Topperworld.in

PPS

B.Tech (1ST YEAR)

Notes

Prepared By: TOPPERWorld

Unit-1

Overview of Computers

What is computer ?

A computer is an electronic data processing device, which accepts and stores data input, processes the data input, and generates the output in a required format.

The purpose of this tutorial is to introduce you to Computers and its fundamentals.

Functionalities of a Computer

If we look at it in a very broad sense, any digital computer carries out the following five functions –

Step 1 – Takes data as input.

Step 2 – Stores the data/instructions in its memory and uses them as required.

Step 3 – Processes the data and converts it into useful information.

Step 4 – Generates the output.

Step 5 – Controls all the above four steps.

Basic components of computer system

By definition, components of a computer system are the primary elements which make the functioning of an electronic device smooth and faster. There are five basic components which include:

- Input Unit
- Output Unit
- Memory Unit
- Control Unit
- Arithmetical and Logical Unit

- **Input Unit**

A computer will only respond when a command is given to the device. These commands can be given using the input unit or the input devices.

For example: Using a keyboard we can type things on a Notepad and the computer processes the entered data and then displays the output of the same of the screen.

The data entered can be in the form of numbers, alphabet, images, etc. We enter the information using an input device, the processing units convert it into computer understandable languages and then the final output is received by a human-understandable language.

- **Output Unit**

When we command a computer to perform a task, it reverts for the action performed and gives us a result. This result is called output. There are various output devices connected to the computer. The most basic of which is a monitor. Whatever we write using a keyboard or click using a mouse, is all displayed on the monitor.

Thus, the output unit gives us the final result once the entire processing is done within the mechanism of a device.

For example: when we visit an ATM, we enter our details like language, pin, amount to be withdrawn, etc. and then the final money which the cash dispenser releases is our outcome. In this case, the cash dispenser acts as an output unit.

To get a list of computer input and output device and the function of the various I/O devices, visit the linked article.

For better understanding and more interactive analysis of the components of the computer, candidates can check the video given below and get detailed information reading the five major components responsible for the functioning of a computer device.

- **Memory Unit**

When we enter the data into the computer using an input device, the entered information immediately gets saved in the memory unit of the Central Processing Unit (CPU). Because of the presence of some existing programming, the Memory Unit transmits the data further to the other parts of the CPU.

Similarly, when the output of our command is processed by the computer, it is saved in the memory unit before giving the output to the user.

- **Control Unit**

This is the core unit which manages the entire functioning of the computer device. It is one of the most essential components of the computer system.

The Control Unit collects the data entered using the input unit, leads it on for processing and once that is done, receives the output and presents it to the user. It can be said to the centre of all processing actions taking place inside a computer device.

Basically, the instructions taken, interpretation of entered data, issuing signals to execute the data and then finally retrieving the data is all done in the Control Unit.

- **Arithmetic & Logical Unit**

As the name suggests, all the mathematical calculations or arithmetic operations are performed in the Arithmetic and Logical Unit of the CPU.

It can also perform actions like a comparison of data and decision-making actions. The ALU comprises circuits using which addition, subtraction, multiplication, division and other numerical based calculations can be performed.

Block diagram of computer system

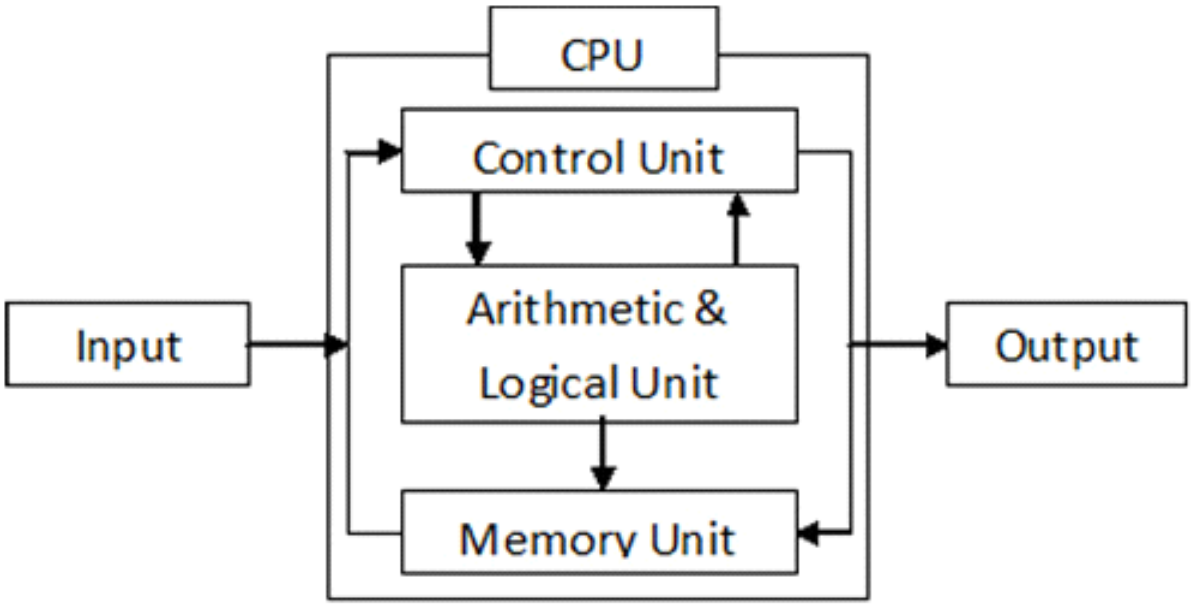


Fig. Block Diagram of Computer

1. Input unit – Input unit is a unit that accepts any input device. The input device is used to input data into the computer system.
Function of input unit:

- It converts inputted data into binary codes.
- It sends data to main memory of computer .

2. Central Processing Unit (CUP) – CPU is called the brain of a computer. An electronic circuitry that carries out the instruction given by a computer program.
CPU can be sub classified into three parts.

i .Control unit (CU)

ii. Arithmetic & Logic unit (ALU)

iii.Memory Unit (MU)

i.Control unit (CU)- the control unit manages the various components of the computer. It reads instructions from memory and interpretation and changes in a series of signals to activate other parts of the computer. It controls and co-ordinate is input output memory and all other units.

ii. Arithmetic & Logic unit (ALU) – The arithmetic logic unit (ALU), which performs simple arithmetic operation such as +, -, *, / and logical operation such as >, <, =<, <= etc.
iii. Memory Unit (MU)- Memory is used to store data and instructions before and after processing. Memory is also called Primary memory or internal memory. It is used to store data temporary or permanently.

Types of memory unit:-

Primary Memory

This is the internal memory that stores the data and instructions of the CPU. It is volatile in nature (data is lost when the power is disconnected).

Secondary Memory

As we know that the primary memory is volatile therefore, we need some devices to store the data permanently so we use some external storage devices for this purpose which we name as the secondary memory. Some examples: CD, DVD,etc.

Function of CPU-

- It controls all the parts and software and data flow of computer.
- It performs all operations.
- It accepts data from input device.
- It sends information to output device.
- Executing programs stored in memory
- It stores data either temporarily or permanent basis.
- It performs arithmetical and logical operations.

- **Output Unit –**Output unit is a unit that constituents a number of output device. An output device is used to show the result of processing.

  Function of Output unit:

- it accepts data or information sends from main memory of computer
- It converts binary coded information into HLL or inputted languages.

Number system:-

When we type some letters or words, the computer translates them in numbers as computers can understand only numbers. A computer can understand the positional number system where there are only a few symbols called digits and these symbols represent different values depending on the position they occupy in the number.

The value of each digit in a number can be determined using –

- The digit
- The position of the digit in the number
- The base of the number system (where the base is defined as the total number of digits available in the number system)

Decimal Number System

The number system that we use in our day-to-day life is the decimal number system. Decimal number system has base 10 as it uses 10 digits from 0 to 9. In decimal number system, the successive positions to the left of the decimal point represent units, tens, hundreds, thousands, and so on.

Each position represents a specific power of the base (10). For example, the decimal number 1234 consists of the digit 4 in the units position, 3 in the tens position, 2 in the hundreds position, and 1 in the thousands position. Its value can be written as

(1 x 1000)+ (2 x 100)+ (3 x 10)+ (4 x l)
$(1 \times 10^3)+ (2 \times 10^2)+ (3 \times 10^1)+ (4 \times l0^0)$
1000 + 200 + 30 + 4
1234

As a computer programmer or an IT professional, you should understand the following number systems which are frequently used in computers.

S.No.	Number System and Description
1	**Binary Number System**

	Base 2. Digits used : 0, 1
2	**Octal Number System** Base 8. Digits used : 0 to 7
3	**Hexa Decimal Number System** Base 16. Digits used: 0 to 9, Letters used : A- F

Binary Number System

Characteristics of the binary number system are as follows –

- Uses two digits, 0 and 1
- Also called as base 2 number system
- Each position in a binary number represents a **0** power of the base (2). Example 2^0
- Last position in a binary number represents a **x** power of the base (2). Example 2^x where **x** represents the last position - 1.

Example

Binary Number: 10101_2

Calculating Decimal Equivalent –

Step	Binary Number	Decimal Number
Step 1	10101_2	$((1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))_{10}$
Step 2	10101_2	$(16 + 0 + 4 + 0 + 1)_{10}$
Step 3	10101_2	21_{10}

Note – 10101_2 is normally written as 10101.

Octal Number System

Characteristics of the octal number system are as follows –

- Uses eight digits, 0,1,2,3,4,5,6,7
- Also called as base 8 number system
- Each position in an octal number represents a **0** power of the base (8). Example 8^0
- Last position in an octal number represents a **x** power of the base (8). Example 8^x where **x** represents the last position - 1

Example

Octal Number: 12570_8

Calculating Decimal Equivalent –

Step	Octal Number	Decimal Number
Step 1	12570_8	$((1 \times 8^4) + (2 \times 8^3) + (5 \times 8^2) + (7 \times 8^1) + (0 \times 8^0))_{10}$
Step 2	12570_8	$(4096 + 1024 + 320 + 56 + 0)_{10}$
Step 3	12570_8	5496_{10}

Note – 12570_8 is normally written as 12570.

Hexadecimal Number System

Characteristics of hexadecimal number system are as follows –

- Uses 10 digits and 6 letters, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- Letters represent the numbers starting from 10. A = 10. B = 11, C = 12, D = 13, E = 14, F = 15
- Also called as base 16 number system
- Each position in a hexadecimal number represents a **0** power of the base (16). Example, 16^0
- Last position in a hexadecimal number represents a **x** power of the base (16). Example 16^x where **x** represents the last position - 1

Example

Hexadecimal Number: $19FDE_{16}$

Calculating Decimal Equivalent –

Step	Binary Number	Decimal Number
Step 1	$19FDE_{16}$	$((1 \times 16^4) + (9 \times 16^3) + (F \times 16^2) + (D \times 16^1) + (E \times 16^0))_{10}$
Step 2	$19FDE_{16}$	$((1 \times 16^4) + (9 \times 16^3) + (15 \times 16^2) + (13 \times 16^1) + (14 \times 16^0))_{10}$
Step 3	$19FDE_{16}$	$(65536 + 36864 + 3840 + 208 + 14)_{10}$
Step 4	$19FDE_{16}$	106462_{10}

Note – $19FDE_{16}$ is normally written as 19FDE.

Arithmetic Of Number System

A **number system** is a set of symbols used to represent values derived from a common base or radix. In a number, the value of each digit can be determined using digit, position of the digit in the number, and the base of the number system. The base is defined as the total number of digits are available in the number system. This is known as positional number system.

Number System	Base	Digit Used
Binary	2	0, 1
Octal	8	0, 1, 2, 3, 4, 5, 6, 7
Decimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Hexadecimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Arithmetic operations like addition, subtraction, multiplication and division are performed in various radix bases. These arithmetic operations may also use r's complement and (r-1)'s complement representation.

These representation techniques hold basic laws for various arithmetic operations:

(i) Unique Existence Law: The sum and product of any two numbers exist uniquely. Where 0 is the identity element for additions and 1 is the identity element for multiplication.

(ii) Associative Law: Addition and multiplication of binary numbers are associative.

(iii) Commutative Law: Addition and multiplication of binary numbers are commutative.

(iv) Distributive Law: Multiplication of binary numbers is distributive over two or more terms in addition.

Now, arithmetic operations for various number systems are explained as following below.

Arithmetic Operations of Binary Numbers:

In the Binary System, there are only two symbols or possible digit values, i.e., 0 and 1. Represented by any device that only 2 operating states or possible conditions. Generally, it is expressed by subscript 2 or binary (B).

Binary arithmetic is essential part of various digital systems. You can add, subtract, multiply, and divide binary numbers using various methods. These operations are much easier than decimal number arithmetic operations because binary system has only two digits: 0 and 1.

Binary additions and subtractions are performed as same in decimal additions and subtractions. When we perform binary additions, there will have two outputs: Sum (S) and Carry (C) . There are four rules for binary addition. These are given as following below,

Input A	Input B	Sum (S) A+B	Carry (C)

0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

There are four rules for binary subtraction. These are given as following below,

Input A	Input B	Subtract (S) A-B	Borrow (B)
0	0	0	0
0	1	0	1
1	0	1	0
1	1	0	0

Borrow 1 is required from next higher order bit to subtract 1 from 0. So, result became 0.

There are four rules for binary multiplication. These are given as following below,

Input A	Input B	Multiply (M) AxB
0	0	0
0	1	0
1	0	0

1	1	1

Whenever at least one input is 0, then multiplication is always 0.

There are four part in any division: Dividend, Divisor, quotient, and remainder. These are given as following rules for binary division,

Input A	Input B	Divide (D) A/B
0	0	Not defined
0	1	0
1	0	Not defined
1	1	1

Whenever divisor is 0, then result is always not defined.

Arithmetic Operations of Octal Numbers:

It has base of 8 digits ranging from 0 to 7 (i.e., 0, 1, 2, 3, 4, 5, 6, 7). Generally, it is expressed by subscript 8 or octal (o).

Arithmetic operations of octal numbers are also performed same as decimal arithmetic operations. The addition table for octal numbers is given as below:

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11

3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

With the help of above table, we can perform additions of octal numbers.

Similarly, we can perform subtraction of octal numbers same as decimal number sutractions. But you can only take borrow of a number 7, which is maximum digit allowed in octal number systems.

In the multiplication of octal numbers, if the product is less than radix of octal (i.e, 8), then we take it as the result, else divide it by radix of octal (i.e., 8) and take the remainder as the LSB (least significant bit). The quotient is taken as carry in the next significant digit.

Similarly, division of octal numbers can be performed by following the rules of division of decimal numbers but maximum allowed digit will be 7.

Arithmetic Operations of Hexadecimal Numbers:

It has base of 16 digits ranging from 0 to F (i.e., 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, and A, B, C, D, E, F). A, B, C, D, E, F are equivalent single digits of 10, 11, 12, 13, 14, 15 respectively. Generally, it is expressed by subscript 16 or Hexa (H) or (h).

Arithmetic operations of hexadecimal numbers can be performed using addition table for hexadecimal numbers which is given as below:

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10

Topperworld.in

2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	7	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

With the help of above table, we can perform additions of hexadecimal numbers.

Subtraction of hexadecimal numbers can be performed by using complement methods or simple as decimal sutractions. The rule of simple hexadecimal subtraction is the digit borrowed from the immediate higher place is counted as 16.

In the multiplication of hexadecimal numbers, if the product is less than radix of hexadecimal (i.e, 16), then we take it as the result, else divide it by radix of hexadecimal (i.e., 16) and take the remainder as the LSB (least significant bit). The quotient is taken as carry in the next significant digit. Using these rules, you can make a table for hexadecimal multiplications.

Similarly, division of hexadecimal numbers can be performed by following the rules of division of decimal numbers but maximum allowed digit will be F(=15 in decimal).

Computer Hardware

Computer hardware is a collective term used to describe any of the physical components of an analog or digital computer. The term *hardware* distinguishes the tangible aspects of a computing device from software, which consists of written, machine-readable instructions or programs that tell physical components what to do and when to execute the instructions.

Hardware and software are complementary. A computing device can function efficiently and produce useful output only when both hardware and software work together appropriately.

Computer hardware can be categorized as being either internal or external components. Generally, internal hardware components are those

necessary for the proper functioning of the computer, while external hardware components are attached to the computer to add or enhance functionality.

What are internal computer hardware components?

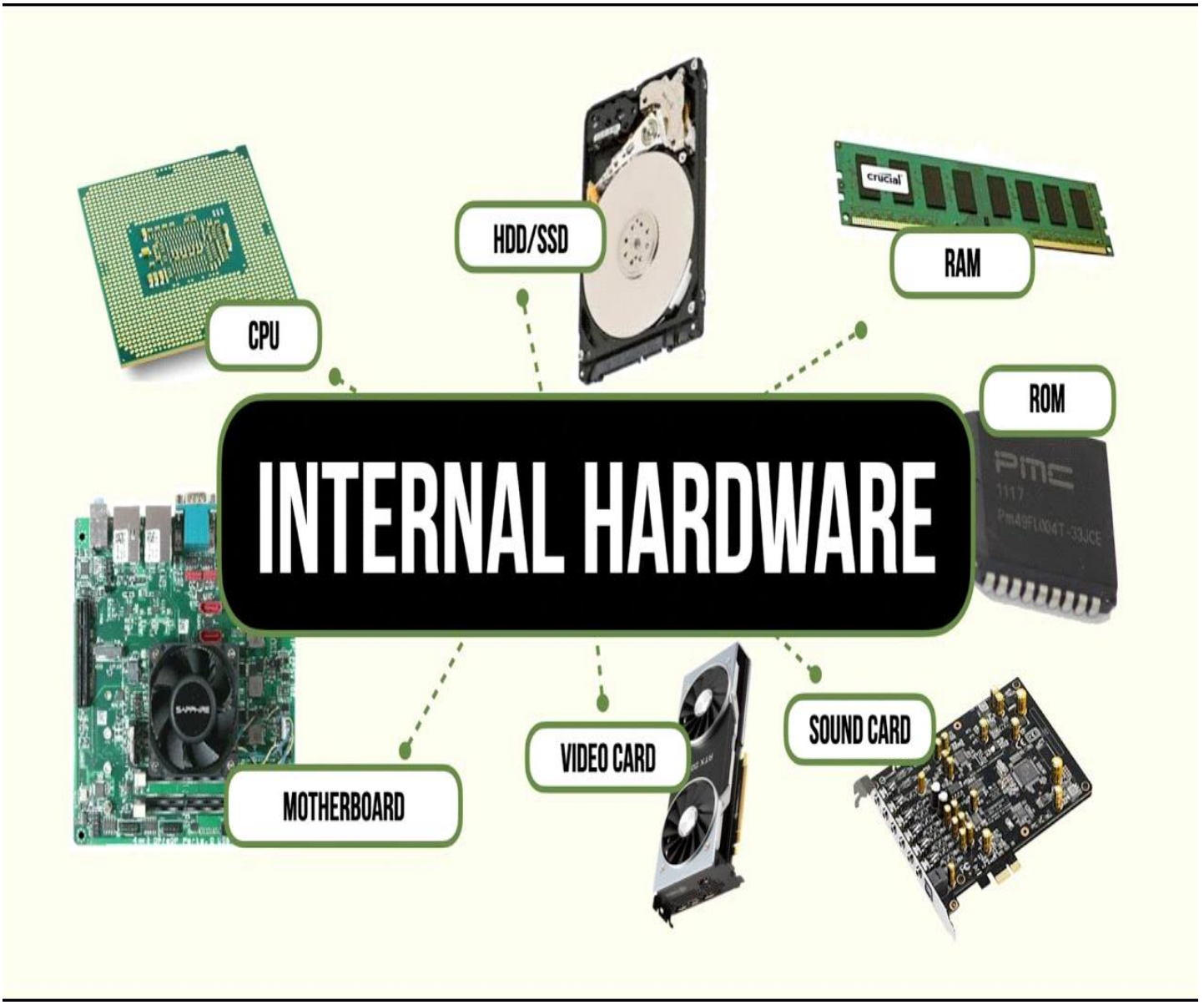
Internal components collectively process or store the instructions delivered by the program or operating system (OS). These include the following:

- **Motherboard.** This is a printed circuit board that holds the central processing unit (CPU) and other essential internal hardware and functions as the central hub that all other hardware components run through.
- **CPU.** The CPU is the brain of the computer that processes and executes digital instructions from various programs; its clock speed determines the computer's performance and efficiency in processing data.
- **RAM.** RAM -- or dynamic RAM -- is temporary memory storage that makes information immediately accessible to programs; RAM is volatile memory, so stored data is cleared when the computer powers off.
- **Hard drive.** Hard disk drive are physical storage devices that store both permanent and temporary data in different formats, including programs, OSes, device files, photos, etc.

- **Solid-state drive (SSD).** SSDs are solid-state storage devices based on NAND flash memory technology; SSDs are non-volatile, so they can safely store data even when the computer is powered down.
- **Optical drive.** Optical drives typically reside in an on-device drive bay; they enable the computer to read and interact with nonmagnetic external media, such as compact disc read-only memory or digital video discs.
- **Heat sink.** This is a passive piece of hardware that draws heat away from components to regulate/reduce their temperature to help ensure they continue to function properly. Typically, a heat sink is installed directly atop the CPU, which produces the most heat among internal components.
- **Graphics process unit.** This chip-based device processes graphical data and often functions as an extension to the main CPU.
- **Network interface card (NIC).** A NIC is a circuit board or chip that enables the computer to connect to a network; also known as a *network adapter* or local area network *adapter*, it typically supports connection to an Ethernet network.

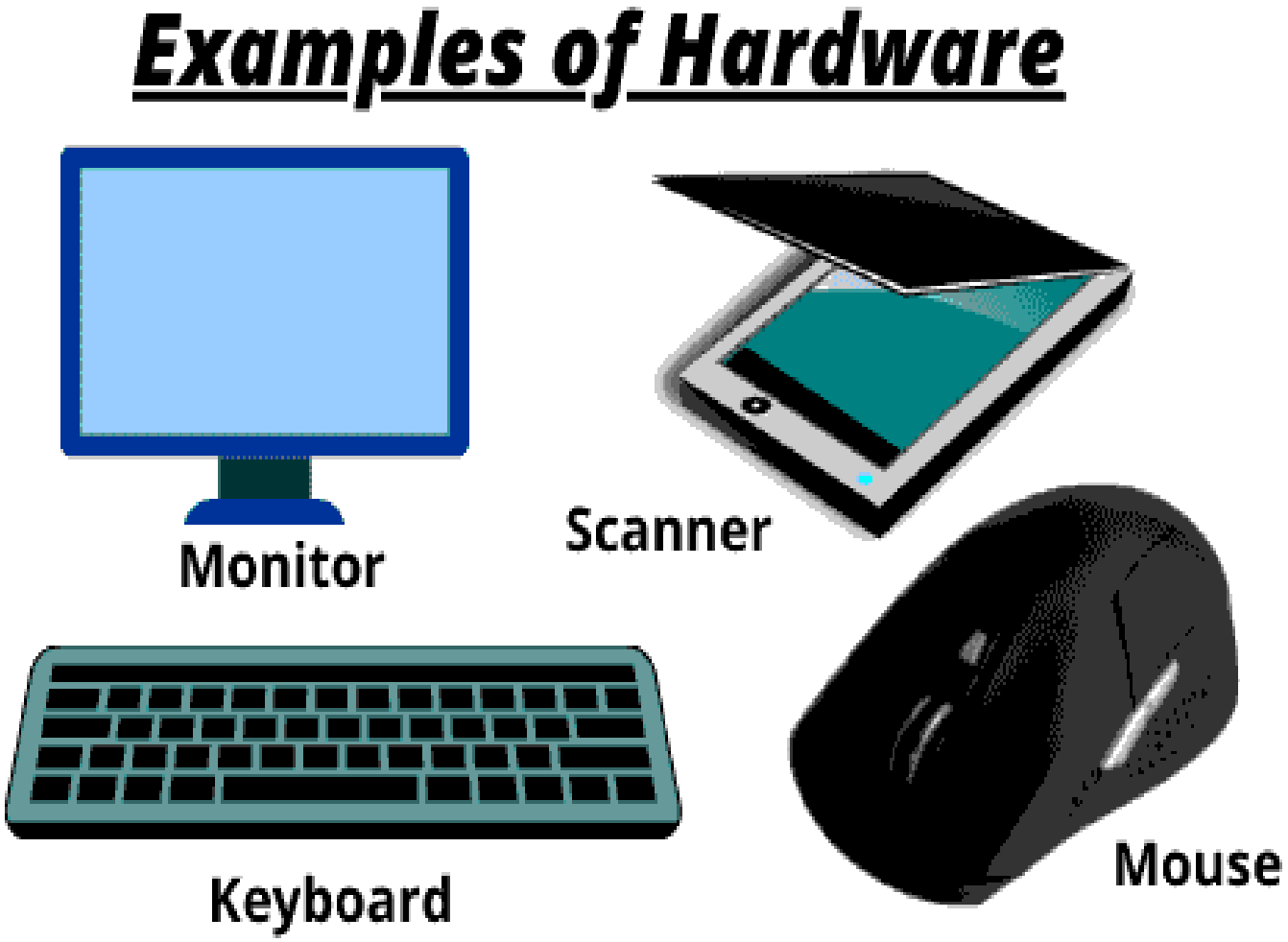
Other computing components, such as USB ports, power supplies, transistors and chips, are also types of internal hardware.

This computer hardware chart below illustrates what typical internal computer hardware components look like.



What are external hardware components?

External hardware components, also called *peripheral components*, are those items that are often externally connected to the computer to control either input or output functions. These hardware devices are designed to either provide instructions to the software (input) or render results from its execution (output).



Common input hardware components include the following:

- **Mouse.** A mouse is a hand-held pointing device that moves a cursor around a computer screen and enables interaction with objects on the screen. It may be wired or wireless.
- **Keyboard.** A keyboard is an input device featuring a standard QWERTY keyset that enables users to input text, numbers or special characters.
- **Microphone.** A microphone is a device that translates sound waves into electrical signals and supports computer-based audio communications.
- **Camera.** A camera captures visual images and streams them to the computer or through a computer to a network device.
- **Touchpad.** A touchpad is an input device, external or built into a laptop, used to control the pointer on a display screen. It is typically an alternative to an external mouse.
- **USB flash drive.** A USB flash drive is an external, removable storage device that uses flash memory and interfaces with a computer through a USB port.
- **Memory card.** A memory card is a type of portable external storage media, such as a Compactflash card, used to store media or data files.

Other input hardware components include joysticks, styluses and scanners.

Examples of output hardware components include the following:

- **Monitor.** A monitor is an output device similar to a TV screen that displays information, documents or images generated by the computing device.

Printer

Printers render electronic data from a computer into printed material. A printer is a hardware output device that is used to generate hard copy and print any document. A document can be of any type such as a text file, image, or the combination of both. It accepts input command by users on a computer or on other devices to print the documents. For example, if you have to submit a project report at your college, you need to create a soft copy of your report and print it with the help of the printer.

Printers are one of the common computer peripheral devices that can be classified into two categories that are **2D and 3D printers**. The 2D printers are used to print text and graphics on a paper, and 3D printers are used to create three dimensional physical objects.

Types of printer

Although there are different types of printers, nowadays, two types of printers are commonly used, which are inkjet and laser printers. A list of all the various types of printers is given below:

- Inkjet Printers
- Laser Printers
- 3D Printers
- LED Printers
- Solid Ink Printers
- Dot Matrix Printers
- Multifunction or All-in-One Printers
- Thermal printer
- Plotter

Inkjet Printers

It is widely used by home and business computer users that prints characters by spraying the ink using magnetic plates on the paper. It contains a **paper feed assembly, ink cartridge, print head, stabilizer bar, and belt.** It stores the ink in cartridges, and uses separate cartridge to print several types of color documents. These colors are a combination of **cyan, magenta, yellow, and black color.** These types of printers have the ability to create high-quality pictures with the help of vivid colors. Furthermore, the inkjet printers are more affordable and easier to use as compared to other printers.

Advantages of Inkjet Printers:

- The Inkjet printers have the ability to produce high-quality output.
- These printers are reasonably fast and easy to use.
- Additionally, these types of printers do not take warm up time.

Disadvantages of Inkjet Printer:

- It may take more time to print.
- Its running cost is high.
- It does not allow the highlighter marker.
- Sometimes, it can produce a wrong warning of an empty cartridge.

Laser printers

The laser printer is one of the common personal computer printers. It was introduced in 1971, and after that it was developed at Xerox PARC by Gary Starkweather. It uses the laser or non-impact photocopier technology to print the text and images on the paper. Whenever it gets input to print any document, a laser beam draws the document on the selenium-coated drum with the help of electric charges.

- When the drum is charged, it is rolled in toner (dry ink powder). The ink follows the image, which has charged on the drum. The ink is

combined with the paper, including heat and pressure, then transferred on a chunk (piece) of paper.

When the document is printed, the excess toner is collected, and an electric charge is removed from the drum. Most of the laser printers are capable of printing only in monochrome. The monochrome laser printer is around ten times cheaper than a color laser printer.

Laserjet Printer

There are many differences between a laser printer and inkjet printer, such as:

- The laser printer contains dry ink, while an inkjet contains **wet ink.**
- An inkjet printer is approximately ten times more expensive than the laser printer as it requires replacing the ink very frequently.
- If a paper is wet, the inkjet printer will print the document with blur, but the laser printer will print clear. The inkjet printer is suitable for printing fewer documents, while the laser printer has the ability to print more documents.

Advantages of a Laser printer:

- These types of printers have a higher paper capacity.
- It is less expensive than the inkjet printer.
- It has the ability to print documents speedily.

- Furthermore, it is able to increase productivity.

Disadvantages of a Laser printer:

- Laser printers may need warm up time.
- Laser printers are bulky as they required the laser technology and imaging drum.
- It requires high voltage leads to small carbon emissions.

3D Printers

One of the best enhancements in the history of printing technology is the 3d printer, which was developed by **Chuck Hull** in **1984**. It produces 3D objects and items by using **quality resin**. It uses materials like plastics, polymers, metal alloys, or even food ingredients.

Application of 3D printers

There are many applications where 3D printers are used, such as **archaeology, aerospace engineering, information systems,**

dentistry, and biotechnology. For example, it might be used to reconstruct ancient artifacts of archaeology physically, which have been destroyed with time.

Advantages of 3D printer:

- The main advantage of a 3D printer is that it allows users to print objects in 3D .
- It has the ability of full customization.
- It is easy to access and cost-effective.
- It prints the documents with better quality.
- It provides users unlimited shapes and geometry.

Disadvantages of 3D printer:

- Its initial and resin costs are high.
- 3D printing is still developing technology.
- It consumes high energy around 50 to 100 more than injection molding.
- It includes limited materials
- The 3D printers are slow as they are limitless for mass customization.

LED Printers

The LED printer is mostly like a laser printer. It uses the drum, ink, and fuser system to print the black or colored documents. Initially, Casio and function developed the LED printers by focusing light throughout the whole length of the drum. These types of printers are non-impact, but instead of using a laser in the printhead they use a light-emitting diode. These printers were invented by the manufacturer OKI in 1989.

There is a difference between laser printers and LED printers; the LED printers use a strip of LEDs to create an ink-attracting static charge on the printing drum while laser printers use a laser and mirror. Although LED printers are not more popular as compared to inkjet and laser printer, they are gently gaining popularity as they provide users the unique benefits.

Advantages of the LED printer:

- The manufacturing of LED printers is cheaper than laser printers.
- These types of printers mostly contain free warranty extensions.
- These printers are capable of printing on thick 3D items.

Solid Ink Printers

A laser class solid ink printer is designed to save space and money on packaging. It uses a unique type of ink technology that is melted into a liquid before being used. As inkjet printers stream the ink directly on the paper, but solid ink printers are different as they stream the ink on the drum. Firstly, transferring the ink to the drum than to the printer is a better way to get a good color combination as a drum can be controlled appropriately than the

moving paper. Xerox acquired the color printer division of Tektronix, Inc., in 2000. The solid ink printer was introduced in the market by Tektronix, Inc., with its crayonlike cartridges.

Advantages of solid ink printer:

- The print quality of a solid ink printer is good as it produces dynamic colors and extraordinary print quality on a broad range of media like envelopes, recycled paper, transparencies, and cardstock.
- The solid ink printers are easy to handle because these are made by non-toxic vegetable oils and do not melt in our hands. Unlike laser or inkjet printers, they do not spill or leak.
- These types of printers are capable of printing the documents fast. Generally, it can print up to 30 pages per minute.
- The ink loading in the solid printer is an easy as it includes sticks in particular shapes that only go to the correct slot. These types of printers contain the ink slot on the top of the printer, and they allow you to replace the ink at any time, including at the time of printing.
- The solid ink printers are designed in that way; they comprise only three main assemblies, print drum, print head, and controller. It takes less space as compared to the color laser printer.
- Furthermore, these printers and their parts are affordable than the color toner cartridges and color laser printers.

Disadvantages of a solid ink printer:

- These printers may require warm-up and cool downtime.

- These are sensitive to UV light, and color may fade under the sunlight with the time.
- These printers consume more power than the laser printer
- These printers are mainly used for a large numbers of printing. If it is turned off in the middle of the printing, it will take some time to warm up again.

Dot Matrix Printers

The dot matrix printers are also known as a pin printer that was released by **IBM in 1957**. Although, in 1970, Centronics created the first dot-matrix impact printer. It strikes an ink ribbon using print heads that place thousands of little dots to form images and text. Nowadays, as compared to laser and inkjet printers, it is less used, as its printing speed slow and generates lower quality images. However, dot matrix printers are still in use in some sectors like package delivery companies and auto part stores.

Advantages of Dot matrix printer:

- A dot matrix printer is less expensive as compared to most of the printers, and it is easily available in the market.
- It has the ability to print carbon copies of a particular print out, unlike non-impact printers.

- Its printing cost is lowest than the other printers.
- Furthermore, its maintenance cost is less expensive than the other printers.

Disadvantages of Dot matrix printer:

- The speed of the dot matrix printer is slow as compared to non-impact printers, and its output is also not of high resolution.
- It creates high noise when the pins hit the ribbon to the paper.
- It may be a time-consuming printer as it requires a single paper to wound up and aligned manually.

Multifunction Printers

Multifunction printer is a hardware device that can perform various tasks such as **printing, scanning, faxing, and copying**. It is also called the All-in-One printer. It is suitable for budget-minded businesses that want to reduce costs, consolidate assets, and improve workflow. Although it is necessary to connect the printer to a computer with the help of a wire or wireless connection, sometimes it can be controlled from their control panel like standalone copiers. Furthermore, no need for more than one unit as it can easily complete multiple tasks within an office or home simultaneously.

Advantages of Multifunction Printers:

- The multifunction printers can save initial cost, as instead of purchasing many printers and fax machines, you can purchase only one large MFG for multipurpose.
- These types of printers are beneficial for home and office as they save a lot of floor space. It may be difficult to find space for the fax machine, printer, and scanner, but a multifunction printer takes limited space and can perform multi-task such as a printing, faxing, and scanning.
- One of the best advantages of a multifunction printer, the operation functionality is faster as compared to laser printers or other printers.

Also, some of the multifunction printers can send print documents, faxes, scan images, and copy all at the same time.

- Furthermore, these printers need only one cable to power the whole setup, which reduces the congestion and electricity required to run the device. Thus, the multifunction printer consumes less power as compared to other printers.

Disadvantages of Multifunction Printers:

- The operating cost of a multifunction printer is high. Its maintenance cost is also expensive as it uses ink speedily as compared to other printers. All functions, such as faxing, copying, and printer, use ink faster.
- These printers perform a task on the basis of first in first out rule. So all tasks stack up in a queue that can cause slow the machine.Thus, some people may have to wait to use the machine.
- When the multifunction printer is out of service or broken down, the all work scanning, printing, copying, faxing will be completely stopped. It can be an issue for an organization, if they have more requirements for faxing, copying, and printing the documents.

Thermal printer

The thermal printer is invented by **Jack Kilby** that is also known as an electrothermal printer, thermal transfer printer, or thermal wax-transfer printer. It utilizes heated pins in order to produce the image on paper. These types of printers are widely used in banking, airline, grocery, entertainment, retail, healthcare industries, fax, and calculator machines.These printers are low-cost and print fast, and do not use ink like other printers. They primarily depend on the thermal paper to produce the images.

This technology is useful for an organization as it is reliable as well as cost-effective. Employees can work continuously without getting interrupted because they do not need to change the cartridges or ribbons in this printer.

Advantages of Thermal printer:

- One of the main advantages of a thermal printer, it does not need any ribbons or cartridges.Thus, the companies can save the time by using it.
- These types of printers are easy to use as they have fewer buttons, including the use of software.
- These are beneficial for offices as they provide noise-free environments.
- These printers are cheaper and available in several models and sizes.
- These printers are faster and more efficient in printing monochromic ones than the other printers.

Disadvantages of Thermal printer:

- Generally, a thermal printer does not produce colors print out well, like standard printers.
- At the time of printing, the high heat can be harmful to the printhead. Consequently, if the printhead breaks down, you have to pay for a repair or buy a new one.

Plotter

A plotter is a hardware device or a printer, which was first invented by **Remington-Rand** in **1953**. It is developed to print vector graphics and uses a pencil, pen, marker, or other writing tools for drawing continuous lines instead of using ink or tonner. It is widely used to print a hard copy of schematics and other similar applications.

There are several types of plotter given in the below list:

- **Drum plotter:** It is also known as roller plotter. It turns the paperback and forth on a drum while pens move left and right. With the help of combining these two directions, the line can be drawn in any direction.
- **Flatbed plotter:** In flatbed plotter, a paper is placed on a large horizontal surface.A traveling bar moves across the surface and draws the lines.
- **Electrostatic plotter:** It draws on paper that is charged negatively with positively charged ink or tonner.
- **Inkjet Plotter:** It is a type of Pinch Roller Plotter, which produces an image with the help of showering little drops of ink on the paper.
- **Cutting Plotter:** It is a new generation device that allows users to cut different shapes. It is mainly used for advertising, sign-making, bill board, and vehicle graphics.

Advantages of Plotter:

- A Plotter is capable of working on the large sheet around 2 feet or more with better quality.
- It can print on sheet, steel, plastic, aluminum, plywood, and any flat sheet material, including paper.
- It has the ability to save all templates and patterns on the disk. Thus, it can overcome the problem to load the same pattern again and again.
- Furthermore, it can draw the same pattern thousand times without any degradation.

Disadvantages of Plotter:

- The size of a plotter is very large than the traditional printers**.**
- Plotters are more expensive as compared to a traditional printer.

Storage Devices

Magnetic storage devices

Nowadays, magnetic storage is commonly founded on hybrid hard drives or extremely large HDDs.

A list is given below of magnetic storage devices:

- **Floppy diskette:** A floppy disk drive (FDD) offers users the benefit of saving data to removable diskettes. FDDs have been replaced with other storage devices like network file transfer and USB.
- **Hard drive:** A hard disk drive (HDD) is used to store data permanently as it is a non-volatile computer storage device, and directly connected

to the disk controller of the computer's motherboard. Usually, it is installed internally in a computer, known as secondary storage device.

- **Magnetic Card:** A magnetic card is a card that may have information about an individual, such as passcodes to enter secure buildings or available recognition on a credit card.
- **SuperDisk:** Imation Corporation developed the disk storage technology. SuperDisk is also known as a LS-240 and LS-120. The drive was most popular with OEM computers and able to store up to 120 MB (Megabyte) on a single disk. Later, it was capable of storing 240 MB and also backward compatible with 1.44 MB disks.
- **Tape cassette:** A tape is a rectangular and flat container that is capable of storing data. As compared to other storage media, it is less expensive and commonly used for backing up a huge amount of data.
- **Zip diskette:** A Zip drive is a hardware data storage device that is an advanced version of the floppy disk. Its functions like a diskette and standard 1.44" floppy drive and developed by Iomega. It became very popular in the late 1990s and capable of storing data that was not possible with ordinary floppy disks.

Optical storage devices

Another type of storage devices are given below:

- Blu-ray disc
- CD-ROM disc
- CD-R and CD-RW disc.
- DVD-R, DVD+RW, DVD+R, and DVD-RW disc.

Flash memory devices

Flash memory is cheaper as well as portable. Due to become more reliable and efficient solution, most magnetic and optical media have replaced by flash memory device.

- **Flash drive:** A USB flash drive is a portable storage device used for data storage that is also known as pen drive, thumb drive, data stick, keychain drive. They are connected to a computer via a USB port and often the size of a human thumb.
- **Memory card:** A memory card is commonly used in digital cameras, printers, MP3 players, PDAs, digital camcorders, game consoles, and handheld computers. The most common memory card format was

CompactFlash for many years, but today are CFexpress, SD, MicroSD, and XQD.

- **Compact Flash (CF):** Compact Flash is a type of flash memory that is commonly found in digital cameras, PDAs and other portable devices. It is a 50-pin connection storage device that is capable of storing data ranging from 2 MB to 128 GB.
- **2:** M.2 is a solid-state drive, introduced in 2014.There are two types of M.2 devices; SATA M.2 and PCIe M.2.
- **MultiMediaCard:** A MultiMediaCard or MMC is an Integrated Circuit that is used in car radios, printers, PDAs, MP3 players, and digital cameras. It acts as external storage for data. The MMCP (MMCplus) and MMCM (MMCmobile / MMCmicro) are the variations of the MMC card.
- **SDHC card (Secure Digital High Capacity):** It uses new technology and an improved version of the standard SD card. It is not backward compatible with SD format devices and has the ability to store data in a range from 4 GB to 32 GB.
- **NVMe (Non-Volatile Memory Express):** It is a device specification that is designed for data centers and standardizes the way devices like SSDs can be connected to a computer's PCI Express bus, introduced by Intel in 2007.
- **Sony Memory Stick:** Sony Memory Stick is a family of flash memory cards, first introduced by Sony in October 1998. It is designed for digital storage in cameras and other Sony products.

- **SmartMedia Card:** A SmartMedia card is a memory card developed by Toshiba that is also known as a solid-state floppy disk card.
- **xD-Picture Card (EXtreme Digital Picture Card):** The xD-Picture Card is a flash memory card that is introduced by Olympus and Fuji in 2002. In **2003**, until the Mini SD card was introduced, the xD cards were the smallest flash memory cards available on the market. H and M/M+ versions of the xD-Picture Card had a capacity of up to 2 GB and the original version up to 512 MB.
- **SSD:** An SSD is a storage medium that is similar to a hard disk drive (HDD). Even without power, it has the ability to maintain stored data in a permanent state. It has higher reliability, noiseless operation, lower power consumption, and faster access time because it has no moving parts, unlike a hard drive.
- **SD Card:** An SD Card, stands for Secure Digital Card, is most commonly used with electronics that are designed to offer high-capacity memory with small size. It is often used in small portable devices like cell phones, digital cameras, digital video camcorders, mp3 players, etc. It is used by more than 400 brands of electronic devices.

Online and cloud storage

The need to store data online and in cloud storage is increasing rapidly.

- **Cloud storage:** Cloud storage is a cloud computing model that transmits and holds data on remote storage systems where a cloud computing provider manages, maintains, and made available data to

users over a network. It offers users the reliability, confidentiality, durability, and 'access data anytime'.

- **Network media:** Network media is used on a computer network such as the Internet, as it is any audio, video, images or text.

Paper storage

Initially, computers were not able to store data on any storage technologies, like flash memory devices, optical storage devices; they had to depend on paper. In modern times, the method of paper storage to store data is rarely used or found.

- **Punch card:** A punch card is also known as Hollerith cards or IBM cards that are able to store data in the form of small punched holes. It is a simple piece of paper stock that was widely used to input data into early computers.
- **OMR:** It stands for optical mark recognition or optical mark reading. It is a method of extracting data from human beings by identifying certain

markings on a document, such as checkboxes and fill-infields, on printed forms. Generally, the OMR process is accomplished by scanning that detects a reflection or transmission with the help of a piece of paper. This technology provides advantages for applications such as ballots, reply cards, surveys, and questionnaires as they need a large amount of hand-filled forms to be processed quickly and with accuracy.

Introduction to Programming Language

Programming Language

As we know, to communicate with a person, we need a specific language, similarly to communicate with computers, programmers also need a language is called Programming language.

Before learning the programming language, let's understand what is language?

What is Language?

Language is a mode of communication that is used to **share ideas, opinions with each other**. For example, if we want to teach someone, we need a language that is understandable by both communicators.

What is a Programming Language?

A programming language is a **computer language** that is used by **programmers (developers) to communicate with computers**. It is a set of instructions written in any specific language (C, C++, Java, Python) to perform a specific task.

A programming language is mainly used to **develop desktop applications, websites, and mobile applications**.

Types of programming language

1. Low-level programming language

Low-level language is **machine-dependent (0s and 1s)** programming language. The processor runs low- level programs directly without the need of a compiler or interpreter, so the programs written in low-level language can be run very fast.

Low-level language is further divided into two parts -

i. Machine Language

Machine language is a type of low-level programming language. It is also called as **machine code or object code**. Machine language is easier to read because it is normally displayed in binary or hexadecimal form (base 16) form. It does not require a translator to convert the programs because computers directly understand the machine language programs.

The advantage of machine language is that it helps the programmer to execute the programs faster than the high-level programming language.

ii. Assembly Language

Assembly language (ASM) is also a type of low-level programming language that is designed for specific processors. It represents the set of instructions in a **symbolic and human-understandable form**. It uses an assembler to convert the assembly language to machine language.

The advantage of assembly language is that it requires less memory and less execution time to execute a program.

2. High-level programming language

High-level programming language (HLL) is designed for **developing user-friendly software programs and websites**. This programming language requires a compiler or interpreter to translate the program into machine language (execute the program).

The main advantage of a high-level language is that it is **easy to read, write, and maintain**.

High-level programming language includes **Python, Java, JavaScript, PHP, C#, C++, Objective C, Cobol, Perl, Pascal, LISP, FORTRAN, and Swift programming language**.

A high-level language is further divided into three parts -

i. Procedural Oriented programming language

Procedural Oriented Programming (POP) language is derived from structured programming and based upon the procedure call concept. It divides a program into small procedures called **routines or functions**.

Procedural Oriented programming language is used by a software programmer to create a program that can be accomplished by using a programming editor like IDE, Adobe Dreamweaver, or Microsoft Visual Studio.

The advantage of POP language is that it helps programmers to easily track the program flow and code can be reused in different parts of the program.

The advantage of POP language is that it helps programmers to easily track the program flow and code can be reused in different parts of the program.

Example: C, FORTRAN, Basic, Pascal, etc.

ii. Object-Oriented Programming language

Object-Oriented Programming (OOP) language is **based upon the objects**. In this **programming language, programs are divided into small parts called objects**. It is used to implement real-world entities like inheritance, polymorphism, abstraction, etc in the program to makes the program resusable, efficient, and easy-to-use.

The main advantage of object-oriented programming is that OOP is faster and easier to execute, maintain, modify, as well as debug.

Note: Object-Oriented Programming language follows a bottom-up approach.

Example: C++, Java, Python, C#, etc.

iii. Natural language

Natural language is a **part of human languages** such as English, Russian, German, and Japanese. It is used by machines to understand, manipulate, and interpret human's language. It is used by developers to **perform tasks such as translation, automatic summarization, Named Entity Recognition (NER), relationship extraction, and topic segmentation**.

The main advantage of natural language is that it helps users to ask questions in any subject and directly respond within seconds.

3. Middle-level programming language

Middle-level programming language **lies between the low-level programming language and high-level programming language**. It is also known as the intermediate programming language and pseudo-language.

A middle-level programming language's advantages are that it supports the features of high-level programming, it is a user-friendly language, and closely related to machine language and human language.

Example: C, C++, language

Most commonly used Programming Language

As we all know, the programming language makes our life simpler. Currently, all sectors (like education, hospitals, banks, automobiles, and more) completely depend upon the programming language.

There are dozens of programming languages used by the industries. Some most widely used programming languages are given below -

1. Python

Python is one of the most widely used user-friendly programming languages. It is an open-source and easy to learn programming language developed in the 1990s. It is **mostly used in Machine learning,Artificial intelligence, Big Data, GUI based desktop applications, and Robotics**.

Advantages

- Python is easy to read, easy to understand, and easy to write.
- It integrates with other programming languages like C, C++, and Java.
- Python executes code line-by-line, so it is easy for the programmer to find the error that occurred in the code.
- Python is platform-independent means you can write code once and run it anywhere.

Disadvantages

- Python is not suitable for developing mobile applications and games.
- Python works with the interpreter. That's why it is slower than other programming languages like C and C++.

2. Java

java is a simple, secure, platform-independent, reliable, architecture-neutral high-level programming language **developed by Sun Microsystems in 1995**. Now, Java is owned by Oracle. It is mainly used to develop bank, retail, information technology, android, big data, research community, web, and desktop applications.

Advantages

- Java is easy to write, compile, learn, and debug as compared to other programming languages.
- It provides an ability to run the same program on different platforms.
- It is a highly secured programming language because in java, there is no concept of explicit pointers.
- It is capable of performing multiple tasks at the same time.

Disadvantages

- Java consumes more memory and slower than other programming languages like C or C++.
- It does not provide a backup facility.

3. C

C is a **popular, simple, and flexible general-purpose computer programming language. Dennis M Ritchie develops it in 1972** at AT&T. It is a combination of both low-level programming language as well as a high-level programming language. It is used to design applications like **Text Editors, Compilers, Network devices, and many more**.

Advantages

- C language is easy to learn.
- It is fast, efficient, portable, easy to extend, powerful, and flexible programming language.
- It is used to perform complex calculations and operations such as MATLAB.
- It provides dynamic memory allocation to allocate memory at the run time.

Disadvantages

- In the C programming language, it is very difficult to find the errors.
- C does not support the concepts of constructors, destructors, abstraction, polymorphism, encapsulation, and namespace like OOPs.

4. C++

C++ is one of the thousands of programming languages that we use to develop software. C++ programming language is developed by **Bjarne Stroustrup in 1980**. It is similar to the C programming language but also includes some additional features such as **exception handling, object-oriented programming, type checking, etc**.

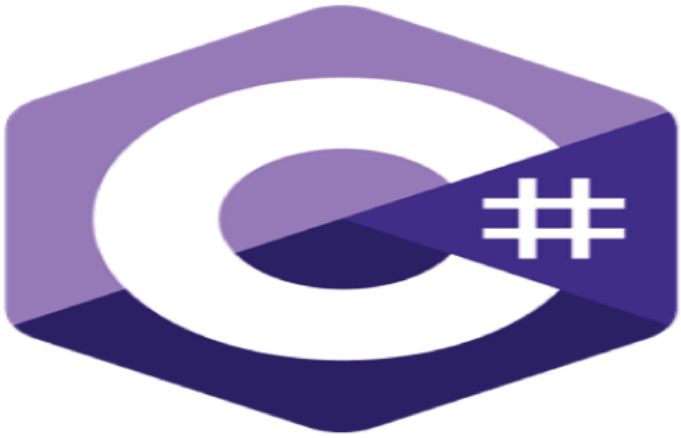
Advantages

- C++ is a simple and portable structured programming language.
- It supports OOPs features such as Abstraction, Inheritance, Encapsulation.
- It provides high-level abstraction and useful for a low-level programming language, and more efficient for general-purpose.
- C++ is more compatible with the C language.

Disadvantages

- C++ programming language is not secured as compared to other programming languages like Java or Python.
- C++ can not support garbage collection.
- It is difficult to debug large as well as complex web applications.

5. C#



C# (**pronounced as C sharp**) is a modern, general-purpose, and object-oriented programming language used with XML based Web services on the .NET platform. It is mainly designed to improve productivity in web applications. It is easier to learn for those users who have sufficient knowledge of common programming languages like C, C++, or Java.

Advantages

- C# is a modern, type-safe, easy, fast, and open-source programming language that is easily integrated with Windows.
- The maintenance of C# (C sharp) is lower than the C++ programming language.
- C# is a pure object-oriented programming language.
- C# includes a strong memory backup facility. That's why it avoids the problem of memory leakage.

Disadvantages

- C# is less flexible because it is completely based on Microsoft .Net framework.
- In C#, it is difficult to write, understand, debug, and maintain multithreaded applications.

6. JavaScript

javascript is a type of **scripting language** that is used on both client-side as well as a server-side. It is developed in the **1990s** for the Netscape Navigator web browser. It allows programmers to implement complex features to make web pages alive. It helps programmers to create dynamic websites, servers, mobile applications, animated graphics, games, and more.

Advantage

- JavaScript helps us to add behavior and interactivity on the web page.
- It can be used to decrease the loading time from the server.
- It has the ability to create attractive, dynamic websites, and rich interfaces.
- JavaScript is a simple, versatile, and lightweight programming language.
- JavaScript and its syntax are easy to understand.

Disadvantage

- JavaScript is completely based on the browser.
- It does not support multiple inheritance.
- It is less secure compared to other programming languages.

7. R

Currently, R programming is one of the popular programming languages that is used in **data analytics, scientific research, machine learning algorithms, and statistical computing.** It is developed in 1993 by Ross Ihaka and Robert Gentleman. It helps marketers and data scientists to easily analyze, present, and visualize data.

Advantages

- R programming provides extensive support for Data Wrangling.
- It provides an easy-to-use interface.
- It runs on any platform like Windows, Linux, and Mac.
- It is an open-source and platform-independent programming language.

Disadvantages

- R programming does not support 3D graphics.
- It is slower than other programming languages.

8. PHP

PHP stands for **Hypertext Preprocessor**. It is an open-source, powerful server-side scripting language mainly used to create static as well as dynamic websites. It is developed by **Rasmus Laird in 1994**. Inside the php, we can also write HTML, CSS and javascript code. To save php file, file extension .php is used.

Advantages

- PHP is a more secure and easy-to-use programming language.
- It supports powerful online libraries.
- It can be run on a variety of operating systems such as Windows, Linux, and Mac.
- It provides excellent compatibility with cloud services.

Disadvantages

- PHP is not capable of handling a large number of applications and not suitable for large applications.
- It is quite difficult to maintain.

9. Go

Go or Golang is an **open-source programming language**. It is used to build simple, reliable, and efficient software. It is developed by **Robert Griesemer, Rob Pike, and Ken Thompson in 2007**.

Advantages

- Go language is easy-to-learn and use.
- It comes with the in-built testing tools.
- Go is a fast programming language.

Disadvantages

- Go language does not support generics.
- It does not support error handling.
- It supports a lack of frameworks.

10. Ruby

Ruby is an open-source, general-purpose, and pure object-oriented programming language released in **1993**. It is used in front-end and back-end web development. It is mainly **designed to write CGI (Common Gateway Interface) scripts**.

Advantages

- Ruby supports various GUI (Graphical User Interface) tools like GTK and OpenGL.
- It is used to develop both internet as well as intranet applications.
- The code written in Ruby is small and contains less number of lines.

Disadvantages

- Ruby is slower than other programming languages.
- It is very difficult for programmers to debug the code written in Ruby.

Introduction to Compiler

- A compiler is a translator that converts the high-level language into the machine language.
- High-level language is written by a developer and machine language can be understood by the processor.
- Compiler is used to show errors to the programmer.

- The main purpose of compiler is to change the code written in one language without changing the meaning of the program.
- When you execute a program which is written in HLL programming language then it executes into two parts.
- In the first part, the source program compiled and translated into the object program (low level language).
- In the second part, object program translated into the target program through the assembler.

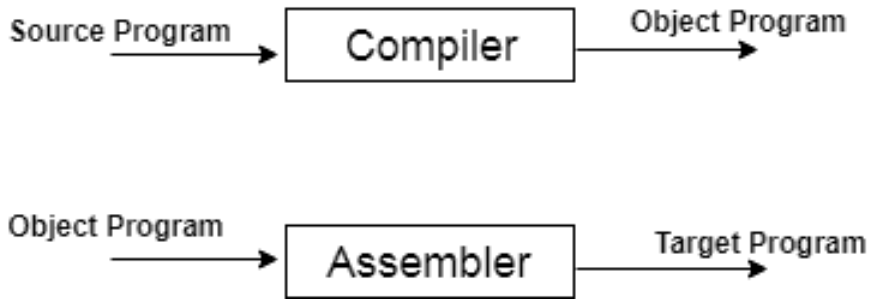


Fig: Execution process of source program in Compiler

Introduction to Interpreter

All high level languages need to be converted to machine code so that the computer can understand the program after taking the required inputs. The software by which the conversion of the high level instructions is performed line-by-line to machine level language, other than compiler and assembler, is known as **INTERPRETER**. If an error is found on any line, the execution stops till it is corrected. This process of correcting errors is easier as it gives line-by-line error but the program takes more time to execute successfully. Interpreters were first used in 1952 to ease programming within the limitations of computers at the time. It translates source code into some efficient intermediate representation and immediately execute
this.

Source programs are compiled ahead of time and stored as machine independent code, which is then linked at run-time and executed by an interpreter. An Interpreter is generally used in micro computer. It helps the programmer to find out the errors and to correct them before control moves to the next statement. Interpreter system performs the actions described by the high level program. For interpreted programs, the source code is needed to run the program every time. Interpreted programs run slower than the compiled programs. **Self-Interpreter** is a programming language interpreter which is written in a language which can interpret itself. For Example- **BASIC** interpreter written in **BASIC**. They are related to self-hosting compilers. Some languages have an elegant and self-interpreter such as Lisp and Prolog. **Need of an Interpreter :** The first and vital need of an interpreter is to translate source code from high-level language to machine language. However, for this purpose Compiler is also there to satisfy this condition. The compiler is a very powerful tool for developing programs in high-level language. However, there are several demerits associated with the compiler. If the source code is huge in size, then it might take hours to compile the source code, which will significantly increase the compilation duration. Here, Interpreter plays its role. They can cut this huge compilation duration. They are designed to translate single instruction at a time and execute them immediately. **Advantage and Disadvantage of Interpreter :**

Advantage of interpreter is that it is executed line by line which helps users to find errors easily.

Disadvantage of interpreter is that it takes more time to execute successfully than compiler.

Applications of Interpreters :

Each operator executed in a command language is usually an invocation of a complex routine, such as an editor or compiler so they are frequently used to command languages and glue languages.

Virtualization is often used when the intended architecture is unavailable.

Sand-boxing

Self-modifying code can be easily implemented in an interpreted language.

Emulator for running Computer software written for obsolete and unavailable hardware on more modern equipment.

Some examples of programming languages that use interpreters are Python, Ruby, Perl, PHP and Matlab. **Top Interpreters according to the computer languages –**

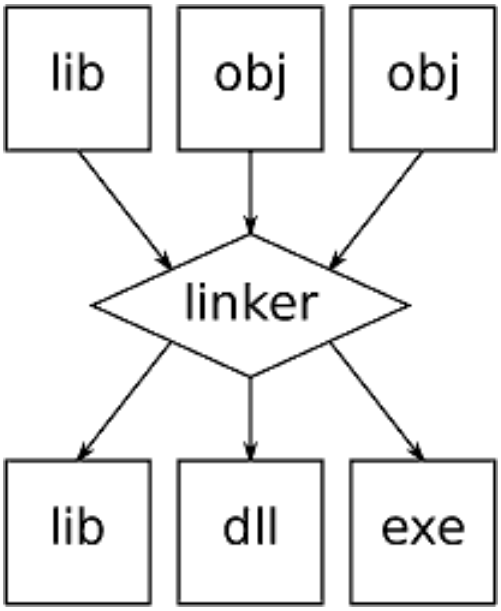
Python- CPython, PyPy, Stackless Python, IronPython
Ruby- YARV, Ruby MRI (CRuby)
JAVA- HotSpot, OpenJ9, JRockIt
Kotlin- JariKo

Introduction to Dubugger

Debugging is the process of detecting and removing of existing and potential errors (also called as 'bugs') in a software code that can cause it to behave unexpectedly or crash. To prevent incorrect operation of a software or system, debugging is used to find and resolve bugs or defects. When various subsystems or modules are tightly coupled, debugging becomes harder as any change in one module may cause more bugs to appear in another. Sometimes it takes more time to debug a program than to code it.

Introduction to Linker

A linker is a program in a system, also known as a link editor and binder, which combines object modules into a single object file. Generally, it is a program that performs the process of linking; it takes one or multiple object files, which are generated by compiler. And, then combines these files into an executable files. Modules are called for the different pieces of code, which are written in programming languages. Linking is a process that helps to gather and maintain a different piece of code into an executable file or single file. With the help of a linker, a specific module is also linked into the system library.



The primary function of the linker is to take objects from the assembler as input and create an executable file as output for the loader, as it helps to break down a large problem into a small module that simplifies the programming task. Usually, computer programs are made up of various modules in which all being a compiled computer programs and span separate object files. The whole program refers to these different compiled modules with the help of using symbols. These separate files are combined by linker into a single executable file. The source code is converted into machine code, and the linking is performed at the last step while compiling the program.

Source code -> compiler -> Assembler -> Object code -> Linker -> Executable file -> Loader

The objects can be collected by linker from a library or runtime library. Most of the linker only consists of files in the output that are referenced by other libraries or object files, and they do not include the whole library. The process of library linking requires additional modules to be linked with some referenced modules; thus, it may be an iterative process. Generally, one or more than one system libraries are linked by default, and libraries are available for different purposes.

In the program's address space, the arranging of the objects is also handled by the linker. The compiler often assumes a fixed base location (like zero), as it seldom knows about the object location where it will reside. The loads, store, and re-targeting of absolute jumps may be involved in the relocating machine code. When the executable output produced by the linker is finally loaded into memory, it may require other relocation pass. Usually, this pass is omitted on the hardware, hardware that offers virtual memory. There is no conflict even at the time all programs load at the same base address because each program is put into its own address space. If the executable file is a position-independent executable, this pass is also omitted on this file.

Introduction to Loader

A loader is a major component of an operating system that ensures all necessary programs and libraries are loaded, which is essential during the startup phase of running a program. It places the libraries and programs into the main memory in order to prepare them for execution. Loading involves reading the contents of the executable file that contains the instructions of the program and then doing other preparatory tasks that are required in order to prepare the executable for running, all of which takes anywhere from a few seconds to minutes depending on the size of the program that needs to run.

The following are the responsibilities of a loader:

- Validate the program for memory requirements, permissions, etc.
- Copy necessary files, such as the program image or required libraries, from the disk into the memory
- Copy required command-line arguments into the stack
- Link the starting point of the program and link any other required library
- Initialize the registers
- Jump to the program starting point in memory

Introduction to Assembler

The Assembler is a Software that converts an assembly language code to machine code. It takes basic Computer commands and converts them into Binary Code that Computer's Processor can use to perform its Basic Operations. These instructions are assembler language or assembly language.

We can also name an assembler as the compiler of assembly language. This is because a compiler converts the high-level language to machine language. On the other hand, an assembler is doing the same task but, for assembly language, the name compiler of assembly language.

What is an Assembly Language?

An assembly language is a low-level language. It gives instructions to the processors for different tasks. It is specific for any processor. The machine language only consists of 0s and 1s therefore, it is difficult to write a program in it. On the other hand, the assembly language is close to a machine language but has a simpler language and code.

We can create an assembly language code using a compiler or, a programmer can write it directly. Mostly, programmers use high-level languages but, when more specific code is

required, assembly language is used. It uses opcode for the instructions.
An **opcode** basically gives information about the particular instruction. The symbolic representation of the opcode (machine level instruction) is called **mnemonics**. Programmers use them to remember the operations in assembly language.

For example ADD A,B

Here, ADD is the mnemonic that tells the processor that it has to perform additional function. Moreover, A and B are the operands. Also, SUB, MUL, DIVC, etc. are other mnemonics.

Types of Assembler

Assemblers generate instruction. On the basis of a number of phases used to convert to machine code, assemblers have two types:

1. One-Pass Assembler

These assemblers perform the whole conversion of assembly code to machine code in one go.

2. Multi-Pass/Two-Pass Assembler

These assemblers first process the assembly code and store values in the **opcode table** and **symbol table**. And then in the second step, they generate the machine code using these tables.

a) Pass 1

- Symbol table and opcode tables are defined.
- keep the record of the **location counter**.
- Also, processes the pseudo instructions.

b) Pass 2

- Finally, converts the opcode into the corresponding numeric opcode.
- Generates machine code according to values of literals and symbols.

Some Important Terms

- **Opcode table:** They store the value of mnemonics and their corresponding numeric values.
- **Symbol table:** They store the value of programming language symbols used by the programmer, and their corresponding numeric values.

- **Location Counter:** It stores the address of the location where the current instruction will be stored.

Browse more Topics under System Software

- Operating system
- Compiler
- Interpreter

Differences Between Compiler and Assembler

Compiler	Assembler
Compiler converts the high-level language source code into machine level language code.	An assembler converts the assembly level language code into the machine level language code.
The input of a compiler is high-level language source code.	Where as, its input is low level assembly code.
Compiler converts the whole source code to machine code at once.	Assembler does not convert the code in one go.
It has the following phases: lexical analysis, syntax analysis, semantic analysis, intermediate code generation, code optimization, code generation and error handling.	An assembler completes the task in two passes.
It produces a machine code in form of mnemonics.	It produces binary code in form of 0s and 1s.
Examples are Java, C, C++ etc. compilers.	Examples of assemblers are GAS, GNU etc.

Problem Analysis: Problem solving techniques

1. Define the problem

Diagnose the situation so that your focus is on the problem, not just its symptoms. Helpful problem-solving techniques include using flowcharts to identify the expected steps of a process and cause and effect to define and analyze root causes.

The sections below help explain key problem-solving steps. These steps support the involvement of interested parties, the use of factual information, comparison of expectations to reality, and a focus on root causes of a problem. You should begin by:

- Reviewing and documenting how processes currently work (i.e., who does what, with what information, using what tools, communicating with what organizations and individuals, in what time frame, using what format).
- Evaluating the possible impact of new tools and revised policies in the development of your "what should be" model.

2. Generate alternative solutions

Postpone the selection of one solution until several problem-solving alternatives have been proposed. Considering multiple alternatives can significantly enhance the value of your ideal solution. Once you have decided on the "what should be" model, this target standard becomes the basis for developing a road map for investigating alternatives. Brainstorming and team problem-solving techniques are both useful tools in this stage of problem solving.

Many alternative solutions to the problem should be generated before final evaluation. A common mistake in problem solving is that alternatives are evaluated as they are proposed, so the first acceptable solution is chosen, even if it’s not the best fit. If we focus on trying to get the results we want, we miss the potential for learning something new that will allow for real improvement in the problem-solving process.

3. Evaluate and select an alternative

Skilled problem solvers use a series of considerations when selecting the best alternative. They consider the extent to which:

- A particular alternative will solve the problem without causing other unanticipated problems.
- All the individuals involved will accept the alternative.
- Implementation of the alternative is likely.
- The alternative fits within the organizational constraints.

4. Implement and follow up on the solution

Leaders may be called upon to direct others to implement the solution, "sell" the solution, or facilitate the implementation with the help of others. Involving others in the implementation is an effective way to gain buy-in and support and minimize resistance to subsequent changes.

Regardless of how the solution is rolled out, feedback channels should be built into the implementation. This allows for continuous monitoring and testing of actual events against expectations. Problem solving, and the techniques used to gain clarity, are most effective if the solution remains in place and is updated to respond to future changes.

Algorithm and Flowchart Representation

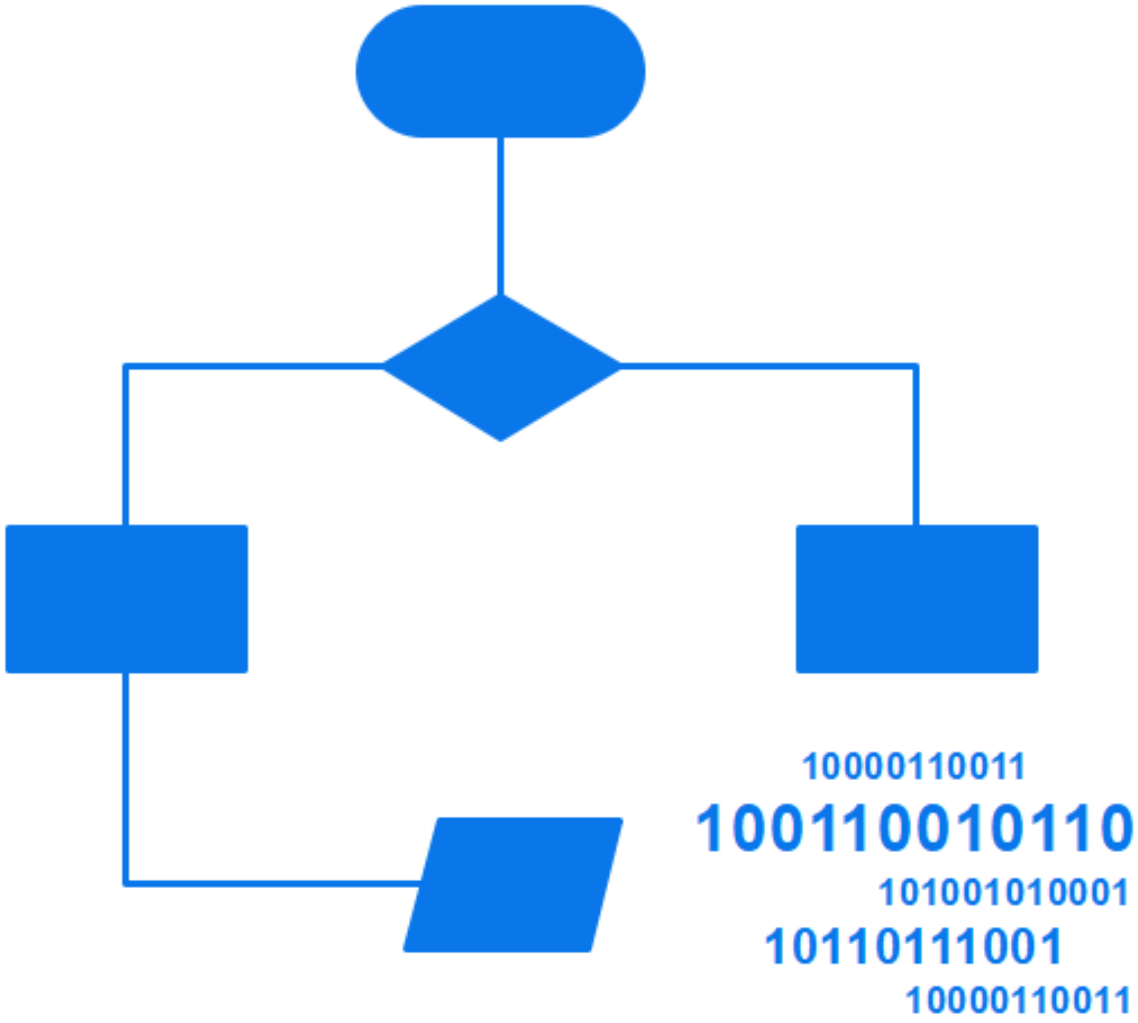
The algorithm and flowchart are two types of tools to explain the process of a program. In this page, we discuss the differences between an algorithm and a flowchart and how to create a flowchart to illustrate the algorithm visually.

Algorithms and flowcharts are two different tools that are helpful 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.

Part 1: Definition of Algorithm

Writing a logical step-by-step method to solve the problem is called the 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 in the process.

An algorithm includes calculations, reasoning, and data processing. Algorithms can be presented by natural languages, pseudocode, and flowcharts, etc.



Part 2: Definition of Flowchart

A flowchart is the graphical or pictorial representation of an algorithm with the help of different symbols, shapes, and arrows to demonstrate a process or a program. With algorithms, we can easily understand a program. The main purpose of using a flowchart is to analyze different methods. Several standard symbols are applied in a flowchart:

Terminal Box - Start / End	
Input / Output	

Process / Instruction	
Decision	
Connector / Arrow	

The symbols above represent different parts 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 certain elements and the relationships between each part.

Part 3: Difference between Algorithm and Flowchart

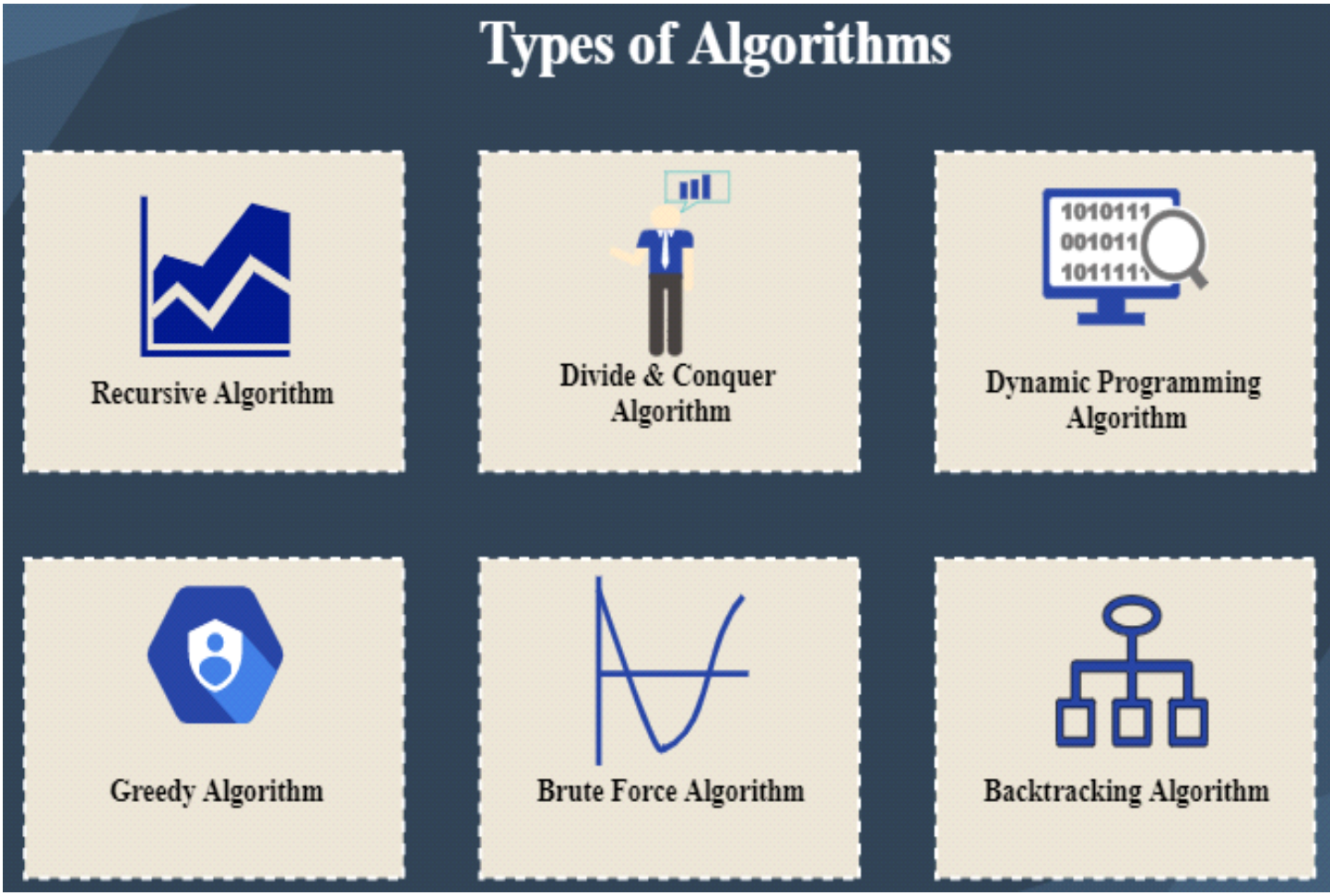
If you compare a flowchart to a movie, then an algorithm is the story of that movie. In other words, **an algorithm is the core of a flowchart.** Actually, in the field of computer programming, there are many differences between algorithm and flowchart regarding various aspects, such as the accuracy, the way they display, and the way people feel about them. Below is a table illustrating the differences between them in detail.

Algorithm	Flowchart
It is a procedure for solving problems.	It is a graphic representation of a process.
The process is shown in step-by-step instruction.	The process is shown in block-by-block information diagram.
It is complex and difficult to understand.	It is intuitive and easy to understand.
It is convenient to debug errors.	It is hard to debug errors.
The solution is showcased in natural language.	The solution is showcased in pictorial format.

It is somewhat easier to solve complex problem.	It is hard to solve complex problem.
It costs more time to create an algorithm.	It costs less time to create a flowchart.

Part 4: Types of Algorithm

It is not surprising that algorithms are widely used in computer programming. However, it can be applied to solving mathematical problems and even in everyday life. Here comes a question: how many types of algorithms? According to Dr. Christoph Koutschan, a computer scientist working at the Research Institute for Symbolic Computation (RISC) in Austria, he has surveyed voting for the important types of algorithms. As a result, he has listed 32 crucial algorithms in computer science. Despite the complexity of algorithms, we can generally divide algorithms into six fundamental types based on their function.



1. Recursive Algorithm

It refers to a way to solve problems by repeatedly breaking down the problem into sub-problems of the same kind. The classic example of using a recursive algorithm to solve problems is the Tower of Hanoi.

2. Divide and Conquer Algorithm

Traditionally, the divide and conquer algorithm consists of two parts: 1. breaking down a problem into some smaller independent sub-problems of the same type; 2. finding the final solution of the original issues after solving these more minor problems separately.

The key points of the divide and conquer algorithm are:

- If you can find the repeated sub-problems and the loop substructure of the original problem, you may quickly turn the original problem into a small, simple issue.
- Try to break down the whole solution into various steps (different steps need different solutions) to make the process easier.
- Are sub-problems easy to solve? If not, the original problem may cost lots of time.

3. Dynamic Programming Algorithm

Developed by Richard Bellman in the 1950s, the dynamic programming algorithm is generally used for optimization problems. In this type of algorithm, past results are collected for future use. Like the divide and conquer algorithm, a dynamic programming algorithm simplifies a complex problem by breaking it down into some simple sub-problems. However, the most significant difference between them is that the latter requires overlapping sub-problems, while the former doesn’t need to.

4. Greedy Algorithm

This is another way of solving optimization problems – greedy algorithm. It refers to always finding the best solution in every step instead of considering the overall optimality. That is to say, what he has done is just at a local optimum. Due to the limitations of the greedy algorithm, it has to be noted that the key to choosing a greedy algorithm is whether to consider any consequences in the future.

5. Brute Force Algorithm

The brute force algorithm is a simple and straightforward solution to the problem, generally based on the description of the problem and the definition of the concept involved. You can also use "just do it!" to describe the strategy of brute force. In short, a brute force algorithm is considered as one of the simplest algorithms, which iterates all possibilities and ends up with a satisfactory solution.

6. Backtracking Algorithm

Based on a depth-first recursive search, the backtracking algorithm focusing on finding the solution to the problem during the enumeration-like searching process. When it cannot satisfy the condition, it will return “backtracking” and tries another path. It is suitable for solving large and complicated problems, which gains the reputation of the

“general solution method.” One of the most famous backtracking algorithm example it the eight queens puzzle.

Part 5: Use Flowcharts to Represent Algorithms

Now that we have learned the definitions of algorithm and flowchart, how can we use a flowchart to represent an algorithm? To create an algorithm flowchart, we need to use a handy diagramming tool like EdrawMax to finish the work.

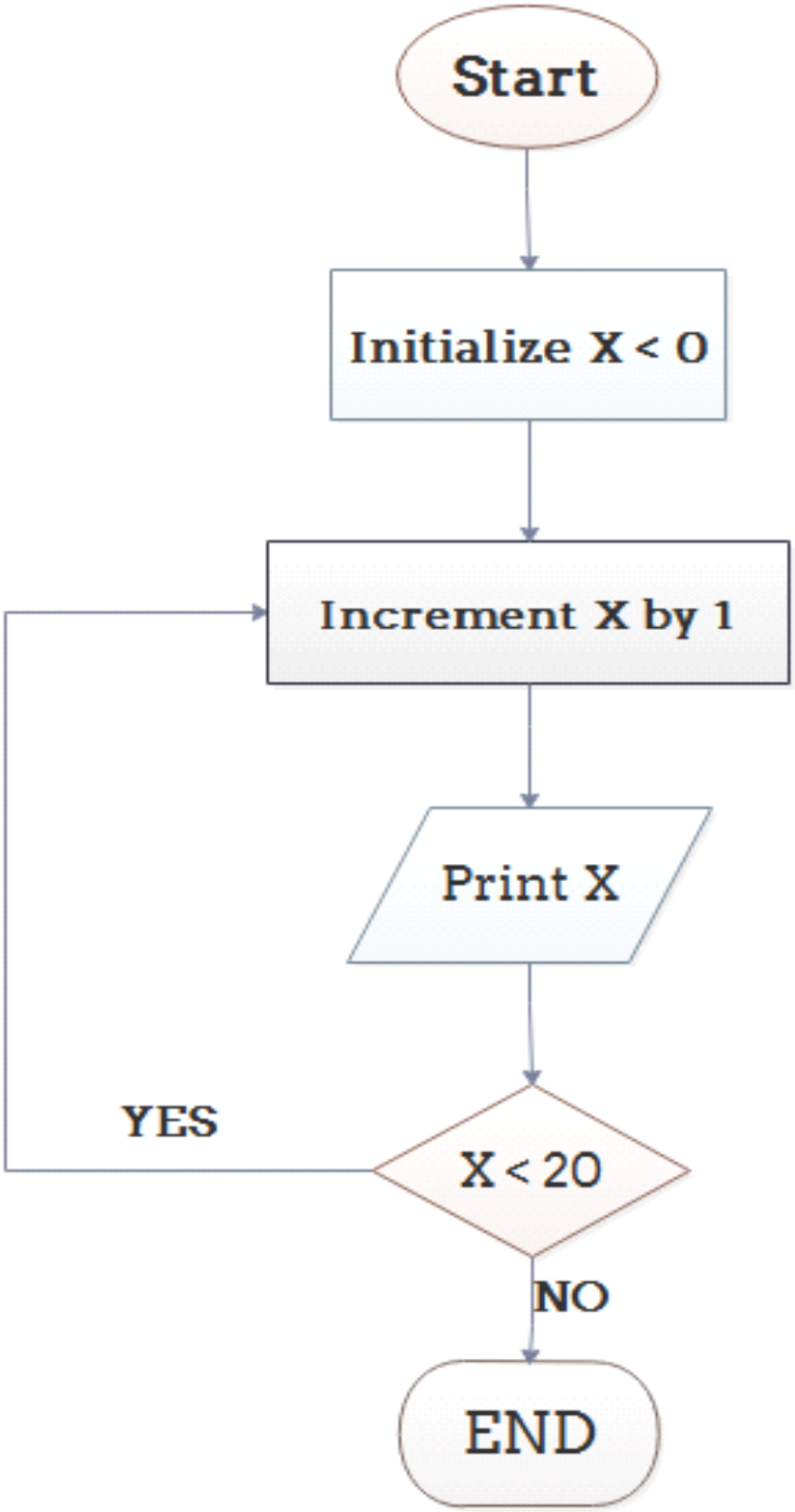
Algorithms are mainly used for mathematical and computer programs, whilst flowcharts can be used to describe all sorts of processes: business, educational, personal, and algorithms. So flowcharts are often used as a program planning tool to organize the program's step-by-step process visually. Here are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

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.

Flowchart:

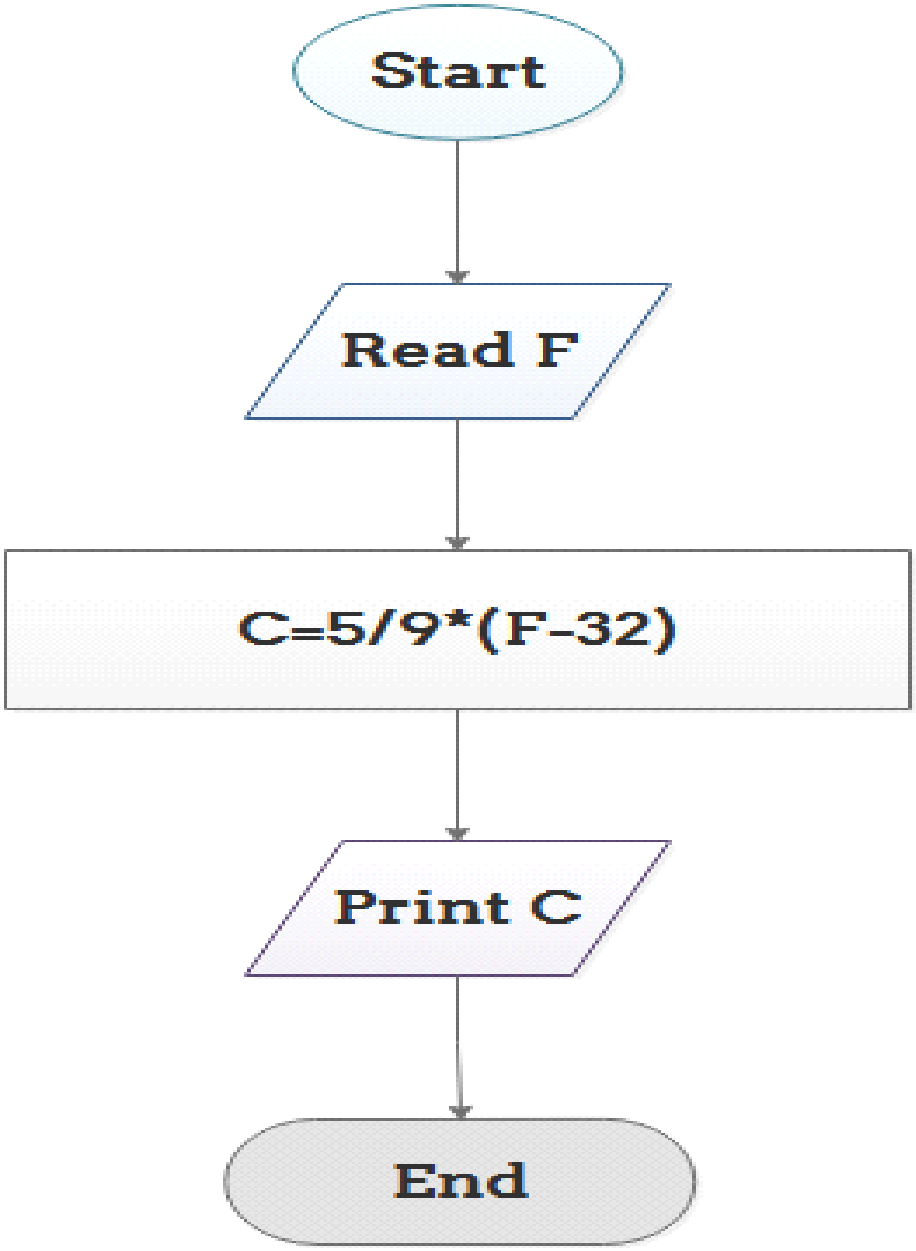

Start
Initialize X < 0
Increment X by 1
Print X
X < 20
YES
NO
END

Example 2: Convert Temperature from Fahrenheit (°F) to Celsius (°C)

Algorithm:

- Step 1: Read temperature in Fahrenheit,
- Step 2: Calculate temperature with formula C=5/9*(F-32),
- Step 3: Print C.

Flowchart:

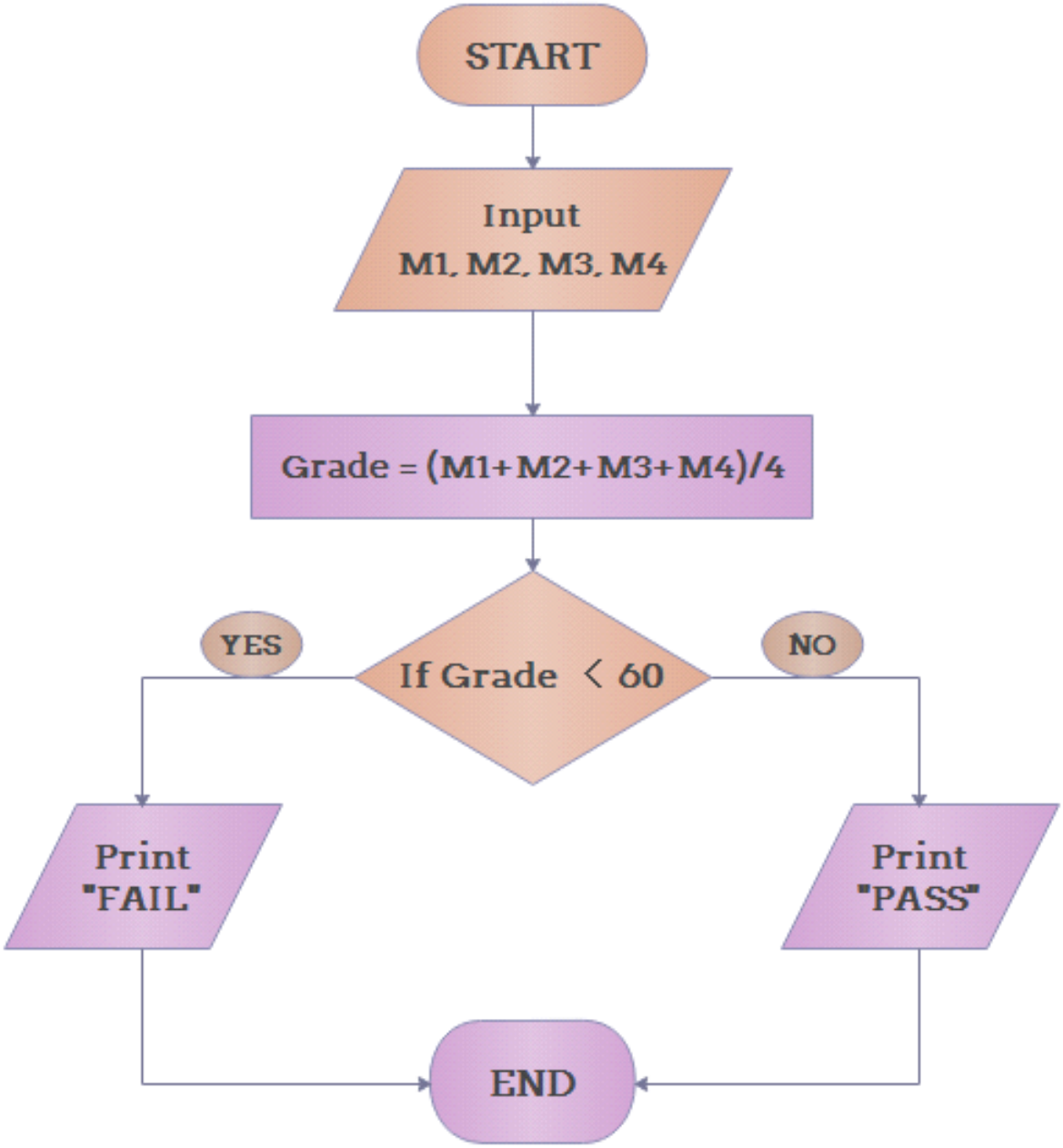


Example 3: Determine Whether A Student Passed the Exam or Not:

Algorithm:

- Step 1: Input grades of 4 courses M1, M2, M3 and M4,
- Step 2: Calculate the average grade with formula "Grade=(M1+M2+M3+M4)/4"
- Step 3: If the average grade is less than 60, print "FAIL", else print "PASS".

Flowchart:



Part 6: Conclusion

From the above, we can come to the conclusion that a flowchart is a pictorial representation of an algorithm, an algorithm can be expressed and analyzed through a flowchart. An algorithm shows you every step of reaching the final solution, while a flowchart shows you how to carry out the process by connecting each step. An algorithm uses mainly words to describe the steps while you can create a flowchart with flowchart symbols to make the process more logical.

Unit-2

Overview of C

Elements of c language

Before going in-depth, we need to understand the basic elements of the C language. It is very necessary to know these elements because it helps you to write C Program. In the below list, I have listed some important elements of the C language.

- Preprocessor directives.
- Functions.
- Variables.
- Statements.
- Comments.

Preprocessor directives:

A pre-processor directive begins with a special character # (pound). When you compile the code, then before compilation, directives tell the preprocessor to perform specific actions. These actions can be like replacing tokens in the text (using the macro), insert the contents of other files into the source file (Using file inclusion like #include <stdio.h>), or perform conditional Compilation (using #if, #else, #endif ..etc).

Function

A function is a collection of statements that perform a task, as the addition of two numbers. In other languages, a function is called procedure or subroutine. In C language there should be at least one function, this mandatory function is the main() function. The main() function is an entry point for c language that means it executes first.

Variables:

A variable defines a location name where we can put value and we can use these values whenever required in the program. In other words, we can say that variable is a name (or identifier) that indicates some physical address in the memory, where data will be stored. The value of a variable can be changed at different times of execution and it may be chosen by the programmer in a meaningful way.

Syntax of variable,

```
Data_Type  Variable_Name;
```

E.g.,
Data_Type V1, V2, V3; V1 V2, and V3 are three variables of the same data type.

In the above example, the Data type should be valid. It can be int, char, float, etc, or any user-defined data type like structure, union, or enum, etc.

Note: Variable names should be unique in the same scope otherwise you will get a compiler error.

Statements:

A statement is a command given to the computer that instructs the computer to take a specific action, like display a message on the console, performing the mathematical operation, and so on. In the C program is the collection of statements and each statement must be terminated with a semicolon(;).

There is a lot of type of statements available in C language these are,

- Labeled Statements (switch and case)
- Expression Statements (optional statement like printf (“welcome”);)
- Compound Statements (function)
- Selection Statements (if,else, ..etc)
- Iteration Statements (for,while, ..etc)

- Jump Statements (goto, ..etc) .

Comment:

Good comments increase the readability of the code. Every module should have good commenting, it helps the developer who comes on the project after you and it also helps to maintain the codebase.

One thing you should remember, if you are commenting on the code which has multiple lines then you should use the preprocessors conditional compilation feature (for example, #if 0 … #endif), it increases the code clarity.

See the below code,

```
//First way: Don't do this
/*
//bit reversal function
unsigned int ReverseTheBits(register unsigned int x)
{
 x = (((x & 0xaaaaaaaa) >> 1) | ((x & 0x55555555) << 1));
 x = (((x & 0xcccccccc) >> 2) | ((x & 0x33333333) << 2));
 x = (((x & 0xf0f0f0f0) >> 4) | ((x & 0x0f0f0f0f) << 4));
 x = (((x & 0xff00ff00) >> 8) | ((x & 0x00ff00ff) << 8));
return((x >> 16) | (x << 16));
}
*/
//Second way: Do this
#if 0
//bit reversal function
```

```
unsigned int ReverseTheBits(register unsigned int x)
{
    x = (((x & 0xaaaaaaaa) >> 1) | ((x & 0x55555555) << 1));
    x = (((x & 0xcccccccc) >> 2) | ((x & 0x33333333) << 2));
    x = (((x & 0xf0f0f0f0) >> 4) | ((x & 0x0f0f0f0f) << 4));
    x = (((x & 0xff00ff00) >> 8) | ((x & 0x00ff00ff) << 8));
    return((x >> 16) | (x << 16));
}
#endif
```

Data Types

Just like the name suggests, here, data types refer to the type of data that we are using in a C program. Whenever we utilise a data type in a C program, we define the variables or functions used in it. We do so because we must specify the type of data that is in use, so that the compiler knows exactly what type of data it must expect from the given program.

Purpose of Data Types in C

Data types used in C language refer to an extensive system that we use to declare various types of functions or variables in a program. Here, on the basis of the type of variable present in a program, we determine the space that it occupies in storage, along with the way in which the stored bit pattern will be interpreted.

A data type specifies the type of data that a variable can store such as integer, floating, character, etc.

Example of Data Types in C

Let us take a look at an example to understand data types better.

For instance, we may want to utilise some numbers such as 5, 8, 600, or maybe a decimal point number such as 43.59, 127.368, 271.49, or maybe a text such as “cappuccino”. Then, the compiler used in C language would handle all of these very differently. Thus, here, we use different data types for defining what data types we want in the program.

Types of Data Types in C

Here are the five major categories into which data types are divided in C language:

Data Type	Example of Data Type
Basic Data Type	Floating-point, integer, double, character.
Derived Data Type	Union, structure, array, etc.
Enumeration	Enums
Void Data Type	Empty Value
Bool Type	True or False

The basic data types are also known as the primary data types in C programming.

Primary Data Types in C

Here are the five primitive or primary data types that one can find in C programming language:

1. Integer – We use these for storing various whole numbers, such as 5, 8, 67, 2390, etc.

2. Character – It refers to all ASCII character sets as well as the single alphabets, such as ‘x’, ‘Y’, etc.

3. Double – These include all large types of numeric values that do not come under either floating-point data type or integer data type.

4. Floating-point – These refer to all the real number values or decimal points, such as 40.1, 820.673, 5.9, etc.

5. Void – This term refers to no values at all. We mostly use this data type when defining the functions in a program.

Various keywords are used in a program for specifying the data types mentioned above. Here are the keywords that we use:

Keyword Used	Data Type
int	Integer
float	Floating-point
void	Void
char	Character

double	Double

Storage Classes in C

Storage classes in C are used to determine the lifetime, visibility, memory location, and initial value of a variable. There are four types of storage classes in C

- Automatic
- External
- Static
- Register

Storage Classes	Storage Place	Default Value	Scope	Lifetime
auto	RAM	Garbage Value	Local	Within function
extern	RAM	Zero	Global	Till the end of the main program Maybe declared anywhere in the program
static	RAM	Zero	Local	Till the end of the main program, Retains value between multiple functions call
register	Register	Garbage Value	Local	Within the function

Automatic

- Automatic variables are allocated memory automatically at runtime.

- The visibility of the automatic variables is limited to the block in which they are defined.

  The scope of the automatic variables is limited to the block in which they are defined.
- The automatic variables are initialized to garbage by default.
- The memory assigned to automatic variables gets freed upon exiting from the block.
- The keyword used for defining automatic variables is auto.
- Every local variable is automatic in C by default.

Example 1

```
#include <stdio.h>
int main()
{
int a; //auto
char b;
float c;
printf("%d %c %f",a,b,c); // printing initial default value of automatic variables a, b , and c.
return 0;
}
```

Output:

```
garbage garbage garbage
```

Example 2

```
#include <stdio.h>
int main()
{
int a = 10,i;
printf("%d ",++a);
{
int a = 20;
```

```
    for (i=0;i<3;i++)
    {
    printf("%d ",a); // 20 will be printed 3 times since it is the local value of a
    }
    }
    printf("%d ",a); // 11 will be printed since the scope of a = 20 is ended.
    }
```

Output:

```
  11 20 20 20 11
```

Static

- The variables defined as static specifier can hold their value between the multiple function calls.
- Static local variables are visible only to the function or the block in which they are defined.
- A same static variable can be declared many times but can be assigned at only one time.
- Default initial value of the static integral variable is 0 otherwise null.
- The visibility of the static global variable is limited to the file in which it has declared.
- The keyword used to define static variable is static.

Example 1

```
#include<stdio.h>
static char c;
static int i;
static float f;
static char s[100];
void main ()
{
printf("%d %d %f %s",c,i,f); // the initial default value of c, i, and f will be printed.
```

```
}
```

Output;

```
0 0 0.000000 (null)
```

Example 2

```
#include<stdio.h>
void sum()
{
static int a = 10;
static int b = 24;
printf("%d %d \n",a,b);
a++;
b++;
}
void main()
{
int i;
for(i = 0; i< 3; i++)
{
sum(); // The static variables holds their value between multiple function calls.
}
}
```

Output:

```
10 24
11 25
12 26
```

Register

- The variables defined as the register is allocated the memory into the CPU registers depending upon the size of the memory remaining in the CPU.

- We can not dereference the register variables, i.e., we can not use &operator for the register variable.
- The access time of the register variables is faster than the automatic variables.
- The initial default value of the register local variables is 0.
- The register keyword is used for the variable which should be stored in the CPU register. However, it is compiler?s choice whether or not; the variables can be stored in the register.
- We can store pointers into the register, i.e., a register can store the address of a variable.
- Static variables can not be stored into the register since we can not use more than one storage specifier for the same variable.

Example 1

```
#include <stdio.h>
int main()
{
register int a; // variable a is allocated memory in the CPU register. The initial default val
ue of a is 0.
printf("%d",a);
}
```

Output:

```
0
```

Example 2

```
#include <stdio.h>
int main()
{
register int a = 0;
```

- printf("%u",&a); // This will give a compile time error since we can not access the address of a register variable.
- }

Output:

```
main.c:5:5: error: address of register variable ?a? requested
printf("%u",&a);
^~~~~~
```

External

- The external storage class is used to tell the compiler that the variable defined as extern is declared with an external linkage elsewhere in the program.
- The variables declared as extern are not allocated any memory. It is only declaration and intended to specify that the variable is declared elsewhere in the program.
- The default initial value of external integral type is 0 otherwise null.
- We can only initialize the extern variable globally, i.e., we can not initialize the external variable within any block or method.
- An external variable can be declared many times but can be initialized at only once.
- If a variable is declared as external then the compiler searches for that variable to be initialized somewhere in the program which may be extern or static. If it is not, then the compiler will show an error.

Example 1

- #include <stdio.h>
- **int** main()
- {
- **extern int** a;
- printf("%d",a);
- }

Output

```
main.c:(.text+0x6): undefined reference to `a'
```

```
collect2: error: ld returned 1 exit status
```

Example 2

```
#include <stdio.h>
int a;
int main()
{
extern int a; // variable a is defined globally, the memory will not be allocated to
a
printf("%d",a);
}
```

Output

```
0
```

Example 3

```
#include <stdio.h>
int a;
int main()
{
extern int a = 0; // this will show a compiler error since we can not use extern an
d initializer at same time
printf("%d",a);
}
```

Output

```
compile time error
main.c: In function ?main?:
main.c:5:16: error: ?a? has both ?extern? and initializer
extern int a = 0;
```

Example 4

```
#include <stdio.h>
int main()
{
```

- **extern int** a; // Compiler will search here for a variable a defined and initialized somewh ere in the pogram or not.
- printf("%d",a);
- }
- **int** a = 20;

Output

```
20
```

Example 5

- **extern int** a;
- **int** a = 10;
- #include <stdio.h>
- **int** main()
- {
- printf("%d",a);
- }
- **int** a = 20; // compiler will show an error at this line

Output

```
compile time error
```

C Operators

An operator is simply a symbol that is used to perform operations. There can be many types of operations like arithmetic, logical, bitwise, etc.

There are following types of operators to perform different types of operations in C language.

- Arithmetic Operators
- Relational Operators
- Shift Operators

- Logical Operators
- Bitwise Operators
- Ternary or Conditional Operators
- Assignment Operator
- Misc Operator

Precedence of Operators in C

The precedence of operator species that which operator will be evaluated first and next. The associativity specifies the operator direction to be evaluated; it may be left to right or right to left.

Let's understand the precedence by the example given below:

- `int value=10+20*10;`

The value variable will contain **210** because * (multiplicative operator) is evaluated before + (additive operator).

The precedence and associativity of C operators is given below:

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right

Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Precedence and Associativity of Operators in C

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right

Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Example Code

```
#include <stdio.h>
main() {
   int a = 20;
   int b = 10;
   int c = 15;
   int d = 5;
   int e;
   e = (a + b) * c / d; // ( 30 * 15 ) / 5
   printf("Value of (a + b) * c / d is : %d\n", e );
   e = ((a + b) * c) / d; // (30 * 15 ) / 5
   printf("Value of ((a + b) * c) / d is : %d\n" , e );
   e = (a + b) * (c / d); // (30) * (15/5)
   printf("Value of (a + b) * (c / d) is : %d\n", e );
   e = a + (b * c) / d; // 20 + (150/5)
   printf("Value of a + (b * c) / d is : %d\n" , e );
   return 0;
}
```

Output

```
Value of (a + b) * c / d is : 90
Value of ((a + b) * c) / d is : 90
```

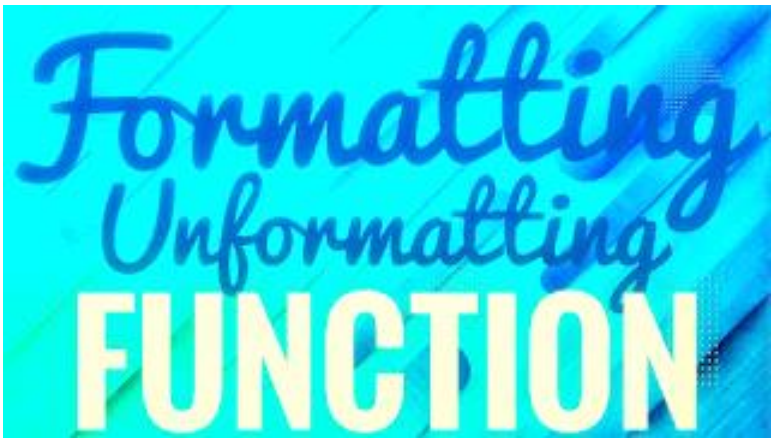
```
Value of (a + b) * (c / d) is : 90
```

Value of a + (b * c) / d is : 50

Formatted and Unformatted Input/Output Function in C

Nikhilesh GuptaOctober 14, 2019

There are two types of Input/Output function in C, which have some advantages as well as some disadvantages over each other. Each of them is used for a specific work to do.



Formatted Function

One type of Input/Output function is **Formatted function** in which input/output is formatted as per our requirements.

There are two types of **Formatted Functions:**

- **printf( )**

  In a **C** program we use "**printf( )**" to print **"character, string, float, integer, octal and hexadecimal values"** in the output screen. In printf( ) the **format specifier** will help the compiler to know whether the output value is **int, float** or of **some other datatype.** Also we can print anything we desire to print by putting that value under double coats (" ").

  Syntax:

  - **printf("Format Specifier", arg 1, arg 2, arg 3, ...arg n);**

  The above syntax is used when you want to print some value.

```c
#include<stdio.h>
main()
{
    int a;
    a=13;
    printf("%d",a);
}
```

Example 1

OR

- **printf("Enter the text you want to print");**

The above syntax is used when you want to print a text directly.

```c
#include<stdio.h>
main()
{
    int a;
    a=13;
    printf("Enter the Text");
}
```

Example 2

-
- # scanf( )

In a **C** program we use "**scanf( )**" to get the values as input to the program of different datatypes like **int, float, double** and **many more.** We use **format specifier** to differentiate the datatypes as Inputs.

Syntax:

- **scanf("Format Specifier", arg 1, arg 2, arg 3, ...arg n);**

In **scanf( ) you have to determine the type data you want to enter.**

```
1 #include<stdio.h>
2 main()
3 {
4     int a;
5     scanf("%d",&a);
6 }
```

Example 3

*In **scanf( )** we have to use **'&'** otherwise it will show you error.*

Unformatted Function

Other type of Input/Output function is **Unformatted function.** This type of Input/Output function doesn't require any **format specifier.**

There are three types of **Unformatted I/O functions**:

▲Character I/O

In this I/O function it doesn't require any datatype because it will only work with **CHAR DATATYPE.**

- ### getchar( ):

  This function can read one character at one time until and unless the user presses the "**Enter Key**". It can only store or take **CHAR** type input from the user.

  Syntax: Variable_Name = getchar( );

```c
#include<stdio.h>
main()
{
    char a;
    a=getchar();
}
```

Example 4

- ## putchar( ):

This the function use to print **one character** in the screen at a time.

Syntax: **putchar(variable name);**

```c
#include<stdio.h>
main()
{
    char a;
    a='A';
    putchar('a');
}
```

Example 5

- ## getch( ) & getche( ):

These function read any alphanumeric character from the standard input device. The character entered is not **displayed** by the **getch( )** function until the enter is pressed.

Syntax: **getch( );** **getche( );**

```c
#include<stdio.h>
main()
{
printf("Enter two value");
getch();
getche();
}
```

Example 6

- ## putch( ):

This function can print any **alphanumeric character** given by the user.

Syntax: **putch(variable name);**

```c
#include<stdio.h>
main()
{
  char a;
  a=getch();
  putch(a);
}
```

Example 7

▲String I/O

In this I/O function we take "**String**" as the input and also print "**String**" as output.

- ## gets( ):

This function is used to take "**String**" as input until and unless the enter key is pressed.

Syntax: **char str[length of the string in number];**
gets(str);

```c
#include<stdio.h>
main()
{
 char a[10];
 gets(a);
 printf("%s",a);
}
```

Example 8

- **puts( ):**
  This function is used to print the **string** which has been stored in the program.

  Syntax: char str[length of str in number]; gets(str); puts(str);

```c
#include<stdio.h>
main()
{
 char a[10];
 gets(a);
 puts(a);
 }
```

Example 9

▲File I/O

A file represents the sequence of bytes on the disk where a group of related data is stored. File is created for permanent storage of **DATA.** It is ready made structure.

Function	description
fopen()	create a new file or open a existing file
fclose()	closes a file
getc()	reads a character from a file
putc()	writes a character to a file
fscanf()	reads a set of data from a file
fprintf()	writes a set of data to a file
getw()	reads a integer from a file
putw()	writes a integer to a file
fseek()	set the position to desire point
ftell()	gives current position in the file
rewind()	set the position to the begining point

Some I/O Function

Control Statements in C:

Control Statements in C are used to execute/transfer the control from one part of the program to another depending on a condition. These statements are also called as conditional statements.

Types of Control Statements in C:

In C Programming, There are mainly two types of Control Statements:

1. **if-else** statement
2. **switch** statement

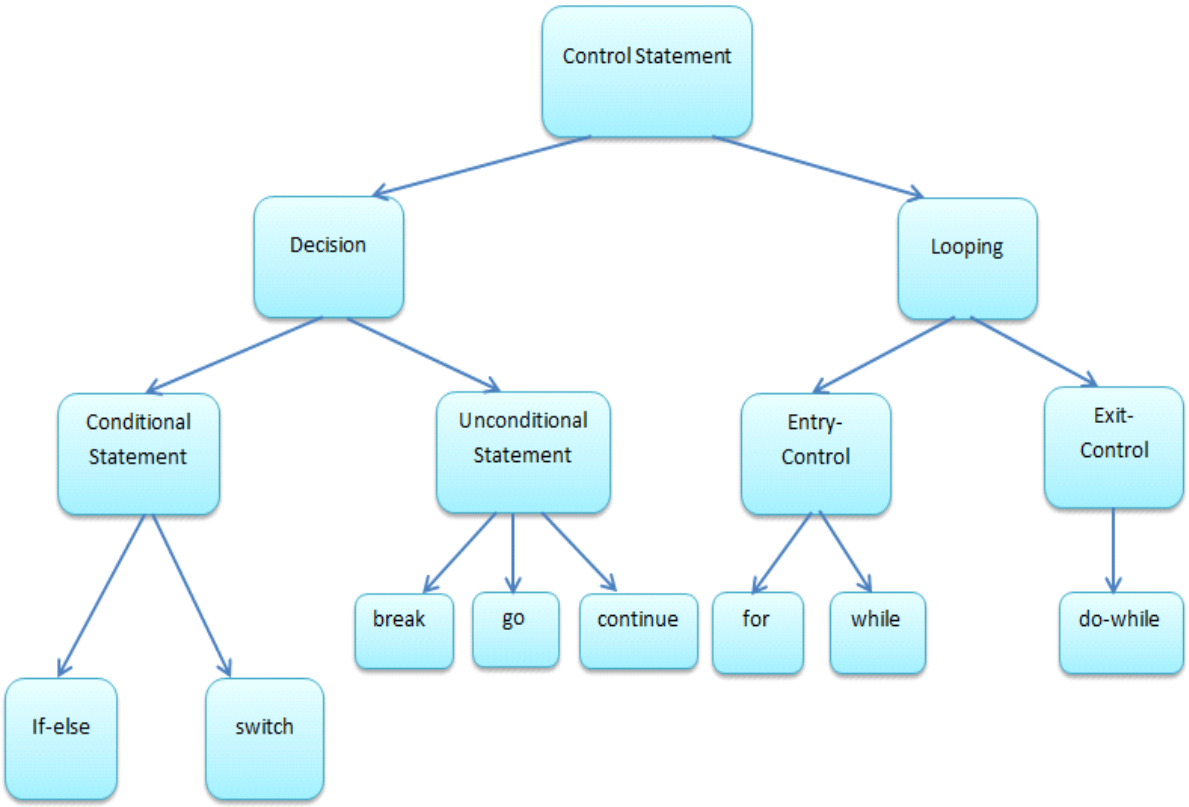


Fig: Control Statements in C

if-else statement:

if-else statement is used to execute a statement block or a single statement depending on the value of a condition. It has the following syntax :

- **if**(condition)
- {
- - - - - - - - - -
- <**true** block>
- - - - - - - - - -
- }
- **else**
- {
- - - - - - - - - -
- <**false** block>
- - - - - - - - - -
- } // where <condition> is a logical expression which will have the value true or false .

Nested if-else Statement:

An **if** statement may have another **if** statement in the < **true block** > and < **false block** >. It is the nesting of if statement within another if statement and nesting of if statement with an else statement. This compound statement is called Nested if-else statement.

- **if**(condition 1)
- **if**(condition 2)
- {
- - - - - - - - - -
- <**true** block 1>
- - - - - - - - - -
- }
- **else**
- {
- - - - - - - - - -
- <**false** block 1>
- - - - - - - - - -
- }
- **else**
- **if**(condition 3)
- {
- - - - - - - - - -
- <**true** block 2>
- - - - - - - - - -
- }
- **else**
- {
- - - - - - - - - -
- <**false** block 2>
- - - - - - - - - -
- }

Switch statement:

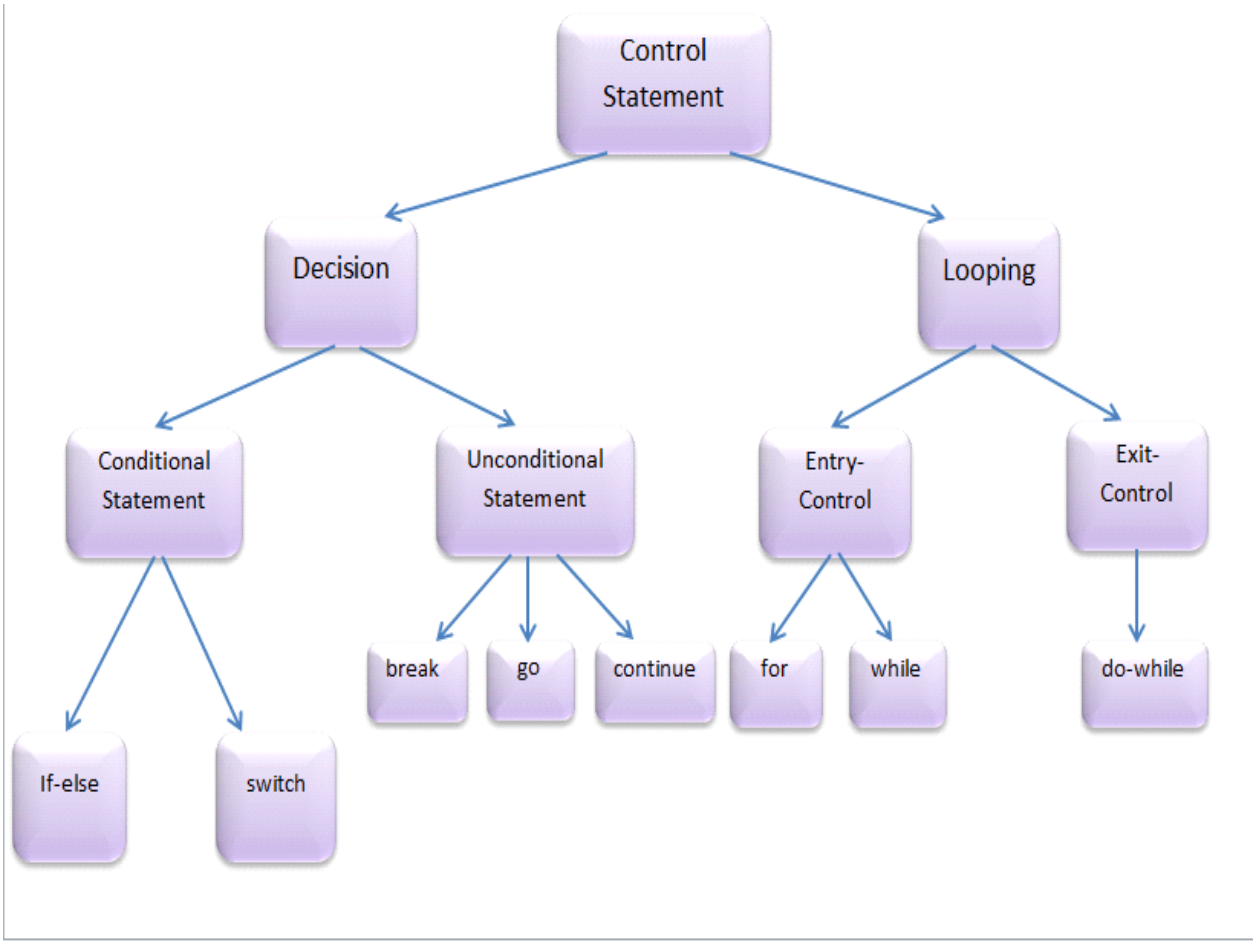
switch statement is used to execute a block of statements depending on the value of a variable or an expression. It has the following syntax :

- **switch** (<expression>)
- {
- **case** <label 1>: {
- - - - - - - - - - -
- <statement 1>

- ----------
- **break**;
- }
- **case** <label 2>: {
- ----------
- <statement 2>
- ----------
- **break**;
- }
- **case** <label 3>: {
- ----------
- <statement 3>
- ----------
- **break**;
- }
- **case** <label n>: {
- ----------
- <statement n>
- ----------
- **break**;
- }
- **default**: {
- ----------
- <**default** statement>
- ----------
- **break**;
- }
- } // where <expression> refers any int or char expression or variable. <label 1>, <label 2>, <label 3>, <label n> are the values which will match with the value of the expression.
- the **break** is a statement that will transfer the control to the end of a **switch** statement

Loop Statements in C are used to execute and repeat a block of statements depending on the value of a condition. There are three types of Loop Control Statements in C language:

1. **for** loop
2. **while** loop
3. **do-while** loop



for loop:

The **for** loop is entry-controlled loop that provides a more concise loop control structure. A **for** loop is used to execute and repeat a block of statements depending on a condition. It has the following syntax:

- **for**(<initial value>;<condition>;<increment>)
- {
- - - - - - - - - - -
- <statement block>
- - - - - - - - - - -
- }

Nested for loop:

This statement block of a for loop lies completely inside the block of another for loop that is called **Nested for** loop or **Nested for** Statement.

- **for**(i=1;i<=3;i++) // outer loop
- {
- - - - - - - - -

- for(j=1;j<=5;j++) // inner loop
- {
- - - - - - - - -
- <statement block>
- - - - - - - - -
- }
- }

while loop:

It is also an entry-controlled loop statement. A **while** loop is used to execute and repeat a block of statements depending on a condition. It has the following syntax:

- while(<condition>)
- {
- - - - - - - - -
- <statement block>
- - - - - - - - -
- } // where <condition> is a relational or logical expression which will have the value true or false.

do-while:

A **do-while** loop is an exit-controlled loop statement. It is used to execute and repeat a block of statements depending on a condition. It has the following syntax :

- do
- {
- - - - - - - - -
- <statement block>
- - - - - - - - -
- }
- while(<condition>) // where <condition> is a relational or logical expression which will have the value true or false.

We can Comparison Between for loop while loop and do-while loop is given below:

for loop vs while loop vs do-while loop:

for loop	while loop	do-while loop

1. A for loop is used to execute and repeat a statement block depending on a condition which is evaluated at the beginning of the loop.	1. A while loop is used to execute and repeat a statement block depending on a condition which is evaluated at the beginning of the loop.	1. A do-while loop is used to execute and repeat a statement block depending on a condition which is evaluated at the end of the loop.
2. A variable value is initialized at the beginning of the loop.	2. A variable value is initialized at the beginning or before of the loop.	2. A variable value is initialized before the loop or assigned inside the loop.
3. The statement block will not be executed when the value of the condition is false.	3. The statement block will not be executed when the value of the condition is false.	3. The statement block will not be executed when the value of the condition is false, but the block is executed at least once irrespective of the value of the condition.
4. A statement to change the value of the condition or to increment the value of the variable is given at the beginning of the loop.	4. A statement to change the value of the condition or to increment the value of the variable is given inside of the loop.	4. A statement to change the value of the condition or to increment the value of the variable is given inside of the loop.
5. A for loop is commonly used by many programmers.	5. A while loop is widely used by many programmers.	5. A do-while loop is used in some cases where the condition need to be checked at the end of the loop.

Unconditional Control Transfer Statements is used to transfer the control to some other place in the program. In C Programming Language, There are four types of unconditional control transfer statements.

1. goto
2. break
3. continue
4. return

goto statement:

The **goto** statement is used to transfer the control from one part of the program to another. It has the following syntax :

```
- - - - - - - - - - -
goto display;
- - - - - - - - - - -
- - - - - - - - - - -
display;
- - - - - - - - - - -
```

when this statement is executed, the control is transferred to the statement lebel **display** which is followed by a colon.

break statement:

The **break** statement is used to transfer the control to the end of a statement block in a loop. It is an unavoidable statement to transfer the control to the end of a switch statement after executing anyone statement block. It has the following syntax :

```
break;
```

continue statement:

The **continue** statement is used to transfer the control to the beginning of a statement block in a loop. It has the following syntax:

```
continue;
```

return statement:

The **return** statement terminates the execution of a function, it returns control to the calling function. It resumes the execution in the calling function immediately. It has the following syntax:

```
jump instruction or statement:
return expression;
```

Unit-3

Arrays

An array is defined as the collection of similar type of data items stored at contiguous memory locations. Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc. It also has the capability to store the collection of derived data types, such as pointers, structure, etc. The array is the simplest data structure where each data element can be randomly accessed by using its index number.

C array is beneficial if you have to store similar elements. For example, if we want to store the marks of a student in 6 subjects, then we don't need to define different variables for the marks in the different subject. Instead of that, we can define an array which can store the marks in each subject at the contiguous memory locations.

By using the array, we can access the elements easily. Only a few lines of code are required to access the elements of the array.

Properties of Array

The array contains the following properties.

- Each element of an array is of same data type and carries the same size, i.e., int = 4 bytes.
- Elements of the array are stored at contiguous memory locations where the first element is stored at the smallest memory location.
- Elements of the array can be randomly accessed since we can calculate the address of each element of the array with the given base address and the size of the data element.

Advantage of C Array

1) Code Optimization: Less code to the access the data.

2) Ease of traversing: By using the for loop, we can retrieve the elements of an array easily.

3) Ease of sorting: To sort the elements of the array, we need a few lines of code only.

4) Random Access: We can access any element randomly using the array.

Disadvantage of C Array

1) Fixed Size: Whatever size, we define at the time of declaration of the array, we can't exceed the limit. So, it doesn't grow the size dynamically like LinkedList which we will learn later.

Declaration of C Array

We can declare an array in the c language in the following way.

- data_type array_name[array_size];

Now, let us see the example to declare the array.

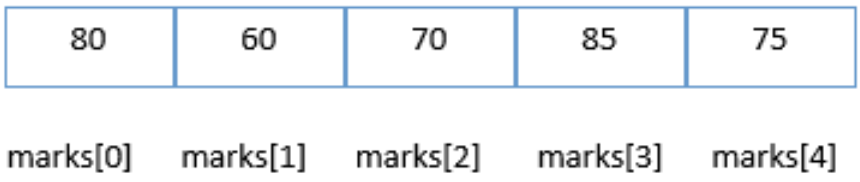
- **int** marks[5];

Here, int is the *data_type*, marks are the *array_name*, and 5 is the *array_size*.

Initialization of C Array

The simplest way to initialize an array is by using the index of each element. We can initialize each element of the array by using the index. Consider the following example.

- marks[0]=80;//initialization of array
- marks[1]=60;
- marks[2]=70;
- marks[3]=85;
- marks[4]=75;



Initialization of Array

C array example

- #include<stdio.h>
- **int** main(){

```
int i=0;
int marks[5];//declaration of array
marks[0]=80;//initialization of array
marks[1]=60;
marks[2]=70;
marks[3]=85;
marks[4]=75;
//traversal of array
for(i=0;i<5;i++){
printf("%d \n",marks[i]);
}//end of for loop
return 0;
}
```

Output

```
80
60
70
85
75
```

C Array: Declaration with Initialization

We can initialize the c array at the time of declaration. Let's see the code.

```
int marks[5]={20,30,40,50,60};
```

In such case, there is **no requirement to define the size**. So it may also be written as the following code.

```
int marks[]={20,30,40,50,60};
```

Let's see the C program to declare and initialize the array in C.

```
#include<stdio.h>
int main(){
int i=0;
```

```
int marks[5]={20,30,40,50,60};//declaration and initialization of array
 //traversal of array
for(i=0;i<5;i++){
printf("%d \n",marks[i]);
}
return 0;
}
```

Output

```
20
30
40
50
60
```

C Array Example: Sorting an array

In the following program, we are using bubble sort method to sort the array in ascending order.

```
#include<stdio.h>
void main ()
{
    int i, j,temp;
    int a[10] = { 10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
    for(i = 0; i<10; i++)
    {
        for(j = i+1; j<10; j++)
        {
            if(a[j] > a[i])
            {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
```

```
            }
        }
    }
    printf("Printing Sorted Element List ...\n");
    for(i = 0; i<10; i++)
    {
        printf("%d\n",a[i]);
    }
}
```

Program to print the largest and second largest element of the array.

```
#include<stdio.h>
void main ()
{
    int arr[100],i,n,largest,sec_largest;
    printf("Enter the size of the array?");
    scanf("%d",&n);
    printf("Enter the elements of the array?");
    for(i = 0; i<n; i++)
    {
        scanf("%d",&arr[i]);
    }
    largest = arr[0];
    sec_largest = arr[1];
    for(i=0;i<n;i++)
    {
        if(arr[i]>largest)
```

```
        {
            sec_largest = largest;
            largest = arr[i];
        }
        else if (arr[i]>sec_largest && arr[i]!=largest)
        {
            sec_largest=arr[i];
        }
    }
    printf("largest = %d, second largest = %d",largest,sec_largest);

}
```

Two Dimensional Array in C

The two-dimensional array can be defined as an array of arrays. The 2D array is organized as matrices which can be represented as the collection of rows and columns. However, 2D arrays are created to implement a relational database lookalike data structure. It provides ease of holding the bulk of data at once which can be passed to any number of functions wherever required.

Declaration of two dimensional Array in C

The syntax to declare the 2D array is given below.

- data_type array_name[rows][columns];

Consider the following example.

- `int twodimen[4][3];`

Here, 4 is the number of rows, and 3 is the number of columns.

Initialization of 2D Array in C

In the 1D array, we don't need to specify the size of the array if the declaration and initialization are being done simultaneously. However, this will not work with 2D arrays. We will have to define at least the second dimension of the array. The two-dimensional array can be declared and defined in the following way.

- `int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};`

Two-dimensional array example in C

```
#include<stdio.h>
int main(){
int i=0,j=0;
int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
//traversing 2D array
for(i=0;i<4;i++){
 for(j=0;j<3;j++){
   printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);
 }//end of j
}//end of i
return 0;
}
```

Output

```
arr[0][0] = 1
arr[0][1] = 2
arr[0][2] = 3
arr[1][0] = 2
arr[1][1] = 3
arr[1][2] = 4
arr[2][0] = 3
arr[2][1] = 4
arr[2][2] = 5
arr[3][0] = 4
arr[3][1] = 5
arr[3][2] = 6
```

C 2D array example: Storing elements in a matrix and printing it.

```
#include <stdio.h>
void main ()
{
   int arr[3][3],i,j;
   for (i=0;i<3;i++)
   {
      for (j=0;j<3;j++)
      {
         printf("Enter a[%d][%d]: ",i,j);
         scanf("%d",&arr[i][j]);
      }
   }
   printf("\n printing the elements ....\n");
   for(i=0;i<3;i++)
   {
      printf("\n");
      for (j=0;j<3;j++)
      {
         printf("%d\t",arr[i][j]);
      }
   }
}
```

Output

```
Enter a[0][0]: 56
Enter a[0][1]: 10
Enter a[0][2]: 30
Enter a[1][0]: 34
```

```
Enter a[1][1]: 21
Enter a[1][2]: 34

Enter a[2][0]: 45
Enter a[2][1]: 56
Enter a[2][2]: 78

 printing the elements ....

56      10      30
34      21      34
45      56      78
```

String Holder

A string is a 1-D array of characters, so an array of strings is a 2-D array of characters. Just like we can create a 2-D array of `int`, `float` etc; we can also create a 2-D array of character or array of strings. Here is how we can declare a 2-D array of characters.

```
1
2
3
4
5
```

```
char ch_arr[3][10] = {
                        {'s', 'p', 'i', 'k', 'e',
'\0'},
                        {'t', 'o', 'm','\0'},
                        {'j', 'e', 'r', 'r',
'y','\0'}
                      };
```

It is important to end each 1-D array by the null character, otherwise, it will be just an array of characters. We can't use them as strings.

Declaring an array of strings this way is rather tedious, that's why C provides an alternative syntax to achieve the same thing. This above initialization is equivalent to:

```
1
2
3
4
5
```

```
char ch_arr[3][10] = {
                        "spike",
                        "tom",
                        "jerry"
                      };
```

The first subscript of the array i.e `3` denotes the number of strings in the array and the second subscript denotes the maximum length of the string. Recall the that in C, each character occupies `1` byte of data, so when the compiler sees the above statement it allocates `30` bytes (`3*10`) of memory.

We already know that the name of an array is a pointer to the 0th element of the array. Can you guess the type of `ch_arr`?

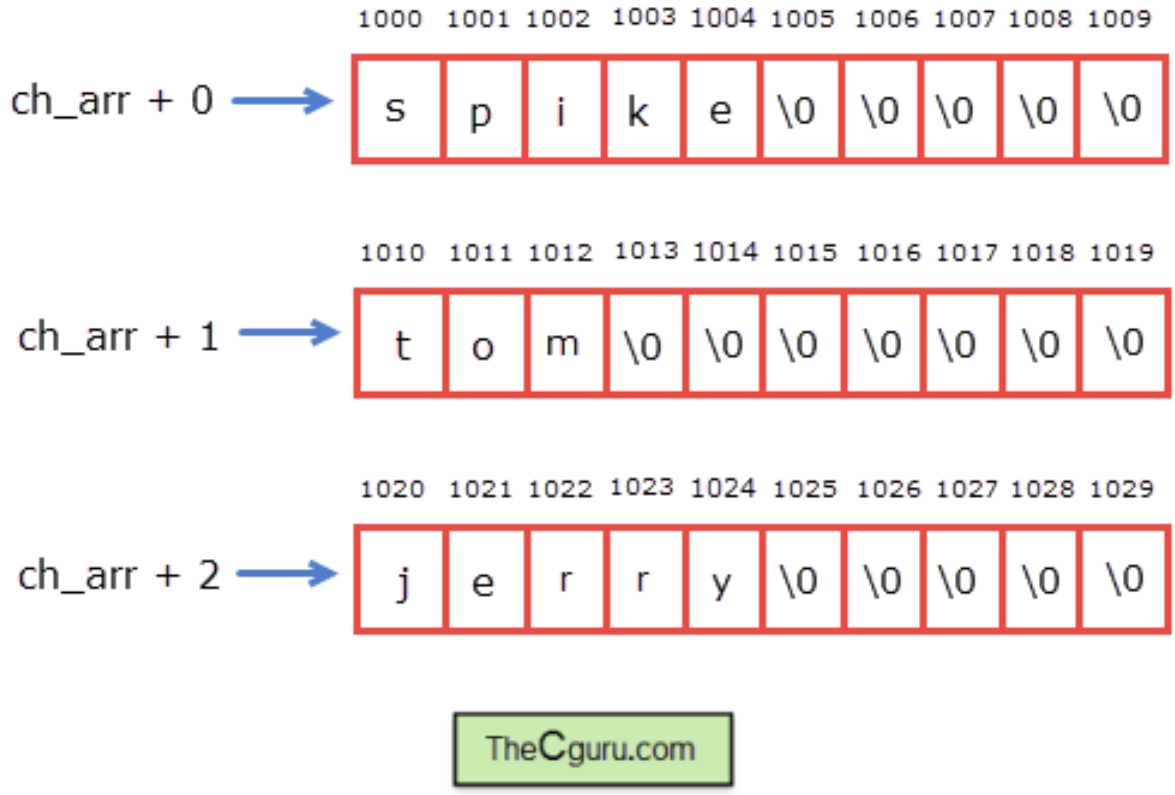
The `ch_arr` is a pointer to an array of `10` characters or `int(*)[10]`.

Therefore, if `ch_arr` points to address `1000` then `ch_arr + 1` will point to address `1010`.

From this, we can conclude that:

`ch_arr + 0` points to the 0th string or 0th 1-D array.
`ch_arr + 1` points to the 1st string or 1st 1-D array.
`ch_arr + 2` points to the 2nd string or 2nd 1-D array.

In general, `ch_arr + i` points to the ith string or ith 1-D array.



We know that when we dereference a pointer to an array, we get the base address of the array. So, on dereferencing `ch_arr + i` we get the base address of the 0th 1-D array.

From this we can conclude that:

`*(ch_arr + 0) + 0` points to the 0th character of 0th 1-D array (i.e `s`)
`*(ch_arr + 0) + 1` points to the 1st character of 0th 1-D array (i.e `p`)
`*(ch_arr + 1) + 2` points to the 2nd character of 1st 1-D array (i.e `m`)

In general, we can say that: `*(ch_arr + i) + j` points to the jth character of ith 1-D array.

Note that the base type of `*(ch_arr + i) + j` is a pointer to `char` or `(char*)`, while the base type of `ch_arr + i` is array of 10 characters or `int(*)[10]`.

To get the element at jth position of ith 1-D array just dereference the whole expression`*(ch_arr + i) + j`.

```
*(*(ch_arr + i) + j)
```

We have learned in chapter Pointers and 2-D arrays that in a 2-D array the pointer notation is equivalent to subscript notation. So the above expression can be written as follows:

```
ch_arr[i][j]
```

The following program demonstrates how to print an array of strings.

```c
#include<stdio.h>

int main()
{
    int i;

    char ch_arr[3][10] = {
                            "spike",
                            "tom",
                            "jerry"
                        };

    printf("1st way \n\n");

    for(i = 0; i < 3; i++)
    {
        printf("string = %s \t address = %u\n", ch_arr + i, ch_arr + i);
    }

    // signal to operating system program ran fine
    return 0;
}
```

Expected Output:

```
1                   string = spike address = 2686736
```

```
2                string = tom address = 2686746
3                string = jerry address = 2686756
```

Some invalid operation on an Array of string

```
1                char ch_arr[3][10] = {
2                                        {'s', 'p', 'i', 'k', 'e',
                 '\0'},
3                                        {'t', 'o', 'm','\0'},
4                                        {'j', 'e', 'r', 'r',
5                'y','\0'}
                                    };
```

It allocates `30` bytes of memory. The compiler will do the same thing even if we don't initialize the elements of the array at the time of declaration.

We already know that the name of an array is a constant pointer so the following operations are invalid.

```
1                ch_arr[0] = "tyke";   // invalid
2                ch_arr[1] = "dragon"; // invalid
```

Here we are trying to assign a string literal (a pointer) to a constant pointer which is obviously not possible.

To assign a new string to `ch_arr` use the following methods.

```
1                strcpy(ch_arr[0], "type"); // valid
2                scanf(ch_arr[0], "type");  // valid
```

Let's conclude this chapter by creating another simple program.

This program asks the user to enter a username. If the username entered is one of the names in the master list then the user is allowed to calculate the factorial of a number. Otherwise, an error message is displayed.

```
1#include<stdio.h>
2#include<string.h>
3int factorial(int );
4
5int main()
```

```
{
    int i, found = 0, n;

    char master_list[5][20] = {
                                    "admin",
                                    "tom",
                                    "bob",
                                    "tim",
                                    "jim"
                                }, name[10];

    printf("Enter username: ");
    gets(name);

    for(i = 0; i < 5; i++)
    {
        if(strcmp(name, master_list[i]) == 0 )
        {
            found = 1;
            break;
        }
    }

    if(found==1)
    {
        printf("\nWelcome %s !\n", name);
        printf("\nEnter a number to calculate the factorial: ");
        scanf("%d", &n);
        printf("Factorial of %d is %d", n, factorial(n));
    }

    else
    {
        printf("Error: You are not allowed to run this program.", name);
    }

    // signal to operating system program ran fine
    return 0;
}

int factorial(int n)
{
    if(n == 0)
    {
        return 1;
    }

    else
    {
        return n * factorial(n-1);
    }
}
```

Expected Output: 1st run:

```
Enter username: admin

Welcome admin !

Enter a number to calculate the factorial: 4
Factorial of 4 is 24
```

2nd run:

```
Enter username: jack
Error: You are not allowed to run this program.
```

How it works:

The program asks the user to enter a name. After the name is entered it compares the entered name with the names in the `master_list` array using `strcmp()` function. If match is found then `strcmp()` returns `0` and the if condition `strcmp(name, master_list[i]) == 0` condition becomes true. The variable found is assigned a value of `1`, which means that the user is allowed to access the program. The program asks the user to enter a number and displays the factorial of a number.

If the name entered is not one of the names in the `master_list` array then the program exits by displaying an error message.

Functions

In c, we can divide a large program into the basic building blocks known as function. The function contains the set of programming statements enclosed by {}. A function can be called multiple times to provide reusability and modularity to the C program. In other words, we can say that the collection of functions creates a program. The function is also known as *procedure*or *subroutine*in other programming languages.

Advantage of functions in C

There are the following advantages of C functions.

- By using functions, we can avoid rewriting same logic/code again and again in a program.

- We can call C functions any number of times in a program and from any place in a program.
- We can track a large C program easily when it is divided into multiple functions.
- Reusability is the main achievement of C functions.
- However, Function calling is always a overhead in a C program.

Function Aspects

There are three aspects of a C function.

- **Function declaration** A function must be declared globally in a c program to tell the compiler about the function name, function parameters, and return type.

- **Function call** Function can be called from anywhere in the program. The parameter list must not differ in function calling and function declaration. We must pass the same number of functions as it is declared in the function declaration.

- **Function definition** It contains the actual statements which are to be executed. It is the most important aspect to which the control comes when the function is called. Here, we must notice that only one value can be returned from the function.

SN	C function aspects	Syntax
1	Function declaration	return_type function_name (argument list);
2	Function call	function_name (argument_list)
3	Function definition	return_type function_name (argument list) {function body;}

The syntax of creating function in c language is given below

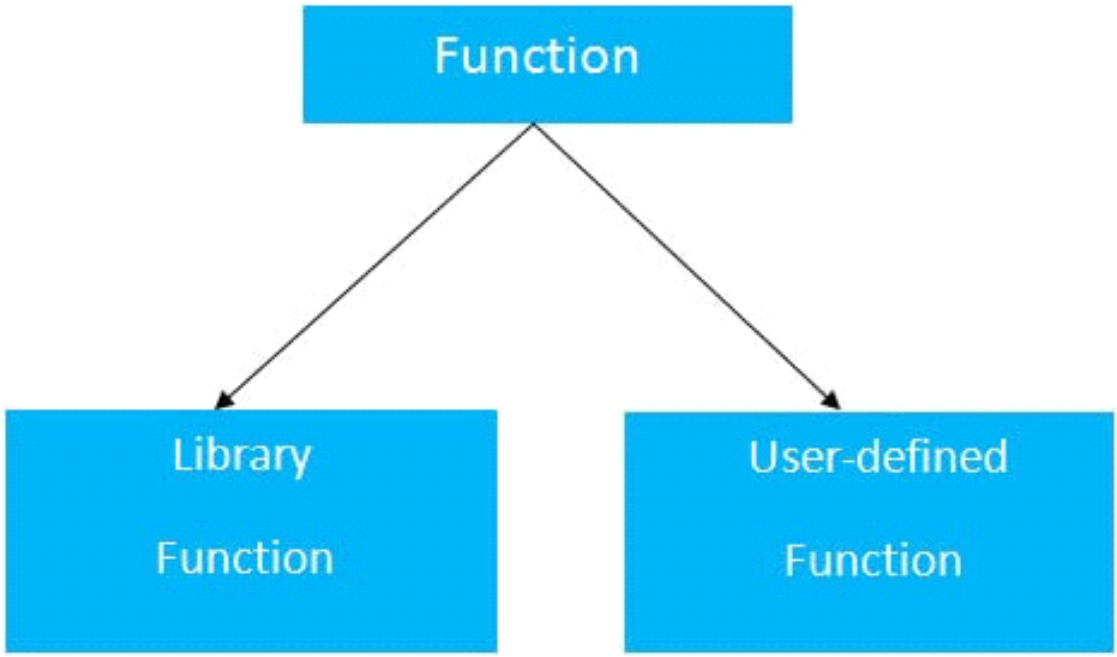
- return_type function_name(data_type parameter...){
- //code to be executed

- }

Types of Functions

There are two types of functions in C programming:

- **Library Functions**: are the functions which are declared in the C header files such as scanf(), printf(), gets(), puts(), ceil(), floor() etc.
- **User-defined functions**: are the functions which are created by the C programmer, so that he/she can use it many times. It reduces the complexity of a big program and optimizes the code.



Return Value

A C function may or may not return a value from the function. If you don't have to return any value from the function, use void for the return type.

Let's see a simple example of C function that doesn't return any value from the function.

Example without return value:

- **void** hello(){
- printf("hello c");

- }

If you want to return any value from the function, you need to use any data type such as int, long, char, etc. The return type depends on the value to be returned from the function.

Let's see a simple example of C function that returns int value from the function.

Example with return value:

- **int** get(){
- **return** 10;
- }

In the above example, we have to return 10 as a value, so the return type is int. If you want to return floating-point value (e.g., 10.2, 3.1, 54.5, etc), you need to use float as the return type of the method.

- **float** get(){
- **return** 10.2;
- }

Now, you need to call the function, to get the value of the function.

Different aspects of function calling

A function may or may not accept any argument. It may or may not return any value. Based on these facts, There are four different aspects of function calls.

- function without arguments and without return value
- function without arguments and with return value
- function with arguments and without return value
- function with arguments and with return value

Example for Function without argument and return value

Example 1

- #include<stdio.h>
- **void** printName();
- **void** main ()

```
{
    printf("Hello ");
    printName();
}
void printName()
{
    printf("topperworld");
}
```

Output

```
Hello topperworld
```

Example 2

```
#include<stdio.h>
void sum();
void main()
{
    printf("\nGoing to calculate the sum of two numbers:");
    sum();
}
void sum()
{
    int a,b;
    printf("\nEnter two numbers");
    scanf("%d %d",&a,&b);
    printf("The sum is %d",a+b);
}
```

Output

```
Going to calculate the sum of two numbers:
```

```
Enter two numbers 10
24

The sum is 34
```

C Library Functions

Library functions are the inbuilt function in C that are grouped and placed at a common place called the library. Such functions are used to perform some specific operations. For example, printf is a library function used to print on the console. The library functions are created by the designers of compilers. All C standard library functions are defined inside the different header files saved with the extension **.h**. We need to include these header files in our program to make use of the library functions defined in such header files. For example, To use the library functions such as printf/scanf we need to include stdio.h in our program which is a header file that contains all the library functions regarding standard input/output.

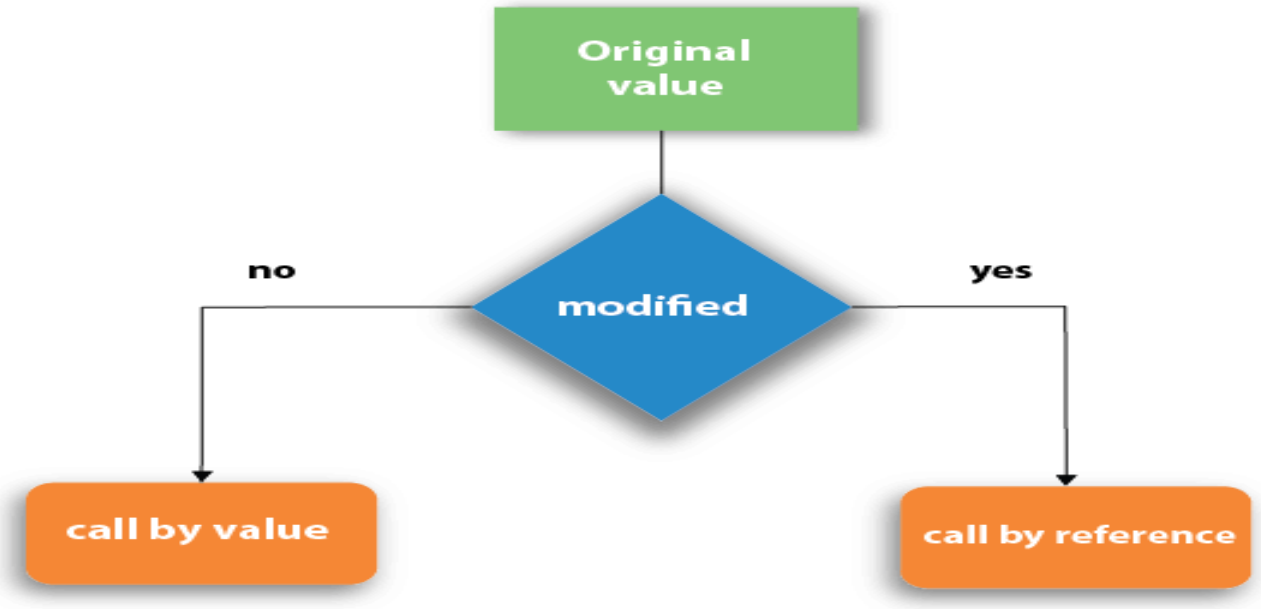
The list of mostly used header files is given in the following table.

SN	Header file	Description
1	stdio.h	This is a standard input/output header file. It contains all the library functions regarding standard input/output.
2	conio.h	This is a console input/output header file.
3	string.h	It contains all string related library functions like gets(), puts(),etc.
4	stdlib.h	This header file contains all the general library functions like malloc(), calloc(), exit(), etc.
5	math.h	This header file contains all the math operations related functions like sqrt(), pow(), etc.
6	time.h	This header file contains all the time-related functions.
7	ctype.h	This header file contains all character handling functions.
8	stdarg.h	Variable argument functions are defined in this header file.
9	signal.h	All the signal handling functions are defined in this header file.
10	setjmp.h	This file contains all the jump functions.
11	locale.h	This file contains locale functions.

12	errno.h	This file contains error handling functions.
13	assert.h	This file contains diagnostics functions.

Call by value and Call by reference in C

There are two methods to pass the data into the function in C language, i.e., *call by value* and *call by reference*.



Let's understand call by value and call by reference in c language one by one.

Call by value in C

- In call by value method, the value of the actual parameters is copied into the formal parameters. In other words, we can say that the value of the variable is used in the function call in the call by value method.
- In call by value method, we can not modify the value of the actual parameter by the formal parameter.

- In call by value, different memory is allocated for actual and formal parameters since the value of the actual parameter is copied into the formal parameter.
- The actual parameter is the argument which is used in the function call whereas formal parameter is the argument which is used in the function definition.

Let's try to understand the concept of call by value in c language by the example given below:

```
#include<stdio.h>
void change(int num) {
    printf("Before adding value inside function num=%d \n",num);
    num=num+100;
    printf("After adding value inside function num=%d \n", num);
}
int main() {
    int x=100;
    printf("Before function call x=%d \n", x);
    change(x);//passing value in function
    printf("After function call x=%d \n", x);
return 0;
}
```

Output

```
Before function call x=100
Before adding value inside function num=100
After adding value inside function num=200
After function call x=100
```

Call by Value Example: Swapping the values of the two variables

```
#include <stdio.h>
void swap(int , int); //prototype of the function
int main()
{
    int a = 10;
    int b = 20;
    printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing the value of a and b in main
    swap(a,b);
    printf("After swapping values in main a = %d, b = %d\n",a,b); // The value of actual parameters do not change by changing the formal parameters in call by value, a = 10, b = 20
}
void swap (int a, int b)
{
    int temp;
    temp = a;
    a=b;
    b=temp;
    printf("After swapping values in function a = %d, b = %d\n",a,b); // Formal parameters, a = 20, b = 10
}
```

Output

```
Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
After swapping values in main a = 10, b = 20
```

Call by reference in C

- In call by reference, the address of the variable is passed into the function call as the actual parameter.
- The value of the actual parameters can be modified by changing the formal parameters since the address of the actual parameters is passed.
- In call by reference, the memory allocation is similar for both formal parameters and actual parameters. All the operations in the function are performed on the value stored at the address of the actual parameters, and the modified value gets stored at the same address.

Consider the following example for the call by reference.

```
#include<stdio.h>
void change(int *num) {
    printf("Before adding value inside function num=%d \n",*num);
    (*num) += 100;
    printf("After adding value inside function num=%d \n", *num);
}
int main() {
   int x=100;
    printf("Before function call x=%d \n", x);
   change(&x);//passing reference in function
    printf("After function call x=%d \n", x);
return 0;
}
```

Output

```
Before function call x=100
Before adding value inside function num=100
After adding value inside function num=200
After function call x=200
```

Call by reference Example: Swapping the values of the two variables

```
#include <stdio.h>
```

```
void swap(int *, int *); //prototype of the function
int main()
{
    int a = 10;
    int b = 20;
    printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing the value of a and b in main
    swap(&a,&b);
    printf("After swapping values in main a = %d, b = %d\n",a,b); // The values of actual parameters do change in call by reference, a = 10, b = 20
}
void swap (int *a, int *b)
{
    int temp;
    temp = *a;
    *a=*b;
    *b=temp;
    printf("After swapping values in function a = %d, b = %d\n",*a,*b); // Formal parameters, a = 20, b = 10
}
```

Output

```
Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
After swapping values in main a = 20, b = 10
```

Difference between call by value and call by reference in c

No.	Call by value	Call by reference

1	A copy of the value is passed into the function	An address of value is passed into the function
2	Changes made inside the function is limited to the function only. The values of the actual parameters do not change by changing the formal parameters.	Changes made inside the function validate outside of the function also. The values of the actual parameters do change by changing the formal parameters.
3	Actual and formal arguments are created at the different memory location	Actual and formal arguments are created at the same memory location

Recursion in C

Recursion is the process which comes into existence when a function calls a copy of itself to work on a smaller problem. Any function which calls itself is called recursive function, and such function calls are called recursive calls. Recursion involves several numbers of recursive calls. However, it is important to impose a termination condition of recursion. Recursion code is shorter than iterative code however it is difficult to understand.

Recursion cannot be applied to all the problem, but it is more useful for the tasks that can be defined in terms of similar subtasks. For Example, recursion may be applied to sorting, searching, and traversal problems.

Generally, iterative solutions are more efficient than recursion since function call is always overhead. Any problem that can be solved recursively, can also be solved iteratively. However, some problems are best suited to be solved by the recursion, for example, tower of Hanoi, Fibonacci series, factorial finding, etc.

In the following example, recursion is used to calculate the factorial of a number.

```
#include <stdio.h>
int fact (int);
int main()
{
    int n,f;
    printf("Enter the number whose factorial you want to calculate?");
    scanf("%d",&n);
```

```
    f = fact(n);
    printf("factorial = %d",f);
}
int fact(int n)
{
    if (n==0)
    {
        return 0;
    }
    else if ( n == 1)
    {
        return 1;
    }
    else
    {
        return n*fact(n-1);
    }
}
```

Output

```
Enter the number whose factorial you want to calculate?5
factorial = 120
```

We can understand the above program of the recursive method call by the figure given below:

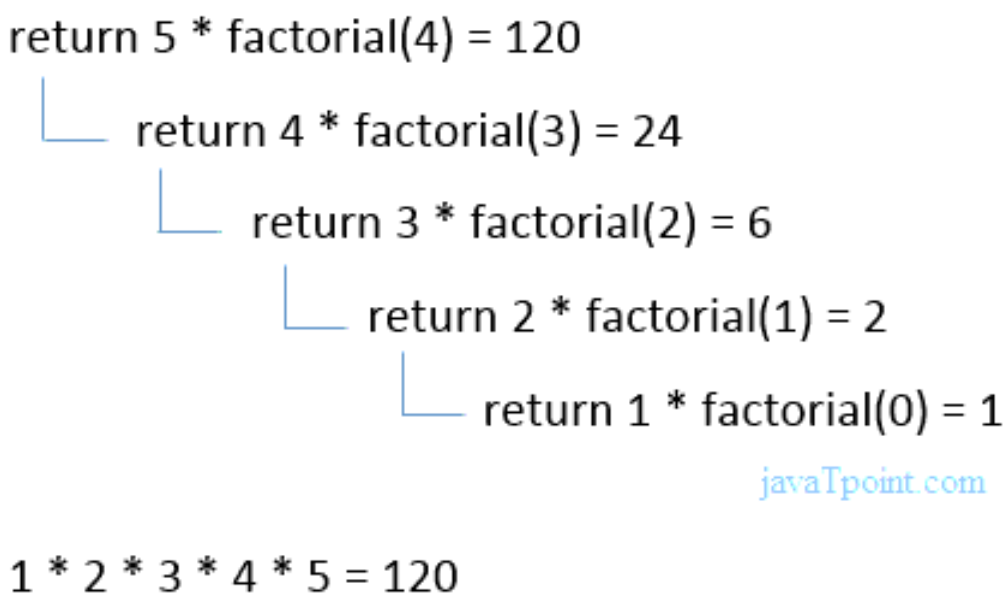


Fig: Recursion

Recursive Function

A recursive function performs the tasks by dividing it into the subtasks. There is a termination condition defined in the function which is satisfied by some specific subtask. After this, the recursion stops and the final result is returned from the function.

The case at which the function doesn't recur is called the base case whereas the instances where the function keeps calling itself to perform a subtask, is called the recursive case. All the recursive functions can be written using this format.

Pseudocode for writing any recursive function is given below.

```
if (test_for_base)
{
    return some_value;
}
else if (test_for_another_base)
{
```

```
    return some_another_value;
}
else
{
    // Statements;
    recursive call;
}
```

Example of recursion in C

Let's see an example to find the nth term of the Fibonacci series

```
#include<stdio.h>
int fibonacci(int);
void main ()
{
    int n,f;
    printf("Enter the value of n?");
    scanf("%d",&n);
    f = fibonacci(n);
    printf("%d",f);
}
int fibonacci (int n)
{
    if (n==0)
    {
    return 0;
    }
    else if (n == 1)
    {
```

```
        return 1;
    }
    else
    {
        return fibonacci(n-1)+fibonacci(n-2);
    }
}
```

Output

```
Enter the value of n?12
144
```

Return an Array in C

An array is a type of data structure that stores a fixed-size of a homogeneous collection of data. In short, we can say that array is a collection of variables of the same type.

For example, if we want to declare 'n' number of variables, n1, n2...n., if we create all these variables individually, then it becomes a very tedious task. In such a case, we create an array of variables having the same type. Each element of an array can be accessed using an index of the element.

Let's first see how to pass a single-dimensional array to a function.

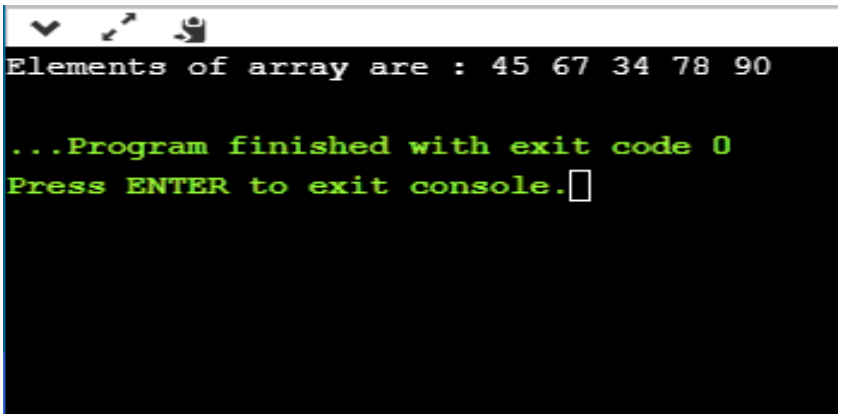
Passing array to a function

```
#include <stdio.h>
void getarray(int arr[])
{
    printf("Elements of array are : ");
    for(int i=0;i<5;i++)
    {
```

```
        printf("%d ", arr[i]);
    }
}
int main()
{
   int arr[5]={45,67,34,78,90};
   getarray(arr);
   return 0;
}
```

In the above program, we have first created the array **arr[]** and then we pass this array to the function getarray(). The **getarray()** function prints all the elements of the array arr[].

Output



Passing array to a function as a pointer

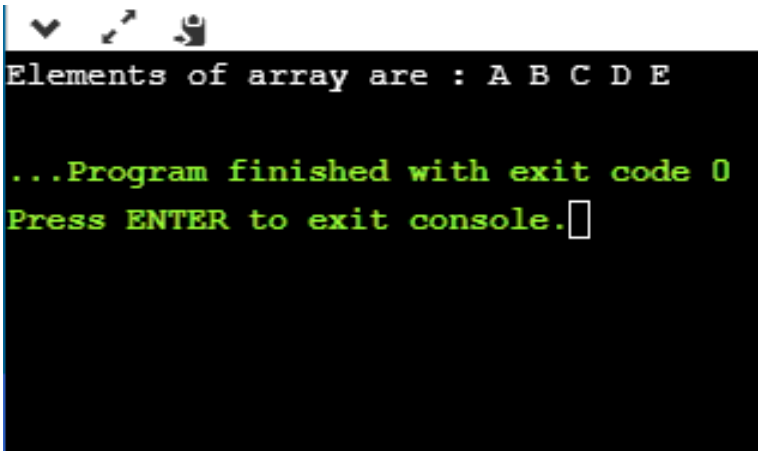
Now, we will see how to pass an array to a function as a pointer.

```
#include <stdio.h>
void printarray(char *arr)
{
   printf("Elements of array are : ");
   for(int i=0;i<5;i++)
   {
       printf("%c ", arr[i]);
```

```
    }
}
int main()
{
  char arr[5]={'A','B','C','D','E'};
  printarray(arr);
  return 0;
}
```

In the above code, we have passed the array to the function as a pointer. The function **printarray()** prints the elements of an array.

Output



Note: From the above examples, we observe that array is passed to a function as a reference which means that array also persist outside the function.

How to return an array from a function

Returning pointer pointing to the array

```
#include <stdio.h>
int *getarray()
```

```
{
    int arr[5];
    printf("Enter the elements in an array : ");
    for(int i=0;i<5;i++)
    {
        scanf("%d", &arr[i]);
    }
    return arr;
}
int main()
{
  int *n;
   n=getarray();
   printf("\nElements of array are :");
   for(int i=0;i<5;i++)
    {
        printf("%d", n[i]);
    }
    return 0;
}
```

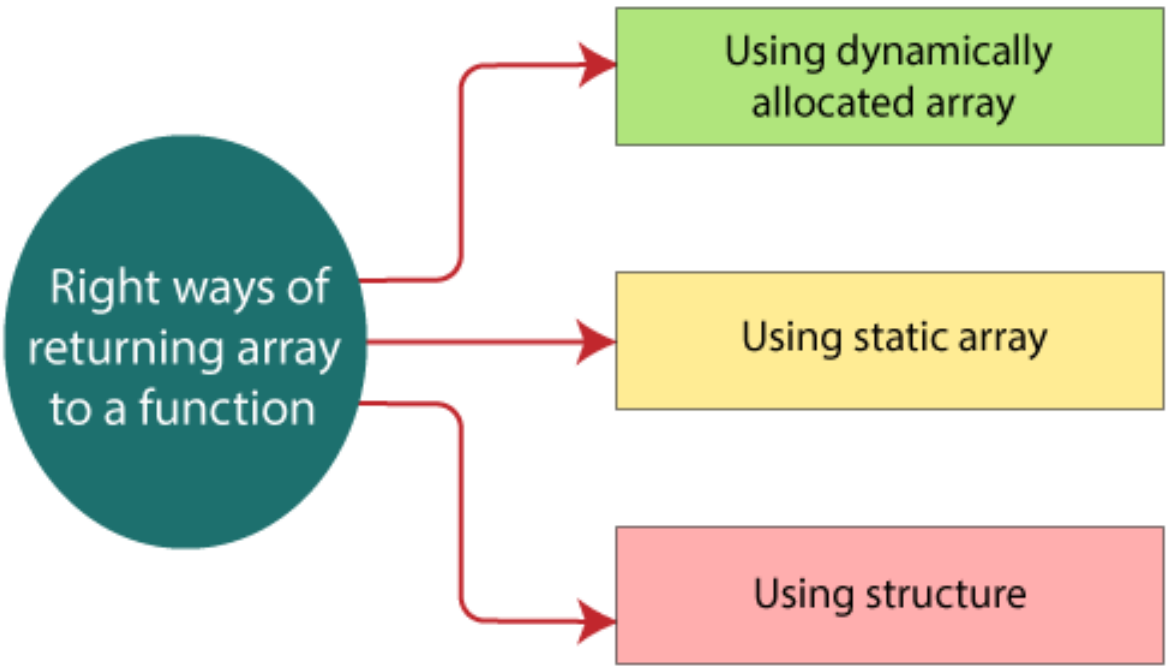
In the above program, **getarray()** function returns a variable 'arr'. It returns a local variable, but it is an illegal memory location to be returned, which is allocated within a function in the stack. Since the program control comes back to the **main()** function, and all the variables in a stack are freed. Therefore, we can say that this program is returning memory location, which is already de-allocated, so the output of the program is a **segmentation fault**.

Output

```
input
main.c:27:12: warning: function returns address of local variable [-Wreturn-local-addr]
Array inside function: 1
2
3
4
5
Array outside function:
Segmentation fault (core dumped)
```

There are three right ways of returning an array to a function:

- **Using dynamically allocated array**
- **Using static array**
- **Using structure**

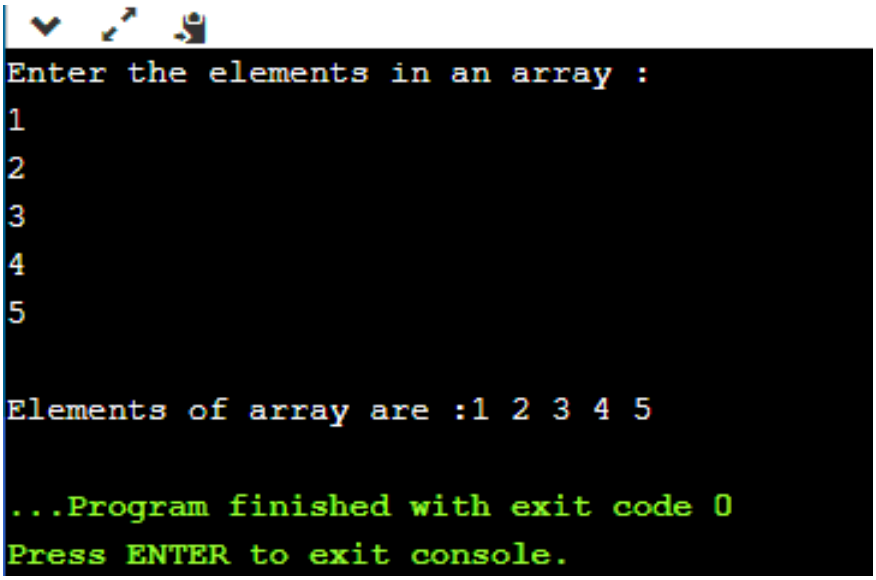


Returning array by passing an array which is to be returned as a parameter to the function.

```
#include <stdio.h>
int *getarray(int *a)
{

    printf("Enter the elements in an array : ");
    for(int i=0;i<5;i++)
    {
        scanf("%d", &a[i]);
    }
    return a;
}
int main()
{
  int *n;
  int a[5];
  n=getarray(a);
  printf("\nElements of array are :");
  for(int i=0;i<5;i++)
    {
        printf("%d", n[i]);
    }
    return 0;
}
```

Output

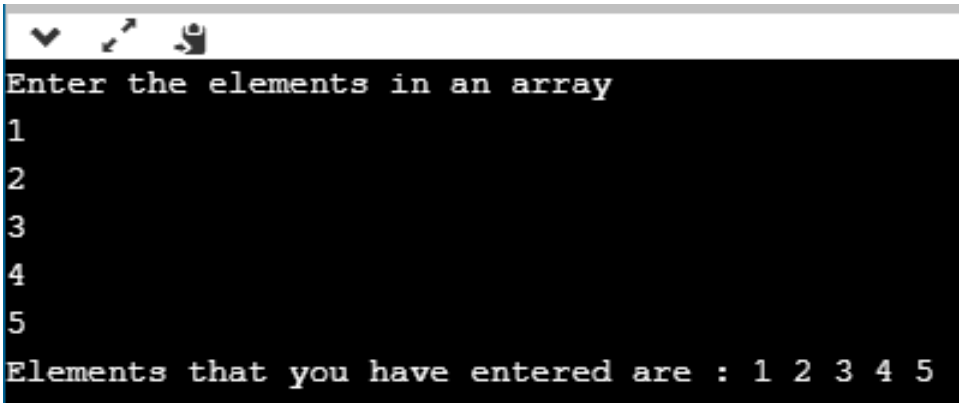


Returning array using malloc() function.

```
#include <stdio.h>
#include<malloc.h>
int *getarray()
{
    int size;
    printf("Enter the size of the array : ");
    scanf("%d",&size);
    int *p= malloc(sizeof(size));
    printf("\nEnter the elements in an array");
    for(int i=0;i<size;i++)
    {
        scanf("%d",&p[i]);
    }
```

```
    return p;
}
int main()
{
   int *ptr;
   ptr=getarray();
   int length=sizeof(*ptr);
   printf("Elements that you have entered are : ");
   for(int i=0;ptr[i]!='\0';i++)
    {
      printf("%d ", ptr[i]);
    }
  return 0;
}
```

Output



Using Static Variable

```
#include <stdio.h>
int *getarray()
{
  static int arr[7];
  printf("Enter the elements in an array : ");
  for(int i=0;i<7;i++)
  {
     scanf("%d",&arr[i]);
  }
  return arr;

}
int main()
{
  int *ptr;
  ptr=getarray();
  printf("\nElements that you have entered are :");
  for(int i=0;i<7;i++)
  {
     printf("%d ", ptr[i]);
  }
}
```

In the above code, we have created the variable **arr[]** as static in **getarray()** function, which is available throughout the program. Therefore, the function getarray() returns the actual memory location of the variable '**arr**'.

Output

```
: ./main
Enter the elements in an array :
10
20
30
40
50
60
70

Elements that you have entered are :10 20 30 40 50 60 70
```

Using Structure

The structure is a user-defined data type that can contain a collection of items of different types. Now, we will create a program that returns an array by using structure.

```
#include <stdio.h>
#include<malloc.h>
struct array
{
    int arr[8];
};
struct array getarray()
{
    struct array y;
    printf("Enter the elements in an array : ");
    for(int i=0;i<8;i++)
    {
        scanf("%d",&y.arr[i]);
```

```
    }
    return y;
}
int main()
{
  struct array x=getarray();
  printf("Elements that you have entered are :");
  for(int i=0;x.arr[i]!='\0';i++)
  {
     printf("%d ", x.arr[i]);
  }
    return 0;
}
```

Unit-4

Pointers

C Pointers

The pointer in C language is a variable which stores the address of another variable. This variable can be of type int, char, array, function, or any other pointer. The size of the pointer depends on the architecture. However, in 32-bit architecture the size of a pointer is 2 byte.

Consider the following example to define a pointer which stores the address of an integer.

```
int n = 10;
int* p = &n; // Variable p of type pointer is pointing to the address of the variable n of type integer.
```

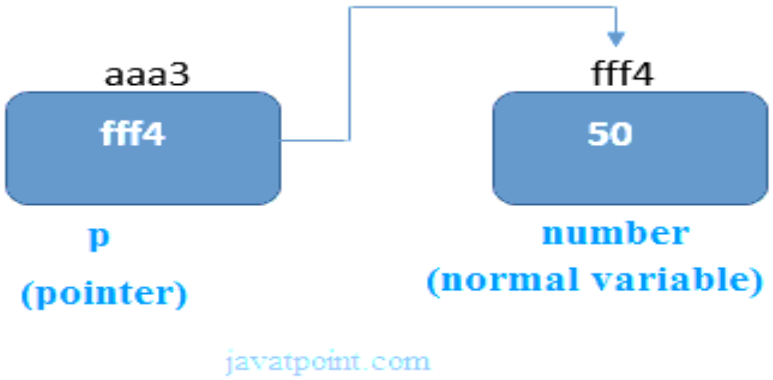
Declaring a pointer

The pointer in c language can be declared using * (asterisk symbol). It is also known as indirection pointer used to dereference a pointer.

- **int** *a;//pointer to int
- **char** *c;//pointer to char

Pointer Example

An example of using pointers to print the address and value is given below.



As you can see in the above figure, pointer variable stores the address of number variable, i.e., fff4. The value of number variable is 50. But the address of pointer variable p is aaa3.

By the help of * (**indirection operator**), we can print the value of pointer variable p.

Let's see the pointer example as explained for the above figure.

- #include<stdio.h>
- **int** main(){
- **int** number=50;
- **int** *p;
- p=&number;//stores the address of number variable
- printf("Address of p variable is %x \n",p); // p contains the address of the number therefore printing p gives the address of number.

- printf("Value of p variable is %d \n",*p); // As we know that * is used to dereferen ce a pointer therefore if we print *p, we will get the value stored at the address co ntained by p.
- **return** 0;
- }

Output

```
Address of number variable is fff4
Address of p variable is fff4
Value of p variable is 50
```

Pointer to array

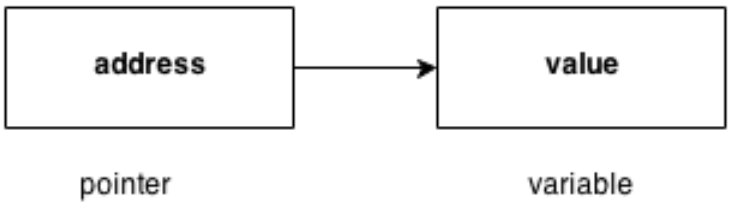
- **int** arr[10];
- **int** *p[10]=&arr; // Variable p of type pointer is pointing to the address of an integer arra y arr.

Pointer to a function

- **void** show (**int**);
- **void**(*p)(**int**) = &display; // Pointer p is pointing to the address of a function

Pointer to structure

- **struct** st {
- **int** i;
- **float** f;
- }ref;
- **struct** st *p = &ref;



Advantage of pointer

1) Pointer **reduces the code** and **improves the performance**, it is used to retrieving strings, trees, etc. and used with arrays, structures, and functions.

2) We can **return multiple values from a function** using the pointer.

3) It makes you able to **access any memory location** in the computer's memory.

Usage of pointer

There are many applications of pointers in c language.

1) Dynamic memory allocation

In c language, we can dynamically allocate memory using malloc() and calloc() functions where the pointer is used.

2) Arrays, Functions, and Structures

Pointers in c language are widely used in arrays, functions, and structures. It reduces the code and improves the performance.

Address Of (&) Operator

The address of operator '&' returns the address of a variable. But, we need to use %u to display the address of a variable.

```
#include<stdio.h>
int main(){
int number=50;
printf("value of number is %d, address of number is %u",number,&number);
return 0;
}
```

Output

```
value of number is 50, address of number is fff4
```

NULL Pointers

It is always a good practice to assign a NULL value to a pointer variable in case you do not have an exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a **null** pointer.

The NULL pointer is a constant with a value of zero defined in several standard libraries. Consider the following program –

```
#include <stdio.h>

int main () {

   int  *ptr = NULL;

   printf("The value of ptr is : %x\n", ptr  );

   return 0;
}
```

When the above code is compiled and executed, it produces the following result –

```
The value of ptr is 0
```

In most of the operating systems, programs are not permitted to access memory at address 0 because that memory is reserved by the operating system. However, the memory address 0 has special significance; it signals that the pointer is not intended to point to an accessible memory location. But by convention, if a pointer contains the null (zero) value, it is assumed to point to nothing.

To check for a null pointer, you can use an 'if' statement as follows –

```
if(ptr)     /* succeeds if p is not null */
if(!ptr)    /* succeeds if p is null */
```

C Pointers & Arrays with Examples

Traditionally, we access the array elements using its index, but this method can be eliminated by using pointers. Pointers make it easy to access each array element.

```
#include <stdio.h>
int main()
{
    int a[5]={1,2,3,4,5};   //array initialization
    int *p;     //pointer declaration
               /*the ptr points to the first element of the array*/

    p=a; /*We can also type simply ptr==&a[0] */

    printf("Printing the array elements using pointer\n");
    for(int i=0;i<5;i++)    //loop for traversing array elements
    {
               printf("\n%x",*p);  //printing array elements
               p++;    //incrementing to the next element, you can also write
p=p+1
```

```
    }
    return 0;
}
```

Output:

```
1
2
3
4
5
```

Adding a particular number to a pointer will move the pointer location to the value obtained by an addition operation. Suppose p is a pointer that currently points to the memory location 0 if we perform following addition operation, p+1 then it will execute in this manner:

Pointer Addition/Increment

Since p currently points to the location 0 after adding 1, the value will become 1, and hence the pointer will point to the memory location 1.

C Pointers and Strings with Examples

A string is an array of char objects, ending with a null character '\ 0'. We can manipulate strings using pointers. This pointer in C example explains this section

```
#include <stdio.h>
#include <string.h>
int main()
{
char str[]="Hello Guru99!";
char *p;
p=str;
printf("First character is:%c\n",*p);
p =p+1;
printf("Next character is:%c\n",*p);
printf("Printing all the characters in a string\n");
p=str;  //reset the pointer
for(int i=0;i<strlen(str);i++)
{
printf("%c\n",*p);
p++;
}
return 0;
}
```

Output:

```
First character is:H
Next character is:e
Printing all the characters in a string
H
```

```
e
l
l
o

G
u
r
u
9
9
!
```

Another way to deal strings is with an array of pointers like in the following program:

```
#include <stdio.h>
int main(){
char *materials[ ] = {  "iron",  "copper",  "gold"};
printf("Please remember these materials :\n");
int i ;
for (i = 0; i < 3; i++) {
  printf("%s\n", materials[ i ]);}
  return 0;}
```

Output:

```
Please remember these materials:
iron
copper
gold
```

Pointers and functions in C

As we discussed in the previous chapter, a pointer can point to a function in C. However, the declaration of the pointer variable must be the same as the function. Consider the following example to make a pointer pointing to the function.

```
#include<stdio.h>
int addition ();
int main ()
{
    int result;
    int (*ptr)();
    ptr = &addition;
    result = (*ptr)();
    printf("The sum is %d",result);
```

```
}
int addition()
{
    int a, b;
    printf("Enter two numbers?");
    scanf("%d %d",&a,&b);
    return a+b;
}
```

Output

```
Enter two numbers?10 15
The sum is 25
```

Dynamic memory allocation in C

The concept of **dynamic memory allocation in c language** *enables the C programmer to allocate memory at runtime*. Dynamic memory allocation in c language is possible by 4 functions of stdlib.h header file.

- malloc()
- calloc()
- realloc()
- free()

Before learning above functions, let's understand the difference between static memory allocation and dynamic memory allocation.

static memory allocation	dynamic memory allocation
memory is allocated at compile time.	memory is allocated at run time.
memory can't be increased while executing program.	memory can be increased while executing program.
used in array.	used in linked list.

Now let's have a quick look at the methods used for dynamic memory allocation.

malloc()	allocates single block of requested memory.
calloc()	allocates multiple block of requested memory.
realloc()	reallocates the memory occupied by malloc() or calloc() functions.
free()	frees the dynamically allocated memory.

malloc() function in C

The malloc() function allocates single block of requested memory.

It doesn't initialize memory at execution time, so it has garbage value initially.

It returns NULL if memory is not sufficient.

The syntax of malloc() function is given below:

- ptr=(cast-type*)malloc(byte-size)

Let's see the example of malloc() function.

```
#include<stdio.h>
#include<stdlib.h>
int main(){
  int n,i,*ptr,sum=0;
    printf("Enter number of elements: ");
    scanf("%d",&n);
    ptr=(int*)malloc(n*sizeof(int));  //memory allocated using malloc
    if(ptr==NULL)
    {
        printf("Sorry! unable to allocate memory");
        exit(0);
    }
    printf("Enter elements of array: ");
    for(i=0;i<n;++i)
```

```
    {
        scanf("%d",ptr+i);
        sum+=*(ptr+i);
    }
    printf("Sum=%d",sum);
    free(ptr);
return 0;
}
```

Output

```
Enter elements of array: 3
Enter elements of array: 10
10
10
Sum=30
```

calloc() function in C

The calloc() function allocates multiple block of requested memory.

It initially initialize all bytes to zero.

It returns NULL if memory is not sufficient.

The syntax of calloc() function is given below:

- ptr=(cast-type*)calloc(number, byte-size)

Let's see the example of calloc() function.

```
#include<stdio.h>
#include<stdlib.h>
int main(){
 int n,i,*ptr,sum=0;
    printf("Enter number of elements: ");
    scanf("%d",&n);
    ptr=(int*)calloc(n,sizeof(int));  //memory allocated using calloc
    if(ptr==NULL)
```

```
    {
        printf("Sorry! unable to allocate memory");
        exit(0);
    }
    printf("Enter elements of array: ");
    for(i=0;i<n;++i)
    {
        scanf("%d",ptr+i);
        sum+=*(ptr+i);
    }
    printf("Sum=%d",sum);
    free(ptr);
    return 0;
}
```

Output

```
Enter elements of array: 3
Enter elements of array: 10
10
10
Sum=30
```

realloc() function in C

If memory is not sufficient for malloc() or calloc(), you can reallocate the memory by realloc() function. In short, it changes the memory size.

Let's see the syntax of realloc() function.

- ptr=realloc(ptr, **new**-size)

free() function in C

The memory occupied by malloc() or calloc() functions must be released by calling free() function. Otherwise, it will consume memory until program exit.

Let's see the syntax of free() function.

- free(ptr)

Structure And Union

What are Structures in C?

Structure stores the different types of elements i.e heterogeneous elements. The struct keyword is used to define structure.

Syntax

```
struct structure_name
{
data_type member1;

data_type memeberN;
};
```

Declaring structure variable

You can declare structure variable in two ways:-

1. By struct keyword within main() function

2. By declaring variable at the time of defining structure.

Example

```
#include <stdio.h>
#include <string.h>
struct student
{
int rollno;
char name[60];
}s1;  //declaring s1 variable for structure
void main( )
{
//store first employee information
s1.rollno=1;
strcpy(s1.name, “intellipaat”);//copying string into char array
//printing first employee information
printf( "Rollno : %d\n", s1.rollno);
printf( "Name : %s\n", s1.name);
}
```

Output

Rollno : 1

Name : intellipaat

What are Unions in C?

Union also stores the different types of elements i.e heterogeneous elements. The union keyword is used to define structure. Union takes the memory of largest member only so occupies less memory than structures.

Syntax

```
union union_name
{
data_type member1;

data_type memeberN;
};
```

Example

```
#include <stdio.h>
#include <string.h>
union student
{
int rollno;
char name[60];
}s1;  //declaring s1 variable for union
void main( )
{
//store first employee information
s1.rollno=1;
strcpy(s1.name, “intellipaat”);//copying string into char array
//printing first employee information
printf( "Rollno : %d\n", s1.rollno);
printf( "Name : %s\n", s1.name);
}
```

Output

Rollno : 3546656

Name:intellipaat

Rollno takes garbage value because name has large memory size. So only name will have actual value.

Uses of Union

To me, the most useful cases are two:

First one is when data in the union will be used to be sent/written using a driver.

```
union employeeData {
   struct eData sEmployee;
   char cEmployeeSerialized[64];
} employee;
```

```
// Get employee data from data base
getEmployeeQuery(int id, &employee.sEmployee);

// Send this data through serial port using driver
// where driver takes an array to be sent
uartTxBuffer(&employee.cEmployeeSerialized);
```

Second is when you work on a limited resources platform (embedded system with low profile microcontroller for instance), and you want to save memory by using a general purpose buffer to store temporary data from different points of your program flow, but data comes in different formats.

```
union multiPurposeBuffer {
   float modulatorOutput[256];
   int interleaverOutput[256];
   char scramblerOutput[1024];
} multiBuffer;
```