

Ministry of Education and Science of Ukraine

**UKRAINIAN STATE UNIVERSITY
OF RAILWAY TRANSPORT**

COMPUTER SCIENCE

**FUNDAMENTALS OF ALGORITHMIZATION
OF BASIC COMPUTATIONAL PROCESSES**

TUTORIAL STUDENT'S BOOK

Kharkiv 2020

UDC 004.421.2
C 10

*Recommended for publication
by Ukrainian State University of Railway Transport
as a tutorial student's book
(October 29, 2019, protocol No. 7)*

Reviewers:

Doctor of Technical Sciences, Professor V Samsonkin (SUIT, Kiev)
Doctor of Technical Sciences, Professor A. Yerokhin (NURE, Kharkiv)
Doctor rerum naturalium O. Panchenko (Onapsis Inc., Berlin, Germany)
Doctor of Technical Sciences, Professor M. Miroshnik (UkrSURT, Kharkiv)

Authors:

S. Bantyukov, V. Merkulov, I. Biziuk, S. Bantyukova

C 10 Computer science. Fundamentals of algorithmization of basic computational processes: Tutorial student's book / S. Bantyukov, V. Merkulov, I. Biziuk, S. Bantyukova. — Kharkiv: UkrSURT, 2020. — 137 p., ill. 71, tables 3.

ISBN

The tutorial student's book contains lecture materials, control assignments, thematic questions and is intended for students to study all specialties and forms of discipline teaching that use computer technology to organize calculations in professional activity.

UDC 004.421.2

ISBN

© S. Bantyukov, V. Merkulov,
I. Biziuk, S. Bantyukova
© Ukrainian State University
of Railway Transport, 2020.

CONTENT

PREFACE	4
INTRODUCTION	5
UNIT 1. THEORETICAL BASIS OF ALGORITHMS BUILDING	7
1.1. Stages of solving problems on the computer	7
1.2. General information about algorithms	17
1.2.1. The concept of the algorithm	17
1.2.2. Objects of action in algorithms and programs	20
1.2.3. Methods for describing algorithms	20
1.2.4. Rules of algorithm construction	24
1.2.5. Properties of algorithms	25
1.2.6. Types of algorithms	26
1.3. Methodologies and classification of algorithms design methods	27
1.4. The problem of choice and analysis of complexity of algorithms	35
1.5. Typical structures of algorithms	41
UNIT 2. LINEAR COMPUTING PROCESSES	49
UNIT 3. OVERALL COMPUTING PROCESSES	53
UNIT 4. CYCLICAL COMPUTING PROCESSES	73
4.1. Simple arithmetic cyclic computing processes	73
4.2. Nested cyclic computing processes	78
4.3. Iterative cyclic computing processes	82
UNIT 5. DESIGNING ARCHITECTURES OF ARCHITECTURE PROCESSING	108
5.1. Concepts and main characteristics of the array	108
5.2. Algorithms for processing one-dimensional arrays	109
5.3. Algorithms for processing two-dimensional arrays	115
5.4. Algorithms for sorting arrays for a given feature	120
BIBLIOGRAPHICAL LIST	137

PREFACE

The introduction of new information technologies in all spheres of modern life has led to the fact that the ability to work on a computer is a necessary attribute of professional activity of any specialist and in many respects determines his rating in society.

Computer disciplines under different names are taught in higher education. Depending on the students' future profession, these training courses vary in content, direction and timing.

Regardless of the specifics, they all have one fundamental feature - the need to constantly adjust the content of the courses, because every few years updated and improved computer hardware and software is introduced to the market. In this regard, the textbooks require significant reworking, and teachers feel the need for carefully crafted teaching materials.

It is a common mistake to think that students need to learn how to use only engineering mathematical software (MathCad, MatLab, FreeLab, SciLab, etc.) to solve their educational tasks, so it is not advisable to teach them programming, since only IT professionals can create high-quality programs.

In fact, knowing the "inner world" of computing, unlike the just simple use of certain applications, allows you to use computers as a highly effective tool. Modern programming systems featuring advanced, high-level algorithmic languages, user-friendly interfaces and powerful built-in editors and debuggers allow you to create sophisticated software products for all professionals in the industry.

In this case, the quality of software projects depends on the commitment and desire of the authors and, first of all, on how well their algorithms are thought out, elaborated and effective.

Possessing the skills of designing high-quality algorithms, the ability to minimize the number of logical errors when compiling them, the ability to quickly find and eliminate the remaining ones is an indispensable feature of a railway engineer armed with modern computer technology and advanced technology.

This tutorial student's book focuses on approaches to constructing efficient algorithms and avoids excessive mathematization of the proposed material.

INTRODUCTION

One of the basic concepts of computer science is the concept of algorithm, as a rule of information transformation. The purpose of this tutorial student's book is to teach you the basics of algorithmization.

This section is the foundation of any computer discipline, but the curriculum has a short term of study. Therefore, some important issues are not addressed, or are not addressed sufficiently deep (*for example: variability in the organization of calculations, the construction of optimal algorithmic structures, solving problems for compiling algorithms, etc.*).

There are a large number of printed materials in various technical fields that provide algorithms for various applications and related programs. The disadvantage of many of them is the lack of systematic use as a component of basic elementary algorithmic constructions, which leads to some problems in developing optimal software projects.

The subjects taught by the Department of Computer Engineering and Control Systems are part of all educational programs of Ukrainian State University of Railway Transport (UkrSURT). They have the purpose of preparing students to independent development of software projects solving such problems as special engineering calculations, synthesis, analysis and optimization of systems, modeling of processes and phenomena, processing of measurement results, etc.

This manual is part of the educational and methodical complex of publications, which cover all the disciplines of the department. The materials collected here have been repeatedly used in the classrooms.

Particularly, the publication includes the concise but complete presentation of the materials and a large number of examples.

The ability to use the materials of the manual in electronic form facilitates both classroom and independent work of students, as well as the activities of teachers in the organization of distance learning, the development of test questions for automated module control.

The purpose of the publication is dictated primarily by the lack of adapted courses in computer science for foreign students, so the material is presented in English (in the presence of the Ukrainian version). It will significantly expand the audience of potential readers

and to orient this contingent to work independently theoretical and practical material.

The language of the tutorial student's book is adapted for students who do not have sufficient grammatical constructions. This is one of important differences from other tutorial student's book.

The theory and test tasks are designed to give students the ability to work with specialized computer literature.

The tutorial contains:

- theoretical fundamentals of algorithm development;
- test questions for topic control;
- control questions that can be used by students for self-control and by teachers for controlling the progress;
- Illustrations that facilitate the teacher in structuring and teaching the material;
- a list of recommended literature to deepen the acquired knowledge and improve practical skills.

UNIT 1. THEORETICAL BASIS OF ALGORITHMS BUILDING

1.1. Stages of solving problems on the computer

The use of computers involves the processing of information using a pre-compiled program, which is especially suitable for the task of laborious tasks that require many calculations. The advent of PCs with advanced tooling software has made computing facilities accessible to all interested professionals and has made it possible to significantly speed up the results of calculations.

It is important to keep in mind the need for extensive preparatory work and the mandatory steps in this process (Fig.1.1) [2].

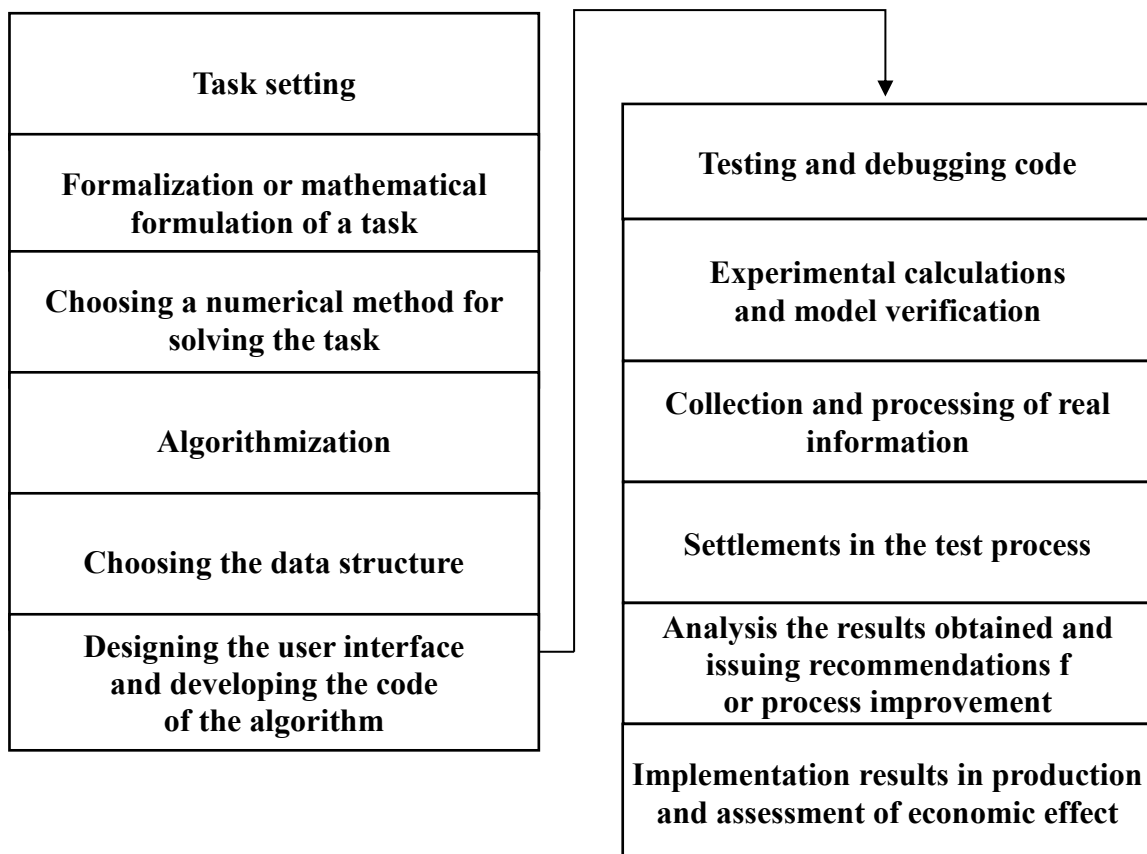


Fig. 1.1. Steps to resolve the tasks on the computer

I stage. Task setting.

This step involves the following:

- definition of an essence of a task and purpose that it has to be reached as a result of its decision;
- made a general research for the problem under study;
- definition of conditions that are caused by the interaction of various factors influencing the process.

On the basis of verbal formulation of the research problem, the variables to be determined are selected, restrictions are written, and the relations between the variables are recorded. At this stage, an analysis is made of: existing analogues; hardware and software; volume and specificity of the output.

Example: Determine triangle height by the set area if it is known that the basis of a triangle is more than height at a certain size. (Fig.1.2).

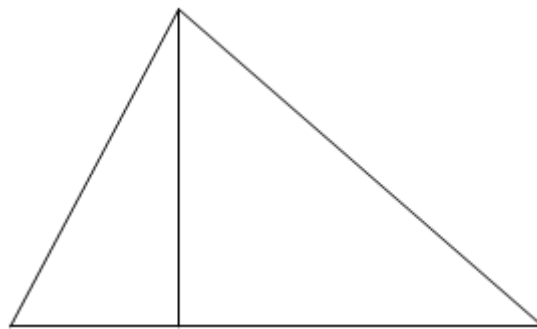


Fig. 1.2. Illustration for example stage II

II stage. Formalization or mathematical formulation of the task.

During this step:

- introduce the system of symbols;
- construct mathematical model of the task which are represented by set of criterion function and system of the equations or inequalities;
- establish the belonging of the solved task to one of known classes of tasks and select the corresponding mathematical apparatus.

As a result, the engineering task takes the form of a formalized mathematical task.

Example: Determine the height of the triangle (x) by the set area (c) if it is known that the base of the triangle is more than the height by (b) (Fig.1.3).

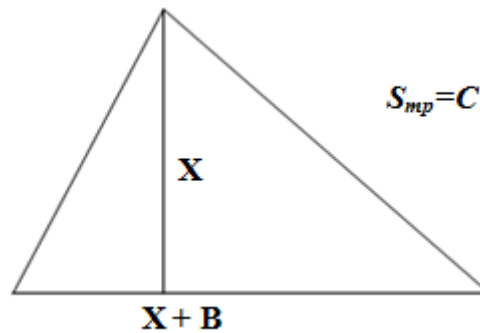


Fig. 1.3. Illustration for stage II

Triangle area $c = 0,5x(x+b) \Rightarrow x^2+bx-2c = 0$

Thus, the mathematical formulation is reduced to finding the true positive square root of the quadratic equation.

Some tasks do not allow or require mathematical formulation (*for example: word processing*).

III stage. Choosing a numerical method for solving the task.

The solution of the problem must be reduced to a sequence of arithmetic and logical operations. The development and study of such methods is dealt with in the mathematics section called numerical analysis.

Example: the true roots of the quadratic equation $ax^2+bx+c=0$ are calculated by the formula:

$$x_{1,2} = (-b \pm \sqrt{b^2 - 4ac}) / (2a),$$

and the value specified integral where v – constant:

$$\int_a^b (x^3 + v) dx,$$

by the formula:

$$y = b^4/4 - a^4/4 + v(b-a).$$

These formulas are based on precise methods for solving a problem (a series of arithmetic and logical actions). However, for most practice tasks, exact methods are either unknown or contain

cumbersome formulas, so numerous methods have been developed at different times to give a rough approximation of the required accuracy.

Example of numerical method – method of rectangles for calculating defined integrals. It does not require the calculation of the primitive, since the integral is replaced by the finite sum of the values of the integrand (Fig. 1.4).

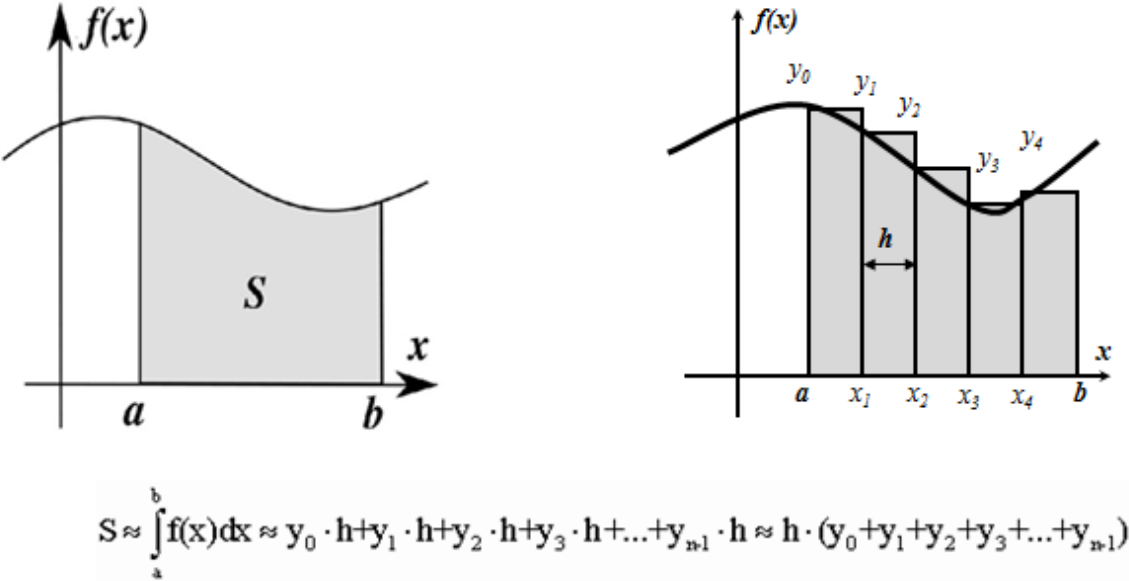


Fig. 1.4. Illustration for stage III

Besides using formulas and manually compiling programs, you can use software tools in the form of specialized mathematical packages or standard library programs. Further, it is recommended to attempt to solve the problem by other methods, including proprietary algorithms and programs, and compare results.

IV stage. Algorithmization.

This step involves the following:

- making a scheme of the algorithm for solving the task;
- decomposition of the computing process into possible components;
- establishing the order of following them;
- description of the contents of each such part.

All the details of this stage are carefully considered in the following material (Fig. 1.5).

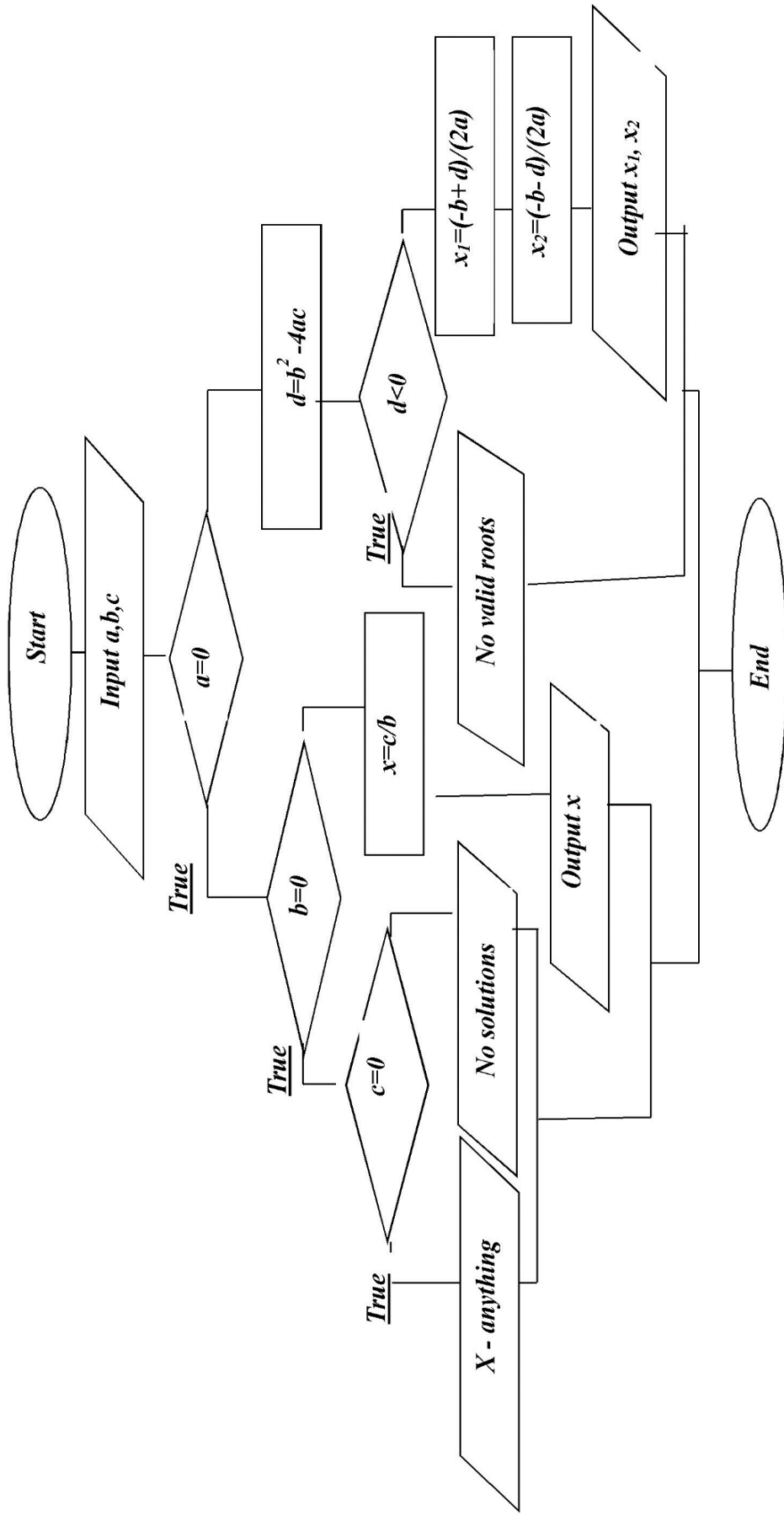


Fig. 1.5. Shows an example of a scheme for finding all the roots of a quadratic equation

V stage. Choosing the data structure.

The data structure is a way of storing data on a computer, to which the algorithm of their processing depends. This choice is usually made taking into account the particularities of the algorithm implementation in one or another programming language.

The well-designed data structure allows for various operations to be performed using as few resources as possible (such as operation time or the amount of RAM).

Data structures are programmed using data types, references, and operations on them, which are performed in the selected algorithmic language. (Fig.1.6)

The development of different types of software has shown that the complexity of implementation and the quality of the final system depends on the choice of the right data structure. Once the data structure is selected, the selection and operation of the algorithm often becomes apparent. However, sometimes things are the other way around - data structures are chosen for the sake of optimizing key tasks with the help of certain algorithms that work best with their type of data structures.

In any case, choosing the right data structure is very important.

VI stage. Designing the user interface and developing the code of the algorithm (Fig.1.7).

The completeness of using the potential capabilities of the available software resources of the project depends on the quality of the user interface, which is an independent characteristic of the software product, comparable in importance to such indicators as reliability and efficiency of use.

The main advantage of a good user interface is that the user always feels that he controls the software, and not the software that controls it.

To create such a feeling of “internal freedom” for the user, the interface must have a number of properties:

- disengagement;
- coherence;
- friendliness;
- feedback;
- simplicity;
- flexibility;
- attractiveness.

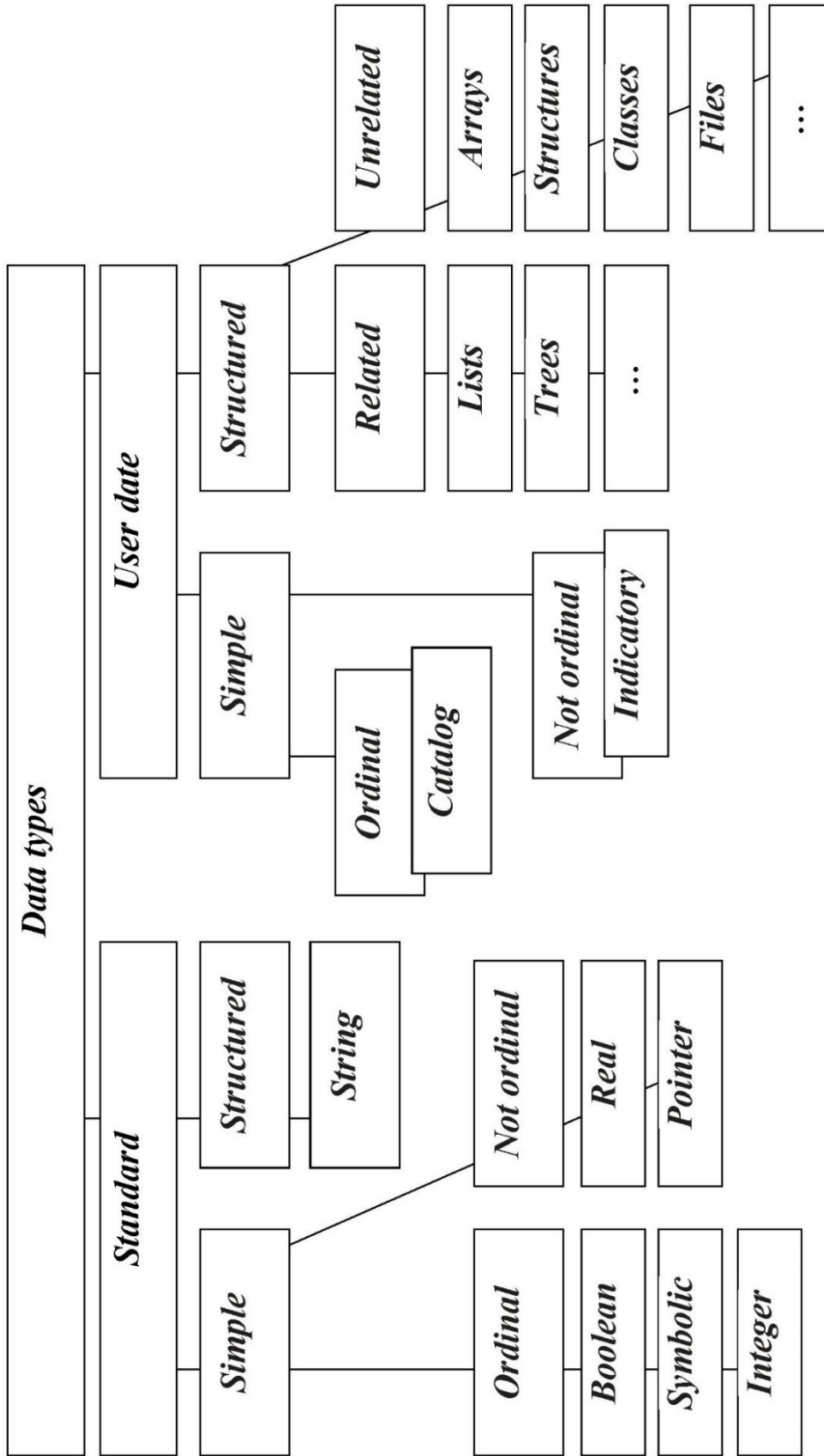
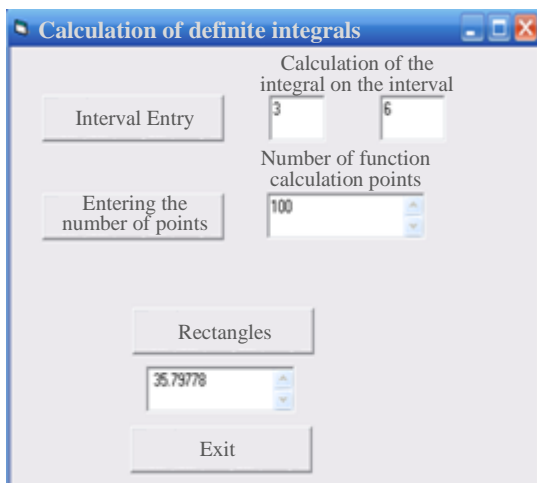


Fig. 1.6. Classification of data types

Programming the task – writing a developed algorithm in the computer source language. It is recommended to use basic designs and data types, expanding the scope of application as you gain experience in programming and practical problem solving.

Criteria for selection of programming languages:

- appropriate to the nature of the problem being solved;
- suitable environment - development, operational, graphic;
- safety;
- possibility hardware control;
- broadcast speed;
- object code performance;
- ability to work with selected data structures;
- service capabilities (debugging tools, working with files, built-in help, navigation);
- integration with teamwork tools.



```

Private Sub CmdLRectangle_Click()
a = Val(x0)
b = Val(xk)
h = (b - a) / n
s = 0 : x = a
Do While x < b
s = s + f(x)
x = x + h
Loop
d = h * s
TxtLRectangle = TxtLRectangle + Str(d)_
+ Chr(13) + Chr(10)
End Sub

```

Fig. 1.7. An example of an interface (form with controls) and the fragment of the program code for the task of calculating of the defined integral by the method of left rectangles

VII stage. Testing and debugging code.

This stage is intended to check the correctness of the program (Fig. 1.8) and correct any detected errors (Fig. 1.9) and provides:

- syntactic debugging;
- debugging semantics and logical structure;
- test calculations and analysis of test results;
- program improvement.

Guarantee the correctness of the result may be, for example:

a) checking the fulfillment of the conditions of the task (for example, for the algebraic equation, the found roots are substituted into the original equation, and the differences between the left and right parts are checked);

b) qualitative analysis of the task;

c) recalculation (if possible by other method).

Example: for algebraic equation $x^2+bx-2c=0$ the founded roots are substituted into the original equation and checked the differences between the left and right parts.

For certainty let's take $b=3$; $c=2$. The equation will look like $x^2+3x-4=0$.

Using formula:

$$x_{1,2} = (-b \pm \sqrt{b^2 - 4ac}) / (2a)$$

Obtain $x_1 = -4$ $x_2 = 1$. We are interested in the positive root $x = 1$. Make a check - substitute the obtained value in the original equation. *Conclusion:* the answer is correct.

```

Private Sub CmdПРЯМ_Click()
    a = Val(x0)
    b = Val(xk)
    h = (b - a) / n
    s = 0: x = a:
    Do While x < b
        s = s + f(x)
        x = x + h
    Loop
    d = h * s:
    TxtПРЯМ = TxtПРЯМ + Str(d) + Chr(13) + Chr(10)
End Sub

```

Fig. 1.8. An example of a semantic error in program code

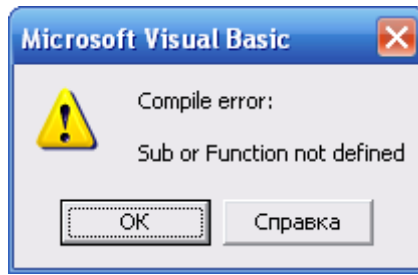


Fig. 1.9. An example of a compile error

The test - is a specially selected source data in combination with the results that the program must produce when they are processed.

Example: data for testing the problem of solving the algebraic equation $ax^2+bx+c = 0$ are presented in the table 1.1.

Table 1.1

Test table for solving the algebraic equation $ax^2+bx+c = 0$

<i>a</i>	<i>b</i>	<i>c</i>	<i>Answer</i>
0	0	0	Any <i>x</i>
0	0	1	No solutions
0	1	0	$x=0$
0	1	1	$x=-1$
1	0	0	$x_1=0; x_2=0$
1	1	0	$x_1=-1; x_2=-1$
1	1	1	No valid roots
2	5	2	$x_1=-2; x_2=0.5$

VIII stage. Experimental calculations and model verification.

The experiment examines the change in the state of the object of study under the influence of the information that is communicated to him.

Testing the adequacy of the model has two goals: to verify the validity of the accepted factors in the modeling of essential factors, hypotheses, assumptions and to establish that the results obtained correspond to a given accuracy allow you to perform the necessary research.

IX step. Collection and processing of real information: preparation of input data for the task.

X step. Settlements in the test process.

XI step. Analysis the results obtained and issuing recommendations for process improvement.

XII step. Implementation results in production and assessment of economic effect.

The contents of the IX-XII steps will be considered in detail when studying the special disciplines of the respective educational programs.

1.2. General information about algorithms

1.2.1. The concept of the algorithm

Programming is to create instructions (programs) for your computer that help you solve specific tasks. The program is based on an algorithm that determines the sequence of operations performed by the machine [3].

In everyday life, we often come across algorithms. The recipe for cooking, the instructions for operating any appliance, or assembling furniture are all algorithmic entries. They surround us everywhere. Most of the actions a person does in accordance with the established rules, without even thinking about executing the algorithm.

The solution to any problem consists of a sequence of operations and, therefore, can be described by an algorithm. At the same time, the execution of each stage and the general order of the stages are clearly defined. The separate step of the algorithm should either be an intermediate problem, the solution of which is already described, or be simple enough to perform without further explanation.

The computer is a software, a device that solves certain problems with its built-in algorithms. If the computer memory does not have an algorithm for solving a specific problem, it will not be able to solve it. The computer performs the actions described in the programs and is not capable of independent "thinking". Just as not capable of independent "thinking" a vacuum cleaner, tractor or bicycle.

The word "algorithm" has its roots in Latinizing the name of Muhammad ibn Musa al-Khwarizmi in a first step to algorismus. Al-Khwarizmi (Persian: خوارزمی) was a Persian mathematician,

astronomer, geographer, and scholar in the House of Wisdom in Baghdad, whose name means "the native of Khwarazm", a region that was part of Greater Iran and is now in Uzbekistan. Now Khovarizm is a small Uzbek city of Khiva. Gradually the form and meaning of the word "algorithmism" was distorted and changed to "algorithm".

One of the earliest German mathematical dictionaries "Vollständiges Mathematisches Lexicon" (Leipzig, 1747) gives the following definition of the word *Algoritmus*: "under this name are combined concepts of four types of arithmetic, namely addition, multiplication, subtraction and division".

The substantial phenomena that led to the emergence of the concept of "algorithm" are traced in mathematics throughout its entire existence.

This is the Euclidean algorithm for finding the largest common multiple of natural numbers, found as far back as the 3rd century BC and surviving to this day.

In the 15th century, the algorithm developed by the Samarkand astronomer Al-Kashi was known, for calculating the number π , which he calculated with 17 valid significant digits after the decimal point.

Initially, the algorithm was understood as verbal rules, schemes, formulas that were used to describe the computing process.

This is not an exact mathematical definition, but only an explanation of the meaning of the word "algorithm".

With algorithms i.e. mathematicians have always dealt with effective procedures that uniquely lead to a result: multiplication by a "column", division by an "angle", a method of eliminating unknowns when solving systems of linear equations, etc.

The development of computational mathematics and computer technology necessitated clarification of the concepts of algorithm as an object of mathematical theory.

A section of discrete mathematics called theory of algorithms has appeared. The founders of the theory of algorithms are the great mathematicians of the 20th century A.I. Kolmogorov, A.A. Markov, A.P. Ershov, A.I. Maltsev, V.A. Uspensky, A.M. Turing, C. Godel, A. Church, A. Thue, E.L. Post and others [1,6].

The isolation and crystallization of the concept of the algorithm was a remarkable achievement of mathematics in the 20th century. This made it possible to solve the problem of the possibility or solution of a

number of problems in algebra, number theory, geometry and other sections of mathematics. Emerging from the intrinsic needs of mathematics itself, algorithm theory has received a diverse field of activity in connection with the development of computers. It is in the last few decades that a great number of specific algorithms have been created and studied. First of all, these include algorithms for numerical solving of problems of physics, mechanics, economics, etc. In addition, there is a large group of algorithms used to solve non-numerical problems, in particular those that arise in connection with the organization of the computers themselves. These are really the basic algorithms that make up the enormous scientific wealth, so to speak, of the material power of applied mathematics. Each such algorithm is a reusable tool, and therefore there are significant questions about developing criteria for evaluating algorithms and how to analyze them.

The concept of an algorithm, like the concept of information, cannot be precisely determined [5]. Therefore, there are a wide variety of definitions - from "naive-intuitive" ("an algorithm is a plan for solving a problem") to "strictly formalized" (normal Markov algorithms).

Definition of modern mathematics:

- the sequence of actions with strictly defined performance rules;
- prescription specifying the content and sequence of operations transforming raw data into the desired result;
- an accurate description of a computational process or any other sequence of actions;
- an exact and complete order about the sequence of performing a finite number of actions necessary to solve any task of this type.

Algorithm — is any system of calculations performed under strictly defined rules, which, after some number of steps obviously leads to the solution of the problem (A. Kolmogorov).

Algorithm — is the exact prescription specifying computation process coming from the variable source data to the desired result (A. Markov).

Algorithm – is formally described calculation procedure, receiving input data (input algorithm or argument), and outputs the calculation result to the output.

1.2.2. Objects of action in algorithms and programs

The objects of description in the schematics of the algorithms, and later in the programs, are constants, variables, expressions (arithmetic and logical) [4].

Constant – it is a value that has a constant value that does not depend on the operation performed in the algorithm. For example, the numbers 7, -1.34, 0.00556 can be constants.

Variable – is a value that can change in the implementation of the algorithm. The variable has its own name – id, *for example: X, v1.*

The value of the variable is stored in the PC memory in a machine word labeled with that name. In computational operations, the value of the variable stored in the PC's memory is used. The variable before computing operations must be defined, that is, it must be given a specific value by input or assignment operations.

The result of the implementation of the algorithm is also a variable or variables whose values are stored in the PC's memory and output as needed.

Expression – sequence of constants, variables, functions, joined by signs of arithmetic, logical or symbolic operations.

Example: $a+b$ – arithmetic expression; $z>5$ – logical expression; "*need* "+" *insert*" – symbolic expression.

1.2.3. Methods for describing algorithms

The algorithm developed must be described so that it is clear to the executor. There are several ways to do this. These include pseudocode, operator diagram, formula description, hierarchical diagrams, Nessi-Schneiderman graphs and algorithm diagrams.

Pseudocode – the way to accurately describe ordinary language sentences, in other words – a verbal description of the action sequences. This method is not obvious and is used only for relatively simple tasks, *for example:*

If $k \leq 5$ then
$a=3$
else
$a=0$

An operator schema provides a string character record with indexes to specify a given sequence of actions, *for example*:

$$A_1 A_2 A_3 P_4 \downarrow_6 A_5^7 A_6^{5,6} \mathcal{A}_7$$

A – calculation, P – verification, \mathcal{A} – end.

Formula description in which an algorithm written in the form of one or more formulas, *for example*:

$$LSR = \frac{\sum_1^N L_i P_i}{\sum_1^N P_i}.$$

Hierarchical diagrams show the algorithm graphically, but not as a flow of control but as a division of a solved tasks into subtasks, *for example* (Fig.1.10):

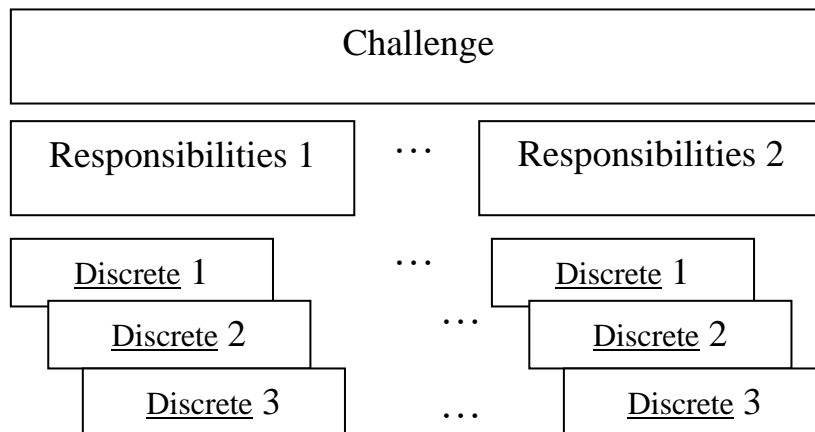


Fig.1.10. Hierarchical diagrams

Nessie-Schneiderman's charts and graphs represent a sequence of executable commands in graphical form, for example (Fig.1.11):

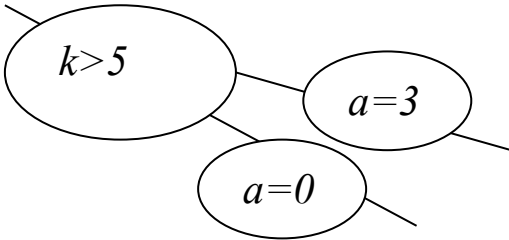


Fig.1.11. Nessie-Schneiderman's graphs

Algorithms are a universal way of documenting algorithms because their appearance is independent of the language in which they are implemented. Another advantage is the high clarity of the schemes. Elements of the algorithm schematics are geometric shapes (symbols), each of which defines a specific action, and flow lines that define the sequence of actions (table 1.2). The type of action and the data (operands) over which the action is performed are written inside the characters in the traditional way (formulas, relations, etc.).

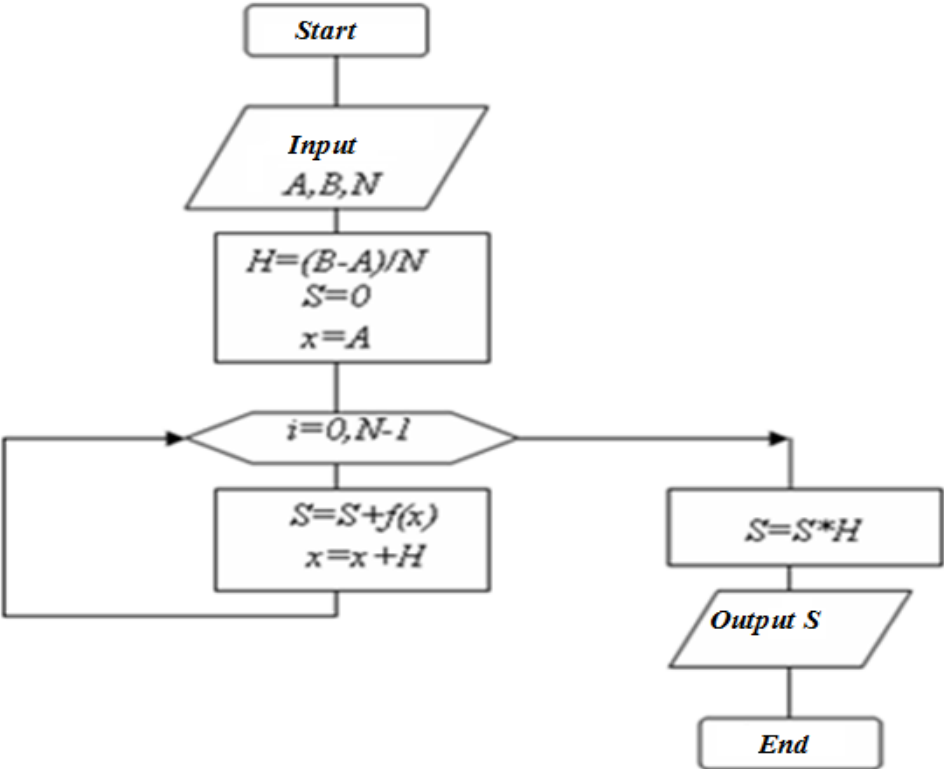
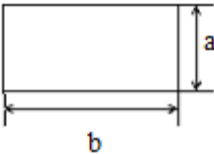









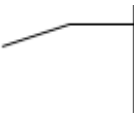


Fig.1.12. Algorithm

In constructing algorithms selected different depth of detail of individual transactions. The diagrams of the algorithms indicate only the flow lines, which have a break and a direction from the bottom up or right to the left. Size (a) the shape of the algorithm (table 1.2) is selected from the series 10, 15, 20 mm. These sizes can increase the number of multiple of 5. Size $b=1.5*a$.

Table 1.2

Name, symbols of symbols (blocks) and functions that they display

Name	Graphic image	Functions
Process		Perform an operation or group of operations
Solution		Selecting the direction of the algorithm according to the specified conditions
Modification		Start of the cyclic process
Input, output		Data input, output
		Data input/output of paper
Subroutine		Using previously created and separately described algorithms
Start, stop		Start, end, stop, sign in, exit in subroutines
Connector (page)		Break the flow line
Connector (interstitial)		Break flow line from another page Connection lines
Flow lines		Explanation to the blocks
Comment		

For easier reading schematics of algorithms, process numbers are assigned to process symbols. The process character numbers are affixed so that they can be read left – right and top – down regardless of the flow direction.

When depicting schematics of algorithms should be guided by a single system of program documentation, which includes the state standards of schematics of algorithms and programs.

1.2.4. Rules of algorithm construction

The construction of algorithms in the flowchart language implies the following rules:

1. The flowchart is built from top to bottom.
2. In any flowchart there is one element corresponding to the beginning and one element corresponding to the end.
3. There must be at least one path from the beginning of the flowchart to any element.
4. There must be at least one path from each flowchart element at the end of the flowchart.

When constructing an algorithm, it is necessary to specify a set of objects with which it will work. The formalized (encoded) representation of these objects is called data. The algorithm starts working with a certain set of data, which is called the input, and as a result of its work produces data that is called the output, i.e. converts input to weekend. Until we have formalized input, we cannot build an algorithm.

The algorithm requires memory. The memory contains the input data with which the algorithm begins to work, intermediate data and output data that are the result of the algorithm.

The memory is discrete, i.e. consisting of separate cells.

A named memory cell is called a variable. In the theory of algorithms, memory sizes are not limited, that is, it is believed that we can provide the algorithm with any amount of memory necessary for work.

1.2.5. Properties of algorithms

Note the basic properties of the algorithms:

- *Discreteness* (discontinuity, separation) – the algorithm should represent the process of solving the problem as the sequential execution of simple (or previously defined) steps. Each action provided by the algorithm is executed only after the execution of the previous one has ended.

- *Determinism* (certainty) – each rule of the algorithm should be clear, unambiguous. After each step, you must indicate which step is performed next, or give a stop command. Due to this property, the execution of the algorithm is mechanical in nature and does not require any additional instructions or information about the task being solved.

Example: Airplane control uses sophisticated algorithms that are performed by a pilot or on-board computer. Each algorithm command determines the unique action of the executor.

- *Efficiency* (finiteness, convergence) – the algorithm should lead to the solution of the problem in a finite number of steps. It is necessary to indicate what to consider as the result of the algorithm. *For example.* The algorithm for adding integers in a decimal number system: firstly, write the numbers in a column; secondly, make up the numbers of the lower rank; thirdly - record the result under horizontal line. If the sum obtained is greater than or equal to the value of the basis of the calculus system (in this case 10), transfer the tens to the higher digit tens; fourthly, repeat paragraphs 2 and 3 for all digits, taking into account the hyphenation from the lower digits (Fig.1.13).

$$\begin{array}{r} 1 \\ 2 5 6 \\ 1 2 8 \\ \hline 3 8 4 \end{array}$$

Fig. 1.13. An example of an algorithm for adding integers in a decimal number system

- *Mass character* – the ability to select input from some set of data (the scope of the algorithm). *For example.* The algorithms of addition, subtraction, multiplication and division can be applied to any numbers in various positional number systems.

- *Formality* – any performer who is able to perceive and execute the instructions of the algorithm can act formally, that is, escape from the content of the task, not delve into its meaning, but only strictly follow the instructions.

- *Efficiency* – the opportunity to solve the task in an acceptable time.

1.2.6. Types of algorithms

The classification of algorithms is carried out depending on the goal, the initial conditions of the problem, ways to solve it, determining the actions of the performer:

- *deterministic* (hard) — sets specific actions denoting them in a single and accurate sequences, thereby providing an unambiguous required or desired result, if process conditions are fulfilled, the task for which the algorithm was developed;

- *probabilistic* (stochastic) — gives a program for solving the problem in several ways or ways leading to the likely achievement of a result;

- *heuristic* — the achievement of the final result of the action program is not unambiguously predetermined, just as the entire sequence of actions is not indicated, all actions of the executor are not identified. For heuristic algorithms include, *for example*, instructions and regulations. These algorithms use universal logical procedures and decision-making methods based on analogies, associations, and past experience in solving similar tasks.

In practice, not just algorithms are needed, but algorithms are good in some broad aesthetic sense. One of the characteristics of the algorithm's quality is the time it takes to execute it. Other characteristics are the adaptability of the algorithm to computers, its simplicity, transparency. As a rule, there are several algorithms for solving the same problem, and it is necessary to decide which one is the best.

1.3. Methodologies and classification of algorithms design methods

Engineering of algorithms and programs – the most critical stage in the life cycle of software products, which determines how much the program being created meets the specifications and requirements from end users. The costs of creating, maintaining and operating software products, the scientific and technical level of development, obsolescence time and much more - all this also depends on the project decisions.

Most of the work of programmers is related to writing source code, testing and debugging programs in one of the programming languages.

Algorithms and programs are the objects of the author's right to intellectual property.

Different programming languages support different programming styles (methodologies or programming paradigms). Part of the programming is to choose one of the languages that is most suitable for solving an existing problem.

Different algorithmic languages require the programmer to have a different level of attention to detail when implementing the algorithm, which often results in a compromise between simplicity and performance (or between the programmer's time and the user's time).

The design methods of algorithms and programs are very diverse, they can be classified according to various criteria, the most important of which are:

- extent of automation of project works;
- accepted methodology of the development process.

By the degree of *automation* of the design of algorithms and programs, we can distinguish:

- methods of traditional (non-automated) engineering;
- methods of computer-aided engineering (CASE-technology and its elements).

Non-automated engineering of algorithms and programs are mainly used in the development of software products that are small in complexity and structural complexity, which do not require the participation of a large number of developers. The complexity of the developed software products is usually small, and the software products

themselves are mainly applied in nature. If these restrictions are violated, the productivity of developers decreases markedly, the quality of development falls, and, paradoxically, the labor costs and the cost of the software product as a whole increase.

Automated engineering of algorithms and programs has arisen with the need to reduce the cost of design work, shorten their execution time, create standard "blanks" of algorithms and programs that are repeatedly replicated for various developments, coordinate the work of a large team of developers, standardize algorithms and programs.

Automation can cover all or individual stages of the software product life cycle, while the work of the stages can be isolated from each other or make up a single complex, performed sequentially in time.

As a rule, an automated approach requires technical and software "re-equipment" of the labor of the developers themselves (powerful computers, expensive software tools, as well as advanced training for developers, etc.).

Computer-aided engineering of algorithms and programs is only possible for large firms specializing in the development of a certain class of software products that occupy a stable position in the software market.

Programming methodology – is a set of ideas, concepts, principles, methods and means that determines the style of writing, debugging and maintaining programs.

Let's list the main *methodologies (paradigms) of programming* along with their inherent types of abstractions:

- *procedure-oriented* – a set of principles, technologies and tools based on the procedural or algorithmic organization of the structure of the program code (procedures - methods, functions, programs);

- *object-oriented* – allow you to connect data with processing procedures into a single whole – classes and objects (Object Pascal, C ++, Java, etc.);

- *logically oriented* – a formalized description of the problem so that the solution follows from the compiled description. It is indicated what is given and what is required to be received, and the search for a solution to the problem is assigned to the computer (Lisp, Prolog);

- *parallel programming* – a method of organizing computer computing in which programs are developed as a set of interacting computing processes that work in parallel (simultaneously).

There are other paradigms.

Structural programming

How to develop a good algorithm? There are a number of useful techniques that mainly relate to structural programming methods.

Classical *procedural programming* requires the programmer to provide a detailed description of how to solve the problem, i.e., the formulation of the algorithm and its special notation. In this case, the expected properties of the result are usually not indicated. The basic concepts of the languages of these groups are the operator and the data. With a procedural approach, operators are combined into groups – procedures.

Structural programming as a whole does not go beyond this direction; it only additionally captures some useful techniques of programming technology.

The beginning of the development of structural design of algorithms and programs falls on the 60s. Structural design is based on consistent decomposition, focused structuring into individual components.

Structural programming, most clearly expressed in the Pascal language (PASCAL), arose during the development of a procedurally-oriented approach, embedded in the historically first Fortran programming language (FORTRAN).

The basis of structured programming techniques make the following provisions:

1. A complex task is broken down into smaller, functionally better-managed tasks. Each task has one input and one output. In this case, the control flow of the program consists of a set of elementary subtasks with a clear functional purpose.

2. The simplicity of the control structures used in the task. The logical structure of the program can be expressed by a combination of three basic structures: follow, branch, and loop.

3. An elaborated algorithms depicted in flowcharts.

4. Using basic structures, the use of an unconditional transition can be completely eliminated, which is an important sign of structural programming.

Structural design methods are a set of technical and organizational principles. Consider some of them.

The first method involves reducing a difficult task to a sequence of more simple tasks. If the problem is complex, then the step-by-step

method should be used, when the general structure of the algorithm is first thought out and fixed without detailed elaboration of its individual parts, but only the basic structures are used. Blocks requiring further detail are indicated by dashed lines. Next, separate blocks are made, without detailing the previous steps. So every step of the way, we're dealing with a simpler task.

The method described is called *top-down* or *private purpose method*.

The second method of algorithm development is known as *the lifting method*. The lifting algorithm begins by making an initial assumption or calculating the initial solution of the problem. Then begins as fast as possible the upward movement from the initial to the best decisions. When the algorithm reaches a point from which it is no longer possible to move up, it stops. Unfortunately, we cannot always guarantee that the final solution obtained by the lifting algorithm will be optimal.

The third method is known as *working backwards*, that is, starting from a goal or decision and moving toward the initial formulation of the problem. Then, if these actions are reversible, we move back from the formulation of the problem to the solution.

Let us dwell in more detail on the most effective, from our point of view, method of the structural approach - top-down design. It is based on the idea of levels of abstraction, which become levels of modules in the program being developed. This allows the programmer to first focus on determining what needs to be done in the program, and only then decide how to do it.

In a top-down development, the initial problem to be solved is divided into a series of sub-tasks subordinated in content to the main task. Such a breakdown is called *drillthrough* or *decomposition*.

At the next stage, these tasks, in turn, are divided into smaller subordinate subtasks, and so on, to the level of relatively small subtasks that require small modules to solve.

Module – is a sequence of logically related operations, designed as a separate part of the program.

Using modules has the following *advantages*:

- the ability to create a program by several programmers;
- simplicity of design and subsequent modifications of the program;

- simplification of program debugging – search and elimination of errors in it;
- the possibility of using ready-made libraries of the most common modules.

Depending on the object of structuring distinguish:

- *functionally-oriented methods* – sequential decomposition of a task or a holistic problem into separate, fairly simple components that have functional certainty;
- *data structuring methods*.

For functionally oriented methods, first of all, given data processing functions are taken into account, in accordance with which the composition and logic of work (algorithms) of individual components of the software product are determined. With a change in the content of the processing functions, their composition, the corresponding information input and output, redesign of the software product is required.

The main emphasis in the structural approach is on modeling data processing processes.

For data structuring methods, analysis, structuring and creation of data models are carried out, in relation to which the necessary composition of functions and processing procedures is established. Software products are closely related to the structure of the processed data, the change of which is reflected in the processing logic (algorithms) and necessarily requires redesigning the software product.

The structural approach uses:

- diagrams of data flows (information and technological schemes), which show the processes and information flows between them, taking into account the “events” that initiate the processing processes;
- integrated domain data structure (infological model, ER-diagrams);
- decomposition diagrams – the structure and decomposition of goals, management functions, applications;
- structural schemes – the architecture of a software product in the form of a hierarchy of interconnected software modules with identification of the relationships between them, the detailed logic of data processing by software modules (flowcharts).

For a complete picture of the software product, descriptive textual information is also required.

Traditional software development approaches have always emphasized the differences between data and its processing. So, information modeling-oriented technologies first specify the data, and then describe the processes using this data.

Structural approach technologies are oriented, first of all, to data processing processes with the subsequent establishment of the necessary data for this and organization of information flows between related processes.

To a greater extent, software products are not a monolith and have a construction structure (architecture) —the composition and interconnection of software modules.

Module – it is an independent part of the program that has a specific purpose and provides specified processing functions autonomously from other program modules.

Thus, the software product has an *internal organization* or *internal structure* formed by interconnected software modules. This is true for complex and multifunctional software products, which are often called *software systems*.

Structuring programs is carried out primarily for the convenience of development, programming, debugging and changes to the software product. As a rule, software systems of great algorithmic complexity are developed by a team of developers (2 - 15 or more people). Manage the development of programs in the application of industrial technologies for the production of programs is possible only on a scientific basis.

The structuring of software products pursues the main objectives:

- to distribute the work among the performers, ensuring their acceptable loading and the required terms for the development of software products;
- build calendar schedules for design work and coordinate them in the process of creating software products;
- control labor costs and the cost of design work, etc.

The structural “partition” of programs into individual components serves as the basis for the selection of tools for their creation, although there is an inverse effect – the choice of tools for a software developer determines the types of software modules.

When creating software products, reusable modules are allocated, their *typification* and *unification* are carried out, due to which the time and labor costs for developing the software product as a whole are reduced.

Some software products use modules from ready-made libraries of standard routines, procedures, functions, objects, data processing methods.

Among the many modules are distinguished (Fig.1.14):

- head module – controls the launch of a software product (exists in the singular);
- control module – provides a call to other modules for processing;
- work modules – perform processing functions;
- service modules and libraries, utilities – perform service functions.

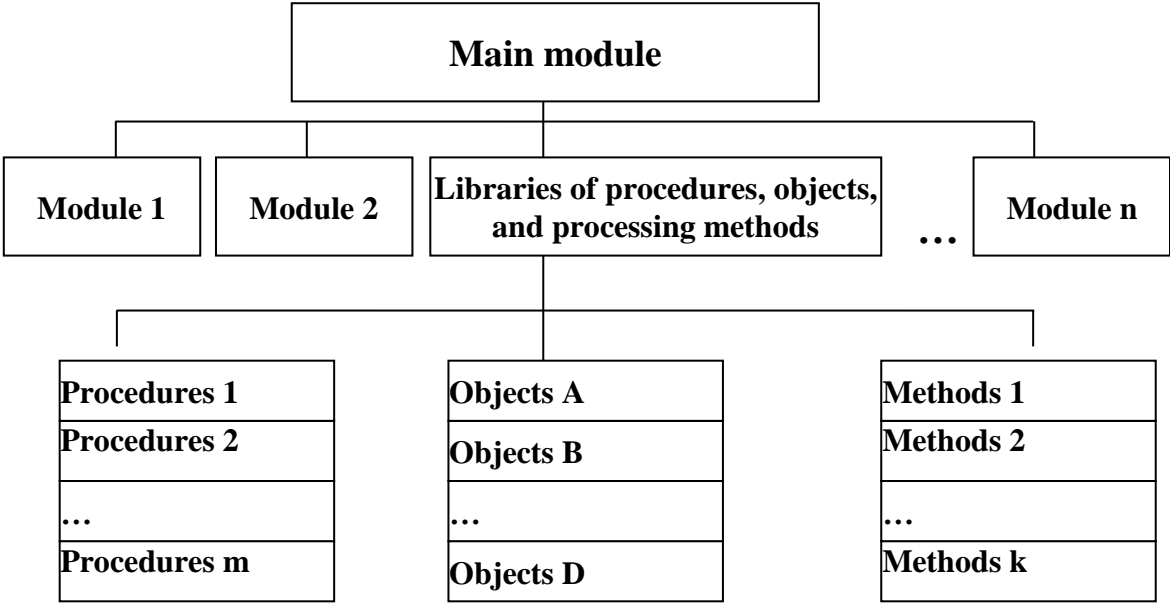


Fig. 1.14. Software product structure

In the process, the software activated the necessary software modules.

Control modules specify the sequence of calls to execute the next module. Information communication of the modules is ensured through the use of a common database or inter-module data transmission through exchange variables.

Each module can be executed as a self-stored file.

For the functioning of a software product, it is necessary to have complete software modules.

Structurally complex software products are developed as software packages, and most often they are of an applied, *application program packages*, is a system of programs designed to solve problems of a particular class.

The components of the APP are united by common data (a database), are informationally and functionally interconnected, and have the property of being *systematic*, i.e. a combination of programs has a new quality that is not available for a particular APP component. The structure of the APP is, as a rule, multimodular.

The private purpose method looks very attractive. But, like most common methods for solving problems or developing algorithms, it is not always easy to transfer to a specific task. A rational choice of simpler tasks is more a matter of art or intuition than of science. Moreover, there is no general set of rules for defining the class of tasks that can be solved with this approach.

Private goals can be set when the following questions are answered:

Is it possible to solve private problems?

Is it possible to ignore some of the remaining tasks by ignoring some conditions?

Is it possible to solve the problem on a case-by-case basis?

Is it possible to develop an algorithm that solves and satisfies all the conditions of the problem but whose output is limited by some subset of all the output?

Is there anything about a task that we didn't understand well enough?

If you try to delve deeper into some of the features of the problem, will it help you come to a solution?

Are we faced with a similar problem, whose solution is known?

Is it possible to change her decision to solve our problem?

Is it possible that this task is equivalent to a known unresolved task?

Today, object-oriented programming has become complementary to structural programming, creating the basis for the development of modern software systems, and declarative programming, expressed in

two different approaches – *functional* and *logical*, has been opposed to it when solving certain classes of problems.

Object-oriented technologies include specialized programming languages and user interface development tools.

At the same time, data and processes are combined into logical entities – objects that have the ability to inherit the characteristics (methods and data) of one or more objects, thereby ensuring the reuse of program code. This leads to a significant reduction in the cost of creating software products, increases the efficiency of the life cycle of software products (the duration of the development phase is reduced). When the program is executed, a message is sent to the object, which initiates the processing of the object data.

If the algorithm is developed, it can be handed to a person for execution. Exactly following the instructions of the algorithm, the performer, even unfamiliar with the solution, will get the solution.

In some cases, the desire to remain anything within the framework of the structural-modular approach leads to unreasonable complication of the algorithm and loss of its naturalness and clarity. In this case, clarity should be preferred.

1.4. The problem of choice and analysis of complexity of algorithms

The algorithm should:

- be easy to understand, translate into code and debug;
- efficiently use computer resources and run as quickly as possible;
- the program should be executed only a few times: the cost of the program is optimized by the cost of writing (rather than executing) the program;
- solving a problem requires significant computational costs: the cost of executing a program can exceed the cost of writing a program, especially with repeated execution.

Note that the first two points contradict each other.

For practice, it's not enough to know that the problem is algorithmically solvable because computer resources (RAM and processor time) are limited, you should choose the most efficient one from equivalent algorithms.

Efficiency is a generalized characteristic for comparing algorithms. Algorithm A1 is more efficient than algorithm A2 if algorithm A1 is executed in less time and (or) requires less computer resources (RAM, disk space, network traffic, etc.).

An efficient algorithm should satisfy the requirements of an acceptable execution time and reasonable resource consumption.

The combination of these characteristics makes up the concept of *algorithm's complexity*. With an increase in the execution time of the algorithm and (or) the resources involved, the complexity increases. Thus, the concepts of efficiency and complexity are inverse relative to each other.

The time of implementation of the algorithm (program execution) is a function of:

- the duration of the input source information in the program;
- time complexity of the algorithm (program);
- quality of the compiled code of the executable program;
- machine instructions that are used for program execution;
- data size;
- source data.

The characteristic of the algorithm, which reflects the time spent on its implementation, is called *time's complexity*.

The characteristic of the algorithm, which reflects the computer resource costs for its implementation, is called *capacitive complexity*.

In quantitative and qualitative estimates of the algorithm associated with the determination of complexity, two approaches are used – *practical* and *theoretical*.

The practical approach is connected with really executed algorithms on specific physical devices. It is characterized by parameters that can be measured and recorded.

The time's complexity with this approach can be expressed in time units (for example, milliseconds) or the number of processor cycles spent on the algorithm.

The capacitive complexity can be expressed in bits (or other units of information), the minimum hardware requirements needed to execute the algorithm, etc.

The main (simplest) operations, analogues of machine instructions:

- assignment of a variable to a value;

- arithmetic operation;
- comparison of two numbers;
- array indexing;
- linking the object;
- function call;
- return from the function.

To evaluate the time complexity, it is necessary – mathematically estimate the time of its execution by counting operations:

- write the algorithm in the form of code of one of the developed programming languages (for example, VB or C ++);
- translate the program into a sequence of machine instructions (for example, bytecodes used in a virtual machine);
- determine for each machine command i the time t_i necessary for its execution;
- find for each machine command i the number of repetitions n_i of command i during the execution of the algorithm;
- calculate the sum of the products $t_i * n_i$ of all machine instructions, which will be *the execution time* of the algorithm.

A practical assessment is not an absolute indicator of the effectiveness of an algorithm. The quantitative values obtained with this approach depend on many factors, such as:

- technical characteristics of the components that make up the computing system. So, the higher the clock speed of the processor, the more elementary operations per unit of time can be performed;
- characteristics of the software environment (the number of running processes, the task scheduler operation algorithm, operating system operating features, etc.);
- selected programming language for the implementation of the algorithm. A program written in a high-level language is likely to run slower and require more resources than a program written in low-level languages with direct access to hardware resources;
- the experience of the programmer who implemented the algorithm. Most likely, a novice programmer will write a less effective program than a programmer with experience.

Thus, an algorithm executed in the same computing system for the same input data can have different quantitative estimates at different points in time. Therefore, the *theoretical approach* to determining complexity is more important.

Theoretical complexity characterizes the algorithm without reference to specific equipment, software, and implementation tools.

In this case, the *time's complexity* is expressed in the number of operations, the clock cycles of the Turing machine, etc.

The capacitive complexity is determined by the amount of data (input, intermediate, output), the number of cells involved on the Turing machine ribbon, etc.

In a theoretical approach to evaluating efficiency, it is believed that the algorithm is executed on some idealized computer for which the execution time of each type of operation is known and constant. It is also believed that the resources of such a computer are infinite, therefore, capacitive complexity is usually not determined with a theoretical approach.

The number of instructions (operations) performed on an idealized computer is selected as the time characteristic of complexity.

Quantitative estimates obtained by the theoretical approach may also depend on a number of the following factors:

- volume of input data. The larger it is, the longer it will take to execute the algorithm;
- selected method of solving the problem. For example, for a large amount of input data, the quick sort algorithm is more effective than the bubble sort algorithm, although it leads to the same result.

The execution time of an algorithm usually depends on the amount of input data, which is understood as the dimension of the problem being solved.

The best way to compare the effectiveness of algorithms is to compare their orders of complexity; applicable to temporal and spatial complexity.

Example: the task of calculating the polynomial values

$$P_n(x) = a_n \cdot x^n + a_{(n-1)} \cdot x^{(n-1)} + \dots + a_i \cdot x^i + \dots + a_1 \cdot x^1 + a_0$$

Specified:

- array of coefficients $A = \{A[0], A[1], \dots, A[n]\}$
- variable value x

Calculate polynomial value $S = P_n(x)$

Version 1 (Fig. 1.15)

For each term, except a_0 to erect x to the set grade by sequential multiplication and then multiply by a factor.

Assessment of time complexity:

- calculation i term ($i=1\dots n$) requires i multiplications;
- in all $1 + 2 + 3 + \dots + n = n(n+1)/2$ multiplications;
- required n additions and one initial value assignment operation a_0 .

The time's complexity of the algorithm:

$$T(n) = n(n+1)/2 + n + 1 = n^2/2 + 3n/2 + 1 \text{ operation.}$$

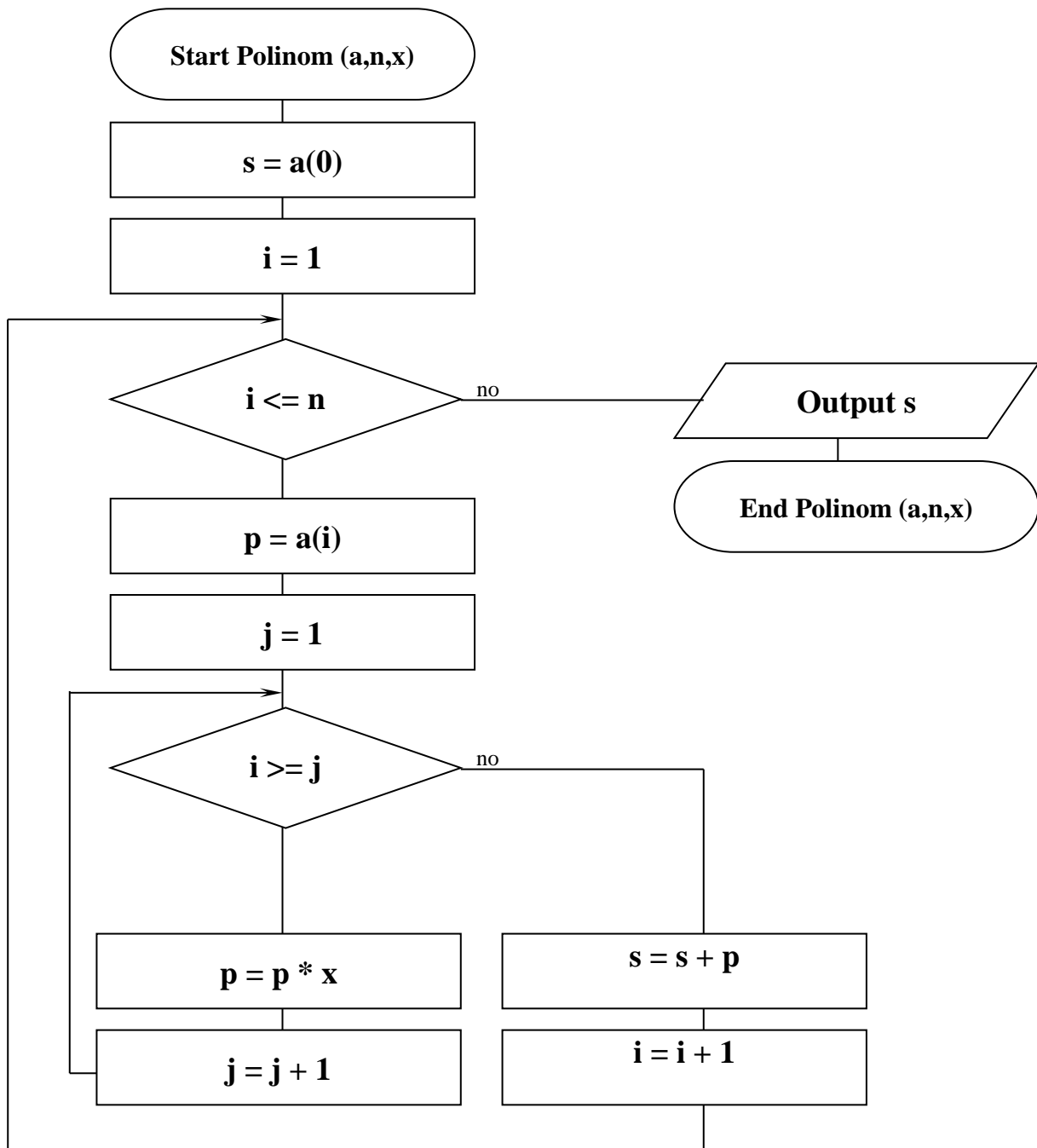


Fig. 1.15. Algorithm version 1

Version 2 (Fig. 1.16).

Horner's scheme

$$P_n(x) = a_0 + x(a_1 + x(a_2 + \dots (a_i + x(a_{n-1} + a_n x)) \dots))$$

$$S_0 = a_n,$$

$$S_1 = S_0 x + a_{n-1},$$

$$S_2 = S_1 x + a_{n-2},$$

...

$$S_i = S_{i-1} x + a_{n-i},$$

...

$$S_n = S_{n-1} x + a_0,$$

$$P_n(x) = S_n$$

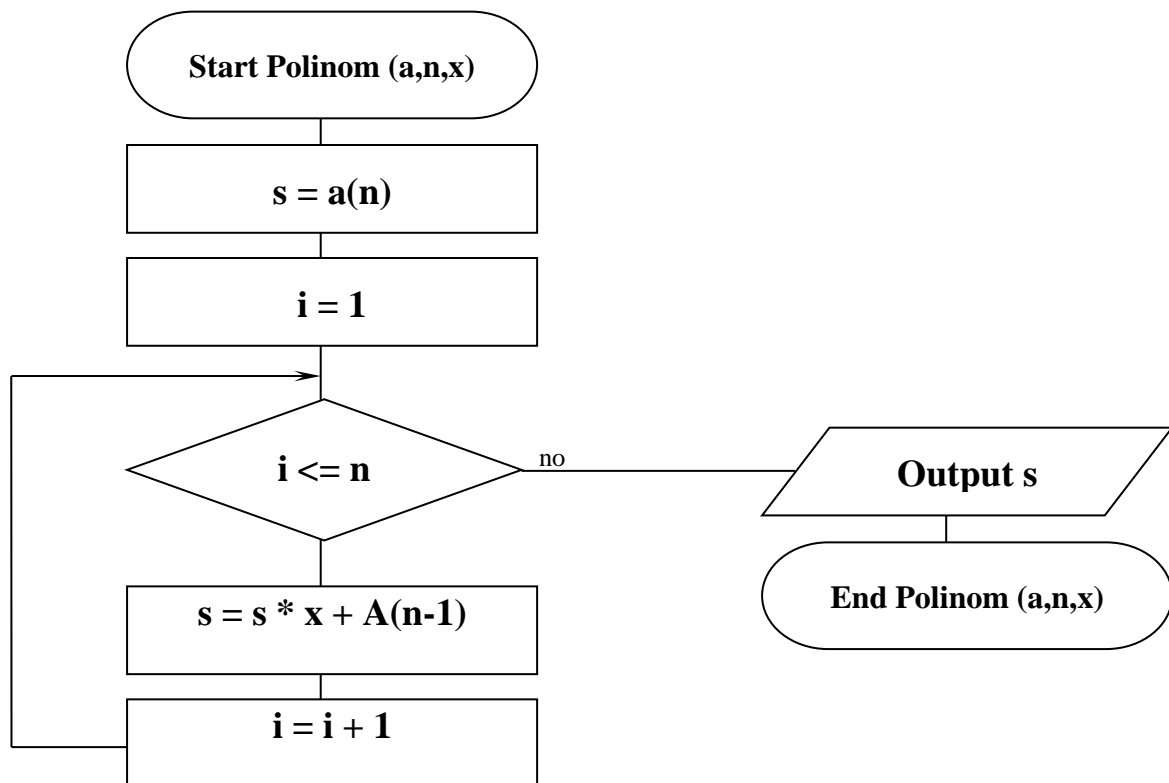


Fig. 1.16. Algorithm version 2

Assessment of time's complexity:

- to calculate S_i required **1** multiplication and **1** addition;
- in total, such an iteration is performed n times.

The time's complexity of the algorithm:

$$T(n) = n \text{ multiplications} + n \text{ additions} = 2n \text{ additions}$$

1.5. Typical structures of algorithms

In drawing algorithms must remember that the same problem can be successfully solved using algorithms that differ from each other. Which algorithms to choose? The requirements of compactness of the algorithm and ease of its understanding will be natural. These requirements are satisfied if the so-called *typical algorithmic structures* are used in the developing of the algorithm.

Typical (basic) structures of algorithms are a limited set of blocks and standard ways of joining them to perform typical action sequences.

Based on the concepts of structural approach, *algorithm of any complexity can be expressed by using basic structures.*

1. Following. As a component is found in almost all algorithmic constructions. Follow-up is characterized by the fact that the actions are performed one after the other: symbols for determining variables, calculations of intermediate and final values, output of calculation results are placed sequentially. This procedure is called natural (Fig. 1.17).

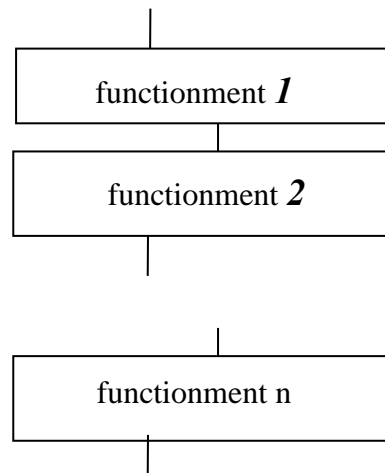


Fig. 1.17. Algorithmic following

2. Branching. An algorithmic design in which, depending on the value of the source data and the result of the condition check, one of two alternative action sequences (branches) is selected.

If the specified condition is fulfilled (result is "yes" or *True*) one branch is selected, if it is not fulfilled (result is "no" or *False*) - another.

To write conditions are selected or that the steps used *logic operations* and *comparison operations*.

Branching is also used when it is necessary to choose one of several alternatives (not two, but three or more variants). To do this, make up a combination of several industries.

The algorithms underlying this structure are called *branching*. Each branch can be of any degree of complexity, but may not contain actions at all.

For each specific set of output data branching is reduced to following.

The construction comes in three main ways:

1. *The branched* algorithmic construction consists of two branches (Fig. 1.18 a).
2. *Bypass* is a separate branching case where one branch contains no action (Fig. 1.18 b).
3. *Multiple Choice* is a generalization of a branch in which one of the actions is selected depending on the value of the control variable (Fig. 1.18 c).

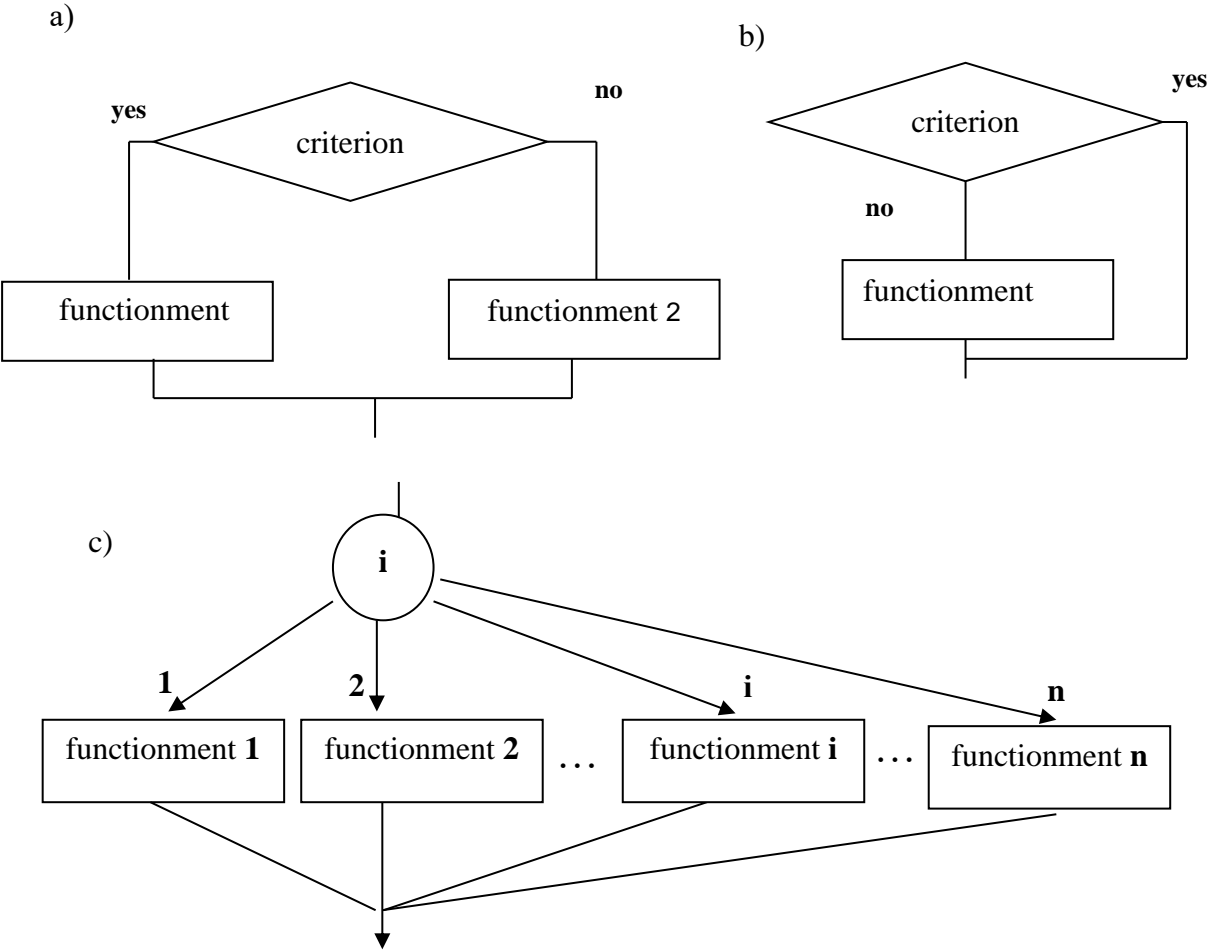


Fig. 1.18 Basic branching structure:
a) branching; b) detour; c) multiple choice

Full branching allows you to organize two branches in the algorithm (or otherwise), each of which leads to a general point of merging, so the algorithm continues regardless of which path was chosen.

Incomplete branching is a special case of branching and assumes the presence of some algorithm actions on only one branch (that), the second branch is absent, ie for one of the results of the check it is not necessary to perform any actions, the control immediately goes to the point of merger.

Multiple choice is a generalization of the branch where depending on the value of the control variable i runs one of several actions.

3. Repeat. Algorithmic construction, which is a sequence of actions performed repeatedly, under the truth of a given condition. Such algorithms are called *cyclic* or *cycles*.

A *loop body* is a sequence of actions that are performed repeatedly (in a loop).

Condition - the condition of the end of the cycle. If the result of checking the condition True - loop ends, if False - loop continues.

If the opposite condition *change* (for example $x > 0 \rightarrow x \leq 0$), *умова закінчення* циклу перетвориться в *умову продовження*.

Prerequisite cycle. The condition is checked early before the loop body is executed, so the loop body may not perform at all (Fig. 1.19 a).

A loop with a condition. The condition check is done at the end after the loop body is executed, so it is always executed at least once (Fig. 1.19 b).

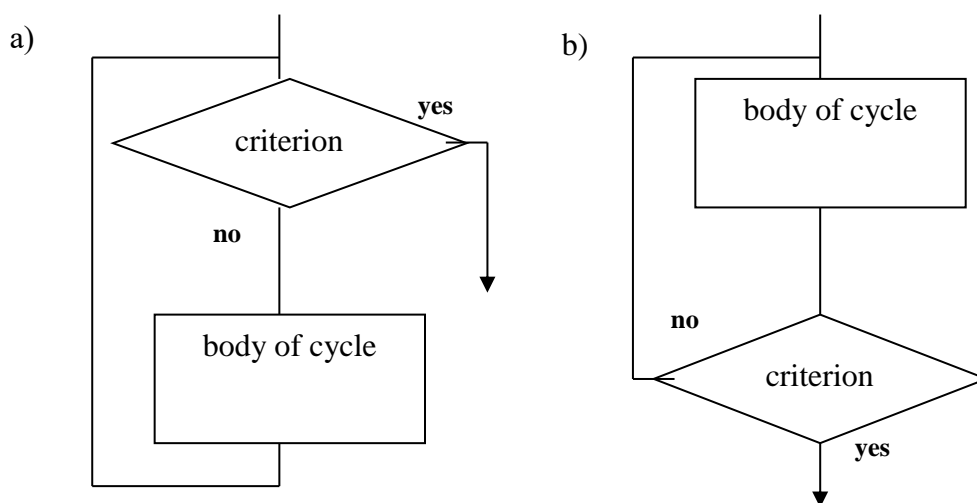


Fig. 1.19. Repeat: a) precondition loop; b) postcondition loop

The peculiarity of all these structures is that they have one input and one output, they can be connected to each other in any sequence. When using basic structures created more complex (combined) structure are allowed to use two approaches:

- *connect* one structure to another, creating a sequence of structures. (Fig. 1.20);
- *replace* the functional blocks of each of the basic structures with the nested structures. (Fig.1.21-1.23).

These rules allow you to build algorithms of any degree of complexity, developing them not only "wide" but also "deep". The resulting algorithms have a clear and clear structure.

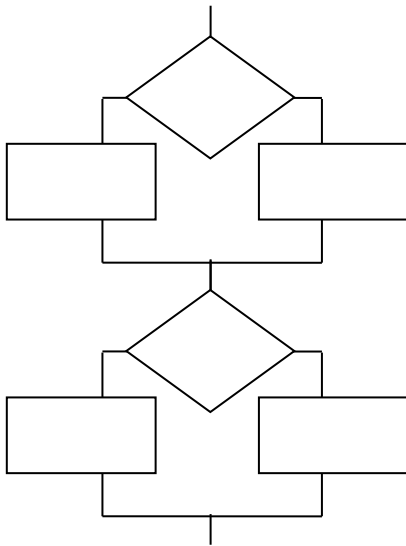


Fig. 1.20. Following branching

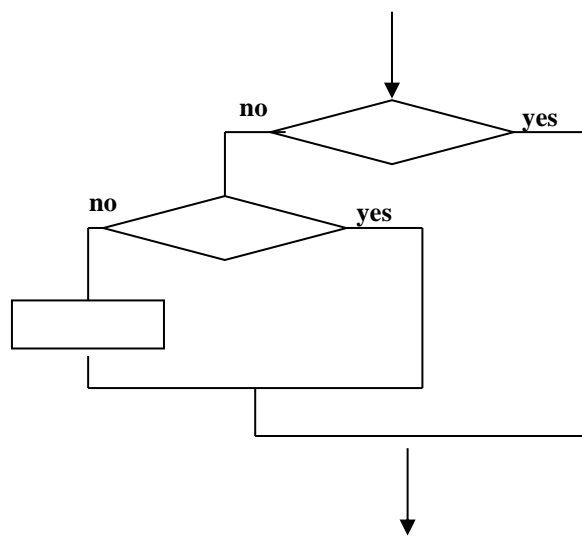


Fig. 1.21. Condition in condition

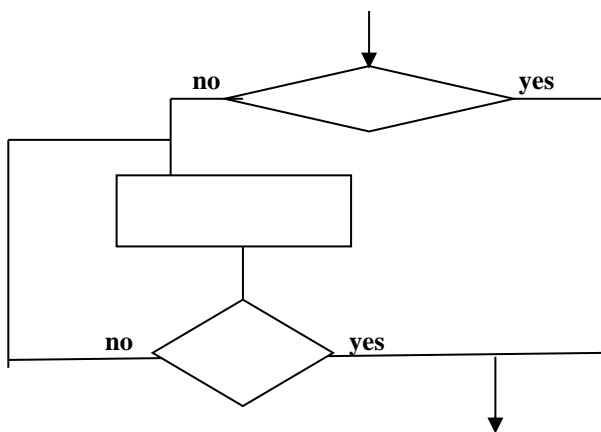


Fig. 1.22. Loop in condition

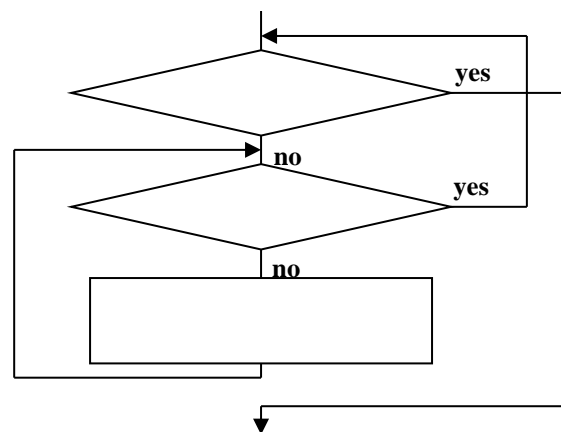
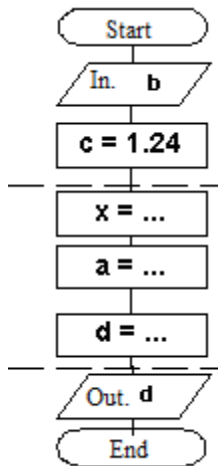




Fig. 1.23. Loop in loop

Control questions and tasks

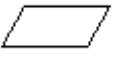
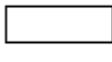

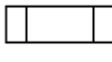
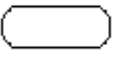
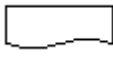
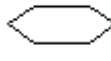
1. What are the steps for solving computer problems?
2. Define the term "algorithm" and call its properties.
3. Define the term "program" for solving the problem on a computer.
4. List the tools for describing the algorithms.
5. X is:



- 1) intermediate calculated value;
 - 2) final calculated value;
 - 3) basic calculated value.
6. Symbol  (process) in the algorithm determines:
- 1) process input/output the information;
 - 2) process of calculation of intermediate and final values;
 - 3) the process of output information on paper;
 - 4) the process of determining the beginning (end) solve the task;
 - 5) the process of using the results of subtask algorithms.

7. Symbol  (solution) in the algorithm determines:
- 1) process input/output the information;
 - 2) process of calculation (solution) of intermediate and final values;
 - 3) the process of selection the action, the implementation of which depends on the condition.

8. Enter symbol that defines the process of input/output of paper:

- 1)  ; 2)  ; 3)  ; 4)  ;
- 5)  ; 6)  ; 7)  .

9. Enter symbol that defines the process of calculation of intermediate and final values:

- 1)  ; 2)  ; 3)  ; 4)  ; 5)  .

10. The stage of solution of the task "Writing the program" involves:

- 1) verbal or schematic description of the final sequence of actions (instructions), the task that leads to the decision;
- 2) description of algorithm in algorithmic language;
- 3) analysis of the obtained results.


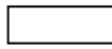

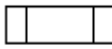



11. The data can be divided into "Input" and:

- 1) Output;
- 2) Variable;
- 3) Constant.

12. *Calculated data (based on changes in the process of solving the task) are divided into:

- 1) Output;
- 2) Input;
- 3) Variable;
- 4) Constant.

13. Enter symbol that defines the process of using the results of subtasks:

- 1)  ; 2)  ; 3)  ; 4)  ;
5)  ; 6)  ; 7)  .

14. The stage of solution of the task "Development of an algorithm" involves:

- 1) a verbal or schematic description of the final sequence of actions that leads to the solution of the task;
- 2) description of algorithm in algorithmic language;
- 3) description of the program code, preparation of help documentation for the user.

15. The stage of solution of the task "Task statement and choice of numerical method" NOT involves:

- 1) collection and processing of input information (forming of input data);
- 2) formalization of the task;
- 3) setting goals and priorities;
- 4) selection and reasoning of mathematical model;
- 5) folding the final sequence of actions that leads to the solution of the task;
- 6) definition of initial information, intermediate and final calculations.

16. The attribute of the algorithm is deterministic (certainty) involves:

- 1) the result or the message about the inability to get a result for the specified input data;
- 2) splitting the algorithm into simple actions;
- 3) possibility to solve many problems by one algorithm;
- 4) possibility of receiving many results with certain input data;
- 5) possibility to select input from multiple data;
- 6) the uniqueness of the result of the input data.

17. *Enter a term that does NOT specify the type of algorithm:

- 1) lineal;
- 2) ramified;
- 3) loop;
- 4) insert;
- 5) combined;

18. The attribute of the algorithm is discretion involves:

- 1) a verbal or schematic description of the final sequence of actions that leads to the solution of the task;
- 2) the uniqueness of the result of the input data;
- 3) the result or a message that the result cannot be obtained;
- 4) splitting the algorithm into simple actions;
- 5) possibility to solve many problems by one algorithm;
- 6) possibility of receiving many results with certain input data;
- 7) possibility to select input from multiple data.

19. The property of the algorithm results involves:

- 1) a verbal or schematic description of the final sequence of actions leading to the decision;
- 2) the uniqueness of the result of the input data;
- 3) the result or a message that the result cannot be obtained;
- 4) splitting the algorithm into simple actions;
- 5) possibility to solve many problems by one algorithm;
- 6) possibility of receiving many results with certain input data;
- 7) possibility to select input from multiple data.

20. The property of the mass algorithm involves:

- 1) the result or the message about the inability to get a result for the specified input data;
- 2) splitting the algorithm into simple actions;
- 3) possibility to solve many problems by one algorithm;
- 4) possibility of receiving many results with certain input data;
- 5) possibility to select input from multiple data.

UNIT 2. LINEAR COMPUTING PROCESSES

As stated earlier, the linear computing process is characterized by the fact that the steps it breaks are performed one by one without checking the conditions. Another name is *the basic following structure* [7].

For implement such a process it is necessary to:

First, *generate the input*.

These actions can be done in user input mode, such as "enter b, y".

Or assign values to variables constant, *for example*: $b=4.7$, $y=-3.1$, where 4.7 i -3.1 the corresponding real numbers – constant (Fig. 2.1).

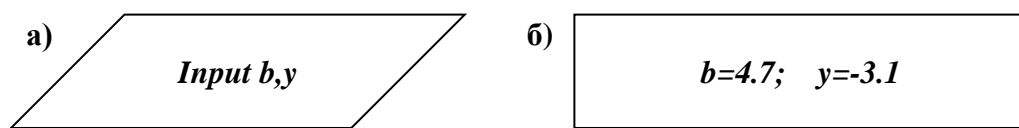


Fig. 2.1. Definition of input data: a) variable; b) constant

Second, *write down the formulas to calculate*.

It is necessary to observe the sequence of computation expressions. It should be remembered that the implementation of the program with non-user-assigned numeric values, numeric variables will be assigned zero values (Fig. 2.2), *for example*:

A rectangular box containing the mathematical formula $x=(a+b)/a^2$.

Fig. 2.2. Calculations

Thirdly, *organize the output of the results* after the calculation. To control the results can output data, which were introduced for calculations, *for example*: constant values (Fig. 2.3).

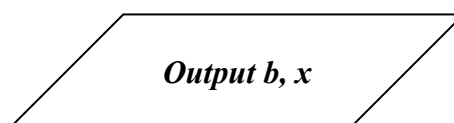


Fig. 2.3. Output constant **b** and estimated **x**

Consider the task that has a linear structure.
 For example, algorithm of calculation function:

$$Y = 3 + \frac{\ln(x^2 + 1)}{2x^4 + a + 5},$$

where $x = \frac{a + b}{b^2},$

$$a = 3.45$$

b – any number, not equal 0 (to avoid dividing by zero).

Will create the algorithm. In creating the algorithm (Fig. 2.4) we will use *the method of calculating the parts*. To do this, denote, for example, the numerator by the symbol Y1, the denominator by Y2.

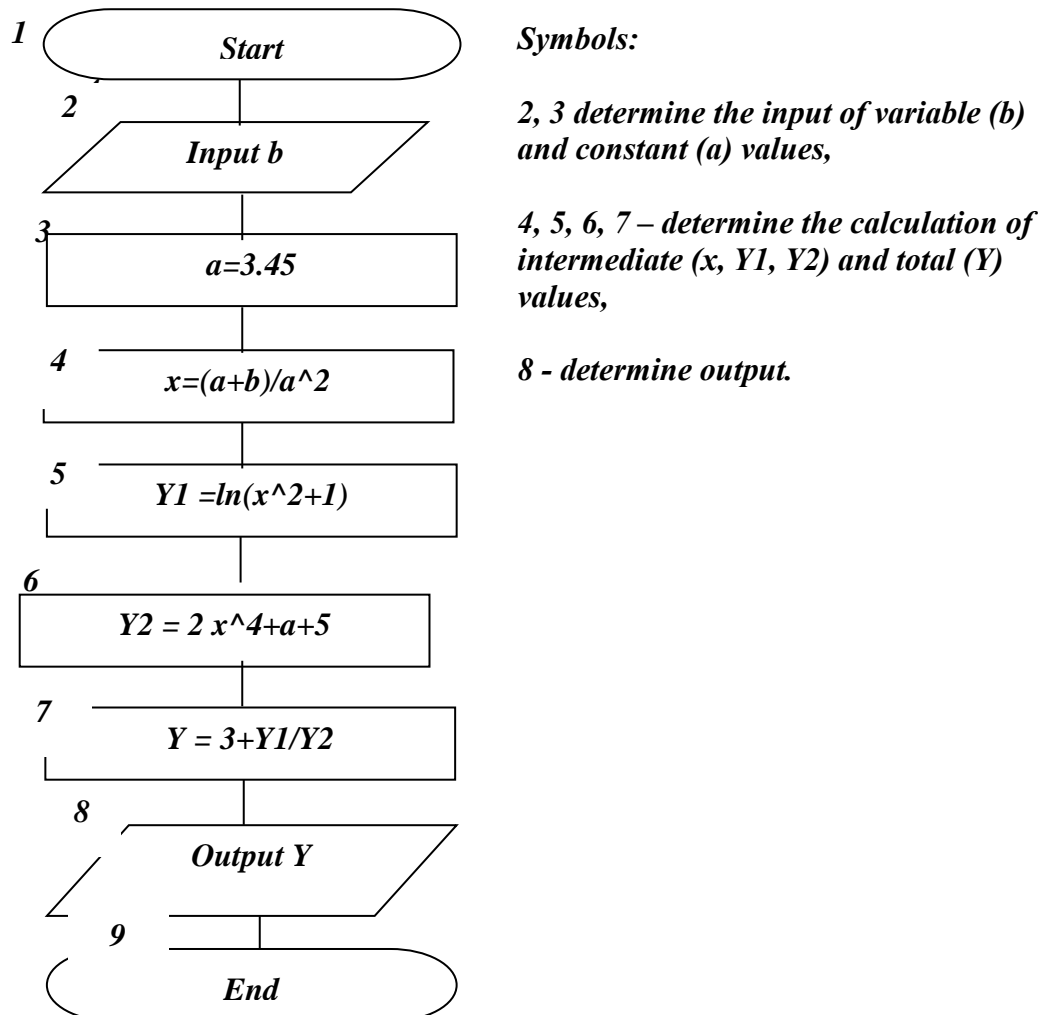


Fig. 2.4. Algorithm

All symbols of the algorithm are arranged sequentially. The values of X, Y1, Y2, Y are stored in memory with their corresponding names (identifiers).

It should be noted that the given algorithm is one of many possible options for solving this task. For example, you can swap the symbols (blocks) 2 and 3; calculate Y without dividing the process into separate parts, immediately after writing down the X formula for calculating Y, method of calculating the numerator (Y1) and the denominator (Y2) – symbols 5 and 6 can also be any.

Note the symbols 2 and 3. The result of their implementation is identical: in the corresponding memory cell will be placed numerical value. However, if the symbol 2 indicates the input of any number from the keyboard (the schema identifies the variable ID and but not the value), then the symbol 3 - assigning a variable a specific value (the schema identifies the variable ID and value itself).

The symbol 8 (output) is depicted and filled in like the symbol of the input. Such technology allows to avoid errors by making algorithms.

Control questions and tasks

1. Define the term of "linear computing process".
2. Give instructions for types that make computing a linear process.
3. Specify the steps that lead to the calculation the arithmetic

expression: $x + 2 * y - \sin^2(y)$

- 1) 1 – calculating sine function,
2 – exponentiation,
3 – multiplication,
4 – addition,
5 – subtraction;
- 2) 1 – exponentiation,
2 - calculating sine function,
3 – multiplication,
4 – addition,
5 – subtraction;
- 3) 1 - calculating sine function,
2 – exponentiation,
3 - addition,
4 – multiplication,
5 – subtraction.

4. Specify the steps that lead to calculating arithmetic expressions:

$$y = e^{a+2b^c}$$

- 1) 1 – exponentiation b to 3,
2 – multiplying the previous result by 2,
3 – addition a,
4 – calculating the exponent;
- 2) 1 – calculating the exponent,
2 – exponentiation b to 3,
3 – multiplying the previous result by 2,
4 – addition a;
- 3) 1 – multiply 2 and b,
2 – exponentiation b to 3,
3 – addition a,
4 – calculating the exponent.

5. Specify the sequence of processing variables

$$z = 2f^k \quad a = \sqrt[7]{d+k^2} \quad k = 3 \quad d = z \sin 2\pi, \quad \text{where } f \text{ – any value}$$

- 1) a,k,f,z,d; 2) k,f,z,d,a; 3) k,f,a,z,d; 4) k,z,f,d,a.

6. Specify the sequence of processing variables.

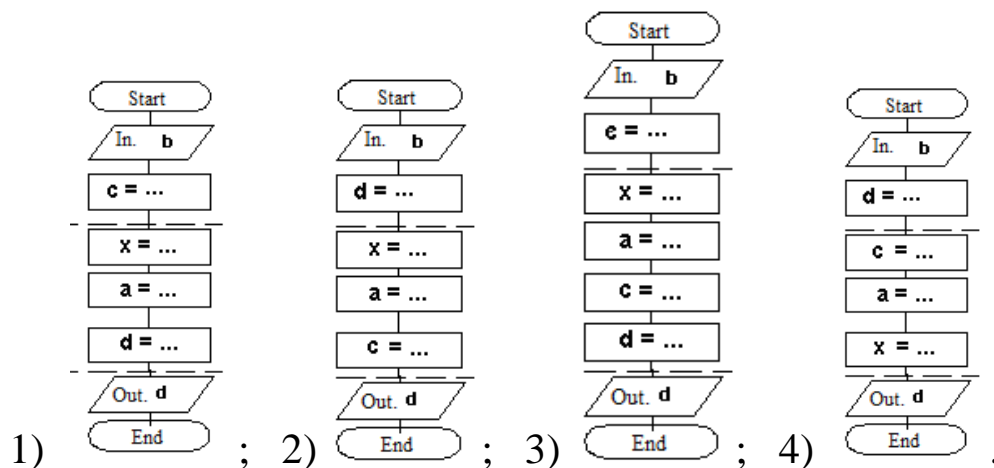
$$x = e^{y+1} \quad z = 2a + 24 \quad a = x^3, \quad \text{where } y \text{ – any value}$$

- 1) x,a,y,z; 2) y,x,a,z; 3) a,z,y,x; 4) x,y,z,a.

7. *Specify a scheme for calculating values

$$x = |b| + 33 \quad d = 3 + \frac{e^{a+1}}{r} \quad a = \ln(x) - c \quad c = 1.24,$$

where b – any value



UNIT 3. OVERALL COMPUTING PROCESSES

The branching is an algorithm in which the choice of action depends on the fulfillment of certain conditions and values of the input data or intermediate results.

These algorithms have several options for calculations, each represented by a separate branch of computing. The branch is selected by the control part of the algorithm.

The control part – "solution" symbols (otherwise called logical or conditional) are connected for a given set of input data or intermediate results guaranteed execution of actions on a single branch of the algorithm. Selecting of variant given logical attitude or logical expression.

Logical attitude – sequential writing of constants, variables, arithmetic expressions, combined by attitude signs:

$$=, <, >, \neq, \leq, \geq.$$

The attitude is compared by two numeric or symbolic values, and if the result is "True", produced 1, otherwise – 0 (Fig. 3.1).

The control part

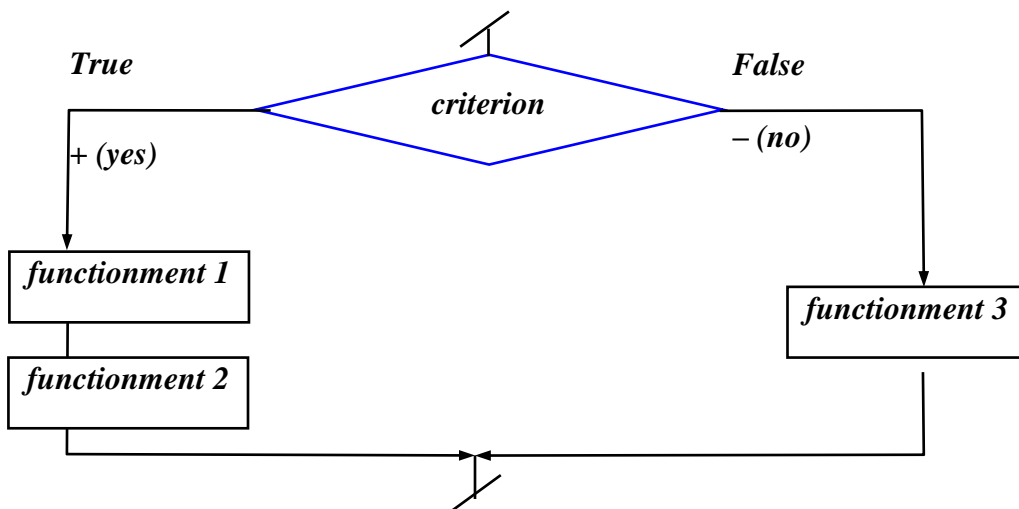


Fig. 3.1. Logical attitudes generally

For example: $56=41$ False result (0) (Fig. 3.2a),
 $15\leq 20$ True result (1) (Fig. 3.2b).

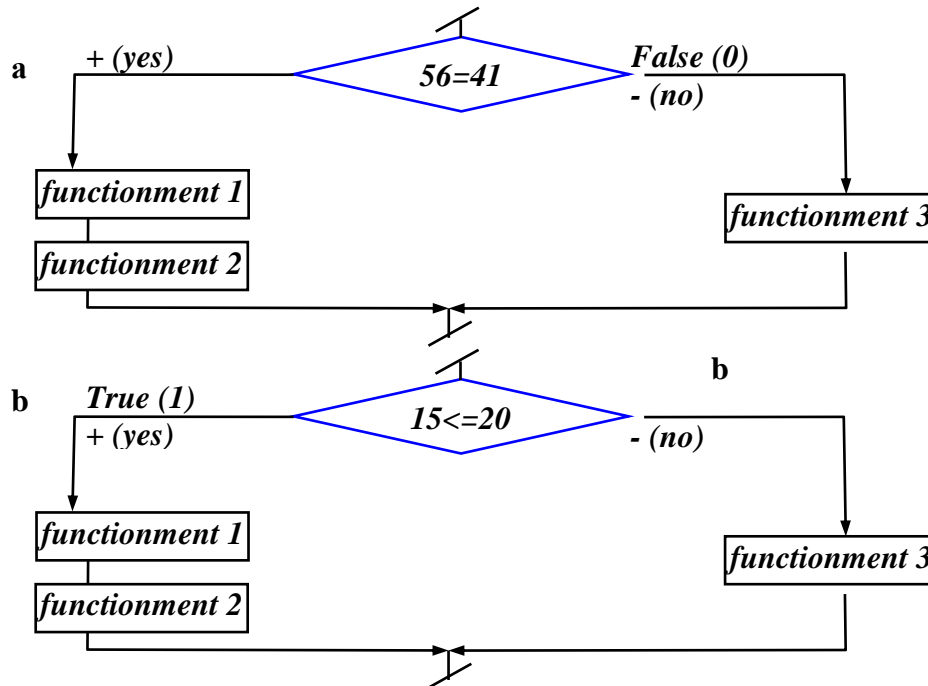


Fig. 3.2. Logical attitude

a – to analyze conditions $56=41$; b – to analyze conditions $15\leq 20$

If necessary, you can change the condition to the opposite, for example: to split information stream into two parts – negative values x and positive values x and zero (Fig. 3.3a). For this condition can be used a schemes shown in the Fig. 3.3 b, and the Fig. 3.3 c.

Logical expression – sequential recording of logical attitudes, combined by signs of logical operations:

logical multiplication (conjunction) or operation "AND" is indicated by a sign \wedge (Fig. 3.4);

logical addition (disjunction) or operation "OR" is indicated by a sign \vee (Fig. 3.5);

logical negation (inversion) or operation "NOT" is indicated by a sign \neg .

The result of logical operations given in the truth table and logic scheme (Fig. 3.6).

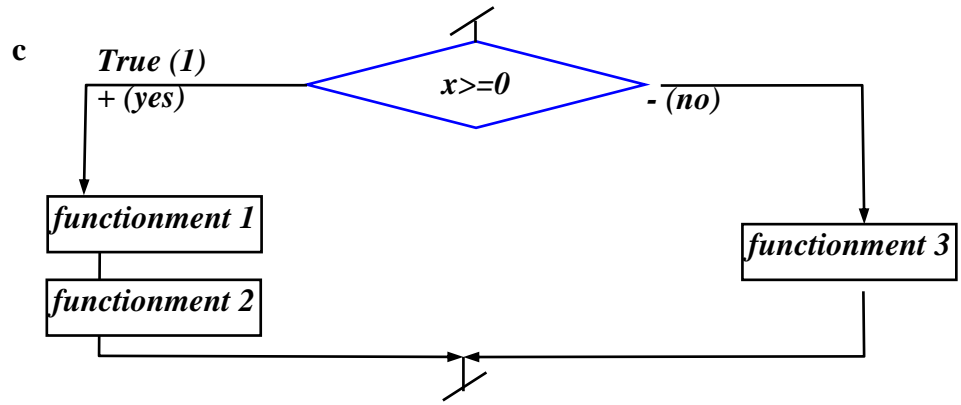
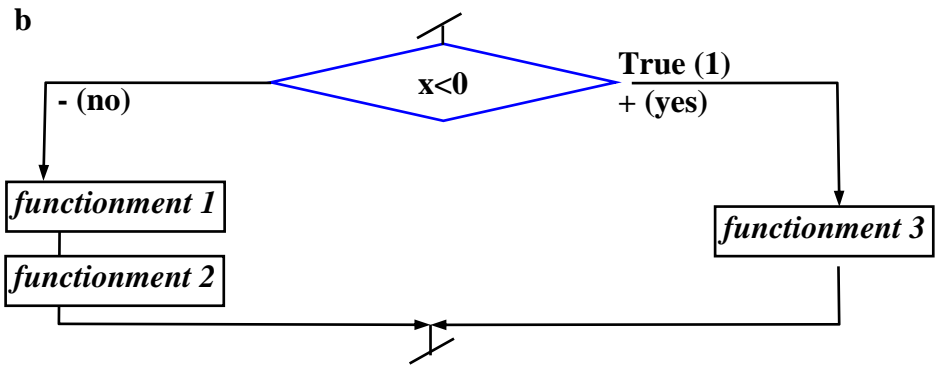
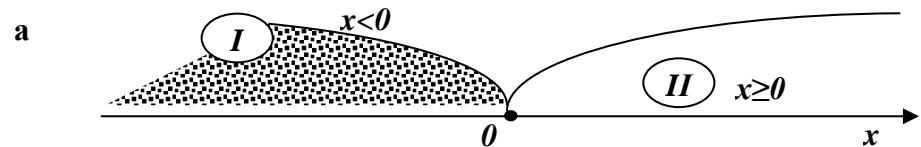


Fig. 3.3. Methods to implement a logical attitude

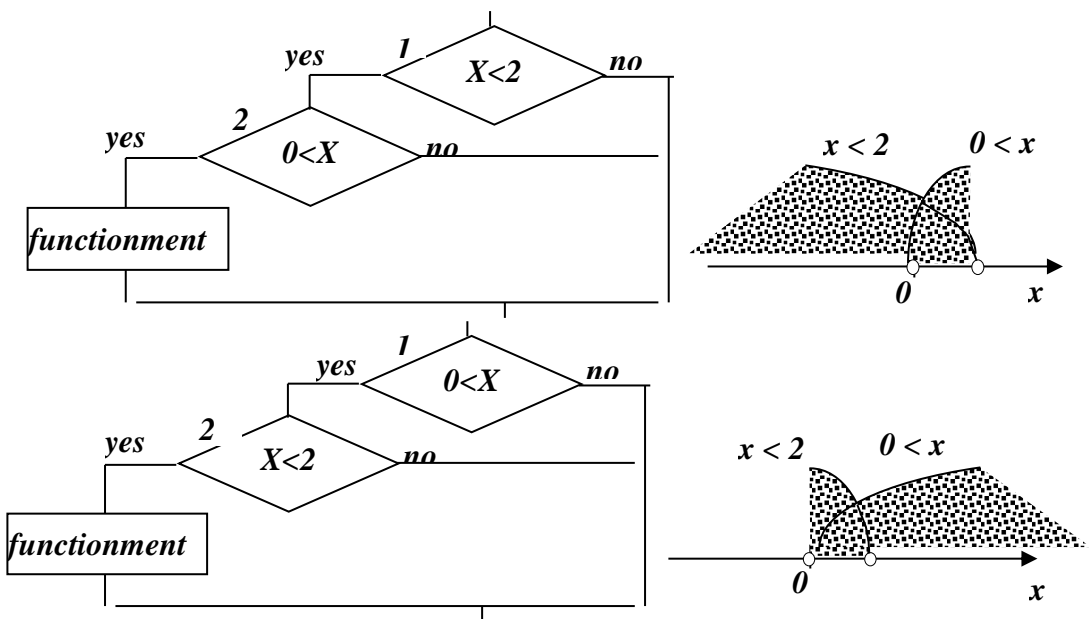


Fig. 3.4. Variants of conjunction for logical expression $0 < X < 2$

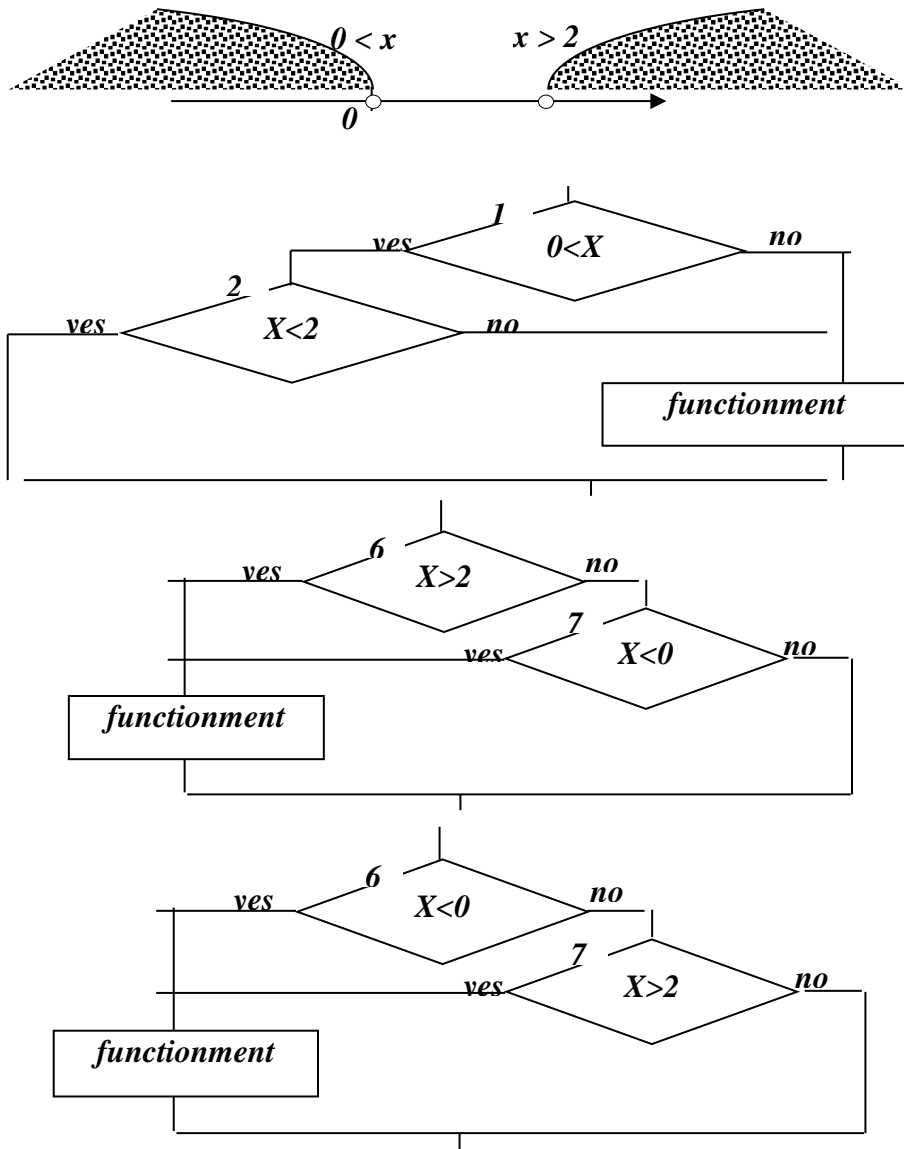


Fig. 3.5. Variants of disjunction for logical expression $0 < X$ or $X > 2$

Table 3.1

The truth table of conjunction, disjunction, inversion

$F = \neg a$	
a	F
0	1
1	0

$F = a \wedge b$		
a	b	F
0	0	0
0	1	0
1	0	0
1	1	1

$F = a \vee b$		
a	b	F
0	0	0
0	1	1
1	0	1
1	1	1

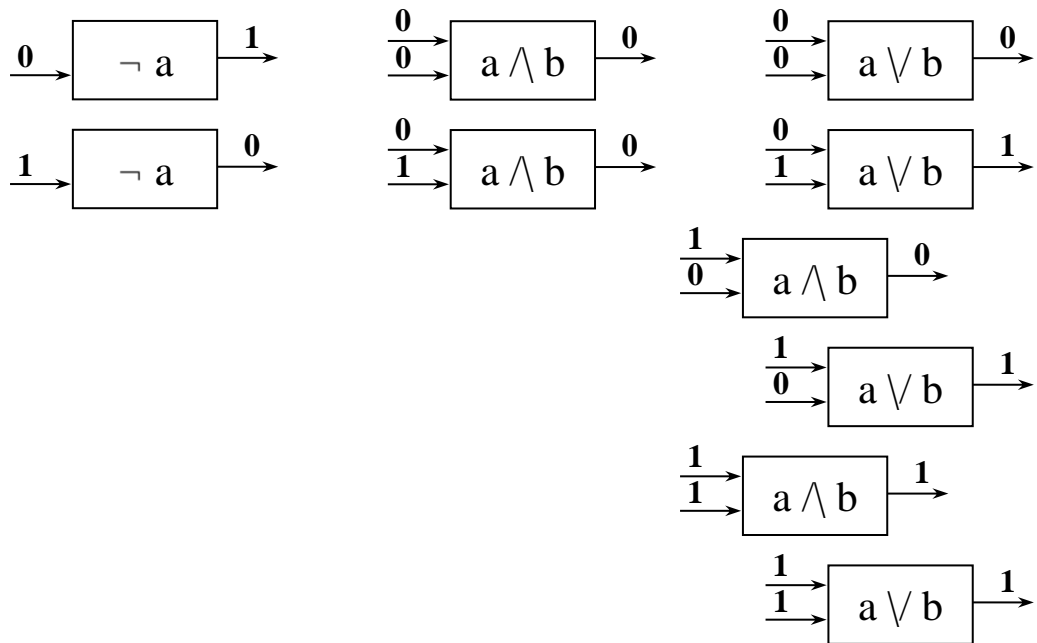


Fig. 3.6 Logical scheme

Logical operations are performed in order of decreasing priority as follows: \neg , \wedge , \vee . For example (Fig. 3.7):

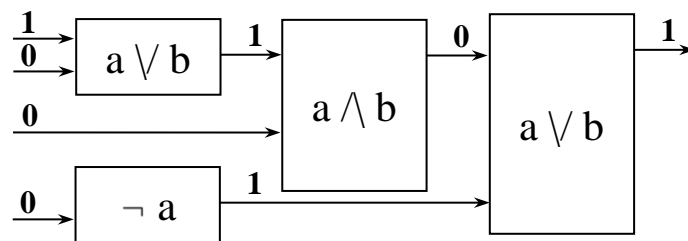


Fig. 3.7. Complex logic scheme

For example: if $a=0$; $b=1$; $c=0$ logical expression $\neg a \wedge b \vee (\neg c) \wedge a \vee b \wedge (\neg a) \wedge (\neg b)=1$ (Fig. 3.8).

You can change the priority using parentheses. For example, if $a=0$; $b=0$; $c=0$; $d=1$ logical expressions take on different meanings:

$$\neg a \wedge b \wedge (c \vee d) = \neg 0 \wedge 0 \wedge (0 \vee 1) = 1 \wedge 0 \wedge 1 = 0$$

$$\neg a \wedge b \wedge c \vee d = \neg 0 \wedge 0 \wedge 0 \vee 1 = 1 \wedge 0 \vee 1 = 1$$

If a=0; b=1; c=0 то

1) make the operations inversion (\neg)

get

2) make the operations conjunctions (\wedge)

отнимаемо

3) make the operations disjunctions (\vee)

get

$$\begin{array}{l}
 (\neg 0) \wedge 1 \vee (\neg 0) \wedge 0 \vee 1 \wedge (\neg 0) \wedge (\neg 1) \\
 (\underline{\neg 0}) \wedge 1 \vee (\underline{\neg 0}) \wedge 0 \vee 1 \wedge (\underline{\neg 0}) \wedge (\underline{\neg 1}) \\
 \underline{1} \wedge 1 \vee \underline{1} \wedge 0 \vee 1 \wedge \underline{1} \wedge \underline{0} \\
 \underline{1} \wedge 1 \vee \underline{1} \wedge 0 \vee \underline{1} \wedge \underline{1} \wedge \underline{0} \\
 \underline{1} \vee \underline{0} \vee \underline{1} \wedge \underline{0} \\
 \underline{1} \vee \underline{0} \vee \underline{0} \\
 \underline{1} \vee \underline{0} \vee \underline{0} \\
 \underline{1} \vee \underline{0} \vee \underline{0} \\
 \underline{1}
 \end{array}$$

Fig. 3.8. Example of calculating logical expressions

Consider examples of making algorithms that have a branched structure.

For example 1. Variants of algorithms are shown in the Fig. 3.9–3.11.

$$y = \begin{cases} 2 * x + 3, & \text{if } x < 0 \\ 4 * x - 7, & \text{if } x \geq 0 \end{cases} \quad \text{where } x - \text{any number}$$

The output data states two conditions ($X < 0$) i ($X \geq 0$), but in the algorithm it is enough to write down one of them (*for example* $X < 0$). This symbol has two outputs: for symbol 4, if $X < 0$, and for symbol 5 (alternative), if $X \geq 0$. The second condition may be: otherwise (in other cases).

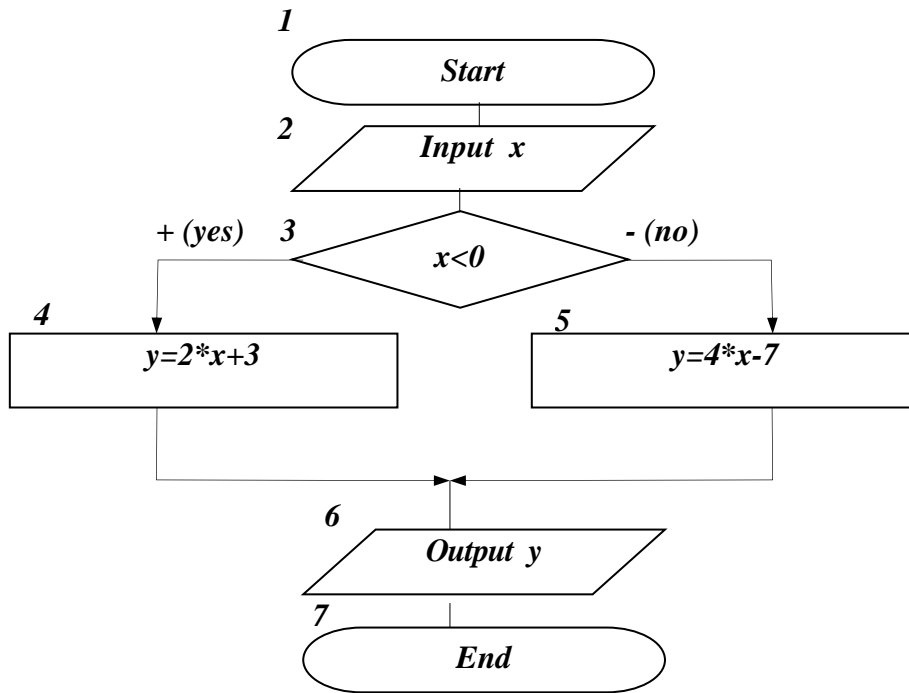


Fig. 3.9. A branching algorithm with a simple condition (logical attitude version 1)

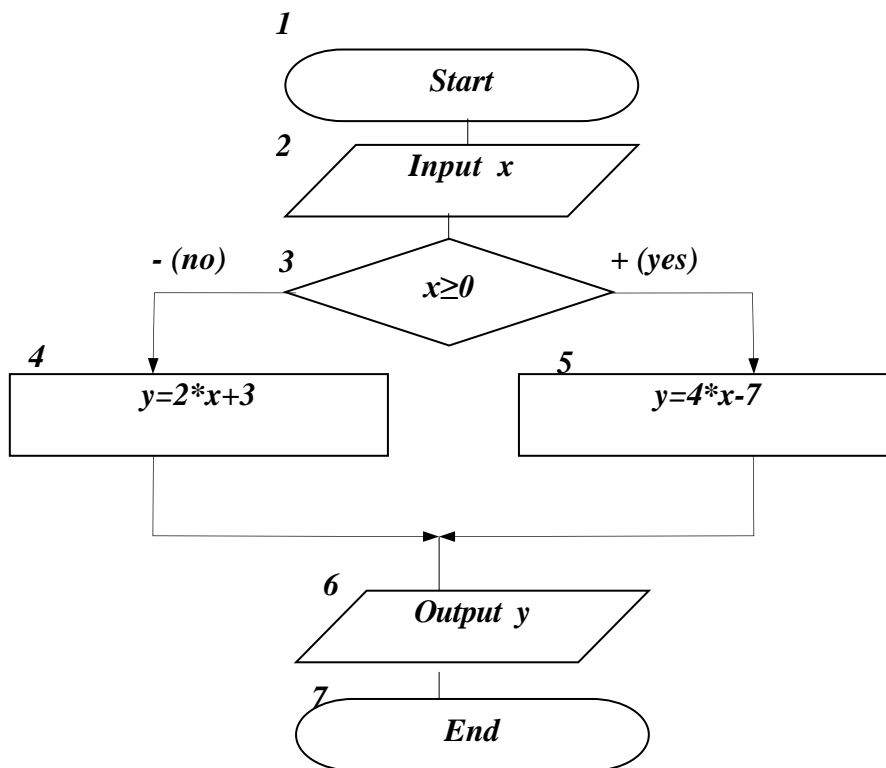


Fig. 3.10. A branching algorithm with a simple condition (logical attitude version 2)

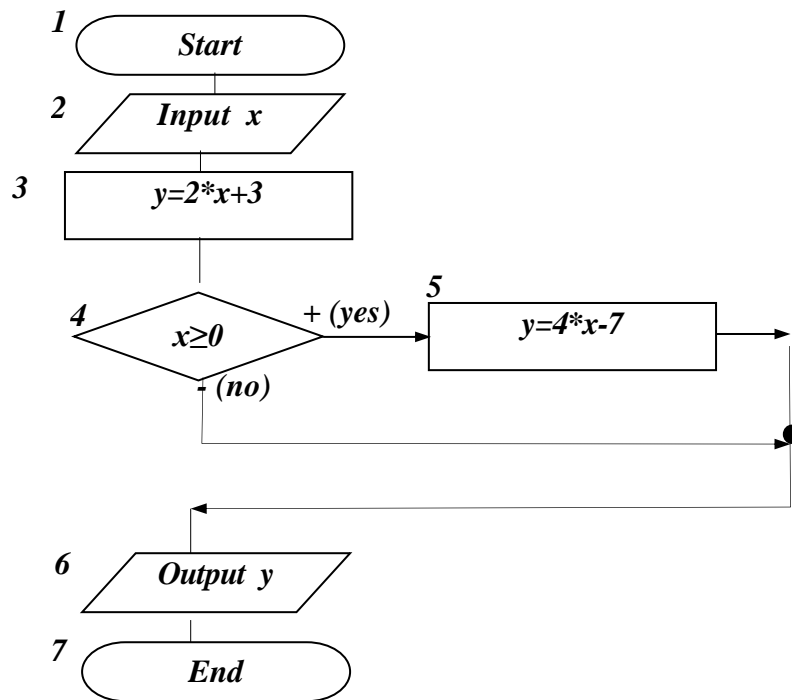


Fig. 3.11. A branching algorithm with a simple condition (logical attitude version 3)

For example 2. Variants of algorithms are shown in the Fig. 3.12-3.13.

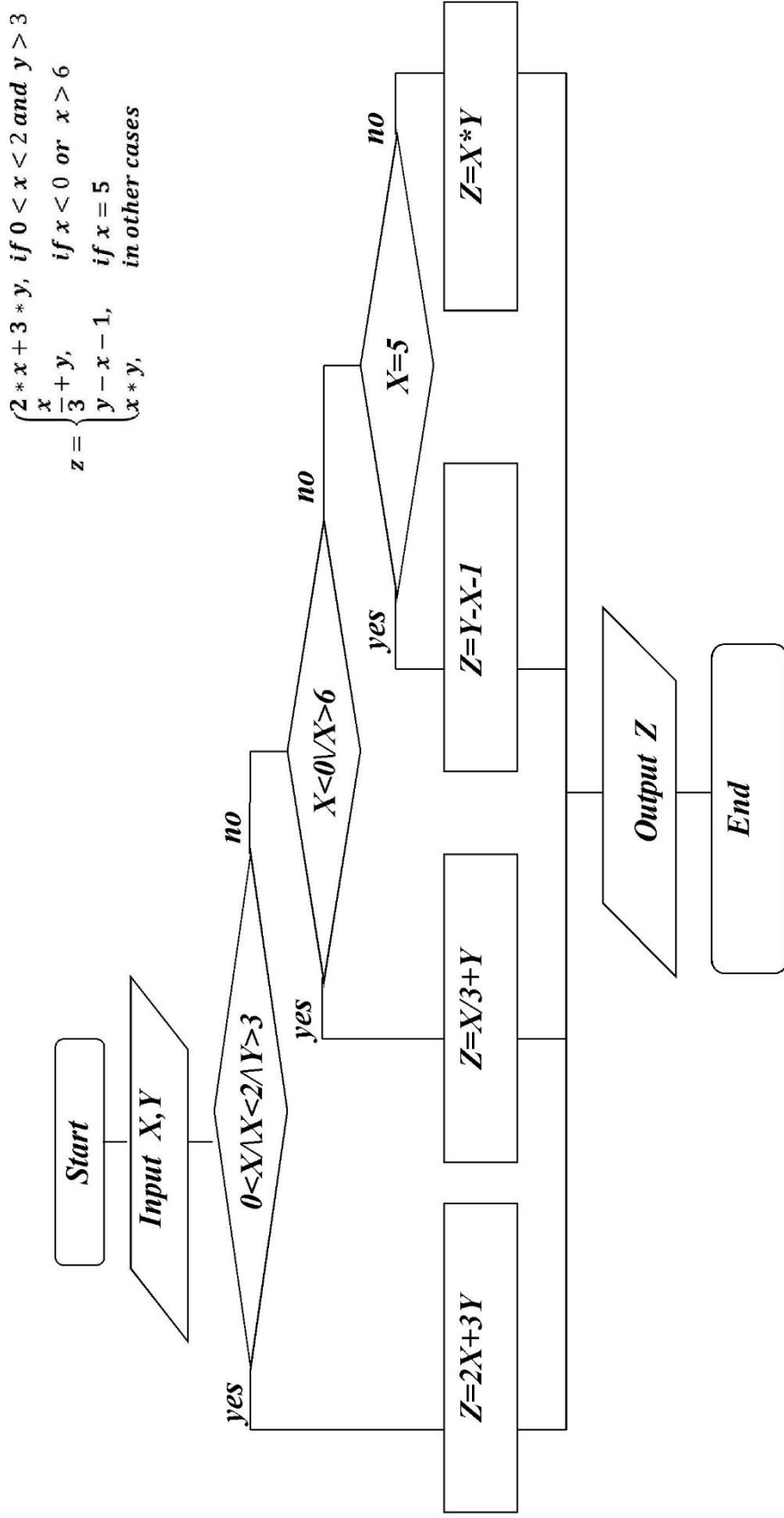


Fig. 3.12. Branched algorithm with a complex condition (logical expression variant 1)

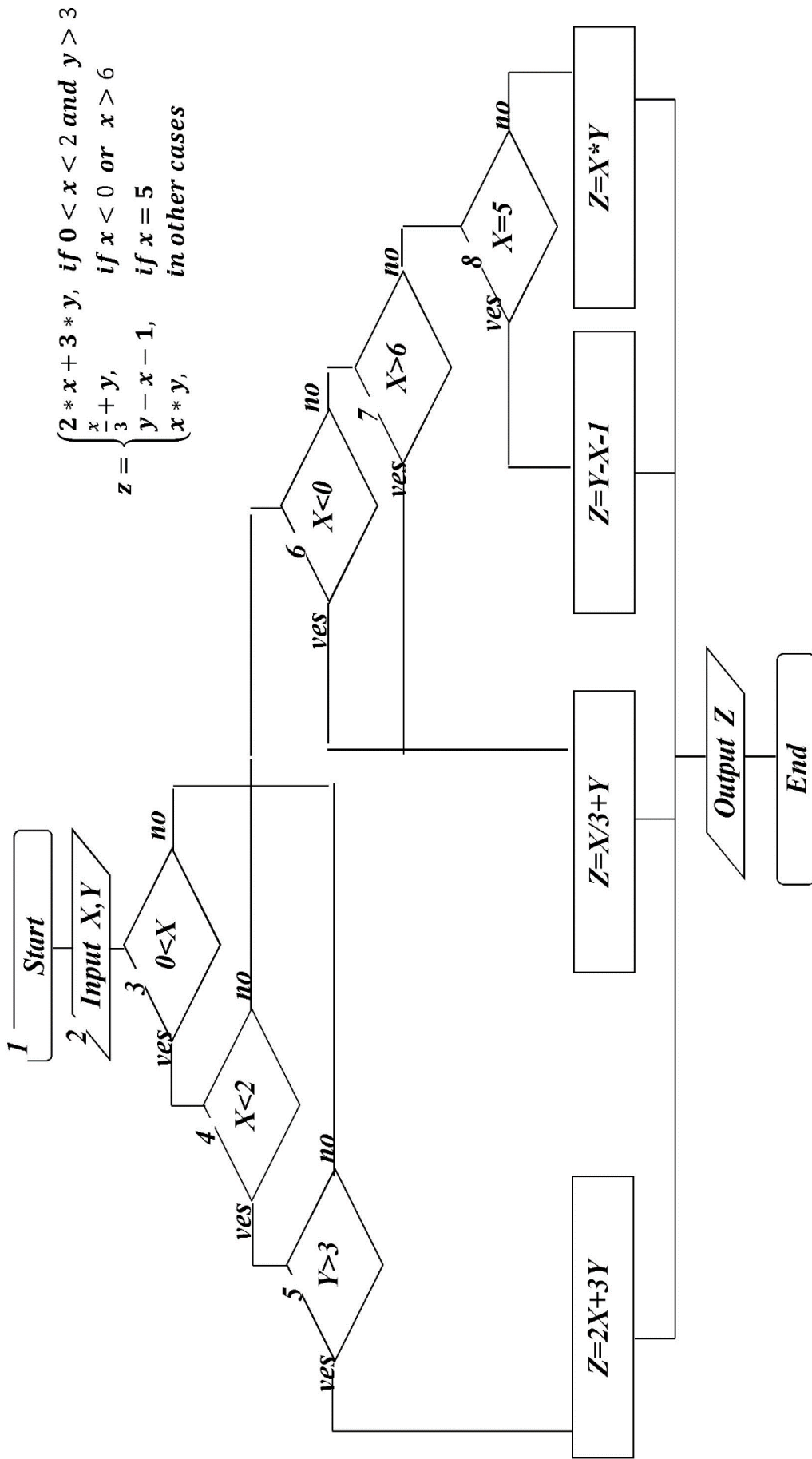


Fig. 3.13. Branched algorithm with a complex condition (logical expression variant 2)

For example 3. The algorithms is shown on the Fig. 3.14.

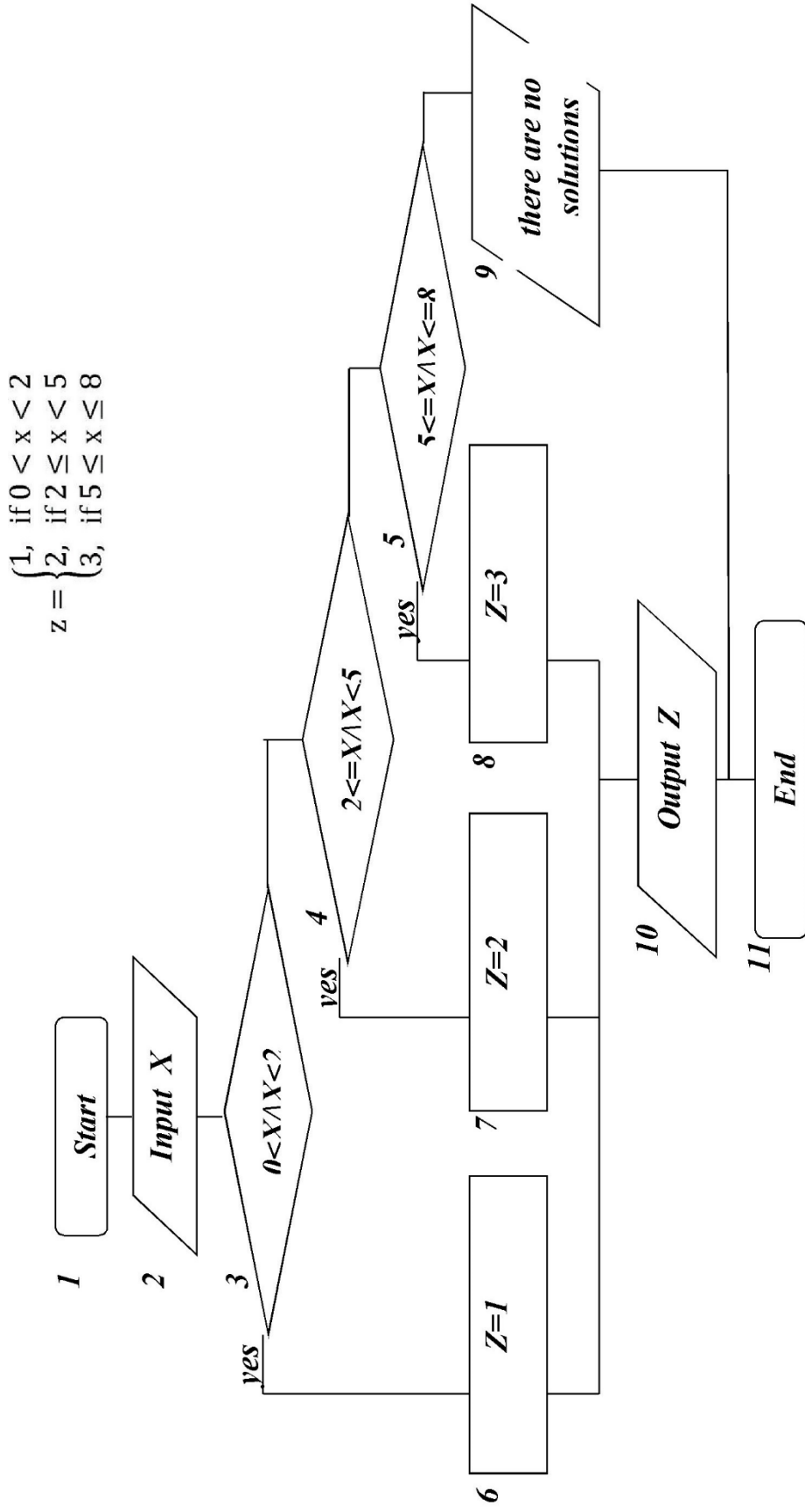
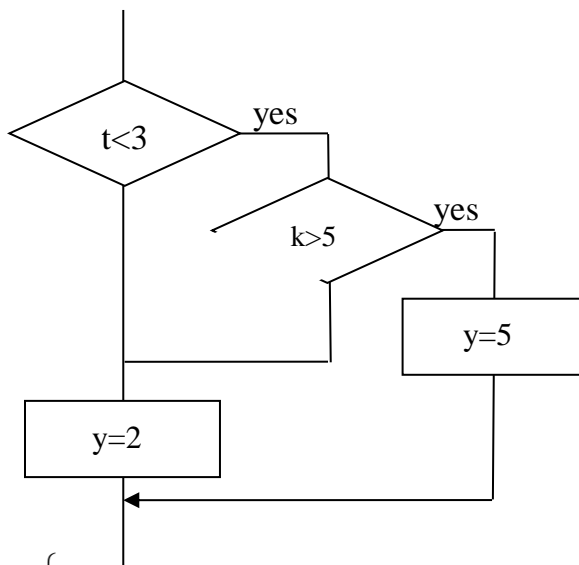


Fig. 3.14. Branched algorithm with an undefined function for certain non-specified argument values

Control questions and tasks

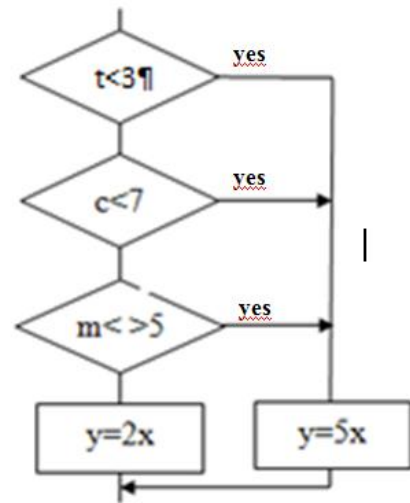
1. Define the concept "branched computing process".
2. Define the concept "logical attitude".
3. Give the list of attitude operations.
4. Define the concept "logical expression".
5. What logical operations are used in logical expressions?
6. Write down formulas for the calculations that correspond to the given fragments of the schematics of the algorithms.

7.

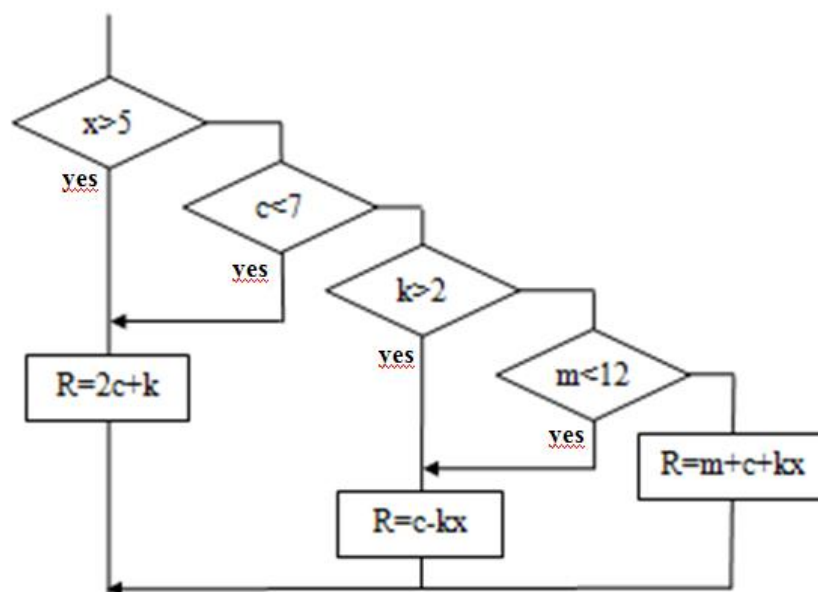


Y = {

R = {



Y = {



8. The branched algorithm is called:

- 1) in which the choice of action depends on the performance of the condition;
- 2) in which it is possible to make many calculations;
- 3) in which actions are performed sequentially - one by one.

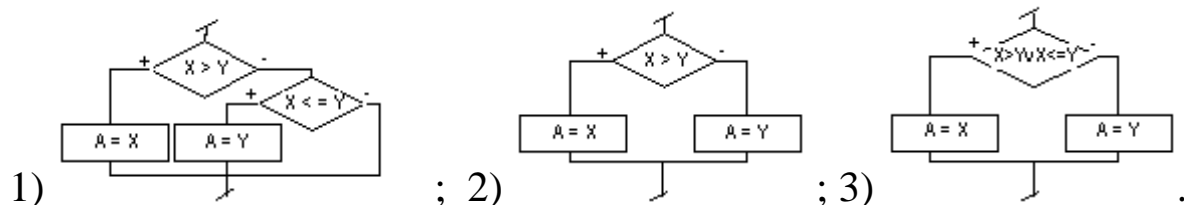
9. An algorithm which selected what to do next depends on the performance of certain conditions is called:

- 1) branched;
- 2) distributed;
- 3) lineal;
- 4) loop;
- 5) iterative.

10. *Selecting branches in branching algorithm can be:

- 1) the result of performing logical reasoning;
- 2) the result of performing a logical definition;
- 3) the result of performing a logical attitude;
- 4) the result of executing a logical expression.

11. Choose the right scheme to calculate the variable $a = \begin{cases} x, & \text{if } x > y \\ y, & \text{if } x \leq y \end{cases}$



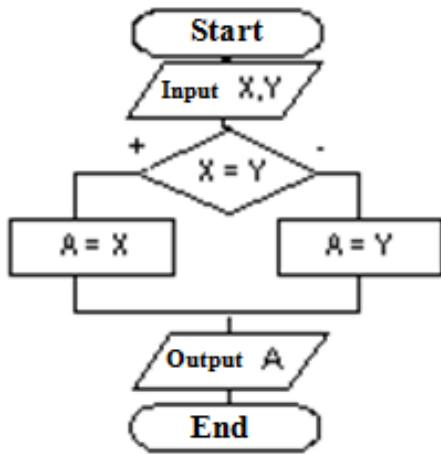
12. Part branched algorithm that controls intended:

- 1) determining variable or constant values;
- 2) unambiguous select one of the options settlement;
- 3) output results;
- 4) calculations.

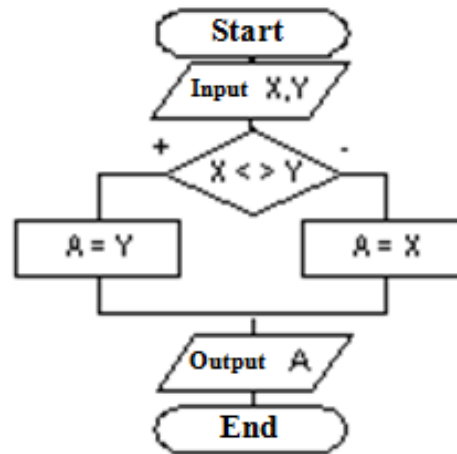
13. *In the part of the branched control algorithm, the condition is specified by:

- 1) logical expression;
- 2) logical attitude;
- 3) logical consideration;
- 4) logical determinant.

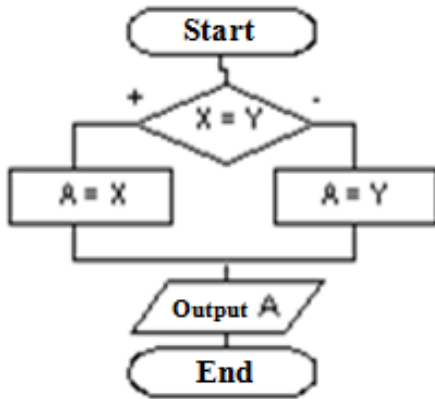
14. Choose the right scheme to calculate the variable: $a = \begin{cases} x, & \text{if } x = y \\ y, & \text{if } x \neq y \end{cases}$



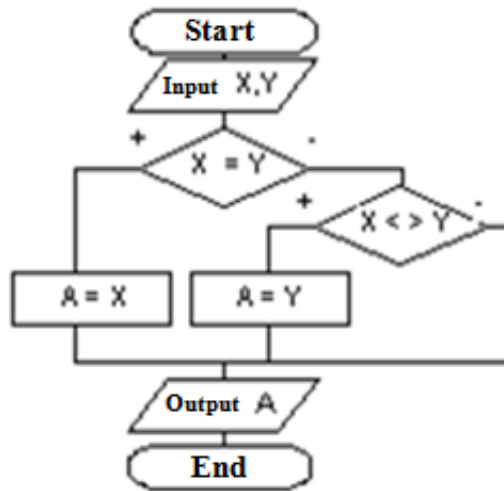
1)



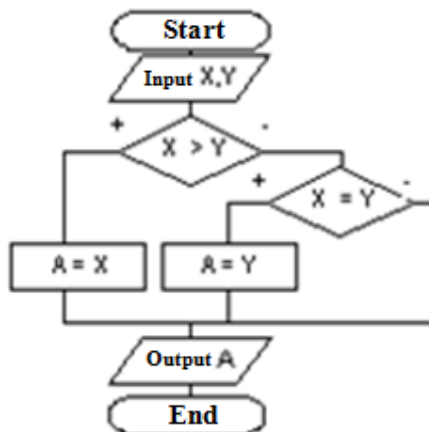
; 2)



3)



; 4)



15. *X is:

- 1) intermediate value;
- 2) final value;
- 3) input value;
- 4) control variable;
- 5) the calculated value.

16. When calculating which variable does NOT use branching?

$$x = \begin{cases} t+1, & \text{if } t < 1 \\ t, & \text{if } t = 1 \\ t-1, & \text{if } t > 1 \end{cases} \quad p = \begin{cases} 1+z, & \text{if } z = 0 \text{ and } y = 0 \\ z+y, & \text{if } z * y < 0 \\ 25, & \text{in other cases} \end{cases} \quad y = \begin{cases} a^3 + x, & \text{if } a < x < 10 \text{ and } a < 10 \\ a + \sin(x), & \text{if } 10 < x \leq 15 \text{ and } a = x \\ y - x - 1, & \text{if } x = 20 \text{ or } 25 < a \leq 30 \\ e^{x-a}, & \text{in other cases} \end{cases}$$

$$z = a^2 + y^3 - 5 \text{ where } t, a - \text{any values}$$

- 1) x; 2) y; 3) z; 4) p.

17. Enter the correct order for the values to be calculated

$$x = \begin{cases} t+1, & \text{if } t < 1 \\ t, & \text{if } t = 1 \\ t-1, & \text{if } t > 1 \end{cases} \quad p = \begin{cases} 1+z, & \text{if } z = 0 \text{ and } y = 0 \\ z+y, & \text{if } z * y < 0 \\ 25, & \text{in other cases} \end{cases} \quad y = \begin{cases} a^3 + x, & \text{if } a < x < 10 \text{ and } a < 10 \\ a + \sin(x), & \text{if } 10 < x \leq 15 \text{ and } a = x \\ y - x - 1, & \text{if } x = 20 \text{ or } 25 < a \leq 30 \\ e^{x-a}, & \text{in other cases} \end{cases}$$

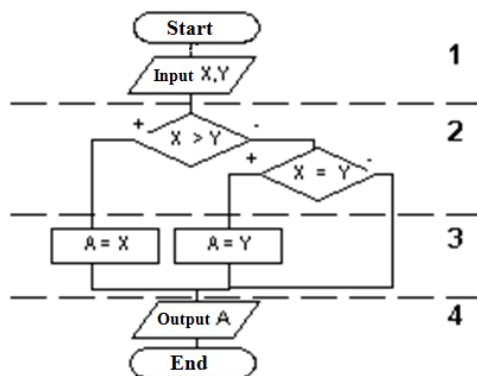
$$z = a^2 + y^3 - 5 \text{ where } t, a - \text{any values:}$$

- 1) t,a,x,y,z,p;
- 2) t,a,p,y,z,x;
- 3) t,a,x,z,y,p;
- 4) t,a,z,p x,y;
- 5) t,a,p,z,x,y.

18. Logical attitude – is:

- 1) sequential writing of variables, constants, arithmetic expressions and standard functions, joined with signs of logical operations (AND, OR, NOT);
- 2) sequential writing of variables, constants, arithmetic expressions and standard functions, joined by symbols of the comparison operation (>,<,<=,>=,= etc.);
- 3) sequential writing of logical expressions combined by signs of logical operations (AND, OR, NOT etc.);
- 4) exceeding the value of one mathematical value over another.

19. Define the sequence of the branched projection algorithm:



- 1) 1 – input of initial data (variable and constant),
2 – control part,
3 – calculation part,
4 – output of intermediate and final values;
- 2) 1 – input of initial data (variable and constant),
2,3 – control part,
4 – output of intermediate and final values;
- 3) 1 – input of initial data (variable and constant),
2 – calculation part,
3 – control part,
4 – output of intermediate and final values.

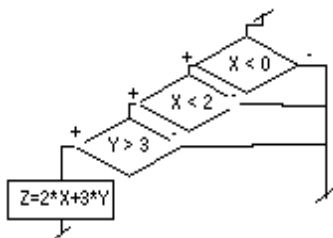
20. Sequential writing of variables, constants, arithmetic expressions and standard functions, joined by the symbols of the comparison operation ($>$, $<$, \leq , \geq , $=$ и т.д.) is called:

- 1) logical expression;
- 2) logical definition;
- 3) logical consideration;
- 4) logical attitude.

21. Sequential writing of expressions combined with signs of logical operations is called:

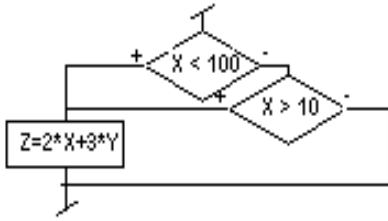
- 1) logical attitude;
- 2) logical definition;
- 3) logical consideration;
- 4) logical expression;
- 5) logical determinant.

22. The part of the control algorithm contains:



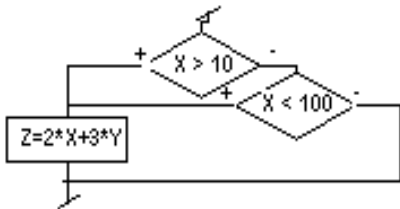
- 1) logical addition (operation OR);
- 2) logical multiplication and logical addition;
- 3) logical multiplication (operation AND);
- 4) logical objection (operation NOT).

23. When calculating a variable Z will be used X:



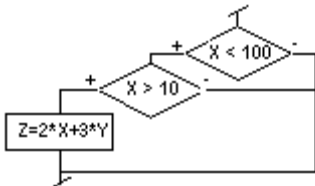
- 1) over 10;
- 2) less 100;
- 3) from 10 to 100;
- 4) the value of the full range.

24. When calculating a variable Z will be used X:



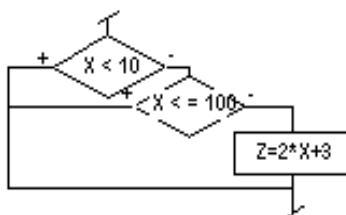
- 1) over 10;
- 2) less 100;
- 3) from 10 to 100;
- 4) the value of the full range.

25. When calculating a variable Z will be used X:



- 1) over 10;
- 2) less 100;
- 3) from 10 to 100;
- 4) the value of the full range.

26. When calculating a variable Z will be used X:



- 1) over 100;

- 2) less 10;
- 3) from 10 to 100;
- 4) the value of the full range.

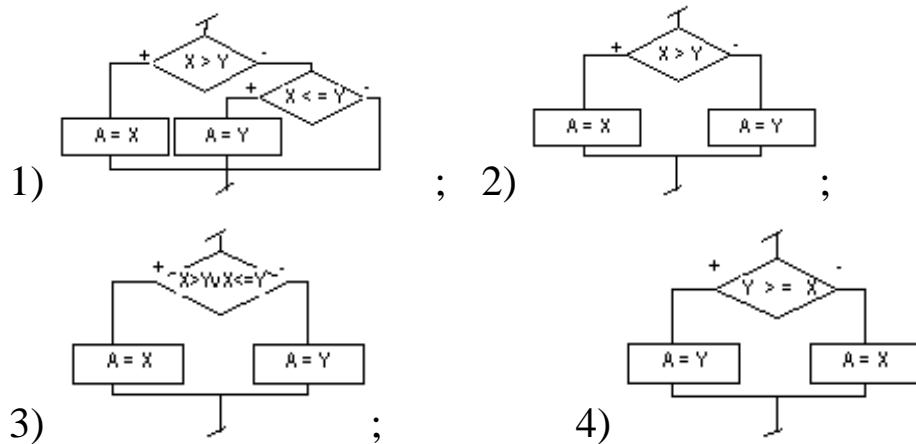
27. *Name the attitude operations used to organize logical attitude:

- 1) \geq ; 2) \leq ; 3) \vee ; 4) \wedge ; 5) \neg ; 6) $<$.

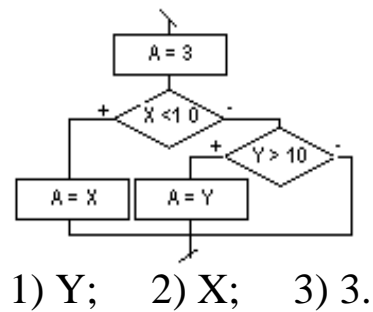
28. *Name the logical operations used to organize the logical expressions:

- 1) \geq ; 2) \leq ; 3) \neg ; 4) \neq ; 5) $<$; 6) \vee ; 7) \wedge .

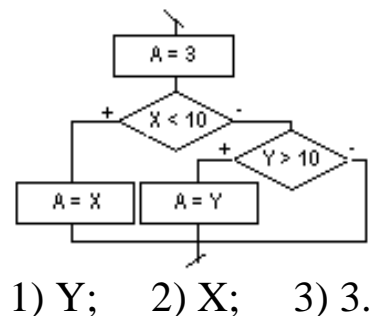
29. *Select the correct scheme for calculating the variable A



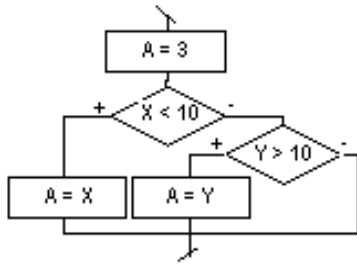
30. What is the variable A, if X=22 and Y=55 :



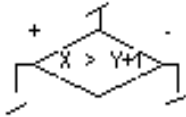
31. What is the variable A, if X=2 and Y=55 :



32. What is the variable A, if X=22 and Y=5 :



- 1) Y; 2) X; 3) 3.

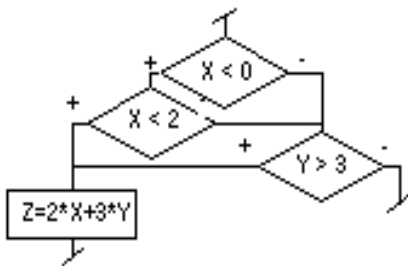
33. Part of the algorithm , that controls is NOT used:

- 1) logical operations;
- 2) logical attitude;
- 3) arithmetic operations.

34. *The result of calculating a logical expression or a logical relationship can be a value:

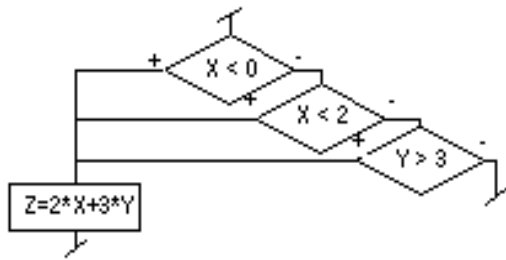
- 1) True;
- 2) False;
- 3) Not Determined;
- 4) Fault.

35. The part of the control algorithm contains:



- 1) logical multiplication (operation AND);
- 2) logical addition (operation OR);
- 3) logical multiplication and logical addition;
- 4) logical objection (operation NOT).

36. The part of the control algorithm contains:



- 1) logical multiplication (operation AND);
- 2) logical addition (operation OR);
- 3) logical multiplication and logical addition;
- 4) logical objection (operation NOT).

37. Specify the condition under which values are selected $X \in [10, 100)$ or $Y \in [10, \infty)$:

- 1) $10 \leq X \wedge X < 100 \vee Y \geq 10$;
- 2) $10 \leq X \vee X < 100 \wedge Y \geq 10$;
- 3) $X \leq 10 \wedge X < 100 \vee Y \geq 10$;
- 4) $X \leq 10 \vee X < 100 \wedge Y \geq 10$.

38. Specify the condition under which values are selected $X \in [10, 100)$ or $Y \in [10, \infty)$ or $Z = 0$:

- 1) $10 \leq X \wedge X < 100 \vee Y \geq 10 \wedge Z = 0$;
- 2) $10 \leq X \vee X < 100 \wedge Y \geq 10 \vee Z = 0$;
- 3) $X \leq 10 \wedge X < 100 \vee Y \geq 10 \wedge Z = 0$;
- 4) $X \leq 10 \vee X < 100 \wedge Y \geq 10 \vee Z = 0$.

UNIT 4. CYCLICAL COMPUTING PROCESSES

4.1. Simple arithmetic cyclic computing processes

In the practice of engineering calculations have to perform multiple calculations on the same mathematical dependencies at different values of the input variables. Such repetitive processes are called loops.

The cyclic algorithm is called algorithm that contains a sequence of operations that are performed repeatedly. Such algorithms are based on typical repetition structures. Using loops can greatly reduce the flow chart and size of the program.

Conditions ending cycle can be checked before executing the loop – a cycle with a prerequisite, or after executing a loop, a loop with a condition.

For images of the diagram cyclic structure can be used "solution" together with the symbol "process" or special character "modification".

In the loops are parameters (counter managing variable) – the numerical variable, which is used to organize repetitions of the cycle.

According to the method of specifying the number of repetitions of the loops body distinguish arithmetic (regular) and iterative cycles.

In iterative cycles, the number of repetitions usually cannot be determined before execution and depends on the specified condition for exiting the loop.

Arithmetical is loops, the number of repetitions of which can be explicitly specified in the condition of the task or, if necessary, calculated before the start of its execution. The loop parameter, in this case, is determined by the boundaries and step of its change, which are given by constants, variables, or expressions.

Arithmetic cycles with one parameter are called simple, with several – inserted.

To arrange a simple arithmetic loop, follow these steps:

- 1) set the initial value of the loop parameter;
- 2) check the condition of the end of the loop – whether the value of the cycle parameter is included in the interval of its change. If the parameter value is no more than the final value at a positive step or at least the initial value at a negative step, then the loop body is executed. Otherwise, the loop is exited;

3) to perform the given calculations and, if necessary, output of results;

4) change cycle parameter by step size;

5) back to item 2.

For example. To make the algorithm of calculation of function values: $y = \sin(xa) - b$, $a = 9.63$, $b = 5.1$. Loop parameter x changes in the interval from $x_n = 1$ to $x_k = 3$ in increments $h_x = 0.5$

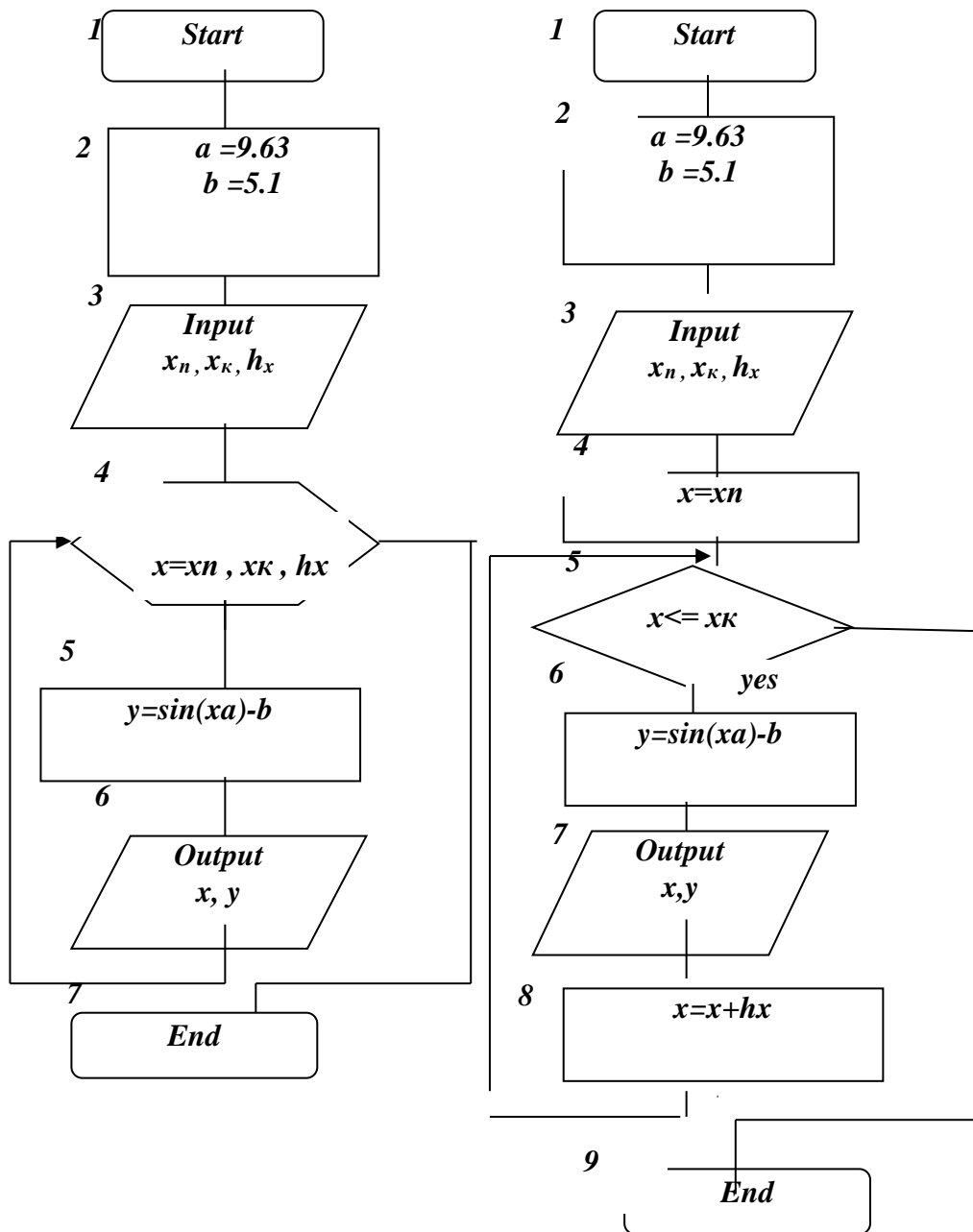


Fig. 4.1. A cyclic algorithm with a symbol
a - "modifier"; b - "solution"

The number of repetitions of the arithmetic loop is calculated by the formula:

$$N = \left] \frac{x_k - x_n}{hx} \left[+ 1 = \left] \frac{3-1}{0.5} \left[+ 1 = 5$$

where x_n – the initial value of the loop parameter;
 x_k – the final value of the loop parameter;
 h_x – the step of changing the parameter;
 $\left] \cdot \left[$ – the operation of taking an integer part of a number.

The task algorithm is a simple cyclic computing process with an implicitly specified number of repetitions.

In the algorithm shown in Fig. 4.1a, the loop is organized using the symbol 4 - "modifier". This symbol defines the initial, final value of the parameter of cycle X and the step of changing it. Calculation of the function performed in the symbol 5.

Fig. 4.1б shows a diagram of this algorithm with the symbol "solution". Its symbols 4, 5, and 8 correspond to the symbol 4 of the scheme in Fig. 4.1a.

Assignment of symbols:

symbols 2-3 – formation of input data;
symbol 4 – setting the initial value of parameter X;
symbol 5 – check the condition of the end of the loop;
symbol 6 – calculating functions;
symbol 7 – output the calculated value of the function and the current value of the cycle parameter;
symbol 8 – calculating the next value of the cycle parameter.

A recurrent expression is an expression that describes any element of a sequence of numbers, where each subsequent element is calculated from the previous one.

For example, formula $X=X+H$ means: to content X add H and write the result in X, that is, the recurrent expression binds consistently calculated values. The input data for the next step is the results of the previous one.

Recurrent expressions are also used to calculate the sum of the finite data output.

To calculate the sum of a number of data, follow these steps:
 generate initial data;
 determine the initial state of the adder, which will be implemented
 accrued;
 organize accrued cycle by adding new values to the sum of all previous;
 output the result.

For example. Make the algorithm for calculating the sum of 15 values of function $Y = \sin(Ax+B)$.

Initial value $X=2$; step change $H=1.5$; A, B – arbitrary numbers.

In the algorithm on Fig. 4.2 the number of repetitions is specified in the task condition.

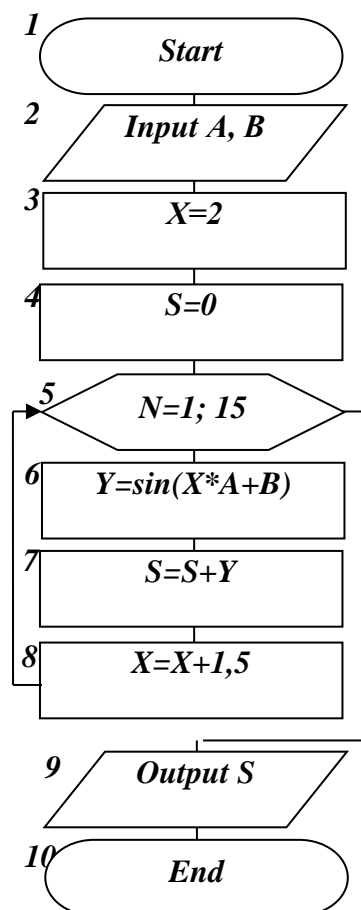


Fig. 4.2

Assignment of symbols:
 symbols 2-3 – formation of input data;
 symbol 4 – preparation (zeroing) of the adder;

symbols 5-8 – organization of a loop with a parameter N (the number of calculation functions Y); calculation of the function Y; adding received values; increment X the magnitude step; symbol 9 – output value.

For example. Make algorithm for calculation of factorial $Y=N!$ Factorial – is the product numbers of natural series from 1 to N. The algorithm for calculating the factorial is in Fig. 4.3.

Assignment of symbols:

symbol 2 – input N. N – variable, so given different values can be calculated factorial of any number;

symbol 4 – preparation of the variable Y, in which the product will accumulate. Given that any number will remain important in the multiplication of 1 to store the product initial value adder will be equal to 1;

symbols 5-7 – organization of cycle for accumulation of product of N values of X;

symbol 8 – output value.

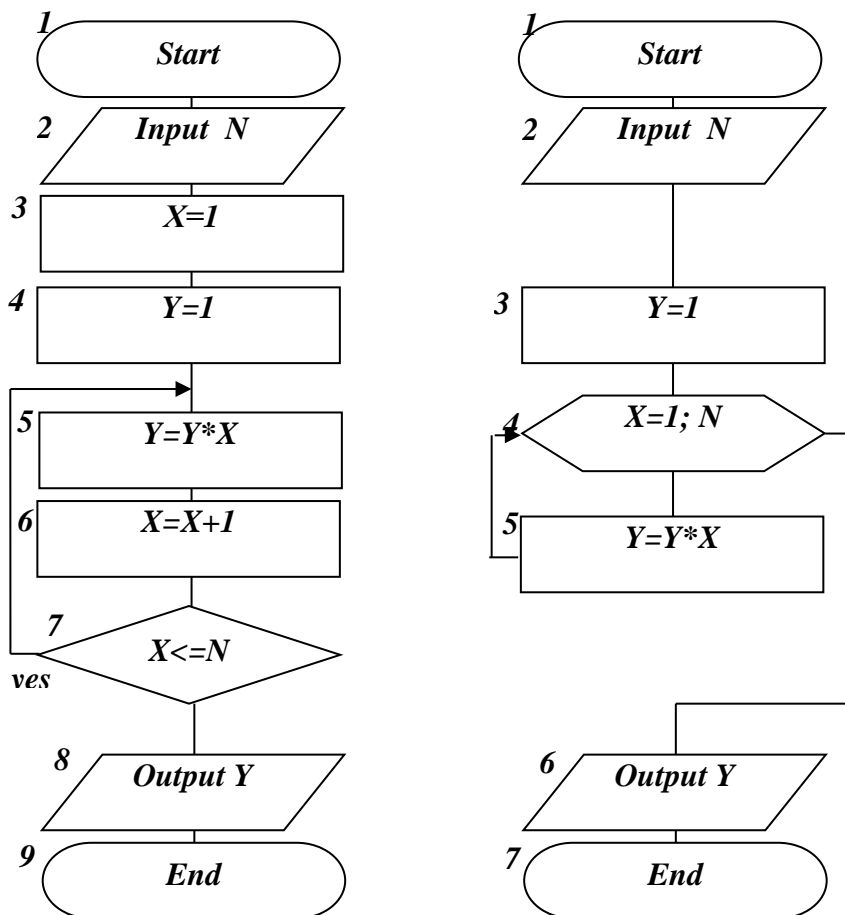


Fig. 4.3 Algorithm for calculation of factorial $Y=N!$

4.2. Nested cyclic computing processes

Along with simple cyclical process in constructing algorithms used nested cyclic computing processes..

A nest is called a loop that contains one or more cycles. A cycle that covers other cycles is called an external cycle, and others – internal. The basic rule for constructing nested loops – is the coverage of an external cycle of one or more internal.

Nested loops are used when calculating functions or calculating expressions that depend on several arguments. Each cycle is organized in the same way as a simple cycle.

For example. Make the algorithm (Fig. 4,4) for calculating the function $y = a \sin(x + a)$, if $x \in [0; 1.7]$, $h_x = 0.1$, $a \in [0; 2]$, $h_a = 0.2$

Loop parameters change sequentially, that is, when one sense parameters of the loop setting internal cycle consistently takes all its value.

The result will be a table:

x=0	a=0	y=...
x=0	a=0.2	y=...
x=0	a=0.4	y=...
.....		
x=0	a=2	y=...
x=0.1	a=0	y=...
x=0.1	a=0.2	y=...
x=0.1	a=0.4	y=...
.....		
x=1.7	a=2	y=...

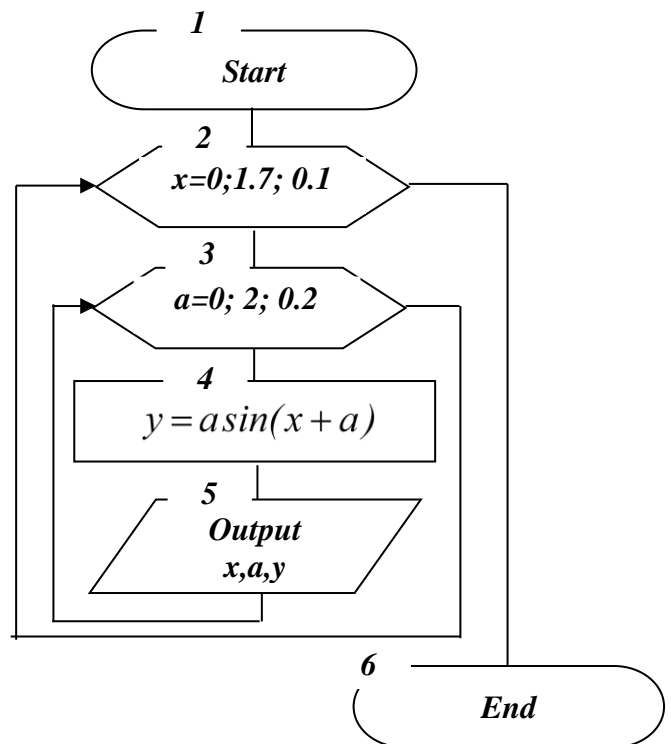


Fig. 4.4

The total number of calculations (execution of symbols 4 and 5) is determined by the product of the number of repetitions of the outer loop and the inner loop. The appropriate calculations:

$$N = ((1.7 - 0) / 0.1 + 1) * ((2 - 0) / 0.2 + 1) = 198$$

that is, 198 calculations will be performed, and the output will result in 198 lines. Nested in cyclic processes can be not only other loops, but also branching, and fragments of linear type.

For example. Make the algorithm for calculating and deriving the values of a function:

$$y = \begin{cases} a \sin(x+a), & \text{if } x < 0 \text{ and } a < x \\ -b - \cos(x-a) & \text{in other cases} \end{cases}$$

where $x \in [-3; 3]$, $h_x = 0.1$, $a \in [-10; 10]$, $h_a = 0.5$.

This algorithm (Fig. 4.5) is a nested cyclic branching process.

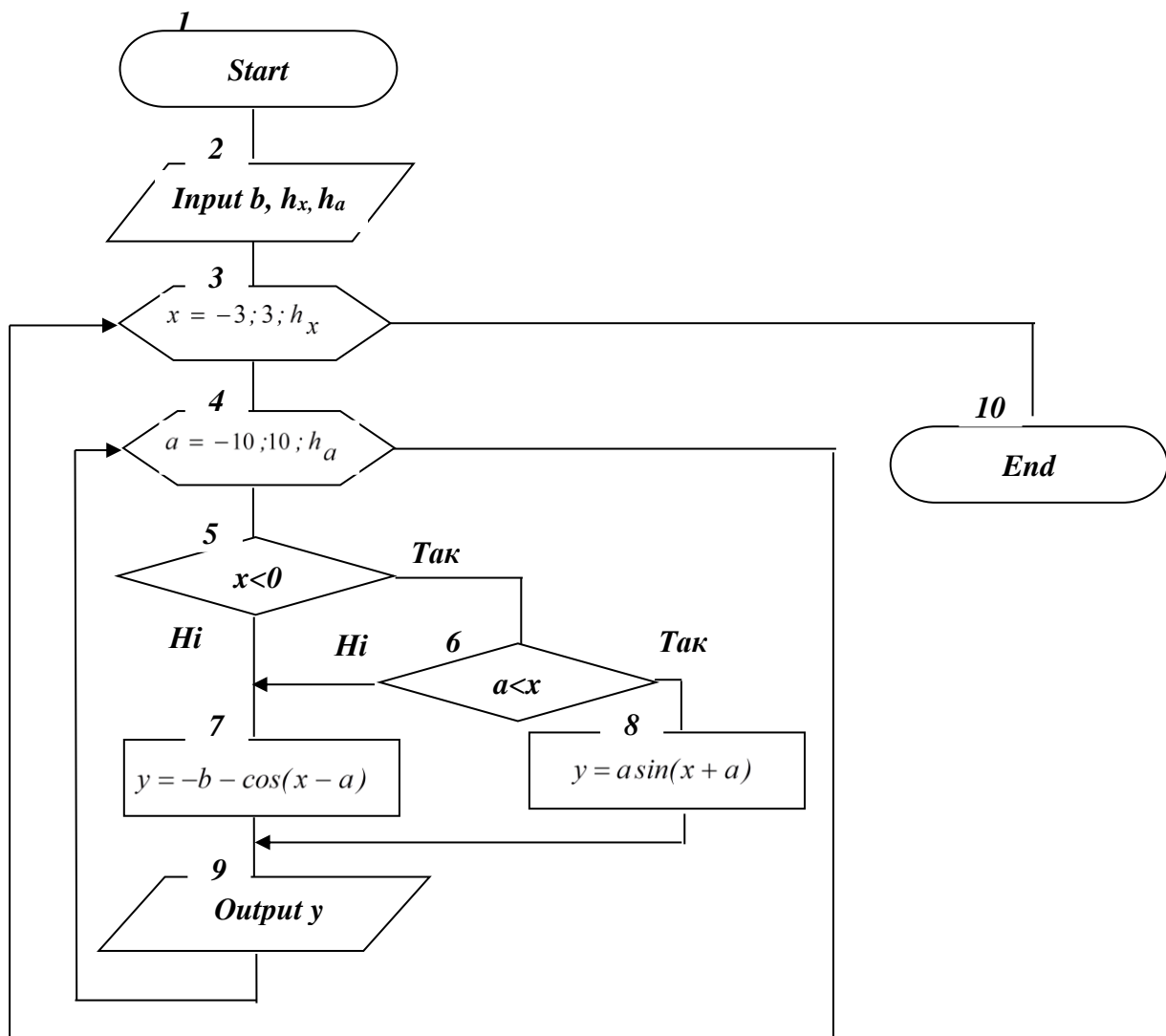


Fig. 4.5

An internal computing process is present in the inner loop. Each branch can contain fragments of branching linear type (symbols 7, 8, 9).

The nesting depth, that is, the number of open cycles on the plot of the algorithm, is not limited. Restrictions may appear later, when writing programs due to insufficient capacity used programming system.

For example. Make the algorithm for calculating and deriving the values of a function:

$$Y = \sum_{x=0}^{10} \left(\frac{A!}{\sum_{i=1}^n \frac{i}{x^2 + 1}} + \prod_{i=1}^n \frac{x A!}{i!} \right).$$

Segmenting by parts, we get:

$$p = \sum_{i=1}^n \frac{i}{x^2 + 1}; \quad f = i!; \quad q = \prod_{i=1}^n \frac{x}{f}; \quad B = A!$$

Each of the variables p, f, q, B is determined by sequential accumulation in the respective loops. Here symbols 5, 10, 15, 16 i 17 – are recurrent ratios for the accumulation of sums and products. This is a nesting: value p is used to calculate individual additions in the sum Y, and so the accumulation cycle Y (symbols 6-17) covers the accumulation cycle of value p (symbols 8-10).

Similarly, value q is the separate application for the amount Y, and the calculation loop q (symbols 11-16) is nested in the calculation loop Y. For calculating q need to calculate the value of factorial i! – value f. This is done in a loop (symbols 13-15), that is nested in the calculation loop q, because the value f is under the sign of the product q and depends on the variable i.

Factorial value A! – value B is calculated in a loop (symbols 3-5), that is not nested. This is based on the conditions of the task, because A! can be taken from the sign of sum and product.

So it is necessary to analyze other tasks that require the construction of nested structures.

In this example, you need to calculate the values of sums and outputs several times. For accumulate sums or product must be used recurrence relations. In the algorithm (Fig. 4.6) are used secondary values.

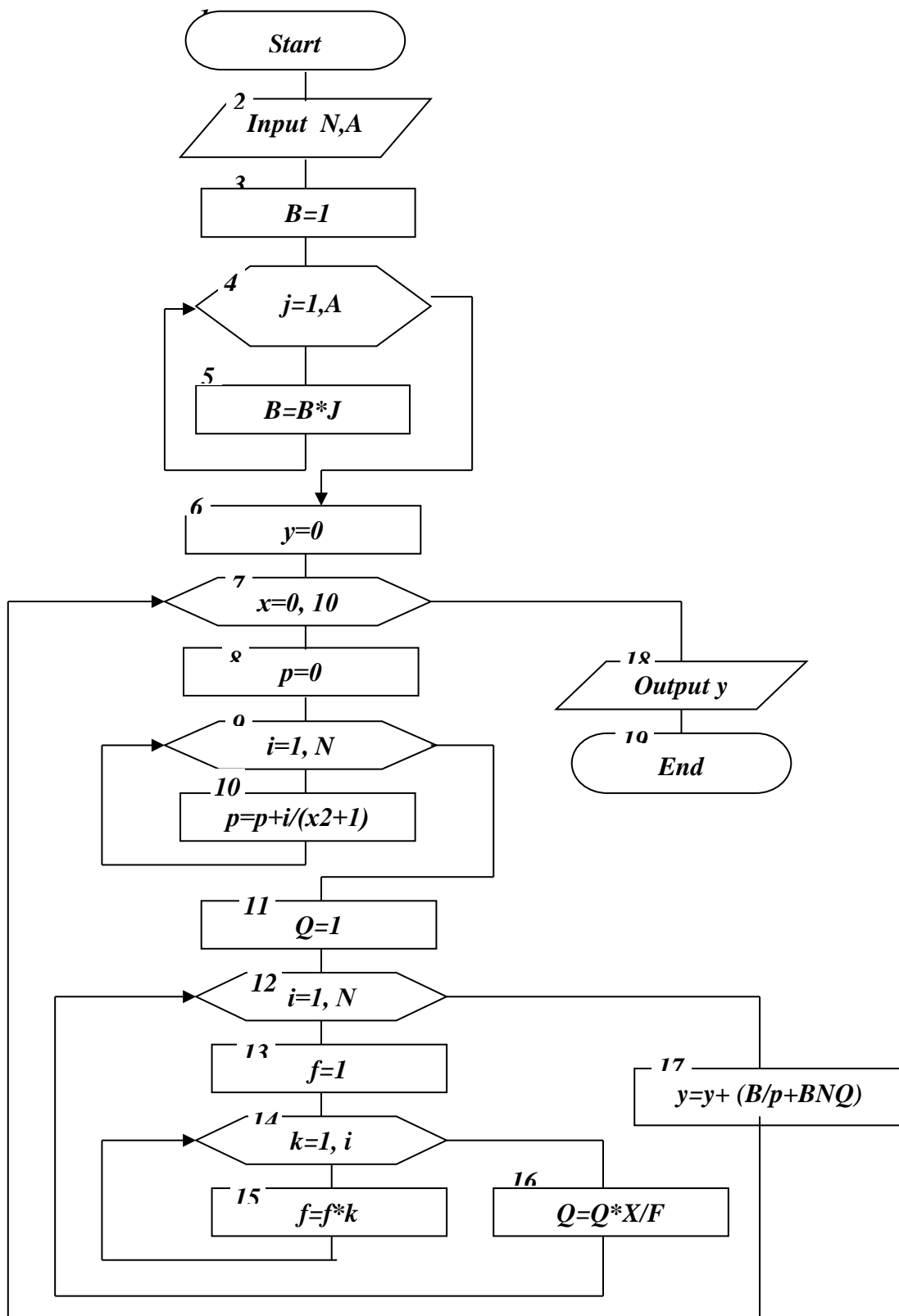


Fig. 4.6

4.3. Iterative cyclic computing processes

In addition to arithmetic loops in engineering practice are used iterative cyclic computing processes.

Iteration loop – it is a computational process in which the number of repetitions of a loop body is unknown and depends on the condition of achieving the desired result. In such algorithms, it is necessary to ensure that the cycle exit condition is fulfilled, that is, the convergence of the iterative process.

For example. Calculate the function Y, that represented by the sum of the elements of an infinite convergent number series:

$$y = \frac{1}{1!+a} - \frac{2}{2!+a^2} + \frac{3}{3!+a^3} - \frac{4}{4!+a^4} + \dots + (-1)^{i+1} \frac{i}{i!+a^i} + \dots$$

Based on that, that the number row is infinite, then for practical calculations they are limited by the number of elements, guided by the set accuracy E calculate the sum Y.

A convergent numerical series – these are a series of values (serie of elements), the value of each is less than the previous value of this serie. Virtually calculating the sum of the elements is stopped on the element, which is lower in value than the set accuracy E. All the following elements (which are also less than E), disregard. If the values of the elements change as shown in Fig. 4.7, then the amount will be included only the first four elements. The fifth element and the next ones in the amount will not come out.

Iterative algorithms for calculating sums of infinite series contain the following steps:

- input data;
- setting the initial value of the sum and the auxiliary variables, if it necessary;
- calculating the value of the current element;
- comparing a series element with a given accuracy E;
- if the element is not less E, then it is added to the accumulated sum, change the values of the auxiliary variables, after which the next element is calculated, and the loop repeats;
- if the element is less than E, then the accumulation of the sum is stopped and the result is output.

When constructing an iterative computational process, the use of the symbol "modifier" is invalid (the upper value of the loop parameter is unknown).

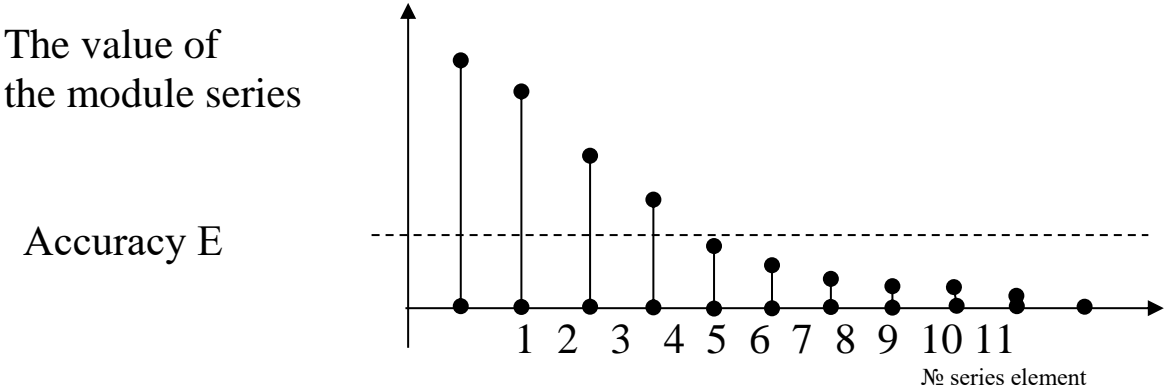


Fig. 4.7. The dependence the value of an element of a convergent serie on its number

On Fig. 4.8 the algorithm for finding the sum Y. The accumulation of the sum of elements is performed in a symbol 10 using recurrence dependency $Y = Y+Z$, where Z – the value of the next calculated addition of the sum.

Each following addition is different from the previous one by values, that included in it (the numerator and denominator change). To track these changes, the algorithm used auxiliary variable i. It specifies the extension number, that is calculated, i used to define the current number $Z=i/(F+a i)$.

In the formula for Z includes values factorial i!, which is calculated in symbols 5-7 and marked as F.

Symbols 3 and 4 set the initial value of the sum Y and value of i.

Symbol 9 – checking the need for adding Z to the sum. We add, if $|Z|>E$ and its value cannot be ignored. Check series, which may have negative elements, is performed using the value of the module.

Exit the iterative loop can be performed at any step when the specified precision condition is reached. If the given example will introduce each time different values a, then the value of the current term will be different. Therefore, it is impossible to predict the number of repetitions at the same accuracy in advance.

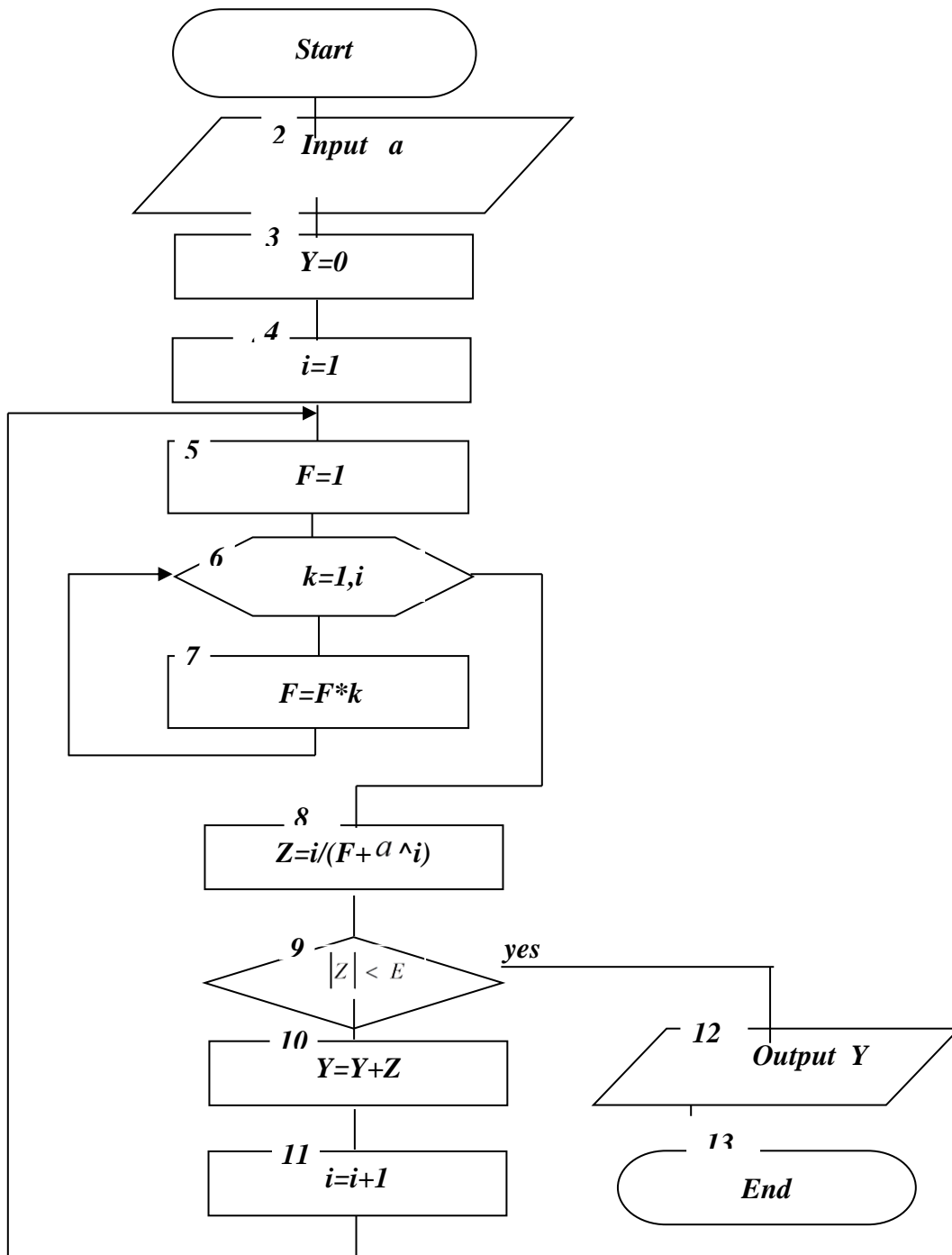


Fig. 4.8

For example (Fig. 4.9), an approximate calculation of the square root of the number X: $Y = \sqrt{X}$

Assume $Y_0 = X$ Each of the following values is calculated by the preceding to the formula: $Y_i = 1/2 * (Y_{i-1} + X / Y_{i-1})$

We finish the iterative process when the condition is fulfilled

$$|Y_i - Y_{i-1}| < E,$$

where $E = 10^{-5}$ – given accuracy of finding the root
 $Y = \sqrt{4}$ $Y_0 = 4$ $Y_1 = 2.5$ $Y_2 = 2.05$ $Y_3 = 2.0006$ etc.

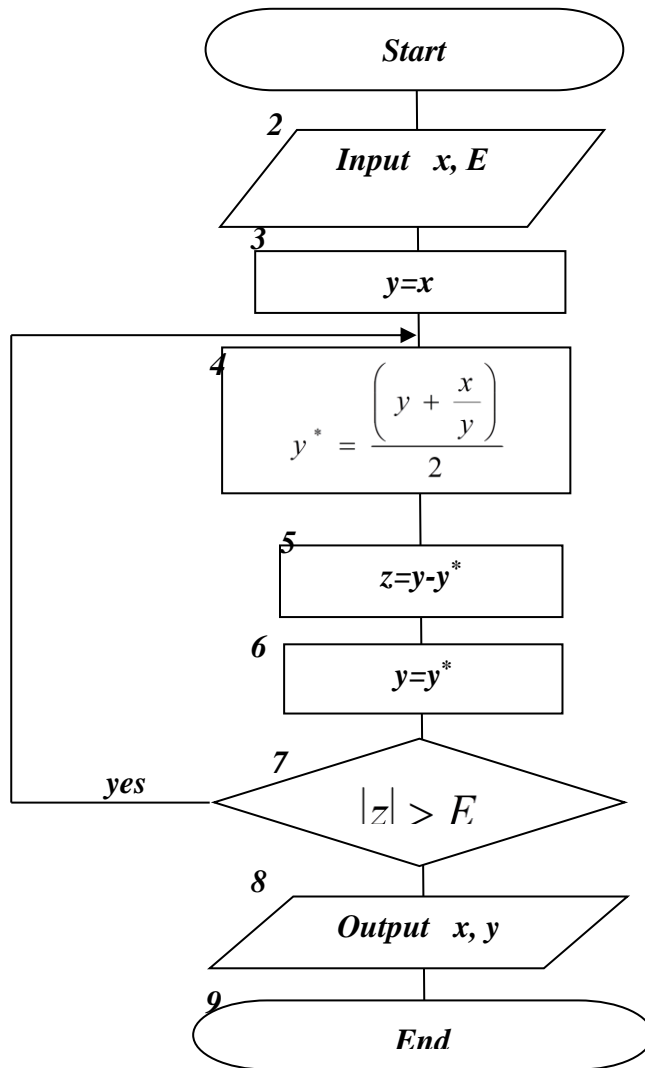


Fig. 4.9

Use of cyclic algorithms for tasks of finding extreme values of functions.

Point x_0 is called the point of local (relative) maximum for a function $f(x)$, if the value of the function at this point more, than the value of the function at the closest points.

Similarly, point x_0 is called the point of local (relative) minimum for a function $f(x)$, if the value of the function at this point is less, than the value of the function at the closest points.

Global extremum (Fig. 4.10) – is the largest (smallest) the function of all local maximums (minimums).

In solving engineering problems, as a rule, this is the value you need to find. There are cases of several equal global extremes in different parts of the domain the function definition.

The function can be defined in various ways: tabular, analytical (using the formula), descriptive and graphic.

The tabular way is, that all the numeric values of the argument are in one line, and the value of the function – in the second line so that each value of the argument corresponds to a specific value of the function.

In the analytical method, the function is given by a mathematical formula, by which the value of Y is calculated by a given value of X.

In the descriptive way, the relations between X and Y is expressed in a verbal description, for example, Y is the largest integer not exceeding X.

Graphical method – image in a rectangular coordinate system of a curve line $Y=f(X)$.

With any method of setting a function to calculate the extremum must have a certain set of values X and $Y=f(X)$ at a given research interval.

Regardless of the type of problem to find extremum can be used the same algorithm, that is, the minimization task can be easily transformed into a maximization task, by changing the sign of the function to the opposite.

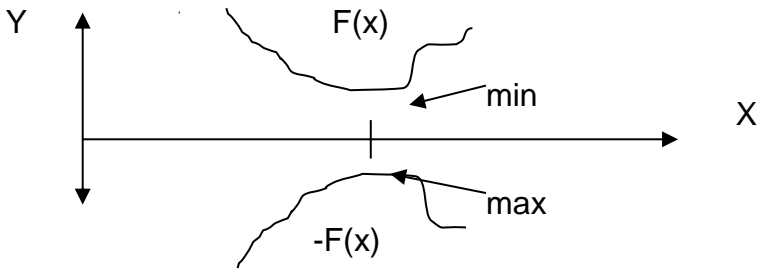


Fig. 4.10

Because the analytic expression of a function is constant over the interval of definition, then its different values arise when changing the parameters (arguments) of this function. Therefore, the procedure for determining the extreme value of the function is performed in a loop.

If as the argument changes gradually, the function also changes gradually, then it is said that the function is continuous.

In this small change of the argument corresponds to a small change of function. Let's give a strict definition.

Function $y = f(x)$ is called continuous at the point x_0 , if defined in some vicinity of this point (including the point itself) and the limit of function at a point x_0 is equal to the value of the function at the very point:

$$\lim_{x \rightarrow x_0} f(x) = A = f(x_0) = f(\lim_{x \rightarrow x_0} x)$$

Geometrically the continuity of a function at an interval means that the graph of this function at a given interval is a solid line without jumps and breaks. In other words, the individual points on the graph of a continuous (interval) function can be connected by a solid line.

They say that the point x_0 is a breakpoint for the function $y = f(x)$, if the function exists around this point (in the point x_0 the function may or may not exist), but in the point x_0 continuity conditions are satisfied.

It should be noted that there are gaps of 1st or 2nd kind of function $f(x)$ does not affect the following method, because it can provide verification of current measured value function and reject critical points, providing the appropriate message in the algorithm.

Consider the example of algorithm of maximum function (similar for the minimum).

Before starting the cycle to determine the maximum value of the function $Y=f(x)$ calculate the first value of the function, taken as a maximum, that is $Y_{max}=Y_1$. To calculate the next function value, we change the value of the loop parameter by a step i , if a new parameter value is acceptable, calculate the corresponding value of the function. Obtained function value Y_2 , compare with Y_{max} .

If $Y_2 > Y_{max}$, then Y_{max} takes on value Y_2 ($Y_{max}=Y_2$), otherwise Y_{max} retains its value, and after changing the cycle parameter, the process is repeated.

All this can be written by mathematical dependence:

$$Y_{max} = \begin{cases} Y_i, & \text{if } Y_i > Y_{max} \\ Y_{max}, & \text{if } Y_i \leq Y_{max} \end{cases}$$

where Y_i – the current value of the function.

Exit the loop is achieved when the parameter reaches the upper limit of the interval.

It should be noted that the solving such problems, it is not about the minimum or maximum value of the function, but about the minimum or maximum among its calculated values. This is because the computer calculates the discrete values of the function at the corresponding discrete values of the parameter, and the real minimum or maximum may be between them (Fig. 4.11).

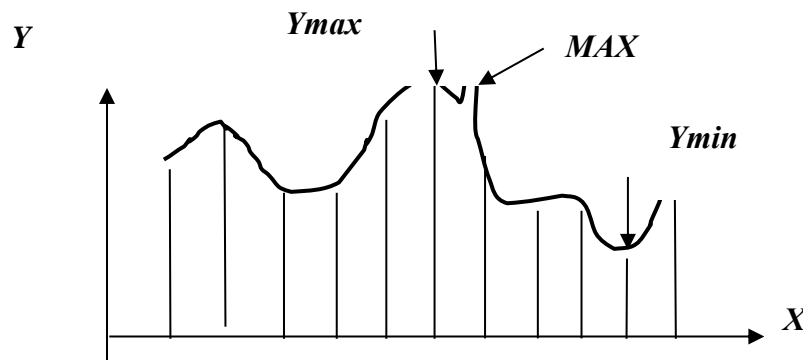


Fig. 4.11

It is possible to increase the accuracy of determination of the extremum by reducing the step of changing the loop parameter.

For example (Fig. 4.12). Make the algorithm for finding the maximum value of the function (Y_{\max}) $Y = |A| \cdot \text{EXP}(B \cdot X)$, if the parameter X changes from 0 to 4 with step $h=0.5$.

Also determine the value of the argument at which the maximum is reached (X_{\max}), and what account will be among the calculated values Y_{\max} (m).

symbol 2 – input of variable a,b;

symbol 3 – assignment the initial value of the loop parameter X ;

symbol 4 – calculating the first value of a function;

symbol 5 – variable and is assigned a value 1

symbol 6 – variable m assigned value i; variable X_{\max} assigned the first argument value; variable Y_{\max} assigned the first functions value;

symbol 7 – organization of a loop by parameter X ;

symbol 8 – calculating the next value of the function;

symbol 9 – increasing the value of the counter i for 1;

symbol 10 – comparing the next value of a function with Y_{max} .
 If the next function value is greater than Y_{max} , then Y_{max} accepts this value. Variable m assigned value i ; variable X_{max} привласнюється поточне значення аргументу (symbol 11), otherwise Y_{max} saves its value. The transition to calculation the next parameter X .

symbol 12 – output Y_{max} , X_{max} , m

symbol 13 – exit from the algorithm.

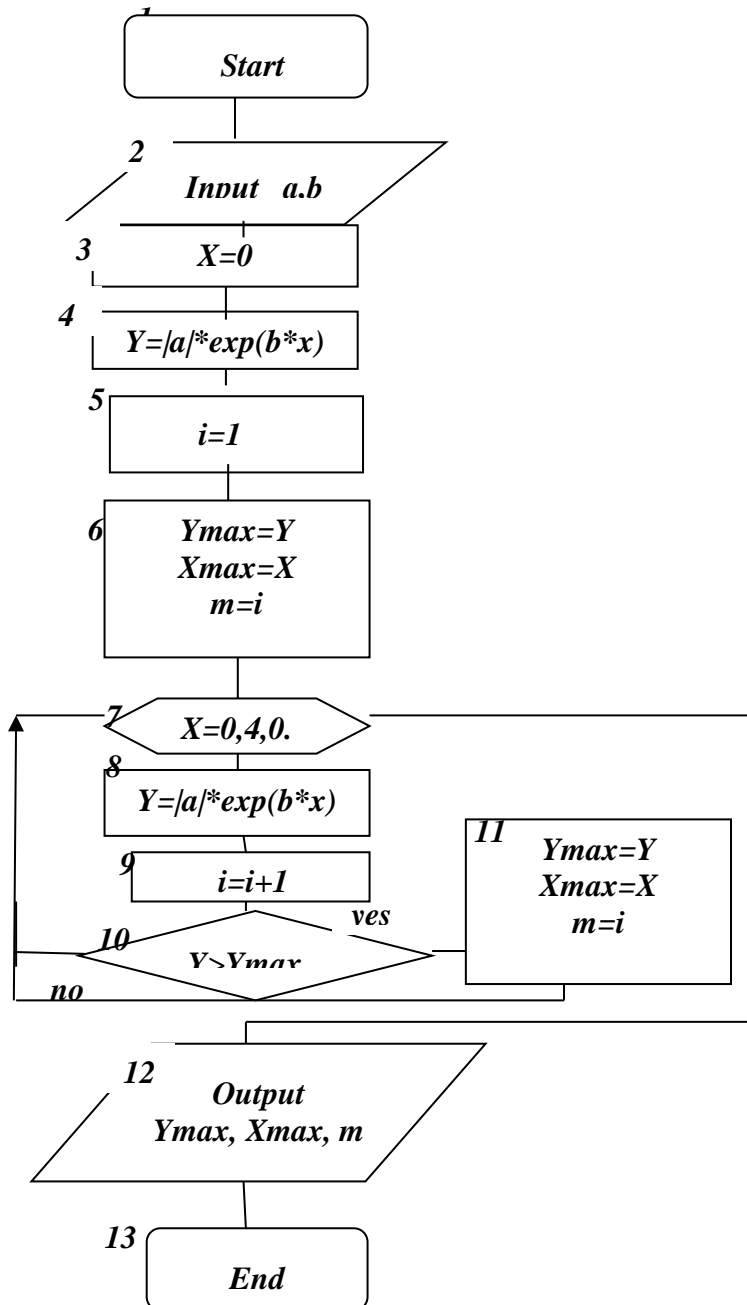
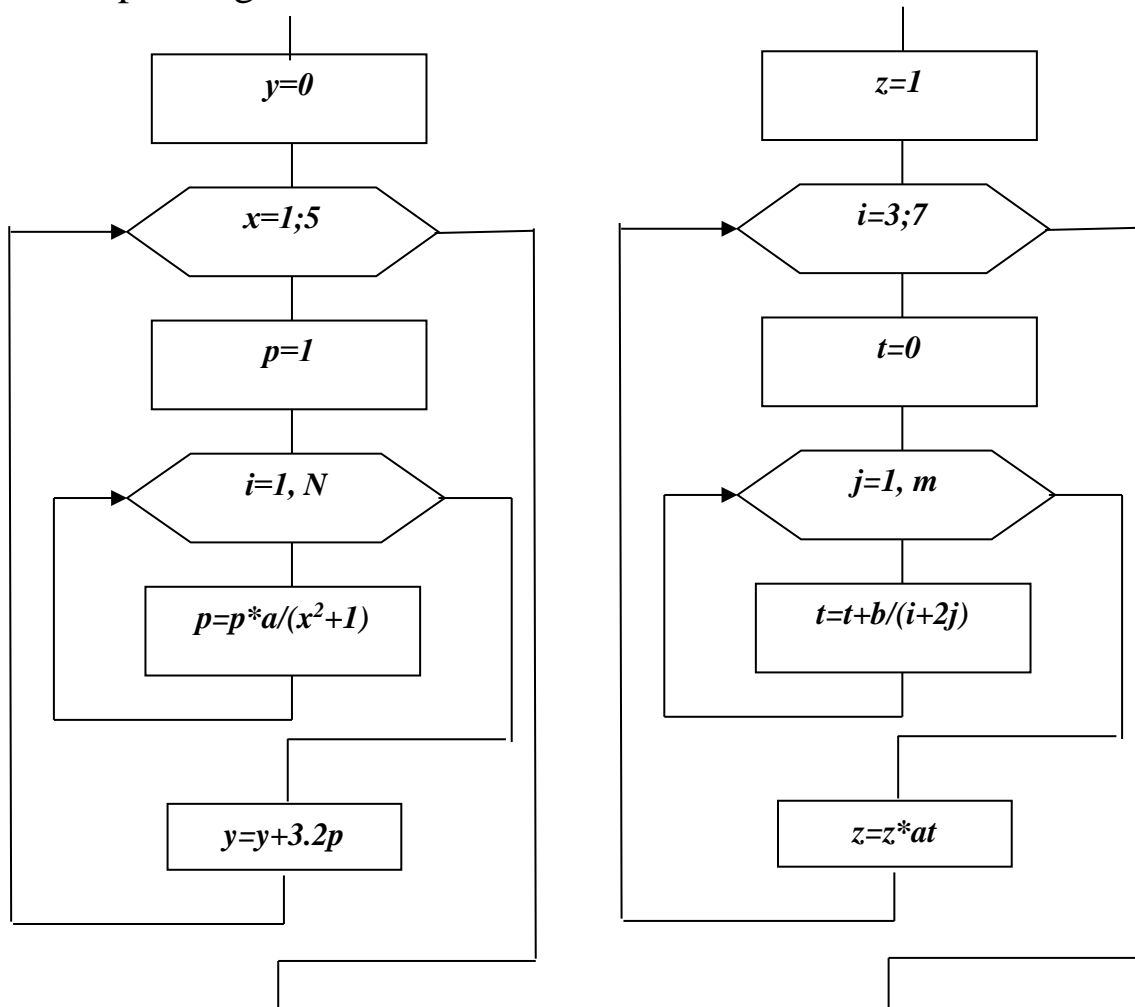


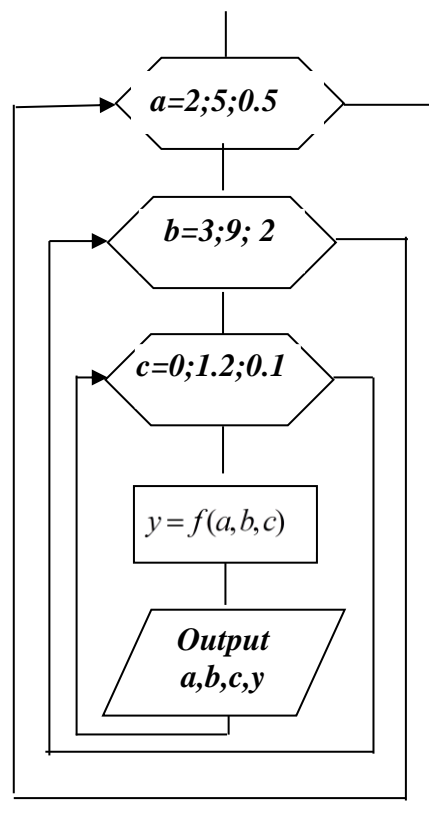
Fig. 4.12

Control questions and tasks

1. Define the concept of "loop", "cyclic calculating process".
2. Give a classification of loops by:
 - the way of setting repetitions;
 - the location of the loop exit condition;
 - grade of nesting.
3. Define the concept of "loop parameter".
4. What steps should be taken to make a simple arithmetic loop?
5. Define the concept of "recurrent expression".
6. Give examples of recurrence relations.
7. Provide loop structures with precondition and postcondition.
8. Define the concept of "nested cyclic calculating process".
9. Specify principles for nested loop construction.
10. For the algorithm below, write down the terms of the tasks they are accomplishing:



11. For the algorithm below, write down the term of the task they are accomplishing.
12. How determined the number of repetitions of calculating operators in nested cyclic processes (give the formula)?
13. Is the depth of nesting of cyclical processes limited?
14. For what tasks are used nested cyclic processes?
15. For what purposes can recursive relations be used in nested cyclic processes?
16. Give examples of tasks conditions that contain nested cyclic calculating processes.
17. For the algorithm below, calculate the number of "y" values that will be displayed.



18. Define the concept of "iterative cyclic calculating process".
19. Give the main differences between iterative and arithmetic loops.
20. When is the iteration loop completed and what should be provided?
21. What should be the numeric series for the iterative process of finding its sum to coincide? Provide a diagram of the value of a row element from its sequence number.
22. When the value of a row element is checked when the iteration process completes, and when – module of this value?

23. Provide an algorithm for calculating the value of a function Y , which is the sum of the elements of an infinite number that matches:

$$y = \frac{2}{1+a} - \frac{3}{2+a^2} + \frac{4}{3+a^3} - \frac{5}{4+a^4} + \dots + \frac{i+1}{1+a^i} + \dots$$

24. Define the concept of "the largest" (MAX) and "the smallest" (MIN) the value of the function per segment.

25. What type of algorithm is used to find the MAX and MIN values of a function?

26. What is a local minimum?

27. What is the global maximum?

28. Which function is called continuous?

29. * The cycles describe:

- 1) branching processes;
- 2) linear processes;
- 3) iterative processes;
- 4) regular processes.

30. Can a step in the regular cycle be negative?

- 1) yes, it can;
- 2) no, it can't;
- 3) it can, but only if it is an integer;
- 4) it depends on the BASIC version.

31. Can a step in a regular cycle be a fractional number?

- 1) yes, it can;
- 2) no, it can't;
- 3) it can, but only if it is a positive number;
- 4) it depends on the BASIC version.

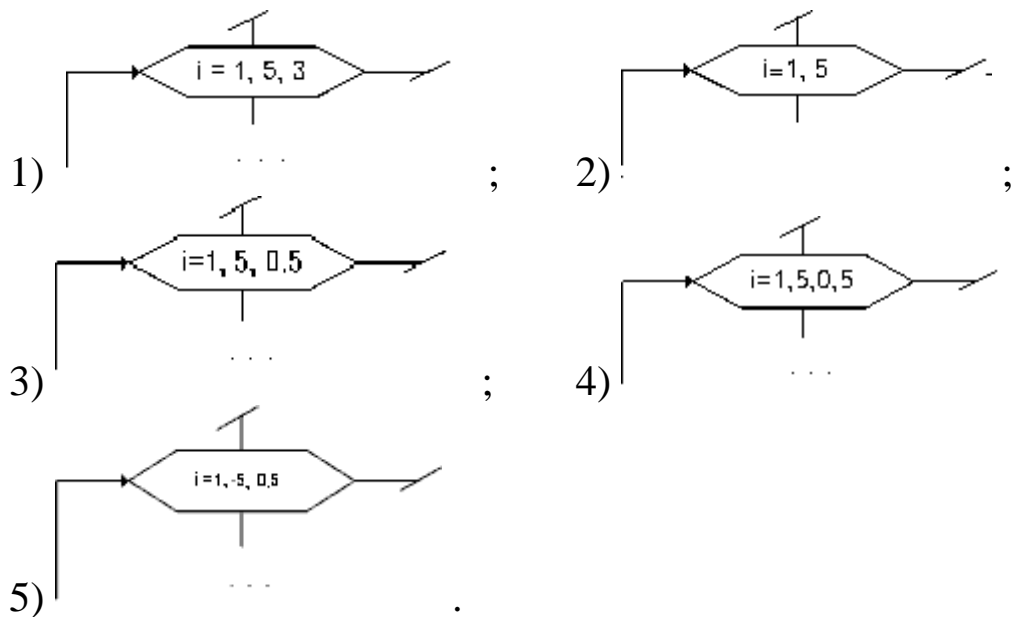
32. *Cycles by location of the condition of leaving the loop are divided into:

- 1) prerequisite loops;
- 2) fasting loops;
- 3) regular loops;
- 4) iterative loops;
- 5) simple loops;
- 6) nested loops.

33. *Regular loop is:

- 1) a loop with no calculated number of repetitions;
- 2) in which using arithmetic actions (+,-,*,/);
- 3) with a clearly specified number of repetitions;
- 4) in which the loop parameter takes only integer values;
- 5) with the estimated number of repetitions.

34. *Specify the correct recorded modifiers:



35. The results of the calculations are NOT:

- 1) Intermediate;
- 2) Result;
- 3) Input.

36. *Specify two known types of cycles:

- 1) with an unknown number of repetitions;
- 2) with a known number of repetitions;
- 3) with an unknown condition for exiting the cycle;
- 4) with an branched condition of exit from the cycle.

37. *Specify the types of cycles that you know of:

- 1) regular;
- 2) unconditional;
- 3) with the transconditional;
- 4) with the mezacondition;

- 5) with the precondition;
- 6) with the condition of fasting.

38. *The nesting loops are divided into:

- 1) prerequisite loops;
- 2) fasting loops;
- 3) simple loops;
- 4) nested loops;
- 5) regular loops;
- 6) iterative loops.

39. *Loops for the way repetitions are organized are divided into:

- 1) regular loops;
- 2) prerequisite loops;
- 3) fasting loops;
- 4) iterative loops;
- 5) nested loops.

40. A feature of the prerequisite cycle is the fact that:

- 1) actions in the loop body may not be performed any time;
- 2) actions in the loop body must be performed at least once;
- 3) actions in the loop body must be performed before printing;
- 4) all values in the loop must be calculated before the start.

41. A feature of the loop with the condition of fasting is the fact that:

- 1) actions in the loop body must be performed at least once;
- 2) actions in the loop body may not be performed any time;
- 3) actions in the loop body must be performed after data entry;
- 4) all values in the loop must be calculated after the end.

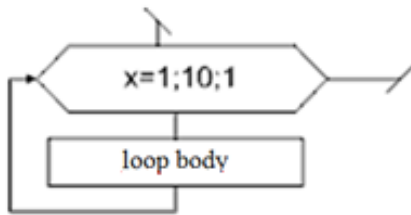
42. The number of repetitions in regular cycles may NOT be:

- 1) a predefined integer;
- 2) a predefined real number;
- 3) null;
- 4) negative (for example, if step = -1).

43. The step in regular cycles should NOT be:

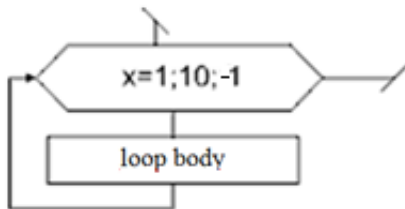
- 1) variable;
- 2) constant;
- 3) arithmetic expression;
- 4) negative number;
- 5) positive number;
- 6) null;
- 7) fractional;
- 8) real number.

44. Specify how many times the loop body defined by the parameter x will be executed?



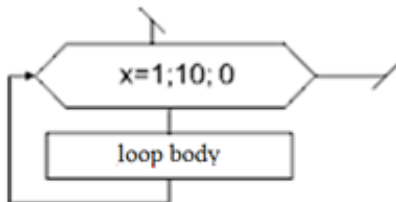
1) 10; 2) 9; 3) never; 4) an infinite number of times (looping); 5) 1.

45. Specify how many times the loop body defined by the parameter x will be executed?



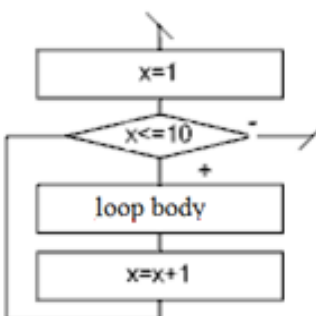
1) 10; 2) 9; 3) never; 4) an infinite number of times (looping); 5) 1.

46. Specify how many times the loop body defined by the parameter x will be executed?



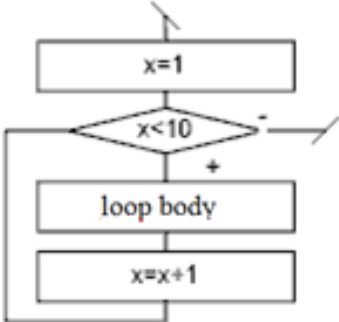
1) 10; 2) 9; 3) never; 4) an infinite number of times (looping); 5) 1.

47. Specify how many times the loop body defined by the parameter x will be executed?



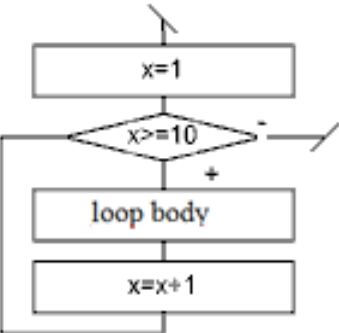
1) 10; 2) 9; 3) never; 4) an infinite number of times (looping); 5) 1.

48. Specify how many times the loop body defined by the parameter x will be executed?



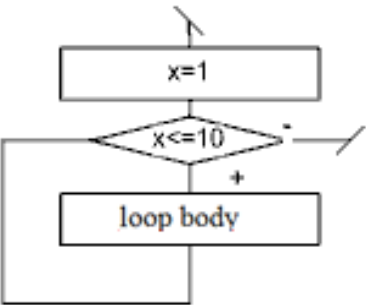
1) 10; 2) 9; 3) never; 4) an infinite number of times (looping); 5) 1.

49. Specify how many times the loop body defined by the parameter x will be executed?



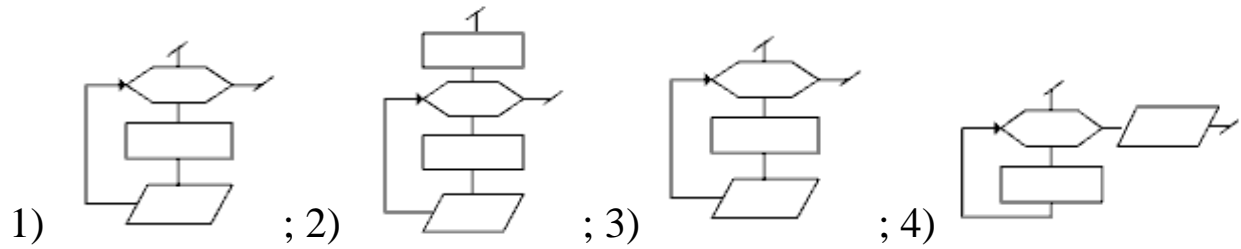
1) 10; 2) 9; 3) never; 4) an infinite number of times (looping); 5) 1.

50. Specify how many times the loop body defined by the parameter x will be executed?

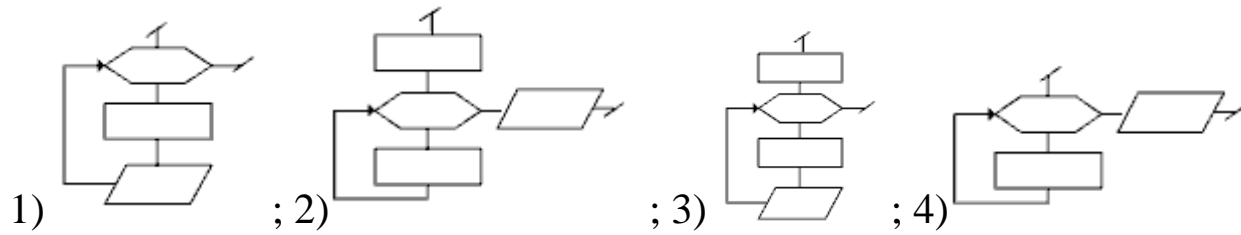


1) 10; 2) 9; 3) never; 4) an infinite number of times (looping); 5) 1.

51. Specify an algorithm diagram that describes the task "tab function":



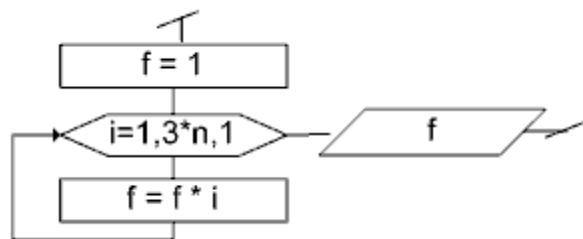
52. Specify an algorithm diagram that describes the task "finding sum, product (factorial)":



53. If $y = \sum_{x=5}^8 x + \prod_{i=1}^n \frac{n!}{i}$ the nested loop is used in the calculation of:

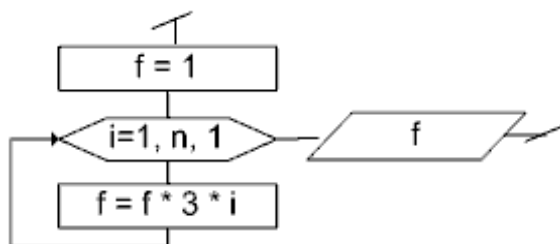
- 1) sum; 2) product; 3) factorial; 4) no nested loops.

54. Specify the calculation described by the scheme:



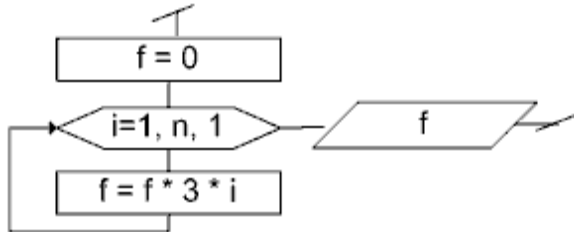
- 1) $f = \sum_{i=1}^n 3^i$ 2) $f = \prod_{i=1}^n 3^i$ 3) $f = (3n)!$ 4) there is no such expression.

55. Specify the calculation described by:



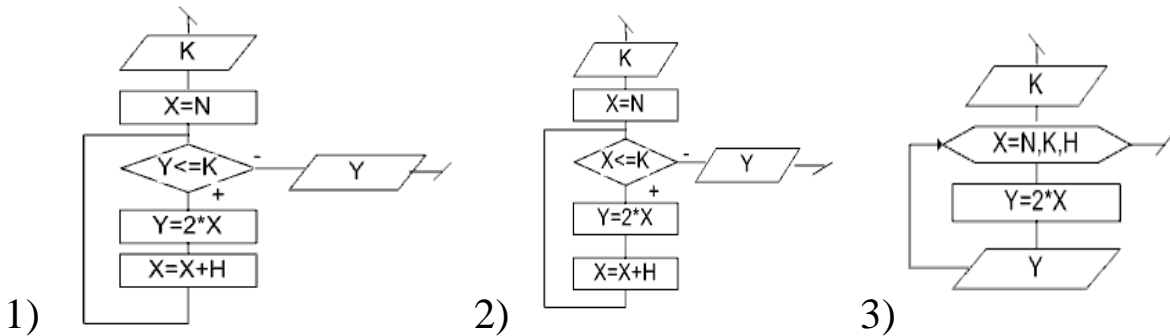
- 1) $f = \sum_{i=1}^n 3^{*i}$ 2) $f = \prod_{i=1}^n 3^{*i}$ 3) $f = (3n)!$ 4) there is no such expression.

56. Specify the calculation described by:



- 1) $f = \sum_{i=1}^n 3^{*i}$ 2) $f = \prod_{i=1}^n 3^{*i}$ 3) $f = (3n)!$ 4) there is no such expression.

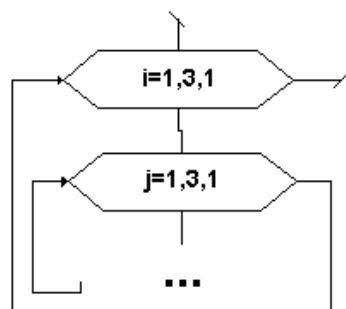
57. Specify cycles that are NOT regular:



58. The number of repetitions in a nested loop is defined as:

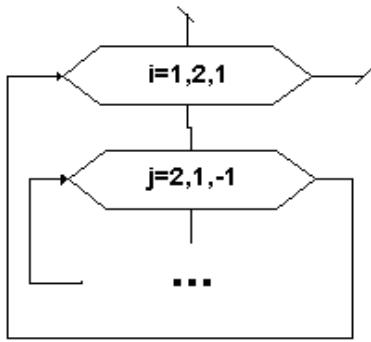
- 1) the sum of the number of repetitions of external and internal loops;
- 2) the difference between the number of repetitions of the external and the sum of the number of repetitions of the internal loop;
- 3) the product of the number of repetitions of external and internal loops.

59. How many times does the loop body run?



- 1) 3; 2) 6; 3) 9; 4) ∞ ; 5) never (loop organized incorrectly).

60. How many times does the loop body run?



1) 1; 2) 4; 3) 2; 4) ∞ ; 5) never (loop organized incorrectly).

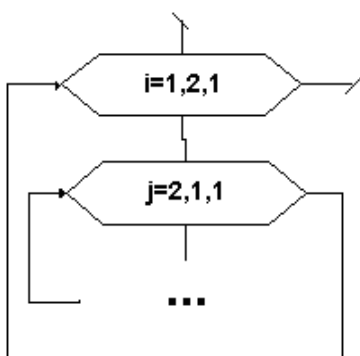
61. Specify tasks that use nested loops:

1) $y = \sum_{x=0}^{10} x \prod_{i=1}^{11} \frac{1}{i}$; 2) $y = \sum_{x=0}^{10} \prod_{i=1}^n \frac{x}{i!}$; 3) $y = 11! + \sum_{x=0}^{10} x + \prod_{i=1}^{11} \frac{1}{i}$

62. Nested is a loop that contains:

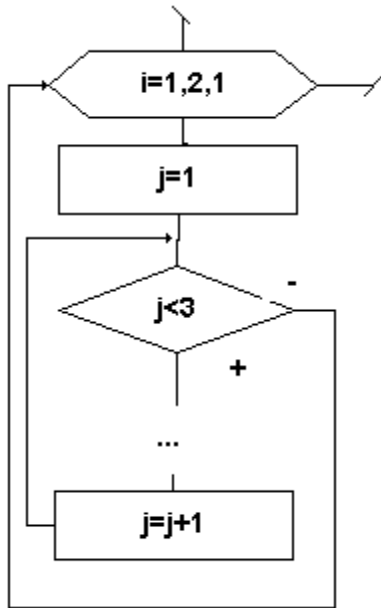
- 1) one or more other loops;
- 2) one or more branches;
- 3) only the calculation of the sum or application.

63. How many times does the loop body run?



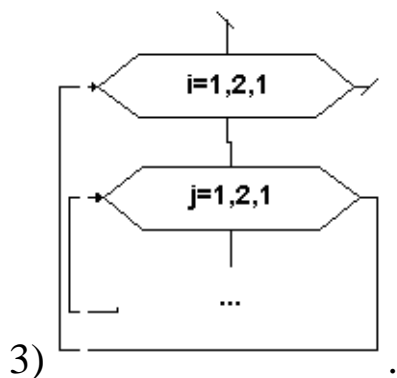
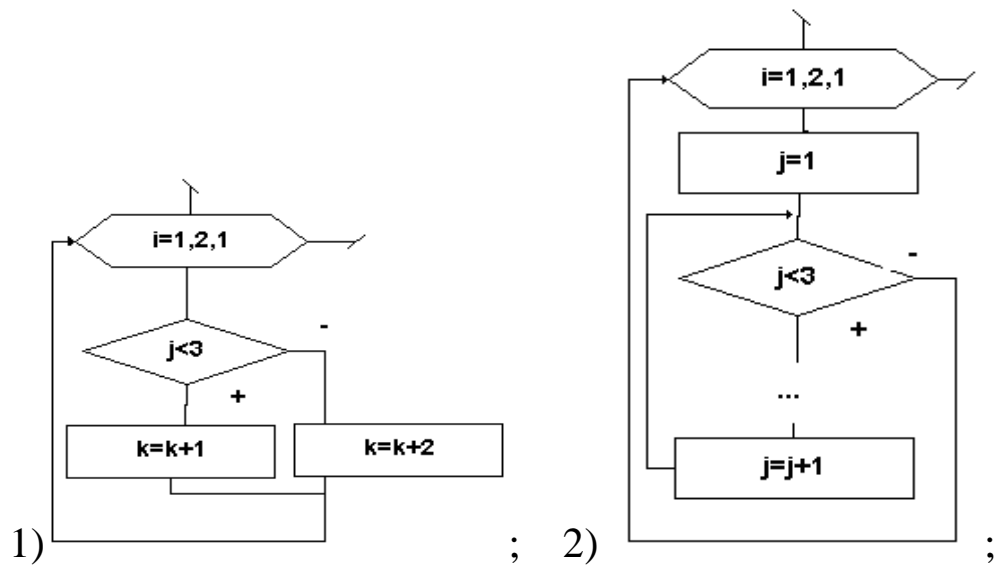
1) 4; 2) 1; 3) 2; 4) ∞ ; 5) never (loop organized incorrectly).

64. How many times does the loop body run?

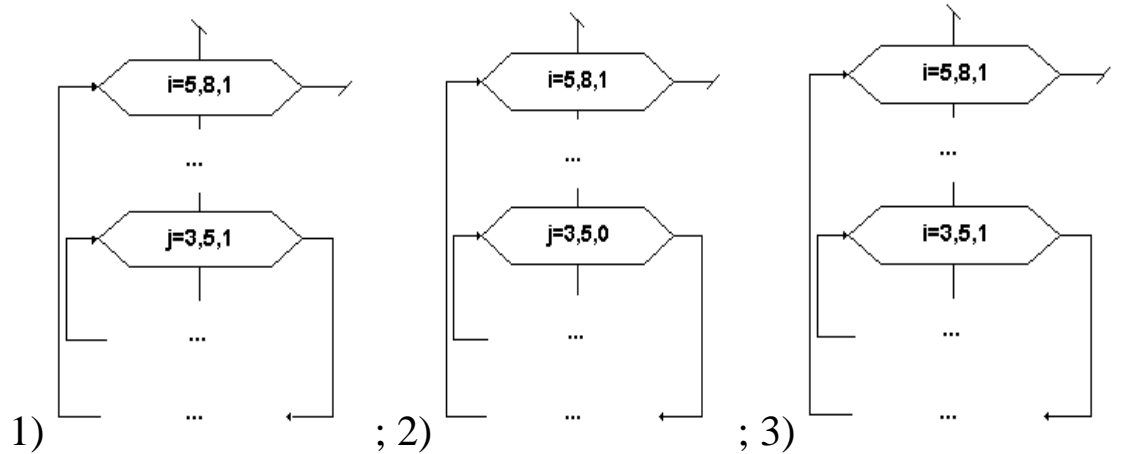


1) 4; 2) 1; 3) 2; 4) ∞ ; 5) never (loop organized incorrectly).

65. Specify cycles that are NOT nested:



66. Specify properly organized schemas for loops:



67. Specify tasks that do NOT use nested loops:

1) $y = \sum_{x=0}^{10} \left(\frac{1}{\sum_{i=1}^n \frac{i}{x^2+1}} + \prod_{i=1}^n \frac{x}{i!} \right)$; 2) $p = \sum_{i=1}^n \frac{i}{x^2+1}$; $q = \prod_{i=1}^n \frac{x}{2i!}$;

3) $p = \sum_{j=1}^n \frac{j}{x^2+1}$; $f = i!$; $q = \prod_{k=1}^n \frac{x}{k}$.

68. Specify tasks that use nested loops:

- 1) $y = a * \sin(x + a)$ if $x \in [0; 1.7]$, $hx = 0.1$, $a = 0.2$;
- 2) $y = a * \sin(x + a)$ if $a = 0.2$;
- 3) $y = a * \sin(x + a)$ if $x \in [0; 1.7]$, $hx = 0.1$, $a \in [3; 7]$, $ha = 3$.

69. Specify tasks that use nested loops:

1) $y = \sum_{x=5}^8 x + \prod_{i=1}^n \frac{n}{i!}$; 2) $y = \sum_{x=5}^8 x + \prod_{i=1}^n \frac{n!}{i}$;

3) $y = \sum_{x=5}^8 x + \prod_{i=1}^{n!} \frac{n}{i}$; 3) $y = \sum_{x=5}^{8!} x + \prod_{i=1}^n \frac{n}{i}$.

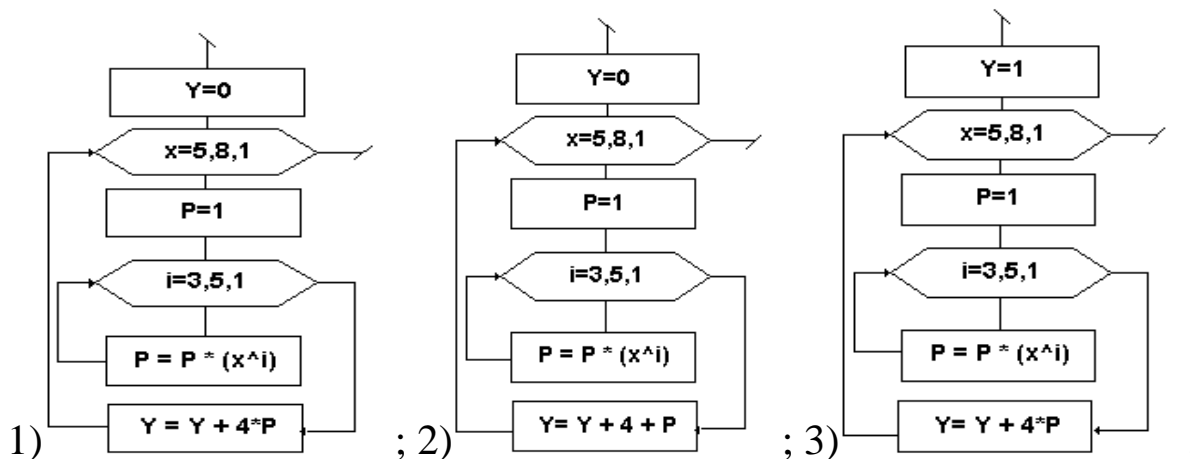
70. Specify tasks that use nested loops:

1) $y = \sum_{x=5}^n \frac{i}{x^2+n!}$; 2) $y = \prod_{i=1}^5 2i \sum_{x=1}^4 a+j$;

$$3) y = \sum_{x=5}^8 x + \prod_{i=1}^n \frac{n!}{i};$$

$$4) y = \sum_{x=5}^8 x! + \prod_{i=1}^n \frac{n}{i}.$$

71. Specify tasks that do NOT use nested loops:



72. If $y = \sum_{x=5}^8 x + \prod_{i=1}^n \frac{n!}{i}$ the nested loop is used in the calculation of:

- 1) sum; 2) product; 3) factorial; 4) no nested loops.

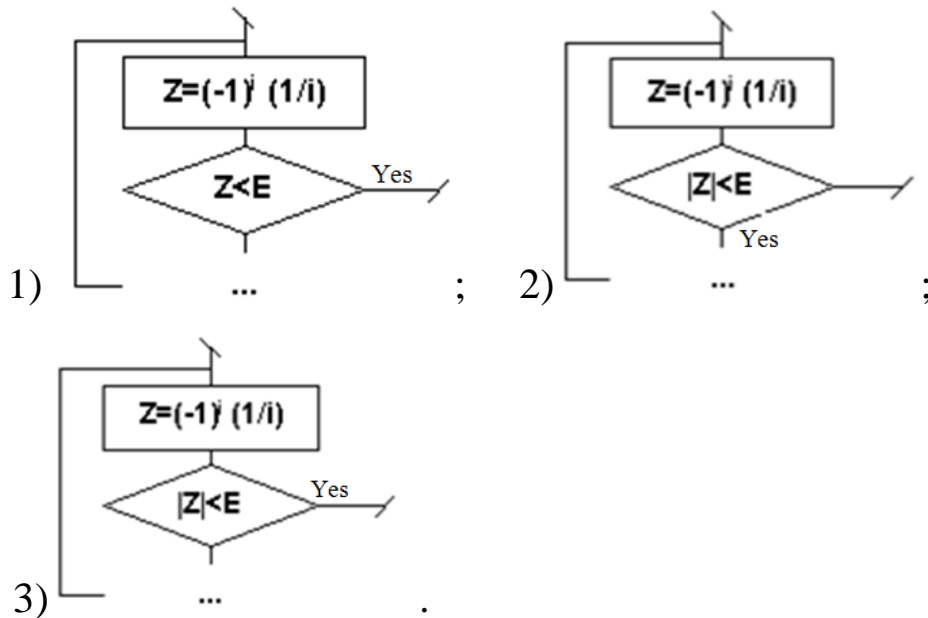
73. If $y = \sum_{x=5}^8 x + \prod_{i=1}^n \frac{n}{i!}$ the nested loop is used in the calculation of:

- 1) sum; 2) product; 3) factorial; 4) no nested loops.

74. If $y = \sum_{x=5}^{8!} x + \prod_{i=1}^n \frac{n}{i}$ the nested loop is used in the calculation of:

- 1) sum; 2) product; 3) factorial; 4) no nested loops.

75. Specify the snippet where the loop exit is incorrectly organized (the task of calculating the sum of series $S = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \dots \pm \frac{1}{n}$, that converging with precision $E = 10^{-5}$):



76. Specify tasks that use nested loops:

1) $y = \sum_{i=1}^n 2 \frac{i+1}{\prod_{x=3}^n 3x}$; 2) $y = \prod_{i=1}^5 2i \sum_{x=1}^4 a+j^2$; 3) $y = \sum_{x=5}^8 x + \prod_{i=1}^n \frac{n!}{i}$.

77. Specify tasks that use nested loops:

1) $y = \sum_{i=1}^n 2 \frac{i+1}{\prod_{x=3}^n 3x^i}$; 2) $y = \prod_{i=1}^5 2i \sum_{x=1}^4 a+j^2$; 3) $y = \sum_{i=1}^n 2 \frac{i}{\prod_{x=3}^n 3x}$.

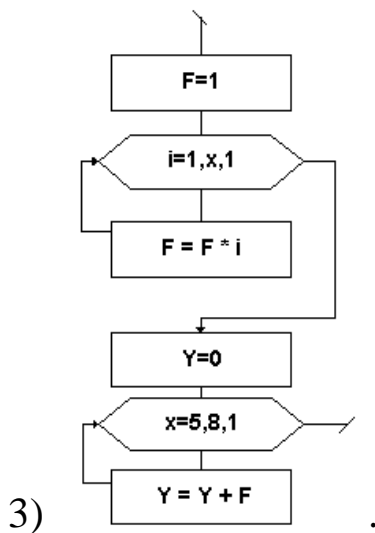
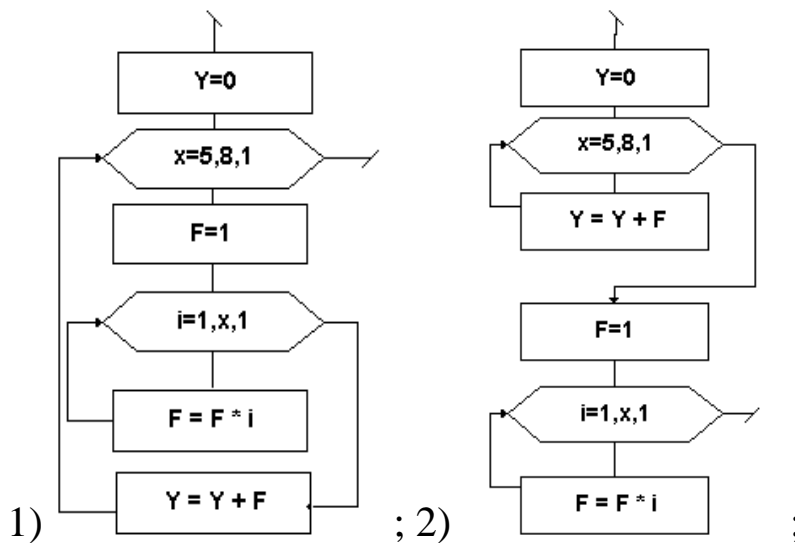
78. Is it possible to describe an iterative loop using a modifier?

1) no; 2) yes; 3) only if this is the loop with precondition.

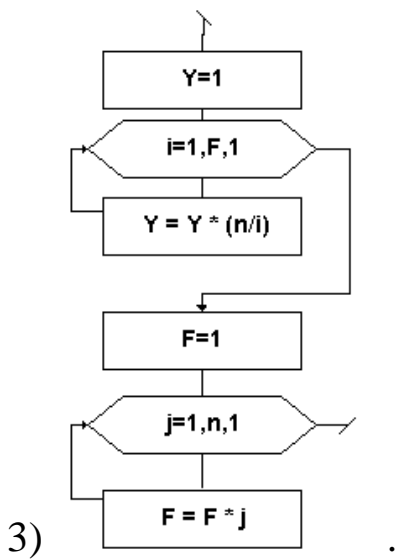
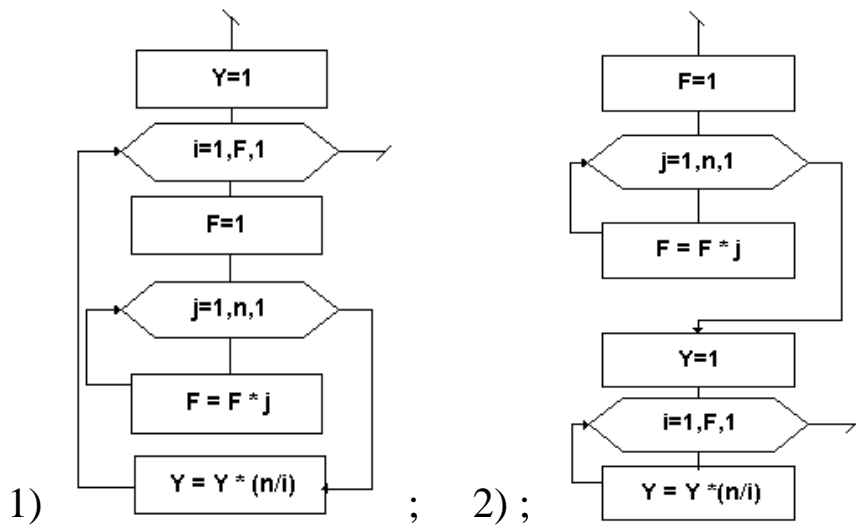
79. If $y = \sum_{x=5}^8 x! + \prod_{i=1}^n \frac{n}{i}$ the nested loop is used in the calculation of:

- 1) sum; 2) product; 3) factorial; 4) no nested loops.

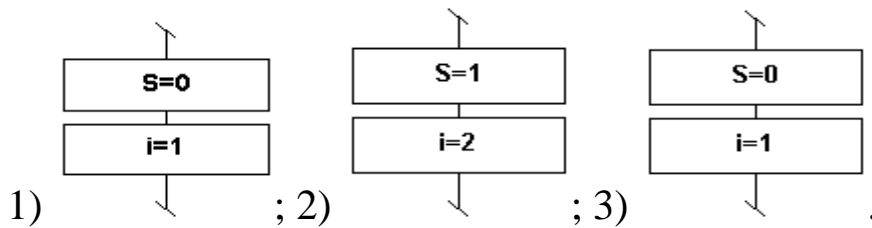
80. Specify the scheme for the calculation $y = \sum_{x=5}^8 x!$:



81. Specify the scheme for the calculation $y = \prod_{i=1}^n \frac{n!}{i}$:



82. Where is the error in "zeroing"? (the task of calculating the sum of series $S = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \dots + \frac{1}{i} + \dots$, that converging with precision $\epsilon = 10^{-5}$)

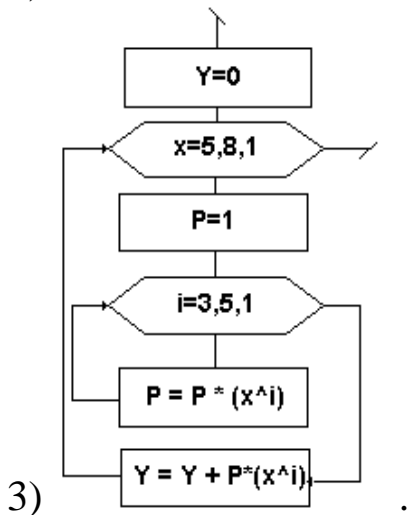
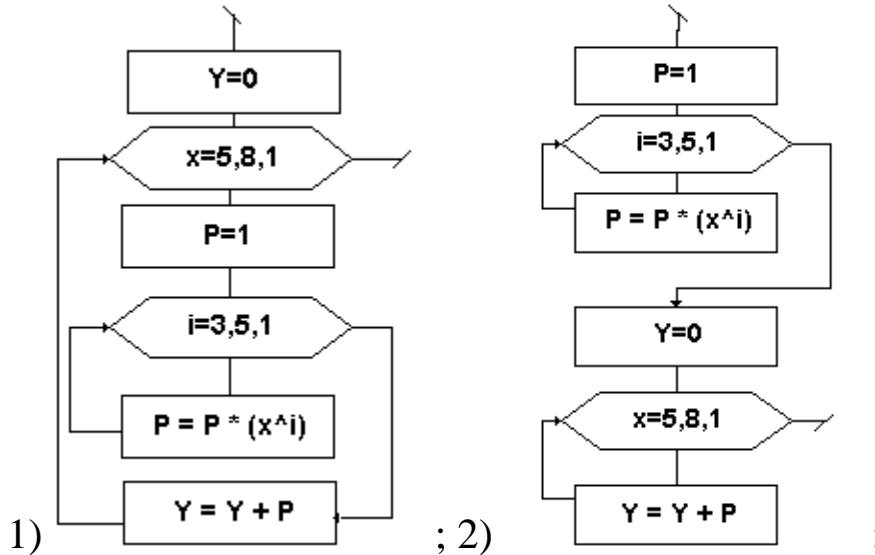


83. *Iterative is called:

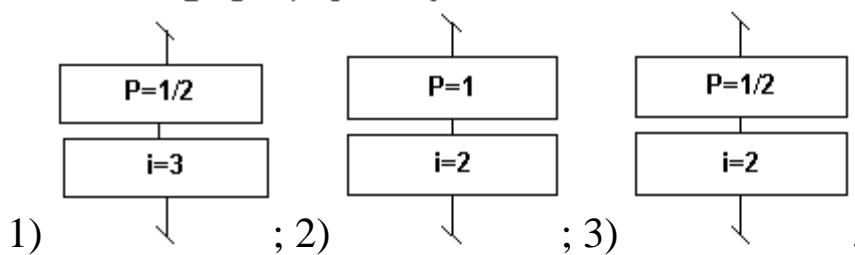
- 1) a cyclical process in which the number of repetitions is unknown in advance;
- 2) a cyclic process in which the number of repetitions is known in advance;
- 3) a recurring part of the program that uses the results of the previous step at each step;

4) the loop that ends as a result of an external interference with a program that contains that loop.

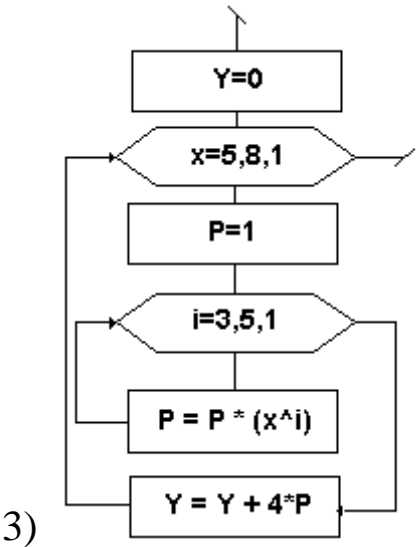
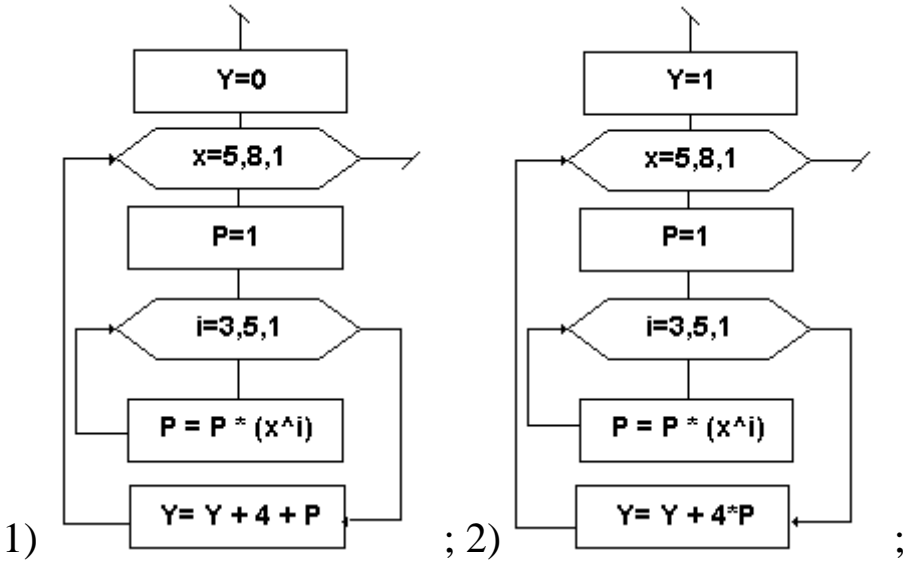
84. Specify the scheme for the calculation $y = \sum_{x=5}^8 \prod_{i=3}^5 x^i$:



85. Where is the error in "zeroing"? (the task of calculating the sum of series $P = \frac{1}{2} * \frac{1}{3} * \frac{1}{4} * \frac{1}{5} * \dots * \frac{1}{i} * \dots$, that converging with precision $\epsilon = 10^{-5}$)



86. Specify the scheme for the calculation $y = \sum_{x=5}^8 4 \prod_{i=3}^5 x^i$:



UNIT 5. DESIGNING ARCHITECTURES OF ARCHITECTURE PROCESSING

5.1. Concepts and main characteristics of the array

In the practice of engineering calculations, the solution of many tasks involves the processing of large sets of homogeneous data, for example, multiple results of measurements of the same physical quantity. If all the elements of an object are of the same type, then the variable denoted by these elements is homogeneous and can be represented as an array.

Array – a set of similar elements, ordered by the numerical values of the indices.

The name of the array indicate the all ordered set of elements in general.

The element of the array is called the index variable. A list of indexes is added to the name to indicate an individual element of the array which allows for access to a particular item.

Index – constant, variable or expression. The index value must be a positive integer (because it's a number), so it is always rounded down, transformed and stored in this form.

Index list – ordered sequence of indices, separated by commas. Each index has its own range of change, called the limit pair.

Index variables are handled according to the same rules as scalar (simple) variables.

All elements of the same array not only have a common name, but are usually placed in consecutive cells of computer memory.

The location of an element can be defined as one (one-dimensional arrays) or multiple values (multidimensional arrays) of indices.

The characteristics of the arrays are dimension and size.

The number of indices determines the dimension of the array.

The number of array elements determines the size of the array.

For example,

$A_i; Z_9; S_{j+2}; S_{k-3}$ – elements of one-dimensional arrays;

$A_{i,j}; Z_{9,j}; S_{2*i,j+2}$ – elements of two-dimensional arrays.

Array processing algorithms have blocks for processing array elements and relevant index values. Typically, these blocks are components of cyclic algorithms. This is because the processing of any element of an array is the same sequence of actions, and by organizing a loop by the number of array elements, you can handle all the elements using the same commands.

The characteristic of such algorithms – each repetition of loop processing should attend the next item. Therefore, they are called redirect loops – moving to the next element is carried out by increasing the memory address (usually on 1).

In mathematics, one-dimensional tables are called vectors or columns. For input, output and processing of one-dimensional arrays are used simple loops. *For example*, $X=\{x_i\}$, $i=1,5$ – one-dimensional array of 5 elements.

The two-dimensional table contains $n*m$ elements, and each element has two indexes. The first index shows the row number and the second shows the column number at which the element is intersected. In mathematics, two-dimensional tables are called matrices.

For example, $A=\{a_{ij}\}$, $i=1,n$; $j=1,m$. If $n=5$ and $m=4$ then

$A_{11}, A_{12}, A_{13}, A_{14}$

$A_{21}, A_{22}, A_{23}, A_{34}$

...

$A_{51}, A_{52}, A_{53}, A_{54}$

For input, output and processing of two-dimensional arrays typically used nested loops.

Working with any array consists of three stages:

- setting values of array elements;
- data processing according to the conditions of a specific task;
- output of calculation results.

5.2. Algorithms for processing one-dimensional arrays

Consider the implementation of input-output elements of a one-dimensional array.

On Fig. 5.1 shows a diagram of the algorithm input values one-dimensional array elements $K(20)$ of the keyboard. The user enters the array elements sequentially, starting with the index element $i=1$ (symbol 3). On Fig. 5.2 shows a diagram of the algorithm input values of the array

elements. The procedure of input-output is implemented by a cyclic algorithm.

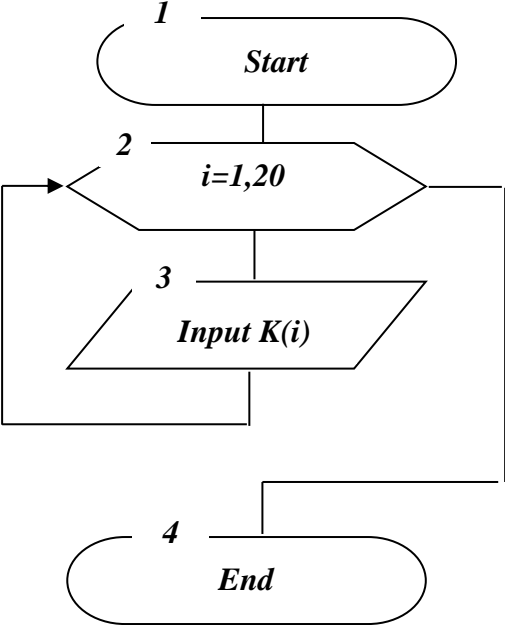


Fig. 5.1

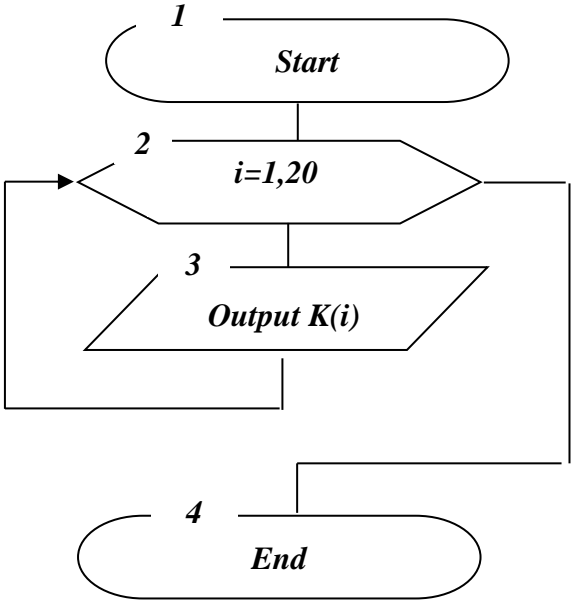


Fig. 5.2

On Fig. 5.1 i 5.2 arrays have a constant size of 20 elements.

For an array of any size, you must first enter the size of the array, and then – the array elements (Fig. 5.3).

If multiple arrays are the same size, they can be entered in one loop (Fig. 5.4).

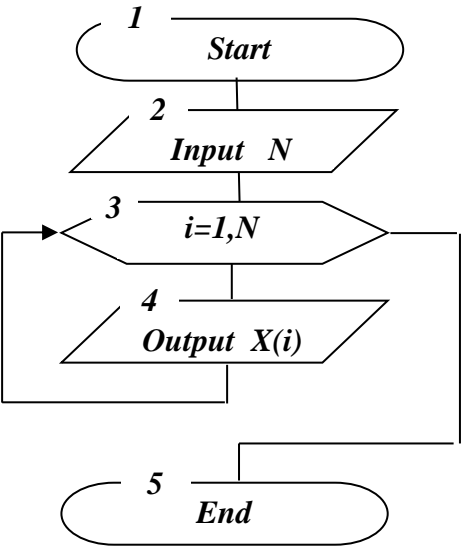


Fig. 5.3

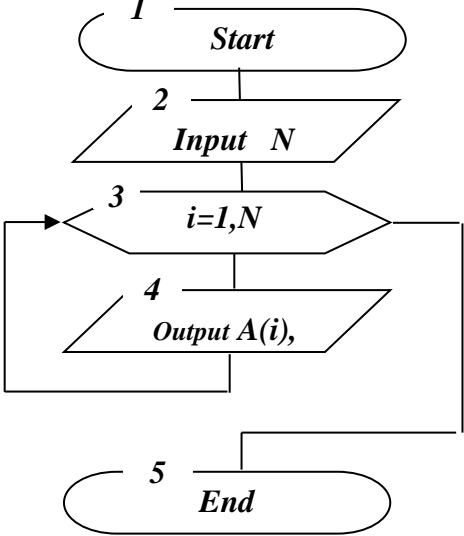


Fig. 5.4

For the sake of brevity, the following representation is allowed: symbols instead of 2-4 (Fig. 5.5) can be drawn:

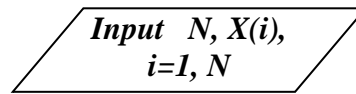


Fig. 5.5

For example. Schemes of algorithms for calculating sums and products of finite number of elements of an array are cyclic algorithms in which the cycle parameter is the order number of the element – i .

The diagram in Fig. 5.6 implements the calculation of the sum of the elements of the array $K(20)$. The sum is accumulated in variable S when the array is viewed sequentially. The initial state the amount of S , is zero, set symbol 3.

The arithmetic mean of the SR elements of the array is determined by the formula:

$$SR = \frac{S}{N},$$

where S – the sum of the elements,
 N – number of elements.

Symbol 6 calculates the SR value after determining the sum of 20 array elements. The diagram in Fig. 5.7 implements the calculation of the product of the elements of the array $K(20)$, located in pairs. It is similar to the previous one, except that the initial state of the accumulator for the product P is equal to one.

For example, Generate an array of arbitrary numbers using the random number generator RND, which is available in almost all algorithmic languages. To obtain such numbers, it is only necessary to specify the range $[a; b]$, in which it is necessary to obtain the numbers and the number of such numbers. Random number generator RND generates real numbers in the range $[0; 1]$, distributed by a evenly act. The formula for obtaining a number in a given range looks like $x = a + (b - a) * RND$.

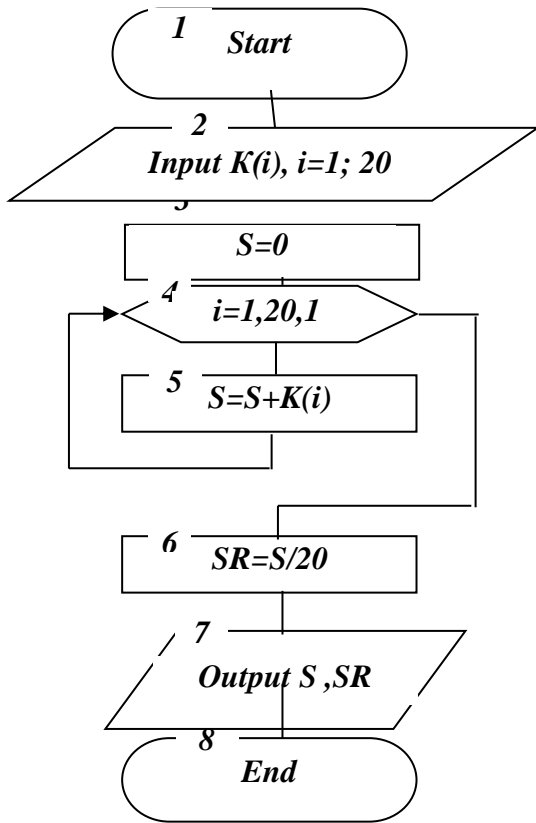


Fig. 5.6

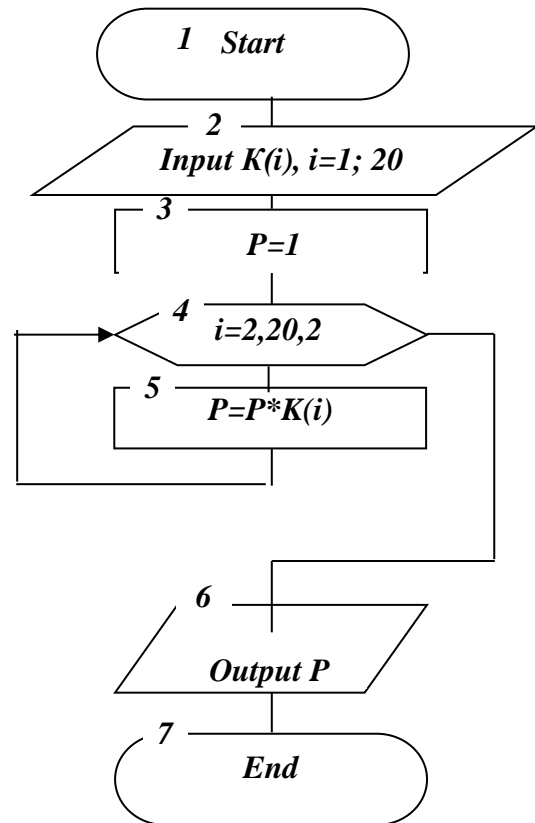


Fig. 5.7

If we make the process of obtaining numbers X cyclical, then we will be able to form an array X of a given size (Fig. 5.8).

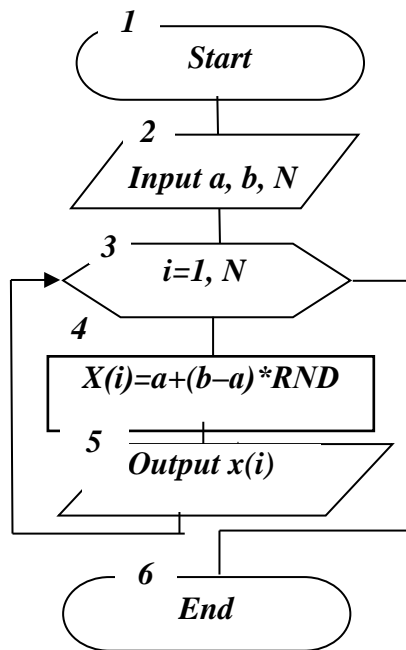


Fig. 5.8

For example, The algorithm shown in Fig. 5.9, allows the array elements $A(20)$ create two arrays X and Z.

Array X consists of negative array A, array Z – with elements from key feature $A(i) > 12$.

Sequential view of array elements A we find items that satisfy the search criteria (symbols 5, 6). Symbols 9 and 10 array elements A, recorded in the corresponding arrays. Serial number of the array element X is given by a variable j, to array Z – by variable k. Before writing the next element in each of the arrays, the values of the corresponding variable characters 7, 8 are increased by 1. The last values of the variables j, k will correspond to the total number of elements of the arrays X and Z.

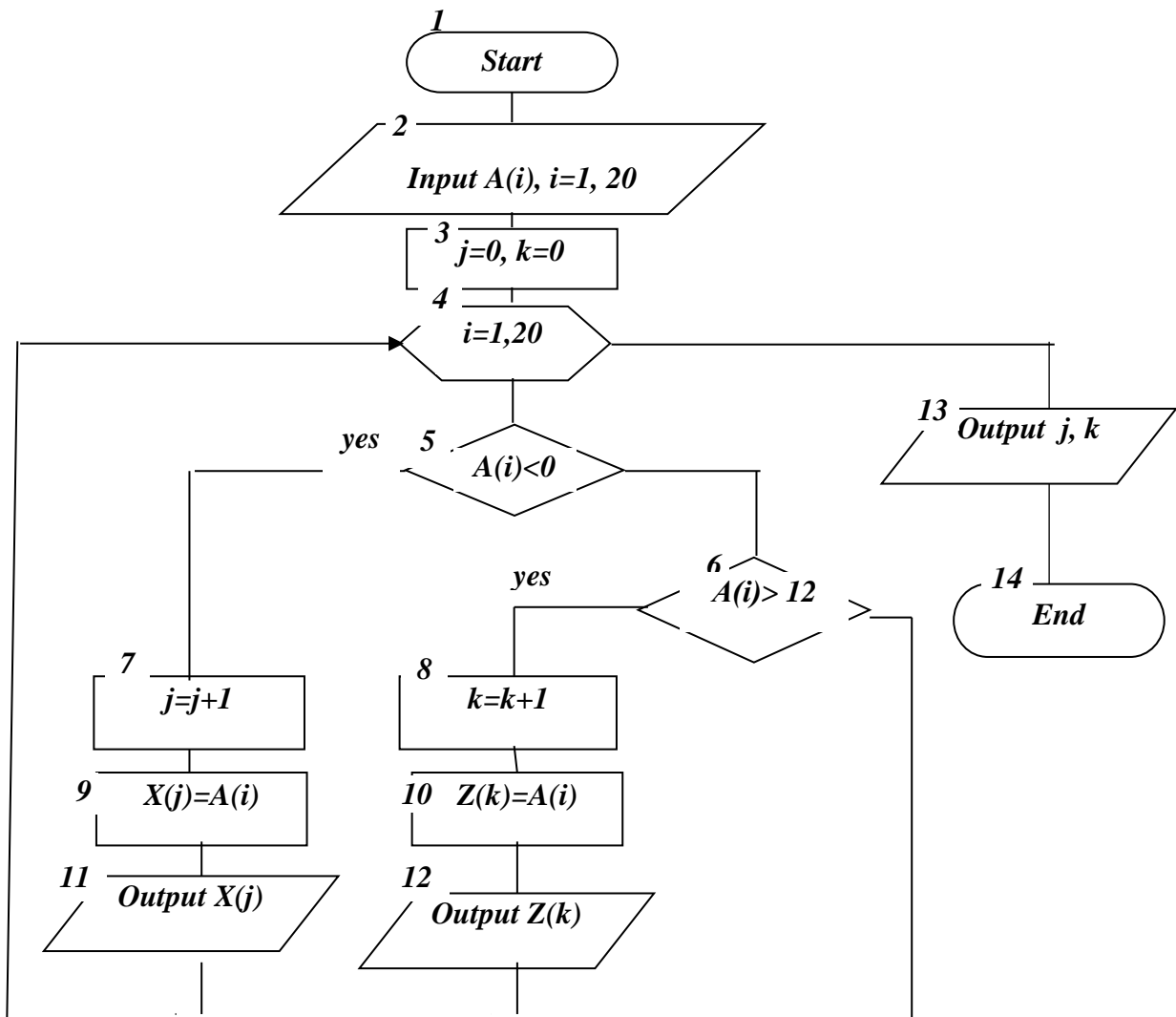


Fig. 5.9

For example. Array X consisting of N numerical elements. Develop a schema for finding the element's highest value (MAX variable).

In the algorithm (Fig. 5.10) in finding the maximum element of each element is compared with the variable MAX, which in block 3 is assigned the value of any element (in the example – element 1). Symbol 6 the value of the variable MAX will replace, if the element $X(i) > MAX$. After viewing all elements of the array, the value of the variable MAX will be equal to the greatest value $X(i)$ and will be displayed.

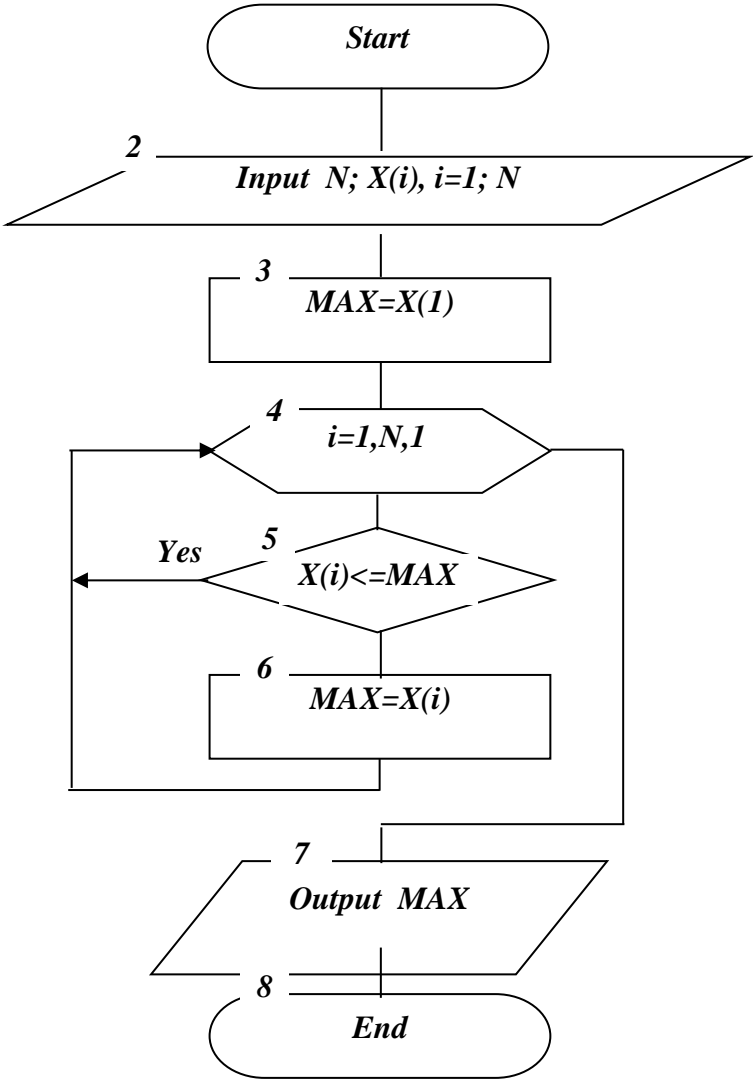


Fig. 5.10

5.3. Algorithms for processing two-dimensional arrays

Procedures for input, processing and output of elements of two-dimensional arrays are based on nested cyclic computing algorithms. The procedure of input and output of elements of an array can occur both on rows and on columns (swap parameters external and internal loops).

On Fig. 5.11 the algorithm of matrix input is given $X(M, N)$ arbitrary size by rows.

On Fig. 5.12 the algorithm of matrix input is given $Y(12, 6)$ by columns.

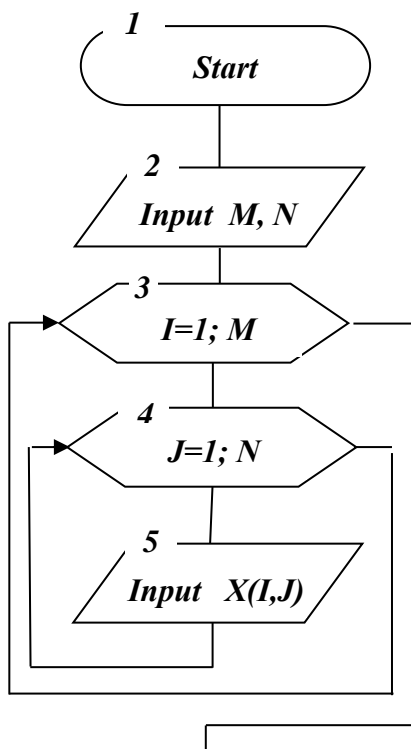


Fig. 5.11

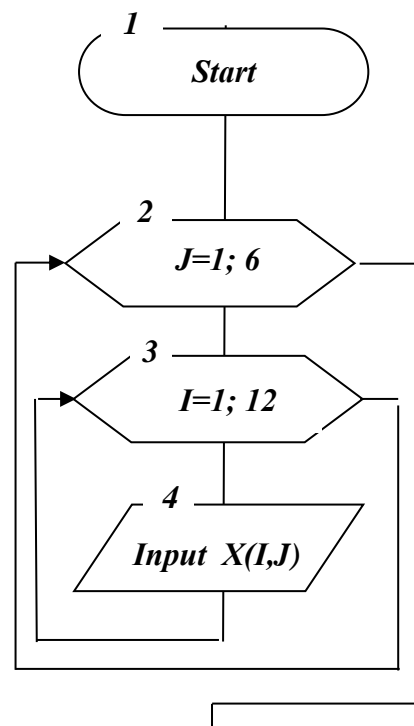


Fig. 5.12

Fig. 5.13 illustrates a brief entry of the matrix input procedure $X(M,N)$.

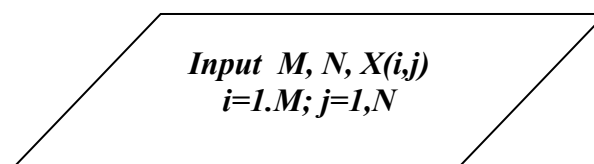


Fig. 5.13

For example. A two-dimensional array is specified $X(k, k)$ numeric values (k rows and k columns). Array $X(k,k)$ – square matrix.

Perform the following data processing in this array:

- determine the difference R between the product and the sum of the elements of the array satisfying the condition $X(i, j) > 10$;
- determine the arithmetic mean of the elements in each row and write them into a one-dimensional array $Y(k)$;
- determine the number of negative elements in each column of the array $X(k, k)$, place results in a one-dimensional array $Z(k)$;
- determine the value of the maximum element (max) on the main diagonal of the array $X(k, k)$

The algorithm for solving the problem is given in Fig. 5.14.

Can be distinguish the basic patterns of construction algorithms for processing two-dimensional arrays:

1. To handle elements in an array row, you must first open the row modifier.
2. To handle elements in an array column, you must first open the column modifier.
3. For processing elements in each row of the array:
 - open the row modifier;
 - set the initial value of the desired value;
 - open the column modifier.
4. For processing elements in each column of the array:
 - open the column modifier;
 - set the initial value of the required value;
 - open the row modifier.

Symbol 2 – input k – the size of the array X .

Symbols 3-5 – loop input elements of the array $X(k,k)$ by rows.

Symbol 6 – assigning initial values of sum $S=0$ and product $P=1$ elements that satisfy the condition $X(i,j)>10$.

Symbols 7-10 sum accumulation loop S and product P . Symbol 9 checks whether the performance of the condition $X(i,j)>10$.

Symbols 11, 12 – calculating and outputting the difference value R between the product P and the sum S .

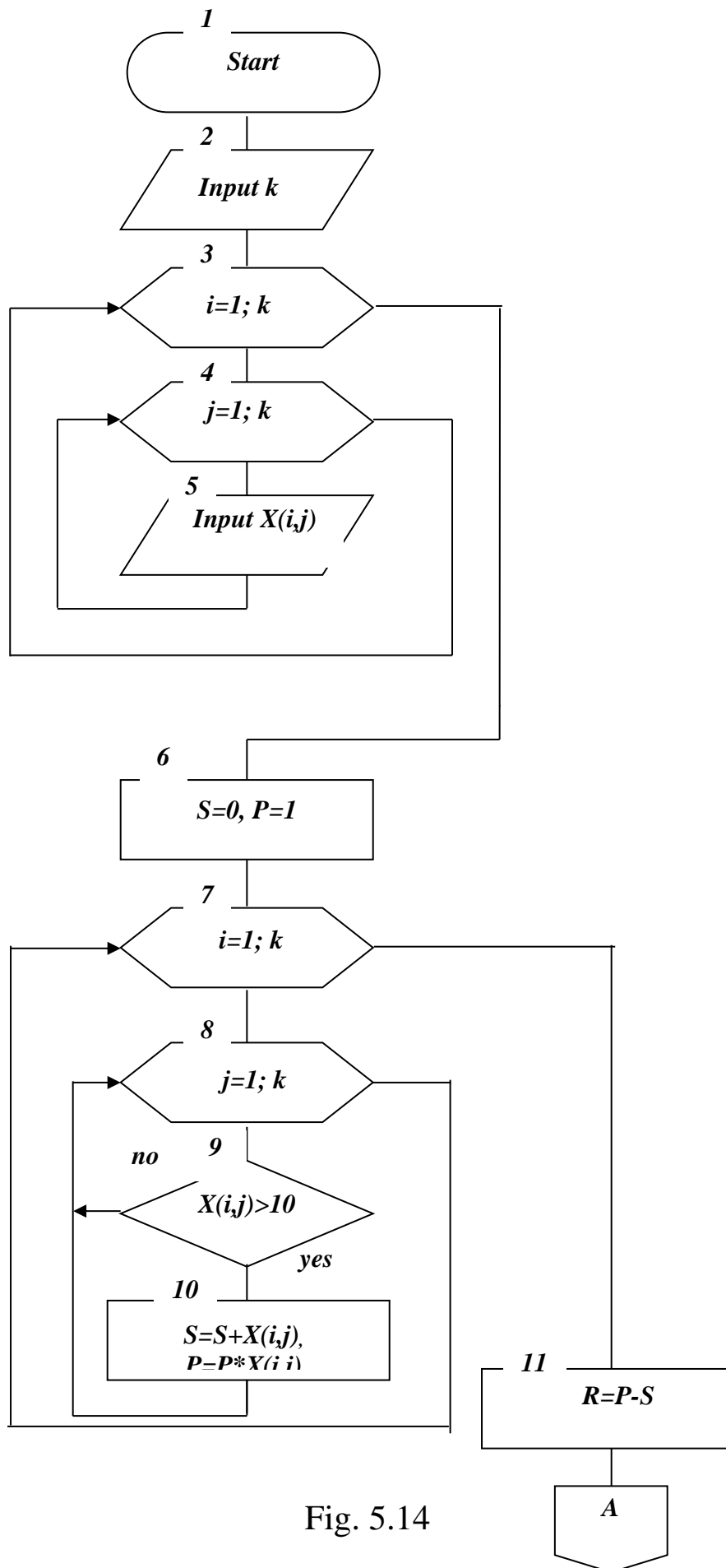


Fig. 5.14

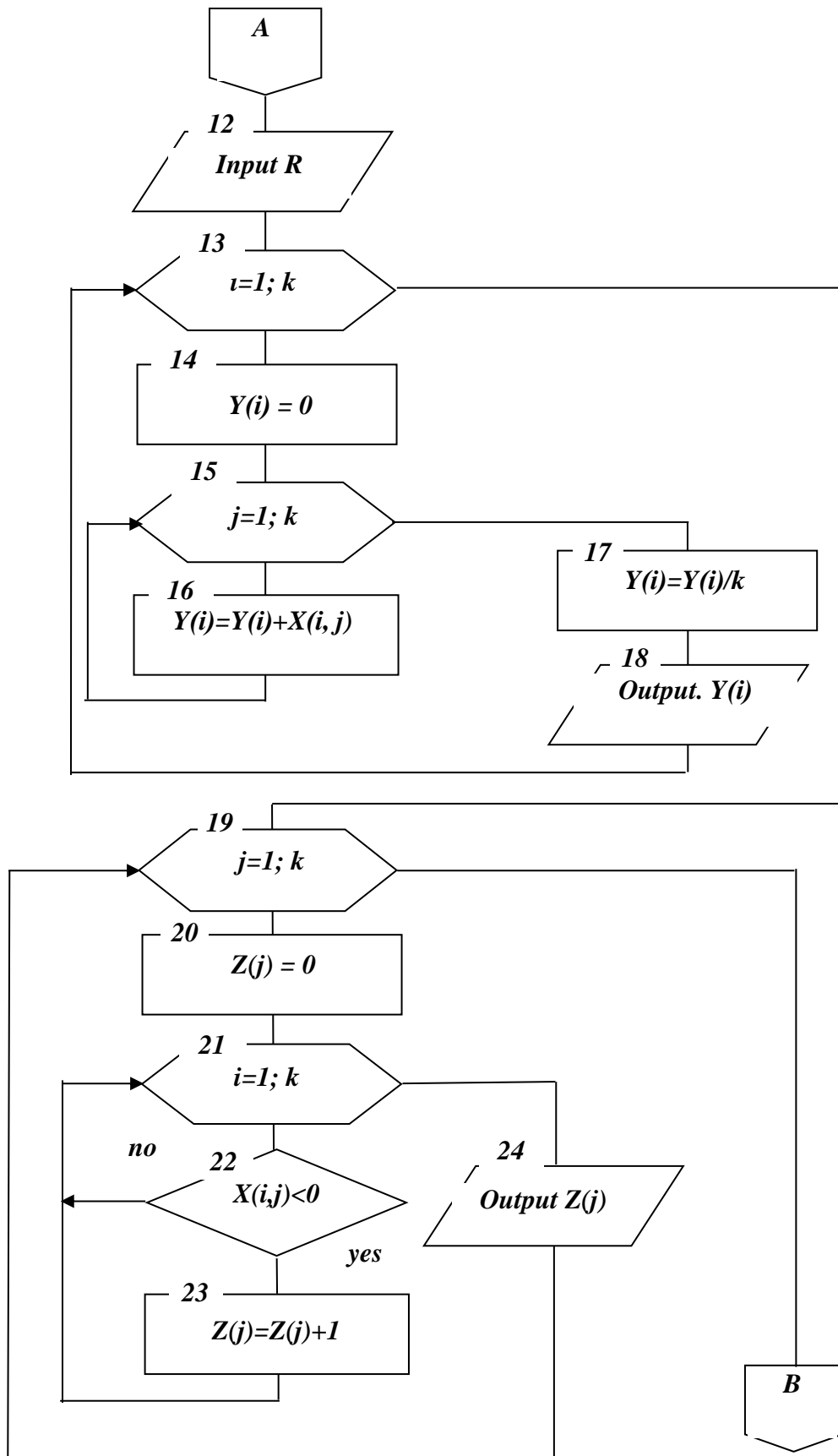


Fig. 5.14 (continuance)

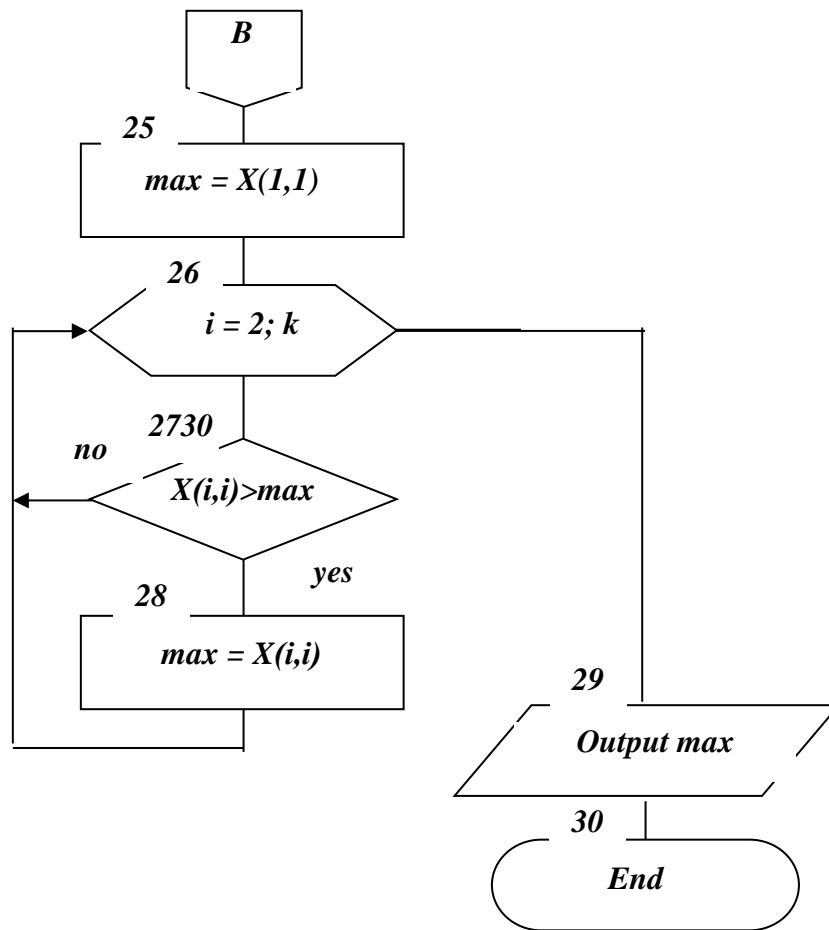


Fig. 5.14 (end)

Symbols 13-18 – calculating and outputting the values of the arithmetic mean of each row in the array $X(k,k)$. Processing is done in rows (parameter i in the external loop). Zeroing $Y(i)=0$ – symbol 14, occurs after setting the row number. Accumulating the sum of elements is due to the addition of value $X(i,j)$ in each internal loop – symbol 16. Calculation and output of arithmetic mean of row elements $Y(i)$, symbols 17, 18, occurs after processing of all row elements (exit from the inner loop, symbol 15).

Symbols 19-24 – calculating and displaying the number of negative elements in each column of the matrix $X(k,k)$. Processing is done by columns (parameter j in the external loop). Zeroing the number of negative elements $Z(j)=0$, symbol 20, occurs after the column number is specified j by the symbol 19. Accumulating the number of elements is due to the addition of 1 in each cycle if the condition $X(i,j)<0$.

Symbols 25-29 – determine and output the value of the maximum element on the main diagonal of the matrix. The initial value of the

maximum element is the value of the element $X(1,1)$. The comparison of the values of the elements of the matrix with the current maximum value occurs from the second element ($i=2$; k , symbol 26). Because the elements of the main diagonal have the same indexes $X(i,i)$, then the search for the maximum element occurs in a simple loop with one parameter i .

5.4. Algorithms for sorting arrays for a given feature

Sorting an array – sorting elements in ascending or descending order. Sorting algorithms vary in complexity and efficiency – depending on the length of time of the array (Fig. 5.15).

Sorting by choice. Search for max element number. It replaces the last element of the array, and the last element contains the maximum element (swapped). Then part of the array is taken without the last element (the last element has finally taken the place of the maximum element). With this residue, the same is done as with the output array. And so until the length of the last residue is equal to one (Fig. 5.16).

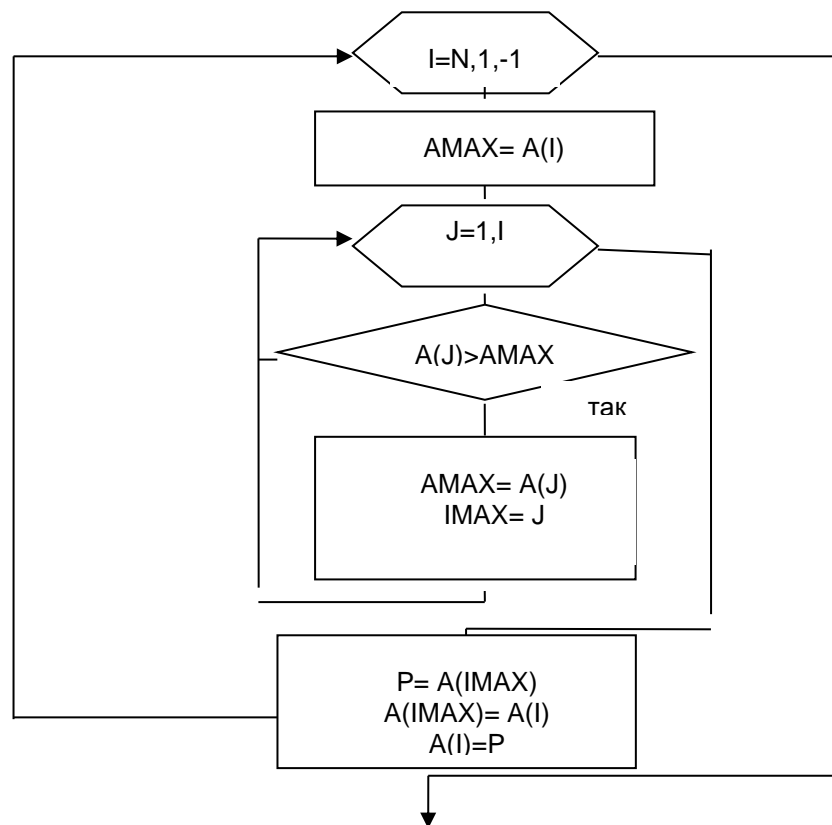


Fig. 5.15. Sorting by choice

Intuitive sorting. Adjacent elements are compared in pairs.

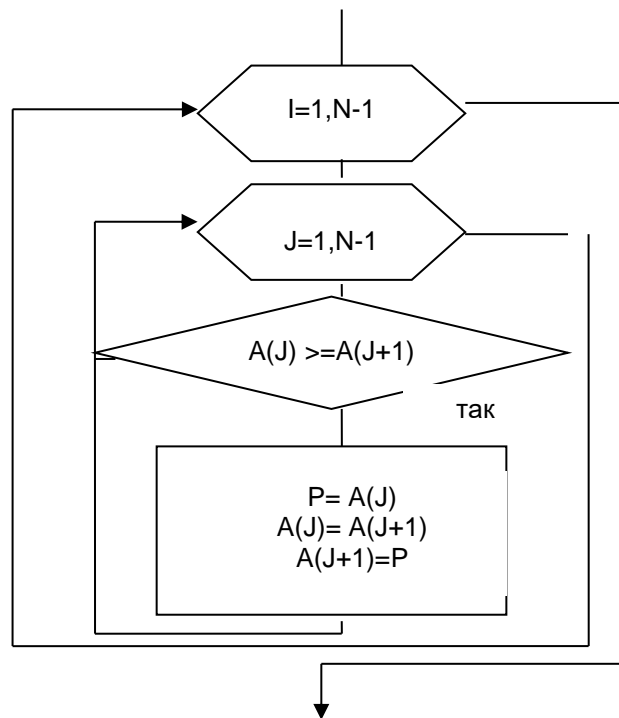


Fig. 5.16 Intuitive array sorting.

Each comparison is checked – if the previous element is greater than the next, they are reversed. One passage contains $(n-1)$ -to check. Then the iteration is repeated: totally $(n-1)$ iteration. Works slowly.

Bubble method. External loop – set the number of iterations; internal loop – set the number of comparisons. Consistently compare the i -th element ($i=1,N$) with the following, starting from $i+1$ -th to the end. If a smaller element is found, it is replaced by an i -th. As a result, each iteration the smaller (“lighter”) are moved to the beginning of the array – “go up like bubbles of air” (Fig. 5.17).

This method is worse than selection sort method, because the exchange of values occurs every time the comparison condition is met, however, it is effective when sorting an almost sorted array.

Improved bubble method. The sign of permutations is set to zero (ordering indicator) – $H=0$. Adjacent elements of the array are compared in pairs. At each comparison makes a check – if the previous element is greater than the next, they are reversed, thus set $H=1$. When all comparisons are made, the condition is checked $H=1$, if it is done, then pairwise comparisons are repeated (Fig. 5.18).

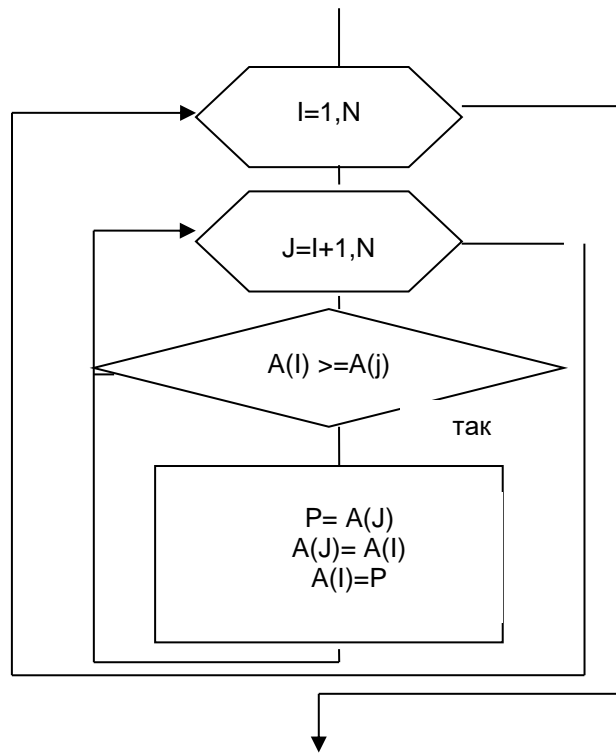


Fig. 5.17. Bubble method

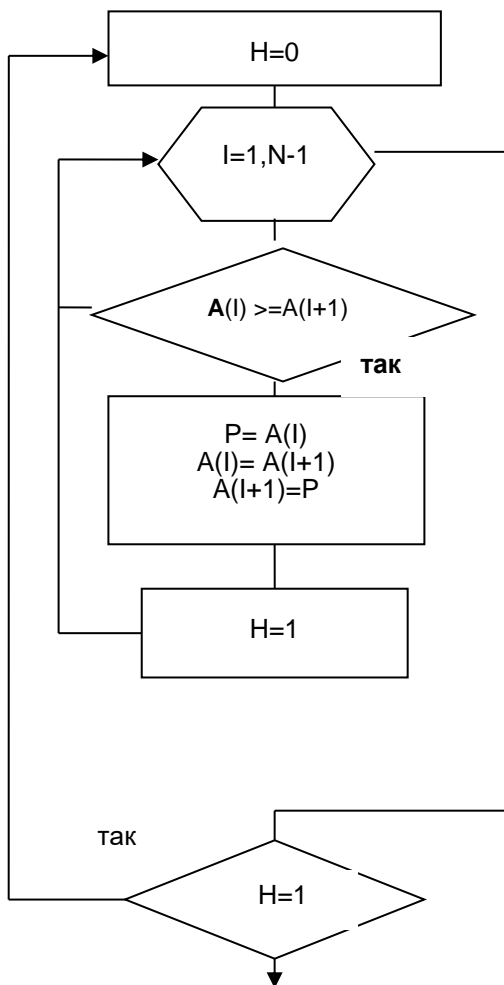
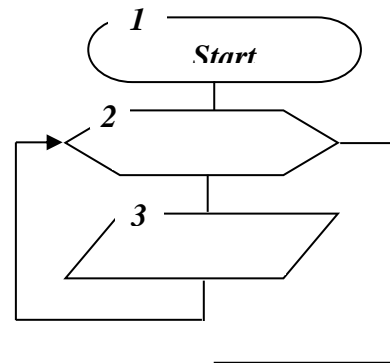
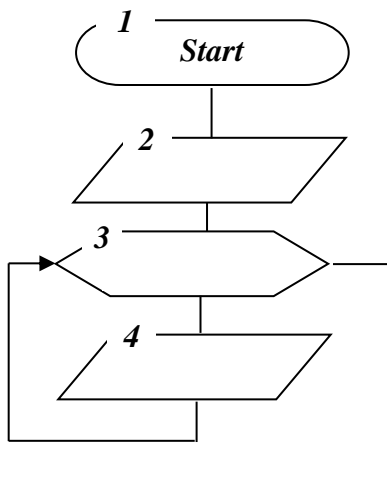


Fig. 5.18 Improved bubble method

Control questions and tasks

1. Define the concept "Array".
2. What are the characteristics of an array?
3. What in mathematics is analogous to the concept of "one-dimensional array"?
4. Expand the concept of "array size".
5. How are array elements stored in the computer's memory?
6. Expand the concept of "the dimension of the array".
7. What type of algorithm is used to enter and process array elements?
8. What is the key attribute for array elements?
9. Write an algorithm for input of elements of array A of arbitrary size A(M).
10. Write an algorithm for entering elements of an array D consisting of 17 elements.



11. How to determine the size of a two-dimensional array that has "M" rows and "N" columns?
12. What structure should the introduction of computer algorithms for two-dimensional arrays?
13. How are two-dimensional array elements stored in the computer's memory?
14. In what sequence is it necessary to organize cycles when processing two-dimensional arrays by rows?
15. Define the concept: "array ordering", "key feature" when ordering array elements.
16. Give a list of known algorithms for sorting arrays.
17. What type of algorithm is used to organize arrays?

18. Describe algorithms:

- 1) sorting "By choice";
- 2) sorting by "Bubbles" method;
- 3) sorting with the improved "Bubbles" method.

19. The index value in parentheses after the array name defines the:

- 1) size of the array;
- 2) dimension of the array;
- 3) data type of the array.

20. The number of indexes placed in parentheses after the array name determines:

- 1) the data type of the array;
- 2) the number of rows in multidimensional arrays;
- 3) the number of columns in multidimensional arrays;
- 4) the dimension of the array.

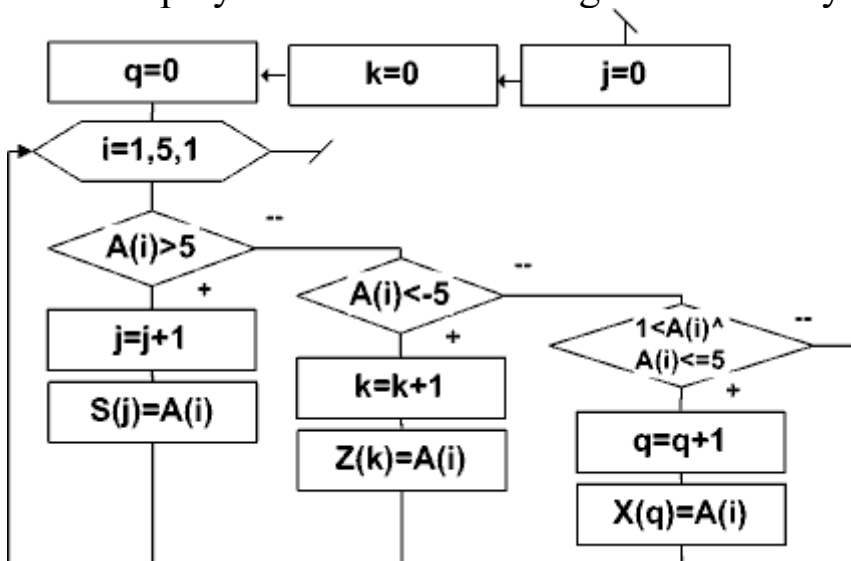
21. What are the steps of processing arrays:

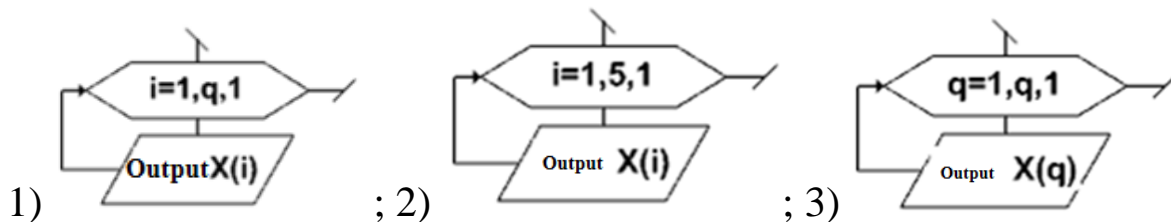
- 1) filling the array, array processing, outputting results;
- 2) description of the array, filling the array, outputting results;
- 3) input/output of array elements, filling the array, array processing, outputting results.

22. An array is an ordered set of similar elements,

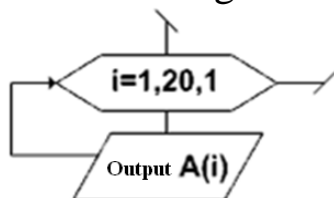
- 1) marked with one name;
- 2) distinguished by their names;
- 3) distinguished by their ordinal numbers (indexes);
- 4) intended for summation of elements.

23. Display the elements of the generated array X:





24. The diagram describes the task:

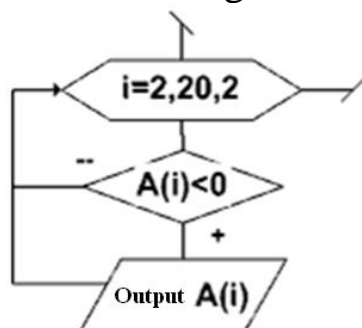


- 1) filling of one-dimensional array A of arbitrary length;
- 2) filling in one-dimensional array A consisting of 20 numbers;
- 3) filling in two-dimensional array A consisting of 20 numbers;
- 4) no correct answer.

25. *An index in arrays can be defined by:

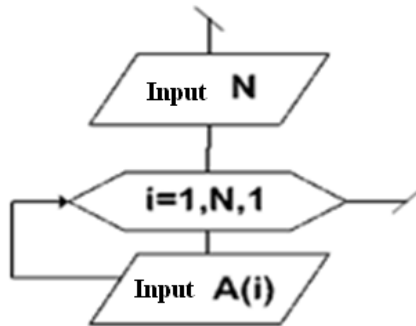
- 1) constant; 2) variable; 3) arithmetic expression;
- 4) a set of constants; 5) a set of variables.

26. The diagram describes the task of displaying:



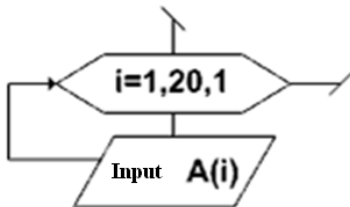
- 1) elements having even ordinal numbers;
- 2) elements having odd ordinal numbers;
- 3) positive elements having even ordinal numbers;
- 4) positive elements having odd ordinal numbers;
- 5) negative elements having even ordinal numbers;
- 6) negative elements having odd sequence numbers;
- 7) no correct answer.

27. The scheme describes the task:



- 1) filling one-dimensional array A consisting of 20 numbers;
- 2) filling two-dimensional array A consisting of 20 numbers;
- 3) filling one-dimensional array A of arbitrary length;
- 4) no correct answer.

28. The scheme describes the task:



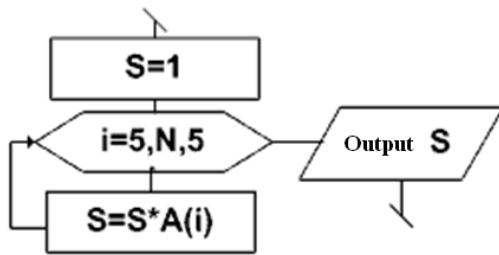
- 1) filling of one-dimensional array A of arbitrary length;
- 2) filling one-dimensional array A consisting of 20 numbers;
- 3) filling two-dimensional array A consisting of 20 numbers;
- 4) no correct answer.

29. The scheme describes the task of finding the sum:



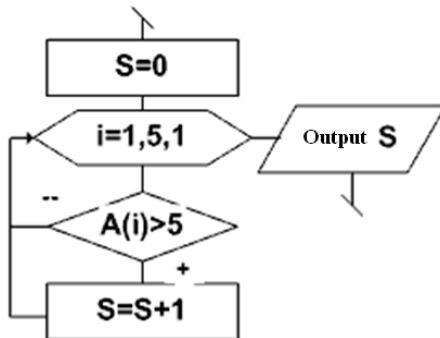
- 1) the first 5 elements of the array;
- 2) each 5th element of the array;
- 3) the last 5 elements of the array;
- 4) array elements bigger than 5;
- 5) no correct answer.

30. The scheme describes the task of finding:



- 1) product of the first 5 elements of the array;
- 2) sum of the last 5 elements of the array;
- 3) product of array elements bigger than 5;
- 4) sum of array elements bigger than 5;
- 5) product of each 5 elements of the array.

31. The scheme describes the task of finding:



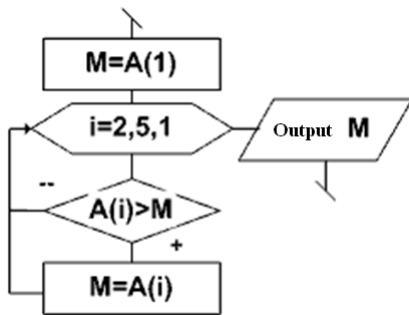
- 1) sum of array elements bigger than 5;
- 2) number of array elements bigger than 5;
- 3) product of array elements bigger than 5;
- 4) arithmetic mean of array elements bigger than 5.

32. The scheme describes the task of finding the sum of the elements of the array:



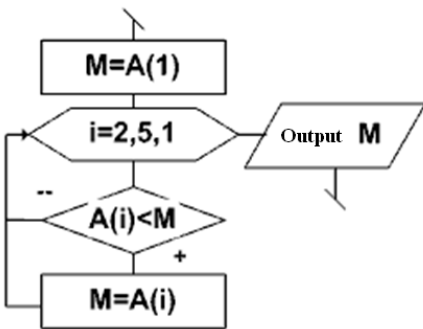
- 1) with numbers 1,4,7,10,13;
- 2) with numbers 1,3,5,7,9;
- 3) with numbers 1,5,9,13,17;
- 4) with numbers 1,6,11,16,21;
- 5) with numbers 1,2,3,4,5.

33. The scheme describes the task:



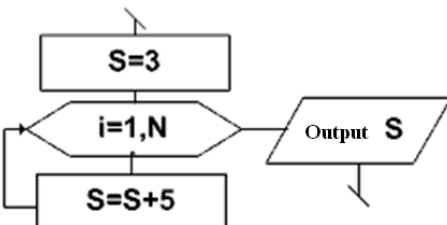
- 1) search the minimum element in array A;
- 2) formation of array of elements lying in the range $2 \leq A(i) \leq 5$;
- 3) search for the maximum element in array A;
- 4) search for elements larger than M in the entire array A.

34. The scheme describes the task:



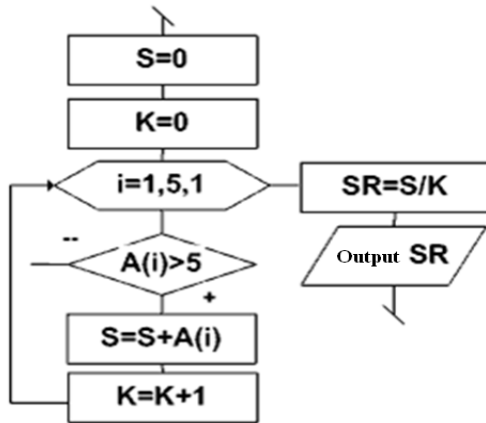
- 1) search for the first maximal element in array A;
- 2) search for the last minimum element in array A;
- 3) search for the last maximum element in array A;
- 4) search for the first minimal element in array A.

35. The scheme describes the task of finding the sum of the array elements:



- 1) with numbers 1,4,7,10,13 .
- 2) with numbers 1,3,5,7,9 .
- 3) with numbers 1,5,9,13,17 .
- 4) no correct answer.

36. The scheme describes the task of calculation and printing:



- 1) number of array elements bigger than 5;
- 2) number of array elements bigger than 5;
- 3) sum of array elements bigger than 5;
- 4) product of array elements bigger than 5;
- 5) no correct answer.

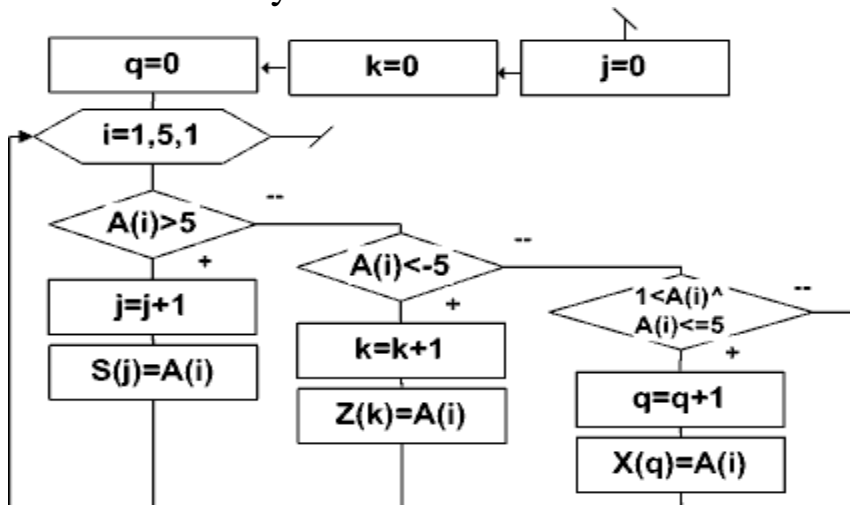
37. The size of the array A

A

2	4	6
13	10	1
-4	12	-1
9	1	0
6	9	1

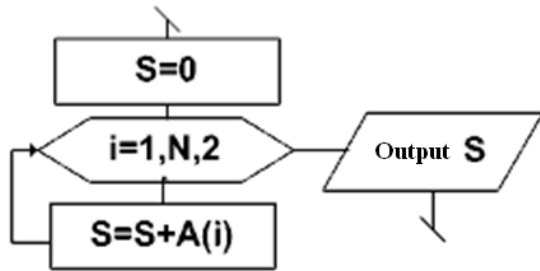
- 1) 15;
- 2) 5;
- 3) 3;
- 4) 2;
- 5) 1.

38. Which array is formed of elements smaller than 5?



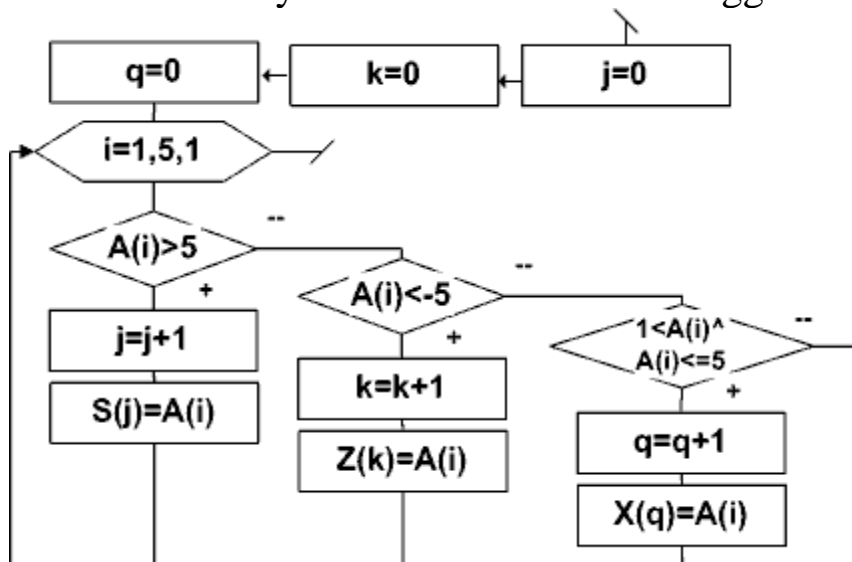
- 1) array Z;
- 2) array X;
- 3) array S;
- 4) array A;
- 5) no correct answer.

39. The scheme describes the task of finding the sum of array elements:



- 1) with numbers 1,4,7,10,13;
- 2) with numbers 1,3,5,7,9;
- 3) with numbers 1,6,11,16,21;
- 4) with numbers 1,5,9,13,17;
- 5) with numbers 1,2,3,4,5;
- 6) no correct answer.

40. Which array is formed of elements bigger than 5?



- 1) array Z;
- 2) array X;
- 3) array S;
- 4) array A;
- 5) no correct answer.

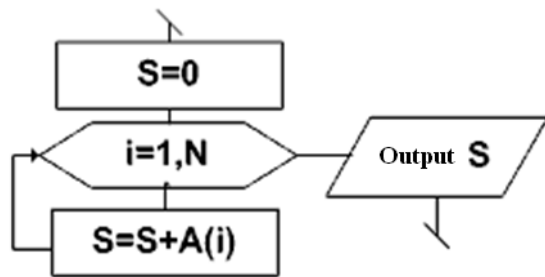
41. The dimension of the array A:

A

2	4	6
13	10	1
-4	12	-1
9	1	0
6	9	1

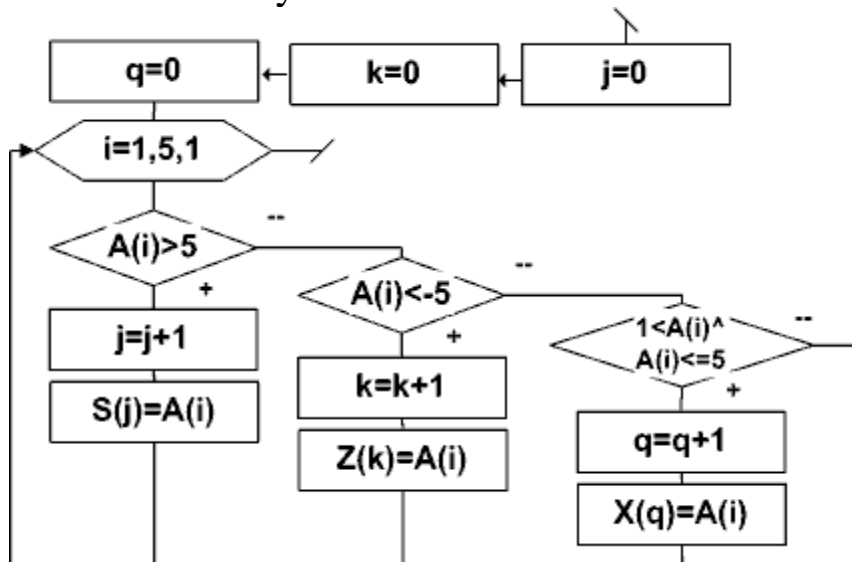
- 1) 15;
- 2) one-dimensional;
- 3) two-dimensional;
- 4) 5;
- 5) 3.

42. The scheme describes the task of finding the sum of array elements:



- 1) with numbers 1,4,7,10,13;
- 2) with numbers 1,3,5,7,9;
- 3) with numbers 1,5,9,13,17;
- 4) with numbers 1,2,3,4,5;
- 5) no correct answer.

43. Which array is formed of elements smaller than -5?



- 1) array Z; 2) array X; 3) array S; 4) array A; 5) no correct answer.

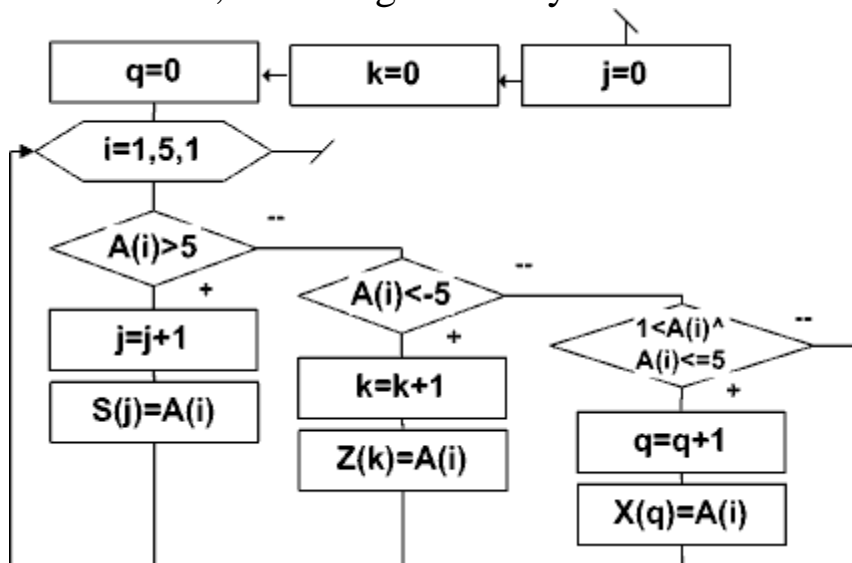
44. The size of the array A:

A

2
13
-4
9
6

- 1) 5; 2) one-dimensional; 3) two-dimensional; 4) 2; 5) 3.

45. Which array is formed of elements belonging to the range from -5 to 1, including boundary values?



1) array Z; 2) array X; 3) array S; 4) array A; 5) no correct answer.

46. The dimension of the array A

A

2
13
-4
9
6

1) 5; 2) one-dimensional; 3) two-dimensional; 4) 2; 5) 3.

47. The index values in parentheses after the array name are determined by:

- 1) the size of the array;
- 2) the dimension of the array;
- 3) the data type of the array.

48. The number of indices in parentheses after the array name determines:

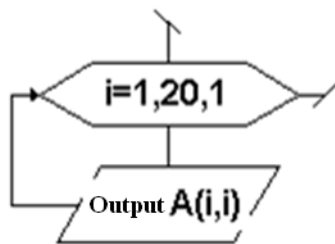
- 1) the dimension of the array;
- 2) the data type of the array;
- 3) the number of rows in multidimensional arrays;
- 4) the number of columns in multidimensional arrays.

49. *The index in the array can be determined by:
 1) constant; 2) variable; 3) arithmetic expression;
 4) a set of constants; 5) a set of variables.

50. What are the steps (sequence) of processing arrays?

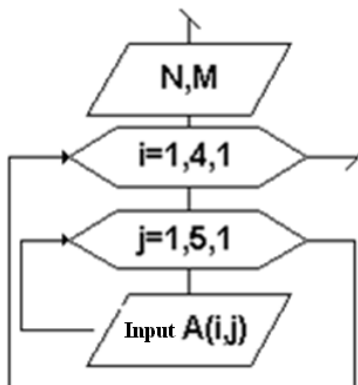
- 1) filling the array, array processing, outputting results;
- 2) input/output elements of the array, filling the array, array processing, outputting results;
- 3) description of the array, filling the array, outputting results.

51. The scheme describes the task of displaying the elements of the array:



- 1) that are in even rows;
- 2) that are in odd rows;
- 3) that are in odd columns;
- 4) that are in even columns;
- 5) that are on the main diagonal;
- 6) of odd elements;
- 7) no correct answer.

52. The scheme describes the task of filling:



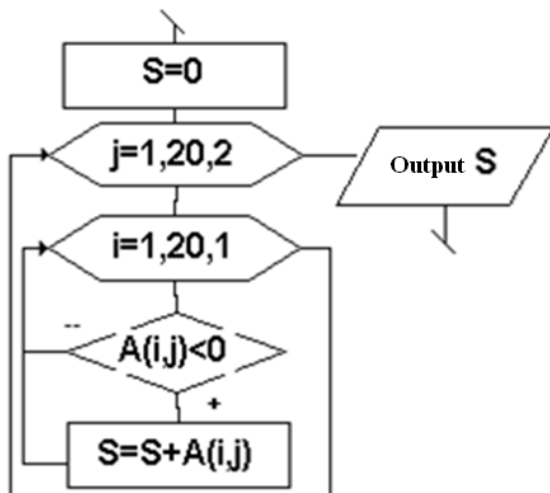
- 1) two-dimensional array A consisting of 20 numbers;
- 2) one-dimensional array A consisting of 20 numbers;
- 3) one-dimensional array A of arbitrary length;
- 4) two-dimensional array A of arbitrary length.

53. The scheme describes the task of displaying the negative elements of the array:



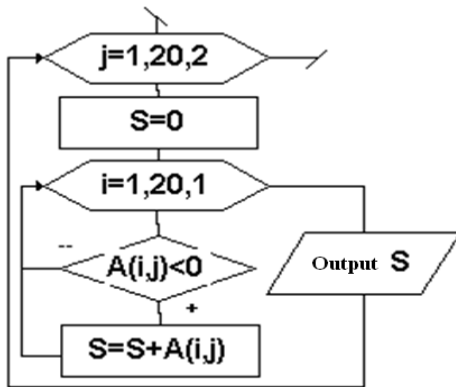
- 1) that are in even rows;
- 2) that are in odd rows;
- 3) that are in odd columns;
- 4) that are in even columns;
- 5) that are on the main diagonal;
- 6) of odd elements;
- 7) no correct answer.

54. The scheme describes the task of finding the sum:



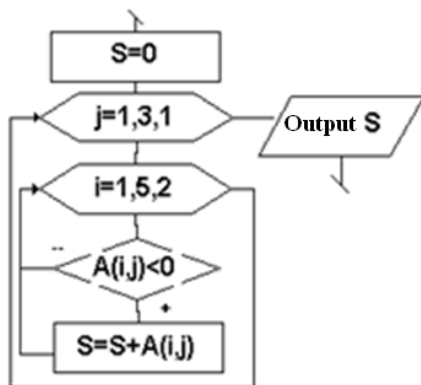
- 1) of the negative elements of an array lying in odd columns;
- 2) of positive elements of an array lying in odd columns;
- 3) of the negative elements of an array lying in even columns;
- 4) of the negative elements of an array lying in odd rows.

55. The scheme describes the task of finding the sum:



- 1) of the negative array elements in each even column;
- 2) of the negative array elements in each odd column;
- 3) of all the negative elements of an array in all even columns;
- 4) of all the negative elements of an array in all odd columns.

56. As a result of the scheme



A

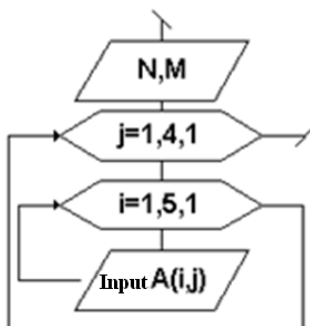
2	0	1
13	10	-5
-4	5	-1
1	-5	0
1	10	1

for the array

S=.

- 1) 5; 2) -5; 3) 15; 4) 0.

57. As a result of the scheme



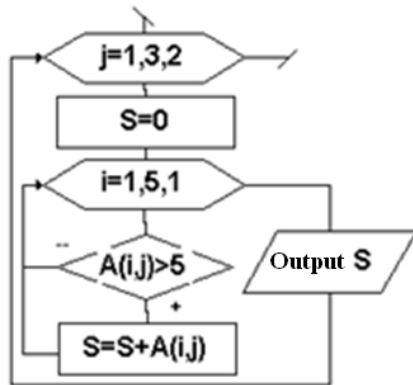
entering data into array A will be done by:

- 1)rows; 2)columns; 3)only in odd rows; 4)only in odd columns;
- 5)only on the main diagonal; 6)above the main diagonal;
- 7)below the main diagonal.

58. *Array – is an ordered set of similar elements

- 1) labeled with one name;
- 2) that are distinguished by their names;
- 3) that differ in their ordinal numbers; (indexes)
- 4) intended to sum elements;

59. As a result of the scheme S=.



A

2	0	1
13	10	-5
-4	5	-1
1	-5	0
1	10	16

for the array

- 1) 13, 16
- 2) 16
- 3) 13
- 4) 13, 20, 16

BIBLIOGRAPHICAL LIST

1. Donald Ervin Knuth. The Art of Computer Programming 1. Fundamental Algorithms (ART OF COMPUTER PROGRAMMING VOLUME 1) / Addison Wesley, 7. Juli 1997. 650 p. ISBN: 9780201896831.
2. Steven S Skiena. The Algorithm Design Manual 2nd ed., Springer, 14. November 2011, 730 p. ISBN: 978-1848000698.
3. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest. Introduction to Algorithms /The MIT Press, 20. Dezember 2013, Buchlänge 1292p. ISBN: 9780262033848.
4. Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom. Database Systems / Pearson New International Edition: The Complete Book: Pearson Education Limited, 17. Juli 2013. 1140 p. ISBN: 129202447X.
5. Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser. Data Structures and Algorithms in Java: International Student Version: Wiley, 12. August 2014. 720 p. ISBN: 1118808576.
6. Martin Fowler. Patterns of Enterprise Application Architecture / Addison Wesley, 5. November 2002. 512 p. ISBN: 0321127420.
7. Основи алгоритмізації базових обчислювальних процесів: навч. посіб. / В. С. Меркулов, В. О. Гончаров, І. Г. Бізюк та ін. Харків: ХарДАЗТ, 2007. 164 с. ISBN: 5-7763-0073-8.

TUTORIAL STUDENT'S BOOK

Bantyukov Sergiy,
Merkulov Victor,
Biziuk Iryna
etc.

COMPUTER SCIENCE

**FUNDAMENTALS OF ALGORITHMIZATION
OF BASIC COMPUTATIONAL PROCESSES**

Відповідальний за випуск Бізюк І. Г.

Редактор Еткало О. О.

Підписано до друку 09.10.19 р.

Формат паперу 60x84 1/16. Папір писальний.

Умовн.-друк. арк. 9,5. Тираж 100. Замовлення №

Видавець та виготовлювач Український державний університет
залізничного транспорту,
61050, Харків-50, майдан Фейєрбаха, 7.
Свідоцтво суб'єкта видавничої справи ДК № 6100 від 21.03.2018 р.