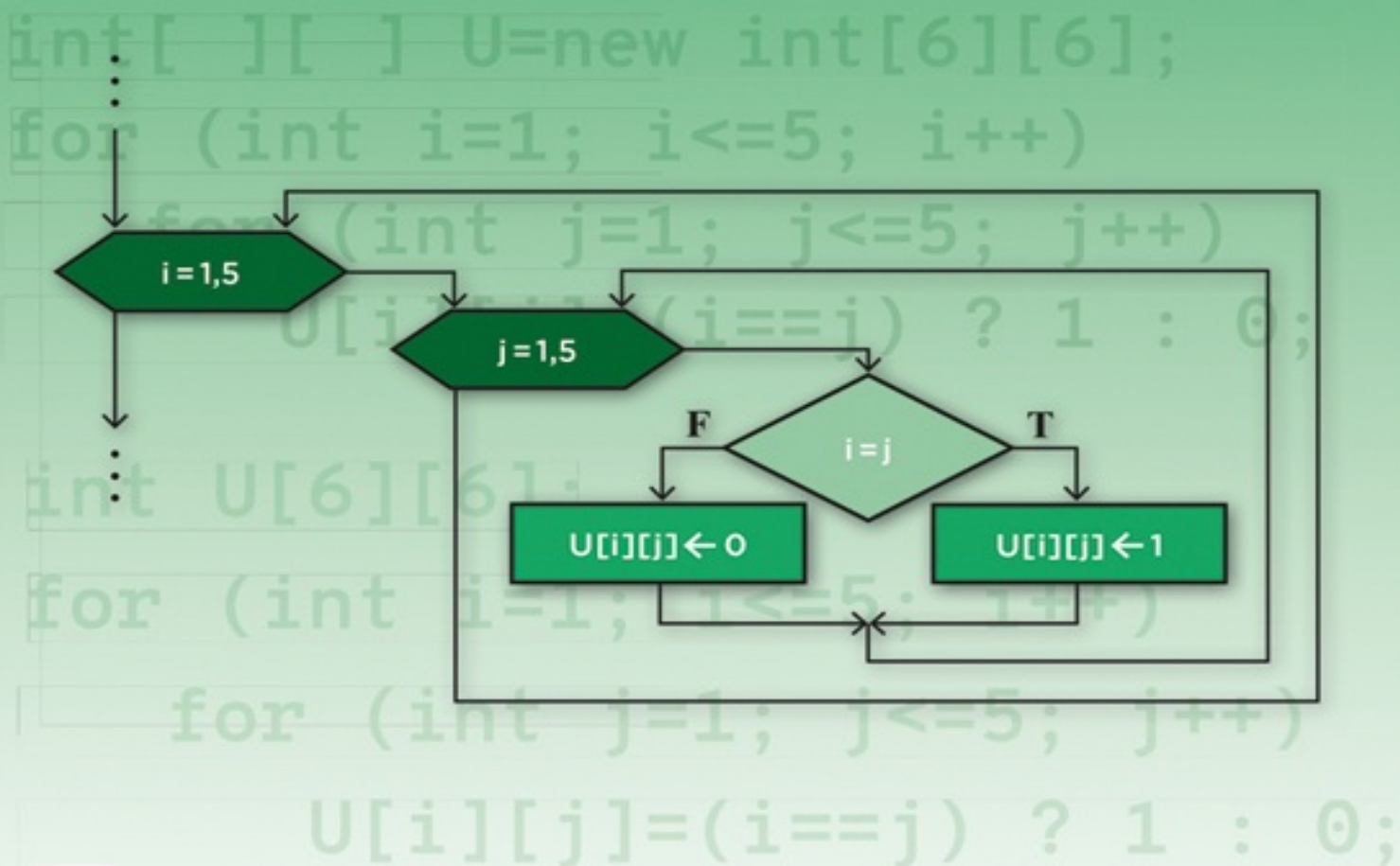


Ali S. Janfada

ELEMENTARY SYNCHRONOUS PROGRAMMING

IN C++ AND JAVA VIA ALGORITHMS



Ali S. Janfada

Elementary Synchronous Programming

Also of interest



C++ Programming

Li Zheng, Yuan Dong, Fang Yang, 2019

ISBN 978-3-11-046943-1, e-ISBN (PDF) 978-3-11-047197-7,

e-ISBN (EPUB) 978-3-11-047066-6



Trusted Computing

Dengguo Feng, Yu Qin, Xiaobo Chu, Shijun Zhao, 2017

ISBN 978-3-11-047604-0, e-ISBN (PDF) 978-3-11-047759-7,

e-ISBN (EPUB) 978-3-11-047609-5



Technoscientific Research

Roman Z. Morawski, 2019

ISBN 978-3-11-058390-8, e-ISBN (PDF) 978-3-11-058406-6,

e-ISBN (EPUB) 978-3-11-058412-7



Web Applications with Javascript or Java, vol. 1

G. Wagner, M. Diaconescu, 2017

ISBN 978-3-11-049993-3, e-ISBN (PDF) 978-3-11-049995-7,

e-ISBN (EPUB) 978-3-11-049724-3



Web Applications with Javascript or Java, vol. 2

G. Wagner, M. Diaconescu, 2019

ISBN 978-3-11-050024-0, e-ISBN (PDF) 978-3-11-050032-5,

e-ISBN (EPUB) 978-3-11-049756-4

Ali S. Janfada

Elementary Synchronous Programming

In C++ and Java via Algorithms

DE GRUYTER

Author

Dr. Ali S. Janfada

Urmia University

Faculty of Science

Department of Mathematics

Urmia, 11km SERO Road, Iran

a.sjanfada@urmia.ac.ir

ISBN 978-3-11-061549-4

e-ISBN (PDF) 978-3-11-061648-4

e-ISBN (EPUB) 978-3-11-061673-6

Library of Congress Control Number: 2019938420

Bibliographic information published by the Deutsche Nationalbibliothek

The Deutsche Nationalbibliothek lists this publication in the Deutsche

Nationalbibliografie; detailed bibliographic data are available on the Internet at <http://>

dnb.dnb.de.

© 2019 Walter de Gruyter GmbH, Berlin/Boston

www.degruyter.com



In memory of my parents
and
to my family

Foreword

In 1975, when I received my B.Sc. degree in mathematics, I knew nothing about programming except for some scattered codes. In fact, in 1973, when I was passing the computer programming course, there were not personal computers on the market like today, and running a program including punching the codes on the punch cards, correcting the administrative mistakes, and re-punching them, as well as observing the required time for their successful implementation on the only central computer of the university and obtaining the outcomes would last almost a week. Therefore, I could completely implement a maximum of ten programs during a semester. After my employment at Urmia University in 1989, computer programming was one of the courses I eagerly taught in mathematics and physics due to my great interest in programming. This passion directed me toward success in writing my first textbook [17] in Farsi in 1994, which soon became one of the most widely used applicable books in the computational field, especially for postgraduate students in physics. Interestingly, my relative skill in programming was one of the reasons for my success during a Ph.D. program on algebraic topology in 2000, as the year of mathematics, at Manchester University, UK. After completing my doctoral degree, I continued teaching computer programming courses,

and the initial ideas for writing the present book were born in my mind in 2004. I first implemented my idea in the form of the pamphlets, and then, I tested it as a textbook [18] in Pascal. In addition, I frequently modified and improved these materials and then collected them as a pamphlet in C++, while drawing upon the assistance of the interested and apt students. Later on, I published them as a textbook [19], which was extensively used by my colleagues in other universities. All these materials were designed and implemented based on the algorithm, which demonstrated desired results in teaching the algorithm-based programming. In recent years, I have found that the above-mentioned programming languages can be simultaneously presented using the algorithms due to the proximity of their codes. The suggestion and request of De Gruyter Publications for publishing the present book encouraged me to simultaneously write the program for both C++ and Java languages, for which I appreciate the staff of this publications.

The current textbook aimed to provide the reader with a more convenient, better, and faster method of programming via the algorithm in both C++ and Java languages. In fact, this is the slogan of the current book: *You will be a professional programmer whenever you become a skilled algorithm designer since a program is nothing but an algorithm with the special codes of a programming language.*

This book covers nearly all the curriculum topics taught in computer science fundamentals and programming for mathematics, physics, and engineering except for computer science during the B.Sc. program and thus can be used as a

textbook for these courses. Actually, the course is more efficient if the instructor considers the applied examples related to each field. Further, the knowledge of high school mathematics is a prerequisite for employing the present book, and therefore, it is useful for high school graduates as well.

It is worth mentioning that this is not a reference book for programming in C++ and Java language; however, it seeks to encourage the readers to develop their skills in algorithm writing for the problems in which mathematical calculations are applied, and simultaneously teach them to translate the algorithm into C++ and Java codes. Therefore, without addressing the computer fundamentals and programming details, algorithm-writing principles and techniques are dealt with after preliminary discussions on algorithm-writing and programming concepts. Furthermore, the book strives to gradually strengthen the readers' ability in this regard so that they can identify and analyze the mental commands which are issued and implemented in their brains for solving a computational mathematics problem and try to design an algorithm based on their understanding and analyses.

To achieve this goal, we introduce a maximum of ten algorithmic templates, which are considered the basic components of the algorithms, and attempt to teach our readers which template to use for each subjective instruction and how to arrange such templates in order to obtain a desired algorithm. Moreover, the translation of each template into C++ and Java programming languages are explained with various details, descriptions, and examples. Additionally, since each

algorithm includes several templates, the translation of the program into C++ and Java codes will be easy by knowing the codes of each template. In addition, we can even translate our algorithms to any programming language if we know the codes of each template in that programming language.

To implement our strategy, in **Chapter 1**, we will explain that most of our daily routines result from one or more algorithms. Then, the algorithm is defined, followed by describing the steps for writing an algorithm. The flowchart, among all types of algorithms, is the most transparent and simplest type and thus, it is easier to write, execute, and translate into the codes of each language. Accordingly, in this book, flowcharts are regarded as the framework of our algorithms and wherever we talk about the algorithm, we imply the flowcharts.

In **Chapters 2** and **3**, we will take a brief look at the programming alphabet in C++ and Java languages, as well as some of its preliminary concepts and practice the provided explanations by implementing several introductory programs, respectively. These two chapters guide the reader to write simple algorithms and programs. Furthermore, **Chapter 3** describes the nature of object-oriented programming in both C++ and Java languages. Due to the computational purpose of the book, most programs are such that the employed functions (methods) are less dependent on the nature of object-oriented programming.

Our main task begins in **Chapter 4**, which explains how to employ the basic algorithm templates. Sub-algorithms (subprograms) are considered the main frameworks of the

algorithms (programs), which are more perceptible in large algorithms (programs). Moreover, in **Chapter 5**, by addressing a variety of sub-algorithms and subprograms, we elaborate on how each type of sub-algorithm (subprogram) is written and called in the main algorithms (programs). By going through these two chapters, a novice reader learns to walk. In other words, he learns how to create intermediate algorithms and translate them into C++ and Java codes.

Additionally, automated and conditional loops are covered in **Chapters 6** and **7**, respectively. The ability of algorithm-writing is substantially strengthened in these two chapters using numerous examples. In addition, algorithmic templates are completed in the above-mentioned chapters. By studying these chapters, the reader learns how to create suitable algorithms and write the desired programs

Acquiring skills in algorithm writing and programming is provided in **Chapters 8** and **9**, in which we deal with one-dimensional and two-dimensional arrays called also vectors and matrices, respectively. It should be noted that more than half of the applications use arrays. In particular, **Chapter 9** discusses a variety of methods for solving linear equation systems.

In this book, several guidelines are presented in order to solve the problems at the end of the **chapters (4-10)**, which help the reader to get some ideas from this section in the cases they fail to solve these problems.

Sometimes, we do not understand part of a movie when we watch it. In such a case, we rewind the film for further understanding and review it several times. We may even watch

it slowly in order to finally figure out what is going on in that section. Given the nature of this course, students are advised to consider and apply three points for learning useful materials.

The first point is to try to understand what processes your mind goes through for doing a daily routine or solving a computational problem. More precisely, try to find out what is going on in the brain by repeatedly reviewing and even slowly moving such processes. Then, identify and analyze the implemented processes.

Next, classify your own perceptions of this recognition and analysis in a set of regular instructions and design an algorithm using such a set so that the other person who runs this algorithm obtains the same answer as your mind did.

Finally, learn the components (templates) of an algorithm, which are up to a maximum of ten, use them appropriately, and try to improve your skills in designing an algorithm by further practice.

We write the name of all sub-algorithms, functions (method), templates, and statements, as well as the keywords and for loop specifications in code font (the same as what was just used for 'for'), while the names of the variables are written in the italic font. In general, we are bound to use the code font everywhere in the flowchart or program, and we apply italic fonts wherever we speak of the variables in the text. However, we employ the font of the table itself in the tables, which is nearly the same as the two above-mentioned fonts.

Numbering the tables and figures is by the number of involved examples.

For the readers' convenience, all the programs of the book are provided on De Gruyter website [www. degruyter.com](http://www.degruyter.com) (search: Elementary Synchronous Programming).

In every human endeavour, there are a considerable number of people worthy of respect and gratitude due to their invaluable moral, intellectual, and technical support. While not being able to name them all, I have selected to highlight a few.

I would like to gratefully appreciate Virayeshyar Editing Company, especially Javad Gholami, Associate Professor of Applied Linguistics at Urmia University, for his valuable guidance and fast and meticulous editing of the present book. I also thanks Amin Habibzadeh and Mohammad Mohammadi for their helps in editing parts of the book.

My sincere thanks are due to Saeed Matlabjoo for his valuable suggestions regarding some issues, as well as his assistance in solving several systemic problems in writing and working with the format of the publications.

I am also profoundly grateful to the staff of the Walter de Gruyter Publishing House for all the publishing stages of this book. I owe special thanks to Katja Schubert, the Production manager, who patiently and constantly provided me with technical guidance. In addition, I would like to thank Leonardo Milla, Aneta Cruz-Kaciak and Angelika Sperlich for all their helps during the preparation of the book.

Special thanks go to two people without whom I could not publish this book in due time and with such quality. My son, Erfan, who translated the original text from Persian to English and patiently modified several English mistakes after the

author's technical reviews and revisions. His encouragement has always inspired me. Further, Ehsun Rasoulion, a very active student of the author at undergraduate and postgraduate levels, was very helpful in carrying the burden of converting all the C++ programs to Java and their implementation. He, also, significantly contributed to the implementation of some C++ programs, at times supported the author, technically studied the entire text of the book, and finally, provided useful suggestions.

My heartfelt gratitude goes to my dear wife who provided a quiet environment for me throughout the years of writing the present book, as well as the other books and helped me progress in other research studies.

Finally, I apologize for all the respected readers for the weaknesses and shortcomings which may have skipped my attention. I sincerely request you, the Reader, to acknowledge such shortcomings to the author.

Spring 2019, Ali S. Janfada

Contents

Foreword

1 Basic concepts of Algorithm

- 1.1 Algorithm
- 1.2 Flowchart

2 Fundamental concepts of programming in C++

- 2.1 Primary concepts
 - 2.1.1 Reserved words
 - 2.1.2 Identifiers
 - 2.1.3 Data types
 - 2.1.4 Variables
 - 2.1.5 Constants
 - 2.1.6 Operators
 - 2.1.7 Library (predefined) functions
 - 2.1.8 Arithmetic and logical expressions
- 2.2 Introduction to programming in C++ language
 - 2.2.1 Output statement
 - 2.2.2 Input statement
 - 2.2.3 Formatted output
- 2.3 Pointers

3 Fundamental concepts of programming in Java

- 3.1 Primary concepts

- 3.1.1 Data types
- 3.1.2 Literals and variables
- 3.1.3 Operators
- 3.2 Introduction to programming in Java
 - 3.2.1 Output and input statements
 - 3.2.2 Formatted output
- 3.3 Object-oriented programming (OOP) system
 - 3.3.1 Objects and class
 - 3.3.2 Types of variables
 - 3.3.3 Constructors and destructors
 - 3.3.4 Destructors and namespaces (C++ only)
 - 3.3.5 Static elements
 - 3.3.6 The this keyword

4 Decision making and branching templates

- 4.1 The if-else template
- 4.2 The if template
- 4.3 The if-else-if template
- 4.4 The switch statement
- 4.5 More applications of the if template
 - 4.5.1 Transferring the program execution
 - 4.5.2 Terminating the program execution

Exercises

5 Sub-algorithms and subprograms

- 5.1 Sub-algorithms
- 5.2 Subprograms
 - 5.2.1 Functions
 - 5.2.2 Multi-return sub-algorithm (subprograms)

5.3 Self-calling (recursive) functions
Exercises

6 Automated loops

6.1 The for template

6.2 Series

Exercises

Supplementary exercises

7 Conditional loops

7.1 The while and do-while templates

7.2 More applications of the conditional loops

7.3 The if-goto loops (C++ only)

Exercises

Supplementary exercises

8 One-dimensional arrays

8.1 vectors

8.2 More applications of the arrays

Exercises

Supplementary exercises

9 Two-dimensional arrays

9.1 Matrices

9.2 Solving linear equations system

9.2.1 Direct ways

9.2.2 Iterative methods

Exercises

Supplementary exercises

Hints for the exercises

Bibliography

Index

1 Basic concepts of Algorithm

1.1 Algorithm

The word **algorithm** is derived from the name of a great Iranian Muslim mathematician, astronomer, and philosopher named al-Khawrizmi¹, who lived in the second century anno hegirae. In general, by an algorithm, a set of usually predefined and orderly instructions is meant to be followed to solve a particular type of problem. One may perform various kinds of algorithms without being aware of them, in their routine and daily lives. For instance, the process of preparing a meal requires several instructions since different stages are needed in cooking and preparing it for serving. Take as another example, attending a church or mosque to say one's praying. This ritual needs several processes which can be turned into an algorithm. The same applies to robots. These machines are designed based on a special algorithm which includes a set of instructions to implement them in a way that when such instructions are implemented successively, they move harmonically and perform the predesigned tasks accurately. Other instances could be constructing a building, driving a car, taking medicine, setting the table, reading a book, taking a trip, publishing a book, directing an office, and many other types of activities. When a mathematical problem is suggested in a classroom, and the

corresponding data are presented afterwards. Consequently, particular solutions are expected to be obtained. Accordingly, a problem-solving algorithm is implemented² in order to arrive at specific results. Even the processes which follow in somebody's mind during solving a problem in their personal lives, some kinds of algorithms occur somewhat differently, for these algorithms are not predefined vividly.

Having understood a problem, one could investigate various steps to approach the solution. That, in turn, requires several other phases to arrive at a certain finding to the posed problem. In the following, a more concise and accurate notion of the construct of a standard algorithm is elaborated and defined.



An algorithm refers to a set of instructions applied in solving a problem. Accordingly, four different phase are required to perform this activity. These particular characteristics include: **concise expressions**, **adequate details**, **the order of the phase**, and **the completion phase**. Hence, the presences of all these features are absolutely essential to perform a standard algorithm.

There are many discussions which lack all those four characteristics mentioned in the definition given above. For instance, suppose the instructions to take a particular medicine are given as follows. Dissolve a teaspoon suspension in a glass of water and drink it three times a day for ten days. It is obvious that this instruction does not include some of the features mentioned in the construct definition given above. First, the exact capacity of a teaspoon, or a glass is not specified precisely.

Second, it is not clear whether “three times a day” means one per every eight hours or after each daily meal. Third, the order of pouring the medicine and water inside the glass are not explained concisely. In some cases, the accurate order of instruction seems to play a phenomenal role such as pouring boiling oil and water.

From now on, by algorithm, the one with all of the four mentioned features is meant. As mentioned earlier, different types of algorithms have already been implemented in our daily lives. It is worth mentioning that the ideas that are made in the brain can be transformed into algorithms, not those based on the feelings erupting from the heart. In other words, all human activities caused by the five senses could be expressed in the form of various algorithms.

To further elucidate the relationship between algorithms and the processes happening in the brain, an example is given to elaborate it and make a feeling of the issue. Assume you have been requested utter even numbers from 2 to 10. You quote 2, 4, 6, 8, and 10, immediately. Nonetheless, have you ever wondered the processes which occur in the brain to mention them so promptly? To further expatiate the issue, imagine a video records the reactions and the processes which happen in your brain while enumerating the even numbers from 2 to 10. That is, the processes which starts off ever since the question is posed and duration in which the brain’s processes is to provide an answer. Having recorded the process, watch the recorded item in a slow motion. Then you can visualize different processes the brain applies not only in analysing the issue but arriving at a solution to the posed problem that, in turn, is changed into an

algorithm. Thus, if that algorithm is given to somebody else to implement, they will arrive at the same solution that have already been achieved.

In short, the answer to the posed question is divided into two parts. The first part examines what the brain's commands, step by step.

Put the number 2 (the first even number) in the "memory". Then, till the number in the memory does not exceed 10, repeatedly, write up or utter the number in the memory and then add 2 to it and substitute for the preceding number in the memory.

The second part analyses these processes in the brain very carefully and patiently in full details in order to convert it into an algorithm.

Algorithm 1.1(a).

1. In a particular "location", first put number 2;
2. Write the number in the location;
3. Add 2 units to the number in this location and substitute the result for the preceding one;
4. If the number in this location is less than or equal to 10, continue implementing the algorithm from the instruction number 2;
5. Or else, complete implementing the algorithm.

By the "place" in the above algorithm, we mean the "memory" mentioned in the brain's commands. In the brain, this algorithm is, indeed, implemented quickly and the answer to the above question is achieved. We too, along with the brain, but slower

than it, implement this algorithm precisely and record the results in the table below.

Tab. 1.1(a): Results of implementing **Algorithm 1.1(a)**.

Memory (place):	2	2+2=4	4+2=6	6+2=8	8+2=10	10+2=12
Value in the memory:		2	4	6	8	10

The core aim of this book is to create and gradually strengthen the ability to imagine a slow motion (video) of the brain's process and analyse the details of it, also, express them as an algorithm in the readers. The extreme importance of the ability in designing algorithms can be understood in the following fact which is the ***slogan of the book***.



You will be a good programmer whenever you become a skilful algorithm designer!

Yes, programming is as simple as this, provided that you possess **elegance, precision, and sensitivity**; elegance in imagining and analysing the brain's commands, precision in designing an algorithm, and sensitivity in the correctness of the algorithm and fixing the probable mistakes.

Expressing the recent algorithm in the *writing form* seems vulgar. By establishing and defining appropriate symbols and variables, we could express it in a simple and *symbolic form*. We present the requested even number with the variable E. The

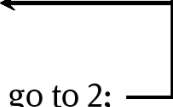
variable E is, in fact, the “memory” in the brain’s commands or the “place” in Algorithm 1.1(a). We will use the assignment symbol $E \leftarrow 2$ to assign (save) 2 to E and the $E \leftarrow E+2$ in order to add two units to the last value saved to E and substitute for E ; that is, after adding 2 units to the last value of E in the memory, the last value is removed and the new value substitutes (saves) for it.



The direction of the arrow in these standard symbols should always be from right to left. Using a symbol such as $2 \rightarrow E$ is incorrect. The reason will be explained in [Chapter 4](#).

With the given symbols, we modify [Algorithm 1.1\(a\)](#) into a simpler form.

Algorithm 1.1(b).

1. $E \leftarrow 2$;
2. Write E ; 
3. $E \leftarrow E+2$;
4. If $E \leq 10$, go to 2; —
5. End.

In this algorithm, there exists a loop, (instructions which are repeated several times) where the direction of the repetition is marked by an arrow. Note that the words like “read, write (print), go to, if, then, terminate”, etc., also, some of the algebraic operations used for the algorithm implementer are assumed to be predefined.

Now we modify **Table 1.1(a)** to **Table 1.1(b)**, called the **implementation table**³.

Tab. 1.1(b): Implementation table of **Algorithm 1.1(b)**.

Processing	E	Output
before the loop	2	
first iteration	4	2
second iteration	6	4
third iteration	8	6
forth iteration	10	8
fifth iteration	12	10



Rule 1 of arranging implementation table: Create columns according to the number of variables. Also, consider a column for the resulting outputs of the algorithm. Now in the top row, we specify the column heads. The second row, named “before the loop”, is assigned for the results of the instructions before the loop, if any. Then, in each row, we write down the results of each iteration of the loop. The final row dedicated to the instructions (mainly one or more printing) implemented after exiting the loop.

Although, arranging this table is often simple, its importance is not less than the importance of the given steps for writing an algorithm.



If there is any **carelessness** in arranging the implementation table, we might not be able to notice the malfunctioning of the algorithm and **all of our efforts for writing an algorithm would be wasted.**

When arranging the implementation table, we are, in fact, putting ourselves in the place of the implementer of the algorithm, which could be a human or a machine, and we are implementing the instructions of the algorithm in the given order. What is seen in the output column of the implementation table will be displayed, with a specific pattern, in the output unit of the computer.

Why are we stressing the importance of algorithms? This is primarily because an algorithm is the structural language of a computer program in any programming language. Undoubtedly, all computer programs are rooted in one or several algorithms. Computers are basically founded on algorithms. Hence, without algorithms, a computer is nothing but a bunch of useless pieces.

Generally, when we are solving a problem, we are thinking at different moments, subconsciously, about cases such as: Where should I start from? What should be done now? Is it time to get the input? What inputs? Is it time to insert the results? What results? Is it time to make a (double or multiple) decision? What decision? Is it time to form a loop? What loop? From where should I start the loop and how should I end it? Has the problem ended completely?

Several steps are conducted for answering each of the above-raised questions. The position of these steps and their order are very important as they lead to the right direction of solving the problem.

When we write an algorithm, we express the solution of the problem in a stereotyped manner. In other words, we arrange the steps required to solve the problem next to each other in a stereotyped way regarding their position and order.

The algorithm of a problem plays the role of the solution of it and writing a program in a programming language is like expressing this solution in that language so the computer can execute it. What is important is the solution of the problem. This is because in order to express it in a programming language, we only need to be familiar with the rules and regulations of that language; however, if we do not know the solution, even if we know many programming languages, they will be of no use since we do not know what to write.

If you take a look at cites [4, 7, 13, 17–19, 22] and the similar programming references based on algorithms (flowcharts), you will get a more thorough understanding of this fact. The author of these books have tried to represent the similarities of the syntaxes of the statements of the C++, Java, Pascal, and Fortran languages and to conclude that if you know a programming language, you could learn another programming language very readily in a short period of time. At least for the computational purposes in the fields of Mathematics, Physics, and most of engineering courses, the learning of how to program is based on understanding the algorithm. You will definitely confirm this fact once you complete reading this book.

On the other hand, if the solution is not correct, what we write is nothing but a waste of time. In other words, the more we haste in writing the program, the more time we spend and the later in achieving a good program. Conversely, if we pay

more attention to the details of the solution of the problem and express them as an algorithm, writing a program will be nothing but a simple translation of the algorithm with the use of a few specific formats and a limited set of rules.

Of what has been discussed about algorithms until now, we can provide some concluding remarks on the general steps of writing an algorithm.



Steps of writing an algorithm:

First step (solving the problem). We solve the problem with the scientific method of the relevant field and write down the required formulas if there are any;

Second step (analysing the solution). We picture, stage by stage, the responses and processes of the brain in arriving at the solution with complete elegance. We review the details of the solution from the brain's point of view and identify and analyse all of the steps regarding their position and order; **Third step (writing the algorithm).** By defining the variables, if necessary, and with a lot of precision and patience, we review the analysis the solution to the posed problem in the second step and we write down each stage regarding its position and order in the language of algorithms (we will start studying the language of algorithms from [Chapter 3](#));

Fourth step (implementing the algorithm). We should be sensitive in the correctness of the algorithm: We arrange an implementation table and write down the results of implementing the algorithm in the third step inside the table.

These four steps not only show the routine procedure of algorithm writing but also create the basis of programming. What follows illustrates these steps:

solving the problem → analysing the solution → writing the algorithm → implementing the algorithm

↓
answer

↓
answer

If these steps are conducted correctly, and, if the first answer, which is derived from the brain's process, matches the last answer, which is derived from implementing the algorithm, then the job is almost complete; there is only one more step remaining for programming and that is nothing but a straightforward translation:



Fifth step (writing the program). Regarding the rules of a chosen programming language, we translate the algorithm of the third step to the desired language.

Several algorithms could be written for a single problem. However, a good algorithm should, 1) work correctly; 2) have fewer steps; 3) not confuse the reader with a proliferation of decisions and additional branches such as, if-then, or, goto statements; 4) get the results as fast as possible (require the least implementation time).

The following three algorithms are equivalent with **Algorithm 1.1(b)**.

1.1(c). 1. $E \leftarrow 2$; 2. Write E; 3. $E \leftarrow E+2$; 4. If $E > 10$ End; 5. Go to 2.

1.1(d). 1. $E \leftarrow 0$; 2. $E \leftarrow E+2$; 3. Write E; 4. If $E < 10$ go to 2; 5. End.

1.1(e). 1. $E \leftarrow 2$; 2. If $E > 10$ End; 3. Write E; 4. $E \leftarrow E+2$; 5. Go to 2.

This is while the following algorithm has less value compared to **Algorithm 1.1(b)** and the three algorithms mentioned above


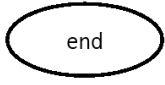




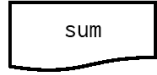
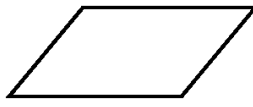
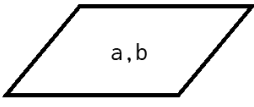
since it does not meet the second feature of a good algorithm.


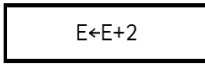
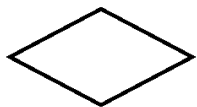
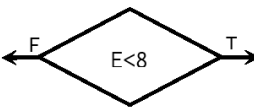

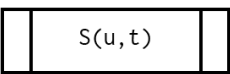
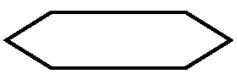
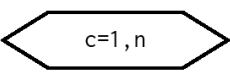

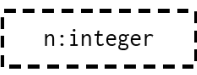
1.1(f). 1. $E \leftarrow 2$; 2. If $E > 10$ go to 6; 3. Write E ; 4. $E \leftarrow E+2$; 5. Go to 2; 6. End.

1.2 Flowchart

In the previous section, we got familiar with algorithms and the two types of written and symbolic forms for writing an algorithm. We saw that the symbolic form was simpler than the written one. An even simpler way of writing an algorithm is by writing it in a *figurative form*. An algorithm which is written with this illustrative method is called a **flowchart**. In other words, a flowchart is an algorithm written with the language of shapes. The following table describes some of the common constructive shapes used in drawing a flowchart including their effects and an example of each one of them. For more information concerning algorithms and flowcharts see the references [2, 6, 7].

Tab. 1.2: Constructive shapes used in flowcharts and their applications.

Shape	Application	Example	Meaning
	beginning of algorithm or sub-algorithm, end of algorithm, or returning from sub-algorithm	 	terminate the algorithm! terminate the sub-algorithm and return to the calling unit!
	determining a position or transferring to a position		transfer the control (of implementation) to the position P!
	outputting the results (on the screen)		print the value of sum (on the screen)!
	inputting the data (from the keyboard)		read the values of a and b (from the keyboard)!

Shape	Application	Example	Meaning
	computation, if any, and assignment		add 2 to the last value of E and substitute (save) for E!
	decision making and branching		if $E < 8$, continue T-direction; otherwise continue F-direction!
	calling sub-algorithms, except functions		call the sub-algorithm S with the mentioned arguments!
	implementation or repetition of do loop		implement (repeat) the do loop with the mentioned specifications!
	side explanation		take the variable n as an integer!

These constructive shapes are connected to each other by arrows. The direction of the arrows represents the flow of the algorithm.



Rule of directions. In this book, we adopt the branches **towards the right side** as the **T-direction** (T for True) and the branches **towards the left side** as the **F-direction** (F for False). If the direction of any branch is downwards, the priority will be with the right and left directions. In other words, in a right-down double branch, the downwards direction will be F because we have already assumed the right direction as T. Also, in a left-down double branch, the downwards direction will be T because we have already assumed the left direction as F. These details are illustrated in the following figure.



As an example, **Figure 1.1** displays **Algorithm 1.1(b)** visually.

Having the fact that the constructive shapes of a flowchart demonstrate, explicitly, the related instructions, we avoid writing unnecessary phrases such as “read” for inputs, “write” for outputs, or “if” for decisions and branches. Also, for simplicity, several assignment and substitution instructions could be written in a single rectangle. In this case, upon entering this shape, all the instructions will be implemented from top to down.

In compliance with the mentioned cases, and according to the recent statements, one can claim that flowcharts are translatable to any programming language.

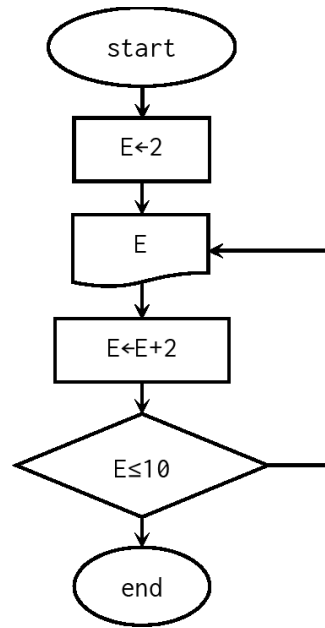


Fig. 1.1: Flowchart of **Algorithm 1.1(b)**.

Since flowcharts are visual algorithms, organizing their implementation table is similar to that of an algorithm. After organizing the implementation table, we implement the algorithm in the specified direction and, if necessary, give it appropriate data and then write down the results of implementing every instruction in its appropriate location inside the table. The input data must be appropriate regarding their number and their type so that they cover all of the different states (like even, odd, negative, positive, zero etc.), and no state is left untested as it is possible that, for example, an incorrect algorithm would accidentally work correctly only for positive numbers, so if negative numbers or zero are not tested, the flaw of the algorithm would not be revealed.

1.1.1. Exercise. Change the flowchart of **Figure 1.1** in a way that instead of writing the numbers themselves, calculate and write their sum.

1.1.2. Exercise. Draw flowcharts of Algorithms 1.1(c), 1.1(d), 1.1(e), and 1.1(f).

Convention. Through the book, instead of the phrase “the flowchart of Figure A” we adopt writing “Figure A” or “Flowchart A”, for simplicity.



In flowcharts, to prevent any bustle in the diagrams, some details like printed designs, long calculation formulas, and non-significant and huge details are not written and are applied only in programming. In choosing the constructive shapes of the flowchart in **Table 1.2**, we have focused on their applications. However, different authors may choose different shapes to construct flowcharts.

It is interesting to know that in the middle of the 1960s, computer terminals were used for time-sharing access to central computers. Before the advent of personal computers (PCs) in the early 1970s, computers were generally large, and costly systems were owned only by large corporations such as universities and government agencies. End users generally could not directly interact with the machine. Instead, in order to run a program, a number of assignments for computer were gathered up by punching some cards. A punched card was a flexible write-once medium that encoded data, most commonly 80 characters. Often, each statement, was punched on one card. Groups or "decks" of cards formed programs and collections of data. Users could create cards using a desk-sized keypunch with a typewriter-like keyboard. After the job was completed, users could collect the results. In some cases, it could take hours or

days between submitting a job to the computing centre and receiving the output. If a small error, no matter how trivial, occurred in a card, the program would give an error.

Now, back to the drawing board! To correct this error, one would have had to punch another entire card, replace it with the error card, rearrange the cards and input them to the computer. On the other hand, the outputs were written on a printing paper, and there were no displaying monitors. Sometimes, in the process of running a program, the existence of an output statement inside an infinite loop would waste a pack of papers, like the program related to [Algorithm 1.2](#).

Algorithm 1.2.

1. $E \leftarrow 2$;
2. Write E; 
3. $E \leftarrow E+2$;
4. Go to 2. 

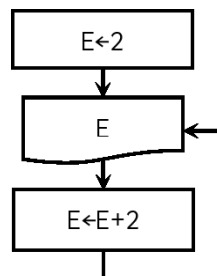


Fig. 1.2: Main part of [Algorithm 1.2](#)

We say that the implementation of this algorithm is locked, that is, we have an endless loop which, if not stopped anyway, will continue to implement forever.

We should be glad that nowadays correcting mistakes in programs is much faster and without any waste of resources.

Generally, the time span needed from starting the programming until acquiring its results has reached its least possible amount.

Algorithm 1.3. Write the main part of an algorithm to swap the values of m and n .

Solution. At first glance it might look as if the two instructions below could do this:

$$\begin{array}{c} m \leftarrow n \\ m \leftarrow n \quad n \leftarrow m \\ n \leftarrow m \end{array}$$

To see if this is correct or not, we assume that the values 5 and 10 are saved for m and n , respectively, in the memory. We implement these instructions in succession. Implementing the first instruction will assign the value of n , which is 10, for m . That is to say that the previous value for m , which was 5, is removed and the new value 10 is saved for it. Now by implementing the second instruction, we will have the value 10 for both m and n indicating that the above-stated instructions do not give us what is requested. Draw the implementation table!

The above problem is similar to a situation in which we have two gauges, one containing rice and the other containing sugar, and we want to swap the contents of the two gauges. To do this, we have no other choice but to use an auxiliary gauge. If we mark the rice gauge m , the sugar gauge n , and the auxiliary gauge k , then, to swap the contents of the gauges m and n , first we pour the contents of one of the gauges, say m , into the k

gauge; afterwards, we pour the contents of the n gauge inside m . Finally, we empty k into n .

Likewise, to solve our problem, we consider an auxiliary variable k . Now we act as we did in dealing with the problem of gauges: 1. $k \leftarrow m$; 2. $m \leftarrow n$; 3. $n \leftarrow k$. The flowchart of this part of the algorithm, which is called the **swap algorithm**, is depicted in **Figure 1.3**.

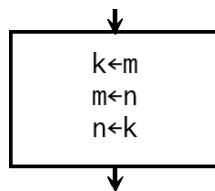
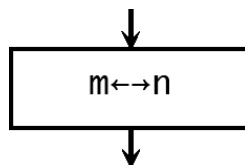


Fig. 1.3: The swap algorithm



Notation. Since this part of the algorithm will alternatively be used in the sequel, for simplicity in drawing algorithms, hereafter instead of the given template, for swapping the values of m and n by the swap algorithm method, we will use the front template in flowcharts



1.3.1. Exercise. Write the algorithm to swap the values of m and n without the use of an auxiliary variable.

1.3.2. Exercise. Write an algorithm for switching the values of x , y and z in a way that the value of x would be transferred to y ,

the value of y to z , and that of z to x .

Algorithm 1.4. Assume that you are asked to read the six “positive” integer numbers and determine the maximum number.

		7	12	9	24	11	18		
7	12		9		24		11		18

Solution. You instantly write 24. How could your brain fulfil this? What would you do if you were given 100 numbers instead of 6 numbers? Could you have determined the maximum number as fast as you did now? For this problem, one might get a pen and paper and by starting from the first number, read a number each time and after comparing it with the next number (next several numbers, instantly but one by one) compare the read number with the number on the paper and, if the read number is larger, cross the number on the paper and write the larger one on the paper.

One might do this by whispering it to oneself instead of writing it on the paper. The act is the same in both ways. We will analyse, at a slow motion, the reaction of the brain in this method for 100 numbers:

Reserve two places in the memory: the first for writing the maximum number and the second for counting the read numbers. Put zero in the first place. The reason for this is that our method is comparing and replacing the larger one and all of the positive numbers are larger than zero. Put 1 (the counter of the first number) in the second place. Afterwards, until exactly

100 numbers are not read, each time read a number and after increasing the counter, compare it with the value existing in the first place. If the read number is larger than that, substitute it; otherwise, terminate the algorithm by writing the last number in the first place.

This is the slow-motion visualization of the brain's reaction, and it is from here that an algorithmic idea is derived from it.

Before continuing to transfer the brain's commands into the form of an algorithm, we want you to ponder over this problem and try to draw a flowchart for it. It is likely that your flowchart may be different from ours which is completely natural because we all think in disparate ways.

In order to settle this idea as a flowchart (visual algorithm), first we define the requested variables:

- n*: the number which is supposed to be read;
- max*: the first place in the memory to write the maximum of read numbers;
- c*: the second place in the memory to write the counter of read numbers.

Defining the variables has two major advantages. First, the implementer of the algorithm gets to know each variable and its role. Thus, there is no ambiguity in implementing the algorithm. Second, when writing a program in any programming language, the data type of each variable, say, integer, real, character, etc. is declared to the computer in an appropriate position with the codes of the desired programming language. We will further discuss this in the next two chapters in the C++ and Java

languages. Hence, if we know the variables well, we will be able to choose an appropriate data type for each respective one.

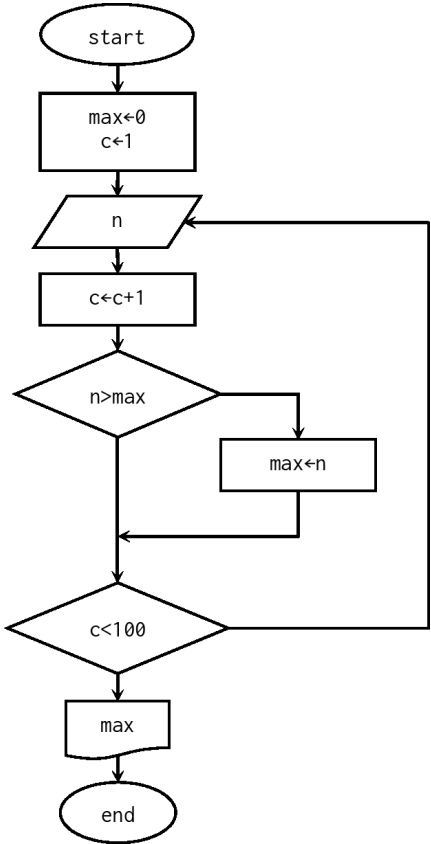


Fig. 1.4(a): A testing flowchart for Algorithm 1.4.

Now we transform this idea into the form of an algorithm. To begin with, we assign 0 and 1 as the initial values of *max* and *c*, respectively. Then, in each repetition we first read the number *n* and instantly increase *c* by one. Next, the read number *n* is compared with the last number assigned for *max*; if the read number *n* is bigger, it is substituted for *max*. Well, how do we write the condition of repeating the loop? Let us test the condition “if *c* < 100, then repeat the loop from the part of reading *n*”; after implementing the algorithm, if this condition worked correctly in the implementation table, then we are done.

Otherwise, we correct it. Also, we put printing the value of *max* upon exiting the loop. **Figure 1.4(a)** illustrates these steps.

We organize the implementation **Table 1.3** of **Flowchart 1.4(a)** for the six numbers given in the hypothesis of **Algorithm 1.4** instead of 100 numbers (see **Tab. 1.3**). We follow the fifth repetition of the loop: the number 11 is read, one unit is added to the value 5 *c*, and the new value 6 substitutes for *c*. Then, in the first condition, the F-direction right is followed and since the second condition is not true, the F-direction down is followed to exit the loop. Although the output is correct, the numbers are not finished yet! Therefore, the algorithm is not functioning properly and to remedy it, we may change the second condition to $c \leq 6$ or $c = 6$ (in the case of one hundred numbers to $c \leq 100$ or $c = 100$) (see **Fig. 1.4(b)**).

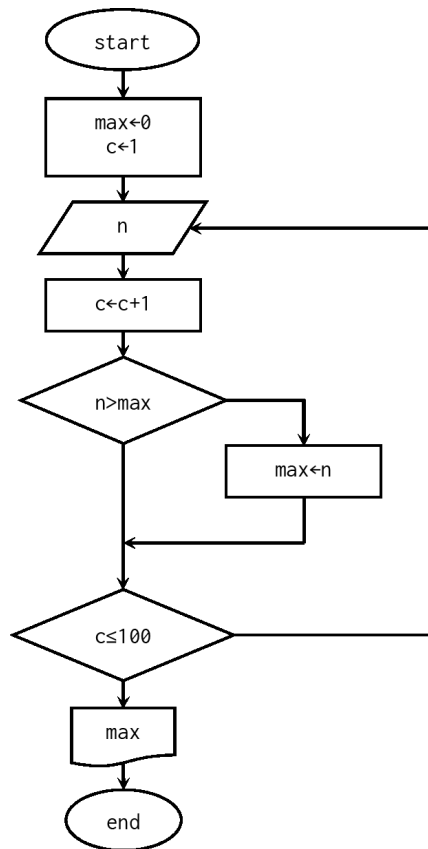


Fig. 1.4(b): Modification of **Flowchart 1.4(a)**.

Tab. 1.3: Implementation table of **Flowchart 1.4(a)**.

Processing	n	c	max	output
before the loop		1	0	
first repeat	7	2	2	
second repeat	12	3	12	
third repeat	9	4	12	
forth repeat	24	5	24	
fifth repeat	11	6	24	
after the loop				24

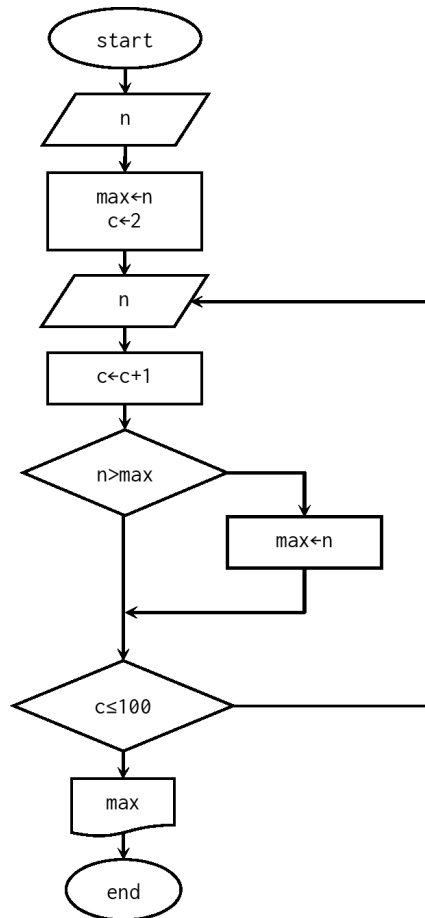


Fig. 1.4(c): Modification of **Flowchart 1.4(b)** to work for arbitrary numbers.

We witnessed that organizing the implementation table, **First**, showed that the algorithm is not working correctly; **Second**, revealed where and how to fix the problem.



Rule 2 of arranging implementation table. In organizing an implementation table, we limit the number of repetitions if there are too many of them and we choose the data which is supposed to be read in an appropriate and varied manner.

Question. Does **Flowchart 1.4(b)** work for 100 “arbitrary”, not just positive, numbers? If not, then how should we modify it to

have a desirable flowchart?

Answer. It is clear that the answer is negative. For instance, if all the numbers are negative, then the algorithm would not work correctly, and the output will be 0. What should we do?

The answer would be trivial if we were aware of the essence of the numbers. For example, if we knew that all the numbers are greater than -1000 then we could have taken that number as the initial value of *max* and left the other parts of the flowchart unchanged. However, if we do not have such an assumption, then the solution is to read the first number before the loop and take that as the initial value of *max*. [Figure 1.4\(c\)](#) shows this process.

Though reorganizing the implementation table, we see that one more alteration needs to be done: we should take 2 as the initial value of *c*. In fact, we start counting from 2 because one number is read before the loop.

To this point, we have only introduced the algorithm and flowchart with a few examples. We will start the techniques of writing algorithms from [Chapter 4](#).



Convention. From now on, throughout the present book, wherever we speak of algorithms, we mean visual algorithms, that is, flowcharts.

2 Fundamental concepts of programming in C++

C++ is a middle-level programming language and is the successor of the C programming language which was first introduced by Denis Ritchie at the AT&T's Bell Laboratories in the USA in 1972. Denise Ritchie used the concepts of BCPL and B to develop C and added data typing as well as some other powerful features.

The inception of C++ programming language began in 1979 when Bjarne Stroustrup was working on his Ph.D. dissertation. He started working on a new language with an object-oriented paradigm, Simula, and mixed it with the features of C programming language. In 1983, he included some add-on features such as classes and called the "C with Class" as C++. Also, C++ is interpreted as C+1, denoting one (skill) more than C; for, as we will see in the sequel, the ++ operator adds 1 to its operand. The first commercial edition of C++ programming language was released in October 1985. C++ is standardized by the Joint Technical Committee ISO/IEC JTC 1 of the International Organization for Standardization (ISO) and the International Electro technical Commission (IEC) that develops and facilitates standards within the fields of programming languages, their environments and system software interfaces. So far, five revisions of the C++ standard have been published, and currently the next revision, named C++20, is under way. These revisions are shown in [Table 2.1](#).

Tab. 2.1: Revisions of the C++ standard.

Year	C++ standard	Informal name
1998	ISO/IEC 14882: 1998	C++ 98
2003	ISO/IEC 14882: 2003	C++ 03
2011	ISO/IEC 14882: 2011	C++ 11, C++ 0x
2014	ISO/IEC 14882: 2014	C++ 14, C++ 1y
2017	ISO/IEC 14882: 2017	C++ 17, C++ 1z
2020	to be determined	C++ 20

C++ runs on a variety of platforms, such as Windows, Mac OS, GNU/Linux, and the various versions of UNIX. Anyone who has used either an Apple Macintosh or a PC running Windows has indirectly used C++ because the primary user interfaces of these systems are written in C++.

2.1 Primary concepts

Since the aim of this book is elementary programming, in this chapter we will introduce the fundamental concepts which are essential for elementary programming in C++ and leave the supplementary details for the books involved with advanced programming in C++.

2.1.1 Reserved words

A **reserved word (or keyword)** is a word defined for the C++ compiler for a certain purpose and cannot be used for any other means except for comments. It is “reserved from use elsewhere”. This is a syntactic definition, and a reserved word may have no meaning. C++ programming language has 95 reserved words of which only 32 basic reserved words were also present in the C programming language and have been carried over into C++. Having these number of reserved words, C++, is still one of the fastest and most efficient programming languages. Some of the C++ reserved words that will be used alternatively in this book are summarized in [Table 2.2](#).

Tab. 2.2: Some frequently used reserved words in C++.

break	case	char	class	const	continue
define	default	do	double	else	float
for	goto	if	int	long	namespace
private	public	return	short	signed	sizeof
static	string	switch	unsigned	using	void
while					

The C++ compiler is case sensitive; therefore, For and for will show different behaviours in the C++ compiler.

2.1.2 Identifiers

An **identifier** is a name used to identify a variable, constant, function, subprogram, or other user-defined items. This naming is more important, especially in variables.



Rule of naming identifiers. An identifier is a combination of letters A to Z or a to z, underscores, and digits 0 to 9, provided that the first character cannot be a number.

For example, the names below can be used as identifiers:

counter pi epsilon a X2 t6 tik_tak



- Although identifiers may be longer, they must differ in the first 31 characters if you want to be sure that your programs are runnable in most versions of C++. Some of the old versions of C++ recognize only the first 8 characters of an identifier.
- An identifier should not include the blank space character.
- The first character of an identifier is not recommended to be an underscore character. Beginning identifiers with an underscore is considered poor programming style.
- An identifier cannot have two consecutive underscores.
- An identifier cannot be a reserved word. Reserved words have predefined special meanings for the compiler.
- We will use the above rule and notes, which are valid for all versions of C++, for naming identifiers. However, in the recent versions some other characters are allowed in naming identifiers.

For example, the names below are forbidden to be used as an identifier:

`λ t'19 t_19 t-19 t,19 t__19 -y 6ab i^2 a/b sin(x) f(3)`

Here, the symbol “`_`” stands for the whitespace character (spacebar key) on the keyboard and we will use it for this purpose wherever an emphasis is required or if a failure to its expression causes confusion.

2.1.3 Data types

The name of a program bears no quantity and, only contains a nominal value. By a **data**, we mean an identifier containing a quantity. In order for a data to carry information, it should have a type. C++ offers the programmer a rich assortment of built-in as well as user-defined data types. The basic data types are as follows.

- 1) Integer data type **int**, used for integers.
- 2) Floating point data type **float**, used for real numbers with less precision.
- 3) Double floating point data type **double**, used for real numbers with high precision.

Several basic types, like the numerical types mentioned above, can be modified using one or more of the type modifiers short, long, signed, and unsigned. All the numerical types, in two integer and real groups, are gathered in [Tables 2.3](#) and [2.4](#), respectively including their range of values which can be stored in the data as well as the size of the data, in bytes, which is saved in the memory. The modifier signed does not make any changes and we have not used it in the tables. For example, the data types int and signed int are the same.

Tab. 2.3: Types of integer data.

Type	Range	Size (bytes)
int	-2,147,483,648 to 2,147,483,648	4
unsigned int	0 to 4,294,967,295	4

short int	-32,768 to 32,767	2
unsigned short int	0 to 65,535	2
long int	-2,147,483,648 to 2,147,483,647	4
unsigned long int	0 to 4,294,967,295	4
long long	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	8
unsigned long long	0 to 18,446,744,073,709,551,615	8

In [Table 2.4](#), a E b means $a \times 10^b$.

Tab. 2.4: Types of floating point (real) data.

Type	Range	Size (bytes)
float	$\pm 3.4028 \text{ E } -38$ to $\pm 3.4028 \text{ E } 38$	4
double	$\pm 1.7977 \text{ E } -308$ to $\pm 1.7977 \text{ E } 308$	8
long double	$\pm 1.1897 \text{ E } -4932$ to $\pm 1.1897 \text{ E } 4932$	12

There are two reasons for distinguishing the two types of integers and reals. First is the size of the relevant data stored in the memory. As seen in the tables, the integer data sizes are relatively less than those of the real data. The second reason is that these two types show completely different behaviours in calculations. In particular, calculating with integer data is much quicker.

- 4) Character data type **char**, which carries a character. Each character data is stored in one byte of the memory. In C++, each character data should be written between two apostrophes, for example:

'd' '1' '\ ' '8' '1' ' _ '



Whitespace is also considered as a character.

The two character types are shown in [Table 2.5](#).

Tab. 2.5: Types of character data.

Type	Range	Size (bytes)
char	-128 to 127	1
unsigned char	0 to 255	1

The range and size of data might be different from those shown in the above table, depending on the compiler and the computer system one is using. By the sizeof operator (see the operators subsection below), one can produce the correct size of various data types on the used computer.

- 5) String data type **string**, which stores a sequence of letters, digits, and other characters. In programs, a string must be placed between a pair of quotation marks. For example, "The value of st1.id_no = " is a string. We will study storage of strings in **Chapter 8**.
- 6) Boolean type **bool**, which are true or false. The default numeric value of true is 1 and false is 0 and these data types can be used in mathematical expressions, although we do not recommend this.
- 7) Valueless data type **void**, which will be discussed later.

There is also another wide character data type `wchar_t`, which we will not use in this book. Each data appears in either constant or variable form in the program. We emphasise that identifiers are mostly used in naming variables, constants, subprograms, and user-defined functions. In this book, we will be dealing mostly with integer and real data types.

2.1.4 Variables

A **variable** is an identifier which could be changed during the program. The variables used in a program must have a type. Defining the variables and determining their types in a program is done by the variable declaration syntax as shown below.



Syntax (variable declaration):

a data type _ one or more variables ;

For example, the part

```
int x, y;  
float m, n;  
char c1, c2;  
double d;  
long int p;
```

of a program, declares to the compiler that the variables `x` and `y` are of `int` type, `m` and `n` of `float` type, `c1` and `c2` of `char` type, `d` of `double` type, and `p` of `long int` type.

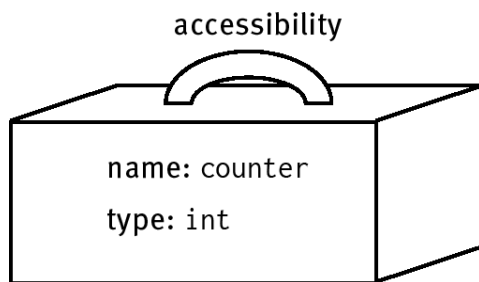
When we want to use a variable in a program, we should first choose a name for the variable based on the role of the variable in the program. For example, the names `sum` for the summation of a series and `count` for the counter of input numbers may be suitable names. We could even use abbreviations for naming. For example, `matA` would be an appropriate name for the matrix `A`. Anyway, the choice of the names is up to the

programmer with no limitations for it. We should only follow the rule of naming identifiers. In this book, for the task of simplicity and avoiding large size programs, we will use short names for variables.

After choosing a name for a variable, we choose a type for it resembling in some sense its application. For instance, it is clear that a variable named count, which acts as a counter of the input numbers in the program, should be an integer type. If a variable is used in a program without declaration, an error will show up.

The variables discussed above are normal variables in C++. On the other hand, C++ is an object-oriented programming language in which feature each variable has an access specifier. An access specifier specifies the positions from where the variable can be accessed. The object-oriented feature of C++ will be discussed in [Section 3.3](#).

Since variables have an essential role in programs, a clear understanding of them is necessary. For this purpose, according to the figure below, a variable could be imagined as a box whose name is the name of the variable, and its content is an amount (numeric or non-numeric) with the same type of the declared name. Moreover, with the object-oriented feature each variable has an accessibility area.



2.1.5 Constants

A **constant** is an identifier which has a fixed value in the program and cannot be changed under any circumstances. Defining the constants and specifying their types in a program is done by one of the following constant declaration syntaxes:



Syntax (constant declaration):

```
#define_ name of constant_ value of constant;  
const_ a data type _ name of constant = value of constant;
```

For example,

```
#define s '*';  
const int n = 100;  
const signed char x = 'a';
```



Declaring variables and constants can be done at any point in the program. However, most programmers prefer to do this at the beginning of the program.

By a **block** in programming we mean a sequence of declarations and statements within a pair of curl braces {}. The content of a block always starts at a new line, shifted two or three spaces to the right. The opening brace { of a block can be placed at the same line of the element that opens this block. The closing brace } should be written on a separate line placed at the same column as the beginning of the element that opens the block. The block-style mentioned above is one of the essential styles of programming. We often ignore using the curly braces for a block which contains a single statement. The rule related to the grouping of the blocks is explained in [Chapter 4](#).

2.1.6 Operators

An **operator** is a symbol that makes the compiler perform specific mathematical or logical manipulations. An operator acts on either single or double operands surrounding it. Operators are divided into several categories.

1) **Arithmetic operators.** These operators in C++ are gathered in [Table 2.6](#).

Tab. 2.6: Arithmetic operations.

Operator	Effect	Example	Result of example
+	addition	2 + 9	11
-	subtraction	7 - 2.5	4.5
*	multiplication	4 * 6	24
/	division and quotient	9.0 / 4 9 / 4	2.25 2
%	remainder	10 % 3	1
++	increase by one	10++	11
--	decrease by one	10--	9

Considering an important note when working with the / operator is necessary.



If at least one of the operands of the / operator is a real value, the result will be real, On the other hand, if both of them are integers, the result will be the quotient of the numerator by the denominator.

Two examples in [Table 2.5](#) resemble the above-mentioned fact. In such cases, if we want to use the / operator for the purpose of division, one reliable method is to write the phrase (float) before the expression. For example, if the values 10 and 8 have already been stored for the variables r and s, respectively, the result of (float)r/s would be the exact value of the division, which is 1.25.



- The result for each of the operators +, -, and * is integer only when both the operands are integers.
- The right side operand of both operators / and % must be nonzero. Otherwise, either an error will show up or the results will be incorrect.
- Both operands of the % operator should be an integer.
- If one of the operands of the % operator is negative, then we will have the following rules:
 - If $a < 0$ and $b > 0$, then $a \% b$ is equivalent to $-(a \% b)$;
 - If $a > 0$ and $b < 0$, then $a \% b$ is equivalent to $-(a \% -b)$;
 - If $a < 0$ and $b < 0$, then $a \% b$ is equivalent to $(-a \% -b)$.
- If either of the operators ++ and -- comes before its operand in calculations, at first, this operator acts, and then the result participates in the sequel calculations. However, if it comes after the operands, the current value of the operand is used and afterwards the operator acts on its operand.

To clarify the last explanation, we provide three examples in [Table 2.7](#). First of all, it is necessary to mention that the expression $i++$ in algorithm language is written as $i \leftarrow i+1$. In the first two examples of the table, we assume that the value 10 has already been saved for the integer variable x in the memory, and, in the third example, we further suppose that the integer variable y holds the value 15 in the memory.

Tab. 2.7: Examples of the operators ++ and --.

Example	Algorithmic meaning	Result
$y = ++x$	$x \leftarrow x + 1$	x: 11
	$y \leftarrow x$	y: 11
$y = x++$	$y \leftarrow x$	x: 11
	$x \leftarrow x + 1$	y: 10
$m = ++x + y++$	$x \leftarrow x + 1$	x: 11

$m \leftarrow x + y$	m: 26
$y \leftarrow y + 1$	y: 16

2) Relational (comparative) operators. Table 2.8 specifies these operators in C++.

Tab. 2.8: Relational (comparative) operators.

Operator:	<	<=	>	>=	==	!=
Math meaning:	<	≤	>	≥	=	≠

The result of the effect of these operators on their operands will be a logical value (1 for true and 0 for false). If the operands are characters, the comparison will be made based on the order of the character s' locations in the ASCII code table.

3) Logical operators. These operators in C++ appear as shown in Table 2.9.

Tab. 2.9: Logical operators.

Operator	Notation in logic	Meaning
!	~	negation
&&	^	and
	v	or



- The! operator has only one operand. !p is true whenever p is false and vice versa.
- p&&q is true if and only if both p and q are true.
- p||q is false if and only if both p and q are false.

4) Assignment operators. Table 2.10 lists the assignment operators supported by C++. With the exception of the first one, all operators are compound.

Tab. 2.10: Assignment operators.

operator	name	example	equivalent
=	assignment	x = y	x = y
+=	addition assignment	x += y	x = x+y
-=	subtraction assignment	x -= y	x = x-y
*=	multiplication assignment	x *= y	x = x*y
/=	division assignment	x /= y	x = x/y
%=	remainder assignment	x %= y	x = x%y

The expression `x = y` makes the compiler assign (substitute) the value of `y` to `x`. This assignment operator is also called the **assignment statement** since it behaves like a statement (see the next section).



- Do not confuse the `==` relational operator with the `=` assignment operator. Using the `==` operator instead of the `=` **statement** often causes an error. However, using the `=` **operator** instead of the `==` operator does not trigger an error. Nonetheless, it could engender delicate logical errors which could eventually lead to serious problems.
- There must not be any space between the two characters in the compound assignment operators; for example, `+=` is illegal.

Apart from the above-cited classical operators, there are also some miscellaneous operators.

5) **The ? operator** (two-way branching). This operator has the syntax below:



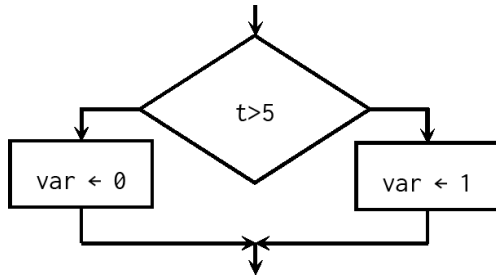
Syntax (? operator):

condition ? expression 1 : expression 2

Effect: If the condition is true, then it returns the value of expression 1; otherwise, it returns the value of expression 2. For example, consider the following assignment statement.

```
var = (t > 5) ? 1 : 0;
```

Here, `var` is assigned the value 1 if `y` is greater than 5 and 0 if it is not. The effect of this assignment can be illustrated in the language of flowchart as displayed in the following figure.



6) **The , operator.** The syntax of this operator is as follows.



Syntax (, operator):

expression 1 , expression 2 , ... , expression n

Effect: The purpose of the comma operator is to string together several expressions. The value assigned to a comma-separated list of expressions is the value of the rightmost expression. The values of other expressions will be discarded. This means that the rightmost expression will become the value of the entire comma-separated expression. Two assignment examples are presented in [Table 2.11](#) to clarify this further.

Tab. 2.11: Two examples of the , operator.

Example (assignment statement)	Result
<code>var = (count = 19 , i = 10 , count + 1);</code>	<code>var: 20</code>
<code>i = (i++, i + 100 , 999 + i);</code>	<code>i: 1010</code>

As you can see, the second (middle) expressions in both examples are discarded. The parentheses are necessary because the comma operator has a lower precedence than the assignment operator.

7) **The sizeof operator.** This operator returns the size of its operand, which is the number of bytes it occupies in the memory. For example, if the variable `var` is declared as a characteristic variable for the compiler, then the result of `sizeof_var` will be 1. As another example, the result of `sizeof(int)` is 2. In the latter example, the parentheses are necessary; otherwise, the compiler will identify `int` as a variable instead of a data type.

There are other types of operators like bitwise operators and other miscellaneous operators which are not used throughout this book.

2.1.7 Library (predefined) functions

In every programming language, certain standard mathematical functions with mathematical applications, so called **library functions**, are predefined for the compiler of the processor of that language. Most of these functions, like $\sin(x)$, appear in all of the programming languages with the same name. However, some of them, like $[x]$, the round down of x (the least integer not greater than x), are defined by different names in various languages. In C++ language, the most frequently used library functions are summarized in [Table 2.12](#). These functions are available in all of the recent versions of C++. Advanced versions might contain additional functions. It should be noted that there are some library functions applied for other than mathematical purposes but, in this book, we will not use such functions.

Tab. 2.12: Some library functions supported by C++.

Function	Argument(s) type	Result type	Effect
$\sin(x)$, $\cos(x)$, $\tan(x)$	int or float	float	trigonometric function of x
$\sinh(x)$, $\cosh(x)$, $\tanh(x)$	int or float	float	hyperbolic trigonometric function of x
$\operatorname{asin}(x)$, $\operatorname{acos}(x)$, $\operatorname{atan}(x)$	int or float	float	inverse trigonometric function of x
$\exp(x)$	int or float	float	exponential of x
$\log(x)$	int or float	float	(natural) logarithm of x
$\log_{10}(x)$	int or float	float	logarithm of x in base 10
$\operatorname{pow}(x, y)$	int or float	int or float	x to the power of y
$\operatorname{sqrt}(x)$	int or float	int or float	square root of x
$\operatorname{abs}(x)$	int or float	int	integer absolute value of x
$\operatorname{fabs}(x)$	int or float	int or float	real absolute value of x
$\operatorname{int}(x)$	int or float	int	integer part of x
$\operatorname{ceil}(x)$	int or float	int	round up x
$\operatorname{floor}(x)$	int or float	int	round down x
$\operatorname{fmod}(x, y)$	int or float	int or float	remainder of x/y

Function	Argument(s) type	Result type	Effect
$\operatorname{max}(x, y)$	int or float	int or float	maximum of x and y
$\operatorname{min}(x, y)$	int or float	int or float	minimum of x and y

Note that the arguments of trigonometric functions are in radian.

In [Chapter 5](#), we will see that, in addition to the library functions, a programmer can define any other needed function in subprograms which are usable in any other program.



- Note the limitation of some arguments. For example, the argument of the functions `log()` and `log10()` must be positive and that of `sqrt` must not be negative. The same note should be taken for the function `pow`.
 - Never use the name of any library function for naming any identifier like a variable or a constant in a program; otherwise, that function will lose its effect, which in turn could cause problems in the results of the program.
-

2.1.8 Arithmetic and logical expressions

In mathematics, an expression such as

$$\frac{2 \sin x + 10}{x^2}$$

containing numbers, variables, mathematical functions, algebraic operators, fractions, or/and pairs of parentheses is called an algebraic expression. An **arithmetic expression** in C++ is a combination of variables, constants, library functions, arithmetic operators, fractions, or pairs of parentheses. The result of an arithmetic expression is a number. The **logical expressions** are defined similarly. Each one of the expressions below is an example of an arithmetic expression in C++:

`sin(x + 2)` `12.0 * t` `-2 * exp(2.0) + k / 4` `abs(u - v) / 10`



In mathematics, `xy` means `x` multiplied by `y` but, in C++ the phrase `xy` means a variable with this name. Always bear in mind that:

- Write the multiplication of `x` in `y` as `x*y` not `xy`;
 - Write the multiplication of 2 in `t` as `2*t` not `2t`;
 - Write the multiplication of -12 in 4.5 as `-12*4.5` not `-12(4.5)`;
 - Write the division of `(x-y)` by `t` as `(x-y)/t` not `x-y/t`;
 - Write the division of 1 by `ab` as `1/(a*b)` not `1/a*b`.
 - Avoid writing expressions which result in an illegal value such as a number divided by zero.
-

When several arithmetic operators act in succession, they act with a top-down priority

* / %
+ -

and subject to the following rule. Note that the = assignment operator has lower priority than all the arithmetic operators.



Rule of priority of operators. if an operator has a higher or equal priority with respect to the next operator, it acts. Otherwise, the operator does not act until the next operator is compared to its next one likewise. When an operator acts, it results in a numeral amount. Now, the previous operators which had been passed over, are executed. Afterwards, we go to the next operators and continue this procedure until we acquire the result of the expression.

During this procedure, 1) whenever we encounter an open parenthesis, at first, we calculate the amount inside the parentheses until it closes. Then we continue the procedure. In the case of nested pairs of parentheses, the priority is with the inner one; 2) whenever we encounter a library function, first, that function affects its argument(s), thereby resulting in a numeral amount, then we continue the procedure.

The priority of some arithmetic operators together with the assignment operator in the two examples below are shown under each operator, as under scripts:

z =₆ p *₁ r %₂ q +₄ w /₃ x -₅ y ;

y =₆ a *₁ x -₄ x %₂ b *₃ x +₅ c ;

Generally, the mentioned rule is used for arithmetic, relational, logical, and assignment operators with a top-down priority as shown below.

() used for function call
 () used for grouping in type
 ! ++ -- signs: + -
 * / %
 + -
 < <= > >=
 == !=
 &&
 ||
 = += -= *= /= %=

Since in this book we are dealing with only the above-mentioned operators, we skip the discussion about other standard and miscellaneous operators and their priorities. Now we

are in a position where we could simply and accurately translate algebraic expressions to C++ codes. Regarding the following notes is strongly recommended.



- To get accurate results from the program and avoid any probable mistakes, pay attention to the type of the variables and constants when choosing their names.
- By considering the priority of the operators, you could avoid writing extra parentheses unless you have doubt in using them, in which case, take caution and use the parentheses.
- Pay extra attention when choosing the type of the library functions and the type of their arguments. Especially when writing the arguments of the trigonometric functions based on radians.
- Note that two relational operators cannot follow each other consecutively. For example, the expression $-2 < x < 2$ is illegal in C++. Instead, you should write: $(-2 < x) \&\& (x < 2)$. While considering the priority of the relational operators and the $\&\&$ operator in the expression above, the existence of parentheses is necessary.

Example. A number of algebraic expressions are written as C++ codes below:

$$\frac{5(x+y)+2z}{x-y} : (5 * (x + y) + 2 * z) / (x - y)$$

$$\pi R^2 + \frac{k+m}{R} : \text{pi} * (R * R) + (k + m) / R$$

$$4 \cos(x + 2y) - 2 \sin^2(x^2) : 4 * \cos(x + 2 * y) - 2 * \text{pow}(\sin(x * x2), 2)$$

$$\frac{\rho}{\frac{\ln x + \sqrt{e^x - e^{-x}}}{|xy|}} : \text{ro} / (\log(x) + \text{sqrt}(\exp(x) - \exp(-x))) / \text{fabs}(x * y)$$

$$\frac{5(x+y)+2z}{x-y} : (5 * (x + y) + 2 * z) / (x - y)$$

$$\pi R^2 + \frac{k+m}{R} : \text{pi} * (R * R) + (k + m) / R$$

$$4 \cos(x+2y) - 2 \sin^2(x^2) : 4 * \cos(x + 2 * y) - 2 * \text{pow}(\sin(x * x2), 2)$$

$$\frac{\rho}{\frac{\ln x + \sqrt{e^x - e^{-x}}}{|xy|}} : \text{ro} / (\log(x) + \text{sqrt}(\exp(x) - \exp(-x))) / \text{fabs}(x * y)$$

Exercise. Translate the expressions below to C++ codes (what we mean by expression is only the calculation, also, the e number is assumed as the Napier number which could be written as $\exp(1)$).

$$t \sqrt{|m^2 - n|}, \sin(x^3) \cos^2(2y), 2 + \sqrt{2 + \sqrt{2 + \sqrt{2 + \sqrt{2}}}}, \cos(\sqrt{e}), \ln x + \frac{\rho}{\frac{\sqrt{e^x + e^{-x}}}{|xy|}}$$

$$\frac{a^2+c}{(y^3+2)(x^3-5)}, \sin^3(\tan(\log(x))), \sin^2(x \cos(2y)) + \sin x^2 \cos(2y).$$

$$t \sqrt{|m^2 - n|}, \sin(x^3) \cos^2(2y), 2 + \sqrt{2 + \sqrt{2 + \sqrt{2 + \sqrt{2}}}}, \cos(\sqrt{e}), \ln x + \frac{\rho}{\frac{\sqrt{e^x + e^{-x}}}{|xy|}}$$

$$\frac{a^2+c}{(y^3+2)(x^3-5)}, \sin^3(\tan(\log(x))), \sin^2(x \cos(2y)) + \sin x^2 \cos(2y).$$

2.2 Introduction to programming in C++ language

Based on computer type, the reader is responsible for installing and using an appropriate Integrated Development Environment (IDE) to work with. The related programming techniques are discussed in the present book. In addition, all C++ programs are run in **Dev C++ 5.11 IDE** workspace.

Statements are fragments of the C++ program which are executed in sequence. The body of any function is a sequence of statements. The functions are discussed in [Chapter 5](#). For the present purpose, it suffices to indicate that a **(main) program** is the function



A C++ program pattern:

```
int main() {  
    body of the function (program)  
    return 0;  
}
```

Hereafter, the italic phrases are not per se parts of the codes of the program in programming. Instead, what is required is replaced for these phrases. Before explaining the details of the above pattern, some programming styles are expressed which help have clear programs for us as well as the users reading and executing these programs. The block-style were previously explained ([Subsection 2.1.5](#)). We describe some more styles as follows.

- A blank is used after the comma and semicolon separators.
- A blank is employed before and after the binary and relational operations. For example, the expressions `a = b + c` and `i < -1` are considered well-read compared to `a=b+c` and `i<-1`, respectively; however, this style is not applied in two-column programming since space is limited.
- No blank is utilized before the unitary operators `++` and `--`; the expression `j ++` is illegal.
- No blank is used between the function name and the parenthesis of its arguments; Never use `prime (n)`; instead, use `prime(n)`.
- A blank is applied between each of the keywords `if`, `for`, `switch`, and `while` and the parenthesis just after them; for instance, use `if (b > 0)`, `for (i=1; i<=n; i++)`, `while (t==0)`.
- Avoid making long lines (i.e., not longer than 120 characters).
- A long statement can be written in any number of lines.
- More than one statement can be written in a single line although it is not recommended.
- The two consecutive input are separated by a space. Never use comma or semicolon for this purpose.

The body of a function is a compound statement which, in turn, is a sequence of statements and/or declarations (with at least one element) surrounded by curly braces:

{ *statements and/or declarations* }

Other types of statements are expression, selection (deciding and branching), iteration (loop), and jump statements. An expression followed by a semicolon is a statement. Most statements are expression statements in C++. The other types of statements are dealt with in [Chapters 4, 6, and 7](#). Additionally, various types of subprograms including functions are covered in [Chapter 5](#).

The operating system of the computer starts executing the program from:

```
int main()
```

and upon completing the body of the program, the statement:

```
return 0;
```

takes the amount zero to the operating system and announces the termination of the program. We may use the function `void main()` instead of the function `int main()` and remove the `return 0` statement.

The first programming statements, namely, the essential ones when working with any programming language, are output and input statements.

2.2.1 Output statement

The process of writing the outputs is conducted using the output `cout` statement with the following syntax.



Syntax (output statement):

```
cout << expression 1 << expression 2 << ... ;
```

Upon executing this statement, `expression 1`, `expression 2`, and the like are written in the output unit, respectively, after their probable calculation if there is any. Then, the cursor remains in its position for another reading or writing (if there are any). Here, the output unit is supposed to be the screen. The expressions in the list of `cout` statement could be:

- 1) Constant, variable, arithmetic expression, or any other kind of expression;
- 2) A string, such as "The sum is: `␣`". In this case, the exact phrase between the pair of quotation marks is printed, which is:

The sum is: `␣`

Throughout this book, strings in the C++ programs are mainly used for the purpose of input notifications or output headings.

- 3) A format (output layout). The formats are discussed later;

- 4) A escape sequence or control character in [Table 2.13](#) (these characters should be placed between a pair of quotation marks, for example "\t");
- 5) endl, which is the same as "\n".

Tab. 2.13 The escape sequences or control characters.

control character	name	effect
\n	new line	move to the beginning of the next row
\t	horizontal tab	move to the beginning of the next eight column (next tab)
\v	vertical tab	move to the beginning of the next eight row
\b	backspace	back up one character removing it
\r	carriage return	move down a line
\a	bib	sound a bib
\"	double quote	write the " character
\	backslash	display the \ character
\?	question mark	display the ? character
\:	quotation mark	display the : character

Program P2_1 is the first program in C++.

```
// Program P2_1 the first program in C++
#include <iostream>
using namespace std;
int main() {
    cout<<"Welcome to C++ programming world";
    return 0;
}
```

Output:

Welcome to C++ programming world

The namespace statement

```
using namespace std;
```

employed in most C++ platforms are discussed in [Section 3.3](#). Assume that this statement provides a wide variety of facilities for the program including output and input statements.

Since input and/or output data exist in nearly every program, we have the **preprocessor directive**

```
#include <iostream>
```

in every program posited at the beginning of the program before the function int main(). This pre-processor directive provides the input/output stream for the program by loading the iostream header. In some old versions, the suffix . h is added to the headers.



Hereafter, the namespace statement and the pre-processor directives are counted as parts of a program.

In general, pre-processor directives are the lines included in the code of the programs preceded by the hash sign, #. These lines are not program statements, instead, they are considered directives for the pre-processor. The pre-processor examines the code before the actual compilation of the code begins and resolves all these directives before any code is actually generated by the regular statements. To put it differently, a pre-processor directive opens a gate of C++ capabilities and facilities for the program, corresponding to the concerned header. There are only a few pre-processor directives in C++ which individually explained whenever needed.

For clearing the work pad, the statement

```
clrscr;
```

is frequently used. In this case, we should place the pre-processor directive

```
#include <conio.h>
```

at the beginning of the program.

In some of the versions, in order to see the output of the program, the statement

```
getch();
```

should be written before the ending statement return 0.

The following is another program, in which the header math. h is loaded to the program by the pre-processor directive.

```
// Program P2_2 to use mathematical functions
#include <iostream>
#include <math.h>
using namespace std;
int main() {
    cout<<"sin(1) = " /* string */<<sin(1)<<"\n\n";
    cout<<"1.5 * 2.9 = "<<1.5 * 2.9; //multiply 1.5 by 2.9

    return 0;
}
```

Output:

```
sin(1) = 0.841471
```

```
␣
```

```
1.5 * 2.9 = 4.35
```

Be aware that in the above program, whatever is between the pairs of quotation marks is directly transferred to the output and written without participating in the process of executing the program except for the control characters. The expression `"\n\n"` is used to create an empty line. Equivalently, `endl<<endl` can be employed for this purpose. The two statements:

```
cout<<"sin(1) = "<<sin(1)<<"\n\n";  
cout<<"1.5 * 2.9 = "<<1.5 * 2.9;
```

are equivalent to the following statements:

```
cout<<"sin(1) = "<<sin(1);  
cout<<"\n\n1.5 * 2.9 = "<<1.5 * 2.9;
```

If we have the statements:

```
cout<<"sin(1) = "<<sin(1);  
cout<<"1.5 * 2.9 = "<<1.5 * 2.9;
```

then, the malformed output

```
sin(1) = 0.8414711.5 * 2.9 = 4.35
```

is the result. Examine it for yourself!

The programmer may want to provide additional comments of the program for the user within the program which should not participate in the process of executing the program. There are two ways to write comments. Line comments come after two forward slashes `//` up to the end of the line. On the other hand, block comments start with a forward slash and an asterisk `/*` and terminated with an asterisk and a forward slash `*/`. Further, block comments can extend across as many lines as needed. You can find examples for the use of such comments in Program P2_2. It is noteworthy that the purpose of using comments is for the transparency of the program so that if anyone else reads the program, they can easily understand what the program is about and what the details of the program are.

2.2.2 Input statement

So far, we found how the constants are declared and determined, as well as how to declare the variables. There are three ways of determining a value for the variables.

- 1) When declaring the variables, for example the part

```
int x, y = 10;
char c1 = 'a', c2 = '1';
```

declares the two variables *x* and *y* as integer types and puts the value 10 in *y*. Then, it declares the variables *c1* and *c2* assigning the character a and 1 to these variables, respectively. Bear in mind not to confuse the character 1 with the number 1.

2) After declaring the variables using the assignment operator (statement) with the syntax below:



Syntax (assignment statement):

a variable = an arithmetic phrase;

For example,

```
int u, v;
char k, s;
u = 0;
v = 0;
k = '?'; s = '*';
```

The two assignment statements:

```
v = 0;
u = 0;
```

are equivalent to the frequent assignment statement

```
u = v = 0;
```

In any frequent assignment statement, the numbers are assigned from right to the left.

The assignment statement, in addition to determining values for the variables can be used for calculation and substitution. For instance, the statement

```
i = i+1;
```

adds 1 to the last value of *i* in the memory, and substitutes the result for *i*.



The two expressions $k = 2$ and $2 = k$ are equal in theoretical studies, while they are completely different in programming. The statement $k = 2$ specifies an assignment statement whereas $2 = k$ is illegal. Furthermore, the statement $u = \sin(h)$ is an

assignment statement while $\sin(h) = u$ is illegal.

3) Using the `cin` (input) statement, which is used in the general syntax below:



Syntax (input statement):

```
cin >> variable 1 >> variable 2 >> ... ;
```

For example, in the part

```
float a, b;  
cin >> a >> b;
```

first the variables a and b are declared as real types. Then, the next statement reads their values from the input unit, respectively, and saves the values for these variables in the memory. The input unit is assumed to be the keyboard.



When entering the inputs, separate two consecutive inputs by one or more space characters. Moreover, never use a comma, semicolon, or other separators to separate the inputs.

After inputting the necessary inputs, we push the Enter key, denoted by the \leftarrow symbol, in order to continue running the program. Then, after entering all the inputs of the concerned `cin` statement, and before pressing the Enter key, we can promptly input the data related to the next `cin` statement in the current line if there is another `cin` statement in the program. Of course, another way is to press the Enter key to complete the process of inputting the data of the former `cin` statement and then, place the data of the latter `cin` statement in the next line. Generally, the data of one or more input statements can be placed in one or more lines. For example, if we assume that a , i , and j are all declared as `int` types, based on the above discussion, upon executing the two statements

```
cin >> a;  
cin >> i >> j;
```

the data can be entered in any of the following four ways:

4 5 -6 \leftarrow	4 \leftarrow	4 5 \leftarrow	4 \leftarrow
	5 -6 \leftarrow	-6 \leftarrow	5 \leftarrow
			-6 \leftarrow

Therefore, the two above-mentioned statements are supposed to be equivalent to the statement

```
cin>>a>>i>>j;
```

and its inputs behave the same. For example, consider the program below.

```
// Program P2_3: A simple program with various statements
#include <iostream>
#include <math.h>
using namespace std;
int main() {
    const double pi = 3.14;
    const double e = 2.7182;
    const char star = '*';
    int a, b, c, d;
    double u;
    cout<<"This is a simple program in C++"<<endl;
    cout<<pi<<" isn't an integer"<<endl;
    cout<<"Enter two integers: ";
    cin>>a>>b;
    u = sqrt(a / b);
    cout<<star<<"\t"<<u<<floor(u)<<endl;
    cout<<"exp(1) = "<<exp(1)<<" , e = "<<e<<"\n\n";
    cout<<a / 10<<(double) a / 10<<endl;
    cout<<"Enter two integers: ";
    cin>>c>>d;
    cout<<c<<d<<c * d;
    return 0;
}
```

The explanations related to running Program P2_3 for some of the inputs and outputs resulting from such programs are provided in the following passage (recall that the sign ← stands for pushing the Enter key). By running the program, the following lines are displayed in the screen:

```
This is a simple program in C++
3.14 isn't an integer
Enter two integers: 13 4←
```

In fact, after declaring the constants and variables, the output statement prints the string “This is my first simple program in C++” in the list of the cout statement and the cursor is transferred to the next line with the endl expression. In the new line, the constant value π is

printed, followed by the string “_isn't an integer”. Again, the cursor is transferred to the next line. In this line, at first, the input notification “Enter two integers:_” is written to request the user to enter two integers. Next, the user inputs two integers, for example, 14 and 3, respectively, separated by a space, and presses the Enter key. Now, the integers 13 and 4 are stored in the memory for the int type variables a and b , respectively.

Let's continue running the program. The next statement is an assignment statement in which first, the expression a / b is calculated in the argument of the library function `sqrt()`. In this expression, the result is 3 which is the integer quotient of a over b since both of the operands of $/$ are integers. Now, the square root of this number (i.e., 3) is assigned to the real variable u . The next output statement first prints the `*` character. Then the control character `\t` shifts the cursor one tab right (the eighth column) and prints two items in its list without any spacing:

```
*_____1.732051
```

After the control character, the first and second items are the value 1.73205 of u , and the round down of u which is 1, respectively. The fifth item in the list of `cout` statement is `endl` which transfers the cursor to a new line.

Clearly, the outputs are mixed together which is definitely not good. In the next subsection, this problem is fixed with formatted outputs and then the designing of the outputs is delivered to the programmer.

We continue the running process. The next `cout` statement prints the following five items in its list: the output heading “`exp(1)_=_`”, the value 2.71828 of `exp(1)` (with the default five digits precision), the output head “`,_e_=_`”, and the constant data 2.7182 declared for the constant e . In addition, the last item in the list of the present `cout` statement, namely “`\n\n`”, creates an empty line. The output explained in the present paragraph is:

```
exp(1)_=_2.71828,_e_=_2.7182
_
```

In the next `cout` statement, the two values 1 and 1.3 are combined together and written as:

```
11.311.3
```

The first one is considered the integer quotient of a by 10 while the second one is the real value of a divided by 10. Recall that we should write the (double) prefix in order to have the real value of a division in which the two operands of the $/$ operator are integers. The third item, `endl`, takes the cursor to a new line.

In the continuation of the program, the input notification “Enter two integers:_” requests the user to input two integers. Input 4 and press the Enter key. The running is incomplete. Even if we press the Enter key several times, there is no result since we have not yet entered the second input. Now enter 8:

```
Enter two integers: 4↵
↵
↵
8↵
```

By pressing the Enter key, the values 4 and 8 substitute for the integer variables c and d , respectively. Then, the cout list, which is the values of the given inputs and their multiplication, is written in succession without any spacing:

4832

Now, the running process is completed. When coding (translating) an algorithm to a language, we strongly recommend considering the notes below. These notes are applied in Program P2_3.



- Take the **types** of the variables proportionate to the types in the algorithm.
 - Provide suitable **input notifications**, before the input statements using the cout statement. This helps the user to know the number, type, and the essence of the data which are entered.
 - Provide appropriate **output headings** for the outputs to present their features.
-

2.2.3 Formatted output

When running Program P2_3, it was found that several outputs were mixed together and appeared in an ugly form. The outputs in C++ can be designed in various forms called **formats** which we will only refer to the common ones.

- 1) The required width for an output can be provided by the setw(n) format in which n is an appropriate integer or an integer variable, the value of which is already stored in the memory.

Effect: The output next to the setw(n) is written in n columns with this format.

If this width for the intended output,

- is more than that of the required output, then, we have extra blank columns on the left side of the output.
- is less than the width of the required output or it is an illegal number such as a negative number or a number larger than what is considered for the format setw(), it is ignored and thus, the output is written in a minimum number of columns.

- 2) The decimal precision can be provided by the format setprecision(n), where n is replaced by an appropriate integer or an integer variable, for which an appropriate value is already defined.

Effect: The decimal numbers are written with n precision digits which includes the number of valuable decimal digits. If the value of n for the intended output,

- is more than the width of the required output, extra number of valueless decimal

digits is added from the rightmost columns. Depending on the programming language version, these valueless decimal digits can be zero or other digits which when rounded up, leading to the original number.

- is less than the width of the required output, the intended number is rounded up in the case where the number of integer digits is not less than n ; otherwise it is written in the floating-point form “a E b”, that is, $a \times b^{10}$.
- is an illegal number such as a negative number or a number larger than what is considered for the format `setprecision()`, it is ignored and the output is written with the default precision defined for the format.



By default, the float and double data types are written with six and ten valuable digits, respectively. Note that these numbers may differ depending on the version of the compiler.

To use the two above-mentioned formats, the header `iomanip. h` should be loaded by the following pre-processor directive.

```
#include <iomanip.h>
```

In Program P2_4 below, we run the same statements as in Program P2_3 with the same inputs while with certain formats in order to observe the effect of the above formats. You can further examine the above-mentioned formats by varying the numbers in the arguments of the formats.

```
// Program P2_4: the previous program with formatted outputs
#include <iostream>
#include <math.h>
#include <iomanip>
#include <conio.h>
using namespace std;
int main() {
    const double pi = 3.14;
    const double e = 2.7182;
```



```

const char star = '*';
int a, b, c, d;
double u;
cout<<"This is a simple program in C++"<<endl;
cout<<pi<<" isn't an integer"<<endl;
cout<<"Enter two integers: ";
cin>>a>>b;
u = sqrt(a / b);
cout<<setw(2)<<star<<"\t"<<setw(3)<<u<<" "<<floor(u)<<endl;
cout<<"exp(1) = "<<setprecision(17)<<exp(1)<<setprecision(5)
    <<", e = "<<e<<"\n\n";
cout<<a / 10<<setw(6)<<(double) a / 10<<setw(3)<<star<<endl;
cout<<"Enter two integers: ";
cin>>c>>d;
cout<<"c: "<<c<<", d: "<<d<<", c*d: "<<c*d;
return 0;
}

```

input/output:

```

This is a simple program in C++
3.14 isn't an integer
Enter two integers: 13 4
_*1.73205
exp(1)=2.7182818284590451, e=2.7182
1
1.3_*
Enter two integers: 4 8
c: 4, d: 8, c*d: 32

```

It is worth mentioning that, unfortunately, the `setprecision()` format continues to the subsequent real numbers until another `setprecision()` format is notified. Several systematic methods exist to deactivate it, however, a simple way is to active `setprecision(5)` since the default precision is 5 in most C++ compilers.

The object-oriented aspect of the two programming languages C++ and Java are synchronously explained in [Section 3.3](#). We end this chapter with the concept of pointers.

2.3 Pointers

Actually, a variable is a name for a piece of memory which holds its value. When the program instantiates a variable, a free memory address is automatically assigned to the variable and any allocated value is stored in this memory address. For example, when the statement:

```
int x;
```

is executed, a location of memory is associated with the variable *x*. It is noteworthy that the program needs not to care about the physical address of the data in the memory; it simply uses the identifier whenever it needs to refer to the variable. However, obtaining the address of a variable during the runtime is useful for a program in order to access the data cells which are at a certain location relative to such variable.

In general, the address of a variable can be obtained by preceding the name of a variable with an **&** symbol known as the **address-of operator**. For example, the statement:

```
x = &var;
```

assigns the address of the variable *var* to *x*; using the above statement, we no longer assign the content of the variable itself to *x* while its address should be assigned in the memory.

Obtaining the address of a variable is not very useful by itself. The ***** **dereference operator** allows us to access the value at a particular address.

Having the address-of and dereference operators added to our toolkits, we can now discuss the pointers. A **pointer** is a variable which holds a memory address as its value. Further, it is one of the flexible and strong facilities of C++ programming language which is not supported by Java in which the other facilities are used instead. Furthermore, the pointer variables in C++ are declared similar to the normal variables with an asterisk between the data type and the variable name in either of the following three alternative ways:

```
data type*_variable or function;
```

```
data type_*variable or function;
```

```
data type_*_variable or function;
```

For example, the statement:

```
int *x;
```

(read “integer *x* pointer” which is a commonly used shorthand for “a pointer to an integer *x*”) declares *x* as a pointer to the int data type. The data type is not the type of the pointer itself, instead, it is the type of the data which the pointer points to. In fact, **x* returns the value located at the address specified by *x*.



- When declaring a pointer variable, put the asterisk next to the name of the variable.
 - When declaring a function, put the asterisk of a pointer return value next to the type.
-

For instance,

```
int *s, i, *x = &var;
double* getArea();
```

To clarify the above discussion, consider the following program.

```
// Program P2_5 to apply pointers
#include <iostream>
using namespace std;
int main () {
    int var = 20, *x;
    x = &var;
    cout<<"Value of var variable: "<<var<<endl;
    cout<<"Address stored in x variable: "<<x<<endl;
    cout<<"Value of * x variable: "<<*x<<endl;
    return 0;
}
```

Output:

```
Value of var variable: 20
Address stored in x variable: 0x6ffe34
Value of *x variable: 20
```

In the above program, first, the actual variable *var* (with the initial value of 20) and the pointer variable *x* are declared, which are both of int type. Then, the address of *var* is stored in *x*. Finally, the real value of the variable *var*, the address of *var* stored in the variable *x*, and the value in this address of the memory are printed, respectively. It is worth mentioning that the address mentioned in the output may vary depending on the used machine.

This section is finished with the fact that a pointer to a const value, known as the **const pointer**, is a pointer which points to a constant value. The **const** keyword is used before the data type in order to declare a const pointer. For example, consider the two statements below:

```
const int con = 5;  
const int *c = &con;
```

The first statement declares the constant *con* of int type while the second one stores its address in the const pointer *c*. However, the following statement is not allowed since we cannot change a constant value:

```
*c = 6;
```

3 Fundamental concepts of programming in Java

Java is the brainchild of Java pioneer James Gosling, who traces Java's core idea of, "Write Once, Run Anywhere" back to the work he did in graduate school. James Gosling along with other teammates, namely, Mike Sheridan and Patrick Naughton (called as the 'Green' Team) initiated Java language project for Sun Microsystems for digital devices such as set-top boxes, televisions, and the like in June 1991. Over time, the team added features and refinements which extended the heirloom of C++ and C, resulting in a new language called 'Oak', named after a tree outside Gosling's office, until it was discovered that a programming language named Oak already existed. Thus, the name was altered to 'Green', the name of their team. As the story goes, after many hours of searching for a new name, the development team went out for a coffee and the name Java was born. In fact, Java was a nickname selected to specify the coffee originated from the small Indonesian island called Java.

Java is a high-level popular and general-purpose programming language and computing platform. In fact, this programming language is known since it is easy to use, fast, object-oriented, robust, platform independent, multi-threading, secure, portable, and highly efficient. According to Oracle, the

company which owns Java, Java runs on more than 3 billion devices worldwide.

In addition, Java programming is considered an extremely diverse language and is used for a variety of purposes such as developing desktop and web (i.e., [Linkden.com](#), [Snapdeal.com](#) and the like) applications, enterprise, mobile operating system (i.e., Android), navigation systems, e-business solutions, smart cards, robotics, games, and so on. Actually, Java is applicable everywhere!

All the revisions of Java and their release dates are summarized in [Table 3.1](#).

Tab. 3.1: Revisions of Java.

Version	Release date	Version	Release date
JDK Beta	1995	Java SE 6	December 2006
JDK 1.0	January 1996	Java SE 7	July 2011
JDK 1.1	February 1997	Java SE 8 (LTS)	March 2014
J2SE 1.2	December 1998	Java SE 9	September 2017
J2SE 1.3	May 2000	Java SE 10 (18.3)	March 2018
J2SE 1.4	February 2002	Java SE 11 (18.9 LTS)	September
J2SE 5	September 2004	Java SE 12 (19.3)	March 2019

The older version of Java SE 8 (LTS) is still supported. Further, the Java SE 11 (18.9 LTS) is the latest version followed by the future release, namely, the Java SE 12 (19.3).

Of course, the information is updated in accord with the time of writing the present book. There are large amounts of valuable concepts in Java. However, in the current chapter, only the

necessary fundamental concepts of elementary programming introduced, which are needed for the intended purpose we have in mind regarding the programming for the mathematical calculations. For extensive studies the reader is referred to the standard books and websites.

3.1 Primary concepts

In this section, the primary concepts of Java programming language are discussed and compared to the C++ programming language. The definition of all the primary concepts of C++, previously provided in [Section 2.1](#), are valid in Java. Therefore, we only explain the reserved words, data types, operators, and library functions, as well as declaring variables and literals (constants in C++) in Java. We start with the reserved words or the keywords as the alphabet of Java. Some of the frequently used reserved words in Java are represented in [Table 3.2](#). Those in bold are the ones which are utilized in the present book.

Tab. 3.2: Some frequently used reserved words or keywords in Java.

abstract	assert	boolean	break	byte	case
catch	char	class	const	continue	default
double	do	else	enum	extends	false
final	finally	float	for	goto	if
implements	import	instanceof	int	interface	long
native	new	null	package	private	protected
public	return	short	static	strictfp	super
switch	synchronized	this	throw	throws	transient
true	try	void	volatile	while	

Similar to the C++ compiler, the Java compiler is case-sensitive and therefore, the behaviours of For and for are different in both compilers.

Although goto and const are no longer used in the Java programming language, they are still cannot be used as identifiers.

The rule of naming identifiers in Java slightly differs by C++.



The rule of naming the identifiers in Java.

- All the identifiers should begin with either a letter 'a' to 'z' or 'A' to 'Z', \$ symbol or an underscore.
 - After the starting character, an identifier can have any combination of the above-mentioned characters.
 - The whitespace character cannot be employed in an identifier.
 - A Java keyword cannot be utilized as an identifier.
-

3.1.1 Data types

The Java language has a rich implementation of data types which specify the size and type of values that can be stored in an identifier. In Java, data types are classified into primitive and non-primitive (reference) categories. The reference data types are discussed in [Section 3.3](#). Primitive data types contain four integer data types including byte, short, int, long, as well as two floating point data types float and double, character data type char, Boolean data type boolean, and finally, the void data type void. A brief description of each data type is presented as follows.

- 1) Integer data type **byte** is used to save memory when dealing with a large number of integers which are in the range of this data type ([Tab. 3.3](#)). This is because a byte data type occupies the least space of the memory needed for a numerical data type, which includes eight bits (one byte).
- 2) Integer data type **short** is utilized to save memory for the integers lying in the range of this data type ([Tab. 3.3](#)). The space in memory which a short data type occupies is 2 times larger than that of byte.
- 3) Integer data type **int** is employed for integers of normal size. Furthermore, an int data type in the memory is 4 and 2 times larger than a byte and a short, respectively.
- 4) Integer data type **long** is applied when a wider range is needed compared to the int. It needs an eight-byte space to be stored in the memory.
- 5) Floating point data type **float** is used for real numbers with

less precision.

- 6) Floating point data type **double** is utilized for real numbers with high precision.
- 7) Character data type **char** is employed to store a Unicode character in 2 bytes of the memory while the C++ language uses 1 byte to store the char data types. This is because C++ supports only ASCII codes for character data types which includes only the English letters and symbols and to do this 1 byte is sufficient. However, Java language supports the Unicode characters (i.e., letters and symbols) of more than 18 international languages and 1 byte of memory is insufficient for storing all these characters. Moreover, each character data in Java, compared to C++, should be written between two apostrophes such as '*' or the whitespace character ' ' which is demonstrated as '␣' whenever an emphasis is required or if a failure to its expression causes confusion.
- 8) Boolean data type **boolean** represents only one bit of information for either true or false. The corresponding boolean types in C++ are 1 and 0, respectively.

In this book, mainly the int, float, and double data types are used whereas the long and char data types are occasionally employed. However, the boolean data types rarely utilized. Java fails to support the implementation of unsigned integers while C++ supports it. **Table 3.3** demonstrates the size, range, and default values of these data types in which the floating-point notation 'a E b' means $a \times 10^b$.

Tab. 3.3: Primitive data types and their size, range, and default values.

Type	Size (bytes)	Range	Default
byte	1	-128 to 127	0
short	2	-32,768 to 32,767	0
int	4	-2,147,483,648 to 2,147,483,647	0
long	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0
float	4	$\pm 1.40129846432481707 \text{ E } -45$ to $\pm 3.40282346638528860 \text{ E } +38$	0.0
double	8	$\pm 4.94065645841246544 \text{ E } -324$ to $\pm 1.79769313486231570 \text{ E } +308$	0.0
char	2	0 to 65535 ('\u0000' to '\uffff')	'\u0000'
boolean	1 bit	false or true	false

Besides the above-mentioned basic data types, there are valueless data type void and string data type String in Java. A string is often a combination of any Unicode characters placed between a pair of quotation marks (e.g., "Note that $10 \% 3 = 1$ "). Formatted prints are considered one of the most common applications of the strings which is discussed in [Section 3.2](#).

3.1.2 Literals and variables

Unlike C++, we have no constants in Java, instead, literals play the role of the constants. A **literal** is a source code representation of a fixed value which is assigned to a variable in a program. Literals occur in the following four types.

- 1) String literals are enclosed in double quotes. For example, the statement

```
String s = "this is a string";
```

declares a string variable named *s* and assigns it the string literal inside the double quotes.

- 2) Character literals are enclosed in single quotes and contain only one character. For instance,

```
char star = '*';
```

- 3) Boolean literals which are either true or false. Unlike C++, these values fails to correspond to 1 and 0.

- 4) Numerical literals can contain either integer or floating-point values. For example,

```
float e = 2.8271;
```

A **variable** is an identifier associated with a value which can be modified during the program. Each variable in a program should have a type. Variables in Java are declared by the same syntax as in C++.



Syntax (variable declaration):

A data type _ one or more variables ;

or

A data type _ one variable = a literal ;

where, the literal data type is identical to that of the corresponding variable. For instance, consider the codes:

```
int a,b;  
float pi = 3.14;
```

The first line declares the `int` variables *a* and *b* to the compiler while the second line declares the variable *pi* of float type and assigns it the literal 3.14.

Java includes local, instance, and class variables. These types of variables are examined in [Section 3.3](#).

3.1.3 Operators

Java programming language uses the same arithmetic, relational (comparative), logical, and assignment operators which C++ apply, and their behaviours are highly comparable. In a number of cases, Java operators exert different results compared to their C++ equivalents. The division `/` operator generates an exception if we divide an integer by zero. For instance, `1.0/0.0` causes positive infinity (Infinity on the screen) while `-1.0/0.0` leads to negative infinity (-Infinity on the screen). Additionally, `1.0/(-1.0/0.0)`, which is 1 over negative infinity and `0.0/0.0`, which is arithmetically undefined, cause in minus zero (`-0.0` on the screen) and the not-a-number value (NaN on the screen), respectively.

As shown, Java defines positive and negative zeroes and infinities, as well as not-a-number values which indicate different habits as an operand in logical expressions. For example, the logical expression of `0.0 == -0.0` is true whereas `0.0 > -0.0` is false. Since NaN is unordered, all the comparison

operators return false if either operand is NaN, except for != which always returns true if either operand is NaN.

The results of some (mainly binary) calculations concerned with the above-mentioned values are represented in [Table 3.4](#). In this table, the results are what appear on the screen. The variable p is assumed a positive real number which can be replaced in the program calculations by, say, 1.0. In addition, the variable inf is supposed to be the infinity. This variable may be replaced by 1.0/0.0 in the program calculations.

Tab. 3.4: Abnormal calculations in Java.

Expression	Result in Java	Expression	Result in Java	Example	Result in Java
$(\pm 0.0)(\pm 0.0)$	0.0	$\sqrt{-p}$	NaN	$\pm p / \pm inf$	0.0
$(\pm 0.0)(\mp 0.0)$	-0.0	$\sqrt{-0.0}$	-0.0	$\pm p / \mp inf$	-0.0
$(\pm 0.0)(\pm p)$	0.0	$\pm 0.0 / \pm 0.0$	NaN	$\pm inf / \pm 0.0$	Infinity
$(\pm 0.0)(\mp p)$	-0.0	$\pm 0.0 / \mp 0.0$	NaN	$\pm inf / \mp 0.0$	-Infinity
$(\pm 0.0)(\pm inf)$	NaN	$\pm 0.0 / \pm p$	0.0	$\pm inf / \pm p$	Infinity
$(\pm 0.0)(\mp inf)$	NaN	$\pm 0.0 / \mp p$	-0.0	$\pm inf / \mp p$	-Infinity
$(\pm inf)(\pm inf)$	Infinity	$\pm 0.0 / \pm inf$	0.0	$\pm inf / \pm inf$	NaN
$(\pm inf)(\mp inf)$	-Infinity	$\pm 0.0 / \mp inf$	-0.0	$\pm inf / \mp inf$	NaN
$(\pm p)(\pm inf)$	Infinity	$\pm p / \pm 0.0$	Infinity	$inf - inf$	NaN
$(\pm p)(\mp inf)$	-Infinity	$\pm p / \mp 0.0$	-Infinity		

The ? and , (comma) operators have the same actions in C++ and Java programs. The sizeof() operator is not defined in Java since the size of primitives in Java is fixed in all platforms.

Further, the arithmetic and logical expressions have similar behaviours in C++ and Java programs. In particular, in Java, we follow the same priority for the operators as in C++. Several library functions which are mainly used in Java are

summarized in [Table 3.5](#) analogous to [Table 2.12](#) with minor differences.

Tab. 3.5: Some library functions supported by Java.

Function	Argument(s) type	Result type	Effect
Math.sin(x)	double	double	$\sin(x)$
Math.cos(x)	double	double	$\cos(x)$
Math.tan(x)	double	double	$\tan(x)$
Math.sinh(x)	double	double	$\sinh(x)$
Math.cosh(x)	double	double	$\cosh(x)$
Math.tanh(x)	double	double	$\tanh(x)$
Math.asin(x)	double	double	$\sin^{-1}(x)$
Math.acos(x)	double	double	$\cos^{-1}(x)$
Math.atan(x)	double	double	$\tan^{-1}(x)$
Math.exp(x)	double	double	e^x
Math.log(x)	double	double	$\ln(x)$
Math.log10(x)	double	double	$\log_{10}(x)$
Math.pow(x,y)	double	double	x^y
Math.sqrt(x)	double	double	\sqrt{x}
Math.abs(x)	int, long, float, double	same type	$ x $
Math.ceil(x)	double	double	round up x
Math.floor(x)	double	double	round down x
Math.max(x,y)	int, long, float, double	same type	maximum of x and y
Math.min(x,y)	int, long, float, double	same type	minimum of x and y
Math.PI			3.14159265358979323846
Math.E			2.7182818284590452354

3.2 Introduction to programming in Java

Installing and using Java depends on the user's computer, which is the reader's responsibility. The programming

techniques are discussed in this book. All Java programs in the present book are run in **eclipse IDE 2018-09** workspace.

We start with a Java program pattern.



A Java program pattern:

```
package P3;
specifier class class name {
    /*
    * line comments
    */
    body of program

    public static void main(String[] args) {
        main body
    }
}
```

As before, the italic phrases in the program codes are not parts of the codes per se and what is required by these phrases is placed for them.

The **package** statement defines a namespace in which classes are stored. The package is used to organize the classes based on functionality. Suppose it as a folder in a file directory. These packages are employed to avoid the name conflicts and to write better maintainable code. However, the package statement cannot appear anywhere in the program. In fact, it should be in the first line of the program or it can be omitted in which case, the class names are put into the default package which has no name.

Further, packages in Java can be divided into built-in and user-defined categories. There exist various built-in packages such as java, lang, awt, javax, swing, net, io, util, sql, and the like while the user-defined packages are written by the programmer for specific purposes. In the current book, we use no package in our programs.

A C++ program was a sequence of separate fragments containing declarations, statements and functions (subprograms) altogether under the control of a main function which results in running the program. Contrarily, a Java program is often gathered in a unit class including separate data members and member functions (methods) which form the body of program in the above pattern. Here again, a main method (the void main() method in the above pattern) results in running the program.

Actually, a method is a compound statement which includes a sequence of statements and declarations within a pair of curly braces. An expression followed by a semicolon is considered a statement. The other types of statements are expression, selection (decision and branching), iteration (loop), and jump statements. A number of other types of statements are addressed in **Chapters 4, 6, and 7**. Moreover, various types of subprograms including the methods are dealt with in **Chapter 5**. The classes and their various aspects along with data members and methods are discussed in details in the next section.

Similar to C++ programs, there are two ways for writing the comments. Line comments come after two forward slashes // up to the end of the line. However, block comments begin with a

forward slash and an asterisk `/*` and terminate with an asterisk and a forward slash `*/`. Additionally, these comments can extend across as many lines as needed.

It is strongly recommended to consider the programming styles at the beginning of [Section 2.2](#). The output and input statements are regarded as the basic handling statements in all programming languages.

3.2.1 Output and input statements

We start with the output statement. Consider Program P3_1 (compare to Program P2_1).

```
// Program P3_1 the first program in Java
public class P3_1 {
    public static void main(String[] args) {
        System.out.print("Welcome to Java programming world");
    }
}
```

This simple program, having P3_1 class name, contains only the main body which is an output statement. This statement prints the string inside the double quotes:

Welcome to Java programming world

In general, there are two normal (unformatted) print statements.



Syntax (print statement):

```
System.out.print(armument 1 + armument 2 + ...);
```

```
System.out.println(armument 1 + armument 2 + ...);
```

where, each of the arguments may be one of the following items:

1. Constant, variable, arithmetic expressions, or any other kind of expression;
2. A string, for instance, "The sum is: ". In this case, the exact phrase between the pair of quotation marks is printed, as:

The sum is:␣

3. The escape sequences or the control characters are provided in **Table 3.6**. These characters should be placed between a pair of quotation marks, for example, "\t";

Tab. 3.6 The frequently used escape sequences or control characters.

escape sequence	name	effect
\n	new line	move to the beginning of the next row
\t	horizontal tab	move to the beginning of the next eight column (next tab)
\b	backspace	back up one character removing it
\r	carriage return	move down a line
\f	form feed	move the next page
\"	double quote	write the " character
\'	single quote	display the ' character

Note that the two consecutive arguments in the print statement are separated by a + operation. A space is used before and after this separator to distinguish the successive arguments. Nevertheless, this may cause a conflict, particularly, when we have two consecutive arithmetic expressions in which case the + operation arithmetically adds these expressions. Therefore, we can separate such expressions with the null argument "" or spaces in order to prevent this inconvenience. The three examples along with their outputs in [Table 3.7](#) clarify this fact.

Tab. 3.7: Use the null argument "" or spaces between two consecutive arithmetic expressions.

Statement	Output
<code>System.out.print(1 + 2);</code>	3
<code>System.out.print(1 + "" + 2);</code>	12
<code>System.out.print(1 + " " + 2);</code>	1 2

The difference between the two above-mentioned printing statements is that the `println()` statement positions the cursor onto the next line after printing the desired arguments while the `print()` statement leaves the cursor on the current line. As a result, the escape character "\n" has the same effect as `println()` if we put it as the last argument of the `print()` statement. However, these two print statements have their own applications.

There exists no pre-processor directive in Java. However several head statements are contributed for certain purposes.

Consider Program P3_2 analogous to Program P2_2 in which the mathematical sine function is invoked.

```
// Program P3_2 to use mathematical functions
public class P3_2 {
    public static void main(String[] args) {
        System.out.print("sin(1) = " /* string */ + Math.sin(1.0) + "\n\n");
        System.out.print("1.5 * 2.9 = " + 1.5 * 2.9);
    }
}
```

Output:

```
sin(1) = 0.8414709848078965
_
1.5 * 2.9 = 4.35
```

The same explanations after Program P2_2 are established here. In particular, the expression “\n\n” is used to create an empty line, and the following three pairs of statements are equivalent:

```
System.out.print("sin(1) = " + Math.sin(1.0) + "\n\n");
System.out.print("1.5 * 2.9 = " + 1.5 * 2.9);
```

```
System.out.print("sin(1) = " + Math.sin(1.0));
System.out.print("\n\n1.5 * 2.9 = " + 1.5 * 2.9);
```

```
System.out.println("sin(1) = " + Math.sin(1.0) + "\n");
System.out.print("1.5 * 2.9 = " + 1.5 * 2.9);
```

That is because, as mentioned above, the `print("\n")` is the same as `println()`.

We continue with the input statement. There are several ways to obtain input from the input unit. Here, the common

input statements are employed. In Java, four steps should be followed to read the inputs from the input unit:

1. Importing the Scanner class by the head statement:

```
import java.util.Scanner;
```

at the top of the program before starting the class;

2. Providing the input facility for the Java compiler by the statement:

```
Scanner scan = new Scanner(System.in);
```

often at the beginning of the class body in which the inputs are to be read;

3. Reading the single input n (here with int data type) by the statement:

```
int  $n$  = scan.nextInt();
```

4. Closing the input facility by the statement:

```
scan.close();
```

often at the end of the class body in which the statement in item 2 is contained.

The scanner object *scan* in statements 2, 3, and 4 is selected by the programmer (The concepts “class” and “object” are discussed in the next section). Throughout the present book, the name *read* is applied for the numeric scanning in order to have a uniform scanner name. Then, having imported the Scanner

class (the head statement in item 1), we keep fixed using the statement

```
Scanner read = new Scanner(System.in);
```

to provide input facility; the statement

```
int n = read.nextInt();
```

to read the int type variable *n*; and the statement:

```
read.close();
```

to close the input facility. The aspects of these statements are not discussed any more. At present, we only use the above steps to read the inputs. With the statement

```
int n = read.nextInt();
```

the int type variable *n* is read. It is noteworthy that the variable *n* may not be individually declared any more. In addition, the same statements are used for other numeric data types. It suffices to replace “Int” by the data type name starting with an uppercase letter (e.g., nextShort(), nextDouble() and the like). Further, the reading of char type inputs is left to the next section.

The two above Programs P3_1 and P3_2 were analogous to Programs P2_1 and P2_2. The strategy of using the same programs in both C++ and Java languages provides the chance to compare the similarities and differences, along with synchronous learning of the details related to the aspects of the

programs. We continue our strategy in the following couple of examples.

As in C++ programs, there are three methods of storing a value for the variables in Java including: along with declaration, after the declaration by assignment statements, and using output statements. Furthermore, the way of entering the inputs in Java is the same as C++. In particular, inputs may be entered in one or more lines, and the two consecutive inputs should only be separated by a space. Consider Program P3_3 analogous to Program P2_3.

```
// Program P3_3: A simple program with various statements
import java.util.Scanner;
public class P3_3 {
    public static void main(String[] args) {
        Scanner read = new Scanner(System.in);
```



```

double pi = 3.14;
double e = 2.7182;
char star = '*';
double u;
System.out.println("This is a simple program");
System.out.println(pi + " isn't an integer");
System.out.print("Enter two integers: ");
int a = read.nextInt();
int b = read.nextInt();
u = Math.sqrt(a/b);
System.out.println(star + "\t" + u + Math.floor(u));
System.out.print("exp(1) = " + Math.exp(1) + ", e = " + e + "\n\n");
System.out.println(a/10 + "" + (double) a/10);
System.out.print("Enter two integers: ");
int c = read.nextInt();
int d = read.nextInt();
System.out.println(c + "" + d + "" + c * d);
read.close();
}
}

```

Some of the inputs and outputs resulting from the above program are provided in the following passage. Recall that the sign \leftarrow stands for pressing the Enter key.

By running the program, the following lines are displayed on the screen:

```

This is a simple program in Java
3.14 isn't an integer
Enter two integers: 13_4 $\leftarrow$ 

```

In fact, after declaring the literals and variables, the first output statement prints its string argument “This is my first simple program in Java” and then the cursor is transferred to the next line due to the `println()` type of output statement. In the new line, the constant value of π and, following it, the string “`_`isn't

an integer” is printed by the second print statement. One more, the cursor is moved to the next line. In this line, first, the input notification “Enter two integers:␣” is written to request the user to enter two integers using the println() statement which leaves the cursor in the current line. Next, the user inputs two integers, for example, 13 and 4, respectively, separated by a space, and presses the Enter key. Now, the integer numbers 13 and 4 are stored in the memory for the variables *a* and *b*, respectively.

Let’s continue running the program. The next statement is an assignment statement. In this statement, first, the expression a / b is calculated in the argument of the library function Math.sqrt(). In this expression, the result is 3 which is the integer quotient of *a* by *b* since both of the operands of / are integers. Now, the square root of 3 is assigned to the real variable *u*. The next output statement prints first the * character. Then, the control character \t shifts the cursor one tab right (the eighth column). Finally, the current print statement continuous printing the last two items in its list without any spacing:

```
*_____1.73205080756887721.0
```

The first item after the control character is the value 1.7320508075688772 of *u* and the second one is the round down of *u* which is 1.0. Then, the cursor is transferred to a new line.

As it is evident, the outputs are mixed together which is definitely not good. This problem is solved by formatted outputs

and the designing of the outputs is delivered to the programmer.

We continue the running process. The next output statement prints the five arguments which are mentioned below in its arguments list: first, the output heading “exp(1) $_$ = $_$ ”; second, the value 2.718281828459045 of exp(1) with the default 15 digits precision; third, the output heading “ , $_$ e $_$ = $_$ ”; fourth, the constant value 2.7182 which is previously declared for the literal e . The fifth and last item in the list of the present print statement, “\n\n”, creates an empty line. The output explained in the present paragraph is:

```
exp(1) $\_$ = $\_$ 2.718281828459045,  $\_$ e $\_$ = $\_$ 2.7182
 $\_$ 
```

In the next print statement, the two values 1 and 1.3 are mixed together and written as:

11.3

The first value is the integer quotient of a by 10 and the second one as the last argument is the real value of a divided by 10. The middle argument fails to affect anything while if we remove it, the result 2.3 is obtained which is $1 + 1.3$. Recall that we should write the (double) prefix in order to have the real value of a division in which the two operands of the / operator are integers.

In the continuation of the program, the input notification “Enter two integers: $_$ ” requests the user to input two integers. Input 4 and press the Enter key. The running is incomplete.

There are no results even if we press the Enter key several times since we have not yet entered the second input. Now, enter 8:

```
Enter two integers: 4↵  
↵  
8↵
```

By pressing the Enter key, the values 4 and 8 substitute for the int type variables *c* and *d*, respectively. Afterwards, the arguments of the next print statement, which is the value of the given inputs and their multiplication, is written in succession, without any spacing:

4832

Now, the executing process is completed. We strongly recommend considering the notes below when coding (translating) an algorithm in any programming language. These notes are applied in Program P3_3.

-
- Take the **types** of the variables proportionate to the types in the algorithm.
 - Provide suitable **input notifications** before input statements by print() statement. This helps the user to know the number, type and the essence of the data which would be entered.
 - Provide appropriate **output headings** for the outputs to present their features.
-

3.2.2 Formatted output

In Program P3_3, we experienced some mixed outputs together which frequently appear in an unacceptable form. Java programming language provides a variety of facilities, called **formats**, for the programmer to design the outputs. For the purpose of calculation, we only express commonly used numerical formats. We adopt using the italic brackets for optional items. In general, we employ the following syntax in order to have a formatted print.



Syntax (formatted print):

```
System.out.printf("format string" [, argument 1, argument 2, ... ]);
```

where, format string contains the string literals and format specifiers. Moreover, arguments are required only if there are format specifiers in the format string. Additionally, format specifiers appear in the following pattern:

```
%[flags] [width] [.precision] conversion character
```

The aspects of the above items are explained as follows.

Flags: Various flags are defined for the Java compiler out of which three of them are frequently used in the programs associated with the calculations.

- :left-justify (default is to right-justify);
- + : output a plus (+) or minus (–) sign for a numerical value;
- 0 : forces numerical values to be zero-padded (default is blank padding).

Width: Specifies the field width for outputting the argument and represents the minimum number of characters to be written to the output. Include space for the expected commas and a decimal point in the determination of the width for numerical values. The width is ignored if it is less than the minimum number of characters needed for the output.

Precision: Used to restrict the output depending on the conversion. It specifies the number of digits of precision when outputting floating-point values. Numbers are rounded to the specified precision. Finally, the valueless zeroes are padded to reach the declared precision if the precision is more than the real decimal digits of the output.

Conversion characters: These characters are:

- d : decimal integer (byte, short, int, long);
- f : floating-point number (float, double);
- c : character, capital C uppercases the letters;
- s : string, capital S uppercases all the letters in the string;
- \n : newline platform, specific newline character; use %n instead of \n for greater compatibility.

Several examples are presented in [Table 3.8](#)

Tab. 3.8: Examples of formatted outputs.

Format string	Argument	Result	Format string	Argument	Result
"%d"	123	123	"%.3f"	123.9876	123.988
"%+d"	123	+123	"%+.3f"	123.9876	+123.498
"%5d"	123	__123	"%10.3f"	123.9876	____123.988
"%+d"	123	_+123	"%5.3f"	-123.9876	-123.988
"%05d"	123	00123	"%.0f"	124.9876	125
"%3c"	'A'	__A	"%12.5f"	-124.9876	____-124.98760
"%s"	"Class"	Class	"%010.3f"	123.9876	000123.987
"%5S"	"Class"	_CLASS	"%06.1f"	123.9876	124.0

We run Program P3_3 with the same inputs while with certain formats in order to observe the effect of the above formats. You can further examine the mentioned formats by varying the numbers flags, width, precision, and conversion characters.

```
// Program P3_4: the previous program with formatted outputs
import java.util.Scanner;
public class P3_4 {
    public static void main(String[] args) {
        Scanner read = new Scanner(System.in);
        double pi = 3.14;
        double e = 2.7182;
        char star = '*';
        double u;
        System.out.println("This is a simple program in Java");
        System.out.println(pi + " isn't an integer");
        System.out.print("Enter two integers: ");
        int a = read.nextInt();
        int b = read.nextInt();
        u = Math.sqrt(a/b);
        System.out.printf("%2c %10.2f %.0f\n", star, u, Math.floor(u));
        System.out.printf("exp(1) = %.17f, e = %.4f\n\n", Math.exp(1.0), e);
        System.out.printf("%7s %.17f%5s %.4f\n\n", "exp(1) =",
            Math.exp(1.0), ", e =", e);
        System.out.printf("%d %6.2f %2c\n", a/10, (double) a/10, star);
        System.out.print("Enter two integers: ");
        int c = read.nextInt();
        int d = read.nextInt();
        System.out.printf("c: %d, d: %d, c*d: %d", c, d, c*d);
        read.close();
    }
}
```

input/output:

```
This is my first simple program in Java
3.14 isn't an integer
Enter two integers:_14 3 ←
_*_____1.73_1
exp(1)_=_2.71828182845904500,_e_=_2.7182
_
exp(1)_=_2.71828182845904500,_e_=_2.7182
_
1____1.40__*
Enter two integers:_4 8 ←
c:_4,_d:_8,_c*d:_32
```


As shown, in the above program the following two statements have the same effects. It is the readers responsibility to discover the details!

```
System.out.printf("exp(1) = %.17f, e = %.4f\n\n", Math.exp(1.0), e);  
System.out.printf("%7s %.17f%5s %.4f\n\n", "exp(1) =", Math.exp(1.0), ", e =", e);
```

Now, we are ready to synchronously explain the object-oriented aspect of the two programming languages C++ and Java.

3.3 Object-oriented programming (OOP) system

The C++ programming mainly aims to introduce the concept of object-orientation to the C programming language: C++ is equivalent to C+1 in programming codes which means one feature more than C, namely, “Class”. In fact, C++ encapsulates high and low-level language features. Therefore, it is considered as an intermediate level language. In addition, C++ allows both procedural and object-oriented programming. Using the object-orientation feature of C++ is not felt a necessity due to the calculation natures of our subprograms except for a few cases and thus, we prefer simplicity in the codes instead.

Contrarily, Java is a high-level, general-purpose, class-based, object-oriented programming language which is designed to minimize implementation dependencies.

An object is a real-world entity such as stone, glass, book, umbrella, bike and the like. Further, object-oriented programming is a methodology or paradigm for designing a program using the objects and classes. It simplifies software

development and maintenance by providing the properties including object, class, inheritance, polymorphism, abstraction, and encapsulation. We only discuss and use the notions of object and class since the present book seeks to adhere to mathematical calculations.

In general, by an object, we mean any entity which has state and behaviour where, state and behaviour mean data and functionality, respectively. In other words, the object-oriented approach is extremely close to the real world and its applications since the state and behaviour of these objects are nearly the same as the real-world objects. In the next subsection, the concepts of objects and class are synchronously dealt with in C++ and Java programming languages. The differences are explained whenever necessary.

3.3.1 Objects and class

A class is a blueprint of the objects, that is, a collection of similar objects. In other words, an **object** is an instance of a class. Furthermore, an object can be physical and logical while a class is only a logical entity. Finally, class is invisible to the world whereas the object is visible.

In general, a **class** is somehow a user-defined data which has the **elements** containing data members, member functions (methods), constructors, and the like. The following is the syntax for defining a class in C++ and Java with a minor difference.



Syntax (defining class):

```
class class name {  
    access specifier  
    data members  
    member functions  
}; // Unlike Java, a class terminates with a semicolon in C++
```

By class definition, we indicate to define a structure or a blueprint while not to exclusively define a data; that is, to what the objects of that class type contain and what operations can be performed on the objects. The following are examples of the class in C++ and Java.

C++ codes:

```
class Test {  
    private:  
        int data1;  
    public:  
        float data2, data3;  
        void function1() {  
            data1 = 1;  
        }  
        float function2(int k) {  
            data2 = k * k;  
            return data2;  
        }  
};
```

Java codes:

```
class Test {  
    private int data1;  
    public float data2, data3;  
    public void function1() {  
        data1 = 1;  
    }  
    public float function2(int k) {  
        data2 = k * k;  
        return data2;  
    }  
}
```

Based on the above discussion, defining a class in C++ terminates with a semicolon following the closed curly bracket '}'. However, in Java, a class ends without any semicolon.

Here, Test is a class which has three data members data1, data2, and data3, along with two member functions including function1() and function2(). A **data member** is a variable which is declared in any class by using any primitive data types, (e.g., int) or derived data type (e.g., user-named class data type).

A **member function**, which we refer to as a **method**, is a function which has its definition or prototype within or outside its class definition. A method operates on any object of the class of which it is a member, and has access to all the data members of a class for that object.

Every method in Java should be a part of a class which is different from that of C++. Methods are generally divided into built-in and user-defined categories. The built-in methods are part of the compiler package such as System.out.println() and System.exit(0). Hereafter, by a method, we mean a user-defined method.

The **access specifier** specifies the accessibility of an element. If it is considered as private, then the data members and methods in its range can be accessed only from inside the same class. The compiler throws an error if we attempt to access private data from outside of the class. Another access specifier is public, in which case, data members and methods are accessible from anywhere, inside or outside of the class. Moreover, there exists another access specifier named protected in which data members and methods can be accessed in the derived class or within the same class. The protected specifiers are not discussed and used in this book. The public specifier is mainly used in both languages C++ and Java.

One of the differences between C++ and Java in the object-oriented programming is that in Java a specifier is a modifier while it is written as a label in C++. This fact can be experienced in the above Test class. Another difference is in the default specifiers. In C++, the access modifier for an element is private if we specify no access modifiers by default for that element inside the class. However, the default specifier for Java programs is the private-package which implies that the element is accessible from inside the same package to which the class belongs.

The method function1() in the above examples has neither return value (of void type) nor has it any parameter. We may put the void keyword inside the parenthesis only in C++ programs if there is no parameter. Contrarily, the method function2() has float-type return value. Additionally, it has one parameter of int type. In general, a method can have any data type or it is void. In addition, it can be void of any parameter or it contains one or more parameters. No method can be defined outside the class in Java since it is a class-based programming language. However, in C++, a method can be defined outside the class using the :: (scope resolution) operator, as demonstrated in the following codes.

C++ codes only:

```
double Test::function3(double t) {
    data3 = t * t - 2 * t + 1;
    return data3;
}
```

In this case, this method should be declared inside the class as follows.

```
double function3(double);
```

There are two worthwhile notes related to the methods in C++ which need attention. First, the declaration is a statement with a semicolon at the end. Second, writing the name of the parameter in the declaration statement is optional. In the above declaration statement, only the data type of the parameter is written. Now, return to the synchronous approach.

When a class is defined, only the specifications of the objects is defined while no memory is allocated to the objects. Actually, memory is allocated to an object as soon as it is created with the following syntaxes in both languages.



Syntax (creating an object):

C++ codes:

```
class name_object name;
```

Java codes:

```
class name_object name = new_class name();
```

For example, the statements:

C++ codes:

```
Test obj1, obj2;
```

Java codes:

```
Test obj1 = new Test();  
Test obj1 = new Test();
```

creates two objects *obj1* and *obj2* of the class `Test`. As shown, the objects are created one by one in Java while in C++, the objects are created in groups as if they are in class data type.

We can access the data members and methods by using a `.` (dot) operator in both languages. For instance, the statement:

```
obj2.function1();
```

calls the `function1()` method inside the `Test` class for the objects *obj2*. Similarly, the data member *data2* can be accessed as:

```
obj1.data2 = 356.07;
```

Notice that, the private members can be accessed only from inside the class. Accordingly, you can write the statement

```
obj2.function1();
```

anywhere (inside or outside the class `Test`) in the above-mentioned examples. However, a code such as

```
obj1.data1 = 6.5;
```

should constantly be inside the `Test` class.

On the other hand, in the input statement defined in [Section 3.2](#), the `println()`, `print()`, and `printf()` are built-in methods which have already been defined for the compiler of Java aiming at printing their arguments. Furthermore, the `nextInt()` is a built-in method for reading the `int` type inputs. Therefore, when we write `read.nextInt()`, in fact, we call the `nextInt()` method in the `Scanner` class for the object *read*.

In **Sections 2.2** and **3.1**, various (primitive) data types were presented. There is another data type, named **reference data types**, which is any instantiable class type that stands for the objects of a class, and the array type which points to the arrays. The word "reference" is selected for these data types since they are handled "by reference"; in other words, the address of the object or array is stored in a variable, passed to methods, and the like. By comparison, primitive types are handled "by value"; the actual primitive values are stored in variables and passed to the methods.

Most of the above discussions are demonstrated in Programs P3_5.

C++ codes:

```
// Program P3_5 to illustrate objects
// and class in C++ Programming
#include <iostream>
using namespace std;
class Box {
private:
    char name;
public:
    float length, breadth, height;
    void setName(char name) {
        this->name=name;
        cout<<"Name of box set to : "
            <<name<<endl;
    }
    void setDimensions(float l, float b,
                       float h) {
        length=l;
        breadth=b;
        height=h;
    }
    char getName() {
        return this->name;
    }
    float getVolume() {
        return length*breadth*height;
    }
    float getSurface() {
        return 2*(length*breadth+breadth*height
                +height*length);
    }
};
```

Java codes:

```
// Program P3_5 to illustrate objects
// and class in C++ Programming
import java.util.Scanner;
class Box {
    private char name;
    public float length=0;
    public float breadth=0;
    public float height=0;
    public void setName(char name) {
        this.name=name;
        System.out.println("name of the box "
            + "set to " + name);
    }
    public void setDimensions(float l,
                              float b, float h) {
        length=l;
        breadth=b;
        height=h;
    }
    public char getName() {
        return this.name;
    }
    public float getVolume() {
        return length * breadth * height;
    }
    public float getSurface() {
        return 2*(length*breadth+breadth*height
            +height*length);
    }
}
//*****
```

```

//*****
int main() {
    Box B1, B2;
    char s1, s2;
    float l1, l2, b1, b2, h1, h2;
    cout<<"Enter the name of box1: ";
    cin>>s1;
    B1.setName(s1);
    cout<<"Enter the name of box2: ";
    cin>>s2;
    B2.setName(s2);
    cout<<"Enter length, breadth, height "
        <<"of "<<B1.getName()<<": ";
    cin>>l1>>b1>>h1;
    B1.setDimensions(l1, b1, h1);
    cout<<"Enter length, breadth, height "
        <<"of "<<B2.getName()<<": ";
    cin>>l2>>b2>>h2;
    B2.setDimensions(l2, b2, h2);
    cout<<"volume of box "<<B1.getName()
        <<": "<<B1.getVolume()<<endl;
    cout<<"surface of box "<<B2.getName()
        <<": "<<B2.getSurface()<<endl;
    return 0;
}

```

Input/output:

```

Enter the name of box1: A↵
Name of box set to : A
Enter the name of box2: B↵
Name of box set to : B
Enter length, breadth, height of A: 1 2 3↵
Enter length, breadth, height of B: 1 2 3↵
volume of box A: 6
surface of box B: 22

```

```

public class P3_5 {
    public static void main(String args[]) {
        Scanner read=new Scanner(System.in);
        Box B1=new Box();
        Box B2=new Box();
        char s1, s2;
        float l1, l2, b1, b2, h1, h2;
        System.out.print("Enter the name of box1: ");
        s1=read.next().charAt(0);
        B1.setName(s1);
        System.out.print("Enter the name of box2: ");
        s2=read.next().charAt(0);
        B2.setName(s2);
        System.out.print("Enter length, breadth, "
            + "height of " + B1.getName() + ": ");
        l1=read.nextFloat();
        b1=read.nextFloat();
        h1=read.nextFloat();
        B1.setDimensions(l1, b1, h1);
        System.out.print("Enter length, breadth, "
            + "height of " + B2.getName() + ": ");
        l2=read.nextFloat();
        b2=read.nextFloat();
        h2=read.nextFloat();
        B2.setDimensions(l2, b2, h2);
        System.out.println("volume of box "
            + B1.getName() + ": " + B1.getVolume());
        System.out.println("surface of box "
            + B2.getName() + ": " + B2.getSurface());
        read.close();
    }
}

```

Input/output:

```

Enter the name of box1: A↵
Name of box set to : A
Enter the name of box2: B↵
Name of box set to : B
Enter length, breadth, height of A: 1 2 3↵
Enter length, breadth, height of B: 1 2 3↵
volume of box A: 6.0
surface of box B: 22.0

```

In each of the above programs, five public methods are declared, along with the declaration of one private and three public data members as instance variables. The first public void method, setName(), receives a char type argument and print it following the message name of box set to:_. The second void

method, `setDimensions()`, receives its three float arguments *l*, *b*, and *h* and then assigns them to the instance variables *length*, *breadth*, and *height*, respectively. The two above-mentioned methods have parameters but no return. On the other hand, the three remaining methods have no parameters but return values. The third char-type method, namely `getName()`, returns the private instance variable *name*. Notice the use of the `this` keyword in this method (see [Section 3.3.6](#)). Finally, the float-type methods `getVolume()` and `getSurface()` return, respectively, the volume and surface of the associated boxes with dimensions *length*, *breadth*, and *height*, as instance variables. The running of the program is clear.

3.3.2 Types of variables

As in C++, a block in Java is a group of one or more statements enclosed in curly braces `{}`. For example,

```
{int i, j; i = 100; j = 200;}
```

A block is itself a type of statement. Variables in both C++ and Java languages include the following types.

- 1) Local variables. These variables are declared in methods, constructors (refer to the next subsection), or blocks. A **local variable** is created (memory allocated to them) when the involved method, constructor, or block is executed and disappeared once it terminates. Local variables are visible only within the method, constructor, or block in which the variable is declared. We cannot use access specifiers for

local variables. Moreover, a local variable fails to be defined with the static keyword ([Subsection 3.3.5](#)). In the following sample class, the local variable *age* is declared within a method.

C++ codes:

```
class Student {
public:
void StAge() {
int age=0;
age=age+5;
cout<<"Student age is: "<<age;
}
};
```

Java codes:

```
class Student {
public void StAge() {
int age=0;
age=age+5;
System.out.print("Student age is: "
+ age);
}
}
```

- 2) Instance (or non-static) variable. An **instance variable** is the one which is declared without the static keyword in a class outside any method, constructor, or block. They are called instance variables since they are object-specific and are not shared among the objects. In other words, an instance variable is created when an object of the class is created and disappeared when the object is disappeared. Unlike the local variables, we may use access specifiers for instance variables. The default access specifier is used if no access specifier is specified. The instance variables can be directly accessed by calling the variable name inside the class. In other words, these variables are visible for all the methods, constructors, and blocks in the class. However, instance variables should be called using the completely qualified name:

object name.variable name

when instance variables are given accessibility. For example, in Programs P3_6 below, the public float variable

InitialSum is an instance variable which is called in the main() methods for the object *s* in both programs.

- 3) Class (or static) variable. A **class variable** is declared with the static keyword in a class and common for all the objects of the class. There exists exactly one copy of this variable regardless of how many times the class is instantiated. Further details about the class variables are presented in [Subsection 3.3.5](#).

In C++, there exists another type of variable, called a **global variable**, which is declared in the main unit.

3.3.3 Constructors and destructors

A **constructor** is a special method which automatically initializes an object of its class when it is created. It is called constructor since it constructs the initial values of data members of the class. A constructor differs from a normal method in the following items:



- A constructor has the same name as the class itself;
 - A constructor has no return value;
 - A constructor is constantly public ;
 - A constructor is automatically called when an object is created;
-

There are three types of constructors including default, parameterized, and copy constructors. A **default constructor** is

the one which takes no parameter. There is a default constructor in Programs P3_6.

C++ codes:

```
// Program P3_6 to illustrate the
// default constructor
#include <iostream>
using namespace std;
class Sum {
public:
    float InitialSum;
    Sum() {
        InitialSum=0;
    }
};
//*****
int main() {

    Sum s;
    cout<<"InitialSum s: "<<s.InitialSum;
    return 0;
}
```

Output:

InitialSum s: 0

Java codes:

```
// Program P3_6 to illustrate the
// default constructor
class Sum {
    public float InitialSum;
    public Sum() {
        InitialSum=0;
    }
//*****
    public static void main(String[] args) {
        Sum s=new Sum();
        System.out.print("InitialSum s: "
            + s.InitialSum);
    }
}
```

Output:

InitialSum s: 0.0

In the program above, as soon as the object is created the constructor is called and the object s is initialized to 0.

The initialization of the object members is necessary. The C++ and Java compilers provide a default constructor implicitly even if the programmer fails to explicitly define a constructor. In Programs P3_6, if we remove the constructor, along with its body, the value of InitialSum is printed as the default value:

Output In C++ compiler:

InitialSum s: 6.30584e-044

Output In Java compiler:

InitialSum s: 0.0

Unlike Java, in the C++ compiler a compiler-dependent value is stored for *s*. Therefore, it would be better that we initialize the constructor and its body.

It is possible to pass parameters to a constructor. Typically, these parameters help initialize an object when it is created. A **parameterized constructor** is used to initialize various data members of different objects with varied values when they are created. Additionally, parameters are simply added to a parameterized constructor for its creation. Each of the Programs P3_7 represents a parameterized constructor which is called for the object *P1* in the main() unit.

C++ codes:

```
// Program P3_7 to illustrate the copy
// and parameterized constructors
#include <iostream>
using namespace std;
class Point {
private:
    double x, y;
public:
    Point(double x1, double y1) {
        x=x1;
        y=y1;
    }
    Point(const Point &P2) {
        x=P2.x;
        y=P2.y;
    }
    double getX() {return x;}
}
```

Java codes:

```
// Program P3_7 to illustrate the copy
// and parameterized constructors
class Point {
private double x, y;
public Point(double x1, double y1) {
    x=x1;
    y=y1;
}
public Point(Point P2) {
    x=P2.x;
    y=P2.y;
}
public double getX() {return x;}
public double getY() {return y;}
/*****
public static void main(String[] args) {
    Point P1=new Point(1.3, -3.5);
}
```

```

    double getY() {return y;}
};
/*****
int main() {
    Point P1(1.3, -3.5);
    Point P2(P1);
    cout<<"P1.x="<<P1.getX()<<" , P1.y="
        <<P1.getY()<<endl;
    cout<<"P2.x="<<P2.getX()<<" , P2.y="
        <<P2.getY();

    return 0;
}

```

```

Point P2=new Point(P1);
System.out.println("P1.x=" + P1.getX()
    + " , P1.y=" + P1.getY());
System.out.println("P2.x=" + P2.getX()
    + " , P2.y=" + P2.getY());
}
}

```

Output:

```

P1.x=1.3, P1.y=-3.5
P2.x=1.3, P2.y=-3.5

```

Output:

```

P1.x=1.3, P1.y=-3.5
P2.x=1.3, P2.y=-3.5

```

The initial values should pass as parameters to the constructor function when an object is declared in a parameterized constructor. However, the normal method of object declaration may fail to work. In C++, the constructors can be called explicitly or implicitly as follows.

```

Point P1 = Point(10, 15);    \\ explicit call
Point P1(10,15);           \\ implicit call

```

The **copy constructor** is a constructor which creates an object by initializing it with an object of the same class already created. A copy constructor has the following general function prototype:

C++ codes:

Java codes:

```

class name (const class name &old_object);    class name (class name old_object);

```

In each of the above Programs P3_7, the new object *P2* is initialized to the old object *P1*. It is worth mentioning that, for

any class which has no user-defined copy constructor, the compiler creates a default copy constructor per se.

Constructors may be overloaded. In fact, when we have both default and parameterized constructors defined in the class, it implies there exist overloaded constructors with or without any parameters. In addition, we can have any number of constructors in a class differing in parameter lists. In each of the following programs P3_8, two constructors with different parameters are defined thus, the constructors are overloaded.

C++ codes:

```
// Program P3_8 to apply the
// overloaded constructors
#include <iostream>
using namespace std;
class Student {
public:
    int id;
    double grade;
    Student(double g) {
        id=191001;
        grade=g;
    }
    Student(int i, double g) {
        id=i ;
        grade=g ;
    }
};
//*****
int main() {
    Student st1(12.5);
    Student st2(191002, 16.75);
    cout<<"st1.id="<<st1.id
        <<" , st1.grade="<<st1.grade;
    cout<<"\nst2.id="<<st2.id
        <<" , st2.grade="<<st2.grade;
}
```

Output:

```
st1.id=191001, st1.grade=12.5
st2.id=191002, st2.grade=16.75
```

Java codes:

```
// Program P3_8 to apply the
// overloaded constructors
class Student {
    public int id;
    public double grade;
    public Student(double g) {
        id=191001;
        grade=g;
    }
    public Student(int i, double g) {
        id=i ;
        grade=g ;
    }
    //*****
    public static void main(String[] args) {
        Student st1=new Student(12.5);
        Student st2=new Student(191002, 16.75);
        System.out.println("st1.id=" + st1.id
            + " , st1.grade=" + st1.grade);
        System.out.println("st2.id=" + st2.id
            + " , st2.grade=" + st2.grade);
    }
}
```

Output:

```
st1.id=191001, st1.grade=12.5
st2.id=191002, st2.grade=16.75
```

In the main part of the above codes, if we write

C++ codes:

```
Student st1;
```

Java codes:

```
Student st1=new Student()
```

then, a compile-time error is encountered since no default constructor is defined.

3.3.4 Destructors and namespaces (C++ only)

A **destructor** is a special methods which destructs an object or, equivalently, de-allocate the memory which was already allocated to the object. Further, a destructor method is automatically called when the object goes out of the scope:

- the function terminates;
- the program ends;
- a block containing local variables ends.

More than one destructor cannot be used for an object in a program. Destructors have the same name as the class preceded by a tilde ‘~’ symbol while they take no parameter and fail to return anything. Program P3_9 aims to highlight the concept of the destructor.

C++ codes only:

```
// Program P3_9 to highlight the concept of destructor
#include <iostream>
using namespace std;
class ABC {
public:
    ABC() {                //constructor is defined
        cout<<"The control is in constructor"<<endl;
    }
    ~ABC() {              //destructor is defined
        cout<<"The control is in destructor"<<endl;
    }
};
//*****
int main() {
    ABC abc1;            //constructor is called
    cout<<"Function main is terminating..."<<endl;
    return 0;
}
```

Output:

```
The control is in constructor
Function main is terminating...
The control is in destructor
```

In the above program, the message The control is in constructor is printed when the constructor is called, followed by printing Function main is terminating... in the main() function. Then, the object abc1, which was created before, goes out of scope, the destructor is called, and The control is in destructor is printed in order to de-allocate the memory consumed by abc1.

We now explain the statement

```
using namespace std;
```

used in most of the C++ programs. The **namespace** is a container for the identifiers. It puts the names of its members in a distinct space so that they are unable to conflict with the

names in other namespaces or global namespaces. The syntax of namespaces is as follows.



Syntax (namespace):

```
namespace identifier {  
    entities  
}
```

where, the identifier is considered any valid identifier and the *entities* are the set of variables, classes, objects, and functions which are included within the namespace. For example, consider the following codes:

```
namespace mySpace {  
    float r, s;  
}
```

The variables *r* and *s* are normal variables declared within a namespace called *mySpace*. We should use the `::` operator in order to access these variables from the outside of *mySpace* namespace. For instance, to access the previous variables from the outside of *mySpace*, we can write:

```
mySpace::r  
mySpace::s
```

Two namespaces are used in Program P3_10.

C++ codes only:

```
// Program P3_10 to create two namespaces
#include <iostream>
using namespace std;
namespace ns1 {
    int value() {return 5;}
}
//*****
namespace ns2 {
    const double x = 100;
    double value() {return 2*x;}
}
//*****
int main() {
    cout<<ns1::value()<<'\n'; // access value function within ns1
    cout<<ns2::value()<<'\n'; // access value function within ns2
    cout<<ns2::x<<'\n';      // access variable x directly
    return 0;
}
```

Output:

```
5
200
100
```

All the files in the C++ standard library declare all of its entities within the std namespace. In fact, the statement

```
using namespace std;
```

requires the compiler to take anything which is in the std namespace and dump it in the global namespace. However, it increases the probability for name conflicts since a bunch of extra names, which were unexpected, were added to the global namespace, and that might butt the heads with some of your own names. Accordingly, it suffices to use the concerned identifiers using the :: operator in the programs which were involved with only a few identifiers in order to avoid this

inconvenience. For instance, `std::cout` notifies the compiler that it requires the `cout` identifier which is in the `std` namespace. Therefore, the following two programs (both in the C++ codes) lead to the same outputs: Sample text.

```
#include <iostream>
using namespace std;
int main() {
    cout<<"Sample text";
    return 0;
}

#include <iostream>
int main() {
    std::cout<<"Sample text";
    return 0;
}
```

Throughout the present book, we adopt the left pattern in our programs in C++ language and use the statement

```
using namespace std;
```

at the top of the programs just after the pre-processor directives.

3.3.5 Static elements

The **static** keyword in C++ and Java is mainly used for memory management. Static elements are allocated to memory only once through a program in the static storage area and they have a scope until the program terminates. The static element can be:

- A local variable in functions (C++ only);
- A data member;
- A method in class;
- A block.

A **static local variable** in C++ exists only inside a function where it is declared (similar to a local variable). Moreover, its lifetime begins when the function is called while it only terminates when the (main) program ends. However, the lifetime of a local variable ends upon terminating the associated function. A static local variable is demonstrated in Programs P3_11 in C++ codes.

C++ codes only:

```
// Program P3_11 to declare and use a static local variable
#include <iostream>
using namespace std;
void getCount() {
    static int count = 0;
    count++;
    cout<<"static local variable count: "<<count<<endl;
}
//*****
int main() {
    getCount();
    getCount();
    getCount();
    return 0;
}
```

Output:

```
static local variable count: 1
static local variable count: 2
static local variable count: 3
```

In the above program, the function `getCount()` is called 3 times. During the first call, the variable `count` is declared as the static local variable and is initialized to 0 by default. Then, the `count` is increased by 1 which is displayed on the screen. When the function `getCount()` returns, variable `count` still exists since it is a static variable. During the second and third function calls, the

previously storage value for *count* is increased by 1 without initializing the 0 value to *count* and then, it is displayed on the screen. The output of Program P3_11, if the *count* is not specified as a static variable, is as:

```
static local variable count: 1
static local variable count: 1
static local variable count: 1
```

Different copies of a normal data member are created with the associated objects when that data member is declared in a class. In some cases, a common data member is needed which should be the same for all the objects. We cannot perform this using normal data members and thus, static data members are required in this respect. A **static data member**, known as a class member, is declared with the static keyword. It is noteworthy that a static data member cannot be private. It is called a class member since it belongs to the class instead of the objects. Additionally, this data member makes the program memory efficient, that is, it saves the memory. No matter how many objects of the class are created, there is only one copy of the static data member. In other words, one single copy of static data member is shared between all the objects of that class. When the first object is created, a static data member is initialized to zero if no other initialization is presented.

In C++, the declaration of a static data member is unable to be considered a definition. The data member is declared in a class scope while the definition is performed at the file scope using the scope resolution :: operator as:

data type_class name::static data member name;

In the left C++ program P3_12 below, the static data member *count* is defined by the following statement after the class StA.

```
int StA::count;
```

The static data member, along with the definition may be initialized in C++. For example, we may initialize the above-mentioned static data member *count* to, say, 1 and write the following statement instead of the above one.

```
int StA::count = 1;
```

The above paragraph is valid in C++. However, in Java, static data members cannot be initialized outside the class. Instead, they can be separately initialized in the static blocks. A **static block** is a block of statements inside a Java class which is executed before the main method at the time of class loading. By default (if no value is initialized), the value of a numerical static data member is considered as 0 in both C++ and Java languages.

In addition, static data members can be directly accessed by the class name while no object is required to access these variables. The following codes represents how a static member can be accessed:

C++ codes:

class name : static data member name

Java codes:

class name . static data member name

Programs P3_12 are examples of static data members in C++ and Java.

C++ codes:

```
// Program P3_12 to declare and
// use a static data member
#include <iostream>
using namespace std;
class StA {
public:
    static int count;
    StA() {
        count++;
    }
};
int StA::count;
//*****
int main() {
    StA a;
    cout<<"static data member for first "
        <<"object: "<<StA::count<<endl;
    StA b;
    cout<<"static data member for second "
        <<"object: "<<StA::count<<endl;
    StA c;
    cout<<"static data member for third "
        <<"object: "<<StA::count<<endl;
    return 0;
}
```

Output:

```
static data member for first object: 1
static data member for second object: 2
static data member for third object: 3
```

Java codes:

```
// Program P3_12 to declare and
// use a static data member
class StA {
    public static int count;
    public StA() {
        count++;
    }
//*****
    public static void main(String[] args) {
        StA a=new StA();
        System.out.println("static data member "
            + " for first object: " + a.count);
        StA b=new StA();
        System.out.println("static data member "
            + " for second object: " + a.count);
        StA c=new StA();
        System.out.println("static data member "
            + " for third object: " + a.count);
    }
}
```

Output:

```
static data member for first object: 1
static data member for second object: 2
static data member for third object: 3
```

In each of these programs, we have created the StA class with a static data member *count* declared inside the class. By default, *count* is initialized to 0. Of course, in C++ the static data members should be defined outside the class. This is done in the left program using the :: operator. In the default constructor of this class we are incrementing the static variable *count* by 1. So

every time an object is created this value is incremented. The output of the programs show this fact for three objects created in the main() function.

A **static method**, also called a class method, belongs to the class rather than the objects of a class. In other words, allocating static storage, a static method is independent of any particular object of the class. That is to say that, a static method can be called directly using the class name itself rather than creating an object and calling from the object. A static method can only access static data members and other static methods with the following statements in C++ and Java.

C++ codes:

class name : : *method name* ();

Java codes:

class name . *method name* ();

The most common example of a static member is Java's main() method. This special method is referred to as the main method or the driver method. Programs P3_13 are examples of static methods.

C++ codes:

```
// Program P3_13 to declare and
// use a static method
#include <iostream>
using namespace std;
class StB {
public:
    static int count, k;
    static int getCount() {
        callCount();
        count++;
        return count;
    }
    static void callCount () {
        k++;
        cout<<"In call "<<k;
    }
};
int StB::count=10;
int StB::k;
//*****
int main() {
    cout<<"Calling the static member "
        <<"function:"<<endl;
    cout<<" , count is: "
        <<StB::getCount()<<endl;
    cout<<" , count is: "
        <<StB::getCount()<<endl;
    cout<<" , count is: "
        <<StB::getCount()<<endl;
    return 0;
}
```

Output:

```
Calling the static method:
In call 1, count is: 11
In call 2, count is: 12
In call 3, count is: 13
```

Java codes:

```
// Program P3_13 to declare a
// use a static method
class StB {
    public static int count, k;
    public static int getCount() {
        callCount();
        count++;
        return count;
    }
    public static void callCount() {
        k++;
        System.out.print("In call " + k);
    }
    static {
        count=10;
    }
    //*****
    public static void main(String[] args) {
        System.out.println("Calling the "
            + "Static method:");
        System.out.println(", count is: "
            + StB.getCount());
        System.out.println(", count is: "
            + StB.getCount());
        System.out.println(", count is: "
            + StB.getCount());
    }
}
```

Output:

```
Calling the static method:
In call 1, count is: 11
In call 2, count is: 12
In call 3, count is: 13
```

In each of the Programs P3_13, we created two static data members *count* initialized to 10 in the program as well as *k* which is initialized to 0 by default. Furthermore, two static methods *getCount()* and *callCount()* are created which access to these static data members. Moreover, the static method

callCount() is called in the static method getCount(). As shown in the left program, both static data members are defined after ending the class. However, as previously mentioned, this is unnecessary in Java unless we want to initialize a value to the data member. In the right program, we initialized the static data member to 10 within a static block. As demonstrated in both programs, the static methods are directly called without creating any object.

3.3.6 The **this** keyword

The **this** keyword is commonly used in both C++ and Java languages. There are various applications of this keyword in both languages. However, due to the calculation aim of the present book, we do not need all the applications in our programs and only one feature is used in several programs.

Moreover, the **this** keyword in C++ is an important pointer accessible only within the non-static member functions of a class and points to the object for which the member function is called. However, in Java, the **this** keyword is an instance variable which refers to the current object in a method or constructor.

As explained above, using the **this** keyword is nearly the same in both languages. The most common use of the **this** keyword in both C++ and Java is to resolve the ambiguity between the instance variables and parameters with the same name. This can be accomplished by the following syntaxes:

C++ codes:

```
this->var = var;
```

Java codes:

```
this.var = var;
```

where, the variable *var* is simultaneously regarded as an instance variable and a parameter. In the left C++ codes, *this->var* means *(this*).var*, in the literature of the pointers ([Section 2.3](#)). To clarify the above discussions, consider Programs P3-14.

C++ codes:

```
//Program P3_14 to apply 'this' keyword
#include <iostream>
using namespace std;
class Stud {
public:
    int id;
    double grade;
    Stud (int id, double grade) {
        this->id=id;
        this->grade=grade;
    }
    void display() {

        cout<<"ID: "<<id<<" , Grade: "
            <<grade<<endl;
    }
};
//*****
int main() {
    Stud st1= Stud (191005, 15.75);
    Stud st2= Stud (191008, 17.25);
    st1.display();
    st2.display();
    return 0;
}
```

Output:

```
ID: 191005, Grade: 15.75
ID: 191008, Grade: 17.25
```

Java codes:

```
//Program P3_14 to apply 'this' keyword
class Stud {
    public int id;
    public double grade;
    public Stud (int id, double grade) {
        this.id=id;
        this.grade=grade;
    }
    public void display() {
        System.out.println("ID: " + id
            + " , Grade: " + grade);
    }
}
//*****
public static void main(String args[]) {
    Stud st1=new Stud (191005, 15.75);
    Stud st2=new Stud (191008, 17.25);
    st1.display();
    st2.display();
}
}
```

Output:

```
ID: 191005, Grade: 15.75
ID: 190087, Grade: 17.25
```

In the above-mentioned programs, the parameters and instance variables *id* and *grade* are homonymous. Therefore, the **this** keyword is used to distinguish between the local and instance variables. Removing the **this** keyword, along with -> in the left and . in the right programs, the following output is displayed:

Output (the left program):

ID: 0, Grade: 4.94066e-324
ID: 46, Grade: 2.2727e-322

Output (the right program):

ID: 0, Grade: 0.0
ID: 0, Grade: 0.0

In the left output, the strange numbers stand for the existence of errors.

4 Decision making and branching templates

In many studies and branches of science, there is a limited set of infrastructure objects and tools in which all the elements are subject to specific rules and principles. These elements act as the building blocks and have a certain names in every case. For instance, the periodic table of the elements in chemistry, regular figures and curves in geometry, numbers in arithmetic, prime numbers in number theory, and building materials in the construction industry are several examples of various sets of the building blocks.

As previously explained in **Chapter 1**, algorithms are the root bases of the elementary programming in every programming language. This set of infrastructure objects are called templates in algorithm designing. Each template in an algorithm corresponds to a statement in a program. Fortunately, a limited number of templates exist which facilitate memorization of each statement; however, this is not enough. What is worth mentioning is when, where, and how to use these statements appropriately.

In the remaining chapters of this book, it is attempted to introduce and gradually strengthen the ability to employ these templates in an optimized and systematic way. Facilitating and organizing the algorithms, as well as having the ability to

translate the templates into the codes of any programming language, are two major advantages of the templates. Therefore, the algorithms created by such templates can be translated into any programming language. In other words, writing a program for the algorithm in a certain programming language, includes translating the algorithm of that problem into the codes of the programming language, template by template, after constructing the algorithm of a specific problem.

Templates are divided into simple and compound types. A simple or a 1-shape template indicates that the corresponding statement of that template is a single reserved word without any block. The input cin, output cout, and the assignment = statements correspond to simple templates and are referred to as input (in the shape of a parallelogram), output (in the shape of a torn paper), and assignment (in the shape of a rectangle), templates, respectively. Bear in mind that the assignment operator is regarded as the assignment statement. In addition, more instructions in the same rectangle are counted as a simple template.

Further, a compound template includes more than one shape, or its corresponding statement has one or more reserved words along with several blocks. The templates in the algorithms are generally categorized into decision making and branching (the conditional), looping, and jumping groups. The decision making and branching templates are studied in the present chapter. Furthermore, **chapters 5, 6 and 7** are dedicated to the looping templates. The jumping templates are scattered

in [Chapters 4, 6, and 7](#) and occasionally applied in the remaining chapters.

The present chapter first examines three two-way branching templates, namely the if, if-else, and if-else-if ones. Then, two jumping goto and exit templates are presented for transmitting the implementations and termination, respectively. The final section delves into a multi-way branching template, called the switch template.

Accordingly, the flowchart of each template and its translation into both C++ and Java codes are provided. Then, the result of template implementation and other related details are explained. Finally, the template is applied in various examples.

It is strongly recommended to consider the following important rule regarding the blocks ([Subsection 2.1.5](#)).



The rule of grouping. A block encompassing more than one statement should be placed between a couple of curl braces {}.

Failure to comply with this rule leads to an error or incorrect results. For example, consider the following small programs in both C++ and Java codes regardless of the notion of codes. As seen, the ranges of if, else, and the body of the main program (method), as the specific blocks, are separately grouped and are shifted two spaces to the right.

C++ codes:

```
// Sample program.
#include <iostream>
using namespace std;
int main() {
    int k;
    cout<<"Enter the integer k: ";
    cin>>k;
    if (k>0) {
        cout<<"Inside: k="<<k<<endl;
        k++;
    }
    else {
        cout<<"Inside: k="<<k<<endl;
        k--;
    }
    cout<<"Outside: k="<<k<<endl;
    return 0;
}
```

Java codes:

```
// Sample program.
import java.util.Scanner;
public class Sample_program {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int k;
        System.out.print("Enter the integer k: ");
        k=read.nextInt();
        if (k>0) {
            System.out.println("Inside: k=" + k);
            k++;
        }
        else {
            System.out.print("Inside: k=" + k);
            k--;
        }
        System.out.print("Outside: k=" + k);
        read.close();
    }
}
```

Although writing the statements inside the blocks (two or three columns ahead) fails to affect the program running, it has two advantages. It makes the program clear and well-built by separating various blocks in different groups. Moreover, it enables the reader to easily analyze and perceive the program by distinguishing such blocks in a variety of groups.



Conventions in algorithm-writing. The following conventions are adopted in algorithm-writing, as well as programming used either in the examples or exercises (in the texts or at the end of chapters).

- The word "read" means "read from the input unit (keyboard)";
- The word "print" indicates "print on the output unit (screen)";
- The word "receive" implies "receive from the call unit (see [Chapter 5](#))";

- The word “return” denotes “return to the call unit (see [Chapter 5](#))”;
 - For the sake of simplicity, we write: print MESSAGE instead of print “MESSAGE”. This may not be confused since the phrase “MESSAGR” is in the code font.
-

4.1 The if-else template

The **if-else template** appears when there is a two-way branch. Given the rule of distributions in the flowcharts represented in [chapter 1](#), the flowchart related to this template along with the syntaxes of the if-else statement in both C++ and Java codes is illustrated as follows.

The if-else template	Syntax in C++ codes	Syntax in Java codes
-----------------------------	----------------------------	-----------------------------

Implementation: If the condition is true, first, the T-path (the if-range in programming) including block A is implemented and then the implementation control is transferred to position P. Otherwise, the F-path (the else-range in programming) encompassing block B is implemented and the control is transferred to position P.



The statements in the if-range and else-range, should be grouped by {} if they are more than one.

The above-mentioned problem related to disregarding the grouping notice can be observed in the next example.

4.1. Example (The hanging problem). This problem arises when more than one statement is presented between the if and else parts without curly braces in the if-else

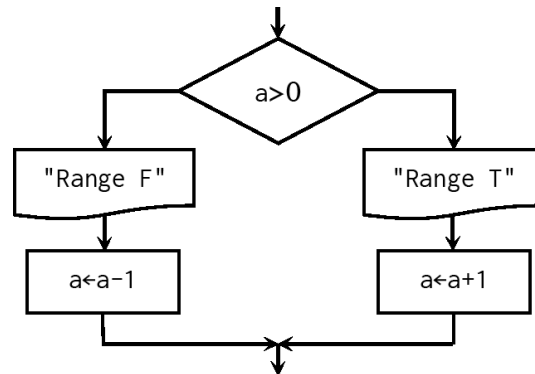


Fig. 4.1(a): The hanging problem flowchart (without problem).

statements. This example is presented for further clarification. Translate the sample flowchart in **Figure 4.1(a)** into both C++ and Java codes.

Solution. The following if-else statements are the answer to the above-mentioned problem:

C++ codes:

```
if (a>0) {
    cout<<"Range T";
    a=a+1;
}
else {
    cout<<"Range F";
    a=a-1;
}
```

Java codes:

```
if (a>0) {
    system.out.print("Range T");
    a=a+1;
}
else {
    system.out.print("Range F");
    a=a-1;
}
```

If grouping the if-range is disregarded, then we will encounter *the hanging if problem*: the statement $a = a + 1$ is hanging. Therefore, there will be a “misplaced if” error. Additionally, the existence of an error alerts the presence of a problem which

needs to be solved. The worse situation happens when the else-range is expressed without grouping as:

C++ codes:

```
else
  cout<<"Range F";
  a = a - 1;
```

Java codes:

```
else
  system.out.print("Range F");
  a=a-1;
```

In fact, in this case, no error appears while only the statement after else is processed as the else-range. In other words, the corresponding flowchart is shown in **Figure 4.1(b)**.

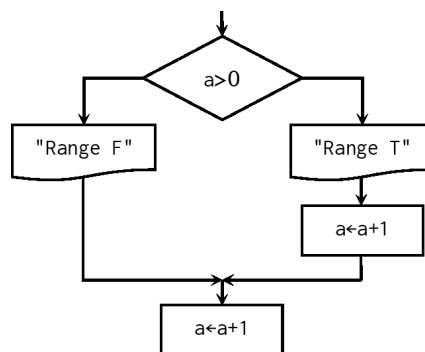


Fig. 4.1(b): The flowchart of the hanging problem (with a problem).

Therefore, using the grouping rule is extensively emphasized due to the great importance of the subject. However, rarely ever, if the ranges are extremely short to place the whole statement in a single line, this rule can be ignored to simplicity and reduce the size of the program. For example:

C++ codes:

```
if (i>0) i++; else i--;
```

Java codes:

```
if (i>0) i++; else i--;
```

4.2. Example. Construct a flowchart to read the integer number a . Then, print the read number followed by the message “ $_$ is

positive” if the number is positive; otherwise print n followed by the message “ n is not positive”.

Solution. The required flowchart is depicted in **Figure 4.2**. This flowchart is translated into both C++ and Java codes in Programs P4_2.

C++ codes:

```
// Program P4_2 to use the
// if-else statement
#include <iostream>
using namespace std;
int main() {
    int a;
    cout<<"Enter an integer: ";
    cin>>a;
    if (a>0)
        cout<<a<<" is positive";
    else
        cout<<a<<" is not positive";
    return 0;
}
```

Java codes:

```
// Program P4_2 to use the
// if-else statement
import java.util.Scanner;
class P4_2 {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int a;
        System.out.print("Enter an integer: ");
        a=read.nextInt();
        if (a>0)
            System.out.print(a + " is positive");
        else
            System.out.print(a + " is not positive");
        read.close();
    }
}
```

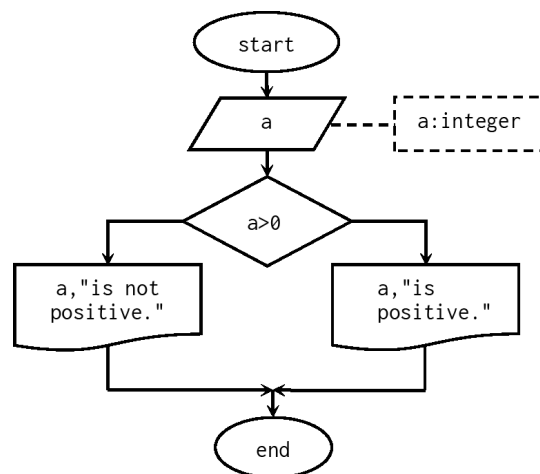


Fig. 4.2: A simple use of the if-else template.

4.2 The if template

If one of the two blocks in the if-else template is null, it is placed in the F-path in order to standardize it. In other words, if we have only one block, it is put in the T-path. This trend is guaranteed since every condition has a negation. Therefore, even if a block is posited in the F-path, it can be moved to the T-path by negating the condition. The obtained template, which is a special case of the if-else template, is called the **if template**. The F-path in the flowchart of the if-else template collapses since block B does not exist in this flowchart. In addition, block B together along with the else reserved word in the syntaxes of the if-else statement of both codes is removed. Consequently, the representation of the if template, as well as the syntaxes of the if statement in both codes is as follows.

The if template	Syntax in C++ codes	Syntax in Java codes
-----------------	---------------------	----------------------

Implementation: The T-path (the if-range in programming) including block A is first implemented if the condition is true and then the control is transferred to position P. Otherwise, the control is transferred to position P.



The statements in the if-range should be grouped by {} if they more than one.

Although the if statement has only one reserved word, it always contains a block (the if-range).



The rule of the intersection of the T- and F-paths. Intersect the T- and F-paths in the flowcharts of the if and if-else templates exactly below the involved condition.

This rule is one of the important tasks in drawing the flowcharts and has three advantages. First, it makes the flowchart clear and legible, in particular from the implementation point of view. Second, the flowcharts constructed using this rule, are translatable into any programming language. Third, this rule reduces the probable errors arising from disregarding the grouping rule. Some of these features are experienced in the next example.

4.3. Example. Draw a flowchart to read a positive integer number. Then, print the output if it is positive:

The given number is positive

Print the output if it is zero:

The given number is zero

And print the output if it is negative:

The given number is negative

Solution. Here, we are facing a three-way branching. Therefore, two if-else templates are used for this purpose. One of these three branching is put in the T-path of an if-else template while the other two branching are placed in the F-path. Afterwards,

the two-way branching in the F-path is separated using another if-else template. Finally, the flowchart displayed in **Figure 4.3(a)** is obtained. Programs P4_3_A display the translation of this flowchart into both C++ and Java codes.

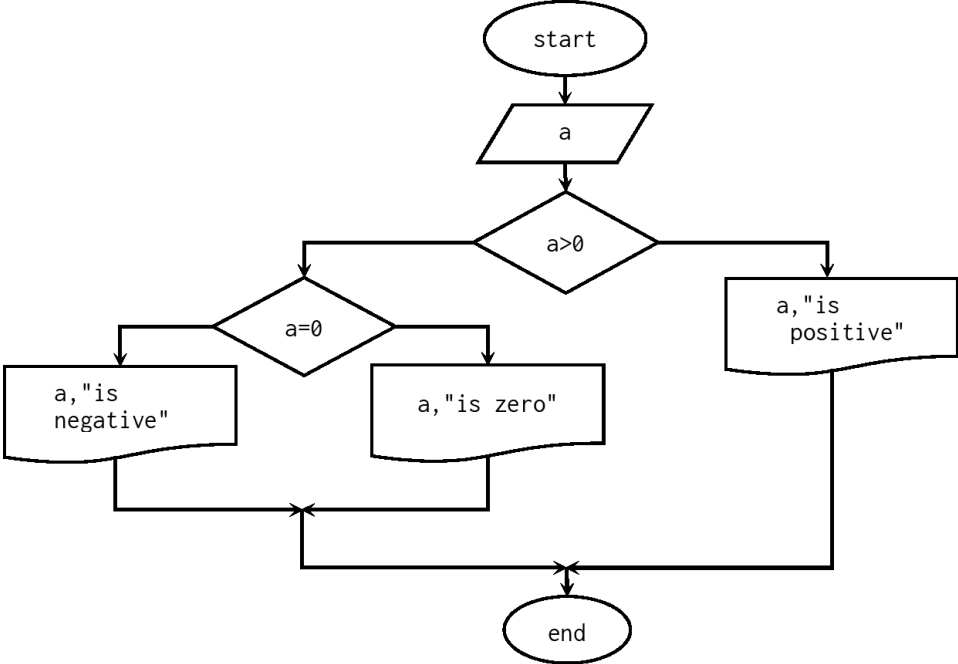


Fig. 4.3(a): Nested conditional templates.

C++ codes:

```
// Program P4_3_A to use nested
// the if-else statements
#include <iostream>
using namespace std;
int main() {
    int a;
    cout<<"Enter an integer: ";
    cin>>a;
    if (a>0)
        cout<<a<<" is positive";
    else
        if (a==0)
            cout<<a<<" is zero";
        else
            cout<<a<<" is negative";
    return 0;
}
```

Java codes:

```
// Program P4_3_A to use nested
// the if-else statements
import java.util.Scanner;
class P4_4_A{
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int a;
        System.out.print("Enter an integer: ");
        a=read.nextInt();
        if (a>0)
            System.out.print(a + " is positive");
        else
            if (a==0)
                System.out.print(a + " is zero");
            else
                System.out.print(a + " is negative");
        read.close();
    }
}
```

Question. Why didn't we comply with the grouping rule for the else range from the outer if-else template?

Answer. Since only one statement (the compound if-else statement) existed in this respect.

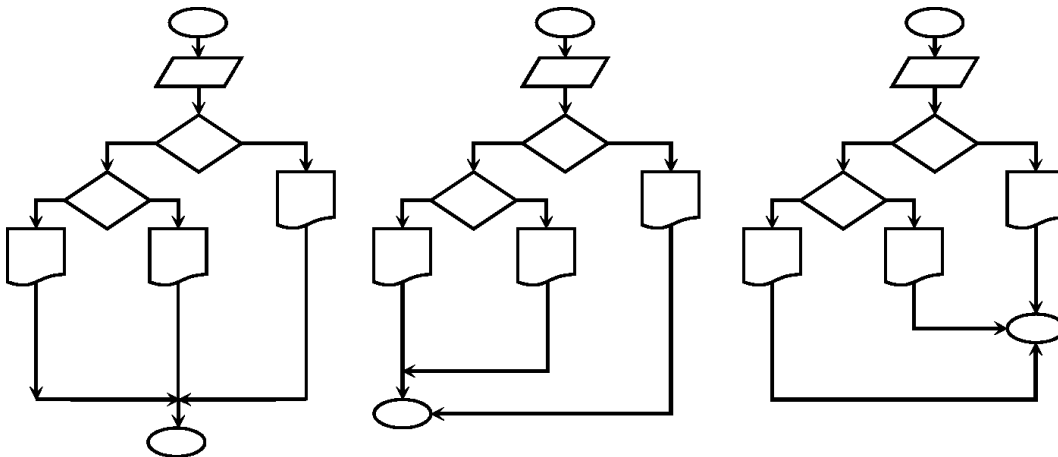


Fig. 4.3(b): The rule of T- and F-paths is not used.

As illustrated in **Figure 4.3(a)**, the body of the T-path of the external conditional template is itself another conditional template, that is, the if-else template. In this case, we are supported to have two *nested conditional templates (nested conditional statements* in programming). As shown, this flowchart is totally transparent and complete. Having applied the rule of T- and F-paths, one can observe the clearness and legibility of **Flowchart 4.3(a)**. However, the three flowcharts in **Figure 4.3(b)** lack this feature.

Further, every IF reserved word in the IF-ELSE-ENDIF statement ends with an ENDIF reserved word in some programming languages including Fortran. The translation of **Flowchart 4.3(a)** into Fortran codes is as follows:

```
READ*, a
IF a.GT.0
    PRINT*, a, 'is positive'
ELSE
    IF a.EQ.0
        PRINT*, a, 'is zero'
    ELSE
        PRINT*, a, 'is negative'
    ENDIF
ENDIF
```

In the main model of **Figure 4.3(a)**, the positions of each ENDIF is clearly determined in the intersection of the T- and F-paths of the involved IF-ELSE-ENDIF template. However, no unique place can be determined for each ENDIF in the three models of **Figure 4.3(b)**

It occasionally happens to terminate the implementation of a flowchart before closing the entire flowchart with the keyword “end”. Any programming languages has a one-keyword statement for this purpose. For example, the keywords

STOP in Fortran, halt in Pascal, and exit(0) in both C++ and Java play the role of termination. However, the following rule is recommended in the flowcharts.

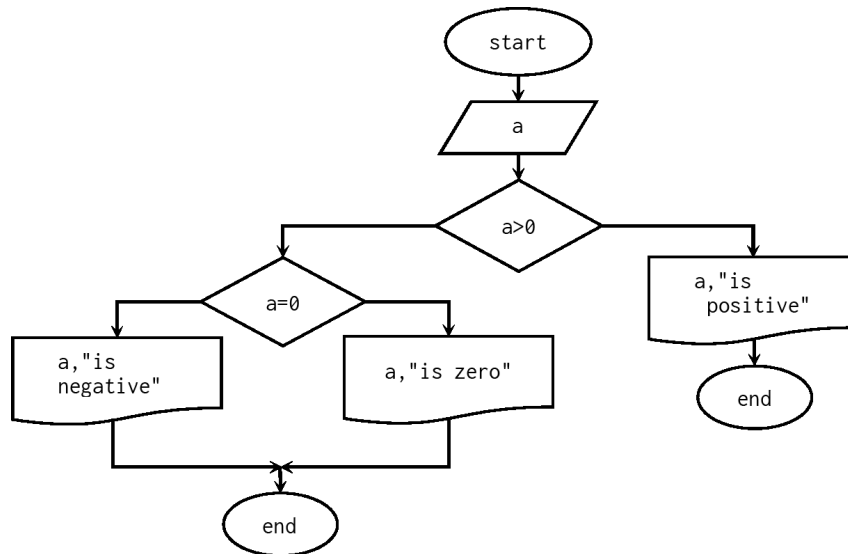


Fig. 4.3(c): The existence of two “end” keywords (not recommended).



Close each flowchart with only one ellipse end-shape with the keyword “end” inside.

Flowchart 4.3(c) is not recommended for **Example 4.3**, although we cannot claim that it is incorrect. One may use a common word like “stop”, or specialized keyword exit(0) instead of the keyword “end” inside the right hand end-shape.

4.4. The dangling else problem. This classic problem is created when no matching else is found for each if. This problem basically occurs when the nested if or if- else statements are employed without grouping by {}. Let us explain this fact in

more details. There are two if and if-else statements both in the one-line form in the following part:

```
if (c1) i++;  
if (c2) j++; else k++;
```

Either of the statements is clear and there is no problem. Now consider the following on-line statement:

```
if (c1) if (c2) r++; else s++;
```

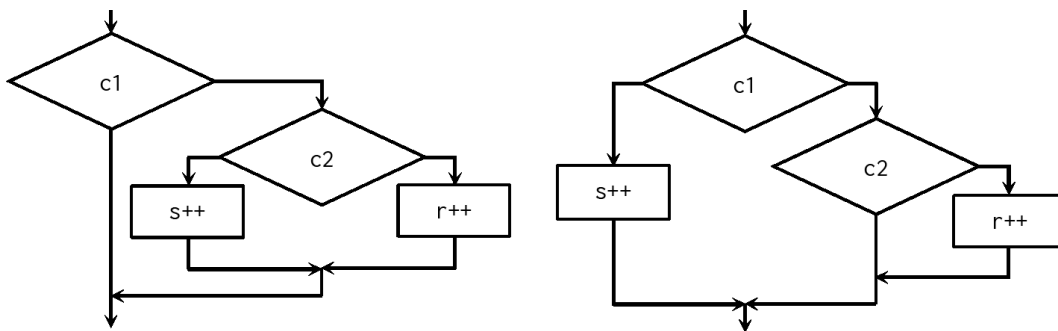


Fig. 4.4: Dangling problem flowchart.

In this example, `r++` is unambiguously executed when both conditions are true. However, this question arises that the else clause is dangling on which if? One may interpret `s++` as being executed when the condition `c1` is false (thus, attaching the else to the first if) or true and the condition `c2` is false (therefore, attaching the else to the second if). In other words, the previous statement is found as either of the following expressions whose flowcharts are displayed in **Figure 4.4**. The codes of the right flowchart is as follows:

C++ codes:

```
if (c1) {  
    if (c2)  
        r++;  
}  
else  
    s++;
```

Java codes:

```
if (c1) {  
    if (c2)  
        r++;  
}  
else  
    s++;
```

However, the codes of the left flowchart:

C++ codes:

```
if (c1) {  
    if (c2)  
        r++;  
    else  
        s++;  
}
```

Java codes:

```
if (c1) {  
    if (c2)  
        r++;  
    else  
        s++;  
}
```



The rule of dangling else. The reserved word else is always paired with the most recent reserved word if.

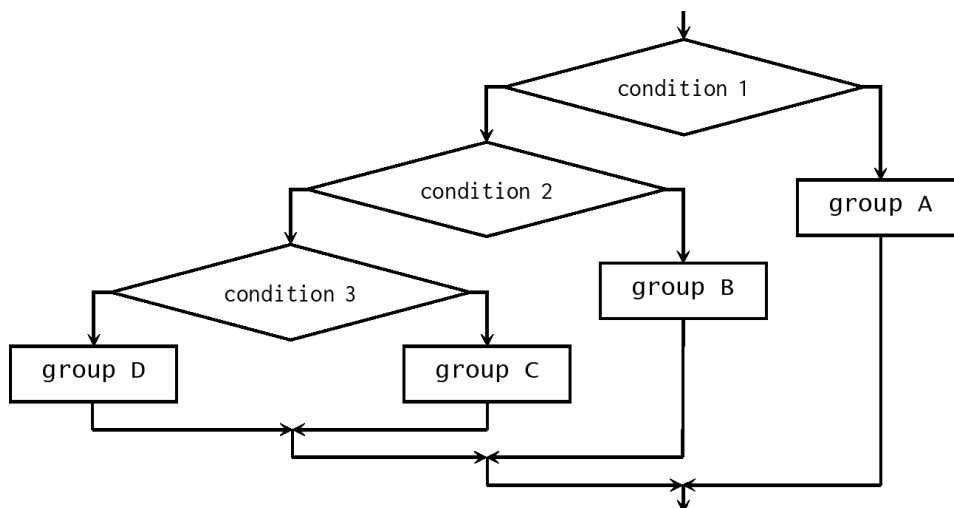


Fig. 4.5(a): Multi-way decision with nested if-else templates.

Accordingly, the expression


```
if (c1) if (c2) r++; else s++;
```

is equivalent with the second expression provided above. Therefore, applying the grouping rule is helpful for avoiding similar ambiguities.

4.3 The if-else-if template

4.5. Observation. Occasionally, there is a variety of nested if-else templates in algorithms, where a multi-way (multi-condition) decision is taken. An example of these circumstances including three conditions is represented in [Figure 4.5\(a\)](#).

The condition is evaluated from the top of the ladder downwards. Considering the statements pinpointed so far, translating [Flowchart 4.5\(a\)](#) into both codes are as follows.

C++ codes:

```
if (condition 1 )  
    block A  
else  
    if (condition 2 )  
        block B  
    else  
        if (condition 3 )
```

Java codes:

```
if (condition 1 )  
    block A  
else  
    if (condition 2 )  
        block B  
    else  
        if (condition 3 )
```

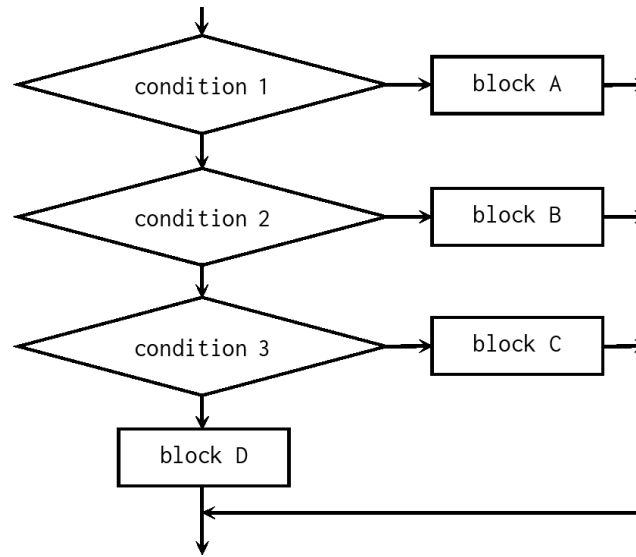


Fig. 4.5(b): The if-else-if template.

block C
 else
block D

block C
 else
block D

As it is shown, we face three nested if-else templates. From the implementation point of view, the previous flowchart is equivalent to the one in **Figure 4.5(b)**. This template is referred to as the **if-else-if template**. The syntaxes of the if-else-if statement in both C++ and java codes are presented below.

C++ codes:

```
if (condition 1)
    block A
else if (condition 2)
    block B
else if (condition 3)
    block C
else
    block D
```

Java codes:

```
if (condition 1)
    block A
else if (condition 2)
    block B
else if (condition 3)
    block C
else
    block D
```



The statements in each block, should be grouped by {} if they are more than one.

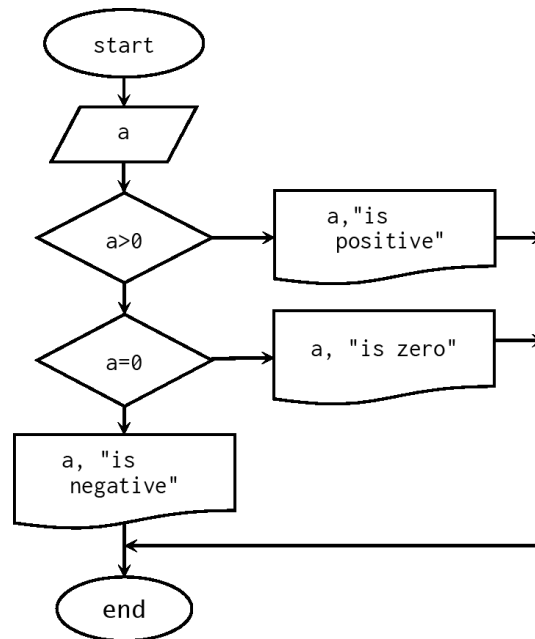


Fig. 4.5(c): Flowchart 4.3(a) with the if-else-if template.

The clearness and simplicity in the flowchart and codes of the if-else-if template can be perceived by comparing the flowcharts 4.5(a) and 4.5(b) and their codes.

It should be noted that in the multi-way branching, it is always possible to provide the nested if-else template as Figure 4.5(a) by negating the conditions if necessary and, following it, the if-else-if template as Figure 4.5(b). As an example, Flowchart 4.5 (c) is a redrawing of Flowchart 4.3 (a). Programs P4_5_C are the same as Program P4_3_A using the if-else-if statement instead of the nested if-else statements.

C++ codes:

```
// Program P4_5_C to rewrite Program
// 4_3_A with the if-else-if statement
#include <iostream>
using namespace std;
int main() {
    int a;
    cout<<"Enter an integer: ";
    cin>>a;
    if (a>0)
        cout<<a<<" is positive";
    else if (a==0)
        cout<<a<<" is zero";
    else
        cout<<a<<" is negative";
    return 0;
}
```

Java codes:

```
// Program P4_5_C to rewrite Program
// 4_3_A with the if-else-if statement
import java.util.Scanner;
class P4_5_C {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int a;
        System.out.print("Enter an integer: ");
        a=read.nextInt();
        if (a>0)
            System.out.print(a + " is positive");
        else if (a==0)
            System.out.print(a + " is zero");
        else
            System.out.print(a + " is negative");
        read.close();
    }
}
```

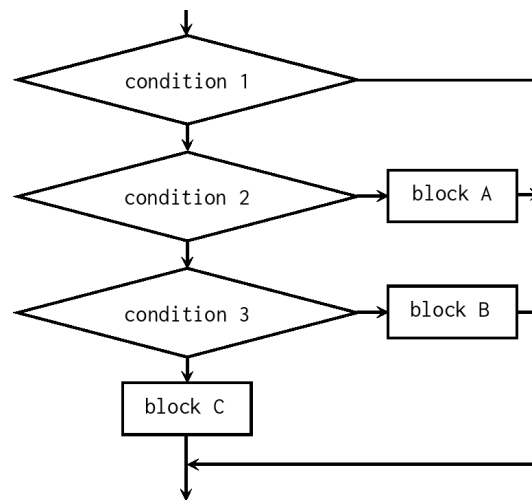


Fig. 4.6(a): The if-else-if template cannot be used.

Considering the following rule is necessary when using the if-else-if templates.



The rule of making the if-else-if template. The last F-path block in the if-else-if template may be empty, while none of the T-paths should be empty. Otherwise, a

combination of the if-else-if template and the nested if-else templates should be used.

The next example clarifies the issue described in the above rule.

4.6. Example. The if-else-if template cannot be used for **Flowchart 4.6(a)** since it fails to satisfy the above rule. Furthermore, it is not consistent with the standard templates proposed in this book. Use a combination of the nested if-else templates and the if-else-if template to draw an equivalent flowchart for **Figure 4.6(a)**.

Solution. Negate the condition 1 to create an if template. Moreover, an if-else-if template may be applied for the rest of the flowchart. The resulted flowchart is depicted in **Figure 4.6(b)**

4.6_1. Exercise. Translate **Flowchart 4.6(b)** to both C++ and Java codes.

Question. Is it necessary to group the block corresponding the T-path block of the outer if template in programming?

Answer. The answer is no, since only one if-else-if template exists.

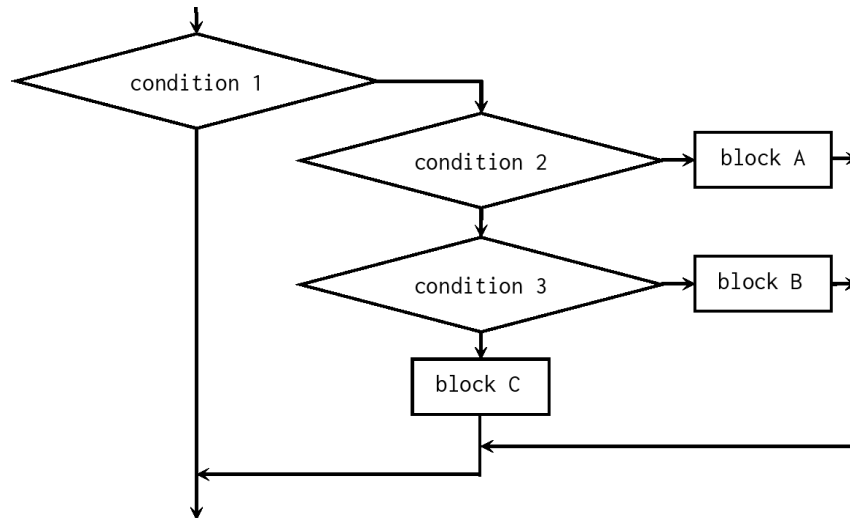


Fig. 4.6(b): An if-else-if template nested in an if template.

4.7. Example. Draw a flowchart to read the three integer numbers a , b , and c . Then, if one of the inequalities $a < 10$, $b < 20$, or $c < 30$ holds, then print Yes. Otherwise, print No.

Solution. The required flowchart is drawn using the if-else-if template in **Figure 4.7(a)**.

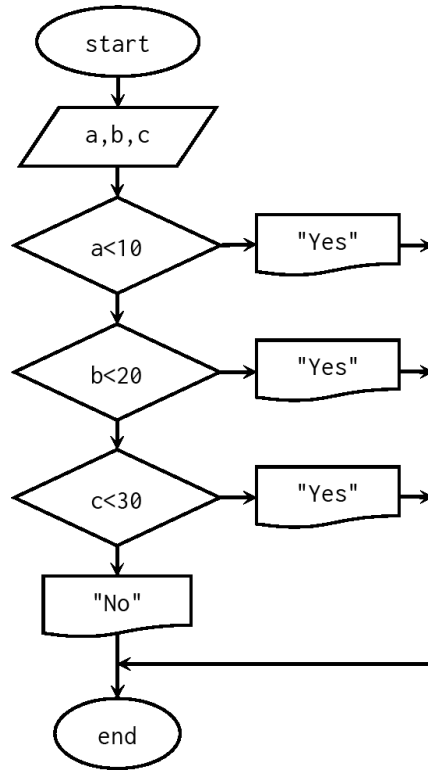


Fig. 4.7(a): Print Yes if one of the three conditions is true; otherwise, print No.

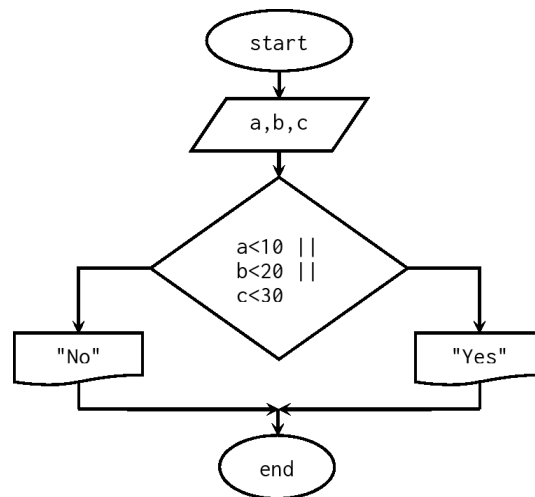


Fig. 4.7(b): Merging the conditions in [Figure 4.7\(a\)](#) with the `||` operator.

Programs P4_7 is the transferred codes of [Flowchart 4.7\(a\)](#) in both C++ and Java languages.

Java codes:

C++ codes:

```
// Program P4_7: another use of
// the if-else-if statement
#include <iostream.h>
using namespace std;
int main() {
    int a, b, c;
    cout<<"Enter three integers: ";
    cin>>a>>b>>c;
    if (a<10)
        cout<<"Yes";
    else if (b<20)
        cout<<"Yes";
    else if (c<30)
        cout<<"Yes";
    else
        cout<<"No";
    return 0;
}
```

```
// Program P4_7: another use of
// the if-else-if statement
import java.util.Scanner;
class P4_7 {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int a, b, c;
        System.out.print("Enter three integers: ");
        a=read.nextInt();
        b=read.nextInt();
        c=read.nextInt();
        if (a<10)
            System.out.print("Yes");
        else if (b<20)
            System.out.print("Yes");
        else if (c<30)
            System.out.print("Yes");
        else
            System.out.print("No");
        read.close();
    }
}
```

We can combine the three conditions and use the logical `||` operator based on [Figure 4.7\(b\)](#) since the same print exists in all the three T-paths.

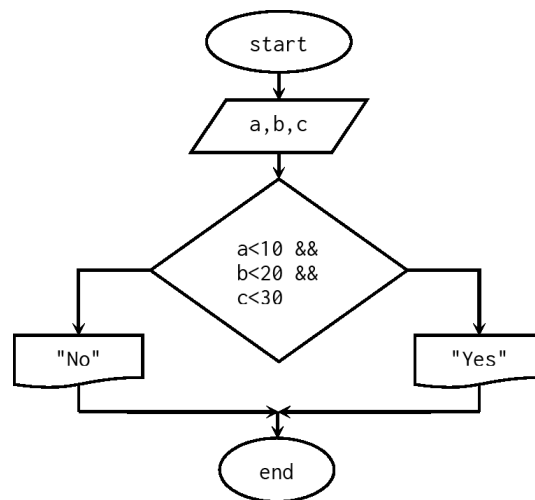


Fig. 4.8(a): Flowchart in [Figure 4.7\(b\)](#) with the operator `&&` instead of `||`.

4.7.1. Exercise. Translate **Flowchart 4.7(b)** into both C++ and Java codes.

4.7.2. Exercise. Draw a deformation of **Flowchart 4.7(a)** using an if-else template and the `||` operator. Then, translate it into both C++ and Java codes.

4.8. Observation. The flowcharts in **Figures 4.7(a)** and **4.7(b)** are equivalent. The former is drawn using the if-else-if template while there a conditional template in the latter one which is depicted employing the `||` operators. **Figure 4.8(b)** is similar to **Figure 4.7(b)** which is drawn applying the `&&` rather than the `||` operators.

Based on the above-mentioned observation, we attempt to draw a flowchart in which the if-else-if template is used instead of the `&&` operators. Note that the composed proposition

$$a < 10 \ \&\& \ b < 20 \ \&\& \ c < 30$$
$$a < 10 \ \&\& \ b < 20 \ \&\& \ c < 30$$

is true if and only if all the three conditions are true. Therefore, the part of the flowchart which prints Yes acts as **Flowchart 4.8 (b)**. When is No printed? Considering the value of the above composed proposition, the phrase “No” is printed when at least one of the conditions above is false. Then, No is printed in the F-path of all the three conditions. The following rule is applied to close several opened conditional templates.



The rule of closing the conditional templates. The opened conditional templates are closed from the innermost to the outermost ones, respectively.

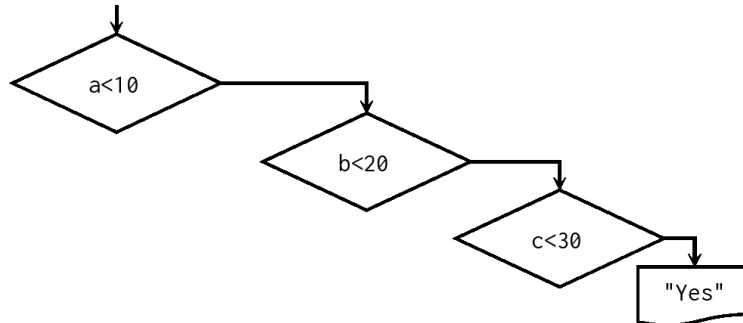


Fig. 4.8(b): Part of the flowchart equivalent to [Figure 4.8\(a\)](#) which prints Yes.

According to the above rule, first, close the template involved with the condition $c < 30$. Taking into account this if-else template as the T-path of the condition $b < 20$, close this middle if-else template. Finally, consider this double nested if-else templates as the T-range of the condition $a < 10$ and close the outermost if-else template. [Figure 4.8\(c\)](#) illustrates the obtained flowchart.

[Flowchart 4.8\(c\)](#) cannot be directly translated into the program codes using the if-else-if statement since its structure is not only the same as the structure of the if-else-if template in [Figure 4.5\(a\)](#) but also it is the mirror symmetric to that structure. Accordingly, the same structure as that of [Figure 4.5\(a\)](#) is obtained if we negate the conditions and thus, the if-else-if statement is applied to translate it into the codes of the program.

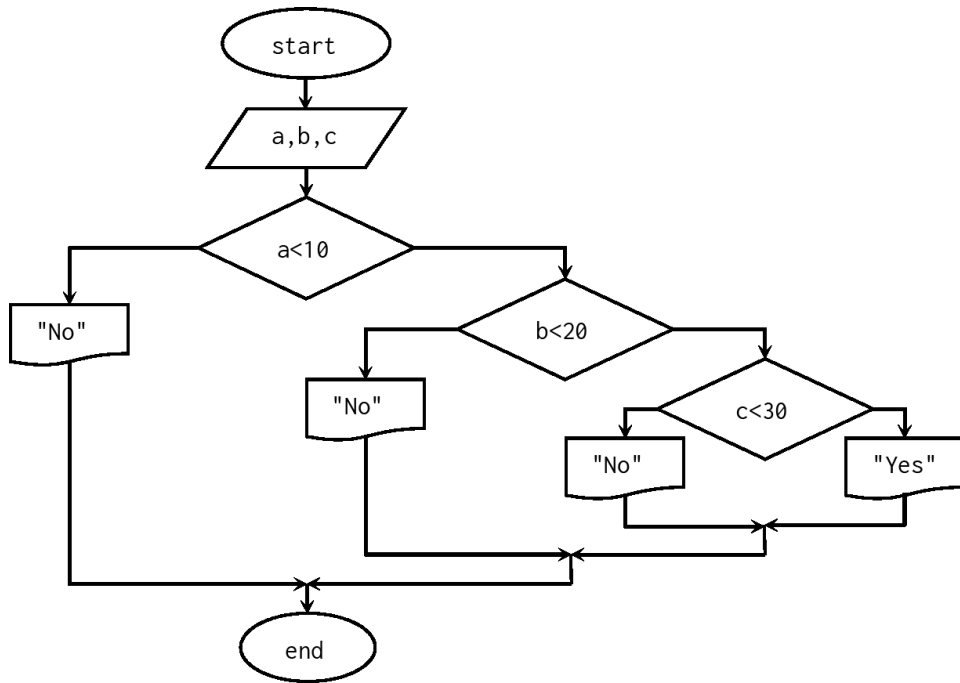
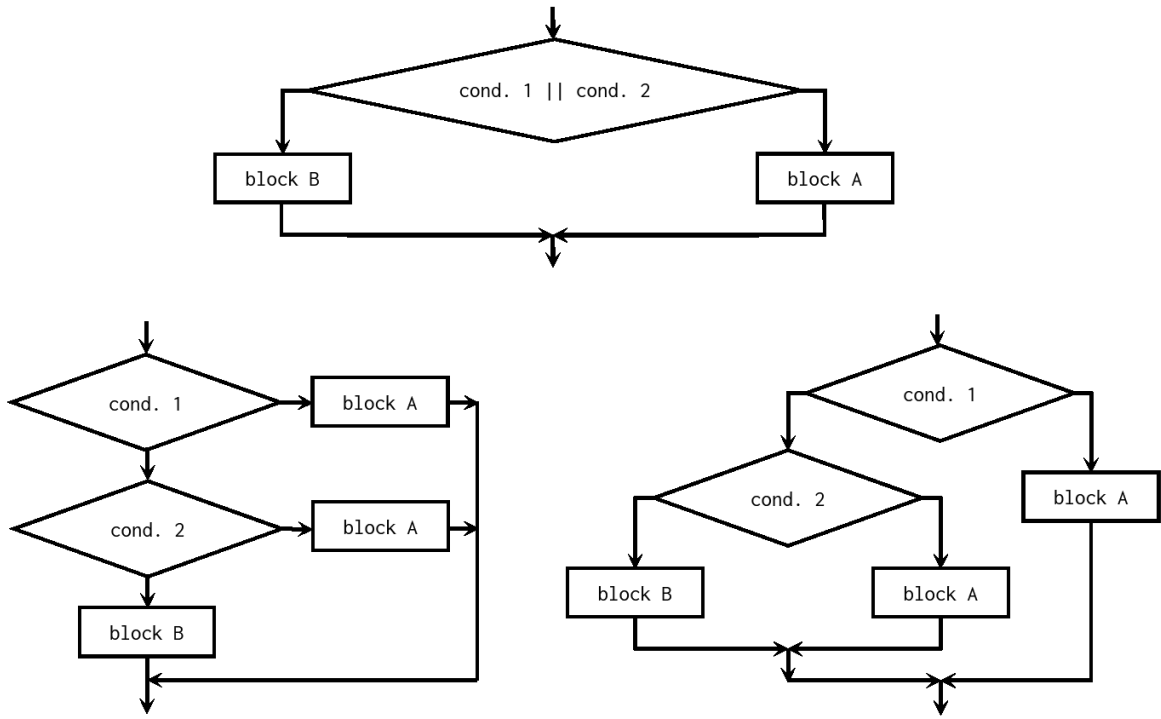
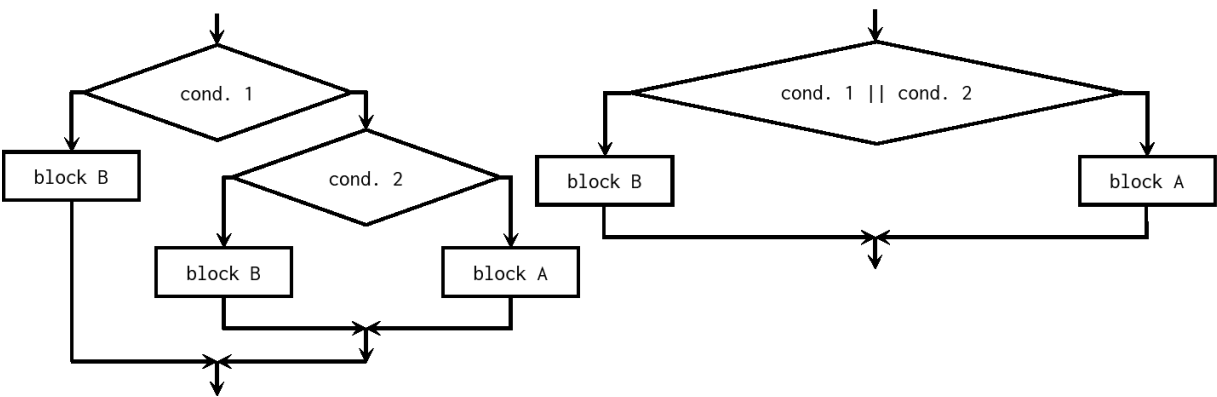


Fig. 4.8(c): A flowchart equivalent to [Figure 4.8\(c\)](#) with the nested if-else templates.

The rule of merging the conditions by the || operator: The following three flowcharts are equivalent.



The rule of merging the conditions by the && operator: The following two flowcharts are equivalent.



4.8.1. Exercise. Redraw **Flowchart 4.8(c)** negating its conditions, and translate it into both C++ and Java codes.

Based on the recent discussions, we can impose a rule for merging the && and || operators.

4.9. Example. Draw a flowchart to read the coefficients a , b , and c of the quadratic equation $ax^2 + bx + c = 0$, in which a is assumed to be nonzero, and then calculate and print the roots accompanied by the appropriate headings.

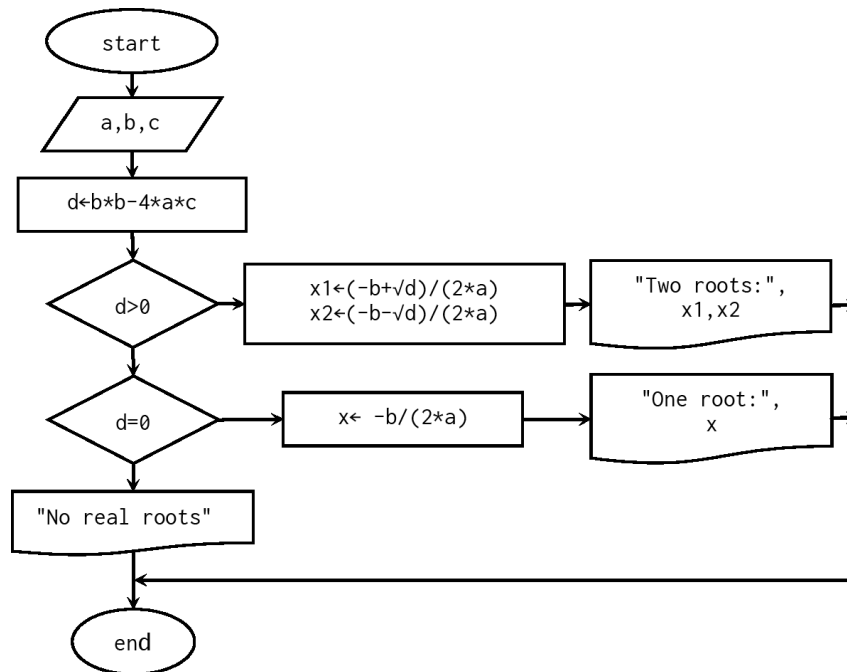


Fig. 4.9: Calculating and printing the roots of the quadratic equation $ax^2 + bx + c = 0$.

Solution. Let $d = b^2 - 4ac$ is the discriminant of this polynomial. Then, the three arguments are obtained as follows:

1. If $d > 0$, we have two distinct real roots:

$$x_1 = \frac{-b + \sqrt{d}}{2a}, \quad x_2 = \frac{-b - \sqrt{d}}{2a}$$

$$x_1 = \frac{-b + \sqrt{d}}{2a}, \quad x_2 = \frac{-b - \sqrt{d}}{2a}$$

2. If $d = 0$, the equation has only one real root:

$$x = \frac{-b}{2a}$$

3. If $d < 0$, there is no real root for the equation.

The above arguments are illustrated in **Flowchart 4.9**.

Sometimes, the algorithm of certain problems are not derived from a special idea, instead, they are easily written by one or more formulas, and probably followed by a short discussion. In such algorithms one only needs to pay attention to the correct use of the formulas and the related discussions. So far, all the examples were approximately of this type. The translations of **Flowchart 4.9** into both C++ and Java codes are displayed in Programs P4_9.

C++ codes:

```
// Program P4_9 to find the
// roots of an equation.
#include <iostream>
#include <math.h>
using namespace std;
int main() {
    float a, b, c, d;
    double x1, x2, x;
    cout<<"Enter the coefficients: ";
    cin>>a>>b>>c;
    d=b*b-4*a*c;
    if (d>0) {
        x1=(-b+sqrt(d))/(2*a);
        x2=(-b-sqrt(d))/(2*a);
        cout<<"Two roots: "<<x1
            <<" and "<<x2;
    }
    else if (d==0) {
        x=-b/(2*a);
        cout<<"One root: "<<x;
    }
    else
        cout<<"No real roots!";
    return 0;
}
```

Input/output:

```
Enter the coefficients: 1 4 -5↵
Two roots: 1 and -5
```

Java codes:

```
// Program P4_9 to find the
// roots of an equation.
import java.util.Scanner;
class P4_9 {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        float a, b, c, d;
        double x1, x2, x;
        System.out.print("Enter the "
            + "coefficients: ");
        a=read.nextFloat();
        b=read.nextFloat();
        c=read.nextFloat();
        d=b*b-4*a*c;
        if (d>0) {
            x1=(-b+Math.sqrt(d))/(2*a);
            x2=(-b-Math.sqrt(d))/(2*a);
            System.out.print("Two roots: " + x1
                + " and " + x2);
        }
        else if (d==0) {
            x=-b/(2*a);
            System.out.print("One root: " + x);
        }
        else
            System.out.print("No real roots!");
        read.close();
    }
}
```

Input/output:

```
Enter the coefficients: 1 4 -5↵
Two roots: 1.0 and -5.0
```

In some cases, the nature of the given problem requires using the consecutive conditional templates. Therefore, we should be careful not to fall into the trap of the nested conditional templates. The next example demonstrates such a case.

4.10. Example. Draw a flowchart to read three numbers of a , b , and c , sort them in ascending order, and finally, print the sorted numbers.

Solution. Every individual can propose an exclusive solution to this problem. To this end, a method called “*the bubble sorting algorithm*” is employed. In this method, starting from the first number, two consecutive numbers are compared each time. If the former one is greater than the latter, then we swap the two numbers. Afterwards, we go one number forward and repeat the same process for the new pair of numbers. Accordingly, the biggest number, among the others, is transferred to the last position, that is, it rises up like a bubble. Then, this number is put aside and the process is repeated for the remaining numbers until only one number is left. Therefore, the numbers are sorted in ascending order. The swap algorithm (see [Chapter 1](#)) is used to swap the consecutive numbers.

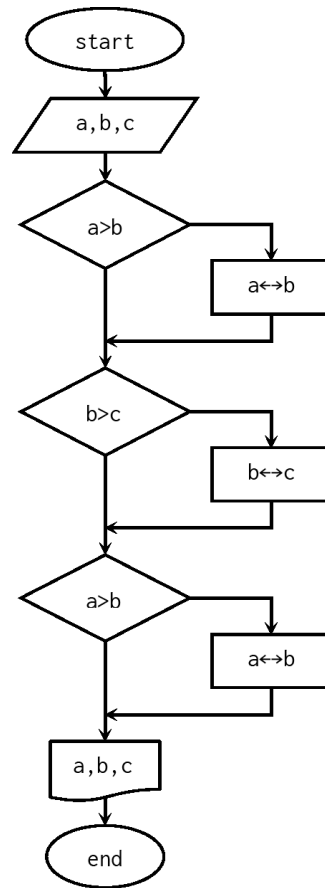


Fig. 4.10: Bubble sorting method for three numbers.

The above-mentioned method is used for the three given numbers a , b , and c . In the first step, a is compared with b . If a is greater than b , then they are swapped. Afterwards, b is compared with c . They swap only if b is greater than c . To this point, the biggest number, among others, is shifted to the end. In the second step, the values of b and c (probably the new ones) are compared and then swapped if the former is greater than the latter one. The reaction of the brain to perform this process is too quick to trace. The required flowchart is depicted in **Figure 4.10**. In **Chapter 8**, we will focus on using this method for any set of numbers.

Worth to mention that the first and third if templates are the same in Flowchart 4.10. In fact, the values of a and b may vary from their previous values after exiting the second if template. This fact is understood if the implementation table is arranged for $a = 12$, $b = 8$ and $c = 5$. Program P4_10 is the translation of Flowchart 4.10 into both C++ and Java codes.

C++ codes:

```
// Program P4_10 to sort three
// numbers by the bubble method
#include <iostream>
using namespace std;
int main() {
    float a, b, c, t;
    cout<<"Enter three numbers: ";
    cin>>a>>b>>c;
    if (a>b)
        {t=a; a=b; b=t;}
    if (b>c)
        {t=b; b=c; c=t;}
    if (a>b)
        {t=a; a=b; b=t;}
    cout<<"The sorted numbers: "
        <<a<<" ", "<<b<<" ", "<<c;
    return 0;
}
```

Input/output:

```
Enter three numbers: 23 -13 11↵
The sorted numbers: -13, 11, 23
```

Java codes:

```
// Program P4_10 to sort three
// numbers by the bubble method
import java.util.Scanner;
class P4_10 {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        float a, b, c, t;
        System.out.print("Enter three numbers: ");
        a=read.nextFloat();
        b=read.nextFloat();
        c=read.nextFloat();
        if (a>b)
            {t=a; a=b; b=t;}
        if (b>c)
            {t=b; b=c; c=t;}
        if (a>b)
            {t=a; a=b; b=t;}
        System.out.print("The sorted numbers: "
            + a +", " + b+ ", " + c);
        read.close();
    }
}
```

Input/output:

```
Enter three numbers: 23 -13 11↵
The sorted numbers: -13, 11, 23
```

4.10.1. Exercise. Apply the bubble sorting method to draw a flowchart in order to sort four numbers in ascending order. Further, write the codes of its program.

4.4 The switch statement

4.11. Observation. This time start with a program. Suppose that the five ranks of *A*, *B*, *C*, *D*, and *E* corresponds to the titles very good, good, average, weak, and failed, respectively. Programs P4_11_A receive the earned rank by the user and determine and print its corresponding title using the if-else-if statement. Moreover, the programs are written in a way to identically treat both uppercase and lowercase letters.

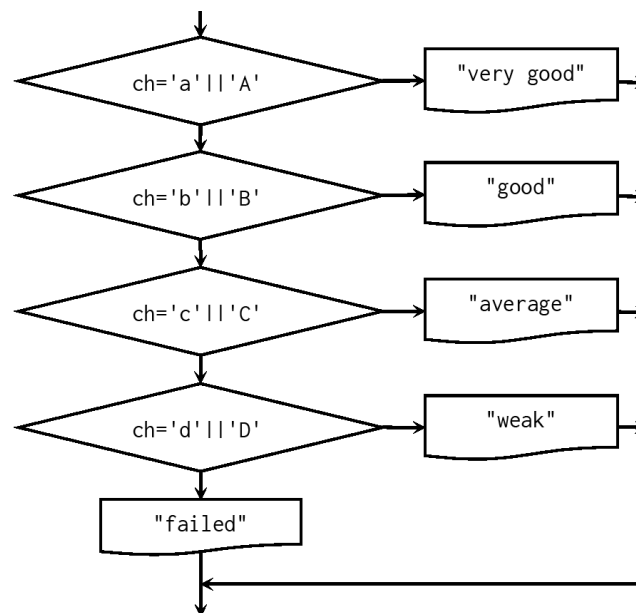


Fig. 4.11(a): Flowchart of Program P4_11 using the if-else-if template.

C++codes:

```
// Program P4_11_A to specify
// a title for any given rank
// using the if-else-if statement
#include <iostream>
using namespace std;
int main() {
    char ch;
    cout<<"Enter your rank: ";
    cin>>ch;
    if (ch=='a' || ch=='A')
        cout<<"very good";
    else if (ch=='b' || ch=='B')
        cout<<"good";
    else if (ch=='c' || ch=='C')
        cout<<"average";
    else if (ch=='d' || ch=='D')
        cout<<"weak";
    else
        cout<<"failed";
    return 0;
}
```

Java codes:

```
// Program P4_11_A to specify
// a title for any given rank
// using the if-else-if statement
import java.util.Scanner;
class P4_11_A {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        char ch;
        System.out.print("Enter your rank: ");
        ch=read.next().charAt(0);
        if (ch=='a' || ch=='A')
            System.out.print("very good");
        else if (ch=='b' || ch=='B')
            System.out.print("good");
        else if (ch=='c' || ch=='C')
            System.out.print("average");
        else if (ch=='d' || ch=='D')
            System.out.print("weak");
        else
            System.out.print("failed");
        read.close();
    }
}
```

The flowchart of Programs P4_11 (Fig. 4.11(a)) drawn with another pattern in Figure 4.11(b). These two flowcharts are equivalent from the executive point of view. In cases

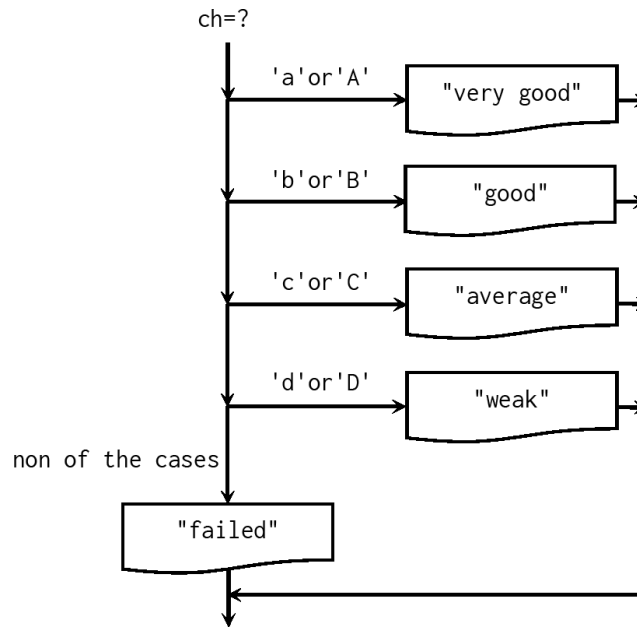


Fig. 4.11(b): Flowchart of Program P4_11 using the switch template.

where we are dealing with this kind of multi-way branching, we use the **switch template** with a flowchart similar to the one displayed in **Figure 4.11(b)**.

Programs P4_11_B are the same as Programs P4_11_A. The only difference is that in the latter programs the switch template substitutes for the if-else-if template in the former one.

C++codes:

```
// Program P4_11_B to specify
// a title for any given rank
// using the switch statement
#include <iostream.h>
using namespace std;
int main() {
    char ch;
    cout<<"Enter your rank: ";
    cin>>ch;
    switch (ch) {
        case 'a': case 'A':
            cout<<"very good";
            break;
        case 'b': case 'B':
            cout<<"good";
            break;
        case 'c': case 'C':
            cout<<"average";
            break;
        case 'd': case 'D':
            cout<<"weak";
            break;
    }
```

Java codes:

```
// Program P4_11_B to specify
// a title for any given rank
// using the switch statement
import java.util.Scanner;
class P4_11_B {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        char ch;
        System.out.print("Enter your rank: ");
        ch=read.next().charAt(0);
        switch (ch) {
            case 'a': case 'A':
                System.out.print("very good");
                break;
            case 'b': case 'B':
                System.out.print("good");
                break;
            case 'c': case 'C':
                System.out.print("average");
                break;
            case 'd': case 'D':
                System.out.print("weak");
                break;
        }
```

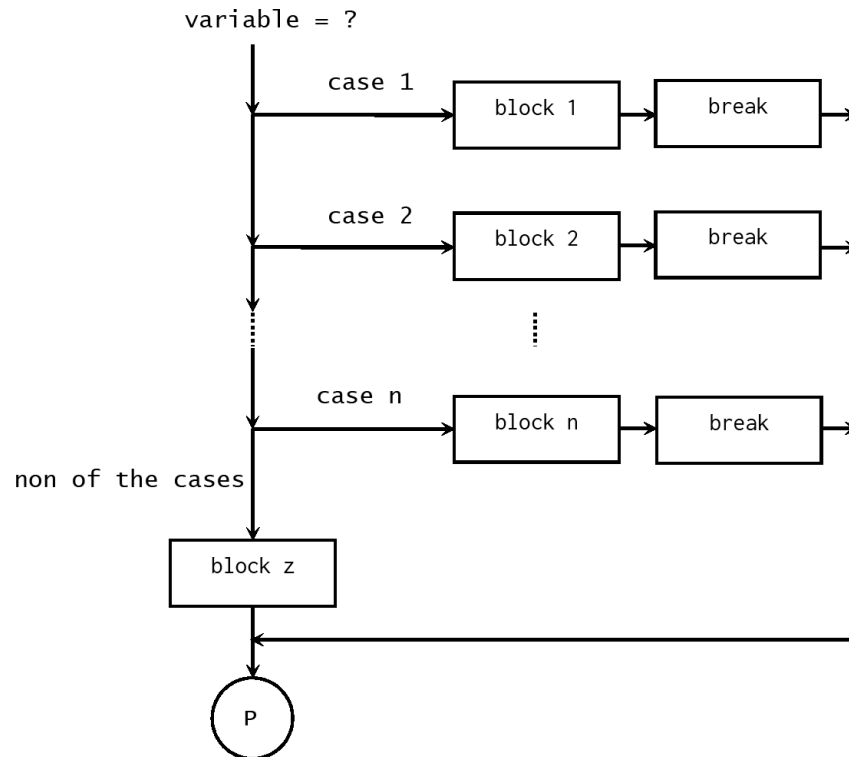


Fig. 4.11(c): The switch template.

```

default:
  cout<<"failed";
  break;
}
return 0;
}

break;
default:
  System.out.print("failed");
  break;
}
read.close();
}
}

```

In general, we use the switch template with the flowchart as in **Figure 4.11(c)** in the multi-way branching.

Implementation: If the variable satisfies in

- the case 1, then, block 1 first runs, afterwards, the break statement terminates this case and finally, the control transfers to position P;
- the case 2, then, block 2 first runs, then the break statement terminates such a case and the control transfers to position

P;

...

- the case n , then, block n is run, the break statement terminates the case, and eventually, the control transfers to position P;
- none of the cases, then, block z is first run, the break statement terminates this case, and the control transfers to position P.

Translating of the switch template into both C++ and Java codes is written in the following parts.

C++ codes:

```
Switch ( variable ) {  
  case case 1 :  
    block 1  
    break;  
  case case 2 :  
    block 2  
    break;  
  . . .  
  case case n :  
    block n  
    break;  
  default  
    block z  
}
```

Java codes:

```
Switch ( variable ) {  
  case case 1 :  
    block 1  
    break;  
  case case 2 :  
    block 2  
    break;  
  . . .  
  case case n :  
    block n  
    break;  
  default  
    block z  
}
```

Several notes should be highlighted regarding the switch template.



- The variable of the switch statement only admits one of the two int or char data types. Therefore, we should use the conditional templates if we want to work with other types of data.
 - No block needs to be grouped by {} since the break statement terminates the corresponding case.
 - Even the break may be ignored in block z.
 - The block z along with the break keyword may not exist.
 - Not every case needs to contain a break. If no break appears, the flow of control *falls through* to the subsequent case until a break is appeared.
-

Additionally, each of the case 1, case 2, ..., case n expressions should be in the same data type as the variable. In addition, each of these expressions may have several cases. In this circumstance, each expression is stated in the range of a case and the establishment of one of these expressions for the switch variable suffices to run the corresponding block. For example, consider the program parts below.

C++ codes:

```
switch (m) {
  case 1: case 2: case 3:
    cout<<"one, two, or three";
    break;
  case 4:

    cout<<"four";
    brak;
  default:
    cout<<"not known";
}
```

Java codes:

```
switch (m) {
  case 1: case 2: case 3:
    System.out.print("one, two, or three");
    break;
  case 4:

    System.out.print("four");
    brak;
  default:
    System.out.print("not known");
}
```

If the value read for the integer m is:

- equal to 1, 2, or 3, then the phrase “one, two, or three” is printed;
- equal to 4, then phrase “four” is printed;
- a value other than the above values, then the phrase “not known” is printed.

Besides the above application, the **break statement**, as a jumping statement, has another usage. Other applications of the break statement will be discussed in details in **Chapters 6** and **7** where the loops are studied.

4.5 More applications of the if template

In this section, further applications of the if template are explained in which the if-range is cut by either a jumping or termination statement. Additionally, in the subsequent subsections, both jumping goto and exit(0) statements are presented. Indeed, these statements are single reserved words.

4.5.1 Transferring the program execution

To transfer (jump) from one part of the program to the other one (after or before the position of the statement), there are statements in both C++ and Java languages with the following syntaxes.

C++ codes:

```
goto_label;
```

Java codes:

```
break_label;
```

Execution: Resume the program at the statement with the mentioned label. In other words, the running control is transferred to the statement with the mentioned label.



- A label can be an identifier or a positive number often from 1 to 255.
- A statement with the mentioned label should appear in the program. This statement is written as below:

label : the desired statement ;

- The label may correspond to a block of statements:

label : { block of statements }

4.12. Example. In Programs P4_12_A we employ the two above-mentioned statements.

C++ codes:

```
// Program P4_12 to apply
// the goto statement
#include <iostream>
using namespace std;
int main() {
    int n, g=1000;
    cout<<"Enter a nonnegative integer: ";
    cin>>n;
    switch (n) {
        case 0:    goto C; break;
        case 10:   goto B; break;
        case 100:  goto A; break;
        default:   cout<<"Out of range";
                  goto X;
    }
    A: g=g+n;
    B: g=g+n;
    C: g=g+n;
    cout<<g;
    X: {};
    return 0;
}
```

Input/output:

```
Enter a nonnegative integer: 10↵
1020
```

Java codes:

```
// Program P4_12 to apply
// the break statement
import java.util.Scanner;
class P4_12 {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int n, g=1000;
        System.out.println("Enter a "
            + "non-negative integer: ");
        n=read.nextInt();
        X: {
            A: {
                B: {
                    C: {
                        switch (n) {
                            case 0:    break A;
                            case 10:   break B;
                            case 100:  break C;
                            default:   System.out.print(
                                "Out of range");
                                break X;
                        }
                    }
                }
            }
            g=g+n;
        }
        g=g+n;
    }
    g=g+n;
    System.out.println(g);
}
read.close();
}
```

Input/output:

```
Enter a nonnegative integer: 10↵
1020
```

This program in both C++ and Java codes, reads the nonnegative integer n and then, if the value of n ,

- is 0, the statement with the label C is executed and the value 1000 is printed for g ;
- is 10, the program resumes at the statement labelled B and therefore, the value 1020 printed for g ;

- is 100, the program is continued from the statement with the label A and the value 1300 is printed for *g*;
- otherwise, the message Out of range is printed and the program is terminated .

Question. Why are the break statements practically useless in the left program?

Answer: Look for the answer within the goto statement.

Unless necessary, attempt not to use the goto statement since it decreases the speed of the program execution and confuses its reader. Fortunately, we rarely need to use the goto statement since there are various loops to fill the gap of using the goto statement. Moreover, the goto statement is not employed throughout this book, except for several times. Although this statement is a Java keyword, it is not used in Java.

4.5.2 Terminating the program execution

The statements with the following syntaxes exist in the C++ and Java languages for successful termination of the program.

C++ codes:

```
exit(0);
```

Java codes:

```
System.exit(0);
```

Execution: Terminate the program execution at this point.



We should place the `stdlib.h` pre-processor statement at the beginning of the program in order to use the `exit(0)` statement in C++ codes.

4.13. Example. For example, we use the statement

C++ codes:

```
exit(0);
```

Java codes:

```
System.exit(0);
```

in Programs P4_12 instead of the statement

C++ codes:

```
goto B;
```

Java codes:

```
break B;
```

and thus Programs P4_13 are acquired. Explain the outputs of this program in any case!

C++ codes:

```
// Program P4_13 to use the
// exit(0) statement
#include <iostream>
using namespace std;
int main() {
    int n, g=1000;
    cout<<"Enter a nonnegative integer: ";
    cin>>n;
    switch (n) {
        case 0: goto C; break;
        case 10: cout<<"Exit...";
                exit(0); break;
        case 100: goto A; break;
        default: cout<<"Out of range";
                goto X;
    }
    A: g=g+n;
    B: g=g+n;
    C: g=g+n;
    cout<<g;
    X: {};
    return 0;
}
```

Input/output:

```
Enter a nonnegative integer: 10↵
Exit...
```

Java codes:

```
// Program P4_13 to use the
// System.exit(0) statement
import java.util.Scanner;
class P4_13 {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int n, g=1000;
        System.out.print("Enter a "
            + "non-negative integer: ");
        n=read.nextInt();
        X: {
            A: {
                B: {
                    C: {
                        switch (n) {
                            case 0: break A;
                            case 10: System.out.print(
                                "Exit... ");
                                System.exit(0);
                                break B;
                            case 100: break C;
                            default: System.out.print(
                                "Out of ange");
                                break X;
                        }
                    }
                }
            }
            g=g+n;
        }
        g=g+n;
        System.out.println(g);
    }
    read.close();
}
}
```

Input/output:

```
Enter a nonnegative integer: 10↵
Exit...
```

In either program, label B and the related break statement are useless.

4.14. Example. Write an algorithm to read the Cartesian coordinates of a point with the names a and b , respectively, determine the position of that point in the coordinate plane, and finally, print it with appropriate headings.

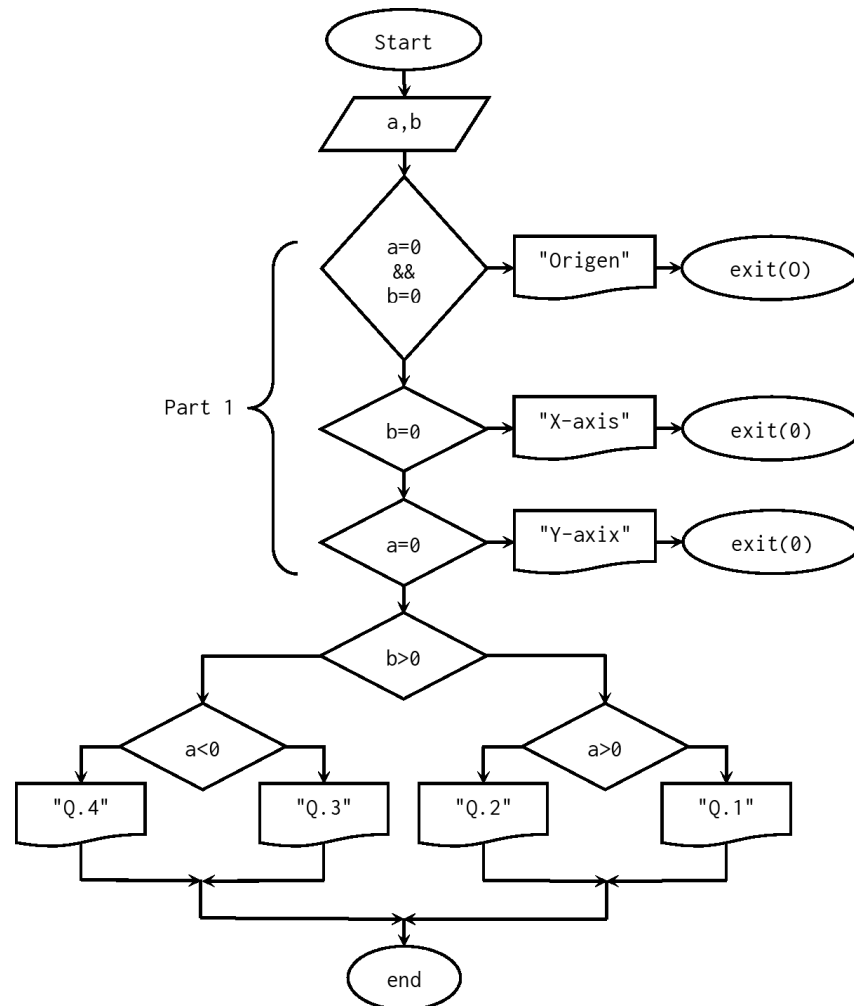


Fig. 4.14: Determine and printing the position of a point in the coordinate plane.

Solution. We draw the flowchart in a way that if the point is at the origin, on the x-axis, or on the y-axis in Part 1, the algorithm is terminated by separately printing them with appropriate headings using the `exit(0)` statement in C++ and `System. exit(0)`

statement in Java. Otherwise, the position of the point is determined and printed by mentioning the number of the coordinate quarter. The flowchart is depicted in **Figure 4.14**. Programs P4_14 are the translation of this flowchart into both C++ and Java codes.

C++ codes:

```
// Program P4_14 to indicate the
// position of a point on the XY-axis
#include <iostream>
#include <stdlib.h>

using namespace std;
int main() {
    float a, b;
    cout<<"Enter the XY-coordinates: ";
    cin>>a>>b;
    if (a==0 && b==0) {
        cout<<"Origin";
        exit(0);
    }
    if (b==0) {
        cout<<"X-axis";
        exit(0);
    }
    if (a==0) {
        cout<<"Y-axis";
        exit(0);
    }
    if (b>0)
        if (a>0)
            cout<<"Quarter 1";
        else
            cout<<"Quarter 2";
    else
        if (a<0)
            cout<<"Quarter 3";
        else
            cout<<"Quarter 4";
    return 0;
}
```

Java codes:

```
// Program P4_14 to indicate the
// position of a point on the XY-axis
import java.util.Scanner;
class P4_14 {

    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        float a, b;
        System.out.print("Enter the "
            + "XY-coordinates: ");
        a=read.nextFloat();
        b=read.nextFloat();
        if (a==0 && b==0) {
            System.out.print("Origin");
            System.exit(0);
        }
        if (b==0) {
            System.out.print("X-axis");
            System.exit(0);
        }
        if (a==0) {
            System.out.print("Y-axis");
            System.exit(0);
        }
        if (b>0)
            if (a>0)
                System.out.print("Quarter 1");
            else
                System.out.print("Quarter 2");
        else
            if (a<0)
                System.out.print("Quarter 3");
            else
                System.out.print("Quarter 4");
        read.close();
    }
}
```

Of course, Part 1 in Flowchart 4.13 can be considered as an if-else-if template and translated to both codes as follows.

C++ codes:

```
if (a==0 && b==0) {
    cout<<"Origin";
    exit(0);
}
else if (b==0) {
    cout<<"X-axis";
    exit(0);
}
else if (a==0) {
    cout<<"Y-axis";
    exit(0);
}
```

Java codes:

```
if (a==0 && b==0) {
    System.out.print("Origin");
    System.exit(0);
}
if (b==0) {
    System.out.print("X-axis");
    System.exit(0);
}
if (a==0) {
    System.out.print("Y-axis");
    System.exit(0);
}
```

4.14_1. Exercise. In the codes of Part 1 presented in the above codes, what results are obtained if one, two, or all the termination statements are removed? Discuss the answer in all cases.

Exercises

In the following exercises: arrange the implementation table, if needed, write the complete program, and provide appropriate input notifications and output headings, if any. Writing an algorithm stands for drawing its flowchart.

4.1. Write an algorithm to read two real numbers and print the integer numbers 1, -1, or 0 if their multiplication is positive, negative, or zero, respectively.

4.2. Write an algorithm to read an integer n and if the number is a multiple of 3, then print the output below:

The value of n is a multiple of 3

A similar message is printed if it is a multiple of 5 or 7. Note that a number can simultaneously be the multiple of two or all the numbers 3, 5, and 7. In this case, two or three messages are printed.

4.3. Write an algorithm to read a positive integer n and then determine whether the number is a full square or not by printing one of the messages YES or NO.

4.4. Write an algorithm to read the real numbers A , B , and C and print the message YES in the case that these three numbers form the edges of a triangle; otherwise, print the message NO.

4.5. Repeat the **exercise 4.4** for a right-angle triangle.

4.6. Repeat the **exercise 4.4** for an equilateral triangle.

4.7. Repeat the **exercise 4.4** for an isosceles triangle.

4.8. Write an algorithm to read the radius of a circle, calculate the area of the circle, inscribed square, and circumscribed square of the circle, and finally, print them.

4.9. Write an algorithm to read three integers. Then, calculate and print the average of these integers if the first number is even; otherwise, calculate and print the squares sum of the integers.

4.10. The number of digits of a positive integer is equal to one plus the integer part of its logarithm (based on 10). Write an algorithm to read an arbitrary integer, then calculate and print the number of its digits.

4.11. Write an algorithm to read an integer number n . Then,
– if n is negative and its last digit is 0 or 5 (i.e., it is divisible by 5), print the output:

The value of n is negative and divided by 5

– if n is non-negative and its last digit is 0 or 5, print the output:

The value of n is non-negative and divided by 5

– if n is none of the above, print the output:

The value of n is not divided by 5

4.12. Assuming that the year 1980 is a leap year, write an algorithm to read the number n , as a year (after or before the year 1980), and determine if it is a leap year. If yes, print YES; otherwise, print NO.

4.13. Write an algorithm to print the number of days of a month by reading the count n of that month. Suppose the year is not a leap year. If n is not in the range of 1 to 12, print the message: Illegal number.

4.14. Write an algorithm to read the two numbers m , (as months), and d , (as days), and then calculate and print the number of a day in a year which corresponds to the d -th day of the m -th month (e.g., by reading 2 and 21, print: 52).

4.15. In **Exercise 4.14**, modify the algorithm in a way to print the message Illegal month if the month number m is in the illegal rang and the message Illegal day if the day number n is in the illegal range; otherwise, the process in **Exercise 4.14** is performed.

In the **exercises 4.16** and **4.17** take 0 for Monday, 1 for Sunday, ..., and 6 for Friday.

4.16. Assuming that the first day of the current year is Saturday, write an algorithm to read n , (as the count of a day in the year), and then determine and print which day of the week it is.

4.17. Repeat the **exercise 4.16** in the case that the first day f of the year is either read.

4.18. Totally 90 two-digit numbers 10 to 99 are known to exist. Write an algorithm to read the positive integer n . Then, print the words First, Second, or Third if it is one of the first, second, or third 30 numbers, respectively. Eventually, print the message Not two-digit number if n is not a two-digit number. Do this task in duplicate using the switch and if-else-if templates.

4.19. Write an algorithm to read each time a pair of numbers m , (as a month), and d , (as a day). Then, print the message Illegal

month if the month number m is in the illegal range and the message Illegal day if the day number n is in the illegal range; otherwise, calculate and print a day of the week corresponding to the d -th day of the m -th month. The termination happens whenever the read values for m and d are both zero.

4.20. Turn your computer input into a calculator: write an algorithm to read a real number a , a character, and a real number b , each separated with a blank space and if that character is,

- +, calculate and print the value of $a + b$;
- -, calculate and print the value of $a - b$;
- *, calculate and print the value of $a * b$;
- /, calculate and print the value of a / b (announce a division by zero with the message Divided by zero);
- otherwise, print: illegal character

Repeat the above process until the character ‘.’ is entered.

4.21. Write an algorithm to read a letter c and if the input for c is,

- one of the letters b or B, print Black;
- one of the letters w or W, print White;
- one of the letters r or R, print Red;
- one of the letters g or G, print Green;
- or if other than the letters above, print Not in range.

Afterwards, asks the user if they want to exit the program, by printing the message

Quit?

Terminate the program if the letter n or N is entered and repeat the program from reading c if n or N is entered; if a letter other than the above four characters is entered, ask the user to type Y or N by printing the message:

Enter 'Y' or 'N':

5 Sub-algorithms and subprograms

5.1 Sub-algorithms

Assume that we know the programming for reading the entries of a matrix, as well as, calculating the sum and multiplication of two matrices along with the inverse and determinant of a matrix. Then, suppose that we were asked to write the following program:

Write a program to read a positive integer n with $n \geq 2$. Then, read the entries of the three $n \times n$ matrices A , B , and C , individually, where C is assumed invertible. Finally, compute and print the value of $\det(AB + C^{-1})$.

If we were to write the required task within one unit of such a large program, we would encounter several problems including: the complexity in writing the different parts and connecting to each other, repeating the same parts, and most importantly, testing the correctness of the program and correcting lots of runtime errors. This is, if not impossible, a very exhausting task.

What is the solution? Clearly, we can divide this program into several subprograms each performing a specific task. For example, in the recent program, we could use the subprograms with specific tasks as follows: (1) To read the entries of a matrix and save them in the memory, (2) To get two matrices, add them, and return the result, (3) To get two matrices, multiply them, and return the result, (4) To get a matrix, invert it, and return the result, and (5) To get a matrix, compute its determinant, and return the result. Afterwards, a main program manages the tasks of these subprograms. More precisely, the main program calls the above-mentioned subprograms in a reasonable order, to arrive at the requested value after reading the value of n (Tab. 5.1).

Tab. 5.1: Managing the subprograms by the main program

Number of subprogram	Input to subprogram	Procedure
(1)	A	saving in the memory
(1)	B	saving in the memory
(1)	C	saving in the memory
(3)	A and B	return AB as, say, X

(4)	C	return C^{-1} as , say, Y
(2)	X and Y	return $X + Y$ as, say, Z
(5)	Z	return $\det(Z)$

Eventually, the main algorithm, prints the final returned result and the program is terminated. Several motivations for using the sub-algorithms (subprograms) are mentioned below.

- Writing the algorithm becomes simpler since the tasks of a large calculation are distributed among the small sub-algorithms and a main algorithm plays the role of a director. Actually, writing the sub-algorithms and the main algorithm is simpler due to their small size and limited job;
- Testing the correctness of the algorithm is summarized to checking each of the small sub-algorithms and, therefore, the problem of testing the algorithm is resolved. In particular, correcting the algorithm becomes easier in the single sub-algorithms and the main algorithm if there are any probable errors;
- Repeating the same parts of the program is avoided and thus, the time is not wasted;
- Saving every sub-algorithm in auxiliary memories for using in other future programs, independently.



The rule of constructing the sub-algorithms. The sub-algorithms are written exactly like the algorithms and all the templates in writing the algorithms can be applied to the sub-algorithms.

In practice, a sub-algorithm does nothing unless another sub-algorithm or the main algorithm, which is named the “call unit”, calls it. In this case, none, one, or several values enter the sub-algorithm and thus, its implementation starts. As a result, one or both of the following are processed.

1. A task such as reading, printing, assigning, and the like is performed;
 2. One or several values return to the call unit.
-



The rule of starting the sub-algorithms. At the start of a sub-algorithm, the name of the sub-algorithm, along with the (formal) parameters inside a pair of parentheses is placed inside an ellipse shape which is the sub-

algorithm prototype in algorithm-writing. This instruction is called the defining instruction of the sub-algorithm.

For example,

Data_reader(a,b,c)

Power(x,n)

The parameters of a sub-algorithm are divided into two types. The first type, called *the value parameter*, brings values from the call unit to the sub-algorithm. All the parameters in Java are value parameters. In addition to the value parameters, there is a second type parameter in C++ called *the reference parameter* which sends out the results from the sub-algorithm to the call unit. Some value parameters may play the role of the reference parameters in C++ programs, which are discussed later on.

Throughout the present book, we start the main algorithm with the shape below:

start

In addition, it is sufficient to write only the name of parameters in algorithm-writing; however in programming, the data type of each parameter should be added before its name.



The rule of ending the sub-algorithms. When implementing a sub-algorithm is completed, the implementation control of the algorithm is shifted after the position from where the sub-algorithm was called.

The shape

return ?

is used at the end of a sub-algorithm instead of the shape



which was used at the end of the main algorithms.

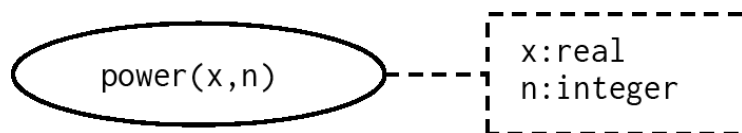
The carrier variable of the return single value is written instead of '?'. However, we have an empty return if there is no return value to the call unit. Although the return statement may not be written in programs of these circumstances, the empty return instruction is written in the algorithms for these cases. This preserves the standard shape of the flowcharts related to the sub-algorithms indicating that the sub-algorithms should have at least one return instruction.

When a sub-algorithm is called, the values (expressions) that are passed to the call unit are called the **actual parameters** or **arguments**. Later on, we will find that the types of sub-algorithms and the ways of their calling vary. Nevertheless, there is one common point in all the calling processes. In other words, the name of the sub-algorithm along with a pair of parentheses probably with several arguments inside occurs in the calling instruction of a sub-algorithm, for instance, `power(y,8)`. Further, following the parameters in the defining instruction of a sub-algorithm, the arguments in the calling instruction of a sub-algorithm inside the call unit are divided into value and reference types. The value arguments carry certain values to the sub-algorithm and the reference arguments take the results coming from the sub-algorithm. The terms "arguments" and "parameters" are used interchangeably. **Figure 5.0** displays the data flow from the call unit to the sub-algorithm (subprogram) and vice versa by the parameters and arguments.



The rule of parameter-argument. The number, order, and type of the parameters should be compatible with the number, order, and type of their corresponding arguments.

For example, in the defining instruction



considering the type of the defined parameters x and n , we can write: `power(y,8)` in the calling instruction in order to call the sub-algorithm for the real value y and

integer value 8 which correspond the value parameters x and n , respectively. However, none of the following forms is allowed:

`power(y, 8.0)`, `power(8,y)`, `power(y)`.



- Repetition is allowed in the arguments, however, none of the parameters can be repetitive. For instance, the following defining instruction is illegal.

`Roots(a, b, b)`

- The arguments and their corresponding parameters can be either homonymous or non-homonymous. For example, the form `power(x,n)` may occur as the form `power(u,k)` in the above-mentioned calling instruction.
-

5.2 Subprograms

The translation of sub-algorithm and main algorithm is called the subprogram and main program in programming, respectively. Similar to most programming languages, the order of writing the main program and the subprograms is not important in C++ and Java languages. However, professional programmers prefer to place the main program at the beginning of their programs. This strategy is followed in most programs of the current book.

We regard the main program and each of the sub programs as a unit. Furthermore, we may occasionally refer to the main program as the main unit and to each of the sub programs with the name of that unit. In particular, the call unit refers to a unit which calls a subprogram. Moreover, the call unit may be either the main program or a subprogram. By a complete program we mean the main program as well as all its subprograms, if any.

The variables declared in the main program and the subprograms are referred to as the global and local variables, respectively.

Depending on the number of the returned values, the sub-algorithms (subprograms) are classified into one-return (function), no-return (void) and multi-return types. There is no multi-return subprogram in Java programming language. The no-return (void) and 1-return subprograms are called **methods** in this

language. In other words, a method is a collection of statements which perform some specific tasks and frequently returns one result to the call unit. However, it is occasionally used to perform certain tasks such as reading, printing, assigning, and the like without returning anything. Think of a method as a subprogram which acts on the value parameters and mostly returns a value. The methods in the current book are mainly of this type.

The methods in Java act like the functions (i.e., no-return or 1-return) in C++. Therefore, the words “function” and “method” have the same meaning in programming. Of course, the method which is the synonym of the technique is excluded. In fact, the word “method” is commonly used after the Java programming language appearance and before that, the word “function” is used for this purpose.



Convention. Throughout this book,

- When we talk of sub-algorithms (subprograms), we mean all types including the functions or methods.
 - In the present book, the words “function” and “method” are interchangeably used, which refer to the same concept.
-

The value of each value argument is copied to the corresponding value parameter at the calling time. This type of parameter passing is called **pass-by-value**. We have already explained that all the Java parameters are value parameters since all the Java sub-algorithms (subprograms) are of function type. In other words, all the Java parameters are strictly pass-by-value while C++ has **pass-by-reference** parameters which refer to the reference parameter as well.

The flow of data passing from a subprogram to a call unit and vice versa in the multi-return subprograms is depicted in [Figure 5.0](#). It is worth mentioning that all the parameters are of value type in the functions (methods) and thus, the single value of the (1-return) function is probably returned by the return statement by means of the function carrier.

Definition:

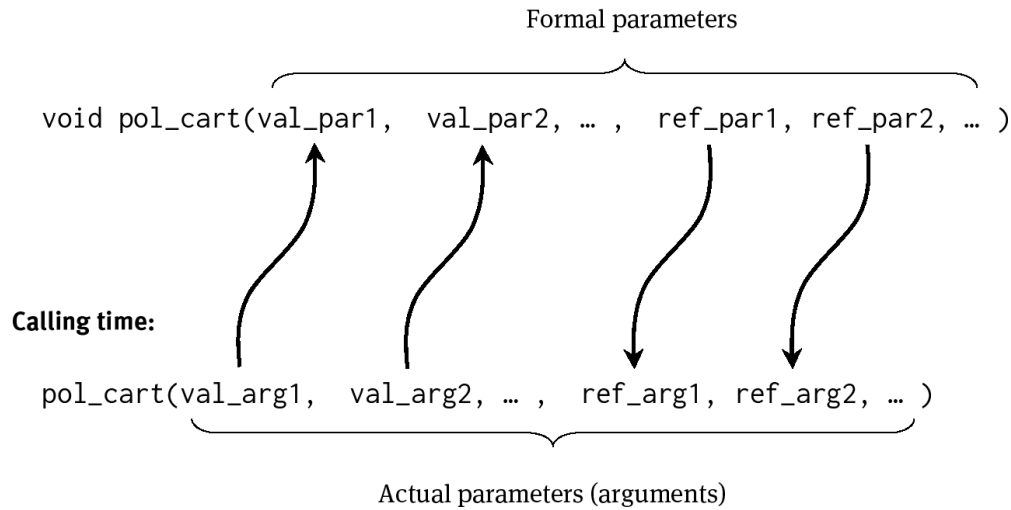


Fig. 5.0: The flow of data via the parameters and arguments in the multi-return subprograms.

Considering the above discussions, the classification of subprograms is revised and summarized in two categories, namely methods, which occur in both C++ and Java languages and multi-return sub-programs which are only dedicated to C++. In the coming chapters, the majority subprograms are function type.

5.2.1 Functions

We start with the function which exactly returns one value. This type of sub-algorithm is used when a programmer wants to construct a function which is not in the list of the library functions. Therefore, this category is referred to as the user-defined function. Furthermore, in cases where the return of only one value to the call unit is required it is advised to use this type of sub-algorithm since its construction and call are easy.

5.1. Example. The sign function for a real number x is defined as follows.

$$\text{sign}(x) = \begin{cases} -1, & \text{if } x < 0, \\ 0, & \text{if } x = 0, \\ 1, & \text{if } x > 0. \end{cases}$$

Write a function sub-algorithm to receive a real value and return sign(x).

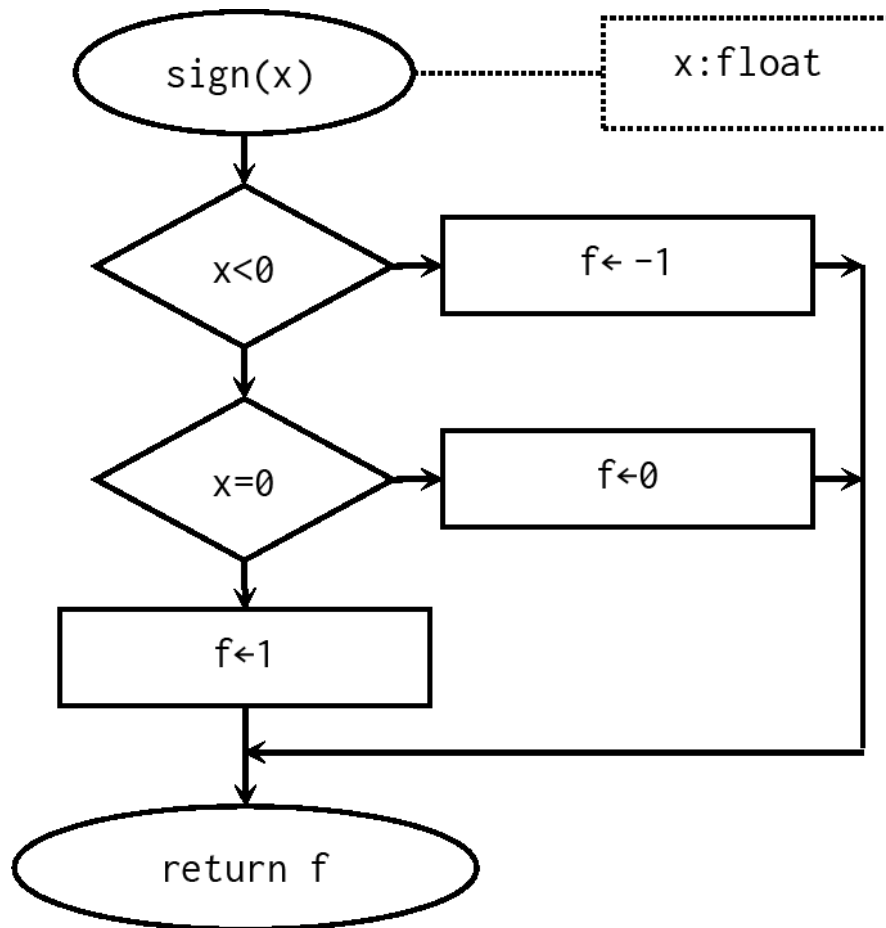


Fig. 5.1(a): The carrier name is different from the function name.

Solution. This function is clearly defined in **Flowchart 5.1(a)**. In this function the name of the carrier related to the result is different from the name of the function. Of course, we may select the same name for both carrier and function (**Fig. 5.1(b)**).

Moreover, if the carrier is to return the single value of a function, it is reasonable for the carrier to occur at least in one assignment instruction in order to obtain a value to return. Occasionally, there is no carrier and thus the return value of a function is directly returned. **Flowchart 5.1(c)** is of this type.

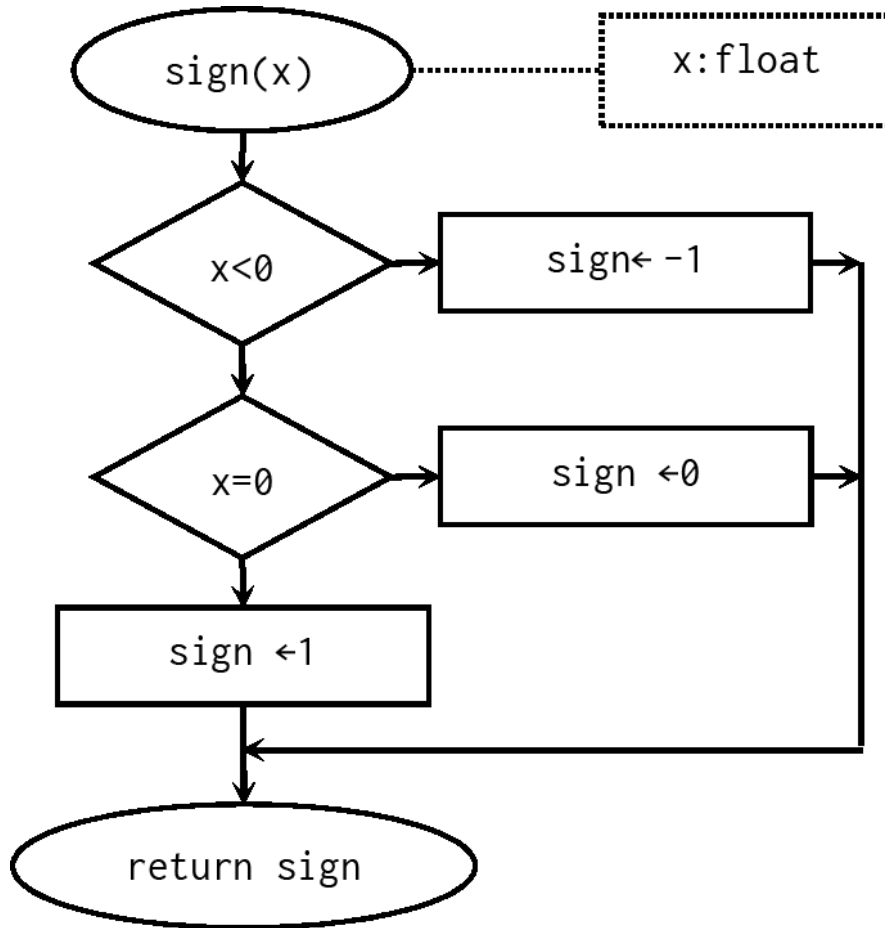


Fig. 5.1(b): The carrier name is homonym to the function name.

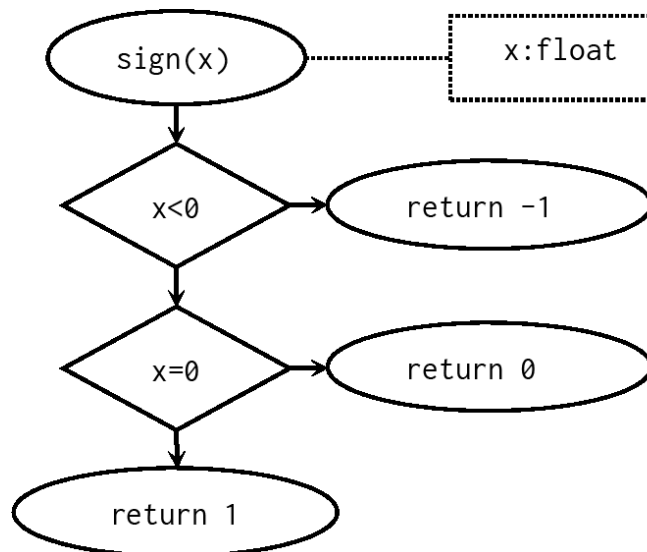


Fig. 5.1(c): There is no carrier and the return value is returned directly.



The syntax of the function (method) defining statement. The statement for defining a function subprogram in programming is written as follows.

C++ codes:

```
data type_name(parameters) {  
    function body  
}
```

Java codes:

```
modifier_static_data type_name(parameters) {  
    method body  
}
```

Recall from [Section 3.2](#) that the keyword `static` is used for the static methods for which the memory is allocated only once at the time of class loading. These methods are common to every object such that it is known as *member* or *class* method. Conversely, non-static methods for which the keyword `static` is not written, the memory is allocated multiple times whenever a method is called. These methods are specific to an object, therefore, they are known as *instance* methods. Based on the purpose of the present book, we use the `static` keyword in our methods.

The name of method is an identifier and should be selected following the rule of naming identifiers. In addition, the data type is the value type of the data returned to the call unit. In the void methods, the void keyword is written instead of data type. The following is an example of this type.

C++ codes:

```
void Sign(double x) {  
    if (x>0)  
        cout<<"Sign: 1";  
    else if (x=0)  
        cout<<"Sign: 0";  
  
    else  
        cout<<"Sign: -1";  
}
```

Java codes:

```
static void Sign(double x) {  
    if (x>0)  
        System.out.print("Sign: 1");  
    else if (x=0)  
        System.out.print("Sign: 0");  
  
    else  
        System.out.print("Sign: -1");  
}
```

In [Example 5.2](#) we find another function of this type.

We leave an empty parenthesis for the parameters if there is no parameter. An example of this type of function is as follows.

C++ codes:

```
double e() {
    return exp(1.0);
}
```

Java codes:

```
static double e() {
    return Math.exp(1.0);
}
```

In this book, all the functions except for a few examples have at least one parameter and exactly one return value. The form of parameters, if any, is explained below.



The rule of the parameter list in functions. All the parameters of a function, if any, are value parameters and the list of parameters is of the following pattern:

data type_first parameter, data type_second parameter, ..., data type_last parameter

As shown, two consecutive parameters are separated by a comma. For example, in the defining statement

C++ codes:

```
double Pad(float x, float y, int n)
```

Java codes:

```
static double Pad(float x, float y, int n)
```

the parameters x and y are declared as float type while the parameter n is declared as int type for the function Pad() and the data type of the return result for this function is double.

Given the above-mentioned discussions, [Flowcharts 5.1\(a\)](#), [5.1\(b\)](#), and [5.1\(c\)](#) are translated into both C++ and Java codes.

C++ codes:

```
// Subprogram 5.1(a)
int sign(double x) {
    int f;
    if (x<0)
        f=-1;
    else if (x==0)
        f=0;
    else
```

Java codes:

```
// Subprogram 5.1(a)
static int sign(double x) {
    int f;
    if (x<0)
        f=-1;
    else if (x==0)
        f=0;
    else
```

```

    f=1;
    return f;
}

// Subprogram 5.1(b)
int sign(double x) {
    int sign;
    if (x<0)
        sign=-1;
    else if (x==0)
        sign=0;
    else
        sign=1;
    return sign;
}

```

```

// Subprogram 5.1(c)
int sign(double x) {
    if (x<0)
        return -1;
    else if (x==0)
        return 0;
    else
        return 1;
}

```

```

    f=1;
    return f;
}

// Subprogram 5.1(b)
static int sign(double x) {
    int sign;
    if (x<0)
        sign=-1;
    else if (x==0)
        sign=0;
    else
        sign=1;
    return sign;
}

```

```

// Subprogram 5.1(c)
static int sign(double x) {
    if (x<0)
        return -1;
    else if (x==0)
        return 0;
    else
        return 1;
}

```



Note 1 of the functions. None of the parameters should be defined as a local variable anymore in the body of a subprogram. In other words, a parameter is simultaneously unable to be a local variable.

For example, writing a statement like

```
float x;
```

in the body of each of the three above-mentioned subprograms is illegal.



Note 2 of the functions. We should not read any parameter in the body of a sub-algorithm (subprogram) anymore; the value of any parameter reaches the sub-algorithm (subprogram) by the call unit.

For instance, reading x is completely illegal in each of Flowcharts or Subprograms 5.1(a), 5.1(b), and 5.1(c).



Note 3 of the functions. We should not print the return value, if any, in the body of a sub-algorithm (subprogram) anymore; this value is going to be returned to the call unit.

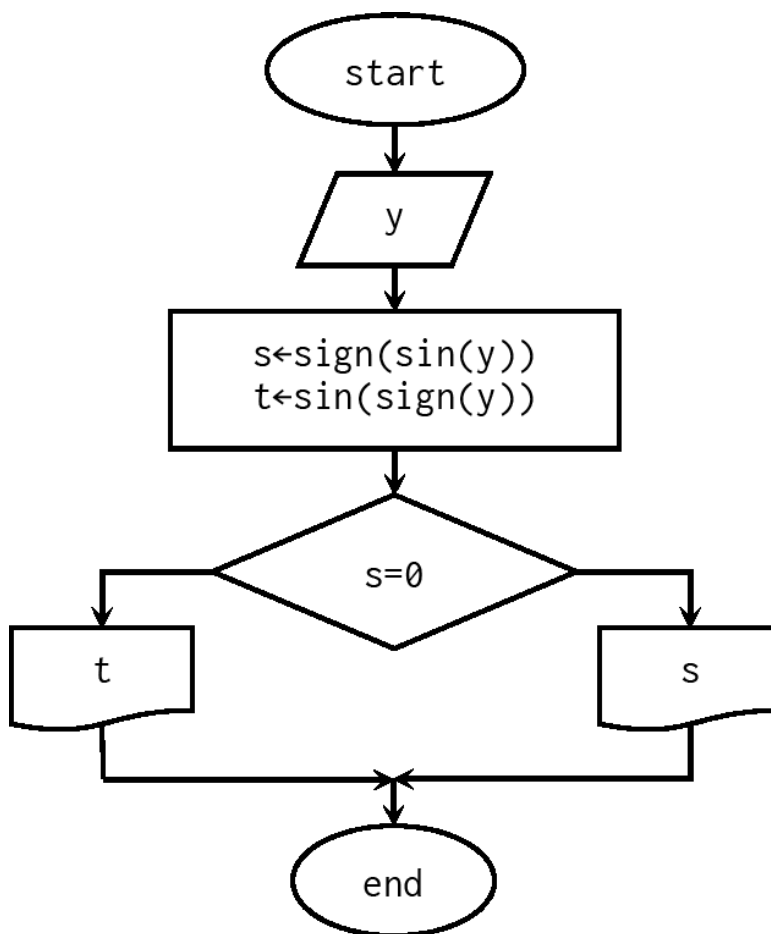


Fig. 5.1(d): A main program calling the function `sign()`.

For instance, printing f in **Flowchart or Subprogram 5.1(a)** as well as printing $sign$ in **Flowchart or Subprogram 5.1(b)** is illegal.

Now we explain how to call a function.



The rule of calling a 1-return function. Calling a function with a return value in algorithm-writing and programming is similar to calling a library mathematical function.

More precisely, a function with a return value is called by its name followed by the arguments which may occur in either of the following forms:

- In the list of an output statement;
- In a logical expression, especially, in an if template condition;
- In an arithmetic expression;
- As an argument of another function or a library function.

In general, the above-mentioned call phrase may occur anywhere a library function probably occurs. The occurrence of the above sign() function can be observed in the following examples:

```
if (sign(x)==0)    switch (sign(k))    System.out.println(sin(sign(t));  
u=sqrt(sign(y));  y=pow(x, sign(n));  cout<<sin(sign(t));
```

The sub-algorithm of the sign() function is called in the main algorithm of **Flowchart 5.1(d)**.

As shown, the function sign() and the library function sin() are equally treated upon calling. Programs P5_1 are the codes of the main Algorithm 5.1(d) along with the function sub-algorithm 5.1(a).

C++ codes:

```
// Program P5_1 containing two units:
// A function sub-program, sign,
// and a main program
#include <iostream>
#include <math.h>
using namespace std;
int sign(double x) {
    int f;
    if (x<0)
        f=(-1);
    else if (x==0)
        f=0;
    else
        f=1;
    return f;
}
//*****
int main() {
    double y, t;
    int s;
    cout<<"Enter a real number: ";
    cin>>y;
    s=sign(sin(y));
    t=sin(sign(y));
    if (s==0)
        cout<<"sign(sin("<<y<<"))="<<s;
    else
        cout<<"sin(sign("<<y<<"))="<<t;
    return 0;
}
```

Input/output:

```
Enter a real number: 1↵
sin(sign(1))=0.841471
```

Java codes:

```
// Program P5_1 containing two units:
// A method sub-program, sign,
// and a main program
import java.util.Scanner;
class P5_1{
    static int sign(double x) {
        int f;
        if (x<0)
            f=-1;
        else if (x==0)
            f=0;
        else
            f=1;
        return f;
    }
    //*****
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        double y, t;
        int s;
        System.out.print("Enter a real number: ");
        y=read.nextDouble();
        s=sign(Math.sin(y));
        t=Math.sin(sign(y));
        if (s==0)
            System.out.print("sign(sin(" + y
                + ")) = " + s);
        else
            System.out.print("sin(sign(" + y
                + ")) = " + t);
        read.close();
    }
}
```

Input/output:

```
Enter a real number: 1↵
sin(sign(1.0)) = 0.8414709848078965
```

In C++ programming language, the short function subprograms can immediately be written after the preprocessor statements using inline manner. Accordingly, the inline keyword is added before the defining statement. These subprograms frequently occupy a single line since they are short. In this regard, an example is provided as follows.

```
inline float h(float x) {return exp(x) + exp(-x);}
```

Typically, both of the following subprograms are equivalent to the above inline function.

```
float h(float x) {  
    return exp(x) + exp(-x);  
}
```

```
float h(float x) {  
    float h;  
    h=exp(x) + exp(-x);  
    return h;  
}
```

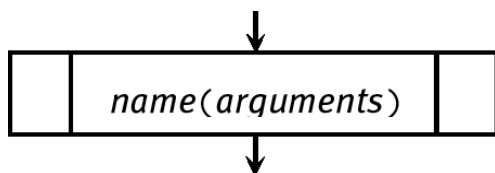
Two common methods exist for writing a complete program including the main program and its subprogram(s). The first way is to write the main program after all the subprograms. There is nothing to say about this way. In the second way, which is the preference of most programmers, the main program is written before all the subprogram(s).

In C++ language, all the subprograms should be predefined before the main program when using the second way. This can be performed using a predefined statement which is a copy of the defining statement with a semicolon at the end. Of course, we may remove the name of the parameters and only leave their data types. Actually, the predefined statement specifies to the compiler about the structure of the function and its parameters.

Calling a void function is quite different.



The rule of calling a void function. Calling a void function in algorithm-writing is conducted by an instruction with the following shape:



Similarly, a void function is called by a statement with the following syntax in programming: *name(arguments)*.

Similar to all the parameters, the arguments are all the value arguments. The parenthesis empty is left empty if there is no argument. The next example includes a void function.

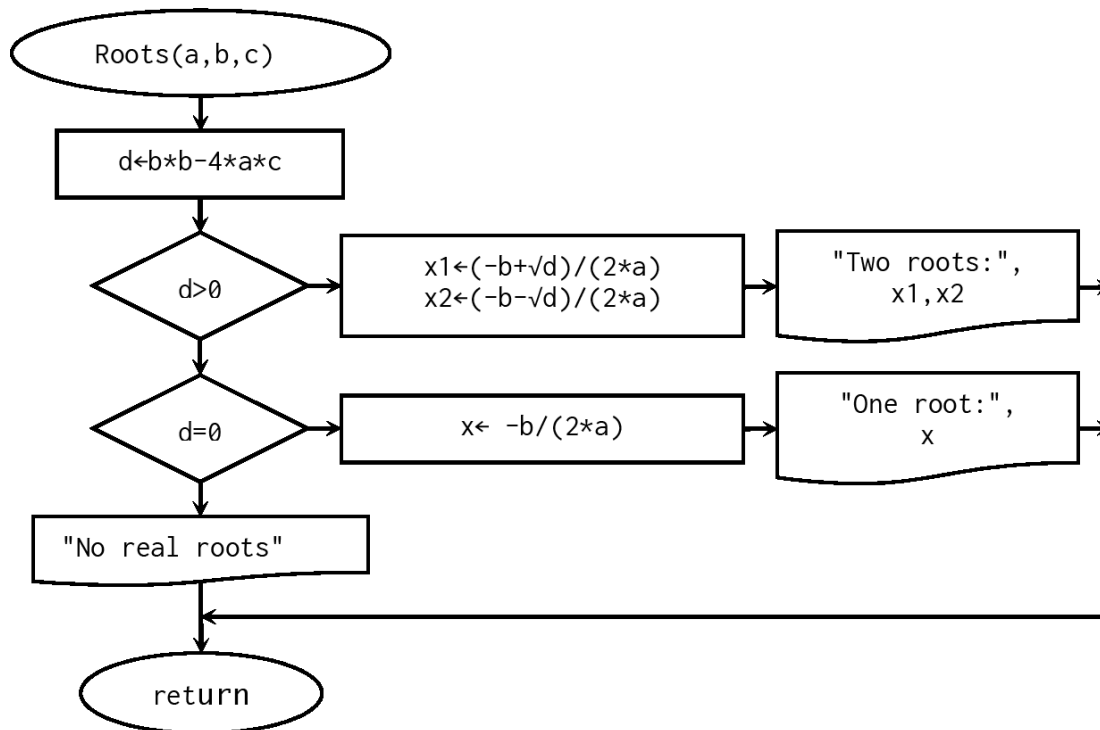


Fig. 5.2(a): Receiving the coefficients and determining the roots of $ax^2 + bx + c = 0$.

5.2. Example. Write a function named `Roots()` to receive the coefficients a , b , and c of the quadratic equation $ax^2 + bx + c = 0$ in which a is assumed to be nonzero and then calculate and print the roots accompanied by appropriate headings. Next, write another function named `Heading()` to print an appropriate heading for the above-mentioned function. Finally, write a main algorithm to call these sub-algorithms.

Solution. The first function has already been worked out in [Example 4.9](#). Remove the start and end parts as well as the reading instruction in [Flowchart 4.9](#) and take the resulted flowchart as the body of the required void function. Then, [Flowchart 5.2\(a\)](#) is obtained. [Flowchart 5.2\(b\)](#) is a sketch of printing a heading. This function has no return value or parameter.

Ultimately, the above functions are called in the main algorithm of [Flowchart 5.2\(c\)](#).

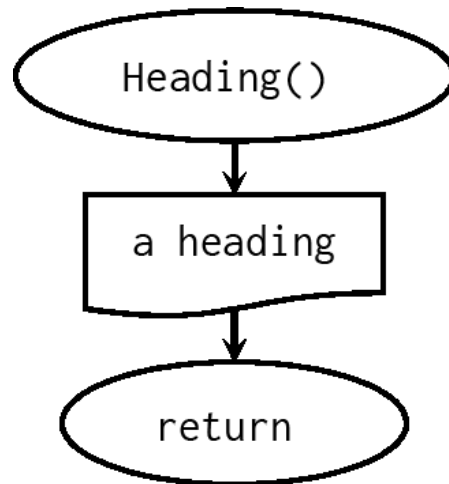


Fig. 5.2(b): Sketch of printing a heading.

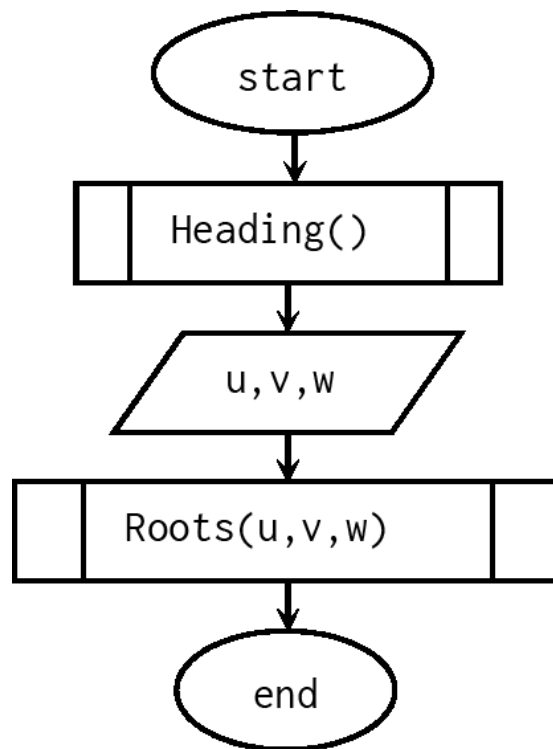


Fig. 5.2(c): Calling the two functions Roots() and Heading().

In this main algorithm, the related heading is first printed using the Heading function. Then, the quadratic polynomial coefficients named u , v , and w are read and the Roots function is called for these three values. Programs P5_2 are the codes of the main Algorithm 5.2(c) along with the two function sub-algorithms 5.2(a) and 5.2(b).

C++ codes:

```
// Program P5_2 containing two
// sub programs: "Heading" to print a
// heading and "Roots" to investigate
// and print the roots of a polynomial
#include <iostream>
#include <math.h>
using namespace std;
void Heading() {
    cout<<"*****"<<endl;
    cout<<"* Roots of      *"<<endl;
    cout<<"* a*x^2+b*x+c=0 *"<<endl;
    cout<<"*****"<<endl;
    return;
}
//*****
void Roots(double a, double b,
           double c) {
    double d, x, x1, x2;
    d=b*b-4*a*c;
    if (d>0) {
        x1=(-b+sqrt(d))/(2*a);
        x2=(-b-sqrt(d))/(2*a);
        cout<<"Two roots: "<<x1<<" , "<<x2;
    }
    else if (d==0) {
        x=-b/(2*a);
        cout<<"Just one root: "<<x;
    }
}
```

Java codes:

```
// Program P5_2 containing two
// sub programs: "Heading" to print a
// heading and "Roots" to investigate
// and print the roots of a polynomial
import java.util.Scanner;
class P5_2 {
    static void Heading() {
        System.out.println(
            "*****");
        System.out.println(
            "* Roots of      *");
        System.out.println(
            "* a*x^2+b*x+c=0 *");
        System.out.println(
            "*****");
    }
    //*****
    static void Roots(double a, double b,
                     double c) {
        double d, x, x1, x2;
        d=b*b-4*a*c;
        if (d>0) {
            x1=(-b+Math.sqrt(d))/(2*a);
            x2=(-b-Math.sqrt(d))/(2*a);
            System.out.print("Two roots: " + x1
                            + " , " + x2);
        }
        else if (d==0) {
```

```

else
    cout<<"No real roots";
return;
}
//*****
int main() {
    double u, v, w;
    Heading();
    cout<<"Enter the coefficients: ";
    cin>>u>>v>>w;
    Roots(u, v, w);
    return 0;
}

```

Input/output:

```

*****
* Roots of      *
* a*x^2+b*x+c=0 *
*****
Enter the coefficients: 1 3 -10↵
Two roots: 2, -5

```

```

x=-b/(2*a);
System.out.print("Just one root: " + x);
}
else
    System.out.print("No real roots");
}
//*****
public static void main(String[] args) {
    Scanner read=new Scanner(System.in);
    double u, v, w;
    Heading();
    System.out.print("Enter the "
        + "coefficients: ");
    u=read.nextDouble();
    v=read.nextDouble();
    w=read.nextDouble();
    Roots(u, v, w);
    read.close();
}
}

```

Input/output:

```

*****
* Roots of      *
* a*x^2+b*x+c=0 *
*****
Enter the coefficients: 1 3 -10↵
Two roots: 2.0, -5.0

```

5.2.2 Multi-return sub-algorithm (subprograms)

We first consider this type of sub-algorithms in C++. The behaviours of the value and reference parameters are explained following by pass-by-value and pass-by-reference concepts. As already mentioned, the Java language fails to support this type of subprograms. Therefore, we attempt to write equivalent subprograms for the C++ multi-return subprograms in Java.

An argument as a variable has both a value and a unique reference (address). Both pass-by-value and pass-by-reference approaches are used to pass the arguments to the subprogram. However, either the value or the reference can be passed to the subprogram via an argument. In the pass-by-reference approach, the reference of the argument in the memory of the computer is passed to the called subprogram whereas, in the pass-by-value approach the real value of the argument is passed to the called subprogram. In other words, a copy of the value argument is

first stored in the temporary reference of the subprogram when it is going to pass the called subprogram and then, it is passed to the subprogram. Finally, its temporary references are totally disappeared after terminating the subprogram.

The difference between pass-by-reference and pass-by-value approaches is that modifications made to the arguments passed in by reference in the subprogram can affect the call unit while those applied to the arguments passed in by value in the subprogram are unable to affect the call unit. Further, arguments passed by the value can be variables (e.g., x), literals (e.g., 107), and expressions (e.g., $\sin(x) + 2x / y$) while the reference arguments have to be variables.

We use the multi-return sub-algorithms (subprograms) when there is more than one return to the call unit or when sub-algorithms (subprograms) are needed to modify some arguments. The defining instruction for this type of sub-algorithms (subprograms) is similar to the void functions. The only difference is that, unlike the functions where all the parameters were value parameters, the parameters here are a combination of the value and reference parameters.

The rule of the parameter list for multi-return subprograms. The parameters of a multi-return subprogram are of the following form:

data type_‘c’ first parameter, data type_‘c’ second parameter, ..., data type_‘c’ last parameter

where, ‘c’ stands for the characteristic symbol of the parameter which is empty for the value parameters and & for the reference parameters.

As shown in the above rule, the reference parameter is indicated by following the parameter name through an & symbol. Then, the compiler passes the memory address of the actual parameter instead of the value. The & symbol may stick to the parameter. For example, in the defining statement

```
void Pol_Cart(float ro, float theta, float &x, float &y)
```

both float variables *ro* and *theta* are value parameters while the float variables *x* and *y* are reference parameters. Moreover, the notes of function subprograms mentioned in [Subsection 5.2.1](#) are applied here with slight differences.

Notes of the multi-return subprograms in C++: 1. None of the parameters should be defined as a local variable in the body of a subprogram anymore. In other words, a parameter is unable to be a local variable at the same time.

2. We should not read any value parameter in the body of a sub-algorithm (subprogram) anymore; the value of any parameter reaches to the sub-algorithm (subprogram) by the call unit.
 3. We should not print any reference parameter in the body of a sub-algorithm (subprogram) anymore; these parameters are going to be returned to the call unit.
-

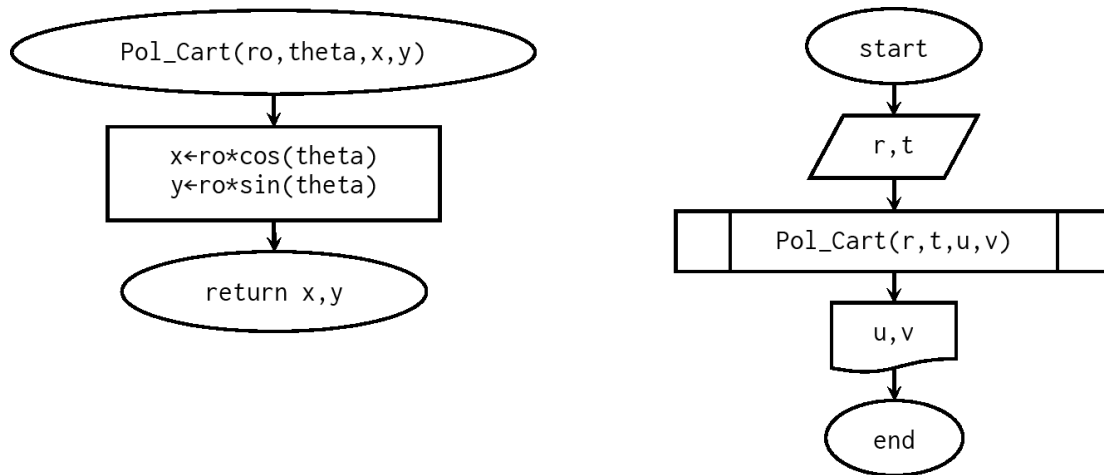


Fig. 5.3: Converting the polar coordinate ($ro, theta$) to the Cartesian coordinate (x, y).

Additionally, the calling instruction for multi-return sub-algorithms (subprograms) is similar to the void functions with the difference that, unlike the functions where all the arguments were value arguments, the arguments here are a combination of the value and reference arguments. In the next example a multi-return sub-program is experienced.

5.3. Example. The Cartesian coordinates (x, y) of a point are calculated from its polar coordinates (ρ, θ) as follows.

$$x = \rho \cos(\theta), \quad y = \rho \sin(\theta). \quad \mathbf{x = \rho \cos(\theta), \quad y = \rho \sin(\theta).}$$

Write a sub-algorithm named Pol_Cart to receive the polar coordinates of a point and then calculate and return its Cartesian coordinates. Next, write a main algorithm to read the polar coordinates of a point and then calculate and print its Cartesian coordinates calling the Pol_Cart sub-algorithm.

Solution. The requested algorithm is drawn in [Flowchart 5.3](#) and its translation is written in Program P5_3 in C++ codes.

C++ codes only:

```
// Program P5_3 to convert polar coordinates
// to Cartesian coordinates using a subprogram.
#include <iostream>
#include <math.h>
using namespace std;
Pol_Cart(double ro, double theta, double &x, double &y) {
    x = ro*cos(theta);
    y = ro*sin(theta);
    return x, y;
}
//*****

int main() {
    double r, t, u, v;
    cout<<"Enter the polar coordinates: ";
    cin>>r>>t;
    Pol_Cart(r, t, u, v);
    cout<<"The Cartesian coordinates are ("<<u<<", "<<v<<");
    return 0;
}
```

Input/output:

```
Enter the polar coordinates: 1 0↵
The Cartesian coordinates are (1, 0)
```

Writing the return statement including its reference parameters at the end of the subprogram is optional in the multi-return subprograms. However, writing an empty return statement is illegal. In other words, the return statement should always be accompanied by the return parameters.

Now, we program for Algorithm 5.3 in Java. Since Java is strictly pass-by-value, therefore, the Cartesian coordinate should be used as the objects of the class Cart. The following synchronous programs are employed to examine the object-orientation feature in both C++ and Java.

C++ codes:

```
// Program P5_3_00P to convert polar
// coordinates to Cartesian coordinates
// using an OOP subprogram.
#include <iostream>
#include <math.h>
using namespace std;
class Cart {
public:
double x, y;
};
//*****
static Cart Pol_Car(double ro,
double theta) {
Cart P;
P.x=ro*cos(theta);
P.y=ro*sin(theta);
return P;
};
//*****
int main() {
double P_x, P_y, r, t;
cout<<"Enter the polar coordinates: ";
cin >>r>> t;
P_x=Pol_Car(r, t).x;
P_y=Pol_Car(r, t).y;
cout<<"The Cartesian coordinates: ("
<<P_x<< ", "<<P_y<<")";
return 0;
}
```

Input/output:

```
Enter the polar coordinates: 1 0↵
The Cartesian coordinates are (1, 0)
```

Java codes:

```
// Program P5_3_00P to convert polar
// coordinates to Cartesian coordinates
// using a subprogram.
import java.util.Scanner;
class Cart {
public double x, y;
}
//*****
class P5_3_00P {
static Cart Pol_Car(double ro,
double theta) {
Cart P=new Cart();
P.x=ro*Math.cos(theta);
P.y=ro*Math.sin(theta);
return P;
}
//*****
public static void main(String[] args) {
double r, t, P_x, P_y;
System.out.print("Enter the polar "
+ "coordinates: ");
Scanner read=new Scanner(System.in);
r=read.nextDouble();
t=read.nextDouble();
P_x=Pol_Car(r, t).x;
P_y=Pol_Car(r, t).y;

System.out.print("Cartesian coordinates:"
+ " (" + P_x + ", " + P_y + ")");
read.close();
}
}
```

Input/output:

```
Enter the polar coordinates: 1 0↵
The Cartesian coordinates are (1.0, 0.0)
```

Based on the rule, any reference parameter should have the & symbol. In addition, the parameter should have the & symbol if it is of both value and parameter types. In the subprogram of the next example, the parameters *m* and *n* are of both value and reference types.

5.4. Example (Swap algorithm). The swap algorithm was discussed in [Chapter 1](#). Write a subprogram, named swap, to receive two integer values of m and n , swap their value, and return them with the same parameters. Then, write a main algorithm to call the subprogram swap() for the two read integers.

Solution. The requested algorithm in C++ codes is found in Program P5_4.

```
// Program P5_4 to interchange two integers using the swap function
#include <iostream>
using namespace std;
void swap(int&, int&);
int main() {
    int m, n;
    cout<<"Enter two integers m and n: ";
    cin>>m>>n;
    swap(m, n);
    cout<<"The interchanged values: m="<<m<<" and n="<<n;
    return 0;
}
//*****
void swap(int& m, int& n) {
    int k;
    k = m; m = n; n = k;
}
```

Input/output:

```
Enter two integers m and n: 1 2↵
The interchanged values: m=2 and n=1
```

The following synchronous Programs P5_4_OOP show the object-oriented feature in C++ and Java.

C++ codes:

```
// Program P5_4_OOP to swap two objects
#include <iostream>
using namespace std;
class Var {
    public:
        int no;
        Var(int no) {
            this->no=no;
        }
};
//*****
void swap(Var &v1, Var &v2) {
    int k;
    k=v1.no;
    v1.no=v2.no;
    v2.no= k;
}
//*****
int main() {
    Var v1=Var(1);
    Var v2=Var(2);
    swap(v1, v2);
    cout<<"v1.no = "<<v1.no<<endl;
    cout<<"v2.no = "<<v2.no<<endl;
    return 0;
}
```

Output:

```
v1.no = 2
v2.no = 1
```

Java codes:

```
// Program P5_4_OOP to swap two objects
class Var {
    int no;
    Var(int no) {
        this.no=no;
    }
}
//*****
class swap {
    public static void swap(Var v1, Var v2) {
        int k;
        k=v1.no;
        v1.no=v2.no;
        v2.no=k;
    }
    //*****
    public static void main(String[] args) {
        Var v1=new Var(1);
        Var v2=new Var(2);
        swap(v1, v2);
        System.out.println("v1.no = " + v1.no);
        System.out.println("v2.no = " + v2.no);
    }
}
```

Output:

```
v1.no = 2
v2.no = 1
```

In each of the Programs P5_4_OOP, there exists a constructor which constructs the initial values of int-type data members (instance variable) *no*. In the left C++ program, the method `swap()` receives the arguments `v1` and `v2` as the objects created in the main method, call the instance variable *no* in the class `Var`, and finally, swap and return them. These tasks are performed in the right Java program within the class `swap`. The run of the program is clear.

5.3 Self-calling (recursive) functions

Self-calling is a technique in which a sub-algorithm (subprogram) calls itself one or more times in its uncompleted implementation (execution) process. In this case, the

following tasks are performed in the implementation process of this calls. Considering these tasks leads to a better understanding of the notion of self-calling and therefore an accurate method of applying this notion in algorithm-writing (programming).

- 1) Imagine that, a temporary layered memory called the “stack memory” is created in each calling of the sub-algorithm and the values of arguments are copied to a layer of this memory such that each layer is distinguished from the previous layer and stacked on that layer. It is noteworthy that all the arguments are of value type.
- 2) Further, the self-calling process is repeated up to the reflexing point. By the reflexing point we mean the first completed sub-algorithm in the last calling in which there is no more self-calling and the sub-algorithm is completely implemented. With the arrangement of the layers of the stack memory in the first above- mentioned task, the top layer is related to the last calling. The top layer of the stack memory is disappeared when the last calling is completely implemented. Then, the implementation control is transferred just after the point from where the final self-calling was conducted. Now, the uncompleted implementation of this sub-algorithm is completed and the corresponding layer of the stack memory is disappeared. This procedure is repeated until the first uncompleted sub-algorithm called from the call unit is completed. Finally, the bottom layer related to the first call is disappeared and the implementation control is transferred just after the point from where the first calling from the call unit was performed.

One of the most important applications of the self-calling technique is the calculation of recursive equations or recursive functions. The most popular recursive example is the factorial function (relation).

5.5. Example. The factorial of the non-negative integer n is defined by the recursive relation

$$n! = n(n - 1)! \quad n! = n(n - 1)!$$

or by the following recursive function:

$$\text{fact}(n) = \begin{cases} 1, & \text{if } n = 0, \\ n \text{ fact}(n - 1), & \text{otherwise.} \end{cases}$$
$$\text{fact}(n) = \begin{cases} 1, & \text{if } n = 0, \\ n \text{ fact}(n - 1), & \text{otherwise.} \end{cases}$$

Write a function to receive a non-negative integer n and then calculate and return the factorial of n using the recursive technique.

Solution. The definition is clear and we only need an if-else template. **Flowchart 5.5 (a)** visualizes this sub-algorithm.

The main algorithm represented in **Flowchart 5.5(b)** reads an integer m and prints a message if the number is negative. Otherwise, it calculates and prints the factorial of the given number by calling the fact() sub-algorithm. Programs P5_5 combined Sub-algorithm 5.5(a) and main Algorithm 5.5(b) in C++ and Java codes.

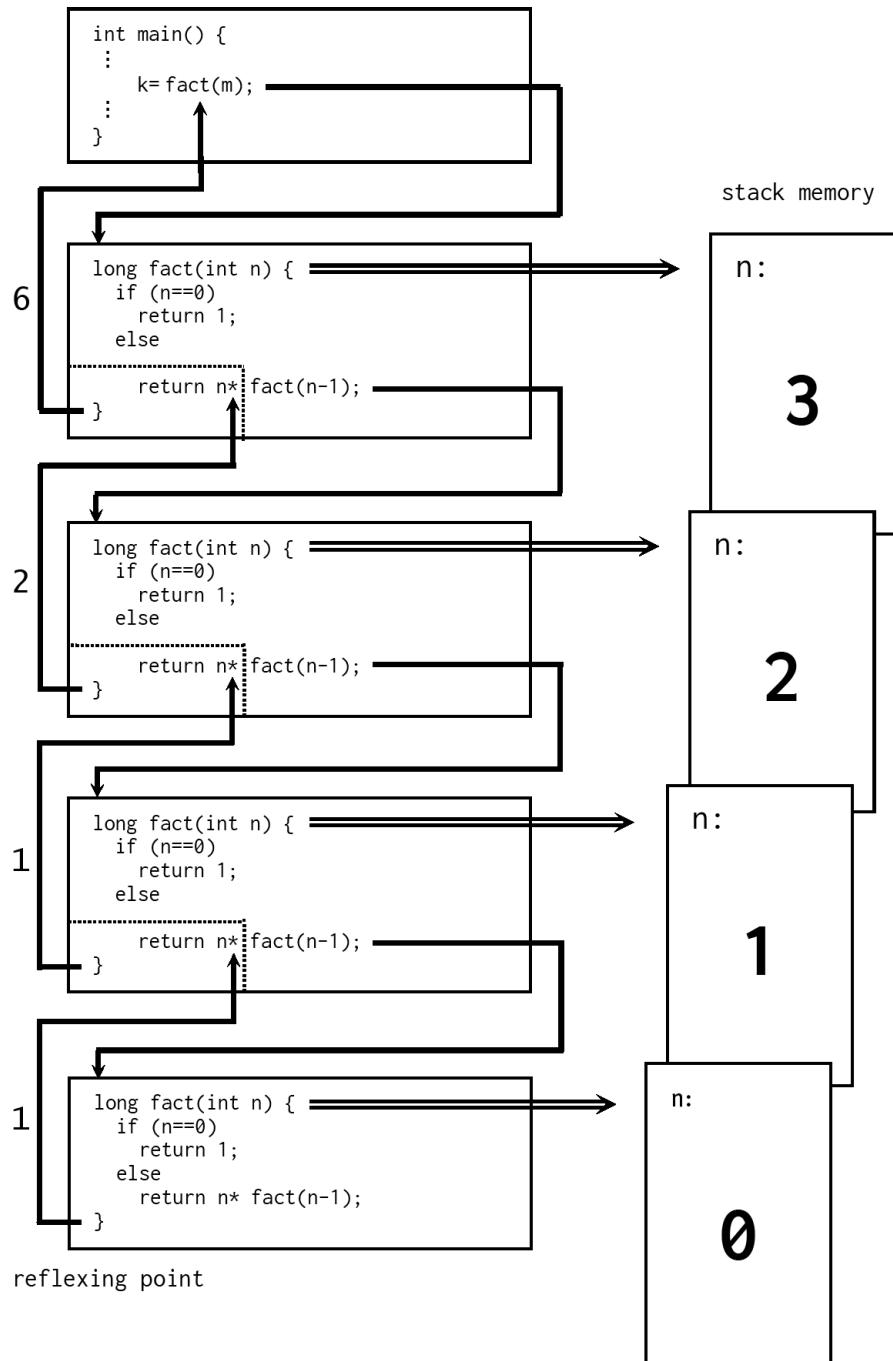


Fig. 5.5(c): Self-calling process in the recursive function `fact()`.

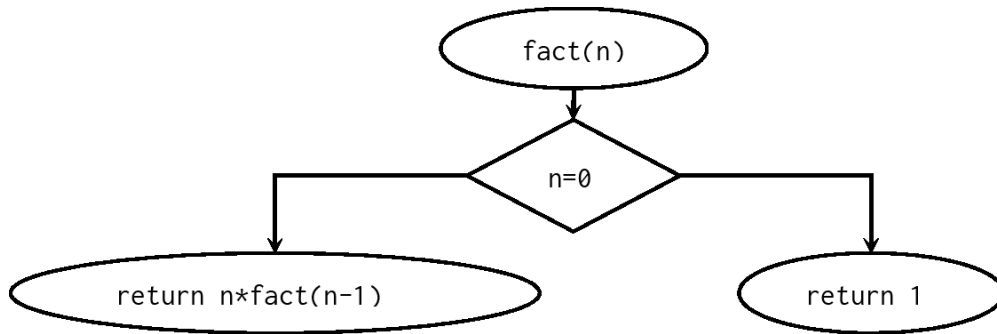


Fig. 5.5(a): Calculating the factorial of n using the recursive technique.

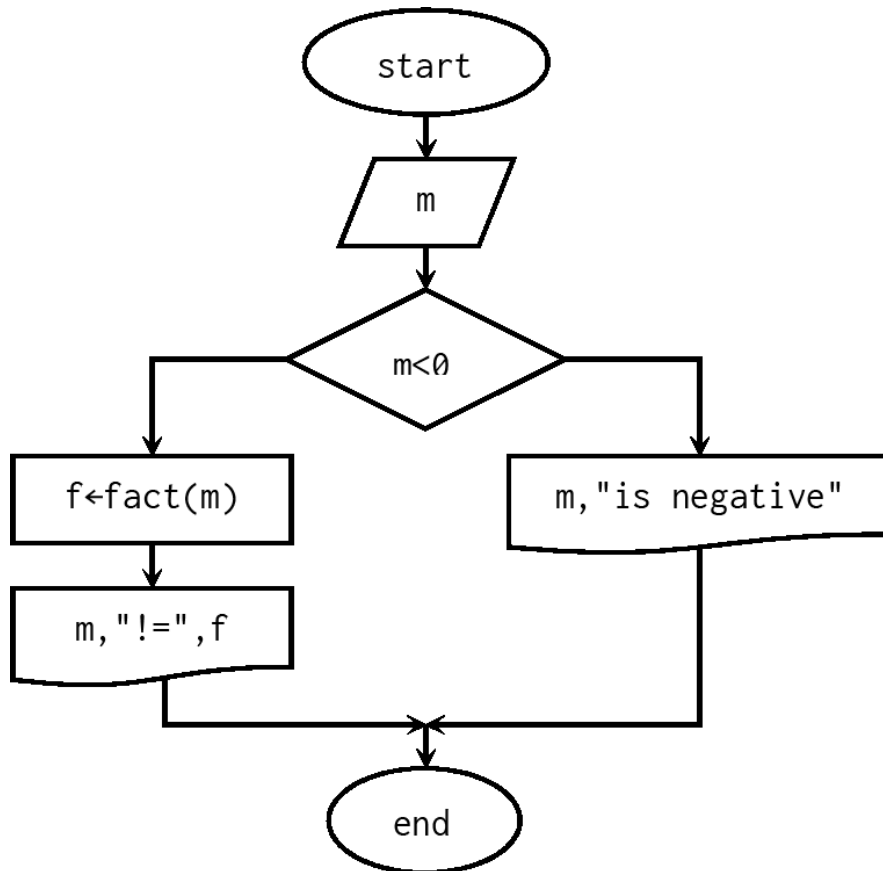


Fig. 5.5(b): Calculating the factorial of a read number calling the function fact().

C++ codes:

```
// Program P5_5 to write down the
// factorial function recursively
#include <iostream.h>
using namespace std;
long fact(int n) {
    if (n==0)
        return 1;
    else
        return n*fact(n-1);
}
//*****
int main() {
    int m;
    long int k;
    cout<<"Enter an integer: ";
    cin>>m;

    if (m<0)
        cout<<m<<" is negative";
    else {
        k=fact(m);
        cout<<m<<"! = "<<k;
    }
    return 0;
}
```

Input/output:

```
Enter an integer: 4↵
10! = 3628800
```

Java codes:

```
// Program P5_5 to write down the
// factorial method recursively
import java.util.Scanner;
class P5_5 {
    static long fact(int n) {
        if (n==0)
            return 1;
        else
            return n*fact(n-1);
    }
//*****
public static void main(String[] args) {
    Scanner read=new Scanner(System.in);
    long k;
    System.out.print("Enter an integer: ");
    int m;

    m=read.nextInt();
    if (m<0)
        System.out.print(m + " is negative");
    else {
        k = fact(m);
        System.out.print(m + "! = " + k);
    }
    read.close();
}
}
```

Input/output:

```
Enter an integer: 4↵
10! = 3628800
```

As explained at the beginning of the section, **Figure 5.5(c)** illustrates the self-calling process in the recursive function of **Flowchart 5.1(a)** from the call unit up to the reflexing point and then back to the call unit bringing the required value of the factorial of n for $n = 3$. The dotted lines indicates that the implementation of the surrounded region is uncompleted. This illustration is for the C++ codes. Furthermore, the same is true for Java codes.

In **Example 5.5**, we had a 1-return function in the implementation of which the function was once called by the call unit and three times by itself. In the next example, a void function calls itself.

5.6. Example. Using the recursive technique, write a function named Back() to receive an integer n which is supposed ≤ 9 and print the integers 9 backward to n .

Solution. In the basic body of the required void function, using an if-else template, we call the function itself for $n + 1$ if $n < 9$. Otherwise, we print n . This sub-algorithm is displayed in Flowchart 5.6(a). The main algorithm in Flowchart 5.6(a) calls the sub-algorithm Back() for the argument of 7. Programs P5_6 demonstrate the translation of Flowchart 5.6(a) into C++ and Java codes.

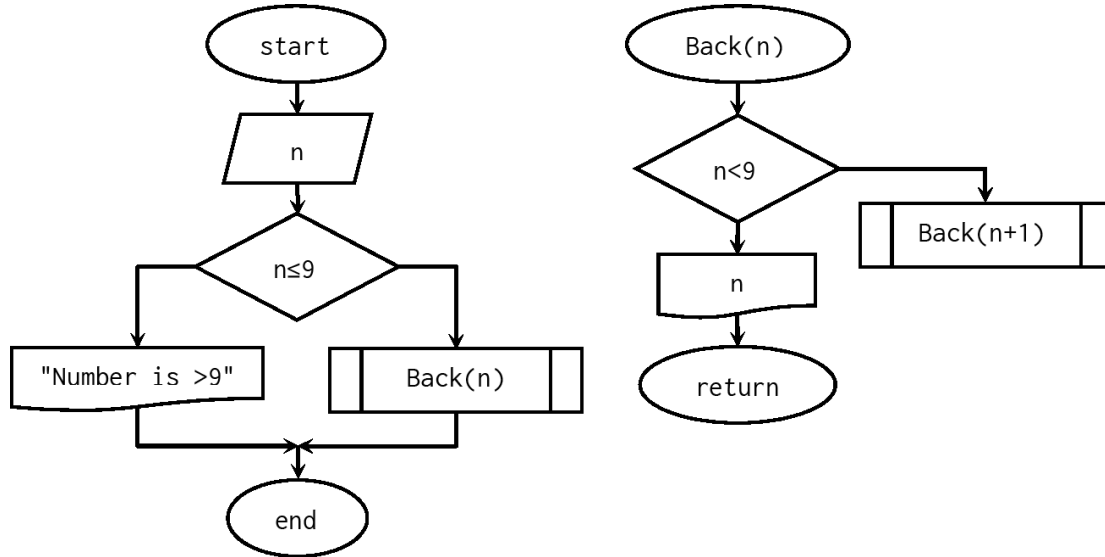


Fig. 5.6(a): Printing the numbers 9 backward to n .

C++ codes:

```
// Program P5_6 a recursive subprogram
// printing 9 backwards to n
#include <iostream>
using namespace std;
void Back(int n) {
    if (n<9)
        Back(n+1);
    cout<<n<<endl;
}
//*****
int main() {
    int n;
    cout<<"Enter an integer:";
    cin>>n;
    Back(n);
    return 0;
}
```

Input/output:

```
Enter an integer:7↵
9
8
7
```

Java codes:

```
// Program P5_6 a recursive subprogram
// printing 9 backwards to n
import java.util.Scanner;
class P5_6 {
    static void Back(int n) {
        if (n<9)
            Back(n+1);
        System.out.println(n);
    }
    //*****
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        System.out.print("Enter an integer:");
        int n=read.nextInt();
        Back(n);
        read.close();
    }
}
```

Input/output:

```
Enter an integer:7↵
9
8
7
```

Based on what was mentioned at the beginning of the section, **Figure 5.6(b)** depicts the self-calling process in the void function of **Flowchart 5.6(a)** from the call unit up to the reflexing point and then back to the call unit printing the integers 9 backward to $n = 7$. The part of the program below the dotted lines is incomplete in the downward direction while it is completed in the upward direction. This illustration is for the C++ codes. Moreover, the same is true for Java codes.

The program starts executing from the `back(n)` call statement in the main program and the direction of the arrows clearly displays the execution direction of the program reaching the reflexing point. In this direction, the main program calls the subprogram once and the subprogram calls itself twice. In each of the three above-mentioned calls, one layer is created in the stack memory. In the reflexing point, the subprogram is completely executed printing the value 9 of the concerned layer in the stack memory. Then, the execution control goes back to the uncompleted part of the penultimate subprogram and the procedure is continued printing 8 and then 7 until the `return 0` statement terminates the main program.

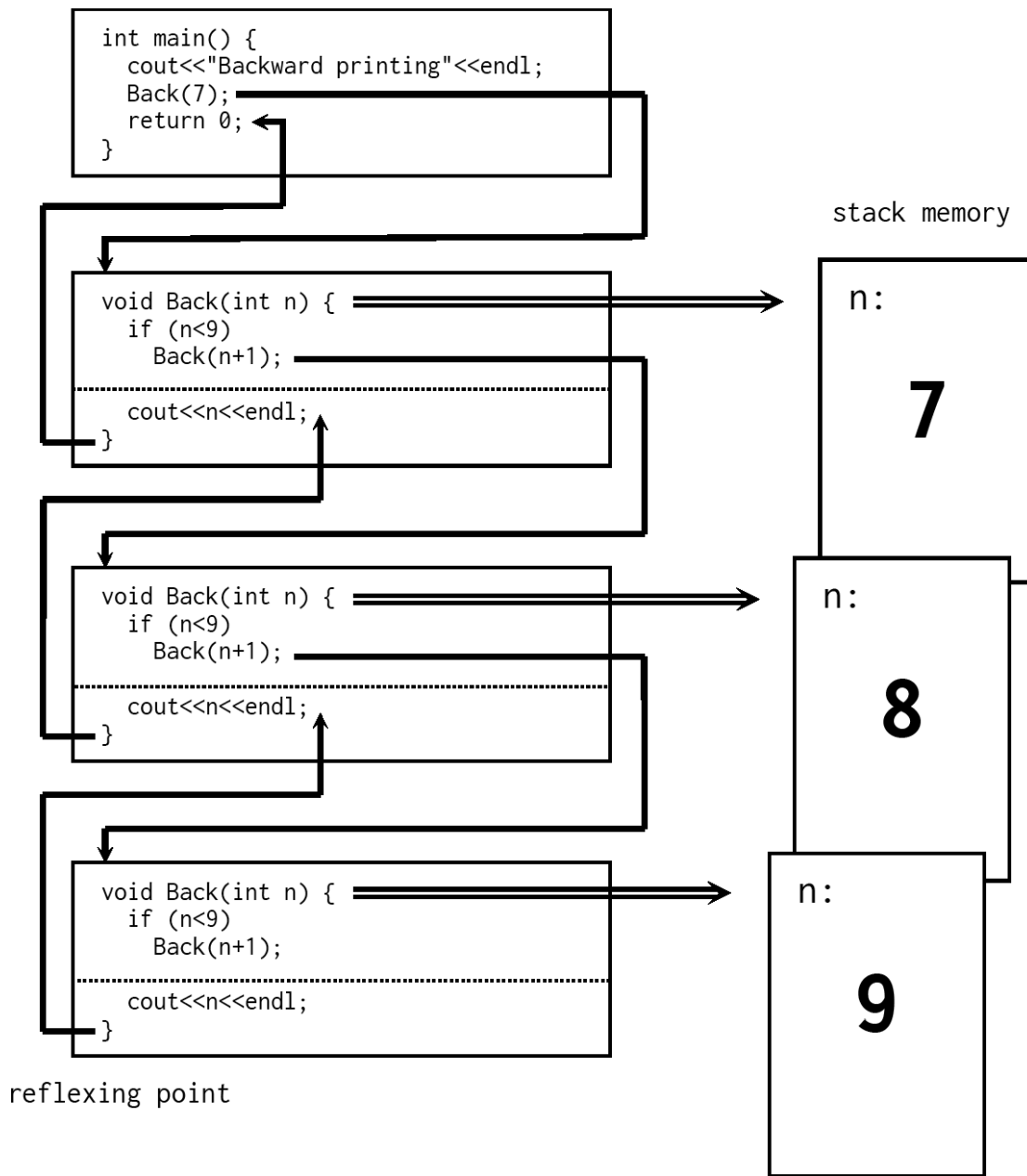


Fig. 5.6(b): Self-calling process in the recursive void function `Back()`.

Question. There should be a reflexing point in every self-calling subprogram. What happens if we write the statement

`Back(n+1);`

in the functions of Programs P5_6 instead of the following if statement?

```
if n<9 then
    Back(n+1);
```

Answer. In this case, there is no reflexing point and the subprogram frequently calls itself. However, considering that the capacity of the stack memory is limited, the memory overflows after a while and the stack overflow error is encountered since one layer is created in the stack memory in each calling.

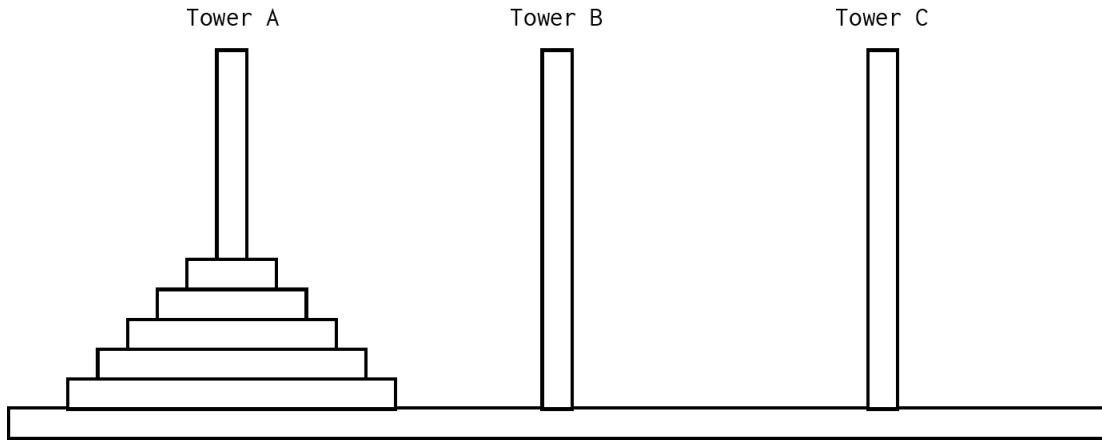


Fig. 5.7: Puzzle of Hanoi Towers.

Additionally, the historical puzzle of the Hanoi towers is another popular example of the self-calling void functions.

5.7. Example (Puzzle of Hanoi Towers). Consider three towers and several disks of distinct sizes on a tower, say A, as shown in [Figure 5.7](#).

In this puzzle, we move the disks from Tower A to Tower C using Tower B as an auxiliary tower subject to the following conditions:

1. Only the top disk can be picked up in every move;
2. The disk should be placed on another Tower if it is picked up;
3. A big disk fails to be placed on a smaller disk.

This puzzle, which is attributed to the monks of the Brahma temple, includes three golden towers and sixty-four golden disks. Considering the above-mentioned conditions, it was believed that upon moving all the sixty-four disks from one tower to another the world would end if every move takes one second. In his book [\[14\]](#), Georges Gamow proves that life on the earth ends much sooner than that time.

Write an algorithm named `move` to receive the number n of disks and the char type variables `pegA`, `pegB`, and `pegC` and then moves the n disks from Tower `pegA`

to Tower *pegC* using Tower *pegB* as an auxiliary tower. Each movement is by printing its direction.

Solution. The required algorithm is described below.

1. If $n = 1$ then, move the single disk from Tower A to Tower C with one move;
2. Else, the movement is performed by recursive techniques as follows:
 - 2.1. Move $n - 1$ disks from Tower A to Tower B employing Tower C;
 - 2.2. Move the only remaining (biggest) disk in Tower A to Tower C with one move;
 - 2.3. Move $n - 1$ disks from Tower B to Tower C utilizing Tower A.

It is recommended to experiment the above-mentioned moves for the cases $n = 2, 3,$ and 4 by yourself. You can observe that the moves of the previous case are repeated twice (once in step 2.1 and once again in step 2.3) in each case.

Considering the clear algorithm described above, the program can be directly written without the need for drawing a flowchart. To do this, the defining statement of the function is first written:

C++ codes: `void move(int n, char pegA, char pegB, char pegC)`

Java codes: `static void move(int n, char pegA, char pegB, char pegC)`

We only need an if-else statement. The above description puts forward the fact that we are dealing with a recursive process hanging on the if-range (case $n = 1$) in which case the move direction is printed as follows:

C++ codes: `cout<<"Move a disk from "<<pegA<<" to "<<pegC<<endl;`

Java codes: `System.out.println("Move a disk from " + pegA + " to " + pegC);`

Now, the else-range is clear if we pay attention to the order of the origin, destination, and auxiliary towers in the defining statements of the function. We are ready now to write the required function. The main units of Programs P5_7 read the number n of the disks and call the function `move` for n and the corresponding character arguments 'A', 'B', and 'C' to character parameters *pegA*, *pegB*, and *pegC*, respectively.

C++ codes:

```
// Program P5_7 The recursive manner for
// movements of TOWERS OF HANOI puzzle.
#include <iostream.h>
using namespace std;
void move(int n, char pegA,
          char pegB, char pegC) {
    if (n==1)
        cout<<"Move a disk from "
            <<pegA<<" to "<<pegC<<endl;
    else {
        move(n-1, pegA, pegC, pegB);
        move(1, pegA, pegB, pegC);
        move(n-1, pegB, pegA, pegC);
    }
    return;
}
//*****
int main() {
    int n;
    cout<<"Enter the number "

        <<"of disks: ";
    cin>>n;
    move(n, 'A', 'B', 'C');
    return 0;
}
```

Input/output:

```
Enter the number of disks: 4↵
Move a disk from A to B
Move a disk from A to C
Move a disk from B to C
Move a disk from A to B
Move a disk from C to A
Move a disk from C to B
Move a disk from A to B
Move a disk from A to C
Move a disk from B to C
Move a disk from B to A
Move a disk from C to A
Move a disk from B to C
Move a disk from A to B
Move a disk from A to C
Move a disk from B to C
```

Java codes:

```
// Program P5_7 The recursive manner for
// movements of TOWERS OF HANOI puzzle.
import java.util.Scanner;
class P5_7 {
    static void move(int n, char pegA,
                    char pegB, char pegC) {
        if (n==1)
            System.out.println("Move a disk from "
                                + pegA + " to " + pegC);
        else {
            move(n-1, pegA, pegC, pegB);
            move(1, pegA, pegB, pegC);
            move(n-1, pegB, pegA, pegC);
        }
    }
    //*****
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int n;
        System.out.print("Enter the number "

            + "of disks: ");
        n=read.nextInt();
        move(n, 'A', 'B', 'C');
        read.close();
    }
}
```

Input/output:

```
Enter the number of disks: 4↵
Move a disk from A to B
Move a disk from A to C
Move a disk from B to C
Move a disk from A to B
Move a disk from C to A
Move a disk from C to B
Move a disk from A to B
Move a disk from A to C
Move a disk from B to C
Move a disk from B to A
Move a disk from C to A
Move a disk from B to C
Move a disk from A to B
Move a disk from A to C
Move a disk from B to C
```

As shown, the first seven moves transfers three disks from A to B using C. In addition, the eighth move transfers the biggest disk from A to C. Finally, the last seven moves transfer the three disks from B to C employing A.

5.7.3. Exercise. Illustrate the process of executing the programs P5_7 as displayed in [Figures 5.5\(c\)](#) and [5.5\(b\)](#).

Exercises

In the following exercises: (1) Arrange the implementation table, if needed, (2) Write the complete program, and (3) Provide appropriate input notifications and output headings, if any. In addition, the user-defined functions in the texts of the current and previous chapters may be used unless otherwise is explicitly specified.

5.1. The number of digits of a positive integer n can be calculated using the following formula.

$$d(n) = \lfloor \log_{10}(n) \rfloor + 1. \quad d(n) = \lfloor \log_{10}(n) \rfloor + 1.$$

where, $\lfloor x \rfloor$ refers to the nearest integer less than or equal to x . Write a function named `digits()` to receive the positive integer n and then calculate and return the number of digits of n using the above formula. Next, write a main algorithm to read the integer m and calling the function `digits()`, print the number of its digits if m is positive; otherwise, print the message Not positive.

5.2. Write a function to receive two real numbers and return the integers 1, -1, or 0 if their multiplication is positive, negative or zero, respectively. Finally, call this function in an appropriate main algorithm.

5.3. Write a function to read the positive integer n and determine whether it is an entire square or not by returning one of the two numbers 1 or 0, respectively. Eventually, call this function in an appropriate main algorithm.

5.4. Write a function to receive the integer n and determine whether it is odd or even by returning the characters 'o' or 'e', respectively. Ultimately, call this function in an appropriate main algorithm.

5.5. Write a function for defining the Kronecker function below (take i and j as real numbers):

$$K(i, j) = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases} \quad K(i, j) = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases}$$

Then, call this function in an appropriate main algorithm.

5.6. Write a function to receive a real number and return its integer and decimal part to the call unit. Then, call this function in an appropriate main algorithm.

5.7. Write a function named `Tri()` to receive three positive real numbers A , B , and C and return 1 if they form the sides of a triangle; otherwise, return 0. Next, call this function in an appropriate main algorithm.

5.8. Write a function named `R_Tri()` to receive three positive real numbers A , B , and C and return 1 if they form the sides of a right triangle; otherwise, return 0. Finally, call this function in an appropriate main algorithm.

5.9. Write a main algorithm to read three positive real numbers A , B , and C . Next, print the message `Not all positive` if none of these numbers are positive (at least one of them is negative or zero); otherwise, calling the function `Tri()`, print the `Not triangle` message if they are not the sides of a triangle; otherwise, calling the `R_Tri` algorithm, print the message `Right triangle` if they are the sides of a right triangle; otherwise, print the message `Trinangle`.

5.10. Write a function to receive the two numbers m and d as months and days, respectively, and then calculate and return the number of a day in a year which corresponds to the d -th day of the m -th month (e.g., by receiving 2 and 21, return: 52).

5.11. Write a function to receive the first day f of the current year and the count n of a day in the year and then determine and return which day of the week that day is. Next, write a main algorithm to read a number (from 0 to 6) as the first day of the year and determine what day of the week the first day of every month is, calling the above-mentioned function.

5.12. Write a function to receive a number f (from 0 to 6), a pair of integers m and d , as the first day of the year, the month, and the day, respectively, and calculate and return a day of the week with is in correspondence with the d -th day of the m -th month. Next, write a main algorithm to read an integer g (from 0 to 6), as the

first day of the year. Then, each time read a pair of integers n , as the month, and t , as the day, and

- Terminate the algorithm if both n and t are equal to 0;
- Print the message error if one of the numbers n or t is out of its legal range;
- Otherwise, calculate a day of the week which is related to the t -th day of the n -th month by calling the above function and then print it as one of the days of the week.

5.13. The Fibonacci sequence is a sequence in which the first two terms are equal to 1 and from the third term each term is the sum of the two previous terms before that term:

$$f_1 = f_2 = 1, \text{ and for } n \geq 3, f_n = f_{n-1} + f_{n-2}.$$

Some of the terms of this sequence are:

$$1, 1, 2, 3, 5, 8, 13, 21, \dots$$

Write a function to receive a positive integer n and then calculate and return the n -th term of the sequence using the self-calling technique. Next, write a main algorithm to read a positive integer n and print the message Not positive if it is not positive and if it is positive, calculate and print the n -th term of the sequence using the above-mentioned function.

5.14. The terms of the sequence f are recursively defined as:

$$f_1 = 1, f_2 = 2, \text{ and for } n \geq 3, f_n = f_{n-1}^2 + f_{n-2}^2.$$

Using the self-calling technique, write a function to receive a positive integer n and then calculate and return the n -th term of the sequence. Now, write a main algorithm to read a positive integer n . Next, print the message Not positive if it is not positive; otherwise, calculate and print the n -th term of the sequence using the above function.

5.15. The polynomials P_n at the point x , with $-1 < x < 1$, are calculated by the following recurrence relations.

$$P_0(x) = 1, P_1(x) = x, \text{ and for } n \geq 3, P_n(x) = \frac{2n-1}{n}P_{n-1}(x) - \frac{n-1}{n}P_{n-2}(x).$$

$$P_0(x) = 1, P_1(x) = x, \text{ and for } n \geq 3, P_n(x) = \frac{2n-1}{n}P_{n-1}(x) - \frac{n-1}{n}P_{n-2}(x).$$

Arrange a function to receive n and x and then calculate and return $p_n(x)$.
 $P_n(x)$. Next, write a main algorithm to read the values of n and x and, calling the above function, calculate and print the value of $P_n(x)$ in the format below if $n \geq 0$ and $-1 < x < 1$:

the value of n -th polynomial at x is: the return value

Otherwise, print the message below:

Error: illegal value x or n

5.16. Considering that $x^0 = 1$, $x^0 = 1$, and for a positive integer n , $x^n = xx^{n-1}$, $n, x^n = xx^{n-1}$, and $x^{-1} = x^{-1} = 1/x$, $1/x$, define a recursive function f to receive the real number x and the integer n and calculate and return the value of $f_n(x) = x^n$. $f_n(x) = x^n$. Assume that the receiving values of x and n are such that x^n is not undefined. Next, write a main algorithm to read the value of x and n and print the following message in the undefined cases $x = 0$ and $n \leq 0$:

the value of x ^ the value of n is not defined

Otherwise, print the value of x^n with the following format calling the function above:

the value of x ^ the value of n = the recursive value

5.17. The number of combinations of r distinct objects from n objects, with $0 \leq r \leq n$, denoted by $C(n, r)$, is calculated by the following recursive function:

$$C(n, r) = \begin{cases} 1, & \text{if } r = n \text{ or } r = 0, \\ n, & \text{if } r = 1 \\ C(n-1, r) + C(n-1, r-1), & \text{otherwise.} \end{cases}$$

$$C(n, r) = \begin{cases} 1, & \text{if } r = n \text{ or } r = 0, \\ n, & \text{if } r = 1 \\ C(n-1, r) + C(n-1, r-1), & \text{otherwise.} \end{cases}$$

Write a function to receive the integer values n and r and then calculate and return the value $C(n, r)$ using the self-calling technique. It is assumed that the receiving values n and r satisfy the inequality $0 \leq r \leq n$. Next, write a main algorithm which reads the two values n and r . Afterwards, prints the following format if the condition $0 \leq r \leq n$ is not established:

C(the value of n , the value of r) is not defined

Otherwise, print $C(n, r)$ in the following format calling the above function:

C(the value of n , the value of r) = recursive value

6 Automated loops

Any process that repeatedly implements a set of instructions, called the range of a loop, is called a **loop**. Designing this range is the most fundamental part of the job in creating a loop. In other words, knowing what to do in each repetition of the loop and how to use such repetition properly to design the algorithm is of great importance.

A kind of loop is based on one or several variables so that, first, the initial values are often assigned to them before starting the loop. Then, a process is performed on the variables in each repetition of the loop. Finally, repeating or exiting the loop is determined by a condition called the repetition (or exiting) condition of the loop. This kind of loop is known as the **conditional loop** which is discussed in the next chapter.

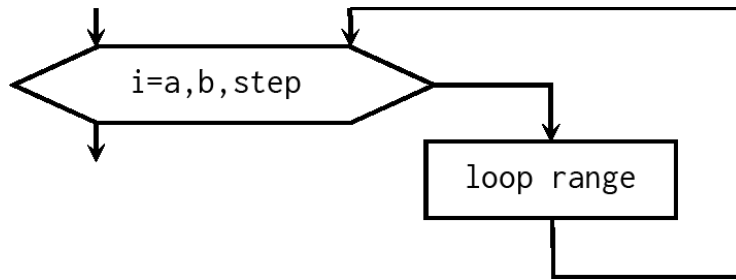
Another kind of loop is called **the automated loop** which depends on a counter. Three tasks are automatically performed by the compiler in this type of loop:

1. The initial value for the counter is assigned;
2. A constant (positive or negative) amount, called the growth of the loop, is added to the previous amount of the counter in each repetition of the loop;
3. The number r of the repetitions is calculated by the compiler and the range of the loop repeats to the number of r times.

That is why this kind of loop is named the automated loop.

6.1 The for template

The automated loop template, called hereafter the “**for template**”, or the “**for loop**” is used in the flowcharts with the shape displayed below.



The translation of this template into both C++ and Java codes, which is called the **for statement**, is as follows.

```
for (i=a; i<=b; i=i+step) {
    loop range
}
```



The loop range, in the case of more than one statement, should be grouped by {}.

The items “*i*”, “*a*”, “*b*”, and “*step*” are called the “variable, initial value, final value”, and “growth value” of the loop, respectively. The phrase “*i=a,b,step*” is called the **specification** of the loop and is written in the front of the codes without any spacing after commas. Moreover, only after the semicolon will be used the space; for instance, we write: `for (i=k+1; i<=n; i+=2)`.

For running the mentioned for statement, the compiler first calculates the number *r* of repetitions using the following formula:

$$r = \max \left\{ \left[\frac{b - a + \text{step}}{\text{step}} \right] \right\},$$

where, $[k]$ stands for the least integer close to k . Then, the run process is performed in the following way: repeat the loop range to the number of r times, starting from $i = a$, in such a way to add the amount of the step to the previous amount of i in each repetition.

For example, consider the specification $i=1,3,9$. The number of repetitions is $r = 3$ and hence the loop range is run 3 times. The first time, 1 is assigned to i and the range is run. The second time, 3 units is added to the previous amount of i , and the range is run for $i = 4$. Finally, the amount of i is increased to 7 and the range is run the third time for $i = 7$. Several notes should be considered in this respect.



- The growth increment $i=i+step$ can be written as $i+=step$ using the compound addition assignment $+=$. Especially, one-unit increasing is written as $i++$ and one-unit decreasing as $i--$.
 - The initial, final, and growth values should not vary inside the loop range.
 - The variable of the loop should not be manipulated.
 - The initial value is always used in the first repetition for assigning to the variable. However, the final value may be useless. This value, indeed, plays the role of a border so that the exit from the loop happens upon passing the value of the variable from this amount.
 - From now on, if the growth is not written in the flowcharts, then it will be assumed as 1 for simplicity.
-

There are two circumstances in which **incompatibility** occurs in the variables of a loop:

- 1) $a < b$ and $step < 0$, for example $i=1,3,-1$;
- 2) $a > b$ and $step > 0$, for example $i=4,1,2$;

In these cases, the number of repetitions is equal to zero and, therefore, the range of the loop never runs. It is as though there is no loop whatsoever.

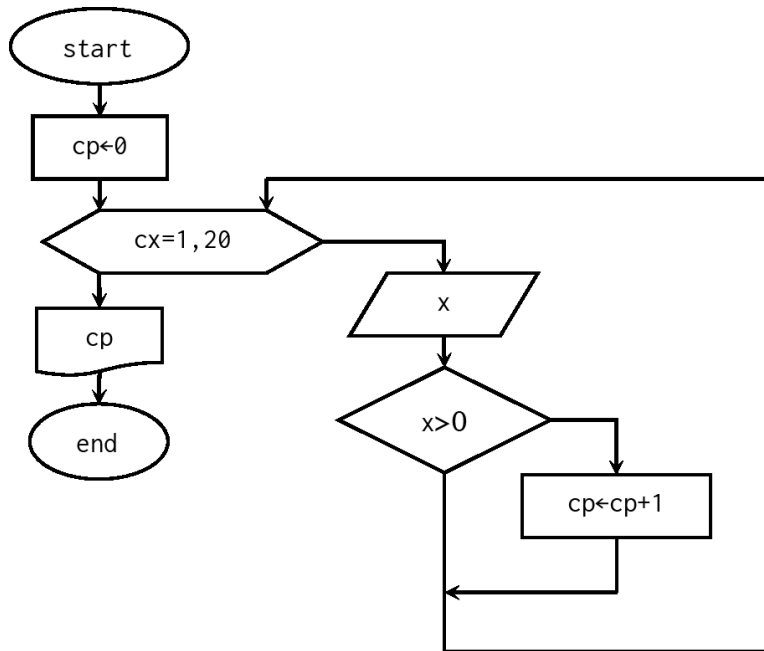


Fig. 6.1: Determining the number of positive integers among 20 integers.

Two issues are highlighted in constructing of the for loops, properly recognizing the specification of the loop, and determining the range of the loop.

6.1. Example. Write an algorithm to read 20 integers, and then determine and print the number of positive integers among these integers.

Solution. Denote the number of positive and read integers by cp and cx , respectively. Here, we need a for loop with 20 repetitions since 20 numbers should be read. The specification of this loop is: $cx=1,20$. What should be processed in each repetition of the loop? Well, first a number x is read. Then, one unit is added to the cp if this number is positive. Assigning an initial value to cp is missing. This should be conducted before starting the loop. Why? Arrange the implementation table for four or five various data in order to get the answer to this question and assign the initial value to cp in duplicate before starting the loop and inside the loop and then compare the results. The resulted flowchart is illustrated in **Figure 6.1**. Programs P6_1 are the translation of this flowchart into both C++ and Java codes.

C++ codes:

```
// Program P6_1 to determine the number
// of positive integers using a for loop
#include <iostream>
using namespace std;
int main() {
    int x, cp=0, cx;
    for (cx=1; cx<=20; cx++) {
        cout<<"Enter an integer: ";
        cin>>x;

        if (x>0)
            cp++;
    }
    cout<<"Number of positive integers: "<<cp;
    return 0;
}
```

Java codes:

```
// Program P6_1 to determine the number
// of positive integers using a for loop
import java.util.Scanner;
class P6_1 {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int x, cp=0, cx;
        for (cx=1; cx<=20; cx++) {
            System.out.print("Enter an "
                + "integer: ");
            x=read.nextInt();
            if (x>0)
                cp++;
        }
        System.out.print("Number of "
            + "positive integers: " + cp);
        read.close();
    }
}
```

In **chapter 5**, the factorial of a non-negative integer was calculated using a recursive subprogram. In the next example, we calculate directly this factorial.

6.2. Example. Write an algorithm to read a non-negative integer n and then calculate and print $n!$ using the following formula:

$$n! = 1 \times 2 \times \cdots \times n. \quad n! = 1 \times 2 \times \cdots \times n.$$

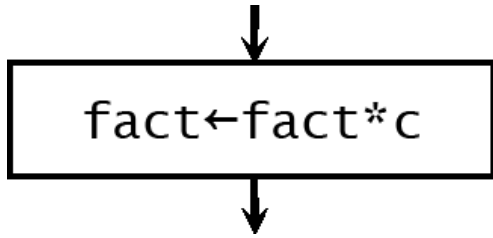
Solution. Here, we are dealing with a normal count from 1 to n . Therefore, an automated loop with the specification $c=1,n$ is constructed.

When n equals to 1 we expect the output 1;

When n equals to 2 we expect the output 1×2 ;

When n equals to 3 we expect the output $1 \times 2 \times 3$, and the like.

As seen, we have a repetitive multiplication which is taken as the *fact*. Furthermore, the new number in each repetition of the loop is multiplied to the previous amount of the *fact* and substituted for it. Therefore, the instruction which is implemented in each repetition of the loop is:



In fact, this is the range of our loop. Finally, assigning the initial value 1 to the *fact* is the only process left to finish the loop. Why 1? The implementation table reveals the answer. The result of our discussion is depicted in [Figure 6.2\(a\)](#).

The implementation table of this algorithm for $n = 4$ is presented in [Table 6.2\(a\)](#).



- If a variable is read only once, then it is not necessary to place it in the implementation table. Thus, we did not write n in [Table 6.2\(a\)](#).
 - It would be better to write the details of the calculation in the implementation table in order to compare the requested calculation in the problem with the results of the implementation table. In [Table 6.2\(a\)](#), the details of the calculation are written instead of writing only the resulted numbers in the fact column. In this case, the output column is not necessary.
-

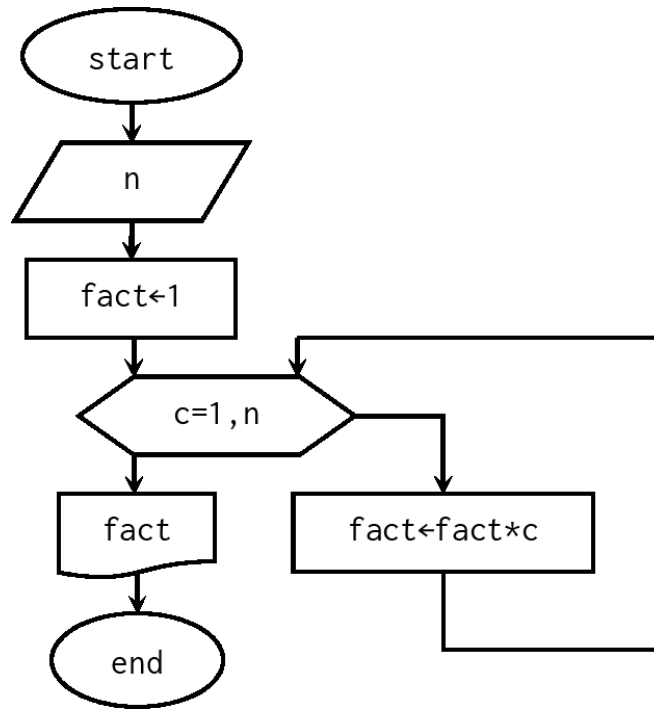


Fig. 6.2(a): Factorial main algorithm.

Tab. 6.2(a): Implementation table of Flowchart 6.2(a) for $n = 4$.

position	c	fact	output
before the loop		1	
first repetition	1	$1! = 1 \times 1$	
second repetition	2	$2! = 1! \times 2$	
third repetition	3	$3! = 2! \times 3$	
fourth repetition	4	$4! = 3! \times 4$	
after the loop			24

Does Table 6.2(a) work either for $n = 0$? The answer is yes. This is due the existence of an incompatibility. As a result, the loop never runs. However, the

initial number of *fact* is 1 which is accidentally 0! The translation of **Flowchart 6.2(a)** into both C++ and Java codes can be seen in Programs P6_2_A.

The long type has a higher capacity in the calculations compared to the int type. Accordingly, one can use this type in order to use the maximum capacity.

The for statement is used in various types. One of these types which is not mostly used is employed for the infinite repetitions:

for (;;)
loop range

C++ codes:

```
// Program P6_2_A to compute the
// factorial of a non-negative number
#include <iostream>
using namespace std;
int main() {
    long fact;
    int c, n;
    cout<<"Enter an integer: ";
    cin>>n;
    fact=1;
    for (c=1; c<=n; c++)
        fact*=c;
    cout<<n<<"!="<<fact;
    return 0;
}
```

Input/output:

```
Enter an integer: 4↵
10! = 3628800
```

Java codes:

```
// Program P6_2_A to compute the
// factorial of a non-negative number
import java.util.Scanner;
class P6_2_A {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        long fact;
        int c, n;
        System.out.print("Enter an integer: ");
        n=read.nextInt();
        fact=1;
        for (c=1; c<=n; c++)
            fact=fact*c;
        System.out.print(n + "!=" + fact);
        read.close();
    }
}
```

Input/output:

```
Enter an integer: 4↵
10! = 3628800
```

In this case, a combination of control buttons is used in different systems to exit the infinite loop. For example, pushing the two buttons “ctrl” and “break” terminates the loop in most systems.

The for statement used so far is in the following form:

assignment statements related to the loop

```
for (i=a; i<=b; step)
```

loop range

Here, the “*assignment statements related to the loop*” may be displaced before $i=a$. Moreover, the loop range may be moved after $i=a$, in which case the for statement is ended and thus a semicolon must be added after the closed parenthesis. Note that, except for the two semicolon separators in the specification of the loop, the other statements must be separated by the commas. Given the above explanations, the following four group of statements are equivalent in both C++ and Java languages.

```
fact=1;  
for (c=1; c<=n; c++)  
fact * =c;
```

```
for (fact=1, c=1; c<=n; c++)  
fact * =c;
```

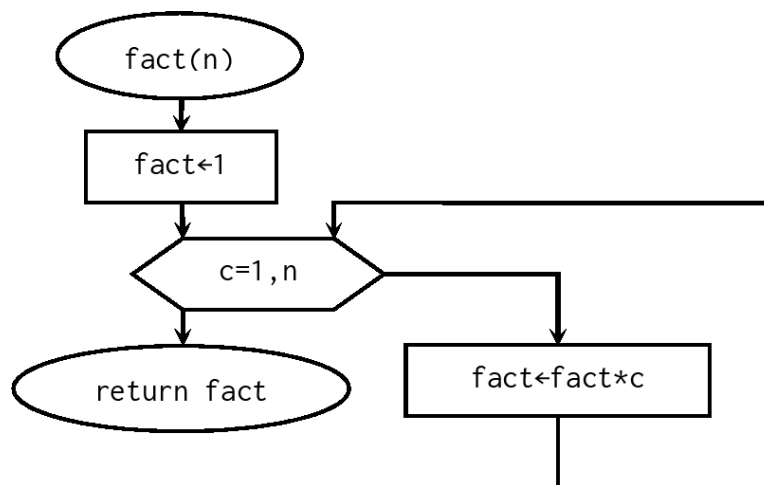


Fig. 6.2(b): Factorial function; the carrier and function have the same name.

```
fact=1;  
for (c=1; c<=n; fact*=c, c++);  
  
for (fact=1, c=1; c<=n; fact*=c, c++);
```

We modify **Flowchart 6.2(a)** to a sub-algorithm named `fact()` (**Fig. 6.2(b)**).

We respected two essential points in the process of modifying this subalgorithm from Algorithm 6.2(a).

- Considering the fact that there exists exactly one return, we choose a function named *fact* and determine its only parameters as *n*. This function receives a nonnegative number *n* and then calculates and returns its factorial.
- The instructions of reading *n* and printing *fact* are removed. Instead, *n* is received from the call unit to the subprogram and the *fact* returns to that unit.

The name of the carrier and function are the same in [Flowchart 6.2\(b\)](#). However, they may be different, as in [Figure 6.2\(c\)](#).

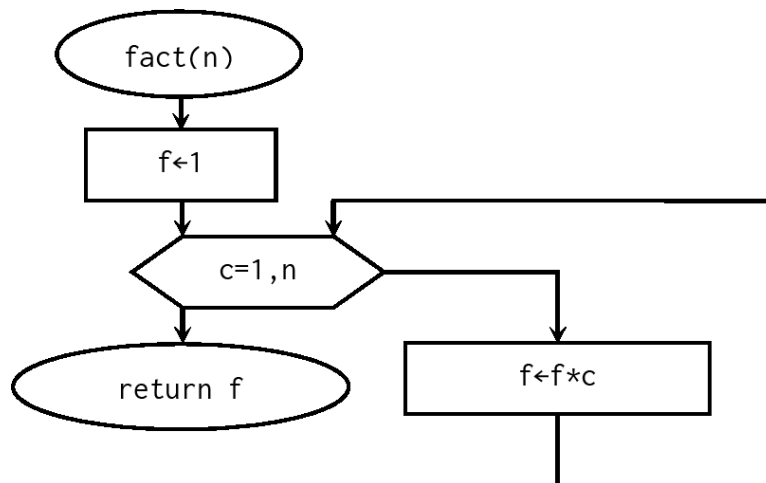


Fig. 6.2(c): Factorial function; the carrier and function have different names.

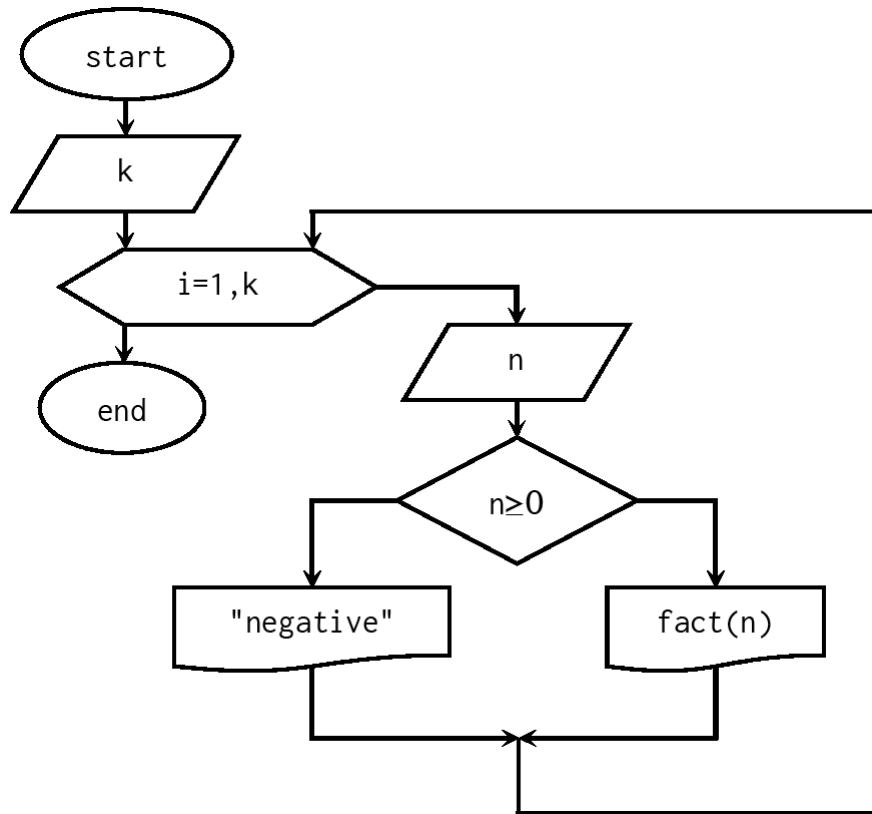


Fig. 6.2(d): A main algorithm calling fact.

The function sub-algorithm fact() is applied several times in this book. The corresponding subprogram (method) can be saved in the memory of the computer for further uses. The translation of **Flowchart 6.2(c)** into C++ and Java codes is represented in the following methods:

C++ codes:

```

long fact(int n) {
  int c;
  long f;
  for (f=1, c=1; c<=n; f*=c, c++);
  return f;
}
  
```

Java codes:

```

static long fact(int n) {
  int c;
  long f;
  for (f=1, c=1; c<=n; f*=c, c++) ;
  return f;
}
  
```

The main algorithm 6.2(d) reads first the positive integer *k*. Then, it reads *k* integers one by one. After each reading, if the read number is non-negative, it calculates and prints the factorial of the number using the sub-algorithm fact(); otherwise, it prints a message. Programs P6_2_B hold the translation of

this main algorithm, as well as the sub-algorithm fact() into both C++ and Java codes.

C++ codes:

```
// Program P6_2_B to investigate the
// factorial of k read integers
// using the function fact()
#include <iostream>
using namespace std;
long fact(int);

int main() {
    int i, k, n;
    cout<<"Enter the number of "
         <<"read integers: ";
    cin>>k;
    for (i=1; i<=k; i++) {
        cout<<"Enter an integer: ";
        cin>>n;
        if (n>=0)
            cout<<n<<"!="<<fact(n)<<"\n";
        else
            cout<<n<<" is negative \n";
    }
    return 0;
}
//*****
long fact(int n) {
    int c;
    long f;
    for (f=1, c=1; c<=n; f*=c, c++);
    return f;
}
```

Input/output:

```
Enter the number of read integers: 3↵
Enter an integer: -2↵
-2 is negative
Enter an integer: 0↵
0!=1
Enter an integer: 10↵
10!=3628800
```

Java codes:

```
// Program P6_2_B to investigate the
// factorial of k read integers
// using the function fact()
import java.util.Scanner;
class P6_2_B {
    public static void main(String[] args) {

        Scanner read=new Scanner(System.in);
        int i, k, n;
        System.out.print("Enter the number of "
                        + "read integers: ");
        k=read.nextInt();
        for (i=1; i<=k; i++) {
            System.out.print("Enter an integer: ");
            n=read.nextInt();
            if (n>=0)
                System.out.println(n + "!=" + fact(n));
            else
                System.out.println(n + " is negative");
        }
        read.close();
    }
}
//*****
static long fact(int n) {
    int c;
    long f;
    for (f=1, c=1; c<=n; f*=c, c++) ;
    return f;
}
}
```

Input/output:

```
Enter the number of read integers: 3↵
Enter an integer: -2↵
-2 is negative
Enter an integer: 0↵
0!=1
Enter an integer: 10↵
10!=3628800
```

6.3. Example. Write an algorithm to read the positive integer n and calculate and then print the following sum.

$$1! + 2! + 3! + \dots + n!$$

Solution. In this algorithm, as well as both the oncoming Algorithms 6.5 and 6.6, where a factorial calculation is involved, a simple way is to write an algorithm in which the fact() sub-algorithm is used. However, we will consider these three algorithms using direct techniques.

Concentrating on **Table 6.2(a)**, we will notice that the factorial of the new number is calculated in each repetition of the loop. Therefore, the requested algorithm will be achieved if we consider this factorial as the general term of a repetitive *sum* and place the calculation of this repetitive *sum* after the calculation of this factorial. Accordingly, we only need to put the initial value of *sum*, which we consider as 0, before the loop. The above-mentioned discussion is summarized in **Figure 6.3**.

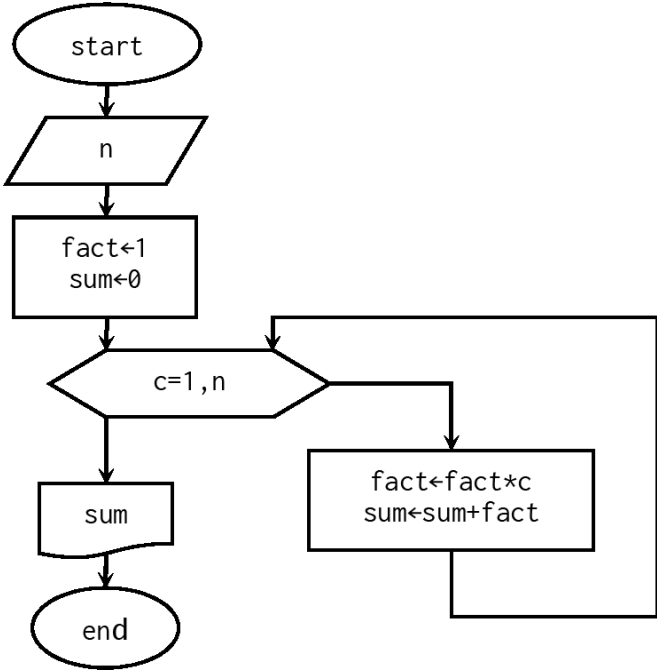


Fig. 6.3: Calculating the sum $1! + 2! + \dots + n!$.

Implementing the above algorithm for $n = 4$ is demonstrated in **Table 6.3**. The output column is removed since the calculation for sum is written in details.

Tab. 6.3: Implementation table of Algorithm 6.3 for $n = 4$.

position	c	fact	sum
before the loop		1	0
first repetition	1	$1! = 1 \times 1$	1!
second repetition	2	$2! = 1! \times 2$	$1! + 2!$
third repetition	3	$3! = 2! \times 3$	$1! + 2! + 3!$
fourth repetition	4	$4! = 3! \times 4$	$1! + 2! + 3! + 4!$

Programs P6_3 represents this algorithm in both C++ and Java codes.

C++ codes:

```
// Program P6_3 to compute the sum
// 1! + 2! + ... + n!
#include <iostream>
using namespace std;
int main() {
    long fact, sum;
    int c, n;
    cout<<"Enter an integer: ";
    cin>>n;

    for (fact=1, sum=0, c=1; c<=n;
        fact*=c, sum+=fact, c++);
    cout<<"sum="<<sum;
    return 0;
}
```

Input/output:

Enter an integer: 4↵
sum=33

Java codes:

```
// Program P6_3 to compute the sum
// 1! + 2! + ... + n!
import java.util.Scanner;
class P6_3 {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        long fact, sum;
        int c, n;
        System.out.print("Enter an integer: ");

        n=read.nextInt();
        for (fact=1, sum=0, c=1; c<=n;
            fact*=c, sum+=fact, c++);
        System.out.print("sum="+sum);
        read.close();
    }
}
```

Input/output:

Enter an integer: 4↵
sum=33

The importance of the order in writing the instructions inside the rectangle is emphasized. Therefore, to realize this importance, swap the position of both

instructions inside the rectangle in the range of the loop and rearrange the implementation table anew!

6.4. Example. Write an algorithm to read the positive integer n and then calculate and print the following sum.

$$1 + (1 + 2) + (1 + 2 + 3) + \dots + (1 + 2 + \dots + n).$$
$$1 + (1 + 2) + (1 + 2 + 3) + \dots + (1 + 2 + \dots + n).$$

Solution. Comparing this formula with that provided in [Example 6.3](#), we will notice that the calculation operator of the general term is multiplication in [Example 6.3](#) whereas the operator is an addition here. As a result, at first glance, it seems obvious that we only need to change the multiplications operator in [Example 6.3](#) to addition operator. How much do you agree with this obvious assumption? Arrange the implementation table! In the first repetition, it is found that the algorithm does not work. By looking at the problem occurred in the table, it is found that we only need to take 0 instead of 1 as the initial value of *fact* in order to fix this problem



The implementation table should be rearranged anew even for the most obvious changes in the flowchart.

6.4.1. Exercise. Draw the complete flowchart and arrange the implementation table for $n = 3$. In addition, translate the resulted flowchart into both C++ and Java codes.

6.5. Example. Write an algorithm to read the positive integer n and calculate and print the sum below.

$$1 - \frac{1}{2!} + \frac{1}{3!} - \dots + (-1)^{n+1} \frac{1}{n!}.$$
$$1 - \frac{1}{2!} + \frac{1}{3!} - \dots + (-1)^{n+1} \frac{1}{n!}.$$

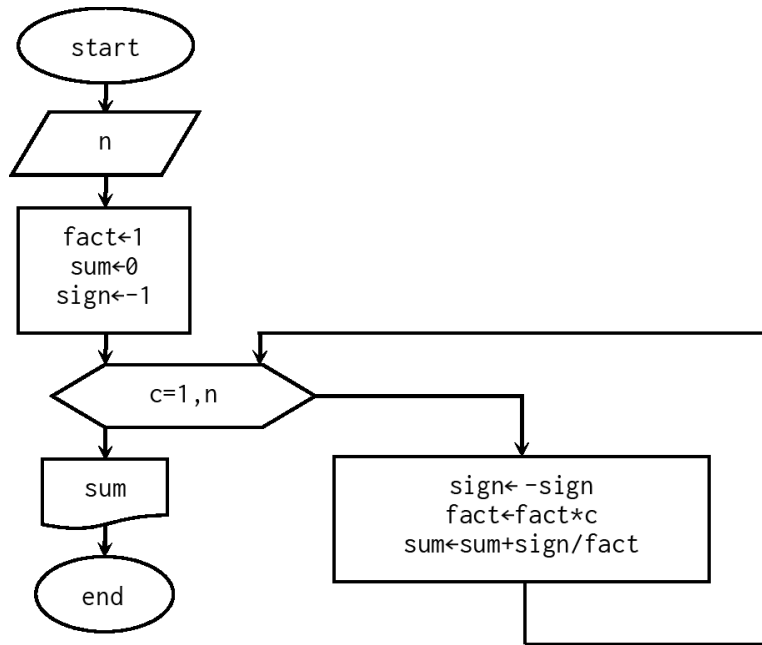


Fig. 6.5: Calculating the sum $1 - 1/2! + 1/3! - \dots (-1)^{n+1}/n!$.

Solution. Ignoring the signs for a moment, the difference between the above sum and that of **Example 6.3** is that the general term is added to the previous repetitive sum in each stage in **Example 6.3** and is reversed here. How do we apply the signs? There are two general techniques to do this. In the first technique, we consider a new variable named *sign* and take -1 as its initial value before the loop. Then, before calculating the *sum* in each repetition of the loop, we change the sign of the *sum* with the instruction $sign \leftarrow -sign$ and then multiply it to the general term. Accordingly, the sign of the general term turns positive and negative alternatively. This algorithm is illustrated in **Figure 6.5**.

6.5.1. Exercise. Arrange the implementation table for $n = 3$. Further, translate **Flowchart 6.5** into both C++ and Java codes.

In the second technique, we change the factorial instruction in the form below:

$fact \leftarrow -fact * c$

Furthermore, the initial value of the *fact* should be changed to -1 ; otherwise, the first term obtains the negative sign.

6.5.2. Exercise. Draw the complete flowchart in this case and arrange the implementation table for $n = 3$. Moreover, translate the obtained flowchart into both C++ and Java codes.

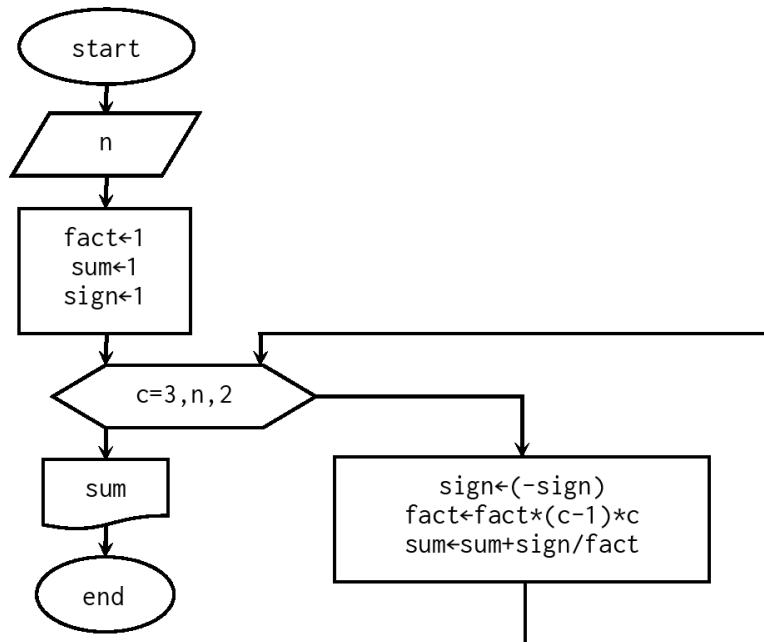


Fig. 6.6: Calculating the sum $1 - 1 / 3! + 1 / 5! - \dots (-1)^{(n+1)/2} / n!$.

6.6. Example. Write an algorithm to read a positive odd integer n . Then calculate and print the sum

$$1 - \frac{1}{3!} + \frac{1}{5!} - \dots + (-1)^{\frac{n+1}{2}} \frac{1}{n!}.$$

$$1 - \frac{1}{3!} + \frac{1}{5!} - \dots + (-1)^{\frac{n+1}{2}} \frac{1}{n!}.$$

Solution. There is more difference this time. The counter increases two units and this causes some complexity. Consider **Flowchart 6.5** and change the growth amount to 2. Now, assume that the third repetition with $c = 3$ is implemented and the result $3!$ is obtained for the *fact*. Increasing the growth amount by 2, the new amount of c gets 5. Certainly, the result of $fact*c$, that is, $3! \times 5$ will not be $5!$. Something is missing. What should be put instead of the dots in $3! \times \dots \times 5$ to arrive at $5!$? Obviously it is 4 which is $c - 1$. Therefore, the factorial instruction should be changed to

fact←fact*(c-1)*c

Now, implementing the recent instruction for $c = 1$ leads to 0, which is problematic. As a result, we should take 3 as the initial value of the loop and assign the initial value 1 (the first term) to the *sum*, instead of 0 in order to fix this problem. Now, the repetitive sum starts with the second term and thus the initial value of the *sign* should be changed to 1. More importantly, all these changes are inspired by the implementation table. **Flowchart 6.6** depicts the result of the above argument and **Table 6.6** is the implementation table for $n = 5$.

Tab. 6.6: Implementation table of Algorithm 6.6 for $n = 5$.

position	c	fact	sign	sum
before the loop		1	1	1
first repetition	3	$3! = 1 \times 2 \times 3$	-1	$1 - 1/3!$
second repetition	5	$5! = 3! \times 4 \times 5$	1	$1 - 1/3! + 1/5!$

Flowchart 6.6 is displayed in C++ and Java codes in Programs P6_6.

C++ codes:

```
// Program P6_6 to compute the sum
// 1-1/3!+1/5!-...+(-1)^((n+1)/2)*1/n!
#include <iostream>
using namespace std;
int main() {
    int c, sign, n;
    long fact;
    float sum;
    cout<<"Enter a positive odd integer: ";
    cin>>n;
    for (fact=1, sum=1, sign=1, c=3;
        c<=n; sign=-sign, fact*=(c-1)*c,
        sum+=(float)sign/fact, c+=2);
    cout<<"sum="<<sum;
    return 0;
}
```

Input/output:

```
Enter a positive odd integer: 5↵
sum=0.841667
```

Java codes:

```
// Program P6_6 to compute the sum
// 1-1/3!+1/5!-...+(-1)^((n+1)/2)*1/n!
import java.util.Scanner;
class P6_6 {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int c, sign, n;
        long fact;
        float sum;
        System.out.print(
            "Enter a positive odd integer: ");
        n=read.nextInt();
        for (fact=1, sum=1, sign=1, c=3; c<=n;
            sign=-sign, fact*=(c-1)*c,
            sum+=(float) sign/fact, c+=2);
        System.out.print("sum=" + sum);
        read.close();
    }
}
```

Input/output:

```
Enter a positive odd integer: 5↵
sum=0.84166664
```

6.7. Example (Khayyam-Pascal triangle). The binomial coefficients are calculated using the following formula:

$$c(i, j) = \frac{j!}{i!(j-i)!}, \quad i = 0, 1, \dots, j$$

$$c(i, j) = \frac{j!}{i!(j-i)!}, \quad i = 0, 1, \dots, j$$

Starting with $j = 0$, consecutively increase the amount of j , and write the results for each j in a separate row. Accordingly, a number of integers arrayed in the shape of a triangle are obtained, called the **Khayyam-Pascal triangle**. The following pattern illustrates this triangle for $j = 0, 1, 2, 3, 4, 5$, and 6 .

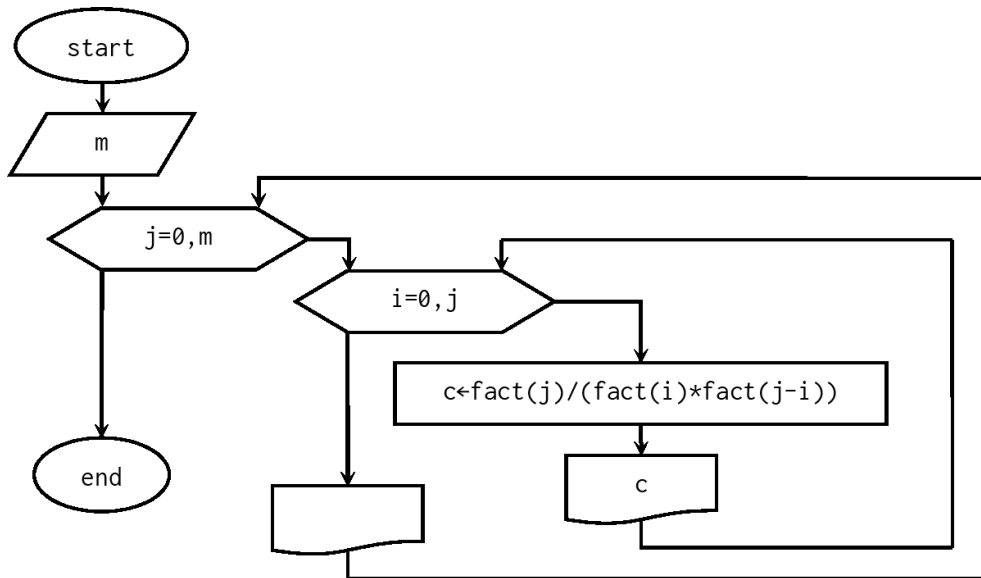


Fig. 6.7: Khayyam-Pascal triangle algorithm.

```

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1

```

```

1      1
1      2      1
1      3      3      1
1      4      6      4      1
1      5      10     10     5      1
1      6      15     20     15     6      1

```

Write an algorithm to read the positive integer m and produce this triangle. then, print it like the above shape using the function `fact()`.

Solution. Clearly, two for loops produce and print this triangle. The for loop with the j variable organizes the rows and that with the i variable calculates and prints the numbers of each row (Fig. 6.7).

In addition, the inner loop produces and prints one row of the triangle for each value of j received from the outer loop. Further, an appropriate space should be considered between the two outputs in this printing. Then, the current row is broken, using the statement `cout<<endl` and the process is continued. The break in the row is displayed in the flowchart by an empty print. The result is programs P6_7.

C++ codes:

```
// Program P6_7 to produce
// Khayyam_Pascal triangle
#include <iostream>
#include <iomanip>

using namespace std;
long fact(int);
int main() {
    int i, j, m;
    long int c;
    cout<<"Enter the number of the rows: ";
    cin>>m;
    for (j=0; j<=m; j++) {
        for (i=0; i<=j; i++) {
            c=fact(j)/(fact(i)*fact(j-i));
            cout<<setw(4)<<c;
        }
        cout<<endl;
    }
    return 0;
}
//*****
long fact(int n) {
    int c;
    long f;
    for (f=1, c=1; c<=n; f*=c, c++);
    return f;
}
```

Input/output:

```
Enter the number of the rows: 6↵
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
```

Java codes:

```
// Program P6_7 to produce
// Khayyam_Pascal triangle
import java.util.Scanner;
class P6_7 {

    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int i, j, m;
        long c;
        System.out.print(
            "Enter the number of rows: ");
        m=read.nextInt();
        for (j=0; j<=m; j++) {
            for (i=0; i<=j; i++) {
                c=fact(j)/(fact(i)*fact(j-i));
                System.out.printf("%-4d", c);
            }
            System.out.println();
        }
        read.close();
    }
}
//*****
static long fact(int n) {
    int c;
    long f;
    for (f=1, c=1; c<=n; f=f*c, c++);
    return f;
}
}
```

Input/output:

```
Enter the number of rows: 6↵
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6
```

There is a function named pow() with two arguments in the list of library functions which takes the first argument to the power of the second. In the following example, we design a sub-algorithm with the same action.

6.8. Example. Write a function to get the real number x and integer n and then calculate and return x^n . x^n .

Solution. Choose a function named `power()` for this mean. Recall that the name of a function, as an identifier cannot be any keyword. A clear idea in this calculation is that, whether n is negative or positive, x is multiplied $|n|$ times by itself. This is conducted in **Figure 6.8(a)**.

The output of the for template is $p = n^{|n|}$. $p = n^{|n|}$. Now, return $1/p$ if n is negative, and p otherwise using an if-else template. The complete function is illustrated in **Figure 6.8(b)**. Of course, the direction selected for $n = 0$ is not a matter of importance due to the incompatibility which occurs in the loop.

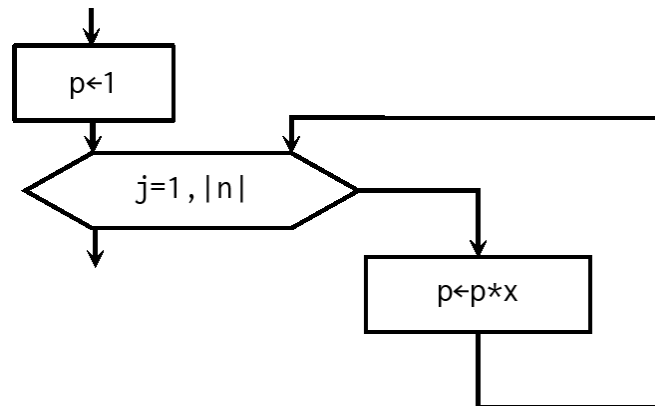


Fig. 6.8(a): Part of power algorithm (x to the power of $|n|$)

What does happen when $n = 0$? In this condition, there is an incompatibility of in the for loop and thus the result of the amount of p will be the initial amount 1. Therefore, there is no difference between selecting to put this condition in one path of the if-then template or the other. We placed it here in the F-path.

Comparing **Flowchart 6.8(a)** with the factorial calculation part of **Flowchart 6.2(a)**, the logic in both parts is found to be equivalent while they differ only in the contents.

As shown in **Flowchart 6.8(a)**, the variable of the loop totally failed to appear in its range. The completed flowchart is illustrated in **Figure 6.8(b)**.



The variable of the loop takes the role of counting. It may appear in the range or not.

In mathematics, the form x^n is called the indeterminate form if $x = 0$ and $n \leq 0$. The main Algorithm 6.8(c) reads the real number x and the integer n . Then, if x^n is in the indeterminate form, it prints a message; otherwise, the amount of x^n is calculated and printed calling the function `power()`. Programs P6_8 demonstrate this algorithm.

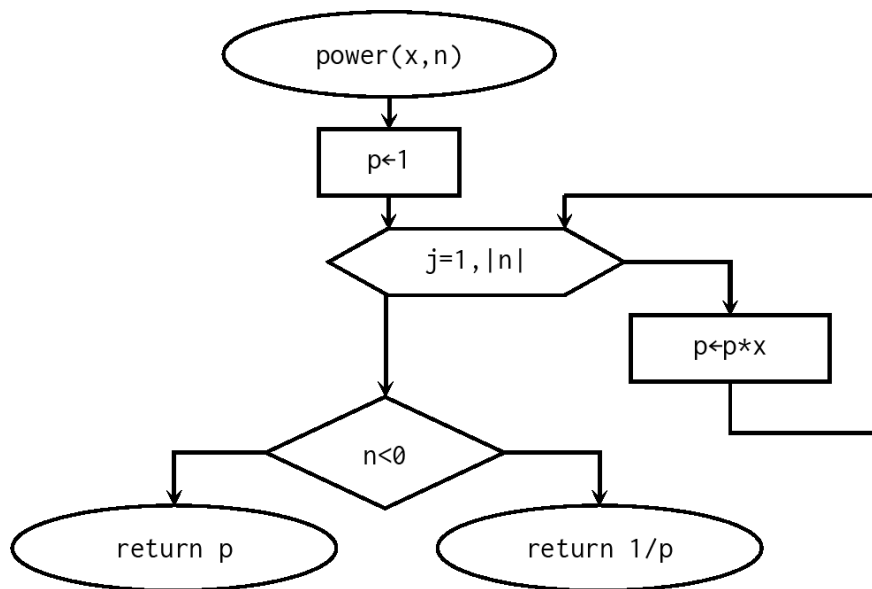


Fig. 6.8(b): Completed power algorithm (x to the power of n).

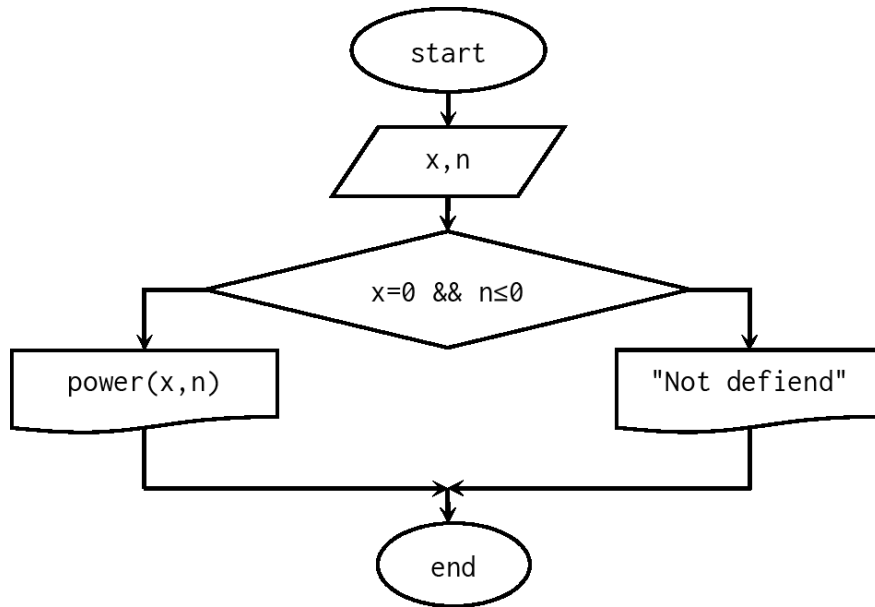


Fig. 6.8(c): A Main algorithm calling the power algorithm.

C++ codes

```
// Program p6_8 for integer power
#include <iostream>
#include <math.h>
using namespace std;
float power(float, int);
int main() {
    int n;
    float x;
    cout<<"Enter x and n: ";
    cin>>x>>n;
    if (x==0 && n<=0)
        cout<<"Not defined";
    else
        cout<<power(x, n);
    return 0;
}
//*****
float power(float x, int n) {
    float p;
    int j;
    for (p=1, j=1; j<=abs(n); p*=x, j++);
    if (n<0)
        return 1/p;
    else
        return p;
}
```

Java codes:

```
// Program P6_8 for integer power
import java.util.Scanner;
class P6_8 {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int n;
        float x;
        System.out.print("Enter x and n: ");
        x=read.nextFloat();
        n=read.nextInt();
        if (x==0 && n<=0)
            System.out.print("Not defined");
        else
            System.out.print(power(x, n));
        read.close();
    }
    //*****
    static float power(float x, int n) {
        float p;
        int j;
        for (p=1, j=1; j<=Math.abs(n);
            p*=x, j++);
        if (n<0)
            return 1/p;
        else
            return p;
    }
}
```

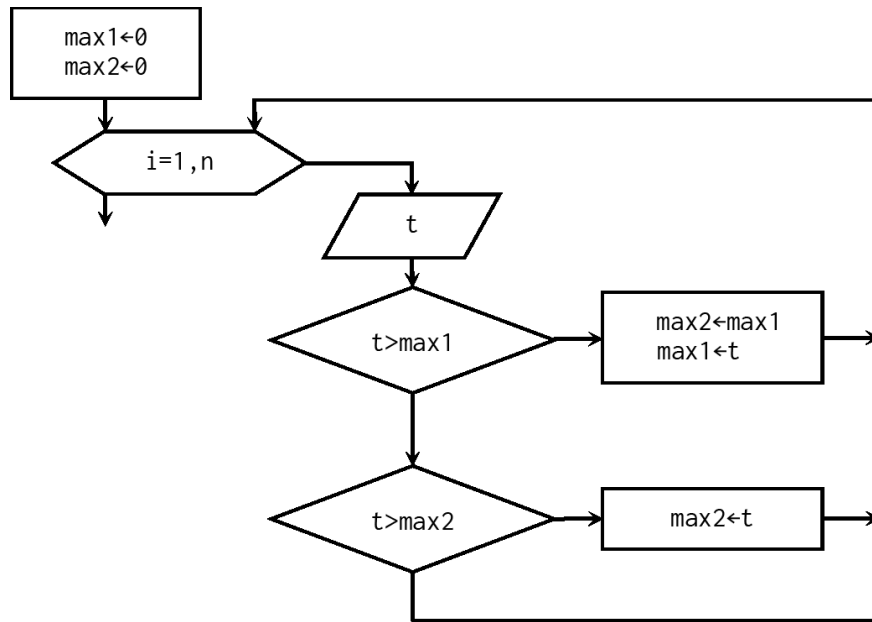


Fig. 6.9(a): Main tasks in the ranking algorithm.

6.9. Example. In a class of n students, two students with the top grade are to be selected to receive awards. Assuming that all the grades are distinct, write an algorithm to read each time the ID number and grade of a student and then print the top two grades together with their ID numbers.

Solution. Let us first describe and analyse an equivalent problem in a simple way: Determine two students from a class of n students who are the tallest between their classmates and announce their height together with their ID numbers, assuming there are no two students with the same height.

We examine the solution of this problem as follows. Consider two empty places and mark them as the $max1$ and $max2$ students. Assume the empty places as students with zero heights. Now implement the following instructions:

1. Call a student named the t student
2. If the t student is taller than the $max1$ student, then substitute the $max1$ student for the $max2$ and the t student for the $max1$.
3. Otherwise, if the t student is taller than the $max2$ student, then substitute the t student for the $max2$.

Suppose that after implementing the above instructions, the student who takes no place or loses its place is asked to leave the class.

Examining all the students this way, the *max1* and *max2* students are the required ones.

As shown in **Flowchart 6.9(a)**, the three above instructions are the range of a for template with 40 repetitions. The empty places considered at first are indeed the zero initial values of *max1* and *max2*.

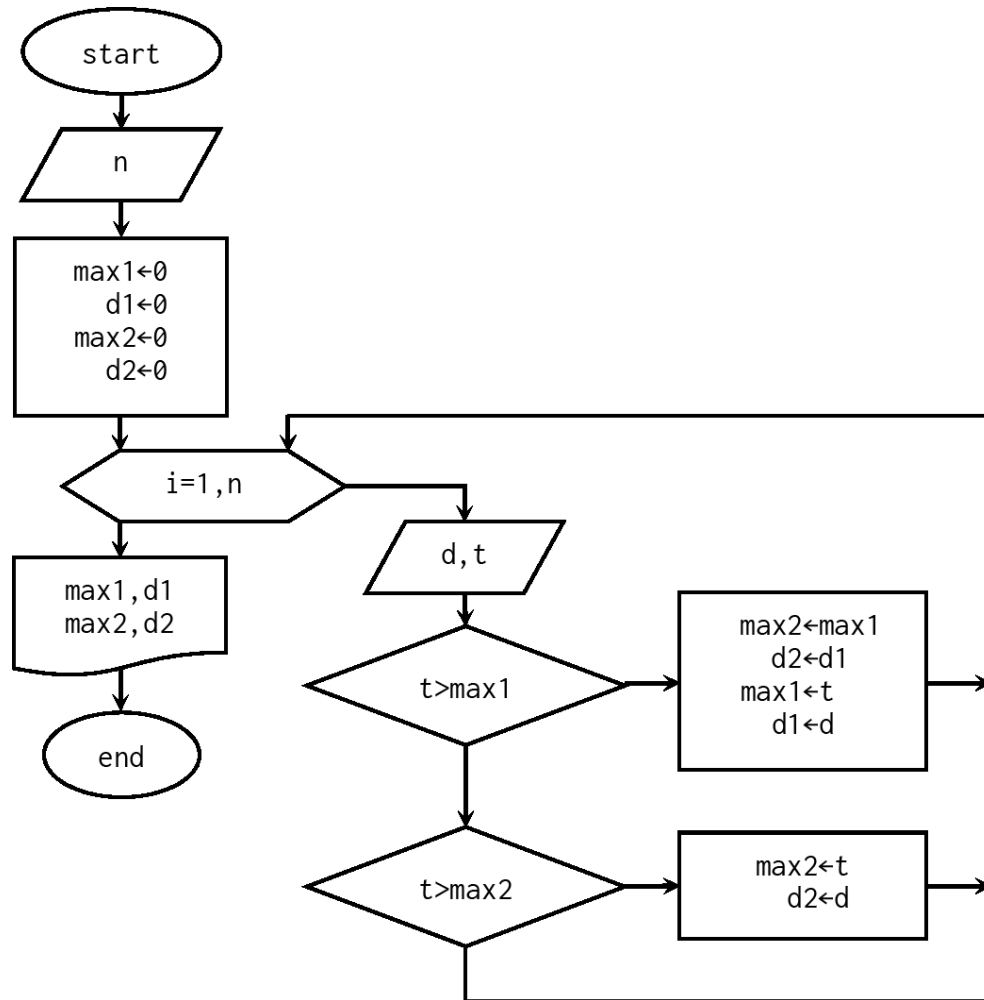


Fig. 6.9(b): Completed ranking algorithm.

What about the ID numbers? We know that the grade and the ID number are two related quantities, as if they are two faces of a coin. Therefore, in correspondence with the three variables *t*, *max1*, and *max2*, we select the other three variables for their ID number, as *d*, *d1* and *d2*, respectively. The complete flowchart is depicted in **Figure 6.9(b)**. Programs P6_9 is the translation of **Flowchart 6.9(b)** into C++ and Java codes.

C++ codes:

```
// Program P6_9 for ranking the
// first two students among n students
#include <iostream>
using namespace std;
int main() {
    float t, max1=0, max2=0;
    int n, i, d, d1=0, d2=0;
    cout<<"Enter the number of students: ";
    cin>>n;
    for (i=1; i<=n; i++) {
        cout<<"Enter the ID and average: ";
        cin>>d>>t;
        if (t>max1) {
            max2=max1; d2=d1;
            max1=t; d1=d;
        }
        else if (t>max2) {
            max2=t; d2=d;
        }
    }
    cout<<"First rank: "<<max1
        <<" with ID: "<<d1<<endl;
    cout<<"Second rank: "<<max2
        <<" with ID: "<<d2;
    return 0;
}
```

Input/output:

```
Enter the number of students: 4↵
Enter the ID and average: 19221001 12.75↵
Enter the ID and average: 19221002 19.25↵
Enter the ID and average: 19221003 14↵
Enter the ID and average: 19221004 18.5↵
First rank: 19.25 with ID: 19221002
Second rank: 18.5 with ID: 19221004
```

Java codes:

```
// Program P6_9 for ranking the
// first two students among n students
import java.util.Scanner;
class P6_9 {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        float t, max1=0, max2=0;
        int n, i, d, d1=0, d2=0;
        System.out.print("Enter the number "
            + "of students: ");
        n=read.nextInt();
        for (i=1; i<=40; i++) {
            System.out.print(
                "Enter the ID and average: ");
            d=read.nextInt();

            t=read.nextFloat();
            if (t>max1) {
                max2=max1; d2=d1;
                max1=t; d1=d;
            }
            else if (t>max2) {
                max2=t; d2=d;
            }
        }
        System.out.println("First rank: "
            + max1 + " with ID: " + d1);
        System.out.print("Second rank: "
            + max2 + " with ID: " + d2);
        read.close();
    }
}
```

Input/output:

```
Enter the number of students: 4↵
Enter the ID and average: 19221001 12.75↵
Enter the ID and average: 19221002 19.25↵
Enter the ID and average: 19221003 14↵
Enter the ID and average: 19221004 18.5↵
First rank: 19.25 with ID: 19221002
Second rank: 18.5 with ID: 19221004
```

6.9.1. Exercise. Arrange the implementation table of the previous flowchart for five students with various inputs.


6.9.2. Exercise. Repeat **Example 6.9**, this time without the assumption that all the grades are distinct.

6.9.3. Exercise. Repeat **Example 6.9** for three top ranks instead of the two, first, by assuming that all the grades are distinct and the second time in a general case.

We briefly and usefully explain the jumping statements of the `continue` and `break`. In **Chapter 4**, we used the statement `break` to break out of the `switch` statement. This statement can be employed in the loops as well. When the `break` statement is encountered inside a loop, the loop is immediately terminated and the program control continues at the next statement following the loop. However, the `continue` statements are only used in the loops. In the case where the `continue` statement is in the range of a loop, the control is shifted to the first statement of the loop and the loop is continued. As an example, the direction of the run control is demonstrated in the following parts for both the `continue` and `break` statements.

The `continue` statement in C++ and Java


```
for (...) {  
    ...  
    continue;  
    ...  
}
```



The `break` statement in C++ and Java

```
for (...) {  
    ...  
    break;  
    ...  
}
```

out of loop ←



On the other hand, the Java programming language supports more facilities for the `continue` and `break` statements using the suffix label with the syntax below:

The labelled `continue` statement in Java

```
continue label;
```

Effect: `continue` from the *label* posited up.

Several notes should be considered in this regard.

The labelled `break` statement in Java

```
break label;
```

Effect: `break` and run from the *label* posited down.

- As mentioned above, the label in the continue statement should be placed up while it may appear up or down for the break statement.
- The statement after the label should be a loop or a block.
- In the loops, the above-mentioned syntax is the same as the statement without the suffix if the label is exactly before the loop where the statement is in.
- We can have the statements before the labels whereas no statement should exist between a label and the loop or block after that.

A prototype for the continue and break statements (with or without the label) can be found in the following model in which some comments are explained on the run flow in the literature of the codes in the loops.

```

outer:
outer loop {
  inner:
  inner loop {
    continue;          // go to the label 'inner' and continue
    continue inner;    // the same as continue (above)
    continue outer;    // go to the label 'outer' and continue
    break;             // break out of inner loop
    break inner;       // the same as break (above)
    break outer;       // break out of the outer loop
  }
}

```

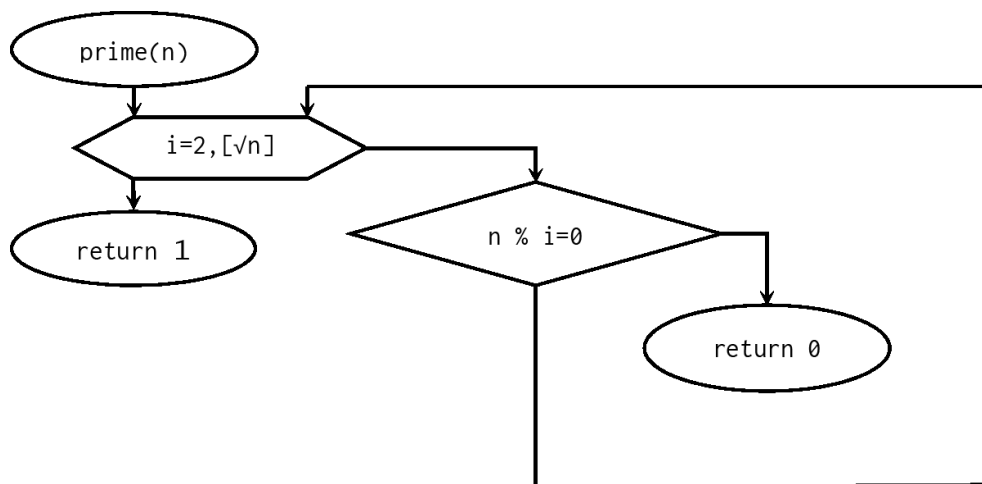


Fig. 6.10(a): The function prime() (the first technique: returning 0 upon divisibility).

It is worth mentioning that the labelled continue and break statements are only used in Java. We will employ the continue and break statements in various examples in the remaining chapters of the book.

6.10. Example. Design a function sub-algorithm named `prime()` in order to receive an integer n , assuming that $n > 1$. Then, determine whether or not the number is a prime number by returning one of the integers 1 or 0, respectively (one can return the Boolean values `true` or `false` instead of 1 or 0, respectively).

Solution. Recall that a positive integer $n > 1$ is called prime if its only divisors are 1 and n . Furthermore, all the prime numbers are odd except for 2. Therefore, we should work on the divisors of the number n in a for template. To this end, it suffices to consider the divisibility of n to the numbers from 2 to $n - 1$ or, even less, to $[\sqrt{n}]$, where $[k]$ stands for the least integer close to k . Accordingly, four various techniques are recommended for solving this problem.

First technique. Consider the divisibility of n by the variable of the loop inside it. Upon reaching the first divisibility, return 0. However, the loop applies all the repetitions if no divisibility occurs. This implies that the number n is prime and hence 1 should be returned after exiting the loop. **Flowchart 6.10(a)** visualizes this technique. The translation of the first technique is represented below.

C++ codes:

```
int prime(int n) {
    int i;
    for (i=2; i<=floor(sqrt(n)); i++)
        if (n%i==0)
            return 0;
    return 1;
}
```

Java codes:

```
static int prime(int n) {
    int i;
    for (i=2; i<=Math.floor(Math.sqrt(n)); i++)
        if (n%i==0)
            return 0;
    return 1;
}
```

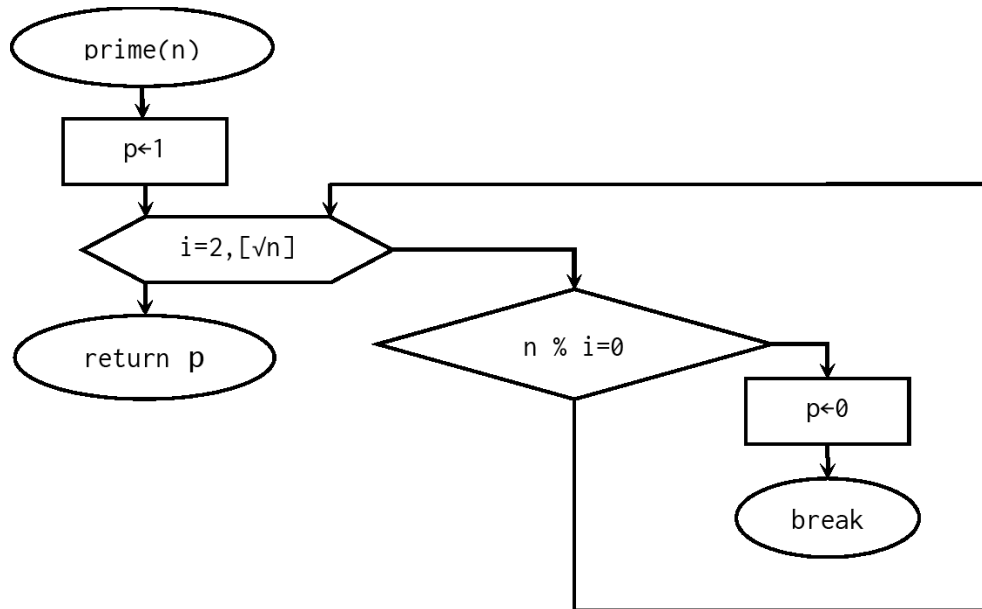


Fig. 6.10(b): The function prime() (the second technique: breaking upon divisibility).

Second technique. At first, assume that n is prime and assign 1 to p , which is the carrier of the function. If the divisibility occurs, then substitute 0 for p which means that n is not prime. After exiting the loop, the value of p returns the result of prime-testing. This technique is demonstrated in **Flowchart 6.10 (b)** in which a break instruction prevents unnecessary repetitions. The subprogram in accord with the flowchart is similar to that of the first technique.

The subprogram in accord with the **Flowchart 6.10(b)** is presented below:

C++ codes:

```

int prime(int n) {
    int i, p;
    p=1;
    for (i=2; i<=floor(sqrt(n)); i++)
        if (n%i==0) {
            p=0; break;
        }
    return p;
}

```

Java codes:

```

static int prime(int n) {
    int i, p;
    p=1;
    for (i=2; i<=Math.floor(Math.sqrt(n)); i++)
        if (n%i==0) {
            p=0; break;
        }
}

```

Third technique. The number of divisors except for 1 is counted by the variable t in this technique. The value zero for t implies that the number n is prime after exiting the loop; otherwise, it is not prime. **Figure 6.10(c)** is the

flowchart of this technique. The corresponding subprogram of the third technique is:

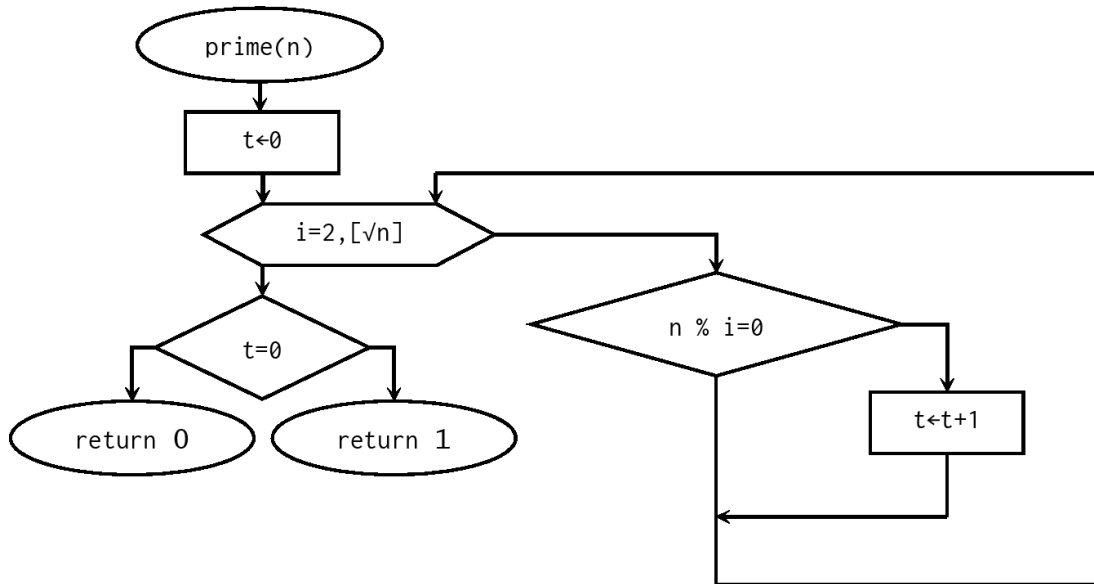


Fig. 6.10(c): The function prime() (the third technique: counting the divisors).

C++ codes:

```

int prime(int n) {
    int i, t;
    t=0;
    for (i=2; i<=floor(sqrt(n)); i++)
        if (n%i==0) t++;
    if (t==0)
        return 1
    else
        return 0;
}
  
```

Java codes:

```

static int prime(int n) {
    int i, t ;
    t=0;
    for (i=2; i<=Math.floor(Math.sqrt(n)); i++)
        if (n%i==0) t++;
    if (t==0)
        return 1
    else
        return 0;
}
  
```

Fourth technique. Use a variable named *s* as a switch. By a switch we mean a variable which records a conversion of status related to a condition in the loop, possibly after one or several instructions. We assign 0 to *s* (switch off) before starting the loop. Upon the occurrence of divisibility, the value of *s* changes to 1 (switch on). Then, we return the result based on whether the switch is off or on. **Flowchart 6.10(d)**, analogy to **Flowchart 6.10(b)**, illustrates this technique.

Comparing **Flowcharts 6.10(c)** and **6.10(d)**, an equivalency is observed between their logic and their codes.

C++ codes:

```
int prime(int n) {
    int i, s;
    s=0;
    for (i=2; i<=floor(sqrt(n)); i++)
        if (n%i==0) {
```

Java codes:

```
static int prime(int n) {
    int i, s;
    s=0;
    for (i=2; i<=Math.floor(Math.sqrt(n)); i++)
        if (n%i==0) {
```

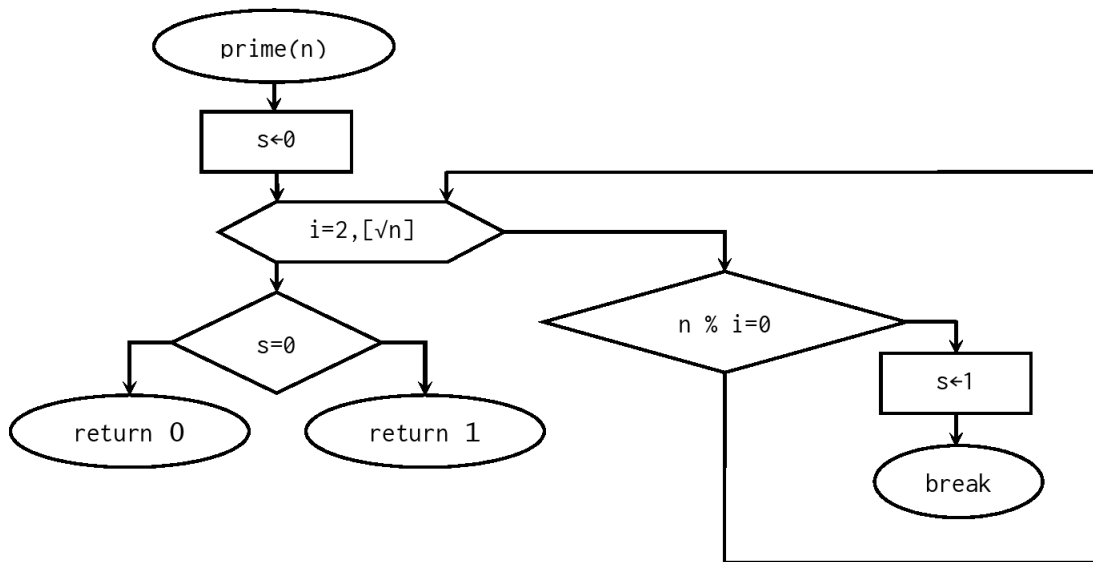


Fig. 6.10(d): The function prime() (the forth technique: using a switch).

```

    p=0;
    break:
  }
  if (s==0)
    return 1
  else
    return 0;
}

```

```

    p=0;
    break:
  }
  if (s==0)
    return 1
  else
    return 0;
}

```

Have you thought that how the function prime() (any technique) deals with the cases $n = 2$ or 3 ? You can find the answer in the incompatibilities of the loop.

6.11. Example. Write a main algorithm to read a an integer n with $n > 2$. Then, determine and print the prime numbers smaller than n using the function prime().

Solution. Figure 6.11 displays the required flowchart and Programs P6_11 hold its translation into C++ and Java codes.

C++ codes:

```
// Program P6_11 to determine and
// print the prime numbers less
// than k using the function fact()
#include <iostream>
#include <math.h>
using namespace std;
int prime(int);
int main() {
    int k, n;
    cout<<"Enter an integer k>2: ";
    cin>>k;
    for (n=2; n<k; n++)
```

Java codes:

```
// Program P6_11 to determine and
// print the prime numbers less
// than k using the function fact()
import java.util.Scanner;
class P6_11 {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int k, n;
        System.out.println("Enter an integer k>2: ");
        k=read.nextInt();
        for (n=2; n<k; n++)
            if (prime(n)==1)
```

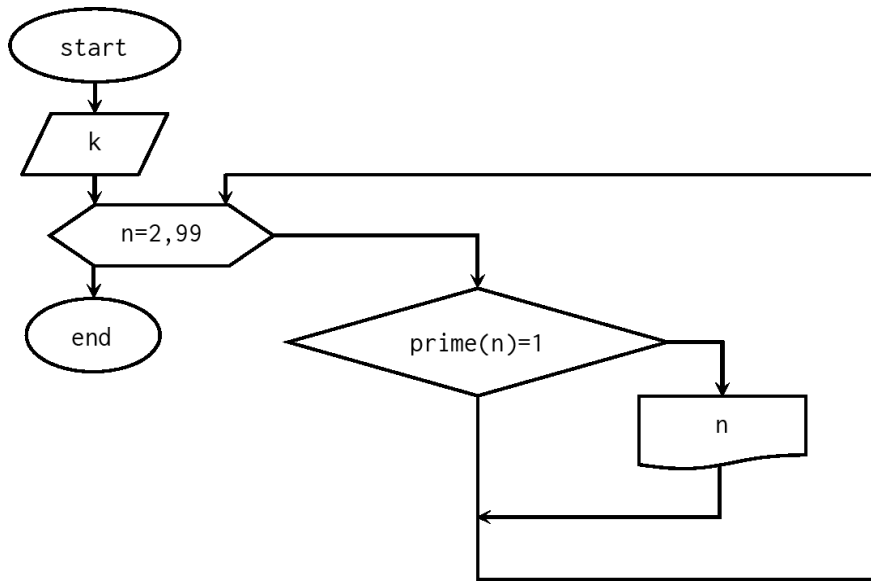


Fig. 6.11: Printing the prime numbers less than 100 calling the function prime().

```

    if (prime(n)==1)
        cout<<n<<endl;
    return 0;
}
//*****
int prime(int n) {
    int i, p;
    p=1;
    for (i=2; i<=floor(sqrt(n)); i++)
        if (n%i==0)
            return 0;
    return p;
}

```

Input/output:

```

Enter an integer k>2: 10↵
2
3
5
7

```

```

        System.out.println(n);
        read.close();
    }
    //*****
    static int prime(int n) {
        int i, p;
        p=1;
        for (i=2; i<=Math.floor(Math.sqrt(n)); i++)
            if (n%i==0)
                return 0;
        return p;
    }
}

```

Input/output:

```

Enter an integer k>2: 10↵
2
3
5
7

```

In **Flowchart 6.11**, we can write the condition as the prime(n). In this case, the return integer values 1 and 0 should be replaced by the Boolean values true and false, respectively, in the right Java program while, in the left C++ program, there is no need to perform this change. Recall that in C++, the integers 1 and 0 are equivalent to the Boolean values true and false, respectively.

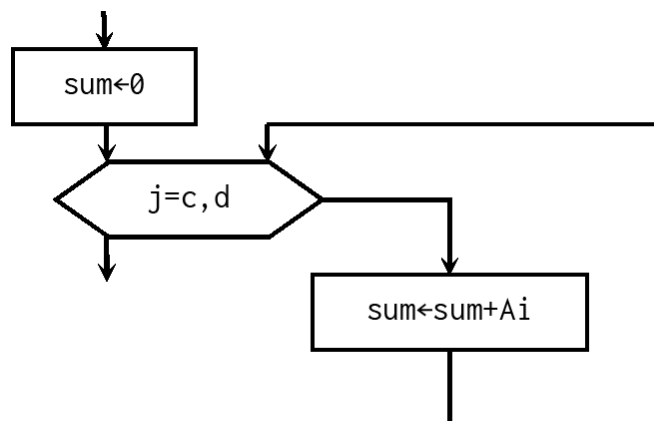


Fig. 6.12(a): Calculating single series.

6.2 Series

6.12. Series via the for template. The for templates are widely used in the algorithms and programs concerned with the series calculations due to their counting nature. In general, to calculate the single series

$$\sum_{i=a}^b A_i,$$

we can construct **Flowchart 6.12(a)**. In the series above, assume that A_i is the general term of the series which is normally dependent on i . Moreover, the algorithm of the following double series can be constructed as **Flowchart 6.12 (b)**.

$$\sum_{i=a}^b \sum_{j=c}^d A_{ij}$$

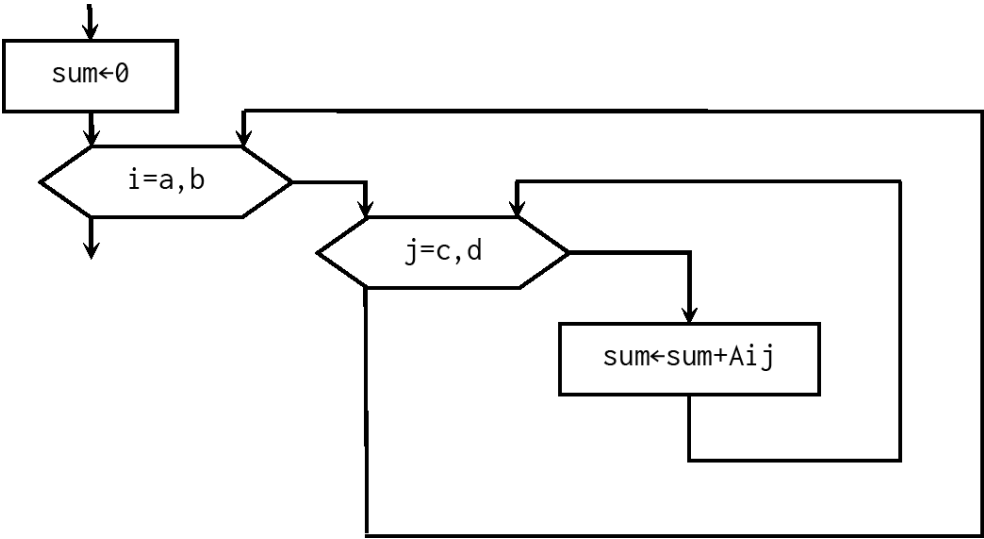


Fig. 6.12(b): Calculating double series.

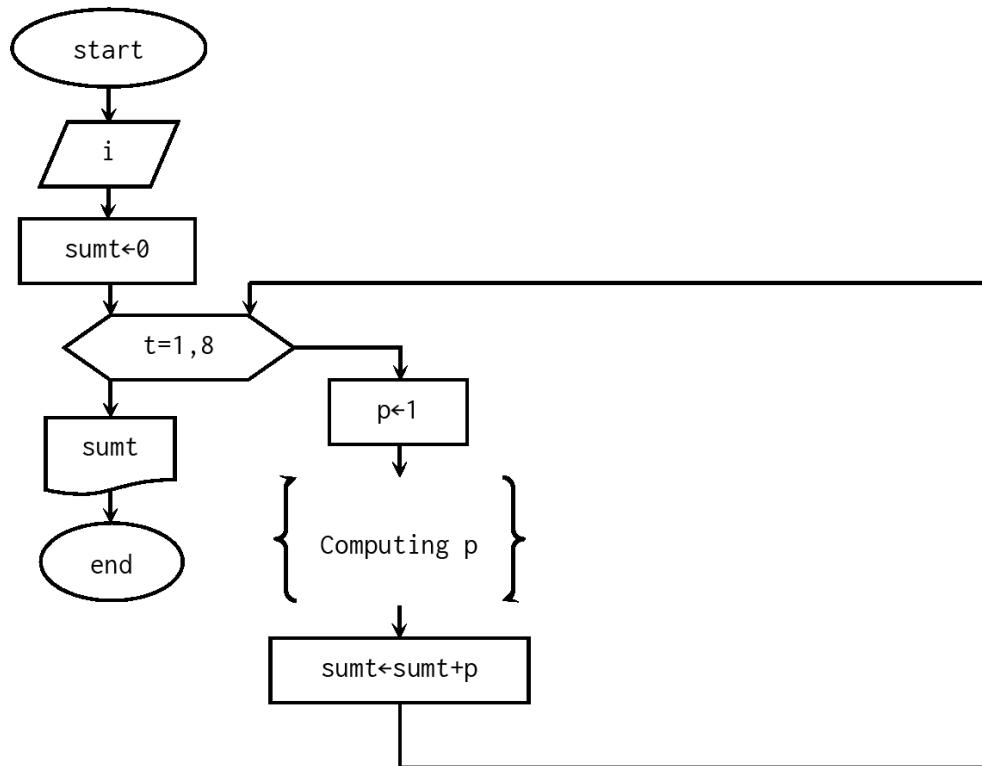


Fig. 6.13(a): Sketch of calculating the single series in Example 6.13.

Here again, A_{ij} is assumed as the general term of the series which depends on both i and j . The double series pattern can be naturally extended to the multi series.

6.13. Example. Write an algorithm to read the positive integer i and then calculate and print the following series.

$$\sum_{t=1}^8 (i^2 + t^2)^{i+t}.$$

Solution. Combine Flowcharts 6.8(a) and 6.12(a) and add the input and output templates to it. Accordingly, change the name of the general term from A_i to p in Flowchart 6.12(a) and add it to the repetitive sum, which we name $sumt$ (Fig. 6.13(a)).

Considering that the power of $i + t$ is positive, we use a special form of Flowchart 6.8(a) shown in Figure 6.13(b) to calculate $p = (i^2 + t^2)^{i+t}$. The completed algorithm is obtained by putting Flowchart 6.13(b) in its position in Flowchart 6.13(a) (Fig. 6.13(c)).

6.13.1. Exercise. Write the programs of the completed Algorithm 6.13(c) in both C++ and Java codes.

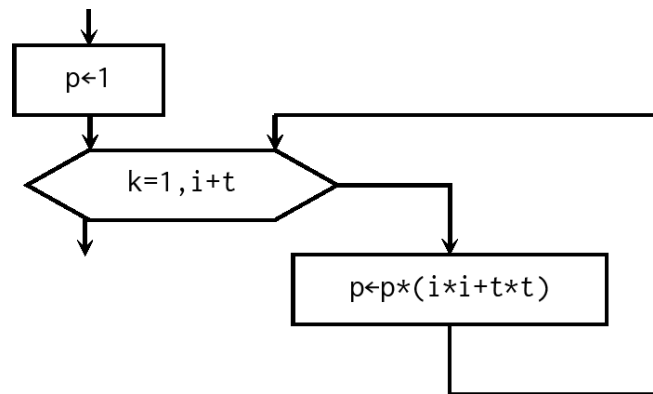


Fig. 6.13(b): Calculating $i^2 + t^2$ to the power of $i + t$.

The above algorithm can be written in a more simple way using the library function `pow()`, or calling the function `power()` in Example 6.8. This is performed in Flowchart 6.13(d). Programs P6_13 are the translation of Flowchart 6.13(d) into C++ and Java codes.

C++ codes:

```

// Program P6_13 to compute a
// single series
#include <iostream>
#include <math.h>
using namespace std;
int main() {

```

Java codes:

```

// Program P6_13 to compute a
// single series
import java.util.Scanner;
class P6_13 {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);

```

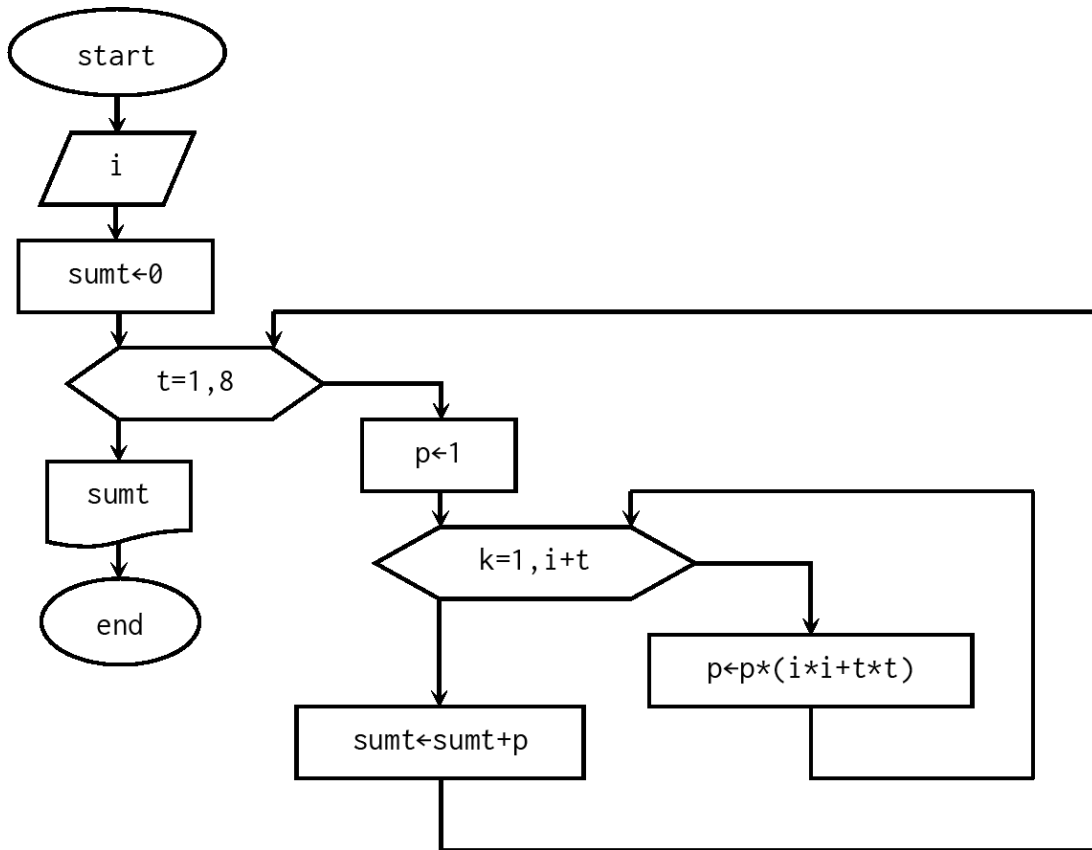


Fig. 6.13(c): Calculating the single series in [example 6.13](#) (complete algorithm).

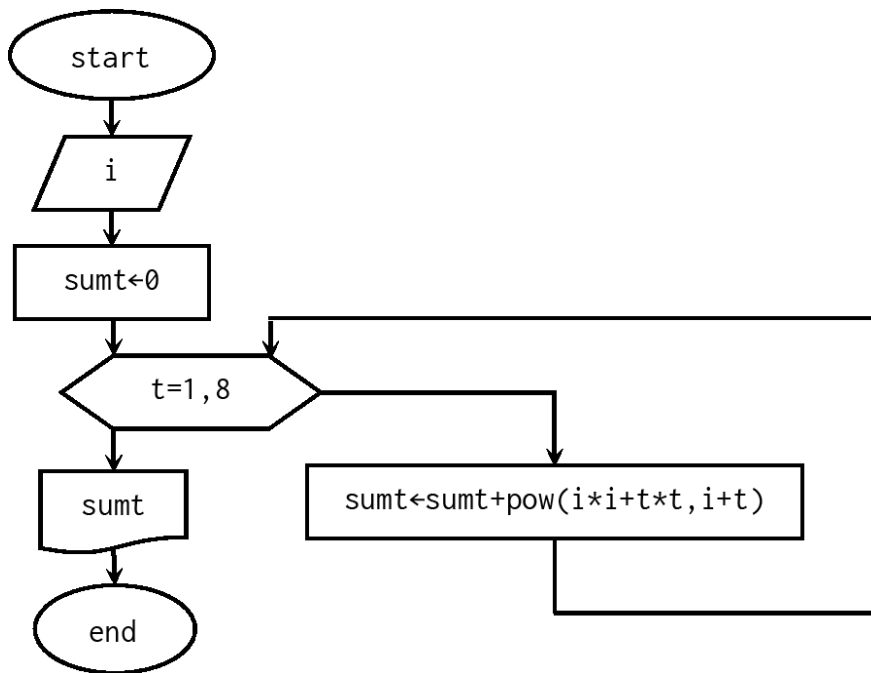


Fig. 6.13(d): Calculating the single series in [Example 6.13](#) (using the pow() function).

```

int i, t;
double sumt;
cout<<"Enter a positive integer: ";
cin>>i;
sumt=0;
for (t=1; t<=8; t++)
    sumt+=pow(i*i+t*t, i+t);
cout<<"The answer is: "<<sumt;
return 0;
}

```

Input/output:

Enter a positive integer: 3↵
The answer is: 3.14158e+020

```

int i, t;
double sumt;
System.out.print("Enter a positive "
                + "integer: ");
i=read.nextInt();
sumt=0;
for (t=1; t<=8; t++)
    sumt+=Math.pow(i*i+t*t, i+t);
System.out.print("The answer is: " + sumt);
read.close();
}
}

```

Input/output:

Enter a positive integer: 3↵
The answer is: 3.141582482478335E20

6.14. Example. Write an algorithm to calculate and print the following series.

$$\sum_{t=1}^8 (i^2 + t^2)^{i+t}$$

Solution. As shown in [Flowchart 6.14](#), this double series can be created using the pattern of [Flowchart 6.12\(b\)](#).

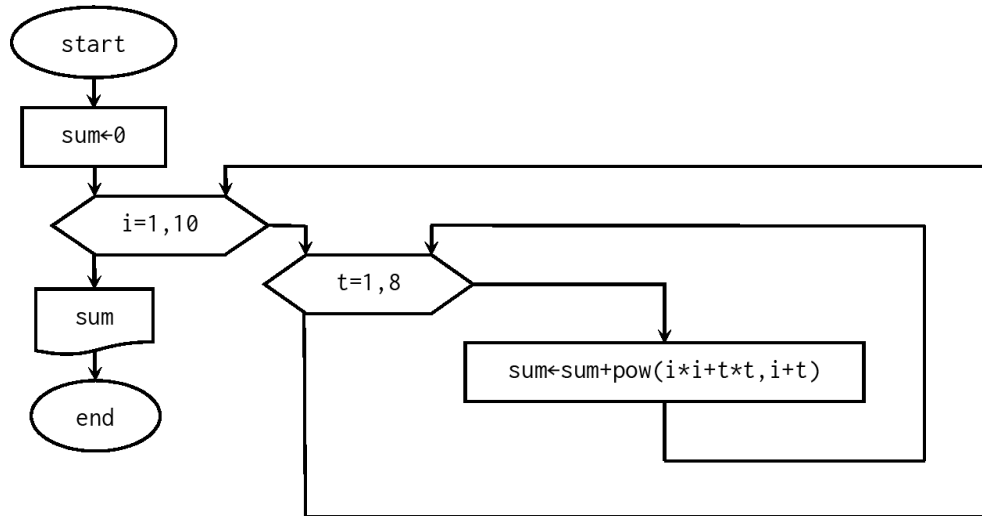


Fig. 6.14: Calculating the double series in [Example 6.14](#) (using the pow() function).

Tab. 6.14: Implementation table of [Flowchart 6.14](#) for $i = 1, 2$ and $t = 1, 2, 3$.

i	t	sumt	sum
			0
1		0	
	1	$(1^2 + 1^2)^{1+1}$	
	2	$(1^2 + 1^2)^{1+1} + (1^2 + 2^2)^{1+2}$	
	3	$(1^2 + 1^2)^{1+1} + (1^2 + 2^2)^{1+2} + (1^2 + 3^2)^{1+3}$	$(1^2 + 1^2)^{1+1} + (1^2 + 2^2)^{1+2} + (1^2 + 3^2)^{1+3}$
2		0	
	1	$(2 + 1^2)^{2+1}$	
	2	$(2 + 1^2)^{2+1} + (2^2 + 2^2)^{2+2}$	
	3	$(2 + 1^2)^{2+1} + (2^2 + 2^2)^{2+2} + (2^2 + 3^2)^{2+3}$	$(1^2 + 1^2)^{1+1} + (1^2 + 2^2)^{1+2} + (1^2 + 3^2)^{1+3} + (2 + 1^2)^{2+1} + (2^2 + 2^2)^{2+2} + (2^2 + 3^2)^{2+3}$

The details of implementing [Flowchart 6.14](#) for i from 1 to 2 and t from 1 to 3 are summarized in [Table 6.14](#). The following rule should be highlighted in arranging the implementation table for the nested loops.



The rule of implementation table for nested loops. The best technique to arrange an implementation table for the nested loops is to allocate necessary parts of the table each for a repetition of the outer loop and write the results of implementing the range of the inner loop in the related part.

Programs P6_14 translate **Flowchart 6.14** into C++ and Java codes.

C++ codes:

```
// Program P6_14 to compute a
// double series
#include <iostream>
#include <math.h>
using namespace std;
int main() {
    double sum=0;
    for (int i=1; i<=10; i++)
        for (int t=1; t<=8; t++)
            sum+=pow(i*i+t*t, i+t);
    cout<<"The answer is: "<<sum;
    return 0;
}
```

Output:

The answer is: 7.37964e+039

Java codes:

```
// Program P6_14 to compute a
// double series
class P6_14 {
    public static void main(String[] args) {
        double sum=0;
        for (int i=1; i<=10; i++)
            for (int t=1; t<=8; t++)
                sum+=Math.pow(i*i+t*t, i+t);
        System.out.println("The answer is: " + sum);
    }
}
```

Output:

The answer is: 7.37963600959878E39

6.15) Trapezoidal approximated integration. The trapezoidal method is one of the methods for the approximation integration of the definite integral

$$\int_a^b f(x)dx.$$

As shown in **Figure 6.15(a)**, the integration interval $[a, b]$ is divided into n equal parts and the split points are marked as $a = x_0, x_1, \dots, x_{n-1}, x_n = b$

$a = x_0, x_1, \dots, x_{n-1}, x_n = b$ in this method.

The mentioned integral is the area under the curve of $y = f(x)$ which restricted between the two lines $x = a$ and $x = b$. Additionally, the amount of this integral is approximately equal to the sum of the areas related to the

trapezoids created by the division subintervals of the integration interval. Given a straightforward calculation, it can be checked that this sum can be obtained by the formula:

$$T = \frac{1}{2}h (f(a) + 2f(x_1) + 2f(x_2) + \dots + 2f(x_{n-1}) + f(b)),$$

$$T = \frac{1}{2}h(f(a) + 2f(x_1) + 2f(x_2) + \dots + 2f(x_{n-1}) + f(b)),$$

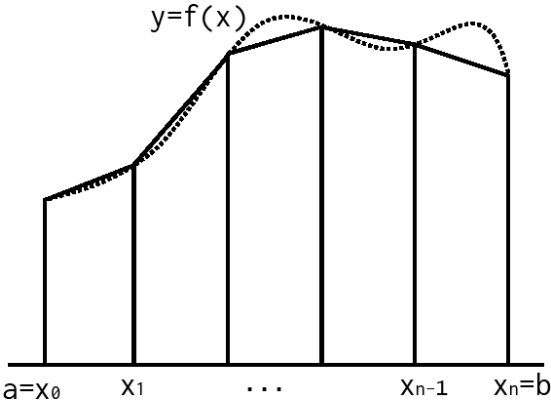


Fig. 6.15(a): The sum of the trapezoids' areas is the approximated integral.

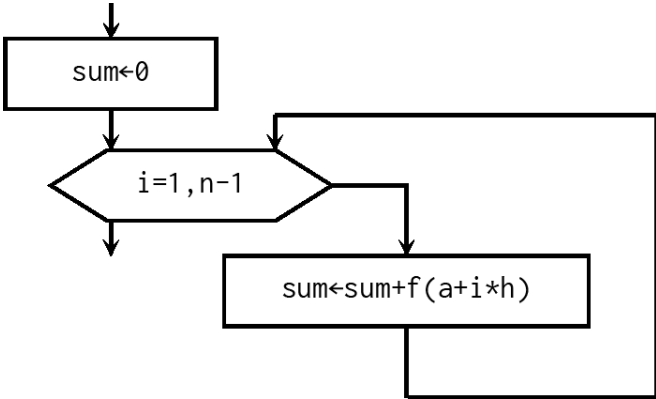


Fig. 6.15(b): Calculating the series of the trapezoid method.

where h is the length of each division subinterval. To calculate the approximated integral

$$\int_a^b e^{-x^2} dx, \int_a^b e^{-x^2} dx,$$

sketch an algorithm including the following parts using the trapezoidal technique.

1. The function sub-algorithm $f()$ which receives a real value x and returns $e^{-x^2} \cdot e^{-x^2}$.
2. The function sub-algorithm $\text{Trap}()$ which receives the integration endpoints a and b as well as the number of division points n , and returns the approximated integral

$$\int_a^b f(x) dx$$

calling the function $f()$ in the first part.

3. The main algorithm which reads the values of the endpoints a and b as well as the division points n and, calling the function $\text{Trap}()$, prints the approximated value of the integral

$$\int_a^b e^{-x^2} dx.$$

Solution. The required approximated value is the sum of the areas of the above-mentioned trapezoids, denoted by T . Note that $x_i = a + ih$ in the above-mentioned formula for T . Therefore, the formula can be rewritten as:

$$T = \frac{1}{2}h \left(f(a) + 2 \sum_{i=1}^{n-1} f(a + ih) + f(b) \right).$$

$$T = \frac{1}{2}h \left(f(a) + 2 \sum_{i=1}^{n-1} f(a + ih) + f(b) \right).$$

The calculation of the series

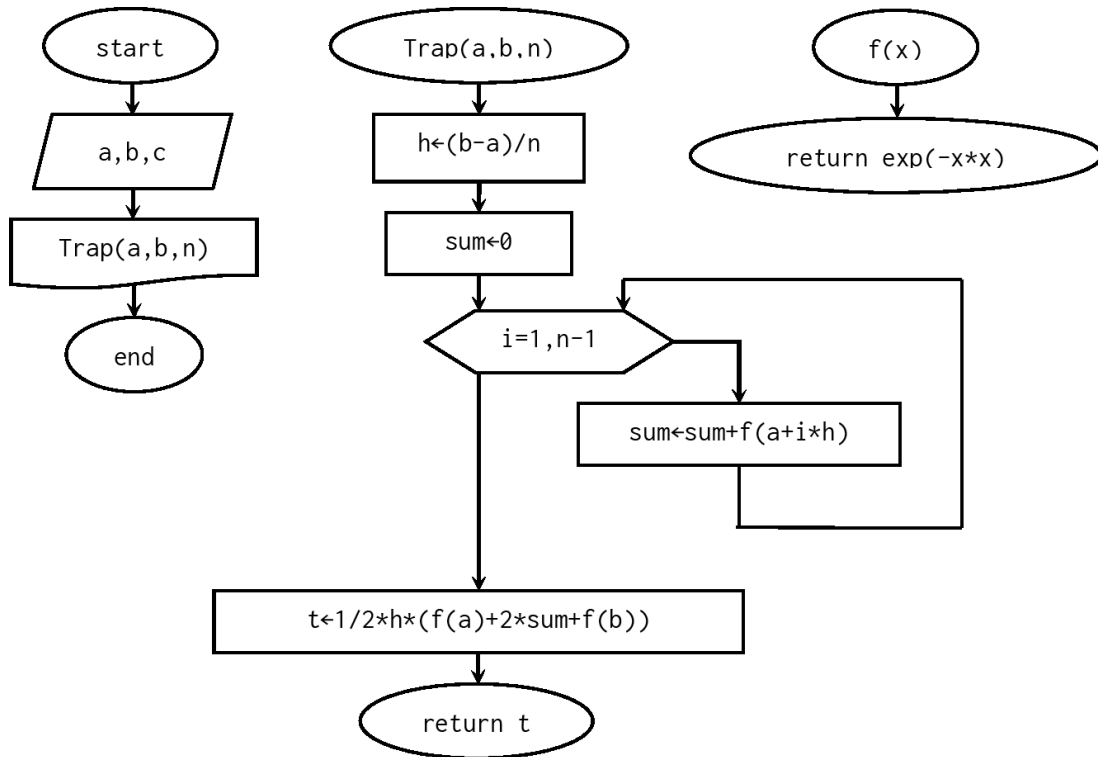


Fig. 6.15(c): Integral approximation by the trapezoid method (completed algorithm).

$$\sum_{i=1}^{n-1} f(a + ih)$$

is the basic part in the above formula. This is conducted in [Flowchart 6.15\(b\)](#). The three required parts are illustrated in [Flowchart 6.15\(c\)](#). Programs P6_15 are the corresponding programs in C++ and Java codes. Programs P6_15 can be used for any function provided that the rule of f is changed in the program.

C++ codes:

```
// Program P6_15 to approximate a
// definite integral by the
// Trapezoid technique. To use,
// replace the involved function
#include <iostream>
#include <math.h>
using namespace std;
double Trap(double, double, int);
int main() {
    double a, b;
    int n;
    cout<<"Enter the end points: ";
    cin>>a>>b;
    cout<<"and number of divisions: ";
    cin>>n;
    cout<<"Approximated amount is: "
        <<Trap(a, b, n);
    return 0;
}
//*****
double f(double x) {
    return exp(-x*x);
}
//*****
double Trap(double a, double b,
            int n) {
    double h, sum;
    h=(b-a)/n;
    sum=0;
    for (int i=1; i<=(n-1); i++)
        sum+=f(a+i*h);
    return 1.0/2*h*(f(a)+2*sum+f(b));
}
```

Input/output:

```
Enter the end points: 1 10↵
and number of divisions: 50↵
Approximated amount is: 0.141392
```

Java codes:

```
// Program P6_15 to approximate a definite
// integral by the Trapezoid technique.
// To use, replace the involved function
import java.util.Scanner;
class P6_15 {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        double a, b;
        int n;
        System.out.print("Enter the end points: ");
        a=read.nextDouble();
        b=read.nextDouble();
        System.out.print("and number of divisions: ");
        n=read.nextInt();
        System.out.print("Approximated amount is: "
            + Trap(a, b, n));
        read.close();
    }
    //*****
    static double f(double x) {
        return Math.exp(-x*x);
    }
    //*****
    static double Trap(double a, double b,
                        int n) {
        double h;
        double sum;
        h=(b-a)/n;
        sum=0;
        for (int i=1; i<=(n-1); i++)
            sum+=f(a+i*h);
        return 1.0/2*h*(f(a)+2*sum+f(b));
    }
}
```

Input/output:

```
Enter the end points: 1 10↵
and number of divisions: 50↵
Approximated amount is: 0.14139148362689577
```

Exercises

In the following exercises: (1) Arrange the implementation table, if needed, (2) Write the complete program, (3) Provide appropriate input notifications and output headings, if any. In addition, the user-defined functions in the text

of the current and the previous chapters may be used unless otherwise is explicitly specified.

6.1. Write an algorithm to read a positive integer n and then calculate and print the sum of the numbers from 1 to n .

6.2. Write an algorithm to read a positive integer n and then calculate and print the sum of the first n even numbers.

6.3. Write an algorithm to read a positive integer n and then calculate and print the sum of the odd numbers smaller than n .

6.4. The square of a positive integer n can be derived from the sum of n consecutive positive odd numbers. For example,

$$6^2 = 1 + 3 + 5 + 7 + 9 + 11. \quad 6^2 = 1 + 3 + 5 + 7 + 9 + 11.$$

Write an algorithm to read a positive integer n and then calculate its square using the above-mentioned method and print it together with n itself.

6.5. Write an algorithm to read two positive integers m and n and then calculate and print their multiplication using the following consecutive sum:

$$nm = m + m + \dots + m \text{ (} n \text{ times).}$$
$$nm = m + m + \dots + m \text{ (} n \text{ times).}$$

6.6. Write an algorithm to read a positive integer n , which is a multiple of 4, and then calculate and print the sum of the multiples of 4, from 4 to n .

6.7. Write an algorithm to read the three positive integers a , b , and m ($a < b$) and then determine and print the number of the multiples of m from a and b .

6.8. Write an algorithm to read the ID number and grade of 20 students and then determine the students with grade < 12 and print them together with their ID numbers. Finally, print the number of these students.

6.9. An airplane at an altitude of h passes above the point P . If its speed is v , then its distance from the point P at the moment t can be calculated using the

formula $d = \sqrt{h^2 + (vt)^2}$. Write an algorithm to read the speed v and altitude h and then calculate and print the distance of the airplane at the moments $t = 1, 2, \dots, 60$.

6.10. Write an algorithm to read the employee number n , the monthly working hours h , and the hourly wage s of the employees in an office including 300 members one by one and then calculate and print the monthly salary of these employees. If an employee has worked over 200 hours per month, the hourly wage will be multiplied by 1.5 for their overtime hours.

6.11. Write an algorithm to read 50 numbers and then separately calculate and print the following items.

- The positive numbers;
- The sum of the positive numbers;
- The number of positive numbers;
- The negative numbers;
- The sum of the negative numbers;
- The number of negative numbers;
- The number of zeroes.

6.12. Write an algorithm to read a positive integer n . Then read n real numbers and determine and print their maximum and minimum.

6.13. Write an algorithm to read the positive integer n and then print the prime numbers smaller than n .

6.14. Write an algorithm to calculate and print the number of primes smaller than 1000 and in the form $4n + 1$.

6.15. Write an algorithm to read the positive even integer n and the real number x and then calculate and print the following sum.

$$1 - \frac{x}{2!} + \frac{x^2}{4!} - \dots + (-1)^{\frac{n}{2}} \frac{x^{\frac{n}{2}}}{n!}.$$

In Exercises 6.16 to 6.23, write the requested algorithm using the necessary user-defined functions in the current chapter, and once again without using such a functions.

6.16. Write an algorithm to read a positive n and then calculate and print the following sum.

$$\sum_{i=1}^n i^i.$$

6.17. Write an algorithm to calculate and print the following sum.

$$\sum_{i=1}^5 i^{i!}.$$

6.18. Write an algorithm to read a positive odd integer n and then calculate and print the following sum.

$$1 - 3^{3!} + 5^{5!} - \dots + (-1)^{\frac{n-1}{2}} n^{n!}.$$

6.19. Write an algorithm to calculate and print the following sum:

$$\sum_{i=1}^5 i!^{i!}.$$

6.20. Write an algorithm to read a positive odd integer n and then calculate and print the following sum.

$$1 - 3!^3 + 5!^5 - \dots + (-1)^{\frac{n-1}{2}} n!^n.$$

$$1 - 3!^3 + 5!^5 - \dots + (-1)^{\frac{n-1}{2}} n!^n.$$

6.21. Write an algorithm to read the real number x and calculate and print the following sum.

$$\sum_{i=1}^7 (-1)^{i+1} i^2 x^i.$$

6.22. Write an algorithm to calculate and print the following sum.

$$\sum_{n=1}^5 \sum_{i=1}^{10} (i^n + n^i).$$

6.23. Write an algorithm to read the positive integer n and then calculate and print the following sum.

$$\sum_{i=1}^{15} i^n + \left(\sum_{i=1}^{15} i \right)^n.$$

6.24. Write an algorithm to calculate and print the following sum.

$$\sum_{i=1}^3 \sum_{j=1}^4 \sum_{k=1}^5 (i + i^j + i^{j^k}).$$

6.25. The approximated integration of the definite integral

$$\int_a^b f(x) dx$$

using the Simpson method is calculated by the formula

$$s = \frac{1}{3}h (f(a) + 4f(x_1) + 2f(x_2) + \dots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(b)).$$

$$S = \frac{1}{3}h(f(a) + 4f(x_1) + 2f(x_2) + \dots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(b)).$$

where a, b, n , and h are the same as in the trapezoidal method in [Example 6.15](#). Here, n is assumed even. To calculate the approximated integral

$$\int_a^b e^{-x^2} dx$$

by the Simpson method, design an algorithm including the three parts mentioned in [Example 6.15](#). Denote the function of the approximated integration by `Simp()` instead of `Trap`.

6.26. Consider the following continued 3-fraction.

$$1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{x}}}$$

Write an algorithm to read the value of the real number x and the positive integer n and then calculate and print a similar continued n -fraction.

6.27. Write an algorithm to read the positive integer n and the real number x and then calculate and print the first n terms of the following series.

$$\frac{1}{x} + \frac{1}{1+\frac{1}{x}} + \frac{1}{2+\frac{1}{1+\frac{1}{x}}} + \frac{1}{3+\frac{1}{2+\frac{1}{1+\frac{1}{x}}}} + \dots$$

$$\frac{1}{x} + \frac{1}{1 + \frac{1}{x}} + \frac{1}{2 + \frac{1}{1 + \frac{1}{x}}} + \frac{1}{3 + \frac{1}{2 + \frac{1}{1 + \frac{1}{x}}}} + \dots$$

6.28. Write an algorithm to read the real number x and then calculate and print the first 10 terms of the following series.

$$\frac{1}{x} - \frac{1}{x+2x^2} + \frac{1}{x+2x^2+3x^3} - \frac{1}{x+2x^2+3x^3+4x^4} + \dots$$

$$\frac{1}{x} - \frac{1}{x + 2x^2} + \frac{1}{x + 2x^2 + 3x^3} - \frac{1}{x + 2x^2 + 3x^3 + 4x^4} + \dots$$

6.29. Write an algorithm to read a positive integer n and then calculate and print all of its positive divisors smaller than n .

6.30. A perfect number is defined to be a positive integer n which is equal to the sum of its positive divisors smaller than n . For example, 28 is a perfect number since $28 = 1 + 2 + 4 + 7 + 14$. Write a function named `perfect()` to receive a positive integer n and then return 1 if n is perfect and 0 otherwise. Afterwards, write a main algorithm to read a positive integer n and specify whether it is a perfect number by printing an appropriate message. Use the function `perfect()` in the main algorithm.

Supplementary exercises

6.1*. Consider the following six 4-line patterns.

```

*           * * * *           *           * * * *           *           * * * *
* *        * * *           * *        * * *           * *        * * *
* * *      * *           * * *      * *           * * *      * * *
* * * *    *           * * * *    *           * * * *    *
(1)        (2)          (3)        (4)          (5)        (6)

```

Now, write a program for each pattern to read a positive integer n and print the n -line corresponding pattern.

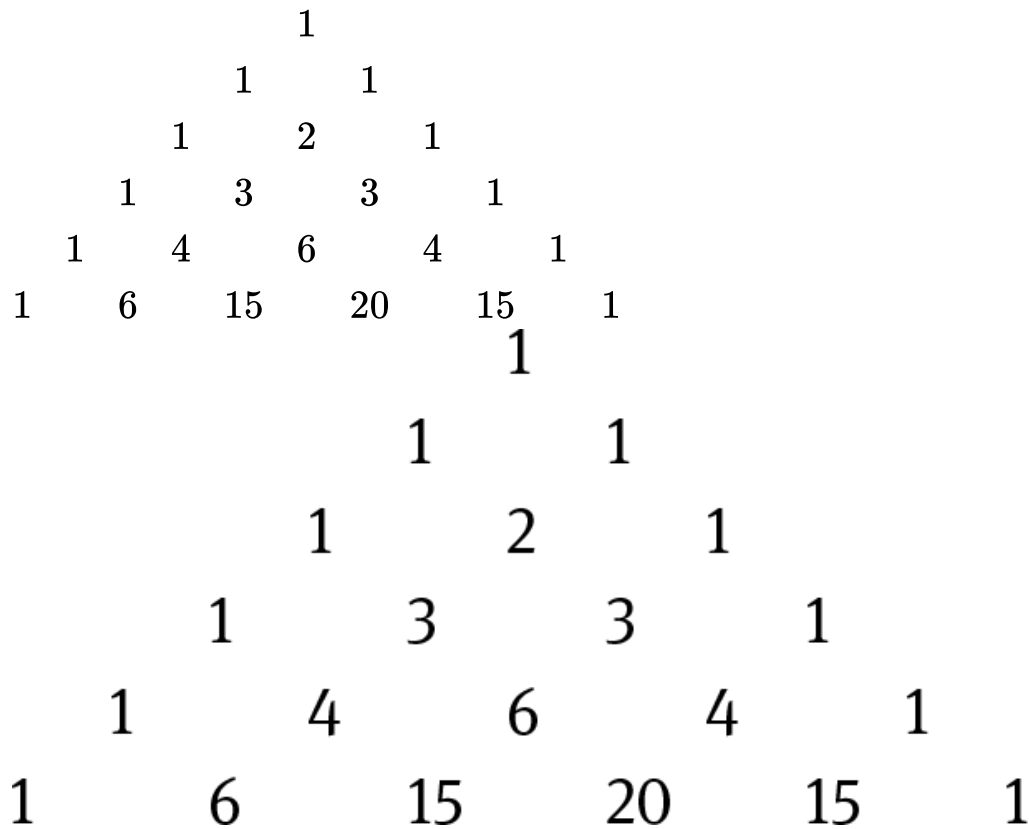
6.2*. Write an algorithm to read the positive integer n and print an n -row triangle as follows.

```

                1
              1  2
            1  2  6
          1  2  6  24
        1
       ⋮
      1  ...
     1  2  6  24
    ⋮
   1  ...          n!

```

6.3*. Modify Algorithm 6.7 so that upon reading n for m , print the Khayyam-Pascal triangle in the following pattern.



6.4 *. Write an algorithm to read the positive integer n and then determine and print the digits of n from right to left.

6.5*. Write an algorithm to determine and print the four-digit positive integers in which the first two digits from left are even while the other two digits are odd. Further, the number of such integers should be printed.

6.6*. Write an algorithm to read a positive integer n and take it to the power of its largest prime divisor and finally, print this integer. For example, if 20 is read then print 20^5 .

6.7*. The sum of all the divisors of the positive integer n , except for n itself, is denoted by $\sigma(n)$. The number n is called perfect, abundant, and deficient if $\sigma(n) = n$, $\sigma(n) > n$, and $\sigma(n) < n$, respectively. Write an algorithm to separately determine and print the perfect, abundant, and deficient numbers less than 100. Furthermore, the number of perfect, abundant, and deficient numbers are printed.

6.8*. For a positive integer n , the Euler phi function, namely, $\varphi(n)$, is defined as the number of all the positive integers smaller than n , which are coprime to n . For example, $\varphi(14) = 6$ since 1, 3, 5, 9, 11, and 13 are the only positive integers less than 14 being coprime to 14. Moreover, this function can be calculated using the following formula.

$$\varphi(n) = n \left(1 - \frac{1}{p_1}\right) \left(1 - \frac{1}{p_2}\right) \dots \left(1 - \frac{1}{p_r}\right) = n \left(\frac{p_1-1}{p_1}\right) \left(\frac{p_2-1}{p_2}\right) \dots \left(\frac{p_r-1}{p_r}\right),$$

$$\varphi(n) = n \left(1 - \frac{1}{p_1}\right) \left(1 - \frac{1}{p_2}\right) \dots \left(1 - \frac{1}{p_r}\right) = n \left(\frac{p_1-1}{p_1}\right) \left(\frac{p_2-1}{p_2}\right) \dots \left(\frac{p_r-1}{p_r}\right),$$

where p_1, p_2, \dots, p_r **p_1, p_2, \dots, p_r** are distinct prime factors of n . Write an algorithm to read the positive integer n and then calculate and print $\varphi(n)$ using the above formula and once again, directly without using it.

6.9*. Write a main algorithm to read a positive number k and then determine and print the perfect numbers less than k using the function `perfect()` in **Exercise 6.30**.

7 Conditional loops

7.1 The while and do-while templates

In [Chapter 6](#), we observed that the automated loops perform three processes including giving the initial value to the variable of the loop, increasing the growth value in each repetition, and controlling the repetition of the loop automatically which in the algorithm (programming) an instruction (a statement) is used for this purpose. On the other hand, in the conditional loops, which we will study in the present chapter, the situation is somehow different. Especially, the repetition of the loop is controlled by a condition named the condition of loop repetition or simply the condition of the loop.

Depending on whether this condition is at the beginning or the end of the loop, the conditional loops are divided into two templates called the **while** or **do-while templates** or **loops**. In fact, the position of the condition is as the position of the while keyword in the name of the template. In this chapter, these two templates are thoroughly investigated. Normally, the following steps are used in creating the conditional loop templates regardless of their order.

The first step. Determining the initial values of the loop, if there are any. These initial values exist in most loops and the first repetition of the loop is created with their help. The best location for the initial values is before the start of the loop. In some loops, however, we distinguish these values after the second or even the third repetition.

The second step. Designing the range of the loop based on the processes performed in each repetition of the loop. Sometimes, especially when the initial values of the loop are not specified, writing the range of the loop from the second repetition onwards is feasible. In such circumstances, we first create the range of the loop and then look for initial values which can create the first repetition using this range. This step, like the automated loops, is of great importance.

The third step. Evaluating the condition for the repetition of the loop and its appropriate location. Clearly, we can place the condition of the loop either at the start or the end in most loops. However, in some loops we are bound to use only one of the start-condition (while) or end-condition (do-while) templates of the loop.

7.1. Example. Write an algorithm to read 20 integers and then determine and print the number of positive integers among these integers.

Solution. This is exactly the first example in [Chapter 6 \(Example 6.1\)](#) for which we wrote an algorithm ([Flowchart 6.1](#)) using a for loop. In this loop, the initial values, growth value, and controlling the repetition of the loop were automatically determined and a single instruction corresponded to all these three tasks.

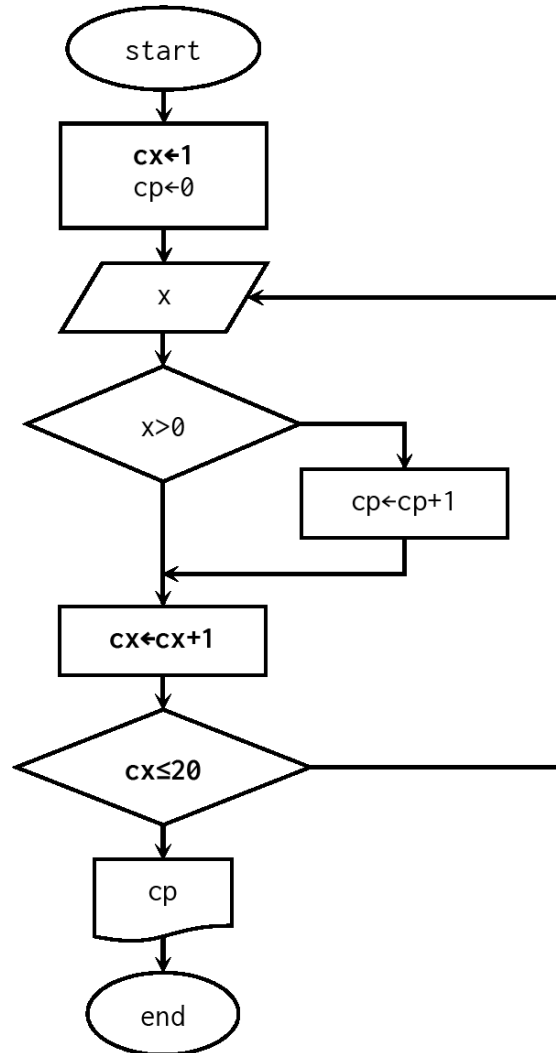


Fig. 7.1(a): Flowchart of [Example 6.1](#) using the do-while template.

In this chapter, we decided to write this algorithm using a conditional loop. Therefore, the three above-mentioned tasks should now be performed using three individual instructions.

1. Before the loop, where no integer has yet been read, take the initial value of cx as 0.
2. We already know the basic instructions of the range of the loop, namely, those which were implemented in each repetition. Often, the best place for increasing the growth value is after these basic instructions of the range of the

loop. Nevertheless, we can occasionally put the growth value increment before the basic instructions of the range or even insert it among such instructions.

3. Finally, the condition of the repetition is located at the end of the loop so that the loop repeats until 20 counts for the read numbers is completed.

The flowchart obtained based on the do-while template is illustrated in **Figure 7.1(a)**. In this flowchart, the above-mentioned instructions are demonstrated in bold font.

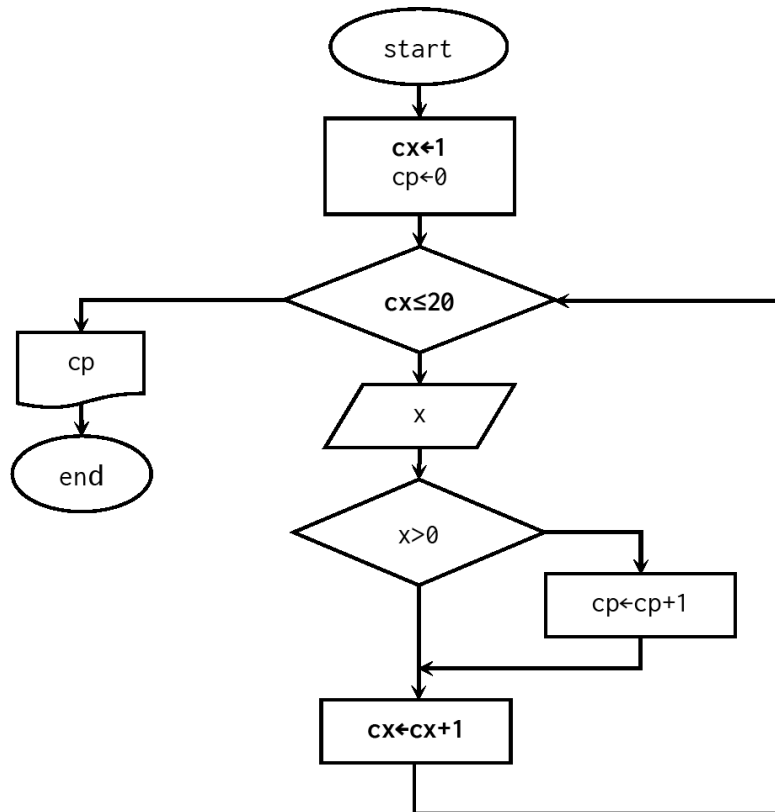


Fig. 7.1(b): Flowchart 7.1(a) using the while template.

Finally, the implementation table is arranged for the following 5 data, instead of 20.

-2, 0, 19, -23, 14 -2, 0, 19, -23, 14

Tab. 7.1: Implementation table of Flowchart 7.1(a) for 5 data instead of 20.

position	x	cx	cp	output
before the loop		1	0	
first repetition	-2	2		
second repetition	0	3		
third repetition	19	4	1	
forth repetition	-23	5		
fifth repetition	14	6	2	
after the loop				2

In **Flowchart 7.1(a)**, the condition is located at the end of the loop. **Flowchart 7.1(b)** is designed for the same algorithm in a way that the condition is at the start of the loop. we leave the reader to arrange the implementation table for **Flowchart 7.1(b)** with the same input data as above for better understanding.

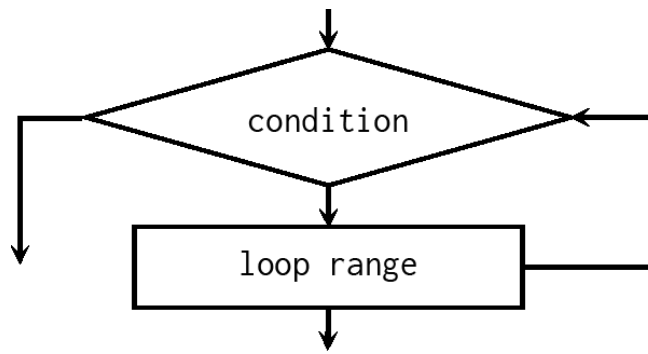


The other template can be created by changing the location related to the condition of the loop from the start of the loop to the end or vice versa using one of the conditional loop templates. Accordingly, several other changes need to be made. After replacing the template, it is strongly recommended to arrange an implementation table for the resulted template in order to be assured the correctness of the algorithm.

7.1.1. Exercise. Modify **Example 7.1** in such a way that the integers themselves along with the number of positive integers are printed. Apply this change in both **flowcharts 7.1(a)** and **7.1(b)**.

7.1.2. Exercise. Now, modify the algorithm so that it calculates and prints the number of positive and negative integers and zeroes together with appropriate output headings.

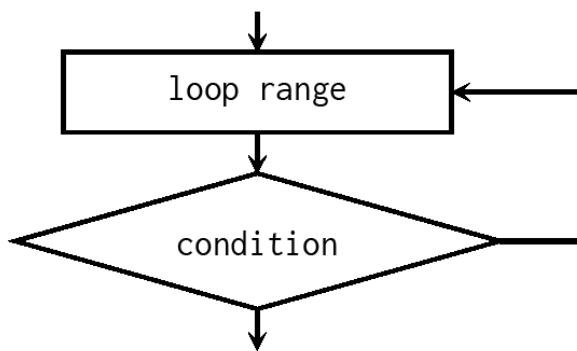
In general, the flowcharts of the while and do-while templates along with their identical codes in C++ and Java are summarized as follows.



The while template

C++ and Java codes:

```
while (condition) {
    loop range
}
```



The do-while template

C++ and Java codes:

```
do {
    loop range
} while (condition)
```

Implementation of the while template: Implement (repeat the implementation of) the loop range *while* the condition is true. Thus, exit the loop as soon as the condition is false.

Implementation of the do-while template: *Do* implement the loop range and repeat its implementation *while* the condition is true. Therefore, exit the loop as soon as the condition is false.

In fact, every logical condition has a negation having the opposite logical value. As a result, both while and do-while loops can be made in such a way that the T- and F-paths on these templates stay on the same directions as the above flowchart pattern. Of course, there are nearly similar flowcharts for these two conditional loop templates in all the programming languages.



The loop range should be grouped by {} if it contains more than one statement.

Programs P7_1_A translate **Flowchart 7.1(a)** into C++ and Java codes.

C++ codes:

```
// Program P7_1_A to determine the
// number of positive integers
// using the do-while statement
#include <iostream>
using namespace std;
int main() {
    int x, cp=0, cx=1;
    do {
        cout<<"Enter an integer: ";
        cin>>x;
        if (x>0)
            cp++;
        cx++;
    } while (cx<=20);
    cout<<"Number of positive integers: "<<cp;
    return 0;
}
```

Java codes:

```
// Program P7_1_A to determine the
// number of positive integers
// using the do-while statement
import java.util.Scanner;
class P7_1_A {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int x, cp=0, cx=1;
        do {
            System.out.print("Enter an "
                +"integer: ");
            x=read.nextInt();
            if (x>0)
                cp++;
            cx++;
        } while (cx<=20);
        System.out.print("Number of "
            +"positive integers: " + cp);
        read.close();
    }
}
```

Furthermore, Programs P7_1_B translate [Flowchart 7.1\(b\)](#) into C++ and Java codes.

C++ codes:

```
// Program P7_1_B to determine the
// number of positive integers
// using the while statement
#include <iostream>
using namespace std;
int main() {
```

Java codes:

```
// Program P7_1_B to determine the
// number of positive integers
// using the while statement
import java.util.Scanner;
class P7_1_B {
    public static void main(String[] args) {
```



```

int x, cp=0, cx=1;
while (cx<=20) {
    cout<<"Enter a number: ";
    cin>>x;
    if (x>0)
        cp++;
    cx++;
}
cout<<"Number of positive integers: "<<cp;
return 0;
}

```

```

int x, cp=0, cx=1;
Scanner read=new Scanner(System.in);
while (cx<=20) {
    System.out.print("Enter an "
                    +"integer: ");
    x=read.nextInt();
    if (x>0)
        cp++;
    cx++;
}
System.out.println("Number of "
                  +"positive integers: " + cp);
read.close();
}
}

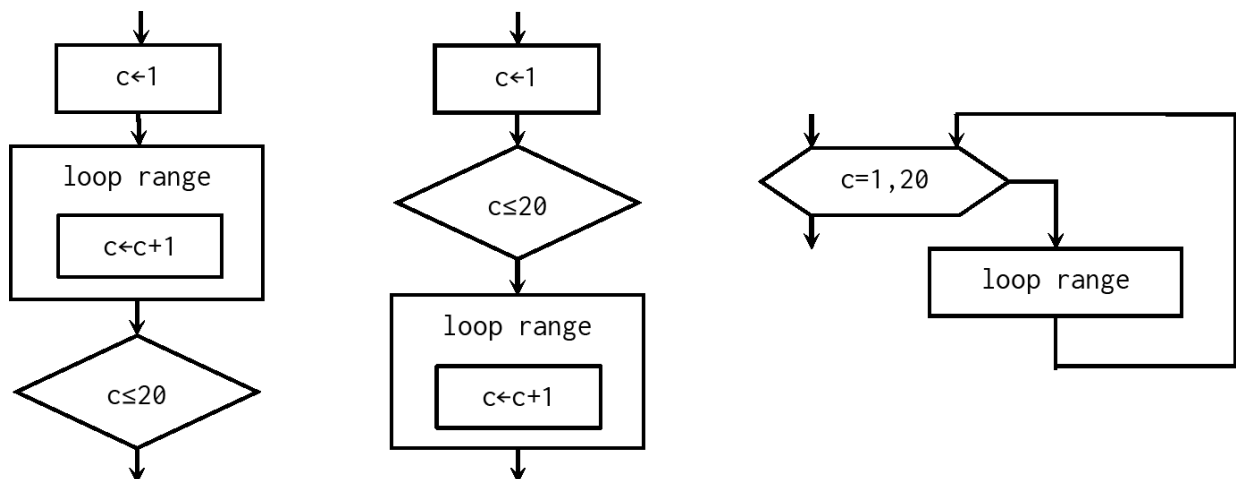
```

As shown, despite the position regarding the condition of the loop, there is another major difference between these two loops in their flowcharts.



The basic difference between the two conditional loop templates. In the while template, the range of the loop may never be implemented while the range of the loop is implemented at least once in the do-while template. This is exactly due to the position of the conditions of the loops.

Comparing [Flowcharts 7.1\(a\)](#) and [7.1\(b\)](#) with [Flowchart 6.1](#), it is observed that the following three flowcharts are equivalent from the implementation point of view.



The algorithm in [Example 7.1](#) is an instance indicating that writing the algorithms is possible using any loop template including the automated or either of the conditional

loop templates. In particular, the automated loop templates can be transferred to either of the conditional loops.



Due to the simplicity of the automated loops both in flowchart and programming codes, it is strongly recommended to use automated loops whenever possible, unless the problem requires employing the conditional loops.

As previously mentioned, before **Example 6.10** in **Chapter 6**, the `continue` and `break` statements may appear in the loops. A general application of these statements in the nested loops are represented in a prototype, accompanied by some notes, in which any loop may be one of the three loops of `for`, `while`, or `do-while`. Recall that the labelled `continue` and `break` statements are exclusively used in Java programming language.

Furthermore, the direction of the run control was demonstrated for both the `continue` and `break` statements in the `for` template. In the following parts, this direction is shown for the `while` and `do-while` templates.

The `continue` statement in C++ and Java

```
while (...) {  
    ...  
    continue;  
    ...  
}
```

The `break` statement in C++ and Java

```
while (...) {  
    ...  
    break;  
    ...  
}  
out of loop
```

The `continue` statement in C++ and Java

```
do {  
    ...  
    continue;  
    ...  
} while (...)
```

The `break` statement in C++ and Java

```
do {  
    ...  
    break;  
    ...  
} while (...)  
out of loop
```

7.2. Example. Several exam marks out of 20 will be read from the input unit. However, the number of marks is not known. Write an algorithm to read all the marks and then calculate and print their average.

Solution. Define the variables x , n , and sum for the read marks, number of read marks, and their repetitive sum, respectively. Then, assign the initial values of 0 to both n and sum before using any loop since no mark is yet read.

Since the read marks are from 0 to 20, we attempt to use a while template so that, while $0 \leq x \leq 20$, first, an x (a mark) is read. Then, the number n is increased one unit and the x is added to the sum , namely, the repetitive sum. Moreover, it suffices to enter either a negative number or a number greater than 20 in order to exit the loop. Then, the average, which is sum / n , is printed. The obtained flowchart is displayed in **Figure 7.2**. Programs P7_2 represent the codes of **Flowchart 7.2**.

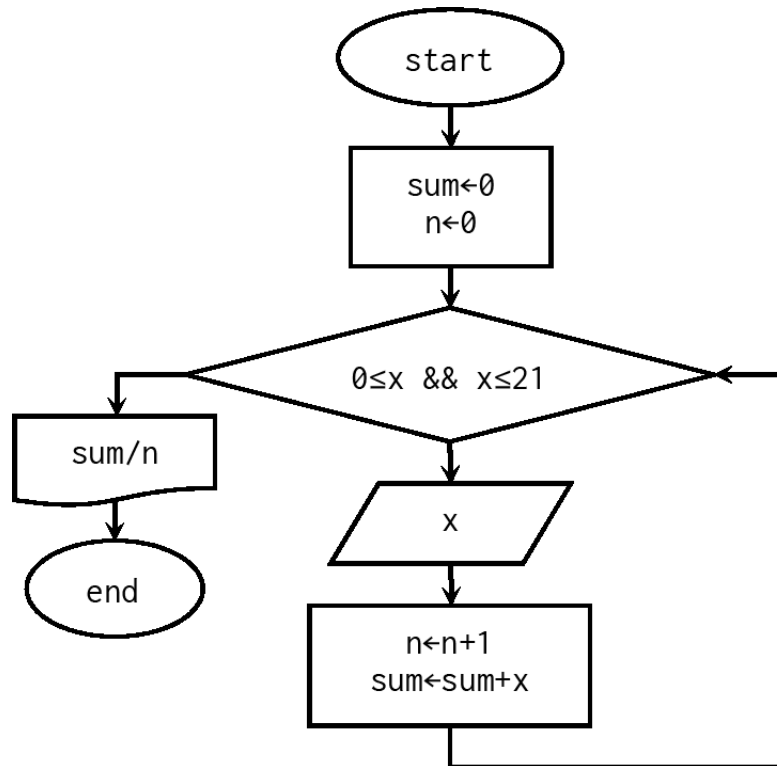


Fig. 7.2: Determining the average of unknown number of marks.

C++ codes:

```
// Program P7_2 to compute the mean
// value of an unknown number of exam
// marks using the do-while statement
#include <iostream>
using namespace std;
int main() {
    int n=0;
    float sum=0, x;
    while ((0<=x)&&(x<=20)) {
        cout<<"Enter an exam mark: ";
        cin>>x;
        n++;
        sum+=x;
    }
    cout<<"The mean value is: "<<sum/n;
    return 0;
}
```

Input/output:

Enter an exam mark: 12↵
Enter an exam mark: 14.5↵
Enter an exam mark: 19.25↵
Enter an exam mark: 8↵
Enter an exam mark: 21↵
The mean value is: 14.95

Java codes:

```
// Program P7_2 to compute the mean
// value of an unknown number of exam
// marks using the do-while statement
import java.util.Scanner;
class P7_2 {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int n=0;
        float sum=0, x;
        while ((0<=x)&&(x<=20)) {
            System.out.print("Enter an exam mark: ");
            x=read.nextFloat();
            n++;
            sum+=x;
        }
        System.out.print("The mean value is: "
            + sum/n);
        read.close();
    }
}
```

Input/output:

Enter an exam mark: 12↵
Enter an exam mark: 14.5↵
Enter an exam mark: 19.25↵
Enter an exam mark: 8↵
Enter an exam mark: 21↵
The mean value is: 14.95

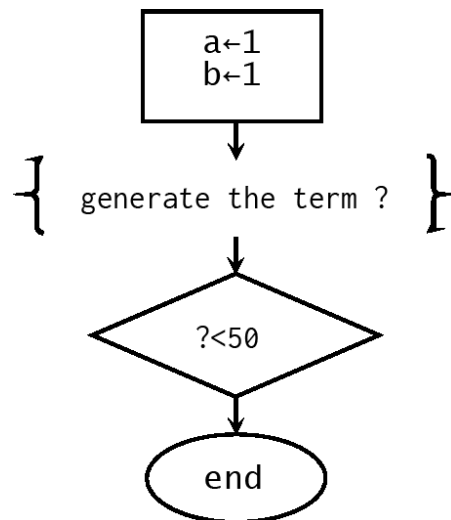


Fig. 7.3(a): Sketch of the Fibonacci sequence algorithm.

As seen, it is difficult to solve [Example 7.2](#) using the for template. However, we can easily use the for template if the upper bound of the number of marks is known.

7.2.1. Exercise. Assume that the number of marks is at most 40. Using the for template, write an algorithm for [Example 7.2](#) and then write the programs.

The following paragraph is another example in which it is impossible to employ the for template.

7.3. Example. The Fibonacci sequence is one of the famous sequences in which the first and second terms are 1 and from the third item onward, each item is the sum of the two previous terms. A few terms of this sequence are as follows.

1, 1, 3, 5, 8, 13, 21, ... **1, 1, 3, 5, 8, 13, 21, ...**

Write an algorithm to generate and print the terms of this sequence which are smaller than 50.

Solution. First, take two variables a and b with the initial value 1. These are the first two terms of the sequence. We intend to generate the terms smaller than 50. We do this in a do-while template while the term is less than 50. So far, we have [Flowchart 7.3\(a\)](#).

Take one of the variables a or b , say a , as the sequence generator. Then, add a to b and substitute the result for a . An implementation table indicates that if we continue this way, we are stuck after two repetitions. Additionally, the table represents that a and b should be swapped after adding up to overcome this problem ([Fig. 7.2\(b\)](#)).

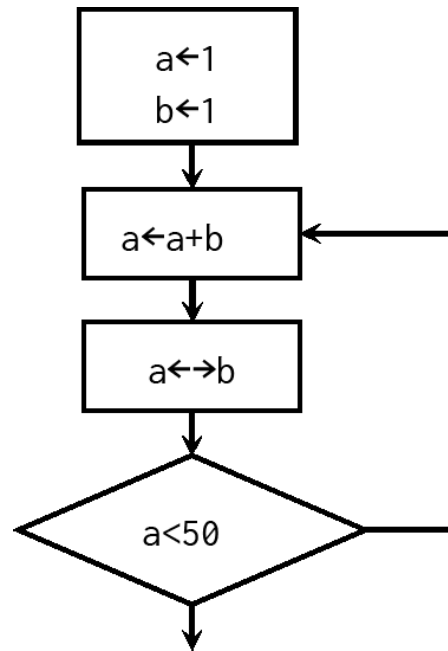


Fig. 7.3(b): Generating the terms of Fibonacci sequence without print.

Arrange the implementation table again (Tab. 7.3(a)). This time it generates the sequence. In this table, we generated the terms less than 7, instead of 50.

Tab. 7.3(a): Generating the terms of the Fibonacci sequence less than 7.

position	a(generator)	b
before the loop	1	1
first repetition	2	
	<u>1</u>	2
second repetition	3	
	<u>2</u>	3
third repetition	5	
	<u>3</u>	5
forth repetition	8	
	<u>5</u>	8

Concentrating on the underlined terms in the table, we can guess the location of the print template. It should be at the start of the loop. Now, the required flowchart is completed. Figure 7.3(c) depicts the completed flowchart. Programs P7_3_A represent the codes of the completed flowchart 7.3(c).

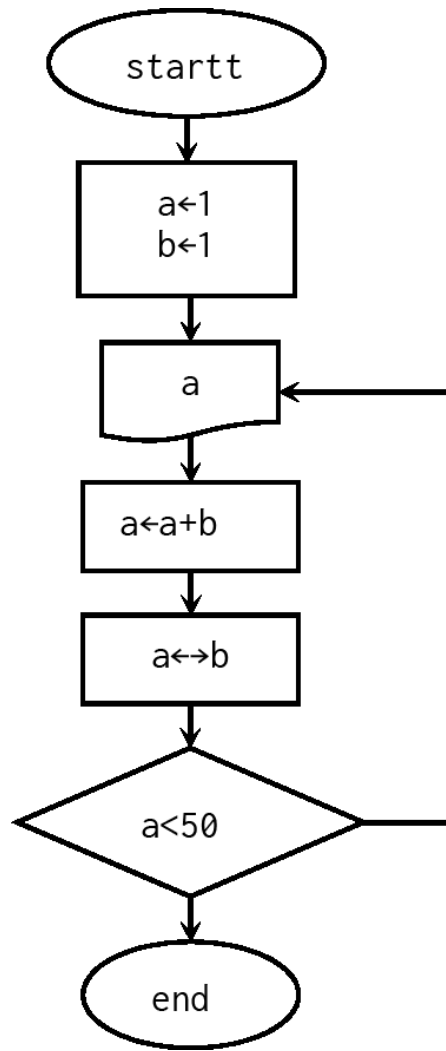


Fig. 7.3(c): Completed Fibonacci algorithm.

C++ codes:

```
// Program P7_3_A to write down the terms
// of Fibonacci sequence less than 50
#include <iostream>
using namespace std;
int main() {
    int a=1, b=1, h;
    cout<<"Fibonacci sequence"
    <<" less than 50:"<<endl;
    do {
        cout<<a<<endl;
        a+=b;
        h=a; a=b; b=h;
    } while (a<50);
    return 0;
}
```

Output:

Fibonacci sequence less than 50
1 1 2 3 5 8 13 21 34

Java codes:

```
// Program P7_3_A to write down the terms
// of Fibonacci sequence less than 50
class P7_3_A {
    public static void main(String[] args) {
        int a=1, b=1, h;
        System.out.println("Fibonacci sequence"
            + " less than 50:");
        do {
            System.out.print(a + " ");
            a+=b;
            h=a; a=b; b=h;
        } while (a<50);
    }
}
```

Output:

Fibonacci sequence less than 50
1 1 2 3 5 8 13 21 34

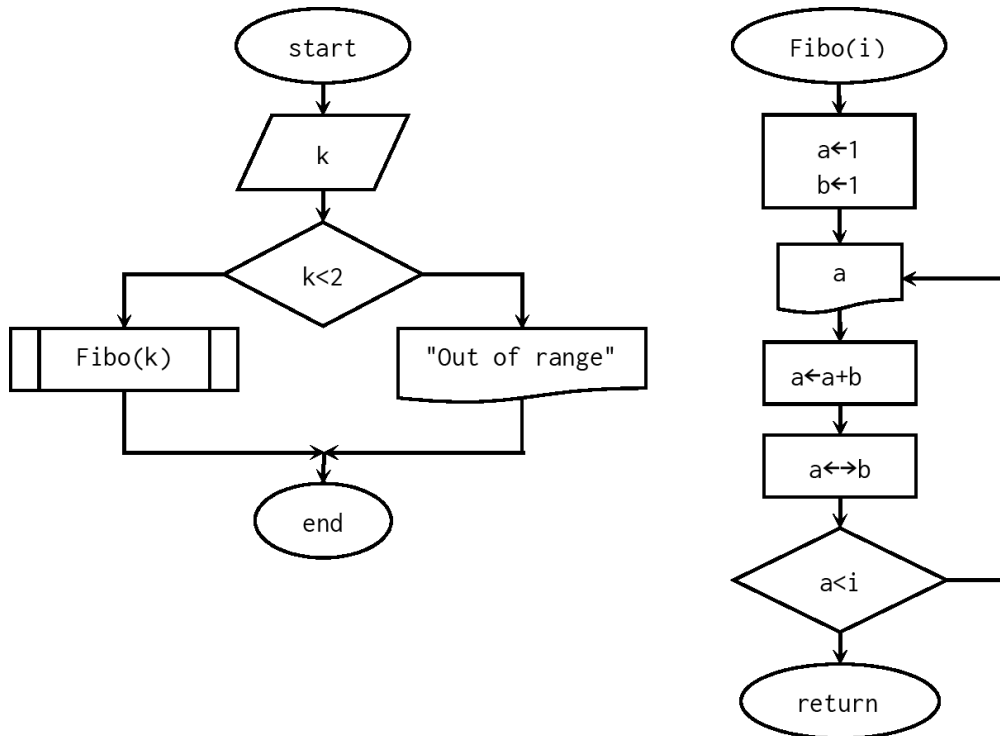


Fig. 7.3(d): Calling Fibonacci sub-algorithm in a main algorithm.

Next, we divide Algorithm 7.3(c) into a sub-algorithm and a main algorithm so that the sub-algorithm, named Fibo(), receives the positive integer *i* and prints all the terms of

the Fibonacci sequence smaller than i . Moreover, the main algorithm reads an integer k from the input and prints a message if $k < 2$. Otherwise, this algorithm prints all the terms smaller than k using the sub-algorithm `Fibo()`. The result is represented in [Flowchart 7.3\(d\)](#). Programs `P7_3_B` indicate the translation of these two units into C++ and Java codes.

C++ codes:

```
// Program P7_3_B to write down the
// terms of Fibonacci sequence less
// than an integer using a subprogram
#include <iostream>
using namespace std;
void Fibo(int);
int main() {
    int k;
    cout<<"Enter an integer: ";
    cin>>k;
    if (k<2)
        cout<<"Out of range";
    else
        Fibo(k);
    return 0;
}
//*****

void Fibo(int i) {
    cout<<"The Fibonacci sequence less "
        <<"than "<<i<<":\n";

    int a=1, b=1, h;
    do {
        cout<<a<<" ";
        a+=b;
        h=a; a=b; b=h;
    } while (a<i);
    return;
}
```

Input/output:

```
Enter an integer: 50↵
The Fibonacci sequence less than 50:
1 1 2 3 5 8 13 21 34
```

Java codes:

```
// Program P7_3_B to write down the
// terms of Fibonacci sequence less
// than an integer using a subprogram
import java.util.Scanner;
class P7_3_B {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        System.out.print("Enter an integer: ");
        int k=read.nextInt();
        if (k<2)
            System.out.println("Out of range");
        else
            Fibo(k);
        read.close();
    }
    //*****
    static void Fibo(int i) {
        System.out.println("The Fibonacci "
            +"sequence less than " + i + ":");
        int a=1, b=1, h;
        do {
            System.out.print(a + " ");
            a+=b;
            h=a; a=b; b=h;
        } while (a<i);
    }
}
```

Input/output:

```
Enter an integer: 50↵
The Fibonacci sequence less than 50:
1 1 2 3 5 8 13 21 34
```

Writing an algorithm to generate the n -th term of the Fibonacci sequence by recursive method would be easier for any positive integer n . If we denote the n -th term by f_n ,

f_n , then we have the following recursive relations

$$f_n = f_{n-1} + f_{n-2}, \quad f_1 = 1, \quad f_2 = 1,$$

$$f_n = f_{n-1} + f_{n-2}, \quad f_1 = 1, \quad f_2 = 1,$$

or, equivalently, as a recursive function,

$$f(n) = \begin{cases} 1, & \text{if } n = 1 \text{ or } 2 \\ f(n-1) + f(n-2), & \text{otherwise.} \end{cases}$$

$$f(n) = \begin{cases} 1, & \text{if } n = 1 \text{ or } 2 \\ f(n-1) + f(n-2), & \text{otherwise.} \end{cases}$$

The generating function of this sequence, named Fib, is illustrated in **Flowchart 7.3(e)** using the above-mentioned method. This function receives a positive integer n and then calculates and returns the n -th term of the Fibonacci sequence. Additionally, the main algorithm of this flowchart reads an integer k from the input and prints a message if k is not positive. Otherwise, it determines and prints the k -th term of the sequence using the function Fib(). Programs P7_3_C are considered the translation of these two units into C++ and Java codes.

C++ code:

```
// Program P7_3_C to generate the
// k-th terms of Fibonacci sequence
// by recursive method
#include <iostream>
using namespace std;
int Fib(int);
int main() {
    int k;
    cout<<"Enter an integer: ";
    cin>>k;
    if (k<=0)
```

Java codes:

```
// Program P7_3_C to generate the
// k-th terms of Fibonacci sequence
// by recursive method
import java.util.Scanner;
class P7_3_C {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int k;
        System.out.print("Enter an integer: ");
        k=read.nextInt();
        if (k<=0)
```

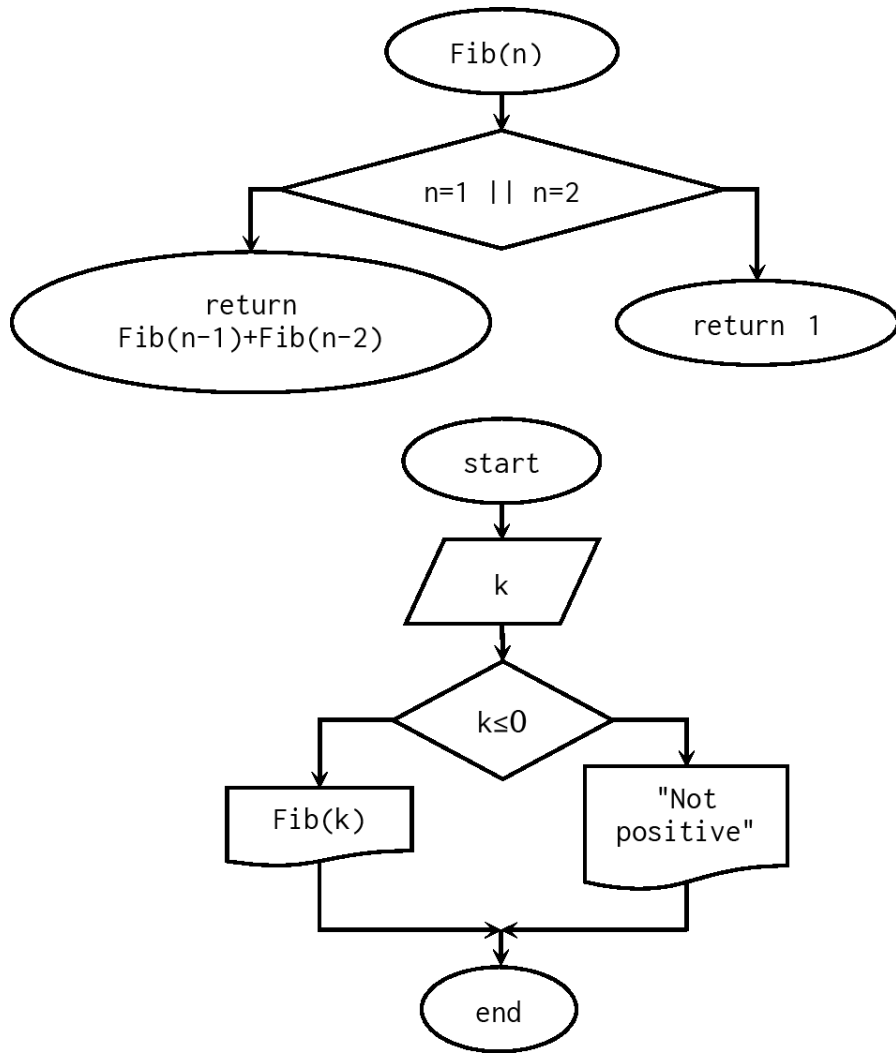


Fig. 7.3(e): Generator the n -th term of Fibonacci sequence by recursive method.

```

    cout<<"Not positive";
else
    cout<<k<<"-th term: "<<Fib(k);
return 0;
}
//*****
int Fib(int n) {
    if (n==1 || n==2)
        return 1;
    else
        return Fib(n-1)+Fib(n-2);
}

```

Input/output:

Enter an integer: 9↵
 9-th term: 34

```

    System.out.println("Not positive");
else
    System.out.print(k + "-th term: " + Fib(k));
read.close();
}
//*****
static int Fib(int n) {
    if (n==1 || n==2)
        return 1;
    else
        return Fib(n-1)+Fib(n-2);
}
}

```

Input/output:

Enter an integer: 9↵
 9-th term: 34

7.4. Example. Write an algorithm to read an integer and print its reverse. For example, -491 is the reverse of -194.

Solution. The brain reaction is first analyzed in order to get an insight on how to write the algorithm for a positive integer. Then, the algorithm is extended to any integer. The details regarding the process of working with number 234 are described in [Table 7.4\(a\)](#).

Tab. 7.4(a): The brain reaction and issued commands for reversing the number 234.

status	the number being processed	last digit	new number
the number	234		
first step	23	4	4
second step	2	3	43
third step	0	2	432

The brain reaction and the issued commands in each step are analyzed:

1. Take the last digit from 234; what is left is 23; write the removed digit (as the new number): 4.
2. Take the last digit from 23; what is left is 2; put the removed number in front of the recent 4: 43.
3. Take the last digit from 2 (2 itself); what is left is 0 (nothing); put the removed number in front of the recent 43: 432.

Now the mentioned analysis is steered to reach an algorithm. To do this, we need to approach some of the above actions to equivalent actions from the algorithm viewpoint:

- Last digit: the remainder of the division by 10;
- What is left: the quotient of the division by 10;
- Putting the digit r in front of the recent number x : substituting the number $x * 10 + r$ for x .

To continue, define some variables:

n : the number being processed;

$newn$: the new number obtained by putting a digit in front of a certain number;

r : the remainder of the division by 10.

We are now going to rewrite the three commands issued from the brain reaction in the literature of the algorithm as follows.

1. Assign $n \% 10$ to r ; substitute $n / 10$ for n ; assign r to $newn$.
2. Assign $n \% 10$ to r ; substitute $n / 10$ for n ; substitute $newn * 10 + r$ for $newn$.
3. Assign $n \% 10$ to r ; substitute $n / 10$ for n ; substitute $newn * 10 + r$ for $newn$.

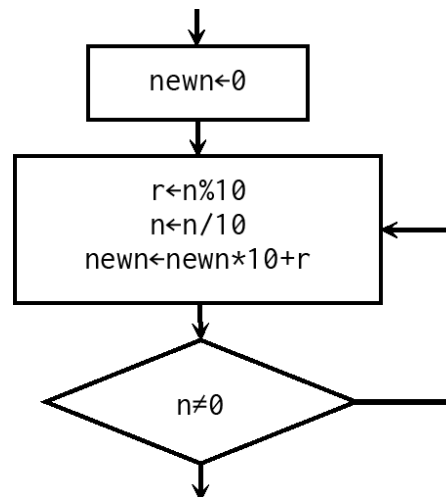
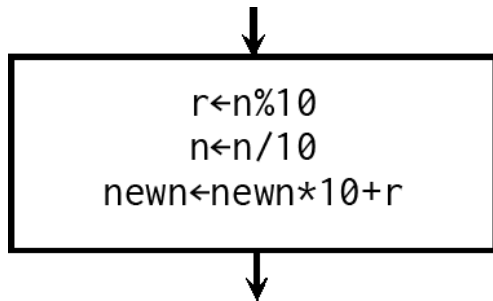


Fig. 7.4(a): The do-while template constructed so far for reversing a positive integer.

As seen, there is a loop which is repeated three times and the following instructions are implemented in each repetition of the loop (Note that assigning zero to the initial value of $newn$, the instruction “assign r to $newn$ ” becomes equivalent to “substitute $newn * 10 + r$ for $newn$ ”).



How long should we continue this loop? Until there is a (positive) number, that is, until $n \neq 0$. As shown in [Flowchart 7.4\(a\)](#), so far, we have constructed a do-while template with the initial value of 0 for *newn*.

[Table 7.4\(a\)](#) is only displayed for explaining the brain reaction and the issued commands. [Table 7.4\(b\)](#) is the real implementation table for 234.

Tab. 7.4(b): Implementation table of [Flowchart 7.4\(a\)](#) for $n = 234$.

status	n	r	newn
before the loop	234		0
first repetition	23	4	4
second repetition	2	3	43
third repetition	0	2	432

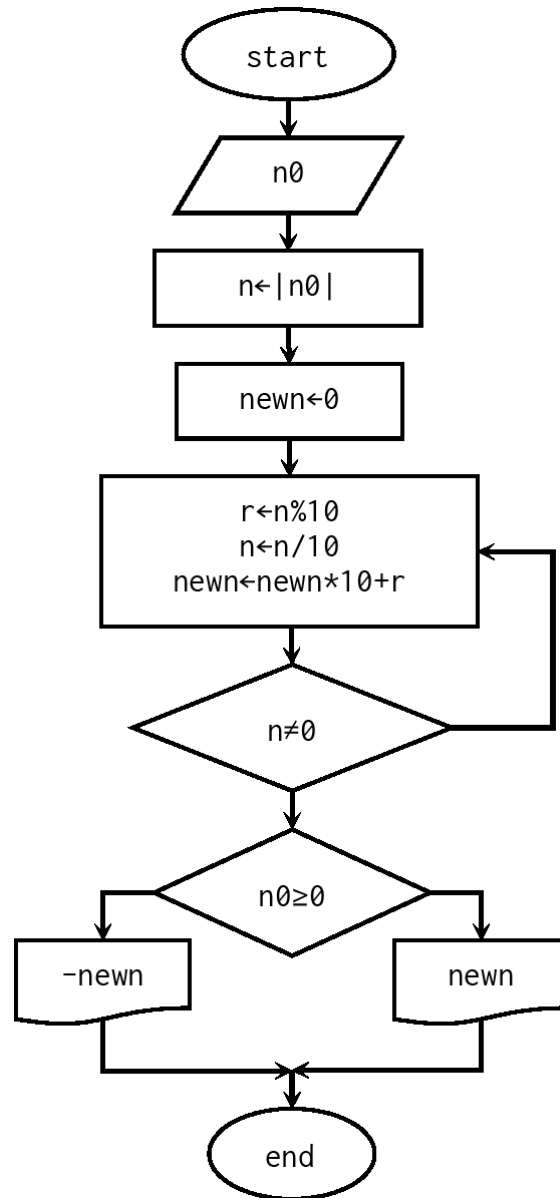


Fig. 7.4(b): Completed algorithm to reverse an integer.

Flowchart 7.4(a) is the most essential part on which the algorithm is based. This flowchart displays a positive integer. Assume that we do not know whether or not the given integer is positive. Take the original number as n_0 and substitute its absolute value for n . Continue the process of Flowchart 7.4(a). Upon the exit of the loop, print $newn$ if the original number is positive; otherwise $-newn$ is printed. Flowchart 7.4(b) is the completed algorithm.

7.4.1. Exercise. Transform the do-while template in Flowchart 7.4(b) into the while template and arrange the implementation table for both positive and negative integers in duplicate.

Now the process of **Flowchart 7.4(a)** is considered as the main body of a function sub-algorithm, named `rev()`. In fact, the function `rev()` is taken into account for calculating and returning the reverse of the positive integer n . Finally, the remaining duties are assigned to a main algorithm. **Figure 7.4(c)** includes these two units.

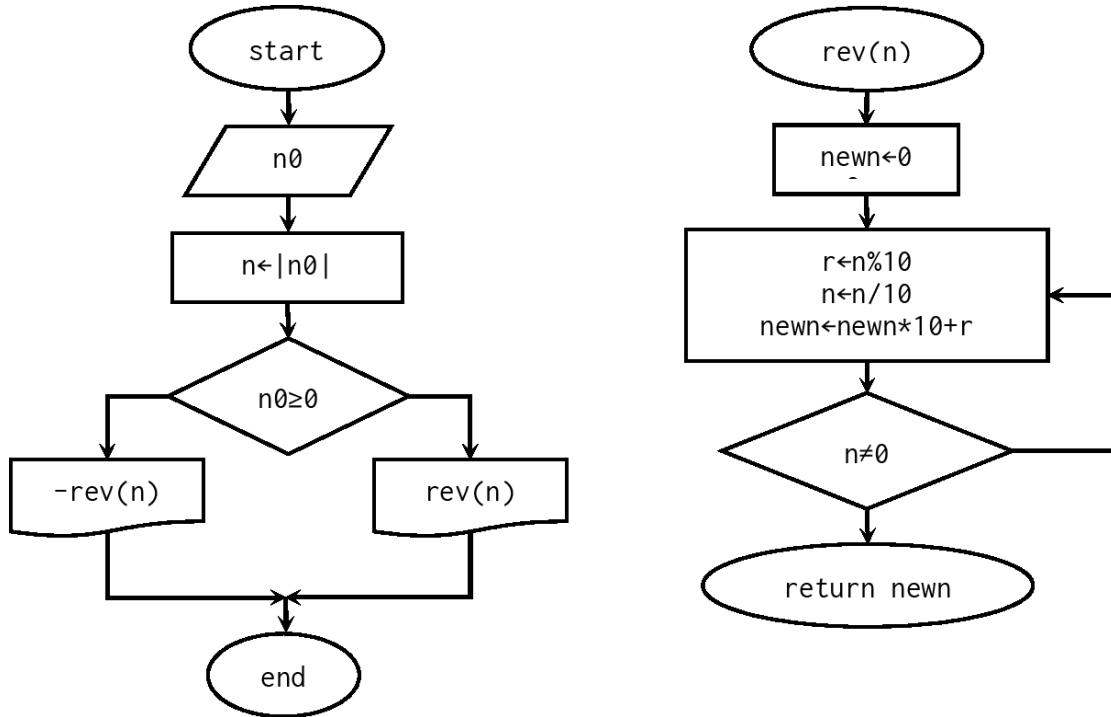


Fig. 7.4(c): Reversing an integer using the `rev()` function.

Programs P7_4 translate the two units of Flowchart 7.4(d) into C++ and Java codes.

C++ codes:

```
// Program P7_4 to reverse an integer
// using the rev() function
#include <iostream>
#include <math.h>
using namespace std;
int rev(int);
int main() {
    int n0, n;
    cout<<"Enter an integer: ";
    cin>>n0;
    n=abs(n0);
    if (n0>=0)
        cout<<"The reversed form is: "
            <<rev(n);
    else
        cout<<"The reversed form is: "
            <<-rev(n);
    return 0;
}
//*****
int rev(int n) {
    int newn=0, r;
    do {
        r=n%10;
        n=n/10;
        newn=newn*10+r;
    } while (n!=0);
}
```

Java codes:

```
// Program P7_4 to reverse an integer
// using the reve() method
import java.util.Scanner;
class P7_4 {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int n0, n;
        System.out.print("Enter an integer: ");
        n0=read.nextInt();
        n=Math.abs(n0);
        if (n0>=0)
            System.out.print("The reversed "
                + "form is: " + rev(n));
        else
            System.out.print("The reversed "
                + "form is: " + -rev(n));
        read.close();
    }
//*****
    static int rev(int n) {
        int newn=0, r;
        do {
            r=n%10;
            n=n/10;
            newn=newn*10+r;
        } while (n!=0);
        return newn;
    }
}
```

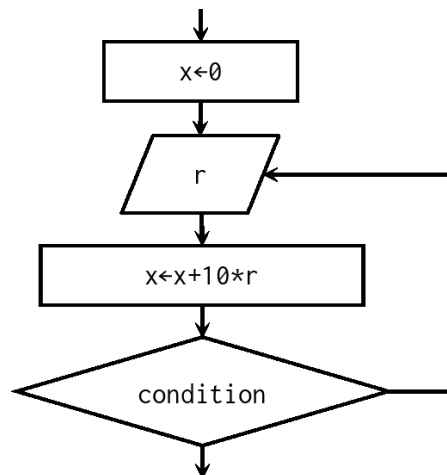


Fig. 7.5(a): Putting digits in front of each other from left to right.

```

return newn;
}
}

```

Input/output:

Enter an integer: -1230456↵
The reversed form is: -6540321

Input/output:

Enter an integer: -1230456↵
The reversed form is: -6540321

7.5. Example. (a) We are given repeatedly a digit r and requested to put it in front of the previous one from left to right while a condition holds. Write a part of the algorithm for this purpose;

(b) Repeat part (a) putting the digits behind each other from right to left;

(c) Repeat part (a) after a decimal point.

Solution. Denote by x the number obtained by putting the digit r in each part. We applied part (a) in the algorithm of [Example 7.4](#). [Figure 7.5\(a\)](#) recalls this part. In fact, we multiply x by 10 before putting the digit r in front of x in order to open up a place for r . Then, r is inserted in its place by adding $x * 0$ up to r .

For part (b) we use a positioner variable named p for ones, tens, hundreds, and the like to arrange the digits from right to left behind each other. This time, we multiply p by r before putting the digit r behind x to open up the right position of r . Then, r is inserted in its position by adding the $p * r$ up to x . Finally, the positioner is multiplied by 10 to create a new position for the next repetition. The obtained part is illustrated in [Flowchart 7.5\(b\)](#).

The arrangement of digits for part (c) is similar to part (a), namely, from left to right. However, the real processing depends on part (b) with the difference that, this time, the positioner variable p provides positions for the one tenth, one hundredth, and the like. [Flowchart 7.5\(c\)](#) illustrates the algorithm for this part.

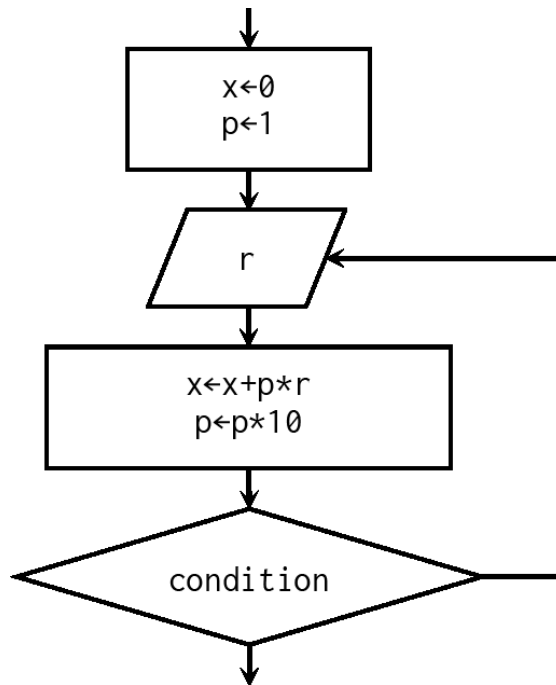


Fig. 7.5(b): Putting digits behind each other from right to left.

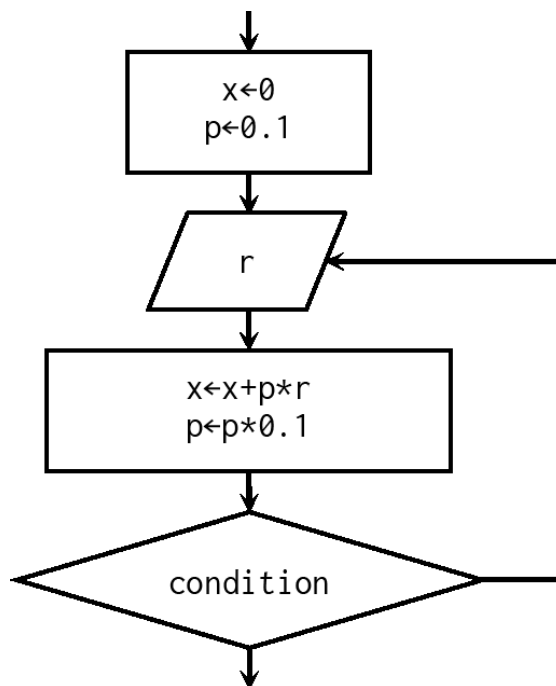


Fig. 7.5(c): Putting digits in front of each other from left to right, after a decimal point.

7.2 More applications of the conditional loops

In this section, converting positive real numbers to the base-2 numeral system, Euclidean algorithm (to find the greatest common divisor), the generalized Euclidean algorithm, and primary decomposition of positive integers are examined.

7.6. Example. Write a function (method) to receive a positive real number, convert it to the base-2 numeral system, and then return the result.

Solution. We start with the positive integer n . Theoretically, to convert n into the base-2 numeral system one must divide it continuously by 2 and write the quotient and remainder. This is repeated until the zero quotient appears. Now, the required number is obtained if the remainders are arranged behind each other from right to left. Applying the mentioned process for $n = 11$ is summarized in [Table 7.6\(a\)](#).

Tab. 7.6(a): Converting 11 to the base-2 numeral system.

status	quotient by 2	remainder by 2
first step	5	1
second step	2	1
third step	1	0
fourth step	0	1

The number 1011 is the required number. We analyze the problem further. As in [Example 7.4](#), the brain reaction and issued commands are analyzed in each step.

1. Divide 11 by 2; what is left (by removing the remainder) is 5; write the remainder (as the new number): 1;
2. Divide 5 by 2; what is left (by removing the remainder) is 2; put the remainder behind the recent 1: 11;
3. Divide 2 by 2; what is left (by removing the remainder) is 1; put the remainder behind the recent 11: 011 = 11;
4. Divide 1 by 2; what is left (by removing the remainder) is 0 (nothing); put the remainder behind the recent 011: 1011.

Further, the details of each step are provided. First, to approach some of the actions to the equivalent actions from the algorithm viewpoint:

- What is left: the quotient of the division by 2;
- Putting the digit r behind the recent number x : [Flowchart 7.5\(b\)](#).

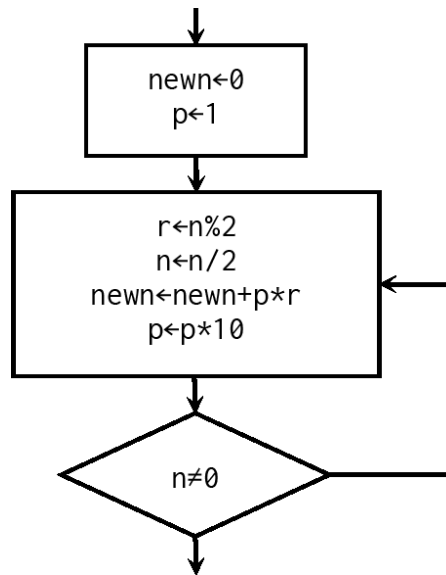


Fig. 7.6(a): Merging Flowcharts 7.4(a) and 7.5(b).

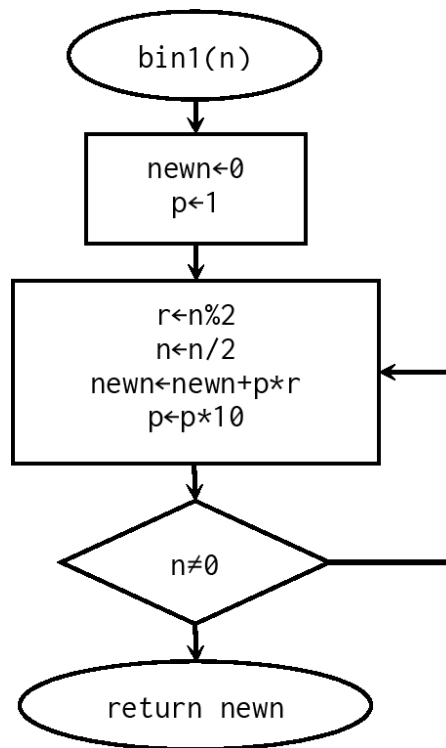


Fig. 7.6(b): A function to transfer an integer to the base-2 numeral system.

Define the following variables:

n : the number being processed;

$newn$: the new number obtained by putting a digit behind a certain number;

r : the remainder of the division by 2.

Now, inspired by [Flowchart 7.4\(a\)](#) and considering [Flowchart 7.5\(b\)](#) for putting a digit behind each one from right to left, [Flowchart 7.6\(a\)](#) is the merge of these two flowcharts.

The required function (method) for integers is represented by `bin1`. As visualized in [Flowchart 7.6\(b\)](#), this function receives the integer n , converts it to the base-2 numeral system and then returns the result with the carrier `newn`.

We arrange the implementation table for number 11 in order to test the correctness of the function `bin1()`.

Tab. 7.6(b): Implementation table for the function `bin1()` in [Flowchart 7.6\(b\)](#).

status	n	r	p	newn	return
before the loop	11		1	0	
first repetition	5	1	10	1	
second repetition	2	1	100	11	
third repetition	1	0	1000	011	
fourth repetition	0	1	10000	1011	1011

7.6.1. Exercise. Furthermore, the function `bin1()` can be designed using the logic of [Flowchart 7.4\(a\)](#) with an appeal to the function `rev()`: (a) Modify [Flowchart 7.4\(a\)](#) by changing the division 10 to 2 in the remainder r and quotient n . Then, the returning number from the function will be the reversed binary representation, (b) An appeal to the function `rev()`, now, returns the binary representation. Write the completed algorithm and test it for number 11.

Now we continue with a decimal number t . Mathematically, to convert a decimal number t to the base-2 numeral system, one should multiply it continuously to 2 and each time keep the integer part of the resulted number. This process is continued until the number of the requested decimals is acquired. Now, the desired number is obtained if we arrange the integers obtained in the above process from left to right after the decimal point. The mentioned process is applied for $t = 0.863$ and the result is obtained up to four decimals precision in [Table 7.6\(c\)](#).

Tab. 7.6(c): Converting the number 0.863 to the base-2 numeral system.

status	multiply the number by 2	integer part of the resulted number
first step	1.726	1
second step	1.452	1
third step	0.904	0

The result is 0.1101. As the previous part, we analyze the brain reaction and issued commands in each step.

1. Multiply 0.863 by 2; what is left is 0.726; take the integer part of the resulted number and put it after the decimal point (as a new number): 0.1;
2. Multiply 0.726 by 2; what is left is 0.452; take the integer part of the resulted number and put it in front of the recent decimal number: 0.11;
3. Multiply 0.452 by 2; what is left is 0.902; take the integer part of the resulted number and put it in front of the recent decimal number: 0.110;
4. Multiply 0.902 by 2; what is left is 0.808; take the integer part of the resulted number and put it in front of the recent decimal number: 0.1101.

Like the previous part, we explain the details by stating the equivalent parts of some actions from the algorithm viewpoint. The details regarding each step are provided. First, to approach several actions to the equivalent actions:

- What is left: the decimal part of the number after multiplying by 2;
- Putting the digit r (the integer part of the number after multiplying by 2) in front of the recent decimal number x : [Flowchart 7.5\(c\)](#).

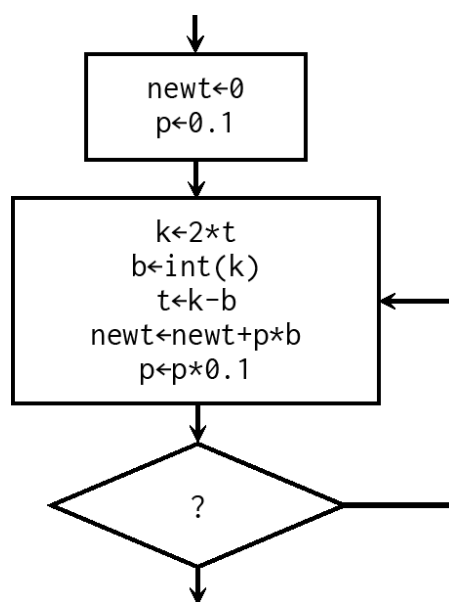


Fig. 7.6(c): The tasks performed in each repetition regarding the discussions so far.

Define the following variables:

t : the number being processed;

$newt$: the new number obtained by putting a digit in front of the recent decimal number;

b : the integer part of the number after multiplying by 2.

Now, considering the above discussion, the do-while loop can be written as depicted in [Flowchart 7.6\(c\)](#).

The loop in [Flowchart 7.6\(c\)](#) terminates only if the decimal digits of the number being processed ends; like 0.125 whose decimal digits terminates after three times multiplying the number by 2. If this does not occur, the loop is terminated after a specific decimal precision determined by the user. To do this, consider a counter variable i with the initial value of 0 and increase it in each repetition. Now, the loop repeats while $i < m$. The function `bin2()` in [Flowchart 7.6\(d\)](#) receives a decimal number t and an integer m . Finally, it converts t to the base-2 decimal system and returns it with m decimal digits. The result of the implementation of [Flowchart 7.6\(d\)](#) for $t = 0.863$ and $m = 4$ is demonstrated in [Table 7.6\(d\)](#).

Tab. 2: implementation table for [Flowchart 7.6\(d\)](#) for $t = 0.863$ and $m = 4$.

status	i	t	k	b	newt	p	return
before the loop	0	0.863			0	0.1	
first repetition	1	0.726	1.726	1	0.1	0.01	
second repetition	2	0.452	1.452	1	0.11	0.001	
third repetition	3	0.904	0.904	0	0.110	0.0001	
fourth repetition	4	0.808	1.808	1	0.1101	0.00001	0.1101

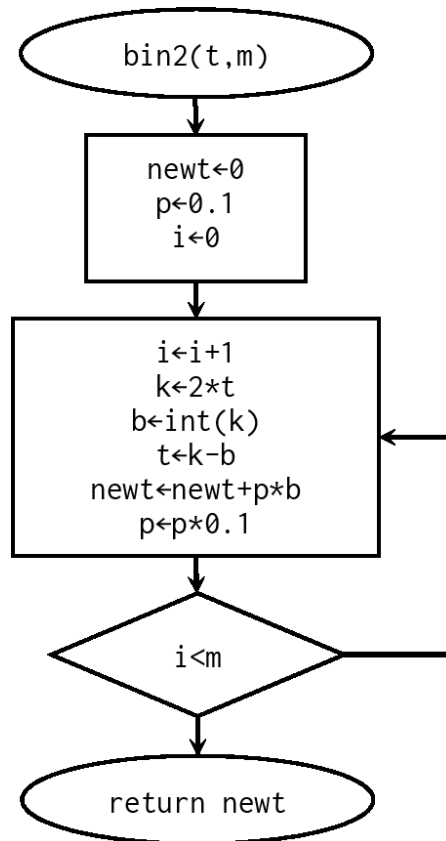


Fig. 7.6(d): A function to transfer a decimal number to the base-2 numeral system.

Now, we write a main algorithm which reads a positive real number u and a positive integer m . Then, it prints the representation of u in the base-2 decimal system with the precision of m decimal digits using the two functions `bin1()` and `bin2`. To do this, the integer and decimal parts of u are first separated and converted to the base-2 numeral system. Then, the results are added and printed. **Flowchart 7.6(e)** and Programs P7_6 are the outcomes of this process (note the formatted prints in the programs).

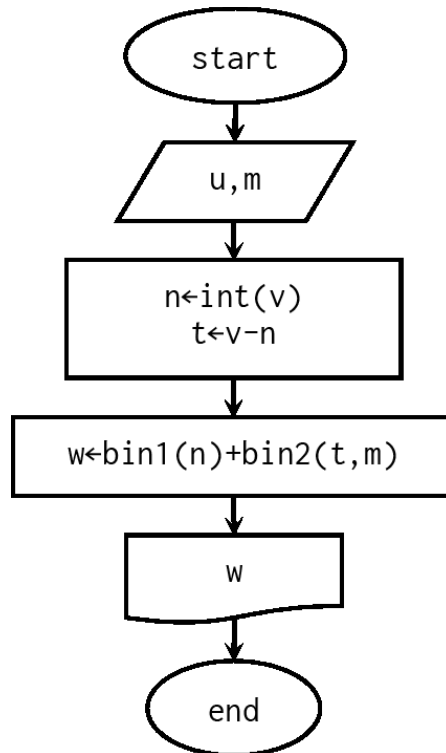


Fig. 7.6(e): Representing a real number in the base-2 numeral system with m precision digits.

C++ codes:

```

// Program P7_6 to convert the positive
// number u to the base-2 numeral system
// with the precision of m decimal digits
#include <iostream>
#include <math.h>
#include<iomanip>
using namespace std;
int bin1(int);
double bin2(float, int);
int main() {
    int n, m;
    double u, w, t;
    cout<<"Enter u and m: ";
    cin>>u>>m;
    n=int(u);
    t=u-n;
    w=bin1(n)+bin2(t, m);
    cout<<"The base-2 representation of "<<u
        <<"\nwith "<<m<<" decimal digits: "
        <<setprecision(10)<<w;
    return 0;
}
//*****
int bin1(int n) {

```

Java codes:

```

// Program P7_6 to convert the positive
// number u to the base-2 numeral system
// with the precision of m decimal digits
import java.util.Scanner;
class P7_6 {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int n, m;
        double u, w, t;
        System.out.print("Enter u and m: ");
        u=read.nextFloat();
        m=read.nextInt();
        n=(int)u;
        t=u-n;
        w=bin1(n)+bin2(t, m);
        System.out.printf(
            "The base-2 representation of %-20.4f"
            + "\nwith %1d"
            + " decimal digits: %-20.5f",
            u, m, w);
        read.close();
    }
}
//*****
static int bin1(int n) {

```

```

int r, p=1, newn=0;
do {
    r=n%2; n=n/2;
    newn=newn+p*r;
    p=p*10;
} while (n!=0);
return newn;
}

```

```

//*****
double bin2(float t, int m) {
    double k, newt=0, p=0.1;
    int i=0, b;
    do {
        i=i+1;
        k=2*t;
        b=int(k);
        t=k-b;
        newt=newt+p*b;
        p=p*0.1;
    } while (i<m);
    return newt;
}

```

Input/output:

Enter u and m: 27.0939 5↵
 The base-2 representation of 27.0939
 with 5 decimal digits: 11011.00011

```

int r, p=1, newn=0;
do {
    r=n%2; n=n/2;
    newn=newn+p*r;
    p=p*10;
} while (n!=0);
return newn;
}

```

```

//*****
static double bin2(double t, int m) {
    double k, newt=0, p=0.1;
    int i=-0, b;
    do {
        i=i+1;
        k=2*t;
        b=(int)k ;
        t=k-b;
        newt=newt+p*b;
        p=p*0.1;
    } while (i<m);
    return newt;
}

```

Input/output:

Enter u and m: 27.0939 5↵
 The base-2 representation of 27.0939
 with 5 decimal digits: 11011.00011

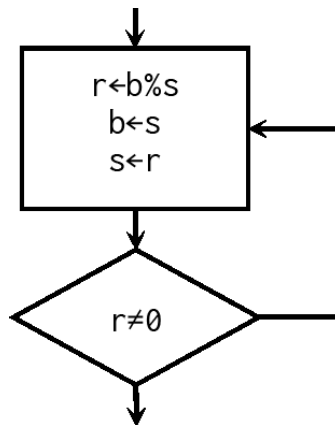


Fig. 7.7(a): Instructions implemented in each repetition of the Euclidean algorithm.

7.6.2. Exercise. The main Algorithm 7.6(e) converts the positive real numbers to the base-2 numeral system. Modify this algorithm in such a way to work for any real number and write the programs.

In the previous examples, the loops had initial values for the variables. The variables in the conditional loop template of the following example do not need any initial values.

7.7) The Euclidean algorithm to compute the greatest common divisor (gcd). Write an algorithm to read the two positive integers b and s and then determine and print their greatest common divisor using the Euclidean algorithm.

Solution. To get a general idea, we apply the Euclidean algorithm to compute the gcd of two numbers 87 and 24. Define the larger and smaller numbers as the dividend and divisor and denote them by b and s , respectively. Divide b by s and record the remainder, which is denoted by r . Next, substitute the divisor for the dividend and the remainder for the divisor. Repeat this process while the remainder is non-zero. The last non-zero remainder will be the gcd. [Flowchart 7.7\(a\)](#) and [Table 7.7\(a\)](#) summarizes the above discussion.

Tab. 7.7(a): The Euclidean algorithm applied for 87 and 24.

status	b (dividend)	s (divisor)	r (remainder)
first step	87	24	15
second step	24	15	9
third step	15	9	6
fourth step	9	6	3
fifth step	6	3	0

Considering the above-mentioned argument, the following instructions are implemented in each repetition of a do-while template while the remainder is non-zero.

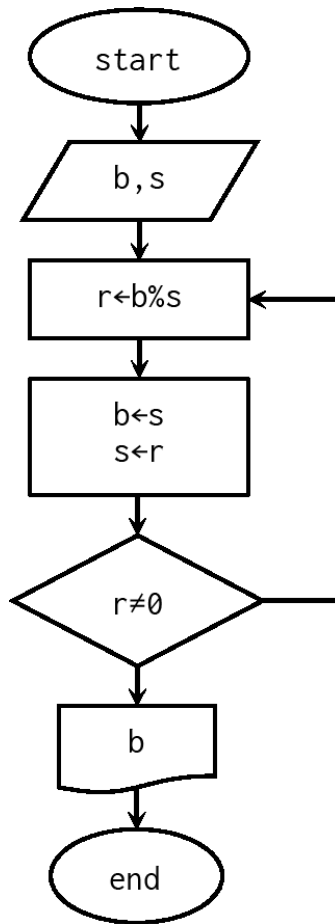


Fig. 7.7(b): Calculating the gcd using a do-while template.



Be careful not to misplace the two instructions $b \leftarrow s$ and $s \leftarrow r$; otherwise the algorithm will be completely wrong. Verify this fact by arranging an implementation table!

After exiting the loop, which variable should we print as the gcd? We look for the answer in the implementation [Table 7.7\(b\)](#).

Tab. 7.7(b): Implementation table for [Flowchart 7.7\(a\)](#).

status	b	s	r
before the loop	87	24	
first repetition	24	15	15
second repetition	15	9	9
third repetition	9	6	6
fourth repetition	6	3	3
fifth repetition	3	0	0

As shown in [Table 7.7](#), the answer to the above question is *b*. [Flowchart 7.7\(b\)](#) displays the resulted algorithm.

It is noteworthy that if we start with $b = 24$ and $s = 87$ in [Table 7.7\(b\)](#), the same result is obtained. Examine it! Thus, the order of reading small or large numbers is not important. The C++ and Java codes of [Flowchart 7.7\(b\)](#) can be observed in Programs P7_7_A.

C++ code:

```
// Program P7_7_A to compute the
// gcd of two integers
#include <iostream>
using namespace std;
int main() {
    int b, s, r;
    cout<<"Enter two integer: ";
    cin>>b>>s;
    cout<<"gcd("<<b<<","<<s<<")=";
    do {
        r=b%s;
        b=s;
        s=r;
    } while (r!=0);
    cout<<b;
    return 0;
}
```

Input/output:

```
Enter two integer: 87 24↵
gcd(87,24)=3
```

Java codes:

```
// Program P7_7_A to compute the
// gcd of two integers
import java.util.Scanner;
class P7_7_A {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int b, s, r;
        System.out.print("Enter two integers: ");
        b=read.nextInt();
        s=read.nextInt();
        System.out.print("gcd(" + b + "," + s + ")=");
        do {
            r=b%s;
            b=s;
            s=r;
        } while (r!=0);
        System.out.print(b);
        read.close();
    }
}
```

Input/output:

```
Enter two integer: 87 24↵
gcd(87,24)=3
```

If we change the do-while template in [Flowchart 7.7\(b\)](#) to a while template, [Flowchart 7.7\(c\)](#) is obtained.

Should we still print *b* upon exiting the loop? The answer is again in the implementation [Table 7.7\(c\)](#).

Tab. 7.7(c): Implementation table for [Flowchart 7.7\(c\)](#).

status	b	s	r
before the loop	87	24	
first repetition	24	15	15
second repetition	15	9	9
third repetition	9	6	6
fourth repetition	6	3	3

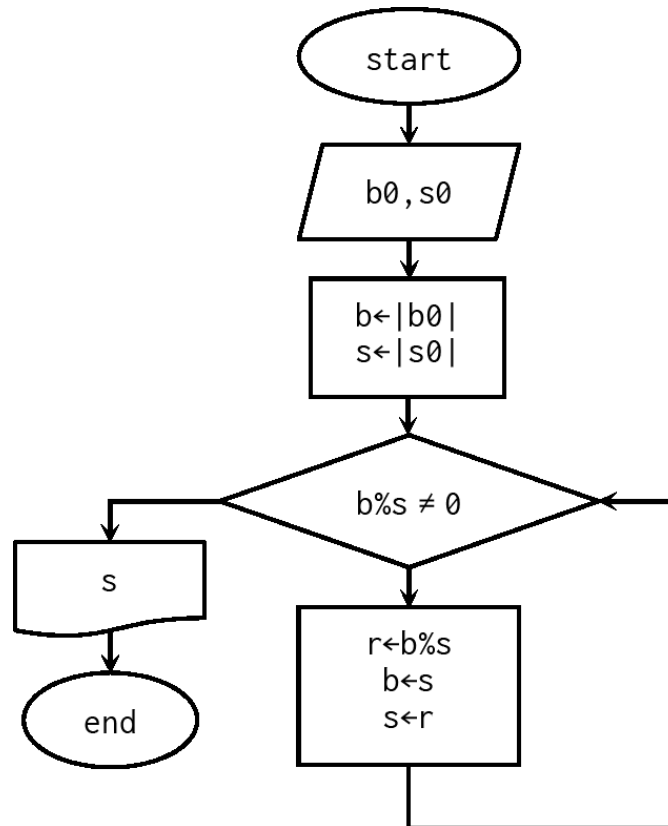


Fig. 7.7(c): Calculating the gcd using a while template.

This time, we should print either s or r . At the beginning of this algorithm, we substitute the absolute values of the read integers b_0 and s_0 for b and s , respectively, to avoid having a negative gcd. Of course, we can instead change the printing value to positive (write the absolute value of r instead of r in the print statement).

7.7.1. Exercise. Write the programs of [Flowchart 7.7\(c\)](#).



When we transfer the condition of the loop from the start to the end or vice versa, the condition and/or the outputs may change. Anyway, the implementation table should be arranged again to check the correctness of the algorithm.

We transform the process of [Flowchart 7.7\(c\)](#) into a function, named `gcd()`, which receives two arbitrary none-zero integers b and s and then calculates and returns their greatest common divisor. This function is depicted in [Flowchart 7.7\(d\)](#).

The main unit in Programs P7_7_B reads the two integers b_0 and s_0 . Then, it prints a message if at least one of these integers is zero. Otherwise, using the function `gcd()`, it calculates and prints the greatest common divisor.

C++ codes:

```
// Program P7_7_B to compute the  
// gcd using the gcd() function  
#include <iostream>  
#include <math.h>  
using namespace std;
```

Java codes:

```
// Program P7_7_B to compute the  
// gcd using the gcd() method  
import java.util.Scanner;  
class P7_7_B {  
    public static void main(String[] args) {
```

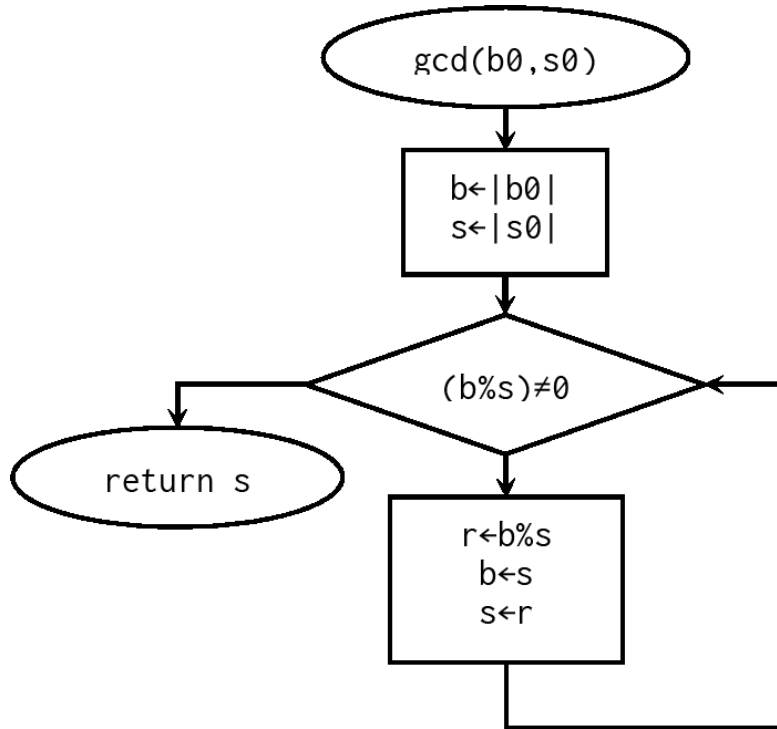


Fig. 7.7(d): A function to calculate the gcd.

```

int gcd(int, int);
int main() {
    int b0, s0;
    cout<<"Enter two integer: ";
    cin>>b0>>s0;
    if (b0*s0==0)
        cout<<"Not both non-zero";
    else
        cout<<"gcd("<<b0<<","<<s0<<")="
            <<gcd(b0, s0);

    return 0;
}
//*****
int gcd(int b0, int s0) {
    int b, s, r;
    b=abs(b0);
    s=abs(s0);
    while (b%s!=0) {
        r=b%s;
        b=s;
        s=r;
    }
    return s;
}

```

Input/output:

Enter two integer: 87 24↵
gcd(87,24)=3

```

Scanner read=new Scanner(System.in);
int b0, s0;
System.out.print("Enter two integers: ");
b0=read.nextInt();
s0=read.nextInt();
if (b0*s0==0)
    System.out.print("Not both non-zero");
else
    System.out.print("gcd(" + b0 + "," + s0
        + ")=" + gcd(b0, s0));

read.close();
}
//*****
static int gcd(int b0, int s0) {
    int b, s, r;
    b=Math.abs(b0);
    s=Math.abs(s0);
    while (b%s!=0) {
        r=b%s;
        b=s;
        s=r;
    }
    return s;
}
}

```

Input/output:

Enter two integer: 87 24↵
gcd(87,24)=3

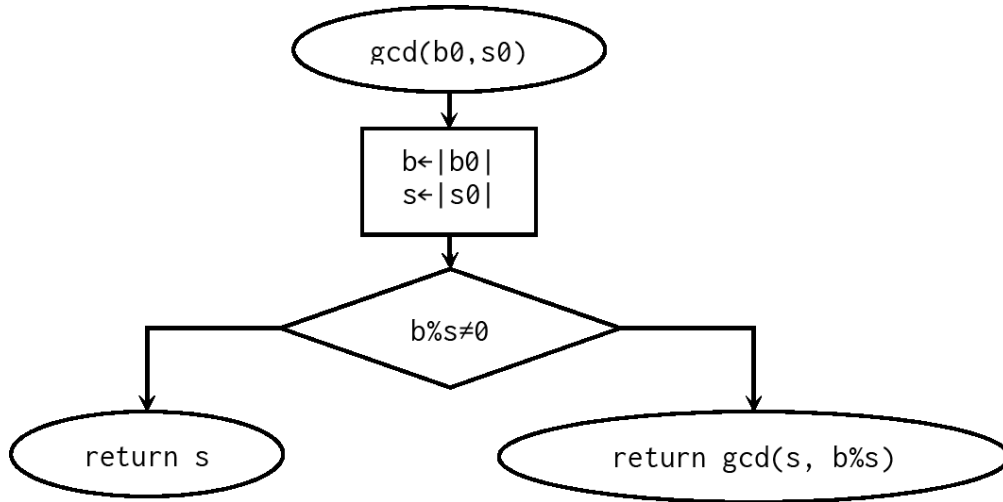


Fig. 7.7(e): A recursive function to Calculate the gcd.

Now, we transform the process in the function of **Flowchart 7.7(d)** into a function with the same name as gcd() using the recursive method (**Flowchart 7.7(e)**). Concentrate on **Flowcharts 7.7(d)** and **7.7(e)** and find the analogy in their logic! The subprogram associated with the function in **Flowchart 7.7(e)** is represented as follows.

C++ codes:

```
int gcd(int b0, int s0) {
    b=abs(b0);
    s=abs(s0);
    if (b%s!=0)
        return gcd(s, b%s);
    else
        return s;
}
```

Java codes:

```
static int gcd(int b0, int s0) {
    b=Math.abs(b0);
    s=Math.abs(s0);
    if (b%s!=0)
        return gcd(s, b%s);
    else
        return s;
}
```

7.8) The generalized Euclidean algorithm. Complete the algorithm of **Flowchart 7.7(c)** so that, in addition to finding the gcd of the two non-zero integers b_0 and s_0 , it calculates the two integers x and y leading to $\text{gcd}(b_0, s_0) = r = b_0 x + s_0 y$. **Solution.** First let b_0 and s_0 be positive. It suffices to find only one of the two required integers, say x , since knowing b_0, s_0, r , and x , the integer y can be obtained from the above relation as follows.

$$y = \frac{r - x b_0}{s_0}$$

Furthermore, the Euclidean algorithm is described in two simultaneous processes in order to further clarify the issue. In one process, calculate the algorithm itself to the point of achieving the gcd result and in the other one, compute the remainders as a linear combination of b and s . These two simultaneous processes are provided in **Table 7.8(a)**.

Tab. 7.8(a): calculating the remainders as a linear combination of b and s .

Euclidean algorithm	remainders
$b = s q_1 + r_1$ $b = s q_1 + r_1$	$r_1 = 1b - q_1 s$ $r_1 = 1b - q_1 s$
$s = r_1 q_2 + r_2$ $s = r_1 q_2 + r_2$	$r_2 = s - q_2 r_1 = -q_2 b + \lambda_2 s$ $r_2 = s - q_2 r_1 = -q_2 b + \lambda_2 s$
$r_1 = r_2 q_3 + r_3$ $r_1 = r_2 q_3 + r_3$	$r_3 = r_1 - q_3 r_2 = (1 - (-q_2) q_3) b + \lambda_3 s$ $r_3 = r_1 - q_3 r_2 = (1 - (-q_2) q_3) b + \lambda_3 s$

$$\begin{array}{r}
r_3 = r_1 - q_3 r_2 = (1 - (-q_2)q_3)b + \lambda_3 s \\
r_2 = r_3 q_4 + r_4 \quad r_4 = r_2 - q_4 r_3 = (-q_2 - (1 + q_2 q_3)q_4)b + \lambda_4 s \\
\vdots \qquad \qquad \qquad \vdots
\end{array}$$

This table aims to calculate the coefficients of b . Therefore, the coefficient of s is not explained in details. The third row in the right column is reviewed in order to find how the coefficients of b are calculated in each step:

$$\begin{aligned}
r_3 &= r_1 - q_3 r_2 = (1 - (-q_2)q_3)b + \lambda_3 s. \\
r_3 &= r_1 - q_3 r_2 = (1 - (-q_2)q_3)b + \lambda_3 s.
\end{aligned}$$

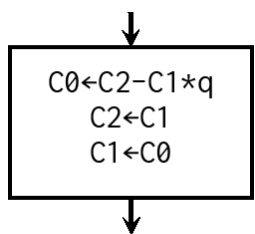
From the algorithm viewpoint, we define the following variables to state the coefficient of b in this row:

- q : the quotient in the current row;
- $C0$: the coefficient of b in the current row;
- $C1$: the coefficient of b in the previous row;
- $C2$: the coefficient of b in the two rows before.

Now, as represented in **Table 7.8(a)**, the coefficient in the current row is obtained from the following relation:

$$C0 = C2 - C1 q. \quad C0 = C2 - C1 q.$$

As shown in the fourth row of **Table 7.8(a)**, $C1$ and $C0$ substitute for $C2$ and $C1$, respectively, in order to transfer the calculation to the next rows. Precisely, the following instructions are implemented in the quoted orders:



Finally, the initial values of $C2$ and $C1$ are left to determine. These values should be selected in such a way that in the first row, $C0$ would be equal to 1, independent of the quotient q . For this purpose, $C2$ and $C1$ should have the initial values 1 and 0, respectively.

In **Flowchart 7.7(c)**, there is a while template which calculates the gcd of the read integers b_0 and s_0 . Therefore, considering the simultaneous processes in the **table 7.8**

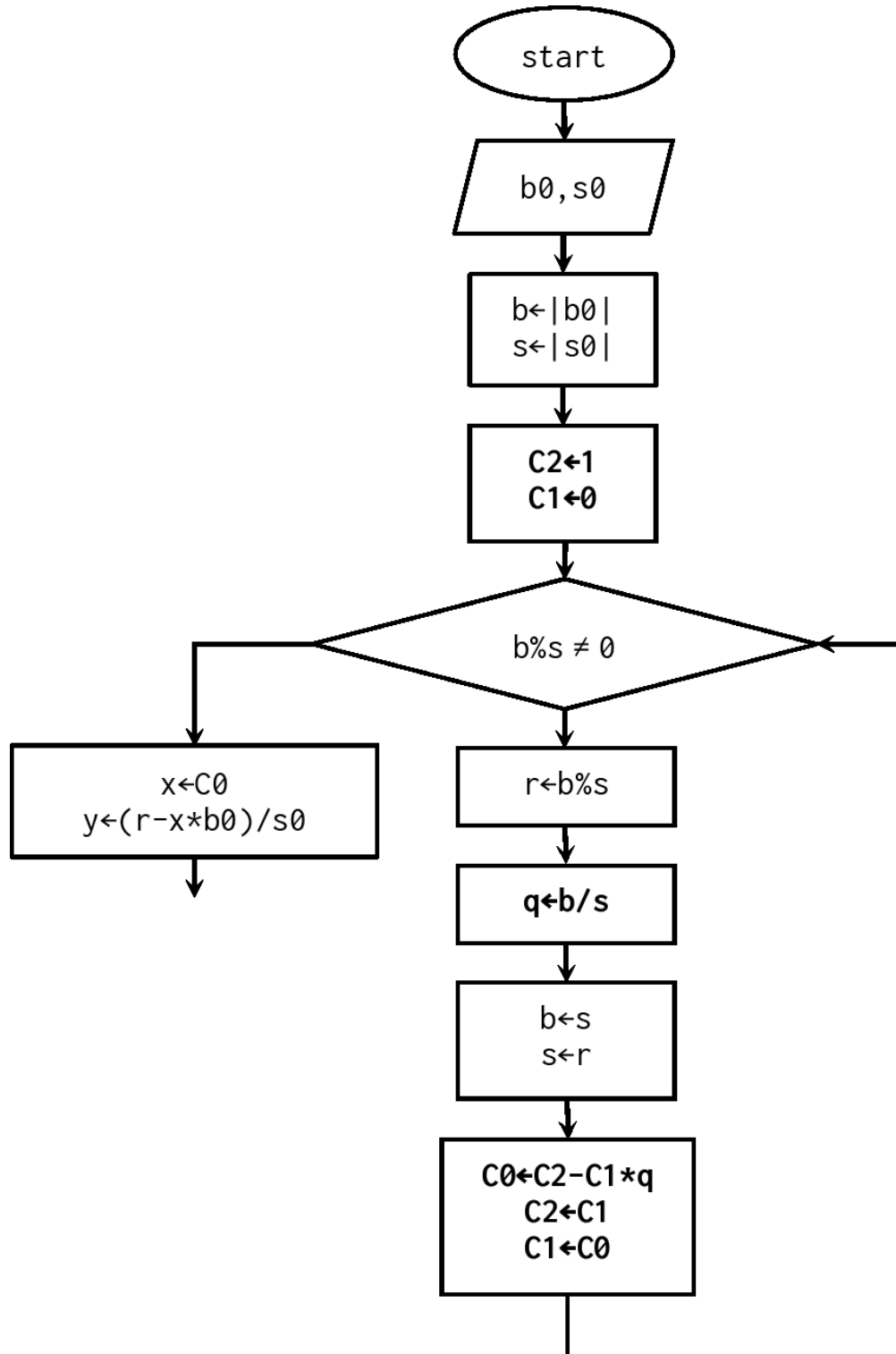


Fig. 7.8(a): Euclidean algorithm and extended Euclidean algorithm simultaneously.

and the above arguments, we can extend the while template of this flowchart to **Flowchart 7.8(a)** in which the new instructions are represented in a bold format. The required integers are:

$$x = C0, \quad y = \frac{r - xb0}{s0}.$$

The implementation **Table 7.8(b)** is the extended version of **Table 7.7(c)** for **Flowchart 7.8(a)**. As mentioned above, the required integer x is indeed $C0$.

Tab. 7.8(b): Simultaneous calculations of the coefficient of b and the gcd.

status	b	s	r	q	C2	C1	C0
before the loop	87	24			1	0	

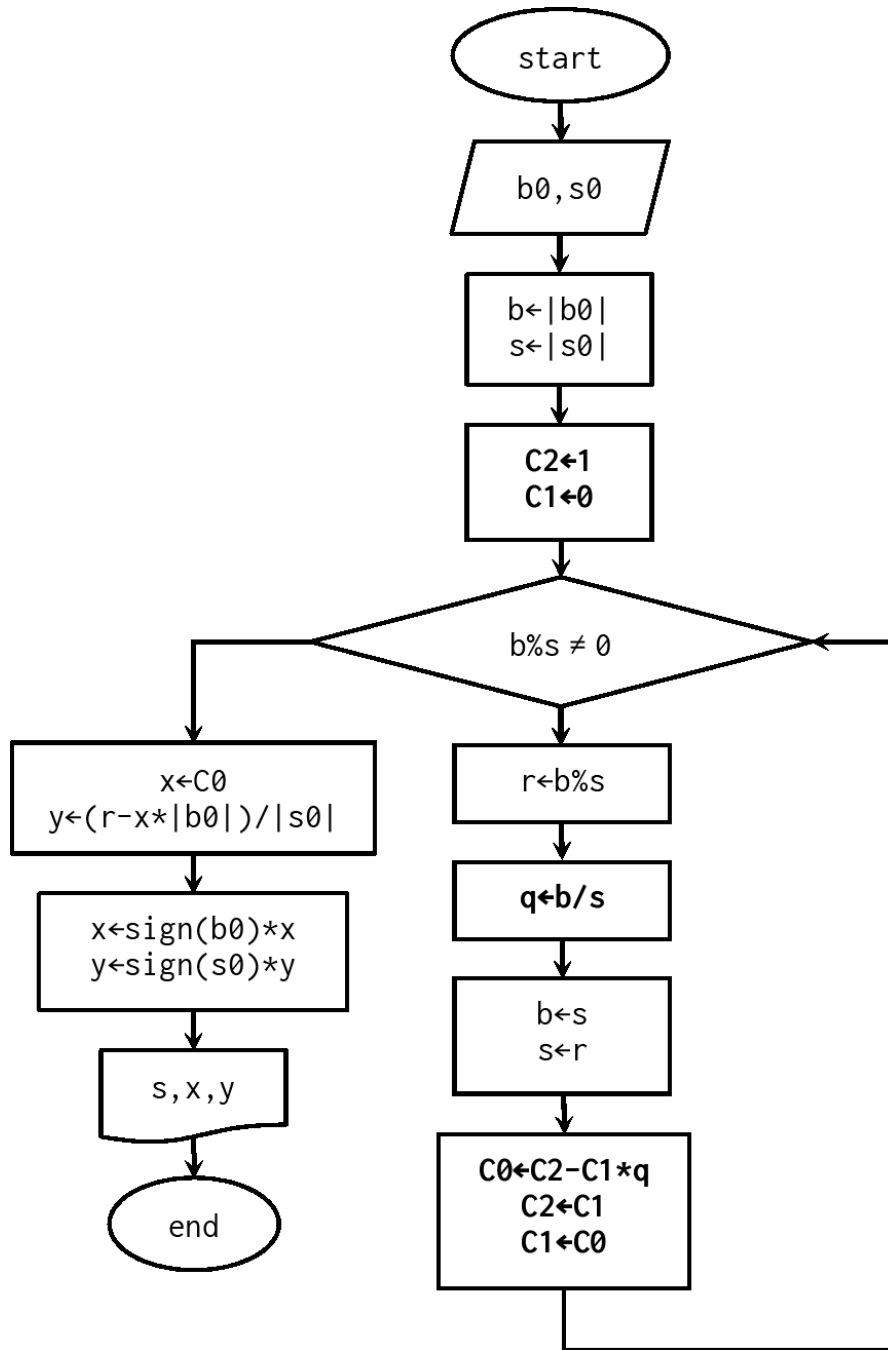


Fig. 7.8(b): Euclidean algorithm and extended Euclidean algorithm simultaneously (completed).

status	b	s	r	q	C2	C1	C0
first repetition	24	15	15	3	0	1	1
second repetition	15	9	9	1	1	-1	-1
third repetition	9	6	6	1	-1	2	2
fourth repetition	6	3	3	1	2	-3	-3

Now, let b_0 and s_0 be any non-zero integers. Then the required integers x and y are:

$$x = C_0, \quad y = \frac{r - x|b_0|}{|s_0|}.$$

Moreover, in this case, the linear combination is as follows.

$$r = b_0(x \times \text{sign}(b_0)) + s_0(y \times \text{sign}(s_0))$$
$$r = b_0(x \times \text{sign}(b_0)) + s_0(y \times \text{sign}(s_0))$$

The completed flowchart is illustrated in [Figure 7.8\(b\)](#) with the codes in Programs P7_8.

C++ codes:

```
// Program p7_8 to extend the Euclidean
// Algorithm: finding the integers
// x,y such that bx + sy = gcd(b,s)
#include <iostream>
#include <math.h>
using namespace std;
int sign(int k) {
    return k/abs(k);
}
int main() {
    int b0, s0, b, s, r, q, C0, C1,
        C2, x, y;
    cout<<"Enter two integers: ";
    cin>>b0>>s0;
    b=abs(b0); s=abs(s0);
    C2=1; C1=0;
    while (b%s!=0) {
        r=b%s;
        q=b/s;
        b=s;
        s=r;
        C0=C2-C1*q;
        C2=C1;
        C1=C0;
    }
    cout<<"gcd("<<b0<<","<<s0<<")="<<s;
    x=C0;
    y=(r-abs(b0)*x)/abs(s0);
    x=sign(b0)*x;
    y=sign(s0)*y;
    cout<<"("<<x<<")("<<b0<<")+("<<y
        <<")("<<s0<<")"<<endl;
    return 0;
}
```

Input/output:

```
Enter two integers: 87 24↵
gcd(87,24)=3=(-3)(87)+(11)(24)
```

Java codes:

```
// Program p7_8 to extend the Euclidean
// Algorithm: finding the integers
// x,y such that bx + sy = gcd(b,s)
import java.util.Scanner;
class P7_8 {
    static int sign(int k) {
        return k/Math.abs(k);
    }
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int b0, s0, b, s, r=0, q, C0=0, C1,
            C2, x, y;
        System.out.print("Enter two integers: ");
        b0=read.nextInt();
        s0=read.nextInt();
        b=Math.abs(b0); s=Math.abs(s0);
        C2=1; C1=0;
        while (b%s!=0) {
            r=b%s;
            q=b/s;
            b=s;
            s=r;
            C0=C2-C1*q;
            C2=C1;
            C1=C0;
        }
        System.out.print("gcd(" + b0 + "," + s0
            + ")=" + s);
        x=C0;
        y=(r-Math.abs(b0)*x)/Math.abs(s0);
        x=sign(b0)*x;
        y=sign(s0)*y;
        System.out.print("(" + x + ")(" + b0
            + ")+" + y + ")(" + s0 + ")");
        read.close();
    }
}
```

Input/output:

```
Enter two integers: 87 24↵
gcd(87,24)=3=(-3)(87)+(11)(24)
```

It is worth mentioning that we should initialize the variables r and $C0$ to 0 in the right Java program. However, in the left program this is not necessary.

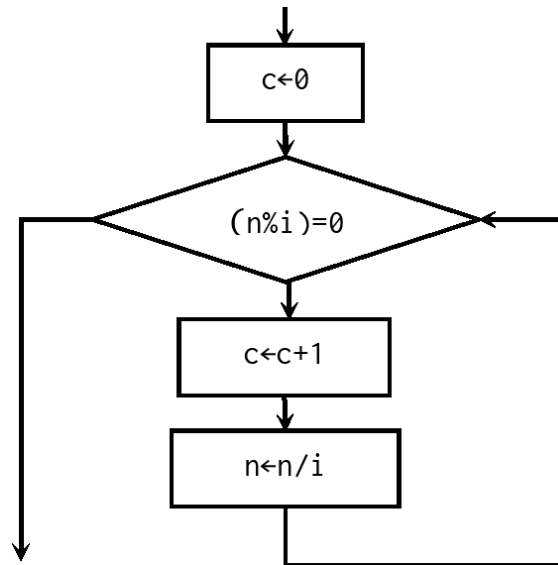


Fig. 7.9(a): Counting the prime factor i .

7.8.1. Exercise. Write a function, named `egcd()` to receives two non-zero integers b_0 and s_0 . Then, calculate and return the gcd of these two numbers as well as the two integers x and y mentioned in the extended Euclidean algorithm. Afterwards, write a main algorithm to read the two integers b_0 and s_0 . Next, print a message if at least one of them is zero. Otherwise, calculate and print the returned values, using the function `egcd()`. Finally, write the complete programs.

7.8.2. Exercise. Repeat the previous exercise using a do-while template instead of while template.

7.9. Example. Write an algorithm to read a positive integer n and decompose it as a multiplication of its prime factors and then print it like the following output for the input 504:

$$504 = 2^3 * 3^2 * 7^1$$

Solution. This problem needs more attention. We should look for the prime factors of n in the range of integers from 2 to n . Accordingly, we perform the following tasks in the range of a for loop with the specification $i=2,n$:

1. Pass the i through the `prime()` function filter and count the number of the i inside n in the F-path of an if template. To do this, define a counter named c with the initial value of 0. Then, inside the range of a while template, while n is divisible to i , first, increase c by one and then, take an i from n by substituting the n/i for n . **Flowchart 7.9(a)** displays this count.

2. How do we print the required format after exiting this while template? It is clear that the i is the (prime) factor of n if it exists at least once inside n . Thus, we check whether c is positive. If so, we attempt to print the required format in the F-path of an if template with the rule as follows. What is clear is that in each repetition, the value of the factor i along with its power c is printed in the form i^c followed by the $*$ character except for the last repetition in which there is no asterisk. Accordingly, we divide the printing task into two sections due to whether or not the repetition is the last one which is happened only when n is 1. Therefore, if $n = 1$, we terminate the program by printing the string:

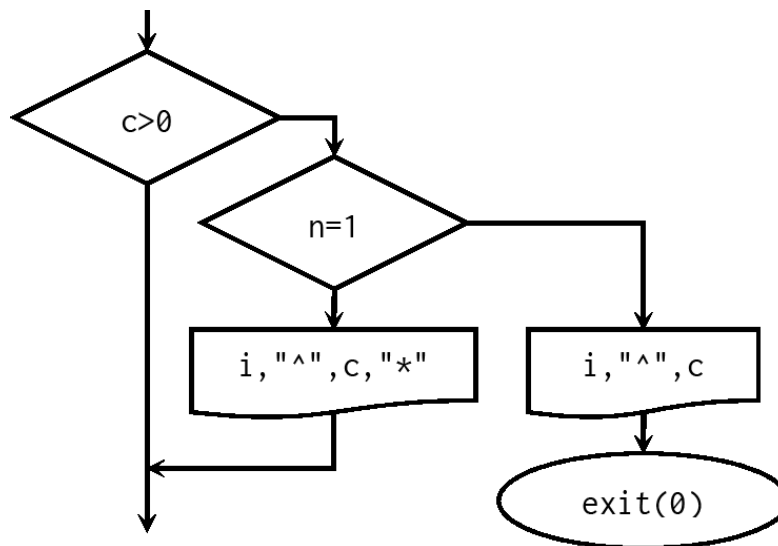


Fig. 7.9(b): Printing the required format.

`i, "^", c`

Otherwise, the string:

`i, "^", c, "*"`

is printed in each repetition. It is noteworthy that this printing formats are different in C++ and Java (see Programs P7_9). **Flowchart 7.9(b)** illustrates this printing design.

Only the part “504=” is left from the required print format. This can be performed by printing the string:

`n, "="`

before the for template. The above discussion, is summarized in the completed **Flowchart 7.9(c)**. Programs P7_9 is the translation of this algorithm into C++ and Java codes.

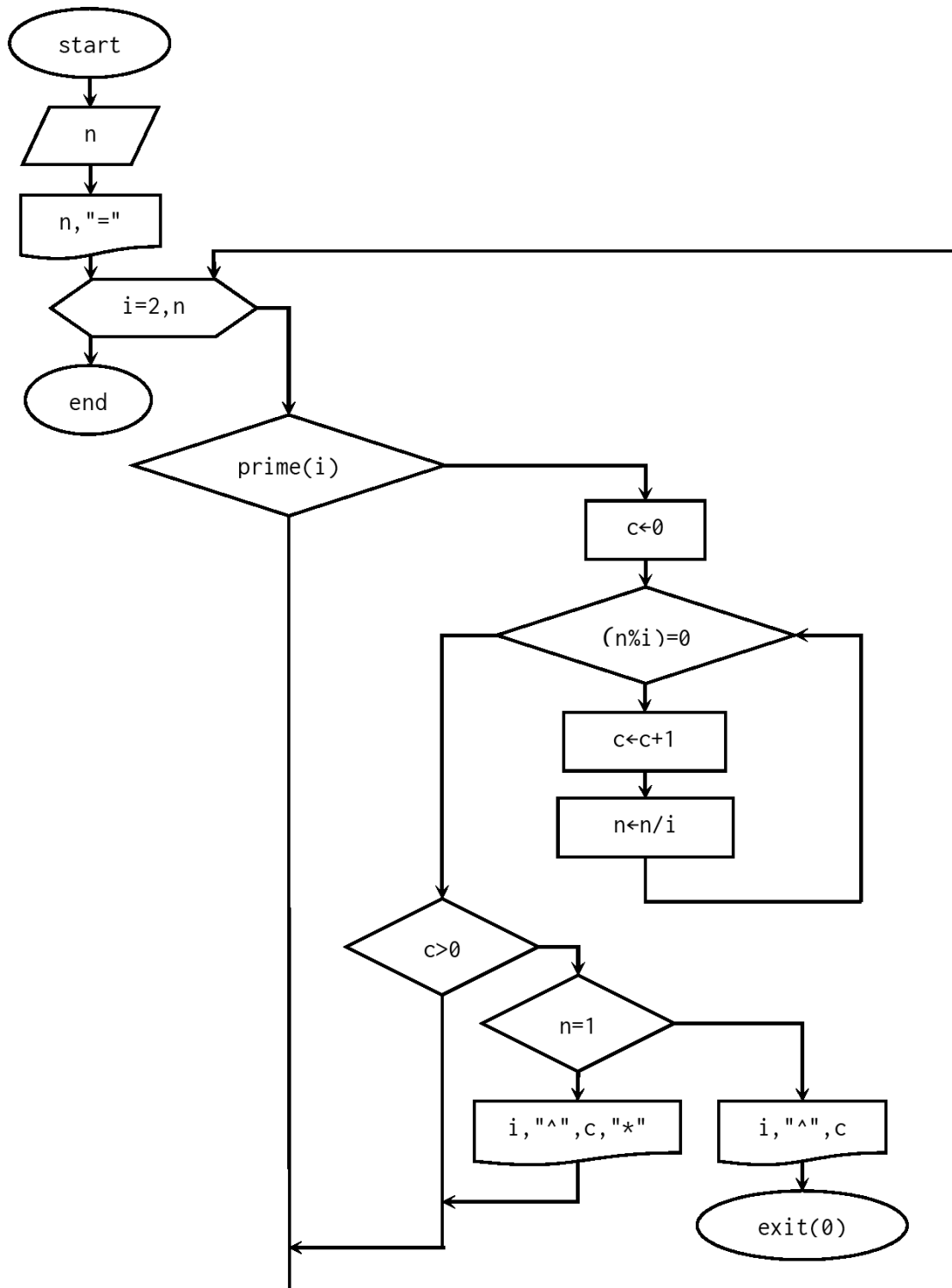


Fig. 7.9(c): Primary decomposition of n along with the formatted print.

C++ code:

```
// Program P7_9: Primary decomposition
// of a positive integer
#include <iostream>
#include <stdlib.h>
#include <math.h>
using namespace std;
int prime(int);

int main() {
    int i, n, c;
    cout<<"Enter an integer: ";
    cin>>n;
    cout<<n<<"=";
    for (i=2; i<=n; i++) {
        if (prime(i)==1) {
            c=0;
            while (n%i==0) {
                c++;
                n=n/i;
            }
            if (c>0) {
                if (n==1) {
                    cout<<i<<"^"<<c;
                    exit(0);
                }
                cout<<i<<"^"<<c<<"*";
            }
        }
    }
    return 0;
}
//*****
int prime(int n) {
    int i, p=1;
    for (i=2; i<=floor(sqrt(n)); i++)
        if (n%i==0)
            p=0;
    return p;
}
```

Input/output:

```
Enter an integer: 504↵
504=2^3*3^2*7^1
```

Java codes:

```
// Program P7_9: Primary decomposition
// of a positive integer
import java.util.Scanner;
class P7_9 {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int i, n, c;

        System.out.print("Enter an integer: ");
        n=read.nextInt();
        System.out.print(n+"=");
        for (i=2; i<=n; i++) {
            if (prime(i)==1) {
                c=0;
                while (n%i==0) {
                    c++;
                    n=n/i;
                }
                if (c>0) {
                    if (n==1) {
                        System.out.print(i + "^" + c);
                        System.exit(0);
                    }
                    System.out.print(i + "^" + c + "*");
                }
            }
        }
        read.close();
    }
    //*****
    static int prime(int n) {
        int i, p=1;
        for (i=2; i<=Math.floor(Math.sqrt(n)); i++)
            if (n%i==0)
                p=0;
        return p;
    }
}
```

Input/output:

```
Enter an integer: 504↵
504=2^3*3^2*7^1
```

7.9.1. Exercise. The completed **Flowchart 7.9(c)** as well as Programs 7_9 do not work for 0, 1, and the negative integers. Modify the flowchart and the programs so that they

work for any integer.

In numerical computations, a certain quantity is often approximated in a series of specific iterated procedures so that in each step the approximated amount becomes more accurate compared to the previous step. In such computations, if we need an accuracy of n decimal digits, then we have to repeat these steps until the absolute value of two consecutive approximations is smaller than

$$5 \times 10^{-(n+1)} = \overset{n \text{ times}}{0.00 \dots 05} 5 \times 10^{-(n+1)} = \overset{n \text{ times}}{0.00 \dots 05}$$

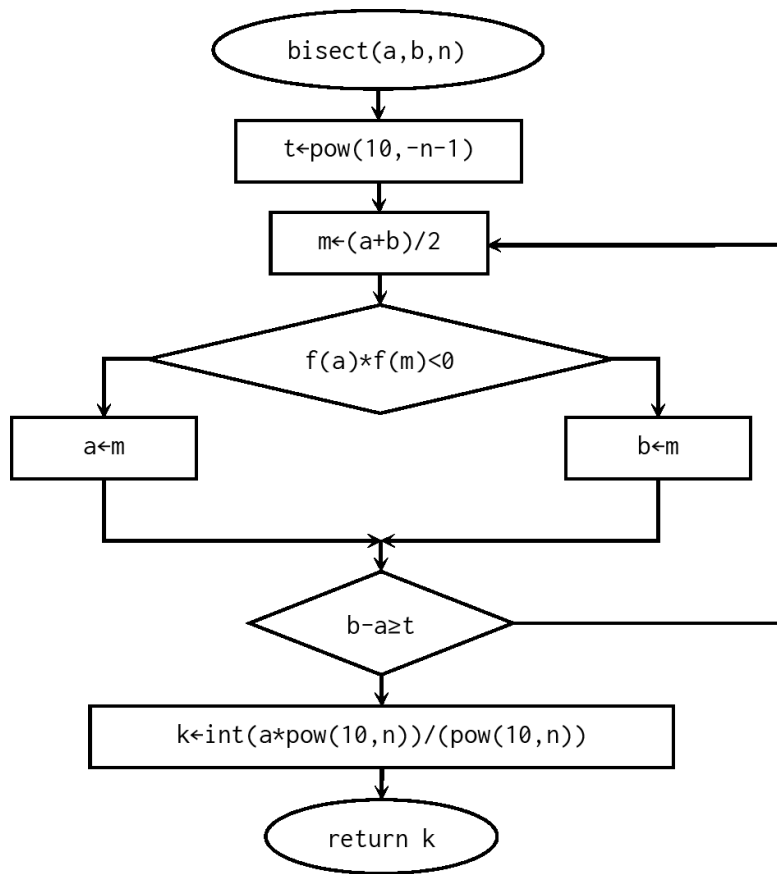


Fig. 7.10: Finding the approximated root of a function f using the bisection method.

This amount is called the *tolerance*. In this case, the last approximation, or the approximation before the last one, up to n decimal digits, is the desired approximation.

7.10. Example. The bisection method is one of the methods of finding the approximated root of a function f . In this method, an interval $[a, b]$ is guessed in which $f(a)$ and $f(b)$ have different signs, that is, f has a root. Now, we halve the interval and take the half in which the values of the function at the endpoints have different signs (f has a root). We

may repeat this procedure as many times as needed. Write the algorithm of a function to receive the endpoints a and b and the positive integer n and then calculate and return the approximated root of f with the precision of n decimal digits.

Solution. Take the tolerance as t . Find the middle point of the interval $[a, b]$ in the range of a do-while template and mark it as m . Now, take m as the end point of the new interval (substitute m for b) if $f(a)$ and $f(m)$ have different signs; otherwise, take it as its start point (substitute m for a). Repeat this procedure while the value of $b - a$ is greater than or equals to the tolerance t . Now the value of either a or b with the precision of n decimal digits is the required approximation. To separate this value from, say a , it suffices to multiply it by 10^n , 10^n , take the integer part of the obtained number, and finally, divide it by 10^n 10^n again. The resulted algorithm is depicted in **Flowchart 7.10**.

In the main unit of Programs P7_10, first, the amounts of a and b are read. If $f(a)$ and $f(b)$ have the same signs, a message is printed. Otherwise, calling the function `bisect()`, the approximated root of the functions $f(x) = x^6 - x - 1$ is calculated and printed with the precision of seven decimal precision.

C++ codes:

```
// Program P7_10 to find the
// approximated root of y=f(x)
// using the bisection method
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;
double f(double);
double bisect(double, double, int);
int main() {
    float a, b;
    int n;
    cout<<"Enter the endpoints: ";
    cin>>a>>b;
    cout<<"Enter the number of decimal "
        <<"precision: ";
    cin>>n;
    if (f(a)*f(b)>=0)
        cout<<"Illegal interval";
    else
        cout<<setprecision(8)
            <<bisect(a, b, n);
    return 0;
}
//*****
double f(double x) {
    return pow(x, 6)-x-1;
}
//*****
double bisect(double a, double b,
              int n) {
    double m, t, k;
    do {
        t=pow(10, -n-1);
        m=(a+b)/2;
        if (f(a)*f(m)<0)
            b=m;
        else
            a=m;
    } while (b-a>=t);
    k=int(a*pow(10, n))/(pow(10, n));
    return k;
}
```

Java codes:

```
// Program P7_10 to find the
// approximated root of y=f(x)
// using the bisection method
import java.util.Scanner;
class P7_10 {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        float a, b;
        int n;
        System.out.print("Enter the "
            + "endpoints: ");
        a=read.nextFloat();
        b=read.nextFloat();
        System.out.print("Enter the number of "
            + "decimal precision: ");
        n=read.nextInt();
        if (f(a)*f(b)>=0)
            System.out.print("Illegal interval");
        else
            System.out.printf("%-10.7f",
                bisect(a, b, n));
        read.close();
    }
    //*****
    static double f(double x) {
        return Math.pow(x, 6)-x-1;
    }
    //*****
    static double bisect(double a, double b,
                        int n) {
        double m, t, k;
        do {
            t=Math.pow(10, -n-1);
            m=(a+b)/2;
            if (f(a)*f(m)<0)
                b=m;
            else
                a=m;
        } while (b-a>=t);
        k=(int) (a*Math.pow(10, n))
            /(Math.pow(10, n));
        return k;
    }
}
```



```
}
```

Input/output:

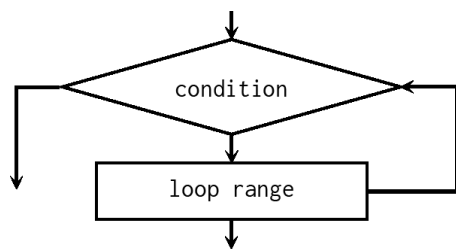
```
Enter the endpoints: 1 2↵
Enter the number of decimal precision: 7↵
1.1347241
```

```
Enter the endpoints: 1 2↵
Enter the number of decimal precision: 7↵
1.1347241
```

7.3 The if-goto loops (C++ only)

In the C++ language, the programs of the while and do-while templates can be written using the two statements if and goto as in **Figure 7.11**.

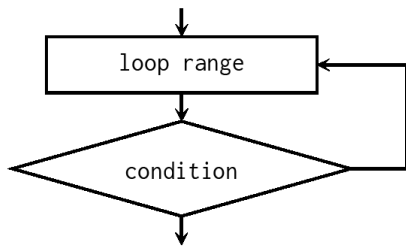
Using the if-goto loops is not recommend since it confuses the reader of the program. Recall that the goto statement is not defined for the compiler of Java language although it is counted as a keyword. Practically, often there is no need to use the goto statement, even in C++. To observe this fact, we provide several various examples in the following patterns. The italic phrases are replaced by the real codes used by the user in these examples. The first columns (from left), are valid codes in C++ using the goto statement. In the second or third columns, which are valid in both C++ and Java, the existence of label L is not needed in practice and it is written just for comparisons.



The while template

C++ codes:

```
100: if (condition) {
    loop range
    goto 100;
}
```



The do-while template

C++ codes:

```
100: {
    loop range
} if (condition) goto 100;
```

Fig. 7.11: The if-goto loops equivalent to the while and do-while templates.

A part with the goto statement:

```
L:
while (condition 1) {
    block 1
    if (condition 2)
```

Equivalent part with the continue statement:

```
L:
while (condition 1) {
    block 1
    if (condition 2)
```

```

    if (condition 2)
        goto L;
    block 2
}    block 2

```

```

    if (condition 2)
        continue;
    block 2
}

```

A part with the goto statement:

```

while (condition 1) {
    block 1
    if (condition 2)
        goto L;
    block 2
}
L:

```

Equivalent part with the break statement:

```

while (condition 1) {
    block 1
    if (condition 2)
        break;
    block 2
}
L:

```

A part with the goto statement:

```

    block 1
    if (condition)
        goto L;
    block 2
L: block 3

```

Equivalent part without the goto statement:

```

    block 1
    if (!condition)
        block 2
L: block 3

```

A part with the goto statement:

```

    block 1
L: block 2
    block 3
    if (condition)
        goto L;

```

Equivalent part with the do-while statement:

```

    block 1
    do {
L: block 2
        block 3 } while (condition);

```

A part with goto :

```

    if (condition)
        goto L;
    block 1
L:

```

Equivalent part without goto :

```

    if (condition) {}
    else {
        block 1
    }
L:

```

Equivalent part without goto :

```

    if (!condition) {
        block 1
    }
L:

```

A part with goto :

```

    if (condition)
        goto L;
    else {
        block 1
    }
    block 2
L:

```

Equivalent part without goto :

```

    if (!condition) {
        block 1
        block 2
    }
L:

```

Equivalent part without goto :

```

    if (!condition) {
        block 1
        block 2
    }
L:

```

Exercises

In the following exercises: (1) Arrange the implementation table, if needed, (2) Write the complete program, (3) Provide appropriate input notifications and output headings, if any. In addition, the user-defined functions in the text of the current and the previous chapters may be used unless otherwise is explicitly specified.

7.1. The approximated amount of the number π is calculated using the following formula.

$$\pi \approx 4 \left(\frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots \right).$$

Write an algorithm to calculate and print the approximated amount of π with the sum of the ten first terms of this series.

7.2. Write an algorithm to calculate and print the approximated amount of π with the precision of 4 decimal digits using the formula in [Exercise 7.1](#).

7.3. The approximated amount of the Neperian logarithm of the positive number x is calculated using the formula

$$\ln(x) \approx \frac{x-1}{x} + \frac{1}{2} \left(\frac{x-1}{x} \right)^2 + \frac{1}{3} \left(\frac{x-1}{x} \right)^3 + \dots$$

Write an algorithm to read the positive number x and print the undefined message if it is non-positive; otherwise, calculate and print the amount of $\ln(x)$ with the sum of the first ten terms of the above series.

7.4. Repeat [Exercise 7.3](#) with the precision of 4 decimal digits instead of the ten first terms of the series.

7.5. Write an algorithm to read the two positive integers m and n and then calculate and print the quotient and remainder of the division of m by n using the repeated subtractions. For example, the equation $13 - 4 - 4 - 4 = 1$ indicates that the quotient of the division 13 by 4 is 3 and the remainder is 1.

7.6. The first two terms of a series are 1 and 2 and from the second term onwards, the distance between the two terms is one more than the distance of the previous two terms. Some of the terms of this series are:

		1	2	4	7	11	16	...	
1	2	4		7		11	16	...	

Write an algorithm to read a positive integer n , where n is assumed greater than 2. Then, calculate and print the n first terms of this series, as well as their sum.

7.7. Write a function to receive a positive integer n , where n is assumed greater than 2. Then, calculate and return the n -th term of the sequence in [Exercise 7.6](#). To do this, use the direct method and once again use the recursive method.

7.8. Write an algorithm to determine and print the terms of the Fibonacci sequence between 100 and 150.

7.9. Write an algorithm to determine and print the number and sum of the terms of the Fibonacci sequence smaller than 100.

7.10. Write an algorithm to produce and print the first 20 terms of the Fibonacci sequence.

7.11. Write a function named SumDig() to receive a positive integer n and then calculate and return the sum of its digits.

7.12. Write an algorithm to read the positive integer n . Then, read n positive integers and determine and print the numbers which the sum of their digits are greater than 45. Finally, print the number of these numbers.

7.13. Write an algorithm to read a positive integer n and then determine whether or not n is divisible to the sum of its digits by printing an appropriate message.

7.14. Write an algorithm to read a positive integer n . Then, read n integers one by one and each time print the reverse of the read number.

7.15. A symmetric integer is an integer which is equal to its reverse. For example, 1221 is symmetrical while 867 is not. Write an algorithm to read a positive integer n and then determine whether or not n is symmetric by printing one of the messages YES or NO after calculating its reverse.

7.16. Write an algorithm to calculate and print the four-digit symmetric integers and the number of such integers.

7.17. Write a function named DecPart() to receive a real number t and then calculate and return the decimal part of t as an integer by removing the decimal point.

7.18. Write an algorithm to find the reverse of an arbitrary real number t . For example, the reverse of 12.345 is 543.21.

7.19. Repeat the previous exercise directly without using [Exercise 7.17](#).

7.20. Write an algorithm to read an integer n with $n > 2$. Then, read n integers from the input one by one and calculate and print their greatest common divisor. It should be mentioned that the greatest common divisor of a set of numbers is determined recursively. For example

$$\gcd(a, b, c) = \gcd(\gcd(a, b), c) \quad \gcd(a, b, c) = \gcd(\gcd(a, b), c)$$

7.21. A polynomial is called primitive if its coefficients are prime to each other. Write an algorithm to read the degree and the coefficients of a polynomial and determine whether or not it is primitive by printing one of the messages Primitive or Not primitive.

7.22. What is the possibility that a pair of integers from 1 to n are coprime? Write an algorithm to read the positive integer n and then calculate and print this possibility. For example, for $n = 4$, among 10 ($= 2 + 3 + 4$) pairs of numbers from 1 to 4, six of them are coprime; therefore, the possibility is 0.6 ($= 6 / 10$).

7.23. Write an algorithm to read a positive integer n and then print the first prime number to the n -th one.

7.24. Write an algorithm to read a positive integer n and then calculate and print the n -th prime number.

7.25. A perfect integer is defined in [Exercise 6.26](#). Write an algorithm to read a positive integer n and print the perfect numbers from the first to the n -th one.

7.26. Write an algorithm to read a positive integer n and then calculate and print the n -th perfect number.

7.27. Write an algorithm to read a positive integer n having at most eight digits. Then, determine whether or not n is a factorial of an integer by printing an appropriate message. If yes, determine the integer k for which $k! = n$.

7.28. For a positive integer n , the phi Euler function of n , denoted by $\varphi(n)$, is defined to be the number of positive integers smaller than n and coprime to n . Write an algorithm to read the positive integer n and then calculate and print $\varphi(n)$. Additionally, print all the positive integers smaller than n and coprime to n .

7.29. Write a sub-algorithm to receive the two arbitrary integers b and s and find their greatest common divisor, g , together with the two numbers x and y so that $g = bx + sy$. Then, return g , x , and y to the caller unit. Write this sub-algorithm with two ways: once using the while template and once again with the do-while template.

7.30. The approximated amount of the n -th root of the real number t can be calculated using the following repetitive way in which a presupposed root x_0 should be selected beforehand.

$$x_{i+1} = \frac{1}{n}(n-1)x_i + \frac{t}{x_i^{n-1}}.$$

Write an algorithm to read the amount of t , n , and the presupposed root x_0 from the input and then calculate the amount of the n -th root of t and print it with three decimal digits precision.

7.31. Solve Exercises 4.19, 4.20, 4.21, and 5.12 using either whole or do-while loop.

Supplementary exercises

7.1*. Write an algorithm to read the integers a , c , and m . Then, print a message if $\text{gcd}(a, m)$ is not divisible by c . Otherwise, solve the following congruent equation.

$$ax \equiv c \pmod{m}.$$

7.2*. Write an algorithm to read the integers b , c , m , and n . Then, print an appropriate message if $\text{gcd}(m, n) \neq 1$. Otherwise, solve the following congruent equations system.

$$\begin{aligned} x &\equiv b \pmod{m} \\ x &\equiv c \pmod{n}. \end{aligned}$$

7.3*. Write a function named $\text{akmodm}()$ to receive the integers a , k , and m and then calculate and return the value of a^k modulo m . The amounts of a and k are assumed none-negative and smaller than m .

7.4*. Suppose that n is a positive integer. The set of positive integers less than and coprime to n form a group with the multiplication modulo n . Assume that a is a member of this group. Then,

$$\gcd(a, n) = 1, \text{ for all } 0 < a < n.$$

$$\gcd(a, n) = 1, \text{ for all } 0 < a < n.$$

The smallest positive integer k satisfying the

$$a^k \equiv 1 \pmod{n}, a^k \equiv 1 \pmod{n},$$

is called the order of a modulo n . Write a function named `order()` to receive two positive integers a and n , with the above properties, and calculate and return the order of a modulo n . Next, write a main algorithm to read two positive integers a and n and print a message if they do not possess the above properties. Otherwise, calculate and print the order of a modulo n using the function `order()`.

7.5*. Write a function to receive a positive integer and then determine and return its reverse using the recursive method.

7.6*. Write a function named `dec()` to receive a positive integer n in the base-2 numeral system, convert it into the base-10 numeral system, and return the result.

7.7*. Write an algorithm to read a positive integer n and group it in triples, from right to left and then print this integer. For example, the read integer 5176254 is printed as 5,176,254. The solution for this exercise is simpler if the arrays are used (**Exercise 8.13***).

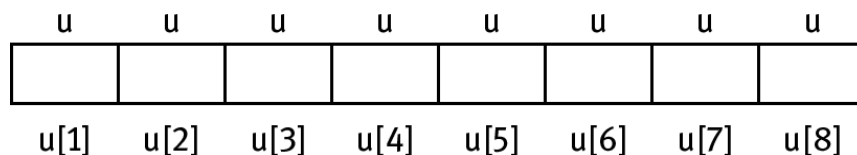
7.8*. Do the following process to convert a positive integer n from the base-2 numeral system to the base-8 numeral system: arrange n in the groups of threes from right to left, convert each group to the base-10 (usual) numeral system, and put it in its location. The resulting number is the representation of the number in the base-8 numeral system. Write an algorithm to read a positive integer n in the base-2 numeral system, convert it into the base-8 numeral system, and then print the result.

7.9*. To convert a number from the base-8 numeral system into the base-2 numeral system do as in the previous exercise but in an opposite direction: convert each digit into the base-2 numeral system using the `bin1()` function in **Example 7.6** and place it in its position. The obtained number is the representation of the number in the base-2 numeral system. Write an algorithm to read the positive integer n in the base-8 numeral system, convert it into the base-2 numeral system, and then print the result.

8 One-dimensional arrays

8.1 vectors

In previous chapters, we addressed independent and non-homonymous variables in algorithm writing and programming. Now, we start working with variables named arrays which include one or more indices. These variables have the same name but different indices. In the process of solving certain problems, we have to deal with a large number of variables having similar behaviours in the program. In these situations, defining all the variables independently consumes the time of the algorithm, increases its size and thus makes it useless. Therefore, we use arrays which have homonymous variables and different indices in order to overcome this problem. An important advantage of using arrays is in object-oriented programming: arrays, as the predefined objects, pass by the reference. In the present chapter, we concentrate on one-dimensional arrays and remove the phrase “one-dimensional”. This type of array is occasionally regarded as a vector or a list. For example, an array of eight entries is illustrated in the following diagram.



As shown, eight different variables exist in this diagram, all having the same name while different indices. In theoretical studies, these variables are written as u_1, u_2, \dots, u_8 .

All of the entries of an array should be of the same data type which is then associated with the array data type. For instance, an array is integer if all of its entries are integers. The declaration of arrays in programming is as follows.

declaration of arrays in C++:

data type array name[array length];

declaration of arrays in Java:

data type_array name[]=new_*data type*[array length];

For example, the statement

in C++:

int u[8];

in Java:

int u[]=new int[8];

declares the variable u as a one-dimensional integer array with a length of 9. As regards the entry u_n in programs, we write $u[n]$. By default, the index of the entries starts from 0 in both C++ and Java compilers. However, we may start from index 1 if necessary, in which case, the length should be added up by one unit (for the zero index). For example, if we want to work with u_0, u_1, \dots, u_7 we introduce u as above. On the other hand, we introduce u with a length of 9 if we want to work with u_1, u_2, \dots, u_8 :

in C++:

int u[9];

in Java:

int u[]=new int[9];

In algorithm writing, we may declare an array in different ways inside a rectangle with dashed border, depending on what feature we need. For example, there are three patterns in the following figure. In the leftmost pattern, the name, along with the starting and ending indices are declared. In the middle one, the data type, name, and the length are the user's emphasis and finally, in the rightmost pattern, the declaration focuses on the data type, name, as well as the starting and ending indices.

u[1...8]

int u[9]

int u[1...8]

8.1. Example. The mean M and the variance V for n numbers are calculated using the following formulae.

$$M = \frac{\sum_{i=1}^n x_i}{n}, \quad V = \frac{\sum_{i=1}^n x_i^2 - nM^2}{n-1}.$$

Write an algorithm to read an integer $n > 1$. Then, read n real numbers and calculate and print their mean and variance.

Solution. We first write an algorithm without using the arrays. To do this, denote the number which is supposed to be read each time as x and use the variables $sum1$ and $sum2$ for the repetitive sums of the x and the x^2 , respectively.

Use a for template and assign 0 as the initial value of $sum1$ and $sum2$ before the loop. At each repetition of the loop we should calculate the repetitive sums $sum1$ and $sum2$ after reading x . Then, calculate and print M and V upon exiting the loop. These discussions are displayed in the form of an algorithm in [Flowchart 8.1\(a\)](#).

Programs P7_8_A indicate the codes of [Flowchart 8.1\(a\)](#).

C++ codes:

```
// Program P8_1_A to calculate the
// mean and variance of n numbers
// without using the arrays
#include <iostream>
using namespace std;
int main() {
    double x, sum1, sum2, M, V;
    int i, n;
    cout<<"Enter an integer n>1: ";
    cin>>n;
```

Java codes:

```
// Program P8_1_A to calculate the
// mean and variance of n numbers
// without using the arrays
import java.util.Scanner;
class P8_1_A {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        double x, sum1, sum2, M, V;
        int i, n;
        System.out.print("Enter an integer n>1: ");
```

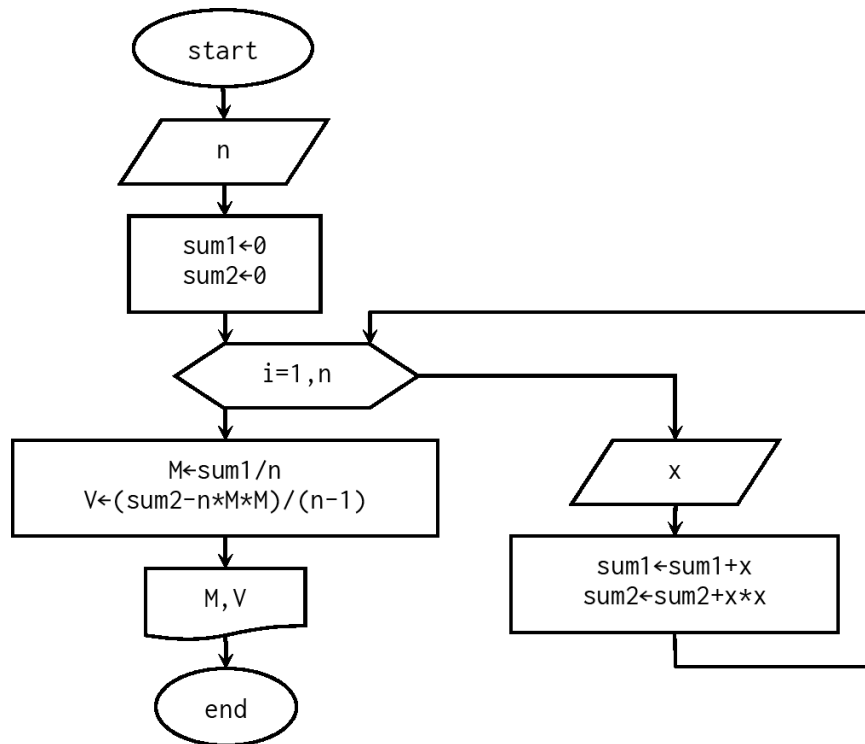


Fig. 8.1(a): Calculating the mean and variance without using the arrays.

```

sum1=sum2=0;
for (i=1; i<=n; i++) {
    cout<<"Enter a numbers: ";
    cin>>x;
    sum1+=x;
    sum2+=x*x;
}
M=sum1/n;
V=(sum2-n*M*M)/(n-1);
cout<<"Mean: "<<M<<" , Variance: "
<<V;
return 0;
}

n=read.nextInt();
sum1=sum2=0;
for (i=1; i<=n; i++) {
    System.out.print("Enter a numbers: ");
    x=read.nextDouble();
    sum1+=x;
    sum2+=x*x;
}
M=sum1/n;
V=(sum2-n*M*M)/(n-1);
System.out.print("Mean: " + M
+ " , Variance: " + V);
read.close();
}
}
  
```

Input/output:

```

Enter an integer n>1: 4↵
Enter a numbers: 14.5↵
Enter a numbers: 25.5↵
Enter a numbers: 89↵
Enter a numbers: 50↵
Mean: 44.75, Variance: 1090.42
  
```

Input/output:

```

Enter an integer n>1: 4↵
Enter a numbers: 14.5↵
Enter a numbers: 25.5↵
Enter a numbers: 89↵
Enter a numbers: 50↵
Mean: 44.75, Variance: 1090.4166666666667
  
```

Now, arrays are used to write this algorithm. To achieve this, an array named x is defined with 100 hypothetical entries (a hypothetical length of 101) and the indexed variable entries of this array are applied in the calculation. Accordingly, **Flowchart 8.1(b)** is obtained.

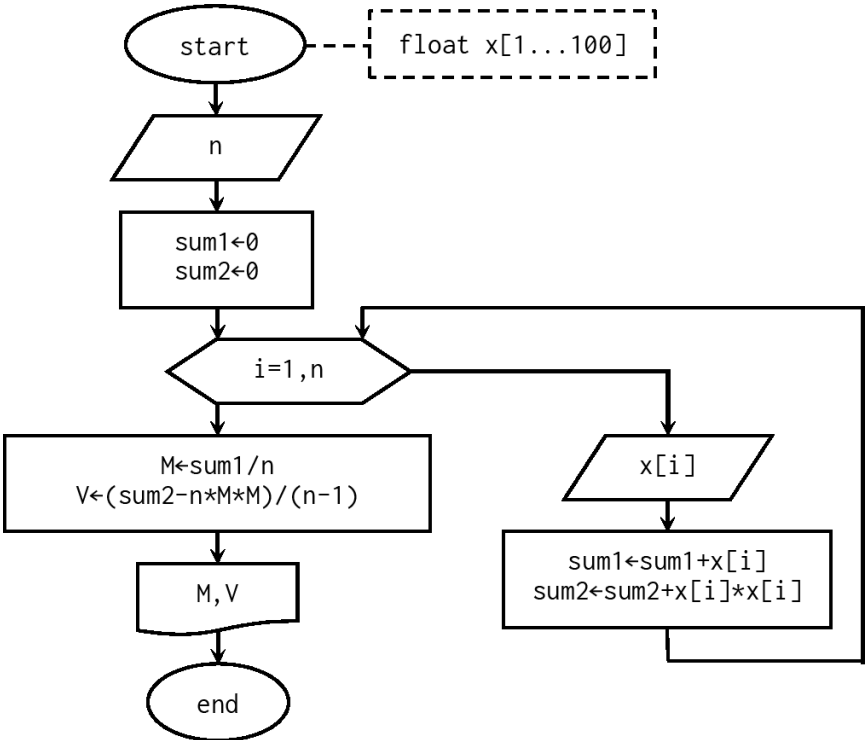


Fig. 8.1(b): Calculating the mean and variance using the arrays.

Maybe one objects that the appearance of the flowchart is now a little complicated! Therefore, what is the benefit of using the arrays? Well, a similar initial idea which is used to write the algorithm without using the arrays may not always occur in everyone’s mind. Further, the speed of running the program is normally higher while less space is occupied in the memory when using the arrays. Furthermore, writing the algorithm without using the arrays is often difficult when facing a large number of data of the same type. For example, can you write an algorithm m) without using the arrays to read two 100-component vectors, add them (componentwise), and store the result of this vector

addition in the third 100-component vector? How many variables are used? How many assignment statements?

Programs P8_1_B are related to **Flowchart 8.1(b)**.

C++ codes:

```
// Program P8_1_B to calculate the
// mean and variance of n numbers
// using the arrays
#include <iostream>
using namespace std;
int main() {
    double x[101], sum1, sum2, M, V;
    int i, n;
    cout<<"Enter an integer n>1: ";

    cin>>n;
    sum1=sum2=0;
    for (i=1; i<=n; i++) {
        cout<<"Enter a numbers: ";
        cin>>x[i];
        sum1+=x[i];
        sum2+=x[i]*x[i];
    }
    M=sum1/n;
    V=(sum2-n*M*M)/(n-1);
    cout<<"Mean: "<<M<<" , Variance: "
    <<V;

    return 0;
}
```

Input/output:

```
Enter an integer n>1: 4↵
Enter a numbers: 14.5↵
Enter a numbers: 25.5↵
Enter a numbers: 89↵
Enter a numbers: 50↵
Mean: 44.75, Variance: 1090.42
```

Java codes:

```
// Program P8_1_B to calculate the
// mean and variance of n numbers
// using the arrays
import java.util.Scanner;
class P8_1_B {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        double x[]=new double[101];
        double sum1, sum2, M, V;

        int i, n;
        System.out.print("Enter an integer n>1: ");
        n=read.nextInt();
        sum1=sum2=0;
        for (i=1; i<=n; i++) {
            System.out.print("Enter a numbers: ");
            x[i]=read.nextDouble();
            sum1+=x[i];
            sum2+=x[i]*x[i];
        }
        M=sum1/n;
        V=(sum2-n*M*M)/(n-1);
        System.out.print("Mean: " + M
            + " Variance: " + V);
        read.close();
    }
}
```

Input/output:

```
Enter an integer n>1: 4↵
Enter a numbers: 14.5↵
Enter a numbers: 25.5↵
Enter a numbers: 89↵
Enter a numbers: 50↵
Mean: 44.75, Variance: 1090.4166666666667
```

Note that, at least two inputs should be entered when running these programs. Otherwise, we encounter the “division by zero” error in

C++ and NaN (not a number) result in Java when calculating the variance.

The next example confirms the fact that it is occasionally difficult to write the algorithm without employing the arrays.

8.2. Example. Write an algorithm to read 20 integer entries of an array and put them reversely (from the end to the start) in another array. Then, print the two arrays separately.

Solution. We arrange the algorithm so that it reads 20 entries of an array named a by a for template. Then, the algorithm puts these entries reversely in another array named b by another for template.

To this end, we only need to assign a_i to b_{21-i} a_i to b_{21-i} in the range of the loop. Finally, the algorithm prints both arrays separately using two for templates. The obtained flowchart is displayed in **Figure 8.2(a)** with the Programs P8_2 in C++ and Java codes.

Of course, as shown in **Flowchart 8.2(b)**, the two first loops can be merged. Programs P8_2 depict **Flowchart 8.2(b)** in both C++ and Java codes.

C++ codes:

```
// Program P8_2 to put the entries of  
// an array to another array reversely
```

Java codes:

```
// Program P8_2 to put the entries of  
// an array to another array reversely
```

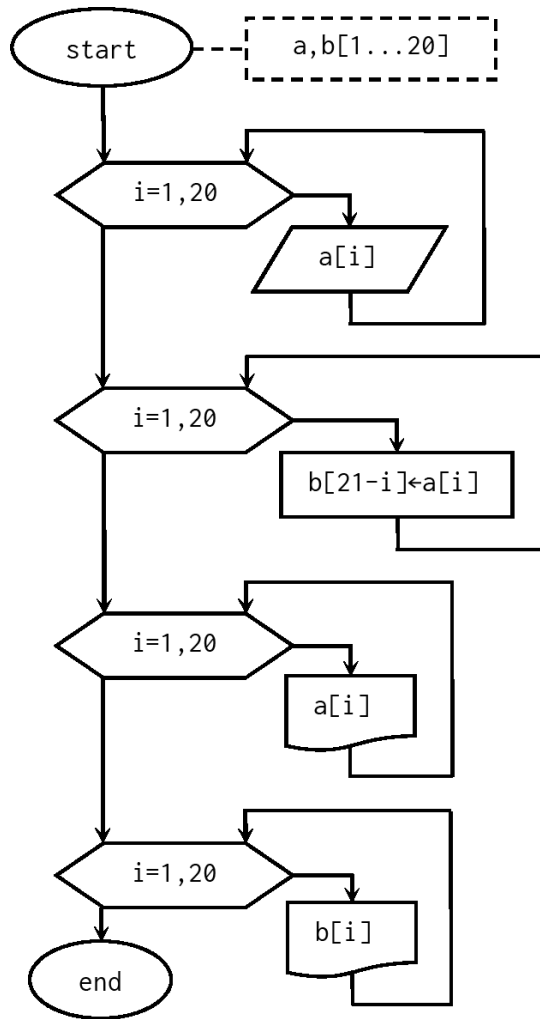


Fig. 8.2(a): Reversely putting the entries of an array in another one.

```

#include <iostream>
using namespace std;
int main() {
    int a[21], b[21], i;
    cout<<"Enter the array a: \n";
    for (i=1; i<=20; i++) {
        cin>>a[i];
        b[21-i]=a[i];
    }
    cout<<"The array a is: \n";
    for (i=1; i<=20; i++)
        cout<<a[i]<<" ";
    cout<<"\nThe array b is: \n";
    for (i=1; i<=20; i++)
        cout<<b[i]<<" ";
    return 0;
}

```

```

import java.util.Scanner;
class P8_2 {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int a[]=new int[21];
        int b[]=new int[21];
        int i;
        System.out.println("Enter the array a: ");
        for (i=1; i<=20; i++) {
            a[i]=read.nextInt();
            b[21-i]=a[i];
        }
        System.out.println("The array a is: ");
        for (i=1; i<=20; i++)
            System.out.println(a[i] + " ");
        System.out.println("\nThe array b is: ");
        for (i=1; i<=20; i++)
            System.out.print(b[i] + " ");
        read.close();
    }
}

```

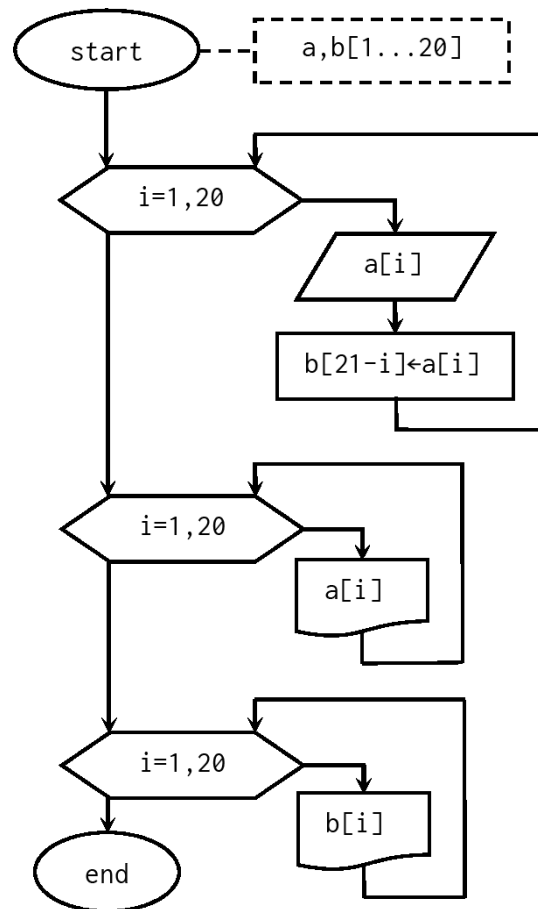
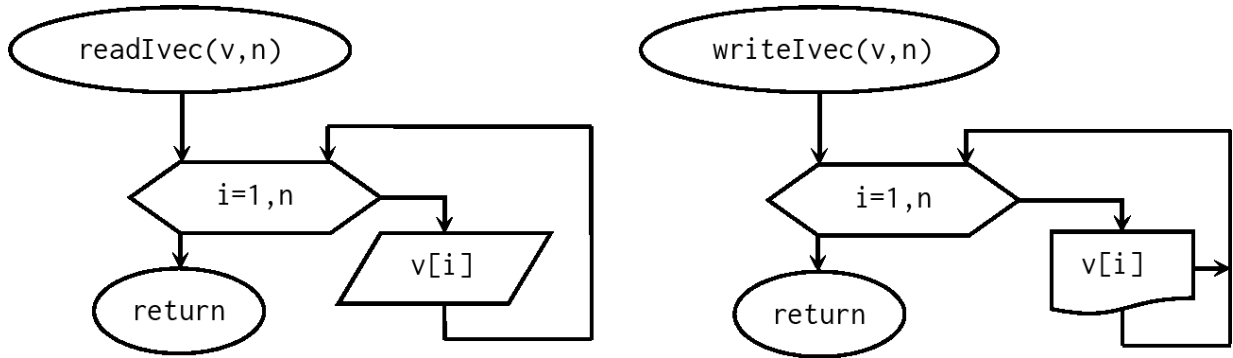



Fig. 8.2(b): Merging the two first arrays in [Flowchart 8.2\(a\)](#).

As displayed in [Flowcharts 8.2\(a\)](#) and [8.2\(b\)](#), printing the array using a for template is repeated two times in Programs P8_2. From now on, sub-algorithms (subprograms) are employed for reading and printing the entries of the arrays (vectors) in order to reduce the time and size of writing algorithms (programs) involved with the arrays. The following flowcharts illustrate the sub-algorithms for reading and writing the integer vectors having n entries started from index 1. We refer to these sub-algorithms as readIvec() and writeIvec() (I for int).



Additionally, similar sub-algorithms named `readFvec()` and `writeFvec()` (F for float) will be used in the case of real entries (The same can be written for double data type). The subprograms of these sub-algorithms are as follows.

C++ codes for readIvec:

```
void readIvec(int u[], int n) {
    for (int i=1; i<=n; i++)
        cin>>u[i];
    return;
}
```

C++ codes for readFvec:

```
void readFvec(float u[], int n) {
    for (int i=1; i<=n; i++)
        cin>>u[i];
    return;
}
```

C++ codes for writeIvec:

```
void writeIvec(int u[], int n) {
    for (int i=1; i<=n; i++)
        cout<<u[i]<<" ";
    return;
}
```

C++ codes for writeFvec:

```
void writeFvec(float u[], int n) {
    for (int i=1; i<=n; i++)
        cin>>u[i];
    return;
}
```

Java codes for readIvec:

```
static void readIvec(int u[], int n) {
    Scanner read=new Scanner(System.in);
    for (int i=1; i<=n; i++)
        u[i]=read.nextInt();
    read.close();
}
```

Java codes for readFvec:

```
static void readFvec(float u[], int n) {
    Scanner read=new Scanner(System.in);
    for (int i=1; i<=n; i++)
        u[i]=read.nextFloat();
    read.close();
}
```

Java codes for writeIvec:

```
static void writeIvec(int u[], int n) {
    for (int i=1; i<=n; i++)
        System.out.print(u[i] + " ");
}
```

Java codes for writeFvec:

```
static void writeFvec(float u[], int n) {
    for (int i=1; i<=n; i++)
        System.out.print(u[i] + " ");
}
```



We can write the length of a one-dimensional array in the open form[] when using it as the parameter of a subprogram. More importantly, the number of entries of the array, which is one less than the real length when starting with index 1, should appear as one of the parameters of the subprogram. Eventually, depending on the need, select an appropriate printing format when using the subprograms writeIvec and writeFvec.

Consider the codes of the above-mentioned readIvec() method. If we use this method more than once, the error is encountered when inputting the data. The reason is that in the first using of readIvec() method, the facility for inputting the data (of int type) is closed by executing the statement

```
read.reset();
```

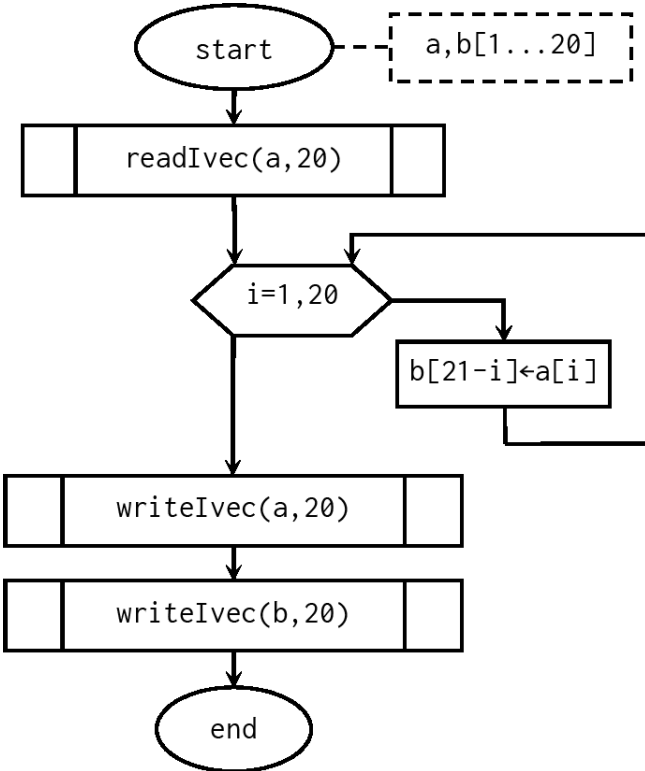


Fig. 8.2(c): Redrawing Flowchart 8.2(a) using the subprograms readIvec() and writeIvec().

at the end of the method. One way to resolve this problem is to use the hasNext() built-in method as follows.

```

static void readIvec(int u[], int n) {
    Scanner read = new Scanner(System.in);
    for (int i=1; i<=n; i++) {
        if (read.hasNext())
            u[i] = read.nextInt();
        read.reset();
    }
}

```

The **hasNext()** is a built-in method of Java Scanner class which returns true if this scanner has another token in its input.

Nevertheless, there are other ways which can be found in the following rule.



The rule of multi-using the reading methods. There are three ways to prevent any input error when we use a reading method with the read.close() statement at the end:

1. Using the hasNext() built-in method as explained above.
2. Inactivating the read.close() statement by adding double slashes:

```
// read.close();
```

3. Changing the read.close() statement to the the read.reset() statement for resetting the Java Scanner class.

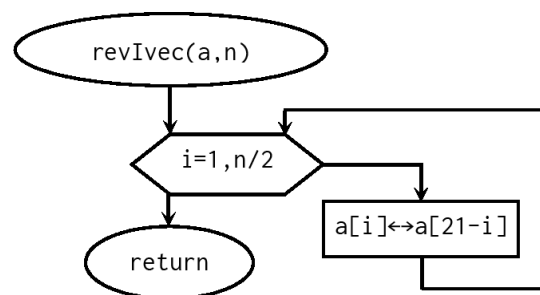


Fig. 8.3: Reversing the positions of the entries of an array.

4. Transferring the read.close() statement to another appropriate place where there is no reading item

As displayed in **Figure 8.2(c)**, **Flowchart 8.2(a)** can now be written in the simple form.

8.2.1. Exercise. Write the programs of **Flowchart 8.2(c)**.

8.3. Example. Write a sub-algorithm, named `revive()`, to receive the n -entry integer array a , reverse the position of entries (from the end to the beginning) and return the reversed array a .

Solution. The method which we are going to use is to swap the values of the entries a_i and a_{21-i} in pairs, appealing to the swap algorithm. The entries a_i and a_{21-i} should be swapped in the range of a for template with the specification $i=1, n/2$ due to the nature of swapping in pairs. It is noteworthy that one may write the similar subprogram named `revFvec()` for the arrays of float types. It suffices to change `int` to `float`.

Flowchart 8.3 illustrates the above discussion. Programs `P8_3` read a 20-entry integer array and then print the reversed array using the sub-algorithm `revive()`.

Question. What would be the result if we wrote n instead of $n / 2$ as the final value of the loop?

Answer. No task is performed since the reversion is repeated.

C++ codes:

```
// Program P8_3 to reverse the
// order of the entries of an array
// without using any other array
#include <iostream>
using namespace std;
void writeIvec(int u[], int n) {
    int i;
    for (i=1; i<=n; i++)
        cout<<u[i]<<" ";
```

Java codes:

```
// Program P8_3 to reverse the
// order of the entries of an array
// without using any other array
import java.util.Scanner;
class P8_3 {
    static void writeIvec(int u[], int n) {
        for (int i=1; i<=n; i++)
            System.out.print(u[i] + " ");
        System.out.println();
```

```

    cout<<endl;
    return;
}
//*****
void readIvec(int u[], int n) {
    for (int i=1; i<=n; i++)
        cin>>u[i];
    return;
}
//*****
void revIvec(int a[], int n) {
    int i, t;
    for (i=1; i<=10; i++) {
        t=a[i];
        a[i]=a[21-i];
        a[21-i]=t;
    }
}
//*****
int main() {
    int a[21];
    cout<<"Enter the array a:"<<endl;
    readIvec(a, 20);
    revIvec(a, 20);
    cout<<"The reordered array: "<<endl;
    writeIvec(a, 20);
    return 0;
}

}
//*****
static void readIvec(int u[], int n) {
    Scanner read=new Scanner(System.in);
    for (int i=1; i<=n; i++) {
        u[i]=read.nextInt();
    }
    read.close();
}
//*****
static void revIvec(int a[], int n) {
    int i, t;
    for (i=1; i<=10; i++) {
        t=a[i];
        a[i]=a[21-i];
        a[21-i]=t;
    }
}
//*****
public static void main(String[] args) {
    Scanner read=new Scanner(System.in);
    int a[]=new int[21];
    System.out.println("Enter the array a: ");
    readIvec(a, 20);
    revIvec(a, 20);
    System.out.println("The reordered array : ");
    writeIvec(a, 20);
    read.close();
}
}
}

```



The rule of calling the array-return methods. The number of the entries of the array should be determined when calling a subprogram which returns an array. In addition, the length of the array should be identified while declaring an array in the call unit.

8.4. Example. Write an algorithm to read 20 entries of an integer array named *a*. Then, put the negative and none-negative entries in the arrays named *b* and *c*, respectively. Finally, print the entries of the arrays *b* and *c*.

Solution. Since it is not clear how many entries b and c have, we will declare them as 20-entry arrays. First, the array a is read using the sub-algorithm readIvev(). The way we use is to take an entry a_i in each repetition of a for template with the specification $i=1,20$. Next, we put a_i in b if it is negative; otherwise we put it in c . In which index of b or c do we put the entry a_i in each case? For this purpose, two *index maker* variables nb and nc are employed to make indices for b and c , respectively. Further, we assign 0 to these variables before the loop in order to use them. Then, we first make an index in the involved case by increasing the previous value of the related index maker variable by one unit and then put the taken a_i in the index just made. The obtained flowchart is displayed in **Figure 8.4**. Programs P8_4 translate this algorithm into C++ and Java codes.

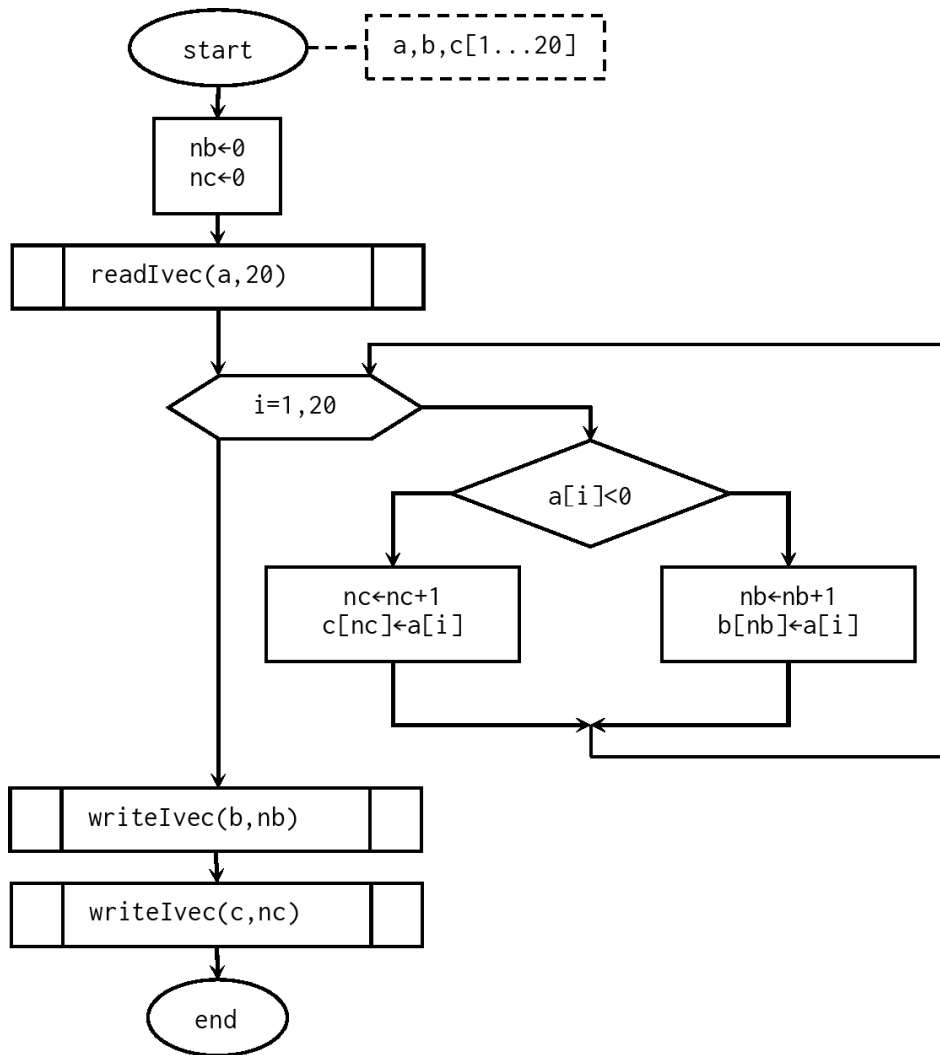


Fig. 8.4: Putting the negative and non-negative entries of a in b and c , respectively.

C++ codes:

```
// Program P8_4 to put the negative
// and non-negative entries of an
// array in two separate arrays
#include <iostream>
using namespace std;
void writeIvec(int u[], int n) {
    for (int i=1; i<=n; i++)
        cout<<u[i]<<" ";
    cout<<endl;
    return;
}
//*****
void readIvec(int u[], int n) {
    for (int i=1; i<=n; i++)
        cin>>u[i];
    return;
}
//*****

int main() {
    int nb, nc, a[21], b[21], c[21];
    nb=nc=0;
    cout<<"Enter the array a: \n";
    readIvec(a, 20);
    for (int i=1; i<=20; i++)
        if (a[i]<0) {
            nb+=1;
            b[nb]=a[i];
        }
        else {
            nc+=1;
            c[nc]=a[i];
        }
    cout<<"The array b: \n";
    writeIvec(b, nb);
    cout<<"The array c: \n";
    writeIvec(c, nc);
    return 0;
}
```

Java codes:

```
// Program P8_4 to put the negative
// and non-negative entries of an
// array in two separate arrays
import java.util.Scanner;
class P8_4 {
    static void writeIvec(int u[], int n) {
        for (int i=1; i<=n; i++)
            System.out.print(u[i] + " ");
        System.out.println();
    }
    //*****
    static void readIvec(int u[], int n) {
        Scanner read=new Scanner(System.in);
        for (int i=1; i<=n; i++) {
            u[i]=read.nextInt();
        }
        read.close();
    }
    //*****
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int nb, nc;
        int a[]=new int[21];
        int b[]=new int[21];
        int c[]=new int[21];
        nb=nc=0;
        System.out.println("Enter the array a: ");
        readIvec(a, 20);
        for (int i=1; i<=20; i++)
            if (a[i]<0) {
                nb+=1;
                b[nb]=a[i];
            }
            else {
                nc+=1;
                c[nc]=a[i];
            }
        System.out.println("The array b: ");
        writeIvec(b, nb);
        System.out.println("The array c: ");
        writeIvec(c, nc);
        read.close();
    }
}
```

8.4.1. Exercise. Modify **Flowchart 8.4** and Programs 8_4 so that the entries of *a* are read inside the used for loop instead of reading the array *a* using the subprogram readIvec().

8.5. Example. In Flowchart 1.8(c), we first read the first number and assigned it to the variable *max* to find the maximum number between 50 read integers. Next, we performed the necessary process using a do-while loop. Do the same tasks using the arrays. Employ the for template instead of the do-while template.

Solution. Given the generality of this algorithm in mind, **Flowchart 8.5 (a)** can easily be drawn. Programs P8_5_A are related to **Flowchart 8.5 (a)**.

C++ codes:

```
// Program P8_5_A to determine the
// maximum element of an array
#include <iostream>
using namespace std;
void readIvec(int [], int);
int main() {
    int u[51], i, max;
    cout<<"Enter the array: ";
    readIvec(u, 50);
    max=u[1];
    for (i=2; i<=50; i++)
```

Java codes:

```
// Program P8_5_A to determine the
// maximum element of an array
import java.util.Scanner;
class P8_5_A {
    public static void main(String[] args) {
        int u[]=new int[51];
        int i, max;
        System.out.println("Enter the array: ");
        readIvec(u, 50);
        max=u[1];
        for ( i=1; i<=50; i++)
```

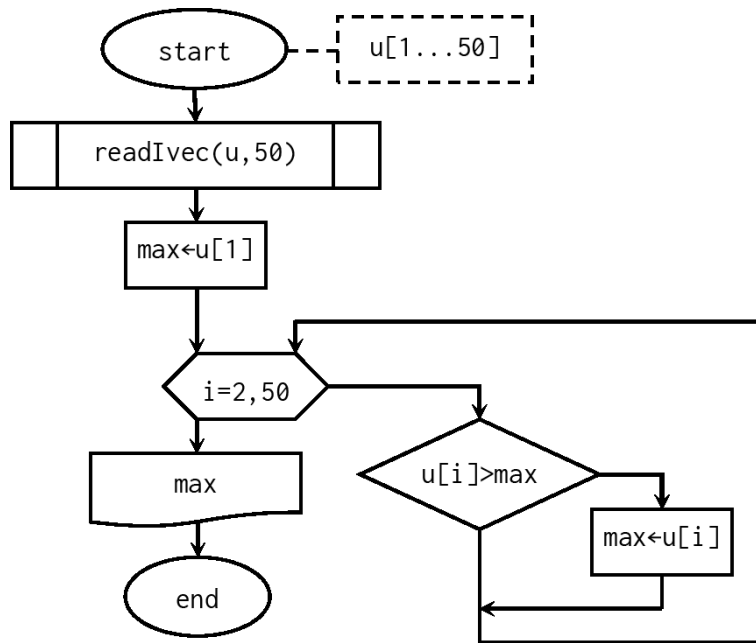


Fig. 8.5(a): Calculating the maximum of 50 numbers.

```

    if (u[i]>max)
        max=u[i];
    cout<<"Maximum is: "<<max;
    return 0;
}
//*****
void readIvec(int u[], int n) {
    for (int i=1; i<=n; i++)
        cin>>u[i];
    return;
}

```

```

    if (u[i]>max)
        max=u[i];
    System.out.println("Maximum is: " + max);
}
//*****
static void readIvec(int u[], int n) {
    Scanner read=new Scanner(System.in);
    for (int i=1; i<=n; i++) {
        u[i]=read.nextInt();
    }
    read.close();
}
}

```

As shown in **Flowchart 8.5(b)**, **Flowchart 8.5(a)** is divided into a function named `max()` which is responsible for finding the maximum member and a main algorithm.

8.5.1. Exercise. Write a similar function named `min()` for finding the minimal entries of the received array u .

Programs P8_5_B are the translation of **Flowchart 8.5(b)** into C++ and Java codes.

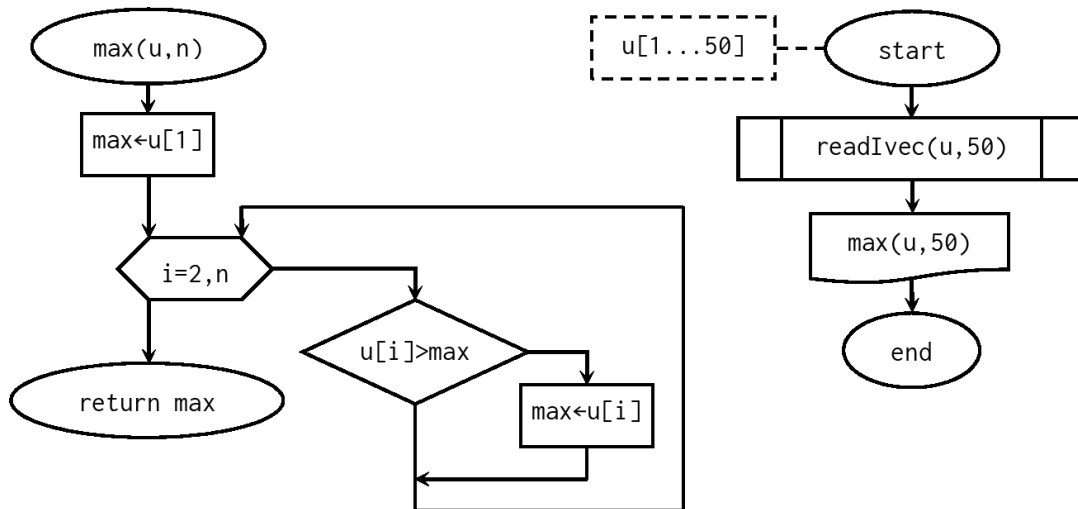


Fig. 8.5(b): A function to calculate the maximum of an n -entry array and a main algorithm.

C++ codes:

```

// Program P8_5_B to determine the
// maximum member of an array,
// using the max() function
#include <iostream>
using namespace std;
void readIvec(int [], int);
int max(int [], int);
int main() {
    int n, u[51];
    cout<<"Enter the size of array: ";
    cin>>n;
    cout<<"Now enter the array: \n";
    readIvec(u, n);
    cout<<"Maximum is: "<<max(u, n);
}
//*****
void readIvec(int u[], int n) {
    for (int i=1; i<=n; i++)
        cin>>u[i];
    return;
}
//*****
int max(int u[], int n) {
    int max=u[1];
    for (int i=2; i<=n; i++)
        if (u[i]>max)
            max=u[i];
    return max;
}

```

Input/output:

Enter the size of array: 5↵

Java codes:

```

// Program P8_5_B to determine the
// maximum member of an array,
// using the max() method
import java.util.Scanner;
class P8_5_B {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int n, u[]=new int[51];
        System.out.print("Enter the size of array: ");
        n=read.nextInt();
        System.out.print("Now enter the array: ");
        readIvec(u, n);
        System.out.print("Maximum is: " + max(u, n));
        read.close();
    }
    //*****
    static void readIvec(int u[], int n) {
        Scanner read=new Scanner(System.in);
        for (int i=1; i<=n; i++) {
            u[i]=read.nextInt();
        }
        read.close();
    }
    //*****
    static int max(int u[], int n) {
        int max=u[1];
        for (int i=2; i<=n; i++)
            if (u[i]>max)
                max=u[i];
        return max;
    }
}

```

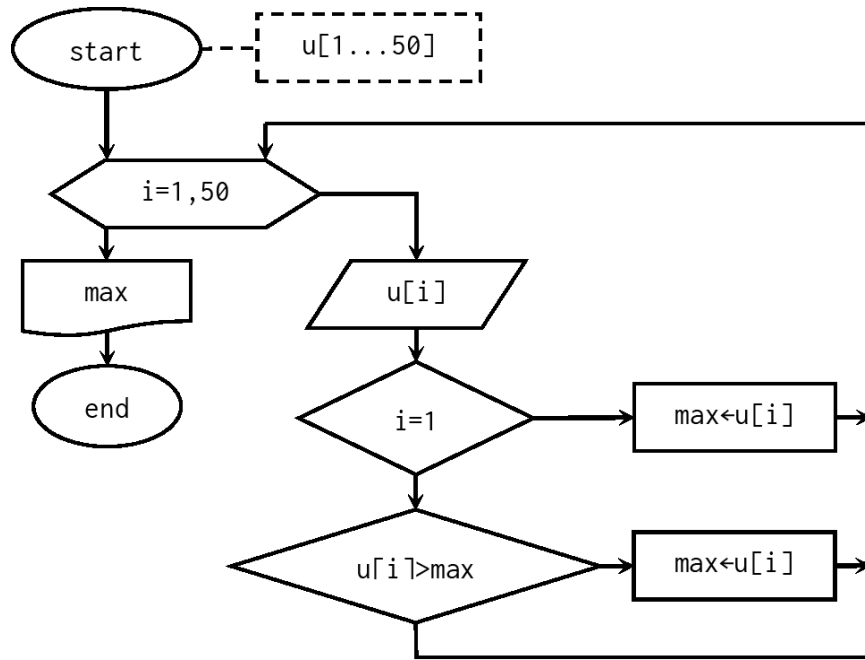


Fig. 8.5(c): Combining the comparing and assigning tasks of flowchart 8.5(a).

Now enter the array:
 2 -5 9 -12 4↵
 Maximum is: 9

Input/output:

Enter the size of array: 5↵
 Now enter the array:
 2 -5 9 -12 4↵
 Maximum is: 9

In the algorithm of Figure 8.5(a), the entries of the array were first read by the sub-algorithm readIvec(). Then, the first entry was assigned to the variable *max* and the comparing and assigning were processed by a for template. All these processes can be combined in a single for template (Fig. 8.5(c)).

Since the F-paths of the two conditions in Flowchart 8.5(c) are the same, we can modify Flowchart 8.5(c) to Flowchart 8.5(d) appealing to the rule of merging the conditions by the || operator in Chapter 4.

8.5.2. Exercise. Write the programs of Flowcharts of 8.5(c) and 8.5(d) in both C++ and Java codes.

8.5.3. Exercise. Modify each of **Flowcharts 8.5(a), 8.5(c)** and **8.5(d)** in such a way that they print the location of the maximum and the maximum itself. In the case where the maximum is repeated, print the first location of the maximum. Finally, write the programs in both C++ and Java codes.

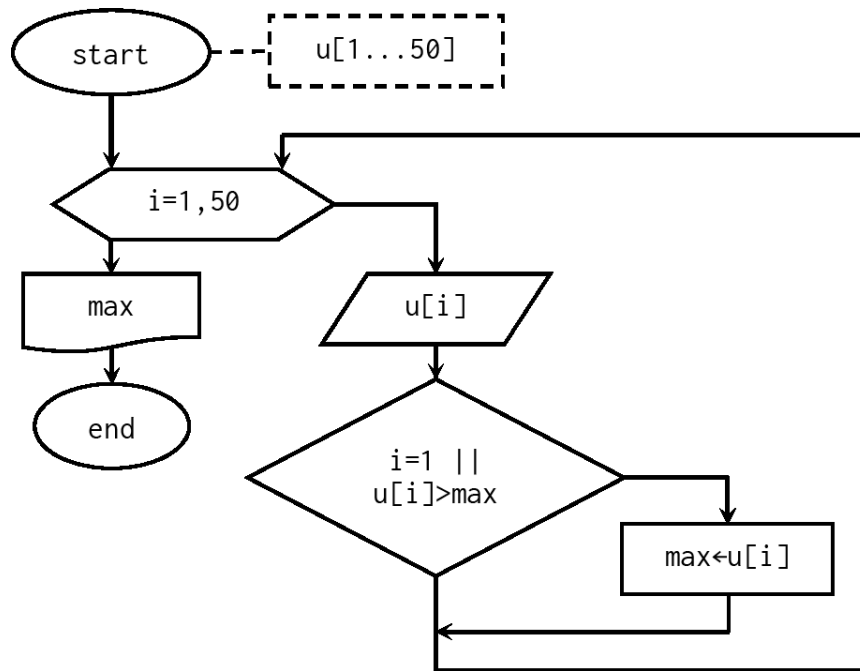


Fig. 8.5(d): Merging the conditions of **Flowchart 8.5(c)** using the `||` operator.

8.5.4. Exercise. Repeat **Exercise 8.5.3** with the difference that print all the locations if the maximum is repeated.

8.6. Example. Write an algorithm to read the identification number (ID) and grade of 40 students of a class one by one. Then, read an ID number and print the grade of that student if it is between the 40 read ID numbers. Otherwise, print the message Not in list.

Solution. First the ID numbers and grades of the students are read and saved in the arrays *N* and *G*, respectively. Then, the intended number by the name *m* is read and searched for among the entries of the array *N* using the switch variable *s* with the initial value of 0 (switch off). Next, 1 to *s* is assigned (switch on) in the range of a for loop and the

index i is instantly stored in the memory with the name j upon matching m with an ID number N_i .

Finally, after exiting the loop, the quoted message is printed if the switch is off; otherwise, the grade of the student corresponding to the number N_j , that is G_j , is printed. The result of the above process is illustrated in **Flowchart 8.6(a)**.

Question. Suppose that we take out the instruction $j \leftarrow i$ and then print G_i instead of G_j after exiting the loop. What is printed? Find the answer by examining.

Programs P8_6_A are the translations of **Flowchart 8.6(a)** into both C++ and Java codes.

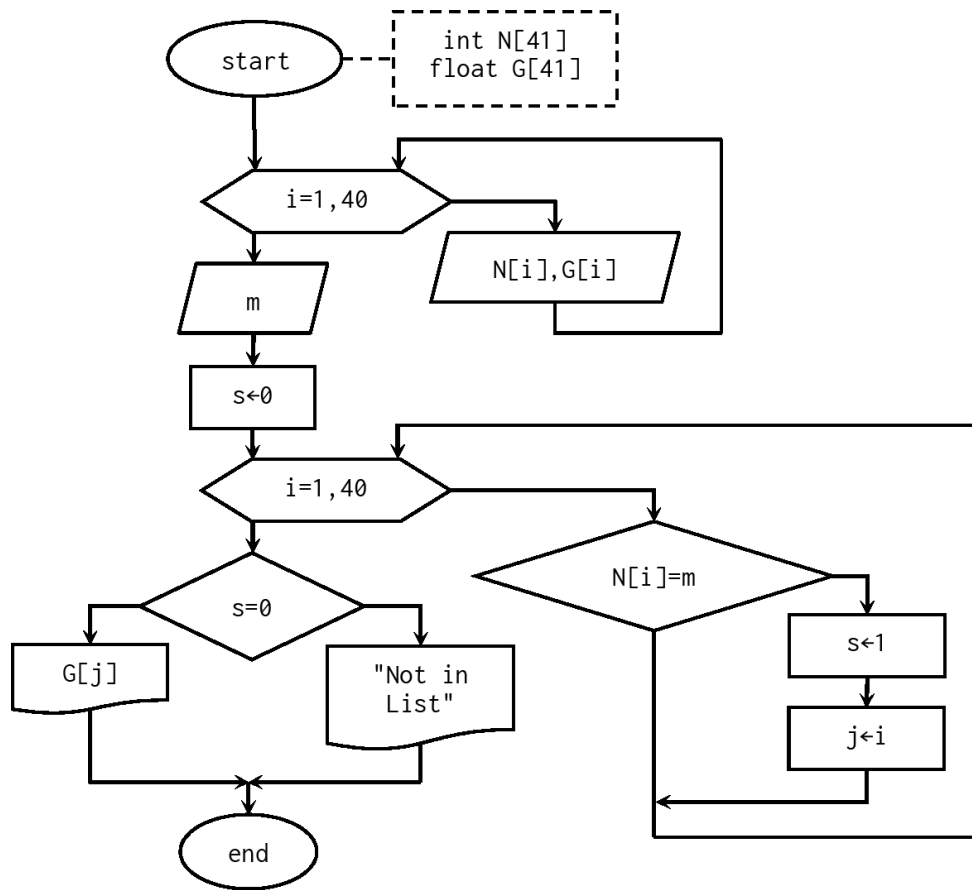


Fig. 8.6(a): Searching a specific student among 40 students and printing its average.

C++ codes:

```
// Program P8_6_A to search a
// student among 40 student and
// print the corresponding
// grade, if exists
#include <iostream>
using namespace std;
int main() {
    int N[41];
    float G[41];
    int i, j, s, m;
    for (i=1; i<=40; i++) {
        cout<<"Enter ID and grade: ";
        cin>>N[i]>>G[i];
    }
    cout<<"Enter the ID for search: ";
    cin>>m;
    s=0;
    for (i=1; i<=40; i++)
        if (N[i]==m) {
            s=1;
            j=i;
        }

    if (s==0)
        cout<<"The ID is not in the list";
    else
        cout<<"The grade of "<<N[j]
            <<" is: "<<G[j];
    return 0;
}
```

Input/output for four students:

```
Enter the size of array: 5↵
Enter ID and grade: 19221003 12.5↵
Enter ID and grade: 19221004 9↵
Enter ID and grade: 19221006 18.75↵
Enter ID and grade: 19221009 14.5↵
Enter the ID for search: 19221006↵
The grade of 19221006 is: 18.75
```

Java codes:

```
// Program P8_6_A to search a student
// among 40 student and print the
// corresponding grade, if exists
import java.util.Scanner;
class P8_6_A {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int N[]=new int[41];
        float G[]=new float[41];
        int i, j=0, s, m;
        for (i=1; i<=40; i++) {
            System.out.print("Enter ID and grade: ");
            N[i]=read.nextInt();
            G[i]=read.nextFloat();
        }
        System.out.print("Enter the ID for "
            + "search: ");
        m=read.nextInt();
        s=0;
        for (i=1; i<=40; i++)
            if (N[i]==m) {
                s=1;
                j=i;
            }
        if (s==0)
            System.out.print("The ID is not in "
                + "the list ");
        else
            System.out.print("The grade of " + N[j]
                + " is: " + G[j]);
        read.close();
    }
}
```

Input/output for four students:

```
Enter the size of array: 5↵
Enter ID and grade: 19221003 12.5↵
Enter ID and grade: 19221004 9↵
Enter ID and grade: 19221006 18.75↵
Enter ID and grade: 19221009 14.5↵
Enter the ID for search: 19221006↵
The grade of 19221006 is: 18.75
```

It is noteworthy that initializing the variable to 0 is necessary for the right Java program.

There is a simple way to solve this example: in the range of the for loop, print the grade of the student and terminate the algorithm upon matching the number m with an ID number N_i . N_i . Otherwise, the quoted message is printed after exiting the loop. The result is **Flowchart 8.6(b)** and Programs P8_6_B.

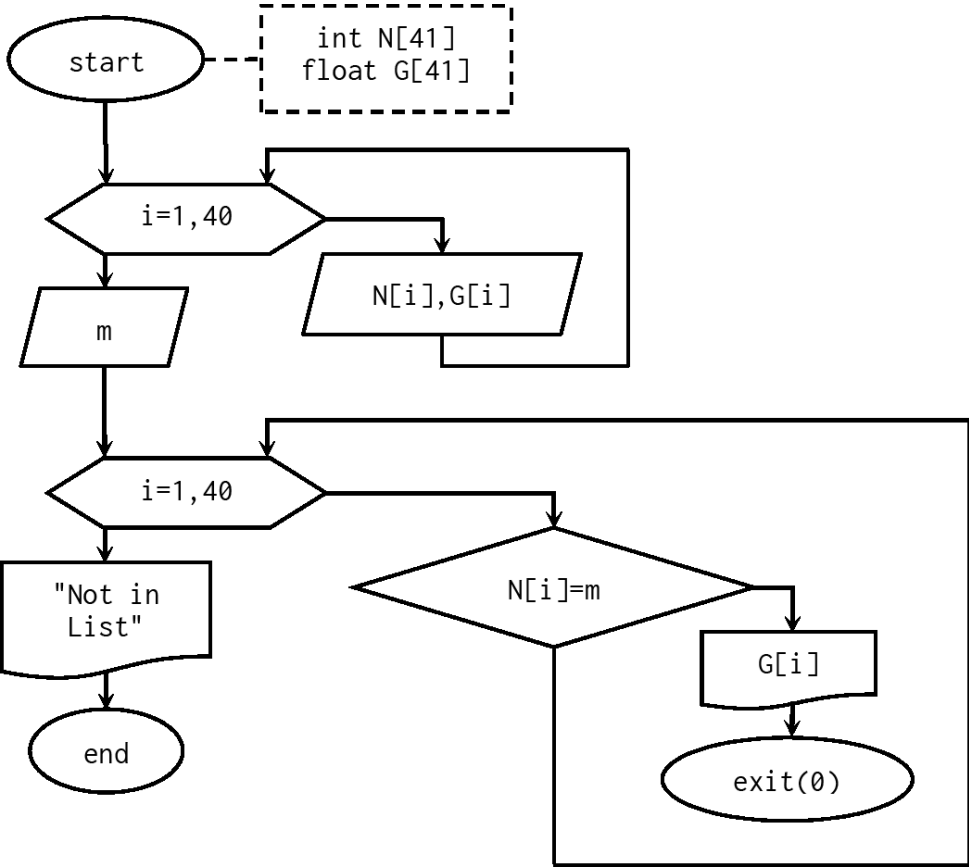


Fig. 8.6(b): A simple way for Example 8.6 instead of Flowchart 8.6(a).

C++ codes:

```

// Program P8_6_B to search a
// student among 40 student and
// print the corresponding
// grade, if exists
#include <iostream>
#include <stdlib.h>
using namespace std;
int main() {
    int N[41];
    float G[41];
    int i, m;
    for (i=1; i<=40; i++) {
        cout<<"Enter ID and grade: ";
        cin>>N[i]>>G[i];
    }
    cout<<"Enter the ID for search: ";
    cin>>m;
    for (i=1; i<=40; i++)
        if (N[i]==m) {
            cout<<"The grade of "<<N[i]
                <<" is: "<<G[i];
            exit(0);
        }
    cout<<"The ID is not in list";
    return 0;
}

```

Java codes:

```

// Program P8_6_B to search a student
// among 40 student and print the
// corresponding grade, if exists
import java.util.Scanner;
class P8_6_B {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int N[]=new int[41];
        float G[]=new float[41];
        int i, m;
        for (i=1; i<=40; i++) {
            System.out.print("Enter ID and grade: ");
            N[i]=read.nextInt();
            G[i]=read.nextFloat();
        }
        System.out.print("Enter the ID for search: ");
        m=read.nextInt();
        for (i=1; i<=40; i++)
            if (N[i]==m) {
                System.out.print("The grade of " + N[i]
                    + " is: " + G[i]);
                System.exit(0);
            }
        System.out.print("The ID is not in list ");
        read.close();
    }
}

```

8.7. Example. Write a function named `search()` to receive an integer m and the n entries of the integer array named a . Then, it returns 1 if m exists among the entries of a ; otherwise, it returns 0. Afterwards, write a main algorithm to read the ID number and grade of 40 students of a class. Next, read a single ID number and print the message `Founded` if it is between the 40 students; otherwise, print the message `Not founded`. One may return the Boolean literals `true` or `false` instead of 1 and 0, respectively. This function may be modified to any data type of the m and the array a .

Solutions. We use a switch, say s . First, turn the switch off (assign 0 to s). Then, in the range of a for template, turn the switch on (assign 1 to

s) upon matching m with an entry a_i . a_i . Finally, return the value of s after exiting the loop. These processes are all summarized in **Flowchart 8.7(a)**.

The main algorithm in **Figure 8.7(b)** is as required. Programs P8_7 combined both **flowcharts 8.7(a)** and **8.7(b)** in the literature of codes.

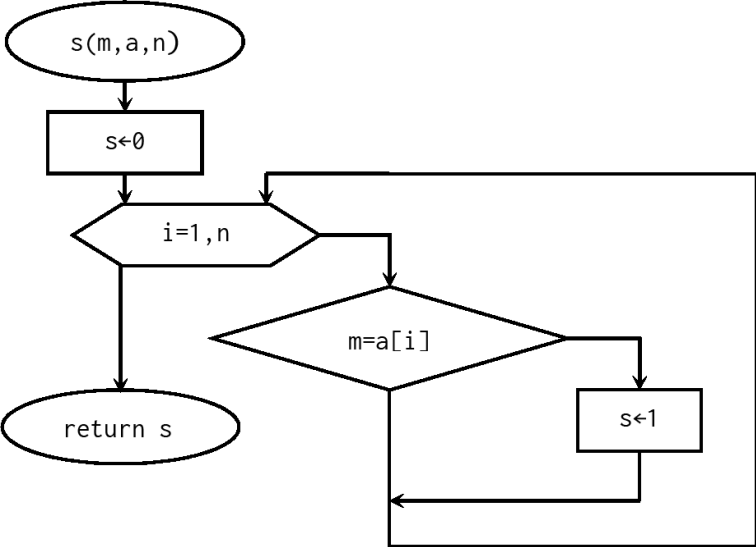


Fig. 8.7(a): Searching a number among the entries of an array.

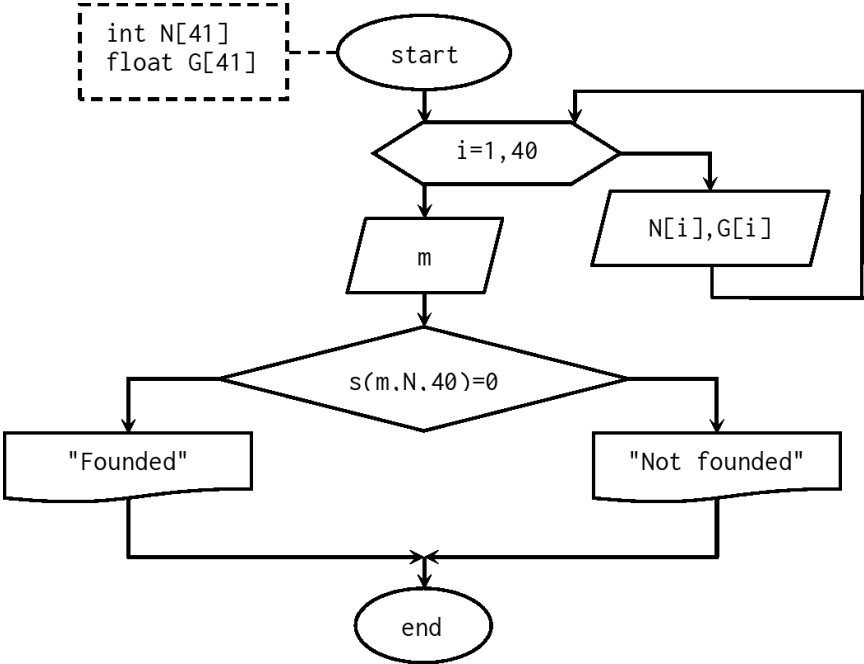


Fig. 8.7(b): Searching an ID number among 40 numbers using the search() function.

C++ codes:

```

// Program P8_7 to search a student
// among 40 student and print the
// result, using the search function
#include <iostream>
using namespace std;
int s(int, int [], int);

int main() {
    int N[41];
    float G[41];
    int i, m;
    for (i=1; i<=40; i++) {
        cout<<"Enter ID and grade: ";
        cin>>N[i]>>G[i];
    }
    cout<<"Enter the ID that want "
        <<"to be searched: ";
    cin>>m;
    if (s(m, N, 40)==0)
        cout<<"The ID is not founded";
    else
        cout<<"The ID is founded";
    return 0;
}
//*****
int s(int m, int G[], int n) {
    int s=0;
    for (int i=1; i<=n; i++)
        if (m==G[i])
            s=1;
    return s;
}

```

Java codes:

```

// Program P8_7 to search a student
// among 40 student and print the
// result, using the search method
import java.util.Scanner;
class P8_7 {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int N[]=new int[41];
        float G[]=new float[41];
        int i, m;
        for (i=1; i<=40; i++) {
            System.out.print("Enter ID and grade: ");
            N[i]=read.nextInt();
            G[i]=read.nextFloat();
        }
        System.out.print("Enter the ID that want "
            + "to be searched: ");
        m=read.nextInt();
        if (s(m, N, 40)==0)
            System.out.print("The ID is not founded ");
        else
            System.out.print("The ID is founded ");
        read.close();
    }
//*****
    static int s(int m, int G[], int n) {
        int s=0;
        for (int i=1; i<=n; i++)
            if (m==G[i])
                s=1;
        return s;
    }
}

```

8.7.1. Exercise. Modify the main program in **Figure 8.7(b)** in such a way to determine and print the index in which the matching happens if the number m is found in the array a . Then, redraw **Flowchart 8.7(b)** and rewrite Programs P8_7 with this idea.

8.8. Example. Write an algorithm to read 20 entries of an integer array named a . Then remove the repetitive entries and put the remained entries in another integer array named b , respectively. In

other words, the entries of the array b are the same as the array a without the repeating ones. Finally, separately print the arrays a and b .

Solution. The method used in writing this algorithm is as follows. The n is regarded as the index maker variable of the array b . First, a_1 is substituted for b_1 . Then, the entry a_i is considered in a for template with the specification $i=2,20$ and, using the search() function, this entry is searched for among the entries of b which are created to this point. If a_i is not found between the entries of b , the index maker of b creates a new index and the picked entry is placed in the array b with the created index. This method is visualized in **Flowchart 8.8(a)**.

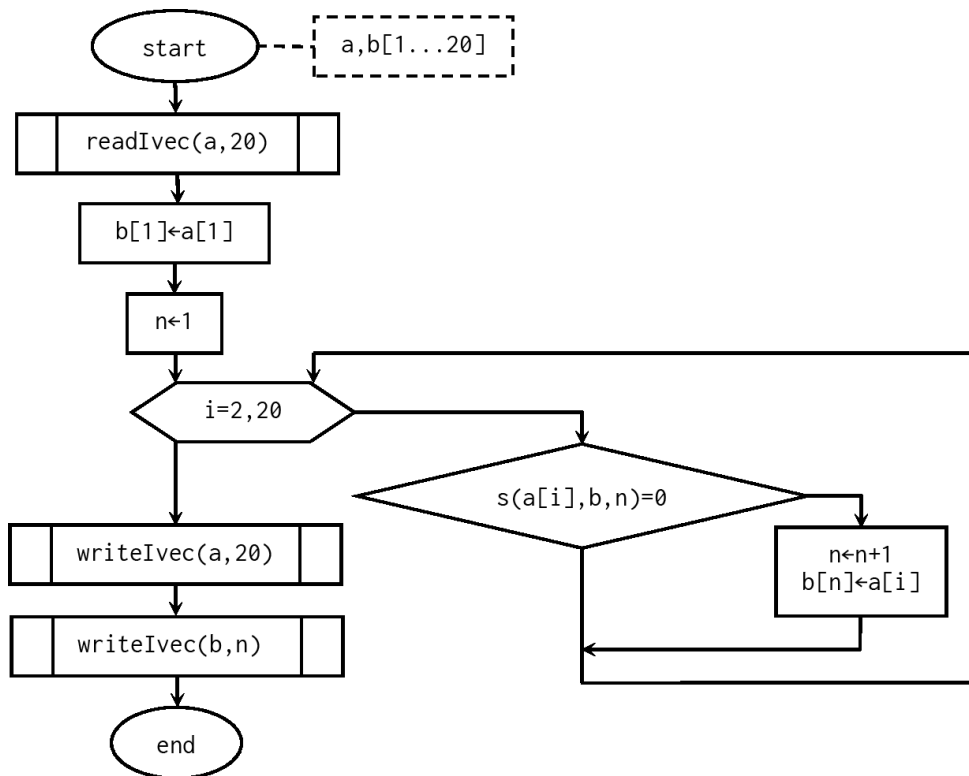


Fig. 8.8(a): Removing the repeated entries of an array.

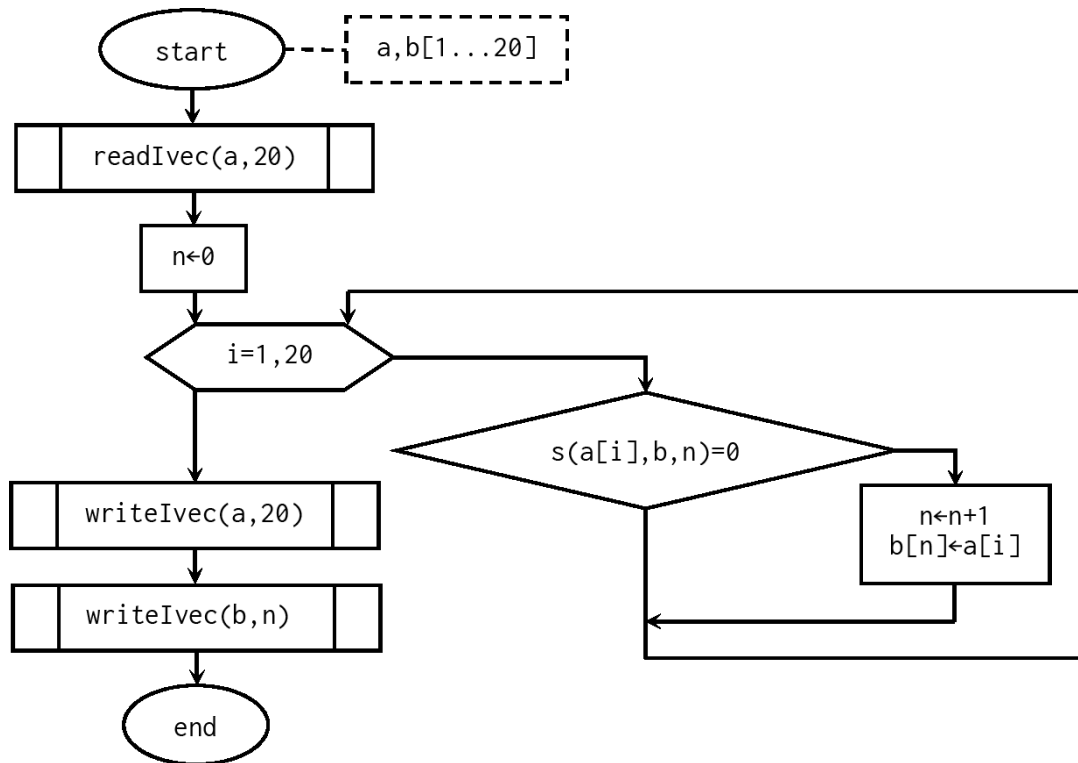


Fig. 8.8(b): Removing the repeated entries of an array (improved version).

Now, let us improve the above algorithm. Remove the instruction $b[1] \leftarrow a[1]$, take 0 as the initial value of the index maker variable n , and then start the for template with the initial value of 1. Accordingly, the incompatibility in the for loop of the search() function causes the index maker n to be equal to 1 and a_1 substitutes for b_1 . **Flowchart 8.8(b)** illustrates the improved version of the algorithm. Programs P8_8 are the codes of the improved **Flowchart 8.8(b)** in the C++ and Java languages.

C++ codes:

```
// Program P8_8 to remove the
// repeated entries of an array
#include <iostream>
using namespace std;
void writeIvec(int u[], int n) {
    for (int i=1; i<=n; i++)
        cout<<u[i]<<" ";
    cout<<endl;
    return;
}
//*****
void readIvec(int u[], int n) {
    for (int i=1; i<=n; i++)
        cin>>u[i];
    return;
}
//*****
int s(int m, int a[], int n) {
    int s=0;
    for (int i=1; i<=n; i++)
        if (m==a[i])
            s=1;
    return s;
}
//*****
int main() {
    int a[21], b[21];
    int i, n;
    cout<<"Enter the array a: \n";
    readIvec(a, 20);
    n=0;
    for (i=1; i<=20; i++)
        if (s(a[i],b,n)==0) {
            n=n+1;
            b[n]=a[i];
        }
    cout<<"Array a: \n";
    writeIvec(a, 20);
    cout<<"Array b: \n";
    writeIvec(b, n);
    return 0;
}
```

Input/output (for ten numbers):**Java codes:**

```
// Program P8_8 to remove the
// repeated entries of an array
import java.util.Scanner;
class P8_8 {
    static void writeIvec(int u[], int n) {
        for (int i=1; i<=n; i++)
            System.out.print(u[i] + " ");
        System.out.println();
    }
    //*****
    static void readIvec(int u[], int n) {
        Scanner read=new Scanner(System.in);
        for (int i=1; i<=n; i++)
            u[i]=read.nextInt();
        read.close();
    }
    //*****
    static int s(int m, int a[], int n) {
        int s=0;
        for (int i=1; i<=n; i++)
            if (m==a[i])
                s=1;
        return s;
    }
    //*****
    public static void main(String[] args) {
        int a[]=new int[21];
        int b[]=new int[21];
        int i, n;
        System.out.println("Enter the array a: ");
        readIvec(a, 20);
        n=0;
        for ( i=1; i<=20; i++)
            if (s(a[i],b,n)==0) {
                n=n+1;
                b[n]=a[i];
            }
        System.out.println("Array a: ");
        writeIvec(a, 20);
        System.out.println("Array b: ");
        writeIvec(b, n);
    }
}
```

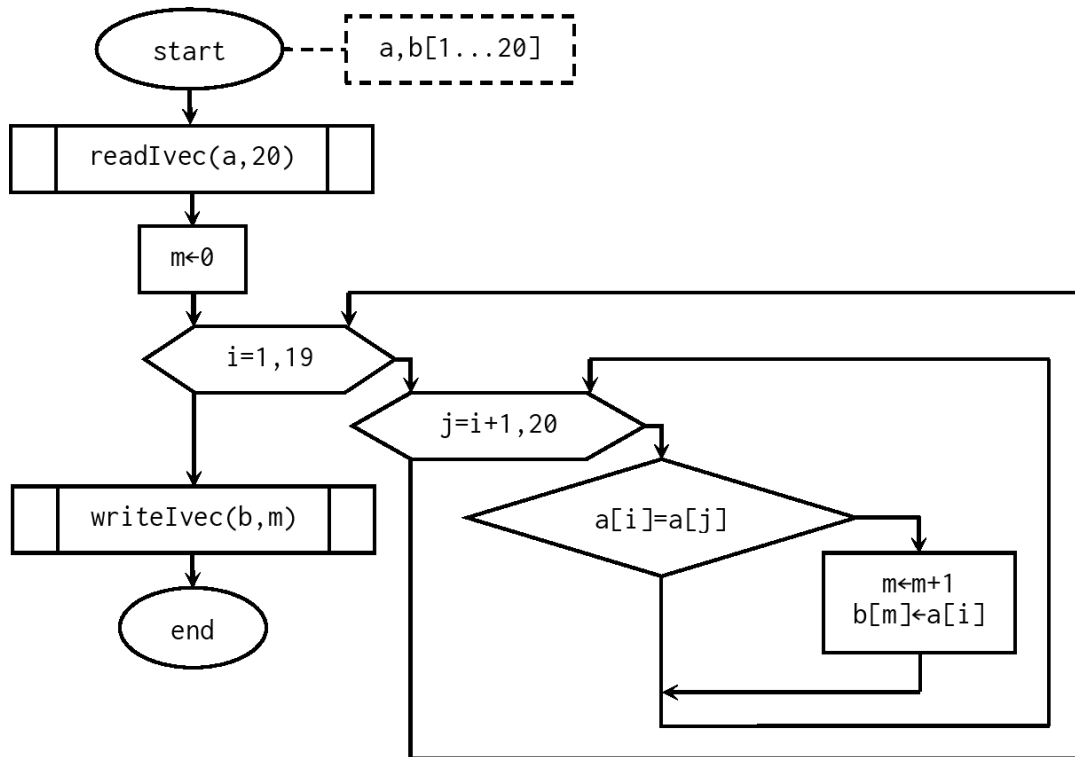


Fig. 8.9(a): Storing the repeated numbers of an array in another array in a special case.

Enter the array a:
 -1 2 4 5 2 9 5 2 -4 3↵
 Array a:
 -1 2 4 5 2 9 5 2 -4 3
 Array b:
 -1 2 4 5 9 -4 3

Input/output (for ten numbers):

Enter the array a:
 -1 2 4 5 2 9 5 2 -4 3↵
 Array a:
 -1 2 4 5 2 9 5 2 -4 3
 Array b:
 -1 2 4 5 9 -4 3

8.9. Example. Write an algorithm to read the entries of a 20-entry integer array a and store its repetitive entries in another array b . Then print the array b . Note that the repeated prints are not allowed.

Solution. First, we work with a simple case: it is supposed that the entries of a are not repeated more than once. In an outer for template with the specification $i=1,19$, we pick an entry a_i and compare it with the entries of a after a_i in an inner for loop with the specification $j=i+1,n$. Then, we store it in b if a_i matches with an

a_j . a_j . Therefore, we make a new index by increasing the index maker variable m by one unit, and then store the picked a_i a_i in b with the new index. The result is displayed in **Flowchart 8.9(a)**.

Now, we examine the general case where the entries of a may be repeated any times. In this case, we *nominate* the picked a_i a_i in the matching path $a_i = a_j$ $a_i = a_j$ to be stored in b and store it in b using the above-mentioned method only if it has not already stored in b . **Flowchart 8.9(b)** depicts this procedure.

Next, we attempt to improve the algorithm. An appeal to the rule of merging the conditions using the && operator (**Chapter 4**) leads to the improved **Flowchart 8.9(c)**.

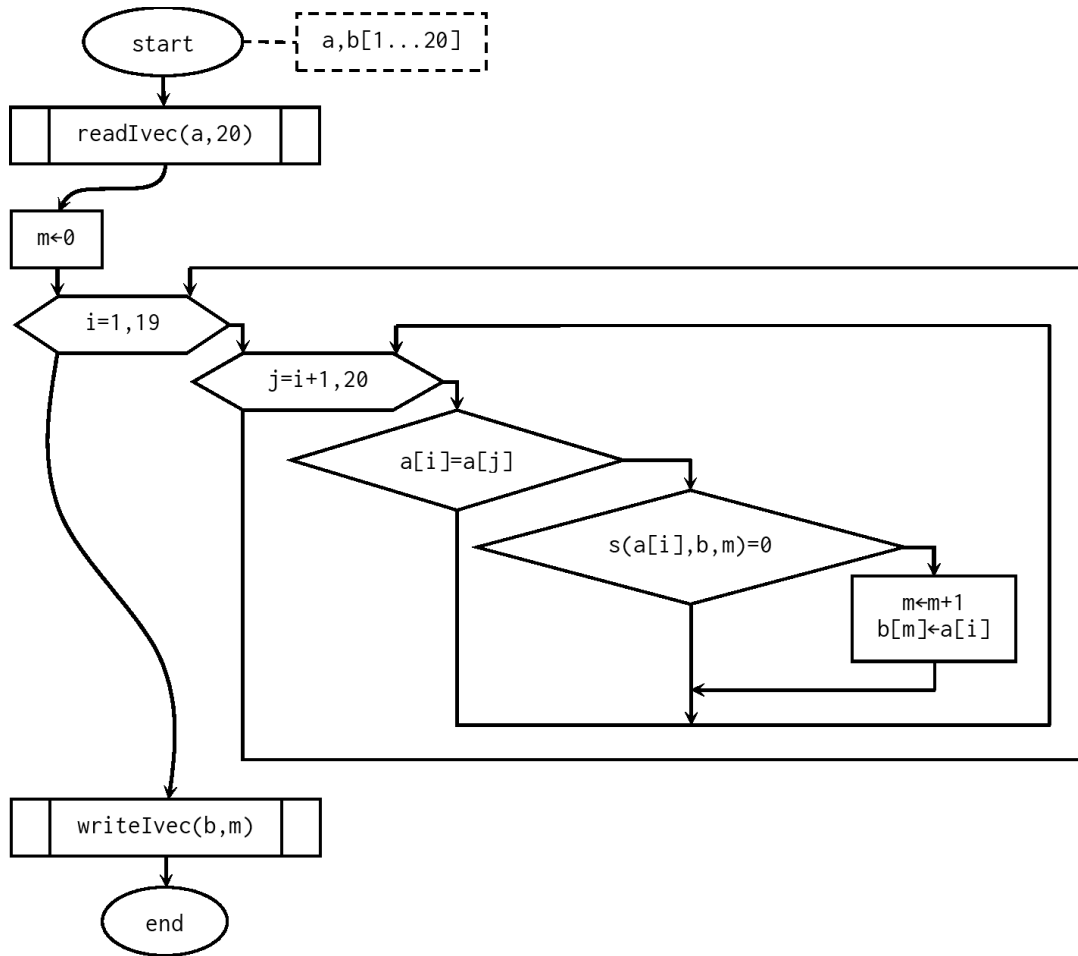


Fig. 8.9(b): Storing the repeated numbers of an array in another array in the general case.

Programs P8_9 represent the codes of the improved **Flowchart 8.9(c)** together with all of its subprograms.

C++ codes:

```
// Program P8_9 to store the
// repeated numbers of an array
// in another array
#include <iostream>
using namespace std;
void writeIvec(int u[], int n) {
    for (int i=1; i<=n; i++)
        cout<<u[i]<<" ";
    cout<<endl;
    return;
}
//*****
void readIvec(int u[], int n) {
    for (int i=1; i<=n; i++)
```

Java codes:

```
// Program P8_9 to store the
// repeated numbers of an array
// in another array
import java.util.Scanner;
class P8_9 {
    static void writeIvec(int u[], int n) {
        for (int i=1; i<=n; i++)
            System.out.print(u[i] + " ");
        System.out.println();
    }
    //*****
    static void readIvec(int u[], int n) {
        Scanner read=new Scanner(System.in);
        for (int i=1; i<=n; i++)
```

```

        cin>>u[i];
    return;
}
//*****
int s(int m, int a[], int n) {
    int s=0;
    for (int i=1; i<=n; i++)
        if (m==a[i])
            s=1;
    return s;
}
//*****
int main() {
    int a[21], b[21];
    int i, j, m;
    cout<<"Enter the array a: \n";
    readIvec(a, 20);
    m=0;
    for (i=1; i<=19; i++)
        for (j=i+1; j<=20; j++)
            if (a[i]==a[j]
                && s(a[i], b, m)==0) {
                m=m+1; b[m]=a[i];
            }
    cout<<"The repeated entries "
        <<"in the array b: \n";
    writeIvec(b, m);
    return 0;
}

```

Input/output (for ten numbers):

```

Enter the array a:
-1 2 4 5 2 9 5 2 -4 3↵
The repeated entries in the array b:
2 5

```

```

        u[i]=read.nextInt();
    read.close();
}
//*****
static int s(int m, int a[], int n) {
    int s=0;
    for (int i=1; i<=n; i++)
        if (m==a[i])
            s=1;
    return s;
}
//*****
public static void main(String[] args) {
    int a[]=new int[21];
    int b[]=new int[21];
    int i, j, m;
    System.out.println("Enter the array a: ");
    readIvec(a, 20);
    m=0;
    for (i=1; i<=19; i++)
        for (j=i+1; j<=20; j++)
            if (a[i]==a[j] && s(a[i], b, m)==0) {
                m=m+1; b[m]=a[i];
            }
    System.out.println("The repeated entries "
        + "in the array b: ");
    writeIvec(b, m);
}
}

```

Input/output (for ten numbers):

```

Enter the array a:
-1 2 4 5 2 9 5 2 -4 3↵
The repeated entries in the array b:
2 5

```

8.9.1. Exercise. Modify Algorithm 8.9 and Programs 8_9 so that they print the number of repetitions along with the repetitive entries.

8.10. Example. Write an algorithm to read the entries of the two integer arrays *a* and *b*, each with 20 entries and then store their common entries in another integer array named *c*. Finally, print the entries of *c*.

Solution. First, in a simple case, we assume that none of the arrays a and b have repetitive entries. We consider m as the index maker variable for the array c . In each repetition of a for loop with the specification $i=1,20$, we take an entry a_i of a and search for it among the entries of b , using the `search()` function in **Example 8.7**. Then, we store the a_i in the array c if we find it in the array b . To do this, as before, we make a new index by increasing the index maker variable m by one unit, and then store the picked a_i in b with the created index.

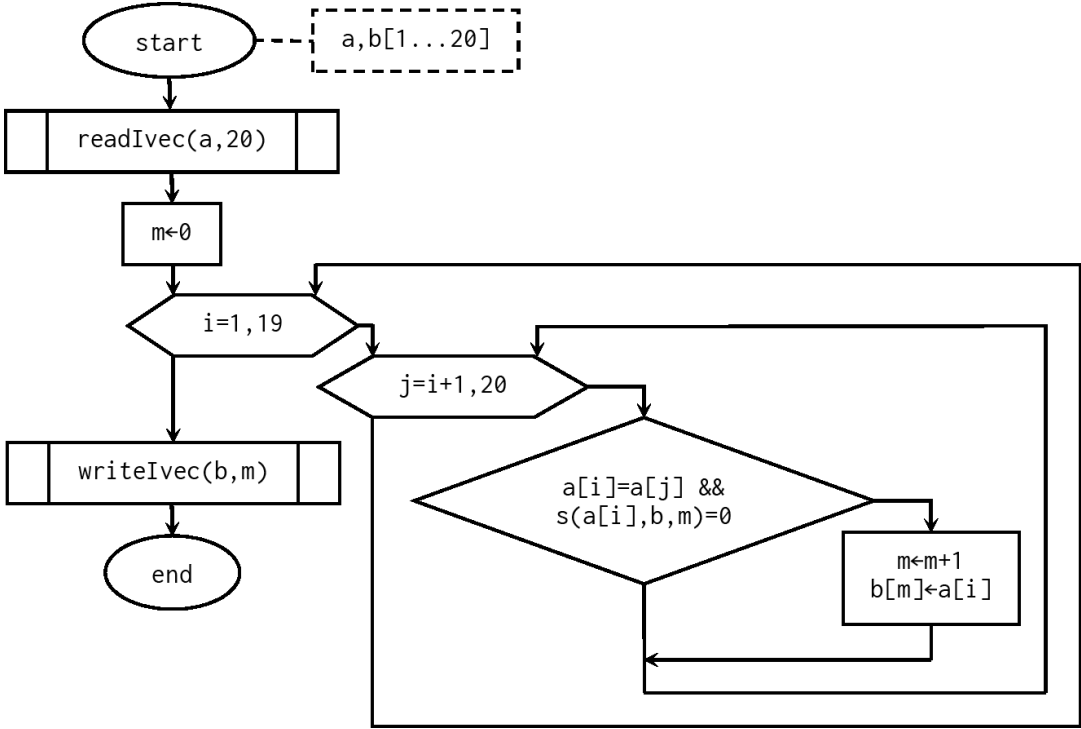


Fig. 8.9(c): Improvement **Flowchart 8.9(b)** by merging the conditions using the `&&` operator.

Flowchart 8.10(a) displays the result to this point.

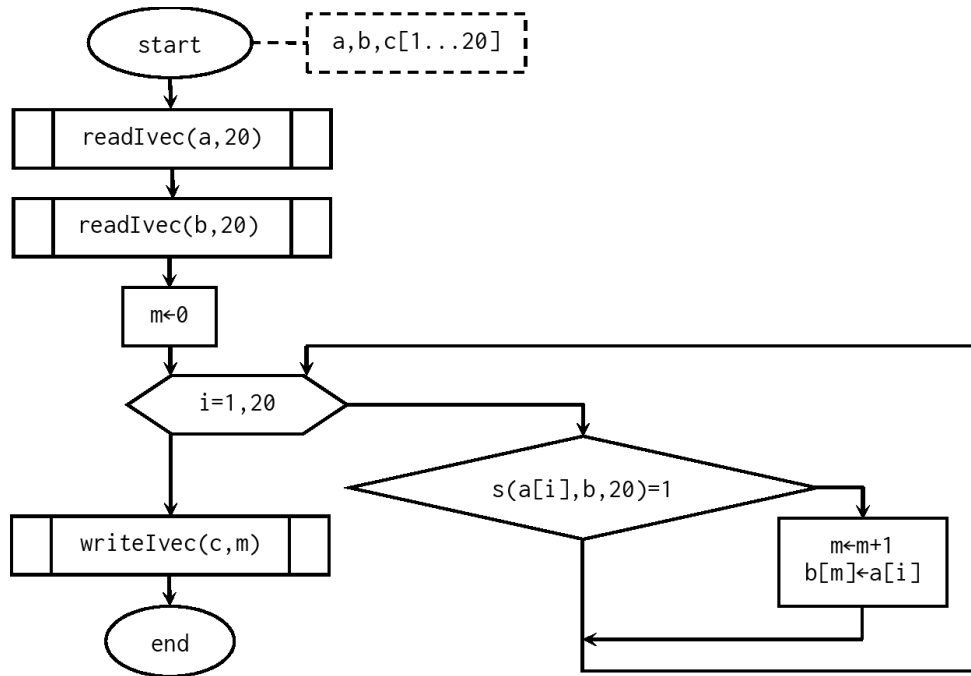


Fig. 8.10(a): Storing the common entries of two arrays in another array in a special case.

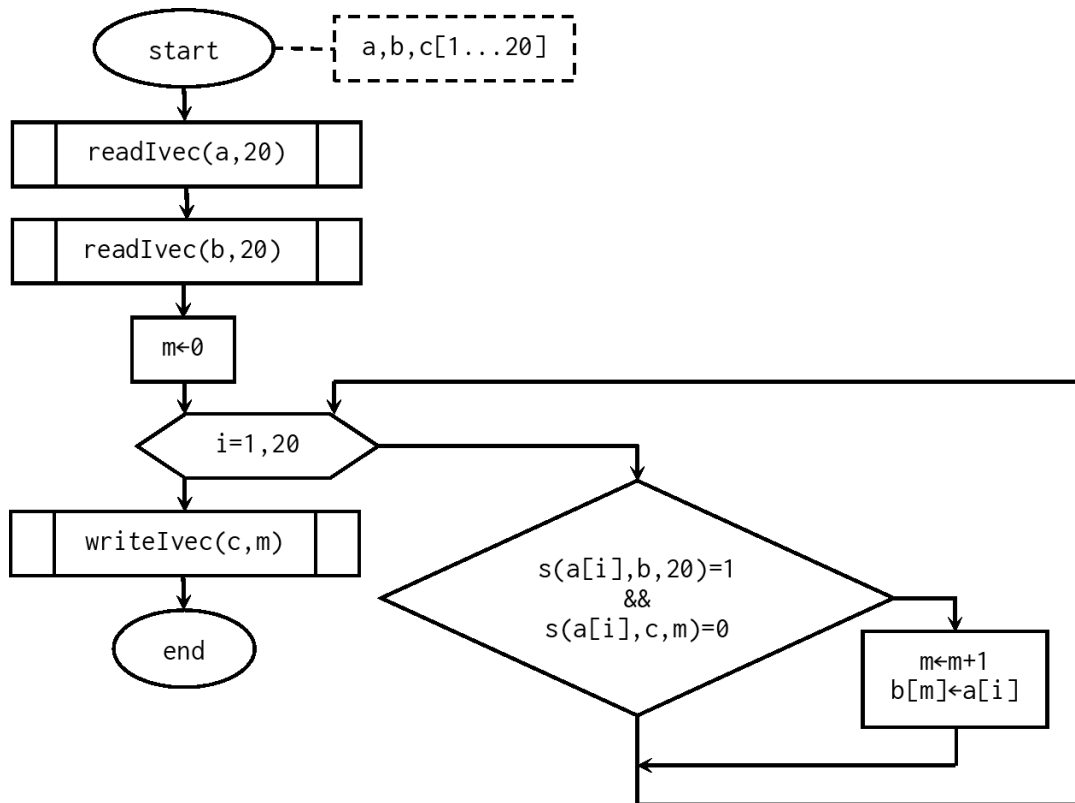


Fig. 8.10(b): Storing the common entries of two arrays in another array in the general case.

Now, we consider the general case in which the arrays a and b can have repeating entries. In this case, we store the picked entry a_i in the array c only if it is not already stored in c ; that is, if a_i is in b but not in c . **Flowchart 8.10(b)** illustrates the result. The general **case 8.10(b)** is translated into C++ and Java codes in Programs P8_10.

C++ codes:

```
// Program P8_10 to store the
// common entries of two arrays
// a and b in the array c
#include <iostream>
using namespace std;
void writeIvec(int u[], int n) {
    for (int i=1; i<=n; i++)
        cout<<u[i]<<" ";
    cout<<endl;
    return;
}
//*****
void readIvec(int u[], int n) {
    for (int i=1; i<=n; i++)
        cin>>u[i];
    return;
}
//*****
int s(int m, int a[], int n) {
```

Java codes:

```
// Program P8_10 to store the
// common entries of two arrays
// a and b in the array c
import java.util.Scanner;
class P8_10 {
    static void writeIvec(int u[], int n) {
        for (int i=1; i<=n; i++)
            System.out.print(u[i] + " ");
        System.out.println();
    }
    //*****
    static void readIvec(int u[], int n) {
        Scanner read=new Scanner(System.in);
        for (int i=1; i<=n; i++)
            if (read.hasNext())
                u[i]=read.nextInt();
        read.reset();
    }
    //*****
}
```

```

int s=0;
for (int i=1; i<=n; i++)
    if (m==a[i])
        s=1;
return s;
}
//*****
int main() {
    int a[21], b[21], c[21];
    int m=0;
    cout<<"Enter the array a: \n";
    readIvec(a, 20);
    cout<<"Enter the array b: \n";
    readIvec(b, 20);
    for (int i=1; i<=20; i++)
        if (s(a[i], b, 20)==1 &&
            s(a[i], c, m)==0) {
            m=m+1;
            c[m]=a[i];
        }
    cout<<"The common entries : \n";
    writeIvec(c, m);
    return 0;
}

```

Input/output (for ten numbers):

```

Enter the array a:
-1 2 9 4 -4 5 7 3 2 -1↵
Enter the array b:
5 7 -1 3 4 7 2 -1 3 -1↵
The common entries :
-1 2 4 5 7 3

```

```

static int s(int m, int a[], int n) {
    int s=0;
    for (int i=1; i<=n; i++)
        if (m==a[i])
            s=1;
    return s;
}
//*****
public static void main(String[] args) {
    int a[]=new int[21];
    int b[]=new int[21];
    int c[]=new int[21];
    int m=0;
    System.out.println("Enter the array a: ");
    readIvec(a, 20);
    System.out.println("Enter the array b: ");
    readIvec(b, 20);
    for (int i=1; i<=20; i++)
        if (s(a[i], b, 20)==1 && s(a[i], c, m)==0) {
            m=m+1;
            c[m]=a[i];
        }
    System.out.println("The common entries: ");
    writeIvec(c, m);
}
}

```

Input/output (for ten numbers):

```

Enter the array a:
-1 2 9 4 -4 5 7 3 2 -1↵
Enter the array b:
5 7 -1 3 4 7 2 -1 3 -1↵
The common entries :
-1 2 4 5 7 3

```

8.2 More applications of the arrays

In this section, two sorting methods for numbers are considered. Then, the Lagrange interpolation polynomial for finding the approximated root of a function is examined. Finally, a method for computing the factorial of large positive integers is exhibited.

8.11. Example (The bubble sorting method). One of the common methods is the bubble method. We used it for sorting three numbers in the algorithm of **Example 4.10**. Write an algorithm to read first the size m (at most 40) of the integer array a . Then, read the m entries of a and sort them in ascending order using the bubble method.

Solution. In this method, we sort m numbers in ascending order in $m-1$ steps in a way that in each step a number is compared with the latter number such that if it is larger, then their amounts are swapped using the swap algorithm. As a result, the larger number rises upward (moves to the end) like a bubble and is put aside. Then the recent procedure is performed with the remaining numbers. In order to clarify this procedure, take the four integers:

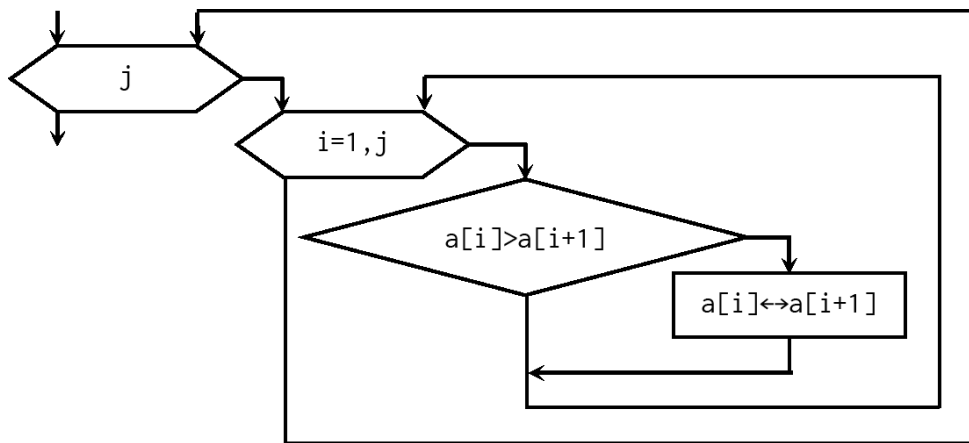


Fig. 8.11(a): Sketch of the bubble sorting algorithm in ascending order.

12 9 24 5 **12** **9** **24** **5**

Following the above-mentioned procedure, in the first step,

12 is swapped with 9:	9	12	24	5
12 is not swapped with 24:	9	12	24	5
24 is swapped with 5:	9	12	5	24

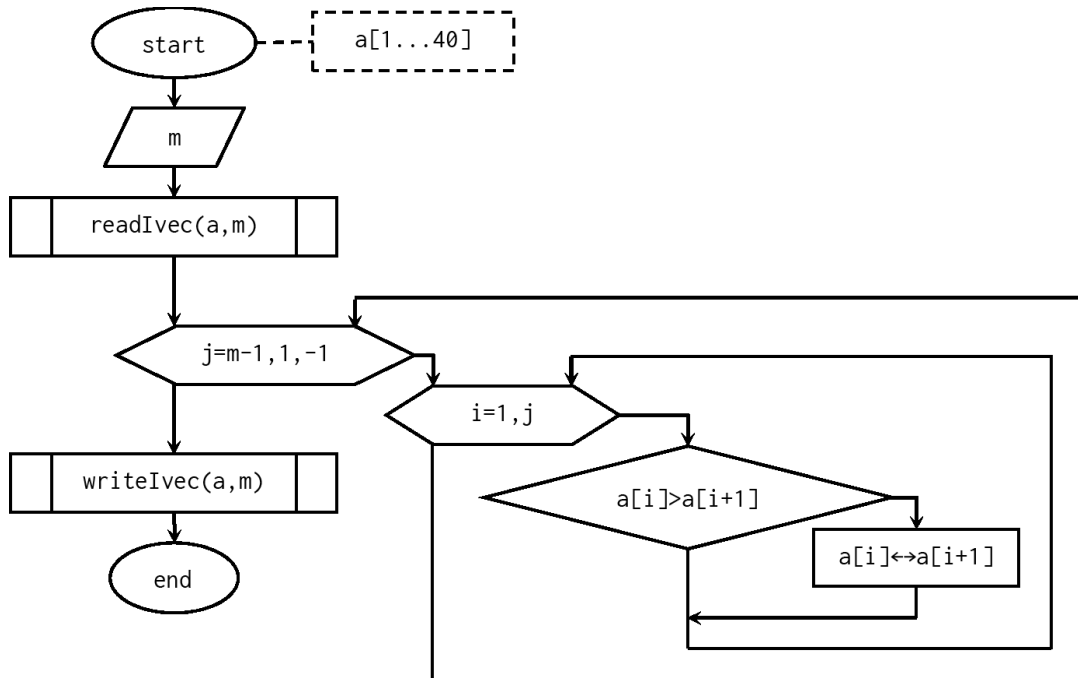


Fig. 8.11(b): Another way for the bubble sorting algorithm in ascending order.

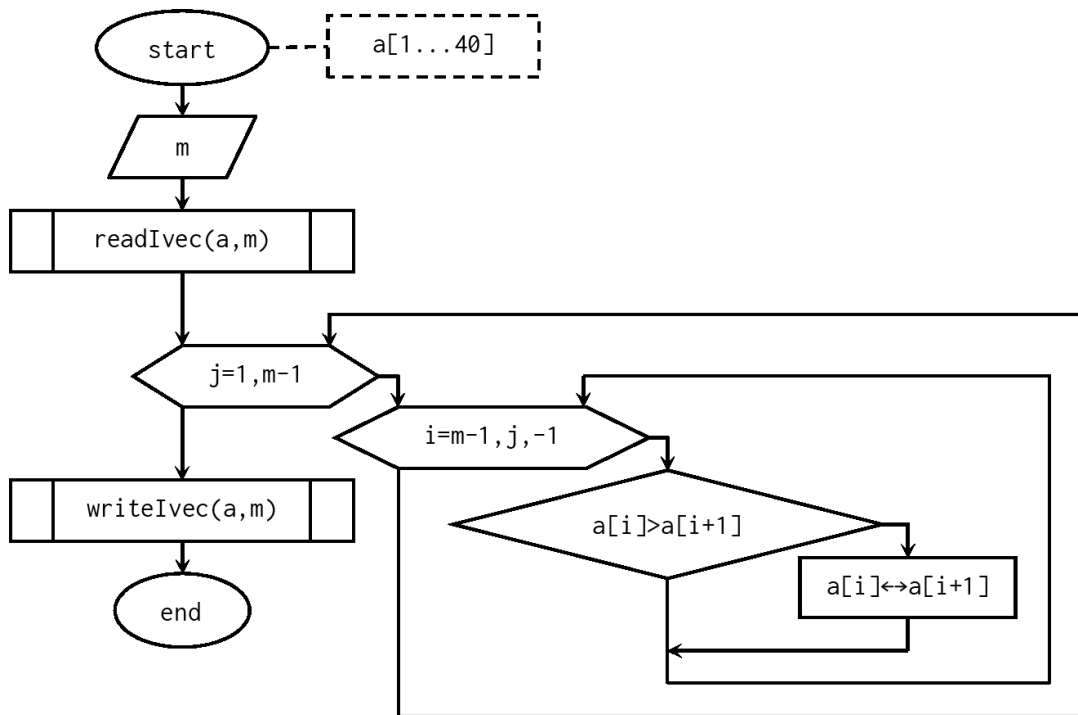


Fig. 8.11(c): Another way for the bubble sorting algorithm in ascending order.

Now, 24 is put aside (underlined) and in the second step, we work on the three remaining numbers. Thus,

9 is not swapped with 12: 9 12 5 24
 12 is swapped with 5: 9 5 12 24

The number 12 is put aside and the two remaining numbers are compared in the third step:

9 is swapped with 5: 5 9 12 24

As seen, several steps are repeated and several processes are performed in each step. Therefore, we need two nested for loops. The outer loop gives the number j , as the times of processes in each step, to the inner loop. Now, the inner loop takes an index i from 1 to the received number j (the times of processes in each step) and compares the entry a_i with a_{i+1} such that it swaps them if $a_i > a_{i+1}$. Up to this point, we have **Flowchart 8.11(a)**.

Next, the specification of the outer loop is remained to find. To do this, we look at the indices that are used in each step:

First step: 1 2 **3**
 Second step: 1 **2**
 Third step: **1**

The number of processes which are performed in each step is written in the bold font: 3, 2, 1. Considering the number of entries used in the above pattern, namely 4, we found the specification of the outer for loop to this pattern: $j=3,1,-1$. Accordingly, the required algorithm for the m entries is now illustrated in **Figure 8. 11(b)**. Programs P8_11 are the translations of **Flowchart 8.11(b)**.

C++ codes:

```
// Program P8_11 to sort the
// entries of an array in ascending
// order by the bubble method
#include <iostream>
using namespace std;
void writeIvec(int u[], int n) {
    for (int i=1; i<=n; i++)
        cout<<u[i]<<" ";
    cout<<endl;
    return;
}
//*****
void readIvec(int u[], int n) {
    for (int i=1; i<=n; i++)
        cin>>u[i];
    return;
}
//*****
int main() {
    int a[41];
    int i, j, t, m;
    cout<<"Entre the size of the array: ";
    cin>>m;
    cout<<"Enter the array: ";
    readIvec(a, m);
    for (j=m-1; j>=1; j--)
        for (i=1; i<=j; i++)
            if (a[i]>a[i+1]) {
                t=a[i];
                a[i]=a[i+1];
                a[i+1]=t;
            }
    cout<<"The sorted array: ";
    writeIvec(a, m);
    return 0;
}
```

Java codes:

```
// Program P8_11 to sort the
// entries of an array in ascending
// order by the bubble method
import java.util.Scanner;
class P8_11 {
    static void writeIvec(int u[], int n) {
        for (int i=1; i<=n; i++)
            System.out.print(u[i] + " ");
        System.out.println();
    }
    //*****
    static void readIvec(int u[], int n) {
        Scanner read=new Scanner(System.in);
        for (int i=1; i<=n; i++)
            u[i]=read.nextInt();
        read.close();
    }
    //*****
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int a[]=new int[41];
        int i, j, t, m;
        System.out.print("Enter the size "
            + "of the array: ");
        m=read.nextInt();
        System.out.print("Enter the array: ");
        readIvec(a, m);
        for (j=m-1; j>=1; j--)
            for (i=1; i<=j; i++)
                if (a[i]>a[i+1]) {
                    t=a[i];
                    a[i]=a[i+1];
                    a[i+1]=t;
                }
        System.out.print("The sorted array: ");
        writeIvec(a, m);
        read.close();
    }
}
```

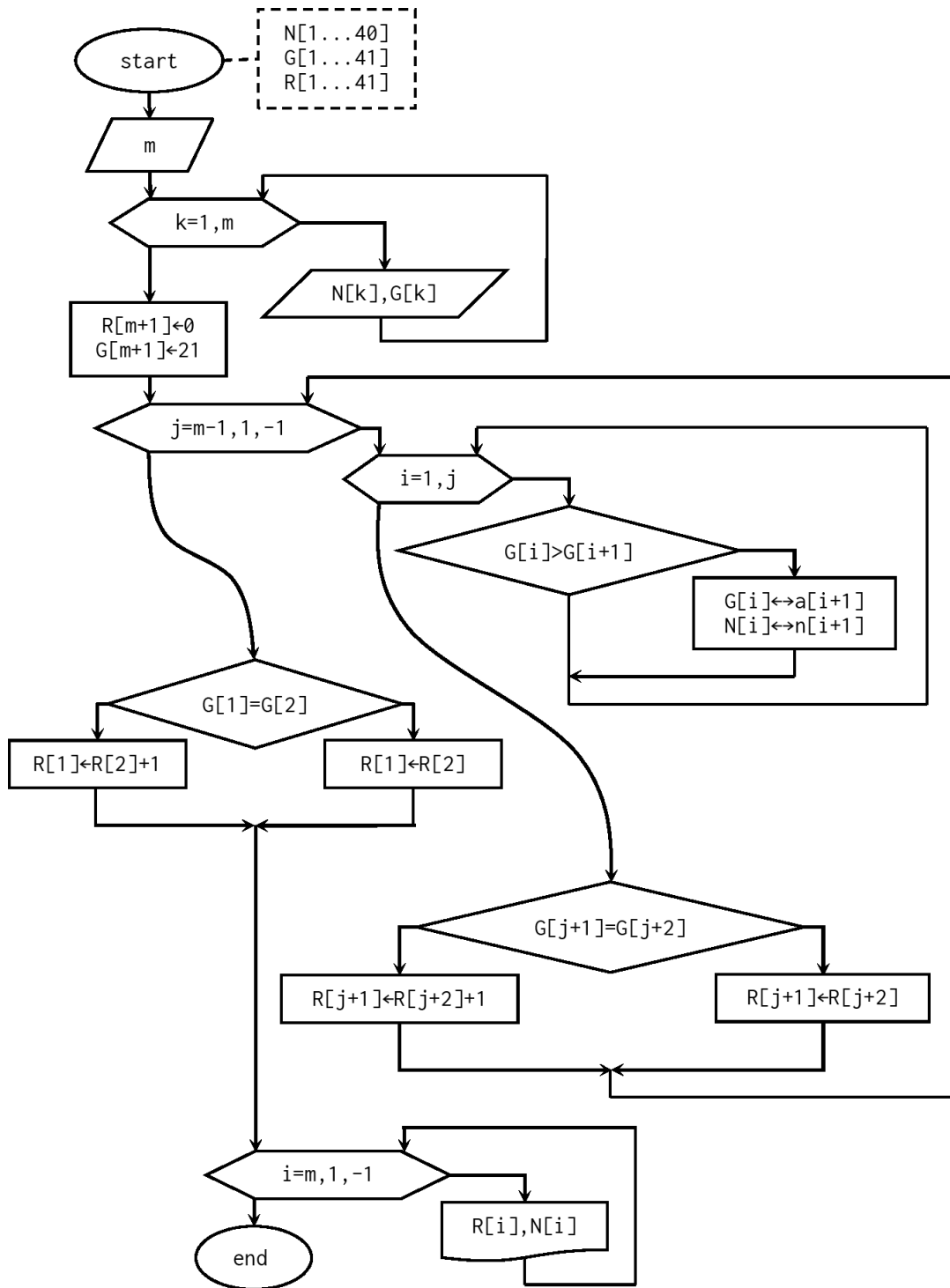



Fig. 8.12(a): Ranking the grades of 40 students.

8.11.1. Exercise. Write a similar algorithm and programs as above for the descending order.

In Algorithm 8.11(b), the *largest* number is transferred to the *end* in each step (each repetition of the outer loop). Moreover, in **Flowchart 8.11(c)**, the *smallest* amount is transferred to the *start* in each step (each repetition of the outer loop). The results are the same, that is, sorting in ascending order using the bubble method.

8.11.2. Exercise. Arrange the implementation table for both **flowcharts 8.11(b)** and 8.11(c) using five various integers.

8.12. Example. Design an algorithm in a way that each time it reads the ID number and grade of one of the m students of a class which m is assumed to be at most 40. Then, it ranks the students and prints the ranks together with the ID numbers in order from top rank downwards. Trivially, two equal grades have the same ranks.

Solution. Denote the arrays of ID numbers, grades, and ranks by N , G , and R , respectively. We assume the grades out of 20. Recall that the larger number is transferred to the end in each step in the algorithm of ascending sorting using the bubble method (**Fig. 8.11(b)**). We construct the main process of this algorithm based on this concept. More precisely, we transfer the largest grade, together with the corresponding ID to the end in each step in the inner for loop of the

bubble sorting algorithm. These are actually the grade G_{j+1} and the corresponding ID, N_{j+1} . Then, we determine the ranks after exiting from the inner loop: we compare the grades G_{j+1} and G_{j+2} to perform the ranking process such that it takes the rank of G_{j+2} , namely R_{j+2} , if the grade G_{j+1} is equal to G_{j+2} ; otherwise, it takes one rank more than the recent one R_{j+2} .

We have two gaps in the ranks. First, there is no rank to compare with R_m . R_m . To fill this gap, we assign the virtual initial rank 0 to R_{m+1} and simultaneously assign the virtual initial grade 21 to G_{m+1} and locate these initial values before the outer for loop.

Additionally, R_1 has not yet been determined. To fill this gap, use the same ranking process as above, taking $j = 0$ after exiting the outer loop.

Now, it is time to print. The printing is performed from top rank downwards. **Flowchart 8.12(a)** covers what has been mentioned so far.



The length of these arrays are declared one unit more than their real length since the virtual values other than the real values are used for the grade and rank.

The second gap can more easily be filled. It suffices to remove the ranking process for $j = 0$ after exiting the outer loop and, instead, change the initial value of the outer for loop from 1 to 0. In fact, it is the incompatibility in the inner for loop that settles the rank R_1 . R_1 . The improved flowchart is shown in **Figure 8.12(b)**. The codes of **Flowchart 8.12(b)** can be found in Programs P8_12.

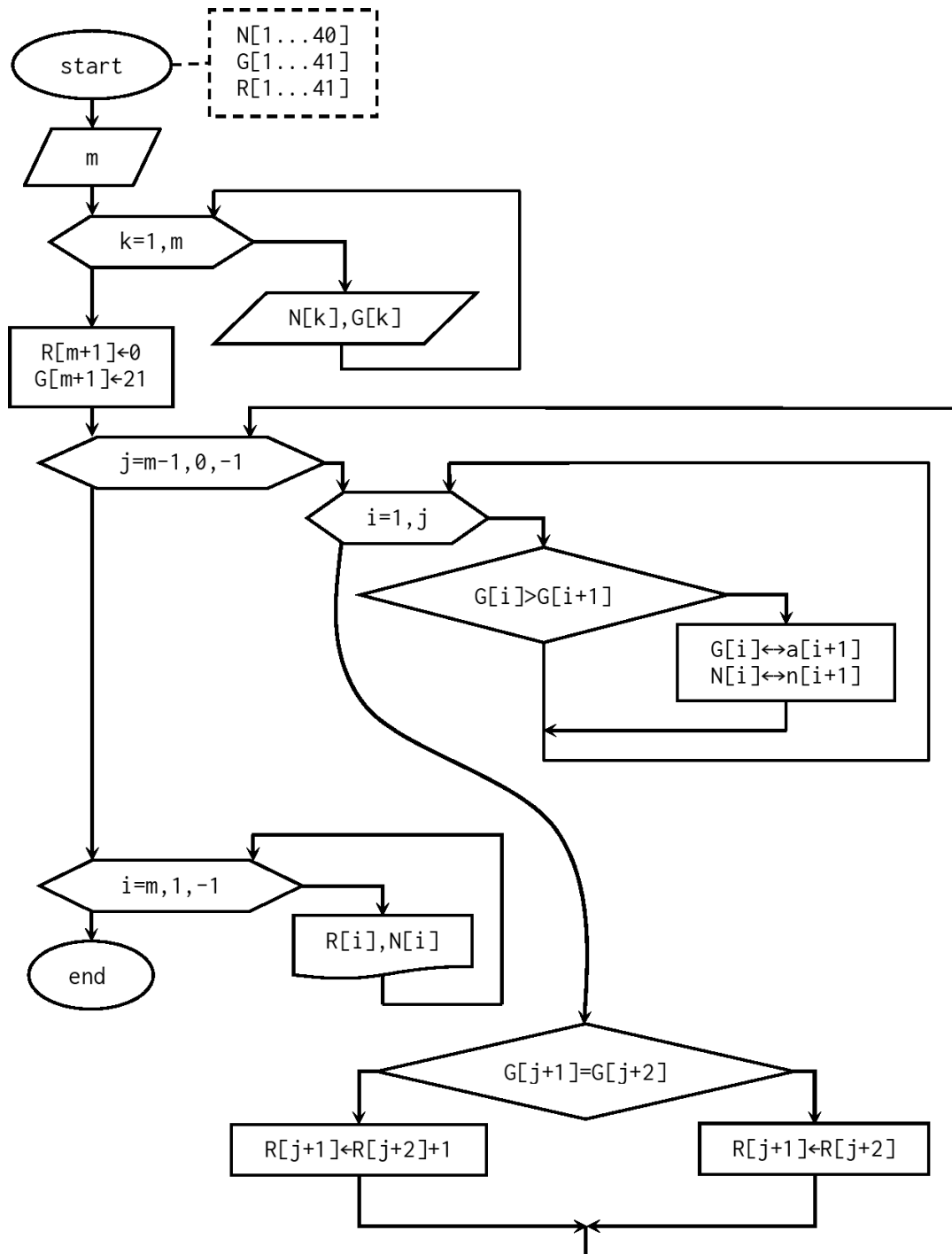


Fig. 8.12(b): Ranking the grades of 40 students (improved version).

C++ codes:

```
// Program P8_12 to rank the grades
// of m students and print them
// from top downward
#include <iostream>
#include <iomanip>
using namespace std;
int main() {

    int N[41], R[42], i, j, k, t, m;
    float G[42], s;
    cout<<"Entre the number of Students: ";
    cin>>m;
    for (k=1; k<=m; k++) {
        cout<<"Enter the ID and grade: ";
        cin>>N[k]>>G[k];
    }
    R[m+1]=0;
    G[m+1]=21;
    for (j=m-1; j>=0; j--) {
        for (i=1; i<=j; i++)
            if (G[i]>G[i+1]) {
                t=N[i]; N[i]=N[i+1]; N[i+1]=t;
                s=G[i]; G[i]=G[i+1]; G[i+1]=s;
            }
        if (G[j+1]==G[j+2])
            R[j+1]=R[j+2];
        else
            R[j+1]=R[j+2]+1;
    }
    cout<<"Rank      ID"<<endl;
    cout<<"----  -" <<endl;
    for (i=m; i>=1; i--)
        cout<<setw(4)<<R[i]<<setw(10)
            <<N[i]<<endl;

    return 0;
}
```

Input/output:

```
Entre the number of Students: 4↵
Enter the ID and grade: 19221001 12.5↵
Enter the ID and grade: 19221002 16↵
Enter the ID and grade: 19221003 7↵
Enter the ID and grade: 19221004 14.25↵
Rank      ID
----  -
1 19221004
```

Java codes:

```
// Program P8_12 to rank the grades
// of m students and print them
// from top downward
import java.util.Scanner;
class P8_12 {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);

        int N[]=new int[41];
        int R[]=new int[42];
        int i, j, k, t, m;
        float G[]=new float[42];
        float s;
        System.out.print("Enter the number "
            + "of Students: ");
        m=read.nextInt();
        for (k=1; k<=m; k++) {
            System.out.print("Enter the ID "
                + "and grade: ");
            N[k]=read.nextInt();
            G[k]=read.nextFloat();
        }
        R[m+1]=0;
        G[m+1]=21;
        for (j=m-1; j>=0; j--) {
            for (i=1; i<=j; i++)
                if (G[i]>G[i+1]) {
                    t=N[i]; N[i]=N[i+1]; N[i+1]=t;
                    s=G[i]; G[i]=G[i+1]; G[i+1]=s;
                }
            if (G[j+1]==G[j+2])
                R[j+1]=R[j+2];
            else
                R[j+1]=R[j+2]+1;
        }
        System.out.println("Rank      ID");
        System.out.println("----  -");
        for (i=m; i>=1; i--)
            System.out.printf("%4d %9d\n",
                R[i], N[i]);

        read.close();
    }
}
```

Input/output:

```
Entre the number of Students: 4↵
```

```

1 19221002
2 19221004
3 19221001
4 19221003

```

```

-----
Enter the ID and grade: 19221001 12.5↵
Enter the ID and grade: 19221002 16↵
Enter the ID and grade: 19221003 7↵
Enter the ID and grade: 19221004 14.25↵
Rank      ID
-----
1 19221002
2 19221004
3 19221001
4 19221003

```

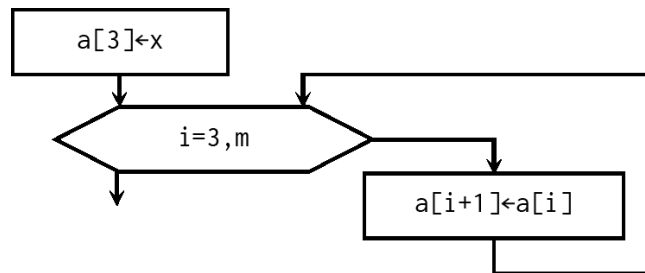


Fig. 8.13(a): Wrong attempt to insert x between the second and third entry.

8.13. Example. Write an algorithm to read first the size m (at most 40) of an integer array a and then, read the entries of a . Next, read an integer x and insert it between the second and third entry. Finally, print the new entries of the $(m + 1)$ -entry array a .

Solution. There are two important points in this algorithm. First, we should declare the array a with a length of at most 42 since one entry is going to be added to the number of the entries of a . The second point is how to insert the read integer x between the second and third entry of the array. We want the read integer x to be accommodated as the third entry, the third entry as the fourth entry, the fourth entry as the fifth entry and the like.

We must be careful in the process of replacing these entries. Only a small carelessness will disarrange the algorithm. For example, we will end up with **Flowchart 8.13(a)** if we intend to write an algorithm based exactly on the above argument.

This flowchart leads to an algorithm which is completely wrong. We arrange the implementation table of **Flowchart 8.13(a)** for $m = 5$

and the value of x as in **Table 8.13(a)** in order to observe the incorrectness.

Tab. 8.13(a): Implementation table for **Flowchart 8.13(a)** for $m = 5$.

status	x	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
data in the memory	100	10	20	30	40	50	
before the loop				100			
first repetition					100		
second repetition						100	
third repetition							100

As illustrated, the algorithm does not work right. Therefore, the insertion process should be started from the final cell of the array which is empty in order to overcome this error. The entries pulled one back, up to the third entry using a for template with the specification $i=3,5$. Finally, x is substituted for a_3 after exiting the loop. The result can be observed in **Flowchart 8.13(b)**.

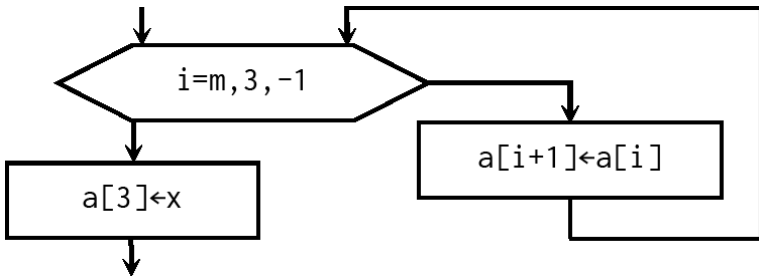


Fig. 8.13(b): Right attempt to insert x between the second and third entry.

Now the implementation **Table 8.13(b)** is rearranged which guarantees the correctness of **Flowchart 8.13(b)**.

Tab. 8.13(b): Implementation table for **Flowchart 8.13(b)** for $m = 5$.

status	x	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
data in the memory	100	10	20	30	40	50	
first repetition							50
second repetition						40	
third repetition					30		
after the loop				100			

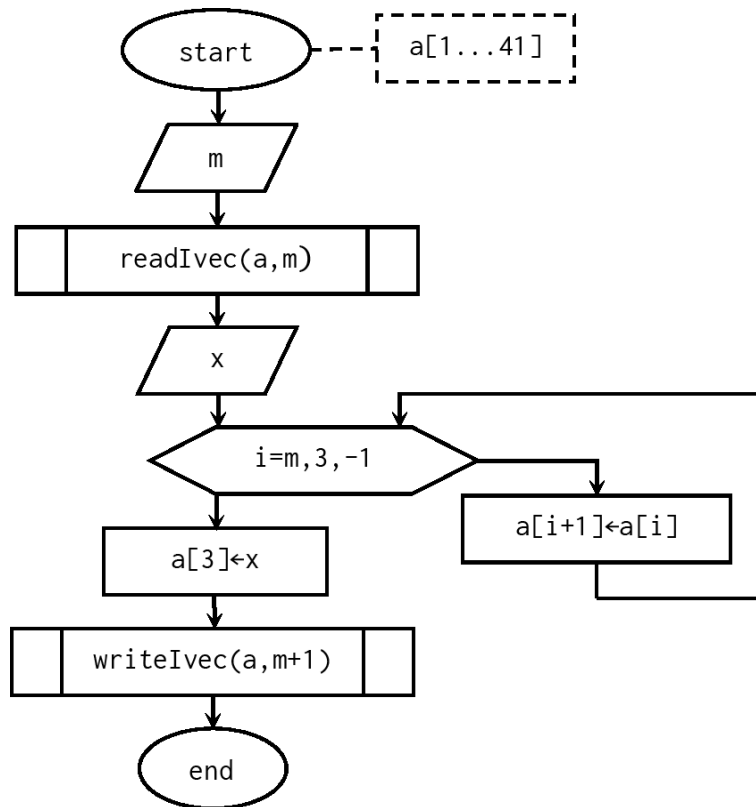


Fig. 8.13(c): Algorithm for reading x and inserting it between the second and third entry.

The codes for the completed **Flowchart 8.13(c)** can be found in Programs P8_13.

C++ codes:

```
// Program P8_13 to insert the
// read number x between the second
// and third entries of the array
#include <iostream>
using namespace std;
void writeIvec(int u[], int n) {
    for (int i=1; i<=n; i++)
        cout<<u[i]<<" ";
    cout<<endl;
    return;
}
//*****
void readIvec(int u[], int n) {
    for (int i=1; i<=n; i++)
        cin>>u[i];
    return;
}
//*****
int main() {
    int a[42], i, x, m;
    cout<<"Entre the size of array: ";
    cin>>m;
    cout<<"Enter the array a: ";
    readIvec(a, m);
    cout<<"Enter the integer x: ";
    cin>>x;
    for (i=m; i>=3; i--)
        a[i+1]=a[i];
    a[3]=x;
    writeIvec(a, m+1);
    return 0;
}
```

Input/output:

Enter an integer n>1: 3

Entre the size of array: 5↵
Enter the array a: 1 2 3 4 5↵
Enter the integer x: 100↵
1 2 100 3 4 5

Java codes:

```
// Program P8_13 to insert the
// read number x between the second
// and third entries of the array
import java.util.Scanner;
class P8_13 {
    static void writeIvec(int u[], int n) {
        for (int i=1; i<=n; i++)
            System.out.print(u[i] + " ");
        System.out.println();
    }
    //*****
    static void readIvec(int u[], int n) {
        Scanner read=new Scanner(System.in);
        for (int i=1; i<=n; i++)
            u[i]=read.nextInt();
        read.reset();
    }
    //*****
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int a[]=new int[42];
        int i, x, m;
        System.out.print("Enter the size "
            + "of array: ");
        m=read.nextInt();
        System.out.print("Enter the array a: ");
        readIvec(a, m);
        System.out.print("Enter the integer x: ");
        x=read.nextInt();
        for (i=m; i>=3; i--)
            a[i+1]=a[i];
        a[3]=x;
        writeIvec(a, m+1);
        read.close();
    }
}
```

Input/output:

Enter an integer n>1: 3

Entre the size of array: 5↵
Enter the array a: 1 2 3 4 5↵
Enter the integer x: 100↵
1 2 100 3 4 5

Note that in the right Java program, the read.reset() method is used due to the multi-use of the readIvec() method.

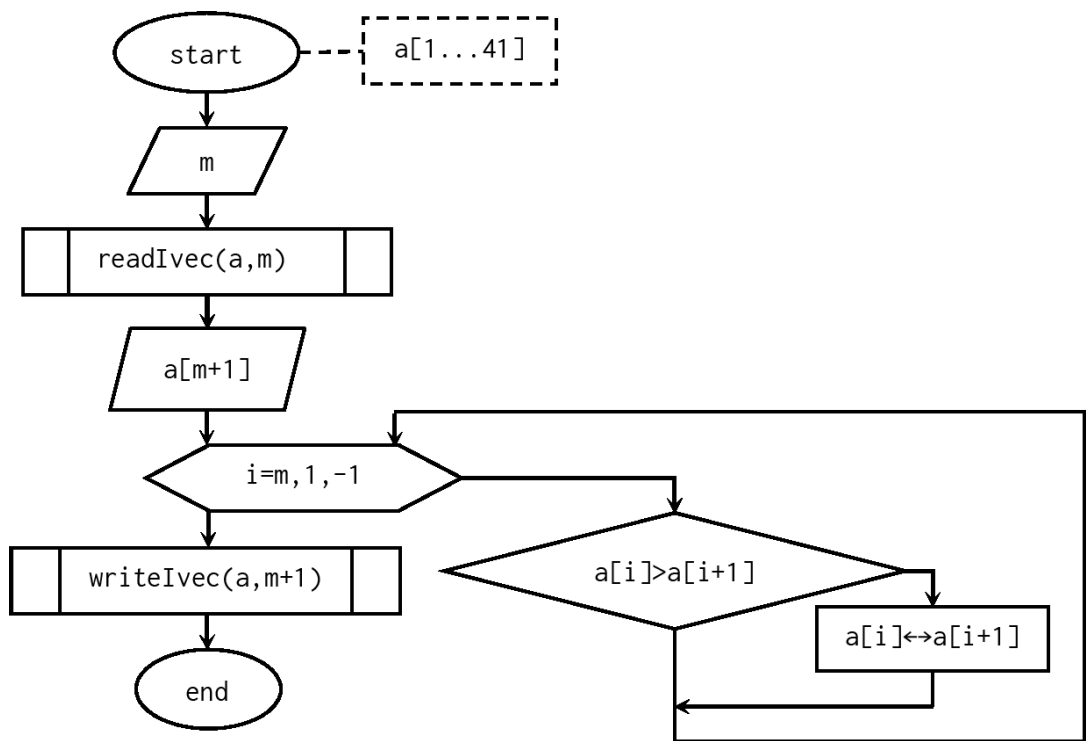


Fig. 8.14: Inserting a read number in an ascending array preserving the order.

8.14. Example. Write an algorithm to read the m entries of an integer array sorted in ascending order followed by the reading m . Then, read an integer and insert it inside the array preserving the ascending order. In other words, the $m + 1$ entries of the modified array are still in ascending order.

Solution. This problem is somehow similar to the algorithm of Figure 8.13(c). The main difference is that in Figure 8.13(c) it is known that the read number is to be inserted between the second and third entry; therefore, we can easily design the flowchart. However, the position in which the number is to be inserted is not clear in the present problem. A simple way is to place the read number in the $(m + 1)$ -st empty cell. Then use the first step of Flowchart 8.11(c) for m numbers here instead of the 40 numbers there. It means that we use the inner

for loop with the specification $i=40,1,-1$. As shown in **Flowchart 8.11(c)**, this loop which starts from $i = 40$ downwards, compares the entry a_i with a_{i+1} and swaps them if $a_i > a_{i+1}$ in each repetition. Accordingly, the entry a_{41} sits in the required place by necessary iterated one-back movements. **Flowchart 8.14** is as required.

We arrange the implementation table of **Flowchart 8.14** $m = 4$ and the read number as in **Table 8.14**.

Tab. 8.14: Implementation table for **Flowchart 8.14** for $m = 4$.

status	a[1]	a[2]	a[3]	a[4]	a[5]
before the loop	10	20	30	40	25
first repetition				25	40
second repetition			25	30	
third repetition					
fourth repetition					

8.14.1. Exercise. Write the codes of **Flowchart 8.14** in both languages C++ and Java.

8.14.2. Exercise. As demonstrated in **Table 8.14**, implementing the algorithm has no effect on the result from the third repetition onwards. Modify **Flowchart 8.14(a)** in such a way that these extra repetitions are not processed. In addition, write the programs of the obtained flowchart. Think of modifying the if template.

8.15. Example (The insertion sorting method). The insertion is another method of sorting a set of numbers in ascending or descending order. In this method, the numbers are read one by one and inserted inside the previously sorted numbers preserving the order. Write an algorithm to read first the size m of the integer array a having at most 40 entries. Then, read the entries of a one by one and sort them in ascending order using the insertion method.

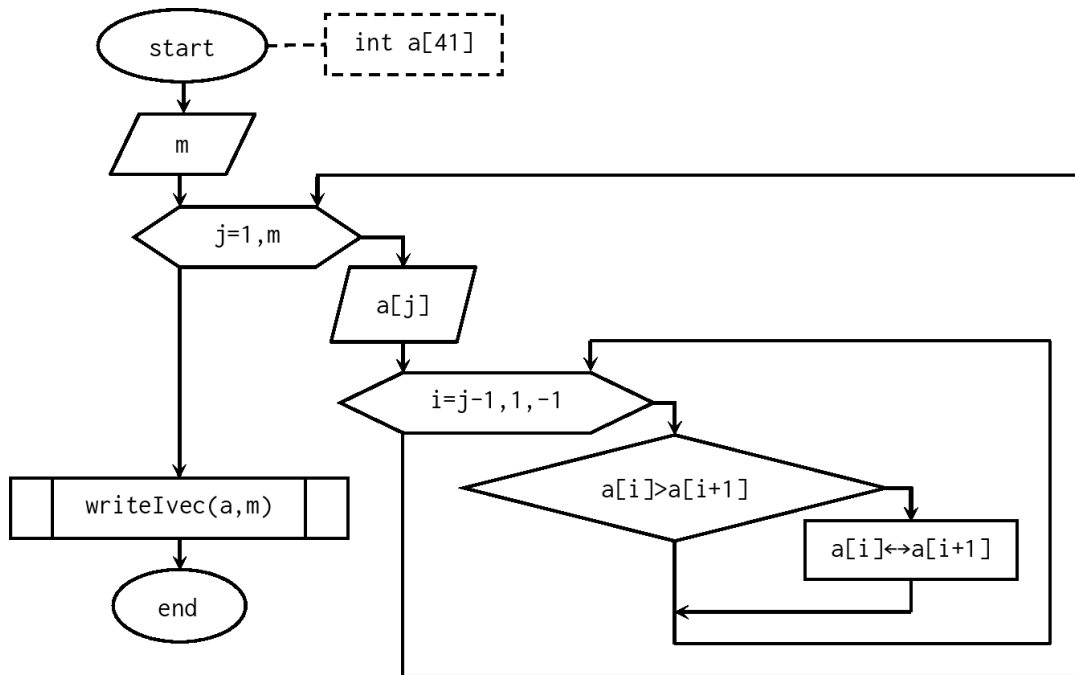


Fig. 8.15: Sorting an array in ascending order by the insertion sorting method.

Solution. The idea in [Flowchart 8.14](#) is used for this algorithm. To be precise, the entries are read one by one with a for template. Upon reading an entry, it is inserted inside the previously sorted numbers preserving the ascending order. The above discussion is summarized in [Flowchart 8.15](#). It should be mentioned that, the basic sorting is started from the second entry. Actually, the first entry is read due to the occurrence of incompatibility in the inner loop. Programs P8_15 are codes of this algorithm.

C++ codes:

```
// Program P8_15 to sort 20 entries
// of an array by insertion method
#include <iostream>
using namespace std;
void writeIvec(int u[], int n) {
    for (int i=1; i<=n; i++)
        cout<<u[i]<<" ";
    cout<<endl;
    return;
}
//*****
int main() {
    int a[41], i, j, t, m;
    cout<<"Entre the size of array: ";
    cin>>m;
    cout<<"Enter "<<m<<" integer "
    <<"entries one by one: \n";
    for (j=1; j<=m; j++) {
        cin>>a[j];
        for (i=j-1; i>=1; i--)
            if (a[i]>a[i+1]) {
                t=a[i];
                a[i]=a[i+1];
                a[i+1]=t;
            }
    }
    cout<<"The sorted entries: ";
    writeIvec(a, m);
    return 0;
}
```

Input/output:

```
Entre the size of array: 4↵
Enter 4 integer entries one by one:
2↵
9↵
6↵
3↵
The sorted entries: 2 3 6 9
```

Java codes:

```
// Program P8_15 to sort 20 entries
// of an array by insertion method
import java.util.Scanner;
class P8_15 {
    static void writeIvec(int u[], int n) {
        for (int i=1; i<=n; i++)
            System.out.print(u[i] + " ");
        System.out.println();
    }
    //*****
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int a[]=new int[41];
        int i, j, t=0, m;
        System.out.print("Enter the size of array: ");
        m=read.nextInt();
        System.out.println("Enter m integer "
            + "entries, one by one: ");
        for (j=1; j<=m; j++) {
            a[j]=read.nextInt();
            for (i=j-1; i>=1; i--) {
                if (a[i]>a[i+1]) {
                    t=a[i];
                    a[i]=a[i+1];
                    a[i+1]=t;
                }
            }
        }
        System.out.print("The sorted entries: ");
        writeIvec(a, m);
        read.close();
    }
}
```

Input/output:

```
Entre the size of array: 4↵
Enter 4 integer entries one by one:
2↵
9↵
6↵
3↵
The sorted entries: 2 3 6 9
```

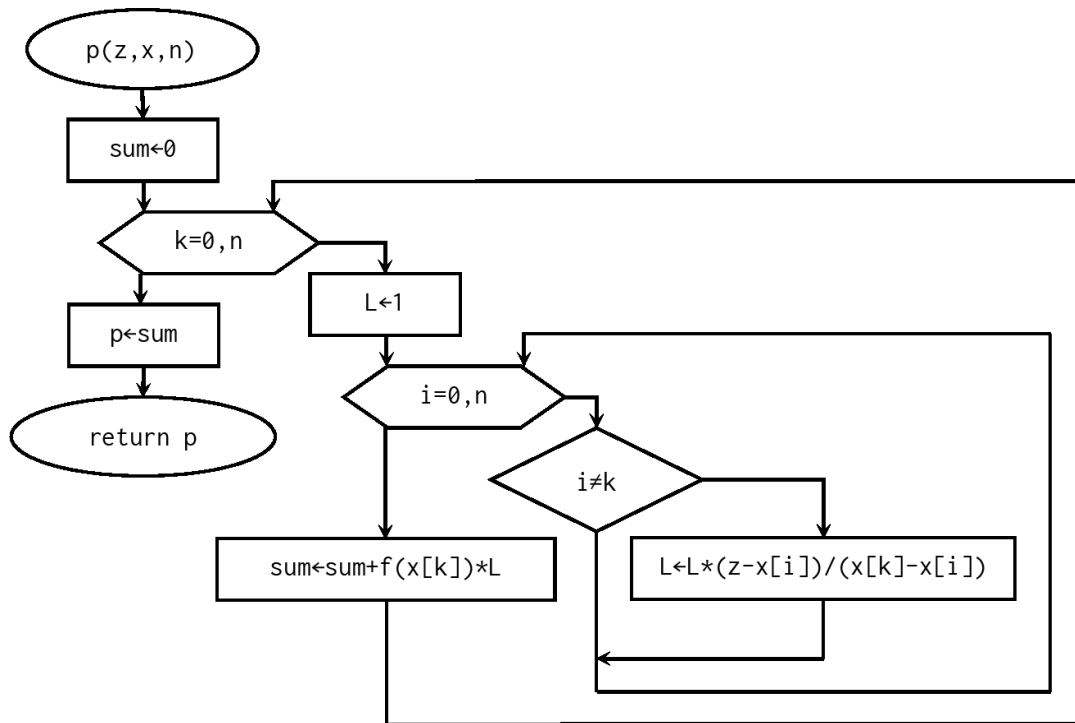


Fig. 8.16: The n -degree Lagrange interpolation polynomial.

8.15.1. Exercise. Modify the previous algorithm and the programs so that all the entries are read and then sorted in ascending order using the insertion method. How many arrays do you use? One or two? Attack to both cases!

8.15.2. Exercise. Split Flowchart 8.15 into a sub-algorithm and a main algorithm. In fact, the sub-algorithm receives an array, sorts it in ascending order using the insertion method, and then returns the sorted array.

8.16. Example (The Lagrange interpolation polynomial). One way to approximate the function f at a point z is to use the n -degree Lagrange interpolation polynomial of the function f based on the knots x_0, x_1, \dots, x_n at the point z which is defined as follows.

$$p(z) = \sum_{k=0}^n f(x_k)L_k(z), \quad L_k(z) = \prod_{\substack{i=0 \\ i \neq k}}^n \frac{z - x_i}{x_k - x_i}.$$

$$p(z) = \sum_{k=0}^n f(x_k)L_k(z), \quad L_k(z) = \prod_{\substack{i=0 \\ i \neq k}}^n \frac{z - x_i}{x_k - x_i}.$$

The knots should be accumulated in the neighborhood of the point z . Write a function to receive the knots x_0, x_1, \dots, x_n x_0, x_1, \dots, x_n as the entries of the $(n + 1)$ -entry real array x as well as the value of the point z and then calculate and return $p(z)$.

Solution. The structure of this algorithm is clear: we have an additive series the common term of which is a multiplicative series. Just note the conditional common term in the multiplicative series. The resulted function is depicted in [Figure 8.16](#). We use this method in the main unit of Programs P8_16 to approximate the function $f(x) = \sin(x) + \cos(x)$. To do this, the main unit first reads the value of n . Then, it reads the x_0, x_1, \dots, x_n . x_0, x_1, \dots, x_n . Finally, the value of z is read. Now, the method $p()$ is called for z as well as the n -entry array x and then the return value is printed. Eventually, the exact amount of $f(1)$ is computed and printed for comparison with the above approximated value.

C++ codes:

```
// Program P8_16 to approximate a
// function at a point by the
// Lagrange interpolation polynomial
#include <iostream>
#include <math.h>
#include <iomanip>
using namespace std;
inline double f(double x) {
    return sin(x)+cos(x);
}
//*****
double p(double z, double x[],
          int n) {
    double sum=0, L;
    for (int k=0; k<=n; k++) {
        L=1;
        for (int i=0; i<=n; i++)
            if (i!=k)
                L=L*(z-x[i])/(x[k]-x[i]);
        sum=sum+f(x[k])*L;
    }
    return sum;
}
//*****
int main() {
    int n;
    double z, x[21];
    cout<<"Enter a positive integer n: ";
    cin>>n;
    cout<<"Enter the notes x[0] to "
         <<"x["<<n<<"]: \n";
    for (int i=0; i<=n; i++)
        cin>>x[i];
    cout<<"Enter the point z: ";
    cin>>z;
    cout<<"The approximated amount: "
         <<p(z, x, n)<<endl;
    cout<<"The computer amount: "
         <<setprecision(8)<<f(1);
    return 0;
}

```

Input/output:**Java codes:**

```
// Program P8_16 to approximate a
// function at a point by the
// Lagrange interpolation polynomial
import java.util.Scanner;
class P8_16 {
    static double f(double x) {
        return Math.sin(x)+Math.cos(x);
    }
    //*****
    static double p(double z, double x[],
                    int n) {
        double sum=0, L;
        for (int k=0; k<=n; k++) {
            L=1;
            for (int i=0; i<=n; i++)
                if (i!=k)
                    L=L*(z-x[i])/(x[k]-x[i]);
            sum=sum+f(x[k])*L;
        }
        return sum;
    }
    //*****
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int n;
        double z, x[]=new double[21];
        System.out.print("Enter a positive "
                        + "integer n: ");
        n=read.nextInt();
        System.out.println("Enter the notes x[0] "
                          + "to x[" + n + "]: ");
        for (int i=0; i<=n; i++)
            x[i]=read.nextDouble();
        System.out.print("Enter the point z: ");
        z=read.nextDouble();
        System.out.println("The approxiamted "
                          + "amount: " + p(z, x, 5));
        System.out.print("The exact amount: "
                          + f(1));
        read.close();
    }
}

```

Input/output:


```

Enter a positive integer n: 5↵
Enter the notes x[0] to x[5]:
0.1 0.2 0.3 0.4 0.5 0.6↵
Enter the point z: 1↵
The approximated amount: 1.38188
The exact amount: 1.3817733

```

```

Enter a positive integer n: 5↵
Enter the notes x[0] to x[5]:
0.1 0.2 0.3 0.4 0.5 0.6↵
Enter the point z: 1↵
The approxiamted amount: 1.38188464485998
The exact amount: 1.3817732906760363

```

We studied two methods of writing algorithms for calculating the factorial of nonnegative integers, the recursive and direct ways in **Chapters 5** and **6**, respectively. The programs of these algorithms are practically usable for calculating the factorial of small numbers in most of the computers and compilers in both C++ and Java languages. In the next example we will use the arrays to write an algorithm usable for determining the factorial of large integers.

8.17. Example. Design a sub-algorithm named LargFact() to calculate and print the factorial of a rather large number m using the arrays.

Solution. Here, the usual formula is employed:

$$m! = 1 \times 2 \times 3 \times \dots \times m. \quad m! = 1 \times 2 \times 3 \times \dots \times m.$$

We use the idea behind the multiplication of a digit to a positive integer, taught to the students in the primary schools. Let us explain this manner for the multiplication of 859×7 . To have a clear explanation, insert a valueless zero to the left of 859, and put the digits of 0859 from left to right, respectively, into the 1-st, 2-nd, 3-rd, and 4-th entry of an array as follows.

1-st entry	2-nd entry	3-rd entry	4-th entry
0	8	5	9

The explanation is as follows.

1. Multiply the 4-th entry 9 to 7 (= 63); write the right digit **3**; hold on the transferring digit 6;

2. Multiply the 3-rd entry 5 to 7 and add the result to the previous transferring digit (= 41); write the right digit **1**; hold on the transferring digit 4;
3. Multiply the 2-nd entry 8 to 7 and add the result to the previous transferring digit (= 60); write the right digit **0**; hold on the transferring digit 6;
4. Multiply the 1-st entry 0 to 7 and add the result to the previous transferring digit (= 6); write the right digit **6**; hold on the transferring digit 0.

Now, if we stored the “right digits” in an array, say a , in each step from the 4-th entry downwards, we would have the following entries of a :

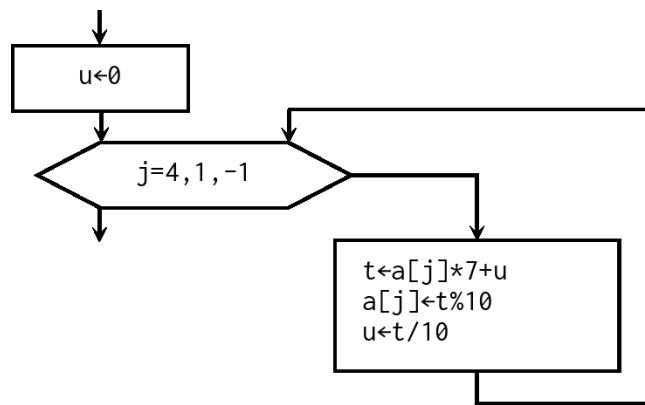


Fig. 8.17(a): The multiplication of an integer by 7 using the primary idea.

a_4	a_3	a_2	a_1
6	0	1	3

Arranging the entries from left to right, we get the real value of the multiplication $859 \times 7 = 6013$. Two points should be noticed here. First, we may use the above idea for any number instead of 7, provided that the necessary valueless zero digits are inserted to the left of 879. Second, nothing is lost if we take the original array the same as a .

Now, we steer the idea in the above multiplication, which is mentioned in four steps, to reach an algorithm. To do this, we need to approach some of the above actions to the equivalent actions from the algorithm viewpoint:

- Right digit: the remainder of the division by 10;
- Transferring digit: the quotient of the division by 10;

To continue, define some variables:

- t : the amount of the multiplication of an entry to 7 and adding the result to the previous transferring digit;
- u : the quotient of the division by 10.

Now, we rewrite the above-mentioned four steps in the literature of algorithm as follows.

1. Assign $a_4 \times 7$ **$a_4 \times 7$** to t ; substitute the remainder of $t / 10$ for a_4 ; **a_4** ; assign the quotient of $t / 10$ to u ;
2. Assign $a_3 \times 7 + u$ to t ; **$a_3 \times 7 + u$ to t** ; substitute the remainder of $t / 10$ for a_3 ; **a_3** ; assign the quotient of $t / 10$ to u ;
3. Assign $a_2 \times 7 + u$ to t ; **$a_2 \times 7 + u$ to t** ; substitute the remainder of $t / 10$ for a_2 ; **a_2** ; assign the quotient of $t / 10$ to u ;
4. Assign $a_1 \times 7 + u$ to t ; **$a_1 \times 7 + u$ to t** ; substitute the remainder of $t / 10$ for a_1 ; **a_1** ; assign the quotient of $t / 10$ to u .

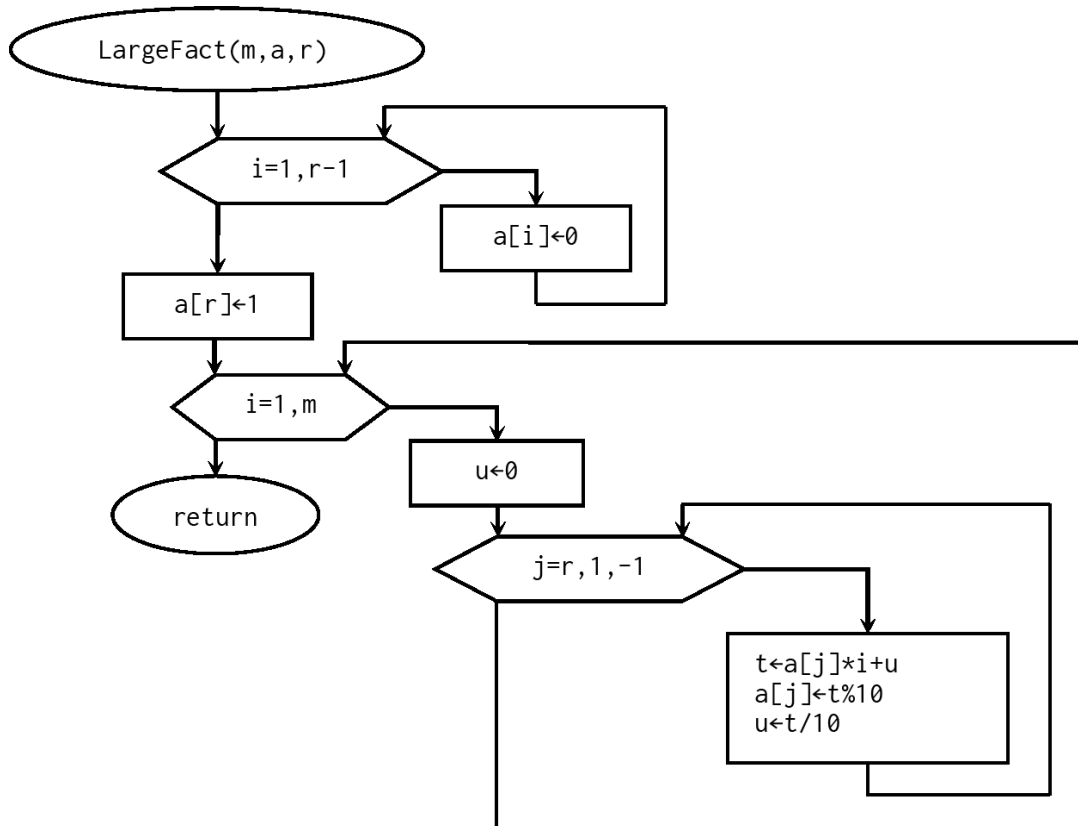


Fig. 8.17(b): Calculating the factorial of a large number and storing the digits in an array.

It is noteworthy that the constructions in the first step would be the same as the other steps by assigning the initial value zero to u . As shown, we have a for loop repeated four times and the above steps can now be settled down in **Flowchart 8.17(a)**.

Now, take $720 = 6!$ instead of the above number 859. Then, the multiplication above will be $6! = 720 \times 7 = 5040 = 7!$ Therefore, if we take the array a as:

a_4	a_3	a_2	a_1
0	0	0	1

and consider **Flowchart 8.17(a)** (with replacing 7 by i) as the range of an outer for loop with the specification $i=1,7$, we will get the amount of $5040 = 7!$ upon exiting this outer loop. To extend this for m instead of 7 and r instead of 4, we arrive at **Flowchart 8.17(b)** of the required sub-algorithm. The number m is a positive integer which we want to calculate its factorial and r is the number of digits of $m!$. This number is not often known and an upper bound should be guesstimated for it. The sub-algorithm `LargFact()` receives the positive integer m and the guesstimated integer r . Then it calculates and returns the array a including the digits of $m!$ in the natural order.

Table 8.17 thoroughly demonstrated the implementation table of the sub-algorithm `LargFact()` for $m = 5$ and $r = 3$.

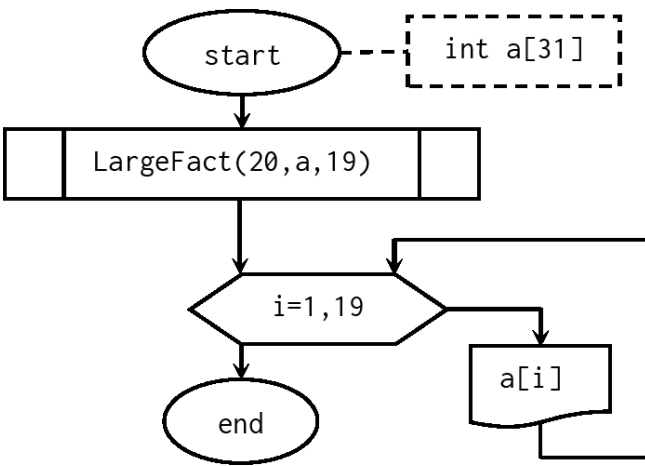


Fig. 8.17(c): Calculating 20! using the sub-algorithm `LargFact()`.

Tab. 8.17: Implementation table of the sub-algorithm `LargFact()` for $m=5$ and $r=3$.

status	i	j	t	u	a[1]	a[2]	a[3]
before the outer loop					0	0	1
before the inner loop	1			0			
first repetition		3	1				1
second repetition		2	0				
third repetition		1					
before the inner loop	2			0			
first repetition		3	2				2
second repetition		2	0				
third repetition		1					
before the inner loop	3			0			
first repetition		3	6				6
second repetition		2	0				
third repetition		1					
before the inner loop	4			0			
first repetition		3	24	2			4
second repetition		2	2	0		2	
third repetition		1	0				
before the inner loop	5			0			
first repetition		3	20	2			0
second repetition		2	12	1		2	
third repetition		1	1	0	1		

In the main algorithm of [Flowchart 8.17\(c\)](#), the amount of $20!$, is calculated, using the sub-algorithm `LargeFact()`. Of course, we know that $20!$ is a 19-digit number. These digits, which are the entries of the array a , are printed successively, without any space. This may be performed in any programming language.

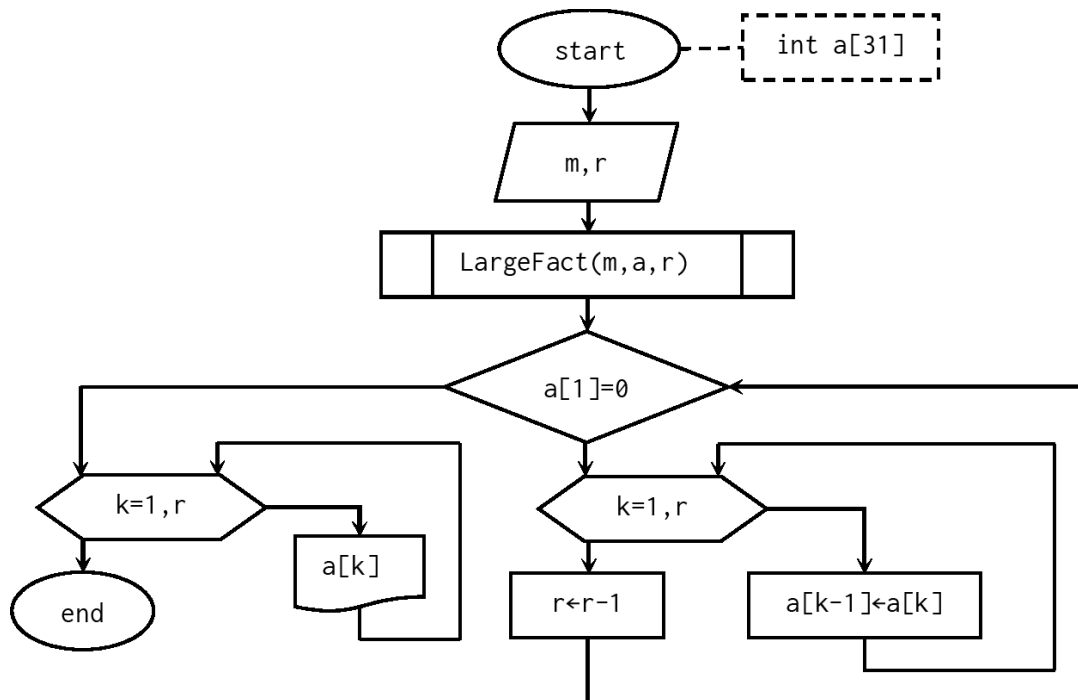


Fig. 8.17(d): Calculating the $m!$ (for large m) and printing it removing the leftmost zero digits.

8.17.1. Exercise. Write the complete programs of [Flowchart 8.17\(c\)](#) in both C++ and Java codes.

It is worth mentioning that, in general, the number of digits of $m!$ is not always known and an upper bound should be guessed for this purpose. Some valueless zeroes mainly appear on the left of the real amount of $m!$ For example, we will have the following output if we write 22 instead of 19 in the main algorithm of [Figure 7,18\(c\)](#):

0002432902008176640000

The main algorithm in [Figure 8.17\(d\)](#), reads the values of m and r and then receives the amount of $m!$ within the r -entry array a by calling the sub-algorithm `LargeFac()`. Next, using a while template, while the first entry of a (the leftmost digit of $m!$) is zero, it is removed and then

all the other entries are pulled one back in each repetition. Finally, the guesstimated number of digits, r , is decreased by 1.

The codes of **Flowchart 8.17(d)** in both C++ and Java languages can be observed in Programs P8_17.

C++ codes:

```
// Program P8_17 to calculate the
// factorial of large integers
// using the subprogram LargFact()
#include <iostream>
using namespace std;
void LargeFact(int, int [], int);
int main() {
    int a[31];
    int m, r, k;
    cout<<"Enter the number m: ";
    cin>>m;
    cout<<"Guess a maximum length for "
         <<m<<"!:";
    cin>>r;
    LargeFact(m, a, r);
    while (a[1]!=0) {
        for (k=1; k<=r; k++)
            a[k-1]=a[k];
        r=r-1;
    }
    cout<<m<<"!=";
    for (k=1; k<=r; k++)
        cout<<a[k];
    return 0;
}
//*****
void LargeFact(int m, int a[],
               int r) {
    int u, t, i, j;
    for (i=1; i<=r-1; i++)
        a[i]=0;
    a[r]=1;
    for (i=1; i<=m; i++) {
        u=0;
        for (j=r; j>=1; j--) {
            t=a[j]*i+u;
            a[j]=t%10;
            u=t/10;
        }
    }
    return;
}
```

Java codes:

```
// Program P8_17 to calculate the
// factorial of large integers
// using the method LargFact()
import java.util.Scanner;
class P8_17 {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int a[]=new int[31];
        int m, r, k;
        System.out.print("Enter the number m: ");
        m=read.nextInt();
        System.out.print("Guess a maximum "
            + "length for " + m + "!:" );
        r=read.nextInt();
        LargFact(m, a, r);
        while (a[1]!=0) {
            for (k=1; k<=r; k++)
                a[k-1]=a[k];
            r=r -1;
        }
        System.out.print(m + "!=");
        for (k=1; k<=r; k++)
            System.out.print(a[k]);
        read.close();
    }
    //*****
    static void LargFact(int m, int a[],
                        int r) {
        int u, t, i, j;
        for (i=1; i<=r-1; i++)
            a[i]=0;
        a[r]=1;
        for (i=1; i<=m; i++) {
            u=0;
            for (j=r; j>=1; j--) {
                t=a[j]*i+u;
                a[j]=t%10;
                u=t/10;
            }
        }
    }
}
```



```
}
```

Input/output:

```
Enter the number m: 27↵  
Guess a maximum length for 27!: 30↵  
27!=10888869450418352160768000000
```

```
}
```

Input/output:

```
Enter the number m: 27↵  
Guess a maximum length for 27!: 30↵  
27!=10888869450418352160768000000
```

A point about the length (upper bound) of the guesstimated value should be noticed.



Extra care should be taken upon assigning the guessed length! We will not achieve a correct answer if this length is smaller than the necessary one.

8.17.2. Exercise. Another way for preventing the appearance of the meaningless digits on the left side of the number is to replace a possible meaningless zero digit with a space character in the main algorithm. Write a main algorithm to perform this print.

Exercises

In the following exercises: (1) Arrange the implementation table, if needed, (2) Write the complete program, (3) Provide appropriate input notifications and output headings, if any. In addition, the user-defined functions in the text of the current and the previous chapters may be used unless otherwise is explicitly specified.

8.1. Write an algorithm to produce the first 20 terms of the Fibonacci sequence and save them in the array F . Then, read the positive integer n (which is less than 20) and calculate and print the sum

$$\sum_{i=1}^n iF_i, \sum_{i=1}^n iF_i,$$

where F_i is the i -th term of the Fibonacci sequence.

8.2. Write a sub-algorithm named `FibArray()` to receive a positive integer n , calculate the first n terms of the Fibonacci sequence, and return them as the entries of an integer array, say F . Then, write a main algorithm to calculate and print the first 20 terms of the Fibonacci sequence using the above sub-algorithm.

8.3. Write a sub-algorithm to receive the n -entry integer array a and an integer t . Then determine and return the first position of the occurrence of t in the array. If t does not occur in the array, announce its position as 0.

8.4. Write an algorithm to read the ID number and grade of 40 students and then store them in the arrays N and G , respectively. Next, read a grade g and print the Not in list message if it is not in the list of grades. Otherwise, print the ID numbers with this grade and the number of these students.

8.5. Write an algorithm to read 20 integers, sort them in ascending order after removing the repeating elements, and print the sorted numbers.

8.6. Write an algorithm to read the 20-entry integer arrays a and b . Then, put those entries in the array a , which are not equal to any entry of the array b in the array c . Repetition is not allowed. Finally, print the three arrays separately.

8.7. Write an algorithm to read the positive integers m and n which are at most 100. Then, read the m -entry array A and n -entry array B and combine them in a new array C so that it is discarded if a number from B exists in A . Finally, print C . An example is demonstrated below.

$A : 3, 7, 2, 5, 9, 3, 6$

$B : 12, 4, 6, 2, 3, 9, 8$

$C : 3, 7, 2, 5, 9, 3, 6, 12, 4, 8$

$A: 3, 7, 2, 5, 9, 3, 6$

$B: 12, 4, 6, 2, 3, 9, 8$

$C: 3, 7, 2, 5, 9, 3, 6, 12, 4, 8$

8.8. In [Exercise 8.7](#), sort the resulted numbers in both ascending and descending orders and then print them separately.

8.9. Write a function named `repeat()` to receive the n -entry integer array a and the integer x and calculate and return the number of repetitions of x in the array.

8.10. Write an algorithm to read the entries of a 20-entry integer array and determine the first repeated number, its position, and the number of its repetitions. Then, print them separately. An appropriate message should be printed if there is no such number.

8.11. Write a sub-algorithm to receive an integer array with n entries named u . Then, determine and return the number with the most repetition. Additionally, determine and print the largest one if there is more than one entry with this property. For example, the output for the following inputs should be 7 since it is the last number which is most repeated (three times).

2 4 7 4 9 4 6 6 7 1 2 6 7 1
2 4 7 4 9 4 6 6 7 1 2 6 7 1

8.12. Divide Algorithm 8.11, sorting in ascending order by the bubble sorting method, into a sub-algorithm named Sort and a main algorithm. In other words, the sub-algorithm Sort() receives an n -entry array, sorts it in the ascending order by the bubble sorting method, and then returns the sorted array.

8.13. Write an algorithm to read the 40 entries of an integer array and, using the bubble sorting method, sort the first 20 entries in ascending order and the second 20 entries in descending order and print them separately. The entries should be sorted into ascending and descending orders simultaneously and in the same loop.

8.14. Write an algorithm to read two 20-entry integer arrays a and b which are sorted in a descending order. Then, combine them in the 40-entry array c and sort c in ascending order. Finally, print the sorted array c .

8.15. Write a sub-algorithm named Rank() to receive the ID number and grade of n students with the names N and G , respectively. Then, determine a rank for each student and return the ranks in an integer array, say R . For the same grades consider the same ranks. Next, write a main algorithm to read the ID number and grade of 40 students and then print three separate lists by calling the sub-algorithm rank(): the first list in the ascending order of the ID numbers (the array N), the second one in the ascending order of the grades (the array G), and finally, the third list in the ascending order of the ranks (the array R). Every list contains 40 rows and in each row, the ID number, grade, and rank of each student are positioned, respectively.

8.16. Write an algorithm to read 40 entries of a real array a . Then read the integer m ($1 \leq m \leq 39$) and the real number t and insert the number t between the m -th and $(m + 1)$ -th entries of the array a .

8.17. Write an algorithm to read 40 real numbers and save them in the real array a . Then, read 7 real numbers one by one and insert them between the fifth and sixth entry.

8.18. Write an algorithm to read the 40 entries of the real array a . Then, read a positive integer q ($1 \leq q \leq 19$) as the count of the numbers which will be inserted. Afterwards, in q steps, each step read the two numbers m and t and insert t in the position between m and $m + 1$ in the array ($1 \leq m \leq q$). Finally, print the new $(40 + q)$ -entry array a .

8.19. Write an algorithm to read the 20 entries of an ascending real array a . Then, read a number n ($1 \leq n \leq 19$) as the count of numbers which will be read. Next, in n steps, each step read a real number and insert it in the array a so that the ascending order of the array is preserved.

8.20. Repeat [Exercise 8.12](#) for the insertion sorting method.

8.21. Write a sub-algorithm named `Bin()` to receive the positive integer n , convert it to the base-2 numeral system, and return the result in the k -entry integer array e . Compare the solution to the sub-algorithm of [Figure 7.6\(b\)](#) which is written without using the arrays.

8.22. Write a sub-algorithm to receive the positive integer k , transfer it to the base-16 numeral system, and return the result to the calling unit. Then, write a main algorithm to read a positive integer k , convert it to the base-16 numeral system, and print it using the above sub-algorithm.

8.23. Write an algorithm to read the 20 entries of the integer array a . Then, calculate and print the greatest common divisor of the entries. Finally, compare this exercise with [Exercise 7.21](#) which is written without using the arrays.

Supplementary exercises

8.1*. Write an algorithm to read the ID number and grade of 40 students with the names N and G , respectively. Then, store the ranks of the students with the name R using the sub-algorithm `rank()` in [Exercise 8.15](#). Next, ask the user to enter one of the following numbers for the stated purposes within the input announcement as follows:

Enter one of the following numbers:

- 0: exit the program
- 1: print an ascending list of the array N together with G and R
- 2: print a descending list of the array G together with N and R
- 3: print an ascending list of the array R together with N and G
- 4: read an ID number and print it together with its grade and rank

After entering one of the above numbers, the algorithm performs the task stated in front of that number and the implementation continues until the user enters the zero number. If the user enters any number other than the above-mentioned numbers, the algorithm prints the message Wrong number and continues from the beginning.

8.2*. Repeat the previous exercise, this time for the following input announcement:

Enter one of the following numbers:

- 0: exit the program
- 1: print the average of the grades
- 2: print the standard deviation of the grades
- 3: print the maximal grade and the related ID number(s)

- 4: print the minimal grade and the related ID number(s)
- 5: print the number of grades equal to the average
- 6: print the number of grades less than the average
- 7: print the number of grades greater than the average
- 8: print the grade with the most repetition and the related ID numbers

8.3*. Write an algorithm to read thirty real numbers and determine whether or not these numbers are orderly sorted and if so, are they sorted in ascending or descending order? It is supposed that not all the numbers are zero.

8.4*. Write an algorithm to read the entries of the 40-entry integer array a . Then, read an integer x and remove that entry and pull all the subsequent entries one back if it matches an entry of the array. Finally, print the deformed array a with possibly less entries.

8.5*. Write an algorithm to read the entries of a 20-entry integer array and remove its repeated entries without using another array. Then, print the deformed array. For example:

input array: 2 5 6 5 2 3 7 6
 output array: 2 5 6 3 7

8.6*. Write an algorithm to read the entries of the 40-entry integer array a . Then determine the repeated entries of this array followed by the number of repetition for each case. Repetition for the repeated entries is not allowed.

8.7*. Write a sub-algorithm named `prim_dec()` to read a positive integer n . Then, in the primary decomposition

$$n = p_1^{r_1} p_2^{r_2} \dots p_k^{r_k} \quad n = p_1^{r_1} p_2^{r_2} \dots p_k^{r_k}$$

store the prime factors in an array named p and the multiplicity of each factor in another array named r .

8.8*. Consider the two numbers m and n having the following primary decompositions:

$$m = p_1^{r_1} p_2^{r_2} \dots p_k^{r_k}, \quad n = p_1^{s_1} p_2^{s_2} \dots p_k^{s_k}.$$

$$m = p_1^{r_1} p_2^{r_2} \dots p_k^{r_k}, \quad n = p_1^{s_1} p_2^{s_2} \dots p_k^{s_k}.$$

Then, their greater common divisor and their least common multiple (gcd and lcm, respectively) are obtained from the formulas

$$\text{gcd} = p_1^{u_1} p_2^{u_2} \dots p_k^{u_k}, \quad \text{lcm} = p_1^{v_1} p_2^{v_2} \dots p_k^{v_k},$$

$$\text{gcd} = p_1^{u_1} p_2^{u_2} \dots p_k^{u_k}, \quad \text{lcm} = p_1^{v_1} p_2^{v_2} \dots p_k^{v_k},$$

where, for any i ($1 \leq i \leq k$), u_i is the minimum of r_i and s_i , and v_i is the maximum of r_i and s_i . Write an algorithm to read the two numbers m and n and then calculate and print their gcd and lcm using the mentioned manner.

8.9*. For a positive integer n , the Mobius function μ is defined as follows.

$$\mu(n) = \begin{cases} 1 & \text{if } n = 1 \\ 0 & \text{if } p^2 | n, \text{ for some prime } p, \\ (-1)^k, & \text{if } n = p_1 p_2 \dots p_k, \text{ for distinct primes } p_i \\ 1, & \text{if } n = 1, \\ 0, & \text{if } p^2 | n, \text{ for some prime } p, \\ (-1)^k, & \text{if } n = p_1 p_2 \dots p_k, \text{ for distinct primes } p_i. \end{cases}$$

Write a function, named `mu()`, to receive the positive integer n and then calculate and return the Mobius function $\mu(n)$.

8.10*. Write an algorithm to read the degrees m and n of two polynomials

$$f(x) = f_0 + f_1x + \cdots + f_mx^m, \quad g(x) = g_0 + g_1x + \cdots + g_nx^n.$$

$$f(x) = f_0 + f_1x + \cdots + f_mx^m, \quad g(x) = g_0 + g_1x + \cdots + g_nx^n.$$

Then, read the coefficients of the functions f and g , separately. Finally, calculate their sum and print it in the standard form of a polynomial. The following is an example for the output:

$$f(x)+g(x)=5+7*x^2+2*x^5$$

8.11*. Repeat the previous exercise, this time for multiplication instead of summation. Recall that for the two polynomials $f(x)$ and $g(x)$ as in [Exercise 8.10*](#), the multiplication is defined as the function h by the following rules.

$$h(x) = \sum_{i=1}^{m+n} h_i x_i, \quad h_i = \sum_{s=0}^i f_s g_{i-s}.$$

$$h(x) = \sum_{i=1}^{m+n} h_i x^i, \quad h_i = \sum_{s=0}^i f_s g_{i-s}.$$

8.12*. Given the positive integers a and n , an efficient technique for calculating a^n is to transfer n to the base-2 numeral system:

$$n = \sum_{i=0}^k e_i 2^i.$$

$$n = \sum_{i=0}^k e_i 2^i.$$

Now we can write:

$$b = a^n = \prod_{i=0}^k a^{e_i 2^i},$$
$$b = a^n = \prod_{i=0}^k a^{e_i 2^i}.$$

Write an algorithm to read the positive integers a , n , and m and then calculate and print a^n modulo m .

8.13*. Solve **Exercise 7.7*** using the arrays.

9 Two-dimensional arrays

At the beginning of [Chapter 8](#), some information was provided regarding introducing one-dimensional arrays in algorithms and programs. This information is also true for two-dimensional arrays, which we will refer to as matrices from now on. In particular, as the one-dimensional case, the two-dimensional arrays, as predefined objects, pass by reference. The only difference is that here, we are dealing with double indices instead of a single index and each index is used inside a pair of []. In this chapter, matrices are studied in more details and then solving linear equations systems are examined.

9.1 Matrices

The first step in matrix related programming is how to read and write the matrices. In other words, how to read matrices row by row, as they are, from the input and write them based on the matrix structure in the output. To this end the simplest pattern is to use two nested for loops for row-reading the matrix $A = (A_{ij})_{m \times n}$ in the C++ and Java codes as follows:

C++ codes:

```
for (int i=1; i<=m; i++) {  
    for (int j=1; j<=n; j++)  
        cin>>A[i][j];  
}
```

Java codes:

```
for (int i=1; i<=m; i++)  
    for (int j=1; j<=n; j++)  
        A[i][j]=read.nextInt();
```

This pattern satisfies our need for row-reading (reading row by row) matrices although matrices can be entered in any form by running this part provided that the row order of the entries is kept. For example, all

the rows of a matrix can be continuously entered in a single input row. In addition, each entry could enter in one row.

From now on, the following subprograms named readImat() and readFmat() are used to read the integer and real (floating point) matrices, respectively:

C++ codes:

```
void readImat(int A[][6],
              int m, int n) {
    for (int i=1; i<=m; i++) {
        for (int j=1; j<=n; j++)
            cin>>A[i][j];
    }
}
```

Java codes:

```
static void readImat(int A[][],
                    int m, int n) {
    Scanner read=new Scanner(System.in);
    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
            A[i][j]=read.nextInt();
    read.close();
}
```

C++ codes:

```
void readFmat(float A[][6],
              int m, int n) {
    for (int i=1; i<=m; i++) {
        for (int j=1; j<=n; j++)
            cin>>A[i][j];
    }
}
```

Java codes:

```
static void readFmat(float A[][],
                    int m, int n) {
    Scanner read=new Scanner(System.in);
    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
            A[i][j]=read.nextInt();
    read.close();
}
```

The rule of multi-using the reading methods in **Chapter 8** is held for two-dimensional arrays. Moreover, the following notes should be highlighted regarding the two-dimensional arrays.



- Whenever two-dimensional arrays are applied as the parameters of a subprogram in the C++ language, the length of the first dimension can be written in an open way [] while the length of the second dimension should be determined. Further, we may change the value of the length of the second dimension depending on our need upon calling the involved subprogram.
- The lengths of both dimensions may be written in an open way [][] in Java language.
- The values of the lengths within the integer variables should be part of the parameters in both languages.

- The real length of both dimensions should be determined in the main program.
-

Now, the writing pattern of a matrix in the form of matrix structure (row by row) is investigated. Consider the following parts for writing the matrix $A = (A_{ij})_{m \times n}$, $A = (A_{ij})_{m \times n}$, in the C++ and Java codes:

C++ codes:

```
for (int i=1; i<=m; i++) {
    for (int j=1; j<=n; j++)
        cout<<A[i][j];
```

Java codes:

```
for (int i=1; i<=m; i++) {
    for (int j=1; j<=n; j++)
        System.out.print(A[i][j]);
```

Then, all the entries are written in a single row, which is not what we desire. Therefore, we should break a row after exiting it using the `cout<<endl` statement in C++ (the `System.out.println()` statement in Java) and then transfer it to the beginning of the next row:

C++ codes:

```
for (int i=1; i<=m; i++) {
    for (int j=1; j<=n; j++)
        cout<<A[i][j];
    cout<<endl;
```

Java codes:

```
for (int i=1; i<=m; i++) {
    for (int j=1; j<=n; j++)
        System.out.print(A[i][j]);
    System.out.println();
```

Suppose that a 2×4 matrix is previously saved in the memory as follows.

a_{11}	a_{12}	a_{13}	a_{14}	a_{21}	a_{22}	a_{23}	a_{24}			
1	2	3	4	5	6	7	8			
a_{11}	a_{12}	a_{13}	a_{14}	a_{21}	a_{22}	a_{23}	a_{24}			
1	2	3	4	5	6	7	8			

Then, running the above parts of the program result in the following output:

1234

5678

Furthermore, we can add one or several space characters after writing $A[i][j]$ in order to create spaces between the entries:

C++ codes:

```
cout<<A[i][j]<<" ";
cout<<endl;
```

Java codes:

```
for (int i=1; i<=m; i++) {
  for (int j=1; j<=n; j++)
    System.out.print(A[i][j] + " ");
  System.out.println();
}
```

However, the problem generally fails to be solved by doing this. For example, if we had the number 22222 instead of the entry a_{ij} in the above matrix, then we would end up with the following scrambled print even with the use of space characters:

```
1__22222__3__4
5__6__7__8
```

We use an appropriate formats for spacing in prints in order to solve this problem. For example, if we use the formats:

C++ codes:

```
cout<<setw(7)<<A[i][j];
```

Java codes:

```
System.out.format("%7d", A[i][j]);
```

in the above parts, then we will have the following output:

```
_____1__22222_____3_____4
_____5_____6_____7_____8
```

This visualization satisfies our desire. Generally, from now on, we use the following subprograms named writeImat() for printing the integer $m \times n$ matrices:

C++ codes:

```
void writeImat(int A[][6],
              int m, int n) {
    for (int i=1; i<=m; i++) {
        for (int j=1; j<=n; j++)
            cout<<setw(7)<<A[i][j];
        cout<<endl;
    }
}
```

Java codes:

```
static void writeImat(int A[][],
                    int m, int n) {
    for (int i=1; i<=m; i++) {
        for (int j=1; j<=n; j++)
            System.out.format("%7d", A[i][j]);
        System.out.println();
    }
}
```

Moreover, the following subprograms named writeFmat() are used for printing the real $m \times n$ matrices:

C++ codes:

```
void writeFmat(float A[][6],
              int m, int n) {
    for (int i=1; i<=m; i++) {
        for (int j=1; j<=n; j++)
            cout<<setw(10)<<A[i][j];
        cout<<endl;
    }
}
```

Java codes:

```
static void writeFmat(float A[][],
                    int m, int n) {
    for (int i=1; i<=m; i++) {
        for (int j=1; j<=n; j++)
            System.out.format("%10f", A[i][j]);
        System.out.println();
    }
}
```



The width of the spacing format in the above patterns of printing is changed depending on the size of the necessary length.

9.1. Example. Write an algorithm to produce and print the 5×5 identity matrix in the name of U .

Solution. The identity matrix is defined in a way that the entries on the main diagonal are considered 1 while the remaining entries are 0. We have two variable indices (rows and columns) in this definition. Therefore, we produce the identity matrix inside two nested for loops in as displayed in **Flowchart 9.1(a)**.

Programs P9_1_A translate **Flowchart 9.1(a)** into the C++ and Java codes. The feature of the ? operator is used in these programs.

C++ codes:

```
// Program P9_1_A to produce and
//print the identity 5 * 5 matrix
#include <iostream>
#include <iomanip>
using namespace std;
void writeImat(int [][][6], int, int);
int main() {
    int U[6][6];
    for (int i=1; i<=5; i++)
```

Java codes:

```
// Program P9_1_A to produce and
//print the identity 5 * 5 matrix
class P9_1_A {
    public static void main(String[] args) {
        int U[][]=new int[6][6];
        for (int i=1; i<=5; i++)
            for (int j=1; j<=5; j++)
                U[i][j]=(i==j) ? 1 : 0;
        System.out.println("The identity 5 * 5 ")
```

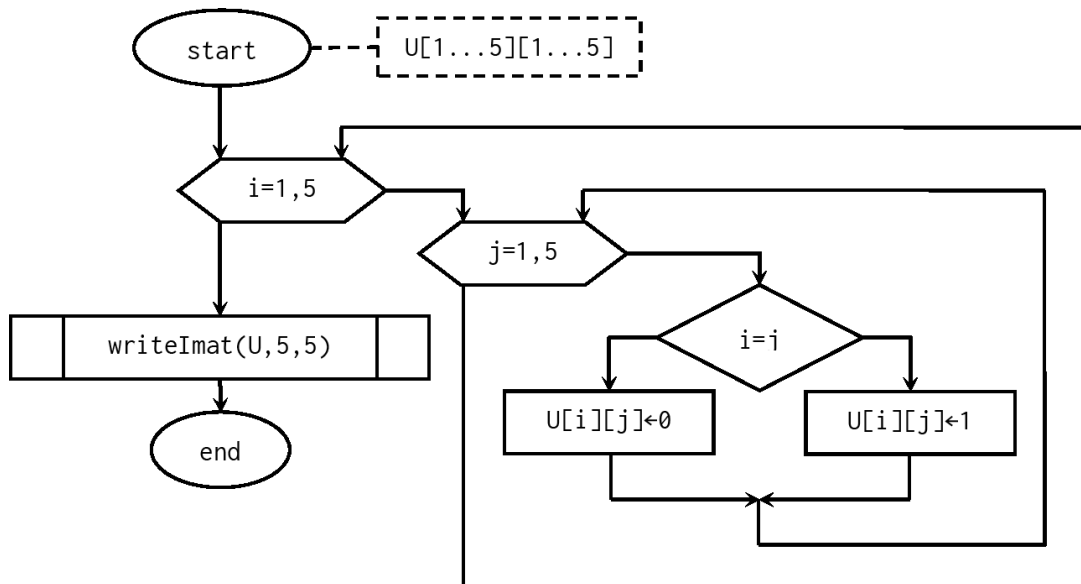


Fig. 9.1(a): Producing the 5 × 5 identity matrix.


```

    for (int j=1; j<=5; j++)
    U[i][j]=(i==j) ? 1 : 0;
    cout<<"The identity 5 * 5 matrix:\n";
    writeImat(U, 5, 5);
    return 0;
}
//*****
void writeImat(int A[][6],
               int m, int n) {
    for (int i=1; i<=m; i++) {
        for (int j=1; j<=n; j++)
            cout<<setw(3)<<A[i][j];
        cout<<endl;
    }
}

```

Output:

```

The identity 5 * 5 matrix:
 1  0  0  0  0
 0  1  0  0  0
 0  0  1  0  0
 0  0  0  1  0
 0  0  0  0  1

```

```

    + "matrix:");
    writeImat(U, 5, 5);
}
//*****
static void writeImat(int A[][],
                     int m, int n) {
    for (int i=1; i<=m; i++) {
        for (int j=1; j<=n; j++)
            System.out.format("%-3d", A[i][j]);
        System.out.println();
    }
}

```

Output:

```

The identity 5 * 5 matrix:
1  0  0  0  0
0  1  0  0  0
0  0  1  0  0
0  0  0  1  0
0  0  0  0  1

```

Now, a sub-algorithm named `idImat()` is written to produce the $n \times n$ identity matrix and then return it within the integer array `U`. **Flowchart 9.1(b)** is the required sub-algorithm.

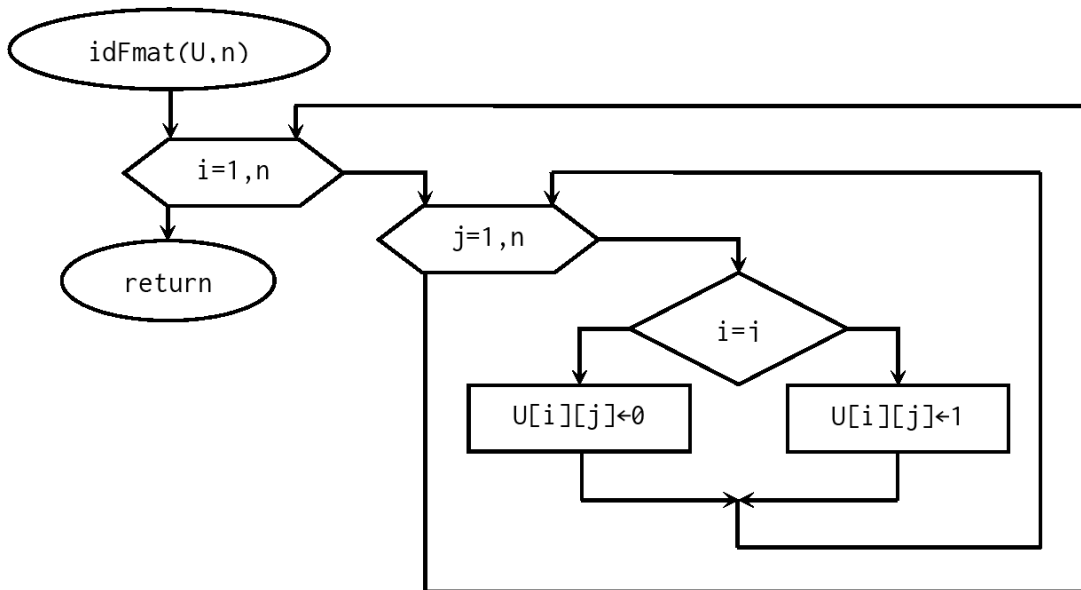


Fig. 9.1(b): A sub-algorithm to produce the $n \times n$ identity matrix.

Each of the Programs P9_1_B first reads the size of the identity matrix with the name n which is assumed to be at most 10. Then, it prints the $n \times n$ identity matrix calling the subprogram `idImat()`. Eventually, the subprograms `idImat` in both programs are written without using the `?` operator.

C++ codes:

```
// Program P9_1_B to produce and
// print the identity 5 * 5 matrix
// using the idImat subprogram
#include <iostream>
#include <iomanip>
using namespace std;
void writeImat(int A[][11], int, int);
float idImat(int A[][11], int);
int main() {
    int n, U[11][11];
    cout<<"Enter the size of "
        <<"identity matrix: ";
    cin>>n;
    idImat(U, n);
    cout<<"The identity n * n matrix:\n";
    writeImat(U, n, n);
    return 0;
}
//*****
void writeImat(int A[][11],
               int m, int n) {
    for (int i=1; i<=m; i++) {
        for (int j=1; j<=n; j++)
            cout<<setw(3)<<A[i][j];
        cout<<endl;
    }
}
```

Java codes:

```
// Program P9_1_B to produce and
// print the identity 5 * 5 matrix
// using the idImat subprogram
import java.util.Scanner;
class P9_1_B {
    public static void main(String[] args) {
        Scanner read=new Scanner(System.in);
        int n, U[][]=new int[11][11];
        System.out.print("Enter the size of "
            + "identity matrix: ");
        n=read.nextInt();
        System.out.println("The identity n * n "
            + "matrix:");

        idImat(U, n);
        writeImat(U, n, n);
        read.close();
    }
    //*****
    static void writeImat(int A[][],
                          int m, int n) {
        for (int i=1; i<=m; i++) {
            for (int j=1; j<=n; j++)
                System.out.format("%-3d", A[i][j]);
            System.out.println();
        }
    }
}
```

```

} //*****
//*****
float idImat(int U[][11], int n) {
    for (int i=1; i<=n; i++)
        for (int j=1; j<=n; j++)
            if (i==j)
                U[i][j]=1;
            else
                U[i][j]=0;
}

//*****
static void idImat(int U[][], int n) {
    for (int i=1; i<=n; i++)
        for (int j=1; j<=n; j++)
            if (i==j)
                U[i][j]=1;
            else
                U[i][j]=0;
}
}

```

Input/output:

```

Enter the size of identity ma-trix: 3↵
The identity n × n ma-trix:
 1  0  0
 0  1  0
 0  0  1

```

Input/output:

```

Enter the size of identity ma-trix: 3↵
The identity n × n ma-trix:
 1  0  0
 0  1  0
 0  0  1

```

Recall that the arrays are passed by reference in both C++ and Java languages. This is why we write no return value in the return instruction at the end of the sub-algorithms. Nevertheless, we do write the return instruction in sub-algorithms to keep their standard structure although the return statement is not written in the subprograms.

The main process of the sub-algorithm idImat() is to produce the identity matrix, denoted in the standard texts by I , and assign it to the integer matrix U . This is analogous to the assignment instruction:

$$U \rightarrow IU \leftarrow I$$

A float version idFmat() of the subprogram idImat() may be written by replacing the data type float instead of int.

An alternative method to produce the identity matrix is to define the entries U_{ij} as 0 in the range of an inner for loop and define the diagonal entry U_{ii} as 1 after exiting that loop. This idea is directly used in [Figure 9.1\(c\)](#) to produce a 5×5 identity matrix in an algorithm. The codes related to the algorithm of [Figure 9.1\(c\)](#) can be found in Programs P9_1_C.

C++ codes:

```
// Program P9_1_C to produce and
// print the identity 5 * 5 matrix
// using an alternative way
#include <iostream>
#include <iomanip>
using namespace std;
void writeImat(int [][][6], int, int);
```

Java codes:

```
// Program P9_1_C to produce and
// print the identity 5 * 5 matrix
// using an alternative way
class P9_1_C {
    public static void main(String[] args) {
        int U[][]=new int[6][6];
        for (int i=1; i<=5; i++) {
```

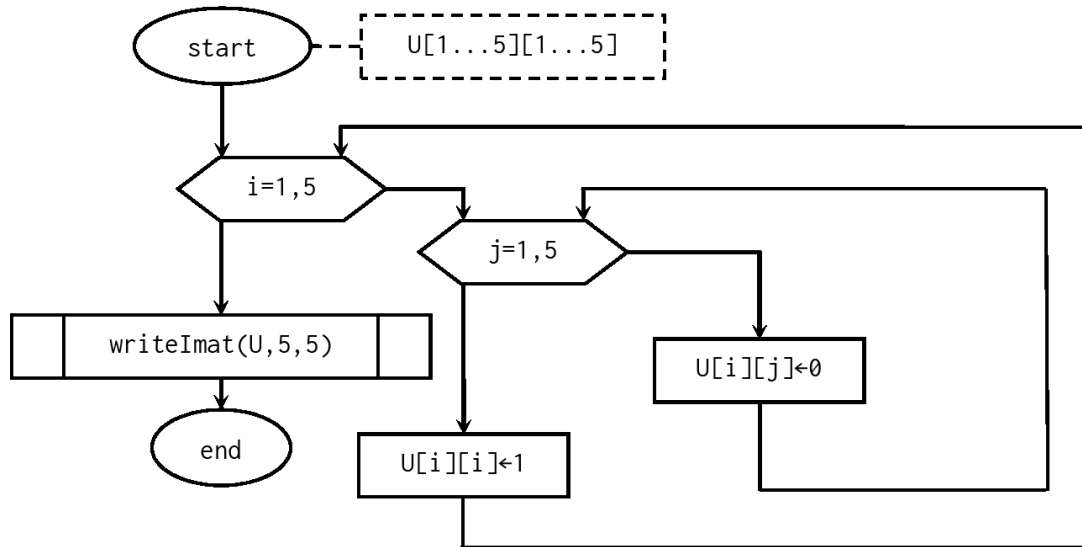


Fig. 9.1(c): Producing a 5 × 5 identity matrix using an alternative manner.

```
int main() {
    int U[6][6];
    for (int i=1; i<=5; i++) {
        for (int j=1; j<=5; j++)
            U[i][j]=0;
        U[i][i]=1;
    }
    cout<<"The identity 5 * 5 matrix:\n";
    writeImat(U, 5, 5);
    return 0;
}
//*****
void writeImat(int A[][6],
               int m, int n) {
    for (int i=1; i<=m; i++) {
        for (int j=1; j<=n; j++)
            cout<<setw(3)<<A[i][j];
        cout<<endl;
    }
}
```

```
        for (int j=1; j<=5; j++)
            U[i][j]=0;
        U[i][i]=1;
    }
    System.out.println("The identity 5 * 5 "
                       + "matrix:");
    writeImat(U, 5, 5);
}
//*****
static void writeImat(int A[][6],
                     int m, int n) {
    for (int i=1; i<=m; i++) {
        for (int j=1; j<=n; j++)
            System.out.format("%-3d", A[i][j]);
        System.out.println();
    }
}
```

9.2. Example. Write an algorithm to read a 6×6 integer matrix named A and then print the message Symmetric if A is equal with its transpose; otherwise, print the message Not symmetric.

Solution. The transpose of the matrix A is a matrix in which the (i, j) -entry is equal to the (j, i) -entry of A . Therefore, as depicted in **Flowchart 9.2**, the algorithm is arranged in a way that as soon as an opposition case of the equality of the (i, j) -entry with the (j, i) -entry is found, prints the Not symmetric message and terminates the program using an if template. The other case occurs when, the (i, j) -entry would be equal to the corresponding (j, i) -entry for each i and j . In this case, the nested loops is completely implemented and the Symmetric message is printed after the natural exit from the outer loop. Programs P9_2 are the translation of this algorithm into C++ and Java codes.

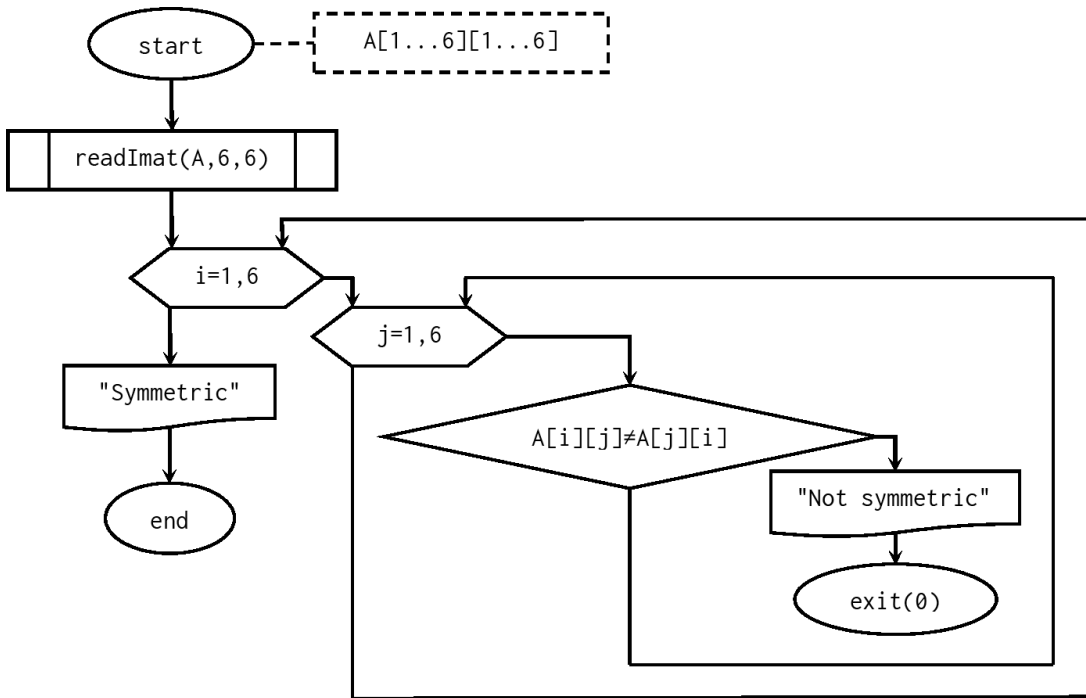


Fig. 9.2: Checking whether or not a 6×6 matrix is symmetric.

C++ codes:

```
// Program P9_2 to check whether or not
// a 6 * 6 integer matrix is symmetric
#include <iostream>
#include <stdlib.h>
using namespace std;
void readImat(int [][][7], int, int);
int main() {
    int A[7][7];
    cout<<"Enter an integer 6 * 6 "
         <<"matrix A:\n";
    readImat(A, 6, 6);
    for (int i=1; i<=6; i++)
        for (int j=1; j<=6; j++)
            if (A[i][j]!=A[j][i]) {
                cout<<"Not symmetric";
                exit(0);
            }
    cout<<"symmetric";
    return 0;
}
//*****
```

Java codes:

```
// Program P9_2 to check whether or not
// a 6 * 6 integer matrix is symmetric
import java.util.Scanner;
class P9_2 {
    public static void main(String[] args) {
        int A[][]=new int[7][7];
        System.out.println("Enter an integer "
            + "6 * 6 matrix A:");
        readImat(A, 6, 6);
        for (int i=1; i<=6; i++)
            for (int j=1; j<=6; j++)
                if (A[i][j] !=A[j][i]) {
                    System.out.print("Not symmetric");
                    System.exit(0);
                }
        System.out.print("symmetric");
    }
}
//*****
static void readImat(int A[][],
    int m, int n) {
    Scanner read=new Scanner(System.in);
```

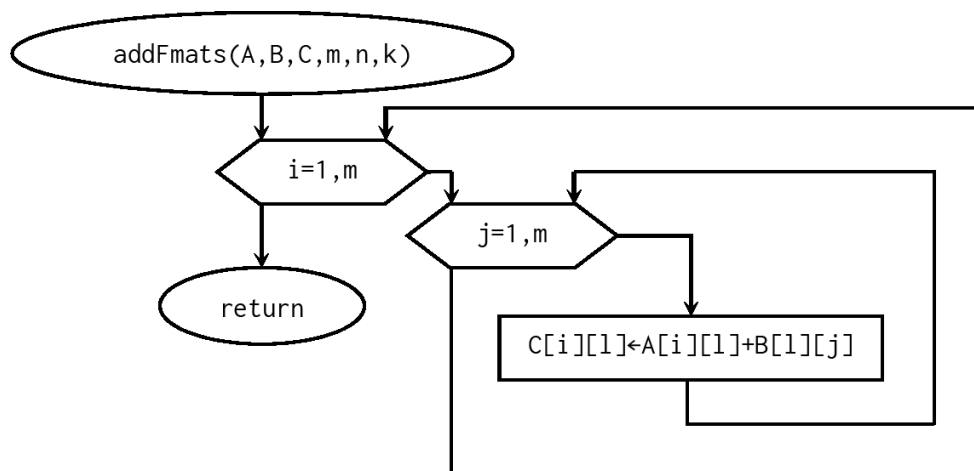


Fig. 9.3: Adding two matrices.

```

void readImat(int A[][7],
              int m, int n) {
    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
            cin>>A[i][j];
}

```

Input/output:

Enter an integer n>1: 3↵

Enter an integer 6 * 6 matrix A:

```

1 2 3 4 5 6↵
2 3 4 5 6 7↵
3 4 5 6 7 8↵
4 5 6 7 8 9↵
5 6 7 8 9 0↵
6 7 8 9 0 1↵
symmetric

```

```

    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
            A[i][j]=read.nextInt();
    read.close();
}

```

Input/output:

Enter an integer n>1: 3↵

Enter an integer 6 * 6 matrix A:

```

1 2 3 4 5 6↵
2 3 4 5 6 7↵
3 4 5 6 7 8↵
4 5 6 7 8 9↵
5 6 7 8 9 0↵
6 7 8 9 0 1↵
symmetric

```

9.3. Example. Write a sub-algorithm named `addFmats()` to receive the two real $m \times n$ matrices named A and B and then calculate and return the matrix sum $C = A + B$.

Solution. The sum $C = A + B$ of two matrices is defined as follows.

$$C_{ij} = A_{ij} + B_{ij}, \quad i = 1, \dots, m, \quad j = 1, \dots, n.$$

As shown in [Figure 9.3](#), this definition can be performed with two nested for loops due to the existence of two variable indices. The codes of this sub-algorithm are as follows.

C++ codes:

```
float addFmats(float A[][5], float B[][5], float C[][5], int m, int n) {
    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
            C[i][j] = A[i][j] + B[i][j];
    }
}
```

Java codes:

```
static void addFmats(float A[][], float B[][], float C[][], int m, int n) {
    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
            C[i][j] = A[i][j] + B[i][j];
    }
}
```



The subprogram addFmat() may be written in alternative forms in various systems and versions of compilers. In particular, in some compilers in C++, the statement `return a[i][j]` is written at the end in order to return the entries of C. Additionally, in some compilers of C++, the data type `void` is accepted for the subprogram.

A similar subprogram named `addImats()` can be written for adding the integer matrices. It suffices to replace the keyword `float` by `int`.

The main units in Program P9_3 read two real 3×4 matrices and then calculate and print their sum using the subprogram `addFmats()`.

C++ codes:

```
// Program P9_3 to add two 3 * 4 real matrices
#include <iostream>
#include <iomanip>
using namespace std;
void readFmat(float [][][5], int, int);
void writeFmat(float [][][5], int, int);
float addFmats(float [][][5], float [][][5], float [][][5], int, int);
int main() {
    float A[4][5], B[4][5], C[4][5];
    cout<<"Enter the matrix A:"<<endl;
    readFmat(A, 3, 4);
    cout<<"Enter the matrix B:"<<endl;
    readFmat(B, 3, 4);
    addFmats(A, B, C, 3, 4);
    cout<<"The matrix C = A + B is:"<<endl;
    writeFmat(C, 3, 4);
    return 0;
}
//*****

void readFmat(float A[][][5], int m, int n) {
    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
            cin>>A[i][j];
}
//*****

void writeFmat(float A[][][5], int m, int n) {
    for (int i=1; i<=m; i++) {
        for (int j=1; j<=n; j++)
            cout<<setw(6)<<A[i][j];
        cout<<endl;
    }
}
//*****

float addFmats(float A[][][5], float B[][][5], float C[][][5], int m, int n) {
    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
            C[i][j] = A[i][j] + B[i][j];
}
```

Input/output:

```
Enter the matrix A:
1 1 1 4↵
6 6 6 3↵
7 7 7 9↵
Enter the matrix B:
4 4 4 1↵
3 3 3 6↵
0 0 0 7↵
```

```
5 5 5 5
The matrix C = A + B is:
  5   5   5   5
  9   9   9   9
 16  16  16  16
```

Java codes:

```
// Program P9_3 to add two 3 * 4 real matrices
import java.util.Scanner;
class P9_3 {
    public static void main(String[] args) {
        float A[][] = new float[4][5];
        float B[][] = new float[4][5];
        float C[][] = new float[4][5];
        System.out.println("Enter the matrix A: ");
        readFmat(A, 3, 4);
        System.out.println("Enter the matrix B: ");
        readFmat(B, 3, 4);
        addFmats(A, B, C, 3, 4);
        System.out.println("The matrix C = A + B is: ");
        writeFmat(C, 3, 4);
    }
    //*****
    static void readFmat(float A[][], int m, int n) {
```

```

Scanner read = new Scanner(System.in);
for (int i=1; i<=m; i++)
    for (int j=1; j<=n; j++)
        A[i][j] = read.nextFloat();
//read.close();
}
//*****
static void writeFmat(float A[][], int m, int n) {
    for (int i=1; i<=m; i++) {
        for (int j=1; j<=n; j++)
            System.out.format("%-7.2f", A[i][j]);
        System.out.println();
    }
}
//*****
static void addFmats(float A[][], float B[][], float C[][], int m, int n) {
    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
            C[i][j] = A[i][j] + B[i][j];
}
}

```

Input/output:

```

Enter the matrix A:
1 1 1 4↵
6 6 6 3↵
7 7 7 9↵
Enter the matrix B:
4 4 4 1↵
3 3 3 6↵
9 9 9 7↵
The matrix C = A + B is:
5.00  5.00  5.00  5.00
9.00  9.00  9.00  9.00
16.00 16.00 16.00 16.00

```

9.4. Example. Write a sub-algorithm named `productFmats()` in order to receive the two real $m \times n$ matrix A and $n \times k$ matrix B and then calculate and return their multiplication matrix C , which is a real $m \times k$ matrix.

Solution. The multiplication $C = AB$ of the two matrices A and B is defined as follows.

$$C_{ij} = \sum_{l=1}^n A_{il}B_{lj}, \quad i = 1, \dots, m, \quad j = 1, \dots, n.$$

$$C_{ij} = \sum_{l=1}^n A_{il}B_{lj}, \quad i = 1, \dots, m, \quad j = 1, \dots, n.$$

As shown, we are dealing with a series. On the other hand, we have two variable indices. Therefore, we calculate this series and assign its result to $C[i][j]$ inside two nested for loops. The result of this process is displayed in **Figure 9.4**. This subprogram is called in Programs P9_4.

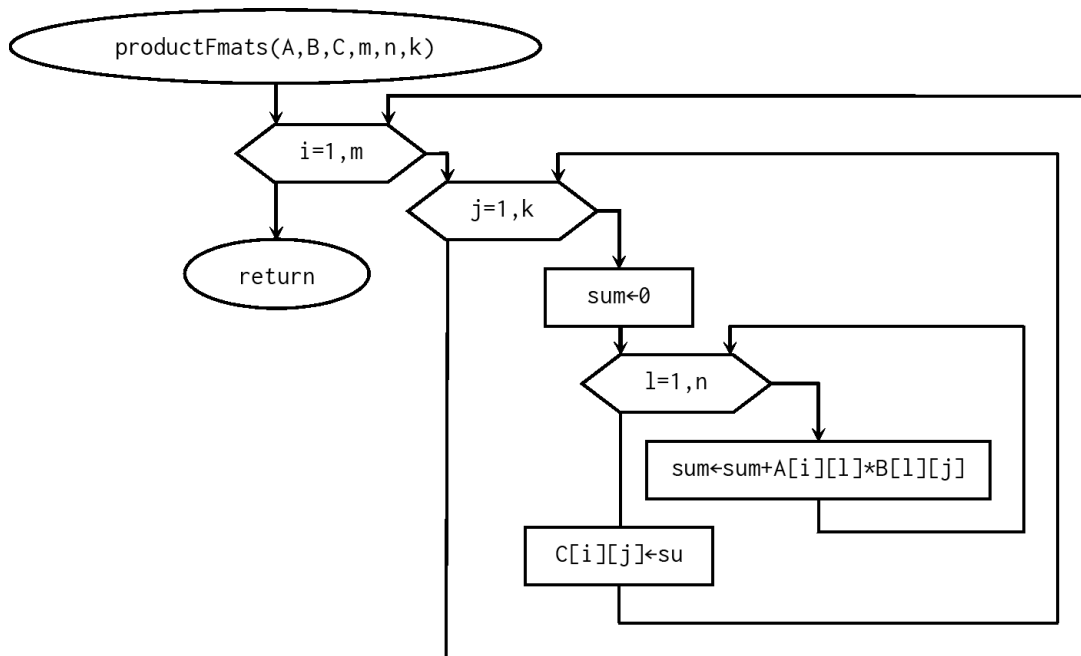


Fig. 9.4: Multiplying two matrices.

C++ codes:

```
// Program P9_4 to multiply the 3 * 4 real matrix A by the 4 * 4 real matrix B
#include <iostream>
#include <iomanip>
using namespace std;
void readFmat(float A[][5], int, int);
void writeFmat(float A[][5], int, int);
float productFmats(float A[][5], float B[][5], float C[][5], int m, int n, int k);
int main() {
    float A[4][5], B[5][5], C[4][5];
    cout<<"Enter the matrix A:"<<endl;
    readFmat(A, 3, 4);
    cout<<"Enter the matrix B:"<<endl;
    readFmat(B, 4, 4);
    productFmats (A, B, C, 3, 4, 4);
    cout<<"The matrix C = AB is:"<<endl;
    writeFmat(C, 3, 4);
    return 0;
}
//*****
void readFmat(float A[][5], int m, int n) {
    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
            cin>>A[i][j];
}
//*****
void writeFmat(float A[][5],int m,int n) {
    for (int i=1; i<=m; i++) {
        for (int j=1; j<=n; j++)
            cout<<setw(6)<<A[i][j];

        cout<<endl;
    }
}
//*****
float productFmats(float A[][5], float B[][5], float C[][5], int m, int n, int k) {
    float sum;
    for (int i=1; i<=m; i++)
        for (int j=1; j<=k; j++) {
            sum=0;
            for (int l=1; l<=n; l++)
                sum += A[i][l] * B[l][j];
            C[i][j]=sum;
        }
}
}
```

Input/output:

Enter the matrix A:

1 1 1 1↵

```

. . . .
2 2 2 2↵
3 3 3 0↵
Enter the matrix B:
0 3 3 3↵
0 3 3 3↵
0 3 3 3↵
0 3 3 3↵
The matrix C = AB is:
    0    12    12    12
    0    24    24    24
    0    27    27    27

```

Java codes:

```

// Program P9_4 to multiply the 3 * 4 real matrix A by the 4 * 4 real matrix B
import java.util.Scanner;
class P9_4 {
    public static void main(String[] args) {
        float A[][] = new float[4][5];
        float B[][] = new float[5][5];
        float C[][] = new float[4][5];
        System.out.println("Enter the matrix A: ");
        readFmat(A, 3, 4);
        System.out.println("Enter the matrix B: ");
        readFmat(B, 4, 4);
        productFmats (A, B, C, 3, 4, 4);
        System.out.println("The matrix C = AB is: ");
        writeFmat(C, 3, 4);
    }
    //*****
    static void readFmat(float A[][], int m, int n) {
        Scanner read=new Scanner(System.in);
        for (int i=1; i<=m; i++)
            for (int j=1; j<=n; j++)
                A[i][j] = read.nextFloat();
    }
}

```

```

//read.close();
}
//*****
static void writeFmat(float A[][], int m, int n) {
    for (int i=1; i<=m; i++) {
        for (int j=1; j<=n; j++)
            System.out.format("%-8.2f",A[i][j]);
        System.out.println();
    }
}
//*****
static void productFmats(float A[][], float B[][], float C[][],
                        int m, int n, int k) {

    float sum;
    for (int i=1; i<=m; i++)
        for (int j=1; j<=k; j++) {
            sum=0;
            for (int l=1; l<=n; l++)
                sum += A[i][l] * B[l][j];
            C[i][j] = sum;
        }
}
}
}

```

Input/output:

```

Enter the matrix A:
1 1 1 1↵
2 2 2 2↵
3 3 3 0↵
Enter the matrix B:
0 3 3 3↵
0 3 3 3↵
0 3 3 3↵
0 3 3 3↵
The matrix C = AB is:
0.00    12.00    12.00    12.00
0.00    24.00    24.00    24.00
0.00    27.00    27.00    27.00

```

9.5. Example. The sum of the entries on the main diagonal of a square matrix is called the *trace* of the matrix. Write a program to read a 5×5 real matrix named T without the need to draw any flowchart while using the previous sub-programs and then calculate and print the trace of the matrix T^3 .

Solution. We denote the multiplication of T and T as well as T and T^2 by T^2 and T^3 , respectively. Now, T^3 is the matrix T^3 . Next, using a for loop, we calculate the sum of the entries on the main diagonal of T^3 which is:

$$\text{Trace}(T^3) = \sum_{i=1}^5 T^3_{ii} .$$

Programs P9_5 are the requested programs.

C++ codes:

```
// Program P9_5 to read the 5 * 5 matrix T
// and then compute the trace of T^3
#include <iostream>
#include <iomanip>
using namespace std;
void readFmat(float [][][6], int, int);
float productFmats(float [][][6], float [][][6], float [][][6], int, int, int);
int main() {
    float Trace;
    float T[6][6], T2[6][6], T3[6][6];
    cout<<"Enter the matrix T:"<<endl;
    readFmat(T, 5, 5);
    productFmats(T, T, T2, 5, 5, 5);
    productFmats(T, T2, T3, 5, 5, 5);
    Trace = 0;
    for (int i=1; i<=5; i++)
        Trace += T3[i][i];
    cout<<"The trace of T^3 is: "<<Trace;
    return 0;
}
//*****
void readFmat(float A[][6], int m, int n) {
    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
            cin>>A[i][j];
}
//*****
float productFmats(float A[][6], float B[][6], float C[][6], int m, int n, int k) {
    float sum;
    for (int i=1; i<=m; i++)
        for (int j=1; j<=k; j++) {
            sum=0;
            for (int l=1; l<=n; l++)
                sum += A[i][l] * B[l][j];
            C[i][j] = sum;
        }
}
}
```

Input/output:

```
Enter the matrix T:
1 0 1 0 1↵
0 1 0 1 0↵
1 0 1 0 1↵
0 1 0 1 0↵
1 0 1 0 1↵
```

The trace of T^3 is: 35

Java codes:

```
// Program P9_5 to read the 5 * 5 matrix T
// and then compute the trace of T^3
import java.util.Scanner;
class P9_5 {
    public static void main(String[] args) {
        int i;
        float Trace;
        float T[][] = new float[6][6];
        float T2[][] = new float[6][6];
        float T3[][] = new float[6][6];
        System.out.println("Enter the matrix T: ");
        readFmat(T, 5, 5);
        productFmats(T, T, T2, 5, 5, 5);
        productFmats(T, T2, T3, 5, 5, 5);
        Trace = 0;
        for (i=1; i<=5; i++)
            Trace += T3[i][i];
        System.out.print("The trace of T^3 is: " + Trace);
    }
    //*****
    static void readFmat(float A[][], int m, int n) {
        int i, j;
        Scanner read=new Scanner(System.in);
        for (i=1; i<=m; i++)
            for (j=1; j<=n; j++)
                A[i][j] = read.nextFloat();
        //read.close();
    }
    //*****
    static void productFmats(float A[][], float B[][], float C[][],
                            int m, int n, int k) {
        float sum;
        for (int i=1; i<=m; i++)
            for (int j=1; j<=k; j++) {
                sum=0;
                for (int l=1; l<=n; l++)
                    sum += A[i][l] * B[l][j];
                C[i][j] = sum;
            }
    }
}
```

Input/output:

Enter the matrix T:

```

enter the matrix T:
1 0 1 0 1↵
0 1 0 1 0↵
1 0 1 0 1↵
0 1 0 1 0↵
1 0 1 0 1↵
The trace of T^3 is: 35.0

```

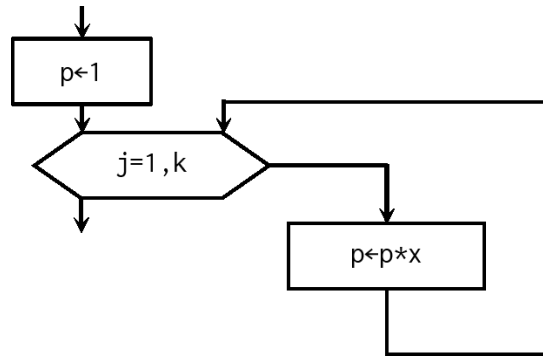


Fig. 9.6(a): Calculating the number x to the power of k (recalling [Flowchart 6.8\(a\)](#)).

In Programs P9_5, we used the technique of repetitive multiplications for calculating the low powers of a matrix. This technique is not an efficient technique for high powers. The next sub-algorithm calculates any power of a square matrix.

9.6. Example. Write a sub-algorithm named `powerFmat()` to receive an $n \times n$ real matrix T and a positive integer k and then calculate and return T^k .

Solution. Recall the algorithm of [Figure 6.8\(a\)](#) for taking the number t to the positive integer power of k , as in [Figure 9.6\(a\)](#).

We use the idea behind the algorithm of [Figure 9.6\(a\)](#). Two points should be noted. First, the analogy to the assignment

$$p \leftarrow 1 \quad P \leftarrow I$$

in the matrix point of view is:

$$p \leftarrow I \quad P \leftarrow I$$

where, I is the identity matrix. This is known for us. In fact, the float version of the sub-algorithm in **Figure 9.1(b)** performs this assignment.

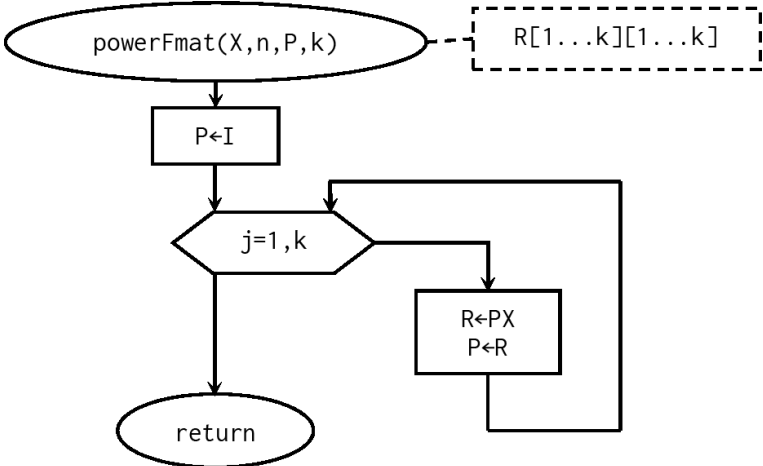


Fig. 9.6(b): Calculating the matrix X to the power of k .

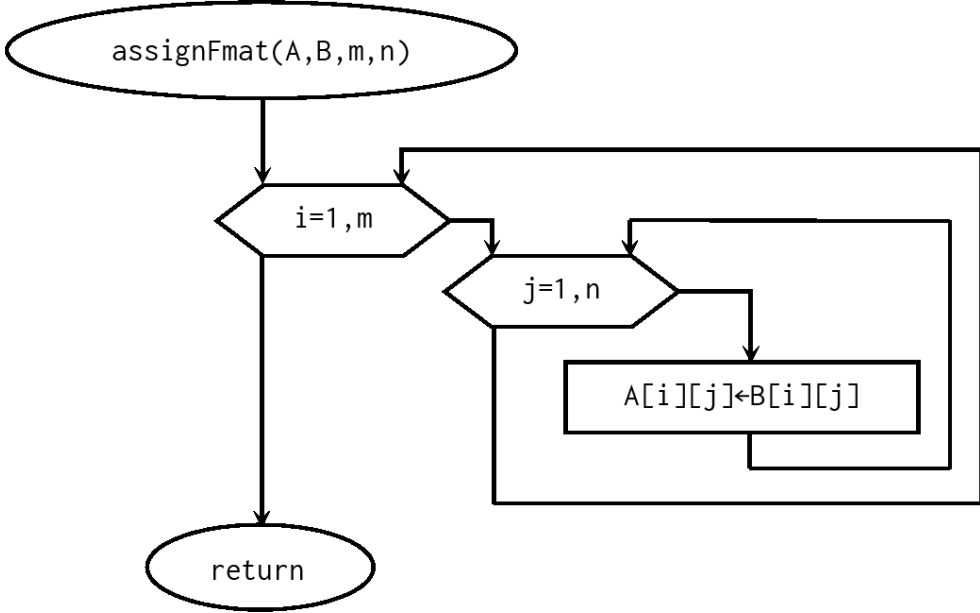


Fig. 9.6(c): Assigning the matrix B to the matrix A .

The second note is that we are unable to directly store the matrix product PX in P due to the nature of the matrix multiplication defined in **Example 9.4**. Instead, we can store PX in another matrix, say R , and then assign R to P . By the discussion provided so far, we can draw **Flowchart 9.6(b)**.

Finally, we provide **Flowchart 9.6(c)** to assign any $m \times n$ matrix B to the matrix A of the same size. Now, we gather the three above flowcharts in Programs 9_6.

C++ codes:

```
// Program P9_6 to take a 4 * 4 matrix A to the positive integer power of 4
#include <iostream>
#include <iomanip>
using namespace std;
void readFmat(float [][][5], int, int);
void writeFmat(float [][][5], int, int);
void powerFmat(float [][][5], int, float [][][5], int );
void productFmats(float A[][][5], float B[][][5], float C[][][5], int m, int n, int k);
void assignFmat(float [][][5], float [][][5], int , int );
void idFmat(float [][][5], int);
int main() {
    int k;
    float A[5][5], A4[5][5];
    cout<<"Enter the matrix A:"<<endl;
    readFmat(A, 4, 4);
    powerFmat(A, 4, A4, 4);
    cout<<"The matrix A4 = A^4 is:"<<endl;
    writeFmat(A4, 4, 4);
    return 0;
}
//*****
void readFmat(float A[][][5], int m, int n) {
    for (int i=1; i<=m; i++)

        for (int j=1; j<=n; j++)
            cin>>A[i][j];
}
//*****
void writeFmat(float A[][][5], int m, int n) {
    for (int i=1; i<=m; i++) {
        for (int j=1; j<=n; j++)
            cout<<setw(6)<<A[i][j];
        cout<<endl;
    }
}
//*****
void productFmats(float A[][][5], float B[][][5], float C[][][5], int m, int n, int k) {
    float sum;
    for (int i=1; i<=m; i++)
        for (int j=1; j<=k; j++) {
            sum = 0;
            for (int l=1; l<=n; l++)
                sum += A[i][l]*B[l][j];
        }
}
```

```

        C[i][j] = sum;
    }
}
//*****
void powerFmat(float X[][5], int n, float P[][5], int k) {
    float R[5][5];
    idFmat(P,4);
    for (int t=1; t<=k; t++) {
        productFmats(P, X, R, n, n, n);
        assignFmat(P, R, 4, 4);
    }
}
//*****
void assignFmat(float A[][5], float B[][5], int m, int n) {
    int i,j;
    for (i=1; i<=m; i++)
        for (j=1; j<=n; j++)
            A[i][j] = B[i][j];
}
//*****
void idFmat(float U[][5], int n) {
    for (int i=1; i<=n; i++)
        for (int j=1; j<=n; j++)
            if (i == j)
                U[i][j] = 1;
            else
                U[i][j] = 0;
}
}

```

Input/output:

Enter the matrix A:

```

2 0 2 0↵
0 2 0 2↵
2 0 2 0↵
0 2 0 2↵

```

The matrix $A^4 = A^4$ is:

```

128    0   128    0
  0   128    0   128
128    0   128    0
  0   128    0   128

```

Java codes:

```
// Program P9_6 to take a 4 * 4 matrix A to the positive integer power of 4
```

```

import java.util.Scanner;
class P9_6 {
    public static void main(String[] args) {
        float A[][] = new float[5][5];
        float A4[][] = new float[5][5];
        System.out.println("Enter the matrix A: ");
        readFmat(A, 4, 4);
        powerFmat(A, 4, A4, 4);
        System.out.println("The matrix A4 = A^4 is:");
        writeFmat(A4, 4, 4);
    }
    /*******
    static void readFmat(float A[][], int m, int n) {
        Scanner read = new Scanner(System.in);
        for (int i=1; i<=m; i++)
            for (int j=1; j<=n; j++)
                A[i][j] = read.nextFloat();
        read.close();
    }
    /*******
    static void writeFmat(float A[][], int m, int n) {
        for (int i=1; i<=m; i++) {
            for (int j=1; j<=n; j++)
                System.out.format("%-9.2f", A[i][j]);
            System.out.println();
        }
    }
    /*******
    static void productFmats(float A[][], float B[][], float C[][],
        int m, int n, int k) {

        float sum;
        for (int i=1; i<=m; i++)
            for (int j=1; j<=k; j++) {
                sum = 0;
                for (int l=1; l<=n; l++)
                    sum += A[i][l]*B[l][j];
                C[i][j] = sum;
            }
    }
    /*******
    static void powerFmat(float X[][], int n, float P[][], int k) {
        float R[][] = new float[5][5];
        idFmat(P,4);
        for (int t=1; t<=k; t++) {
            productFmats(P, X, R, n, n, n);
            assignFmat(P, R, 4, 4);
        }
    }
}

```

```

    }
}
//*****
static void assignFmat(float A[][[]], float B[][[]], int m, int n) {
    int i, j;
    for (i=1; i<=m; i++)
        for (j=1; j<=n; j++)
            A[i][j] = B[i][j];
}
//*****
static void idFmat(float U[][[]], int n) {
    for (int i=1; i<=n; i++)
        for (int j=1; j<=n; j++)
            if (i == j)
                U[i][j] = 1;
            else
                U[i][j] = 0;
}
}
}

```

Input/output:

```

Enter the matrix A:
2 0 2 0↵
0 2 0 2↵
2 0 2 0↵
0 2 0 2↵
The matrix A4 = A^4 is:
128.00   0.00   128.00   0.00
0.00    128.00   0.00    128.00
128.00   0.00   128.00   0.00
0.00    128.00   0.00    128.00

```

9.7. Example. Write a program to read two 4×4 real matrices A and B . Then, calculate and print the following matrix using the needed subprograms which were already studied.

$$AB^5 + BA^5. AB^5 + BA^5.$$

Solution. Denote the B^5 , AB^5 , A^5 , BA^5 , and $AB^5 + BA^5$ by C , D , E , F , and G , respectively. The requested calculations are performed in Programs P9_7.

C++ codes:

```
// Program P9_7 to read two real 4 * 4 matrices
// A and B and then compute and print A*B^5+B*A^5
#include <iostream>
#include <iomanip>
using namespace std;
void readFmat(float [][][5], int, int);
void writeFmat(float [][][5], int, int);

void addFmats(float [][][5], float [][][5], float [][][5], int, int);
void productFmats(float [][][5], float [][][5], float [][][5], int, int, int);
void powerFmat(float [][][5], int, float [][][5], int);
void assignFmat(float [][][5], float [][][5], int, int);
void idFmat(float [][][5], int);
int main() {
    float A[5][5], B[5][5], C[5][5], D[5][5], E[5][5], F[5][5], G[5][5];
    cout<<"Enter the matrix A:"<<endl;
    readFmat(A, 4, 4);
    cout<<"Enter the matrix B:"<<endl;
    readFmat(B, 4, 4);
    powerFmat(B, 4, C, 5);
    productFmats(A, C, D, 4, 4, 4);
    powerFmat(A,4,E,5);
    productFmats(B, E, F, 4, 4, 4);
    addFmats(D, F, G, 4, 4);
    cout<<"The matrix A*B^5+B*A^5 is:"<<endl;
    writeFmat(G,4,4);
    return 0;
}
//*****
void readFmat(float A[][][5], int m, int n) {
    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
            cin>>A[i][j];
}
//*****
void writeFmat(float A[][][5], int m, int n) {
    for (int i=1; i<=m; i++) {
        for (int j=1; j<=n; j++)
            cout<<setw(6)<<A[i][j];
        cout<<endl;
    }
}
//*****
void addFmats(float A[][][5], float B[][][5], float C[][][5], int m, int n) {
    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
```

```

        for (int j=1; j<=n; j++)
            C[i][j] = A[i][j] + B[i][j];
    }
//*****
void productFmats(float A[][5], float B[][5], float C[][5], int m, int n, int k) {
    float sum;
    for (int i=1; i<=m; i++)
        for (int j=1; j<=k; j++) {
            sum=0;
            for (int l=1; l<=n; l++)
                sum += A[i][l] * B[l][j];
            C[i][j] = sum;
        }
}
//*****
void powerFmat(float X[][5], int n, float P[][5], int k) {
    float R[5][5];
    idFmat(P, 4);
    for (int t=1; t<=k; t++) {

        productFmats(P, X, R, n, n, n);
        assignFmat(P, R, 4, 4);
    }
}
//*****
void assignFmat(float A[][5], float B[][5], int m, int n) {
    int i, j;
    for (i=1; i<=m; i++)
        for (j=1; j<=n; j++)
            A[i][j] = B[i][j];
}
//*****
void idFmat(float U[][5], int n) {
    for (int i=1; i<=n; i++)
        for (int j=1; j<=n; j++)
            if (i == j)
                U[i][j] = 1;
            else
                U[i][j] = 0;
}

```

Input/output:

Enter the matrix A:

```

1 0 1 0↵
0 1 0 1↵
1 0 1 0↵
0 1 0 1↵

```

Enter the matrix B:

```
3 0 0 0↵
0 3 0 0↵
0 0 3 0↵
0 0 0 3↵
```

The matrix $A \cdot B^5 + B \cdot A^5$ is:

```
291    0    291    0
   0   291     0   291
291    0    291    0
   0   291     0   291
```

Java codes:

```
// Program P9_7 to read two real 4 * 4 matrices
// A and B and then compute and print  $A \cdot B^5 + B \cdot A^5$ 
import java.util.Scanner;
class P9_7 {
    public static void main(String[] args) {
        float A[][] = new float[5][5];
        float B[][] = new float[5][5];
        float C[][] = new float[5][5];
        float D[][] = new float[5][5];
        float E[][] = new float[5][5];
        float F[][] = new float[5][5];
        float G[][] = new float[5][5];
        System.out.println("Enter the matrix A: ");

        readFmat(A, 4, 4);
        System.out.println("Enter the matrix B: ");
        readFmat(B, 4, 4);
        powerFmat(B, 4, C, 5);
        productFmats(A, C, D, 4, 4, 4);
        powerFmat(A, 4, E, 5);
        productFmats(B, E, F, 4, 4, 4);
        addFmats(D, F, G, 4, 4);
        System.out.println("The matrix  $A \cdot B^5 + B \cdot A^5$  is: ");
        writeFmat(G, 4, 4);
    }
    /*******
static void readFmat(float A[][], int m, int n) {
    Scanner read = new Scanner(System.in);
    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
            A[i][j] = read.nextFloat();
}
    /*******
static void writeFmat(float A[][], int m, int n) {
    for (int i=1; i<=m; i++) {
```

```

        for (int j=1; j<=n; j++)
            System.out.format("%-9.2f", A[i][j]);
        System.out.println();
    }
}
//*****
static void addFmats(float A[][], float B[][], float C[][], int m, int n) {
    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
            C[i][j] = A[i][j] + B[i][j];
}
//*****
static void productFmats(float A[][], float B[][], float C[][],
                        int m, int n, int k) {
    float sum;
    for (int i=1; i<=m; i++)
        for (int j=1; j<=k; j++) {
            sum=0;
            for (int l=1; l<=n; l++)
                sum += A[i][l] * B[l][j];
            C[i][j] = sum;
        }
}
//*****
static void powerFmat(float X[][], int n, float P[][], int k) {
    float R[][]=new float[5][5];
    idFmat(P, 4);
    for (int t=1; t<=k; t++) {
        productFmats(P, X, R, n, n, n);
        assignFmat(P, R, 4, 4);
    }
}
//*****
static void assignFmat(float A[][], float B[][], int m, int n) {
    int i,j;

```

```

    for (i=1; i<=m; i++)
        for (j=1; j<=n; j++)
            A[i][j] = B[i][j];
}
//*****
static void idFmat(float U[ ][ ], int n) {
    for (int i=1; i<=n; i++)
        for (int j=1; j<=n; j++)
            if (i == j)
                U[i][j] = 1;
            else
                U[i][j] = 0;
}
}

```

Input/output:

```

Enter the matrix A:
1 0 1 0↵
0 1 0 1↵
1 0 1 0↵
0 1 0 1↵
Enter the matrix B:
3 0 0 0↵
0 3 0 0↵
0 0 3 0↵
0 0 0 3↵
The matrix A*B^5+B*A^5 is:
291.00  0.00  291.00  0.00
0.00  291.00  0.00  291.00
291.00  0.00  291.00  0.00
0.00  291.00  0.00  291.00

```

There are three elementary row operations on an $m \times n$ matrix as follows.

1. Multiplying one row of A by a nonzero number z ;
2. Replacing the r -th row of A by row r plus z times row s where z is any number and $r \neq s$;
3. Interchanging the two rows of A .

In the following two challenging [Examples 9.8](#) and [9.9](#), we use the elementary row operations to calculate the determinant and the inverse of a square matrix, respectively. There may exist simpler manners to

solve these examples. However, exhibited manners aim to examine and apply several previously taught techniques.

9.8. Example. Write a function named `det()` to receive an $n \times n$ real matrix A and then calculate and return its determinant.

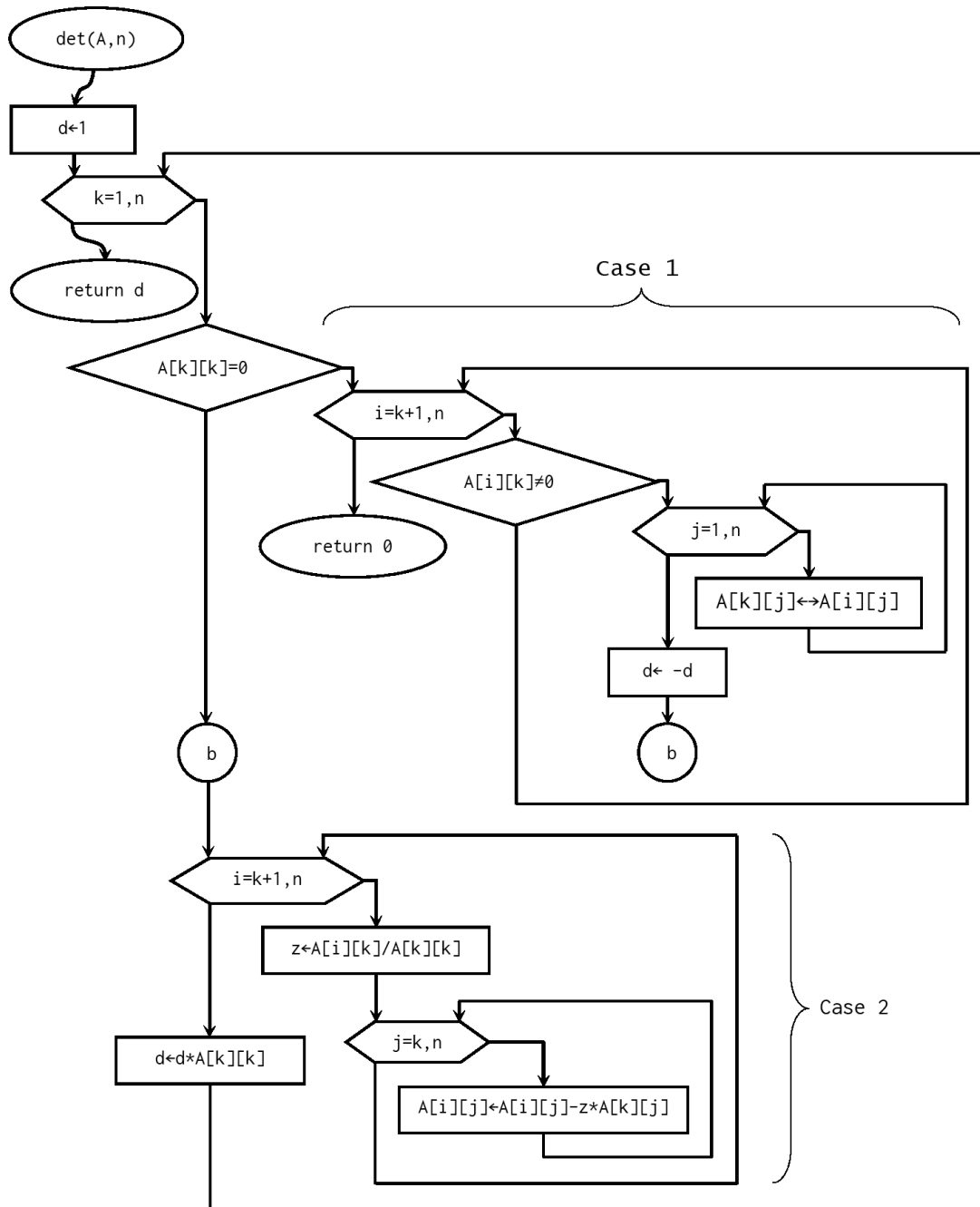


Fig. 9.8: Calculating the determinant of an $n \times n$ matrix using the elementary row operations.

Solution. This is a challenging example. Represent the carrier of the function as d with the initial value of 1. The strategies for calculating the determinant is to transform the given matrix to an upper triangular matrix using the elementary row operations and then exhibit the

multiplication of the (main) diagonal entries as the determinant. To do this, we take the diagonal entry A_{kk} in each repetition, using a *leader* for loop with the specification $k=1,n$ and then check whether or not A_{kk} is zero with an if template. We continue the algorithm in two cases depending on this condition.

Case 1. $A_{kk} = 0$. $A_{kk} = 0$. In this case, we look for a nonzero entry below A_{kk} in the same column using a for loop with the specification $i=k+1,n$. All the corresponding entries of rows i and k are swapped by a for loop if there is such a nonzero entry in some repetition of the recent loop. Then, the sign of d is changed by the preliminary properties of determinant and finally, goes to **Case 2**. However, the non-existence of such a nonzero entry in any repetition implies that the entry in this position of the upper triangular matrix is zero. Therefore, the sub-algorithm is terminated returning the zero value for the determinant.

Case 2. $A_{kk} \neq 0$. $A_{kk} \neq 0$. In this case, which starts with the label b , we turn all the entries below A_{kk} , in the same column, to zeroes. To do this, it suffices to perform the following substitution inside a for loop with the specification $i=k+1,n$.

$$\begin{aligned} \text{row } i &\leftarrow \text{row } i - z \times \text{row } k, \\ \text{ROW } i &\leftarrow \text{ROW } i - z \times \text{ROW } k, \end{aligned}$$

where $z = A_{ik}/A_{kk}$. $z = A_{ik}/A_{kk}$. As seen, the row is fixed while only the columns vary. Therefore, a for loop with the specification $j=k,n$ is used to perform this substitution since the entries A_{i1} to A_{ik-1} have already become zero.

Finally, we should multiply the value of the current diagonal entry to the previous repetitive multiplication of d before ending the current repetition of the leader loop. The above discussions are summarized in **Flowchart 9.8**.

Each of Programs P9_8 reads the size of the matrix A as n which is at most 10. Then, reads an $n \times n$ matrix. Eventually, calculates and prints the determinant of A calling the function `det()`.

C++ codes:

```
// Program P9_8 to compute the determinant of a 4 * 4 matrix using elementary
// row operations. To use for another size n, change the length 5 to n+1
#include <iostream>
using namespace std;
void readFmat(float A[][11], int, int);
float det(float A[][11], int);
float A[11][11];
int main() {
    int n;
    cout<<"Enter the size of the matrix A: ";
    cin>>n;
    cout<<"Now enter the matrix A: \n";
    readFmat(A, n, n);
    cout<<"The determinant of A is: "<<det(A, n);
    return 0;
}

/*****
void readFmat(float A[][11], int m, int n) {
    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
            cin>>A[i][j];
}
*****/
float det(float A[][11], int n) {
    float d, t, z;
    int i, j, k;
    d = 1;
    for (k=1; k<=n; k++) {
        if (A[k][k] == 0) {
            for (i=k+1; i<=n; i++)
                if (A[i][k] != 0) {
                    for (j=1; j<=n; j++) {
                        t = A[k][j]; A[k][j] = A[i][j]; A[i][j] = t;
                    }
                    d = -d;
                }
        }
    }
}
```

```

        goto b;
    }
    return 0;
}
b: for (i=k+1; i<=n; i++) {
    z = (float)(A[i][k] / A[k][k]);
    for (j=k; j<=n; j++)
        A[i][j] = A[i][j] - z * A[k][j];
    }
    d = d * A[k][k];
}
return d;
}

```

Input/output:

```

Enter the dimension of the matrix A: 4↵
Now enter the matrix A:
3 0 2 -1↵
1 2 0 -2↵
4 0 6 -3↵
5 0 2 0↵
The determinant of A is: 20

```

Java codes:

```

// Program P9_8 to compute the determinant of a 4 * 4 matrix using elementary
// row operations. To use for another size n, change the length 5 to n+1
import java.util.Scanner;
class P9_8 {
    public static void main(String[] args) {
        Scanner read = new Scanner(System.in);
        double A[][] = new double[11][11];
        System.out.print("Enter the size of matrix A: ");
        int n = read.nextInt();

        System.out.println("Enter the matrix A: ");
        readFmat(A, n, n);
        System.out.println("The determinant of A is: " + det(A, n));
        read.close();
    }
    //*****
    static void readFmat(double A[][], int m, int n) {
        int i, j;
        Scanner read = new Scanner(System.in);
        for (i=1; i<=m; i++)
            for (j=1; j<=n; j++)
                A[i][j] = read.nextDouble();
        read.close();
    }
}

```

```

}
//*****
static double det(double A[][], int n) {
    double d, t, z;
    int c, h, i, j, k;
    d = 1;
    for (k=1; k<=n; k++) {
b:    {
        if (A[k][k] == 0) {
            for (h=k+1; h<=n; h++)
                if (A[h][k] != 0) {
                    for (c=1; c<=n; c++) {
                        t = A[k][c]; A[k][c] = A[h][c]; A[h][c] = t;
                    }
                    d = -d;
                    break b;
                }
            return 0;
        }
        for (i=k+1; i<=n; i++) {
            z = (double)(A[i][k] / A[k][k]);
            for (j=k; j<=n; j++)
                A[i][j] = A[i][j] - z * A[k][j];
        }
        d = d * A[k][k];
    }
    return d;
}
}
}

```

Input/output:

```

Enter the dimension of the matrix A: 4↵
Now enter the matrix A:
3 0 2 -1↵
1 2 0 -2↵
4 0 6 -3↵
5 0 2 0↵
The determinant of A is: 20.0

```

9.9. Example. Write a subprogram named `invFmat()` to receive the $n \times n$ real matrix A and then calculate and return its inverse if it is invertible; otherwise, prints a message.

Solution. One of the techniques of finding the inverse of the invertible matrix A is to transform A to the identity matrix, using the elementary row operations. Then, apply exactly the same elementary row

operations to the identity matrix I in the same order. The final matrix I is the inverse of A .

Further, the same strategies as in **Example 9.8** with a little differences is simultaneously applied for A and I . Then, the real identity matrix is generated using the sub-algorithm `idFmat()` and is assigned to I . Next, as in **Example 9.8**, we take the diagonal entry A_{kk} **A_{kk}** in each repetition using a *leader* for loop with the specification $k=1,n$ and check whether or not A_{kk} **A_{kk}** is zero. Finally, we continue the algorithm in two cases depending on this condition.

Case 1. $A_{kk} = 0$. $A_{kk} = 0$. This case goes the same line as **Case 1** in **Example 9.8** and the elementary row operations are simultaneously applied for A and I .

Case 2. $A_{kk} \neq 0$. $A_{kk} \neq 0$. This case, which starts with the label b , is divided into two parts. In Part 1, we divide rows k of both A and I by the diagonal entry A_{kk} . **A_{kk}** . Since this entry may vary, we store it in a variable, say p , and divide the mentioned rows by p . In Part 2, we change all the entries below and above A_{kk} , **A_{kk}** , in the same column, to zeroes. In other words, we annihilate all the entries A_{ij} **A_{ij}** with $i \neq j$. To do this, we jump over row k using an if template and apply the following substitution for both A and I .

$$\begin{aligned} \text{row } i &\leftarrow \text{row } i - z \times \text{row } k, \\ \text{row } i &\leftarrow \text{row } i - z \times \text{row } k, \end{aligned}$$

where, this time $z = A_{ik}$ **$z = A_{ik}$** since the division by A_{kk} **A_{kk}** has already been conducted. As before, this substitution is performed

using a for loop with the specification $j=k:n$, since the entries A_{i1} to A_{ik-1} have already reached zero.

Concentrating on the sub-algorithm `det()` in [Example 9.8](#), exactly in the position where the determinant was announced as zero by the instruction `return 0`, we terminate our sub-algorithm and return to the caller unit by printing a message mentioning that the matrix is not invertible. The above discussions are summarized in [Flowchart 9.9](#).

Each of Programs P9_9 reads the size of the matrix A as n which is at most 10. Then, it reads the $n \times n$ matrix A and, calling the subprogram `invFmat()`, it calculates and prints the inverse of A if A is invertible; otherwise, it prints a message.

To check the correctness of the inverse matrix I in Programs P9_9, we multiplied it by the matrix A which was assigned in the matrix $A0$ using the `assignFmat` subprogram in [Example 9.6](#). The inverse matrix is correct if the result of the multiplication is the identity matrix. It is worth mentioning that, we may encounter with the approximated results for the 0- and 1-entries of the identity matrix due to the approximated computations for the real numbers in the system of the computers.

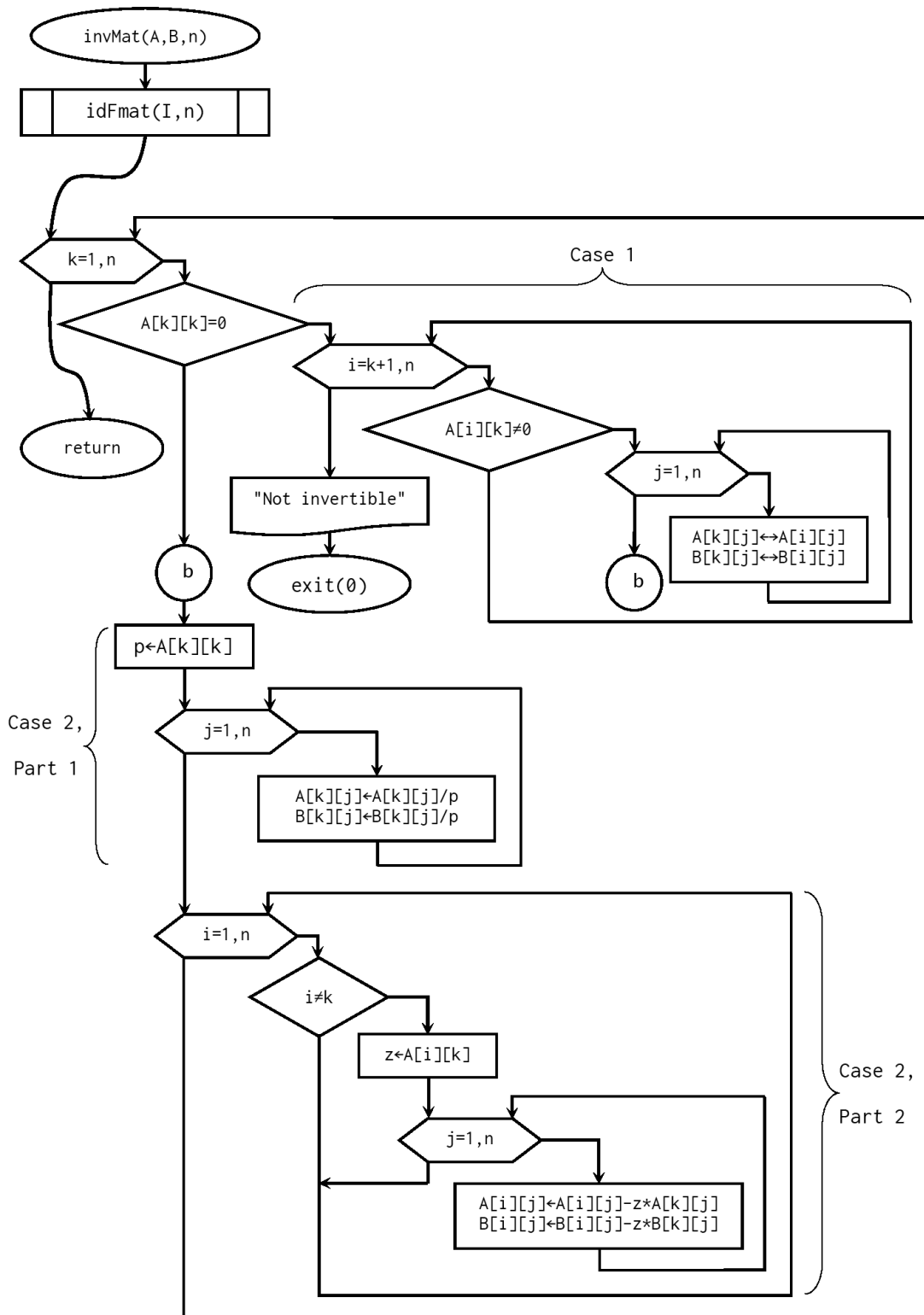


Fig. 9.9: Calculating the inverse of a matrix using the elementary row operations.

C++ codes:

```
// Program P9.9 to compute the inverse of an n * n matrix A
```

```

// Program 15_3 to compute the inverse of an n x n matrix A,
// using the elementary row operations
#include <iostream>
#include <iomanip>
using namespace std;
void readFmat(float A[][11], int, int);
void writeFmat(float A[][11], int, int);
void productFmats(float A[][11], float B[][11], float C[][11], int , int , int );
void invMat(float A[][11], float I[][11], int);
void idFmat(float I[][11], int );
void assignFmat(float A[][11], float I[][11], int , int );
int main() {
    float A[11][11], I[11][11], Id[11][11], A0[11][11];
    int n;
    cout<<"Enter the size of the matrix A: ";
    cin>>n;
    cout<<"Now enter the matrix A: \n";
    readFmat(A, n, n);
    cout<<"\nThe matrix A: \n\n";
    writeFmat(A, n, n);
    assignFmat(A0, A, 4, 4);
    invMat(A, I, n);
    cout<<"\nThe inverse of A: \n\n";
    writeFmat(I, n, n);
    cout<<"\nThe multiplication A by its inverse:\n\n";
    productFmats(A0, I, Id, n, n, n);
    writeFmat(Id, n, n);
}
//*****
void readFmat(float A[][11], int m, int n) {
    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
            cin>>A[i][j];
}
//*****
void writeFmat(float A[][11], int m, int n) {
    for (int i=1; i<=m; i++) {
        for (int j=1; j<=n; j++)
            cout<<setw(14)<<A[i][j];
        cout<<endl;
    }
}
//*****
void productFmats(float A[][11], float B[][11], float C[][11], int m, int n, int k) {
    float sum;
    for (int i=1; i<=m; i++)
        for (int j=1; j<=k; j++) {
            sum = 0;
            for (int l=1; l<=n; l++)
                sum += A[i][l] * B[l][j];
        }
}

```


Input/output:

Enter the size of the matrix A: 3↵

Now enter the matrix A:

1 1 3↵

2 4 9↵

1 3 0↵

The inverse of A:

2.25	-0.75	0.25
-0.75	0.25	0.25
-0.166667	0.166667	-0.166667

The matrix A:

1	1	3
2	4	9
1	3	0

The multiplication A by its inverse:

1	0	0
0	1	0
0	0	1

Java codes:

```
// Program P9_9 to compute the inverse of an n * n matrix A,  
// using the elementary row operations  
import java.util.Scanner;  
class P9_9 {  
    public static void main(String[] args) {  
        Scanner read = new Scanner(System.in);  
        float A[][] = new float[11][11];  
        float I[][] = new float[11][11];  
        float Id[][] = new float[11][11];  
        float A0[][] = new float[11][11];  
        int n;  
        System.out.print("Enter the size of matrix A: ");  
        n = read.nextInt();  
        System.out.println("Now enter the matrix A: ");  
        readFmat(A, n, n);  
        System.out.println("\nThe matrix A: \n");  
        writeFmat(A, n, n);  
        assignFmat(A0, A, n, n);  
        invMat(A, I, n);  
    }  
}
```

```

        invMat(A, i, n),
        System.out.println("\nThe inverse of A is: \n");
        writeFmat(I, n, n);
        System.out.println("\nThe multiplication A by its inverse: \n");
        productFmats(A0, I, Id, n, n, n);
        writeFmat(Id, n, n);
        read.close();
    }
    //*****
    static void readFmat(float A[][], int m, int n) {
        int i, j;
        Scanner read = new Scanner(System.in);
        for (i=1; i<=m; i++)
            for (j=1; j<=n; j++)
                A[i][j] = read.nextFloat();
        read.close();
    }

    //*****
    static void writeFmat(float A[][], int m, int n) {
        for (int i=1; i<=m; i++) {
            for (int j=1; j<=n; j++)
                System.out.format("%-12f", A[i][j]);
            System.out.println();
        }
    }
    //*****
    static void productFmats(float A[][], float B[][], float C[][],
                            int m, int n, int k) {

        float sum;
        for (int i=1; i<=m; i++)
            for (int j=1; j<=k; j++) {
                sum = 0;
                for (int l=1; l<=n; l++)
                    sum += A[i][l] * B[l][j];
                C[i][j] = sum;
            }
    }
    //*****
    static void idFmat(float U[][], int n) {
        for (int i=1; i<=n; i++)
            for (int j=1; j<=n; j++)
                if (i ==j )
                    U[i][j] = 1;
                else
                    U[i][j] = 0;
    }
    //*****
    static void assignFmat(float A[][], float B[][], int m, int n) {
        for (int i=1; i<=m; i++)

```

```

        for (int j=1; j<=n; j++)
            A[i][j] = B[i][j];
    }
    //*****
    static void invMat(float A[][], float I[][], int n) {
        float t, z, p;
        int i, j, k;
        idFmat(I, 4);
        for (k=1; k<=n; k++) {
b:    {
            if (A[k][k] == 0) {
                for (i=k+1; i<=n; i++)
                    if (A[i][k] != 0) {
                        for (j=1; j<=n; j++) {
                            t = A[k][j]; A[k][j] = A[i][j]; A[i][j] = t;
                            t = I[k][j]; I[k][j] = I[i][j]; I[i][j] = t;
                        }
                        break b;
                    }
                System.out.println("The matrix A is not invertible!");
                System.exit(0);
            }
        }
        p = A[k][k];
    }
}

```

```

    for (j=1; j<=n; j++) {
        A[k][j] = A[k][j] / p;
        I[k][j] = I[k][j] / p;
    }
    for (i=1; i<=n; i++)
        if (i != k) {
            z = A[i][k];
            for (j=1; j<=n; j++) {
                A[i][j] = A[i][j] - z * A[k][j];
                I[i][j] = I[i][j] - z * I[k][j];
            }
        }
    }
}
}
}

```

Input/output:

```

Enter the size of the matrix A: 3↵
Now enter the matrix A:
1 1 3↵
2 4 9↵
1 3 0↵

```

The matrix A:

```

1.000000    1.000000    3.000000
2.000000    4.000000    9.000000
1.000000    3.000000    0.000000

```

The inverse of A is:

```

2.250000    -0.750000    0.250000
-0.750000    0.250000    0.250000
-0.166667    0.166667    -0.166667

```

The multiplication A by its inverse:

```

1.000000    0.000000    0.000000
0.000000    1.000000    0.000000
0.000000    0.000000    1.000000

```

9.2 Solving linear equations system

Consider the following system of linear equations including n equations in n unknowns.

$$\begin{aligned}
& A_{11}x_1 + A_{12}x_2 + \cdots + A_{1n}x_n = y_1 \\
& A_{21}x_1 + A_{22}x_2 + \cdots + A_{2n}x_n = y_2 \\
& A_{11}x_1 + A_{12}x_2 + \cdots + A_{1n}x_n = y_1 \\
& A_{21}x_1 + A_{22}x_2 + \cdots + A_{2n}x_n = y_2 \\
& \vdots \\
& A_{n1}x_1 + A_{n2}x_2 + \cdots + A_{nn}x_n = y_n \\
& \vdots \\
& A_{n1}x_1 + A_{n2}x_2 + \cdots + A_{nn}x_n = y_n
\end{aligned}$$

This system can be expressed as the following matrix multiplication.

$$Ax = y. \quad Ax = y.$$

where $A = (A_{ij})_{n \times n}$ $A = (A_{ij})_{n \times n}$ is supposed the coefficients matrix and

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}. \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

It is noteworthy that every column matrix can be regarded as a row matrix and vice versa. Moreover, a row or column matrix can be considered a vector.

The condition for the existence of a solution to this system is that A must be invertible. In other words, its determinant should be nonzero. We study the solution of the system $Ax = y$ $Ax = y$ in two parts.

9.2.1 Direct ways

9.10. Example (Matrix method). If we multiply both sides of the relation $Ax = y$ $Ax = y$ from left to the inverse of A , we obtain the unknown matrix $x = A^{-1}y$ $x = A^{-1}y$ including the solutions. Using the previously mentioned subprograms, write programs in both C++ and Java codes in order to read the of the $n \times n$ coefficients matrix A , as well as the $n \times 1$ matrix y , and then calculate the solutions, if any; otherwise, print a message.

Solution. We have a simple matrix multiplication. Programs P9_10 are the requested programs. In each program, the size of the system $Ax = y$, which is at most 10, is first read. Then, after reading the coefficients matrix A and column vector y , the solutions are calculated.

C++ codes:

```
// Program P9_10 to compute the solutions of a system of
// four equations in four unknowns by direct way
#include <iostream>
#include <iomanip>
using namespace std;
void readFmat(float [][][11], int, int);
void productFmats(float [][][11], float [][][11], float [][][11], int, int, int k);
void det(float [][][11], int);
void invMat(float [][][11], float [][][11], int);

void idFmat(float [][][11], int);
int main() {
    int n;
```

```

float A[11][11], Ainv[11][11], x[11][11], y[11][11];
cout<<"Enter the size of the system Ax=y: ";
cin>>n;
cout<<"Enter the coefficients matrix A:"<<endl;
readFmat(A, n, n);
cout<<"Enter the column matrix y: "<<endl;
readFmat(y, n, 1);
invMat(A, Ainv, n);
productFmats(Ainv, y, x, n, n, 1);
cout<<"\nThe solutions are: \n";
for (int i=1; i<=n; i++)
    cout<<"x("<<i<<")="<<x[i][1]<<endl;
return 0;
}
//*****
void readFmat(float A[][11], int m, int n) {
    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
            cin>>A[i][j];
}
//*****
void productFmats(float A[][11], float B[][11], float C[][11], int m, int n, int k) {
    float sum;
    for (int i=1; i<=m; i++)
        for (int j=1; j<=k; j++) {
            sum = 0;
            for (int l=1; l<=n; l++)
                sum += A[i][l] * B[l][j];
            C[i][j] = sum;
        }
}
//*****
void invMat(float A[][11], float I[][11], int n) {
    float t, p, z;
    int i, j, k;
    idFmat(I, 5);
    for (k=1; k<=n; k++) {
        if (A[k][k] == 0) {
            for (i=k+1; i<=n; i++)
                if (A[i][k] != 0) {
                    for (j=1; j<=n; j++) {
                        t = A[k][j]; A[k][j] = A[i][j]; A[i][j] = t;
                        t = I[k][j]; I[k][j] = I[i][j]; I[i][j] = t;
                    }
                    goto b;
                }
            cout<<"The matrix A is not invertible!";exit(0);
        }
    }
b:  p = A[k][k];
    for (i=1; i<=n; i++) {

```

```

        A[k][j] = A[k][j] / p;
        I[k][j] = I[k][j] / p;
    }

    for (i=1; i<=n; i++)
        if (i != k) {
            z = A[i][k];
            for (j=1; j<=n; j++) {
                A[i][j] = A[i][j] - z * A[k][j];
                I[i][j] = I[i][j] - z * I[k][j];
            }
        }
    }
}

//*****
void idFmat(float U[][11], int n) {
    for (int i=1; i<=n; i++)
        for (int j=1; j<=n; j++)
            if (i == j)
                U[i][j] = 1;
            else
                U[i][j] = 0;
}

```

Input/output:

```

Enter the size of the system Ax=y: 3↵
Enter the coefficients matrix A:
1 1 1↵
4 3 -1↵
3 5 3↵
Enter the column matrix y:
1 6 4↵

```

The solutions are:

```

x(1)=1
x(2)=0.5
x(3)=-0.5

```

java codes:

```

// Program P9_10 to compute the solutions of a system of
// four equations in four unknowns by direct way
import java.util.Scanner;
class P9_10 {
    public static void main(String[] args) {
        Scanner read = new Scanner(System.in);
        int n = read.nextInt();
        float A[][] = new float[n][n];
        float y[] = new float[n];
        for (int i=0; i<n; i++)
            for (int j=0; j<n; j++)
                A[i][j] = read.nextFloat();
        for (int i=0; i<n; i++)
            y[i] = read.nextFloat();
        //*****
        idFmat(A, n);
        //*****
        for (int i=0; i<n; i++)
            System.out.println("x(" + (i+1) + ")=" + A[i][i]);
    }
}

```



```

float A[][] = new float[11][11];
float Ainv[][] = new float[11][11];
float x[][] = new float[11][11];
float y[][] = new float[11][11];
int i, n;
System.out.print("Enter the size of the system Ax=y: ");
n = read.nextInt();
System.out.println("Enter the coefficients matrix A:");
readFmat(A, n, n);
System.out.println("Enter the column matrix y: ");
readFmat(y, n, 1);

invMat(A, Ainv, n);
productFmats(Ainv, y, x, n, n, 1);
System.out.println("\nThe solutions are: \n");
for (i=1; i<=n; i++)
    System.out.println("x(" + i + ")=" + x[i][1]);
read.close();
}
//*****
static void readFmat(float A[][], int m, int n) {
    int i, j;
    Scanner read = new Scanner(System.in);
    for (i=1; i<=m; i++)
        for (j=1; j<=n; j++)
            A[i][j] = read.nextFloat();
//read.close();
}
//*****
static void productFmats(float A[][], float B[][], float C[][],
                        int m, int n, int k) {
    float sum;
    for (int i=1; i<=m; i++)
        for (int j=1; j<=k; j++) {
            sum = 0;
            for (int l=1; l<=n; l++)
                sum += A[i][l] * B[l][j];
            C[i][j] = sum;
        }
}
//*****
static void idFmat(float U[][], int n) {
    for (int i=1; i<=n; i++)
        for (int j=1; j<=n; j++)
            if (i == j )
                U[i][j] = 1;
            else
                U[i][j] = 0;
}
//*****

```

```

static void invMat(float A[][], float I[][], int n) {
    float t, z, p;
    int i, j, k;
    idFmat(I, 4);
    for (k=1; k<=n; k++) {
b:    {
        if (A[k][k] == 0) {
            for (i=k+1; i<=n; i++)
                if (A[i][k] != 0) {
                    for (j=1; j<=n; j++) {
                        t = A[k][j]; A[k][j] = A[i][j]; A[i][j] = t;
                        t = I[k][j]; I[k][j] = I[i][j]; I[i][j] = t;
                    }
                    break b;
                }
            System.out.println("The matrix A is not invertible!");
            System.exit(0);
        }
    }
}

```

```

    }
    p = A[k][k];
    for (j=1; j<=n; j++) {
        A[k][j] = A[k][j] / p;
        I[k][j] = I[k][j] / p;
    }
    for (i=1; i<=n; i++)
        if (i != k) {
            z = A[i][k];
            for (j=1; j<=n; j++) {
                A[i][j] = A[i][j] - z * A[k][j];
                I[i][j] = I[i][j] - z * I[k][j];
            }
        }
    }
}
}
}

```

Input/output:

```

Enter the size of the system Ax=y: 3↵
Enter the coefficients matrix A:
1 1 1↵
4 3 -1↵
3 5 3↵
Enter the column matrix y:
1 6 4↵

```

```

The solutions are:
x(1)=0.9999999
x(2)=0.5
x(3)=-0.5

```

9.11. Example (Cramer method). In this method, the solutions are calculated as follows.

$$x_i = \frac{\det(A_i)}{\det(A)}$$

where the matrix A_i is obtained by replacing the i -th column of A by y .

Solution. This method is rarely used due to the more determinant calculations. However, we leave the reader to use the necessary subprograms and write a program in order to read the $n \times n$ coefficients matrix A as well as the $n \times 1$ matrix y and then calculate the solutions, if any; otherwise, print a message.

9.2.2 Iterative methods

First, two special cases of the system related to linear equations are studied, which are used in the sequel general methods. In this subsection, we deal with x and y as vectors (one-dimensional arrays), write the coefficients matrix in the uppercase or lowercase forms and then represent the system of linear equations in the form $Ax = y$.

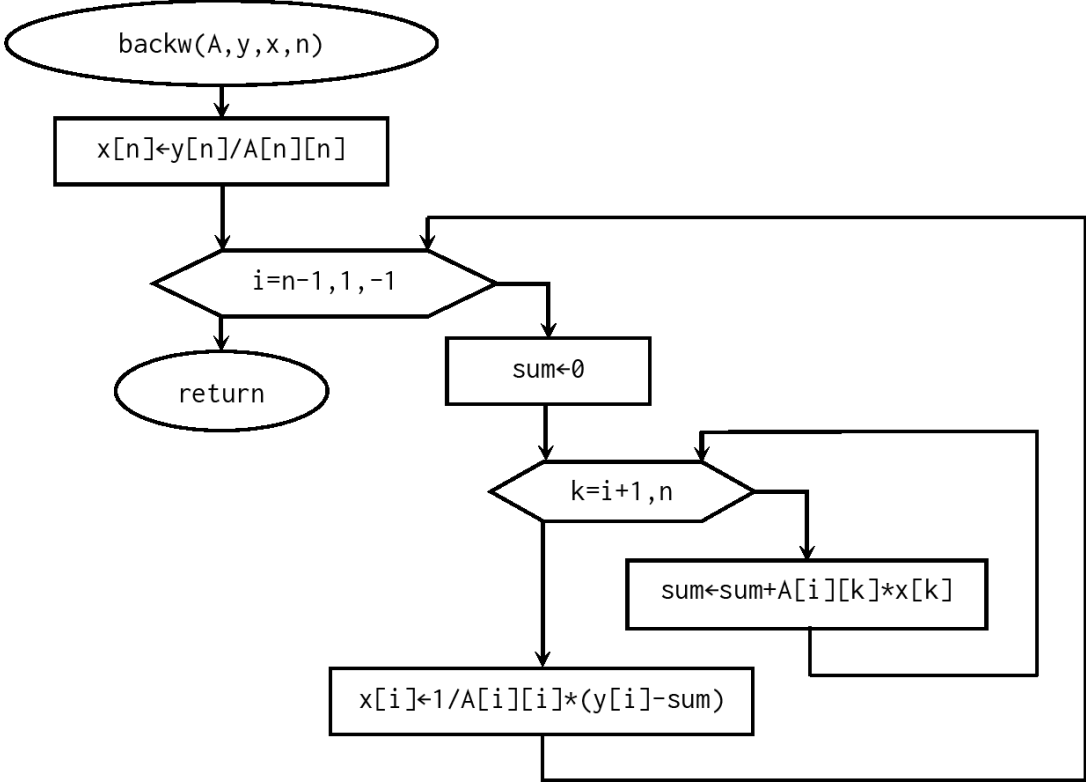


Fig. 9.12: Solving an upper triangular system of equations by the backward method.

9.12) Backward displacement method. Consider the following upper triangular system of linear equations of n equations in n unknowns.

$$\begin{aligned}
A_{11}x_1 + A_{12}x_2 + \cdots + A_{1n}x_n &= y_1 \\
A_{22}x_2 + \cdots + A_{2n}x_n &= y_2 \\
&\vdots \\
A_{11}x_1 + A_{12}x_2 + \cdots + A_{1n}x_n &= y_1 \\
&\quad \quad \quad A_{nn}x_n = y_n \\
A_{22}x_2 + \cdots + A_{2n}x_n &= y_2 \\
&\vdots \\
&\quad \quad \quad A_{nn}x_n = y_n
\end{aligned}$$

This system can be easily solved. We get x_n from the n -th equation, obtain x_{n-1} placing x_n in the $(n-1)$ -th equation, and then continue this process backward. Finally, we arrive at x_1 . The answers are as follows.

$$\begin{aligned}
x_n &= \frac{y_n}{A_{nn}}, & x_i &= \frac{1}{A_{ii}} \left(y_i - \sum_{k=i+1}^n A_{ik}x_k \right), & i &= n-1, \dots, 2, 1. \\
x_n &= \frac{y_n}{A_{nn}}, & x_i &= \frac{1}{A_{ii}} \left(y_i - \sum_{k=i+1}^n A_{ik}x_k \right), & i &= n-1, \dots, 2, 1.
\end{aligned}$$

Flowchart 9.12 displays a sub-algorithm which receives the coefficients matrix A and the vector y and then calculates and returns the solutions vector x .

This flowchart is written in C++ and Java codes as follows.

C++ codes:

```

void backw(float A[][4], float x[],
           float y[], int n) {
    float sum;
    x[n]=y[n]/A[n][n];
    for (int i=n-1; i>=1; i--) {
        sum=0;
        for (int k=i+1; k<=n; k++)
            sum = sum+A[i][k]*x[k];
        x[i]=(y[i]-sum)/A[i][i];
    }
}

```

Java codes:

```

static void backw(float A[][], float x[],
                 float y[], int n)
{
    float sum;
    x[n]=y[n]/A[n][n];
    for (int i=n-1; i>=1; i--) {
        sum=0;
        for (int k=i+1; k<=n; k++)
            sum=sum+A[i][k]*x[k];
        x[i]=(y[i]-sum)/A[i][i];
    }
}

```

9.13. Forward displacement method. This time, we consider the lower triangular system of linear equations of n equations in n unknowns..

$$\begin{array}{rcl}
 A_{11}x_1 & & = y_1 \\
 A_{21}x_1 + A_{22}x_2 & & = y_2 \\
 & \vdots & \\
 A_{n1}x_1 + A_{n2}x_2 + \cdots + A_{nn}x_n & = & y_n \\
 A_{11}x_1 & & = y_1 \\
 A_{21}x_1 + A_{22}x_2 & & = y_2 \\
 & \vdots & \\
 A_{n1}x_1 + A_{n2}x_2 + \cdots + A_{nn}x_n & = & y_n
 \end{array}$$

This system can be solved in a similar way as the backward system with the following solutions.

$$x_1 = \frac{y_1}{A_{11}}, \quad x_i = \frac{1}{A_{ii}} \left(y_i - \sum_{k=1}^{i-1} A_{ik} x_k \right), \quad i = 1, 2, \dots, n.$$

$$x_1 = \frac{y_1}{A_{11}}, \quad x_i = \frac{1}{A_{ii}} \left(y_i - \sum_{k=1}^{i-1} A_{ik} x_k \right), \quad i = 2, 3, \dots, n.$$

We only write the subprogram with a slight changes in the flowchart and subprogram of the backward method.

C++ codes:

```
void forw(float A[][4], float x[],
         float y[], int n) {
    float sum;
    x[1]=y[1]/A[1][1];
    for (int i=2; i<=n; i++) {
        sum=0;
        for (int k=1; k<=i-1; k++)
            sum=sum+A[i][k]*x[k];

        x[i]=(y[i]-sum)/A[i][i];
    }
}
```

Java codes:

```
static void forw(float A[][], float x[],
                float y[], int n) {
    float sum;
    x[1]=y[1]/A[1][1];
    for (int i=2; i<=n; i++) {
        sum=0;
        for (int k=1; k<=i-1; k++)
            sum=sum+A[i][k]*x[k];

        x[i]=(y[i]-sum)/A[i][i];
    }
}
```

Now, we study the system of linear equations $Ax = y$ in a general case in two methods as follows.

9.14. Example (The Gauss elimination method). In this method, the system of linear equations $Ax = y$ is turned to an upper triangular system as in the backward displacement method and then the solutions are calculated using the backward displacement method. Write a sub-algorithm named Gauss() in order to receive the coefficients matrix A and the vector y . Then, calculate and return the solutions vector x using the Gauss elimination method.

Solution. We discuss the solution in four steps:

Form the augmented matrix with the same name A . This matrix is obtained by the matrix A followed by the (column) vector y . This is easily implemented by a for loop.

1. Turn the augmented matrix A to a matrix in which the square part corresponding to the coefficients matrix is the upper triangular using the elementary row operations. This may be performed exactly in a similar manner as in the function `det()` in **Example 9.8** with only two slight differences. The process related to the carrier d is removed. Additionally, the initial values related to the column from n to $n + 1$ are changed.
2. At this point, we have the upper triangular system $Ax = y$ where A is now an upper triangular matrix and the new vector y is the deformation of the original one. However, the current deformed matrix A includes both the mentioned upper triangular matrix and thus, the new vector y and we should separate the vector y from it. To this end, the substitution processes are performed opposite to those in the first step.
3. The solutions are calculated and returned calling the subprogram `backw()`.

Flowchart 9.14 includes all the above steps which clearly described. Programs P9_14 translate this flowchart into C++ and Java codes. Each program reads first the size of the system which is supposed at most 10. Then, it reads the coefficients matrix A and the column vector y and determine the solutions.

Each of the Programs P9_14 reads the size n of the system, which is at most 10, coefficients matrix A , and the vector y . Then, calculates and prints the solution of the linear system $Ax = y$ using the Gauss method.

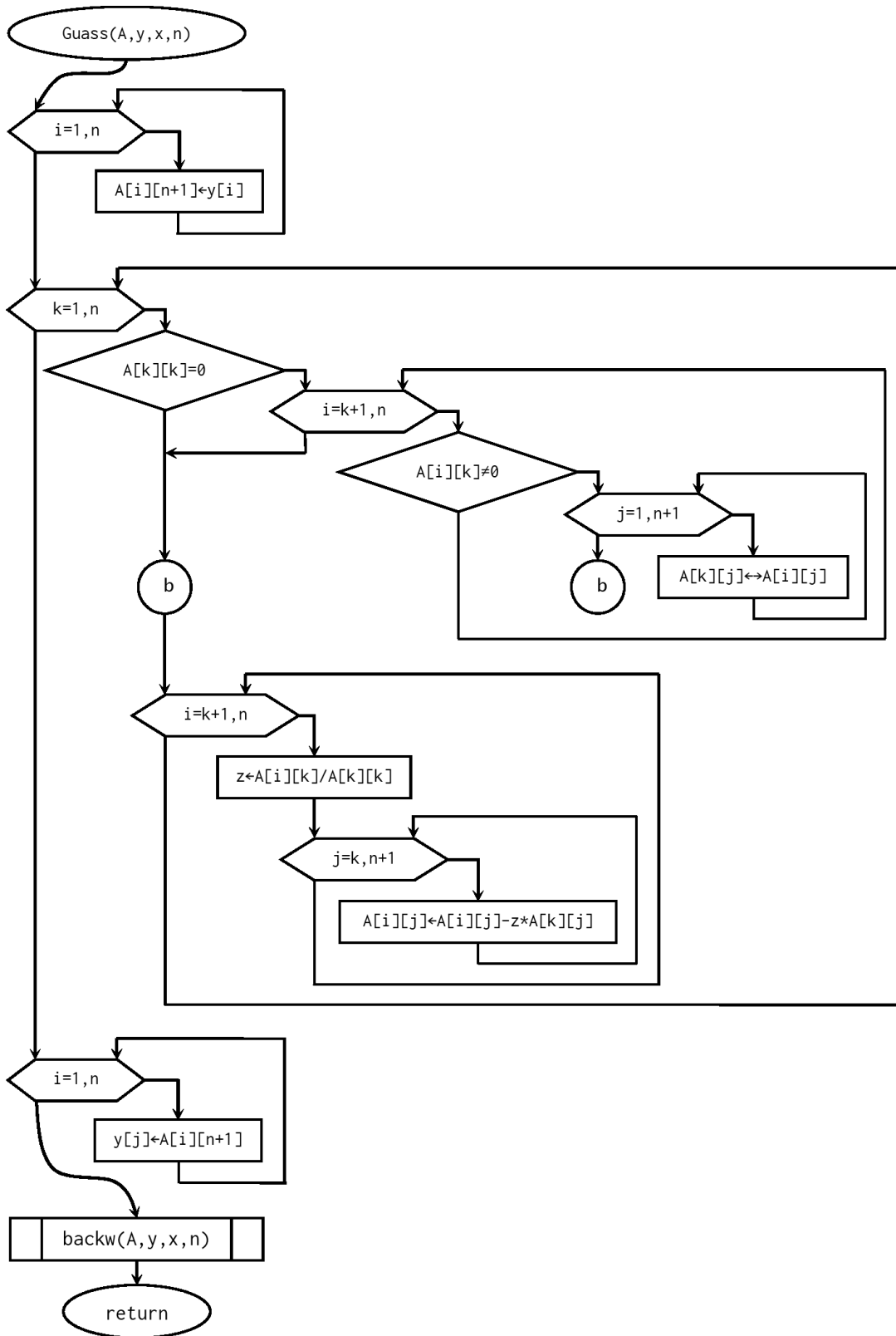


Fig. 9.14: Solving a system of linear equations by the Gauss elimination method.

C++ codes:

```

-----
// Program P9_14 to solve the system of linear equations of
// n equations in n un-knowns by the Gauss elimination method
#include <iostream>
#include <iomanip>
using namespace std;
void readFmat(float [][][11], int, int);
void Gauss(float [][][11], float [], float [], int);
void backw(float [][][11], float [], float [], int);
int main() {
    float A[11][11], x[11], y[11];
    int i, n;
    cout<<"Enter the size of the system of the system Ax=y: ";
    cin>>n;
    cout<<"Enter the coefficients matrix A:"<<endl;
    readFmat(A, n, n);
    cout<<"Enter the column matrix y: "<<endl;
    for (i=1; i<=n; i++)
        cin>>y[i];
    Gauss(A, y, x, n);
    cout<<"The solutions are: "<<endl;
    for (i=1; i<=n; i++)
        cout<<"x("<<i<<")="<<x[i]<<endl;
    return 0;
}
//*****
void readFmat(float A[][][11], int m, int n) {
    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
            cin>>A[i][j];
}
//*****
void Gauss(float A[][][11], float y[], float x[], int n) {
    float t, z;
    int i, j, k;
    for (i=1; i<=n; i++)
        A[i][n+1] = y[i];
    for (k=1; k<=n; k++) {
        if (A[k][k] == 0) {
            for (i=k+1; i<=n; i++)
                if (A[i][k] != 0) {
                    for (j=1; j<=n+1; j++) {
                        t = A[k][j]; A[k][j] = A[i][j]; A[i][j] = t;
                    }
                    goto b;
                }
        }
    }
b: for (i=k+1; i<=n; i++) {
    z = A[i][k] / A[k][k];
    for (j=k; j<=n+1; j++)
        A[i][j] = A[i][j] - z * A[k][j];
}
}

```

```

        }
    }
    for (i=1; i<=n; i++)
        y[i] = A[i][n+1];

    backw(A, y, x, n);
}
//*****
void backw(float A[][11], float y[], float x[], int n) {
    float sum;
    x[n] = y[n] / A[n][n];
    for (int i=n-1; i>=1; i--) {
        sum = 0;
        for (int k=i+1; k<=n; k++)
            sum = sum + A[i][k] * x[k];
        x[i] = 1 / A[i][i] * (y[i] - sum);
    }
}
}

```

Input/output:

Enter the size of the system of the system Ax=y: 3↵

Enter the coefficients matrix A:

1 1 1↵

4 3 -1↵

3 5 3↵

Enter the column matrix y:

1 6 4↵

The solutions are:

x(1)=1

x(2)=0.5

x(3)=-0.5

Java codes:

```

// Program P9_14 to solve the system of linear equations of
// n equations in n un-knowns by the Gauss elimination method
import java.util.Scanner;
class P9_14 {
    public static void main(String[] args) {
        Scanner read = new Scanner(System.in);
        float A[][] = new float[11][11];
        float x[] = new float[11];
        float y[] = new float[11];
        int i, n;
        System.out.print("Enter the size of the system Ax=y: ");
    }
}

```

```

n = read.nextInt();
System.out.println("Enter the coefficients matrix A:");
readFmat(A, n, n);
System.out.println("Enter the column matrix y: ");
for (i=1; i<=n; i++)
    y[i] = read.nextFloat();
Gauss(A, x, y, n);
System.out.println("\nThe solutions are: ");
for (i=1; i<=n; i++)
    System.out.println("x(" + i + ")=" + x[i]);
read.close();

}
//*****
static void readFmat(float A[][], int m, int n) {
    Scanner read = new Scanner(System.in);
    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
            A[i][j] = read.nextFloat();
//read.close();
}
//*****
static void Gauss(float A[][], float x[], float y[], int n) {
    float t, z;
    int i, j, k;
    for (i=1; i<=n; i++)
        A[i][n+1] = y[i];
    for (k=1; k<=n; k++) {
b:    {
        if (A[k][k] == 0) {
            for (i=k+1; i<=n; i++)
                if (A[i][k] != 0) {
                    for (j=1; j<=n+1; j++) {
                        t = A[k][j]; A[k][j] = A[i][j]; A[i][j] = t;
                    }
                    break b;
                }
        }
    }
    for (i=k+1; i<=n; i++) {
        z = A[i][k] / A[k][k];
        for (j=k; j<=n+1; j++)
            A[i][j] = A[i][j] - z * A[k][j];
    }
}
    for (i=1; i<=n; i++)
        y[i] = A[i][n+1];
    backw(A, x, y, n);
}
//*****
static void backw(float A[][], float x[], float y[], int n) {
    float sum;

```

```

x[n] = y[n] / A[n][n];
for (int i=n-1; i>=1; i--) {
    sum = 0;
    for (int k=i+1; k<=n; k++)
        sum = sum + A[i][k] * x[k];
    x[i] = 1 / A[i][i] * (y[i] - sum);
}
}
}

```

Input/output:

Enter the size of the system of the system Ax=y: 3↵

Enter the coefficients matrix A:

1 1 1↵

4 3 -1↵

3 5 3↵

Enter the column matrix y:

1 6 4↵

The solutions are:

x(1)=1

x(2)=0.5

x(3)=-0.5

9.15. Example (The triangular decomposition method). In this method, the coefficients matrix A is decomposed, in some ways which are latter explained, into two upper triangular matrix U and lower triangular matrix L , that is, $A = LU$. Now, the system $Ax = y$ turns to $LUx = y$. Then, we get $Lz = y$ putting $Ux = z$. As shown, we are involved with the two systems of linear equations as follows.

$$Lz = y, \quad Ux = z. \quad LZ = y, \quad Ux = z.$$

where L and U are regarded as lower and upper triangular matrices, respectively. Therefore, having the vector y , we first solve the system $Lz = y$ to achieve the required solutions using the forward displacement method and then get the solutions vector z of this system. Next, we solve the system $Ux = z$ and obtain the required solution vector x using the backward displacement method.

Now, we explain the Doolittle manner of decomposition of the matrix A into two upper and lower triangular matrices U and L , respectively. In addition, there are two other decomposition manners, the Choleski manner and the Crout manner. We cite [??] for more details about these manners. We omit the details of decomposition in the following Doolittle manner.

Consider the matrices L and U as follows.

$$L = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ L_{21} & 1 & 0 & \cdots & 0 \\ L_{31} & L_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ L_{n1} & L_{n2} & L_{n3} & \cdots & 1 \end{bmatrix}, \quad U = \begin{bmatrix} U_{11} & U_{12} & U_{13} & \cdots & U_{1n} \\ 0 & U_{22} & U_{23} & \cdots & U_{2n} \\ 0 & 0 & U_{33} & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & U_{nn} \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ L_{21} & 1 & 0 & \cdots & 0 \\ L_{31} & L_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ L_{n1} & L_{n2} & L_{n3} & \cdots & 1 \end{bmatrix}, \quad U = \begin{bmatrix} U_{11} & U_{12} & U_{13} & \cdots & U_{1n} \\ 0 & U_{22} & U_{23} & \cdots & U_{2n} \\ 0 & 0 & U_{33} & \cdots & U_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & U_{nn} \end{bmatrix}$$

It is proved that, for $i = 1, \dots, n$, $i = 1, \dots, n$, the i -th row of U is:

$$U_{ik} = A_{ik} - \sum_{j=1}^{i-1} L_{ij} U_{jk}, \quad k = i, \dots, n,$$

$$U_{ik} = A_{ik} - \sum_{j=1}^{i-1} L_{ij} U_{jk}, \quad k = i, \dots, n,$$

and the i -th column of L is:

$$L_{ki} = \frac{1}{U_{ii}} \left(A_{ki} - \sum_{j=1}^{i-1} L_{kj} U_{ji} \right), \quad k = i + 1, \dots, n.$$

$$L_{ki} = \frac{1}{U_{ii}} \left(A_{ki} - \sum_{j=1}^{i-1} L_{kj} U_{ji} \right), \quad k = i + 1, \dots, n.$$

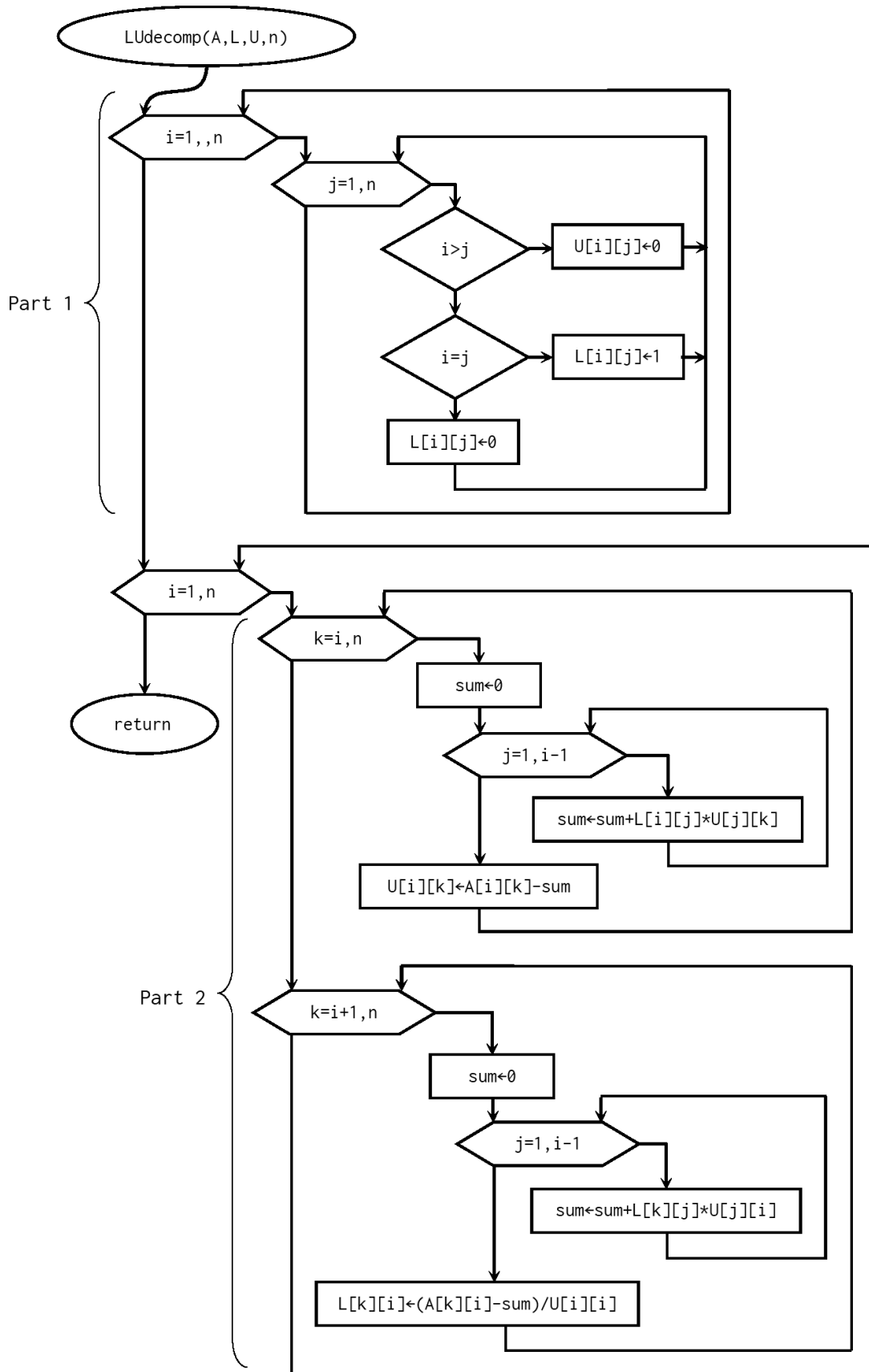


Fig. 9.15: Solving a system of linear equations by the triangle decomposition method.

Write a sub-algorithm named `LUdecomp()` to receive an $n \times n$ matrix A and then calculate and return L and U based on the above relations. Next, write a main program to read the coefficients matrix A of the system $Ax = y$ as well as the vector y . Then, receive the solutions vector z of the system $Lz = y$ calling the sub-algorithm `forw()`. Finally, receive the solutions vector x of the system $Ux = z$ calling the sub-algorithm `backw()` and print the entries of x as the required solutions.

Solutions. We write the sub-algorithm `LUdecomp()` in two parts as follows.

1. Produce the 0- and 1-entries of both matrices L and U (Part 1 in Fig. 9.15);
2. Generate the i -th row of U as well as the i -th column of L for $i = 1, \dots, n$ ($i = 1, \dots, n$) (Part 2 in Fig. 9.15).

The main unit in Programs P9_15,

1. reads the size n of the system $Ax = y$, the $n \times n$ coefficients matrix A of the system, and the vector y , respectively;
2. calls the sub-algorithm `LUdecomp()` for A and receives the matrices L and U ;
3. calls the sub-algorithm `forw()` for L and y and receives the solutions vector z of the system $Lz = y$;
4. calls the sub-algorithm `backw()` for U and z and receives the solutions vector x of the system $Ux = z$;
5. prints the entries of x as the required solutions.

In each of the Programs P9_15, after reading the size n of the linear system (at most 10), coefficients matrix A , and the vector y , first, the lower and upper triangular matrices L and U and then, the z -solutions of the system $Lz = y$, as well as the x -solutions of the system $Ux = z$ are calculated and printed. The x -solutions are indeed the solutions of the linear system $Ax = y$

C++ codes:

```
// Program P9_15 to solve the system of linear equations of n equations
// in n unknowns by the triangular decomposition method
#include <iostream>
#include <iomanip>
using namespace std;
//*****
void readDmat(float A[][11], int m, int n) {
    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
            cin>>A[i][j];
}
//*****
void LUdecomp(float A[][11], float L[][11], float U[][11], int n) {
    for (int i=1; i<=n; i++)
        for (int j=1; j<=n; j++)
            if (i > j)
                U[i][j] = 0;

            else if (i == j)
                L[i][j] = 1;
            else
                L[i][j] = 0;
    for (int i=1; i<=n; i++) {
        for (int k=i; k<=n; k++) {
            int sum = 0;
            for (int j=1; j<=i-1; j++)
                sum += (L[i][j] * U[j][k]);
            U[i][k] = A[i][k] - sum;
        }
        for (int k=i+1; k<=n; k++) {
            int sum = 0;
            for (int j=1; j<=i-1; j++)
                sum += (L[k][j] * U[j][i]);
            L[k][i] = (A[k][i] - sum) / U[i][i];
        }
    }
}
//*****
void forw(float A[][11], float x[], float y[], int n) {
    float sum;
    x[1] = y[1] / A[1][1];
    for (int i=2; i<=n; i++) {
        sum = 0;
        for (int k=1; k<=i-1; k++)
            sum = sum + A[i][k] * x[k];
        x[i] = (y[i] - sum) / A[i][i];
    }
}
//*****
```

```

void backw(float A[][11], float x[], float y[], int n) {
    float sum;
    x[n] = y[n] / A[n][n];
    for (int i=n-1; i>=1; i--) {
        sum = 0;
        for (int k=i+1; k<=n; k++)
            sum = sum + A[i][k] * x[k];
        x[i] = (y[i] - sum) / A[i][i];
    }
}
//*****
void writeDmat(float A[][11], int m, int n) {
    for (int i=1; i<=m; i++) {
        for (int j=1; j<=n; j++)
            cout<<setw(6)<<A[i][j];
        cout<<endl;
    }
}
//*****
int main() {
    int i, n;
    float A[11][11], L[11][11], U[11][11], x[11], y[11], z[11];
    cout<<"Enter the size of the system Ax=y: ";
    cin>>n;
    cout<<"Enter the coefficients matrix A:\n";
}

```

```

readDmat(A, n, n);
cout<<"Enter the vector y:\n";
for (i=1; i<=n; i++)
    cin>>y[i];
LUdecomp(A, L, U, n);
cout<<"\nThe matix L:\n\n";
writeDmat(L, n, n);
cout<<"\nThe matix U:\n\n";
writeDmat(U, n, n);
forw(L, z, y, n);
cout<<"\nThe z-solutions are:\n\n";
for (i=1; i<=n; i++)
    cout<<"z("<<i<<"")="<<z[i]<<endl;
for (i=1; i<=n; i++)
    backw(U, x, z, n);
cout<<"\nThe x-solutions are:\n\n";
for (i=1; i<=n; i++)
    cout<<"x("<<i<<"")="<<x[i]<<endl;
return 0;
}

```

Input/Output:

Enter the size of the system $Ax=y$: 3↵

Enter the coefficients matrix A:

1 1 1↵

4 3 -1↵

3 5 3↵

Enter the vector y:

1 6 4↵

The matix L:

```

1    0    0
4    1    0
3   -2    1

```

The matix U:

```

1    1    1
0   -1   -5
0    0  -10

```

The z-solutions are:

$z(1)=1$

$z(2)=2$

$z(3)=5$

The x-solutions are:

$x(1)=1$

$x(2)=0.5$

$x(3)=-0.5$

Java codes:

```
// Program 9_15 to solve the system of linear equations of n equations
// in n unknowns by the triangular decomposition method
import java.util.Scanner;
class P9_15 {
    static void readFmat(float A[][], int m, int n) {
        Scanner read = new Scanner(System.in);
        for (int i=1; i<=m; i++)
            for (int j=1; j<=n; j++)
                A[i][j] = read.nextFloat();
        //read.close();
    }
    //*****
    static void LUdecomp(float A[][], float L[][], float U[][], int n) {
        for (int i=1; i<=n; i++)
            for (int j=1; j<=n; j++)
                if (i > j)
                    U[i][j] = 0;
                else if (i ==j )
                    L[i][j] = 1;
                else
                    L[i][j] = 0;
        for (int i=1; i<=n; i++) {
            for (int k=i; k<=n; k++) {
                int sum = 0;
                for (int j=1; j<=i-1; j++)
                    sum += (L[i][j] * U[j][k]);
                U[i][k] = A[i][k] - sum;
            }
            for (int k=i+1; k<=n; k++) {
                int sum = 0;
                for (int j=1; j<=i-1; j++)
                    sum += (L[k][j] * U[j][i]);
                L[k][i] = (A[k][i] - sum) / U[i][i];
            }
        }
    }
    //*****
    static void forw(float A[][], float x[], float y[], int n) {
        float sum;
        x[1] = y[1] / A[1][1];
        for (int i=2; i<=n; i++) {
            sum = 0;
            for (int k=1; k<=i-1; k++)
                sum = sum + A[i][k] * x[k];
            x[i] = (y[i] - sum) / A[i][i];
        }
    }
    //*****
    static void backw(float A[][], float x[], float y[], int n) {
```

```

float sum;
x[n] = y[n] / A[n][n];
for (int i=n-1; i>=1; i--) {
    sum = 0;
    for (int k=i+1; k<=n; k++)

        sum = sum + A[i][k] * x[k];
    x[i] = (y[i] - sum) / A[i][i];
}
}
//*****
static void writeFmat(float A[][], int m, int n) {
    for (int i=1; i<=m; i++) {
        for (int j=1; j<=n; j++)
            System.out.format("%9.2f", A[i][j]);
        System.out.println();
    }
}
//*****
public static void main(String[] args) {
    Scanner read = new Scanner(System.in);
    float A[][] = new float[11][11];
    float L[][] = new float[11][11];
    float U[][] = new float[11][11];
    float x[] = new float[11];
    float y[] = new float[11];
    float z[] = new float[11];
    System.out.print("Enter the size of the system Ax=y: ");
    int n = read.nextInt();
    System.out.println("Enter the coefficients matrix A:");
    readFmat(A, 3, 3);
    System.out.println("Enter the vector y:");
    for (int i=1; i<=n; i++)
        y[i] = read.nextFloat();
    LUdecomp(A, L, U, n);
    System.out.println("\nThe matix L:\n");
    writeFmat(L, n, n);
    System.out.println("\nThe matix U:\n");
    writeFmat(U, n, n);
    forw(L, z, y, n);
    System.out.println("\nThe z-solutions are:\n");
    for (int i=1; i<=n; i++)
        System.out.println(z[i]);
    backw(U, x, z, n);
    System.out.println("\nThe x-solutions are:\n");
    for (int i=1; i<=n; i++)
        System.out.println("z(" + i + ")=" + z[i]);
}
}

```

Input/Output.

input/output:

Enter the size of the system $Ax=y$: 3↵

Enter the coefficients matrix A:

1 1 1↵

4 3 -1↵

3 5 3↵

Enter the vector y:

1 6 4↵

The matrix L:

1.00 0.00 0.00

4.00 1.00 0.00

3.00 -2.00 1.00

The matrix U:

1.00 1.00 1.00

0.00 -1.00 -5.00

0.00 0.00 -10.00

The z-solutions are:

1.0

2.0

5.0

The x-solutions are:

$z(1)=1.0$

$z(2)=2.0$

$z(3)=5.0$

Exercises

In the following exercises: (1) Arrange the implementation table, if needed, (2) Write the complete program, and (3) Provide appropriate input notifications and output headings, if any. In addition, the user-defined functions in the text of the current as well as the previous chapters may be used unless otherwise is explicitly specified.

9.1. Write an algorithm to create the multiplication table of the numbers from 1 to 10 and print it with an appropriate format.

9.2. Write an algorithm to read an integer square matrix A and save its transpose on itself.

9.3. Write a sub-algorithm to receive two integer matrices and determine whether or not they are equal by returning one of the numbers 1, for the equal case, or 0, otherwise.

9.4. Write an algorithm to print a 5×5 integer matrix A in which the entries on the main and secondary diagonals are 1 whereas the remaining entries are considered 0.

9.5. Write an algorithm to read a 7×7 integer matrix and swap its main diagonal with the secondary diagonal.

9.6. Write an algorithm to read an $m \times n$ real matrix A and swap the distinct rows r and s of the matrix.

9.7. Repeat the previous exercise for columns instead of rows.

9.8. Write an algorithm to read an $m \times n$ real matrix A and apply the following substitution. It is supposed that $z \neq 0$ and $r \neq t$.

$$\begin{aligned} & r\text{-th row} \leftarrow r\text{-th row} + z \times t\text{-th row.} \\ & \mathbf{r\text{-th row} \leftarrow r\text{-th row} + z \times t\text{-th row.} \end{aligned}$$

9.9. Repeat the previous exercise for columns instead of the rows.

9.10. Write an algorithm to read an $m \times n$ real matrix and multiply its r -th row by the real number z .

9.11. Repeat the previous exercise for columns instead of the rows.

9.12. Write an algorithm to read the two positive integers m and n which are at most 10 and then print all the base $m \times n$ integer matrices. A base matrix E_{ij} is a matrix where the (i, j) -entry is considered 1 while the remaining entries are 0.

9.13. Write an algorithm to read the entries of a 6×6 real matrix and convert it into an upper triangular matrix using the elementary row operations.

9.14. Repeat the above exercise for the lower triangular case.

9.15. Write a function named searchFmat() to receive the $m \times n$ real matrix A and a real number t . Then, return 1 if t is an entry of A ; otherwise, return 0. Now, write a main algorithm to read a 4×6 real matrix as well as 10 real numbers one by one and each time determine whether the number is a member of the matrix by printing one of the messages Yes or No calling the function searchFmat().

9.16. Write an algorithm to read the student number and grade of 36 students and save them in a 2×36 real matrix. Then, read a number and announce whether the corresponding student passed or failed by printing one of the messages Passed or Failed if this number is among the 36 student numbers. Otherwise, print the message Not in list.

9.17. Using the Cramer method, write a program to solve the system $AX = Y$ of 10 equations in 10 unknowns.

9.18. Repeat the previous exercise for the Gauss method.

Supplementary exercises

9.1*. Write a program for printing each of the patterns below. The logic of the program should be in a way so that the pattern can be developed

to larger sizes by changing only a single number.

(1)	(2)	(3)
1	1	10 9 8 7 6
2 3	2 3 2	10 9 8 7
3 4 5	3 4 5 4 3	10 9 8
4 5 6 7	4 5 6 7 6 5 4	10 9
5 6 7 8 9	5 6 7 8 9 8 7 6 5	10

(4)	(5)	(6)
1	1	4 4 4 4 4 4 4
1 2 1	1 2	4 3 3 3 3 3 4
1 2 3 2 1	1 2 3	4 3 2 2 2 3 4
1 2 3 4 3 2 1	1 2 3 4	4 3 2 1 2 3 4
1 2 3 2 1	1 2 3	4 3 2 2 2 3 4
1 2 1	1 2	4 3 3 3 3 3 4
1	1	4 4 4 4 4 4 4

9.2*. Write an algorithm to read the 5×7 real matrix A . Then, calculate and print the maximum entry in each row together with its position. Finally, print the last position of the maximum entry if it is repeated.

9.3*. Write an algorithm to replace every entry of an integer matrix by its reverse using the `rev()` function in [Example 7.4](#).

9.4*. Write an algorithm to read an 8×8 real matrix A and sort its rows alternately in ascending and descending orders. Then, print the resulted matrix.

9.5*. A magic matrix is a square matrix in which the sum of the entries on each row and each column is equal to the sum of the entries of the main diagonal and secondary diagonal. Write a sub-algorithm named `Magic()` to receive an $n \times n$ square integer matrix and determine

whether or not the matrix is a magic matrix by returning one of the integers 1 or 0, respectively.

9.6*. Write a sub-algorithm named $A_{ij}()$ to receive an $n \times n$ square real matrix A , a row number i , and a column number j . Then, return an $(n - 1) \times (n - 1)$ matrix (with the same name A_{hat}) which is obtained from A by removing the row i and column j of A .

9.7*. Write a function to receive a real $n \times n$ matrix A and then calculate and return its determinant using the recursive method.

9.8*. Given an $n \times n$ real matrix A , the adjacent matrix $\text{adj}(A)$ is defined as follows.

$$\begin{aligned} (\text{adj}(A))_{ij} &= (-1)^{i+j} \det(A_{\hat{i}\hat{j}}), \quad i = 1, 2, \dots, n, \quad j = 1, 2, \dots, n, \\ (\text{adj}(A))_{ji} &= (-1)^{i+j} \det(A_{\hat{i}\hat{j}}), \quad i = 1, 2, \dots, n, \quad j = 1, 2, \dots, n, \end{aligned}$$

where the matrix $A_{\hat{i}\hat{j}}$ is obtained from A by removing row i and column j of A . The adjacent matrix is used for calculating the inverse of a matrix A using the following formula.

$$A^{-1} = \frac{1}{\det(A)} \text{adj}(A).$$

Write a sub-algorithm to receive a real square matrix A and then calculate and return A^{-1} using this method.

9.9*. An $m \times n$ matrix A is called row-reduced if:

- The first nonzero entry in each nonzero row of A is equal to 1;
- Each column of A which contains the leading non-zero entry of some row has its other entries 0.

Write an algorithm to read an $m \times n$ real matrix A . Then, convert A into a row-reduced matrix and print it using the elementary row operations.

9.10*. An $m \times n$ matrix A is called a row-reduced echelon matrix if:

- a) A is row-reduced;
- b) Every row of A which has all its entries 0 occurs below every row which has a nonzero entry;
- c) If rows $1, 2, \dots, r$ are the nonzero rows of A , $i = 1, 2, \dots, r$,
 $k_i < k_{i+1} < \dots < k_r$,
 $k_i < k_{i+1} < \dots < k_r$ and if the leading nonzero entry of row i occurs in column k_i then In other words, no zero row occurs above any nonzero row.

Write an algorithm to read a real $m \times n$ matrix A . Then, convert A into a row-reduced echelon matrix using the elementary row operations and then print it. It is assumed that there is no zero row in the resulted row-reduced echelon matrix.

Hints for the exercises

To solve the exercises, you fail to obtain a good result and you are unable to develop your ability in algorithm writing and thus programming if you first refer to their solutions without attacking these exercises. Therefore, you should draw the flowcharts of the exercises by yourself and test them by arranging their implementation table. Then, you should write their program and be assured of their correctness by running it in the computer. It is worth mentioning that the algorithms and the programs should work for various inputs rather than just certain ones. Next, you can take a look at the hints of the exercises in this part. However, your technique may differ from the technique used in the present part which is natural since the techniques suggested here are not unique!

You can follow its hints in this section and solve it accordingly if you are really unable to find the solution for a certain exercise. These hints are written such that they offer the path for solving the exercise and the reader should complete the process of solving the exercise by themselves. There is a hint for nearly all the exercises except for the simple problems and repeated ideas. The codes of mathematical library functions are often in C++. Therefore, in Java you should add the prefix 'Math.' to them.

4.2. A number is a multiple of n if the remainder of the division of that number by n is zero.

4.3. Comparing the real value of \sqrt{n} with the integer part of \sqrt{n} is one way to find whether or not the number n is square.

4.4. In a triangle, any pair of edges satisfy the condition: the sum of two edges is greater than the third edge. Therefore, you should use an if template with the condition containing three parts separated by the && operator.

4.5. In a right-angle triangle, the pair of edges adjacent to the right angle satisfy the Pythagorean relation. Thus, you should use an if template with the condition containing three parts separated by the || operator. Note that the establishment of the Pythagorean relation is sufficient to conclude that the shape is a right-angle triangle.

4.6. Use an if template with a single condition checking the equality of the three numbers.

4.7. Use an if template with the following condition.

$$(a==b \ \&\& \ a+b<c) \ || \ (a==c \ \&\& \ a+c<b) \ || \ (b=c \ \&\& \ b+c<a)$$

4.8. The areas of a circle with the radius r as well as the inscribed and circumscribed squares are πr^2 , r^2 , and $r^2/2$, πr^2 , r^2 , and $r^2/2$, respectively.

4.9. A number is even if the remainder of its division by 2 is zero.

4.10. The required number of digits for the arbitrary integer n is $\log_{10}(\text{abs}(n))+1$.

4.11. Use an if-else-if template.

4.12. Consider the divisibility of $n-1980$ by 4.

4.13. In an if-else-if template, print 31 if $1 \leq n \leq 6$; otherwise, print 30 if $7 \leq n \leq 11$. Furthermore, print 29 if $n = 12$; otherwise, print the requested message. Moreover, you can use the switch template.

4.14. Consider the first six months separately from the second six months in a two-way branching. The required formula can be easily achieved by a manual calculation of two or three cases: in the first six months the required number is $(m - 1) \times 31 + d$ while in the second six months this number is equal to $(m - 7) \times 30 + d + 186$.

4.15. Use an if-else-if template and locate the process in **Exercise 4.14** in the else-part.

4.16. The result of $n + 3$ modulo 7 is the required day.

4.17. The corresponding number for Saturday is 4. Therefore, the result of $n + f - 1$ modulo 7 is the required day.

4.18. Consider the quotient of $n - 10$ divided by 30.

There is no code for the following three exercises in Java language due to the existence of the goto statement if we fail to use the loops. However, the codes can be easily written if either while or do-while loop is used (refer to **Chapter 7**) in which case, there is no need for the goto statement even in C++. The following hints regarding the above-mentioned exercises are valid in the C++ language.

4.19. Use the logic of [Exercise 4.15](#). Label the input template as ‘100’ and add an if template just after the if-else-if template such that the implementation control goes to the label 100 if the condition $m=0$ && $n=0$ is true. Note that the if statement is as follows in the C++ codes:

```
if (m==0 && n==0) goto 100;
```

4.20. Use the switch template. In the division case, print the message if the dividend is zero (you can use the if-else template for this purpose). For repetition, see the hint of the last part of [Exercise 4.19](#).

4.21. Use a switch or an if-then-if template to provide the multi-way branching process. Label the reading c construction as ‘100’. Then, print Quit? and label this construction as ‘200’. Read a character type variable, say p . Next, in an if-else-if template,

- if one of the characters n or N is entered, terminate the algorithm using the `exit(0)` construction;
- otherwise, transfer the implementation control to the label 100 if one of the characters y or Y is entered;
- otherwise, transfer the control to the label 200.

5.3. Refer to the hint of [Exercise 4.3](#).

5.4. Consider the remainder of the division n by 2.

5.7. Refer to the hint of [Exercise 4.4](#).

5.8. Refer to the hint of [Exercise 4.5](#).

5.9. Use the if-else-if template with three conditions. Additionally, the nested if-else templates can be used.

5.10. Refer to the hint of [Exercise 4.14](#).

5.11. Refer to the hint of [Exercises 4.16](#) and [4.17](#).

5.12. Refer to the hint of [Exercise 4.19](#) for the function (consider also the paragraph before this exercise). Use an if-else-if template for the main algorithm. Perform the repetition using the goto instruction. It is easier to use the switch template for printing the names of the week.

5.13. The recursive equation for defining the n -th term of the sequence is as follows.

$$f(n) = \begin{cases} 1, & \text{if } n = 1 \text{ or } 2, \\ f(n-1) + f(n-2), & \text{otherwise.} \end{cases}$$
$$f(n) = \begin{cases} 1, & \text{if } n = 1 \text{ or } 2, \\ f(n-1) + f(n-2), & \text{otherwise.} \end{cases}$$

5.14. This is similar to the Fibonacci sequence (refer to the previous exercise) except that here we are dealing with three cases.

5.15. The recursive function is as follows:

$$P(n, x) = \begin{cases} 1, & \text{if } n = 0, \\ x, & \text{if } n = 1, \\ \frac{2n-1}{n} P(n-1, x) - \frac{n-1}{n} P(n-2, x), & \text{otherwise.} \end{cases}$$
$$P(n, x) = \begin{cases} 1, & \text{if } n = 0, \\ x, & \text{if } n = 1, \\ \frac{2n-1}{n} P(n-1, x) - \frac{n-1}{n} P(n-2, x), & \text{otherwise.} \end{cases}$$

5.16. A recursive function for power is defined as below:

$$\text{Power}(x, n) = \begin{cases} 1, & \text{if } n = 0, \\ x \text{ Power}(x, n - 1), & \text{if } n > 0, \\ \frac{1}{x} \text{ Power}(x, n + 1), & \text{otherwise.} \end{cases}$$

$$\text{Power}(x, n) = \begin{cases} 1, & \text{if } n = 0, \\ x \text{ Power}(x, n - 1), & \text{if } n > 0, \\ \frac{1}{x} \text{ Power}(x, n + 1), & \text{otherwise.} \end{cases}$$

In the main algorithm, print the message using an if-else template, if the undefined case happens; otherwise, call the method.

5.17. Use the same way as [Exercise 5.15](#).

6.4. You need to calculate the series $\sum_{i=1}^n 2i - 1$.

6.5. You need to calculate the series $\sum_{i=1}^n m$.

6.6. Take the variable *sum* with the initial value of 0. In the range of a for template with the specification *i=1,n*, replace *sum* by its addition to *i* if *i* is a multiple of 4. An integer *i* is a multiplication of 4 if the remainder of division *i / 4* is zero. Another way is to

calculate the series $\sum_{i=1}^{n/4} 4i$.

6.7. Similar to the first way in the hint of [Exercise 6.6](#).

6.8. Take the variables *n* and *g* for the ID number and grade. In addition, take the counter variable *c*. Read *n* and *g* in the range of a

for template. Then, increase c by 1 along with printing n and g if $g < 12$. Finally, Print c after exiting the loop.

6.9. You need to calculate the series $\sum_{t=1}^{60} \sqrt{h^2 + (vt)^2}$.

6.10. In the range of a for template with the specification $i=1,300$, first read n , h , and s , and then print $200s + 1.5(h - 200)s$ if $h > 200$; otherwise, print hs as salary.

6.11. Take the variables nP , nZ , and nN for the number of positive, zero, and negative numbers, respectively. Further, take x for the read numbers as well as $sumP$ and $sumN$ for the sum of positive and negative numbers, respectively. First read x in the range of a for template with the specification $i=1,50$. Then, using an if-else-is template, print it with an appropriate heading if $x > 0$ and increase $sumP$ and nP by x and 1, respectively; Otherwise, if $x < 0$, print it with an appropriate heading, increase $sumN$ and nN by x and 1, respectively; otherwise, increase nZ by 1. Eventually, print the required quantities with appropriate headings after exiting the loop.

6.12. Take the variables x , max , and min for the read number, maximum and minimum, respectively. Read the first number and assign it for both max and min . Then, in the range of a for template with the specification $i=2,n$, first read x . Next, substitute x for max if $x > max$ and substitute x for min if $x < min$. Note that these two if templates are successive, not nested. After exiting the loop, max and min are the maximum and the minimum of all the numbers.

6.13. In the range of a for template with the specification $i=2,n-1$, print i if it is a prime number.

6.14. In the range of a for template with the specification $i=5,999,4$, print i if it is prime. Or, in the range of a for template with the specification $i=1,249$, print $4i + 1$ if it is prime.

6.15. Use the logic of [Example 6.4](#).

For [Exercises 6.16](#) to [6.24](#) we hint for the case where we do not use the necessary user-defined functions. Using these functions makes the algorithms easier.

6.16. Use the idea of [Example 6.13](#).

6.17. The main body of [Flowchart 6.2\(c\)](#) calculates the factorial of n . However, the main body of [Flowchart 6.8\(a\)](#) computes x^n if the absolute value is discarded. You should compose these two flowcharts. First, apply the former flowchart replacing the loop specification by $k=1,i$ and then apply the latter flowchart replacing the loop specification and x by $k=1,f$ and i , respectively. What p carries after leaving the latter flowchart is $i^{i!}$. Consider this as the common term of the required series and complete it using a for loop with the specification $i=1,5$.

6.18. Take the variables *sum* and *sign* for the repetitive sum and providing the signs, respectively. Assign 0 and -1 for *sum* and *sign*, respectively. In the range of a for template with the specification $t=1,n,2$, first place the processes for calculating the common term $i^{i!}$ in the hint of [Exercise 6.17](#). Then, change the sign by

substituting $-sign$ for $sign$. Finally, substitute the addition of sum and p for sum .

6.19. Fix the former flowchart in the hint of [Exercise 6.17](#) and change the latter one by interchanging the roles of i and f .

6.20. See the hints of [Exercises 6.18](#) and [6.19](#).

6.21. As mentioned in the hint of [Exercise 6.17](#), the main body of [Flowchart 6.8\(a\)](#) calculates x^n discarding the absolute value of n . In this flowchart, replace the loop specification by $k=1,i$. Now change the sign by substituting $-sign$ for $sign$ and take $pi^2 sign$ as the common term of the given series using a for template with the specification $i=1,7$. Bear in mind to assign -1 to $sign$ as the initial value before the outer loop.

6.22. Create the common term of the double series as follows. Take the variable $p1$ with the initial value of 0. Substitute the multiplication of $p1$ by i for $p1$ in the range of a for template with the specification $k=1,n$. Take another variable $p2$ with the initial value of 0. Then, substitute the multiplication of $p2$ by i for $p2$ in the range of another for template with the same specification. The common term is $p1 + p2$.

6.23. Calculate two series $\sum_{i=1}^{15} i^n$ and $\sum_{i=1}^{15} i$ in a single for template with the specification $i=1,15$ with two repetitive sums, namely, $sum1$ and $sum2$ for which the initial value of 0 are assigned before the loop. The common terms of $sum1$ and $sum2$ are the $p1$ in the hint of [Exercise 6.22](#) and i , respectively.

After exiting the loop, calculate $(sum2)^n$ using **Flowchart 6.8(a)** discarding the absolute value of n . It suffices to replace x by $sum2$.

6.24. We focus on i^{j^k} since the other terms are known. Write `pow(i,pow(j,k))` to use the library function `pow()`. Then, take the variable p with the initial value of 1 in order to be a common term of a series. Substitute the multiplication of p by j for p in the range of a for template with the specification `t=1,k`. Here, p carries j^k with itself. Now, take the variable q with the initial value of 1 and substitute the multiplication of q by i for q in the range of a for template with the specification `t=1,p`. At this point q carries i^{j^k} .

6.25. The same tasks as the trapezoid method are performed with only one replacement in the range of the for loop in the series of function `Trap()` as follows. Substitute the addition of sum and $4f(a + ih)$ for sum if i is odd (the remainder of $i / 2$ is 1); otherwise, substitute the addition of sum and $2f(a + ih)$ for sum . Another way is to substitute the addition of sum and $4f(a + 2ih) + 2f(a + (2i + 1)h)$ for sum in the range of a for template with the specification `i=1,n/2-1`.

6.26. Substitute $x + 1/x$ for x in the range of a for template with the specification `i=1,n`.

6.27. Take the series variable *sum* with the initial value of $1/x$. Substitute the inverse of $i + 1/x$ for x in the range of a for template with the specification $i=1,n-1$ and then take it as the common term of the series.

6.28. First work with positive signs. As shown, the denominator of the k -th term is the series $s = \sum_{i=1}^k ix^i$. $S = \sum_{i=1}^k ix^i$. Now, $1/s$ is the common term of the required 10-term series. Apply the positive and negative signs using a sign maker variable.

6.29. Examine the divisibility of n by i in the range of a for template with the specification $i=1,n/2$.

6.30. For the function, take the variable *sum* with the initial value of 0. In each repetition of the for loop in the algorithm of the previous exercise, instead of printing the divisor, add it to *sum* and substitute the result for *sum*. After exiting the loop, return 1 if $sum = n$; otherwise return 0.

6.1*. We give hints for the odd numbers; for even numbers, it suffices to reverse the specification of the outer loop.

1. In the range of two nested for loops with the specification $i=1,n$ and $j=1,i$, respectively, print “* _”. After exiting the inner loop, break the line. This is performed by `cout<<endl` and `System.out.println()` statements in C++ and Java codes, respectively.
3. In item 1, before the inner loop, provide the necessary space using the following statements:

```
cout<<setw(2*(n-i+1));
```

Do the following tasks in the range of a for loop with the specification $i=1,n$ in Java due to the different nature of the formats in C++ and Java:

- I. Print a whitespace character using the statement `System.out.print(" ")` in the range of a for loops with the specification $j=1,2*(n-i)$;
 - II. Print an asterisk followed by a whitespace character with the `System.out.print("* ")` statement in the range of a for loops with the specification $k=1,i$;
 - III. Break the line using the `System.out.println()` statement.
5. In items 3 as well as 3I, remove the coefficients 2.

6.2*. Use [Example 6.7](#).

6.3*. Use the hint of part 5 in [Exercise 6.1*](#).

6.4*. First, determine the number of digits of n using the function `digits()` of Exercise 5.1 and then assign it to k . Now, assign the remainder of the quotient of $n/10^{k-1-i}$ by 10^i to an integer variable, say t , in the range of a for template with the specification $i=k-1,0,-1$ and then print it.

6.5*. In the range of the for template mentioned in the hint of [Exercise 6.4*](#), print t if $i = 3$ or 2 and t is even; or, $i = 1$ or 0 and t is odd. Count the number of such integers in this path.

6.6*. In the range of a for template with the specification $i=2,[n/2]$, assign i to an integer variable, say k , if it is prime as well as a divisor of n . Here, $[]$ stands for the least integer closer to the number. Then, use the function `prim()` in [Example 6.10](#) in order to distinguish the primality. The integer i is a divisor of n if the

remainder of n by i is zero. After exiting the loop, k is the largest prime divisor of n .

6.7*. Use the hints of [Exercises 6.29](#) and [6.30](#).

6.8*. You have to calculate two multiplicative series

$$P1 = \prod_{i=1}^r p_i, \quad P2 = \prod_{i=1}^r (1 - p_i)$$

$$P1 = \prod_{i=1}^r p_i, \quad P2 = \prod_{i=1}^r (1 - p_i)$$

in the same for loop with the specification $i=1,r$ inside which, in the range of an inner for loop with the specification $j=1,\sqrt{n}$, apply j as the common term of the first series and $1 - j$ as the common term of the second series if j is a prime divisor of n . To distinguish whether or not j is the prime divisor, refer to the hint of [Exercise 6.6*](#).

6.9*. Use the hint of [Exercise 6.13](#).

7.1. Take the counting variable i , summing variable sum with the initial value of 0, and the sign making variable $sign$ with the initial value of -1 . In the range of a while template, apply the following instructions while $i < 19$ (the denominator of the tenth term):

1. Change the mark of $sign$;
2. Add $sign \times 4 / i$ to sum and substitute the result for sum ;
3. Increase i by 2.

Print sum after exiting the loop. You can solve this exercise using the for templates. It suffices to put the first two above-mentioned

instructions in the range of a for template with the specification $i=1,19,2$.

7.2. Use the tool of tolerance quoted before **Example 7.10**. This time use a do-while template. Do the three instructions in the hint of **Exercise 7.1** while $4 / i < 0.00005$. Note the absolute value of two consecutive approximations is $4 / i$. If you have solved **Exercise 7.1** using a for template, select a large number instead of the initial value of 19 of the loop. At the end of the range, terminate the algorithm by printing *sum* if $4 / i < 0.00005$.

7.3. Use the hint of **Exercise 7.1**. Here, the increment of i is by 1 and the common term in each repetition is

$$\frac{1}{i} \left(\frac{x-1}{x} \right)^i \cdot i \left(\frac{x-1}{x} \right)^i.$$

Use the library function `pow()` for powering. Note that the tenth term is the above term for $i = 10$.

7.4. Use the hint of **Exercise 7.2**.

7.5. Take the counter variable q with the initial value of 0 for the quotient. In a while loop, perform the following instructions in each repetition while $m \geq n$:

1. Subtract n from m and substitute the result for m .
2. Increase q by one unit.

After exiting the loop, q and m are the quotient and remainder, respectively.

7.6. To generate the terms of this sequence, we observe that after the third term, each term is equal to the double of the previous term subtracted by the term before the previous one, plus 1. Now use **Example 7.3** replacing the substitution of $a + b$ for a , by $2a - b$ for a . Furthermore, you should change the initial values of a and b . From the viewpoint of recursive functions,

$$f_n = 2f_{n-1} - f_{n-2} + 1. \quad f_n = 2f_{n-1} - f_{n-2} + 1.$$

7.7. Considering the hint of **Exercise 7.6**, use the logic of **Flowchart 7.3(e)**.

7.8. In **Flowchart 7.3(c)** replace 50 by 150 and print a if $a > 100$.

7.9. In **Flowchart 7.3(c)** replace 50 by 100. Then, taking two variables c and sum with initial values of 0, increase sum by a and c by 1 instead of printing a .

7.10. In **Flowchart 7.3(c)**, first take the variable c with initial value of 0. Then, increase c by 1 after printing a . Finally, replace the loop condition by $c < 20$.

7.11. Take the variables r and sum , with the initial values of 0, for the remainder and sum of digits which are returned to the call unit. Then, perform the following instructions in the range of a do-while loop:

1. Assign the remainder of the division n by 10 to r ;
2. Increase sum by r ;
3. Substitute $n / 10$ for n .

Repeat the above instructions while $n > 0$.

7.12. Use the function `SumDig()` of **Exercise 7.11**.

7.13. Use the function SumDig() of [Exercise 7.11](#).

7.14. Take the variable u for the number which is read each time. After reading n , read u inside a for loop with the specification $i=1,n$ and print the reverse of u using the function rev() of [Example 6.3](#).

7.16. In the range of a for loop with the specification $i=1001,9999$, print and count the symmetric integers using the function rev() of [Example 6.3](#).

7.17. Take the variable d for the decimal part of t . Then, d is equal to t subtracted by the integer part of t . Now, in the range of a do-while loop, substitute the multiplication of d by 10 and repeat this while $d \geq 1$. After exiting the loop, first store d in an integer variable, say k , and then return it.

7.18. Use the sub-algorithm DecPart() in [Exercise 7.17](#) to receive the decimal part of t as an integer by removing the decimal point and name it as m in the main algorithm. Moreover, name the integer part of t as n . Calculate the number of digits of m and n and name them as nD and mD , respectively. Now, print first a minus sign if t is negative (use the built-in System.out.print() method to do this in Java). Next, print consecutively the reverse of m in mD columns, the dot character, and the reverse of n in nD columns using the function rev() of [Example 7.4](#). Use appropriate formats in C++ and Java to print the above-mentioned items.

7.19. Take the counter integer c . Substitute the multiplication of t by 10 for t and increase c by 1 in the range of a do-while template. Repeat this range while the remainder of the integer part of t by 10 is 0, that is, while the decimal point moves to the rightmost position. Upon exiting the loop, assign t in an integer variable, say

k . Take the number of digits of k subtract by c as u . Now the real value of the division of k by 10^u 10^u is the required number.

7.20. The function `gcd()` in [Example 7.7](#) is used in this algorithm. After reading n , first read an integer and assign it to an integer variable, say g . Now, in the range of a for loop with the specification $i=2,n$, read an integer, say s , each time and substitute `gcd(g, s)` for g . The value of g is the required number after exiting the loop.

7.21. Use the hint of [Exercise 7.20](#).

7.22. In the range of two nested for loops with the specification $i=1,n$ and $j=i+1,n$, respectively, count the pairs (i, j) which are coprime using the function `gcd()` in [Example 7.7](#). The requested probability follows from the division of this count by $n(n+1)/2$.

7.23. Take the integer variables c (for counting) and k with the initial values of 0 and of 1, respectively. In the range of a do-while loop, first, increase k by 1 to generate an integer. Then, if k is prime, print it and increase the counter c by 1. Repeat these instructions while $c < 20$. Finally, use the function `prime()` to check the primality.

7.24. In the hint of [Exercise 7.23](#), take the printing of k out of the loop.

7.25. Appealing to the function `perfect()` in [Exercise 6.26](#), use the hint of [Exercise 7.23](#).

7.26. Using the function `perfect()` in [Exercise 6.26](#), use the hint of [Exercise 7.24](#).

7.27. In the range of a do-while loop, create the factorial of the consecutive numbers k and compare $k!$ with n , while $k < n$.

7.28. In the range of a for loop with the specification $i=1,n-1$, print i and add 1 to the counter of this number if $\text{gcd}(i, n) = 1$.

7.29. Combine **Flowcharts 7.7(b)** and **7.8(b)**, or, **Flowcharts 7.7(b)** and **7.8(b)**.

7.30. Use the hint of **Exercise 7.2**. Additionally, you can use the recursive method. The return point is the estimated root of x_0 if the recursive methods is employed.

7.30. There are easy flowcharts and codes. It suffices to use the related hints as all or part of the range.

7.1*. In the divisible case, examine i for being the answer (i is the answer if $ai = c$) in the range of a for template with the specification $i=0,m-1$.

7.2*. If $\text{gcd}(m, n) = 1$, examine i and j for being the answers of the first and second congruent equations in the range of two nested for templates with the specification $i=0,m-1$ and $j=0,n-1$, respectively.

7.3*. The trivial way is to calculate the remainder of a^k [a^k by m which is time consuming for the large values of k and may cause the memory overlap error. The other way is to take a variable p . Then, in the range of a for template with the specification $i=0,m-1$, substitute the remainder of pa by m for p (compare the resulted flowchart with **Flowchart 6.8(a)**). In this way, we do not have any memory overlap error, however, it is still time-consuming.

7.4*. In the range of a for loop with the specification $k=1,n$, if a^k modulo n is equal to 1, terminate the sub-algorithm by returning k using the function `akmodm()` of **Exercise 7.3***. In addition, you can write this sub-algorithm using a do-while template with the same range.

7.5*. In this sub-algorithm, use the idea that the reverse of a number is printed starting from the rightmost digit backward. The contents of the sub-algorithm for the receiving number n is as follows. Using an if template print the remainder of n by 10 if it is not zero (use the `System.out.print()` method in Java). Then, call the function for the quotient of n by 10; otherwise return to the call unit.

7.6*. By assigning the initial values of 0 and 1 to the variables sum and p , respectively, apply the following instructions in the range of a do-while loop:

1. Assign the remainder of the division n by 10 to r ;
2. Add the r times p to sum and substitute the result for sum ;
3. Substitute $2p$ for p ;
4. Substitute the quotient of n by 10 for n .

Repeat this range while n is equal to zero (no number is left). Then, return sum after exiting the loop.

7.7*. First, calculate the number of the digits of n using **Exercise 4.1** and then assign it to the variable d . Further, assign the quotient of $d / 3$ to the variable q . Now, print the quotients of n by 1000^q in the range of a for template with the specification $i=q,0,-1$ and then, substitute the remainder of n by 1000^q **1000^q**

for n . To print the comma separator use an if template so that the item "," is printed if $q \neq 0$. The `System.out.print()` statement should be used in the Java codes.

7.8*. In the hint of **Exercise 7.7***, instead of printing each group, convert that group to the base-10 numeral system using the function `dec()` in **Exercise 7.6*** and print it in its position. The `System.out.print()` statement should be used in the Java codes.

7.9*. First, calculate the number of digits of n and assign it to the variable d . Further, assign the quotients of n by 10^{d-1} to the variable p in the range of a for template with the specification `i=d,1,-1` and then replace n by the remainder of n by 10^{d-1} . Furthermore, convert p to the base-2 numeral system using the function `bin1()` in **Example 7.6** and assign the result to q . Now, it is time to print. In Java, use the format `"%03d"` followed by q as the print arguments in order to prevent losing zeroes when q has less than three digits. However, in C++, first, calculate the number of digits of q and assign it to the variable e . Then, using an if-else-if template or a switch template,

- print 00 followed by q if $e = 1$;
- print 0 followed by q if $e = 2$;
- otherwise, print q itself

8.1. Use the function `Fib()` in **Example 7.3**.

8.2. A trivial way is to use the hint of **Exercise 8.1**. Further, another way is by a do-while template using the idea of the function `Fibo()` in **Example 7.3**. The required sub-algorithm `FibArray()` includes two arguments including the array F and the number n of its entries. First, assign the initial value of 1 for the variables a and b and 0 for

the counter variable i . Then, apply the following instructions in the range of a do-while loop:

1. Assign a to F_i ; F_i ;
2. Increase i by 1;
3. Substitute $a + b$ for a ;
4. Swap a and b .

Repeat these instructions while $i < 20$.

8.3. In the range of a for template with the specification $i=1,n$, terminate the sub-algorithm by returning i as soon as t matches with an entry. Exiting the loop implies that there is no matching and thus 0 should be returned.

8.4. Take the counting variable c with the initial value of 0. In the range of a for template with the specification $i=1,40$, print N_i N_i and G_i G_i and increase c by 1 if $g = G_i$. $g = G_i$. After exiting the loop, print the message if $c = 0$; otherwise print c .

8.5. Follow **Flowchart 8.8(b)** until the exit from the loop. Now, use Algorithms 8.11(b) or 8.11(c) for (b, n) instead of $(a, 40)$. For the second part, you may call the sub-algorithm Sort() in **Exercise 8.12**

8.6. Take the index maker variable m with the initial value of 0. In the range of a for template with the specification $i=1,20$, if a_i a_i is not in the array b , and if a_i a_i is not in the m -entry array c , make a new index m (increase m by 1) and assign a_i a_i for c_m c_m using the search() function in **Example 8.7**.

In the above discussion, you can use two successive if templates or merge them using the && operator. It is worth mentioning that

in the second search (searching a_i in c_m), the inconsistency in the loop of the search() function causes the substitution of a_1 for c_1 .

8.7. Take an index maker variable k . First put A in C : Substitute A_i for C_i in the range of a for template with the specification $i=1,m$. Now, in the range of a for template with the specification $i=1,n$, if B_i is not in the array A , make a new index (increase k by 1) and put B_i in C_k using the search() function in **Example 8.7**. After exiting the loop, print the $(m + k)$ -entry array C . Declare A and B with lengths 101 and C with length 201.

8.8. Note that in each of the sorting Flowcharts 7.11(b) or 7.11(c) in ascending order, changing the inequality ' $>$ ' to ' $<$ ' turns it to descending order. Therefore, you can sort C simultaneously in both orders. It suffices to use the same nested loops and the successive if templates.

8.9. Take the counter variable c with the initial value of 0. In the range of a for template with the specification $i=1,n$, increase c by 1 if $x = a_i$.
 $x = a_i$.

8.10. In the range of a for templates with the specification $i=1,20$, if the number of repetitions of a_i in the array a is greater than 1, print the required three items a_i, i, a_i, i , and the number of repetitions of a_i in a and then terminate the algorithm using the function repeat() in Exercise 8.9. Put the message printing after exiting the loop.

8.11. Using the repeat() function in Exercise 8.9, create another array named t such that t_i , for $1 \leq i \leq n$, is the number of repetitions of u_i in the array u . Then, calculate the maximum entry of the array t and assign it in a variable, say e , using the function max() in Example 8.5. Finally in the range of a for template with the specification $j=n,1,-1$, if $t_j = e$, print u_j .

8.12. The array is returned to the call unit with the same name as it enters the sub-algorithm. In Flowchart 8.11(b), remove the middle part including the basic process of sorting and place the instruction of calling the sub-algorithm Sort() instead. Now put the removed part as the basic process of the sub-algorithm Sort() changing 39 to $n - 1$.
 $n - 1$.

8.13. First, it is worth mentioning that in each of the sorting Flowcharts 7.11(b) or 7.11(c) in ascending order, changing the inequality '>' to '<' turns it to descending order. Take one of the above-mentioned flowcharts and replace 39 by 19 in its main part. Then, after the if template in the range of the inner loop use another if template: if $a_{20+i} < a_{21+i}$, $a_{20+i} < a_{21+i}$, then swap them. After exiting the nested loops, print the first half of the array using the sub-algorithm writeIvec() and the second half directly from index 21 onwards.

8.14. First, combine the arrays a and b in the array c . To do this, in the range of a for template with the specification $i=1,20$, substitute a_i and b_i for c_i and c_{20+i} , respectively. Now use Algorithm 8.11(b) or Sub-algorithm Sort() in Exercise 8.12 to sort c .

8.15. In Flowchart 8.12(b), remove the middle part including the basic process of ranking and place the instruction of calling the sub-algorithm Rank() instead. Now, put the removed part as the basic process of the sub-algorithm Rank() changing 39 and 41 to $n - 1$ and $n + 1$, respectively.

Print the first, the second, and the third lists from N_1 to N_{40} , G_{40} to G_1 , and R_1 to R_{40} with appropriate headings in each case, respectively.

8.16. Use the logic of Algorithm 7.13(c) with the exception of replacing 3 by $m + 1$.

8.17 Substitute a_i for a_{i+7} in the range of a for loop with the specification $i=40,6,-1$. After exiting the loop, read one

number and assign it to a_{i+5} in the range of another for template with the specification $i=1,7$. Declare the array a with the length of 48.

8.18. Apply the following instructions in the range of a for template with the specification $k=1,q$:

1. Read t and m ;
2. In the range of a for template with the specification $i=39+k,m+1,-1$, substitute a_i for a_{i+1} ; a_i for a_{i+1} ;
3. Substitute t for a_{m+1} . a_{m+1} .

8.19. First, read the 20-entry ascending array a . Now, use the main body of **Flowchart 8.15** and change the specification of the outer loop to $i=21,20+n$. Finally, print the $(20 + n)$ -entry real array a using the sub-algorithm `writeFvec()` instead of printing the 20-entry integer array a using the sub-algorithm `writeIvec()`.

8.20. Pay attention that in this exercise, the entries of the array should be read one by one in the sub-algorithm itself.

8.21. Assign the initial value of -1 to the index maker variable k . In the sub-algorithm of **Figure 7.6(b)** remove the instructions concerned with the variables new and p and change the range of the do-while template to the following:

1. Make a new index (increase k by 1);
2. Assign the remainder of n by 2 to e_j ;
3. Substitute the quotient of n by 2 for n .

Now reverse the position of entries of the k -entry array e using the sub-algorithm `revive()` in **Example 8.3**.

8.22. Use the hint of [Exercise 8.21](#) with replacing 2 by 16. The final array b should be converted to the string data, however, we perform this, along with the printing processes. To do this, in the main algorithm, first, receive the array b from the sub-algorithm. Then, use a switch (or if-else-if) template in the range of a for template with the specification $i=1,k$ such that in the cases $b_i = 10, 11, 12, 13, 14, 15$, $b_i = 10, 11, 12, 13, 14, 15$, print the characters 'A', 'B', 'C', 'D', 'E', 'F', respectively, and print b_i itself in the default case.

8.23. Assign a_1 to a variable, say g . Then, using the function $\text{gcd}()$ in [Example 7.7](#), each time substitute $\text{gcd}(g, a_i)$ for g in the range of a for template with the specification $i=2,20$. The value of g is the required number after exiting the loop.

8.1*. Take the integer variables k and m . Then, perform the following instructions in the range of a do-while template:

1. Print the pattern mentioned in the exercise;
2. Read k ;
3. If k is 1, sort the array N in ascending order such that the entries of the arrays G and R with the same indices are swapped in the swapping step in either [Flowcharts 8.11\(b\)](#) or [8.11\(c\)](#);
4. Otherwise, do the same tasks as in item 3 for the arrays G , N , and R , respectively, if k is 2. Note that the sorting is in descending order here;
5. Otherwise, do the same works as in item 3 for the arrays R , N , and G , respectively, if k is 3;
6. Otherwise, read m and in the range of a for template with the specification $i=1,40$, if k is 4, print N_i , G_i , and R_i

N_i , G_i , and R_i if $m = N_i$ and then exit the loop using the break statement. Finally, print the message Not in list after exiting the loop;

7. Otherwise, print the message Wrong number.

Repeat the above instructions while $k \neq 0$. You may use either an if-else-if template or a switch template for the instructions 3 to 7.

8.2*. Use the hint of **Exercise 8.1***. Use the ideas in Example 8,1 for items 1 and 2. For item 3, first find the maximum grade using the function max() in **Example 8.5**. Item 4 uses the same way as above, using the function min() in **Exercise 8.5.1** in the text.

For item 5, first calculate the average and name it as M. Second, take a counter variable, say c, with the initial value of 0. Third, in the range of a for template with the specification i=1,40, increase c by 1 if $M = G_i$. Finally, print c after exiting the loop. Items 6 and 7 have the same manner.

For item 8, first, create the array t as mentioned in the hint of **Exercise 8.11**. Then, find the maximum of the array t and name it as x using the function max() in **Example 8.5**. Now, similar to item 5, count the number of the entries N_i with $x = N_i$.

8.3*. Read the thirty numbers as a real array, say a. Perform the following instructions in the range of a for template with the specification i=1,29:

1. Continue the loop if $a_i = a_{i+1}$;
2. Otherwise, if $a_i < a_{i+1}$, print the message Not sorted in the range of a for template with the specification j=i+1,29 and then terminate the algorithm if $a_j > a_{j+1}$. Next, print the message Increasing array after exiting this loop.

3. Otherwise, print the message Not sorted in the range of a for template with the specification $j=i+1,29$ and then terminate the algorithm if $a_j < a_{j+1}$. $a_j < a_{j+1}$. Finally, print the message Decreasing array after exiting this loop.

8.4*. Take the variable n for the number of entries of the array a with the initial value of 40 and the variable i for counting with the initial value 0. Perform the following instructions in the range of a do-while template:

1. Increase i by 1;
2. Do the following if $x = a_i$: $x = a_i$:
 - 2.1. Substitute a_{j+1} for a_j a_{j+1} for a_j in the range of a for template with the specification $j=i,n-1$;
 - 2.2. Decrease n by 1;
 - 2.3. Decrease i by 1;

Repeat the above instructions while $i < n$. Finally, print the n -entry deformed array a after exiting the do-while loop.

8.5*. Take the variable n for the number of entries of the array a with the initial value of 40 and the variables k and i for counting with the initial value of 0 for k . Use two nested do-while loops as follows. Perform the following in the range of the outer do-while loop:

1. Increase k by 1;
2. Assign k to i ;
3. Perform the following instructions in the range of the inner do-while loop:
 - 3.1. Increase i by 1;
 - 3.2. Do the following if $a_k = a_i$: $a_k = a_i$:

- 3.2.1. Substitute a_{j+1} for a_j a_{j+1} for a_j in the range of a for template with the specification $j=i, n-1$;
- 3.2.2. Decrease n by 1;
- 3.2.3. Decrease i by 1;

Repeat the inner do-while loop (from 3.1) while $i < n$.

Repeat the outer do-while loop (from 1) while $k < n$.

Print the n -entry deformed array a after exiting the outer do-while loop.

8.6*. Create two integer arrays T and N for the repeated entries and the number of their repetitions, respectively. Take an index maker variable k . In the range of a for template with the specification $i=1, 40$, calculate the repetition of a_i a_i and assign it to a variable, say r , using the function `repeat()` in **Exercise 8.9**. Then, make a new index (increase k by 1) and store a_i in T_k a_i in T_k if $r > 1$ and a_i a_i has not already stored in the k -entry array T . To check whether or not a_i has already been stored in T , use the function `search()` in **Example 8.7** (see **Example 8.9**). Now, print T_i and N_i T_i and N_i in the range of a for template with the specification $i=1, k$.

8.7*. Take an index maker variable k . Modify **Flowchart 7.9** as follows. Remove the single print before the for loop. After exiting the while loop, make a new index (increase k by 1) if $c > 0$ and then store i and c in p_k and r_k , p_k and r_k , respectively. This is the k -entry arrays p and r which are returned.

8.8*. Using the sub-algorithm `prim_dec()` in **Exercise 8.7***, store the prime factors and their multiplicities of the primary decomposition of m in the km -entry arrays pm and rm , respectively, while storing those of n in the kn -entry arrays pn and rn , respectively. Take the variables

gcd and lcm with the initial values of 1. Furthermore, take the variable Rm and Rn . Now, do the following in the range of a for template with the specification $i=2, \max(m,n)$:

1. Assign 0 to Rm ;
2. In the range of a for template with the specification $j=1, km$, if $i = pm_j$, $i = pm_j$, assign rm_j rm_j to Rm ;
3. After exiting the recent for loop assign 0 to Rn ;
4. In the range of a for template with the specification $j=1, kn$, assign rn_j rn_j to Rn if $i = pn_j$; $i = pn_j$;
5. After exiting the recent for loop, multiply gcd by i to the power of minimum of Rm and Rn and substitute the result for gcd ;
6. Multiply lcm by i to the power of maximum of Rm and Rn and substitute the result for lcm .

Finally, print gcd and lcm after exiting the outer for loop.

8.9*. Using an if-else template, return 1 if $n = 1$; otherwise, do the following:

1. Call the sub-algorithm `prim_dec()` of **Exercise 8.7*** for n , and receive the k -entry arrays p and r . Note that all entries of r are positive;
2. In the range of a for template with the specification $i=1, k$, the second case for the function happens if $r_i > 1$, $r_i > 1$, thus, terminate the sub-algorithm by returning 0;
3. The natural exit of the for loop occurs when all the entries of r are 1. In this circumstance, return -1 to the power of k , or, return 1 instead if k is even (the remainder of k by 2 is zero) using an if-else template; otherwise, return -1.

8.10*. Take the two arrays f and g and declare them with a large lengths, say 100 (or larger if needed). Assign zero for the above arrays as follows. Assign 0 to both f_i and g_i in the range of a for template with the specification $i=1,100$. Now, read m and n and then read the coefficients of f and g : read f_i in the range of a for template with the specification $i=0,m$ and read g_i in the range of another for template with the specification $i=0,n$.

Denote the function $f + g$ by h and let k be its degree which is the maximum of m and n . Define the coefficients of h as follows. Assign $f_i + g_i$ to h_i in the range of a for template with the specification $i=0,k$. Determine the first nonzero coefficient of h as follows. In the range of a for template with the specification $i=0,k$, assign i to a variable, say u if $h_i \neq 0$. Now it is time to print the following:

1. Print the string $f(x)+g(x)=$;
2. Print 0 if $u = k$ ($h = 0$);
3. Print h_0 and increase u by 1 if $u=0$ (the constant coefficient). Be careful not to break the line after printing!
4. Print the following pattern in the range of a for template with the specification $i=u,k$ if $h_i \neq 0$: $h_i \neq 0$:

in C++ codes:

```
cout<<"+"<<h[i]<<"x^"<<i;
```

In Java codes:

```
System.out.print("+" + h[i] + "x^" + i)
```

8.11*. Use the hint of **Exercise 8.10*** with the following changes.

1. Assign $m + n$ (not the maximum of them) to the degree k of h .
2. To define the coefficients of h , in the range of a for template with the specification $i=0,k$, first calculate the series $\sum_{s=0}^i f_s g_{i-s}$

$\sum_{s=0}^i f_s g_{i-s}$ as *sum* (Section 6.2). Then, assign *sum* for h_i .

3. Print the string $f(x)*g(x)=$ in the printing item 1.

8.12*. Receive the k -entry array e calling the sub-algorithm Bin() in **Exercise 8.21** for n . That is to say that,

$$n = \sum_{i=0}^k e_i 2^i.$$

Take 1 as the initial value of b . Take another variable p with the initial value of 1. In the range of a for template with the specification $i=0,k$, do the following:

1. Substitute $a^{pe_i} a^{pe_i}$ for a ;
2. Substitute the remainder of $b \times a$ by m for b ;
3. Substitute $2p$ to p .

After exiting the loop, b is the required number to print.

8.13*. First, calculate the number of digits of n using **Exercise 4.10** and assign it to a variable, say d . Then, store the digits of n from left to right in a d -entry array named D . To do this, store the quotients of n by 10^{d-1} in the entry D_{d+1-i} in the range of a for template with the specification $i=d,1,-1$ and then replace n by the remainder of n by 10^{d-1} . Now, the required format for the output can be performed using the appropriate formats in both C++ and Java.

9.2. Use the swap algorithm in [Example 5.4](#) to swap the entries A_{ij} and A_{ji} inside two nested for templates. Pay attention to the final values of the loop (refer to Algorithm 7.3).

9.3. Compare the corresponding entries inside two nested for loops and return 0 as soon as an inequality occurs. Moreover, return 1 after the (natural) exit from the outer loop.

9.4. The entry A_{ij} is on the secondary diagonal if $i + j = 6$. Therefore, using an if template, if $i = j$ or $i + j = 6$ then, assign 1 to A_{ij} inside two nested for templates; otherwise, assign 0 to A_{ij} .

9.5. In a for template with the specification $i=1,7$, swap the (i, i) - and $(i, 8-i)$ -entries using the swap algorithm.

9.6. In a for template with the specification $j=1,m$, swap the (r, j) - and (s, j) -entries of A by the swap algorithm.

9.8. In a for template with the specification $j=1,m$, apply the following substitution.

$$A_{rj} \leftarrow A_{rj} + zA_{tj} \quad A_{rj} \leftarrow A_{rj} + zA_{tj}.$$

9.10. In a for template with the specification $j=1,m$, apply the following substitution.

$$A_{rj} \leftarrow zA_{rj} \quad A_{rj} \leftarrow zA_{rj}$$

9.12. First, define E as a 10×10 matrix and, initialize zero to all of its entries using two nested for loops. Then, perform the following three

instructions respectively in the range of two nested for loops with the variables i and j :

1. Assign 1 to A_{ij} ; A_{ij} ;
2. Print A using the sub-algorithm `writeMat()`;
3. Assign 0 to A_{ij} . A_{ij} .

9.13. The algorithm of this exercise is similar to **Flowchart 9.8**. It suffices to change the following two instructions in Case 1:

- Remove the instruction which substitutes $-d$ to d ;
- Transfer the implementation control to the beginning of the leading for loop which is supposed to have the label ‘a’ instead of the instruction which returns zero. This can be done by `goto a` and `continue a` statements in C++ and Java codes, respectively.

9.15. Use a similar logic as in the `search()` function of **Example 8.7**. Here two nested for templates should be used. The main algorithm can be achieved using a for template.

9.16. Use the function `searchFmat()` of the previous exercise.

9.17. First, assign the coefficients matrix A in another matrix, say B . Then, apply the following instructions in the range of a for template with the specification `j=1,10`.

1. Swap the vector y and the i -th column of A . To do this, swap the entries y_i and A_{ij} y_i and A_{ij} using a for template with the specification `i=1,10`;
2. Divide $\det(B)$ by $\det(A)$;
3. Print it as x_j . x_j .

9.18. Use the idea in Programs P9_9.

9.1*. For parts 1, 2, and 3, use the hints of the parts 1, 5, and 2 of **Exercise 6.1***, respectively. The other parts have similar algorithms.

9.2*. Apply the following instructions in the range of a for template with the specification $i=1,5$.

1. Assign A_{i1} and 1 to the real variable max and integer variable p , respectively;
2. In the range of a for template with the specification $j=2,7$, substitute A_{ij} for max if it is greater than or equal to max . Further, substitute j for p ;
3. After exiting this inner loop, the (i, p) -entry is the position of the maximum entry in row i . Print max together with this position with an appropriate design and continue.

9.3*. Use the function `rev()` in **Example 7.4**

9.4*. Read the matrix A . Consider the main part of either of Algorithms 8.11(b) or 8.11(c) including the two nested loops. In that algorithm, replace a_i and a_{i+1} by a_i , and a_{i+1} by a_{i+1} , and A_{ki} , and $A_{k(i+1)}$, A_{ki} , and $A_{k(i+1)}$, respectively. In the analogous algorithm for descending order, the inequality ' $<$ ' is replaced by ' $>$ '. Accordingly, you have two algorithms, namely, one for sorting in ascending order of 7 entries in row k and the other for sorting in descending order of the same entries.

Now, in the range of a for template with the specification $k=1,8$, place the above-mentioned ascending algorithm if k is odd and descending algorithm otherwise. Finally, print the deformed matrix A after exiting this loop.

9.5*. First, simultaneously, calculate the sum of the main and the secondary diagonals. For this purpose, take two variables sD and sS with the initial values of 0. Then, substitute the addition of A_{kk} and sD for sD and the addition of $A_{k(n+1-k)}$ and sS for sS in the range of a for template with the specification $k=1,n$. Upon exiting this loop, print the message Not magic and terminate the algorithm if $sD \neq sS$; otherwise, calculate the sum of the entries in the k -th row and compare it with sD (or sS) in the range of a for template with the specification $i=1,n$. To this end, take the variable sR with the initial values of 0. Then, in the range of a for template with the specification $j=1,n$, substitute the addition of A_{ij} and sR for sR . Upon exiting this loop, print the message Not magic and terminate the algorithm if $sD \neq sR$.

Repeat the recent procedure for the column instead of row. Now, print the message Magic after exiting the outer loop.

9.6*. Remove column j of A and pull the entries of the next columns one column backward. To this aim, substitute A_{ik+1} for A_{ik} for A_{ik} in the range of two nested for loops with the specifications $i=1,n$ and $k=j,n-1$, respectively. Similarly, remove row i of A .

9.7*. Define the determinant as follows if $n = 2$.

$$pet = A_{11}A_{22} - A_{12}A_{21}; \quad pet = A_{11}A_{22} - A_{12}A_{21} ;$$

Otherwise, use the method of extending the determinant based on the first row (or the first column). For this purpose, use the function of the previous exercise.

9.8*. Use the sub-algorithm $A_{ij}()$ in **Exercise 9.6*** and the function $\det()$ in **Example 9.8**. Name the adjacent matrix as $adjA$ and define it as follows. In the range of two nested for loops with the specifications $j=1,n$ and $i=1,n$, respectively, first, call the sub-algorithm $A_{ij}()$ for A , i , and j and receive the return matrix in the name of A_{hat} . Now, define the (j, i) -entry of $adjA$ using the following formula.

$$adjA_{ji} = -1^{i+j} \det(A_{hat}), \quad i = 1, 2, \dots, n, \quad j = 1, 2, \dots, n.$$

$$adjA_{ji} = -1^{i+j} \det(A_{hat}), \quad i = 1, 2, \dots, n, \quad j = 1, 2, \dots, n.$$

9.9*. Perform the following in the range of an outer for loop with the specification $k=1,n$:

1. Using an if template, if $A_{kk} = 0$ $A_{kk} = 0$ then, perform the following instructions in the range of an inner for loop with the specification $i=k+1,n$:
 - 1.1. Using an if template, if $A_{ik} \neq 0$ $A_{ik} \neq 0$ then, perform the instructions from the label 'b' onward in Case 2 (Part 1 and Part 2) of **Flowchart 9.9**;
 - 1.2. Otherwise, continue the inner for loop;
2. Otherwise, continue the outer for loop.

9.10*. Use the idea in **Flowchart 9.9** removing the instructions concerned the matrix B . Moreover, transfer the control to the beginning of the leading for loop which is supposed to have the label 'a' instead of the instructions after exiting the inner loop in Case 1. This can be done by the `goto a` statement in C++ codes and `continue a` statement in Java codes.

Bibliography

- [1] Allain A. Jumping into C++. CProgramming.com 2013.
- [2] Atzori R. (Online e-book containing more than 300 flowcharts): <http://www.flowgorithm.org/>
- [3] Burd B. A. Beginning programming with Java for Dummies, 5th Edition. For Dummies 2017.
- [4] Byrne P. and Lyons G. The Effect of Student Attributes on Success in Programming. Proceeding of 6th Annual Conference on Innovation and Technology in Computer Science Education ITiCSE United Kingdom 49–52, 2001.
- [5] Cadenhead R. Sams teach yourself Java in 21 days (Covering Java 8), 3rd Edition. Sam Publishing 2016.
- [6] Chaudhuri A. B. The art of programming through flowcharts and algorithms. Laxmi Publications 2005.
- [7] Cook D. D. Flowgorithm: Principles for Teaching Introductory Programming Using Flowcharts. Proceedings of the 2015 American Society for Engineering Education/Pacific South West. 158–167, 2015.
- [8] Davis G. B. Fortran 77: A structured disciplined style (ISE Editions). McGraw-Hill Education 1984.
- [9] Deitel P. J. and Deitel H. C++: How to program, 10th Edition. Pearson International 2017.
- [10] Eckel B. Thinking in Java, 4th Edition. Prentice Hall 2006.
- [11] Erosa A. M. and Hendren L. J. Taming control flow: a structured approach to eliminating goto statements. Proceedings of the 994 IEEE International Conference on Computer Languages, 229–240, May1994.

- [12] Flanagan D. Java in a Nutshell, 5th Edition. O'Reilly Media Inc. 2005.
- [13] Farrell J. Computer Programming Logic Using Flowcharts. Boyd & Fraser Pub. Co. 1994.
- [14] Gamow G. One, two, three...infinity facts and speculations of science (Dover Books on Mathematics) Revised Edition (1947, revised 1961), Viking Press (copyright renewed by Barbara Gamow, 1974), reprinted by Dover Publications, illustrated by the author; eBook edition, Dover 2012.
- [15] Gomes A. and Mendes A. J. Learning to program-difficulties and solutions. International Conference on Engineering Education-ICEE, 2007: <http://icee2007.dei.uc.pt/proceedings/papers/411.pdf>
- [16] Gomes A. Carmo L. Bigotte E. and Mendes A. J. Mathematics and programming problem solving. Proceeding of the 3rd E-Learning Conference in Computer Science Education (CD-ROM), Coimbra, Portugal, September 2006.
- [17] Janfada A. S. FORTRAN 77: Programming and comprehensive reference (a textbook in Persian language). Urmia University Publications 1994.
- [18] Janfada A. S. Elementary programming in Pascal, related by algorithm (a textbook in Persian language). Urmia University Publications 2009.
- [19] Janfada A. S. Elementary programming in C++, via algorithm (a textbook in Persian language). Urmia University Publications 2018.
- [20] Kernighan B. W. and Ritchie D. M. C Programming Language, 2nd Edition. Prentice Hall 1988.
- [21] Liberty J. and Jones B. Sams teach yourself C++ in 21 days, 5th Edition. Sams Publishing, 2004.
- [22] Merritt S. M. and Stix A. Migrating from Pascal to C++. Springer 1997.
- [23] Moore D., Musciano C., Liebhaber M. J., Lott S. F., and Starr L. "goto considered harmful' considered harmful' considered harmful? Communications of the ACM 30(5): 351-355, 1987.

- [24] Mueller F. and Whalley D. Avoiding unconditional jumps by code replication. Proceedings of the ACM SIGPLAN 1992 conference on Programming language design and implementation, 322–330, 1992.
- [25] Murphy J. Ridout D. and McShane B. Numerical analysis algorithms and computation. Halsted Press 1988.
- [26] Parhami B. Introduction to computer (in Persian language). Iran University of Science and Technology Publishing 1984.
- [27] Peterson W., Kasami T., and Tokura N. On the capabilities of while, repeat, and exit statements. Communications of the ACM 16(8): 503–512, 1973.
- [28] Ramshaw L. Eliminating go to's while preserving program structure. Journal of the ACM (JACM), 35(4): 893–920, 1988.
- [29] Rubin F. 'goto considered harmful' considered harmful. Communications of the ACM, 30(3): 195–196, 1987.
- [30] Savitch W. Problem solving with C++, 10th Edition. Pearson Education 2017.
- [31] Scheid F. Computer science (Schaum's Outline Series). McGraw-Hill Companies 1970.
- [32] Scheinerman E. C++ for Mathematicians: An Introduction for students and professionals. CRC Press 2006.
- [33] Schildt H. C++: The complete reference, 4th Edition. McGraw-Hill Education 2002.
- [34] Schildt H. Herb Schildt's C++ programming cookbook. McGraw-Hill Education 2008.
- [35] Sleeman, D. The Challenges of Teaching Computer Programming. Communications of the ACM. 29(9): 840–841, 1986.
- [36] Sierra K. and Bates B. Head first Java, 2nd Edition. O'Reilly Media Inc. 2005.

- [37] Stroustrup B. The C++ programming language, 4th Edition. Addison-Wesley Professional 2013.
- [38] Stroustrup B. Programming: Principles and practice using C++, 2nd Edition. Addison-Wesley Professional 2014.
- [39] Sutter H. and Alexandrescu A. C++ coding standards: 101 rules, guidelines, and best practices. Addison-Wesley Professional 2004.
- [40] Swan T. Mastering Turbo Pascal 5.5, 3rd Edition. Hayden Books 1989.
- [41] Wing J. M. Computational thinking. Communications of the ACM. 49(3): 33–35, 2006.
- [42] Wolfram, S. How to Teach Computational Thinking. Stephen Wolfram Blog: <http://blog.stephenwolfram.com/2016/09/how-to-teach-computational-thinking/>.

Websites

- [43] A Crash Course from C++ to Java: http://www.horstmann.com/cc/c_to_java.pdf
- [44] artima Scala, consulting, training, books, and tools: <https://www.artima.com/objectsandjava/>
- [45] [BeginnersBook.com](https://beginnersbook.com/) - Tutorials for Beginners: <https://beginnersbook.com/>
- [46] Codequiz.in: <http://www.codequiz.in/category/c-programs-2/>
- [47] Computer Science University of Toronto: <http://www.cs.toronto.edu/>
- [48] [Cprogramming.com](https://cboard.cprogramming.com/c-programming/): <https://cboard.cprogramming.com/c-programming/>
- [49] GeeksforGeeks: <https://www.geeksforgeeks.org/>
- [50] [IncludeHelp.com](https://www.includehelp.com/): <https://www.includehelp.com/>
- [51] [infocodify.com](http://www.infocodify.com/): <http://www.infocodify.com/>
- [52] javaTpoint: <https://www.javatpoint.com/>
- [53] Learning a New Programming Language: Java for C++ Programmers: <http://pages.cs.wisc.edu/~hasti/cs368/JavaTutorial/>

- [54] MathBits.com: <https://mathbits.com/JavaMathBits/JavaResourcesOpening.html>
- [55] Oracle Java Documentation: <https://docs.oracle.com/javase/tutorial/java/>
- [56] Programize: <https://www.programiz.com/java-programming/>
- [57] Simple Snippets, Quality Edu-Tech Videos & Tutorials: <https://simplesnippets.tech/>
- [57] SitePoint Pty Ltd [AU]: <https://www.sitepoint.com/>
- [59] Software Development & Entrepreneurship Tutorials: <http://tutorials.jenkov.com/>
- [60] StudyTonight: <https://www.studytonight.com/>
- [61] Study.com: <https://study.com/>
- [62] ThoughtCo.: <https://www.thoughtco.com/>
- [63] Tutorial Dost: <http://www.tutorialdost.com/>
- [64] TutorialsPoint simply easy learning: <https://www.tutorialspoint.com/java/>
- [65] w3schools.com: <https://www.w3schools.com/>

Index

- access specifier, 1
 - – private, 1
 - – protected, 1
 - – public, 1, 2
- algorithm, 1, 2, 3, 4, 5, 6
 - – bubble sorting, 1
 - – definition, 1
 - – designing, 1
 - – Euclidean, 1
 - – features, 1
 - – figurative form, 1
 - – flow of, 1
 - – general steps of writing, 1
 - – generalized Euclidean, 1
 - – good, 1
 - – implementing, 1, 2, 3, 4
 - – standard, 1
 - – swap, 1, 2
 - – symbolic form, 1
 - – writing, 1, 2, 3
 - – writing form, 1
- algorithm-writing, 1
- al-Khawrizmi, 1

- approximated integration
 - – Simpson, 1
 - – trapezoidal, 1
- argument, 1, 2, 3, 4
- array
 - – as parameter, 1, 2
 - – declaring, 1
 - – length, 1, 2, 3, 4, 5
 - – one-dimensional, 1
 - – two-dimensional, 1
- base-2 decimal system, 1
- bisection method (for finding roots), 1
- Bjarne Stroustrup, 1
- block, 1
 - – static, 1
- brain, 1, 2
 - – commands, 1, 2, 3
 - – process, 1, 2, 3
 - – reaction, 1, 2
- branching
 - – multi-way, 1, 2, 3
 - – two-way, 1, 2
- call unit, 1
- character
 - – control, 1, 2
 - – conversion, 1
 - – whitespace, 1, 2

- class, 1
- constant, 1
 - - declaring, 1
- constructor, 1
 - - calling, 1
 - - copy, 1
 - - default, 1
 - - parameterized, 1
- dangling else problem, 1
- data, 1
- data member, 1
 - - static, 1
- data type, 1, 2
 - - boolean, 1
 - - byte, 1
 - - char, 1, 2, 3
 - - const, 1
 - - double, 1, 2, 3
 - - float, 1, 2, 3, 4
 - - int, 1, 2, 3
 - - long, 1, 2
 - - primitive, 1
 - - reference, 1
 - - short, 1, 2
 - - signed, 1
 - - unsigned, 1
 - - void, 1, 2, 3
- Denis Ritchie, 1

- destructor, 1
- elementary row operations, 1
- Enter key symbol, 1
- escape sequence. *See* control sequence
- expression
 - – algebraic, 1
 - – arithmetic, 1
- Fibonacci sequence, 1, 2
- flowchart, 1
 - – shapes, 1, 2
- format (C++), 1, 2, 3
 - – setprecision, 1
 - – setw, 1
- format (Java), 1, 2
- format specifier (Java)
 - – flag, 1
 - – precision, 1
 - – width, 1
- function, 1, 2, 3
 - – 1-return, 1
 - – body, 1, 2
 - – calling, 1
 - – int main(), 1
 - – library, 1, 2
 - – no-return, 1
 - – recursive, 1, 2
 - – self-calling. *See* recursive function
 - – void (calling), 1

- - void main(), 1
- Gosling, James, 1
- greatest common divisor, 1
- Green Team, 1
- hanging problem, 1
- IDE workspace
 - - C++, 1
 - - Java, 1
- identifier, 1
- implementation table, 1, 2, 3
 - - arranging, 1, 2
- incompatibility (in the for loop), 1
- infinity value (in Java), 1
- input
 - - entering, 1
 - - notification, 1, 2, 3
 - - unit, 1
- keyword, 1, 2, 3
 - - const, 1
 - - static, 1, 2
 - - this, 1
 - - void, 1
- Khayyam-Pascal triangle, 1
- Lagrange interpolation polynomial, 1
- literal, 1
 - - Boolean, 1

- – character, 1
- – numerical, 1
- – string, 1
- loop
 - – automated. See for **loop**
 - – conditional, 1, 2, 3
 - – do-while, 1
 - – for, 1
 - – infinite for, 1
 - – while, 1
- matrix. See two-dimensional array
 - – determinant, 1
 - – identity, 1
 - – inverse, 1
 - – row-reduced, 1
 - – row-reduced echelon, 1
- member function, 1, 2
- method, 1, See *member function*
 - – class, 1, 2
 - – instance, 1
 - – main (Java), 1
 - – member. See class method
 - – static, 1
- namespace, 1
 - – std, 1
- NaN value (in Java), 1
- Naughton, Patrick, 1

- nested
 - – conditional statements, 1
 - – conditional templates, 1
 - – if-else templates, 1
- object, 1, 2
 - – creating, 1
 - – scanner, 1
- operator, 1
 - – &, 1
 - – *, 1
 - – ,, 1, 2
 - – ::, 1
 - – ?, 1, 2
 - – address-of, 1
 - – arithmetic, 1
 - – assignment, 1, 2
 - – comparative. *See relational operator*
 - – dereference, 1
 - – logical, 1
 - – relational, 1
 - – scope resolution, 1, *See :: operator*
 - – sizeof, 1, 2
- output
 - – heading, 1, 2, 3
 - – unit, 1
- package, 1
- parameter, 1, 2, 3

- – actual. See [argument](#)
- – formal. See [parameter](#)
- – pass-by-reference, [1](#)
- – pass-by-value, [1](#)
- – reference. See [pass-by-reference](#)
- – value. See [pass-by-value](#)
- pointer, [1](#)
 - – const, [1](#)
 - – declaring, [1](#)
- preprocessor directive, [1](#)
 - – conio.h, [1](#)
 - – iomanip.h, [1, 2](#)
 - – iostream.h, [1](#)
 - – math.h, [1](#)
- prime numbers, [1](#)
- priority of operators, [1](#)
- problem solving, [1, 2, 3, 4](#)
- program, [1, 2, 3](#)
 - – C++, [1](#)
 - – complete, [1, 2](#)
 - – Java, [1](#)
 - – writing, [1, 2, 3](#)
- programming, [1, 2, 3](#)
 - – object-oriented, [1](#)
 - – styles, [1, 2](#)
- programming language, [1, 2, 3](#)
 - – C, [1, 2](#)
 - – C++, [1](#)

- – Fortran, 1
 - – Java, 1
- punch card, 1
- Puzzle of Hanoi Towers, 1

- reflexing point, 1, 2, 3
- reserved word. See [keyword](#)
- revisions
 - – C++, 1
 - – Java, 1
- rule
 - – 1 of arranging implementation table, 1
 - – 2 of arranging implementation table, 1
 - – calling the array-return methods, 1
 - – calling 1-return function, 1
 - – calling a void function, 1
 - – constructing the sub-algorithms, 1
 - – directions, 1
 - – ending the sub-algorithms, 1
 - – grouping, 1
 - – implementation table for nested loops, 1
 - – intersection of the T- and F-paths, 1
 - – making the if-else-if template, 1
 - – merging the conditions, 1
 - – multi-using the reading methods, 1
 - – naming identifiers, 1
 - – parameter list in functions, 1
 - – parameter-argument, 1
 - – parameters of multi-return subprograms, 1

- – priority of operators, 1
 - – starting the sub-algorithms, 1
- series, 1
 - – double, 1
 - – single, 1
- Sheridan, Mike, 1
- slogan of the book, 1
- solution analysing, 1, 2
- solving linear equations system
 - – backward displacement method, 1
 - – Cramer method, 1
 - – forward displacement method, 1
 - – Gauss eliminated method, 1
 - – matrix method, 1
 - – triangular decomposition method, 1
- sorting method
 - – bubble, 1, 2
 - – insertion, 1
- specification (of the for loop), 1
- statement, 1
 - – assignment, 1, 2
 - – break, 1, 2, 3, 4, 5
 - – cin, 1, 2
 - – clrscr, 1
 - – continue, 1, 2, 3
 - – cout, 1, 2
 - – do-while, 1
 - – exit(0), 1, 2

- – frequent assignment, 1
- – getch(), 1
- – goto, 1, 2, 3
- – if, 1
- – if-else, 1
- – if-else-if, 1
- – print, 1
- – printf, 1
- – println, 1
- – return, 1
- – return 0, 1
- – switch, 1
- – System.exit(0), 1
- – using namespace std, 1
- – while, 1
- sub-algorithm, 1
 - – function. See **function**
 - – multi-return, 1
- subprogram, 1, 2
 - – multi-return, 1
- syntax, 1
 - – assignment statement, 1
 - – constant declaration, 1
 - – continue, 1
 - – creating an object, 1
 - – defining class, 1
 - – do-while, 1
 - – for, 1

- – formatted print, 1
- – function (method) defining statement, 1
- – if-else, 1
- – if-else-if, 1
- – input statement, 1
- – namespace, 1
- – output statement, 1
- – switch, 1
- – this keyword, 1
- – variable declaration, 1, 2
- – while, 1
- template
 - – assignment, 1, 2
 - – compound, 1
 - – conditional, 1
 - – do-while, 1
 - – for, 1
 - – if, 1, 2
 - – if-else, 1
 - – if-else-if, 1
 - – input, 1
 - – output, 1
 - – simple, 1
 - – switch, 1
 - – while, 1
- variable, 1, 2, 3, 4
 - – class, 1

- - declaring, [1](#), [2](#), [3](#), [4](#), [5](#)
- - global, [1](#)
- - instance, [1](#)
- - local, [1](#)
- - static local, [1](#)
- vector. *See* One-dimensional array

Endnotes

1

Muhammad ibn Musa al-Khawrizmi (780 A.D. - 850 A.D.).

2

The verbs “implement” and “run” or “execute” are used for algorithms and programs, respectively.

3

The “implementation table” is used for both algorithms and programs.