**Topic № 1.**

Computer science appeared in the early 40th of the XX century. It was due to the development of computers. Computer science was a combination of the capabilities of computers of that time, mathematical logic and the theory of algorithms.

Later, new directions appeared. It was due to improvements of the computers. They began to be used in a wider range of areas of human activity.

In a number of countries instead of "computer science" the concept of "informatics" is used. The notion of "informatics" arose in the 60s in France. This was the name of the field of knowledge that studies the use of electronic computers for automation of data processing. The word "informatics" is formed from two words "information" and "automation".

Although the concepts of "informatics" and "computer science" can be considered synonymous, there is a slight difference between them. Informatics places emphasis on information processing. However, its emergence and development are inextricably linked with the existence of computer technique. Informatics as a scientific discipline began to form in the 70-80s of the XX century, when computers became more accessible to a diverse audience.

Initially, the computer was a tool for automating labor-intensive calculations. However, it gradually evolved into a tool for working with almost any information, not just a numeric one. To date, there are a huge number of programs and applications designed to work with text, graphics, tables, databases and much more.

From the computer science and informatics began to stand out some scientific directions. Since the material world is diverse, and there are many spheres of human activity, the subject of computer science is also very heterogeneous. Therefore, computer science can be regarded as a complex science, which makes it difficult to unambiguously define its.

In the 1980s, E.P. Ershov gave this definition:

Informatics is a science that is in the process of becoming established, studying laws and methods of accumulation, transmission and processing of information with the help of a computer, as well as the field of human activity connected with the use of computers.

Computer science is closely connected with mathematics, as it relies on its achievements. This is due to the fact that objects of natural and technical sciences, as well as social phenomena can be described with the help of the concepts of mathematics - functions, systems of equations, inequalities, and others.

On the other hand, the subject of informatics - information - is, among other things, a general scientific and social concept.

In fact, the task of informatics is to study the ways in which scientific and technical achievements can be used to process data of diverse nature. Civilization in the XX century came to the point of accumulation of data, when the problem arose of their storage, use, access, transfer, transformation. Computer science is studying the solution of these problems with the help of computer technology.

Currently, processes are actively occurring related to the translation of information accumulated by civilization into the electronic form. It can be expected that many real-world objects will soon find their digital counterpart.

**Cybernetics and Informatics**

Cybernetics is the science of the general principles of management in various systems: technical, biological, social and other. Management is the purposeful organization of a particular process that takes place in the system. Management is the central concept of cybernetics.

Each phase of the control process flows in interaction with the environment. Therefore, in cybernetics, much attention is paid to the study of feedbacks and the concept of a "black box".

The foundations of cybernetics as a science were laid by works on the mathematical logic, that were published in 1948, of the American mathematician Norbert Wiener. Although the term itself was introduced a century earlier by the Frenchman AM Amper.

Are cybernetics and informatics related to each other and how? Yes, they are connected. Basically through the concept of "information". Both sciences operate on information, but do so for different purposes. Therefore, cybernetics and informatics are different sciences having a number of points of contact. In other words, their fields of activity overlap.

Cybernetics is the science about management. The governing body passes information flows through itself. Another thing - the objects of control, through which mainly material flows pass. Information for cybernetics plays the role of a kind of means to which control is provided. All people involved in management are dealing only with information.

The management process is an information process that includes the collection of information, its processing and analysis, decision-making, the development of control actions and their delivery to management objects.

Informatics is the science of how to automate the processing of information. How to receive information, store, transmit, convert. Here, information is a central concept, an object of study. Informatics is engaged in studying the processes of transformation and creating new information more widely. For cybernetics, the central concept is management.

Informatics appeared thanks to the development of computer technology, is based on it and without it is unthinkable. Computer technologies play the role of information processing tools for informatics.

Cybernetics developed by itself, building various models of object management. The basic concept, laid down by N. Wiener, is connected with the development of the theory of control of complex dynamical systems in various fields of human activity. Cybernetics exists regardless of the presence or absence of computers.

**Areas of Computer Science**

Computer science is a complex science. It includes various scientific directions, the development of technology, the study of technics. The only thing that unites all this is the use of computer techniques.

We can distinguish two main areas of computer science - theoretical and applied (practical). The second uses the achievements of the first.

In the field of theoretical computer science, the concept of information, its coding, mathematical logic, theory of algorithms, data structures, the theory of programming languages and some other areas of research are studied.

In general, theoretical computer science develops methods of data management with the help of computers, explores information systems, their role, structure, functioning, patterns.

Unlike theoretical, applied computer science is engaged in the development of specific information systems for various fields of activity. In the area of this computer science, the main directions are:

* hardware, or computer architecture
* computer networks
* cryptography and computer security
* development of programs - software, programming languages
* artificial intelligence - questions of recognition of images and sounds, intellectual translation, research of possibilities of self-development of computer systems, etc.
* databases
* computer modelling
* computer graphics

Informatics is not only a science, but also a sphere of people's activity, an applied discipline that has wide practical application. The development of computer science, the introduction of its achievements allows increasing labor productivity.

The increase in the influence of informatics on all spheres of human life is connected with the transition to an information society, which increasingly consumes and produces information. Therefore, in the areas of computer science, new directions should be expected.

Since computer science develops methods and means for processing various information, its tasks are:

* the study of all known information processes;
* creation of techniques and technologies for information processing;
* effective inculcation of computers in the field of human activities.

So, computer science is a complex scientific and technical discipline that creates new equipment and technologies for solving information and management problems in other areas. Progress in all spheres of society depends on computer science.

**History of Informatics**

The history of informatics as a science began in the second half of the 20th century. This was due to the appearance and dissemination of computers and the beginning of the computer revolution. The advent of computers in the 1940s and 1950s created the necessary hardware support for computer science and informatics, that is, a favorable environment for their development as a sciences.

However, despite its short history, informatics has a long prehistory associated with the peculiarities of accumulation and processing of information at different stages of the development of human society. Thus, the whole history of informatics can be divided into two major stages: history and prehistory.

**Prehistory of Informatics and Computer Science**

The prehistory of informatics began with the emergence of society. In the prehistory there are a number of stages. Each of them is characterized by a sharp increase in comparison with the previous stage of the possibilities of storage, transmission and processing of information.

The first stage is the appearance of a developed oral **speech**. For the ancient people, articulate speech and the language in which they spoke began to play the role of a means of storing and transmitting information.

At the second stage, **writing** appeared. In comparison with the previous stage, the possibility of storing information has sharply increased. Mankind received a kind of artificial external memory. The organization of postal services made it possible to use writing as a means of conveying information, not just storage.

The emergence of writing was a prerequisite for the development of science. Probably at the same stage, the notion of a "natural number" appeared. All nations that had a written language owned the concept of number and used a certain numeral system.

The third stage is **typography**. It can be called the first information technology. The reproduction of information turned out to be streamed. Compared to the previous one at this stage, not so much the possibility of storing information (although here there was a gain: a written source is often a single copy, a printed book is a whole circulation of copies, and therefore a low probability of information loss during storage) how much improved the availability of information for all people, as well as the accuracy of its reproduction, that is, reliability.

The fourth and last stage of the prehistory of informatics is connected with the successes of the exact sciences (primarily mathematics and physics) and the beginning of the scientific and **technological revolution**. This stage is characterized by the emergence of such powerful means of communication as radio, telephone and telegraph, and later television. There were new opportunities for receiving and storing information - photography and cinema. It is very important to add to them the development of methods for recording information on magnetic media (magnetic tapes, disks).

**History of Informatics**

The beginning of the history of informatics as a science is associated with the development of the first computers. There are several reasons for this binding.

Firstly, the term "informatics" came thanks to the development of computer technology, and at first informatics and computer science were understood as the science of computing automation, because the first computers were mostly used for numerical calculations.

Secondly, an important feature of modern computer technologies is the universal final way of presenting information. This also contributed to the allocation of informatics in a separate science. All information, regardless of its form, is stored and processed on a computer in binary form.

It so happened that the computer in one system combined storage and processing of numerical, text (symbolic) and audiovisual (sound, image) information. In this universality was the initiating role of computer technology in the emergence and design of the new science.

Today, computer science and informatics are complex scientific and technical disciplines. They unite a number of fields, such as information theory, cybernetics, programming, modeling, hardware and much more.

**Receiving the Information by a Human**

A man is an animal organism. Therefore, people can receive information through the senses: eyes, ears, nose, tongue, skin and muscles. Accordingly, we distinguish the types of information: visual, auditory, olfactory, gustatory, tactile, muscular sense.

Various kinds of information are not equal among themselves to humans. Most of all we perceive visual information (about 90%), about 5% of the sound, the other types make up a small percentage.

In addition there is information in the world that a person can not obtain via its existing senses. For example, we do not hear ultrasound, do not feel the magnetic field of the Earth, our vision is limited in range. However, information that is not available to us, we can get by using different technical means. Technics converts the data into a form that people can perceive. Often at this person must be able to correctly interpret the data. For example, we can not accurately assess the level of light in the room. The device can accurately show the level. However, to understand and to conclude about adequate lighting, a person must have special knowledge.

The Concept Of An Information Resource

An information resource - it is information that is knowledge, i.e. it has all the attributes of knowledge.

The form of presentation of knowledge is an important for an information resource. Knowledge - it is a reflection of one side or another of the objective reality in the form of ideas, concepts about some object or phenomenon.

But not all knowledge can serve as an information resource. For information resource it is important to its use by mankind. While knowledge and information may remain unused due to a number of reasons. Therefore, today there is the problem of extracting the maximum information from the messages accumulated in the history of humanity and turning it into an information resource. Typically, the content of the books or other knowledge are converted into algorithms and programs. It is part of the work on the formation of an information resource.

An information resource is an intellectual resource, the factor of collective creativity. There is a problem of understanding how knowledge is transformed into a force that acts on production and society.

**Features of an information resource**

1. Inexhaustible. Unlike material resources. With the development of society and the growth of consumption of knowledge, stocks of information resources are not decreasing, and growing.
2. It has the potential significance. An information resource may appear as a driving force only connecting with other resources - experience and qualifications of personnel, equipment, energy and raw materials.
3. Efficiency of application. This is due to the effect of re-manufactured knowledge: obtaining knowledge requires a large amount of effort, while its reproduction is performed with less labor.
4. It includes science into productive forces. Information resources are becoming the basis of production.
5. There is a result of creativity. Any mental work consists of two parts: the routine and creative. The increasing mental work by routine does not lead to an increase in an information resource.

**Forms of an information resource**

An information resource can exist in two forms: active and passive. Passive resources are books, databases, etc.

Active resources are models, algorithms, programs, projects, knowledge bases.

They can be interpreted as a stage of maturation and transformation of information resource into force.

**Model** is a description of the system, showing some its group of properties.

Creating a model of system allows to predict its behavior in a certain range of conditions.

**Algorithms** are classified according to the degree of generality. It is important to seek to establish basic algorithms.

**Programs and projects** is the final synthetic form of existence of an information resource in its life cycle. The program or the project resist entropy of the object. From this standpoint, it is introduced the concept of the program information capacity, which indicates the magnitude of the potential to reduce uncertainty. In the course of implementing the program, object is filled with information.

A model, algorithm, program, project, and especially the knowledge base as an active form of the information resource are anti-entropic tools. However, the program and the project stand out among them its regularity and willingness to direct effects of the information on the object in order to remove his uncertainty.

**Properties of Social Information**

For a man it is not so important quantitative characteristic information. Important properties of information related to the knowledge of the world. For the person information may be important or not, be full or not, etc. In other words, it is important to the quality of social information.

There are various properties information.

1. The value of information. The more important problem to be solved by a person, the more valuable the information that is required to solve the problem.
2. Availability of information. For example, if the text is in the World Wide Web, and you have a connection to the Internet, the information is easier to obtain than if the text was only in the library.
3. Understandable information. A message in Japanese can be incomprehensible to an englishman, even if it contains valuable information.
4. Completeness of information. Enough information to solve specific range of tasks.
5. Redundancy of information.
6. Adequacy. The conformity of the reality.
7. Relevance. The information may matter only in a certain time.
8. Objectivity. The lower the content of the information depends on who it received and processed, so it is more objective.
9. and etc.

A feature of the properties of social information is their temporary nature and dependence on a specific person. Since the same information to someone can be understood, and for someone — no. Today, the information may be relevant, and tomorrow — no.

# Information Processes

In the world there are information flows. Information is transferred from one object to another, wherein information can be modified.

Information is mainly considering the information processes in any way connected with the person. People receive, process, store and transmit, use information in their activities.

Receiving, transmission and storage of information takes place through messages on a particular language (not necessarily natural). Information processing can lead to the emergence of new information or change it.

Man receives information from the senses, and through technical means (such as a telescope). Thus, humanity reflects the world around us in an understandable form for yourself. Man can transmit the information to others with the help of technical means, with their help, he can store it. Using the equipment for storage and transfer of information requires transforming information into another form, because technology does not understand the words and the information is necessary to encode. When submitting information to the person it is decoded into an acceptable form for him.

It should be noted that the new information can be obtained from the old through its processing, generalization and other operations that are characteristic of consciousness (creativity, invention, conclusions, calculations, etc.).

# Information Technologies

When there was no information technologies, information was processed without the aid of computers. With the advent of computers, information systems appeared. They implement a special technologies of gathering, processing, transmission and use of information.

The concept of technology includes a set of scientific and engineering knowledge, embodied in methods of work, a set of material, technical, energy, labor factors of production, ways of their connection to create a product or service.

Information technology include:

* certain sets of material resources (carriers of information, tools to assess its condition, processing, transmission, etc.);
* ways of interaction of material means;
* a set of specific methods of organizing the work of specialists.

Information technologies include procedures that can be grouped by function-time stages:

* data collection and recording;
* transfer of data;
* data encryption;
* storage;
* search;
* calculation processing;
* duplication of information;
* using information, i.e. making decisions.

Typically, the information is subjected to conversion with all procedures, but in some cases, some procedures may be omitted. The sequence of the procedures is also different, and some of them may be repeated.

The procedure for the collection and recording of information is carried out differently on different objects. First, information is collected and then fixed.

Writing in the primary documents is mainly done by hand, so the procedure for collection and registration remains the most labor-intensive. In the context of the automation of control, special attention is given to the use of technical means for collecting and recording information, its storage and transmission through communications channels into a computer in order to create the original document.

Information transfer may be performed remotely via communication channels, which reduces the time of data transmission. For remote transmission of information through communication channels need special hardware. Some of the technical means of collect and record information automatically collect it from sensors installed in the workplace, and transmit it to a computer.

Encoding of information is a procedure for submission of information on storage media in the codes adopted in the computer. In the process of recording information in the media there is the greatest number of errors due to complexity of the operation.

Usually on the computer, information processing is carried out centrally.

In processing the information, arithmetic and logic operations are performed on a computer. Arithmetic operations of processing data include all types of mathematical operations caused by the program. Logic operations provide the appropriate ordering data arrays to be further arithmetic processing. Significant place in the logical operation takes different types of sorting: ordering, distribution, selection, sampling, merging.

Storage and accumulation of information due to the need of its repeated use. Storage is on machine carriers in the form of data arrays.

Search data is a sample of the required data from the stored information, including information retrieval, subject to adjustment or replacement.

Printing reports may be accompanied by the replication procedure.

As a rule, the decision is carried out without the use of specialist technical equipment, but on the basis of careful analysis of information gathered from a computer. Decision-making procedure is complicated by the fact that the expert has to look from the set of feasible solutions most suitable, which minimizes waste of resources (time, labor, material, etc.). Through the use of computers, the degree of analytic processed data increases, and also it provides a gradual transition to automation development of optimal solutions in the process of dialogue of the user with computer system. The mathematical theory of decision-making is based on game theory, operations research.

# Methods for Reproducing and Processing of Information

**Natural methods.** Methods based on the senses. Logical thinking, imagination, comparison, analysis, forecasting, etc.

**Hardware methods.** It's always devices. Tape recorders, telephones, microscopes, video recorders, etc. From the standpoint of computer science, these devices perform one function. They convert the data from the form is not available for natural methods of man, in a form accessible to him. Not always devices can process data created by other devices. In such cases, it is used special devices of data conversion, and not talking about the transformation of the form data, talking about the transformation of their format (modems, home video camera).

**Software methods.** The widespread introduction of computer technology makes it possible to automate the processing of many different kinds of data with a computer. Computer is a particular type of device, which combines both hardware and software methods for processing and presentation of information. These methods constitute the subject area computer science.

Assuming that information is a dynamic object that does not exist in nature itself and formed during the interaction of data and methods, and exists as long as it lasts interaction, and all the rest time exists in the form of data, it is possible to give a definition:

Information - it is the interaction product of the data and methods considered in the context of this interaction.

Context (adequate) method is one that is common to work with a particular type of data. This method must be known by the creator of data and the user of information.

For graphical data (illustrations), a contextual approach is an observation based on the vision. In this case, it refers to the visual or graphic information. For text data it is used contextual reading method, based on vision, knowledge of the alphabet and language. In this case we speak of text information.

For radio waves it is used hardware methods of data transformation via radio or television. Therefore, often used the concept of television information, newscast, news program, etc.

For data stored in the computer, transmitted over networks, it is used the hardware and software methods of computer technics. These methods are called means of information technology, they are included in the subject area of computer science. In this case it is used the notion of computer information.

Form of Information in a Computer Memory

To information can be operated, process and convert it is necessary to present it in the form of any structure. Often, information is provided in the form of a one-dimensional sequences of characters. Under the multi-dimensional representation of information are refering the location of its elements in the two-dimensional plane or in three-dimensional space in the form of drawings, diagrams, volume layouts. Thus there are many characteristics of the information (color, size, position, composition, structure).

Creation of a presentation of information is called its coding. Coding can be seen as translating information from its natural form of existence in the form convenient for storage, transmission and processing. Returning of an information in the original form is called decoding.

Usually, when coding information:

* It becomes easy for the physical play.
* It can be a high speed transfer and process.
* Redundancy of messages decreases.
* There are reliability and safety, as protection against accidental distortion and ease of storage and protection against intentional damage.

However, all of these advantages are relative and are often mutually exclusive. For example, reduction of redundancy can reduce reliability.

Since at the present time to work with the information used computers, it is natural that information theory is very closely related to computer science.

In a computer for storing information, there are a variety of storage devices. The information is stored in the form of binary digital code. This is due to the fact that it is technically easier to implement such a memory. Electronic components, that make up the memory, can only be in one of two stable states. These conditions are compared with 0 and 1.

It is understood that in one memory element can store a minimum amount of information. The minimum amount of information is called a bit. The word "bit" is a contraction of two words: BInary + digiT = BIT.

Consistent association of bits is forming larger units of information — cell. The cell, that consists of eight bits, called a byte.

The hardware of the computer operates the machine words, which represent a certain number of consecutive bits, perceived as a whole. Word length depends on the particular computer.

Memory system of the computer imposes limitation on the capacity and the precision of the numerical information (because it is obvious that computer memory physically consists of a limited number of bits).

**The Amount of Information in Computer Technics**

Information theory is one of the branches of mathematics. Therefore, one of the problems associated with the information is the measurement of its quantity. However, the concept of information can be defined differently. Therefore, it can different measure its quantity. To determine the amount of information the most common formula Hartley and formula Shannon.

In the world of computers, the amount of information is determined with the bits, bytes, kilobytes, megabytes, gigabytes, etc.

**Bit** is the smallest unit of information in computer technics. This is cell, which can be in two states: either 0 or 1.

One bit can only be in two states (0 or 1).

Two bits have four states (00 or 01 or 10 or 11).

Three bits — 8 states (000, 001, 010, 011, 100, 101, 110, 111).

Four bits — 16 states.

It is easy to notice that the number of states (N) of the system is determined by the formula N = 2a, where a - is the number of bits. So 28 = 256. I.e. using 8 bits, we can fixate 256 different states.

In computer technics, 8 bits are called **byte**. Why are eight bits in a byte? Partly due to the fact that 256 states is enough to encode the alphabet of any language in the world. I.e. for each letter assigned one byte, that can be in a suitable condition to the letter.

1 kilobyte = 1024 bytes (or 210), 1 megabyte = 1024 kilobytes, etc.

# Measuring the Amount of Information. Formula Hartley

Suppose we need to find something or to determine in a particular system. There is a way to search as "halving". For example, someone thinks of a number between 1 and 100, and the other person has to guess it, getting answers only a "yes" or "no". The question is asked: number less? The answer "yes" or "no" reduce the search area. If you select a number in the middle, the search range will be divided in half. Further, in the same way again, the range is halved. Ultimately, the unknown number is found.

We calculate how many questions we must ask in order to find a number. Suppose unknown number 27. Start:

1. More than 50? No.
2. More than 25? Yes.
3. More than 38? No.
4. Less than 32? Yes.
5. Less than 29? Yes.
6. More than 27? No.
7. This number is 26? No.

Yes! If the number is not 26, and not more than 27, then this is clearly 27.

To guess a number between 1 and 100 by "halving" method It took seven questions.

But why is it necessary to ask questions like that? For example, you can simply ask: is the number 1? Is the number 2? Etc. But then you need a lot more questions. "Bisection" is the shortest way to find a number.

The amount of information inherent in the "yes" or "no" is one bit. Indeed, the bit can be set to 1 or 0. So, to guess the numbers from 1 to 100, it took seven bits (seven replies).

**N = 2k**

This formula shows how many questions (data bits) are required to specify one of the possible values. N is quantity of values, n is quantity of bits. For example, 100 is less than 27, but more than 26. Yes, we could take a total of 6 questions, if had been make come true 28.

Hartley formula: **k = log2N**. The amount of information (k), needed to determine the particular element is the logarithm base 2 of the total number of elements (N).

# The Amount of Information. Shannon's Equation

Formula Hartley determines the number of necessary information to identify a particular element of the set under the condition that all elements are equiprobable. However, it may be that some elements are more likely (more common) and some less. For example, in the English letter "a" is used more often than the letter "z".

In the case when the probability items are not identical, to determine the amount of information sufficient to identify items using Shannon's formula. It can be obtained from the formula Hartley.

k = logN (formula Hartley)

1/N - the probability of each outcome if they all are equally probable.

-1/N \* log1/N - contribution to k (overall uncertainty, the amount of information) the same outcome.

Let the contribution of each is denoted with letter P with the index 1, 2, 3, 4 .... N.

We get the formula:

k = -P1logP1 – P2logP2 – P3logP3 – … – PNlogPN

The larger k, the more information contained in the system. The maximum value of k is only when all outcomes are equally likely.

**Topic № 2.**

# Numeral Systems. Positional and Nonpositional Numeral Systems

Numeral system is a way of writing numbers. Typically numbers are written with special characters - it is digits (but not always). Two numeral systems are well known. These are Arabic and Roman. The digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 are used in the Arab system; letters I, V, X, L, C, D, M are used in the Roman. Arabic numeral system is a positional, Roman is nonpositional.

In positional notation, the value of the figures depends on its position in the number and in nonpositional - independent. For example:

11 - the first 1 is ten, and the second 1 is one.

II - are both "I" are one.

345, 259, 521 - here numeral 5 denotes five in the first case, in the second - fifty, and the third - five hundred.

XXV, XVII, VII - here, no matter where the figure stood V, it represents always five units. In other words, the value indicating the sign of V, regardless of its position.

Addition, multiplication, and other mathematical operations in a positional number system performed better than nonpositional, since mathematical operations are carried out by simple algorithms (for example, multiplication in a column, a comparison of two numbers).

Positional number systems are the most common. The decimal system is well known. However, binary, octal and hexadecimal systems are common in the technics.

It should be noted the important role of zero. The discovery of this figure in the history of mankind has played a major role in the formation of positional numeral systems.

**Radix** is the number of characters that is used to record numbers.

The **discharge** is a position of digit in the number. **Bit number** - the number of digits that make up the number (eg, 264 - three-digit number, 00010101 - eight-bit number). Bits are numbered from right to left (for example, in number 598, the eight occupies the first discharge, and five — third).

So, **in a positional numeral system, numbers are written in such a way that each successive (movement from right to left) larger than the other for the degree radix**.

One number (value) can be represented in different number systems. Thus, different kind are obtained, but the value remains unchanged.

**Numeral Systems. Positional and Nonpositional Numeral Systems**

Numeral system is a way of writing numbers. Typically numbers are written with special characters - it is digits (but not always). Two numeral systems are well known. These are Arabic and Roman. The digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 are used in the Arab system; letters I, V, X, L, C, D, M are used in the Roman. Arabic numeral system is a positional, Roman is nonpositional.

In positional notation, the value of the figures depends on its position in the number and in nonpositional - independent. For example:

11 - the first 1 is ten, and the second 1 is one.

II - are both "I" are one.

345, 259, 521 - here numeral 5 denotes five in the first case, in the second - fifty, and the third - five hundred.

XXV, XVII, VII - here, no matter where the figure stood V, it represents always five units. In other words, the value indicating the sign of V, regardless of its position.

Addition, multiplication, and other mathematical operations in a positional number system performed better than nonpositional, since mathematical operations are carried out by simple algorithms (for example, multiplication in a column, a comparison of two numbers).

Positional number systems are the most common. The decimal system is well known. However, binary, octal and hexadecimal systems are common in the technics.

It should be noted the important role of zero. The discovery of this figure in the history of mankind has played a major role in the formation of positional numeral systems.

**Radix** is the number of characters that is used to record numbers.

The **discharge** is a position of digit in the number. **Bit number** - the number of digits that make up the number (eg, 264 - three-digit number, 00010101 - eight-bit number). Bits are numbered from right to left (for example, in number 598, the eight occupies the first discharge, and five — third).

So, **in a positional numeral system, numbers are written in such a way that each successive (movement from right to left) larger than the other for the degree radix**.

One number (value) can be represented in different number systems. Thus, different kind are obtained, but the value remains unchanged.

# Binary Numeral System

The binary numeral system uses only two digits. These are 0 and 1. In other words, a deuce is the radix of the binary system. Similarly, a dozen is the radix of the decimal system.

Consider how the numbers are generated in the decimal system to understand the numbers in binary notation.

We have ten characters (0 to 9) in the decimal system. When the count reaches 9, the new discharge is introduced (tens), and the units are reset, and counting starts again. After 19, discharge of tens is incremented by 1; and the units are reset again. And so on. When tens reach up to 9, then there is a third discharge. It's hundreds.

The principles of the formation of numbers are similar in binary and decimal systems. Except that the binary number is formed from only two digits (0 and 1). Once the discharge reaches its limit (i.e. 1), a new discharge appears, and the old is reset.

This is the beginning of counting in binary:  
0 - zero  
1 - one (limit of the discharge)  
10 - two  
11 - three (again limit)  
100 - four  
101 - five  
110 - six  
111 - seven (limit), etc.

**Translation of numbers from binary to decimal**

In the binary system, the length of the numbers are growing rapidly with increasing values. How to define what is it mean the number 10001001? We do not understand how much it is. It would be nice to be able to convert binary numbers to decimal.

In the decimal numeral system, any number can be represented as the sum of units, tens, hundreds, etc. For example:

1476 = 1000 + 400 + 70 + 6

We can represent the number as follows:

1476 = 1 × 103 + 4 × 102 + 7 × 101 + 6 × 100

Look at the record carefully. Here, digits 1, 4, 7 and 6 is a set of digits that make up the number 1476. All of these digits are multiplied by ten raised to the appropriate degree. Ten is the radix of the decimal numeral system. The degree is the discharge of the digit, from which 1 is subtracted.

Similarly, we can decompose any binary number. Only the base there will be 2:

10001001 = 1×27 + 0\*26 + 0×25 + 0×24 + 1×23 + 0×22 + 0×21 + 1×20

If we calculate the sum, we get a decimal number that corresponds to 10001001:

1×27 + 0×26 + 0×25 + 0×24 + 1×23 + 0×22 + 0×21 + 1\*20 = 128 + 0 + 0 + 0 + 8 + 0 + 0 + 1 = 137

I.e. the number 10001001 in radix 2 is the number 137 in radix 10. It can be written as:

100010012 = 13710

**Why is binary numeral system so prevalent?**

The fact that the binary system - it is the language of computer technics. Each digit must be somehow recorded on the physical medium. If this is the decimal system, the need to create a device which may be in ten states. It's complicated. Easier to manufacture the physical element which can only be in two states (e.g. there is current or no current). This is one of the main reasons why the binary numeral system is given so much attention.

**Translation decimal to binary**

You may need to convert decimal to binary. One way is divide by two and collect the remains. For example, to get from the number 77 its binary representation:

77 / 2 = 38 (1 residue)  
38 / 2 = 19 (0 residue)  
19 / 2 = 9 (1 residue)  
9 / 2 = 4 (1 residue)  
4 / 2 = 2 (0 residue)  
2 / 2 = 1 (0 residue)  
1 / 2 = 0 (1 residue)

We are putting together the remains from the end of: 1001101. It is the number 77 in the binary representation. Check out:

1001101 = 1×26 + 0×25 + 0×24 + 1×23 + 1×22 + 0×21 + 1×20 = 64 + 0 + 0 + 8 + 4 + 0 + 1 = 77

# Octal Numeral System

Modern computer technics "understands" only binary numeral system. However, the person finds it difficult to perceive the long record of binary numbers. In addition, the translation of numbers from binary to decimal and back is long and laborious. Therefore, programmers often use other numeral systems: octal and hexadecimal. 8 and 16 are powers of two. Therefore binary is easily convert into them (also perform the reverse operation).

The octal number system uses eight digits (0 to 7). Each digit corresponds to a set of three digits in the binary system:

000 – 0  
001 – 1  
010 – 2  
011 – 3  
100 – 4  
101 – 5  
110 – 6  
111 – 7

Every three digits of binary number must be replaced by their corresponding digits of the octal numeral system. Triplets are formed with the end, and missing digits at the beginning is replaced by zeros. For example:

1011101 = 1 011 101 = 001 011 101 = 1 3 5 = 135

I.e. the number 1011101 in the binary system is the number 135 in octal.

10111012 = 1358

Back translation. Suppose, you want to translate the number 1008 (do not mistake! 100 octal is not 100 decimal) in the binary system.

1008 = 1 0 0 = 001 000 000 = 001000000 = 10000002

Translation of octal number to a decimal can be done on a familiar pattern:

6728 = 6 \* 82 + 7 \* 81 + 2 \* 80 = 6 \* 64 + 56 + 2 = 384 + 56 + 2 = 44210  
1008 = 1 \* 82 + 0 \* 81 + 0 \* 80 = 6410

# Hexadecimal Numeral System

As octal, hexadecimal system is widely used in computer science for ease translating it into binary numbers. The hexadecimal numbers are more compact.

Hexadecimal notation uses numbers from 0 to 9 and the first six Latin letters - A (10), B (11), C (12), D (13), E (14), F (15).

The binary number is divided into groups of four bits, starting from the end. If the number of digits is not divisible, then the first group is appended by zeros ahead. Each group corresponds to digit of hexadecimal numeral system:

0000 — 0  
0001 — 1  
0010 — 2  
0011 — 3  
0100 — 4  
0101 — 5  
0110 — 6  
0111 — 7  
1000 — 8  
1001 — 9  
1010 — A  
1011 — B  
1100 — C  
1101 — D  
1110 — E  
1111 — F

For example:

10001100101 = 0100 1100 0101 = 4 C 5 = 4C5

If necessary, the 4C5 can be converted into a decimal number system as follows (C should be replaced by a corresponding number in the decimal system; it is 12):

4C5 = 4 \* 162 + 12 \* 161 + 5 \* 160 = 4 \* 256 + 192 + 5 = 1221

Maximum two-digit number, which can be obtained using hexadecimal notation, it is FF.

FF = 15 \* 161 + 15 \* 160 = 240 + 15 = 255

255 is the maximum value of one byte. It is equal to 8 bits: 1111 1111 = FF. So it is very convenient to briefly (with two digits) write byte values using the hexadecimal numeral system. Attention! Byte has 256 states, but the maximum value is 255. Do not forget to 0. This is just the 256th state.

# Numeral Systems.

Binary to decimal

101,012 = 1 \* 22 + 0 \* 21 + 1 \* 20 + 0 \* 2-1 + 1 \* 2-2 = 4 + 0 + 1 + 0 + 1/4 = 5,2510

Оctal to decimal

253,318 = 2 \* 82 + 5 \* 81 + 3 \* 80 + 3 \* 8-1 + 1 \* 8-2 = 128 + 40 + 3 + 3/8 + 1/64 = 171 + 0,375 + 0,015625 = 171,39062510

Hexadecimal to decimal

42D16 = 4 \* 162 + 2 \* 161 + 13 \* 160 = 1024 + 32 + 13 = 106910

When translating integers from the decimal number system consistently perform division on the base of the selected number system. The division is performed until the quotient is equal to zero. The number is obtained by "collecting" residues starting from the end.

Decimal to binary

34 / 2 = 17 (0)  
17 / 2 = 8 (1)  
8 / 2 = 4 (0)  
4 / 2 = 2 (0)  
2 / 2 = 1 (0)  
1 / 2 = 0 (1)

3410 = 1000102

Decimal to octal

472 / 8 = 59 (0)  
59 / 8 = 7 (3)  
7 / 8 = 0 (7)

47210 = 7308

Decimal to hexadecimal

924 / 16 = 57 (12)  
57 / 16 = 3 (9)  
3 / 16 = 0 (3)

92410 = 39C16

**Decimal fractions**

Decimals sequentially multiplied by the base of the selected numeral system, until you get a zero fractional part or achieve the required accuracy. Integer is thrown at each subsequent multiplication. The integers of the results constitute a new fraction. Integers are recorded in order from top to bottom.

Decimal fraction to binary

0,225 \* 2 = 0,45  
0,45 \* 2 = 0,9  
0,9 \* 2 = 1,8  
0,8 \* 2 = 1,6  
0,6 \* 2 = 1,2  
0,2 \* 2 = 0,4  
0,4 \* 2 = 0,8  
0,8 \* 2 = 1,6  
…

0,22510 = 0,00111001…2

Decimal fraction to octal

0,225 \* 8 = 1,8  
0,8 \* 8 = 6,4  
0,4 \* 8 = 3,2  
0,2 \* 8 = 1,6  
0,6 \* 8 = 4,8  
…

0,22510 = 0,16314…8

Decimal fraction to hexadecimal

0,225 \* 16 = 3,6  
0,6 \* 16 = 9,6  
0,6 \* 16 = 9,6  
…

0,22510 = 0,699…16

**Operations on Binary Numbers**

In the binary number system arithmetic operations are performed by the same rules as in the decimal system, because they are both positional (along with octal, hexadecimal, etc.).

**Addition**

Addition of one-bit binary numbers is performed by the following rules:

0 + 0 = 0  
1 + 0 = 1  
0 + 1 = 1  
1 + 1 = 10

The least significant bit is overflowed when the two units are summarized. Therefore, the unit is transferred to the most significant bit. Overflow occurs when the sum is equal to the radix (in this case is the number 2) or greater (for the binary system is not relevant).

Let's do the addition of any two binary numbers:

1101

+ 101

------

10010

**Subtraction**

Subtraction of one-digit binary numbers is performed by the following rules:

0 - 0 = 0  
1 - 0 = 1  
0 - 1 = 1 (loan from the most significant bit)  
1 - 1 = 0

Example:

1110

- 101

----

1001

**Multiplication**

Multiplication of one-digit binary numbers is performed by the following rules:

0 \* 0 = 0  
1 \* 0 = 0  
0 \* 1 = 0  
1 \* 1 = 1

Example:

1110

\* 10

------

0000

+ 1110

------

11100

**Division**

The division is performed as well as in the decimal system:

1110 | 10

|----

10 | 111

----

11

10

----

10

10

----

0

**Data coding**

Any information with which modern computer equipment works, converted to a number in the binary numeral system.

The fact that the physical devices (registers, memory locations) can be in two states which correspond to 0 or 1. Using a set of similar physical devices, almost any number in the binary system can be stored in computer memory. Discharge of the number corresponds to the number of memory cells. If eight cells, the number may consist of eight digits.

Coding of integers, fractions and negative, of characters (letters, etc.) is different for each species. For example, to store integers allocated less memory (less cells) than storage fractional regardless of their value.

However, you should always remember that any **information (numeric, text, graphics, sound, etc.) in computer memory is represented as numbers in the binary number system** (almost always).

In general terms, encoding information can be defined as transfer of information provided in the primary message alphabet to a sequence of codes.

It should be understood that any data is somehow encoded information. Information may be presented in different forms: in the form of numbers, text, graphics and other. Transfer from one form to another is a coding.

# The natural numbers (positive integers)

Let the memory cell is equal to 1 byte. Byte contains eight bits. I.e. eight-bit number in the binary numeral system can be stored in one memory cell. Obviously, 00000000 is the minimum number, and 11111111 is the maximum.

If the number 11111111 is converted to decimal system, you get the number 255. I.e. one byte can store positive integers from 0 to 255 inclusive (it is 256 values, that corresponds to 28).

For storage of numbers having a value greater than 255, more bytes are used. Since, two bytes can store a number consisting of 16 binary digits. We can find out the number of possible combinations of zeros and ones to 16-digit number: 216 = 65536. I.e. any number from 0 to 65535 can be stored in two bytes.

We count the number of values that can be saved by using a 4-byte memory location (such cell has 32 bits):

232 = 4 294 967 296,

i.e. more than 4 billion.

**Signed magnitude representation, two's complement**

**Signed magnitude representation**

Signed magnitude representation is a representation of the number in the binary system, in which the first (most significant) bit is given by the number sign. If the number is positive, then 0 is written into the left digit; if the number is negative, then 1 is written.

Thus, using the signed magnitude representation, the seven-digit number in the binary system can be written in eight-digit cell (byte).

For example:

0 00011010 is a positive number

1 00011010 is a negative number

The number of values, that can be placed in the eight-digit cell with a sign in the additional digit, is 256. This coincides with the number of values that can be placed in an eight-digit cell without a sign. However, the range of values is another, it owns values from -128 to 127 inclusive (when translated into decimal system).

However, in computer technique, the signed magnitude representation is used almost exclusively for the presentation of positive numbers.

The so-called *two’s complement* is used for negative numbers. This is due to the convenience of performing operations over the numbers of electronic devices in the computer.

**Two’s complement**

In the two’s complement, as well as in the signed magnitude representation, the first digit is assigned to represent the sign. The signed magnitude representation is used to represent positive numbers; the two’s complement is used to represent negative numbers. Therefore, if the first digit contains 1, then we are dealing with the two’s complement representation and a negative number.

First, all of the other bits are inverted, i.e. they are reversed (0 to 1 and 1 to 0). For example, if 10001100 is the signed magnitude representation of the number, with the formation of its two’s complement, you first need to replace the zeros on the unit and the unit to zero, except the first digit. We get 11110011. But this is not the final form of the two’s complement.

Next, need to add 1:

11110011 + 1 = 11110100

The result is the two’s complement.

The reason why the two’s complement is used to represent negative numbers, due to the fact that it's easier to perform mathematical operations. For example, we have two numbers represented in the signed magnitude. One number is positive, the other is negative. These numbers have to add up. However, they can not simply add up. If one number is negative, then the operation of addition should be replaced by the operation of subtraction. Then the computer must determine which number is larger in absolute value, to determine the sign of the results and determine what to subtract from that. It turns a complex algorithm. Much easier to add numbers if negative numbers are changed to the two’s complement. This can be seen in the examples below.

**The operation of addition of positive and negative numbers represented in the signed magnitude representation**

1. The signed magnitude of 5 is 0000 0101

2. The signed magnitude of -7 is 1000 0111

3. The numbers are compared. In the digit of sign of the result is written sign a greater number.

4. If the numbers have different signs, instead of using the operations of addition is used subtraction the smaller number from the larger number. The first digit is not involved in operation.

\_ 000 0111

000 0101

-------------

000 0010

5. After the operation is taken into account first digit. The result of operations is 1 000 0010 or -210.

**The operation of addition of positive and negative numbers represented in the two’s complement**

1. The signed magnitude of 5 is 0000 0101

2. The signed magnitude of -7 is 1000 0111

3. Formation of the two’s complement of -7.

Inversion: 1 111 1000

Adding units: 1 111 1001

4. The operation of addition.

0 000 0101

+ 1 111 1001

--------------

1 111 1110

Check the result by converting to the signed magnitude.

The two’s complement: 1 111 1110

Subtraction unit: 1 111 1101

Inversion: 1 000 0010 (or -210)

# Real numbers, normalized number

In computer technology, real numbers are numbers that have a fractional part.

Real numbers may contain a large set of digits. For example: 0.0000345 or 10900000 (i.e. it is a very large or very small numbers). For convenience, the real numbers are written in the form of a normalized representation of the number. In this case, the number is written as a product on the radix raised to some degree. For example, in a normalized form, the previous two numbers will look like this: 0.345 \* 10-4 and 0.109 \* 108. Here the numbers 0.345 and 0.109 - mantissa of reals, 10 - radix, 4 and 8 — degree. The dot, that separates whole and fractional part, is placed before the first significant digit (other than 0).

The normalized form of the number is the most convenient to represent fractional numbers in a computer.

It is understood that the normalized representation is not only used for the decimal number system. Here are examples of normalized numbers in binary notation:

101.11 = 0.10111 \* 211

0.001 = 0.1 \* 2-10

Here, the degrees 11 and 10 is a binary form of decimal numbers 3 and 2.

A normalized number is one of the variants of an exponential form of recording of a number.

Let a word consists of two bytes, then the two words are 4 bytes or 32 bits.

Normalized single precision number is stored in memory as follows: number sign - in the 15th bit of the first word (0 for positive and 1 for negative numbers); the degree is placed in 7-14th bits of the first word, and the mantissa occupies remaining 23 bits in two words (0 to 6 bits of the first word and all the bits of the second word). A normalized double-precision number is written into four words, and the mantissa occupies fifty five bits.

The extent of the number represented in floating-point format, ranges from -128 to +127.

Although the mantissa contains 23 bits (for the number of single-precision) or 55 bits (for double-precision numbers), 24 or 56 bits involve in the operations, because most significant bit of the mantissa of normalized numbers are not stored. However, when performing an operation, this bit is automatically restored and taken into account.

Normalized mantissa in the binary system is always represented by a decimal number m, lying in the range 0,5 <= m <1.

Arithmetic operations are performed separately for degrees and mantissas. When the addition is carried out, the degree of the numbers are equalized. When multiplying, the degrees of the numbers add up, and the mantissas are multiplied. When dividing, the degree of the divisor is subtracted from the degree of the dividend, and the mantissas just divide. After the operation, the result is translated into a normalized form. The degree is changed, as each shift by one digit to the left corresponds to a decrease by one degree, and right shift corresponds to an increase by one. The term "floating point" because the binary degree is adjusted after each arithmetic operation, i.e. point "floats" in the number (changes its position). While a fixed-point number, it is rigidly fixed at a particular location.

Arithmetic operations with numbers represented in floating-point format, is much more complicated operations for the same number represented in fixed-point format. But floating point operation allows for automatic scaling in the machine and eliminates the accumulation of absolute error in the calculations (although not eliminate the accumulation of relative error).

# Representation of integers in computer

For simplicity, we will consider the four-bits machine word. This word size provides storage of decimal numbers only from 0 to 15 (when only consider positive integers). However, the observed regularities remain in force for the machine word of any size.

Suppose that the processor is able to increase the (add 1) and to complement (invert) of four-bits word. For example, 1101 is the result of the increase 1100 word. 0011 is the result of addition 1100. Consider the 0000 word, representing the decimal number 0. By increasing the content of the word becomes equal to 0001, corresponding to decimal 1. Continuing successively to increase four-bits words, we arrive at a situation where increasing word 1111 (which is a decimal number 15), we obtain as a result of word of 0000, i.e. 1111 + 1 = (1)0000. We get the wrong arithmetic operation and returned to the original state. This was due to the fact that the memory word may consist of a limited number of bits. Thus, a **computer numerical system is finite and cyclic**.

Bit configuration of 1111 can be taken as code for -1. In this case, 1110 word is interpreted as -2, 1101 as -3, 1000 as -8. Thereby we have a different number system, containing both positive and negative numbers. In this system, half of the configurations (begins with 1) is interpreted as a negative number and the other half (begins with 0) is a positive number or zero. Therefore, the highest bit of a number is called the sign bit. The numerical system with the sign is also finite and cyclical. Arithmetically incorrect result will attempt to increase the number 7 on the unit (0111 + 1 = 1000 = -810).

If the sign bit is zero, the value of the number can be easily calculated. The sign bit is ignored, and the remaining three bits are interpreted as a binary code of a decimal number. For example, the 0110 word is the binary number 110, which is equal to the decimal number 6.

How to understand the value of a negative number? For this purpose it is necessary first to invert the number, and then add one. For example, you want to find the number that is encoded in the 1001 word. To do this, first we perform the inversion operation: 1001 -> 0110. After that, the result is increased by one: 0110 + 1 = 0111. This is the binary code for the decimal number 7. Therefore, 1001 is the value of -7.

**Indicators of transfer and overflow**

Let us consider in more detail the situation, leading to the wrong arithmetic result with an increase of a four-bit number. It arises from the limit of number system of computers. In the unsigned number system, there is a problem with the increase of 1111 word, at the same time there is a transfer of the unit to nowhere. In the case of signed numbers, transfer from most significant bit gives the correct result: 1111 + 0001 = (1)0000 (that's right: -1 + 1 = 0), but the increase 0111 word do result in an error: 0111 + 1 = 1000 (7 + 1 = -8), wherein the transfer takes place in the sign bit.

A processor of a computer (the part that is responsible for the execution of arithmetic operations) has two indicators - the transfer indicator and the overflow indicator. Each indicator comprises a single bit of information, and can be set by a processor (in this case, it is given a value of one) or can be reset (zero). The transfer indicator points to the transfer from the sign bit to beyond of word, and the overflow indicator points to the transfer to the sign bit. After completion of the operation in which the transfer takes place in the most significant bit, the processor sets the overflow indicator; if no such transfer, the overflow indicator is reset. The transfer indicator is handled similarly.

After the operation, the processor indicates the status indicators, and they can be checked. If the indicators point on an incorrect arithmetic result, it is necessary to rectify this situation.

**Topic № 3.**

We have seen how digital computers use the binary system to represent and

manipulate numeric values. We have yet to consider how these internal values

can be converted to a form that is meaningful to humans. The manner in which

this is done depends on both the coding system used by the computer and how the

values are stored and retrieved.

**Binary-Coded Decimal**

*Binary-coded decimal (BCD)* is a numeric coding system used primarily in IBM

mainframe and midrange systems. As its name implies, BCD encodes each digit

of a decimal number to a 4-bit binary form. When stored in an 8-bit byte, the

upper nibble is called the *zone* and the lower part is called the *digit*. (This convention

comes to us from the days of punched cards where each column of the card

could have a “zone punch” in one of the top 2 rows and a “digit punch” in one of

the 10 bottom rows.) The high-order nibble in a BCD byte is used to hold the

sign, which can have one of three values: An unsigned number is indicated with

1111; a positive number is indicated with 1100; and a negative number is indicated

with 1101.

**EBCDIC**

Before the development of the IBM System/360, IBM had used a 6-bit variation

of BCD for representing characters and numbers. This code was severely limited

in how it could represent and manipulate data; in fact, lowercase letters were not

part of its repertoire. The designers of the System/360 needed more information

processing capability as well as a uniform manner in which to store both numbers

and data. In order to maintain compatibility with earlier computers and peripheral

equipment, the IBM engineers decided that it would be best to simply expand

BCD from 6 bits to 8 bits. Accordingly, this new code was called *Extended*

*Binary Coded Decimal Interchange Code* (*EBCDIC*). IBM continues to use

EBCDIC in IBM mainframe and midrange computer systems. The EBCDIC code

is shown in Figure 2.6 in zone-digit form. Characters are represented by appending

digit bits to zone bits. For example, the character *a* is 1000 0001 and the digit

*3* is 1111 0011 in EBCDIC. Note the only difference between upper- and lowercase

characters is in bit position 2, making a translation from upper- to lowercase

(or vice versa) a simple matter of flipping one bit. Zone bits also make it easier

for a programmer to test the validity of input data.

**ASCII**

While IBM was busy building its iconoclastic System/360, other equipment makers

were trying to devise better ways for transmitting data between systems. The

*American Standard Code for Information Interchange* (*ASCII*) is one outcome of

these efforts. ASCII is a direct descendant of the coding schemes used for

decades by teletype (telex) devices. These devices used a 5-bit (Murray) code that

was derived from the Baudot code, which was invented in the 1880s. By the early

1960s, the limitations of the 5-bit codes were becoming apparent. The International

Organization for Standardization (ISO) devised a 7-bit coding scheme that

it called International Alphabet Number 5. In 1967, a derivative of this alphabet

became the official standard that we now call ASCII.

As you can see in Figure 2.7, ASCII defines codes for 32 control characters, 10

digits, 52 letters (upper- and lowercase), 32 special characters (such as $ and #), and

the space character. The high-order (eighth) bit was intended to be used for parity.

*Parity* is the most basic of all error detection schemes. It is easy to implement

in simple devices like teletypes. A parity bit is turned “on” or “off” depending on

whether the sum of the other bits in the byte is even or odd. For example, if we

decide to use even parity and we are sending an ASCII *A*, the lower 7 bits are 100

0001. Because the sum of the bits is even, the parity bit would be set to off and

we would transmit 0100 0001. Similarly, if we transmit an ASCII *C*, 100 0011,

the parity bit would be set to on before we sent the 8-bit byte, 1100 0011. Parity

can be used to detect only single-bit errors. We will discuss more sophisticated

error detection methods in Section 2.8.

To allow compatibility with telecommunications equipment, computer manufacturers

gravitated toward the ASCII code. As computer hardware became more

reliable, however, the need for a parity bit began to fade. In the early 1980s,

microcomputer and microcomputer-peripheral makers began to use the parity bit

to provide an “extended” character set for values between 12810 and 25510.

Depending on the manufacturer, the higher-valued characters could be anything

from mathematical symbols to characters that form the sides of boxes to

foreign-language characters such as n. Unfortunately, no amount of clever tricks

can make ASCII a truly international interchange code.

**Unicode**

Both EBCDIC and ASCII were built around the Latin alphabet. As such, they are

restricted in their abilities to provide data representation for the non-Latin alphabets

used by the majority of the world’s population. As all countries began using

computers, each was devising codes that would most effectively represent their

native languages. None of these were necessarily compatible with any others,

placing yet another barrier in the way of the emerging global economy.

In 1991, before things got too far out of hand, a consortium of industry and

public leaders was formed to establish a new international information

exchange code called Unicode. This group is appropriately called the Unicode

Consortium.

Unicode is a 16-bit alphabet that is downward compatible with ASCII and the

Latin-1 character set. It is conformant with the ISO/IEC 10646-1 international

alphabet. Because the base coding of Unicode is 16 bits, it has the capacity to

encode the majority of characters used in every language of the world. If this

weren’t enough, Unicode also defines an extension mechanism that will allow for

the coding of an additional million characters. This is sufficient to provide codes

for every written language in the history of civilization.

A full Unicode-compliant system will also allow formation of composite characters

from the individual codes, such as the combination of ´ and A to form A. The

algorithms used for these composite characters, as well as the Unicode extensions,

can be found in the references at the end of this chapter.

Although Unicode has yet to become the exclusive alphabet of American

computers, most manufacturers are including at least some limited support for it

in their systems. Unicode is currently the default character set of the Java programming

language. Ultimately, the acceptance of Unicode by all manufacturers

will depend on how aggressively they wish to position themselves as international

players and how inexpensively disk drives can be produced to support an

alphabet with double the storage requirements of ASCII or EBCDIC.

In the computer memory, decimal numbers are coded with the binary numeral system. To do this, there are simple and clear rules for translation. However, today the computer is used not only for the labor-intensive mathematical calculations. Computer memory also stores text and multimedia information. It raises the first question:

**How are symbols (letters) stored in the computer's memory?**

Each letter belongs to a particular alphabet in which characters follow each other. Hence, the symbols can be numbered with consecutive integers.

**Each letter can be associated with a positive integer called a symbol code.**

This code will be stored in computer memory, while displaying on the screen the code will be replaced by a corresponding symbol. To distinguish the representation of numbers and representation of symbols in the computer's memory, the information is also stored on what type of data is encoded in a specific area of memory. **Symbol table is a correspondence of letters and their codes.**In other words, each symbol of the alphabet has a particular code number in the table.

However, there are many alphabets in the world (English, Russian, Chinese, and others.). Therefore, the following question:

**How to encode all alphabets used in the computer?**

In the 60-ies of XX century character encoding table was developed, it was used in all operating systems. This table is called ASCII (American Standard Code for Information Interchange). Later there was an enhanced version of ASCII.

In accordance with the table of ASCII to represent one character is allocated 1 byte (8 bits). A set of eight bits can take 28 = 256 different values. The first 128 values (from 0 to 127) are constant and form the so-called main part of the table, which include decimal digits, letters of the alphabet (uppercase and lowercase), punctuation (dot, comma, brackets, etc..), as well as the space and a variety of special characters (tab, line feed, and others.). Values from 128 to 255 form an additional part of the table, where it is assumed to encode national characters.

Since there are a lot of national alphabets, then there are many options of extended ASCII-tables. This creates additional difficulties. For example, we send a letter, written in one encoding, and the receiver is trying to read it in another. As a result, he sees gibberish. He needs to use a different table for text encoding.

There is another problem. In some languages, alphabets contain too many characters and they do not fit into their assigned positions from 128 to 255 of single-byte encodings.

The third problem - what to do if the text uses multiple languages (such as English and French)? You can not use two tables at once.

Unicode table was developed to solve these problems at one time.

**Unicode standard of coding symbols**

Unicode character encoding standard was developed in the early 90s. This standard allows the use of any language and symbols in the text.

Unicode provides 31 bytes (4 bytes minus one bit) for encoding characters. The number of possible combinations is a huge number: 231 = 2147483684 (i.e. more than two billion). Therefore, Unicode describes alphabets of all known languages, including many mathematical and other special symbols. However, the information capacity of 31-bit Unicode is still too big. So increasingly used abbreviated 16-bit version (216 = 65536 values), which encodes all modern alphabets.

In Unicode, the first 128 codes coincide with the table ASCII.

# Audio coding

**Analog and digital information**

From the point of view of physics, sound is a wave of pressure fluctuations in a particular environment. A sound is something indivisible into parts (continuous) running through time and space. To record a sound on any medium, a level (strength) of a sound can be correlated to any measured media specifications. For example, the power of magnetization of the magnetic tape in its different areas is depended on the characteristics of the sound that was recorded to it. Magnetization may continuously change over the tape, just like the sound parameters are changing in the air. I.e. a magnetic tape perfectly suitable for storing sound. A tape stores a sound in the so-called analog form when the value is changed continuously (smoothly), that is close to the natural sound.

But how sound is stored on a computer. Here, any information is represented in a digital form. The data should be presented by a numbers; therefore, an information in the computer is discrete (separated). To record audio on digital media (eg, hard drive), it is subjected to so-called digitization. The mechanism of digitization is to measure the sound parameters over a certain period of time (very small).

**Sampling and quantization**

When converting an audio information into a digital form it is subjected to sampling and quantization. Sampling is a measurement of the analog signal a myriad of times per second. The resulting value of the analog signal is mapped to a specific value of the previously selected range: 256 (8 bits) or 65536 (16 bits). Quantization is the alignment of the signal level to a certain value range.

It is clear that no matter how often we measure, a part of the information will be lost. However, than more often we do the measurements, then the more accurate will match a digital audio to an analog original. Also, the more bits allocated for coding the signal (quantization), the more accurate matching.

On the other hand, good audio quality will contain more data; therefore, take up more space on digital media.

As an example, the following calculations. For high-quality music recording an analog audio signal is measured more than 44 000 times per second and is quantized by 2 bytes (16 bits give a range of 65536 values). Ie per second, 88000 bytes of information is recorded. It is approximately 86 KB (88000 ÷ 1024). One minute will cost the 5168 Kbytes (86 × 60), ie, almost 5 MB.

**Audio coding**

**Analog and digital information**

From the point of view of physics, sound is a wave of pressure fluctuations in a particular environment. A sound is something indivisible into parts (continuous) running through time and space. To record a sound on any medium, a level (strength) of a sound can be correlated to any measured media specifications. For example, the power of magnetization of the magnetic tape in its different areas is depended on the characteristics of the sound that was recorded to it. Magnetization may continuously change over the tape, just like the sound parameters are changing in the air. I.e. a magnetic tape perfectly suitable for storing sound. A tape stores a sound in the so-called analog form when the value is changed continuously (smoothly), that is close to the natural sound.

But how sound is stored on a computer. Here, any information is represented in a digital form. The data should be presented by a numbers; therefore, an information in the computer is discrete (separated). To record audio on digital media (eg, hard drive), it is subjected to so-called digitization. The mechanism of digitization is to measure the sound parameters over a certain period of time (very small).

**Sampling and quantization**

When converting an audio information into a digital form it is subjected to sampling and quantization. Sampling is a measurement of the analog signal a myriad of times per second. The resulting value of the analog signal is mapped to a specific value of the previously selected range: 256 (8 bits) or 65536 (16 bits). Quantization is the alignment of the signal level to a certain value range.

It is clear that no matter how often we measure, a part of the information will be lost. However, than more often we do the measurements, then the more accurate will match a digital audio to an analog original. Also, the more bits allocated for coding the signal (quantization), the more accurate matching.

On the other hand, good audio quality will contain more data; therefore, take up more space on digital media.

As an example, the following calculations. For high-quality music recording an analog audio signal is measured more than 44 000 times per second and is quantized by 2 bytes (16 bits give a range of 65536 values). Ie per second, 88000 bytes of information is recorded. It is approximately 86 KB (88000 ÷ 1024). One minute will cost the 5168 Kbytes (86 × 60), ie, almost 5 MB.

**Topic № 4.**

The purpose of a color model is to facilitate the specification of colors in some standard generally accepted way. In essence, a color model is a specification of a 3-D coordinate system and a subspace within that system where each color is represented by a single point.

Each industry that uses color employs the most suitable color model. For example, the RGB color model is used in computer graphics, YUV or YCbCr are used in video systems, PhotoYCC\* is used in PhotoCD\* production and so on. Transferring color information from one industry to another requires transformation from one set of values to another. Intel IPP provides a wide number of functions to convert different color spaces to RGB and vice versa.

The screen in graphics mode is divided into points (pixels). The image is formed by combining various colors of all the points.

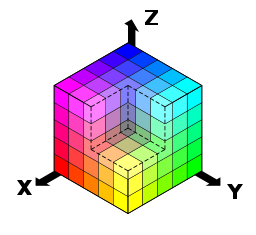
Color coding may be different. To create an image on the screen, the RGB (Red, Green, Blue) color model is commonly used.

All variety of colors is achieved by mixing the three primary colors in various proportions. In other words, any color is characterized by certain proportions of red, green and blue colors. Therefore, on the screen, each pixel of a particular image is described by a set of three colors with a certain level of brightness. Various combination of brightness of these colors (red, green, and blue) creates a whole spectrum of colors that we see on the screen.

How many levels of brightness is for each color? The answer to this question will reveal how many different colors and shades can take each point.

If each of the three may be only two states (0 and 1), we can get only 23 = 8 colors. For example, if the red and green are at the maximum (1), and the blue is off (0), you get a clean yellow.

However, in modern computers, there are gradations of brightness much more than two. Tens of thousands, even hundreds of thousands of colors are obtained. Increasing the resolution (number of pixels) and number of colors increases the amount of information stored in video memory for display a particular image.



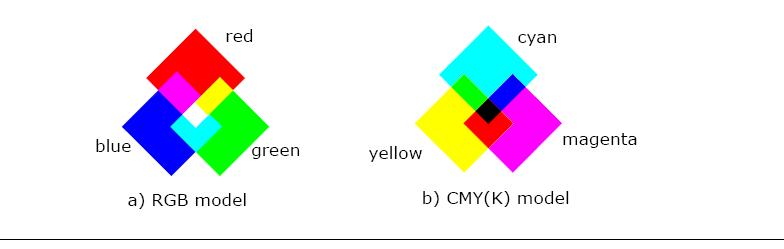
Coding color using the RGB model is conveniently represented as a color cube. Each color corresponds to a point inside the cube. The origin of coordinates corresponds to black (the complete absence of any color). Across the top of the cube is the point corresponding to white (maximum brightness for each color).

If you take any point inside the cube and hold a segment to it from the origin of coordinates, you obtain a smooth transition of a certain color from dark to light shade.

The most saturated colors are located on the faces of the cube, the less saturated colors are inside of the cube (begin to mix into shades of gray).

In the RGB model, each color appears as a combination of red, green, and blue. This model is called additive, and the colors are called primary colors. The primary colors can be added to produce the secondary colors of light (see Figure "Primary and Secondary Colors for RGB and CMYK Models") - magenta (red plus blue), cyan (green plus blue), and yellow (red plus green). The combination of red, green, and blue at full intensities makes white.

**Primary and Secondary Colors for RGB and CMYK Models**



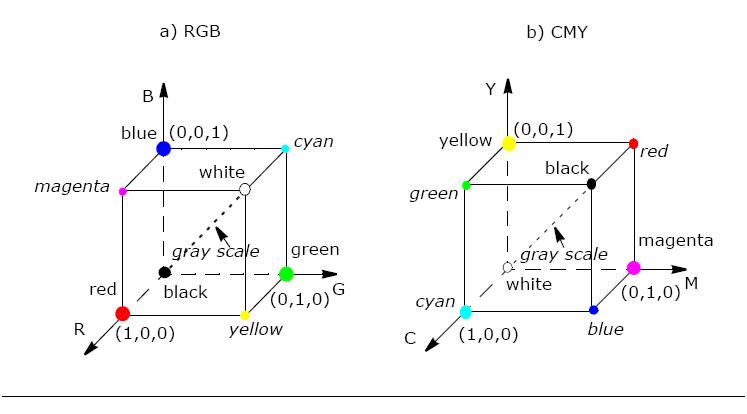
The color subspace of interest is a cube shown in *Figure "RGB and CMY Color Models"* (RGB values are normalized to 0..1), in which RGB values are at three corners; cyan, magenta, and yellow are the three other corners, black is at their origin; and white is at the corner farthest from the origin.

The gray scale extends from black to white along the diagonal joining these two points. The colors are the points on or inside the cube, defined by vectors extending from the origin.

Thus, images in the RGB color model consist of three independent image planes, one for each primary color.

As a rule, the Intel IPP color conversion functions operate with non-linear gamma-corrected images R'G'B'.

The importance of the RGB color model is that it relates very closely to the way that the human eye perceives color. RGB is a basic color model for computer graphics because color displays use red, green, and blue to create the desired color. Therefore, the choice of the RGB color space simplifies the architecture and design of the system. Besides, a system that is designed using the RGB color space can take advantage of a large number of existing software routines, because this color space has been around for a number of years.



However, RGB is not very efficient when dealing with real-world images. To generate any color within the RGB color cube, all three RGB components need to be of equal pixel depth and display resolution. Also, any modification of the image requires modification of all three planes.

CMYK Color Model

The CMYK color model is a subset of the RGB model and is primarily used in color print production. CMYK is an acronym for cyan, magenta, and yellow along with black (noted as K). The CMYK color space is subtractive, meaning that cyan, magenta yellow, and black pigments or inks are applied to a white surface to subtract some color from white surface to create the final color. For example (see [Figure "Primary and Secondary Colors for RGB and CMYK Models"](https://software.intel.com/en-us/node/503873#FIG6-2)), cyan is white minus red, magenta is white minus green, and yellow is white minus blue. Subtracting all colors by combining the CMY at full saturation should, in theory, render black. However, impurities in the existing CMY inks make full and equal saturation impossible, and some RGB light does filter through, rendering a muddy brown color. Therefore, the black ink is added to CMY. The CMY cube is shown in [Figure "RGB and CMY Color Models"](https://software.intel.com/en-us/node/503873#FIG6-3), in which CMY values are at three corners; red, green, and blue are the three other corners, white is at the origin; and black is at the corner farthest from the origin.

YUV Color Model

The YUV color model is the basic color model used in analogue color TV broadcasting. Initially YUV is the re-coding of RGB for transmission efficiency (minimizing bandwidth) and for downward compatibility with black-and white television. The YUV color space is “derived” from the RGB space. It comprises the *luminance* (Y) and two color difference (U, V) components. The luminance can be computed as a weighted sum of red, green and blue components; the color difference, or *chrominance*, components are formed by subtracting luminance from blue and from red.

The principal advantage of the YUV model in image processing is decoupling of luminance and color information. The importance of this decoupling is that the luminance component of an image can be processed without affecting its color component. For example, the histogram equalization of the color image in the YUV format may be performed simply by applying histogram equalization to its Y component.

There are many combinations of YUV values from nominal ranges that result in invalid RGB values, because the possible RGB colors occupy only part of the YUV space limited by these ranges. [Figure "RGB Colors Cube in the YUV Color Space"](https://software.intel.com/en-us/node/503873#FIG6-4) shows the valid color block in the YUV space that corresponds to the RGB color cube RGB values that are normalized to [0..1]).

The Y'U'V' notation means that the components are derived from gamma-corrected R'G'B'. Weighted sum of these non-linear components forms a signal representative of luminance that is called *luma*Y'. (*Luma* is often loosely referred to as *luminance*, so you need to be careful to determine whether a particular author assigns a linear or non-linear interpretation to the term *luminance*).

HSV, and HLS Color Models

The HLS (hue, lightness, saturation) and HSV (hue, saturation, value) color models were developed to be more “intuitive” in manipulating with color and were designed to approximate the way humans perceive and interpret color.

*Hue* defines the color itself. The values for the hue axis vary from 0 to 360 beginning and ending with red and running through green, blue and all intermediary colors.

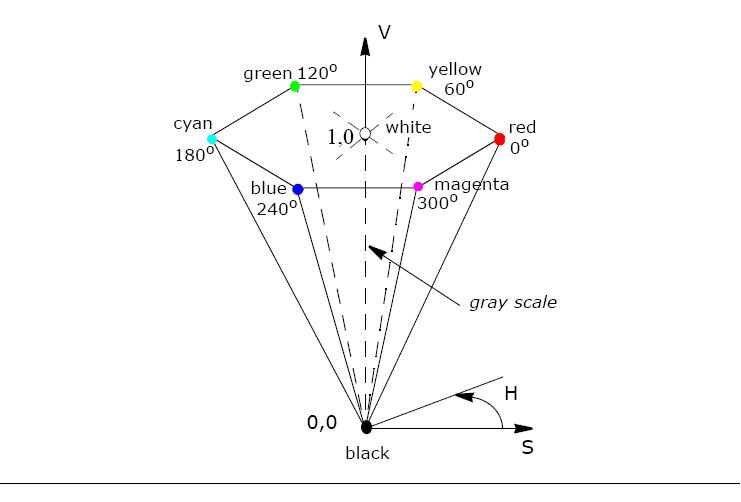
*Saturation* indicates the degree to which the hue differs from a neutral gray. The values run from 0, which means no color saturation, to 1, which is the fullest saturation of a given hue at a given illumination.

Intensity component - *lightness* (HLS) or *value* (HSV), indicates the illumination level. Both vary from 0 (black, no light) to 1 (white, full illumination). The difference between the two is that maximum saturation of hue (S=1) is at *value* V=1 (full illumination) in the HSV color model, and at *lightness* L=0.5 in the HLS color model.

The HSV color space is essentially a cylinder, but usually it is represented as a cone or hexagonal cone (hexcone) as shown in the [Figure "HSV Solid"](https://software.intel.com/en-us/node/503873#FIG6-8), because the hexcone defines the subset of the HSV space with valid RGB values. The *value* V is the vertical axis, and the vertex V=0 corresponds to black color. Similarly, a color solid, or 3D-representation, of the HLS model is a double hexcone ([Figure "HSV Solid"](https://software.intel.com/en-us/node/503873#FIG6-9)) with *lightness* as the axis, and the vertex of the second hexcone corresponding to white.

Both color models have intensity component decoupled from the color information. The HSV color space yields a greater dynamic range of saturation. Conversions from [RGBToHSV](https://software.intel.com/node/7c2e7827-7a7d-484e-868a-4c41ba7766d1#7C2E7827-7A7D-484E-868A-4C41BA7766D1)/[RGBToHSV](https://software.intel.com/node/fc891e9d-0a09-4092-a857-7682aa269405#FC891E9D-0A09-4092-A857-7682AA269405) and vice-versa in Intel IPP are performed in accordance with the respective pseudocode algorithms [[Rogers85]](https://software.intel.com/node/404ef43a-5c56-44d5-b061-9c0948bd8e40#ROGERS85) given in the descriptions of corresponding conversion functions.

**HSV Solid**



Vector images

One way to describe an image using numbers is to declare its contents using position and size of geometric forms and shapes like lines, curves, rectangles and circles; such images are called vector images.

The description of an image can be seen as a “cooking recipe” for how to draw the image, it contains geometrical primitives like lines, curves and cirles describing color as well as relative size, position and shape of elements. When preparing the image for display is has to be translated into a ***bitmap image***, this process is called ***rasterization***.

A vector image is resolution independent, this means that you can enlarge or shrink the image without affecting the output quality. Vector images are the preferred way to represent Fonts, Logos and many illustrations.

Bitmap images

Bitmap-, or raster [[1](http://pippin.gimp.org/image_processing/chap_dir.html#ftn.id2549850)] -, images are “digital photographs”, they are the most common form to represent natural images and other forms of graphics that are rich in detail. Bitmap images is how graphics is stored in the video memory of a computer. The term bitmap refers to how a given pattern of bits in a pixel maps to a specific color.



A rasterized form of the letter 'a' magnified 16 times using pixel doubling

A bitmap images take the form of an array, where the value of each element, called a ***pixel*** picture element, correspond to the color of that portion of the image. Each horizontal line in the image is called a ***scan line***.

The letter 'a' might be represented in a 12x14 matrix as depicted in Figure 3., the values in the matrix depict the brightness of the ***pixels*** (picture elements). Larger values correspond to brighter areas whilst lower values are darker.

Sampling

When measuring the value for a pixel, one takes the average color of an area around the location of the pixel. A simplistic model is sampling a square, this is called a box filter, a more physically accurate measurement is to calculate a weighted Gaussian average (giving the value exactly at the pixel coordinates a high weight, and lower weight to the area around it). When perceiving a bitmap image the human eye should blend the pixel values together, recreating an illusion of the continuous image it represents.

Raster dimensions

The number of horizontal and vertical samples in the pixel grid is called ***Raster dimensions***, it is specified as width x height.

Resolution

Resolution is a measurement of sampling density, resolution of bitmap images give a relationship between pixel dimensions and physical dimensions. The most often used measurement is ppi, pixels per inch .

**Topic 5**

Analog and digital computers

Depending on the type of information being processed, computing machines are divided into two main classes: analog and digital.

An **analog computer** is a computer that operates of information provided in a form of a continuous change of some physical quantities. The physical variables are power of the current in a circuit, a rotation angle of the shaft, speed and acceleration of a body, etc. Using the fact that many phenomena in nature mathematically described by the same equations, analog computers can simulate other processes using a physical process.

A **digital computer** is a computer that operates with information presented in a discrete form. Currently, methods for the numerical solution of many types of equations are developed. This is made possible on digital computers to solve various equations and objectives through a set of simple arithmetic and logical operations. Therefore, if an analog computers are typically designed to address a specific class of problems, i.e., are specialized, a digital computer is typically a general purpose computing device. Electronic computers, which are made using the latest achievements of electronics, are widely spread.

# Computer Generations

In the history of computer technology, several periods are allocated depending on what a basic elements used to manufacture of a computer. The division into periods is conditional to a certain extent, because when an old generation computers were producing, a new generation was beginning to gain momentum.

**There are general tendencies of development of computers:**

1. Increasing the number of elements per unit area.
2. The reduction in size.
3. Increased speed of operation.
4. Reducing the cost.
5. The development of software, on the one hand, and the simplification, standardization of hardware - the other side.

## The zero generation. Mechanical calculators

Background to the emergence of a computer was forming from ancient times, but often the review is begun with the calculating machine of Blaise Pascal, which he designed in 1642. This machine could perform only the operations of addition and subtraction. In the 70s of the same century, Leibnitz built a machine, able to perform not only addition and subtraction, but also multiplication and division.

In the XIX century, Charles Babbage made the great contribution to the future development of computer technology. His Difference Engine was able only to add and subtract. But the results of the calculations is extruded onto the copper plate (analogue means of input-output). Later described by Babbage Analytical Engine was must to perform all four basic math operations. Analytical machine consisted of memory, computational engine and input-output devices (as a computer, but mechanical) and a machine could perform various algorithms (depending on which card was in the input device). Ada Lovelace was writing a program for the Analytical Engine (the first known programmer). In fact, the machine was not realized at that time because of the technical and financial difficulties. World lagged behind the train of Babbage's thought.

In XX century, Conrad Zus, George Stibits, John Atanasoff were constructing automatic counting machines. Atanasoff's machine included a prototype of RAM and used binary arithmetic. Howard Aiken's relay computers "Mark I» and «Mark II» were similar in architecture with Babbage's Analytical Engine.

## The first generation. Bulb computers (1940-1955)

Performance: tens of thousands of operations per second.

**Features:**

* Since the bulbs have substantial dimensions and their thousands, the machines had enormous dimensions.
* As many lamps, they have a tendency to burn out, so the computer is often idle for search and replacement of defective bulbs.
* The lamps emit a large amount of heat, therefore, computers require special powerful cooling system.

**Examples of computers:**

The **Colossus** is a secret development of the British government (Alan Turing was involved in the development). This is the first electronic computer in the world, but he had no influence on the development of computer technology (because of its privacy).

**ENIAC**. Creators: John Mouchli, J. Presper Eckert. Machine weight is 30 tons. Cons: the use of the decimal number system; a plurality of switches and cables.

**Edsak**. Achievement: The first machine with a program in memory.

**Whirlwind I**. Words of small length, work in real-time.

The **Computer 701** (and later models) from IBM. This is the first computer, leading on the market for 10 years.

ENIAC

## The second generation. Transistor computers (1955-1965)

Performance: hundreds of thousands of operations per second.

Compared with bulbs, the transistors reduced the size of computers, increase reliability and the speed of operation (up to 1 million instructions per second) and greatly reduce the heat transfer. Ways to store information are developing: magnetic tape is widely used, drives appear later. A first computer game appeared.

The first transistorized computer TX became the prototype for computers PDP branch company DEC, which can be considered the ancestor of the computer industry, because there was the phenomenon of the mass sale of machines. DEC releases a first mini computer (it size is like a closet). It was noted the appearance of the display.

PDP-7

IBM has also actively working, producing already transistor versions of their computers.

CDC company's 6600 computer, which has developed Seymour Cray, had an advantage over the other computers of the time - it was its speed, which was achieved by parallel execution of commands.

## The third generation. Computers on integrated circuits (1965-1980)

Performance: millions of instructions per second.

An integrated circuit is an electronic circuit made on a silicon crystal. That scheme accommodates thousands of transistors. Consequently, computers of this generation have become smaller, faster and cheaper. Becoming cheaper, computers began to penetrate into various spheres of human activity. Because of this, they became more specialized (ie, were different computers for different tasks).

There was a compatibility problem for the produced models (software for them). IBM was the first company who paid great attention to compatibility.

There was realized multiprogramming in which several executable programs stored in memory. This has the effect of saving processor resources.

## The fourth generation. Computers on large (and very large) integrated circuits (1980- …)

Performance: hundreds of millions of operations per second.

The opportunity came to place thousands of integrated circuits on a single chip instead of one as before. Computer speed has increased significantly. Computers were continuing to become cheaper, and now many people could buy them. This marked the so-called era of personal computers. However, most people are not professional programmers. Consequently, it took the development of user-friendly software so that people can easily use it.

In the late 70's - early 80's popular to use was a computer Apple, developed by Steve Jobs and Steve Wozniak. Later in the mass production it was launched the IBM PC with Intel processor.

IBM PC

Later, superscalar processors appeared, they are able to perform multiple commands at the same time, as well as 64-bit computers appeared.

## The fifth generation?

This include the failed project of Japan. Other sources refer to the fifth generation of computers so called the invisible computers (microcontrollers, embedded in household appliances, cars, etc..) or handheld computers.

There is also a view that the fifth generation of computers should be assigned to the dual-core processors. From this viewpoint, the fifth generation began about 2005.

# Types of Computers

Familiar to most people, personal computers (PC) are not the only type of computers. Typically, computers are classified according to performance and method of use.

## Personal computers (PC)

There are stationary and portable (laptop) PCs.

A personal computers always have a monitor and other peripherals. A PC's block has a motherboard, CPU, different memory (RAM, hard drive), input-output devices, peripherals interfaces and other.

A PC have good extensibility. They are easy to connect a variety of additional devices. On a PC, you can install a wide range of different software.

## Gaming Computers

Compared with personal computers, game computers have enhanced multimedia capabilities (sound, video, interactivity), but there are restrictions on the amount of software, as well as the possibility of further expansion (new devices). Gaming computers do not have the monitor and hard drive.

As an example of a gaming computer can lead Sony PlayStation.

Prices for a gaming computers are typically lower than a PC.

## Mobile devices (Handhelds)

They are similar to personal computers, but less than their size. Commonly used as electronic diaries or to read e-books.

Microcontrollers are set for various household and other electronic devices (cell phones, washing machines, printers, televisions, cars, and others.). They provide the ability to manage the device.

A microcontroller, despite its size, is a full-fledged computing device, because has a memory, processor and input-output means. The program for the microcontroller is normally set by its manufacturer, there is no possibility of change it in the future.

Microcontrollers are manufactured in large quantities.

## Servers

Unlike a PC, servers are more potent, and may not have a monitor, and other peripherals. They are used in the networks.

Servers usually have more memory (RAM and hard drive) and high-speed network interfaces. A server stores data and programs (a file server and an application server). Server's processor is usually manages users and permissions to access the data. Calculations are made on the client computer.

## Mainframe computers

Mainframes are big computers (the size of the room), producing centralized processing of large volumes of data. Users can access through terminals (keyboard + monitor) and / or a PC, mainly intended for input and output. Number of connected terminals is typically several hundred. Mainframes are characterized by high reliability. Mainframe power is greater than PCs and servers, but not much. But they have a high-speed input-output processes and have increased the size of constant memory.

Mainframes are expensive enough (within a million dollars). They are used in large organizations (banks, airports, government agencies).

## Supercomputers

Supercomputers are a very powerful systems (have powerful processor). They appeared in the 60s. They are used for tasks that require large volumes of complex calculations (such as space exploration, weather forecasting), their price is tens of millions of dollars.

## Workstations

Workstations, as well as personal computers, are designed for a single user, how

# History of personal computers

Since 1975, the US began mass production of personal computers (PC). This event is often referred to as a second information revolution (the first information revolution is the appearance of the printing press and printing books in 1445). PCs appeared on the basis of mini- and micro-computers for personal computing, ie for a specialist in a particular subject area at their workplace. PCs began to use by people who do not know how to program. Since 1981, personal computers were produced, having a block-modular design. It was easy to use and relatively cheap machines. They are intended for users who do not have knowledge in computer science and programming. Widespread minicomputers in the early 1970s. dictated by the need to bring the computer to the user. Mini-computer installed in businesses and organizations where using mainframes was uneconomical.

Thus, the PC is a computer designed for individual use. Currently, it is a powerful general-purpose computer; it works successfully both at home and in the workplace in the office, easily connected to various computer networks.

The main features of the PC is the small size, no need for maintenance, low price, functional versatility and ease of upgrade.

The technical basis of the PC is a microprocessor, its development determines the change of generations of personal computers:

1. 8-bit microprocessor (1975 - 1980);
2. 16-bit (1981 - 1985);
3. 32-bit (1986 - 1992);
4. 64-bit (1993 – present).

An important role in the development of the PC played the advent of computers IBM PC, produced by the corporation IBM (USA) in 1981. The PC is based on the Intel-8086 microprocessor. This PC has taken a leading position in the market. Its main advantage is the so-called open architecture, through which users can extend the computer capabilities by adding various peripherals and upgrading a computer.In the future, other companies have begun to create their own PC, but IBM PC became the standard. Today, more than 80% of all sold PCs is based on the architecture of the IBM PC.

Personal computers are used at home and in professional activities. Computers are used to working with text and graphics, the solution of scientific and engineering tasks, entertainment, education and others. The architecture of the PC allows to connect their to networks, expand the set of peripheral equipment. Computers are equipped with advanced software, including operating systems, compilers with algorithmic languages, application packages.

Professional PCs are used in science to solve complex information and industrial applications that require high performance, efficient transmission of large amounts of data, large capacity memory.

By connecting a large set of peripherals, PC functionality is greatly enhanced.

# Von Neumann architecture

In 1946 J. von Neumann, G. Goldstein and A. Burks in their joint article described the new principles of construction and operation of a computer. Subsequently, based on these principles were made the first two generations of computers. In later generations, there were some changes, although the von Neumann principles are relevant today.

Neumann was able to compile scientific research and the discovery of many other scientists and to formulate on their basis of a fundamentally new.

## The principles of von Neumann

1. **Using the binary system in computers.** The advantage to the decimal system is that the device can be made quite simple, arithmetic and logical operations in the binary system is also easy.
2. **Software control of the computer.** Work of a computer is controlled by a program consisting of a set of commands. The commands are executed one after the other. The creation of the machine with a stored program was the beginning of what we now call programming.
3. **Computer memory is used to store not only data, but also software.** Program instructions and data are encoded in binary form, i.e. recording method is the same. Therefore, in certain situations the same actions can be performed over commands as over data.
4. **The cells of computer memory have a addresses that are sequentially numbered.** At any time, you can access any memory location using its address. This principle has opened the possibility to use variables in programming.
5. **Possibility of a conditional jump in the program.** Despite the fact that the commands are executed sequentially, in the programs can be implemented transitions to any code portion.

The most important consequence of these principles can be called that now the program has not been a permanent part of the machine (as for example, the calculator). The program was made possible to easily change. But the equipment remains the same, and very simple.

For comparison, the computer program for ENIAC (which did not stored the program in memory) was defined with a special panel of jumpers. To reprogram the machine (set the jumpers on another) may need far more than a day. Although programs for modern computers can be written years, but they are working on millions of computers after a few minutes of installation on a hard disk.

## How does the von Neumann machine work?

Von Neumann machine consists of a storage device (memory), an arithmetic logic unit (ALU), control unit (CU) and input and output devices.

Programs and data are entered into the memory from the input device through an arithmetic logic unit. All program instructions are stored in adjacent memory cells, and data can be contained in an arbitrary cells. The last command of any program must be the shutdown command.

A command consists of specifying what operation should be performed (of the possible operations on this hardware) and address memory, which stores data on which to perform the specified operation, as well as cell address where need to record it (if need to store in the memory).

ALU performs the specified with commands operations over the specified data.

From ALU, results go in memory or output device. The principal difference between the memory and the output device is that the data is stored in the memory in a form suitable for processing by a computer, and on the output device (printer, monitor, etc.) so as convenient to the person.

The control unit controls all parts of the computer. Other devices receive signals "what to do" from the control unit, and from the other devices the CU receives information about their condition.

The control unit comprises a special register (cell) called "counter". After downloading the program and data in memory, address of the first instruction of the program is written to the counter. CU reads the contents of the memory cell whose address is stored in the program counter, and puts it in a special device - "Command register". CU defines an operation command, "notes" in memory data addresses specified in the command and control of the command. ALU or computer hardware is performing the command.

After each command, the counter changes by one and, therefore, points to the next command of the program. When need to execute a command that does not follow the order of the current command, the special transition command contains the address of the cell where need to transfer control.

# Computer principles

A computer is a technical means of converting information. The basis of his work is contained the same principles of processing electrical signals as in any electronic device:

1. Provided by various physical processes both electric and non-electric nature (letters, numbers, beeps, etc.), the input information is converted into an electrical signal.
2. Signals are processed in the processing unit.
3. Via the converter of output signals, the processed signals are converted into non-electrical signals (the screen image).

Assignment of a computer is a processing various types of information and presenting it in a human readable format.

From the standpoint of functionality, a computer is a system consisting of 4 basic devices that perform certain functions: memory, which is divided into an operational and constant, arithmetic logic unit (ALU), the control unit (CU) and input-output devices (IO). Consider their role and purpose.

A memory is designed for storing information and program instructions. Stored in the memory information is encoded numbers, characters, words, commands, addresses, etc with 0 and 1.

Recording a number in the memory is its location in the cell at the specified address and keeping it there until the selecting with program command. When writing, the previous information stored in the cell is overwritten. When programming, for example, Pascal or C, the cell address is associated with the name of a variable that represents a combination of letters and numbers, selected by the programmer.

Reading a number from memory is its selecting from a cell at the specified address. The copy of the number is transmitted from the memory to the desired device, and the number itself remains in the cell.

Transfer of information means that information is read from one cell and recorded to another.

Address of the cell is formed in the control unit (CU), and then enters the address selection device which opens a channel and connects the desired cell.

Numbers, characters, instructions are stored in the memory on an equal basis and have the same format. The type of data does not matter neither for the memory nor the computer. The types differ only in data processing with a program. The length of the cell is determined with the number of binary digits (bits). Each bit may contain 1 or 0. In modern computers, cell length is a multiple of 8 bits and is measured in bytes. The minimum length of the cell, for which it is possible to form the address, is 1 byte consisting of 8 bits.

The following parameters are used for describing of memory:

* The memory capacity is the maximum number of stored data (in bytes).
* Speed memory is a memory access time determined by the time of reading or the time of recording of information.

The arithmetic logic unit (ALU) performs arithmetic and logical operations.

It should be noted that any arithmetic operation can be implemented using addition.

A complex logic problem is decomposed into simpler tasks, which is enough to analyze only two levels: yes and no.

The control unit (CU) controls the whole course of the computational and logical process in the computer, ie, it acts as a "traffic controller" of information. The control unit reads the command, decode it, and connect the necessary circuitry to accomplish it. Reading of the following command is automatically.

In fact, CU performs the next cycle of action:

* the formation of the address of the next instruction;
* reading of commands from the memory and its interpretation;
* performing a command.

In modern computers, the functions of CU and ALU are performed with a single device called a CPU.

Consider a typical desktop computer. It has a system unit which is the case or

box that houses the motherboard, other printed circuit boards, the storage devices, and the

power supply. The system unit is generally designed in such a way that it can be easily

opened to add or replace modules. The di\_erent components in the system unit are typically

connected together using a bus, which is a set of wires for transferring electrical signals. Each

printed circuit board houses a number of chips, some of which are soldered and the rest are

plugged into the board. The latter permits the user to upgrade the computer components.

Circuits etched into the boards act like wires, providing a path for transporting data from

# one chip to another.

**Processor:** The processor, also called the central processing unit (CPU), is perhaps the

most important part of a computer. It carries out the execution of the instructions of a

# program.

**Chip Sets**: The chipsets provide hardware interfaces for the processor to interact with

# other devices, such as DRAM and graphics cards.

**Motherboard:** The motherboard is the main printed circuit board, and holds the computer's

processor chip(s), ROM chips, RAM chips, and several other key electronic components.

The processor is an important part of a computer, and can be a single chip or

a collection of chips. ROM chips typically contain a small set of programs that start the

computer, run system diagnostics, and control low-level input and output activities. These

programs are collectively called BIOS (basic input output system) in PCs. The instructions

in the ROM chips are permanent, and the only way to modify them is to reprogram the

ROM chips. RAM chips are volatile and hold program and data that is temporary in nature.

A battery powered real-time clock chip keeps track of the current date and time. The

motherboard also typically contains expansion slots, which are sockets into which expansion

cards such as video card, sound card, and internal modem, can be plugged in. An expansion

card has a card edge connector with metal contacts, which when plugged into an expansion

slot socket, connect the circuitry on the card to the circuitry on the motherboard. The

# number of expansion slots in the motherboard determines its expandability.

**Storage Devices**: The commonly used storage devices are oppy disk drives, hard disk

drives, CD-ROM drives, and ZIP drives. A oppy disk drive is a device that reads and

writes data on oppy disks. A typical oppy disk drive uses 3 1

2 -inch oppy disks each of

which can store up to 1.44 MB. A hard disk drive can store billions of bytes on a nonremovable

disk platter. A CD-ROM drive is a storage device that uses laser technology to

read data from a CD-ROM. The storage devices are typically mounted in the system unit.

The ones involving removable media such as the oppy disk drive, the CD-ROM drive, and

the ZIP drive are mounted on the front side of the system unit, and the hard disk drives

# are typically mounted inside the system unit.

**Input/Output Devices**: Two of the commonly used input devices in a desktop computer

are the keyboard and the mouse. A computer keyboard looks similar to that of a typewriter,

with the addition of number keys, as well as several additional keys that control computerspeci

\_c tasks. The mouse is useful in manipulating objects depicted on the screen. Other

commonly used input device is the microphone. The primary output device in a desktop

computer is the monitor, a display device that forms an image by converting electrical

signals from the computer into points of colored light on the screen. Its functioning is very

to a television picture tube, but has a much higher resolution so that a user sitting at close

quarters can clearly see computer-generated data such as text and images. Other frequently

# used output devices are the printer and the speakers.

# Device Controllers:

Each device|keyboard, mouse, printer, monitor, etc|requires special

controller circuitry for transferring data from the processor and memory to the device,

and vice versa. A device controller is designed either as a chip which is placed in the

motherboard or as a printed circuit board which is plugged into an expansion slot of the

motherboard. The peripheral devices are connected to their respective controllers in the

system unit using special cables to sockets called expansion ports. The ports are located

on the backside of the system unit and provide connections through holes in the back of

the system unit. Parallel ports transfer several bits simultaneously and are commonly used

to connect printers to the computer. Serial ports transfer a single bit at a time, and are

commonly used to connect mice and communication equipment to the computer. Device

controllers are very complex. Each logical command from the processor must typically be

decomposed into long sequences of low-level commands to trigger the actions to be performed

by the device and to supervise the progress of the operation by testing the device's

status. For instance, to read a word from a disk, the disk controller generates a sequence of

commands to move the read/write arm of the disk to the correct track, await the rotational

delay until the correct sector passes under the read/write arm, transfer the word, and check

for a number of possible error conditions. A sound card contains circuitry to convert digital

signals from the computer to sounds that play through speakers or headphones that are

connected to the expansion ports of the card. A modem card connects the computer to the

telephone system so as to transport data from one computer to another over phone lines.

A network card, on the other hand, provides the circuitry to connect a computer to other

# computers on a local area network.

# Central processing unit

## Description and purpose of computer processors

In fact, what we today call a computer processor (or a central processing unit - CPU), properly called a microprocessor.

The first processor (Intel 4004) appeared in 1971.

A computer processor is a silicon wafer with millions and billions (today) transistors and channels for the transmission of signals.

Purpose of a CPU is the automatic execution of the program. In other words, it is a major component of any computer.

## Structure of a CPU

The major components of a CPU are an **arithmetic logic unit** (ALU), **registers** and **control unit** (CU). ALU performs basic arithmetic and logic operations. All calculations are made in the binary system. Coherence working of parts of a processor and its relation to other (external to it) devices depends on a control unit. A current command, initial, intermediate and final data (result of the calculation ALU) are temporarily stored in registers. Amount of bits of all registers is the same.

A **processor cache** stores frequently used data and instructions. Contacting the cache is much faster than RAM, so what it is bigger, the better.

## Work of a processor

A processor controlled by a program located in RAM.

Among other things, the control unit is responsible for calling another command and the determination of its type.

After receiving data and command, arithmetic logic unit executes the operation and writes the result to one of the available registers.

A current command is placed in specially designated for her a **command register**. When a current command is processed, the value of the so-called command counter increases. Now it points to the next command (unless, of course, there was no jump command or stop).

A command is often presented as a structure consisting of recording operation (need to perform) and cell addresses of inputs and outputs data. As specified in the command address, data is taken and placed in the usual registers (not in a command register), the obtained result is also first placed in a register, and then moved on to their address specified in the command.

## Processor properties

Processor clock speed today is measured in gigahertz (GHz), previously measured in megahertz (MHz). 1MHz = 1 million cycles per second.

A processor "communicate" with other devices (RAM) via the data, the address and control bus. Amount of bits of a bus is always a multiple of 8, changeable in the historical development of computer technology, and is different for different models, and are not the same for the data bus and address bus.

Bits data bus indicates how much information (number of bytes) can be transferred at a time (per cycle). From the address bus width depends on the maximum amount of memory to which the processor can work.

On power (performance) affect not only the processor clock speed and bit data bus, also important is the amount of cache memory.

# Random-access memory

## Functions of random-access memory (RAM)

* Storage of data and commands for further transfer to a processor (CPU) for processing. Information may come from RAM not immediately to handling processor, and in a more rapid than the RAM, the cache memory of processor.
* Storing the results of calculations performed by a processor.
* Reading (and writing) of contents of cells.

## Features work of RAM

RAM can store data only when the computer is on. Therefore, when it is turned off, the processed data should be stored on a hard disk or other storage medium. When running programs, an information is supplied to the RAM, for example, from a computer hard drive. While we are working with the program, it is present in memory (usually).

Once the work is finished with it, the data are overwritten on a hard disk. In other words, the flows of information in memory are very dynamic.

RAM is a memory with **random access**. It means that data can be read / written from any cell of the RAM at any time. For comparison, for example, a magnetic tape is a memory device with serial access.

## The structure of random-access memory

Random-access memory consists of cells, each of which has its own address. All cells contain the same number of bits. Nearby cells have consecutive addresses. As well as the data, memory address expressed in binary numbers.

Usually one cell contains one byte of information (8 bits), and it is the minimum unit of information, which can be referenced to. However, many commands work with so-called words. The word is a memory area consisting of 4 or 8 bytes (or other options).

## Types of RAM

Accepted provide two types of memory: static (SRAM) and dynamic (DRAM). SRAM is used as the processor cache, and DRAM - just as the computer's RAM.

**SRAM** is composed of flip-flops. Triggers can be only in two states: "on" or "off" (storage one bit). The trigger does not store charge, so switching between states is very fast. However, triggers require more sophisticated production technology. It is reflected in the price of the device. Secondly, consisting of transistors and the connections between them, a trigger takes a lot of space (at the micro level), so SRAM is a relatively large device.

**DRAM** does not contain triggers, and the bit is stored with using one transistor and one capacitor. It is cheaper and more compact. However, capacitors store charge, and the charge-discharge process is longer than switching of trigger. As a result, DRAM is slower. Another minus is a spontaneous discharge of capacitors. The charge has to be regenerated at regular intervals, on what to spend extra time.

## View of memory module

Outside, RAM PC is a module of the chips (8 or 16 pieces) on a board. The module is inserted into a special socket on the motherboard.

By design, memory modules for personal computers are divided into **SIMM** (one-sided arrangement of pins) and **DIMM** (two-sided arrangement of pins). DIMM has a higher data rate than SIMM. Currently, mainly produced DIMM-modules.

The main characteristics of RAM are data capacity and speed. Today memory capacity is expressed in gigabytes.

# Magnetic Disks

**Magnetic disks** are used for long-term storage of information (it will not be erased when you turn off the computer). But when using the disc, the data can be removed from it, and other be recorded.

There are hard disks and floppy disks. Currently, however, floppy disks hardly used. Floppy disks were popular in the 80-90-ies of the last century.

**Floppy disks** are magnetic disks in square plastic cassette size of 5.25 inches (133 mm) or 3.5 inch (89 mm). Floppy disks allow transfer of documents and programs from one computer to another, store information, make backup copies of the information contained on the hard drive.

On a magnetic disk, information is recorded and read with magnetic heads along concentric tracks. When writing or reading data, a magnetic disk rotates around its axis, and its head is brought to a desired track with a special mechanism.

3.5-inch floppy disks have a capacity of 1.44 MB.

Unlike floppy disks, **hard drive** allows to store large amounts of information. The capacity of modern hard disks can be terabytes.

The first hard drive was created by IBM in 1973. It allowed to store up to 16 MB of information. Since the drive has 30 cylinders, divided into 30 sectors, it is designated as 30/30. By analogy with automatic rifles with a caliber 30/30, the disc was nicknamed the "Winchester".

A hard disk is a sealed iron box, inside it there are one or more magnetic disks with a block of heads of read / write and an electric motor. When you turn on the computer, the electric motor spins the magnetic disk at a high speed (several thousand revolutions per minute) and the disk continues to rotate as long as the computer is turned on. Special magnetic heads "hover" over the disc. They write and read information. Heads hover above the disk due to its high speed. If the head touches the disc, the disc would be quickly out of action due to friction.

The following definitions are used in describing a magnetic disk.

A **track** is concentric circles on a magnetic disk, which is the basis for recording an information.

A **cylinder** is a set of magnetic tracks, one above the other in all of working surfaces of the disks of the hard drive.

A **sector** is portion of the magnetic track, it is one of the basic units of data storage. Each sector has its own number.

A **cluster** is the minimum element of the magnetic disk, which operates on the operating system. Each cluster consists of several sectors.

# The logical structure of hard disk drive

Each disk has a logical structure that includes the following elements:

* boot sector;
* file allocation table;
* data area.

The boot sector (Boot Record) takes a sector number 0. It contains a small program IPL2 (Initial Program Loader 2), by which the computer determines whether to load the operating system from the hard disk.

In addition to the boot sector, a hard drive has another area - the main boot sector (Master Boot Record). The fact that a single hard disk may be divided into multiple logical disks. For the main boot sector on the hard drive is always allocated physical sector 1. This sector contains a program IPL1 (Initial Program Loader 1), which defines the boot drive when executed.

The file allocation table is used to store information about the placement of files on the disk drive. Magnetic disk typically uses two copies of the tables which follow one another, and their content is identical. It is done in case, if the disk have any failures, it can always be "repaired" using a second copy of the table. If both copies will be broken, then all the information on the disk will be lost.

The data area occupies most of the space and serves directly for data storage.

# Hard drive partitions

Usually, the hard disk drive is divided into several partitions. This is useful for storing files, and is a prerequisite if you install multiple operating systems on one physical computer's hard disk.

So, a partition of a hard disk is its part used for specific tasks: file system of any type, swap space, etc. Changing contents and a file system of one partition does not affect the other.

In Linux, partitions generally called so: hda1, hda2, hda3, etc. - For the first (or only) a physical hard disk.

If a computer has multiple hard disks, the partitions of the second disk will be referred to as: hdb1, hdb2, hdb3 etc. The third - hdc1, hdc2, hdc3 etc.

The main sections (primary partitions) on each hard drive can be a total of four. Accordingly, from hd\_1 to hd\_4. (The underscore represents a letter of a physical hard disk).

However, it happens that four partitions is not enough. Therefore, one of the main partition is announced as an **extended partition** and is divided into subsections, starting with hd\_5.

Example:

In this example, the computer has one hard drive, which has four primary partitions (although they may be less) - hda1, hda2, hda3, hda4. The last partition is expanded and divided into five parts (hda5, hda6, hda7, hda8, hda9), the total size of which is equal to about 60 gigabytes.

The second column of the table (Filesystem) shows types of file system of partitions. For partitions in Windows operating systems are used the NTFS file system or Fat32. For partitions in Linux - ext4, ext3, ext2 or others.

In this case, hda5 partition is used as a swap partition (linux-swap) in Linux. But partition hda2 contains a bad sector, it can be understood by tag "hidden".

# PC Peripherals

Modern personal computers usually have a variety of peripheral devices.

**Peripherals** are any additional and supplementary devices that connect to a PC to extend its functionality.

Consider some of the peripheral devices.

A **printer** is a device for printing text and graphic information. As a rule, printers work with paper A4 or A3. The most common today laser and inkjet printers, dot matrix printers are already out of items.

In matrix printers, the print head consisted of a number of thin metal needles that when moving along a line at the right moment struck through the ink ribbon, thereby ensuring the formation of characters and images. Dot matrix printers had low speed and print quality.

In inkjet printers, ink is ejected from the molding apertures (nozzles) in the printhead and then sticks to the paper. Imaging takes place as if from separate points – "blots". Inkjet printers are characterized by the high cost of consumables.

In laser printers, running on a drum, the laser beam electrifies it, and an electrified drum attracts the particles of dry ink, and then the image is transferred from the drum to the paper. Then, the sheet of paper passes through the heat drum and the paint under the action of heat is fixed on the paper. Laser printers have a high print speed and quality.

A  **plotter** is a device for printing on the paper of large images, line drawings and other graphic information. A plotter can print graphic information on paper A2 and more. Structurally, it can use a drum roll paper or a horizontal plate.

A  **scanner** is a device that allows to enter graphic information into computer. When driving through a picture (sheet of text, photography, drawing), a scanner converts the image into a numeric format and displays it on a screen.

**Topic № 6.**

**A boolean algebra and the computer logic**

**What is a boolean algebra?**

**A boolean algebra** is the branch of mathematics that arose in the XIX century thanks to the efforts of the English mathematician George Boole. Initially, boolean algebra had no practical value. However, already in the XX century, its provisions have been applied in the description of the functioning and development of various electronic circuits. The laws of Boolean algebra have been used in the design of various parts of the computer (memory, CPU). Although it is not only the scope of application of this science.

What is a boolean algebra? First, it examines the methods of establishing the truth or falsity of complex logic statements using algebraic methods. Second, Boolean algebra does so in a way that complex logical statements are being described with a function whose result can be either true or false (1 or 0). The arguments of the function (simple expression) can also have only two values: 0 or 1.

What is a simple logical statement? It is phrases like "two more than one", "5.8 is an integer". In the first case we have the truth, and the second - the lie. A boolean algebra does not concern the essence of these statements. If someone decides that the statement "Earth is square" is true, then a boolean algebra will accept this as a fact. The fact that a Boolean algebra deals with the calculated result of complex logic statements based on pre-known values of simple statements.

**Logic operations. Disjunction, conjunction and negation**

So how simple logical statements are interconnected to form complex expressions? In natural language we use various unions and other parts of speech. For example, "and," "or", "either", "not", "if", "then". Examples of complex sentences: "He has the knowledge and skills", "she will arrive on Tuesday or Wednesday," "I'll play when the lessons do", "5 is not equal to 6". How do we decide what we are told the truth or not? Somehow logically, even somewhere unconsciously, based on previous experience of life, we understand that the truth in the union "and" occurs if the two simple statements are true. If at least one statement will be false, then the complex statement will be false. But when the union "or" just one simple statement may be true, then the entire expression becomes true.

A boolean algebra has shifted the experience of life in the language of mathematics, it is formalized, introduced strict rules for obtaining unambiguous results. Here, the unions have been called logical operators.

Boolean algebra provides a lot of logical operations. However, three of them deserve special attention, because they can be used to describe other, and therefore, use less diverse devices when constructing circuits. These operations are the **conjunction** (AND), **disjunction** (OR) and the **negation** (NOT). Often conjunction designate the sign &, disjunction - ||, and negation - a bar over a variable indicating a statement.

At the conjunction, true of complex expression occurs only in the case of truth of all simple expressions. In all other cases, a complex expression is false.

When the disjunction, truth of complex expression comes when one simple statement is true, or both at once. It happens that the complex expression consists of more than two simple. In the case of "or" enough if one simple is true, and then all expression will be true.

The negation is a unary operation, because it is performed on a simple expression or over the result of a complex. As a result of the negation, the new statement is obtained, which is the opposite of the original.

**Truth tables**

Logical operations is convenient to describe with the so-called truth tables, which reflect the results of the calculations of complex sentences for different values of the original simple statements. Simple statements are indicated with the variables (for example, A and B).

**Logic basics of the computer**

The computer uses a variety of devices, their work is perfectly described with a boolean algebra. These devices include groups of switches, triggers, adders.

In addition, the relationship between Boolean algebra and computers lies in the numeral system. As you know, computers are using the binary system. Therefore, the computer device can store and convert both the numbers and value of logical variables.

**Switching circuits**

Computers use electric schemes consisting of a plurality of switches. A switch can be in only two states: opened and closed. In the first case - the current passes; in the second — no. Very convenient operation of such schemes is being described using Boolean algebra. Depending on the position of the switches, the signals at the outputs may be getting or not getting.

**Valves, triggers and adders**

The valve is a logical element which receives the one binary values and other outputs depending on its implementation. For example, there are valves that realize the logical multiplication (conjunction), addition (disjunction) and negation.

Triggers and adders is relatively complex devices composed of simpler elements — valves.

A trigger can store one bit, because can exist in two stable states. Generally, triggers are used in registers of the processor.

Adders are widely used in the arithmetic-logic unit (ALU) of the processor and perform the summation of binary digits.

# Boolean Algebra Axioms

For boolean values are commonly used three operations:

* **Conjunction** — logical multiplication (AND) - and, &, ∧.
* **Disjunction** - logical addition (OR) - or, |, v.
* Logical **negation** (NOT) - not, ¬.

Logical expressions can be converted in accordance with the axioms of Boolean algebra:

1. reflexivity  
   a ∨ a = a  
   a ∧ a = a
2. commutativity  
   a ∨ b = b ∨ a  
   a ∧ b = b ∧ a
3. associativity  
   (a ∧ b) ∧ c = a ∧ (b ∧ c)  
   (a ∨ b) ∨ c = a ∨ (b ∨ c)
4. distributivity  
   a ∧ (b ∨ c) = (a ∧ b) ∨ (a ∧ c)  
   a ∨ (b ∧ c) = (a ∨ b) ∧ (a ∨ c)
5. negation of the negation  
   ¬ (¬ a) = a
6. De Morgan laws  
   ¬ (a ∧ b) = ¬ a ∨ ¬ b  
   ¬ (a ∨ b) = ¬ a ∧ ¬ b
7. absorption  
   a ∨ (a ∧ b) = a  
   a ∧ (a ∨ b) = a

# Logic gates

At the base of the construction of computer hardware are the so-called gates. They are simple elements which can be combined with each other, creating a different schemes. Some schemes are suitable for arithmetic operations, and other schemes are used for different computer memory.

A **logic gate** is a device that receives input signals (or only one signal) and depending on which provides a single output signal.

The simplest valve is a transistor inverter which converts the low voltage to high or vice versa (high to low). It can be represented as a transformation of a logic zero to logic unit or vice versa. This is valve NOT.

By connecting a pair of transistors in a different ways, gates OR-NOT and AND-NOT are obtained. These valves are taking two or more input signals. The output signal is always one, and depends on (outputs a high or low voltage) from the input signals. In the case of NOR (OR-NOT) gate, high voltage (logical unit) may be obtained only if a low voltage on all inputs. In the case of NAND (AND-NOT) gate is the opposite: a logic zero is obtained if all input signals are units. As you can see, it's oppositely to the usual logical operations such as the AND and OR. However, commonly used valves AND-NOT and OR-NOT because their implementation easier. They are implemented by two transistors, while logical AND and OR by three.

The output signal of a gate can be expressed as a function of input signals.

Transistors require very little time to switch from one state to another (switching time measured in nanoseconds). And this is one of the significant advantages of schemes constructed on their basis.

# Adder and Half Adder

An arithmetic logic unit (ALU) of the CPU necessarily contains in its composition elements such as adders. These schemes allow to add binary numbers.

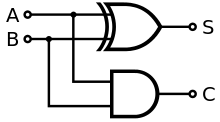
How does the addition of? For example, the binary numbers 1001 and 0011 is required to add up. First, we add up the lower bits (the latest digits): 1 + 1 = 10. I.e. the bit is 0, and the 1 is transferred to the next discharge. Next: 0 + 1 + 1(on the transfer) = 10, i.e. 0 is written again in the current discharge, and the unit will go to the next. The third step: 0 + 0 + 1(from transfer) = 1. As a result, the amount is equal to 1100.

## Half adder

Now we will not pay attention to the transfer from the previous discharge and look at just how a sum of the current discharge is formed. If the two units or two zeros have been given, the amount of the current discharge is equal to 0. If one of the two summands is equal to one, then the sum is equal to one. So results can be obtained using the "Exclusive OR" gate.

The transfer of the 1 to the next discharge occurs if the two summands are equal to one. This is accomplished with valve "AND".

Then the addition within the same discharge (excluding possible transfer 1 from lower) can implement with the scheme shown below, which is called the **half-adder**. The half-adder has two inputs (for summands) and two outputs (for the amount and the transfer). The scheme depicts a half-adder, consisting of valves "exclusive OR" and "AND".



## An adder

In contrast to the half-adder, the adder takes into account the transfer of the previous discharge, so it has not two, but three inputs.

To allow for the transfer, it is necessary to complicate the scheme. In fact, it turns out consisting of two half-adder.

Consider one of the cases. We need to find the amount of 0 and 1, and 1 of the transfer. First, we determine the sum of current discharge. Signals A (0) and B (1) are entering into "Exclusive OR". The output is 1. In the following "exclusive or" already two units are entering. Consequently, the amount will be equal to 0. Now let's see what happens at the transfer. One of the valves "and" gets 0 and 1 (A and B). We are getting 0. In the second valve "AND" the two units come, it gives 1. The passing through the gate "OR" zero from the first "AND" and the units from the second "AND" gives us 1.

Check the operation of the scheme with simple addition: 0 + 1 + 1 = 10. I.e. 0 is stays in the current discharge, and 1 comes in the next. Consequently, the logic scheme works correctly.

# Memory elements, SR flip-flop

Memory is an important part of the computer. This is a device for storing data and instructions. If a computing device has no memory, it is not a computer.

The bit is the elementary unit of computer memory. Therefore, a device must be able to be in two states, i.e. store one or zero. Also, this device must be able to quickly switch from one state to another under external influence, which makes it possible to change the information. Finally, the device must be able to determine its condition, i.e. to provide information about their condition.

A flip-flop is a device that can receive, store and deliver information.

A variety of flip-flops is very large. The simplest of these is the so-called SR flip-flop (or SR latch), which is assembled from two gates. Typically valves "NOR" (OR-NOT) or "NAND" (AND-NOT) is used.

## A SR latch with "NOR" gates

A SR latch "remembers" which its input received a signal "1" the last time. If the signal has been submitted to the S-input, on the output the flip-flop is constantly "informs" that stores unit. If a signal corresponding to one, served on the R-input, the flip-flop output is 0. Despite the fact that the SR flip-flop has two outputs, it means output Q. Underlined Q always has the opposite meaning of Q.

In other words, the input S (set) sets the flip-flop into 1; input R (reset) sets into 0. Installation is made with a high voltage signal (corresponding to one). Just all depends on which the input is applied the signal.

Most of the time the zero signal (low voltage) is applied to the inputs. In this case, the trigger retains its previous state.

The following situations are possible:

* Q = 1, the signal is supplied to S, thus, Q does not change.
* Q = 0, the signal is supplied to S, thus, Q = 1.
* Q = 1, the signal is supplied to R, thus, Q = 0.
* Q = 0, the signal is supplied to R, thus, Q does not change.

The situation in which "1" signals are supplied to both inputs, is unacceptable.

How does the trigger store its state? For example, the flip-flop outputs a logic 0 at the output Q. Then, according to the scheme, this 0 is returned to the S-gate, where is inverted (1 is obtained), and in this form is transmitted to the R-gate. The R-gate again inverts the signal (0 returns), which then goes to the output Q.

Now, suppose, was given "1" signal on the S-input. Now, in the S-gate, signals are received: 1 from S and 0 from Q. Since the gate is NOR type, the output turns out 0. Zero goes on the R-gate, there it is inverted (turns 1). The signal at the output Q becomes equal to 1.

# The Practical Meaning of a Boolean Algebra

The binary half-adder performs binary addition of two one-bit binary numbers (i.e. performs the rules of binary arithmetic):

0 + 0 = 0; 0 + 1 = 1; 1 + 0 = 1; 1 + 1 = 0.

The half-adder has the carry bit. However, the half-adder circuit does not contain a third input to which a carry signal can be fed from the previous discharge of amount binary numbers. Therefore, a half-adder is used only in the least significant bit of a logic circuit for summing multi-bit binary numbers, since there is no carry signal from the previous bits. A full binary adder adds two multi-bit binary numbers, taking into account the transfer of signals from the addition of previous bit.

Combining binary adders in cascade, you can get a logic scheme of an adder for binary numbers with any number of digits. With some modifications, these logic circuits are used for subtraction, multiplication and division of binary numbers. Arithmetic devices of modern computers are built on them.

Adders and half-adders are a single-tact logic circuit. The values of their outputs is uniquely determined by the values of their inputs. They have not the time factor. Along with them there are a multicycle logic circuits in which the value of their outputs are determined not only with the values of their inputs, but also with their status in the previous cycle. Such logic circuits are memory schemes (flip-flops). They are built with the help of feedback from output to input.

For memorizing the input signal, a closed chain from the output to the input of flip-flops is formed with the feedback. This circuit is maintained indefinitely after removal of the input signal, until the appearance of the reset signal.

This memory circuit has another name yet: a latch with separate inputs. In this scheme, there are an memorization input (S) and reset input (R). Also, in computer technique, the flip-flop with calculating input is widely used. It has only one input and one output. This circuit performs division by 2. The status of its output is changed only after the consecutive two input pulses. Combining the flip-flop with calculating input in a serial cascade, it is possible to carry out the division into 2, 4, 8, 16, 32, 64, etc.

A memory scheme plays an important role in the construction of machine control systems of high risk, such as for example the industrial press. To protect the operator's hands, these machines are built with a two-handed control systems. Such systems force an operator to keep both hands on control buttons during each cycle of work the machine. It eliminates the ingress of hands in the danger zone, where a detail is pressed.

Input and output signals of electromagnetic relays take only two values, like statements in a Boolean algebra. When the coil is de-energized, the input signal is zero, and if there is current in the winding, the input signal is equal to one. When the relay is open, the output signal is zero, and if the contact is closed, the output signal is equal to one.

**Bitwise operations**

Logical operations on bits of integers are allowed in many programming languages. Conventional logic operations result in a logical data type. However, bit logical operations change integer, according to certain rules. More precisely bitwise operations alter individual bits of the binary representation of the number, resulting in a change of its decimal value.

For example, in programming language Pascal ordinary logical operations and logic operations on the bits are denoted by the same keywords: not, and, or, xor. The compiler defines what is meant depending on the context of use of these words. Conventional logic operations combine two or more simple logical expressions. For example, (a > 0) and (c != b), (c < a) or (not b), etc. In turn, bitwise operations are performed only on integers (or variables that contain them). For example, a and b, a or 8, not 247.

**How to understand bitwise operations**

1. We translate arbitrary pair of integers in the binary representation.

6710 = 0100 00112

11410 = 0111 00102

2. Now arrange the bits of the second number under the corresponding bits of the first and perform the usual logical operations to the figures, standing in the same discharges first and second numbers. For example, if in the latter (younger) discharge one number is 1, and the other numbers is 0, the logic operation "and" returns 0, and "or" returns 1. We apply the operation "not" only to the first integer.

3. We translate the result in decimal system.

01000010 = 26 + 21 = 64 + 2 = 66

01110011 = 26 + 25 + 24 + 21 + 20 = 64 + 32 + 16 + 2 + 1 = 115

00110001 = 25 + 24 + 20 = 32 + 16 + 1 = 49

10111100 = 27 + 25 + 24 + 23 + 22 = 128 + 32 + 16 + 8 + 4 = 188

4. So, as a result of bitwise operations we obtain the following:

67 and 114 = 66

67 or 114 = 115

67 xor 114 = 49

not 67 = 188

Here's another example of the logical operations on bits. Check it out yourself.

5 and 6 = 4

5 or 6 = 7

5 xor 6 = 3

not 5 = 250

**Why are bitwise operations used?**

Looking at the result of bitwise operations not immediately possible to capture regularity in their results. Therefore, it is not clear why we need such operations. However, they find their application. The bytes are not always store numbers. A byte or memory cell can store a set of flags (set - reset), representing the status of something. With bitwise operations can be checked, which bits in the byte are set to one, the bits can be reset or, conversely, set to one. It is also possible to change the values of the bits reversed.

**Check of bits**

Checking bits is performed using bitwise operations "and". Imagine that there is a byte of memory contents unknown to us. It is known that the logic operation "and" returns 1 if both operands contain 1. If to an unknown number, the bitwise logical multiplication is applied (operation "and") by the number 255 (which is the binary representation is 11111111), as a result we get the unknown number. Those units of the binary representation of 255 will become zeros, which will be multiplied by the bits of the unknown, containing 0. For example, suppose there is an unknown number of 38 (00100110), then checking bits would look like this:

In other words, x and 255 = x.

**Setting bits to zero**

To set to zero any bit of a number, it is necessary to logically multiply its by 0.

Note the following:

1111 1110 = 254 = 255 - 1 = 255 - 20

1111 1101 = 253 = 255 - 2 = 255 - 21

1111 1011 = 251 = 255 - 4 = 255 - 22

1111 0111 = 247 = 255 - 8 = 255 - 23

1110 1111 = 239 = 255 - 16 = 255 - 24

1101 1111 = 223 = 255 - 32 = 255 - 25

1011 1111 = 191 = 255 - 64 = 255 - 26

0111 1111 = 127 = 255 - 128 = 255 - 27

For example, to reset the fourth bit from the end of the number, it is necessary to logically multiplied by 247, or (255 - 23).

**Setting bits to the 1**

To set the bits in the unit, the bitwise operation "or" is used. If we add up logically the binary representation of the x and 00000000, we get the x number itself. But if we write the 1 in any bit of the second summand, the result will be the 1 in this bit.

We also note that:

0000 0001 = 20 = 1

0000 0010 = 21 = 2

0000 0100 = 22 = 4

0000 1000 = 23 = 8

0001 0000 = 24 = 16

0010 0000 = 25 = 32

0100 0000 = 26 = 64

1000 0000 = 27 = 128

Therefore, for example, to set the second-highest bit of the x number in the 1, it should be logical add up with 64 (x or 64).

**Changing bit values**

The bit operations "xor" is used to change the values of the bits reversed. To invert any bit of the number x, the one is being recorded in the same discharge of the second number. If you want to invert all the bits of x, then use the bitwise XOR with the number 255 (1111 1111).

**Operations of a logical shift**

In addition bitwise logical operations in many programming languages provide bit cyclic shift operation to the left or right. For example, in the Pascal programming language, these operations are denoted by shl (left shift) and shr (right shift). A first operand is an integer being operated. The second operand indicates how much positions bits of the first number are shifted to the left or right. For example, 105 shl 3 или 105 shr 4. The number 105 in binary representation is 0110 1001.

The upper bits of the original number is lost when a shift to the left, the lower bits are shifting in their place. Freed significant bits are being filled with zeros.

The lower bits of the original number is lost when the shift to the right, the upper bits are shifting in their place. Freed significant bits are filled with zeros if the original number was positive.

**TOPIC 7**

**Types of Software**

A desktop computer typically comes with pre-installed software. This software can be

categorized into two categories |application software and systems software.

Application Software: Application programs are designed to satisfy end-user needs by

operating on input data to perform a given job, for example, to prepare a report, update

a master payroll \_le, or print customer bills. Application software may be packaged or

custom. Packaged software includes programs pre-written by professional programmers,

and are typically o\_ered for sale in a oppy disk or CD-ROM. Custom software includes

programs written for a highly specialized task.

Systems Software: Systems software enables the application software to interact with

the computer, and helps the computer manage its internal and external resources. Systems

software is required to run applications software; however, the converse is not true. Systems

software can be classi\_ed into three types|utility programs, language translators, and the

operating system. Utility programs are generally used to support, enhance, or expand the

development of application programs. Examples consist of editors and programs for merging

files. A language translator or compiler is a software program that translates a program

written in a high-level language such as C into machine language, which the hardware

can directly execute. Thus a compiler provides the end user with the capability to write

programs in a high-level language.

Operating System: The operating system is a major component of the systems software.

Desktop operating systems allocate and control the use of all hardware resources: the

processor, the main memory, and the peripheral devices. They also add a variety of new

features, above and beyond what the hardware provides. Running the shell provides the end

user with a more \capable" machine, in that the computer system provides direct capability

to specify commands by typing them on a keyboard. The GUI (graphical user interface)

goes one step further by providing the user with a graphical view of the desktop, and letting

the user enter commands by clicking on icons. The multitasking feature of the OS provides

the user with the capability to run multiple tasks \concurrently". The \_le system of the OS

provides the user with a structured way of storing and accessing \permanent" information.

The operating system is thus an important part of most computer systems because it exerts

a major inuence on the overall function and performance of the entire computer. Normally,

the OS is implemented in software, but there is no theoretical reason why it could not be

implemented in hardware!

DeviceDriver (Software Driver): Most application programs need to access input/output

devices and storage devices such as disks, terminals, and printers. Allowing these programs

to perform the low-level IO activity required to directly control an input/output device

is not desirable for a variety of reasons. First, most application programmers would \_nd

it extremely di\_cult to do the intricate actions required to directly control an IO device.

Second, inappropriate accesses of the IO devices by amateur or malicious programmers can

wreck plenty of havoc. The standard solution adopted in computer systems is therefore to

provide a more abstract interface to the application programmer, and let an interface program

perform the required low-level IO activity. This interface program is called a device

driver or software driver. Each device requires speci\_c device driver software, because

each device has its own speci\_c commands whereas an application program uses generic

commands. The device driver receives generic commands from the application program

and converts them into the specialized commands for the device, and vice versa.

**Starting the Computer System**: The Boot Process

Now that you have a good understanding of the role of an operating system in a modern

computer, it would be interesting to learn how the operating system is activated each time

a computer is turned on. When a computer is turned o\_, the data in the registers and

memory are lost. Thus when the computer is turned on, the OS program is not residing in

the main memory, and needs to be brought into main memory from a storage device such as

a diskette or hard disk. In modern computers, this copying is done by executing a program

called the bootstrap program or boot program for short. How can the computer execute

this copy program if the memory contains no useful contents? To solve this dilemma, a

portion of the memory is implemented using non-volatile memory devices such as a readonly

memory (ROM). This memory contains the boot program. When the computer is

turned on, it starts executing instructions from the starting address of the boot program.

The boot program contains code to perform diagnostic tests of crucial system components

and load the operating system from a disk to the main memory. This bootstrap loader may

be comprehensive enough to copy the nucleus of the operating system into memory. Or it

may \_rst store a more comprehensive loader that, in turn, installs the nucleus in memory.

Once loaded, the OS remains in main memory until the computer is turned off.

For copying the OS from a disk drive to the RAM, the computer needs to know how

the disk has been formatted, i.e., the number of tracks and sectors and the size of each

sector. If information about the hard disk were stored in the ROM, then replacing the

hard disk becomes a di\_cult proposition, because the computer will not be able to access

the new hard disk with information about the old disk. Therefore, a computer must have

a semi-permanent medium for storing boot information, such as the number of hard disk

drive cylinders and sectors. For this purpose, it uses CMOS (complementary metal oxide

semiconductor) memory, which requires very little power to retain its contents and can

therefore be powered by battery. The battery helps the CMOS memory to retain vital

information about the computer system con\_guration, even when the computer is turned

o\_. When changing the computer system con\_guration, the information stored in the CMOS

memory must be updated, either by the user or by the plug-and-play feature.

Modern computers cannot work without software. In other words, the computer performs certain tasks if on it are installed programs containing algorithms to perform these tasks.

The rapid development of the software began in the 80s due the proliferation of personal computers (PC). Today, the variety of software is huge. Users and developers should choose the application or development environment from a list proposed by the computer industry. The choice depends on the quality of the software, its price, the prospects of development, providing technical support and others.

Currently, along with the notion of "program" is used the concept of "application". Between them there is no fundamental difference. It is estimated that a **program** is one unit, and an **application** is a set of programs that solve together one or more related tasks. However, this division can be quite arbitrary due to the fact that most simple programs typically include various libraries and third-party modules. On the other hand, to isolate a program from an application so that it runs independently, it may not be possible.

Another thing is the concept of a **software system**, which is a set of programs that solve whole group of tasks. These software systems can serve operating systems, office suites, integrated set of tools for maintenance or testing of a computer.

The development of personal computers has led to the fact that the computers are used primarily for application tasks, rather than numerical data processing tasks (as it was in the early days of the computers). However, the application software can not function on the bare hardware. Application software can work only in pre-installed operating system, through which the software can use the resources of the equipment. Operating systems and other system software play the role of an intermediary. In addition, software development tools are necessary. Therefore, all of the **software can be divided into three types**:

* system software
* development tools
* applications

**System software** is not only operating systems. It's also a variety of utility programs for the diagnosis of computer's resources (such as memory tests), providing to users with a convenient way to interact with the computer work (for example, the command line), as well as maintenance of computer resources (such as disk layout).

Among other functions, an operating system provides a user interface, whose main task is the formation of a convenient environment for working. The user interface can be a graphic, and can be a text interface. An interface is a set of techniques for interaction of two or more entities. The interface can be between users and programs, between programs and between software and hardware.

**Programming tools** are a set of programming languages, tools to automate the process of creating programs, compilers and interpreters.

Programming languages and systems according to the destination are the tools to create a truly useful software. With their help, a programmer creates application and system software and new development tools.

A huge share of the software take **applications** that are divided into universal and specialized. However, this division is conditional.

**Topic № 8.**

# Operating System Concepts

## What is an operating system?

Basically all the various software is divided into system programs and applications. The first group provides operation of the second group on the existing hardware (CPU, disk, memory, input-output devices). **Operating systems (OS) are system software.** One of the tasks of the operating system is to implement algorithms of work with hardware. One may ask: why is it necessary? Because each application could include code to appeal to the hardware. However, it would have fanned the software to large sizes. Furthermore, in applications it would be much the same code, responsible for implementing low-level commands (requests to hardware). Another question: how to solve the problem of co-operation of different programs on the same computer? Therefore, operating systems and other system software are the intermediaries between the application software and computer hardware.

Historically, operating systems are originated as a set of programs and libraries for managing the input and output operations. Programmers used these universal programs. They did not had to wrestle with how to program data read from the floppy or text output to the printer. It was enough to call a function of the connected library, and it was doing all the work (it already had have the code work with physical devices).

Later operating systems became more complex, they had acquired new functions. Computers were becoming more powerful, and it was need to run many programs at once. OS has become to solve a problem of efficient allocation of resources of hardware between running programs. Several users began to work on one computer at a time. OS began to monitor the rights of each and to protect the data. As a result, modern operating systems include a variety of different functions.

In structure an operating system is a set of programs and modules. There is a concept of an **operating system kernel**. The software kernel is protected from interference by users and programmers. Applications refer to the core by means of requests to perform a particular action with hardware. These requests are known as system calls and are special commands.

## Purpose of an Operating System

Thus, an operating system performs two main tasks:

1. **It facilitates (enables) to use the hardware for users and programmers.** For example, an operating system makes it possible to ignore the fact as in reality the data is being processed on the hard disk, but to work with the file concept.
2. **OS provides an efficient use of hardware.** On modern computers multiple programs run simultaneously, so an operating system is responsible for the allocation of memory, CPU registers and the other between running programs at any given time. OS determines the optimum allocation of these resources in time (CPU usage by queue of programs) and space (loading in different parts of the memory of different programs).

## Types of operating systems

### Windows operating systems

Today, the most popular are the Windows operating systems that are proprietary (commercial) product from Microsoft. Windows start their "pedigree" from the DOS operating system and originally were a shell (Windows were being launched from under DOS), increasing the possibility DOS and facilitating to an inexperienced user to work with a computer. Later versions (starting with Windows NT) were complete operating systems.

Windows advantage is considered a user-friendly interface. Among the shortcomings are noted the unreliability of the system.

### Unix-like operating systems

The UNIX operating system has had a great influence on the development of operating systems, laying the foundation of work of modern operating systems. Historically, UNIX was a system for software development. Basically, only programmers worked in UNIX (and indeed in the 70s, only they were working with computers).

UNIX was developed on several fundamental ideas. For example, one small problem must be solved one small program, but a complex task must be solved combination of simple programs.

In UNIX, great attention is paid to the distribution of computer resources among users. This system is a multiterminal (each user is working with a computer via its terminal).

Despite the fact that the Unix-like systems are less popular than Windows, they work on many types of computers.

#### Linux

Linux is a set of Unix-like operating systems (distributions), which are most often freely distributed. One of the unique features of the GNU / Linux systems is the lack of a single geographic development center. Linux and its programs are being written by millions of programmers scattered around the world.

#### MAC OS

This operating system also was created on the basis of the core of UNIX.

MAC OS is Apple's product for her own Macintosh computers.

It is considered a reliable and user-friendly. But unlike Windows is not so popular; since Apple computers are more expensive.

# Types of User Interface

According to the type of user interface there are the text (linear), graphic and voice operating system.

The **user interface** is a set of methods of user interaction with the application. The user interface includes a user interaction with the application and the method of communication.

## Text-based user interface

Text-based operating systems implement a command line interface. The main control device is a keyboard. A command is typed on the keyboard and displayed on the screen. The end of the input of a command is pressing the Enter key. To work with operating systems that have a text interface, you need to know command language of the OS.

The first operating systems have a text interface. Currently, it is often used on the servers.

## Graphical user interface of OS

These operating systems implement an interface based on the interaction with graphical controls. In this case, control devices are a keyboard and mouse. The active element of control is a mouse pointer. Movement of the pointer on the screen is synced with the movement of the mouse. Passive controls are a graphical elements of control of applications (on-screen buttons, icons, radio buttons, check boxes, drop-down lists, menu bar, etc.).

A start screen of graphical operating systems is a system object called "Desktop". It displays objects (files and directories) and control elements.

Graphical operating systems allow to perform most operations in many different ways, for example through the menu, the toolbar, through the windows system and other. Since the operations are performed on the object, it must be pre-selected (highlighted).

The basis of graphical user interface is an organized system of windows and other graphical objects that developers seek to maximize standardization of all the elements and methods of work.

Window is a framed by a rectangular area on the screen that displays the application, document, message. A window is active when with him at the moment the user is working. Performing in the graphical operating system, transactions take place either on the desktop or in any window.

## Speech-based user interface

In the case of SILK-interface (Speech, Image, Language, Knowledge) by voice command on the screen there is movement from one search image to other.

It is assumed that by using a public interface is not necessary to understand the menu. Screen images clearly indicate the subsequent path of moving from one search image to another by semantic relations.

# Text Editor and Word Processor

**Text editors** are programs designed for working with text. They allow to read, write, edit the text, save it to a file, perform a search on the text, and more.

The main difference between a **word processor** and a text editor: word processors allow to format text and paste non-text objects (images, tables, charts, video and audio) into an editable document.

Initially, somewhere in the 70s of XX century, special electronic devices, designed to work with the text, were called word processors. Later, when computers became more versatile machines, have started to name programs-applications.

## Features of text editors

Text editors are designed to work with so-called "plain text". Such text does not include the elements of formatting and non-text objects. According to other features of work with text, editors almost not inferior to word processors, and superior to them in something. For example, many text editors can highlight text syntax representing a code in a programming language. Moreover, code highlighting works for a variety of programming languages.

A simple text editor is a single program. More complex editors may include additional programs, libraries and utilities. In this case, a text editor is not a program, it is a application, as in the case of word processors.

The most well-known and simple text editor is the "Notepad", which comes with the Windows operating system. In GNU/Linux, more complex text editors are common. There are many them. As examples can be called more simple Gedit and more complex Geany.

## Features of word processors

In a sense, word processors can be regarded as a kind of text editors.

Word processors can format a text and paste into it a non-text objects (tables, images and other). Therefore, created in word processing, documents store not only text data but also the text markup commands in a special language, references to related objects or objects themselves in the form of binary code.

Every word processor saves the document in its format. Therefore, a document, created in one application, could not correctly displayed, if you open it in another application.

Word processors have a more complicated interface. In addition to the menu bar, there are always toolbars containing buttons with icons, drop-down lists and others. Clicking on this buttons call certain commands. All commands, possible in a word processor, are in the menu items. On the toolbar, usually the most popular commands are duplicated.

Visibility of toolbars are customizable. So by default in many word processors, are displayed only two panels: "Standard" and "Format".

Text editors can also have toolbars. However, there are not in all editors. While toolbars there are in all word processors.

At present, the most popular word processors are part of the application packages. Thus, MS Word is part of the MS Office package, LibreOffice Writer is included in the LibreOffice packet. Typically office application packages include a word processor and spreadsheet application, application for creating presentations, database management system, and others.

# Graphics software

Need of input graphs, charts, diagrams, drawings in a document made it necessary to create a graphics software. Graphics editors are programs and applications that allow you to create and modify graphical objects.

Vector graphics create images composed of different geometric shapes, which are described in a special language. Also vector graphics editors allow to choose the thickness and line color, shading, fonts, etc. Graphical editors for processing vector graphics are designed for the professional work related to the artistic and technical illustration, with the subsequent color printing. They occupy an intermediate position between the package for computer-aided design (CAD) and publishing systems.

There is also a group of programs that allow you to view images in slide show mode, apply special effects.

Business graphics provides display of information stored in a spreadsheet, database, and other structures in the form of a two- or three-dimensional graphs, pie charts, line graphs, and others. Software packages for working with business graphics are constructors of graphic images of business information, designed to a visual and dynamic way to present the results of some analytical researches.

Scientific graphics involves the creation of mathematical, chemical and other formulas, cartography, and others.

Cognitive computing facilities are a set of virtual devices, software and systems, implementing complex processing of visual information in the form of images, processes, structures. Cognitive graphics allows to present various mathematical formulas and laws to prove complicated theorems in the form of visual images, opens up new possibilities for understanding the laws of functioning of consciousness. Cognitive graphics tools are associated with many advanced information technologies, including hypertext and multimedia.

Graphical editors for processing raster graphics are designed to work with photographs and includes a set of tools for encoding images in digital form.

# Desktop Publishing

Desktop publishing systems are designed for the preparation of brochures, design of magazines and books. They allow to prepare documents and print them.

Desktop publishing software are applications designed for professional publishing, allowing make up in pages different types of documents.

The software packages of this type provides the tools that allow то make the following:

* compose (make up) text;
* use all kinds of fonts and images;
* graphics processing;
* provide output high quality documents;
* and etc.

There are two basic kinds of publishing systems. Publishing systems of the first type are suitable for the preparation of small materials with illustrations, graphs, diagrams, different fonts in the text (for example, newspapers and small magazines).

Desktop publishing of the second type are more suitable for the preparation of large documents, such as books. These applications can import text, prepared with the help of word processors, retaining the formatting.

The main function of desktop publishing software is a layout (placement of the text on the pages of the document, insert pictures, formatting text with different fonts, etc.). In publishing systems, text editing is less convenient than in text editors. So it happens that documents are prepared in two stages: a text is typed in a word processor, and then read its with publishing system and carry out the final preparation of the document.

Publishing systems allow the use of hundreds of types of fonts (type styles and sizes of characters of text) that are displayed on the screen as well as in print. They support the change of figures and charts, creating tables and formulas, and others.

To perform the work of publishing for the majority of users it may be enough of word processor capabilities, which include elements of color highlighting and means of graphic editors.

# Spreadsheets

Many tasks require a tabular presentation of data and their subsequent analysis. For example, tables are used in the balance sheet, the various financial reports, etc. **Spreadsheet software** is used for storing and processing information presented in tabular form.

Spreadsheets allow to create tables and automate the processing of tabular data. In addition, using a spreadsheet, you can perform a variety of economic, accounting and engineering calculations, as well as to build all sorts of charts, perform sophisticated economic analysis, simulate and optimize the solution of various business operations, and more.

Functions of spreadsheet processors are very diverse and include:

* creation and editing of spreadsheets;
* their formating and printing;
* creating multi-sheet documents linked with formulas;
* creating diagrams, their modification and solution of economic tasks with graphical methods;
* working with spreadsheet as a database (sorting, data sampling with requests);
* the creation of the final and summary tables;
* using in the construction information tables from external databases;
* forecasting solution of economic problems by adjusting the parameters;
* solution of optimization problems;
* statistical data processing;
* macros development and other.

Different spreadsheet applications can vary a set of functions and interface. A sheet of spreadsheet is a two-dimensional array of rows and columns. Rows are denoted by numbers and columns - by letters. A place of crossing a row and column is called a cell. The address (ID) of the cell is a letter indicating a column and a number indicating the line number.

There are an absolute and relative cell addressing. Absolute addressing is used more widely. With relative addressing, there is reference to the increment with a sign from the beginning of the desired cells.

# Database Management Systems

A **database management system** (DBMS) is a software combined with a specialized language that is designed to work with databases.

The **database** is a data about a specific topic, in any way organized and stored in the computer's external memory (e.g., hard disk). These data may be updated, extracted, search performed on them, and other operations. The main objective of the organization of database is easy to manipulate the data. A database management system should provide this convenience.

DBMSs are different. In addition, in relation to them, experts may be of three types:

* Developers of database management systems.
* Developers of databases on the specific DBMS.
* Users of databases and DBMSs.

Users input and change an information in a database, looking for information in them, perform a variety of inquiries and others. In other words, users use the database for its intended purpose. A DBMS is an instrument (software), with which they work.

Many database management systems are applications allowing to work with them without knowing a database programming language. Examples of such applications are Microsoft Access and LibreOffice Base. Other DBMSs are suggesting a more serious level of professional knowledge. An example of such DBMS is MySQL.

**DBMSs are classified by a number of attributes: data models (relational, hierarchical, etc.), by the method of access. Today, relational databases are the most common.**

**A relational database management system should allow to create data tables, queries, and make data changes, etc.**

**Topic № 9.**

Simulation is one of the ways of knowing the world.

Modeling concept is rather complicated. It includes a huge variety of ways of modeling: from creation of reduced or enlarged copies of real objects to conclusion of mathematical formulas.

For the research and knowledge of the various phenomena and processes, different ways of modeling are used.

An object that is the result of the simulation, is called a **model**. This is not necessarily a real object. However, it can replace the original in the study and description of the behavior.

Although a model can be an exact copy of an original, but most models have some important elements of the study, and the other are neglected. This simplifies a model. In addition, it is often impossible to create an exact copy of an original. For example, when modeling the behavior of an object in space. We can say that a model is a certain way of describing the real world.

Modeling goes through three stages:

* Creating a model.
* Studying model.
* Application of research results into practice and/or formulation of theoretical conclusions.

There is huge amount of types of modeling and simulation. Here are some examples of the types of models:

**Mathematical models**. This is a model describing specific numeric ratios.

**Graphical models** are used for the visual representation of complex objects. Description of these objects in other ways does not give people a clear understanding.

**Simulation models** allow to observe changes in the behavior of components of a system, conduct experiments changing some parameters of a model.

In a simulation important role plays interdisciplinary connections. Therefore, to create a model it can bring together experts from different fields.

**Features of computer simulation**

Improved computer technology and wide distribution of personal computers have opened vast prospects for research in modeling of processes and phenomena of the world, including the human society.

The computer simulation is a modeling implemented with the help of computer technology.

Availability of certain software is important for computer modeling. In the computer simulation, different software can be used. It can be versatile (text, table, graphics processors) or highly specialized, designed only for certain types of simulation.

Very often computers are used for mathematical modeling. Here, they are invaluable in performing numerical operations, while the task analysis is performed by a man.

Usually, different kinds of computer simulation are complementary. So, if a mathematical formula is very complicated and does not give the obvious idea of the described processes, the graphic and simulation models come to the rescue. Computer visualization can be much cheaper than creating real models.

With the advent of powerful computers, is spread graphical modeling based engineering systems for creating drawings, charts, graphs.

If a system is complex, and there is the need to track each of its elements, computer simulation models can come to the aid. They make it possible to reproduce the sequence of events, and then process the large amount of information.

However, it should be understood that the computer is a good tool to create and study models, but it can not come up models. Abstract analysis of the surrounding world in order to recreate it in the model is done by a man.

**Notion of Algorithm**

In today's society the word "algorithm" is so widespread that intuitive. We understand the algorithm as any sequence of steps to achieve a particular goal. However, theoretical science concept of an "algorithm" is rather complicated.

There is no single definition of the algorithm, although mostly different sources give very similar definitions.

An algorithm contains the following components:

An algorithm is a finite sequence of instructions …

* ... in the language understood to the executor ...
* ... defining the process of solving certain types of tasks ...
* ... and leading to the result that uniquely determined by the admissibility of the original data.

The result of an algorithm is clearly definitely depends on the source data. I.e., same algorithm gives different results for different source data. On the other hand, if the same algorithm repeatedly receives the same data, then it should give the same of times the same result.

The word "algorithm" is derived from the name of the IX century scientist Al-Khwarizmi. He described the rules of performing arithmetic operations in the decimal numeral system. At that time, the word "algorithm" meant the rules of calculations. However, over time the concept of the algorithm is modified and in the XX century under him began to realize any sequence of actions leading to the solution of the problem.

First, the definition of the algorithm has been a problem of mathematics, but over time, the theory of algorithms began to develop due to the influence of discoveries not only in mathematics, but also in computer science. Currently, the algorithm is one of the basic concepts of the informatics.

In other words, it should be understood that the theory of algorithms originated in mathematics and was a search for ways to solve a specific type of tasks by means of a specific set of instructions.

**Algorithm Properties**

1. Readability (in this case, the division into parts), and the ordering. The algorithm should consist of separate actions that are performed one after other.
2. Determinism (unique distinctness). Repeated application of an algorithm to the same set of input data always gives the same result.
3. Formality. The algorithm should not allow varying interpretations of action for the executor.
4. Effectiveness and completeness. The algorithm should be completed within a certain number of steps, and the task to be solved.
5. Mass. A specific algorithm should be applicable to all the same type of tasks.

**An executor and a developer of an algorithm**

Only human can inventing and developing algorithms. But formally (without thinking and evaluating) to perform, can any machine (eg, computers, household appliances). What is the benefit of such a division of labor? The fact that a person is exempt from routine activities, which can often take a long time, and instructs it to machines.

However, machines are not people: devices understand only a limited number of commands and can process the data (objects) not all types. Hence, in the end, a developer must describe an algorithm within permissible commands a specific executor (a machine). An executor's system commands is a set of commands, that it can perform. Objects (data) over which an executor can perform actions, are shaping the environment of executor.

**A programming language is a means of recording of algorithms for computers**

The computer is a quite versatile performer. It can perform a variety of types of algorithms: to do mathematical calculations, to handle text data, modify a graphic, etc. Computers perform many actions is much faster than a human. However, a person and a computer "talk" in different languages: a man says in a natural language (english, russian, and others), and a computer - a formal (machine) language.

A man must somehow "explain" his algorithm to a computer. Programming languages are used for this purpose, as a result of a recording of a algorithm is a program for them.

We can say, a programming language is a kind of intermediary between a person and a computer. A program written in a programming language, then is translated into a machine language with a translator.

**Formal language**

**Language** is a system of symbols and rules for the transmission of messages. There are natural languages in which people communicate, and artificial (or formal).

Language is set by the alphabet, syntax and semantics. A **programming language** is a formal language that provides a description of the specific problems formulated and solved with a person using the computer.

An **alphabet** is an ordered set of characters (**letters**) in a given language or system. Only characters belonging to a given alphabet can be used to build words.

The **syntax** is a set of rules for constructing words, structures and patterns of text in a language or system. Errors when writing the program and relating to only syntax are detected when parsing, carried out by the translator.

A **word** can be defined as an ordered set of symbols in a given alphabet, a word has a specific meaning.

A **translator** is a program that translates a program from one language to another.

**Semantics** is the meaning of each syntactic structure in language or system.

A translator translates syntax of a programming language to commands that are understood by an operating system and processor. A translator can not detect semantic errors. Their search occurs during debugging, testing and use of a program by a man.

The study of algorithms is of great practical significance. This is due to the fact that the creation of the algorithm requires a detailed description of each step of solving the problem, and ultimately step algorithm can be fairly simple to perform with a computer. This means that a solving of tasks can be automated.

However, one should always keep in mind that not all problems have algorithmic solution.

But for those tasks that have algorithmic solution, can be developed various algorithms. But probably the most effective would be only one.

# Flowcharts

An algorithm can be described in different ways: words, the programming language and using flowcharts.

In the language of flowcharts each step of an algorithm is described by the corresponding figures and the sequence of steps is determined by the lines of bonds. Flowcharts are read from top to bottom and left to right.

Flowcharts are useful because they provide easy "readability" of the algorithm. However, this is not always the case. If more or less complex algorithm have been described in terms of flowcharts, the scheme grows to incredible size and loses its visual advantage. Therefore, flowcharts are only good for the description of small algorithms in structured programming.

The language of flowcharts is simple (although there are advanced options). Below there are the figures of flowcharts and their meaning.

* A rectangle is a performing an action (for example, c = a + b).
* A rhombus is a check conditions (for example, a > b). If the condition is met, then the algorithm goes through the "yes" line, if not satisfied - through "no" line.
* A rounded rectangle denotes the beginning and end of a algorithm.
* A beveled rectangle indicates input and output data (for example, obtaining the value of the variable, output of the result on the screen).

This is not a complete description of the flowchart language.

# Types of algorithms

In structured programming, tasks that have an algorithmic solution, can be recorded using the following algorithmic structures:

* **Sequence**. It assumes sequential execution of commands from top to down. If the algorithm is composed only of the sequential structures, it is linear.
* **Selection** (branching). Execution of the program goes on an one of two, several or plurality of branches. Selecting a branch depends on the conditions at the input and the data received here.
* **Cycle**. It suggests the possibility of multiple repetition of certain actions. The number of repetitions depends on the cycle conditions.
* **Function** (subprogram). Commands, separated from the main program, are executed only if they are called from the main program (from any of its place). The same function can be called from the main program any number of times.

## Description of the various algorithmic structures on the flowchart language

### The "if" construct

This is the easiest type of branching. If the result of the calculation condition returns true, the execution of an algorithm goes on the "Yes" branch, which includes an additional expressions. If the condition returns false, then the execution of an algorithm goes on a "No" branch, ie the main branch of a program continues to run.

### The "if-else" construct

If the condition is true, then the execution of an algorithm goes on the "Yes" branch, if the condition is not met (false), then the execution goes on the "No" branch. Whatever the result of the conditions, the program can not go back to the main branch of the program, without additional action.

### The "if-elif-else" construct

The amount of conditions may be different. If the first is true, then after the actions, the program goes to the main branch without checking further conditions. If the first condition returns false, the second condition is checked. If the second condition returns true, the actions included in the second branch of the structure, are implemented. The last block action ("else") is executed only if none of the condition before it has not produced the result true. This algorithmic construct (if - elif - else) should not be confused with algorithmic construct "case".

### The "while" loop

While the condition is satisfied (result of logical expression yields true), the action will be carried out of the loop body. After another implementation embedded action, the condition is tested again. In order for the algorithm is not fixated, in the body of loop (among other actions) must be an expression, as a result of which will change the variable used in the condition. A body of the loop may never be executed if the condition is initially gave false.

### The "do" loop

In this cycle, the first time the condition is checked only after execution of actions of the loop body. If the condition is true, then the body is executed again. Whatever the condition, the body of the loop is executed at least once.

### The "for" loop

In the title of this cycle, three parameters are specified: the initial value of the variable (from), the final value (up) and change it with the help of an arithmetic operation on each "turn" cycle (step)

**Algorithm Examples**

**Euclid's algorithm (finding the greatest common divisor)**

**Euclid's algorithm** is an algorithm for finding the greatest common divisor (GCD) of a pair of integers.

**The greatest common divisor (GCD)** is a number that divides the two numbers without a residue; and itself is divided without residue on any divider of the two numbers. Simply put, this is the largest number that divides the two numbers without a residue.

**Description of the algorithm for finding the GCD by division**

1. The division of a larger number by a smaller number.
2. If it is divisible, then a smaller number is the GCD (should get out of the cycle).
3. If there is a residue, the larger number is replaced for the remainder of the division.
4. Go to step 1.

**Example:**

Find the GCD of 30 and 18.

30/18 = 1 (12 remainder)

18/12 = 1 (6 remainder)

12/6 = 2 (0 remainder). End: the GCD is a divider. GCD (30, 18) = 6

**Trial division ("testing of simplicity")**

**Trial division** is an algorithm for determining a prime number or a composite.

The algorithm is a consistent division of given natural number on all integers, starting with the deuce and ending with a value less than or equal to the square root of this number. If even one divider divides this number without the residue, the number is a complex. If the number does not have any divider dividing it without residue, then the number is prime.

**The sieve of Eratosthenes is an algorithm for determining the prime numbers**

The sieve of Eratosthenes is an algorithm for finding prime numbers up to a given number n. During execution of the algorithm gradually eliminated composite numbers, multiples of simple, starting with 2.

**Machine Post**

A **Post machine** is an abstract (actually nonexistent) computer, designed to clarify (formalization) of the concept of an algorithm. It is a versatile performer, allowing to enter the initial data and read the result of a program.

In 1936, in an article, American mathematician Emil Post described the system with algorithmic simplicity and the ability to determine whether a task is algorithmically solvable. If a task has an algorithmic solution, then it can be represented in the form of instructions for a Post machine.

A Post machine consists of …

1. an endless tape divided by the same cell (section). A cell can be empty (0 or empty) or contain a mark (1 or any other sign);
2. head (carriage), able to move through the tape one cell in one direction or another, and able to check labels, erase and record a label.

The current state of the Post machine is described with the state of the tape and the position of the carriage. The state of the tape is an information what sections are empty and which are marked. A step is a movement of the carriage on the one cell to the left or right. The state of the tape can be changed at runtime.

The carriage is controlled with a program consisting of command lines. Each command has the following syntax:

i K j

i - number of a command, K - action of the carriage, j - number of the next command (reference).

Total for a Post machine, there are six types of commands:

* V j - to put a label and move to the j-th line of the program.
* X j - erase a label, go to the j-th line of the program.
* <- j - move to the left, move to the j-th line of the program.
* -> j - move to the right, go to the j-th line of the program.
* ? j1; j2 - if there is no label in the cell, then go to the j1-th row of the program, or go to the j2-th row of the program.
* ! - the end of the program (stop).

The command "stop" has no a reference.

Options of the end of the program on a Post machine:

1. The command "stop" is the correct stop. It arises as a result of well written algorithm.
2. Performing an invalid command is not right stop. Cases where the head should put a mark where it already exists, or erase a mark where it is no, are emergency (unacceptable).
3. Care to infinity loop. Post machine as a result of the algorithm may never stop (never reach the command "stop" and has never completed an emergency).

Elementary actions (commands) of a Post machine are simpler than commands of a Turing machine. Therefore, programs for the Post machine have more commands than similar programs for a Turing machine.

Why are only two different character (there is a label, no label) quite? The fact that any alphabet can be coded with two characters; depending on only number of alphabet characters in the binary alphabet letter may increase.

**An example of work of a Post machine**

Objective: to increase the number 3 to 4 (change the value in the memory 3 to 4).

A positive integer on the tape of a Post machine is represented with consecutive marks, which is one greater than the given integer. This is due to the fact that one label is zero, and only two - one, etc.

For example, it is known that the carriage is somewhere to the left of labels and overlooks an empty cell. Then the program is to increase the number by one might look like:

1 -> 2  
2 ? 1;3  
3 <- 4  
4 V 5  
5 !

**Turing machine**

In 1936, Alan Turing proposed a universal abstract executor for clarify the concept of an algorithm. The abstractness of the Turing machine is that it is a logical computing construction, not a real computer. The term "universal executor" suggests that the executor can simulate any other performer. For example, operations that real computers perform, can simulate on the universal executor. Subsequently, invented by Turing a computational machine was named a Turing machine. In addition, it is assumed that an universal performer must be able to prove the existence or absence of an algorithm for a particular task.

**What is a Turing machine?**

Turing machine consists of an endless belt on both sides divided into cells, and the automatic (head), which is controlled by a program.

Programs for Turing machines are recorded in a table where the first column and row are containing the letters of the external alphabet and internal state of a machine (internal alphabet). The contents of the table is the command to Turing machine. A letter, that the head reads in a cell (over which it is at the moment), and internal state of the head are determining which command to run.

**To specify a particular Turing machine, it is required to write to the following components:**

* **External alphabet.** The final set (for example, A) whose elements are called letters (symbols). One of the letters of the alphabet (for example, a0) should be a null character.
* **Internal alphabet.** The final set of states of the head (machine). One of the states (for example, q1) should be the initial (run programs). Another one of the states (q0) must be finite (final program) - the stop state.
* **Table transitions.** The behavior of the machine (head) depending on the condition and a readed character.

**An automatic of a Turing machine can perform the following actions in the course of their work:**

* To record an symbol of the external alphabet in a cell (including a blank), replacing is in it (including the blank).
* To move one cell to the left or right.
* To change its internal state.

One command for a Turing machine is a specific combination of three components: an indication of what the character is being written in the cell, where move and what is a new state. Although, a command may not contain all the components (for example, do not change the character, not to move or not to change the internal state).

**An example work of a Turing machine**

For example, on a tape there is a word consisting of the characters #, $, 1, and 0. All the characters # and $ are required to be replaced by zeros. At the time of launch, the head is on the first letter of the word to the left. The program ends when the head is over the null character after the rightmost letter of the word.

Note: The word length and the sequence of characters is not important. The figure is an example of a sequence of commands for a particular case. If another word is on the tape, the sequence of commands will be different. Despite this, the program for a Turing machine (in the picture - the table on the left) is applicable to any word given external alphabet (observed property of the applicability of all the same type of problems — mass).

You can complicate the program. For example, the head is not required on the first and on any word character. Then this program for a Turing machine can be such (but could be different):

There is a shift of the head to the left as long as it is over the null character. After that, the machine goes into the state q2 (team coincide with the teams q1 of the previous program).

**Topic № 10.**

In programming, algorithm are the set of well defined instruction in sequence to solve a program. An algorithm should always have a clear stopping point.

Qualities of a good algorithm

1. Inputs and outputs should be defined precisely.
2. Each steps in algorithm should be clear and unambiguous.
3. Algorithm should be most effective among many different ways to solve a problem.
4. An algorithm shouldn't have computer code. Instead, the algorithm should be written in such a way that, it can be used in similar programming languages.

The basic actions involved in a computer algorithm are:

\_ Specify data values (using abstract data types)

\_ Perform calculations and assign data values

\_ Test data values and select alternate courses of actions including repetitions

\_ Terminate the algorithm

The algorithm-level architecture supports abstract data types and abstract data structures;

the algorithm designer formulates suitable abstract data structures and develops an

algorithm that operates on the data structures so as to solve the problem. Providing abstract

data types enables the algorithm designer to develop more general algorithms that

can be used for di\_erent applications involving di\_erent data types. This is often called

algorithm abstraction. For instance, a sorting algorithm that has been developed without

specifying the data types being sorted, can be programmed to sort a set of integers or a

set of characters. Similarly, when considering a data structure, such as an array, it is often

more productive to ignore certain details, such as the exact bounds of its indices. This is

often called data abstraction.

Algorithm Efficiency: Computer theorists are mainly concerned with discovering the

most efficient algorithms for a given class of problems. The algorithm's efficiency relates its

resource usage, such as execution time or memory consumption, to the size of its input data,

n. The efficiency is stated using the \Big O" notation, O(n). For example, if an algorithm

takes 4n - 2n + 2 steps to solve a problem of size n, we can say that the number of steps is

O(n2). Programmers use their knowledge of well-established algorithms and their respective

complexities to choose algorithms that are best suited to the circumstances. Examples of

such algorithms are quick-sort for sorting data (which has an an (n log n) average running

time), and binary search for searching through sorted data (which has an O(log2n) time).

Algorithms can be specified in different ways. Two common methods are pseudocode

descriptions and flowchart diagrams. A pseudocode description uses English, mathematical

notations, and a limited set of special commands to describe the actions of an algorithm. A

flowchart diagram provides the same information graphically, using diagrams with a finite

set of symbols in the place of the more elegant features of the pseudocode. A computer

cannot directly understand either pseudocode or owcharts, and so algorithm descriptions

are translated to computer language programs, most often by human programmers. Thus,

a computer program is an embodiment of an algorithm; strictly speaking, an algorithm is

a mental concept that exists independently of any representation.

Coded language used by programmers to write instructions that a computer can understand to do what the programmer (or the computer user) wants. The most basic (called low-level) computer language is the machine language that uses binary ('1' and '0') code which a computer can run (execute) very fast without using any translator or interpreter program, but is tedious and complex. The high-level languages (such as Basic, C, Java) are much simpler (more 'English-like') to use but need to use another program (a compiler or an interpreter) to convert the high-level code into the machine code, and are therefore slower.

Computer programming languages are used to to communicate instructions to a computer. They are based on certain syntactic and semantic rules, which define the meaning of each of the programming language constructs.

* Interpreted Programming Languages
* Functional Programming Languages
* Compiled Programming Languages
* Procedural Programming Languages
* Scripting Programming Languages
* Markup Programming Languages
* Logic-Based Programming Languages
* Concurrent Programming Languages
* Object-Oriented Programming Languages

# Interpreted Programming Languages

*An****interpreted language****is a programming language for which most of its implementations execute instructions directly, without previously compiling a program into machine-language instructions. The interpreter executes the program directly, translating each statement into a sequence of one or more subroutines already compiled into machine code. (Wikipedia)*

## ****APL****

Named after the book A Programming Language (Iverson, Kenneth E., 1962), APL is an array programming language. It can work simultaneously on multiple arrays of data. It is interpretive, interactive and a functional programming language.

## BASIC

Developed by John George Kemeny and Thomas Eugene Kurtz at Dartmouth in 1964, it is an acronym for **B**eginner’s **A**ll-purpose **S**ymbolic **I**nstruction **C**ode. It was designed with the intent of giving the non-science people an access to computers.

## Forth

It is a structured imperative programming language, which bases its implementation on stacks. It supports an interactive execution of commands as well as the compilation of sequences of commands.

## Lisp

Lisp is the second-oldest high-level programming language in widespread use today. The name Lisp is derived from ‘List Processing Language’. One of the important data structures that Lisp supports is linked list. Lisp programs deal with source code as a data structure.

## Lua

Members of the Computer Graphics Technology Group developed Lua in 1993. It is an imperative and procedural programming language that was designed as a scripting language. It is known for being simple yet powerful.

## Perl

Perl is a high-level interpreted programming language that supports dynamic programming. It was developed by Larry Wall, a linguist who served as a systems administrator at NASA. It provides the programmers with text processing facilities and has a blend of features adopted from various languages like C, Lisp, and Awk.

## PostScript

It is used in the desktop publishing field and is known as a page description language. It is a dynamically typed stack-based programming language developed by John Warnock, an American computer scientist and Charles Geschke, a notable figure in the field of computer science. These developers went on to found the very well-known company, Adobe Systems.

## Python

It is a high-level programming language that supports imperative, object-oriented, and functional programming paradigms. In its features like the dynamic type system and automatic memory management, it is similar to Perl. Originally released in 1991 by Guido van Rossum, a Dutch computer programmer, Python is an open community-based language whose development is managed by the Python Software Foundation.

## Ruby

The efforts for developing this language initiated in Japan in the 1990s. Similar to Perl, it has a dynamic type system and an automatic memory management. It supports multiple programming paradigms and is a dynamic object-oriented language.

# [Functional Programming Languages](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

***[Functional programming languages](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)****[define every computation as a mathematical evaluation. They focus on the application of functions. Many of the functional programming languages are bound to mathematical calculations.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)*

## [F#](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[It targets the .NET Framework and supports both functional as well as imperative object-oriented programming. Don Syme at the Microsoft Research developed this language, which is now being developed at the Microsoft Developer Division. F Sharp, as it is called, will soon be integrated into the .NET Framework and Visual Studio.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

## [Haskell](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[Named in honor of Haskell Curry, a logician, Haskell is a standardized purely functional language. It supports pattern matching, definable operators, single assignment, algebraic data types and recursive functions.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

# [Compiled Programming Languages](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

*[A](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)****[compiled language](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)****[is a programming language whose implementations are typically compilers (translators that generate machine code from source code), and not interpreters (step-by-step executors of source code, where no pre-runtime translation takes place). (Wikipedia)](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)*

## [Ada](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[It is a statically typed, structured, imperative programming language that is based on Pascal. A team of CII Honeywell Bull that was led by Jean Ichbiah developed Ada. The Ada compilers are validated for mission-critical systems. Ada is an internationally standardized computer programming language.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

## [ALGOL](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[Algorithmic Language, as it is called, is actually a family of imperative programming languages that was developed in the middle 1950s. It proved instrumental in the creation of programming languages like BCPL, B and C. Ole-Johan Dahl and Kristen Nygaard of the Norwegian Computing Center in Oslo were the brains behind Simula.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

## [C](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[Dennis Ritchie at the Bell Telephone Laboratories developed C to be used on the Unix platform. It is a general-purpose, cross-platform, procedural, imperative programming language. It is used for implementing system software and application software and is one of the most-used computer programming languages of today. The development of C++ and C# was influenced by C.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

## [C++](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[It consists of a combination of high-level and low-level language features and is hence considered as a middle-level programming language. Bjarne Stroustrup of Bell Labs developed C++ as an extension of the C language. Originally known as ‘C with Classes’, it came to be known as C++ from 1983. It is a multi-paradigm language that supports procedural programming, generic programming, object-oriented programming, and data abstraction.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

## [C#](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[C Sharp is a multi-paradigm programming language that supports imperative, generic and object-oriented programming. It is a part of the Microsoft .NET Framework. It is similar to C++ in its object-oriented syntax and is also influenced by Java and Delphi.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

## [COBOL](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[The name stands for Common Business-Oriented Language that is designed for the business and finance domain. COBOL 2002 standard supports object-oriented programming. It is one of the very old programming languages that are still in use.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

## [Fortran](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[It is a procedural, imperative, general purpose computer programming language that works well for scientific computations and numeric operations. After IBM developed it in the 1950s, it soon gained popularity in programming. It is very popular in the field of high-performance computing. It is a structured and compiled programming language that is a subset of Fortran95. Fortran 2003, a revised version of Fortran supports object-oriented programming.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

## [Java](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[It is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. Compiled Java code can run on all platforms that support Java without the need for recompilation. It is a very popular language of the modern times.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

## [Objective-C](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[It is a reflective object-oriented programming language that adds messaging services to C.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

## [Smalltalk](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[It is a reflective, object-oriented programming language that supports dynamic typing. Alan Kay, Adele Goldberg, Dan Ingalls, Scott Wallace, Ted Kaehler and their associates at Xerox PARC developed Smalltalk. They designed it for educational use and it soon became popular. VisualWorks is a prominent implementation of Smalltalk. Squeak is a programming language that is in the form of an implementation of Smalltalk. Scratch is a visual programming language based on Squeak.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

## [Turing](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[It was developed by Ric Holt and James Cordy of the University of Toronto, Canada, in 1982. It was named in honor of the British computer scientist, Alan Turing. This Pascal-like language is a freeware since 2007.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

## [Visual Basic](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[It is an event-driven programming language that is packaged with an integrated development environment. It inherits many of its features from BASIC. Its graphical development features make it easy for beginners to learn VB.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

## [Visual FoxPro](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[It is an object-oriented and procedural programming language derived from FoxPro. It is integrated with a relational database system of its own and does not require an additional programming environment. It supports dynamic programming.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

# [Procedural Programming Languages](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

***[Procedural (imperative) programming](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)****[implies specifying the steps that the programs should take to reach to an intended state. A procedure is a group of statements that can be referenced through a procedure call. Procedures help in the reuse of code. Procedural programming makes the programs structured and easily traceable for program flow.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)*[.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

## [Modula-2](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[It is a general-purpose procedural language created in 1978 by Niklaus Wirth at ETH. It is similar to Pascal and has systems programming and multiprogramming features.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

## [Oberon](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[Niklaus Wirth, the man behind Pascal and Modula came up with Oberon in 1986. It was designed as a part of the Oberon operating system. It is similar to Modula-2 but smaller than it.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

## [Component Pascal](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[It is a programming language that seems to be related to Pascal, but is actually incompatible with it. It is actually a variant of Oberon-2. Lagoona is an experimental programming language that supports component-oriented programming, a paradigm of decomposing a system into logical or functional components. Michael Franz, a student of Niklaus Wirth developed Lagoona. Seneca, better known as Oberon-2 is an extension of the Oberon programming language.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

## [MATLAB](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[It is a numerical computing environment and a programming language that enables matrix computations, function plotting, and algorithm implementation. It can also be used for user interface creation. MathWorks created MATLAB.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

## [PL/I](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[It is an imperative computer programming language targeted at scientific and engineering applications. Mainly intended to perform data processing, it also supports structured programming and recursion.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

# [Scripting Languages](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

***[Scripting languages](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)****[are programming languages that control an application. Scripts can execute independent of any other application. They are mostly embedded in the application that they control and are used to automate frequently executed tasks like communicating with external programs.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)*

## [PHP](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[PHP is one of the very popularly used general purpose scripting languages. It is developed for creating dynamic web pages and supports a command line interface capability.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

## [VBScript](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[It is an active scripting language that Microsoft developed as a variation of Microsoft Visual Basic. VBScript is a default component with each of the Desktop releases of Microsoft Windows.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

# [Markup Languages](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

*[A](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)****[markup language](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)****[is an artificial language that uses annotations to text that define how the text is to be displayed.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)*

## [HTML](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[Hypertext Markup Language, abbreviated as HTML, is the most prominent markup language that is used for web pages. It is written in the form of HTML tags that are surrounded by angular brackets. HTML tags describe the appearance of the text in a document and can be embedded into certain other code to affect the web browser behavior. HTML uses the SGML default syntax.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

## [XML](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[The name stands for Extensible Markup Language. It is extensible because it allows the users to define their own XML elements. It supports the sharing of structured data over the Internet and the encoding and serializing of data. It originated as a subset of SGML. XPath is the XML Path Language that is used to select nodes from an XML document. It supports the computation of values. XQuery is used to query the collections of XML data. Extensible Stylesheet Language Transformations (XSLT) is an XML-based language that is used for the transformation of XML documents into human-readable formats. Apache Ant is a tool for the automation of software build processes. It uses XML to describe the build processes.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

## [XHTML](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[It is a markup language that is similar to HTML and follows the XML syntax. It is midway between HTML and XML. XHTML documents allow automated processing of data.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

# [Logic-based Programming Languages](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

***[Logic programming](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)****[is a type of programming paradigm which is largely based on formal logic. Any program written in a logic programming language is a set of sentences in logical form, expressing facts and rules about some problem domain. (Wikipedia)](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)*

## [Prolog](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[It is a general-purpose programming language that supports logic programming and is often linked with artificial intelligence and computational linguistics. The language is declarative and the program logic is expressed in the form of relations. Mercury is a functional logic programming language that is based on Prolog. Strawberry Prolog is a dialect of Prolog, which is supposed to be easy to use. Visual Prolog is a strongly typed extension of Prolog that supports object-oriented programming. It is a compiled logic-based programming language.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

# [Concurrent Programming Languages](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

***[Concurrent programming](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)****[is a computer programming technique that provides for the execution of operations concurrently — either within a single computer, or across a number of systems. In the latter case, the term distributed computing is used. (Wikipedia)](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)*

## [Concurrent Pascal](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[Per Brinch Hansen, a Danish-American computer scientist created Concurrent Pascal for writing operating systems and programming real-time systems.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

# [Object-Oriented Programming Languages](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

***[Object-oriented programming](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)****[(](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)****[OOP](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)****[) is a programming paradigm based on the concept of “objects”, which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods. In OOP, computer programs are designed by making them out of objects that interact with one another. (Wikipedia)](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)*

## [Scala](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

[The name Scala stands for Scalable Language. It is a multi-paradigm programming language, which offers object-oriented and functional programming features.](https://clickhelp.co/free-download/responsive-layouts-getting-started-guide/?utm_source=medium-com&utm_medium=responsive-banner&utm_campaign=blogging)

**Topic № 11.**

A **programming language** is a set of rules and functions that let people use computers, cell phones, tablets, and more devices. Programming languages are designed to make it easy for humans to write complex instructions. They function a lot like human languages: they have explicit grammars and a primitive vocabulary.

A programming language is a language for writing instructions for a machine. A programming language is defined by a grammar.

Many programming paradigms exist which focus on different design aspects. Imperative languages focus on commands and how to accomplish tasks, declerative languages dictate the rules and conditions of a task and do not specify the process of completing it, and functional languages emphasize functions which do not change state, and are guaranteed to always produce the same output for an input.

Languages may be designed to support paradigms, ways of programming.[[1]](https://wiki.scratch.mit.edu/wiki/Programming_Language#cite_note-1) Paradigms include:

* Imperative: Programs use statements
* Functional: Programs use functions and avoid state
* Declarative: Programs state rules of a problems
* Object-oriented: Message passing between objects
* Procedural: Programs are separated into subroutines

Languages usually have various primitive data types which can be expanded with Object-Oriented Programming:

* Numbers, which are parsed as numbers
  + Some languages consider integers, floating-point numbers, and doubles to be different data types.
* [Strings](https://wiki.scratch.mit.edu/wiki/String), which are parsed as text
* [Arrays](https://wiki.scratch.mit.edu/wiki/Array), which are lists of elements
* Objects, which are dictionaries of key-value pairs.
* [Functions](https://wiki.scratch.mit.edu/wiki/Functions), which are pieces of code which can be executed with [Arguments](https://wiki.scratch.mit.edu/wiki/Argument).

In essence, a programming language just provides a framework where a function can be executed with arguments—the rest can be worked around. Usually, the grammar of a language consists of "statements", which are either:

* Assignments: binding some value to a name ([variable](https://wiki.scratch.mit.edu/wiki/Variable))
* Procedure calls
* Special forms: certain specialized procedures which cannot be created using the language itself, for example, IF/ELSE. The inputs to special forms are not immediately evaluated.

We generally write a computer program using a high-level language. A high-level language is one which is understandable by us humans. It contains words and phrases from the English (or other) language. But a computer does not understand high-level language. It only understands program written in **0's** and **1's** in binary, called the machine code. A program written in high-level language is called a source code. We need to convert the source code into machine code and this is accomplished my compilers and interpreters. Hence, a compiler or an interpreter is a program that converts program written in high-level language into machine code understood by the computer.

The difference between an interpreter and a compiler is given below:

|  |  |
| --- | --- |
| Interpreter | Compiler |
| Translates program one statement at a time. | Scans the entire program and translates it as a whole into machine code. |
| It takes less amount of time to analyze the source code but the overall execution time is slower. | It takes large amount of time to analyze the source code but the overall execution time is comparatively faster. |
| No intermediate object code is generated, hence are memory efficient. | Generates intermediate object code which further requires linking, hence requires more memory. |
| Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy. | It generates the error message only after scanning the whole program. Hence debugging is comparatively hard. |
| Programming language like Python, Ruby use interpreters. | Programming language like C, C++ use compilers. |

**Topic № 12.**

C is a general-purpose, procedural, imperative computer programming language developed in 1972 by Dennis M. Ritchie at the Bell Telephone Laboratories to develop the UNIX operating system. C is the most widely used computer language. It keeps fluctuating at number one scale of popularity along with Java programming language, which is also equally popular and most widely used among modern software programmers.

Local Environment Setup

If you want to set up your environment for C programming language, you need the following two software tools available on your computer, (a) Text Editor and (b) The C Compiler.

Text Editor

This will be used to type your program. Examples of few a editors include Windows Notepad, OS Edit command, Brief, Epsilon, EMACS, and vim or vi.

The name and version of text editors can vary on different operating systems. For example, Notepad will be used on Windows, and vim or vi can be used on windows as well as on Linux or UNIX.

The files you create with your editor are called the source files and they contain the program source codes. The source files for C programs are typically named with the extension "**.c**".

Before starting your programming, make sure you have one text editor in place and you have enough experience to write a computer program, save it in a file, compile it and finally execute it.

The C Compiler

The source code written in source file is the human readable source for your program. It needs to be "compiled", into machine language so that your CPU can actually execute the program as per the instructions given.

The compiler compiles the source codes into final executable programs. The most frequently used and free available compiler is the GNU C/C++ compiler, otherwise you can have compilers either from HP or Solaris if you have the respective operating systems.

The following section explains how to install GNU C/C++ compiler on various OS. We keep mentioning C/C++ together because GNU gcc compiler works for both C and C++ programming languages.

Installation on UNIX/Linux

If you are using **Linux or UNIX**, then check whether GCC is installed on your system by entering the following command from the command line −

$ gcc -v

If you have GNU compiler installed on your machine, then it should print a message as follows −

Using built-in specs.

Target: i386-redhat-linux

Configured with: ../configure --prefix=/usr .......

Thread model: posix

gcc version 4.1.2 20080704 (Red Hat 4.1.2-46)

If GCC is not installed, then you will have to install it yourself using the detailed instructions available at <https://gcc.gnu.org/install/>

This tutorial has been written based on Linux and all the given examples have been compiled on the Cent OS flavor of the Linux system.

Installation on Mac OS

If you use Mac OS X, the easiest way to obtain GCC is to download the Xcode development environment from Apple's web site and follow the simple installation instructions. Once you have Xcode setup, you will be able to use GNU compiler for C/C++.

Xcode is currently available at [developer.apple.com/technologies/tools/](https://developer.apple.com/technologies/tools/).

Installation on Windows

To install GCC on Windows, you need to install MinGW. To install MinGW, go to the MinGW homepage, [www.mingw.org](http://www.mingw.org/), and follow the link to the MinGW download page. Download the latest version of the MinGW installation program, which should be named MinGW-<version>.exe.

While installing Min GW, at a minimum, you must install gcc-core, gcc-g++, binutils, and the MinGW runtime, but you may wish to install more.

Add the bin subdirectory of your MinGW installation to your **PATH** environment variable, so that you can specify these tools on the command line by their simple names.

After the installation is complete, you will be able to run gcc, g++, ar, ranlib, dlltool, and several other GNU tools from the Windows command line.

Hello World Example

A C program basically consists of the following parts −

* Preprocessor Commands
* Functions
* Variables
* Statements & Expressions
* Comments

Let us look at a simple code that would print the words "Hello World" −

[Live Demo](http://tpcg.io/3Ty4QP)

#include <stdio.h>

int main() {

/\* my first program in C \*/

printf("Hello, World! \n");

return 0;

}

Let us take a look at the various parts of the above program −

* The first line of the program *#include <stdio.h>* is a preprocessor command, which tells a C compiler to include stdio.h file before going to actual compilation.
* The next line *int main()* is the main function where the program execution begins.
* The next line /\*...\*/ will be ignored by the compiler and it has been put to add additional comments in the program. So such lines are called comments in the program.
* The next line *printf(...)* is another function available in C which causes the message "Hello, World!" to be displayed on the screen.
* The next line **return 0;** terminates the main() function and returns the value 0.

Compile and Execute C Program

Let us see how to save the source code in a file, and how to compile and run it. Following are the simple steps −

* Open a text editor and add the above-mentioned code.
* Save the file as *hello.c*
* Open a command prompt and go to the directory where you have saved the file.
* Type *gcc hello.c* and press enter to compile your code.
* If there are no errors in your code, the command prompt will take you to the next line and would generate *a.out*executable file.
* Now, type *a.out* to execute your program.
* You will see the output *"Hello World"* printed on the screen.

$ gcc hello.c

$ ./a.out

Hello, World!

Make sure the gcc compiler is in your path and that you are running it in the directory containing the source file hello.c.

You have seen the basic structure of a C program, so it will be easy to understand other basic building blocks of the C programming language.

Tokens in C

A C program consists of various tokens and a token is either a keyword, an identifier, a constant, a string literal, or a symbol. For example, the following C statement consists of five tokens −

printf("Hello, World! \n");

The individual tokens are −

printf

(

"Hello, World! \n"

)

;

Semicolons

In a C program, the semicolon is a statement terminator. That is, each individual statement must be ended with a semicolon. It indicates the end of one logical entity.

Given below are two different statements −

printf("Hello, World! \n");

return 0;

Comments

Comments are like helping text in your C program and they are ignored by the compiler. They start with /\* and terminate with the characters \*/ as shown below −

/\* my first program in C \*/

You cannot have comments within comments and they do not occur within a string or character literals.

Identifiers

A C identifier is a name used to identify a variable, function, or any other user-defined item. An identifier starts with a letter A to Z, a to z, or an underscore '\_' followed by zero or more letters, underscores, and digits (0 to 9).

C does not allow punctuation characters such as @, $, and % within identifiers. C is a **case-sensitive** programming language. Thus, *Manpower* and *manpower* are two different identifiers in C. Here are some examples of acceptable identifiers −

mohd zara abc move\_name a\_123

myname50 \_temp j a23b9 retVal

Keywords

The following list shows the reserved words in C. These reserved words may not be used as constants or variables or any other identifier names.

|  |  |  |  |
| --- | --- | --- | --- |
| auto | else | long | switch |
| break | enum | register | typedef |
| case | extern | return | union |
| char | float | short | unsigned |
| const | for | signed | void |
| continue | goto | sizeof | volatile |
| default | if | static | while |
| do | int | struct | \_Packed |
| double |  |  |  |

Whitespace in C

A line containing only whitespace, possibly with a comment, is known as a blank line, and a C compiler totally ignores it.

Whitespace is the term used in C to describe blanks, tabs, newline characters and comments. Whitespace separates one part of a statement from another and enables the compiler to identify where one element in a statement, such as int, ends and the next element begins. Therefore, in the following statement −

int age;

there must be at least one whitespace character (usually a space) between int and age for the compiler to be able to distinguish them. On the other hand, in the following statement −

fruit = apples + oranges; // get the total fruit

no whitespace characters are necessary between fruit and =, or between = and apples, although you are free to include some if you wish to increase readability.

**Topic № 13.**

Computers generate random number for everything from cryptography to video games and gambling. There are two categories of random numbers — “true” random numbers and pseudorandom numbers — and the difference is important for the security of encryption systems.

## What Random Numbers Are Used For

Random numbers have been used for many thousands of years. Whether it’s flipping a coin or rolling a dice, the goal is to leave the end result up to random chance. Random number generators in a computer are similar — they’re an attempt to achieve an unpredictable, random result.

Random number generators are useful for many different purposes. Aside from obvious applications like generating random numbers for the purposes of gambling or creating unpredictable results in a computer game, randomness is important for cryptography.

Cryptography requires numbers that attackers can’t guess. We can’t just use the same numbers over and over. We want to generate these numbers in a very unpredictable way so attackers can’t guess them. These random numbers are essential for secure encryption, whether you’re encrypting your own files or just using an  HTTPS website on the Internet.

## Pseudorandom Numbers

Pseudorandom numbers are an alternative to “true” random numbers. A computer could use a seed value and an algorithm to generate numbers that appear to be random, but that are in fact predictable. The computer doesn’t gather any random data from the environment.

This isn’t necessarily a bad thing in every situation. For example, if you’re playing a video game, it doesn’t really matter whether the events that occur in that game are caused by “true” random numbers or pseudorandom numbers. On the other hand, if you’re using encryption, you don’t want to use pseudorandom numbers that an attacker could guess.

For example, let’s say an attacker knows the algorithm and seed value a pseudorandom number generator uses. And let’s say an encryption algorithm gets a pseudorandom number from this algorithm and uses it to generate an encryption key without adding any additional randomness. If an attacker knows enough, they could work backwards and determine the pseudorandom number the encryption algorithm must have chosen in that case, breaking the encryption.

One mathematical function in C programming that’s relatively easy to grasp is the rand() function. It generates random numbers. Though that may seem silly, it’s the basis for just about every computer game ever invented. Random numbers are a big deal in programming.

A computer cannot generate truly random numbers. Instead, it produces what are known as *pseudo–random numbers*. That’s because conditions inside the computer can be replicated. Therefore, serious mathematicians scoff that any value a computer calls random isn’t a truly random number.

The rand() function is the simplest of C’s random-number functions. It requires the stdlib.h header file, and it coughs up an int value that’s supposedly random. Now, That’s Random demonstrates sample code.

#include <stdio.h>

#include <stdlib.h>

int main()

{

int r,a,b;

puts("100 Random Numbers");

for(a=0;a<20;a++)

{

for(b=0;b<5;b++)

{

r=rand();

printf("%dt",r);

}

putchar('n');

}

return(0);

}

Now, That’s Random uses a nested for loop to display 100 random values. The rand() function in Line 13 generates the values. The printf() function in Line 14 displays the values by using the %d conversion character, which displays int values.

The rand() function is good at generating a slew of random values, but they’re predictable values. To make the output less predictable, you need to *seed* the random-number generator. That’s done by using the srand() function.

#include <stdio.h>

#include <stdlib.h>

int main()

{

unsigned seed;

int r,a,b;

printf("Input a random number seed: ");

scanf("%u",&seed);

srand(seed);

for(a=0;a<20;a++)

{

for(b=0;b<5;b++)

{

r=rand();

printf("%dt",r);

}

putchar('n');

}

return(0);

}

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main()

{

int r,a,b;

srand((unsigned)time(NULL));

for(a=0;a<20;a++)

{

for(b=0;b<5;b++)

{

r=rand();

printf("%dt",r);

}

putchar('n');

}

return(0);

}

The time() function returns information about the current time of day, a value that’s constantly changing. The NULL argument helps solve some problems, but time() returns an ever-changing value.

The (unsigned) part of the statement ensures that the value returned by the time() function is an unsigned integer. That’s a technique known as *typecasting*.

**rand ()**

rand() function is used in C to generate random numbers. If we generate a sequence of random number with rand() function, it will create the same sequence again and again every time program runs. Say if we are generating 5 random numbers in C with the help of rand() in a loop, then every time we compile and run the program our output must be the same sequence of numbers.

**int rand(void):**

returns a pseudo-random number in the range of 0 to RAND\_MAX.

**RAND\_MAX:** is a constant whose default value may vary

between implementations but it is granted to be at least 32767.

**srand()**

The srand() function sets the starting point for producing a series of pseudo-random integers. If srand() is not called, the rand() seed is set as if srand(1) were called at program start. Any other value for seed sets the generator to a different starting point.  
**Syntax:**

**void srand( unsigned seed ):**

Seeds the pseudo-random number generator used by rand()

with the value seed.

**Note:** The pseudo-random number generator should only be seeded once, before any calls to rand(), and the start of the program. It should not be repeatedly seeded, or reseeded every time you wish to generate a new batch of pseudo-random numbers.  
Standard practice is to use the result of a call to **srand(time(0))** as the seed. However, time() returns a time\_t value which vary everytime and hence the pseudo-random number vary for every program call.

**How srand() and rand() are related to each other?**

srand() sets the seed which is used by rand to generate “random” numbers. If you don’t call srand before your first call to rand, it’s as if you had called srand(1) to set the seed to one.  
In short, **srand() — Set Seed for rand() Function**.

Data types in c refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

The types in C can be classified as follows −

|  |  |
| --- | --- |
| **Sr.No.** | **Types & Description** |
| 1 | **Basic Types**  They are arithmetic types and are further classified into: (a) integer types and (b) floating-point types. |
| 2 | **Enumerated types**  They are again arithmetic types and they are used to define variables that can only assign certain discrete integer values throughout the program. |
| 3 | **The type void**  The type specifier *void* indicates that no value is available. |
| 4 | **Derived types**  They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types. |

The array types and structure types are referred collectively as the aggregate types. The type of a function specifies the type of the function's return value. We will see the basic types in the following section, where as other types will be covered in the upcoming chapters.

Integer Types

The following table provides the details of standard integer types with their storage sizes and value ranges −

|  |  |  |
| --- | --- | --- |
| **Type** | **Storage size** | **Value range** |
| char | 1 byte | -128 to 127 or 0 to 255 |
| unsigned char | 1 byte | 0 to 255 |
| signed char | 1 byte | -128 to 127 |
| int | 2 or 4 bytes | -32,768 to 32,767 or -2,147,483,648 to 2,147,483,647 |
| unsigned int | 2 or 4 bytes | 0 to 65,535 or 0 to 4,294,967,295 |
| short | 2 bytes | -32,768 to 32,767 |
| unsigned short | 2 bytes | 0 to 65,535 |
| long | 4 bytes | -2,147,483,648 to 2,147,483,647 |
| unsigned long | 4 bytes | 0 to 4,294,967,295 |

To get the exact size of a type or a variable on a particular platform, you can use the **sizeof** operator. The expressions *sizeof(type)* yields the storage size of the object or type in bytes. Given below is an example to get the size of int type on any machine −

[Live Demo](http://tpcg.io/FnrxP5)

#include <stdio.h>

#include <limits.h>

int main() {

printf("Storage size for int : %d \n", sizeof(int));

return 0;

}

When you compile and execute the above program, it produces the following result on Linux −

Storage size for int : 4

Floating-Point Types

The following table provide the details of standard floating-point types with storage sizes and value ranges and their precision −

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Storage size** | **Value range** | **Precision** |
| float | 4 byte | 1.2E-38 to 3.4E+38 | 6 decimal places |
| double | 8 byte | 2.3E-308 to 1.7E+308 | 15 decimal places |
| long double | 10 byte | 3.4E-4932 to 1.1E+4932 | 19 decimal places |

The header file float.h defines macros that allow you to use these values and other details about the binary representation of real numbers in your programs. The following example prints the storage space taken by a float type and its range values −

[Live Demo](http://tpcg.io/mynMA1)

#include <stdio.h>

#include <float.h>

int main() {

printf("Storage size for float : %d \n", sizeof(float));

printf("Minimum float positive value: %E\n", FLT\_MIN );

printf("Maximum float positive value: %E\n", FLT\_MAX );

printf("Precision value: %d\n", FLT\_DIG );

return 0;

}

When you compile and execute the above program, it produces the following result on Linux −

Storage size for float : 4

Minimum float positive value: 1.175494E-38

Maximum float positive value: 3.402823E+38

Precision value: 6

The void Type

The void type specifies that no value is available. It is used in three kinds of situations −

|  |  |
| --- | --- |
| **Sr.No.** | **Types & Description** |
| 1 | **Function returns as void**  There are various functions in C which do not return any value or you can say they return void. A function with no return value has the return type as void. For example, **void exit (int status);** |
| 2 | **Function arguments as void**  There are various functions in C which do not accept any parameter. A function with no parameter can accept a void. For example, **int rand(void);** |
| 3 | **Pointers to void**  A pointer of type void \* represents the address of an object, but not its type. For example, a memory allocation function **void \*malloc( size\_t size );**returns a pointer to void which can be casted to any data type. |

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C is case-sensitive. Based on the basic types explained in the previous chapter, there will be the following basic variable types −

|  |  |
| --- | --- |
| **Sr.No.** | **Type & Description** |
| 1 | **char**  Typically a single octet(one byte). This is an integer type. |
| 2 | **int**  The most natural size of integer for the machine. |
| 3 | **float**  A single-precision floating point value. |
| 4 | **double**  A double-precision floating point value. |
| 5 | **void**  Represents the absence of type. |

C programming language also allows to define various other types of variables, which we will cover in subsequent chapters like Enumeration, Pointer, Array, Structure, Union, etc. For this chapter, let us study only basic variable types.

## Variable Definition in C

A variable definition tells the compiler where and how much storage to create for the variable. A variable definition specifies a data type and contains a list of one or more variables of that type as follows −

type variable\_list;

Here, **type** must be a valid C data type including char, w\_char, int, float, double, bool, or any user-defined object; and **variable\_list** may consist of one or more identifier names separated by commas. Some valid declarations are shown here −

int i, j, k;

char c, ch;

float f, salary;

double d;

The line **int i, j, k;** declares and defines the variables i, j, and k; which instruct the compiler to create variables named i, j and k of type int.

Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows −

type variable\_name = value;

Some examples are −

extern int d = 3, f = 5; // declaration of d and f.

int d = 3, f = 5; // definition and initializing d and f.

byte z = 22; // definition and initializes z.

char x = 'x'; // the variable x has the value 'x'.

For definition without an initializer: variables with static storage duration are implicitly initialized with NULL (all bytes have the value 0); the initial value of all other variables are undefined.

## Variable Declaration in C

A variable declaration provides assurance to the compiler that there exists a variable with the given type and name so that the compiler can proceed for further compilation without requiring the complete detail about the variable. A variable definition has its meaning at the time of compilation only, the compiler needs actual variable definition at the time of linking the program.

A variable declaration is useful when you are using multiple files and you define your variable in one of the files which will be available at the time of linking of the program. You will use the keyword **extern** to declare a variable at any place. Though you can declare a variable multiple times in your C program, it can be defined only once in a file, a function, or a block of code.

### Example

Try the following example, where variables have been declared at the top, but they have been defined and initialized inside the main function −

#include <stdio.h>

// Variable declaration:

extern int a, b;

extern int c;

extern float f;

int main () {

/\* variable definition: \*/

int a, b;

int c;

float f;

/\* actual initialization \*/

a = 10;

b = 20;

c = a + b;

printf("value of c : %d \n", c);

f = 70.0/3.0;

printf("value of f : %f \n", f);

return 0;

}

When the above code is compiled and executed, it produces the following result −

value of c : 30

value of f : 23.333334

The same concept applies on function declaration where you provide a function name at the time of its declaration and its actual definition can be given anywhere else. For example −

// function declaration

int func();

int main() {

// function call

int i = func();

}

// function definition

int func() {

return 0;

}

## Lvalues and Rvalues in C

There are two kinds of expressions in C −

* **lvalue** − Expressions that refer to a memory location are called "lvalue" expressions. An lvalue may appear as either the left-hand or right-hand side of an assignment.
* **rvalue** − The term rvalue refers to a data value that is stored at some address in memory. An rvalue is an expression that cannot have a value assigned to it which means an rvalue may appear on the right-hand side but not on the left-hand side of an assignment.

Variables are lvalues and so they may appear on the left-hand side of an assignment. Numeric literals are rvalues and so they may not be assigned and cannot appear on the left-hand side. Take a look at the following valid and invalid statements −

int g = 20; // valid statement

10 = 20; // invalid statement; would generate compile-time error

Constants refer to fixed values that the program may not alter during its execution. These fixed values are also called **literals**.

Constants can be of any of the basic data types like *an integer constant, a floating constant, a character constant, or a string literal*. There are enumeration constants as well.

Constants are treated just like regular variables except that their values cannot be modified after their definition.

Integer Literals

An integer literal can be a decimal, octal, or hexadecimal constant. A prefix specifies the base or radix: 0x or 0X for hexadecimal, 0 for octal, and nothing for decimal.

An integer literal can also have a suffix that is a combination of U and L, for unsigned and long, respectively. The suffix can be uppercase or lowercase and can be in any order.

Here are some examples of integer literals −

212 /\* Legal \*/

215u /\* Legal \*/

0xFeeL /\* Legal \*/

078 /\* Illegal: 8 is not an octal digit \*/

032UU /\* Illegal: cannot repeat a suffix \*/

Following are other examples of various types of integer literals −

85 /\* decimal \*/

0213 /\* octal \*/

0x4b /\* hexadecimal \*/

30 /\* int \*/

30u /\* unsigned int \*/

30l /\* long \*/

30ul /\* unsigned long \*/

Floating-point Literals

A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part. You can represent floating point literals either in decimal form or exponential form.

While representing decimal form, you must include the decimal point, the exponent, or both; and while representing exponential form, you must include the integer part, the fractional part, or both. The signed exponent is introduced by e or E.

Here are some examples of floating-point literals −

3.14159 /\* Legal \*/

314159E-5L /\* Legal \*/

510E /\* Illegal: incomplete exponent \*/

210f /\* Illegal: no decimal or exponent \*/

.e55 /\* Illegal: missing integer or fraction \*/

Character Constants

Character literals are enclosed in single quotes, e.g., 'x' can be stored in a simple variable of **char** type.

A character literal can be a plain character (e.g., 'x'), an escape sequence (e.g., '\t'), or a universal character (e.g., '\u02C0').

There are certain characters in C that represent special meaning when preceded by a backslash for example, newline (\n) or tab (\t).

Here, you have a list of such escape sequence codes −

Following is the example to show a few escape sequence characters −

[Live Demo](http://tpcg.io/psy5Zb)

#include <stdio.h>

int main() {

printf("Hello\tWorld\n\n");

return 0;

}

When the above code is compiled and executed, it produces the following result −

Hello World

String Literals

String literals or constants are enclosed in double quotes "". A string contains characters that are similar to character literals: plain characters, escape sequences, and universal characters.

You can break a long line into multiple lines using string literals and separating them using white spaces.

Here are some examples of string literals. All the three forms are identical strings.

"hello, dear"

"hello, \

dear"

"hello, " "d" "ear"

Defining Constants

There are two simple ways in C to define constants −

* Using **#define** preprocessor.
* Using **const** keyword.

The #define Preprocessor

Given below is the form to use #define preprocessor to define a constant −

#define identifier value

The following example explains it in detail −

[Live Demo](http://tpcg.io/p8YVAu)

#include <stdio.h>

#define LENGTH 10

#define WIDTH 5

#define NEWLINE '\n'

int main() {

int area;

area = LENGTH \* WIDTH;

printf("value of area : %d", area);

printf("%c", NEWLINE);

return 0;

}

When the above code is compiled and executed, it produces the following result −

value of area : 50

The const Keyword

You can use **const** prefix to declare constants with a specific type as follows −

const type variable = value;

The following example explains it in detail −

[Live Demo](http://tpcg.io/PnBPfK)

#include <stdio.h>

int main() {

const int LENGTH = 10;

const int WIDTH = 5;

const char NEWLINE = '\n';

int area;

area = LENGTH \* WIDTH;

printf("value of area : %d", area);

printf("%c", NEWLINE);

return 0;

}

When the above code is compiled and executed, it produces the following result −

value of area : 50

Note that it is a good programming practice to define constants in CAPITALS.

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators −

* Arithmetic Operators
* Relational Operators
* Logical Operators
* Bitwise Operators
* Assignment Operators
* Misc Operators

We will, in this chapter, look into the way each operator works.

Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language. Assume variable **A** holds 10 and variable **B**holds 20 then −

[Show Examples](https://www.tutorialspoint.com/cprogramming/c_arithmetic_operators.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + | Adds two operands. | A + B = 30 |
| − | Subtracts second operand from the first. | A − B = -10 |
| \* | Multiplies both operands. | A \* B = 200 |
| / | Divides numerator by de-numerator. | B / A = 2 |
| % | Modulus Operator and remainder of after an integer division. | B % A = 0 |
| ++ | Increment operator increases the integer value by one. | A++ = 11 |
| -- | Decrement operator decreases the integer value by one. | A-- = 9 |

Relational Operators

The following table shows all the relational operators supported by C. Assume variable **A** holds 10 and variable **B** holds 20 then −

[Show Examples](https://www.tutorialspoint.com/cprogramming/c_relational_operators.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| == | Checks if the values of two operands are equal or not. If yes, then the condition becomes true. | (A == B) is not true. |
| != | Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true. | (A != B) is true. |
| > | Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true. | (A > B) is not true. |
| < | Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true. | (A < B) is true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true. | (A >= B) is not true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true. | (A <= B) is true. |

Logical Operators

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then −

[Show Examples](https://www.tutorialspoint.com/cprogramming/c_logical_operators.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| && | Called Logical AND operator. If both the operands are non-zero, then the condition becomes true. | (A && B) is false. |
| || | Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true. | (A || B) is true. |
| ! | Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false. | !(A && B) is true. |

Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows −

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **p** | **q** | **p & q** | **p | q** | **p ^ q** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |

Assume A = 60 and B = 13 in binary format, they will be as follows −

A = 0011 1100

B = 0000 1101

-----------------

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

The following table lists the bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then −

[Show Examples](https://www.tutorialspoint.com/cprogramming/c_bitwise_operators.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| & | Binary AND Operator copies a bit to the result if it exists in both operands. | (A & B) = 12, i.e., 0000 1100 |
| | | Binary OR Operator copies a bit if it exists in either operand. | (A | B) = 61, i.e., 0011 1101 |
| ^ | Binary XOR Operator copies the bit if it is set in one operand but not both. | (A ^ B) = 49, i.e., 0011 0001 |
| ~ | Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. | (~A ) = -60, i.e,. 1100 0100 in 2's complement form. |
| << | Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand. | A << 2 = 240 i.e., 1111 0000 |
| >> | Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. | A >> 2 = 15 i.e., 0000 1111 |

Assignment Operators

The following table lists the assignment operators supported by the C language −

[Show Examples](https://www.tutorialspoint.com/cprogramming/c_assignment_operators.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Simple assignment operator. Assigns values from right side operands to left side operand | C = A + B will assign the value of A + B to C |
| += | Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand. | C += A is equivalent to C = C + A |
| -= | Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand. | C -= A is equivalent to C = C - A |
| \*= | Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand. | C \*= A is equivalent to C = C \* A |
| /= | Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand. | C /= A is equivalent to C = C / A |
| %= | Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand. | C %= A is equivalent to C = C % A |
| <<= | Left shift AND assignment operator. | C <<= 2 is same as C = C << 2 |
| >>= | Right shift AND assignment operator. | C >>= 2 is same as C = C >> 2 |
| &= | Bitwise AND assignment operator. | C &= 2 is same as C = C & 2 |
| ^= | Bitwise exclusive OR and assignment operator. | C ^= 2 is same as C = C ^ 2 |
| |= | Bitwise inclusive OR and assignment operator. | C |= 2 is same as C = C | 2 |

Misc Operators ↦ sizeof & ternary

Besides the operators discussed above, there are a few other important operators including **sizeof** and **? :** supported by the C Language.

[Show Examples](https://www.tutorialspoint.com/cprogramming/c_sizeof_operator.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| sizeof() | Returns the size of a variable. | sizeof(a), where a is integer, will return 4. |
| & | Returns the address of a variable. | &a; returns the actual address of the variable. |
| \* | Pointer to a variable. | \*a; |
| ? : | Conditional Expression. | If Condition is true ? then value X : otherwise value Y |

Operators Precedence in C

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, x = 7 + 3 \* 2; here, x is assigned 13, not 20 because operator \* has a higher precedence than +, so it first gets multiplied with 3\*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

[Show Examples](https://www.tutorialspoint.com/cprogramming/c_operators_precedence.htm)

|  |  |  |
| --- | --- | --- |
| **Category** | **Operator** | **Associativity** |
| Postfix | () [] -> . ++ - - | Left to right |
| Unary | + - ! ~ ++ - - (type)\* & sizeof | Right to left |
| Multiplicative | \* / % | Left to right |
| Additive | + - | Left to right |
| Shift | << >> | Left to right |
| Relational | < <= > >= | Left to right |
| Equality | == != | Left to right |
| Bitwise AND | & | Left to right |
| Bitwise XOR | ^ | Left to right |
| Bitwise OR | | | Left to right |
| Logical AND | && | Left to right |
| Logical OR | || | Left to right |
| Conditional | ?: | Right to left |
| Assignment | = += -= \*= /= %=>>= <<= &= ^= |= | Right to left |
| Comma | , | Left to right |

**Topic № 14.**

When we say **Input**, it means to feed some data into a program. An input can be given in the form of a file or from the command line. C programming provides a set of built-in functions to read the given input and feed it to the program as per requirement.

When we say **Output**, it means to display some data on screen, printer, or in any file. C programming provides a set of built-in functions to output the data on the computer screen as well as to save it in text or binary files.

## The Standard Files

C programming treats all the devices as files. So devices such as the display are addressed in the same way as files and the following three files are automatically opened when a program executes to provide access to the keyboard and screen.

|  |  |  |
| --- | --- | --- |
| **Standard File** | **File Pointer** | **Device** |
| Standard input | stdin | Keyboard |
| Standard output | stdout | Screen |
| Standard error | stderr | Your screen |

The file pointers are the means to access the file for reading and writing purpose. This section explains how to read values from the screen and how to print the result on the screen.

## The getchar() and putchar() Functions

The **int getchar(void)** function reads the next available character from the screen and returns it as an integer. This function reads only single character at a time. You can use this method in the loop in case you want to read more than one character from the screen.

The **int putchar(int c)** function puts the passed character on the screen and returns the same character. This function puts only single character at a time. You can use this method in the loop in case you want to display more than one character on the screen. Check the following example −

#include <stdio.h>

int main( ) {

int c;

printf( "Enter a value :");

c = getchar( );

printf( "\nYou entered: ");

putchar( c );

return 0;

}

When the above code is compiled and executed, it waits for you to input some text. When you enter a text and press enter, then the program proceeds and reads only a single character and displays it as follows −

$./a.out

**Enter a value :** this is test

**You entered:** t

## The gets() and puts() Functions

The **char \*gets(char \*s)** function reads a line from **stdin** into the buffer pointed to by **s** until either a terminating newline or EOF (End of File).

The **int puts(const char \*s)** function writes the string 's' and 'a' trailing newline to **stdout**.

**NOTE:** Though it has been deprecated to use gets() function, Instead of using gets, you want to use [fgets()](https://www.tutorialspoint.com/c_standard_library/c_function_fgets.htm).

#include <stdio.h>

int main( ) {

char str[100];

printf( "Enter a value :");

gets( str );

printf( "\nYou entered: ");

puts( str );

return 0;

}

When the above code is compiled and executed, it waits for you to input some text. When you enter a text and press enter, then the program proceeds and reads the complete line till end, and displays it as follows −

$./a.out

**Enter a value :** this is test

**You entered:** this is test

## The scanf() and printf() Functions

The **int scanf(const char \*format, ...)** function reads the input from the standard input stream **stdin** and scans that input according to the **format** provided.

The **int printf(const char \*format, ...)** function writes the output to the standard output stream **stdout** and produces the output according to the format provided.

The **format** can be a simple constant string, but you can specify %s, %d, %c, %f, etc., to print or read strings, integer, character or float respectively. There are many other formatting options available which can be used based on requirements. Let us now proceed with a simple example to understand the concepts better −

#include <stdio.h>

int main( ) {

char str[100];

int i;

printf( "Enter a value :");

scanf("%s %d", str, &i);

printf( "\nYou entered: %s %d ", str, i);

return 0;

}

When the above code is compiled and executed, it waits for you to input some text. When you enter a text and press enter, then program proceeds and reads the input and displays it as follows −

$./a.out

**Enter a value :** seven 7

**You entered:** seven 7

Here, it should be noted that scanf() expects input in the same format as you provided %s and %d, which means you have to provide valid inputs like "string integer". If you provide "string string" or "integer integer", then it will be assumed as wrong input. Secondly, while reading a string, scanf() stops reading as soon as it encounters a space, so "this is test" are three strings for scanf().

**Topic № 15.**

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators −

* Arithmetic Operators
* Relational Operators
* Logical Operators
* Bitwise Operators
* Assignment Operators
* Misc Operators

We will, in this chapter, look into the way each operator works.

## Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language. Assume variable **A** holds 10 and variable **B** holds 20 then −

[Show Examples](https://www.tutorialspoint.com/cprogramming/c_arithmetic_operators.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + | Adds two operands. | A + B = 30 |
| − | Subtracts second operand from the first. | A − B = -10 |
| \* | Multiplies both operands. | A \* B = 200 |
| / | Divides numerator by de-numerator. | B / A = 2 |
| % | Modulus Operator and remainder of after an integer division. | B % A = 0 |
| ++ | Increment operator increases the integer value by one. | A++ = 11 |
| -- | Decrement operator decreases the integer value by one. | A-- = 9 |

## Relational Operators

The following table shows all the relational operators supported by C. Assume variable **A** holds 10 and variable **B** holds 20 then −

[Show Examples](https://www.tutorialspoint.com/cprogramming/c_relational_operators.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| == | Checks if the values of two operands are equal or not. If yes, then the condition becomes true. | (A == B) is not true. |
| != | Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true. | (A != B) is true. |
| > | Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true. | (A > B) is not true. |
| < | Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true. | (A < B) is true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true. | (A >= B) is not true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true. | (A <= B) is true. |

## Logical Operators

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then −

[Show Examples](https://www.tutorialspoint.com/cprogramming/c_logical_operators.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| && | Called Logical AND operator. If both the operands are non-zero, then the condition becomes true. | (A && B) is false. |
| || | Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true. | (A || B) is true. |
| ! | Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false. | !(A && B) is true. |

## Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows −

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **p** | **q** | **p & q** | **p | q** | **p ^ q** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |

Assume A = 60 and B = 13 in binary format, they will be as follows −

A = 0011 1100

B = 0000 1101

-----------------

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

The following table lists the bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then −

[Show Examples](https://www.tutorialspoint.com/cprogramming/c_bitwise_operators.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| & | Binary AND Operator copies a bit to the result if it exists in both operands. | (A & B) = 12, i.e., 0000 1100 |
| | | Binary OR Operator copies a bit if it exists in either operand. | (A | B) = 61, i.e., 0011 1101 |
| ^ | Binary XOR Operator copies the bit if it is set in one operand but not both. | (A ^ B) = 49, i.e., 0011 0001 |
| ~ | Binary One's Complement Operator is unary and has the effect of 'flipping' bits. | (~A ) = ~(60), i.e,. -0111101 |
| << | Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand. | A << 2 = 240 i.e., 1111 0000 |
| >> | Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. | A >> 2 = 15 i.e., 0000 1111 |

## Assignment Operators

The following table lists the assignment operators supported by the C language −

[Show Examples](https://www.tutorialspoint.com/cprogramming/c_assignment_operators.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Simple assignment operator. Assigns values from right side operands to left side operand | C = A + B will assign the value of A + B to C |
| += | Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand. | C += A is equivalent to C = C + A |
| -= | Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand. | C -= A is equivalent to C = C - A |
| \*= | Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand. | C \*= A is equivalent to C = C \* A |
| /= | Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand. | C /= A is equivalent to C = C / A |
| %= | Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand. | C %= A is equivalent to C = C % A |
| <<= | Left shift AND assignment operator. | C <<= 2 is same as C = C << 2 |
| >>= | Right shift AND assignment operator. | C >>= 2 is same as C = C >> 2 |
| &= | Bitwise AND assignment operator. | C &= 2 is same as C = C & 2 |
| ^= | Bitwise exclusive OR and assignment operator. | C ^= 2 is same as C = C ^ 2 |
| |= | Bitwise inclusive OR and assignment operator. | C |= 2 is same as C = C | 2 |

## Misc Operators ↦ sizeof & ternary

Besides the operators discussed above, there are a few other important operators including **sizeof** and **? :** supported by the C Language.

[Show Examples](https://www.tutorialspoint.com/cprogramming/c_sizeof_operator.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| sizeof() | Returns the size of a variable. | sizeof(a), where a is integer, will return 4. |
| & | Returns the address of a variable. | &a; returns the actual address of the variable. |
| \* | Pointer to a variable. | \*a; |
| ? : | Conditional Expression. | If Condition is true ? then value X : otherwise value Y |

## Operators Precedence in C

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, x = 7 + 3 \* 2; here, x is assigned 13, not 20 because operator \* has a higher precedence than +, so it first gets multiplied with 3\*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

[Show Examples](https://www.tutorialspoint.com/cprogramming/c_operators_precedence.htm)

|  |  |  |
| --- | --- | --- |
| **Category** | **Operator** | **Associativity** |
| Postfix | () [] -> . ++ - - | Left to right |
| Unary | + - ! ~ ++ - - (type)\* & sizeof | Right to left |
| Multiplicative | \* / % | Left to right |
| Additive | + - | Left to right |
| Shift | << >> | Left to right |
| Relational | < <= > >= | Left to right |
| Equality | == != | Left to right |
| Bitwise AND | & | Left to right |
| Bitwise XOR | ^ | Left to right |
| Bitwise OR | | | Left to right |
| Logical AND | && | Left to right |
| Logical OR | || | Left to right |
| Conditional | ?: | Right to left |
| Assignment | = += -= \*= /= %=>>= <<= &= ^= |= | Right to left |
| Comma |  |  |

**Topic № 16.**

Decision making structures require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Show below is the general form of a typical decision making structure found in most of the programming languages −



C programming language assumes any **non-zero** and **non-null**values as **true**, and if it is either **zero** or **null**, then it is assumed as **false** value.

C programming language provides the following types of decision making statements.

|  |  |
| --- | --- |
| **Sr.No.** | **Statement & Description** |
| 1 | [**if statement**](https://www.tutorialspoint.com/cprogramming/if_statement_in_c.htm)  An **if statement** consists of a boolean expression followed by one or more statements. |
| 2 | [**if...else statement**](https://www.tutorialspoint.com/cprogramming/if_else_statement_in_c.htm)  An **if statement** can be followed by an optional **else statement**, which executes when the Boolean expression is false. |
| 3 | [**nested if statements**](https://www.tutorialspoint.com/cprogramming/nested_if_statements_in_c.htm)  You can use one **if** or **else if** statement inside another **if** or **else if** statement(s). |
| 4 | [**switch statement**](https://www.tutorialspoint.com/cprogramming/switch_statement_in_c.htm)  A **switch** statement allows a variable to be tested for equality against a list of values. |
| 5 | [**nested switch statements**](https://www.tutorialspoint.com/cprogramming/nested_switch_statements_in_c.htm)  You can use one **switch** statement inside another **switch** statement(s). |

## The ? : Operator

We have covered **conditional operator ? :** in the previous chapter which can be used to replace **if...else** statements. It has the following general form −

Exp1 ? Exp2 : Exp3;

Where Exp1, Exp2, and Exp3 are expressions. Notice the use and placement of the colon.

The value of a ? expression is determined like this −

* Exp1 is evaluated. If it is true, then Exp2 is evaluated and becomes the value of the entire ? expression.
* If Exp1 is false, then Exp3 is evaluated and its value becomes the value of the expression.

**IF statement**

An **if** statement consists of a Boolean expression followed by one or more statements.

## Syntax

The syntax of an 'if' statement in C programming language is −

if(boolean\_expression) {

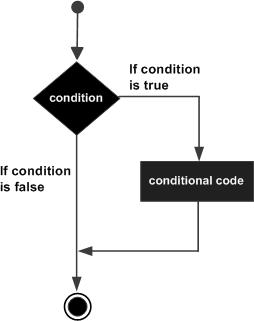
/\* statement(s) will execute if the boolean expression is true \*/

}

If the Boolean expression evaluates to **true**, then the block of code inside the 'if' statement will be executed. If the Boolean expression evaluates to **false**, then the first set of code after the end of the 'if' statement (after the closing curly brace) will be executed.

C programming language assumes any **non-zero** and **non-null**values as **true** and if it is either **zero** or **null**, then it is assumed as **false** value.

## Flow Diagram



## Example

#include <stdio.h>

int main () {

/\* local variable definition \*/

int a = 10;

/\* check the boolean condition using if statement \*/

if( a < 20 ) {

/\* if condition is true then print the following \*/

printf("a is less than 20\n" );

}

printf("value of a is : %d\n", a);

return 0;

}

When the above code is compiled and executed, it produces the following result −

a is less than 20;

value of a is : 10

# C - if...else statement

An **if** statement can be followed by an optional **else** statement, which executes when the Boolean expression is false.

## Syntax

The syntax of an **if...else** statement in C programming language is −

if(boolean\_expression) {

/\* statement(s) will execute if the boolean expression is true \*/

} else {

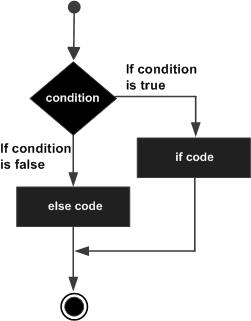
/\* statement(s) will execute if the boolean expression is false \*/

}

If the Boolean expression evaluates to **true**, then the **if block**will be executed, otherwise, the **else block** will be executed.

C programming language assumes any **non-zero** and **non-null**values as **true**, and if it is either **zero** or **null**, then it is assumed as **false** value.

## Flow Diagram



## Example

#include <stdio.h>

int main () {

/\* local variable definition \*/

int a = 100;

/\* check the boolean condition \*/

if( a < 20 ) {

/\* if condition is true then print the following \*/

printf("a is less than 20\n" );

} else {

/\* if condition is false then print the following \*/

printf("a is not less than 20\n" );

}

printf("value of a is : %d\n", a);

return 0;

}

When the above code is compiled and executed, it produces the following result −

a is not less than 20;

value of a is : 100

## If...else if...else Statement

An **if** statement can be followed by an optional **else if...else**statement, which is very useful to test various conditions using single if...else if statement.

When using if...else if..else statements, there are few points to keep in mind −

* An if can have zero or one else's and it must come after any else if's.
* An if can have zero to many else if's and they must come before the else.
* Once an else if succeeds, none of the remaining else if's or else's will be tested.

### Syntax

The syntax of an **if...else if...else** statement in C programming language is −

if(boolean\_expression 1) {

/\* Executes when the boolean expression 1 is true \*/

} else if( boolean\_expression 2) {

/\* Executes when the boolean expression 2 is true \*/

} else if( boolean\_expression 3) {

/\* Executes when the boolean expression 3 is true \*/

} else {

/\* executes when the none of the above condition is true \*/

}

### Example

#include <stdio.h>

int main () {

/\* local variable definition \*/

int a = 100;

/\* check the boolean condition \*/

if( a == 10 ) {

/\* if condition is true then print the following \*/

printf("Value of a is 10\n" );

} else if( a == 20 ) {

/\* if else if condition is true \*/

printf("Value of a is 20\n" );

} else if( a == 30 ) {

/\* if else if condition is true \*/

printf("Value of a is 30\n" );

} else {

/\* if none of the conditions is true \*/

printf("None of the values is matching\n" );

}

printf("Exact value of a is: %d\n", a );

return 0;

}

When the above code is compiled and executed, it produces the following result −

None of the values is matching

Exact value of a is: 100

# C - switch statement

A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each **switch case**.

## Syntax

The syntax for a **switch** statement in C programming language is as follows −

switch(expression) {

case constant-expression :

statement(s);

break; /\* optional \*/

case constant-expression :

statement(s);

break; /\* optional \*/

/\* you can have any number of case statements \*/

default : /\* Optional \*/

statement(s);

}

The following rules apply to a **switch** statement −

* The **expression** used in a **switch** statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
* You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
* The **constant-expression** for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
* When the variable being switched on is equal to a case, the statements following that case will execute until a **break** statement is reached.
* When a **break** statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
* Not every case needs to contain a **break**. If no **break**appears, the flow of control will *fall through* to subsequent cases until a break is reached.
* A **switch** statement can have an optional **default** case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No **break** is needed in the default case.

## Flow Diagram



## Example

#include <stdio.h>

int main () {

/\* local variable definition \*/

char grade = 'B';

switch(grade) {

case 'A' :

printf("Excellent!\n" );

break;

case 'B' :

case 'C' :

printf("Well done\n" );

break;

case 'D' :

printf("You passed\n" );

break;

case 'F' :

printf("Better try again\n" );

break;

default :

printf("Invalid grade\n" );

}

printf("Your grade is %c\n", grade );

return 0;

}

When the above code is compiled and executed, it produces the following result −

Well done

Your grade is B

**Topic № 17.**

You may encounter situations, when a block of code needs to be executed several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times. Given below is the general form of a loop statement in most of the programming languages −



C programming language provides the following types of loops to handle looping requirements.

|  |  |
| --- | --- |
| **Sr.No.** | **Loop Type & Description** |
| 1 | [**while loop**](https://www.tutorialspoint.com/cprogramming/c_while_loop.htm)  Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body. |
| 2 | [**for loop**](https://www.tutorialspoint.com/cprogramming/c_for_loop.htm)  Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable. |
| 3 | [**do...while loop**](https://www.tutorialspoint.com/cprogramming/c_do_while_loop.htm)  It is more like a while statement, except that it tests the condition at the end of the loop body. |
| 4 | [**nested loops**](https://www.tutorialspoint.com/cprogramming/c_nested_loops.htm)  You can use one or more loops inside any other while, for, or do..while loop. |

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

C supports the following control statements.

|  |  |
| --- | --- |
| **Sr.No.** | **Control Statement & Description** |
| 1 | [**break statement**](https://www.tutorialspoint.com/cprogramming/c_break_statement.htm)  Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch. |
| 2 | [**continue statement**](https://www.tutorialspoint.com/cprogramming/c_continue_statement.htm)  Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating. |
| 3 | [**goto statement**](https://www.tutorialspoint.com/cprogramming/c_goto_statement.htm)  Transfers control to the labeled statement. |

The Infinite Loop

A loop becomes an infinite loop if a condition never becomes false. The **for** loop is traditionally used for this purpose. Since none of the three expressions that form the 'for' loop are required, you can make an endless loop by leaving the conditional expression empty.

#include <stdio.h>

int main () {

for( ; ; ) {

printf("This loop will run forever.\n");

}

return 0;

}

When the conditional expression is absent, it is assumed to be true. You may have an initialization and increment expression, but C programmers more commonly use the for(;;) construct to signify an infinite loop.

**NOTE** − You can terminate an infinite loop by pressing Ctrl + C keys.

A **while** loop in C programming repeatedly executes a target statement as long as a given condition is true.

Syntax

The syntax of a **while** loop in C programming language is −

while(condition) {

statement(s);

}

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any nonzero value. The loop iterates while the condition is true.

When the condition becomes false, the program control passes to the line immediately following the loop.

Flow Diagram



Here, the key point to note is that a while loop might not execute at all. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Example

[Live Demo](http://tpcg.io/p4bawJ)

#include <stdio.h>

int main () {

/\* local variable definition \*/

int a = 10;

/\* while loop execution \*/

while( a < 20 ) {

printf("value of a: %d\n", a);

a++;

}

return 0;

}

When the above code is compiled and executed, it produces the following result −

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15

value of a: 16

value of a: 17

value of a: 18

value of a: 19

A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax

The syntax of a **for** loop in C programming language is −

for ( init; condition; increment ) {

statement(s);

}

Here is the flow of control in a 'for' loop −

* The **init** step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
* Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and the flow of control jumps to the next statement just after the 'for' loop.
* After the body of the 'for' loop executes, the flow of control jumps back up to the **increment** statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
* The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the 'for' loop terminates.

Flow Diagram



Example

[Live Demo](http://tpcg.io/Li6I3H)

#include <stdio.h>

int main () {

int a;

/\* for loop execution \*/

for( a = 10; a < 20; a = a + 1 ){

printf("value of a: %d\n", a);

}

return 0;

}

When the above code is compiled and executed, it produces the following result −

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15

value of a: 16

value of a: 17

value of a: 18

value of a: 19

Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the **do...while** loop in C programming checks its condition at the bottom of the loop.

A **do...while** loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.

Syntax

The syntax of a **do...while** loop in C programming language is −

do {

statement(s);

} while( condition );

Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop executes once before the condition is tested.

If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop executes again. This process repeats until the given condition becomes false.

Flow Diagram



Example

[Live Demo](http://tpcg.io/pVAh6H)

#include <stdio.h>

int main () {

/\* local variable definition \*/

int a = 10;

/\* do loop execution \*/

do {

printf("value of a: %d\n", a);

a = a + 1;

}while( a < 20 );

return 0;

}

When the above code is compiled and executed, it produces the following result −

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15

value of a: 16

value of a: 17

value of a: 18

value of a: 19

**Topic № 18.**

# ****LOGICAL OPERATORS IN C:****

* These operators are used to perform logical operations on the given expressions.
* There are 3 logical operators in C language. They are, logical AND (&&), logical OR (||) and logical NOT (!).

|  |  |
| --- | --- |
| **Operators** | **Example/Description** |
| && (logical AND) | (x>5)&&(y<5) It returns true when both conditions are true |
| || (logical OR) | (x>=10)||(y>=10)  It returns true when at-least one of the condition is true |
| ! (logical NOT) | !((x>5)&&(y<5))  It reverses the state of the operand “((x>5) && (y<5))”  If “((x>5) && (y<5))” is true, logical NOT operator makes it false |

# ****EXAMPLE PROGRAM FOR LOGICAL OPERATORS IN C:****

#include <stdio.h>

int main()

{

   int m=40,n=20;

   int o=20,p=30;

   if (m>n && m !=0)

   {

      printf("&& Operator : Both conditions are true\n");

   }

   if (o>p || p!=20)

   {

      printf("|| Operator : Only one condition is true\n");

   }

   if (!(m>n && m !=0))

   {

      printf("! Operator : Both conditions are true\n");

   }

   else

   {

      printf("! Operator : Both conditions are true. " \

      "But, status is inverted as false\n");

   }

}

# ****OUTPUT:****

|  |
| --- |
| && Operator : Both conditions are true || Operator : Only one condition is true ! Operator : Both conditions are true. But, status is inverted as false |

* In this program, operators (&&, || and !) are used to perform logical operations on the given expressions.
* **&& operator** – “if clause” becomes true only when both conditions (m>n and m! =0) is true. Else, it becomes false.
* **|| Operator** – “if clause” becomes true when any one of the condition (o>p || p!=20) is true. It becomes false when none of the condition is true.
* **! Operator**  – It is used to reverses the state of the operand.
* If the conditions (m>n && m!=0) is true, true (1) is returned. This value is inverted by “!” operator.
* So, “! (m>n and m! =0)” returns false (0).

The following examples illustrate the logical operators:

int w, x, y, z;

if ( x < y && y < z )

printf( "x is less than z\n" );

In this example, the **printf** function is called to print a message if x is less than y and y is less than z. If x is greater than y, the second operand (y < z) is not evaluated and nothing is printed. Note that this could cause problems in cases where the second operand has side effects that are being relied on for some other reason.

printf( "%d" , (x == w || x == y || x == z) );

In this example, if x is equal to either w, y, or z, the second argument to the **printf** function evaluates to true and the value 1 is printed. Otherwise, it evaluates to false and the value 0 is printed. As soon as one of the conditions evaluates to true, evaluation ceases.

// expre\_Logical\_AND\_Operator.cpp

// compile with: /EHsc

// Demonstrate logical AND

#include <iostream>

using namespace std;

int main() {

int a = 5, b = 10, c = 15;

cout << boolalpha

<< "The true expression "

<< "a < b && b < c yields "

<< (a < b && b < c) << endl

<< "The false expression "

<< "a > b && b < c yields "

<< (a > b && b < c) << endl;

}

**Topic № 19.**

**Palindrome number in C**

Palindrome number in C language: A palindrome number is a which remains the same on reversal. For example, some palindrome numbers are 8, 121, 212, 12321, -454. To check whether a number is a palindrome or not first we reverse it and then compare the number obtained with the original number, if both are same then the number is palindrome otherwise not.

## Check palindrome number algorithm

To test if a number is palindrome or not, do the following steps:  
1. Get the number from a user.  
2. Reverse it.  
3. Compare it with the number entered by the user.  
4. If both are the same then print palindrome number else print not a palindrome number.

1. #include <stdio.h>
3. int main()
4. {
5. int n, reverse = 0, t;
7. printf("Enter a number to check if it is a palindrome or not**\n**");
8. scanf("%d", &n);
10. t = n;
12. while (t != 0)
13. {
14. reverse = reverse \* 10;
15. reverse = reverse + t%10;
16. t = t/10;
17. }
19. if (n == reverse)
20. printf("%d is a palindrome number.**\n**", n);
21. else
22. printf("%d isn't a palindrome number.**\n**", n);
24. return 0;
25. }

A positive integer is called an Armstrong number of order n if

abcd... = an + bn + cn + dn + ...

In case of an Armstrong number of 3 digits, the sum of cubes of each digits is equal to the number itself. For example:

153 = 1\*1\*1 + 5\*5\*5 + 3\*3\*3 // 153 is an Armstrong number.

## Example #1: Check Armstrong Number of three digits

1. #include <stdio.h>
2. int main()
3. {
4. int number, originalNumber, remainder, result = 0;
5. printf("Enter a three digit integer: ");
6. scanf("%d", &number);
7. originalNumber = number;
8. while (originalNumber != 0)
9. {
10. remainder = originalNumber%10;
11. result += remainder\*remainder\*remainder;
12. originalNumber /= 10;
13. }
14. if(result == number)
15. printf("%d is an Armstrong number.",number);
16. else
17. printf("%d is not an Armstrong number.",number);
18. return 0;
19. }

**Output**

Enter a three digit integer: 371

371 is an Armstrong number.

## Example #2: Check Armstrong Number of n digits

1. #include <stdio.h>
2. #include <math.h>
3. int main()
4. {
5. int number, originalNumber, remainder, result = 0, n = 0 ;
6. printf("Enter an integer: ");
7. scanf("%d", &number);
8. originalNumber = number;
10. while (originalNumber != 0)
11. {
12. originalNumber /= 10;
13. ++n;
14. }
15. originalNumber = number;
16. while (originalNumber != 0)
17. {
18. remainder = originalNumber%10;
19. result += pow(remainder, n);
20. originalNumber /= 10;
21. }
22. if(result == number)
23. printf("%d is an Armstrong number.", number);
24. else
25. printf("%d is not an Armstrong number.", number);
26. return 0;
27. }

**Topic № 20.**

Euclidean algorithms (Basic and Extended)

GCD of two numbers is the largest number that divides both of them. A simple way to find GCD is to factorize both numbers and multiply common factors.

**Basic Euclidean Algorithm for GCD**  
The algorithm is based on below facts.

* If we subtract smaller number from larger (we reduce larger number), GCD doesn’t change. So if we keep subtracting repeatedly the larger of two, we end up with GCD.
* Now instead of subtraction, if we divide smaller number, the algorithm stops when we find remainder 0.

Below is a recursive function to evaluate gcd using Euclid’s algorithm.

/\* GCD of two numbers using Euclid's algorithm\*/

#include <stdio.h>

void main()

{

    int m, n; /\* given numbers \*/

    clrscr();

    printf("Enter-two integer numbers: ");

    scanf ("%d %d", &m, &n);

    while (n > 0)

    {

        int r = m % n;

        m = n;

        n = r;

    }

    printf ("GCD = %d \n",m);

    getch();

}

**Prime number** is a number that is greater than 1 and divided by 1 or itself. In other words, prime numbers can't be divided by other numbers than itself or 1. For example 2, 3, 5, 7, 11, 13, 17, 19, 23.... are the prime numbers.

#### Note: Zero (0) and 1 are not considered as prime numbers. Two (2) is the only one even prime number because all the numbers can be divided by 2.

1. #include<stdio.h>
2. **int** main(){
3. **int** n,i,m=0,flag=0;
4. printf("Enter the number to check prime:");
5. scanf("%d",&n);
6. m=n/2;
7. **for**(i=2;i<=m;i++)
8. {
9. **if**(n%i==0)
10. {
11. printf("Number is not prime");
12. flag=1;
13. **break**;
14. }
15. }
16. **if**(flag==0)
17. printf("Number is prime");
18. **return** 0;
19. }

**Topic № 21.**

The series basically represents sums of natural numbers. First term is sum of single number. Second term is sum of two numbers, and so on. A simple solution is to add the first n natural numbers.

/ CPP program to find n-th term of

// series 1, 3, 6, 10, 15, 21...

#include <iostream>

using namespace std;

// Function to find the nth term of series

int term(int n)

{

    // Loop to add numbers

    int ans = 0;

    for (int i = 1; i <= n; i++)

        ans += i;

    return ans;

}

// Driver code

int main()

{

    int n = 4;

    cout << term(n) ;

    return 0;

}

// CPP program to find the n-th

// term in series 1 3 6 10 ...

#include <bits/stdc++.h>

using namespace std;

// Function to find nth term

int term(int n)

{

    return n \* (n + 1) / 2;

}

// Driver code

int main()

{

    int n = 4;

    cout << term(n);

    return 0;

}

#include<iostream>

using namespace std;

int main()

{

int i,n;

float sum=0;

cout<<"Enter the value of n ";

cin>>n;

for(i=1;i<=n;i++)

sum += 1.0/i;

cout<<"Sum : "<<sum;

return 0;

}

**Topic № 22.**

Fibonacci numbers

The Fibonacci sequence is named after Italian mathematician Leonardo of Pisa, known as Fibonacci:

The Fibonacci numbers *fn* = f(*n*) are the numbers characterized by the fact that every number after the first two is the sum of the two preceding ones. They are defined with the next recurrent relation:



So *f*0 = 0, *f*1 = 1, *fn* = *fn*-1 + *fn*-2.

The Fibonacci sequence has the form

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …

Example. Fill integer array *fib* with Fibonacci numbers (fib[*i*] = *fi*):

#include <stdio.h>

int i, n, fib[47];

int main(void)

{

scanf("%d",&n);

fib[0] = 0; fib[1] = 1;

for(i = 2; i <= n; i++)

fib[i] = fib[i-1] + fib[i-2];

printf("%d\n",fib[n]);

return 0;

}



The biggest Fibonacci number that fits into int type is

*f*46 = 1836311903

The biggest Fibonacci number that fits into long long type is

*f*92 = 7540113804746346429

If you want to find Fibonacci number *fn* for *n* > 92, use BigInteger type.

Example. Find f(*n*) – the *n*-th Fibonacci number with recursion:

#include <stdio.h>

int n;

int fib(int n)

{

if (n == 0) return 0;

if (n == 1) return 1;

return fib(n-1) + fib(n - 2);

}

int main(void)

{

scanf("%d",&n);

printf("%d\n",fib(n));

return 0;

}

Program for Tower of Hanoi

Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:  
1) Only one disk can be moved at a time.  
2) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.  
3) No disk may be placed on top of a smaller disk.

Take an example for 2 disks :

Let rod 1 = 'A', rod 2 = 'B', rod 3 = 'C'.

Step 1 : Shift first disk from 'A' to 'B'.

Step 2 : Shift second disk from 'A' to 'C'.

Step 3 : Shift first disk from 'B' to 'C'.

The pattern here is :

Shift 'n-1' disks from 'A' to 'B'.

Shift last disk from 'A' to 'C'.

Shift 'n-1' disks from 'B' to 'C'.

Input : 3

Output : Disk 1 moved from A to C

Disk 2 moved from A to B

Disk 1 moved from C to B

Disk 3 moved from A to C

Disk 1 moved from B to A

Disk 2 moved from B to C

Disk 1 moved from A to C

*For n disks, total 2n – 1 moves are required.*

eg: For 4 disks 24 – 1 = 15 moves are required.

*For n disks, total 2n-1 function calls are made.*

// C++ recursive function to

// solve tower of hanoi puzzle

#include <bits/stdc++.h>

using namespace std;

void towerOfHanoi(int n, char from\_rod,

                    char to\_rod, char aux\_rod)

{

    if (n == 1)

    {

        cout << "Move disk 1 from rod " << from\_rod <<

                            " to rod " << to\_rod<<endl;

        return;

    }

    towerOfHanoi(n - 1, from\_rod, aux\_rod, to\_rod);

    cout << "Move disk " << n << " from rod " << from\_rod <<

                                " to rod " << to\_rod << endl;

    towerOfHanoi(n - 1, aux\_rod, to\_rod, from\_rod);

}

// Driver code

int main()

{

    int n = 4; // Number of disks

    towerOfHanoi(n, 'A', 'C', 'B'); // A, B and C are names of rods

    return 0;

}

Move disk 1 from rod A to rod B

Move disk 2 from rod A to rod C

Move disk 1 from rod B to rod C

Move disk 3 from rod A to rod B

Move disk 1 from rod C to rod A

Move disk 2 from rod C to rod B

Move disk 1 from rod A to rod B

Move disk 4 from rod A to rod C

Move disk 1 from rod B to rod C

Move disk 2 from rod B to rod A

Move disk 1 from rod C to rod A

Move disk 3 from rod B to rod C

Move disk 1 from rod A to rod B

Move disk 2 from rod A to rod C

Move disk 1 from rod B to rod C

**Topic № 23.**

There are hundreds of different programming languages, but only a few of them are really popular. Learn about some of the most widely used programming languages and what they are being used for in this video lesson.

**Programming Languages**

A program is a set of instructions that tells a computer what to do in order to come up with a solution to a particular problem. Programs are written in a language that computers can understand, known as programming language. Each programming language has its own syntax, which consists of a set of rules that dictate how words and symbols can be put together to form a program. There are hundreds of different programming languages, each with their own logic and syntax. Only a few of them are really popular, but a programmer can easily use a dozen or more languages during a career.

**C, C++, and C#**

C is a general purpose programming language with a long history. C is used for many different types of software, but it is particularly popular for system software, such as operating systems, device drivers and telecommunications applications. C is widely used because it runs very fast. It can also access a computer system's low level functions; this means it is closer to the hardware than some other programming languages. C has become an official standard of the American National Standards Institute, or ANSI. Many other programming languages borrow syntax from C.

C++ is a high-level programming language that builds on its predecessor C by adding object-oriented features to it. C++ is very versatile and can be used for many different applications. C++ is also an official ANSI standard. C++ is used widely for applications that rely more heavily on a graphical user interface, or GUI. For example, many utility programs and device drivers are written in C, while applications software with many user dialogs are written in C++. The name C++ represents the evolution of the C language because the two plus symbols represent the increment operator in C.

C# is a programming language designed by Microsoft. It combines the functionality of C and C++ with Visual Basic. C# is used on many operating systems, not just Windows. C# is one of the languages used in the Microsoft .NET framework. While C++ supports both object-oriented programming and procedural programming, C# is strictly an object-oriented programming language. The name C# was inspired by musical notation where a sharp indicates that the note should be made a semitone higher in pitch. The sharp symbol also resembles the combination of four plus symbols to suggest that the language is an increment of C++.

The Microsoft .NET framework is a programming infrastructure created by Microsoft. It is also referred to as Visual Studio .NET. It is used for building many different types of applications, such as desktop software applications and web services. The .NET framework makes it possible for different programming languages to work together. However, C# is the language specifically developed to create code for the .NET framework.

Objective-C is also derived from C and has been adapted by Apple to develop applications for the Mac OS and iOS operating systems. Objective-C has therefore become very popular for mobile phone apps. The name Objective-C was chosen to make it clear that this was an object-oriented language with its origins in the C language.

**Perl, Ruby, and Python**

A number of programming languages are referred to as scripting languages. Scripting is used to automate tasks which would otherwise be done manually by a computer user. Scripting typically extends the functionality of existing software, but it is not used to create stand-alone software applications from scratch. Scripting is sometimes referred to as 'glue code' since it is widely used to connect system components.

Scripting languages are typically interpreted languages, which means they do not need to be compiled before they can be executed. This makes it easier to code, test and run a script very quickly. Scripting is widely used for behind-the-scenes jobs, such as file processing and system administration, web scripting and networking programming. Some of the most widely used scripting languages are Perl, Ruby and Python. In comparison to languages such as C and C++, these languages are relatively recent. They are also free and open-source software, meaning that anyone is freely licensed to use, share and change the software.

Java technology is both a programming language and a platform.

### The Java Programming Language

The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

|  |  |
| --- | --- |
|  Simple |  Architecture neutral |
|  Object oriented |  Portable |
|  Distributed |  High performance |
|  Interpreted |  Multithreaded |
|  Robust |  Dynamic |
|  Secure |  |

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called Java bytecodes —the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java bytecode instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed.

### The Java Platform

A platform is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Windows 2000, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

The Java platform has two components:

* The Java Virtual Machine (Java VM)
* The Java Application Programming Interface (Java API)

You've already been introduced to the Java VM. It's the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as packages.

Native code is code that after you compile it, the compiled code runs on a specific hardware platform. As a platform-independent environment, the Java platform can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time byte code compilers can bring performance close to that of native code without threatening portability.