**UNIT – 1**

## 1.1 INTRODUCTION TO PROGRAMMING

A program is a set of instructions that tell the computer to do various things; sometimes the instruction it has to perform depends on what happened when it performed a previous instruction. This section gives an overview of the two main ways in which you can give these instructions, or “commands” as they are usually called. One way uses an interpreter, the other a compiler. As human languages are too difficult for a computer to understand in an unambiguous way, commands are usually written in one or other languages specially designed for the purpose.

**Introduction to Computers:**

**1.2 Computer Systems: Introduction to components of a Computer System:**

A computer is a system made of two major components: hardware and software. The computer hardware is the physical equipment. The software is the collection of programs (instructions) that allow the hardware to do its job.

**COMPUTER SYSTEM**

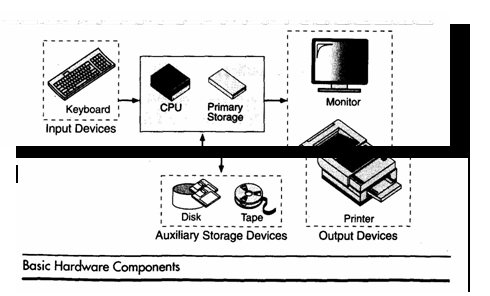
**HARDWARE**

****

**Computer Hardware**

The hardware component of the computer system consists of five parts: input devices, central processing unit (CPU) , primary storage, output devices, and auxiliary storage devices.

The **input device** is usually a keyboard where programs and data are entered into the computers. Examples of other input devices include a mouse, a pen or stylus, a touch screen, or an audio input unit.



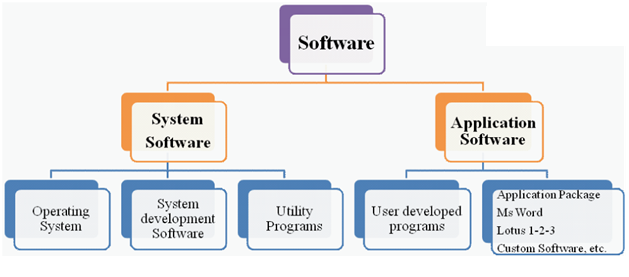
The **central processing unit (CPU**) is responsible for executing instructions such as arithmetic calculations, comparisons among data, and movement of data inside the system. Today’s computers may have one ,two, or more CPUs .**Primary storage** ,also known as main memory, is a place where the programs and data are stored temporarily during processing. The data in primary storage are erased when we turn off a personal computer or when we log off from a time-sharing system.

The **output device** is usually a monitor or a printer to show output. If the output is shown on the monitor, we say we have a **soft copy**. If it is printed on the printer, we say we have a hard copy.

**Auxiliary storage**, also known as **secondary storage** , is used for both input and output.It is the place where the programs and data are stored permanently. When we turn off the computer, or programs and data remain in the secondary storage, ready for the next time we need them.

**Computer Software**

Computer software is divided in to two broad categories: system software and application software .System software manages the computer resources .It provides the interface between the hardware and the users. Application software, on the other hand is directly responsible for helping users solve their problems.



**System Software:**

**System software** consists of programs that manage the hardware resources of a computerand perform required information processing tasks. These programs are divided into three classes: the operating system, system support, and system development.

The **operating system** provides services such as a user interface, file and database access, and interfaces to communication systems such as Internet protocols. The primary purpose of this software is to keep the system operating in an efficient manner while allowing the users access to the system.

**System support software** provides system utilities and other operating services.Examples of system utilities are sort programs and disk format programs. Operating services consists of programs that provide performance statistics for the operational staff and security monitors to protect the system and data.

The last system software category ,**system development software**, includes the language translators that convert programs into machine language for execution ,debugging tools to ensure that the programs are error free and computer –assisted software engineering(CASE) systems.

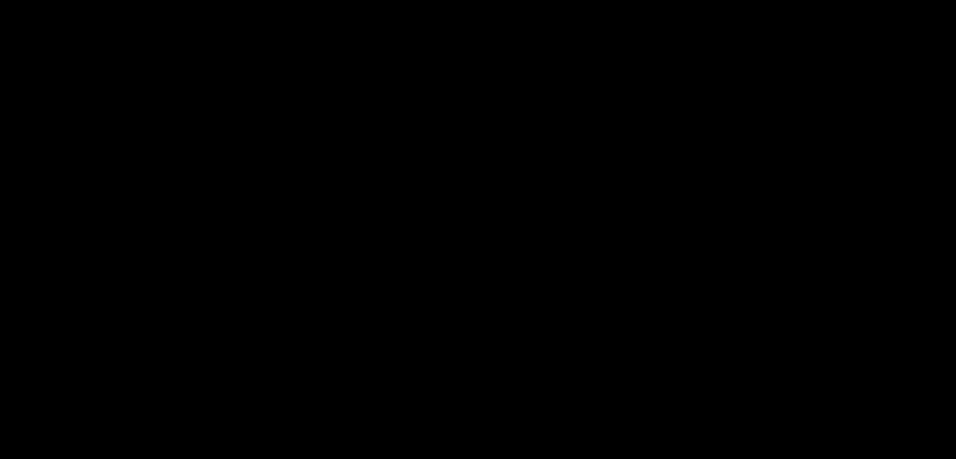
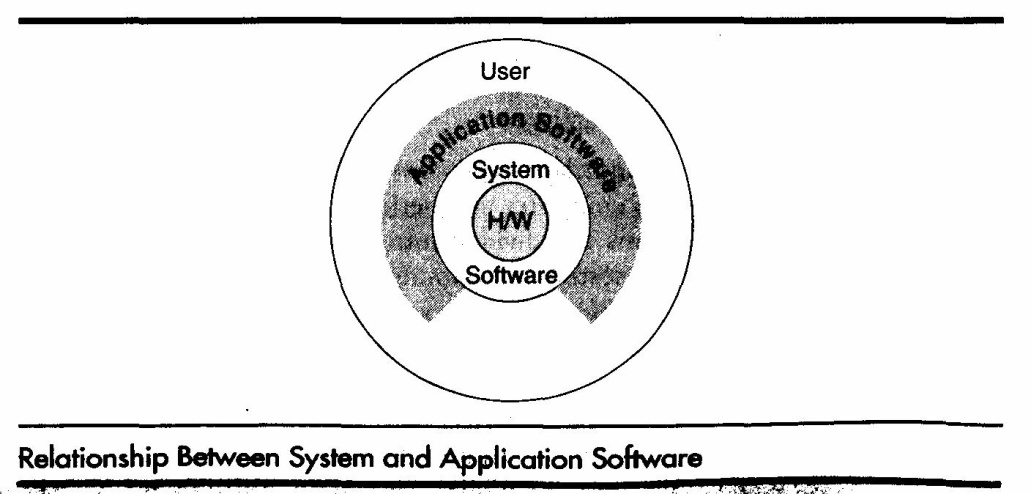
**Application software**

**Application software** is broken in to two classes :general-purpose software andapplication - specific software. **General purpose software** is purchased from a software developer and can be used for more than one application. Examples of general purpose software include word processors, database management systems ,and computer design systems. They are labeled general purpose because they can solve a variety of user computing problems.

**Application –specific software** can be used only for its intended purpose.

A general ledger system used by accountants and a material requirements planning system used by a manufacturing organization are examples of application-specific software. They can be used only for the task for which they were designed they cannot be used for other generalized tasks.

The relation ship between system and application software is shown in fig-2.In this figure, each circle represents an interface point .The inner core is hard ware. The user is represented by the out layer. To work with the system,the typical user uses some form of application software. The application software in turn interacts with the operating system ,which is apart of the system software layer. The system software provides the direct interaction with the hard ware. The opening at the bottom of the figure is the path followed by the user who interacts directly with the operating system when necessary.



# 1.2.1Disk



Alternatively referred to as a **diskette**, a **disk** is a hard or floppy round, flat, and magnetic [platter](https://www.computerhope.com/jargon/p/platter.htm) capable of having information read from and written to it. The most commonly found disks with a computer are the [hard disks](https://www.computerhope.com/jargon/h/harddriv.htm) and floppy disks ([floppy diskette](https://www.computerhope.com/jargon/f/floppydi.htm)) shown in the picture to the right.

When talking about a disk or diskette, you're referring to a floppy diskette or hard disk drive. However, when talking about a [Blu-ray](https://www.computerhope.com/jargon/b/bd.htm), [CD](https://www.computerhope.com/jargon/c/cd.htm), or [DVD](https://www.computerhope.com/jargon/d/dvd.htm), you should use "disc" and not "disk." For example, saying "my movie is on that DVD disc" is the proper usage of "disc." An easy way to remember this is that a "disk" is always a [magnetic storage](https://www.computerhope.com/jargon/m/magnmedi.htm) using magnets, and a "disc" is always an [optical media](https://www.computerhope.com/jargon/o/optidisc.htm) using a lasers or lights.

### Hard drives

A hard-disk drive (HDD) is a non-volatile device used for storage, located inside the computer case. Like the floppy drive, it holds its data on rotating platters with a magnetic upper exterior which are changed or read by electromagnetic tipped arms that move over the disk as it spins.

**1.2.2 Memory**

A memory is just like a human brain. It is used to store data and instructions. Computer memory is the storage space in the computer, where data is to be processed and instructions required for processing are stored. The memory is divided into large number of small parts called cells. Each location or cell has a unique address, which varies from zero to memory size minus one. For example, if the computer has 64k words, then this memory unit has 64 \* 1024 = 65536 memory locations. The address of these locations varies from 0 to 65535.

Memory is primarily of three types −

* Cache Memory
* Primary Memory/Main Memory
* Secondary Memory

**Cache Memory**

Cache memory is a very high speed semiconductor memory which can speed up the CPU. It acts as a buffer between the CPU and the main memory. It is used to hold those parts of data and program which are most frequently used by the CPU. The parts of data and programs are transferred from the disk to cache memory by the operating system, from where the CPU can access them.



**Advantages**

The advantages of cache memory are as follows −

* Cache memory is faster than main memory.
* It consumes less access time as compared to main memory.
* It stores the program that can be executed within a short period of time.
* It stores data for temporary use.

**Disadvantages**

The disadvantages of cache memory are as follows −

* Cache memory has limited capacity.
* It is very expensive.

**Primary Memory (Main Memory)**

Primary memory holds only those data and instructions on which the computer is currently working. It has a limited capacity and data is lost when power is switched off. It is generally made up of semiconductor device. These memories are not as fast as registers. The data and instruction required to be processed resides in the main memory. It is divided into two subcategories RAM and ROM.



**Characteristics of Main Memory**

* These are semiconductor memories.
* It is known as the main memory.
* Usually volatile memory.
* Data is lost in case power is switched off.
* It is the working memory of the computer.
* Faster than secondary memories.
* A computer cannot run without the primary memory.

**Secondary Memory**

This type of memory is also known as external memory or non-volatile. It is slower than the main memory. These are used for storing data/information permanently. CPU directly does not access these memories, instead they are accessed via input-output routines. The contents of secondary memories are first transferred to the main memory, and then the CPU can access it. For example, disk, CD-ROM, DVD, etc.



Characteristics of Secondary Memory

* These are magnetic and optical memories.
* It is known as the backup memory.
* It is a non-volatile memory.
* Data is permanently stored even if power is switched off.
* It is used for storage of data in a computer.
* Computer may run without the secondary memory.
* Slower than primary memories.

**RANDOM ACCESS MEMORY**

RAM (Random Access Memory) is the internal memory of the CPU for storing data, program, and program result. It is a read/write memory which stores data until the machine is working. As soon as the machine is switched off, data is erased.



Access time in RAM is independent of the address, that is, each storage location inside the memory is as easy to reach as other locations and takes the same amount of time. Data in the RAM can be accessed randomly but it is very expensive.

RAM is volatile, i.e. data stored in it is lost when we switch off the computer or if there is a power failure. Hence, a backup Uninterruptible Power System (UPS) is often used with computers. RAM is small, both in terms of its physical size and in the amount of data it can hold.

RAM is of two types −

* Static RAM (SRAM)
* Dynamic RAM (DRAM)

**Static RAM (SRAM)**

The word **static** indicates that the memory retains its contents as long as power is being supplied. However, data is lost when the power gets down due to volatile nature. SRAM chips use a matrix of 6-transistors and no capacitors. Transistors do not require power to prevent leakage, so SRAM need not be refreshed on a regular basis.

There is extra space in the matrix, hence SRAM uses more chips than DRAM for the same amount of storage space, making the manufacturing costs higher. SRAM is thus used as cache memory and has very fast access.

Characteristic of Static RAM

* Long life
* No need to refresh
* Faster
* Used as cache memory
* Large size
* Expensive
* High power consumption

**Dynamic RAM (DRAM)**

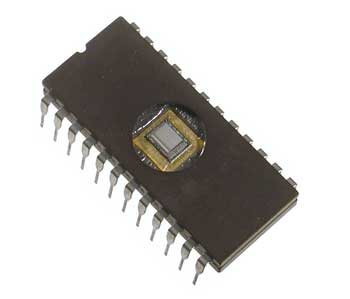
DRAM, unlike SRAM, must be continually **refreshed** in order to maintain the data. This is done by placing the memory on a refresh circuit that rewrites the data several hundred times per second. DRAM is used for most system memory as it is cheap and small. All DRAMs are made up of memory cells, which are composed of one capacitor and one transistor.

Characteristics of Dynamic RAM

* Short data lifetime
* Needs to be refreshed continuously
* Slower as compared to SRAM
* Used as RAM
* Smaller in size
* Less expensive
* Less power consumption

**READ ONLY MEMORY**

ROM stands for **Read Only Memory**. The memory from which we can only read but cannot write on it. This type of memory is non-volatile. The information is stored permanently in such memories during manufacture. A ROM stores such instructions that are required to start a computer. This operation is referred to as **bootstrap**. ROM chips are not only used in the computer but also in other electronic items like washing machine and microwave oven.



Let us now discuss the various types of ROMs and their characteristics.

**MROM (Masked ROM)**

The very first ROMs were hard-wired devices that contained a pre-programmed set of data or instructions. These kind of ROMs are known as masked ROMs, which are inexpensive.

PROM (Programmable Read Only Memory)

PROM is read-only memory that can be modified only once by a user. The user buys a blank PROM and enters the desired contents using a PROM program. Inside the PROM chip, there are small fuses which are burnt open during programming. It can be programmed only once and is not erasable.

**EPROM (Erasable and Programmable Read Only Memory)**

EPROM can be erased by exposing it to ultra-violet light for a duration of up to 40 minutes. Usually, an EPROM eraser achieves this function. During programming, an electrical charge is trapped in an insulated gate region. The charge is retained for more than 10 years because the charge has no leakage path. For erasing this charge, ultra-violet light is passed through a quartz crystal window (lid). This exposure to ultra-violet light dissipates the charge. During normal use, the quartz lid is sealed with a sticker.

**EEPROM (Electrically Erasable and Programmable Read Only Memory)**

EEPROM is programmed and erased electrically. It can be erased and reprogrammed about ten thousand times. Both erasing and programming take about 4 to 10 ms (millisecond). In EEPROM, any location can be selectively erased and programmed. EEPROMs can be erased one byte at a time, rather than erasing the entire chip. Hence, the process of reprogramming is flexible but slow.

**Advantages of ROM**

The advantages of ROM are as follows −

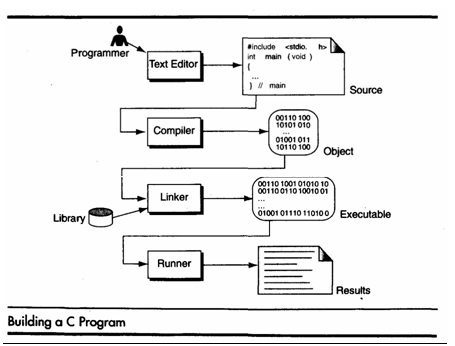
* Non-volatile in nature
* Cannot be accidentally changed
* Cheaper than RAMs
* Easy to test
* More reliable than RAMs
* Static and do not require refreshing
* Contents are always known and can be verified

**1.3 Compilers:**

A compiler is a software program that transforms high-level source code that is written by a developer in a high-level programming language into a low level object code (binary code) in machine language, which can be understood by the processor. The process of converting high-level programming into machine language is known as compilation.

The processor executes object code, which indicates when binary high and low signals are required in the arithmetic logic unit of the processor.

**1.4 Creating and Running Programs:**



Computer hardware understands a program only if it is coded in its machine language. It is the job of the programmer to write and test the program .There are four steps in this process:1.Writing and Editing the program2.Compiliing the program 3.Linking the program with the required library modules 4.Executing the program.

**Writing and Editing Programs**

The software used to write programs is known as a **text editor.** A text editor helps us enter, change, and store character data. Depending on the editor on our system, we could use it to write letters, create reports, or write programs. The main difference between text processing and program writing is that programs are written using lines of code, while most text processing is done with character and lines.

Text editor is a generalized word processor, but it is more often a special editor included with the compiler. Some of the features of the editor are search commands to locate and replace statements, copy and paste commands to copy or move statements from one part of a program to another, and formatting commands that allow us to set tabs to align statements.

After completing a program, we save our file to disk. This file will be input to the compiler; it is known as a **source file.**

**Compiling Programs:**

The code in a source file stored on the disk must be translated into machine language , This is the job of the **compiler.** The c compiler is two separate programs. the **preprocessor** and the **translator.** The preprocessor reads the source code and prepares it for the translator. While preparing the code ,it scans for special instructions known as preprocessor commands. These commands tell the preprocessor to look for special code libraries, make substitutions in the code ,and in other ways prepare the code for translation into machine language. The result of preprocessing is called the translation unit.

After the preprocessor has prepared the code for compilation, the translator does the actual work of converting the program into machine language. The translator reads the translation unit and writes the resulting object module to a file that can then be combined with other precompiled units to form the final program. An object module is the code in machine language. The output of the compiler is machine language code, but it is not ready to run; that is ,it is not executable because it does not have the required C and other functions included.

**Linking Programs:**

A C program is made up of many functions. We write some of these functions, and they are a part of our source program. There are other functions, such as input/output processes and, mathematical library functions, that exist elsewhere and must be attached to our program. The linker assembles all of these functions, ours and systems into our final executable program.

**Executing Programs:**

Once program has been linked, it is ready for execution. To execute a program we use an operating system command, such as run, to load the program into primary memory and execute it. Getting the program into memory is the function of an operating system program known as the loader. It locates the executable program and reads it into memory. When everything is loaded, the program takes control and it begins execution.

In a typical program execution, the reads data for processing , either from the user or from a file. After the program processes the data, it prepares the output. at output can be to the user’s monitor or to a file. When the program has finished its job, it tells the operating system , which then removes the program from memory.

**1.5 Number systems:**

**Number systems Conversions Binary, Decimal, Hexadecimal:**

**Number systems** are the technique to represent numbers in the computer system architecture, every value that you are saving or getting into/from computer memory has a defined number system.

Computer architecture supports following number systems.

* **Binary number system**
* **Octal number system**
* **Decimal number system**
* **Hexadecimal (hex) number system**

**1) Binary Number System**

A Binary number system has only two digits that are **0 and 1**. Every number (value) represents with 0 and 1 in this number system. The base of binary number system is 2, because it has only two digits.

**2) Octal number system**

Octal number system has only eight (8) digits from**0 to 7**. Every number (value) represents with 0,1,2,3,4,5,6 and 7 in this number system. The base of octal number system is 8, because it has only 8 digits.

**3) Decimal number system**

Decimal number system has only ten (10) digits from**0 to 9**. Every number (value) represents with 0,1,2,3,4,5,6, 7,8 and 9 in this number system. The base of decimal number system is 10, because it has only 10 digits.

**4) Hexadecimal number system**

A Hexadecimal number system has sixteen (16) alphanumeric values from **0 to 9** and **A to F**. Every number (value) represents with 0,1,2,3,4,5,6, 7,8,9,A,B,C,D,E and F in this number system. The base of hexadecimal number system is 16, because it has 16 alphanumeric values. Here **A is 10**, **B is 11**, **C is 12**, **D is 13**, **E is 14**and **F is 15**.

**CONVERSIONS:**

**BINARY SYSTEM :**

* Binary to decimal conversion
* Binary to octal conversion
* Binary to hexadecimal conversion

Here is an example of converting binary directly into decimal. We simply add up the place values of each 1 digit in the binary number.

**Binary to decimal conversion**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1001012 = 3710:   |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Exponents | 25 |  | 24 |  | 23 |  | 22 |  | 21 | 20 |  | | Place Values | 32 |  | 16 |  | 8 |  | 4 |  | 2 | 1 |  | | Bits | 1 |  | 0 |  | 0 |  | 1 |  | 0 | 1 |  | | Value | 32 |  |  |  |  | + | 4 |  |  | + 1 |  | = | 37 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |   **Binary to octal conversion**  **Step 1:**Take the given binary number  **Step 2:**Multiply each digit by 2n-1 where n is the position of the digit from the decimal.If it is a decimal number multiply the each digit in the decimal part by http://math.tutorvista.com/js/jsMath/wysiwyg_asciimath/mimetex/mimetex.cgi?%5Cdisplaystyle%5Cfrac%7B%7B1%7D%7D%7B%7B%7B2%7D%7D%5E%7B%7Bm%7D%7D%7D ,m is the position of the digit from the decimal point  **Step 3:**The resultant is the equivalent decimal number for the given binary number.  **Step 4:** Divide the decimal with 8  **Step 5:** Note the remainder  **Step 6:** Continue the above two steps with the quotient till the quotient is zero  **Step 7:** Write the remainder in the reverse order  **Step 8:** The resultant is the required octal number for the given binary number.  **Question:**Convert 10101012 to octal **Solution:**  Given binary number is 10101012  First we convert given binary to decimal  10101012= (1 \* 26) + (0 \* 25) + (1 \* 24) + (0 \* 23) + (1 \* 22) + (0 \* 21) + (1 \* 20)  = 64 + 0 + 16 + 0 + 4 + 0 + 1  = 64 + 21  = 85  (Decimal form)  Now we will convert this [decimal to octal](http://math.tutorvista.com/number-system/decimal-to-octal.html) form  8 | 85  ------- 8 | 10 -- 5 ------- 8 | 1 -- 2 -------  8 | 0 --1  10101012 equivalent octal form is 1258. | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 28 |  | 27 |  | 26 |  | 25 |  | 24 |  | 23 |  | 22 |  | 21 |  | 20 | | Place Values | 256 |  | 128 |  | 64 |  | 32 |  | 16 |  | 8 |  | 4 |  | 2 |  | 1 | | Bits | 1 |  | 1 |  | 1 |  | 1 |  | 0 |  | 1 |  | 0 |  | 0 |  | 0 | | Value | 256 | + | 128 | + | 64 | + | 32 |  |  | + | 8 |  |  |  |  |  |  | = | 488 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Exponents | 27 |  | 26 |  | 25 |  | 24 |  | 23 |  | 22 |  | 21 |  | 20 | | Place Values | 128 |  | 64 |  | 32 |  | 16 |  | 8 |  | 4 |  | 2 |  | 1 | | Bits | 1 |  | 0 |  | 1 |  | 1 |  | 0 |  | 1 |  | 0 |  | 1 | | Value | 128 |  |  | + | 32 | + | 16 |  |  | + | 4 |  |  | + | 1 | = | 181 | |

**Binary to hexadecimal conversion**

**Step 1:** The given number is in binary from.

**Step 2:** First,we have to change the **binary number into decimal number.**

**Step 3:** Then, we count the number of binary digits in the given number. Let there be n numbers.

**Step 4:**Then, we multiply each digit with 2n-1, when n is equal to number of position from right side.

**Step 5:** Add all numbers after multiplication.

**Step 6:** Now, the binary number is in decimal number.

**Step 7:** Now, convert **decimal to hexadecimal**. If the decimal number is less than sixteen, it will be converted by above table.

**Step 8:** If decimal number is greater than sixteen, it should be divided by 16.

**Step 9:** Remainder must be less than 16. (It will be converted by table).

**Step 10:** Then, we write quotient first and then hexadecimal form of remainder together.

**Step 11:** The resultant is in hexadecimal form of given binary number.

**Question 1:**Convert 01011011 in hexadecimal number.  
**Solution:**

The given binary number is 01011011

Now, we convert it first to decimal number

So, 01011011 =(0 ×× 277) + (1 ×× 266) + (0 ×× 255) + (1 ×× 244) + (1 ×× 233) + (0 ×× 222) + (1 ×× 211) + (1 ×× 200)

= (0 ×× 128) + (1 ×× 64) + (0 ×× 32) + (1 ×× 16) + (1 ×× 8) + (0 ×× 4) + (1 ×× 2) + (1 ×× 1)

= 0 + 64 + 0 + 16 + 8 + 0 + 2 + 1

= 91 (decimal form of binary number)

Now, we have to change it into hexadecimal number. So, 91 is greater than 16. So, we have to divide by 16.

After dividing by 16, quotient is 5 and remainder is 11.

Remainder is less than 16.

Hexadecimal number of remainder is B.

Quotient is 5 and hexadecimal number of remainder is B.

So, 5B is the hexadecimal number equivalent to decimal number 91.

**OCTAL SYSTEM**

**Octal to Binary conversion**

**Step 1:** Consider the given octal number

**Step 2:** Let the given number have n number of digits

**Step 3:** Multiply each digit of the number with 8n-1, when the digit is in the nth position from the right end of the number.If the number has decimal part the multiply each digit in the decimal part by http://math.tutorvista.com/js/jsMath/wysiwyg_asciimath/mimetex/mimetex.cgi?%5Cdisplaystyle%5Cfrac%7B%7B1%7D%7D%7B%7B%7B8%7D%7D%5E%7B%7Bm%7D%7D%7D when the digit is in the mth position from the decimal point.

**Step 4:** Add all terms after multiplication

**Step 5:** The obtained value is the equivalent decimal number

**Step 6:** Consider the decimal number, divide it by 2

**Step 7:** Note the remainder

**Step 8:** Continue the above two steps for the quotient till the quotient is zero

**Step 9:** Write the remainders in the reverse order

**Step 10:**The obtained number is the equivalent binary number for the given octal number.

**Question 1:**Convert 418 to a binary number.

**Solution:**

Given number is 418  
  
418

 = (4 \* 81) + (1 \* 80)  
= 4 \* 8 + 1 \* 1  
= 32+1  
= 33(Decimal number)  
  
Now convert this decimal number to a binary number.  
  
2 | 33  
2 | 16 -- 1  
2 | 8  -- 0  
2 | 4  --0  
2 | 2  -- 0  
      1 -- 0  
  
The binary number is 1000012  
  
418 = 1000012

**Octal to Decimal conversion**

**Step 1:**Take the given octal number.

**Step 2:**Find out the number of digits in the number

**Step 3:**Let it have n digits.

**Step 4:**Multiply each digit in the number with 8n-1,when the digit is in the nth position.

**Step 5:**Add all digits after multiplication.

**Step 6:**The resultant is the equivalent decimal to the given octal number.

If octal number contains a decimal point

**Step 7:**Let m digits are there after the decimal

**Step 8:**Multiply each digit after decimal withhttp://math.tutorvista.com/js/jsMath/wysiwyg_asciimath/mimetex/mimetex.cgi?%5Cdisplaystyle%5Cfrac%7B%7B1%7D%7D%7B%7B%7B8%7D%7D%5E%7B%7Bm%7D%7D%7D ,when the digit is the mth position.

All other steps are same as above.

**Example: Convert 57468 to decimal number  
Solution:**

The given number is 57468  
  
57468

= (5 \* 83)+ (7 \* 82) + (4 \* 81) + (6 \* 80)  
=5 \* 512 + 7 \* 64 + 4 \* 8 + 6 \* 1  
= 2560+448+32+6  
= 3046  
  
The equivalent decimal number for 57468 is 3046  
  
57468 = 3046

**Octal to hexadecimal conversion**

**Step 1:** Let the number of digits in the number be n

**Step 2:** Multiply the digits with 8n-1where n is position of digit from the right end of the number.If the number has decimalpart the multiply digits after decimal by http://math.tutorvista.com/js/jsMath/wysiwyg_asciimath/mimetex/mimetex.cgi?%5Cdisplaystyle%5Cfrac%7B%7B1%7D%7D%7B%7B%7B8%7D%7D%5E%7B%7Bm%7D%7D%7D where m is position of the number from the decimal

**Step 3:** Add the terms after multiplication

**Step 4:** The obtained number is equivalent decimal number to the given octal

**Step 5:**Consider the decimal number,divide it by 16

**Step 6:** Note the remainder.

**Step 7:** Continue the process till the quotient in zero

**Step 8:** Write the remainder in the reverse order

**Step 9:** The obtained number is equivalent hexadecimal number to the given octadecimal number.

**Example:**Convert 10028 to hexadecimal  
**Solution:**

The given number is 10028  
  
10028

 = (1 \* 83)+ (0 \* 82) + (0 \* 81) + (2 \* 80)  
=1 \* 512 + 0 \* 64 + 0 \* 8 + 2 \* 1  
= 512 + 0 + 0+ 2  
= 514(decimal number)  
  
Now we convert the above decimal to hexadecimal  
  
16 | 514  
16 | 32   --2  
        2 -- 0  
  
The hexadecimal number is 202  
  
10028 = 20216

**DECIMAL SYSTEM**

**Decimal to Binary conversion**

**Step 1:** Divide given number starting from 2 as suitable.

**Step 2:**Write remainder on the right side of quotient.

**Step 3:**Divide untill quotient will be 0.

**Step 4:**Now write binary number starting from lower end of that divison.

**Step 5:**Now write given number including quotient of lower end that should be starting point.

**Example:**Convert 35 into binary number.  
**Solution:**

The binary number can be calculated by using L division method

2 | 35  
-------  
2 | 17 -- 1  
-------  
2 | 8 -- 1  
-------  
2 | 4 --0  
-------  
2 | 2 -- 0  
--------  
1 -- 0

**answer is (100011)2.**

**Decimal to octal conversion**

**Step 1:**Take the given decimal number

**Step 2:**If the number is less than 8 the octal number is the same

**Step 3:**If the number is greater than 7 then Divide the number with 8

**Step 4:**Note the remainder

**Step 5:**Carry on the step 3 and 4 with the qoutient till it is less than 8

**Step 6:**Write the remainders in reverse order(bottom to top)

**Step 7:**The resultant is the equivalent octal number to the given decimal number

**Question 1:**Convert 128 to octal form.  
**Solution:**

Given decimal number is 128

Start the division process

8 | 128  
8 | 16 -- 0  
8 | 2 -- 0  
8 | 0 –2

**answer is The equivalent octal number for 128 is (200)8.**

**Decimal to Hexadecimal conversion**

**Step 1:**0 to 15 we can covert directly by the table.

**Step 2:** For other numbers. Divide the decimal number by 16.

**Step 3:**Remainder will always be less than 16.

**Step 4**: Quotient will write first

**Step 5:** Convert remainder by the help of table.

**Step 6:** After Quotient we will write the hexadecimal number of remainder.

**Example:**Convert 146 to hexadecimal number?

**Solution:**

146 is greater than 16 , so we have to divide by 16.

After dividing by 16 , quotient is 9 and remainder is 2.

remainder is less than 16.

the hexadecimal number of remainder is 2.

Quotient is 9 and hexadecimal number of remainder is 2.

so, the 92 is the hexadecimal number is equivalent to decimal number 146.

16 | 146  
 -------------  
 16 | 9 -- **2**  
 -------------  
 0 – **9**

**HEXADECIMAL CONVERSIONS**

**Hexadecimal to Binary conversion**

**Step 1:**Take given hexadecimal number

**Step 2:**Find the number of digits in the decimal

**Step 3:**If it has n digits,multiply each digit with 2n-1where the digit is in the nth position

**Step 4:**Add the terms after multiplication

**Step 5:**The resultant is the decimal number equivalent to the given hexadecimal number.Now we have to convert this hexadecimal to binary number.

**Step 6:**Divide the decimal number with 2

**Step 7:**Note the remainder

**Step 8:**Do the above 2 steps for the quotient till the quotient is zero

**Step 9:**Write the remainders in the reverse order.

**Step 10:** The resultant is the required binary number.

**Example:**Convert A2B16 to a equivalent binary number   
**Solution:**

 Given hexadecimal number is A2B  
  
A2B16 = (A \* 162) + (2 \* 161) + (B \* 160)  
= (A \* 256) + (2 \* 16) + (B \* 1)  
= (10 \*256) + 32 + 11  
= 2560 + 43  
  
= 2603(Decimal number)  
  
Now we have to convert 2603 to binary  
  
2 | 2603  
2 |1301 -- 1  
2 | 650 -- 1  
2 | 325 -- 0  
2 | 162 -- 1  
2 | 81  -- 0  
2 | 40 -- 1  
2 | 20 -- 0  
2 | 10 -- 0  
2 | 5 -- 0  
2 | 2 -- 1  
2 | 1 -- 0  
2 | 0 -- 1  
  
The binary number is 1010001010112  
  
A2B16 = 1010001010112

**Hexadecimal to octal conversion**

**Step 1:**Consider the given hexadecimal number

**Step 2:**First count the number of digits in the number

**Step 3:**If n is the position of the digit from the right end then multiply each digit with 16n-1

**Step 4:**Add the terms after multiplication

**Step 5:**Resultant is the equivalent decimal form

**Step 6:**Divide the decimal number with 8

**Step 7:**Note down the remainder

**Step 8:**Continue step 6 and 7 with the quotient, until the quotient is zero

**Step 9:**Write the remainders in reverse order

**Step 10:**The obtained number is the required result

**Example :**Find the equivalent octal form of C116

**Solution:**

 Given hexadecimal number is C1  
  
C116

= (C \* 161) + (1 \* 160)  
= C \* 16 + 1 \* 1  
=12 \* 16 + 1  
= 192 + 1  
  
=193 (Decimal form)  
  
Now we have to convert this decimal to octal  
  
8 | 193  
8 | 24 -- 1  
8 | 3 -- 0  
8 | 0 --3  
  
The octal number is 3018  
  
C116 = 3018

**Hexadecimal to Decimal conversion**

**Step 1:** First we find the number of hexadecimal digits in the number. Let there be n numbers.

**Step 2:** Then we multiply each hexadecimal digit with 16n−116n−1, when n is equal to number of position from right side.

**Step 3:**Then we add each number after multiplication.

**Step 4:**The resultant is equivalent hexadecimal number of the given decimal number.

**Example :**Convert 7B16  into decimal number.  
**Solution:**

 Given hexadecimal number is 7B1616.

7B1616

= 1611 ×× 7 + 1600 ×× B

= 16 ×× 7 + 1 ×× B

= 112 + 1 ×× 11

= 112 + 11

= 123

Answer is 123

**1.6 Introduction to Algorithms: steps to solve logical and numerical problems**

### [Definition of Algorithm](https://www.edrawsoft.com/algorithm-definition.php)

To write a logical step-by-step method to solve the problem is called algorithm, in other words, an algorithm is a procedure for solving problems. In order to solve a mathematical or computer problem, this is the first step of the procedure. An algorithm includes calculations, reasoning and data processing. Algorithms can be presented by natural languages, pseudo code and flowcharts, etc.

**1.6.1 Example 1: Print 1 to 20:**

**Algorithm:**

Step 1: Initialize X as 0,

Step 2: Increment X by 1,

Step 3: Print X,

Step 4: If X is less than 20 then go back to step 2.

**Example 2: Convert Temperature from Fahrenheit (℉) to Celsius (℃)**

**Algorithm:**

Step 1: Read temperature in Fahrenheit,

Step 2: Calculate temperature with formula C=5/9\*(F-32),

Step 3: Print C,

### [Definition of Flowchart](https://www.edrawsoft.com/Flowchart-Definition.php)

A Flow chart is a Graphical representation of an Algorithm or a portion of an Algorithm. Flow charts are drawn using certain special purpose symbols such as Rectangles, Diamonds, Ovals and small circles. These symbols are connected by arrows called flow lines.

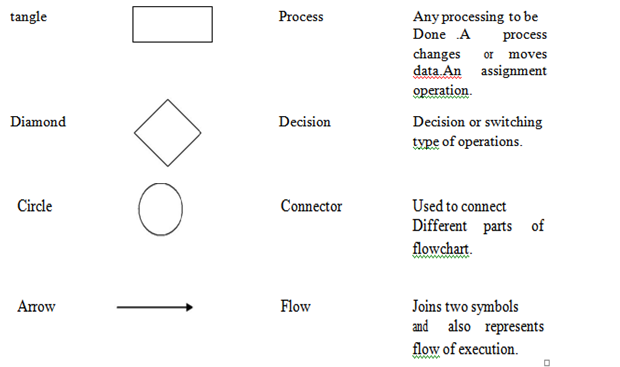
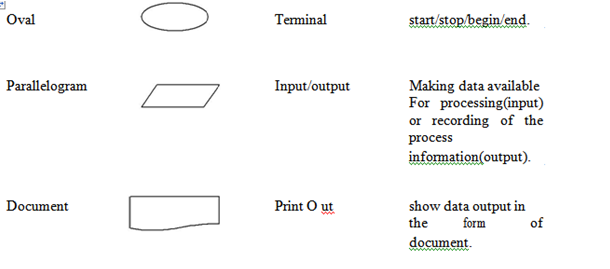
(or)

The diagrammatic representation of way to solve the given problem is called flow

chart.

**The following are the most common symbols used in Drawing flowcharts:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Shape Type**  **of the symbol** | **Symbol** | **Name of the**  **Symbol** | **Meaning** |



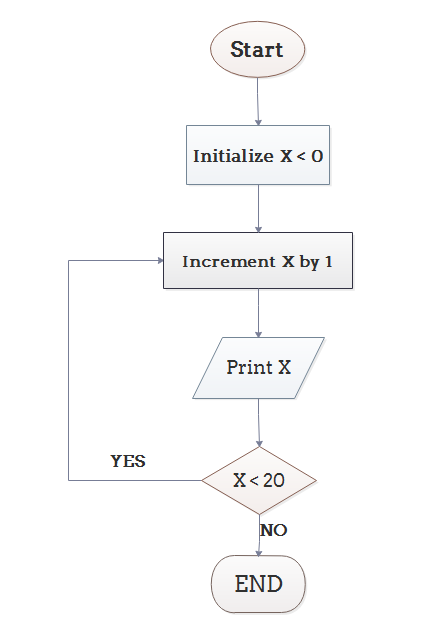
The graphics above represent different part of a flowchart. The process in a flowchart can be expressed through boxes and arrows with different sizes and colors. In a flowchart, we can easily highlight a certain element and the relationships between each part.

### 1.6.2 How to Use Flowcharts to Represent Algorithms

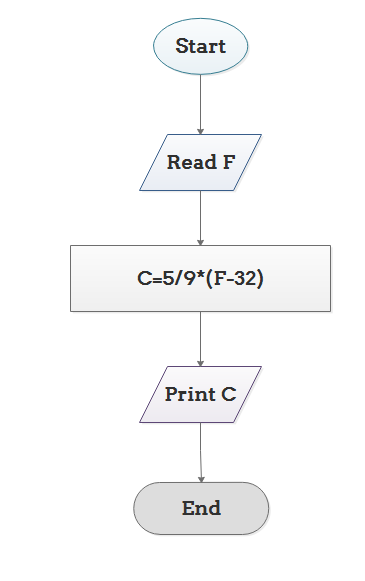
*Now that we have the definitions of algorithm and flowchart, how do we use a flowchart to represent an algorithm?*

Algorithms are mainly used for mathematical and computer programs, whilst flowcharts can be used to describe all sorts of processes: business, educational, personal and of course algorithms. So, flowcharts are often used as a program planning tool to visually organize step-by-step process of a program. Here are some examples:

**Example 1: Flow chart for printing 1 to 20:**



**Example 2: Flow chart for converting Temperature from Fahrenheit (℉) to Celsius (℃)**

****

**PSEUDOCODE:**

Pseudo code is an artificial and informal language that helps programmers develop algorithms. Pseudo code is similar to everyday English, it is convenient and user friendly although it is not an actual computer programming language. Psuedo code programs are not actually executed on computer rather they merely help the programmer “think out” a program before attempting to write it in a programming language such as C.

**1.7 Program Design and structured programming:**

**1.7.1 Program Design Language** (or **PDL**, for short) is a method for designing and documenting methods and procedures in software. It is related to [pseudo code](https://en.wikipedia.org/wiki/Pseudocode), but unlike pseudo code, it is written in plain language without any terms that could suggest the use of any programming language or library.

PDL was originally developed by the company Caine, Farber & Gordon and has been modified substantially since they published their initial paper on it in 1975. It has been described in some detail by [Steve McConnell](https://en.wikipedia.org/wiki/Steve_McConnell) in his book [*Code Complete*](https://en.wikipedia.org/wiki/Code_Complete).

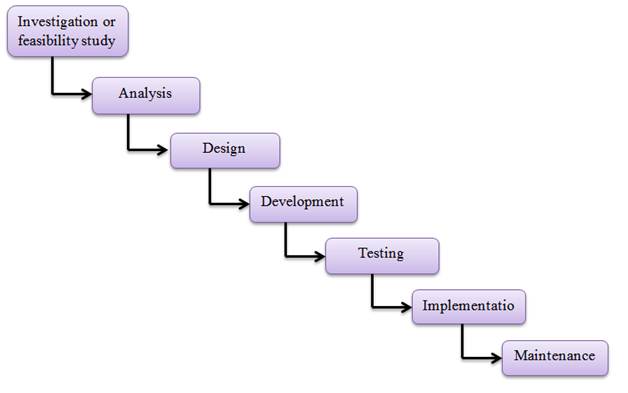
**1.7.2 Program development System Development Method**

A **software development process** is a structure imposed on the development of a software product. This critical process determines the overall quality and success of our program. If we carefully design each program using good structured development techniques, programs will be efficient, error-free, and easy to maintain.

**System Development Life Cycle**

Today’s large-scale modern programming projects are built using a series of interrelates phases commonly referred to as the system development cycle. Although the exact number and names of the phases differ depending on the environment there is general agreement as to the steps that must be followed. One very popular development life cycle developed, this modal consists of between 5 and 6 phases.

The water fall modal starts with systems requirements in this phase the systems analyst defines requirements that specify what the proposed system is to accomplish.



The *requirements* are usually stated in terms that the user understands.

The *analysis* *phase* looks at different alternatives from a systems point of view while the design phase determined how the system will be built.

In the *design* phase the functions of the individual programs that will make up the system are determined and the design of the files and / or the databases is completed.

Finally in the *4th phase code*, we write the programs. After the programs have been written and tested to the programmer’s satisfaction, the project proceeds to the system test.

All of the programs are tested together to make sure of the system works as a whole

The final phase maintenance keeps the system working once it has been put into production.

Although the implication of the water falls approach is that the phases flow in a continuous stream from the first to the last, this is not really the case. As each phase is developed, errors and omissions will often be found in the previous work. When this happens it is necessary to go back to the previous phase to rework it for consistency and to analyze the impact caused by the changes.

**PROGRAM DEVELOPMENT STEPS:**

Program Development is a multistep process that requires that we understand the problem, develop a solution, write the program, and then test it. When we are given the assignment to develop a program, we will be given a program requirements statement and the design of any program interfaces. We should also receive an overview of the complete project so that we will take the inputs we are given and convert them to the outputs that have been specified. This is known as program design.

**Understand the Problem**

The first step in solving any problem is to understand it. By reading the requirements statements carefully, we fully understand it, we review our understanding with the user and the systems analyst to know the exact purpose.

**Develop the solution**

Once we fully understand the problem we need to develop our solution. Three tools will help in this task. 1. Structure chart, 2.Psuedocode &3.Flowcharts. Generally we will use structure chart and either flowchart or Pseudo code

The structure chart is used to design the whole program .Pseudo code and flowcharts are used to design the individual parts of the program.

**Structure chart:** A structure chart, also known as hierarchy chart, shows the functionalflow through our program. The structure chart shows how we are going to break our program into logical steps each step will be a separate module. The structure chart shows the interaction between all the parts (modules) of our program.

We can use flowchart or pseudo code to complete the design of your program will depend on experience and difficulty of the program your designing.

**Write the program**

When we write a program, we start with the top box on the structure chart and work our way to the bottom. This is known as top-down implementation. We will write the programs by using structure chart and flowchart or pseudo code.

**Test the Program**

Program testing can be a very tedious and time- consuming part of program development. As the programmer we are responsible for completely testing our program. In large-development projects test engineers are responsible for testing to make sure all the programs work together.

**1.8 INTRODUCTION TO ‘C’ LANGUAGE:**

**1.8.1 BACK GROUND OF C:**

C language facilitates a very efficient approach to the development and implementation of computer programs. The History of C started in 1972 at the Bell Laboratories, USA where Dennis M. Ritchie proposed this language. In 1983 the American National Standards Institute (ANSI) established committee whose goal was to produce “an unambiguous and machine independent definition of the language C” while still retaining it’s spirit .

C is the programming language most frequently associated with UNIX. Since the 1970s, the bulk of the UNIX operating system and its applications have been written in C. Because the C language does not directly rely on any specific hardware architecture, UNIX was one of the first portable operating systems. In other words, the majority of the code that makes up UNIX does not know and does not care which computer it is actually running on. Machine-specific features are isolated in a few modules within the UNIX kernel, which makes it easy for you to modify them when you are porting to a different hardware architecture.

C was first designed by Dennis Ritchie for use with UNIX on DEC PDP-11 computers. The language evolved from Martin Richard's BCPL, and one of its earlier forms was the B language, which was written by Ken Thompson for the DEC PDP-7. The first book on C was *The C Programming Language* by Brian Kernighan and Dennis Ritchie, published in 1978.

In 1983, the American National Standards Institute (ANSI) established a committee to standardize the definition of C. The resulting standard is known as *ANSI C,* and it is the recognized standard for the language, grammar, and a core set of libraries. The syntax is slightly different from the original C language, which is frequently called K&R for Kernighan and Ritchie. There is also an ISO (International Standards Organization) standard that is very similar to the ANSI standard. It appears that there will be yet another ANSI C standard officially dated 1999 or in the early 2000 years; it is currently known as "C9X."

Algorithms and flowcharts are two different tools used for creating new programs, especially in computer programming. An algorithm is a step-by-step analysis of the process, while a flowchart explains the steps of a program in a graphical way.

**1.8.2 BASIC STRUCTURE OF C LANGUAGE:**

The program written in C language follows this basic structure. The sequence of sections should be as they are in the basic structure. A C program should have one or more sections but the sequence of sections is to be followed.

Documentation section

Linking section

Definition section

Global declaration section

Main function section

{

Declaration section

Executable section

}

Sub program or function section

**DOCUMENTATION SECTION:** comes first and is used to document the use oflogic or reasons in your program. It can be used to write the program's objective, developer and logic details. The documentation is done in C language with /\* and \*/ . Whatever is written between these two are called comments.

**LINKING SECTION:** This section tells the compiler to link the certain occurrencesof keywords or functions in your program to the header files specified in this section.

e.g. #include <stdio.h>

**DEFINITION SECTION:** It is used to declare some constants and assign them somevalue.

e.g. #define MAX 25

Here #define is a compiler directive which tells the compiler whenever MAX is found in the program replace it with 25.

**GLOBAL DECLARATION SECTION :** Here the variables which are used throughout the program (including main and other functions) are declared so as to make them global(i.e accessible to all parts of program)

e.g. int i; (before main())

**MAIN FUNCTION SECTION:** It tells the compiler where to start the executionfrom

main()

{

point from execution starts

}

main function has two sections

* declaration section : In this the variables and their data types are declared.
* Executable section : This has the part of program which actually performs the task we need.

**SUB PROGRAM OR FUNCTION SECTION :** This has all the sub programs or thefunctions which our program needs.

**SIMPLE ‘C’ PROGRAM:**

/\* simple program in c \*/

#include<stdio.h> main()

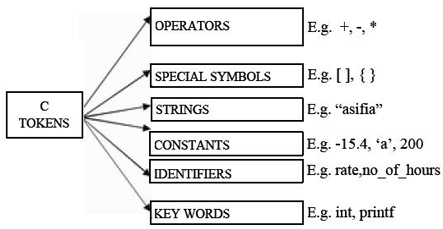
{

printf(“welcome to c programming”);

} /\* End of main \*/

**1.8.3 C-TOKENS :**

Tokens are individual words and punctuations marks in English language sentence. The smallest individual units are known as C tokens.



A C program can be divided into these tokens. A C program contains minimum 3 c tokens no matter what the size of the program is.

**1.8.3.1 DATA TYPES:**

To represent different types of data in C program we need different data types. A data type is essential to identify the storage representation and the type of operations that can be performed on that data. C supports four different classes of data types namely

* **Basic Data types**
* **Derives data types**
* **User defined data types**
* **Pointer data types**

**BASIC DATA TYPES:**

All arithmetic operations such as Addition, subtraction, etc are possible on basic data types.

E.g.: int a,b;

Char c;

**The following table shows the Storage size and Range of basic data types:**

|  |  |  |
| --- | --- | --- |
| **TYPE** | **LENGTH** | **RANGE** |
| Unsigned char | 8 bits | 0 to 255 |
| Char | 8 bits | -128 to 127 |
| Short int | 16 bits | -32768 to 32767 |
| Unsigned int | 32 bits | 0 to 4,294,967,295 |
| Int | 32 bits | -2,147,483,648 to 2,147,483,648 |
| Unsigned long | 32 bits | 0 to 4,294,967,295 |
| Enum | 16 bits | -2,147,483,648 to 2,147,483,648 |
| Long | 32 bits | -2,147,483,648 to 2,147,483,648 |
| Float | 32 bits | 3.4\*10E-38 to 3.4\*10E38 |
| Double | 64 bits | 1.7\*10E-308 to 1.7\*10E308 |
| Long double | 80 bits | 3.4\*10E-4932 to 1.1\*10E4932 |

**DERIVED DATA TYPES:**

Derived data types are used in ‘C’ to store a set of data values. Arrays and Structures are examples for derived data types.

Ex: int a[10];

Char name[20];

**USER DEFINED DATATYPES:**

C Provides a facility called typedef for creating new data type names defined by the user. For Example , the declaration :

**typedef int Integer;**

makes the name Integer a synonym of int.Now the type Integer can be used in declarations ,casts, etc, like,

**Integer num1,num2;**

Which will be treated by the C compiler as the declaration of num1,num2as int variables. “typedef” ia more useful with structures and pointers.

**POINTER DATA TYPES:**

Pointer data type is necessary to store the address of a variable.

**1.9 VARIABLES:**

A quantity that can vary during the execution of a program is known as a variable. To identify a quantity we name the variable for example if we are calculating a sum of two numbers we will name the variable that will hold the value of sum of two numbers as 'sum'.

**IDENTIFIERS :**

* Names of the variables and other program elements such as functions, array,etc,are known as identifiers.
* There are few rules that govern the way variable are named(identifiers).
* Identifiers can be named from the combination of A-Z, a-z, 0-9, \_(Underscore).
* The first alphabet of the identifier should be either an alphabet or an underscore. digit are not allowed.
* It should not be a keyword. Eg: name,ptr,sum After naming a variable we need to declare it to compiler of what data type it is . The format of declaring a variable is

***Data-type id1, id2,.....idn;***

where data type could be float, int, char or any of the data id1, id2, id3 are the names of variable we use. In case of single variable no commas are required.

eg float a, b, c;

Int e, f, grand total;

char present\_or\_absent;

**ASSIGNING VALUES :**

When we name and declare variables we need to assign value to the variable. In some cases we assign value to the variable directly like

a=10;

in our program

In some cases we need to assign values to variable after the user has given input for that.

Eg: we ask user to enter any no and input it.

/\* write a program to show assigning of values to variables \*/

#include<stdio.h>

main()

{

int a; float b;

printf("Enter any number\n"); b=190.5;

scanf("%d",&a); printf("user entered %d", a); printf("B's values is %f", b);

}

**CONSTANTS :**

A quantity that does not vary during the execution of a program is known as a constant supports two types of constants namely Numeric constants and character constants.

**NUMERIC CONSTANTS:**

* Example for an integer constant is 786,-127
* Long constant is written with a terminal ‘l’or ‘L’,for example 1234567899L is a Long constant.
* Unsigned constants are written with a terminal ‘u’ or ‘U’,and the suffix ‘ul’ and ‘UL’ indicates unsigned long. for example 123456789u is a Unsigned constant and 1234567891ul is an unsigned long constant.
* The advantage of declaring an unsigned constant is to increase the range of storage.
* Floating point constants contain a decimal point or an exponent or both. For Eg : 123.4,1e-2,1.4E-4,etc.The suffixes f or F indicate a float constant while the absence of f or F indicate the double, l or L indicate long double.

**CHARACTER CONSTANTS:**

A character constant is written as one character with in single quotes such as ‘a’. The value of a character constant is the numerical value of the character in the machines character set. certain character constants can be represented by escape sequences like ‘\n’. These sequences look like two characters but represent only one.

The following are the some of the examples of escape sequences:

**Escape sequence** **Description**

\a Alert

\b Backspace

\f Form feed

\n New Line

\r Carriage return

\t Horizontal Tab

\v Vertical Tab

String constants or string literal is a sequence of zero or more characters surrounded by a double quote. Example , “ I am a little boy”. quotes are not a part of the string.

To distinguish between a character constant and a string that contains a single character ex: ‘a’ is not same as “a”. ‘a’ is an integer used to produce the numeric value of letter a in the machine character set, while “a” is an array of characters containing one character and a ‘\0’ as a string in C is an array of characters terminated by NULL.

There is one another kind of constant i.e Enumeration constant , it is a list of constant integer values.

**1.10 Syntax and Logical Errors in Compilation:**

**Syntax error**: A **syntax error** is an error in the [syntax](https://en.wikipedia.org/wiki/Syntax_(programming_languages)) of a sequence of characters or [tokens](https://en.wikipedia.org/wiki/Token_(parser)) that is intended to be written in a particular [programming language](https://en.wikipedia.org/wiki/Programming_language).

For [compiled languages](https://en.wikipedia.org/wiki/Compiled_language), syntax errors are detected at [compile-time](https://en.wikipedia.org/wiki/Compiler). A program will not compile until all syntax errors are corrected.

**For example,** consider the statement,

|  |  |
| --- | --- |
| 1 | int a,b: |

The above statement will produce syntax error as the statement is terminated with : rather than ;

**Logical Error:** A **logic error** is a [bug](https://en.wikipedia.org/wiki/Software_bug) in a program that causes it to operate incorrectly, but not to terminate abnormally (or [crash](https://en.wikipedia.org/wiki/Crash_(computing))). A logic error produces unintended or undesired output or other behaviour, although it may not immediately be recognized as such.

Logic errors occur in both [compiled](https://en.wikipedia.org/wiki/Compiler) and [interpreted](https://en.wikipedia.org/wiki/Interpreter_(computing)) languages. Unlike a program with a [syntax error](https://en.wikipedia.org/wiki/Syntax_error), a program with a logic error is a valid program in the language, though it does not behave as intended.

**For example,**

a=5/0;

1.11 **Object and Executable code:**

The programming language code is called as Source code or object code and is obtained after compilation. Source code extension is .c file and object code extension is .obj and executable file is .exe

**1.12 OPERATORS :**

An operator is a symbol that tells the compiler to perform certain mathematical or logical manipulations. They form expressions.

C operators can be classified as

* Arithmetic operators
* Relational operators
* Logical operators
* Assignment operators
* Increment or Decrement operators
* Conditional operator
* Bit wise operators
* Special operators

**1. ARITHMETIC OPERATORS:** All basic arithmetic operators are present in C.

**operator meaning**

+ add

- subtract

\* multiplication

/ division

% modulo division(remainder)

An arithmetic operation involving only real operands(or integer operands) is called real arithmetic(or integer arithmetic). If a combination of arithmetic and real is called mixed mode arithmetic.

**2. RELATIONAL OPERATORS:** We often compare two quantities and depending ontheir relation take certain decisions for that comparison we use relational operators.

**operator meaning**

< is less than

* is greater than

<= is less than or equal to

>= is greater than or equal to

== is equal to

!= is not equal to

**3. LOGICAL OPERATORS:** An expression of this kind which combines two or morerelational expressions is termed as a logical expressions or a compound relational expression. The operators and truth values are

|  |  |  |  |
| --- | --- | --- | --- |
| **op-1** | **op-2** | **op-1 && op-2 op-1 || op-2** | |
| non-zero | non-zero | 1 | 1 |
| non-zero | 0 | 0 | 1 |
| 0 | non-zero | 0 | 1 |
| 0 | 0 | 0 | 0 |
| op-1 | !op-1 |  |  |
| non-zero | Zero |  |  |
| zero | non-zero |  |  |

**4. ASSIGNMENT OPERATORS :** They are used to assign the result of an expressionto a variable. The assignment operator is '='.

**Syntax:** op=exp is variable

op binary operator exp expression

op= short hand assignment operator

**short hand assignment operators**

use of simple assignment operators use of short hand assignment operators

a=a+1 a+=1

a=a-1 a-=1

a=a%b a%=b

**5. INCREMENT AND DECREMENT OPERATORS:**

and == are called increment and decrement operators used to add or subtract. Both are unary and as follows

**Syntax:** ++m or m++ --m or m--

The difference between ++m and m++ is

if m=5; y=++m then it is equal to m=5;m++;y=m; if m=5; y=m++ then it is equal to m=5;y=m;m++;

**6. CONDITIONAL OPERATOR:** A ternary operator pair "?:" is available in C toconstruct conditional expressions of the form

**Syntax:** exp1 ? exp2 : exp3;

It work as

if exp1 is true then exp2 else exp3

**7. BIT WISE OPERATORS:** C supports special operators known as bit wise operatorsfor manipulation of data at bit level. They are not applied to float or double.

**operator meaning**

& Bitwise AND

| Bitwise OR

^ Bitwise exclusive OR

<< left shift

>> right shift

~ one's complement

**8. SPECIAL OPERATORS:**

These operators which do not fit in any of the aboveclassification are , (comma), sizeof, Pointer operators(& and \*) and member selection operators (. and ->). The comma operator is used to link related expressions together. sizeof operator is used to know the sizeof operand.

/\* programs to exhibit the use of operators \*/

#include<stdio.h>

main()

{

int sum, mul, modu; float sub, divi;

int i,j; float l, m;

printf("Enter two integers "); scanf("%d%d",&i,&j); printf("Enter two real numbers"); scanf("%f%f",&l,&m);

sum=i+j;

mul=i\*j;

modu=i%j; sub=l-m; divi=l/m;

printf("sum is %d", sum); printf("mul is %d", mul); printf("Remainder is %d", modu);

printf("subtraction of float is %f", sub); printf("division of float is %f", divi);

}

/\* program to implement relational and logical \*/

#include<stdio.h>

main()

{

int i, j, k;

printf("Enter any three numbers "); scanf("%d%d%d", &i, &j, &k); if((i<j)&&(j<k))

printf("k is largest"); else if i<j || j>k

{

if i<j && j >k

printf("j is largest");

else

printf("j is not largest of all");

}

}

/\* program to implement increment and decrement operators \*/

#include<stdio.h>

main()

{

int i;

printf("Enter a number"); scanf("%d", &i);

i++;

printf("after incrementing %d ", i); i--;

printf("after decrement %d", i); }

/\* program using ternary operator and assignment \*/ #include<stdio.h>

main()

{

int i,j,large;

printf("Enter two numbers "); scanf("%d%d",&i,&j); large=(i>j)?i:j;

printf("largest of two is %d",large);

}

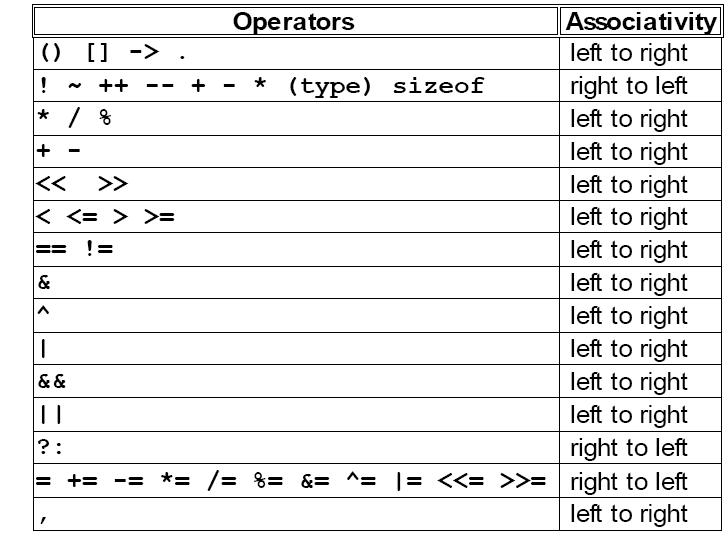
**1.13 EXPRESSIONS AND PRECEDENCE:**

An expression is a sequence of operands and operators that reduces to a single value. Expression can be simple or complex. An **operator** is a syntactical token that requires an action be taken. An **operand** is an object on which an operation is performed.

A simple expression contains only one operator. E.g: 2 + 3 is a simple expression whose value is 5.A complex expression contains more that one operator. E.g: 2 + 3 \* 2. To evaluate a complex expression we reduce it to a series of simple expressions. In this first we will evaluate the simple expression 3 \* 2 (6)and then the expression 2 + 6,giving a result of 8.

The order in which the operators in a complex expression are evaluated is determined by a set of priorities known as **precedence,** the higher the precedence ,the earlier the expression containing the operator is evaluated. If two operators with the same precedence occur in a complex expression ,another attribute of an operator ,its associativity ,takes control. **Associativity** is the parsing direction used to evaluate the expression. It can be either left-to-right or right-to-left. When two operators with the same precedence occur in an expression and their associativity is left-to-right ,the left operator is evaluated first. For example ,in the expression 3\*4/6 ,there are two operators multiplication and division ,with the same precedence and left-to-right associativity. Therefore the multiplication is evaluated before the division.

**The following table shows the precedence and associativity of operators:**



**1.14 EXPRESSION EVALUATION:**

**ARITHMETIC EXPRESSIONS:**

It is a combination of variables, constants and operators arranged according to the syntax of C language.

Some examples A \* B – C

(M + N) \* (X + Y)

**Evaluation of expressions:** Expressions are evaluated using an assignment statement of the form

***Syntax:***

Variable = expression

Eg:

x = x\*y + z-3 \*(z \*y)

**Precedence of arithmetic expressions** is used to evaluate a expression to provide unambiguous way of solving an expression. The highest precedence operator is evaluated then next highest precedence operator until no operator is present.

The precedence or priorities are as follows High \* / %

Low + -

An **expression is evaluated** in left to right and value is assigned to variable in left side of assignment operator.

/\* program to demonstrate evaluation of expressions \*/ #include<stdio.h>

main()

{

float a,b,c,x,y,z; a=9;b=23;c=3; x=a-b/3+c\*2-1; y=a-b/(3+c)\*(2-1); z=a-(b/(3+c)\*2)-1;

printf("values of x,y,z are %d%d%d",x,y,z);

}

**1.15 STORAGE CLASSES:**

**Storage Classes:-** Not only data type is required to declare a variable but its storage classalso has to be mentioned. **(or)**

The variables declared in C programs are totally different from other languages. We can use the same variable names in the C program in separate blocks. When we declare a variable it is available only to specific part or block of the program. Remaining block or other function cannot get access to the variable.

The area or block of the C program from where the variable can be accessed is known as the **scope of variable.** The area or scope of the variable depends on its **storage class** i.e. whereand how it is declared. There are four scope variables in C.

1. Function
2. File
3. Block
4. Function prototype

**A storage class of variable tells us four things:**

1. Where the variable would be stored.
2. The Scope of the variable i.e., in which region of the program the value of variable is actually available for us active.
3. Life time of the variable i.e., how long the variable i.e., how long the variable would be active in the program(longevity or alive).
4. The initial value of the variable if it is not initialized.

**Any variable declared in C can have any one of the four storage classes:**

1. Automatic variables.
2. External variables.
3. Static variables.
4. Register variables.

The variables may also be broadly categorized, depending on the place of their declaration, as

1. Internal variables (*local)*
2. External variables (*global)*

**1*) Automatic variables*** are defined inside a function. A variable declared inside a functionwithout storage class name, by default is an auto variable.

**The features of automatic variables are:-**

|  |  |  |  |
| --- | --- | --- | --- |
| (i) | **Storage** | **:** | Memory |
| (ii) | **Initial value** | **:** | Garbage (or) unpredictable |
| (iii) | **Scope** | **:** | Within the function |

iv) **Life time** **:**Till the control remains in the function.

These variables are created when the function is called and destroyed automatically when the function is exited.

Automatic variables are local to the function in which they are declared. These values cannot be accessed by any other function. The keyword used is „**auto**‟.

**Eg:**

void main ( )

{

auto int a =100;

clrscr ( );

{

auto int a = 300;

{

auto int a = 500;

printf (“\n\n\t a=%d”,a);

}

printf (“\n\n\t a=%d”,a);

}

printf (“\n\n\t a=%d”, a);

getch ( );

|  |  |  |
| --- | --- | --- |
| } |  |  |
| **O/p:** | a = 500 a = 300 | a = 100 |

**Eg 4:**

void show ( );

void main ( )

{

int I;

clrscr ( );

for (i=1; i<5; i++);

show ( )

getch ( );

}

void show ( )

{

int a= 10;

printf (“\n\n\t a=%d”,a);

a++;

}

**2)External variables** are also known as global variables. These variables are declared outside the function and the values of these variables are available to all the functions of the program.

Unlike Local Variables, Global Variables can be accessed by any function in the program.

If same name is given to both the global and local variables priority is given to the local variable. The keyword “ extern” is used to declare these variables.

**The features of external variables are:**

(i) **Storage** **:** memory

(ii) **Initial value :** zero

(iii) **Scope** **:** Global

(iv) **Life time** **:** Till the program comes to an end.

**Example Program:**

#include<stdio.h>

#include<conio.h>

int a=20;

void main( )

{

clrscr( );

fun1( );

fun2( );

fun3( );

printf(“\n In main function a=%d”, a);

}

fun1( )

{

printf(“\n In fun1 a = %d”, a);

}

fun2( )

{

int a = 10;

printf(“\n In fun2 a = %d”,a);

}

fun3( )

{

printf(“\n In fun3 a = %d”, a);

}

**Explanation:** In this program local variable and global variables are declared with the samename in fun2( ). In this case, when fun2( ) is called, the local variable „a‟ of fun2( ) hides the global variable „a‟.

**Note:**

We can declare external variables by using **extern** key word inside the function body. **Ex:** extern int a;

**Eg 1:**

extern int a = 100; // int a;

void main ( )

{

clrscr ( );

printf (“\n\n a=%d”, a);

getch ( );

}

**O/p:** a = 100 //a = 0

1. ***Static variables*** may be of Local (or) global depending upon where it is declared. If it isdeclared outside the function, it is static global otherwise if it declared inside a function block, it is static local.

A static variable is initialized only once and can never be re-initialized. The value of static variable persists at each call and last change made in the variable remains throughout the program execution.

**The features of a static variable are:-**

|  |  |  |  |
| --- | --- | --- | --- |
| (i) | **Storage** | **:** | memory |
| (ii) | **Initial value** | **:** | zero |
| (iii) | **Scope** | **:** | Local to the block in which variable is defined. |
| (iv) | **Life time** | **:** | persists till the end of program execution. |

**Example Programs:-**

**Eg 1:**

void main( )

{

incr( );

incr( );

incr( );

}

void incr( )

{

static int x;

x=x+1;

printf(“%d”, x);

}

The keyword used to declare these variables is “**static**”.

**4*)* RegisterVariables***:*Instead of storing in memory, variables can also be stored in registerof CPU. The advantage of storing in registers is register access is faster than memory access, so, generally frequently accessed variables are kept in registers for faster execution of the program.

**Syntax :** register int count;

The keyword register tells the compiler that the variable list followed by it is kept on the CPU registers. If the CPU fails to keep the variables in CPU registers, in that case the variables are assured as auto and stored in the memory.

**Note:**

1. CPU registers are limited in number. So, we cannot declare more variables as register variables. Compiler automatically converts the register variables into non-register variables once the limit is reached.
2. We cannot use register class for all types of variables. The CPU registers in microcomputer are 16 bit registers. The data types *float* and *double* needs space of more than 16 bits. If we define any variable of these types with register class, no errors will be shown. The compiler treats them as auto variables.

**The features of register variables are:-**

|  |  |  |  |
| --- | --- | --- | --- |
| (i) | **Storage** | **:** | Registers |
| (ii) | **Initial value** | **:** | Garbage |
| (iii) | **Scope** | **:** | Local |
| (iv) | **Life time** | **:** | until the control remains in that function block. |

**Example Programs:-**

**Eg 1:**

void main( )

{

register int i;

for (i=1; i<=5; i++)

printf (“ %d/t”, i);

}

The keyword used to declare these variables is “**register**”.

**1.16 TYPE CONVERSION:**

In an expression that involves two different data types ,such as multiplying an integer and a floating point number to perform these evaluations ,one of the types must be converted.

We have two types of conversions

1. Implicit Type Conversion

2. Explicit Type Conversion

**IMPLICIT TYPE CONVERSION :**

When the types of the two operands in a binary expression are different automatically converts one type to another .This is known as implicit type conversion .

**EXPLICIT TYPE CONVERSION :**

Explicit type conversion uses the unary cast operator ,to convert data from one type to another. To cast data from one type to another ,we specify the new type in parentheses before the value we want converted.

For example ,to convert an integer ,a , to a float, we code the expression like (float) a

**1.17 Conditional Branching and Loops:**

**1.17.1 Writing and evaluation of conditionals**

**STATEMENTS AND BLOCKS :**

A statement causes an action to be performed by the program. It translates directly in to one or more executable computer instructions.

**STATEMENT TYPES:**

**NULL STATEMENT :**

The null statement is just a semicolon (the terminator).

Eg:

//null statement

Although they do not arise often, there are syntactical situations where we must have a statement but no action is required .In these situations we use the null statement.

**EXPRESSION STATEMENT :**

An expression is turned in to a statement by placing a semicolon(;)after it.

**Syntax:**

expression; //expression statement

Eg: a=2;

**RETURN STATEMENT :**

A return statement terminates a function. All functions ,including main, must have a return statement. Where there is no return statement at the end of the function ,the system inserts one with a void return value.

**Syntax:**  return expression; //return statement

The return statement can return a value to the calling function. In case of main ,it returns a value to the operating system rather than to another function. A return value of zero tells the operating system that the program executed successfully.

**COMPOUND STATEMENTS:**

A compound statement is a unit of code consisting of zero or more statements .It is also known as a **block**. The compound statement allows a group of statements to become one single entity. A compound statement consists of an opening brace, an optional declaration and definition section ,and an optional statement section ,followed by a closing brace.

Eg:

{

//Local Declarations int x;

int y; int z;

//Statements

x=1;

y=2;

…

} //End BlocK

**1.18 IF AND SWITCH STATEMENTS :**

We have a number of situations where we may have to change the order of execution of statements based on certain conditions or repeat a group of statements until certain specified conditions are met.

The if statement is a two way decision statement and is used in conjunction with an expression. It takes the following form

If(test expression)

If the test expression is true then the statement block after if is executed otherwise it is not executed

**Syntax:**

if (test expression)

{

statement block;

}

statement–x ;

only statement–x is executed.

/\* program for if \*/ #include<stdio.h> main()

{

int a,b;

printf(“Enter two numbers”); scanf(“%d%d”,&a,&b):

if a>b

printf(“ a is greater”);

if b>a

printf(“b is greater”);

}

**The if –else statement:**

If your have another set of statement to be executed if condition is false then if-else is used

**Syntax:** if (test expression)

{

statement block1;

}

else

{

statement block2;

}

statement –x ;

**Example:** /\* program for if-else \*/

#include<stdio.h>

main()

{

int a,b;

printf(“Enter two numbers”); scanf(“%d%d”,&a,&b):

if a>b

printf(“ a is greater”)

else

printf(“b is greater”);

}

**Nesting of if..else statement :**

If more than one if else statement exists in the program then it is said as Nesting of statement

**Syntax:**

if(text cond1)

{

if (test expression2

{

statement block1;

}

else

{

statement block 2;

}

}

else

{

statement block2;

}

statement-x ;

**if else ladder**

if(condition1)

statement1; else if(condition2) statement 2; else if(condition3) statement n;

else

default statement. statement-x;

The nesting of if-else depends upon the conditions with which we have to deal.

**THE SWITCH STATEMENT:** If for suppose we have more than one valid choices to choose from then we can use switch statement in place of if statements.

**Syntax:** switch(expression)

{.

case value-1

block- 1 break;

case value-2

block- 2 break;

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

default:

default block; break; }

statement–x

In case of Syntax:

if(cond1)

{

statement-1

}

if (cond2)

{

statement 2

}

/\* program to implement switch \*/ #include<stdio.h>

main()

{

int marks,index; char grade[10];

printf(“Enter your marks”); scanf(“%d”,&marks); index=marks/10; switch(index)

{

case 10 : case 9: case 8: case 7:

case 6: grade=”first”; break;

case 5 : grade=”second”; break;

case 4 : grade=”third”; break;

default : grade =”fail”; break;

}

printf(“%s”,grade);

}

**LOOPING :**

Some times we require a set of statements to be executed repeatedly until a\condition is met.

We have two types of looping structures. One in which condition is tested before entering the statement block called entry control.

The other in which condition is checked at exit called exit controlled loop.

**WHILE STATEMENT :**

It helps to execute the statements only if the condition is true.

**Syntax:**

While(test condition)

{

body of the loop

}

It is an entry controlled loop. The condition is evaluated and if it is true then body of loop is executed. After execution of body the condition is once again evaluated and if is true body is executed once again. This goes on until test condition becomes false.

**Example: /\*** program for while \*/ #include<stdio.h> main()

{

int count,n; float x,y;

printf(“Enter the values of x and n”); scanf(“%f%d”,&x,&n);

y=1.0;

count=1;

while(count<=n)

{

y=y\*x;

count++;

}

printf(“x=%f; n=%d; x to power n = %f”,x,n,y);

}

**DO WHILE STATEMENT:**

The while loop does not allow body to be executed if test condition is false. The do while is an exit controlled loop and its body is executed at least once.

Syntax:

do

{

body }while(test condition);

Example: /\* printing multiplication table \*/

#include<stdio.h> #define COL 10 #define ROW 12 main()

{

int row,col,y; row=1;

do

{

col=1; do

{

y=row\*col;

printf(“%d”,y);

col=col+1;

}while(col<=COL);

printf(“\n”);

row=row+1;

}while(row<=ROW);

}

**THE FOR LOOP:**

It is also an entry control loop that provides a more concise structure

Syntax:

for(initialization; test control; increment)

{

body of loop

}

Example: /\* program of for loop \*/ #include<stdio.h>

main()

{

long int p; int n; double q;

printf(“2 to power n “); p=1; for(n=0;n<21;++n)

{

if(n==0)

p=1;

else

p=p\*2;

q=1.0/(double)p;

printf(“%101d%10d”,p,n);

}

}

**BREAK STATEMENT:**

This is a simple statement. It only makes sense if it occurs in the body of a switch, do, while or for statement. When it is executed the control of flow jumps to the statement immediately following the body of the statement containing the break. Its use is widespread in switch statements, where it is more or less essential to get the control .

The use of the break within loops is of dubious legitimacy. It has its moments, but is really only justifiable when exceptional circumstances have happened and the loop has to be abandoned. It would be nice if more than one loop could be abandoned with a single break but that isn't how it works. Here is an example.

#include<stdio.h> #include<stdlib.h> main(){

int i;

for(i = 0; i < 10000; i++){ if(getchar() == 's') break;

printf("%d\n", i);

}

exit(EXIT\_SUCCESS);

}

It reads a single character from the program's input before printing the next in a sequence of numbers. If an ‘s’ is typed, the break causes an exit from the loop.

If you want to exit from more than one level of loop, the break is the wrong thing to use.

**CONTINUE STATEMENT:**

This statement has only a limited number of uses. The rules for its use are the same as for break, with the exception that it doesn't apply to switch statements. Executing a continue starts the next iteration of the smallest enclosing do, while or for statement immediately. The use of continue is largely restricted to the top of loops, where a decision has to be made whether or not to execute the rest of the body of the loop. In this example it ensures that division by zero (which gives undefined behaviour) doesn't happen.

#include<stdio.h> #include<stdlib.h> main(){

int i;

for(i = -10; i < 10; i++){ if(i == 0)

continue; printf("%f\n", 15.0/i); /\*

\* Lots of other statements .....

\*/

}

exit(EXIT\_SUCCESS);

}

The continue can be used in other parts of a loop, too, where it may occasionally help to simplify the logic of the code and improve readability. continue has no special meaning to a switch statement, where break does have. Inside a switch, continue is only valid if there is a loop that encloses the switch, in which case the next iteration of the loop will be started.

There is an important difference between loops written with while and for. In a while, a continue will go immediately to the test of the controlling expression. The same thing in a for will do two things: first the update expression is evaluated, then the controlling expression is evaluated.

**GOTO AND LABELS:**

In C, it is used to escape from multiple nested loops, or to go to an error handling exit at the end of a function. You will need a *label* when you use a goto; this example shows both.

goto L1;

/\* whatever you like here \*/ L1: /\* anything else \*/

A label is an identifier followed by a colon. Labels have their own ‘name space’ so they can't clash with the names of variables or functions. The name space only exists for the function containing the label, so label names can be re-used in different functions. The label can be used before it is declared, too, simply by mentioning it in a goto statement.

Labels must be part of a full statement, even if it's an empty one. This usually only matters when you're trying to put a label at the end of a compound statement—like this.

label\_at\_end: ; /\* empty statement \*/

}

The goto works in an obvious way, jumping to the labelled statements. Because the name of the label is only visible inside its own function, you can't jump from one function to another one.

It's hard to give rigid rules about the use of gotos but, as with the do, continue and the break (except in switch statements), over-use should be avoided. More than one goto every 3–5 functions is a symptom that should be viewed with deep suspicion.

**1.19 & 1.20 FORMATTED INPUT AND OUTPUT STATEMENTS:**

The simplest of input operator is getchar to read a single character from the input device.

**Syntax:** varname=getchar(); //you need to declare varname.

The simplest of output operator is putchar to output a single character on the output device.

putchar(varname)

The getchar() is used only for one input and is not formatted. Formatted input refers to an input data that has been arranged in a particular format, for that we have scanf.

scanf("control string", arg1, arg2,...argn);

Control string specifies field format in which data is to be entered.

arg1, arg2... argn specifies address of location or variable where data is stored.

eg scanf("%d%d",&a,&b);

%d used for integers

%f floats

%l long

%c character

for formatted output you use printf

printf("control string", arg1, arg2,...argn);

/\* program to exhibit i/o \*/

#include<stdio.h>

main()

{

int a,b; float c;

printf("Enter any number"); a=getchar();

printf("the char is "); putchar(a);

printf("Exhibiting the use of scanf");

printf("Enter three numbers");

scanf("%d%d%f",&a,&b,&c);

printf("%d%d%f",a,b,c);

}

**1.20 INTRODUCTION TO STDIN, STDOUT AND STDERR**

**Standard Input (stdin) :**

Standard input is stream data (often text) going into a program. The program requests data transfers by use of the read operation. Not all programs require stream input. For example, the [dir](https://en.wikipedia.org/wiki/Dir_(command)) and [ls](https://en.wikipedia.org/wiki/Ls) programs (which display file names contained in a directory) may take [command-line arguments](https://en.wikipedia.org/wiki/Command-line_interface#Arguments), but perform their operations without any stream data input.

Unless [redirected](https://en.wikipedia.org/wiki/Redirection_(computing)), standard input is inherited from the parent process. In the case of an interactive shell, that is usually associated with the [keyboard](https://en.wikipedia.org/wiki/Keyboard_(computing)).

The [file descriptor](https://en.wikipedia.org/wiki/File_descriptor) for standard input is 0 (zero); <stdio.h> variable is FILE\* stdin;

**Standard Output( stdout):**

Standard output is the stream where a program writes its output data. The program requests data transfer with the write operation. Not all programs generate output. For example, the [file rename](https://en.wikipedia.org/wiki/Rename_(computing)) command (variously called [mv](https://en.wikipedia.org/wiki/Mv), [move](https://en.wikipedia.org/wiki/Move_(command)), or [ren](https://en.wikipedia.org/wiki/Ren_(command))) is silent on success.

Unless [redirected](https://en.wikipedia.org/wiki/Redirection_(Unix)), standard output is inherited from the parent process. In the case of an interactive shell, that is usually the [text terminal](https://en.wikipedia.org/wiki/Text_terminal) which initiated the program.

The [file descriptor](https://en.wikipedia.org/wiki/File_descriptor) for standard output is 1 (one); the [POSIX](https://en.wikipedia.org/wiki/POSIX) *<unistd.h>* definition is STDOUT\_FILENO; the corresponding C *<stdio.h>* variable is FILE\* stdout

# 1.21 COMMAND LINE ARGUMENTS IN C:

The most important function of C/C++ is main() function. It is mostly defined with a return type of int and without parameters :

**Syntax**:

int main() { /\* ... \*/ }

We can also give command-line arguments in C and C++. Command-line arguments are given after the name of the program in command-line shell of Operating Systems.  
To pass command line arguments, we typically define main() with two arguments : first argument is the number of command line arguments and second is list of command-line arguments.

**Syntax**:

int main(int argc, char \*argv[]) { /\* ... \*/ }

or

int main(int argc, char \*\*argv) { /\* ... \*/ }

* **argc (ARGument Count)** is int and stores number of command-line arguments passed by the user including the name of the program. So if we pass a value to a program, value of argc would be 2 (one for argument and one for program name)
* The value of argc should be non negative.
* **argv(ARGument Vector)** is array of character pointers listing all the arguments.
* If argc is greater than zero,the array elements from argv[0] to argv[argc-1] will contain pointers to strings.
* Argv[0] is the name of the program , After that till argv[argc-1] every element is command -line arguments.

**TUTORIAL QUESTIONS**

**This Tutorial corresponds to Unit No: 1**

**Find the output of the following Program:**

1. **#include <stdio.h>   
    int main(void) {   
    int i, j, k;   
    i = -1;   
    j = 1;   
    if(i)   
    j--;   
    if(j)   
    i++;   
    k = i \* j;   
    printf("%d",k);   
    return 0;   
    }**

**Ans:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **#include <stdio.h>   
    int main(void) {   
    float x,y;   
    int i,j;   
    x = 1.5; y = 2.0;   
    i = 2; j = 4;   
    x = x \* y + (float)i / j;   
    printf("%f",x);   
    return 0;   
    }**

**Ans:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **#include <stdio.h>   
    int main(void) {   
    int i = 7, j = i - i;   
    while(i) {   
    i /= 2;   
    j++;   
    }   
    printf("%d",j);   
    return 0;   
    }**

**Ans:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **#include <stdio.h>   
    int main(void) {   
    int i, j = 1;   
    for(i = 11; i > 0; i /= 3)   
    j++;   
    printf("%d",j);   
    return 0;   
    }**

**Ans:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **#include <stdio.h>   
    int main(void) {   
    int i = 1, j = -2, k;   
    k = (i >= 0) || (j >= 00) && (i <= 0) || (j <= 0);   
    printf("%d",k);   
    return 0;   
    }**

**Ans:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**ASSIGNMENT QUESTIONS**

**SET-1:**

1. Design an algorithm and flow chart to find maximum among three numbers.
2. Implement a C program to find factors of given number.

**SET-2:**

1. Design an algorithm and flow chart to check given number is positive or negative or zero
2. Implement a C program to print multiplication table for 1 to n numbers.

**SET-3:**

1. Implement a C program to check given number is Armstrong or not.
2. Design an algorithm and flow chart to convert Celsius to Fahrenheit and Fahrenheit to Celsius.

**SET-4:**

1. Implement a C program to check given number is perfect number or not.
2. Design a structure how to create and run C program.

**SET-5:**

1. Implement a C program to reverse a given number.
2. Design program Development steps.

**SET-6:**

1. Implement a C program to convert decimal number into binary.
2. Discuss about data types.
3. Implement a C program to add two numbers using command line argument.

**IMPORTANT QUESTIONS UNIT – I**

1. What is a flowchart? Explain the different symbols used in a flowchart.

2. (a) Define an Algorithm?

1. What is the use of Flowchart?
2. What are the different steps followed in the program development?

. 3. Explain the different computing environments?

1. Explain the steps for the creation and running c program?
2. Draw a flow chart to find the biggest among three numbers.
3. Write the various steps involved in executing a C program and illustrate with the help of flow chart?
4. What is the difference between break and continue statements ? Explain with examples.
5. What is the purpose of goto statement? How is the associated target statement identified?
6. (a) What are constants?

B.Name the different data types that C supports and explain them in detail.

1. What is meant by looping? Describe any two different forms of looping with Examples.
2. Write a program to print the following outputs using for loop.
   1. 1
3. 2
4. 3 3
5. 4 4 4

|  |  |  |  |
| --- | --- | --- | --- |
| ii) |  | 1 |  |
|  | 2 | 2 |  |
|  | 3 | 3 | 3 |
| 4 | 4 | 4 | 4 |

1. What are the logical operators used in C and illustrate with examples?
2. Whar is the purpose of switch statement ? How does this statement differ from the other statements?
3. State and explain various identifiers in C program. And also discuss about operator precedence in expression evaluation with a suitable example.
4. Explain with a sample program about while, for, do-while and switch statements in C programming
5. Explain the difference between break, goto and continue statements with an example
6. Describe the for loop statement in ‘C’.
7. Explain the difference between while and do-while statements with suitable examples.
8. Write a C program to print digits in reverse order for a given number.
9. Describe type casting with an example.
10. Explain the logical operators with suitable examples.
11. Write C program to print prime numbers in a given series of numbers. For example: numbers from 1 to 100.
12. What is a flowchart? Discuss various symbols used in flowchart. Illustrate with an example.
13. Explain the steps in program development.
14. What is meant by assembly language? Give example. Differentiate between pre test and post test loops.
15. What is the need of type conversion? Discuss type casting. b) List the demerits of go to statement. c) Why is switch statement known as multi way selection?
16. Explain switch statement. Explain its usage with a sample C-program. What are the storage classes in C? Explain their usage with a sample C-program.
17. Explain scope of a variable with an example. Write brief notes on computer languages

**OBJECTIVE QUESTIONS**

1. The C language consist of \_\_\_\_ number of keywords.
2. What will be sum of the binary numbers 1111 and 11001………………
3. Which one of the following is known as the language of the computer [ ]

(A) Programming language

(B) Machine language

(C) High level language

(D) Assembly level language

4. Which of the following is syntactically correct [ ]

(A) for(;); (B) for(); (C) for(,); (D) for(;;);

5. which one of the following is not a translator program [ ]

(A) Assembler (B) Interpreter (C) Linker (D) Compiler

6. . What will be the ASCII Octal value of A [ ]

(A) 100 (B) 101 (C) 110 (D) 111

7. a<<1 IS EQUAL TO \_\_\_\_\_\_\_\_\_\_\_\_\_\_

8. The process of repeating a group of statements in an algorithm is known as\_\_\_\_\_\_\_\_\_\_\_

9. Extend the term CPU \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

10 Monitor, keyboard, mouse and printers are \_\_\_\_\_\_\_\_\_devices

14. C was developed by \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

15. \_\_\_\_\_\_\_\_\_is used to compile your c program

16. Short Integer size is \_\_\_\_ bytes

17. The while loop repeats a statement until the test at the top proves \_\_\_\_\_\_\_

18. The \_\_\_\_\_\_\_\_\_\_statement transfers control to a statement within its body

19. The \_\_\_\_\_\_\_is a unconditional branching statement used to transfer control of the program from one statement to another

20. ANSI stands for \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_