UNIT-1

Components of Computer system (or) functional unit of a computer :

→ Computer is an electronic device that takes input data from the input devices and stores and process these data and produce output.

Data: Data is a collection of unorganized facts.

Information: processed data (or) organized data.

Components of computer can be divided into two catogories

(1.) Hardware

(2.) Software

Hardware: Computer hardware includes the physical parts of a computer, such as central processing unit (CPU), monitor, mouse, keyboard.

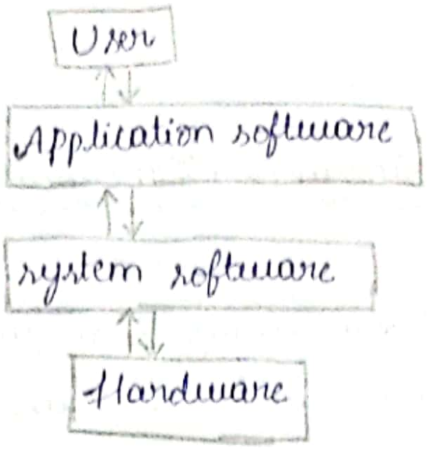
Software: software, instructions that tell a computer what to do. It is mainly of two types named as, system software and application software.

(a.) **System software:** system software acts as the interface between the application software and hardware.

Ex: windows xp.

(b.) **Application software:** Application software acts as an interface between the system software and user.

Ex: MS office, Antivirus etc....



All types of computers follows five basic operations for converting raw input data into information useful to their users.

Operation's:-

(1.) Take input:- The process of entering data and instructions into the computer system.

(2.) Store data:- saving data and instructions so that they are available for processing when required.

(3.) processing data:- performing arithmatic and logical operations on data in order to convert them into useful information.

(4.) Output information:- The process of producing useful information or result for the user such as printed report or visual display.

(5.) Control the workflow:- Directs the manner and sequence in which all the above operations are performed.

- Computer systems consists of three components, they are;

  (i.) Central processing unit.

(ii) Input unit
(iii) Output unit.

(i.) Input unit:- This unit contains devices with the help of which we enter data into the computer. This unit creates a link between the user and the computer. The input devices translates the information into a form understandable by the computer.

(ii.) CPU (central processing unit):-

CPU is considered as the brain of the computer.

CPU performs all types of data processing operations. It stores data. Intermediate results and program. It controls the operations of all parts of the computer.

- CPU itself has the following three components;

(a) ALU (Arithmatic logical unit):

Data entered into computer is sent to RAM, from where it is then sent to ALU, where rest of data processing takes place. All types of processing, such as comparisions, decision making takes place here and once again data is moved to RAM.

(b) Control unit:-

As name indicates, this part of CPU extracts instructions, performs execution, maintains and directs operations of entire system.

(c.) memory unit:

This is unit in which data and instructions given to computer as well as results given by computer are

are stored. unit of memory is "Byte".

$\boxed{\text{1 Byte = 8 Bits.}}$

(iii.) Output unit:-

The output unit consists of devices with the help of which we get the information from the computer. This unit is a link between the computer and the users. Output devices translates the computers output into a form understandable by the users.

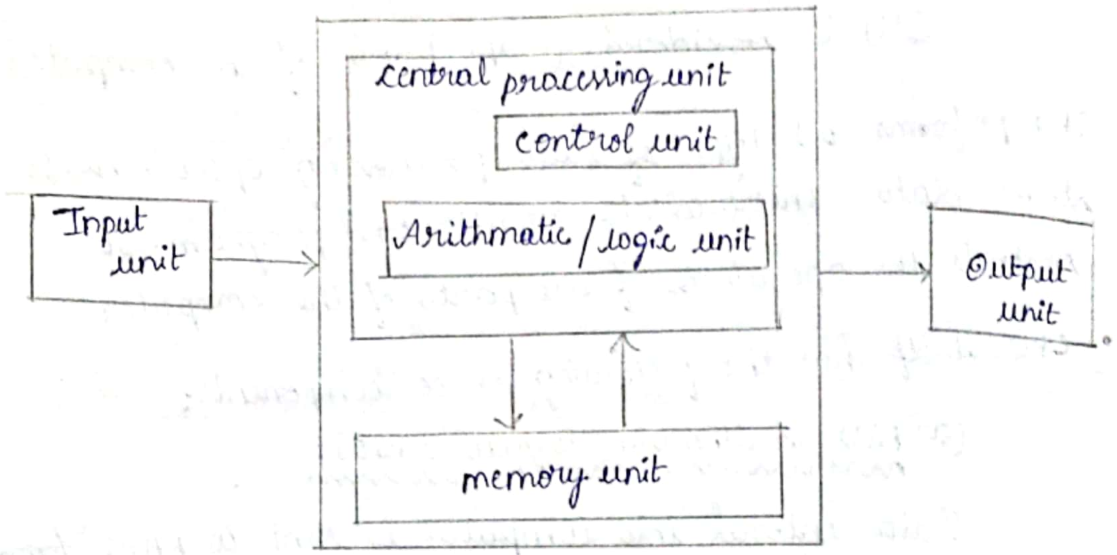


Fig: Von-Neumann Architecture.

Computer language:-

→ computer languages mainly divided into 3-types;

(i.) Machine level languages (1940).
(ii.) Assembly (or) symbolic languages (1950). } low-level languages.
(iii.) High-level languages (1960).

High level language:-

→ The language which can be easily understandable by the user is called as high-level language.

→ Generally these language instructions are in english like language

Assembly language:-

→ Assembly language is also called as "symbolic language".
→ This language uses some symbols to write programs.
Eg:- Move, ADD, SUB........
→ It is also understandable by the user.

Machine language:-

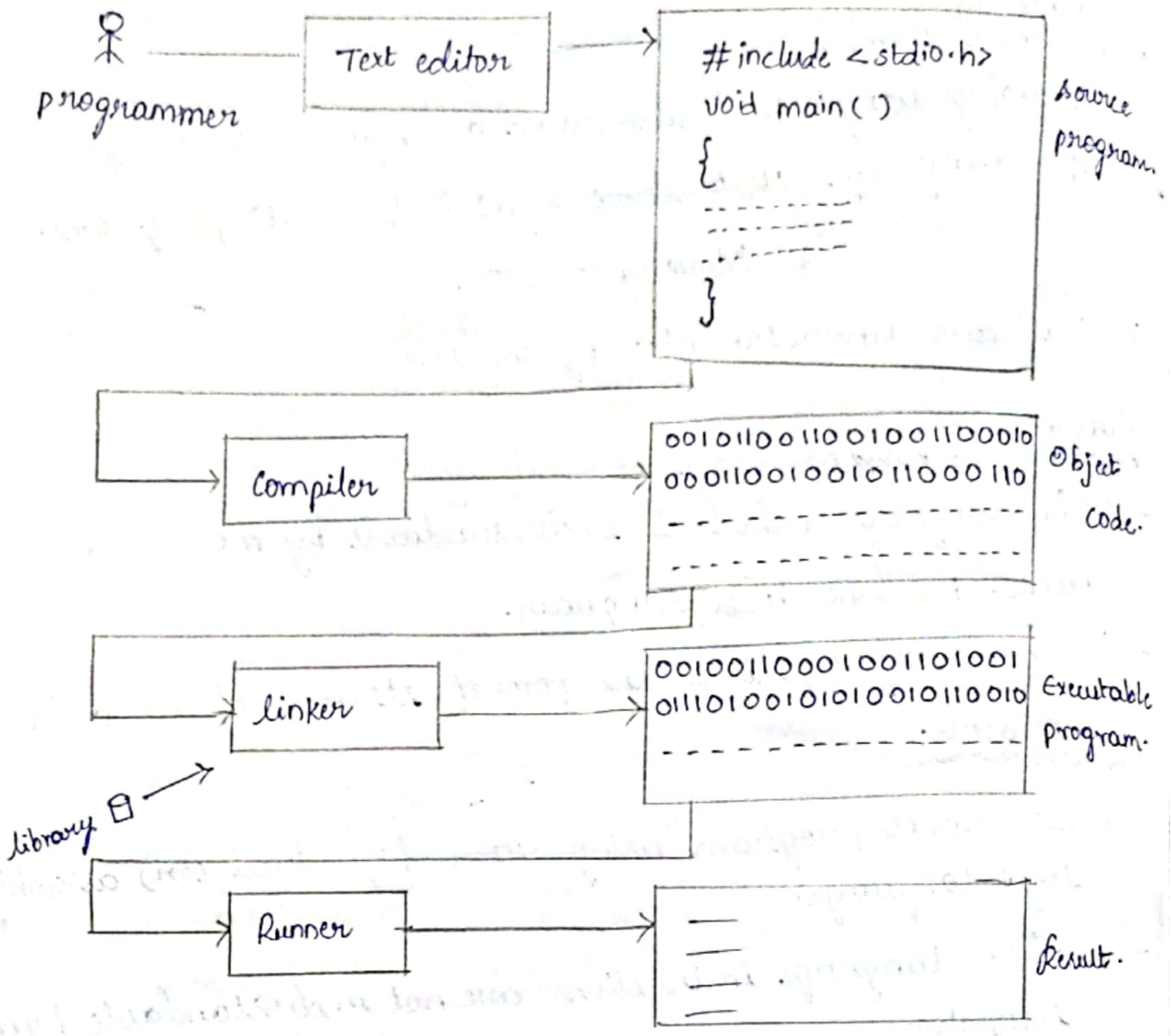
→ The language which is understandable by machine is called machine level language.
→ This language is in the form of streams of 0's and 1's.

Translators:-

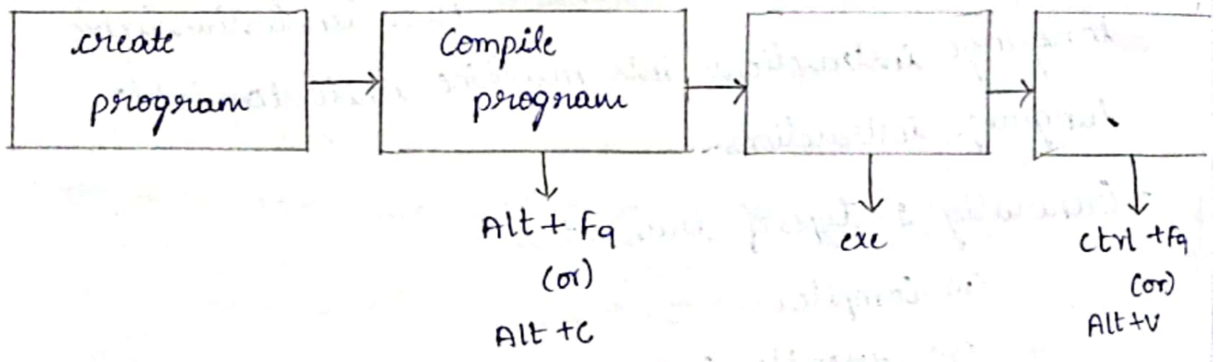
→ user writes program using either high level (or) assembly level language.
→ These language instructions are not understandable by the computer.
→ So we need translator to convert user understandable language instructions into machine understandable language instructions.

- Generally 2-types of translator's
  (1.) Compiler
  (2.) Assembler (interpretor).

Creating and running programs :-



(or)



For output use Alt + F5.

Creating program :-

→ program can be created by using "text editor."

→ we have to enter our program in text editor.

→ After writing program, save that file with extension ".c".

Ex :- some name.c.

Compiling program :-

→ The written program is generally in high level language.

→ That cannot be understandable by the machine (or) computer.

→ So, we need translators like compiler to convert that program in machine language.

→ For compilation we have to use Alt+F9 (or) Alt+C.

→ After compilation compiler shows errors if any exist in the program.

→ If there is no error, it converts the program into machine language, and produces the obj file.

→ Obj stands for object file.

Linking program :-

→ This object program will be linked with library (or) header file by the linker.

→ Then produces "exe" file.

→ "exe" means executable file.

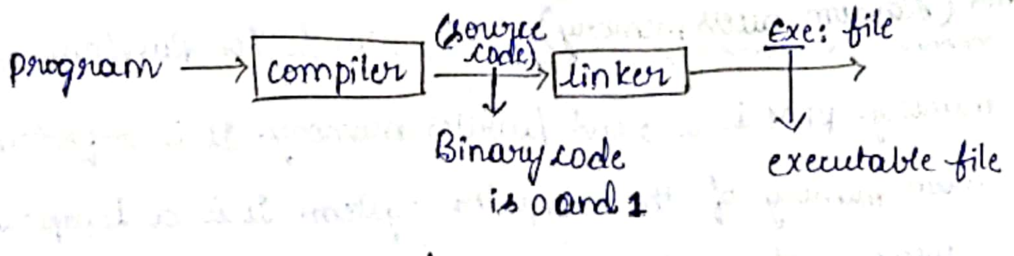
→ We can execute this executable file only.

Executing program :-

→ For execution we use the ctrl+F9 (or) Alt+V.

For Output :-

use Alt+F_5.

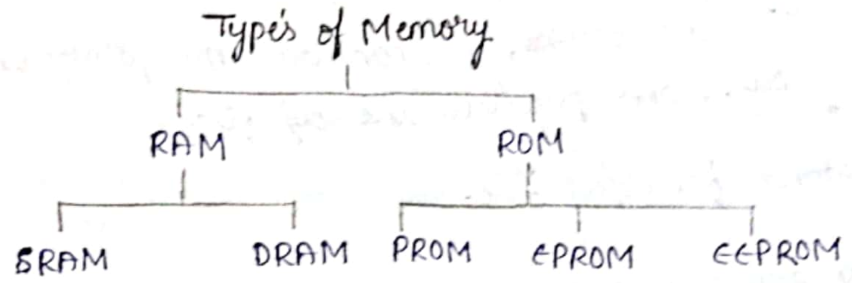


↓

this entire process occurs when you click on $\boxed{\text{Alt+F9}}$

MEMORY:

Memory is the most essential element of a computing system because without it computer cannot perform simple tasks. Computer memory is of two basic ~~parts~~ types. primary memory (RAM and ROM) and secondary memory (hard drive, CD), Random Access memory (RAM) is primary volatile memory and Read only Memory (ROM) is primary non- volatile memory.



Type's of Memory:-

Generally computer system consists of two types of memory; primary memory (or) volatile memory

It is called the internal memory of the computer. And it is also known as main memory or temporary memory. It holds the data and instructions that are presently working on the system or by the CPU. Primary memory is called volatile memory, because when power is switched off it loses all data.

Primary memory is generally of two types;

- RAM
- ROM

RAM (Random access memory):- It stands for Random access memory. RAM is a read /writes memory. It is referred as main memory of the computer system. It is a temporary memory. The information • stored in RAM is lost whenever the power supply to the computer is switched off.

RAM is also of two types which are as follows-

- Static RAM:- Static RAM also known as SRAM. In this RAM the information is stored as long as the power supply is ON. SRAM are of higher coast and consume more power. They have higher speed than Dynamic RAM.

- Dynamic RAM:- Dynamic RAM also known as DRAM. This type of RAM stores information in a very short time basically, a few milliseconds even though the power supply is ON. The Dynamic RAM is cheaper and of moderate speed and also they consume less power.

ROM (read only memory):- It stands for read only memory. ROM is a permanent type of memory. ROM information is not lost when power supply is switched off. The content of ROM is inserted by the computer manufacturer and permanently stored at the time of manufacturing. ROM cannot be overwritten by the computer. It is also called Non-volatile memory.

ROM memory has three types' names which are as following-

- PROM (programmable read only memory):- It is used to write data once and read many. Once a chip has been programmed, the recorded information cannot be changed. It is a non-volatile memory.

- EPROM (Erasable programmable read only memory):-
  EPROM chip can be programmed by erasing the information stored earlier in it.

- EEPROM (Electrically Erasable programmable read only memory):- It is programmed and erased by special electrical waves in milliseconds. A single byte of data or the entire contents of the device can be erased.

Secondary memory (or) Non - volatile memory:-

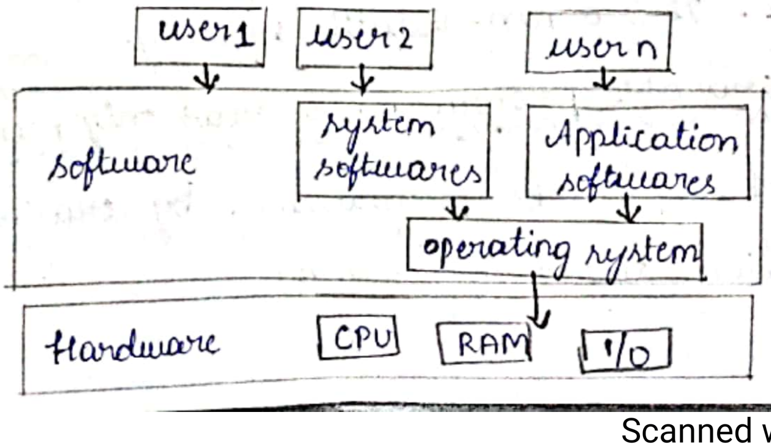
It is an external memory of the computer. It is also known as Auxiliary memory or permanent memory. It is used to store different programs and the information permanently. We call it a non- volatile memory that means the data is stored permanently even if power is switched off.

The secondary storage devices are as follows-

- Floppy disks.
- Magnetic (Hard) disk.
- Magnetic Tapes.
- Pen drive.
- Winchester disk.
- Optical disk (CD, DVD).

Operating system:-

An operating system (OS) is a software that acts as an interface between computer hardware components and the user. Every computer system must have atleast one operating system to run other programs. Applications like Browsers, MS office, Notepad Games, etc., need some environment to run and perform its tasks.



Following are some of important functions of an operating system.

- Memory management
- Processor management
- Device management
- File management
- Security.
- Control over system performance
- Job accounting.

Types of Operating system (OS):-

Following are the popular types of OS (operating system):

- Batch operating system
- Multi tasking / Time sharing OS
- Multi processing OS.
- Real time OS.
- Distributed OS.
- Network OS.
- Mobile OS.

Batch operating system:-

Some computer processes are very lengthy and time-consuming. To speed the same process, a job with a similar type of needs are batched together and run as a group.

The user of a batch operating system never directly interacts with the computer. In this type of OS, every user prepares his or her job on an offline device like a punch card and submit it to the computer operator.

Multi - Tasking / Time - sharing Operating systems :-

Time - sharing operating system enables people located at a different terminal (shell) to use a single computer system at the same time. The processor time (CPU) which is shared among multi users is termed as time sharing.

Real time OS :-

A real time operating system time interval to process and respond to inputs is very small. Examples: Military software systems, space software systems are the real time OS example.

Distributed operating system :-

Distributed systems use many processors located in different machines to provide very fast computation to its users.

Network Operating system :-

Network operating system runs on a server. It provides the capability to serve to manage data, user, groups, security, applications, and other networking functions.

Mobile OS :-

Mobile operating systems are those OS which is especially that are designed to power smartphones, tablets, and wearables devices.

- Some most famous mobile operating systems are Android and iOS.

Numbering system :-

numbering systems are 4-types ;

(i.) Binary number system.
(ii.) Octal number system.
(iii.) Decimal number system.
(iv.) Hexa Decimal number system.

Binary number system :-

→ Bi- means "2" ; nary - means "digits".

→ Binary number system is a number system, which contains 2- digits like "0 and 1".

→ Binary number system is also called as Base 2 number system.

→ Combination of 8 bits called 1-byte.

→ Combination of 4- bits is called 1" nibble.

→ Digital computers understands information that should be in binary format only.

Eg:- $101101_2 \longrightarrow$ Base 2.

Octal number system :-

→ Octal number system is a number system which contains 8-digit from from "0 to 7".

→ Octal number system is also called Base 8 number system.

→ Any number in octal number system should be the combination of 0 to 7 digits only.

Eg:- $735_8 \leftarrow$ Base 8

Decimal number system:-

→ Decimal number system is a number system which contains 10- digits from "0" to "9".

→ It is also called Base-10 number system.

→ Any number in this number system should be the combination of 0 to 9 digits only.

Eg:- 936542_{10} ← Base 10.

Hexa decimal number system:-

→ Hexa means-6, Deci means-10, total contains 16- digits.

→ 10 digits from 0 to 9 and remaining 6 from Alphabets like

A	B	C	D	E	F
10	11	12	13	14	15

→ It is also called Base-16 number system.

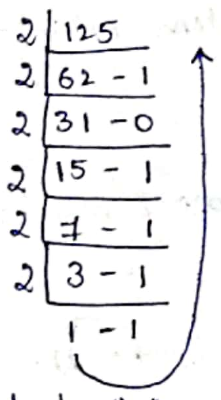
→ Any number should be in the combination of 0 to 9 and A to F digits only.

Eg:- $FA3219_{16}$

Numbering system conversion's:-

Decimal → Binary

125_{10}

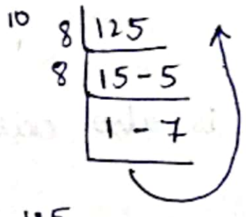


divide by "2".

$\therefore 125_{10} = 1111101_2$

Decimal → Octal

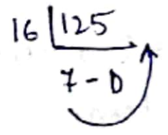
125_{10}



$\therefore 125_{10} = 175_8$

Decimal → Hexa decimal

125_{10}



$\therefore 125_{10} = 7D_{16}$

Binary → Decimal

$10110_{(2)}$

$\frac{2^4}{1}$ $\frac{2^3}{0}$ $\frac{2^2}{1}$ $\frac{2^1}{1}$ $\frac{2^0}{0}$

↓

$= (1\times2^4)+(0\times2^3)+(1\times2^2)+(1\times2^1)+(0\times2^0).$

$= (1\times16) + (0) + (1\times4) + 1(2) + (0\times1)$

$= 16+0+4+2+0$

$= 22_{(10)}$

$\therefore \boxed{10110_2 = 22_{10}}$

Octal → Decimal

~~10110~~

$123_{(8)}$

$\frac{8^2}{1}$ $\frac{8^1}{2}$ $\frac{8^0}{3}$

$= (1\times8^2)+(2\times8^1)+(3\times8^0)$

$= (1\times64)+(2\times8)+(3\times1)$

$= 64+16+3$

$= 83_{(10)}$

$\therefore \boxed{123_8 = 83_{(10)}}$

Hexadecimal → Decimal

$ABC_{(16)}$

$\frac{16^2}{A}$ $\frac{16^1}{B}$ $\frac{16^0}{C}$

$= (10\times16^2)+(11\times16^1)+(12\times16^0)$

$= 2560+176+12$

$= 2748_{10}$

$\therefore \boxed{ABC_{16} = 2748_{10}}$

Octal → binary

$125_{(8)}$ into binary

$8^1 = 2^3$

1 digits in octal = 3 bits in binary.

```
  1    2    5
 /     |     \
001   010   101
```

$= 125_8 = 1010101_2$

Hexa Decimal → Binary

$ABCD_{16}$

$16^1 = 2^4$

1 bit in Hexa decimal = 4 bits in binary.

```
 A      B      C      D
 |      |      |      |
1010   1011   1100   1101
```

A=10 B=11

```
2 | 10        2 | 11
2 | 5 - 0     2 | 5 - 1
2 | 2 - 1     2 | 2 - 1
    1 - 0         1 - 0
```

$\therefore ABCD_{16} = 1010\ 1011\ 1100\ 1101_2$

Hexa Decimal → ~~Octal~~ Binary:

```
 A      B      C      D
 |      |      |      |
1010   1011   1100   1101
```

$\underline{1010}\ \underline{1011}\ \underline{1100}\ \underline{1101}$

125715

$ABCD_{(16)} = 1010\ 1011\ 1100\ 1101_{(2)}$

Algorithms:-

An ~~alog.~~ algorithm is a step by step procedure for solving a problem or to complete a task.

Pseudo code:-

* pseudo code is an informal and artificial language that helps programmers to develop an algorithm.
* Algorithm written in english like language is known as pseudo code.

Example-1:- adding of 2-numbers.

Algorithm name:- addition of 2-numbers.

1. Start
2. Read 2-values
3. Add 2-values
4. store result in another variable.
5. print the variable which contains result.
6. stop.

Example-2:- Average of 3-number.

1. Start
2. Read 3-values assume a,b,c.
3. sum a,b,c
4. Divide sum by 3.
5. store result in another variable like "d."
6. print "d."
7. Stop (or) end program.

Characteristics (or) properties of Algorithm:-

1. Input/output
2. Finiteness
3. Definiteness (or) unambeguous.
4. Effectiveness
5. Generality.

1. Input / Output :- Each algorithm must take 0.1 or more quantities as input should produce atleast one output.

2. Finiteness :- An algorithm should terminate a finite number of steps.

3. Definiteness (or) unambiguous :- Each steps of algorithm must be clear means no ambiguity.

4. Effectiveness :- Algorithm should contain only necessary steps, it should not contains any unnecessary statements.

5. Gen Generality :- Algorithm should run for any type of input and output

Example:-

Additional 2-numbers:-

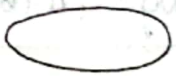
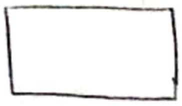
1.— start

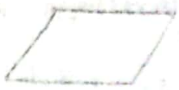
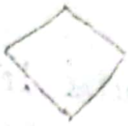
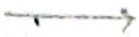
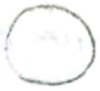
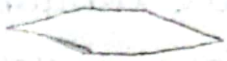
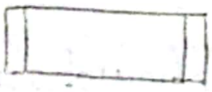
2.— Read 2-values

3.— Add

Q. what is flow chart? It's advantages and disadvantages?

Ans:- Flow chart is the pictorial (or) diagram representation of an algorithm by using different types of boxes and symbols. Flow charts uses the following boxes and symbols.

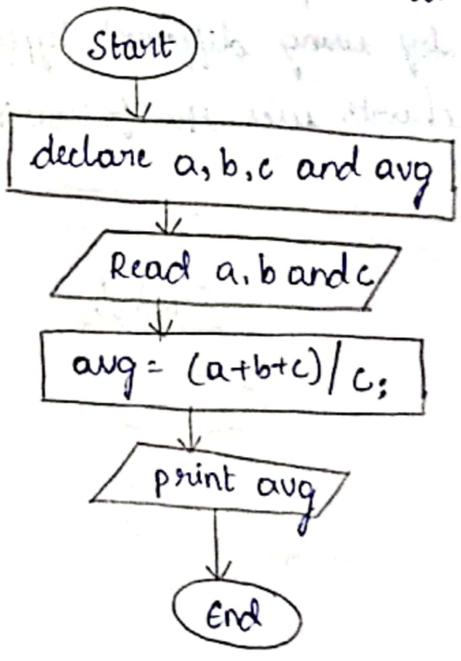
No.	Name	Symbol	usage
1.	Ellipse		start/stop.
2.	Rectangle		Expressions

3.	parallelogram		Input (Read) / output (print).
4.	Rhombus		conditional checking.
5.	Arrow		Flow of solution
6.	Circle		Connector
7.	Elongated Hexagon		Continue
8.	Rectangle with bars		procedure / Function call.

Problem:- add two numbers.

Step-1 : Start
Step-2 : Read A, B
Step-3 : C=A+B
Step-4 : print C
Step-5 : stop.

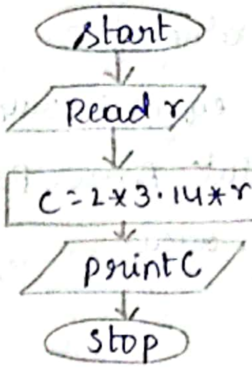
Flow chart for average of three numbers:-



Example 4: Write an algorithm and draw the flowchart to find the circumference of circle

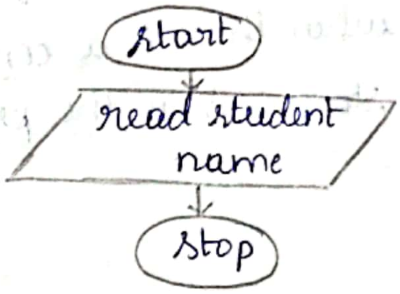
Algorithm

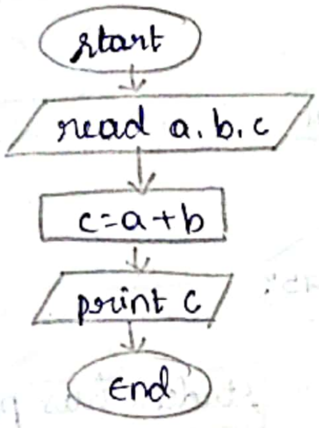
1. Start
2. Read r
3. Calculate C=2*3.14*r
4. Print C
5. Stop.



Example: 3: Algorithm for reading student name

1. Start
2. Read student name
3. End



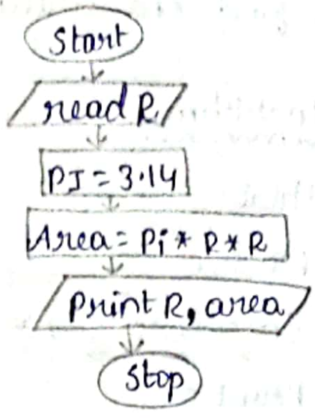


Algorithm for average of three numbers:

Declare a variable a.b.c and avg as int;
Read two numbers a.b and c;
avg = (a+b+c)/3;
~~pring~~ print avg;

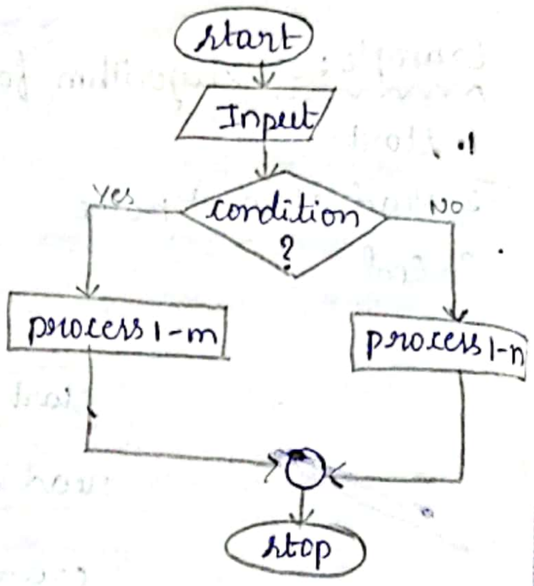
Example :- Algorithm to calculate the area of circle

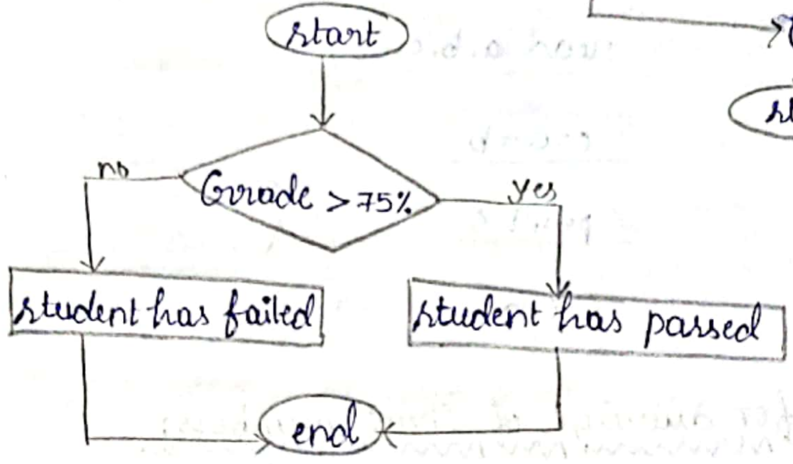
1. Start
2. Read value of R
3. Set PI equal to 3.14
4. Calculate Area = PI*R*R
5. Print R, area
6. Stop



Branched flowchart :-

- It is branched flowchart, when there is a condition statement in the program.





Introduction to problem solving through programming in C :-

A computer is a very powerful machine capable of performing different tasks, but it has no intelligence or thinking power.

The computer performs any tasks exactly in the same manner as it is told to do. This places responsibility on the user to instruct the computer in a correct manner so that the machine is able to perform the required job in a proper way. A wrong o instruction may sometimes prove errors.

The computer cannot ~~it~~ solve the problem on its own, one has to provide step by step solutions of the problem to the computer. In fact, the task of problem-solving is not done by the computer. It is the programmer who has to write down the solution to the problem in terms of simple operations which the computer can understand and execute.

problem-solving is a sequential process of finding information related to a given solution and generating appropriate response options.

In order to solve a problem with the computer, one has to pass through certain stages or steps. They are as follows:

Steps to solve a problem with the computer:-

Step 1: understanding the problem:

Here we try to understand the problem to be solved in totally. Before with go to the next stage or step, we should clearly understand the given problem.

Step-2: Analyzing the problem:-

After understanding thoroughly the problem to be solved, we look at different ways of solving the problem and analyse each of these methods. The idea here is to search for an appropriate solution to the problem.

Step-3: program design (By using algorithm, flowchart and pseudocode):

Here, after analyzing the solution to the problem ~~to be~~ solved, we look at different ways of solving the problem and ans we need to write step by step solution to the problem by using algorithm or flow chart or pseudocode.

Step-4: Coding and implementation:

The last stage of problem-solving is the conversion of the step by step solution into a language that the computer can understand (ex: c-language, JAVA, python). Here, each step is converted to its equivalent instruction or code in the computer language which is done by programmers (or software engineers).

Step-5: Testing:

Here testing engineers will find errors present in our code and then this testing engineers will create error report and send this error report back to programmers, so that programmers will modify errors and then again send code back to testing engineers.

This testing is done until there is no error in our program. After testing our program, it is given to our clients.

Important points to remember:-

Clients:

person who gives project.

(example I want one college software, I give some money to software engineers to make that college software, so here I am considered as client).

Software designers:

person who creates flowchart or algorithm.

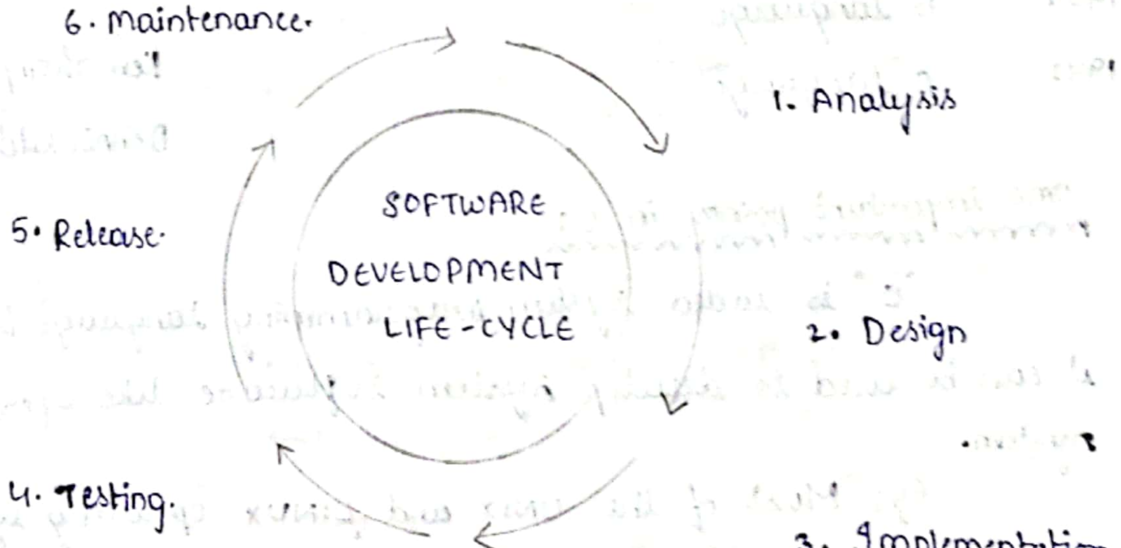
Software engineers / programmers:

person who do coding to our software.

Software tester or testing engineer:

person who perform testing in order to find errors in our software.

Software development life cycle:-



Expressions:-

$c = \underbrace{a+b}$; → operator.
→ operance.

b = a+b* c;

c = b/a;

Introduction to C :-

C is a programming language developed by Dennis Ritchie in the year 1972. at AT and T's Bell laboratories of USA.

AT and T stands for American Telephone and Telegraph Company.

History of C language

1960	ALGOL-60	
1963	CPL (combine programming language)	
1967	BCPL (Basic combine programming language)	Martin Richards
1969	'B' language	Ken Thompson
1972	'C' language	Dennis Ritchie

Some important points in C :

"C" is called system programming language because it can be used to develop system software like operating system.

Eg:- Most of the UNIX and LINUX operating system developed by using C-language only.

→ C is also called procedure oriented programming language because each statements / instruction C-programming language instructs computer machine what to do in a step-by-step manner.

→ C is a case-sensitive language (It treats "a" and "A" differently).

→ C is also called robust language because using C-programming language you can write any complex program, you can develop system software. Every-thing you can do by using C-language. Hence, it is called "Robust" language.

→ "C" follows "Top-down" approach; means Generally user defined function definition written after main () only.

Middle level language :-

→ C-language posses the capability of both low level languages and high level languages. Therefore "C" can be used for writing system software and application software.

Portibility:

→ "C" is a portable language, because the programms written on one machine can be executed on different machine with (or) without minor changes in the program.

How to write first C program:

To write the first C program, open the C console and write the following code.

```
/* title : my first program */
# include <stdio.h>
int main ( )
{
 printf ("Hello C language");
 return 0;
 }
```

#include < stdio.h > stands for standard input output. header file, it is a library functions. the printf() function is defined in stdio.h library. So whenever we want to use printf you need to declare #include < stdio.h >.

int main ( ) The main () function is the entry point of every program is c language. we write our actual program in main () function.

printf ( ) The printf() function is used to generate output, it is used to print data on the console (on output screen).

return 0 The return 0 statement, returns execution status to the OS. The 0 value is used for successful execution and 1 for unsuccessful exeution.

How to compile and run the C program

There are 2 ways to compile and run the c program, by menu and by shortcut.

By menu

- Now click on the compile menu then compile sub menu to compile the C program.
- Then click on the run menu then run sub menu to run the C program.

By shortcut

"or; press ctrl + F9 keys compile and run the program directly.

You will see the following output on user screen.

OUTPUT:

```
Turbo C++ IDE

Hello C Language
```

You can view the user screen any time by pressing the alt + F5 Keys.

Now Press Esc to return to the turbo C++ Console.

Difference between void main and int main in C/C++:-

Sometimes we use int main ( ), or sometimes void main(). Now the question comes into our mind, that what are the differences between these two.

The main ( ) function is like other functions. It also takes arguments, and returns some value. One point we have to keep in mind that the program starts executing from this main ( ) function. The void main ( ) - indicates that the main ( ) function will not return any value, but the int main ( ) indicates that the main ( ) can return integer type data when our program is simple, and it is not going to terminate before reaching the last line of the code, or the code is error free, then we can use the void main ( ).

Int main () program:

```
#include<stdio.h>
int main()
{
printf("Hello ~~to~~ C language");
return 0;
}
```

void main () program:

```
#include<stdio.h>
void main ()
{
printf("Hello ~~to~~ C
                language");
}
```

OUTPUT:- Hello C language

C language: Comments

COMMENTS are optional, just for writing heading or meaning we use comments

→ Comments are two types
 (i.) Single line comment
 (ii.) multiple line comment

Single line comment:-

→ If your documentation contains only one line, then you can use single line comments section.

→ Single line comments section starts with "//".
 Eg:- // This program for adding 2- numbers.

Multiple line Comment section:-

→ If your documentation contains more than one, then you can use multiple comment section.

→ Multiple line comment section starts with "/*" and ends with "*/".

Eg: /* program names : addition of 2 numbers
Author name : vyshu
Date : 1/12/15 */.

Document section is not mandatory section.

Variables in C

A variable is a name of the memory location. variable is used to store data. Its value can be changed, and it can be reused many times.

Syntax to declare a variable. data_type variable_type.

The example of declaring the variable is given below:

int a; // here int is data type and a is variable name

float b=10; // here float is data type, b is variable name and 10 is value.

char c; // here char is data type and c is variable name.

Here, a, b, c ~~ar~~ are variable. The int, float, char are the data types.

We can also provide values while declaring the variables as given below:

1. int a=10, b=20, int c=30; // declaring 2 variable of integatype.
2. float d= 20.8 , e=46.68;
3. char c= 'A';

Rules of for defining variables

- A variable can have alphabets and underscore.
  Example : int a=10; or int_a =10;
- A variable name can start with alphabet, and underscore only. It can't start with a digit.

- No whitespace is allowed within the variable name.
- A variable name must not be any reserved word or keyword, eg. int, float, etc.

Valid variable names:

1. int a; // we can start with an alphabet.
2. int _ab; // we can start with underscore.
3. int a30; // we can give number after alphabet.

Invalid variable names:

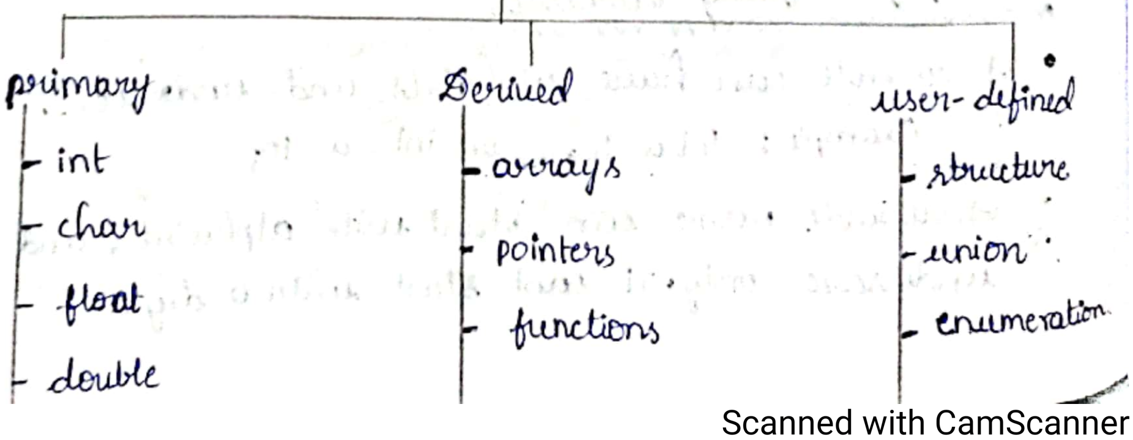
1. int 2; // we must not start with number.
2. int num = 10; // we must not give space between variable name.
3. int float =10; // here float is datatype so we cannot take it as variable name.

Simple program for variable:

```
# include <stdio.h>
void main ()
{
int a=10; //
printf("%d", a);
}
```

OUTPUT: 10.

Data types in c



- Data type specifies what kind of value a variable can store in it.
- Data type determines size of the variable and type of value.

Classifications of data types:

Data types are classified into below types depending on size and type of the value.

- int : An int variable is used to store are integer.

  Syntax: int variable_name = value;

  Example: int a = 10;

- char: It stores a single character and required a single byte of memory in almost all compilers.

  Syntax: char variable_name = value;

  Example : char a = "c";.

- Float : float used to store decimal numbers (numbers with floating point value).

  Syntax: float variable_name = value;

  Example: float a = 10.5 ;

Formatted I/O or format specifiers:

Formatted I/O functions are used to take various inputs from the user and display multiple outputs to the user. These types of I/O functions can help to display the output to the user in different formats using the format specifiers. These I/O supports all data types like int, float, char, and many more. These functions are called formatted I/O functions because we can use format specifiers in these funtions and hence, we can format these functions according to our needs.

Scanned with CamScanner

list of some format specifiers :-

Type	size	Range	Format specifier
char	1 byte	-128 to +127	%c
unsigned char	1 byte	0 to 225	%c
int	2 byte	-32768 to +32767	%d
unsigned int	2 bytes	0 to 65535	%u
long int	4 bytes	-2147483648 to -2147483647	%ld
unsigned long	4 bytes	0 to 4294967295	%lu
float	4 bytes	$\pm 3.4 * 10^{\pm 38}$	%f
double	8 bytes	$\pm 1.7 * 10^{\pm 308}$	%lf
long double	10 bytes	$\pm 3.4 * 10^{\pm 4932}$	%Lf

The following formatted I/O functions will be discussed in this section-

1. printf ()
2. Scanf ()

printf ( ):

printf ( ) function is used in a C program to display any value like float, integer, character, string, etc on the console screen. It is a pre-defined function that is already declared in the stdio.h (header file).

Syntax 1 :

printf ("format specifier", var1, var2, ... varn);

Example: printf ("%d", a);

Syntax 2:

printf ("Enter the text which you want to display");

Scanf ():

Scanf() function is used in the C program for reading or taking any value from the keyboard by user, these values can be of any data type like integer, float, character, string, and ~~an~~ many more. This function is declared in stdio.h (header file), that why it is also a pre-defined function. In scanf () function we use ~~an~~ & (address-of operator) which is used to store the variable value on the memory location of that variable.

Syntax:

Scanf ("Format specifier", &var1, &var2, ---- &varn);

Example: Scanf ("%d", &num1);

Difference between variable initialization and Declaration:-

Example:

int a; // this is variable declaration

Example;

int a=10; // this is variable initialization

Integer program

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

format specifier.

OUTPUT: 10.

character program

```
#include <stdio.h>
void main()
{
char a='h';
printf ("%c", a);
}
```

OUTPUT: h.

float program

```
#include<stdio.h>
void main()
{
float =10.20;
print f ("%f", a);
}
```

OUTPUT: 10.20

```
# include<stdio.h>
void main()
int a;
printf ("Enter number
                 \n");
Scanf ("%d", &a);
{printf ("\n your
          no is %d", a);
}
```

OUTPUT: Enter number
10
your number is 10

```
#include<stdio.h>
void main()
{
float a;
printf ("Enter number\n");
Scanf ("%f", &a);
printf ("\n your no. is
  %f", a);
}
```

OUTPUT: Enter number
10.20
your number is 10.20

```
#include<stdio.h>
void main()
{
char a;
printf ("Enter value
            \n");
Scanf ("%c", &a);
printf ("\n your
   value is %c", a);
}
```

OUTPUT: Enter value
c
your value is c.

Some more examples:-

```
# include <stdio.h>
void main()
{
int a=1;
float b=10.5;
char c='c';
print f ("%d \n% f \n%c", a, b, c);
}
```

OUTPUT:

```
1
10.5
c
```

```
#include <stdio.h>
void main()
{
int a=10, b=20, c=30;
printf ("%d \n %d \n %d\n",a,b,c);
}
```

OUTPUT:-

```
10
20
30
```

```
#include <stdio.h>
void main()
{
float a;
printf ("Enter English marks ");
Scanf ("%f", &a);
printf ("\n your got %f",a);
float b;
printf ("\n Enter social marks");
Scanf ("%f", &b);
printf ("\n you got %f", b);
}
```

OUTPUT:-

```
Enter English marks 10.25
you got 10.250000.
Enter social marks 20.6
you got 20.600000.
```

Delimiters and Escape sequences:-

Delimiters:-

→ They limit the boundary between the basic elements of a program

→ Delimitors are also called as "Seperator's".

Delimeters :-	Uses
: - colon	useful for label.
; - semi-colon	Terminates statement
( ) - parenthesis	used in expression and function
[ ] - square bracket	used for array declaration
{ } - curly brace	scope of statement
# - hash	pre-processor directive
, - comma	variable - separator.

Escape sequences:-

Escape sequence:-	meaning
"\0"	Null (end of string)
"\t"	Horizontal tab
"\n"	New line character
"\r"	carriage return
"\" "	Double quote
"\f"	Form feed
"\b"	Back space
"\a"	system alarm.

C Expressions

An expression is a formula in which operands are linked to each other by the use of operators to compute a value. An operand can be a function reference, a variable, an array element or a constant.

Let's see an example:

a-b;

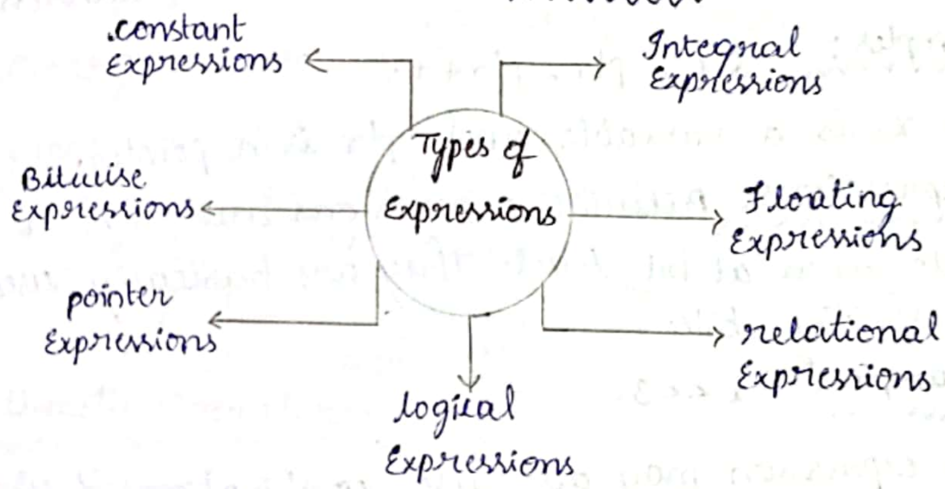
In the above expression, minus character (-) is an operator, and a, and b are the two operands.

Each type of expression takes certain types of operands and uses a specific set of operators. Evaluation of a particular expression produces a specific value.

For example: $x = 9/2 + a - b$;

The entire above line is a statement, not an expression. The portion after the equal is an expression.

Types of Expressions



Types of expressions:-

Expressions may be of the following types:

- Constant expressions: Constant expressions consists of only constant values. A constant value is one that doesn't change.

  Examples:

  x=5, where 5 is constant.

- Integral expressions:- Integral expressions are those which produce integer results.

  Examples: x * y.

  where x and y are integer variables.

- Floating expressions: Float expressions are which produce floating point results.

  Examples: 10.75.

  where x and y are floating point variables.

- Relational expressions: they will be evaluated first and then the results compared. Relational expressions are also known as Boolean expressions.

  ~~x>y && x==10~~ , ~~x==10 || y==5~~.

  Examples: x <= y , x+y>2.

- Logical expressions: logical expressions combine two or more relational expressions.

  Examples: x>y && x==10 , x==10 || y==5.

- pointer expressions: pointer expressions produce address values.

  Examples: &x , ptr, ptr++.

  where x is a variable and ptr is a pointer.

- Bitwise expressions: Bitwise expressions are used to manipulate data at bit level. They are basically used for testing or shifting bits.

  Examples: x << 3.

NOTE: An expression may also use combinations of the above expressions. Such expressions are known as "compound expressions."

C-Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators –

- Arithmetic Operators.
- Relational Operators.
- Logical Operators.
- Bitwise Operators.
- Assignment Operators.
- Misc Operators.

Operators in C language.

Operation type	Operator's type	Operators
Unary operator	Increment/ Decrement operators	++ , --
Binary operator	Arithmetic operators	+, -, *, /, %, ++, --
Binary operator	Relational operators	==, !=, <, >, <=, >=
Binary operator	logical operators	&&, ||, !
Binary operator	Bitwise operators	&, |, ^, ~, <<, >>
Binary operator	special operators	, & sizeof()
Binary operator	Assignment operators	=, +=, -=, *=, /=, %=
Ternary operators	Ternary or conditional operators.	?:

Arithmetic Operators:

The following table shows all the arithmetic operators supported by the C language. Assume variable A holds 10 and variable B holds 20 then-

show Examples

operator	Description	Example
+	Adds two operands.	A+B = 30.
-	subtracts second operand from the first	A- B= -10.

*	multiplies both operands.	A* B=200
/	Divides numerator by de-~~nomerator~~ denomerator.	B/A =2
%	Modulus Operator and remainder of after an integer division.	B%A =0.
++	Increment operator increases the integer value by one.	A++ =11.
--	Decrement operator decreases the the integer value by one.	A-- =9

Relational Operators :-

The following table shows all the relational operators supported by C. Assume variable A holds 10 and variable B holds 20 then -

Show Examples

operator	Description	Example
==	checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A==B) is not true.
!=	checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A!=B) is true
>	checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A>B) is not true.
<	checks if the value of left oper-and is less than the value of right operant. If yes, then the condition becomes true.	(A<B) is true.

>=	checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A>=B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A<=B) is true.

logical operators:-

Following table shows all the logical operators supported by C language. Assume variable A holds 1 and variable B holds 0, then -

Show Examples

Operator	Description	Example
&&	Called logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A&&B) is false.
||	Called logical OR operator. If any of the two operands is non-zero, then the condition becomes true.	(A||B) is true.
!	Called logical NOT operator. It is used to reverse the logical state of its operand. If a condition is true, then logical NOT operator will make it false.	!(A&&B) is true

Bitwise Operators:

Bitwise Operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows -

p	q	p&q	p|q	p^q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume A = 60 and B = 13 in binary format, they will be as follows -

A = 0011 1100
B = 0000 1101

A&B = 0000 1100
A|B = 0011 1101
A^B = 0011 0001
~A = 1100 0011

The following table lists the bitwise operators supported by C. Assume variable "A" holds 60 and variable "B" holds 13, then -

Show Examples

operator	~~Desp~~ Description	Example
&	Binary AND operator copies a bit to the result if it exists in both operands.	(A&B) = 12, i.e., 0000 1100
|	Binary OR operator copies a bit if it exists in either operand.	(A|B) = 61, i.e., 0011 1101
^	Binary XOR operator copies the bit if it set in one operand but not both.	(A^B) = 49, i.e., 0011 0001
~	Binary One's complement operator is unary and has the effect of "flipping" bits.	(~A) = ~(60) i.e., - 0111101

<<	Binary left shift operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary right shift operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

Assignment Operators:-

The following table lists the assignment operators supported by the C language-

Show examples

operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C=A+B will assign value of A+B to C.
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C+=A is equivalent to C=C+A.
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C-=A is equivalent to C=C-A.
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C*=A is equivalent to C=C*A.
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C/=A is equivalent to C=C/A.

%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C%=A is equivalent to C=C%A
<<=	Left shift AND assignment operator.	C<<=2 is same as C=C <<2
>>=	Right shift AND assignment operator.	C>>=2 is same as C=C>
&=	Bitwise AND assignment operator	C&=2 is same as C=C&2.
^=	Bitwise ~~A~~ exclusive OR and assignment operator.	C^=2 is same as C=C^2
|=	Bitwise inclusive OR and assignment operator.	C|=2 is same as C=C|2.

Misc operators ⟼ sizeof & ternary OPERATOR

Besides the operators discussed above, there are a few other important operators including sizeof and ?: supported by the C language.

Show Examples

Operator	Description	Example
Sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4
&	Returns the address of a variable.	&a; returns actual address of variable.
*	Pointer to a variable	*a;
?:	Conditional Expression.	If condition is true ? then value X : otherwise value Y.

Operators precedence in c

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

For example $x = 7+3*2$; here, x is assigned 13, not 20 because operator * has 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 ~~a~~ table, those with the lowest appear at the bottom. Within an expression, highest 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

Arithmetic Operator - Example program

Example.

```
# include <stdio.h>
void main ()
{
int a=21;
int b=10;
int c;
c=a+b;
printf ("addition is %d \n", c);
c=a-b;
printf ("sub of c is %d \n", c);
c=a*b;
printf ("mul of c is %d \n",c);
c=a/b;
printf ("div ~~mod~~ of c is %d \n",c);

c=a%b;
printf ("mod of c is %d\n",c);
 c=a++;     21++  22--
printf ("Increment of c is %d \n",c);
c=a--;   --
 printf (" decrement of c is %d\n",c);
 }
```

OUTPUT:

```
addition of c is 31
sub of c is 11
mul of c is 210
div of c is 2.
mod of c is 1
increment of c is 21
decrement of c is 22
```

Bitwise Operator- Example programs

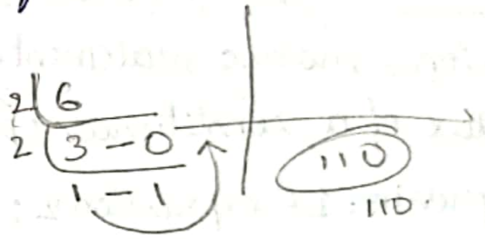
And operator example-

Let's understand the bitwise AND operator through the program.

```
#include <stdio.h>
int main ()
{
int a=6 , b=14 // variable declarations
printf ("The output of the Bitwise AND operator a&b is
        %d", a&b);
return 0;
}
```

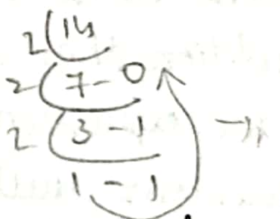
In the above code, we have created two variables, i.e., 'a' and 'b'. The values of 'a' and 'b' are 6 and 14 respectively. The binary value of 'a' and 'b' are 0110 and 1110 respectively. When we apply the AND operator between these two variables,

a AND b = 0110 && 1110 = 0110



OUTPUT:

```
The output of the Bitwise AND operator a&b is 6
.... program finished with exit code 0
press ENTER to exit console. □
```



OR operator Example-

Let's understand the bitwise OR Operator through a program.

```
#include <stdio.h>
int main ()
{
int a=23, b=10; // variable declarations
printf ("The output of the Bitwise OR operator a|b is %d",
        a|b);
return 0;
}
```

OUTPUT:

```
The output of the Bitwise OR operator a|b is 31
.... program finished with exit code 0
press ENTER to exit console.□
```

Ternary Operator (or) Conditional Operator in C

The conditional operator is also known as ternary operator. The conditional statements are the decision-making statements which depends upon the output of the expression. It is represented by two symbols, i.e., '?' and ':'.

As conditional operator works on three operands, so it is also known as the ternary operator.

The behavior of the conditional operator is similar to the "if-else" statement as if-else statement is also a decision-making statement.

Syntax of a conditional operator

```
Expression 1? Expression 2 : expression 3;
```

Meaning of the above syntax

- In the above syntax, the expression 1 is a Boolean condition that can be either true or false value.
- If the expression 1 results into a true value, then the expression2 will execute.
- The expression 2 is said to be true only when it returns a non-zero value.
- If the expression 1 returns false value then the expression 3 will execute.
- The expression 3 is said to be false only when it returns zero value.

let's understand the ternary or conditional operator through an example.

```
# include <stdio.h>
int main ()
{
int age ; // variable declaration
printf ("Enter your age ");
scanf ("%d",& age); // taking user input for age variable
(age >= 18) ? (printf ("eligible for voting ")) : (printf ("not
 eligible for voting")); // conditional operator.
 return 0;
 }
```

In the above code, we are taking input as the "age" of the user. After taking input, we have applied the condition by using a conditional operator. In this condition, we are checking the age of the user. If the age of the user is greater than or equal to 18, then the statement 1 will execute, i.e., (printf ("eligible for voting")) otherwise, statement 2 will execute, i.e., (printf ("not eligible for voting")).

let's observe the output of the above program.

- If we provide the age of user below 18, then the output would be:

```
Enter your age 12
not eligible for voting.
.... program finished with exit code 0
press enter to exit console
```

- If we provide the age of user above 18, then the output would be:

```
Enter your age 24
eligible for voting.
.... program finished with exit code 0
press ENTER to exit console.
```

Increment :-

→ Increment operator always increments the value of a variable by one.

→ Increment operator can be represented by the symbol "++"

→ You can apply increment operator after the variable name (or) before the variable name.

→ If you apply after the variable name that is called post-increment and if it is before variable it is called as pre-increment.

Decrement operator :-

→ Decrement operator always decrease the value of a Variable by "one".

→ "− −" can be used to represent decrement operator.

→ You can apply decrement operator before and also after the variable name.

→ If it is applied before then it is called pre-decrement and if it is after then it is called as post-decrement.

Operator's	Meaning
++a	pre-increment
a++	post-increment
--a	pre-decrement
a--	post-decrement

```
// post decrement and post increment program
#include <stdio.h>
void main ()
{
int a=10;
printf(" post decrem decrement:\n ");
printf ("%d \n", a--); // post decrement
printf ("%d\n", a);
printf ("post increment : \n ");
printf ("%d\n", a++); //post increment
printf ("%d\n", a);
}
```

OUTPUT:

```
post decrement :
10
9
post increment :
9
10
```

```
// pre decrement and pre increment program
#include <stdio.h>
void main ()
{
int a=10;
printf ("pre decrement :\n");
printf ("% d\n", --a); // pre decrement
printf ("%d\n", a);
printf ("pre increment : \n");
printf ("%d\n", ++a); //pre increment
printf ("% d\n", a);
}
```

OUTPUT:

```
pre decrement :
9
9
pre increment :
10
10
```

main function:-

→ main() function is the entry point of any C program.
→ It is the point at which execution of program is started
→ when a C program is executed, the execution control goes directly to the main() function. Every C program have a main() function.

Syntax:
```
void main ()
{
 -----
 -----
}
```

void: it is a keyword in c language, void means nothing, whenever we use void as a function return type then the function return nothing. here main () function not return any value

→ In place of void we can also use int return type of main () function, at that time main () return integer type of value.

→ main is a function which is predefined in C library.

Example:
```
# include <stdio.h>
# include <conio.h>
void main()
{
printf ("this is main function");
}
```

OUTPUT:-
```
  This is main function.
```

Type conversion (or) Type casting:-

→ Type conversion also called "type casting."
→ It is the process of converting one data type value into another data type value.

In C type, conversions are done in 2-ways.

1. Implict type conversion.
2. Explict type conversion.

Implict type conversion:-

→ The type conversion which is done by the system itself (automatically) is called "Implict type conversion".

Example:-

```
#include <stdio.h>
void main
{
int a=10/3 ;
printf("%d",a);
}
```

OUTPUT:- 3.

Explict type conversion:-

→ The type of conversion which is done by the user (manually) is called as explicts type conversion".

Example:-

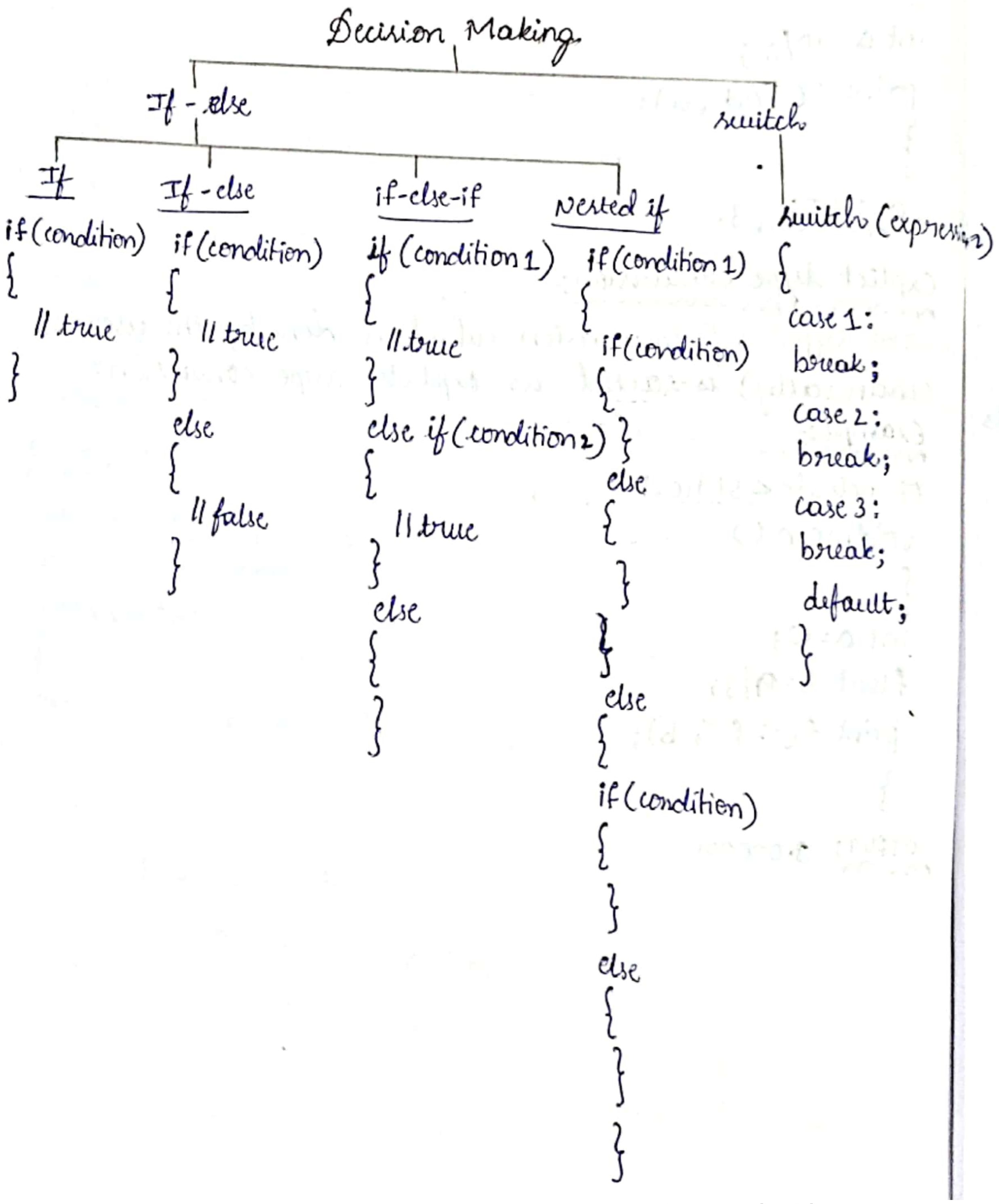
```
#include<stdio.h>
voidmain()
{
int a=10;
float b=9/3;
printf(%f ", b);
}
```

OUTPUT:- 3.000000

Decision Making or Conditional statements in C

(if, if.. else, nested if, if-else-if, switch)

There come situations in real life when we need to make some decisions and based on these decisions, we decide what should we do next. Similar situations arise in programming also where we need to make some decisions and based on these decisions we will execute the next block of code.



If

```
if(condition)
{
    // true
}
```

If - else

```
if(condition)
{
    // true
}
else
{
    // false
}
```

if-else-if

```
if (condition 1)
{
    // true
}
else if(condition 2)
{
    // true
}
else
{
}
```

Nested if

```
if(condition 1)
{
 if(condition)
 {
 }
 else
 {
 }
}
else
{
 if(condition)
 {
 }
 else
 {
 }
}
```

switch

```
switch (expression)
{
 case 1:
 break;
 case 2:
 break;
 case 3:
 break;
 default;
}
```

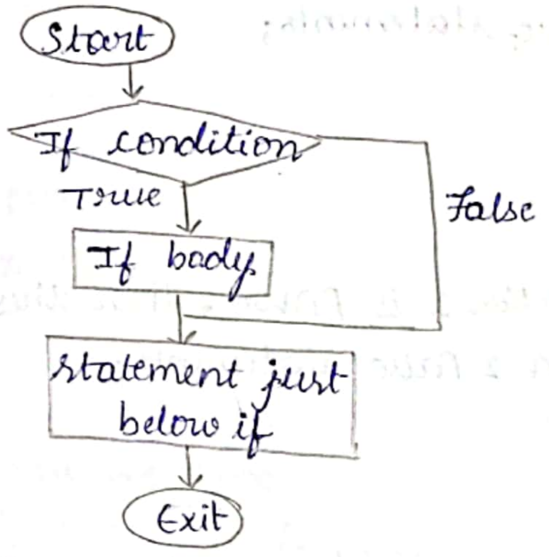
(1.) If statement:

If statement is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.

Syntax:

```
if (condition)
{
 // statements to execute if
 // condition is true
}
```

Flowchart:



example:

```
#include <stdio.h>
int main()
{
 Int i=16;
 if (i>15)
  {
   printf ("value is greater than 15");
  }
 Return 0;
}
```

for clear screen →

```
# include <stdio.h>
int main()
{
int i=16;
if (i>15);
{
clrscr();
printf ("value         ");
}
return 0;
}
```

OUTPUT:-

```
value is greater than 15.
```

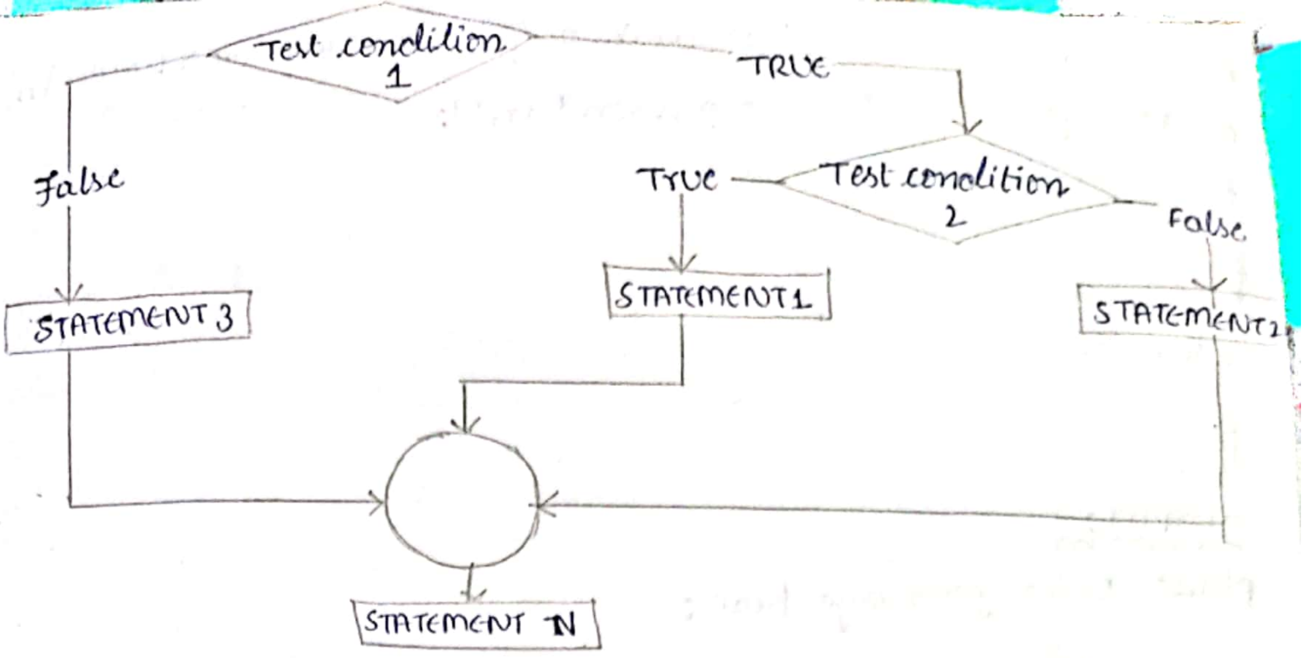
nested-if :

Nested if statements mean an if statement inside another if statement. Yes, both C and C++ allow us to nested if statements within if statements, i.e, we can place an if statement inside another if statement.

Nested if Syntax

```
if (test condition 1)
{
  // If the test condition 1 is TRUE then these it will check
for test condition 2
  if (test condition 2)
  {
  // If the test condition 2 is TRUE, these statements execute
  Test condition 2 True statements;
  }
  else
  {
  // If the c test condition 2 is FALSE, then these statements
  execute Test condition 2 False statements;
  }
  else
  {
  // If the test condition 1 is FALSE then these statements
  will be executed
  Test condition 1 false statements;
  }
```

Flow chart for Nested if :



Example:

```
#include <stdio.h>
int main( )
{
int age;
printf("please Enter your Age Here:\n");
scanf ("%d", &age);
if (age<18)
{
printf("You are Minor. \n");
printf ("Not Eligible to work");
}
else
{
if (age>= 18 && age<=60)
{
 printf (" You are Eligible to work \n");
 printf ("please fill in your details and apply\n");
}
else -> if (age>60)
{
```

```
printf ("You are too old to work as per the Government rules\n
printf ("please collect your pension ! \n");
}
}
return0;
}
```

OUTPUT:-

```
please enter your age here :
27
You are eligible to work
Please fill in your details and apply
```

(2.) if - else statement:

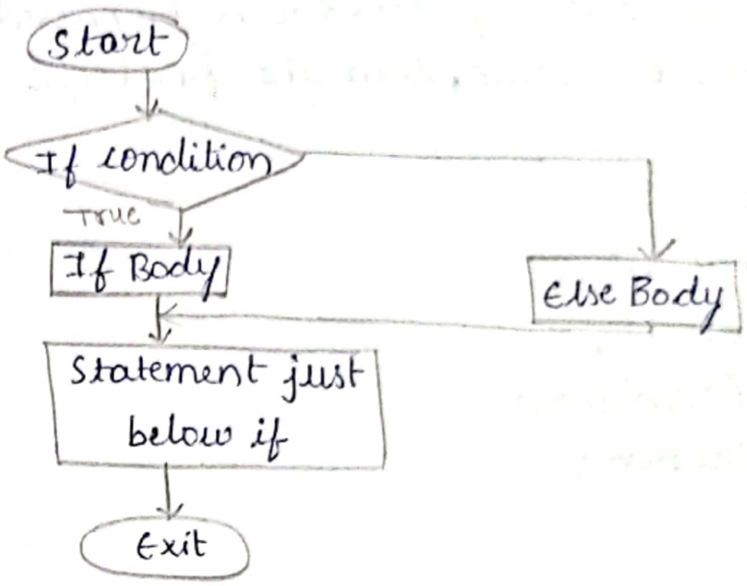
The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But what if we want to do something else if the condition is false. Here comes the C else statement. We can use the else statement with if statement to execute a block of code when the condition is false.

The if statement executes a statement if a specified condition is true. If the condition is false, another statement can be executed.

Syntax:

```
if (condition)
{
  // executes this block if
  // condition is true
}
else
{
    // executes this block if
    // condition is false
}
```

Flowchart:-



Example:-

```
#include <stdio.h>
int main()
{
int a=20; → clrscr
if (a<18)
{
printf("Good day.");
}
else
{
printf("Good evening.");
}
          → return 0;
}
```

OUTPUT:

```
"Good evening."
```

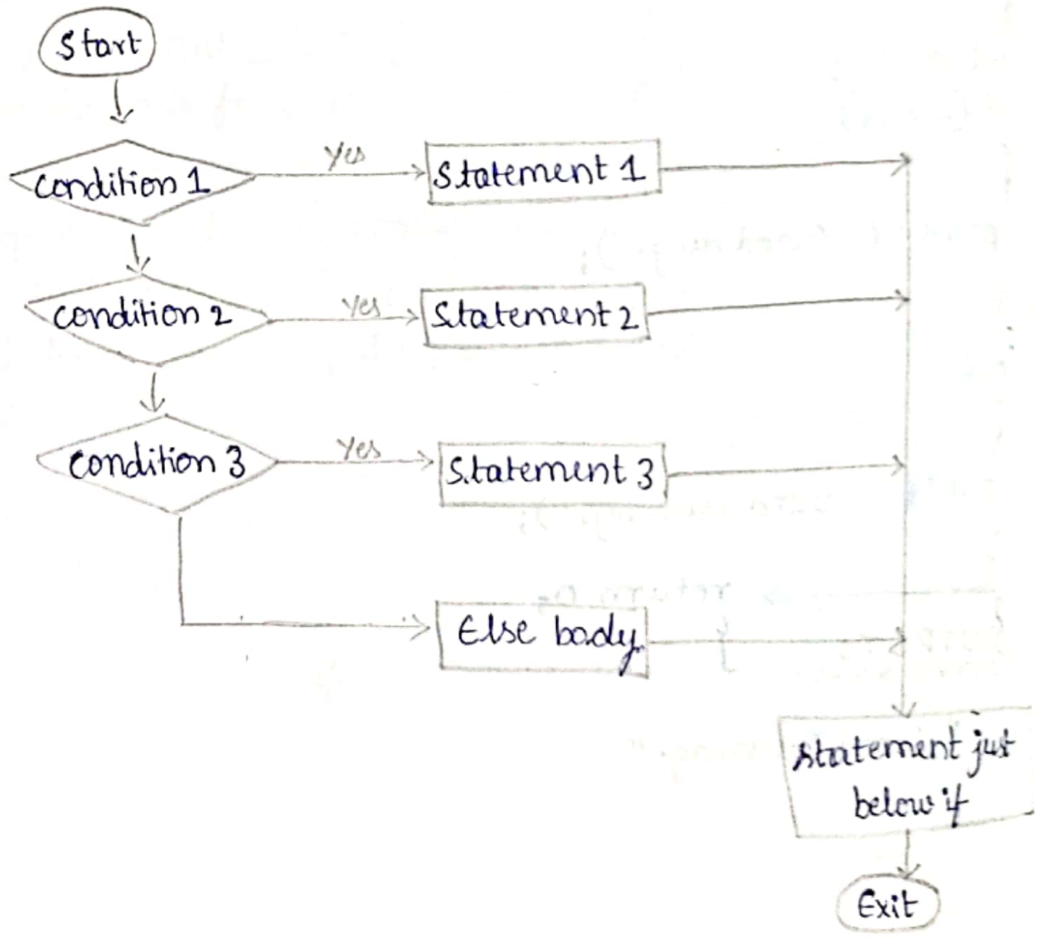
(3.) Elese-if statement in C:

Here, a user can decide among multiple options. The C if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the C else-if ladder is bypassed. If none of the conditions are true, then the final else statement will be executed.

Syntax:

```
if (condition)
     Statement;
 else if (condition)
      statement;
 :
 else
    statement;
```

Flowchart:-



Example:

```
#include <stdio.h>
int main ()
{
 int a=20;  → clrscr ();
 if (a = =10)
{
 printf ("a is 10");
}
 else if (a = =15)
{
 printf ("a is 15");
}
 else if (a = =20)
{
 printf ("a is 20 ");
}
 else
{
 printf ("a is not present");
}
 Return 0;
}
```

OUTPUT :- a is 20.

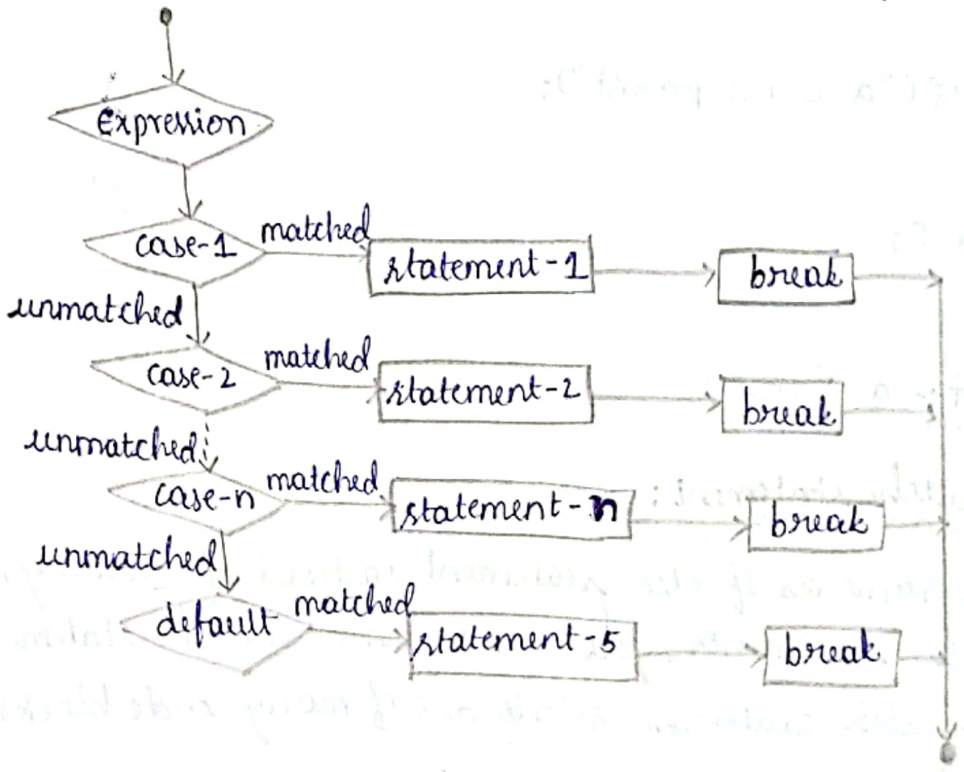
(4.) Switch statement:

It is same as if else statement instead of writing many if.. else statements, you can use the switch statement. The switch statement selects one of many code blocks to be executed.

Syntax:

```
Switch (expression)
 {
 case x :
  // code block
   break;
  case y:
   // code block
  break;
  .
  .
  .
 default:
  // code block
 }
```

Flowchart of switch statement in C



Example:-

```
#include<stdio.h>
int main ()
{
int a=4;
switch (a)
{
case 1:
printf ("Monday");
break;
case 2:
printf ("Tuesday");
break;
case 3;
printf ("wednesday);
break
case 4:
printf (" thursday);
break
case 5:
printf ("Friday");
break.
case 6;
printf ("Saturday");
break
case 7:
printf ("Sunday");
break;
}
return 0;
}
```

OUTPUT:-

```
"Thursday"
```

Loops in C

In programming, sometimes there is a need to perform some operation more than once or n number of times. Loops come into use when we need to repeatedly execute a block of statements.

In loop, the statement needs to be written only once and the loop will be executed n times. In computer programming, a loop is a sequence of instructions that is repeated, until a certain condition is reached.

There are mainly two types of loops:-

1. Entry controlled loops :- In this type of loop, the test condition is tested before entering the loop body. For loop and while loop is entry- controlled loops.
2. Exit controlled loops :- In this type of loop the test condition is tested or evaluated at the end of the loop body. Therefore, the loop body will execute at least once, irrespective of whether the test condition is true or false. the do- while loop is exit controlled loop.

Q.) what is while loop? Discuss Syntax, flow chart and program.

Ans:- while loop:- while loop is a pre-test loop, it first test a specified conditional expression and as long as the conditional expression is true, loop body statements will be executed.

Syntax:- while (condition)
```
{
    ===   } statement.
    ===
    ===
  (inc/dec)
}
```

① Example program:- (Increasing)

```
// while loop program
# include < stdio.h>
void main ()
{
int a=1;
while (a<=10)
{
printf ("%d\n", a);
a++;
}
}
```

OUTPUT:-

```
1
2
3
4
5
6
7
8
9
10
```

② Example program:- (decreasing).

```
// decreasing by using while loop program
#include <stdio.h>
Void main ()
{
int a=99
while (a>12&& a<100)
{
printf ("%d \n",a);
a--;
}
}
```

OUTPUT:-

```
99
98
97
96
.
.
.
.
.
.
.
6
5
4
3
2
```

③Example program:-

```
// printing statement by using while loop program
#include <stdio.h>
 void main()
 {
 int a=0;
 while (a<10)
 {
 printf("how are you?\n");
 a++;
 }
 }
```

OUTPUT:-

```
how are you?
how are you?
how are you?
how are you?
how are you?
how are you?
how are you?
how are you?
how are you?
how are you?
```

Q:) Do while loop:-

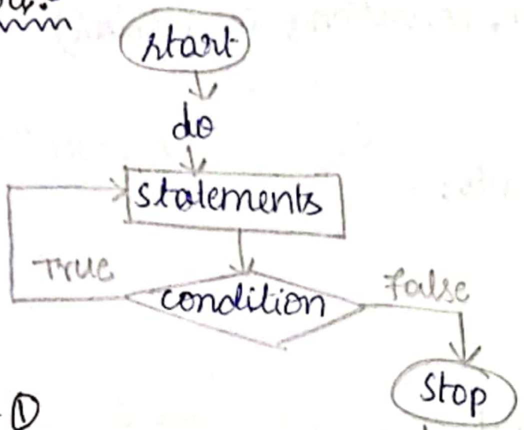
what is do while loop? Discuss syntax, flowchart and example....

Ans:- Do while loop is a post-test loop. It is similar to while loop except it executes its body at least once. weather the condition is true or false. the do-while loop terminates the text expression is evaluated to be zero.

Syntax :- do

```
do
{
        program statements;

}
while(condition);
```

Flowchart:-



Example-①

```
// do while loop program
#include <stdio.h>
void main ()
{
int a=1;
do{
    printf ("%d\n", a);
    a++;
} while (a<10);
}
```

OUTPUT:-

```
1
2
3
4
5
6
7
8
9
```

Example-②

```
// do while loop program
#include <stdio.h>
void main()
{
int a=95;
do{
printf ("%d\n",a);
a--;
} while (a>80 && a<99);
}
```

OUTPUT:-

```
95
94
93
92
91
90
89
88
87
86
85
84
83
82
81
```

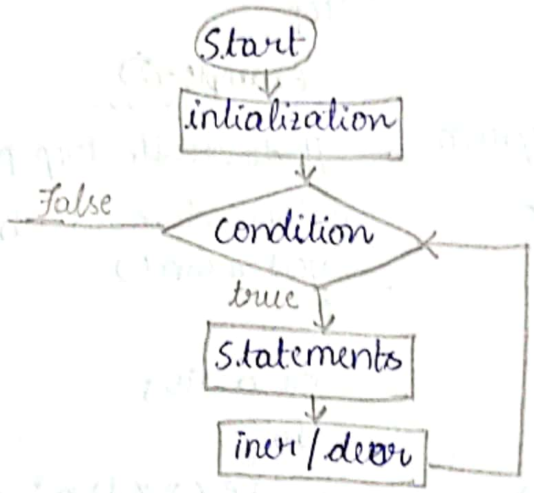
Q.) what is for loop? Discuss syntax, flowchart and Examples?

Ans: For loop execute the statements of program several times repeatedly until a given condition returns false.

Syntax:-

```
for (initialization; condition; incr/decr)
{
    ═══ } statements;
}
```

Flowchart:-



① **Example:-**

```
// For loop
#include <stdio.h>
void main()
{
for (int i=0; i<10; i++)
{
 printf("%d\n", i);
 }
 }
```

OUTPUT:-

```
0
1
2
3
4
5
6
7
8
9
```

Scanned with CamScanner

② Example :-

```
// for loop
# include <stdio.h>
void main ()
{
for (int i=0 ; i<10; i++)
{
 print ("how r u? \n");
}
}
```

OUTPUT :-

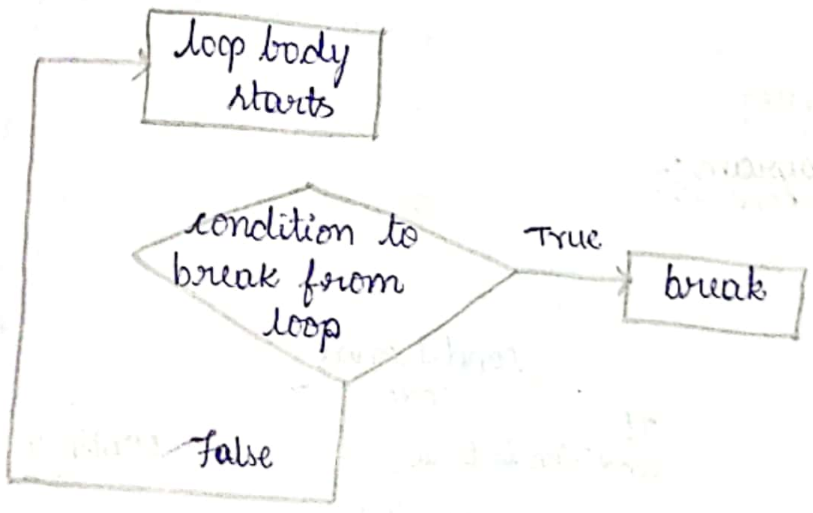
```
how r u ?
how r u ?
how r u ?
how r u ?
how r u ?
how r u ?
how r u ?
how r u ?
how r u ?
how r u ?
```

Break and continue and goto statement in C

Break statement: The break in C or C++ is a loop control statement which is used to terminate the loop. As soon as the break statement is encountered from within a loop, the loop iterations stops there and control returns from the loop immediately to the first statement after the loop.

Syntax :
break;

Flowchart:



Scanned with CamScanner

Example:

```
# include <stdio.h>
{
int main ()
{
for (int a=1; a<10; a++)
{
printf("%d", a);
if (a==4)
break;
}
return 0;
}
```

OUTPUT:-

```
1234
```

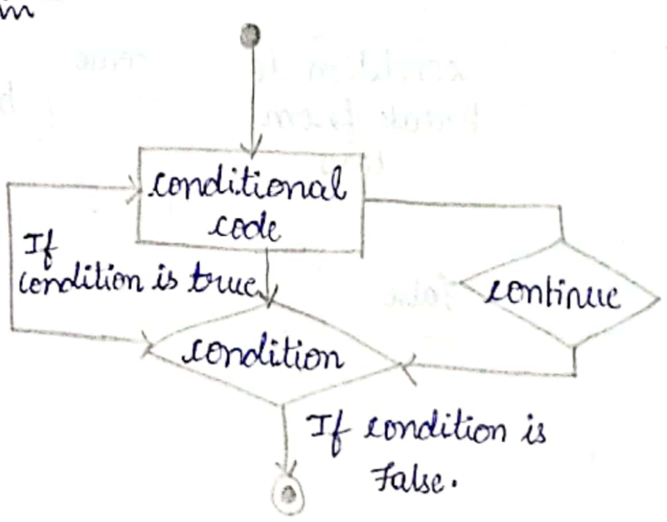
Continue statement:

The continue statement in C programming works somewhat like the break statement. Instead of forcing termination, it forces the next iteration of the loop to take place.

Syntax :-

```
Continue;
```

Flow Diagram:-



Example :-

```
# include < stdio . h >
{
int main ( )
For ( int a=1; a<10; a++ )
{
printf ("%d", a);
if (a==4)
continue ;
}
return 0;
}
```

OUTPUT :-

```
123456789
```

Goto statement:

A Goto statement in C programming provides an unconditional jump from the "goto" to a labeled statement in the same function.

NOTE: Use of goto statement is highly discouraged in any programming language because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a goto can be rewritten to avoid them.

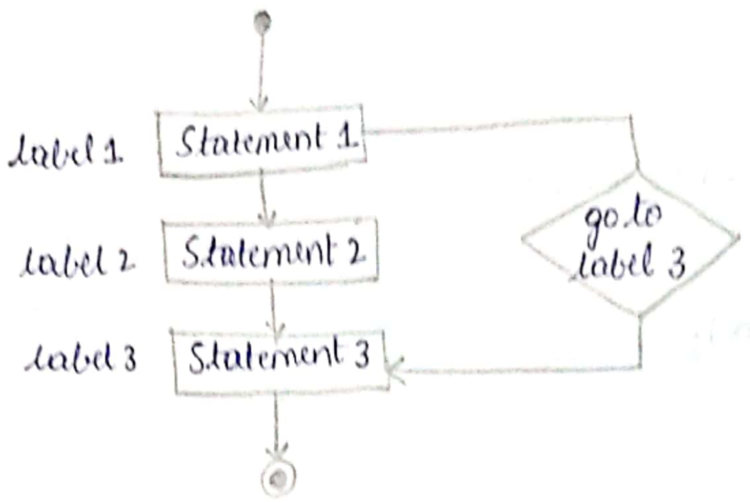
Syntax :-

The syntax for a goto statement in C is as follows-

```
goto label;
:
.
label: statement;
```

Here label can be any plain text except C keyword and it can be set anywhere in the C program above or below to goto statement.

Flow Diagram:-



Example:-

```
#include <stdio.h>
void main()
{
int num, i=1;
printf("Enter the number whose table you want to print?");
Scanf("%d", &num);
table:
 printf("%d x %d = %d\n", num, i, num*i);
 i++;
 if (i<=10)
 goto table;
}
```

OUTPUT:

```
Enter the number whose table you want to print?4
4x1=4
4x2=8
4x3=12
4x4=16
4x5=20
4x6=24
4x7=28
4x8=32
4x9=36
4x10=40
```

```
#include <stdio.h>
void main()
{
int num,a=1;
scanf ("%d", &num);
 table:
 printf ("%dx%d= %d\n", num, a, num*a);
 a++;
 if (a<=10)
 goto table ;
 }
```

OUTPUT:

```
5

5x1=5
5x2=10
5x3=15
5x4=20
5x5=25
5x6 = 30
5x7= 35
5x8 =40
5x9=45
5x10=50
```

Structure of C program

// sample program ——→ Documentation section

include < stdio.h>
include <conio.h> ——→ link section.

void main (), int main () ——→ Definition section.

int a=10; b=20; ——→ Global declaration section

```
main()
{
 Declaration part
 Executable part
 }
```
——→
```
void main(),int main()
{
 clrscr();
printf ("a value function");
Scanf ("%d", &a);
}
```

C - Scope

A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable it cannot be accessed. There are three places where variables can be declared in C programming language -

- Inside a function or a block which is called local variables.
- Outside of all functions which is called global variables.
- In the definition of function parameters which are called formal parameters.

local variables:

variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. local variables are not known to outside functions. The following example shows how local variables are used. Here all the variables a, b, and c are local to main () function.

program:

```
#include<stdio.h>
void main ()
{
int a=10; // here a is local variable because it is written
              inside main function.
printf ("%d", a);
}
```

OUTPUT:-

```
10
```

Global variables:-

variables that are declared outside of a function block and can be accessed inside the function is called global variables.

Global variables are defined outside a function or any specific block, in most of the case, on the top of the C program. These variables hold their values all through the end of the program and are accessible within any of the functions defined in your program.

Any function can access variables defined within the global scope, i.e., its availability stays for the entire program after being declared.

program:

```
# include <stdio.h>
int a=50; // global variable definition
void main ()
{
printf ("%d\n", a);
}
```

OUTPUT: 50.

Lifetime of a variable

Life time: life time of any variable is the time for which the particular variable is present in memory during running of the program. The lifetime of a variable is also known as the storage time of the variable.

Storage areas are of 2 types they are ~~me~~ memory and CPU register

Storage classes specify the scope, lifetime of variables.

To fully define a variable, one needs ~~a~~ to mention not only its "type" but also its storage class.

A variable name identifies some physical location within computer memory, where a collection of bits are allocated for storing values of variable.

Storage class tells us the following factors

- Where the variable is stored (in memory or CPU register);

- What will be the initial value of variable, if nothing is initialized?
- What is the scope of variable (where it can be accessed)?
- What is the life of a variable?

Lifetime
- where the variable is stored?

The lifetime of a variable ~~b~~ defines the duration for which the computer allocates memory for it (the duration between allocation and deallocation of memory).

In C language, a variable can have automatic, static or dynamic lifetime.

- Automatic: A variable with automatic lifetime are created. Every time, their declaration is encountered and destroyed. Also, their blocks are exited.
- Static: A variable is created when the declaration is executed for the first time. It is destroyed when the execution stops / terminates.
- Dynamic: The variables memory is allocated and deallocated through memory management functions.

Storage classes

There are four storages classes in C language-

Storage specifications	storage area	initial values	Scope of a variable	life time of a variable
auto	main memory	Garbage	within block	End of block
extern	main memory	zero	Global multiple files	Till end of program
Static	main memory	zero	within block	Till end of program
Register	CPU Register	zero	within blocks	End of block.

Scanned with CamScanner

Example 1: Following is the C program for automatic storage class-

```
#include <stdio.h>
void main()
{
auto int a=3;
printf("%d",a);
}
```

OUTPUT:-
```
3
```

Example-2: Following is the C program for external storage class-

```
#include <stdio.h>
extern int a=1; /* this"i" is a available throughout program*/
void main ()
{
 printf("%",a);
}
```

OUTPUT:-1

Command line Arguements in C

The arguements passed from command line are called commond line arguments. These arguments are handled by main() function.

To support commond line argument, you need to change the structure of main() function as given below.

Syntax: void maint (int argc, char*argv[]).

In the above statement, the command line arguments have been handled via the main() function, and you have set the arguments where

- argc (ARGument count) denotes the number of arguments to be passed and

• argv [ ](ARGument vector) denotes to a pointer array that is pointing to every argument that has been passed to your program.

You must make sure that in your command line argument, argv[0] stores the name of your program, Similarly argv[1] gets the pointer to the 1st command line argument that has been supplied by the user, and *argv[n] denotes the last argument of the list.

program for Command line Argument

```
#include <stdio.h>
int main (int argc, char *argv[])
{
printf ("program name %s\n", argv[0]);
if (argc = = 2)
{
 printf ("The argument supplied is %s\n", argv[1]);
}
else if (argc>2)
{
printf ("Too many arguments supplied.\n");
}
else
{
print f ("one argument expected.\n");
}
}
```

when the above code is compiled and executed with single argument, it produces the following result.

OUTPUT:-

```
$./a.out testing
The argument supplied is testing
```

when the above code is compiled and executed with a two arguments, it produces the following result.

OUTPUT:-

```
$./a.out testing 1 testing 2
Too many arguments supplied.
```

when the above code is compiled and executed without passing any argument, it produces the following result.

```
$./a.out
one argument expected
```

- It should be noted that argv[0] holds the name of the program itself and argv[1] is a pointer to the first command line argument supplied
- In the above program $./a.out I the name of program.

C-Input and Output

when we say Input, it means to feed some data into a program. An input can be given in the form of a file or from the command line. C programming provides a set of built-in functions to read the given input and feed it to the program as per requirement.

when we say Output, it means to display some data on screen, printer, or in any file. C programming provides a set of built-in functions to Output the data on the computer screen as well as to save it in text or binary files.

The standard files

C programming treats all the devices as files. So devices such as the display are addressed in the same way as files and the following three files are automatically opened when a program executes to provide access to the keyboard and screen.

Standard file	file pointer	Device
Standard input	Stdin	keyboard
Standard output	Stdout	Screen
standard error	Stderr	Screen

The file pointers are the means to access the file for reading and writing purpose. This section explains how to read values from the screen and how to print the result on the screen.

The getchar() and putchar() Functions:

The getchar() and putchar() functions

The getchar() function reads the next available character from the screen and returns it as integer. This function reads only single character at a time. You can use this method in the loop in case you want to read more than one character from the screen.

The putchar() function puts the passed character on the screen and returns the same character. This function puts only single character at a time. You can use this method in the loop in case you want to display more than one character on the screen.

Example:-

```
#include <stdio.h>
int main ( )
{
int c;
printf ("enter avalue:");
c=getchar();
printf (" \n you entered:");
putchar(c);
return0;
}
```

when the above code is compiled and executed, it waits for you to input some text. when you enter a text and press enter, then the program proceeds and reads only a single character and displays it as follows-

```
$./a.out
Enter a value: this is test
You entered: t
```

the gets() and puts () functions:

```
#include <stdio.h>
int main ()
{
char a[100];
printf ("Enter avalue:");
gets (a);
printf (" \n you entered: ");
puts (a);
reutrn 0;
}
```

when the above code is compiled and executed, it waits for you to input some text. when you enter a text and press enter, then the program proceeds and reads the complete line till end, and displays it as follows -

OUTPUT:-

```
$./a.out
Enter a value: this is test
You entered: this is test
```

The Scanf() and printf() Functions

printf():

printf() function is used in a C program to display any value like float, integer, character, string, etc on the console screen. It is a pre-defined function that is already declared in the stdio.h (header file).

Syntax1:

```
printf ("Format Specifier", var1, var2, -----, varn);
```

Example: `printf ("%d", a);`

Syntax2:

```
printf ("Enter the text which you want to display");
```

Scanf():

Scanf() function is used in the C program for reading or taking any value from the keyboard by the user, these values can be of any data type like integer, float, character, string, and many more. This function is declared in stdio.h, that's why it is also a pre-defined function. In scanf() function we use & (address of operator) which is used to store the variable value on the memory location of the variable.

Syntax:

```
Scanf (" Format specifier", & var1, &var2, -------&varn);
```

Example: `Scanf ("%d", &num1);`

Example program:

```
# include <stdio.h>
void main ()
{
int a;
printf ("enter any no");
Scanf ("%d", &a);
printf ("your no is %d", a);
}
```

OUTPUT:-

```
enter any no
10
your no is 10.
```

AND

P	Q	P∧Q
T	T	T
T	F	F
F	T	F
F	F	F

OR

P	Q	P∨Q
T	T	T
T	F	T
F	T	T
F	F	F

p	q	p↓q
T	T	F
T	F	F
F	T	F
F	F	T

p	q	~p	p→q	~p∨q
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T
1	2	3	4	5

Palindrome program in C

palindrome number in C:

A palindrome number is a number that is same after reverse. For example 212, 34543, 343, 131, 48984 are the palindrome numbers.

Program:

```
# include <stdio.h>
int main ()
{
int n, r, sum=0, temp;
printf ( "enter the number= ");
Scanf ("%d", &n);
 temp=n;
while (n>0)
{
 r=n%10;
Sum= (Sum * 10)+r;
n=n/10;
}
If (temp==sum)
printf (" palindrome number");
else
printf (" "not palindrome" ~~palindrome~~ number");
return 0;
}
```

OUTPUT:-

```
enter the number = 151
palindrome number.
enter the number =5621
not palindrome number.
```

C program to check whether a number is prime or not

program:-

```
#include <stdio.h>
int main ()
{
int num;      // Declare the number
printf (" Enter the number\n");
scanf ("%d", &num);   // Initialize the number
int c=0;      // Declare a count variable
for (int i=2, i< num; i++) // check for factors
{
  if (num % i = = 0)
    c++;
}
if (c!=0)     // check whether prime ornot
{
  printf ("Not a prime number\n");
}
else
{
  printf (" prime number\n");
}
  returno;
 }
```

OUTPUT:-

```
Enter the number
11
prime number
```

program to check Even or odd

An even number is an integer that is exactly divisible by 2. For example : 0, 4, 8,

An odd number is an integer that is not exactly divisible by 2. For example : 1, 7, -11, 15.

program:-

```
# include <stdio.h>
 int main ()
 {
int num;
printf ("Enter an integer: ");
Scanf ("%d", &num);
//true if num is perfectly divisible by 2
if (num % 2 ==0)
 printf("% d is even.", num);
else
  printf ("%d is odd.", num);
reutrn0;
 }
```

OUTPUT :-

```
Enter an integer: 7
 7 is odd.
```

Factorial pgm.

C program to compeute the average of three given numbers

C program:-

```
# include <stdio.h>
# include< conio.h>
void main ()
{
int a, b, c, d;
clrscr();
 printf ("\n Enter three numbers: ");
 scanf ("%d %d %d", &a, &b, &c);
 d= (a+b+c)/3;
  printf (" \n average : %d ", d);
  getch ();
 }
```

OUTPUT:-

```
Enter three numbers   10   20   30

Average : 20.
```

	STRUCTURE	UNION
Keyword	The keyboword struck is used to define a structure	used to define a union.
size	when a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is greater than or equal to the sum of sizes of its members.	. when a variable is associat with a structure, compiler allocates the memory by considering size of the largest memory. So, size of union is equal to size of largest numb
memory	Each member within a structure is assigned unique storaage area of location.	memory allocated is shared by individual members of union.
value altering	Altering the value of a member will not affend other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Indiuidual member can be accessed at a time.	only one member can be accessed at a time.
Initialization of members	Several members of a structure can initialize at once.	only the first member of a union can be initialized.

UNIT-2

TOPICS

1.) Arrays, accessing and manipulating elements of array, 1D (dimensional) and 2D (dimensional) arrays.

2.) Strings, strings as arrays, string functions (strlen, strcat, strcpy, strstr, ...)

3.) Structures, initializing structures, unions, array of structures.

4.) pointers, pointers to arrays and structures, use of pointers in self referential structures

5.) Enumeration data type.

TOPIC-1-Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variable for each value.

To create an array, define the data type (link int) and specify the name of the array followed by square brackets [ ].

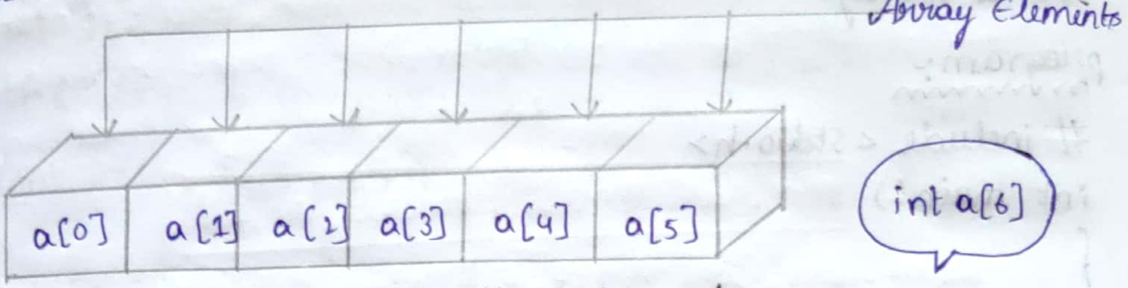
To insert values to it, use a comma-separated list, inside curly braces:

int num[ ]=[25, 50, 75, 100];

we have now created a variable that holds an arry of four integers.

Access the Elements of an Array:-

To access an array element, refer to its index number. Array indexes start with 0: [0] is the first element. [1] is the second element. ect.



1-D Array with 6 Elements

This statement accesses the value of the first element [0] in a:

Example:

```
# include <stdio.h>
int main ()
{
 int a[] = {25 ,50,75, 100};
  printf ("%d", a[0]);
 return0;
 }
```

OUTPUT:-
```
25
```

Change an Array Element

To change the value of a specific element, refer to the index number:

Example:-

```
# include<stdio.h>
int main()
{
int num[]= {25 ,50,75,100};
num[0]=33;
printf ("%d", num [0]);
return 0;
}
```

OUTPUT: 33.

loop through an Array:-

You can loop through an array elements with the for loop. The following example outputs all elements in the ~~a~~ array.

program:-

```
#include <stdio.h>
int main()
{
int a[] = {25, 50, 75, 100};
int i;
for (i=0; i<4; i++)
{
printf ("a[%d]=%d\n", i, a[i]);
}
return 0;
}
```

OUTPUT:-

```
a[0] =25
a[1] =50
a[2]=75
a[3]=100
```

Array declaration:-

we declare array by scanf, for array declaration we must definitely include size of array

Example:-

```
ina[10]; // here 10 is size of array.
```

// array declaration to print 5 values

```
# include <stdio.h>
int main()
{
int a[5];
printf("Enter 5 integers: ");
for(int i=0 ; i<5 ; ++i)
{
 scanf ("%d", &a[i]);
 printf ("your value %d", a[i]);
}
returno;
}
```

OUTPUT:-

```
Enter 5 integers :1234
Your value 1234
```

Set Array size :-

Another common way to create arrays, is to specify the size of the array, and add elements later:

```
# include <stdio.h>
int main ()
{
// Declare an array of four integers:
int a[4];
// Add elements to it
a[0]=25;
a[1] = 50;
a [2] = 75;
a[3]=100;
printf("%d\n", a[0]);
returno;
}
```

OUTPUT:-

```
25
```

2 dimensional array

- 2-D Arrays can be defined as an array of arrays,
- It can also represent a Matrix,
- Each element is represented as Arr [row][coloumn], where Arr[][] is the 2D Array.

	col 1	col 2	col 3	col 4
Row 1	Arr[0][0]	Arr[0][1]	Arr[0][2]	Arr[0][3]
Row 2	Arr[1][0]	Arr[1][1]	Arr[1][2]	Arr[1][3]
Row 3	Arr[2][0]	Arr[2][1]	Arr[2][2]	Arr[2][3]
Row 4	Arr[3][0]	Arr[3][1]	Arr[3][2]	Arr[3][3]

Declaration of 2D-Arrays in C

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

Syntax: data-type arry-name [rows][columns];

Example: int a [4][3];

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

Initializing Two-Dimensional Arrays (2D array):

There are various ways in which a Two-Dimensional array can be initialized.

First Method:

Example:

```
int x[3][3] = {0,1,2,3,4,5,6,7,8};
```

The above array has 3 rows and 3 columns. The ~~err~~ elements in the braces from left to right are stored in the table also from left to right. The elements will be filled in the array in order, the first 3 elements from the left in the first row, the next 3 elements in the second row, and so on.

Second method :-

Example :
int x[3][3] = {{0,1,2}, {3,4,5}, {6,7,8}};

program:

```
# include <stdio.h>
int main()
{
int i=0 , j=0;
int a[3][3] = {{1,2,3}, {2,3,4}, {3,4,5}};
// traversing 2D array
for (i=0; i<3; i++)
{
 for (j=0; j<3; j++)
 {
 printf ("[%d][%d] = %d \n", i, j, a[i][j]);
 } // end of j
 } //end of i
 return0;
 }
```

OUTPUT :-

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

Scanned with CamScanner

C strings

Strings are used for storing text/characters.
For example, "Hello world" is a string of characters.
Unlike many other programming languages, C does not have a string type to easily create string variables.
However, you can use the char type and create an array of characters to make a string in C:

```
char a[] = "Hello world!";
```

Note that you have to use double quotes
To output the string, you can use the printf() function together with the format specifier %s to tell C that we are now working with strings:

Program:

```
#include <stdio.h>
int main()
{
char a[]= "Hello world!";
printf("%s",a);
return0;
}
```

OUTPUT:- Hello world!

String Declaration:-

For string declaration we use scanf, for string declaration definitely we need declare size of string.

Example: char a[10];

program:-

```
//. string declaration
# include <stdio.h>
void main ()
{
char a[10];
printf (" enter text : \n");
scanf ("%s", &a);
printf ("your text is : %s", a);
}
```

OUTPUT:-

```
enter text:
hello
your text is : hello
```

Access strings:-

Since strings are actually arrays in C, you can access a string by referring to its index number inside square brackets [].

This example prints the first character (0) in greetings:

Note that we have to use the %c format specifier to print a single character.

program:-

```
#include <stdio.h>
int main ()
{
char a[] = "Hello world!";
printf ("%c", a[0]);
return 0;
}
```

OUTPUT:- H.

Scanned with CamScanner

Modify strings:-

To change the value of a specific character in a string, refer to the index number, and use single quotes:

Program:-

```
#include<stdio.h>
int main()
{
char a[]= "Hello world!";
a[0]='J';
printf("%s",a);
return 0;
}
```

OUTPUT:- Hello world.

Another way of creating strings:-

In the examples above, we used a "string literal" to create a string variable. This is the easiest way to create a string in C.

You should also note that you can to create a string with a set of characters. This example will produce the same result as the one above:

\0 character is known as the "null termininating character", and must be included when creating strings using this method. It tells C that this is the end of the string.

Program:-

```
#include<stdio.h>
int main()
{
char a[]={'H','e','l','l','o',' ','W','o','r','l','d','!','\0'};
printf("%s\n",a);
return 0;
}
```

OUTPUT:- Hello world!

You should note that the size of both arrays is the same; They both have 13 characters (space also counts as a character by the way), including the \0 character:

C string functions:-

There are many important string functions defined in "string.h" library. ☆

S.NO.	Function	Description
1.	strlen (string name) (length)	returns the length of string name.
2.	strcpy (destination. source) (copy string)	copies the contents of source string to destination string.
3.	strcat (first string. second string) [a].[b]	concats or joins first string with second string. The result of the string is stored in first string.
4.	strcmp (first string, second string) (equal (or) unequal)	compares the first string with second string. If both strings are same, it returns 0.
5.	strrev (string) (reverse)	returns reverse string.
6.	strlwr (string). (lowercase)	returns string characters in lowercase.
7.	strupr (string) (upper case)	returns string characters in upper case.

Function strlen() program:

```
#include <stdio.h>
#include <string.h>
int main()
{
char a[20] = "viha";
printf ("length of string is: %d", strlen (a));
return 0;
}
```

OUTPUT:

```
length of string is:4
```

C copy string: strcpy() program:

```
#include <stdio.h>
#include <string.h>
int main()
{
char a[20]= " vihanvika";
char b [20];
strcpy (a,b);
printf ("value of second string is : %s", b);
return 0;
}
```

OUTPUT:

```
value of second string is : vihanvika.
```

Function strrev() program:

```
#include <stdio.h>
#include <string.h>
int main ()
{
char a[20] = "viha "
printf ("length of string is : %s", strrev(a));
return 0;
}
```

OUTPUT:

```
ahiv.
```

Function strlwr () program:-

```
# include <stdio.h>
# include < string.h>
int main ( )
{
char a[20] = "VIHA";
printf (" lower case is: %s", strlwr(a));
return 0;
}
```

output:-

```
lower case is: viha.
```

function strupr() program:-

```
# include < stdio.h>
# include < string.h>
int main ()
{
char a[20]= "viha";
printf (" upper case is: %s" strupr (a));
return 0;
}
```

output:-

```
upper case is : VIHA.
```

Function strcmp () program:

```
#include <stdio.h>
#include <string.h>
void main()
{
char a[]= "viha", b[]= "viha";
if (strcmp (a,b)= =0)
{
printf ("equal");
}
else
{
printf ("not equal");
}
}
```

OUTPUT:- equal.

```
#include <stdio.h>
#include <string.h>
void main ()
{
char a[] = "viha", b[] = "vihanvika";
if (strcmp(a,b) ==0)
{
printf ("equal");
}
else
{
printf("not equal");
}
}
```

OUTPUT:- not equal.

Function strcat() program:-

```
# include<stdio.h>
# include <string.h>
void main ()
{
char a [] = "vyshali", b[] = "reddy";
printf("combined string is: %s", strcat (a,b));
}
```

OUTPUT:-

```
vyshalireddy..
```

Scanned with CamScanner

Structures in C

Structures (also called structs) are a way to group variables of different data types into one place. Each variable in the structure is known as a member of the structure.

Unlike an array, a structure can contain many different data types (int, float, char, etc.).

Create a structure :-

You can create a structure by using the struct keyword and declare each of its members inside curly braces:

Syntax:

```
Structure structure tag
{
member definition;
member definition;
....
member definition;
};
```

Example :-

```
struct book // structure declaration, here num is structure
                tag name
{
int pages; // Member (int variable)
float price; // Member (float variable)
char author [30]; // Member (char variable)
}; // End the structure with a semicolon
```

Scanned with CamScanner

Initialization and accessing of structures:

we can initialization and accessing of structures in 3 ways.

Example programs:

Method-1:

```
# include < stdio.h>
// create a structure called my structure
struct data
{
int a;
char b;
float c;
}
d={10, 'B', 10.5};
int main ()
{
// print values
printf ("value of a is : %d \n", d.a);
printf ("value of b is : %c \n", d.b);
printf ("value of c is : %f \n", d.c);
return 0;
}
```

OUTPUT:-

```
value of a is :13
value of b is : B
value of c is : 10.500000
```

Scanned with CamScanner

Method-2:-

```
# include < stdio.h >
//create a structure called mystructure
struct data
{
int a;
char b;
float c;
};
int main()
{
struct data = d {10, 'B', 10.5};
//print values
printf ("value of a is : %d\n", d.a);
printf ("value of b is : %c\n", d.b);
printf ("value of c is : %f\n", d.c);
return 0;
}
```

OUTPUT:-

```
value of a is :13
value of b is : B
value of c is : 10.500000
```

Method-3:-

Access structure members

To access members of a structure, use the dot syntax (.):

```
# include <stdio.h>
// create a structure called mystructure
struct data
```

```
{
 int a;
char b;
float c;
 };
int main()
{
struct data d;
// Assign values
d.a=13;
d.b= 'B';
d.c=10.5;
// print values
printf ("value of a is: %d\n", d.a);
printf ("value of b is: %c\n", d.b);
printf (" value of c is : %f\n", d.c);
return 0;
}
```

OUTPUT:-

```
value of a is :13
value of b is : B
value of c is : 10.500000
```

Array of structures in C

An array of structures in C can be defined as the collection of multiple structures variables where each variable contains information about different entities.

The array of structures in C are used to store information about multiple entities of different data types. The array of structures is also known as the collection of structures.

Example:-

```
#include<stdio.h>
# ~~str~~ include< string.h>
struct student
{
int rollno;
char name[10];
};
int main ()
{
 int i;
 struct student std[5];
 printf ("Enter Records of 5 students");
 for (i=0; i<5; i++)
 {
  printf ("\n Enter Rollno:");
  Scanf ("%d", & std [i].rollno);
  printf ("\n Enter Name:");
  Scanf ("%s", & std [i].name);
  }
  printf ("\n Student information list:");
  for (i=0; i<5; i++)
  {
  printf ("\n Rollno:%d , Name:%s", std [i].roll no, std[i]
                                                    .name);
  }
  return 0;
  }
```

OUTPUT:-

```
Enter Records of 5 students
Enter Rollno : 1
Enter Name : vyshali
Enter Rollno: 2
Enter Name: viha
Enter Roll no: 3
Enter name: vihanvi
Enter Roll no: 4
Enter name : Vihanvika
Enter Roll no : 5
Enter Name: vihanth

Student Information list:
Roll no : 1, Name: vyshali
Rollno : 2 , Name: viha
Rollno:3 , Name : vihanvi
Rollno :4 , name: vihanvika
Rollno :5 , Name ; vihanth
```

Union in C

Like structures, union is a user defined data type. In union, all members share the same memory location. To define a union, you must use the union statement in the same way as you did in a structure.

Syntax:-

```
union union tag
{
member definition ;
member definition;
....
member definition;
} variables;
```

Example:

```
union book // union declaration, here num is uniontag
                name
{
int pages; // Member (int variable)
float price ; // Member (float variable)
char author [30]; // Member (char variable)
}; // End the union with a semicolon
```

Example program:

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

union Data
{
int a;
float b;
char c [20];
};
int main ()
{
union Data d;
printf ("Memory size occupied by data: %d\n", size of
                (d));
return 0;
}
```

OUTPUT: Memory size occupied by data :20

Enumeration (or enum) in C

The enum in C is also known as the enumerated type. It is a user-defined data type that consists of integer values, and it provides meaningful names to these values. The use of enum in C makes the program, easy to understand and maintain. The enum is defined by using the enum keyboard.

The following is the way to define the enum is C:

Syntax : enum data {integer- const 1, integer-const 2,.......
integer- const N};

In the above declaration, we define the enum named as data containing 'N' integer constants. The default value of integer - const 1 is 0, integer- const 2 is 1, and so on. We can also change the default value of the integer constants at the time of the declaration.

Example :

```
enum fruits {mango = 3, apple = 4, strawberry = 19,
             papaya = 27};
```

Enumerated type declaration:

program-1 :

```
# include <stdio.h>
enum week {Mon, Tue, Wed, Thur, Fri, Sat, Sun};
void main ()
{
printf ("The value of enum week : %d \t %d \t %d \t %d \t
                                     %d \t %d \t %d \n \n",
        Mon, Tue, Wed, Thur, Fri, Sat, Sun);
}
```

OUTPUT:
```
The value of enum week : 0 1 2 3 4 5 6
```

program-2:

```
# include <stdio.h>
enum week {Mon, Tue, Wed, Thur, Fri, Sat, Sun};
void main ()
{
printf ("%d", day);
}
```

OUTPUT:

```
2
```

program-3

```
#include <stdio.h>
enum week {Mon=10, Tue, Wed, Thur, Fri, Sat, Sun};
void main ()
{
 printf ("The value of enum week: %d\t%d\t%d\t%d\t
                                   %d\t%d\t%d\n\n",
         Mon, Tue, Wed, Thur, Fri, Sat, Sun);
}
```

OUTPUT:-

```
The value of enum week: 10   11   12   13   14   15   16
```

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.

If you have a variable a in your program, &a will given you its address in the memory.

We have used address numerous times while using the scanf() function.

```
Scanf ("%d", &a);
```

Here, the value entered by the user is stored in the address of a variable. Let's take a working example.

Example 1:

```
# include <stdio.h>
int main()
{
int a =43;
printf ("%d\n", a);
printf ("%p\n", &a);
return 0;
}
```

OUTPUT:
```
43
0x7ffe5367c044
```

In the example above, &a is also known as a pointer. A pointer is a variable that stores the memory address of another variable as its value.

The pointer variable points to a data type (like int) of the same type, and is created with the * operator. the address of the variable you're working with is assigned to the pointer:

Example:

```
# include <Stdio.h>
void main()
{
int a=43 ; // An int variable
int *b =&a ; // A pointer variable, with the name b,
            that stores the address of a.
printf ("%d \n", a);
printf ("% p \n", b);
}
```

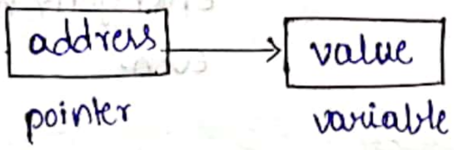
```
OUTPUT :- 43
          0x7ffe5367e044
```

pointer to array

```
int a[10];
int *p[10] = &a;
```

pointer to structure

```
struct st
{
int i;
float f;
} s;
Struct st *p = &s;
```



Advantage of pointer :-

1.) pointer reduces the code and improves the performance,

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.

```
#include <stdio.h>
int main ( )
{
 int a;
 printf ("Enter your no :");
 Scanf ("%d", &a);
 if (a%2 ==0)
 {
  printf ("Even");
 }
 else
 {
 printf ("odd");
 }
 return 0;
}
```

OUTPUT:-

```
Enter your no : 0
Even
Enter your no : 1
odd
Enter your no : 2
Even
Enter your no : 3
odd
Enter your no : 4
even
Enter your no : 100
Even.
```

Type conversions:-

Converting one data type into another data type is called as type conversions.

(i.) Implict type conversion

(ii.) Explict type conversion

(i)
```
#include <stdio.h>
void main ( )
{
 int a = 10/3;
 printf ("%d", a);
}
```
o/p = 3

(ii)
```
#include <stdio.h>
void main ( )
{
 float a = 10/3;
 printf ("%f", a);
}
```
o/p = 3.0000

Implict type conversion :- It is automatically done by the compiler by ①* converting smaller data type into a longer data type.

Explict type conversion :- (type casting)
It is done by the user* ① manually, it will convert ① * larger data type into smaller data type.

UNIT-3

1.) preprocessors, [preprocessor commands like include, define, undef, if, ifdef, ifndef].

2.) Files, text and Binary files, creating, reading, writing, text and Binary files.

3.) Appending data to existing files.

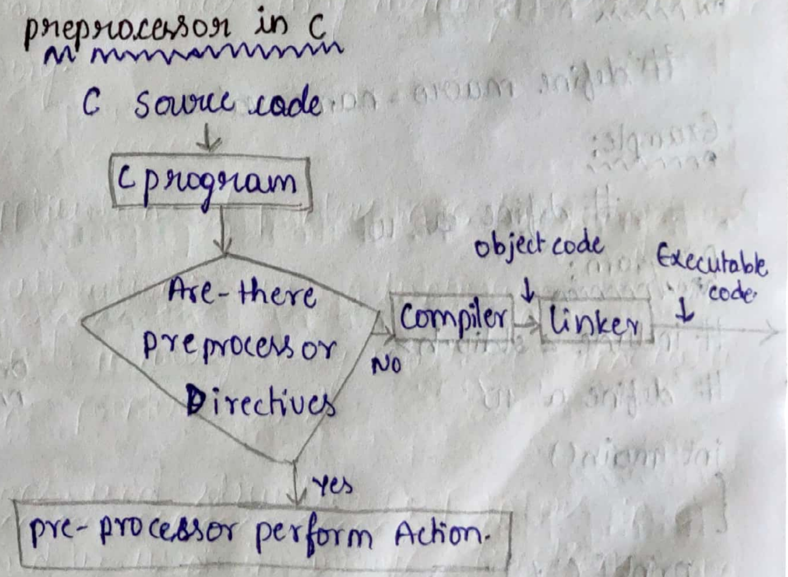
4.) writing and reading structures using binary files

5.) Random access using fseek, ftell, rewind.

C - processor :-

The C ~~pro~~ preprocessor is not a part of the compiler, but is a separate step in the compilation process. In simple terms, a C preprocessor is just a text substitute. A tool and it instructs the compiler to do required pre-processing before the actual compilation. All preprocessor commands begin will a hash symbol (#).

Diagram :-



You can see the intermediate steps in the above diagram; The source code written by programmers is first stored in a file, let the name be "program.c". This file is then processed by preprocessors and an expanded code file is generated named program.i". This expanded file is compiled by the compiler and an object code file is generated by named "program.obj". Finally, the linker links this object code file to the object code of the library functions to generate the executable file".

There are 4- main types of preprocessor directives;

1. Macros
2. File Inclusion
3. Conditional compilation
4. Other directives.

1. Macros: The "#define" directive is used to define a macro. Macros are pieces of code in a program that is given some name. Whenever this name is encountered by the compiler, the compiler replaces the name with the actual value.

Syntax:

```
#define macro-name value
```

Example:

```
#define a 10
```

program:

```
#include <stdio.h>
#define a 10
int main()
{
 printf("%d",a);
}
```

OUTPUT:- 10.

2. File Inclusion:-

this type of preprocessor directive tells the compiler to include a file in the source code program. there are two types of files that can be included by the user in the program. Header files or standard files. these files contain definitions of pre-defined functions like printf (), Scanf () etc. These files must be included to work with these functions. Different functions are declared in different header files. For example, standard I/O functions are in the "stdio.h" file whereas functions that perform string operations are in the "string.h" file

Syntax:-

```
# include <files-name>
 or
# include " file-name"
```

Example:-

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

program:

```
# include <stdio.h>
void main ()
{
 printf ("Hello world!");
}
```

OUTPUT:- Hello world!

3. Conditional Compilation:-

Conditional compilation directives are a type of directive that helps to compile a specific portion of the program or to skip the compilation of some specific part of the program based on some conditions. This can be done with the help of the preprocessing Commands like ;

1. #if def
2. #if n def
3. #if
4. # else
5. # elif
6. # endif

1. #if def :- The #ifdef preprocessor directive checks if macro is defined by #define. If yes, only then it executes the code.

Syntax :

```
#ifdef Macro
 # Code
 # endif
```

program:

```
# include <stdio.h>
# define a 10
void main ()
{
#ifdef a
printf ("hello");
#endif
}
```

OUTPUT: hello.

2. #infndef :- The #ifndef preprocessor directive checks if macro is not defined by # define. If not defined, it executes the code.

Syntax :-

```
#ifndef Macro
// code
# endif
```

program:-

OUTPUT:-

```
# include <stdio.h>
# define. a   10
void main ()
{
# in Ifndef a
printf (" Hello");
# endif
}
```

3. #if #elif #else :

The # if preprocessor directive evaluates the expression or condition. If condition is true, it executes the code otherwise . # else if or # else or # endif code is executed.

Syntax:
```
# if  expression
# if code
# elif  expression
# elif code
# else
# else code
# endif
```

program:-

```
# include < stdio.h >
# define a 15
void main ()
{
    #if a<10
    printf ("value is less than 10");
    #elif a<20
    printf (" value is less than 20");
#else
printf("number not found");
#endif
}
```

OUTPUT: value is less than 20.

4. Other Directives:- A part from the above directives, there are two more directives that are not commonly used. These are: #undef directive : the #undef directive is used to undefine an existing macro. this directive works as:

example:

```
#define a 10
#undef a
```

Program:

```
#include<stdio.h>
#define a 10
#undef a
int main()
{
 printf("%d", a);
}
```

OUTPUT:

```
error message.
```

#pragma Directive: This directive is a special purpose directive and is used to turn on or off some features. this type of directives are compiler-specific, i.e., they vary from compiler to compiler. Some of the #pragma directives are discussed below:

```
#pragma startup
#pragma exit
#pragma warn
```

File handling in C

Files is collection of records (or) it is a place on hard disk, where data is stored permanently.

Declaring a file:

For creating a file, pointer with file is used

Example: FILE *a;

Here,
FILE is type and
a is file pointer

Types of files

There are two types of files in C language which are as follows -

- Text file
- Binary file

Text file :-

- It contains alphabets and numbers and special symbols which are easily understood by human beings.
- An error in a text file can be eliminated when seen.
- In text file, the text and characters will store one char per byte.
- For example, the integer value 4567 will occupy 4 bytes in text file.
- The data format is usually line-oriented. Here, each line is a separate command.

Binary file:-

- It contains 1's and 0's, which are easily understood by computers.
- The error in a binary file corrupts the file and is not easy to detect.

- In binary file, the integer value 1245 will occupy 2 bytes in memory and in file.
- A binary file always needs a matching software to read or write it.
- For example, an MP3 file like VLC player is used to play music.
- MP3 file will not play in an image viewer. in same way A binary file always needs a matching software

→ Files are classified into fallowing

- Sequential files:- Here, data is stored and retained in a sequential manner.
- Random access files:- Here, data is stored and retrieved in in Random way.

C file operations:-

Five major operations can be performed on file are:

- Creation of a new file.
- Opening an existing file.
- Reading data from a file.
- Writing data in a file.
- Closing a file.

To handling files in C, file functions available in the stdio library are:

fopen	Opens a file
fclose	closes a file
getc	Reads a character from a file
putc	writes a character to a file
getw	Read integer
putw	write an integer
fprintf	Prints formatted output to a file
fscanf	Reads formatted input from a file
fgets	Read string of characters from a file
fputs	write string of characters to file.

So far the operations using C program are done on a prompt / terminal which is not stored anywhere. But in the software industry, most of the programs are written to store the information of the program. One such way is to store the information in a file. Different operations that can be performed on a file are:

Opening or creating file:

For opening a file, fopen function is used with the required access modes.

r	Open for reading a file
r+	Open for reading and writing a file.
w	Open for writing and create the file if it doesnot exist. If the file exists then make it blank.
w+	Open for reading and writing and create the file if it doesnot exist. If the file exists then make it blank.
a	Open for appending (writing at the end of the file) and create the file if it doesnot exist.
a+	Open for reading and appending and create the file if it does not exist.

Various modes for binary files:

rb	Open for reading binary file.
rb+	Open for reading and writing binary file
wb	Open for writing and create the binary file if it doesnot exist. If the file exists then make it blank.

wb+ Open for reading and writing and create the binary file if it doesnot exist. If the file exists then make it blank.

ab Open for appending (writing at the end of the binary file) and create the file if it doesnot exist.

ab+ Open for reading and appending and create the binary file if it doesnot exist.

Example: FILE *a;

```
a=fopen ("E:\fileName.txt", "w")
```

• Reading from a file-

The file read operations can be performed using functions fscanf or fgets. Both the functions performed the same operations as that of scanf and gets but with an additional parameter, the ~~po~~ file pointer.

Example: FILE *a;

```
a=fopen ("E:\myfile.txt","r");
fscanf (a, "%d %d %d %d", &b,&c,&d,&e);
```

• Writing a file :-

The file write operations can be performed by the functions fprintf and fputs with similarities to read operations. The snippet for writing to a file is as :

Example: FILE *arr[60];

```
a= fopen ("fileName.txt", "w");
fscanf ( f, "%s", arr);
```

• Closing a file - After every successful file operations, you must always close a file. For closing a file, you have to use fclose function. The snippet for closing a file is given as:

Example :-
```
FILE *a;
    a = fopen ("fileName.txt", "w");
    ......... Some file operations ..........
    fclose(a)
```

Program 1:

```
#include <stdio.h>
void main ()
{
 FILE* a;
 a = fopen ("E:\file.txt", "w");
 fprintf (a, "my name is vyshali");
 fclose(a);
 }
```

By writing above program it will create file.txt text file in local disk E and that file contain text i.e, my name is vyshali.

Program 2:

```
# include <stdio.h>
void main ()
{
FILE *f;
char arr[50];
f = fopen ("file.txt", "r");
while (fscanf (f, "%s", arr) ! = EOF)
{
printf ("%s", arr);
}
fclose(f);
}
```

=> here f is address where file is present, I want to read data present inside the file. So here I too "r" mode where r stands for read.
There are so many characters present inside the file for that purpose I took while loop, inside while loop, I written fscanf, by using fscanf I can read data present inside the file, and data is in the form of string. so include %S, arr.
↳ arr will read file data.

until end of string [where EOF stands for end of string] in order to print file data on output screen. I use print ("%S", arr);

(Example [ch-3 X]
program:-

```
# include <stdio.h>
# define a  15
void main()
{
#if a<10
printf (" value is less than 10");
# elif a<20
printf ("value is less than 20");
#else
printf ("not a valid ");
#endif
}
```

OUTPUT: value is less than 20.

Program:-

```
#include <stdio.h>
void main()
{
 char str[100];
 int i;
 FILE *fp=fopen("data.txt","r");
 if(fp==NULL)
   printf("File opening failed");
 else
 {
  printf("Data in the file is :\n");
  for(i=1;i<=4;i++)
  {
   fgets(str,99,fp);
   puts(str);
  }
  fclose(fp);
 }
}
```

C program to append data to text file :-

This C program will add message entered by user into the data.txt file. Note that, present content will remain as it is, a new content will be appended at the bottom of data.

Program:-

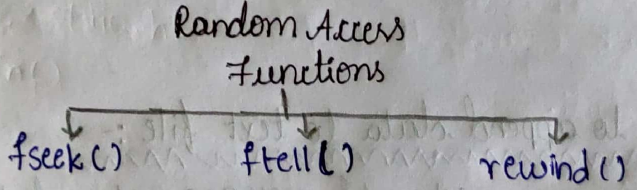
```
#include <stdio.h>
void main()
{
 FILE *fp;
char str[80];
fp=fopen ("data.txt", "a");
printf ("Enter your message:");
gets (str);
fprintf( fp, "%S", str);
Printf ("Your message is appended in data.txt file!");
fclose(fp);
}
```

OUTPUT of program:

```
Enter your message : How are you?
Your message is appended in data.txt file.
Original content of data.txt file is : Hello student.
After execution of program, content of data.txt file is :
 Hello student. How are you?
```

Random access file functions in C

Random access file functions in C are mainly of 3 types, they are:



rewind():

The function rewind() is used to set the position of file to the begining of given stream. It does not return any value.

Syntax of rewind():

```
rewind (FILE* stream);
```

fseek():

fseek() in C language is used to move file pointer to a specific position. Offset and stream are the destination of pointer, given in the function parameters.

Syntax of fseek():

fseek (FILE *stream, long int offset, int whence)

The parameters used in fseek():

- stream - This is the pointer to identify the stream.
- offset - It is used to place cursor in specific position, for example if you give char as 5 then it will place cursor at 5 characters.
- whence - this is the position from where cursor is added.

whence is specified by constants:

- SEEK_END : End of file.
- SEEK_SET : Starting of file.
- SEEK_CUR : Current position of file pointer.

ftell():

ftell in C is used to find out the position of file pointer in the file with respect to starting of the file.

Syntax of ftell() is :

ftell (FILE * pointer)

Example program for rewind:

```
#include <stdio.h>
void main()
{
FILE *a;
char b;
A= fopen ("sample.txt", "r");
printf ("reading file data for the first time\n");
While (b=getc(a)!=EOF)
putchar (b);
Rewind (a); // set the file pointer to starting position of file
printf ("reading file data for the second time\n");
While (b=getc(a)!=EOF)
putchar (b);
fclose(a);
}
```

For example sample.txt file contain ~~tex~~ text as "Engineering college" then I will get output as

OUTPUT:

```
reading file data for the first time
Engineering college
reading file data for the second time
Engineering college.
```

Example program for fseek and ftell:

```
#include<stdio.h>
void main()
{
FILE *a;
char b;
A=fopen ("sample.txt", "r");
fseek (a, 0, SEEK_SET);
b=getc(a);
Printf ("character at %d location is %c\n ", ftell(a), b);
fseek (a, 3, SEEK-CUR);
b=getc(a);
printf ("character at %d location is %c\n", ftell (a), b);
fseek (a,-1, SEE-END);
b= get c (a);
printf ("character at %d location is %c\n", ftell (a), b);

fclose(a);

}
```

OUTPUT:-

For example sample.txt file contain text as "Engineering college" then I will get output as

```
character at 1 location is E
character at 4 location is g
character at 19 location is g.
```

UNIT 4 : functions

1.functions, types of functions, types of user defined functions

2.function declaration and definition

3.function parameters (or) arguments, return type of function

4.standard library functions in c

5.parameter passing techniques(call by value and call by reference)

6.passing pointers and arrays to functions

7.recursion, finding number factorial, Fibonacci series using functions, limitations of recursion

8.dynamic memory allocation(malloc(),calloc(),realloc(),free() functions)

UNIT 4

Functions

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times.

types of function

There are two types of function in C programming:

- Standard library functions
- User-defined functions
-

Standard library functions

The standard library functions are built-in functions in C programming.

These functions are defined in header files. For example,

- The printf(),scanf(),main() are standard library function. This function is defined in the stdio.h header file. Hence, to use the printf(),scanf()functions, we need to include the stdio.h header file using #include <stdio.h>.

EXAMPLE:

```
#include <stdio.h>
void  main()
{
  printf("Hello World!");

}
```

User-defined function

You can also create functions as per your need. Such functions created by the user are known as user-defined functions.

How user-defined function works(structure of function):

```
#include <stdio.h>
void myfun()
{
    ... .. ...
    ... .. ...
}
int main()
{
   myfun();

}
```

The execution of a C program begins from the main() function.
When the compiler encounters myfun();, control of the program jumps to
And, the compiler starts executing the codes inside myfun().
The control of the program jumps back to the main() function once code inside the function definition is executed.

Advantages of user-defined function

1. The program will be easier to understand, maintain and easy to remove errors.
2. Reusable codes that can be used in other programs
3. A large program can be divided into smaller modules. Hence, a large project can be divided among many programmers.

Types of user defined functions in C

Different types of user-defined functions: A function, depending on whether arguments are present or not and whether a value is returned or not, may belong to one of the following categories.

- *Category* I: Functions with no arguments and no return values
- *Category 2:* Functions with no arguments and with return values
- *Category* 3: Functions with arguments and no return values
- *Category 4:* Functions with arguments and with return values

1) Functions with no arguments and no return value:

The following syntax describes how to write a function which neither takes any arguments nor returns any value.

Syntax for function definition:

```
void functionName()     //Return type in the function header should be 'void' and do not give any arguments within the parenthesis.
{
........
........
}//Need not return any value
```

Example program :

```
#include<stdio.h>

 void myFunction()

{

printf("I AM NAGENDRA");

 }

 int main()

{

myFunction();

myFunction();

myFunction();
return 0;

}
```

Output:

```
 I AM NAGENDRA

I AM NAGENDRA

I AM NAGENDRA
```

2) Functions with no arguments and with return values:

The following syntax shows how to write a function which does not take any arguments but returns some value back to the calling function.

Syntax for function definition :

```
returnType functionName()      //Identify what type of value will be returned by the function and accordingly specify the return type and empty

 {
parenthesis since there is no argument
. . . .
. . . .
return value;
}
```

Example program :

```
#include<stdio.h>

int myFunction()

{

printf("I AM NAGENDRA");

return 0;

 }
 int main()
{
myFunction();
myFunction();
myFunction();
return 0;
}
```

Output:

```
 I AM NAGENDRA
 I AM NAGENDRA
I AM NAGENDRA
```

3) Functions with arguments and no return values:

Syntax for function definition:

```
void functionName(typel argl, type2 arg2,...... typeN argN)
{
. . . .
. . . .
return;
}
```

Example program:

```
#include<stdio.h>
void myFun(char a[], int b)
{
printf("%s,%d\n”,a,b);
 }
```

```
int main()
 {
myFun ("madhu", 3);
 myFun ("naveen", 14);
myFun ("yashwant", 30);
 return 0;
 }
```

Output:

```
 madhu,3
 Naveen,14
 Yashwant,30
```

4) Functions with arguments and with return values:

Syntax for function definition:

```
returnType functionName(typel argi, type2 arg2,.........typeN argN)
{
. . . .
. . . .
return value;
}
```

Example program:

```
#include<stdio.h>
int myFun(char a[], int b)
```

```
{
printf("%s,%d\n”,a,b);
return 0;
 }
int main()
 {
myFun ("madhu", 3);
 myFun ("naveen", 14);
myFun ("yashwant", 30);
 return 0;
 }
```

Output:

```
 madhu,3



 Naveen,14
 Yashwant,30
```

One function can call any number of functions:

Program:

```
#include <stdio.h>
void india()
```

```
{
    printf("i am in india\n");
}
void hyderabad()
{
    printf("i am in hyderabad\n");
}
void main()
{
    printf("i am in main function\n");
    india();
    hyderabad();
}
```

Output:

```
i am in main function
i am in india
i am in hyderabad
```

C Function Declaration and Definition

A function consist of two parts:

- **Declaration:** the function's name, return type, and parameters (if any)
- **Definition:** the body of the function (code to be executed)

Example program:

```
#include <stdio.h>

void myFunction();// Function declaration

int main()// The main method

{

  myFunction();  // call the function

  return 0;

}

void myFunction()// Function definition

{

  printf("hello!");

}
```

Output:

```
hello!
```

C Function Parameters (or) arguments

Parameters (or) Arguments:

Information can be passed to functions as a parameter. Parameters act as variables inside the function.

Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma:

Syntax:

```
returnType  functionName(parameter1, parameter2, parameter3)

 {
  //code_to_be_executed
}
```

Example:

```
#include <stdio.h>

void myFun(char a[], int b)

 {

  printf("%s,%d\n”,a,b);

}

int main()

 {

  myFun ("madhu", 3);

  myFun ("naveen", 14);

  myFun ("yashwant", 30);

  return 0;

}
```

Output:

```
madhu,3
```

Naveen,14

Yashwant,30

Return Values (or) Return type:

The void keyword, used in the previous examples, indicates that the function should not return a value. If you want the function to return a value, you can use a data type (such as int or float, etc.) instead of void, and use the return keyword inside the function:

Example:

```
#include <stdio.h>

int myfun(int x, int y)

 {

  return x + y;

}

int main()

{

  printf("Result is: %d", myfun(5, 3));

  return 0;

}
```

Output:

```
Result is:8
```

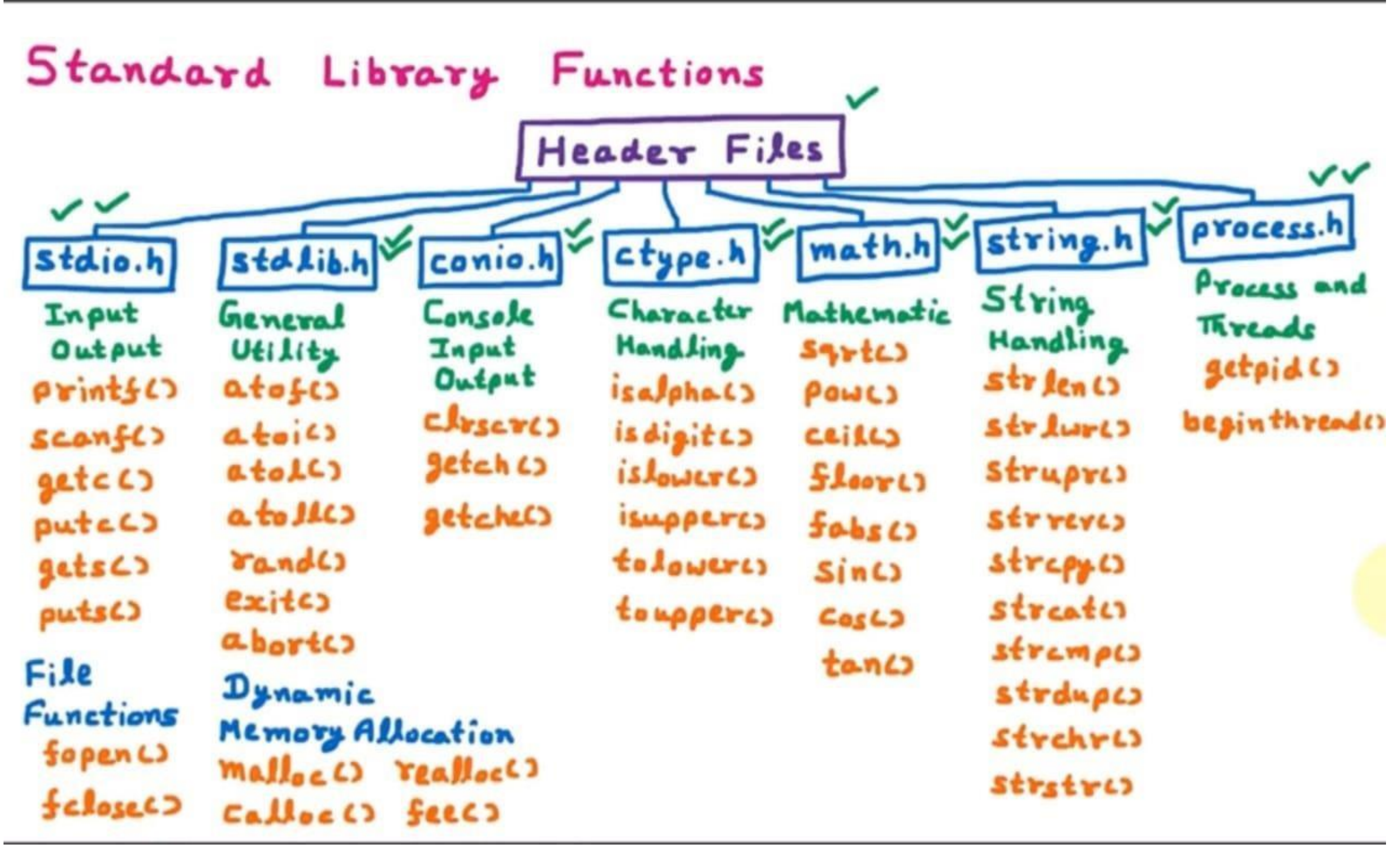
STANDARD LIBRARY FUNCTIONS IN C

Library functions are built-in functions that are grouped together and placed in a common location called library.

Each function here performs a specific operation. We can use this library functions to get the pre-defined output.

All C standard library functions are declared by using many header files. These library functions are created at the time of designing the compilers.

We include the header files in our C program by using **#include<filename.h>**. Whenever the program is run and executed, the related files are included in the C program.



Header File Functions

Some of the header file functions are as follows −

- **stdio.h** − It is a standard i/o header file in which Input/output functions are declared
- **conio.h** − This is a console input/output header file.
- **string.h** − All string related functions are in this header file.
- **stdlib.h** − This file contains common functions which are used in the C programs.
- **math.h** − All functions related to mathematics are in this header file.
- **time.h** − This file contains time and clock related functions.Built functions in stdio.h

Built functions in stdio.h

Let’s see what are the built functions present in stdio.h library function.

Sl.No	Function & Description
1	**printf()**This function is used to print the all char, int, float, string etc., values onto the output screen.
2	**scanf()**This function is used to read data from keyboard.
3	**getc()**It reads character from file.
4	**gets()**It reads line from keyboard.
5	**getchar()**It reads character from keyboard.
6	**puts()**It writes line to o/p screen.
7	**putchar()**It writes a character to screen.
8	**fopen()**All file handling functions are defined in stdio.h header file.

Sl.No	Function & Description
9	**fclose()**Closes an opened file.
10	**getw()**Reads an integer from file.
11	**putw()**Writes an integer to file.
12	**fgetc()**Reads a character from file.
13	**putc()**Writes a character to file.
14	**fputc()**Writes a character to file.
15	**fgets()**Reads string from a file, one line at a time.
16	**f puts()**Writes string to a file.
17	**feof()**Finds end of file.
18	**fgetchar**Reads a character from keyboard.
19	**fgetc()**Reads a character from file.
20	**fprintf()**Writes formatted data to a file.
21	**fscanf()**Reads formatted data from a file.
22	**fputchar**Writes a character from keyboard.

Sl.No	Function & Description
23	**fseek()**Moves file pointer to given location.
24	**SEEK_SET**Moves file pointer at the beginning of the file.
25	**SEEK_CUR**Moves file pointer at given location.
26	**SEEK_END**Moves file pointer at the end of file.
27	**ftell()**Gives current position of file pointer.
28	**rewind()**Moves file pointer to the beginning of the file.
29	**putc()**Writes a character to file.
30	**sprint()**Writes formatted output to string.
31	**sscanf()**Reads formatted input from a string.
32	**remove()**Deletes a file.
33	**flush()**Flushes a file.

Parameter Passing Techniques in C
Or
Inter function communication

There are different ways in which parameter data can be passed into and out of methods and functions. Let us assume that a function *B()* is called from another function *A()*. In this case *A* is called the ***"caller function"*** and *B* is called the ***"called function or callee function"***. Also, the arguments which *A* sends to *B* are called *actual arguments* and the parameters of *B* are called *formal arguments*.

Terminology

- **Formal Parameter :** A variable and its type as they appear in the prototype of the function or method.
- **Actual Parameter :** The variable or expression corresponding to a formal parameter that appears in the function or method call in the calling environment.

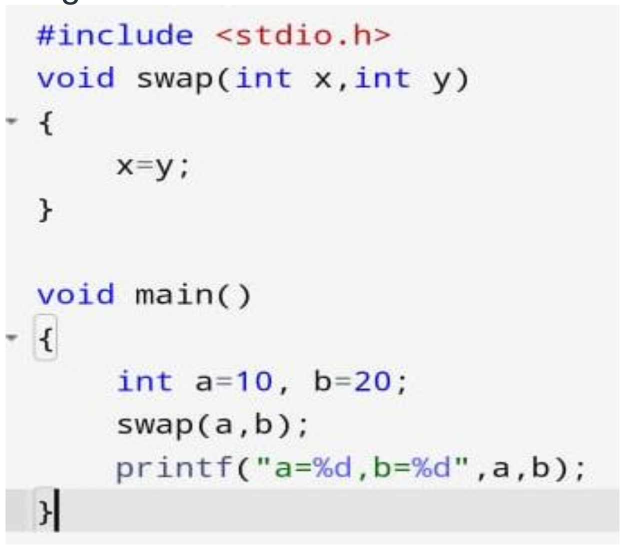
Two methods of Parameter Passing:

1.Pass By Value

Pass by reference

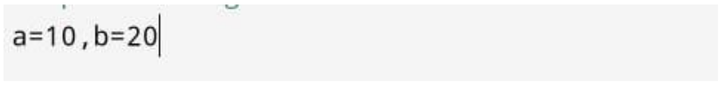
1.Pass By Value: in this method Changes made to formal parameter do not get transmitted back to the formal parameters. Any modifications to the formal parameter variable will not effect the actual parameter. This method is also called as ***call by value***.

Program:



```
#include <stdio.h>
void swap(int x,int y)
{
     x=y;
}

void main()
{
     int a=10, b=20;
     swap(a,b);
     printf("a=%d,b=%d",a,b);
}
```

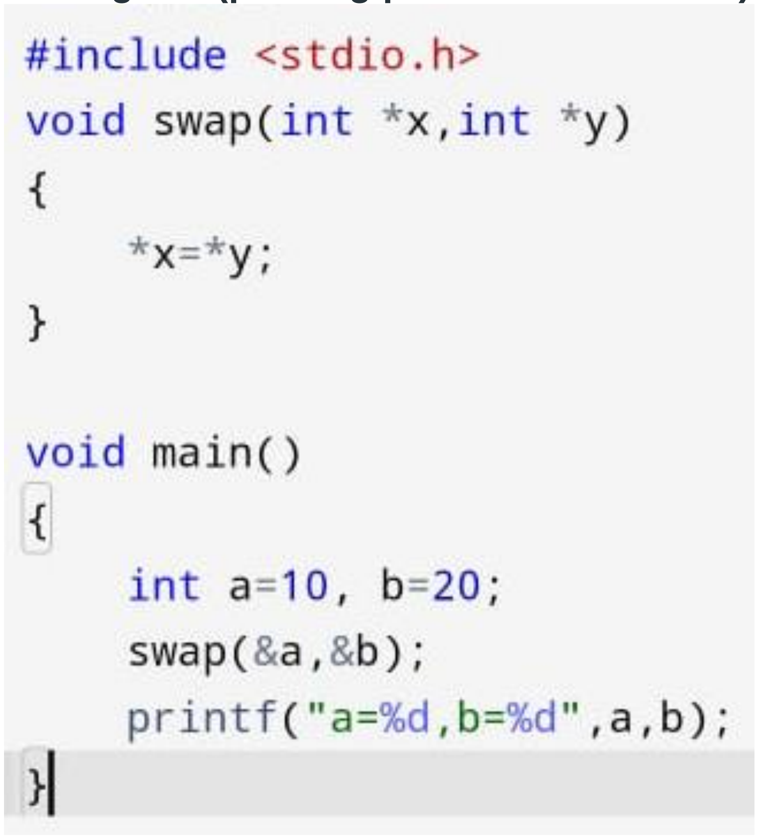
Output:



```
a=10,b=20
```

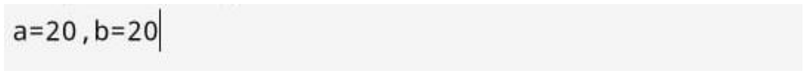
Pass by reference(aliasing): .. Any changes to the formal parameter will effect the actual parameter as formal parameter receives a reference (or pointer). This method is also called as **call by reference**.

Program:(passing pointers to function)



```c
#include <stdio.h>
void swap(int *x,int *y)
{
    *x=*y;
}

void main()
{
    int a=10, b=20;
    swap(&a,&b);
    printf("a=%d,b=%d",a,b);
}
```

Output:



```
a=20,b=20
```

Passing arrays to function

If you want to pass a single-dimension array as an argument in a function, you would have to declare a formal parameter in one of following three ways and all three declaration methods produce similar results because each tells the compiler that an integer pointer is going to be received. Similarly, you can pass multi-dimensional arrays as formal parameters.

Way-1

Formal parameters as a pointer −

```
void myFunction (int *arrayname)
 {
   .
   .
   .
}
```

Way-2

Formal parameters as a sized array −

```
Void    myFunction (int arrayname[10])
 {
   .
   .
   .
}
```

Way-3

Formal parameters as an unsized array −

```
void myFunction (int arrayname[    ])
{
   .
   .
   .
}
```

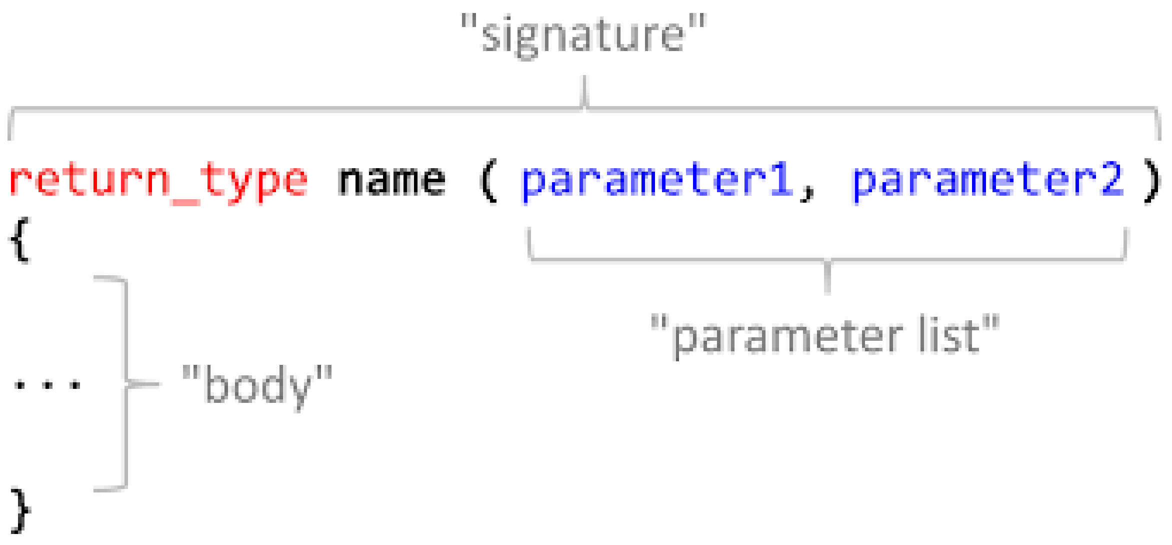
Program:

```
#include<stdio.h>

Void fun(int a[])

{

Printf("%d %d %d %d",a[0],a[1],a[2],a[3]);
}
Void main
{
Int a[]={10,20,30,40};
fun(a);
}
```

Output:

```
10 20 30 40
```

<u>Signature of function</u>



Program:

```
int myFun(char a[  ], int b)
```

```
{
printf("%s,%d\n”,a,b);
return 0;
}
int main()
{
myFun ("madhu", 3);
myFun ("naveen", 14);
myFun ("yashwant", 30);
return 0;
}
```

Output:

```
madhu,3
Naveen,14
Yashwant,30
```

Recursion

Recursion is the technique of making a function call itself. This technique provides a way to break complicated problems down into simple problems which are easier to solve.

Recursion is the process of repeating items in a self-similar way. In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function.

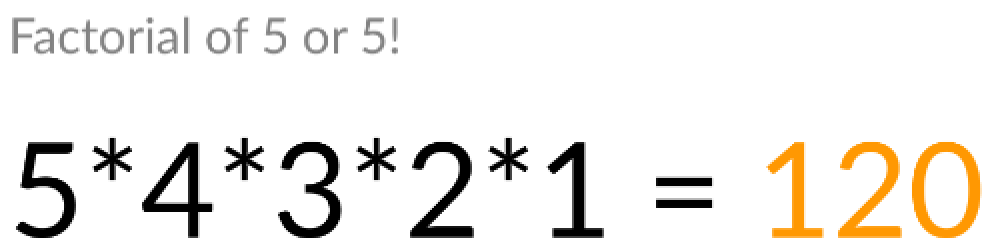
Example:

```
void recursion()
 {
   recursion(); /* function calls itself */
}

void main()
 {
   recursion();
}
```

The C programming language supports recursion, i.e., a function to call itself. But while using recursion, programmers need to be careful to define an exit condition from the function, otherwise it will go into an infinite loop.

Recursive functions are very useful to solve many mathematical problems, such as calculating the factorial of a number, generating Fibonacci series, etc.

Number Factorial:



Factorial of 5 or 5!

5*4*3*2*1 = 120

The following example calculates the factorial of a given number using a recursive function –

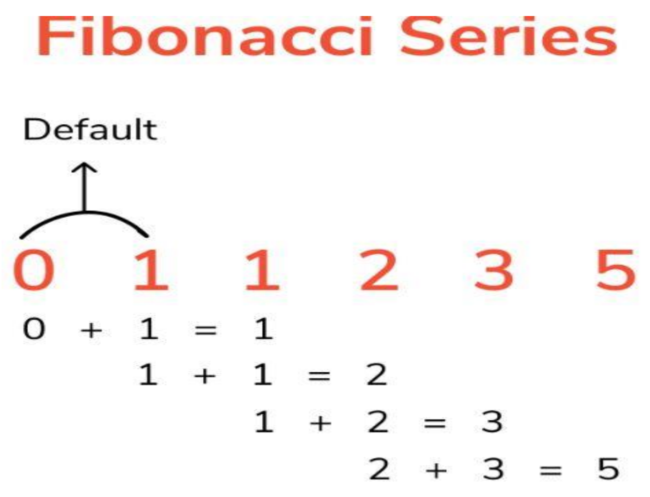
Program:

```
#include <stdio.h>

int factorial(int n)
{

  if(n <= 1)
{
    return 1;
  }
  else  return n * factorial(n - 1);
}

int  main()
 {
  int n = 12;
  printf("Factorial of %d is %d\n",ni, factorial(n));
  return 0;
}
```

Output:

```
Factorial of 12 is 479001600
```

Fibonacci Series:



The following example generates the Fibonacci series for a given number using a recursive function –

Program:

```
#include <stdio.h>

int fibonacci(int i)
 {

   if(i == 0)
 {
      return 0;
   }

   else if(i == 1)
 {
      return 1;
   }
else
return fibonacci(i-1) + fibonacci(i-2);
}

int  main()
{

   int i;

   for (i = 0; i < 10; i++)
 {
      printf("%d\t\n", fibonacci(i));
   }

   return 0;
}
```

Output:

```
0
1
1
2
```

```
3
5
8
13
21
34
```

Disadvantages (or) limitations of Recursion

1)More use of Memory:
As in the process of recursion, the function has to call itself, each other, and add to the stack in each recursive call and they keep their value there until the call is finished. In this process, recursion uses lots of memories.

2)Recursion can be slow:
If it is not implemented properly then it is a very slow process. And it is a very difficult task to write a recursive function with much less speed and low memory. The reason behind its being slow is that it requires the allocation of a new stack frame.

3)Difficult to analyze code:
Recursion is the process of converting a complex problem into less complex but analyzing code is also a complex part. Sometimes it is very difficult to understand code.

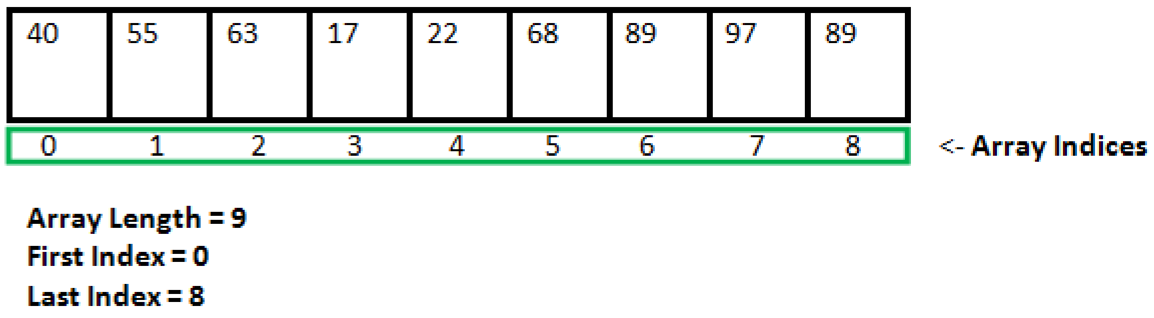
Comparison Table for Advantages and Disadvantages of a Recursion

Advantages	**Disadvantages**
Solve problem which is naturally recursive.	Slower than nonrecursive function

Reduce unnecessary calling of function	Requires lots of memory
Reduce the length of code	Not more effective in terms of space and time
Help in solving data structure problems	Hard to analyze code
Stack evolution and infix	Computer runs out of memory if recursive calls are not properly checked.

Dynamic memory allocation in C

Since C is a structured language, it has some fixed rules for programming. One of them includes changing the size of an array. An array is a collection of items stored at contiguous memory locations.



As it can be seen that the length (size) of the array above made is 9. But what if there is a requirement to change this length (size). For Example,

- If there is a situation where only 5 elements are needed to be entered in this array. In this case, the remaining 4 indices are just wasting memory in this array. So there is a requirement to lessen the length (size) of the array from 9 to 5.
- Take another situation. In this, there is an array of 9 elements with all 9 indices filled. But there is a need to enter 3 more elements in this array. In this case, 3 indices more are required. So the length (size) of the array needs to be changed from 9 to 12.

This procedure is referred to as **Dynamic Memory Allocation in C**. Therefore, C **Dynamic Memory Allocation** can be defined as a procedure in which the size of a data structure (like Array) is changed during the runtime.

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

1. malloc()
2. calloc()

3. realloc()
4. free()

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.

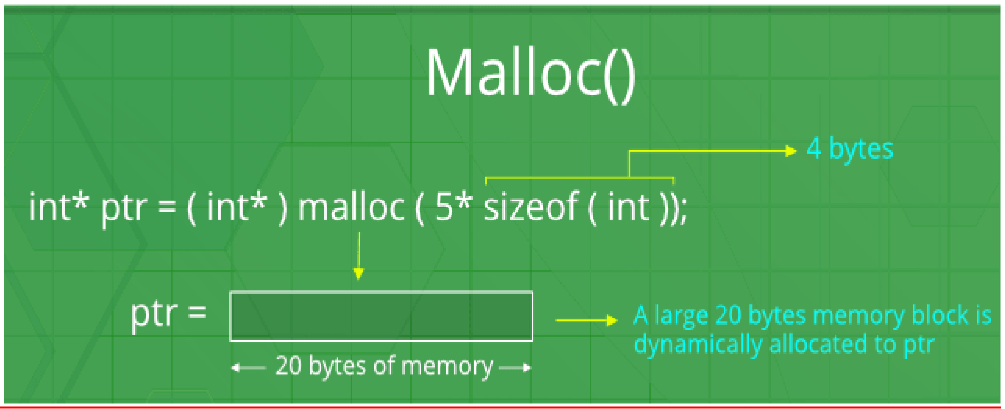
Function	Syntax
malloc()	malloc (number *sizeof(int));
calloc()	calloc (number, sizeof(int));
realloc()	realloc (pointer_name, number * sizeof(int));
free()	free (pointer_name);

Malloc:

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.


Malloc()
4 bytes
int* ptr = (int*) malloc (5* sizeof (int));
ptr =
20 bytes of memory
A large 20 bytes memory block is dynamically allocated to ptr

malloc Example program:

```
#include<stdio.h>

#include<string.h>

#include<stdlib.h>

void main()

{

  char *p;

  //memory allocated dynamically

p = malloc( 15 * sizeof(char) );

  if(p== NULL )

 {

 printf("Couldn't able to allocate memory\n");

 }

else
```

```
 {
strcpy( p,"nagendra");
}
 printf("Dynamically allocated memory content : %s\n", p );
free(p);
}
```

Output:

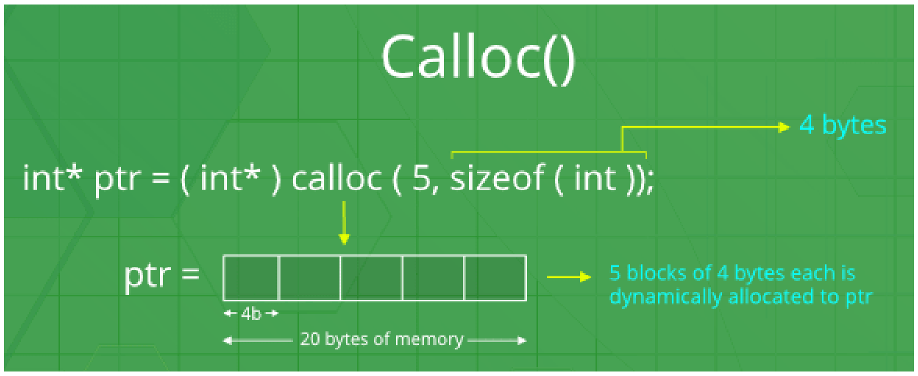
```
Dynamically allocated memory content : nagendra
```

Calloc:

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

It initially initialize all bytes to zero.

It returns NULL if memory is not sufficient.


Calloc()
4 bytes
int* ptr = (int*) calloc (5, sizeof (int));
ptr =
4b
20 bytes of memory
5 blocks of 4 bytes each is dynamically allocated to ptr

Calloc example program:

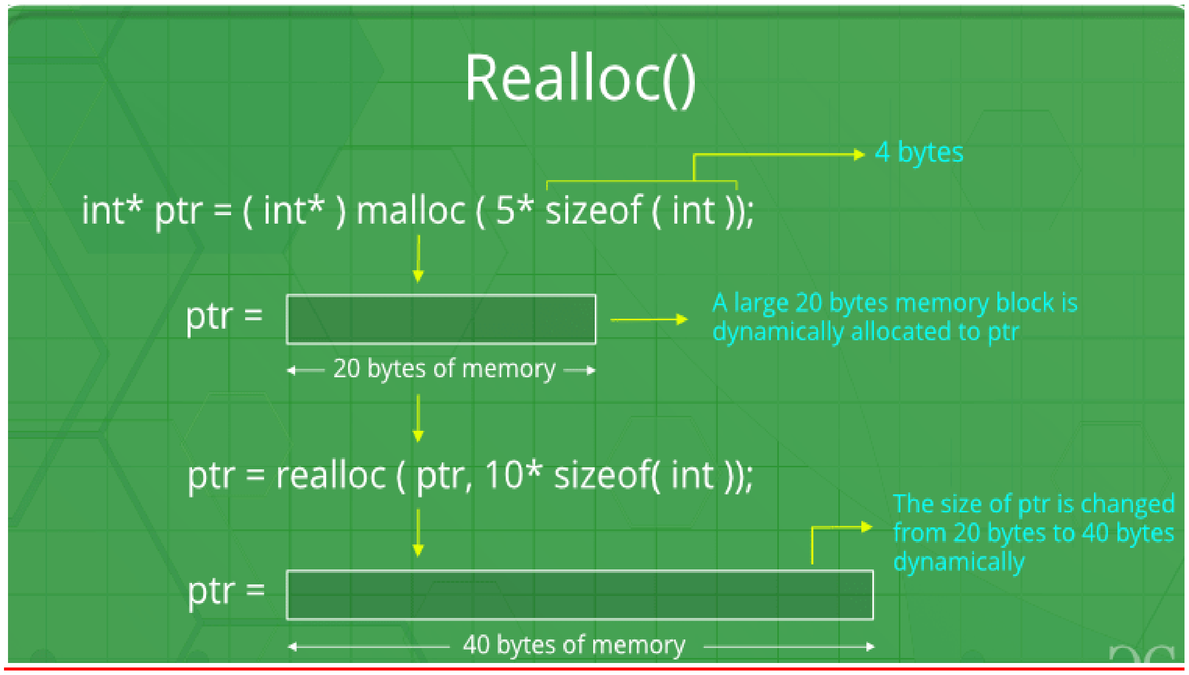
```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

void main()
{
   char *p;
   //memory allocated dynamically
   p = calloc( 15 , sizeof(char) );

   if(p== NULL )
   {
      printf("Couldn't able to allocate memory\n");
   }
   else
   {
      strcpy( p,"nagendra");
   }

   printf("Dynamically allocated memory content : %s\n", p );
   free(p);
}
```

Output:

```
Dynamically allocated memory content : nagendra
```

Realloc:

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


Realloc()
4 bytes
int* ptr = (int*) malloc (5* sizeof (int));
ptr =
A large 20 bytes memory block is dynamically allocated to ptr
20 bytes of memory
ptr = realloc (ptr, 10* sizeof(int));
The size of ptr is changed from 20 bytes to 40 bytes dynamically
ptr =
40 bytes of memory

Example program for realloc:

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

void main()
{
 char *p;
   //memory allocated dynamically
   p = calloc( 15 , sizeof(char) );
   p=realloc(p,100*sizeof(char));

 if(p== NULL )
   {
      printf("Couldn't able to allocate memory\n");
   }
 else
     {
     strcpy( p,"nagendra");
     }
```

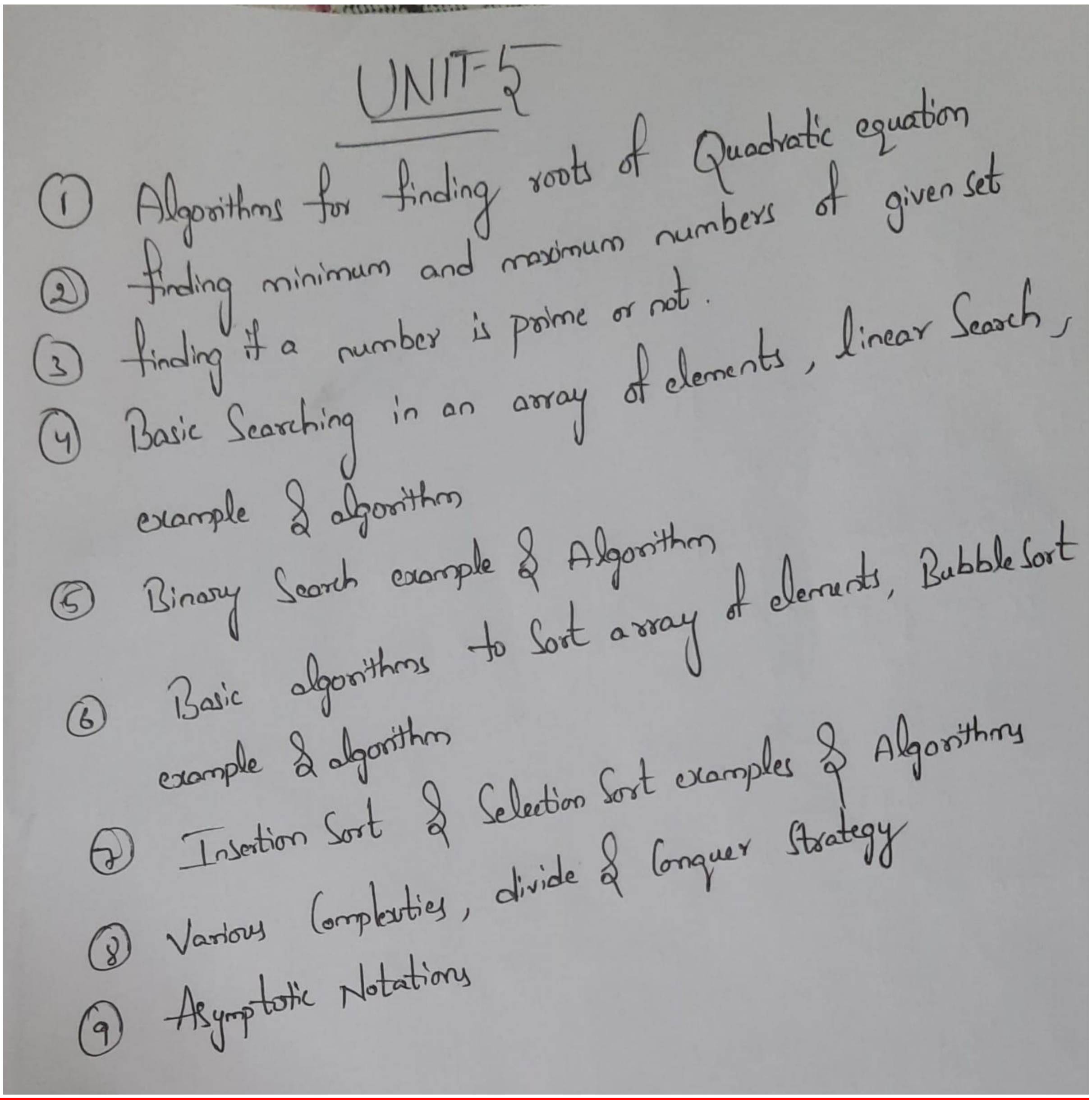
```
    printf("Dynamically allocated memory content : %s\n", p );
    free(p);
}
```

Output:

```
Dynamically allocated memory content : nagendra
```

free() :

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



UNIT-5

1. Algorithms for finding roots of Quadratic equation
2. finding minimum and maximum numbers of given set
3. finding if a number is prime or not.
4. Basic Searching in an array of elements, linear Search, example & algorithm
5. Binary Search example & Algorithm
6. Basic algorithms to Sort array of elements, Bubble Sort example & algorithm
7. Insertion Sort & Selection Sort examples & Algorithms
8. Various Complexities, divide & Conquer Strategy
9. Asymptotic Notations

Algorithm for factorial of given number:

Factorial of a positive integer n is product of all values from n to 1. For example, the factorial of 3 is (3 * 2 * 1 = 6).

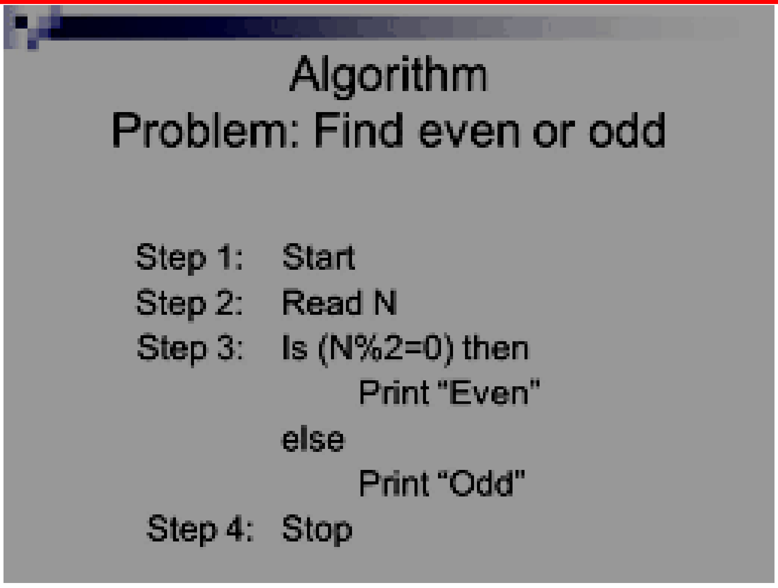
Algorithm

Algorithm of this program is very easy −

```
START
   Step 1 → Take integer variable A
   Step 2 → Assign value to the variable
   Step 3 → From value A upto 1 multiply each digit and store
```

Step 4 → the final stored value is factorial of A
STOP

Algorithm to check number is even or odd



Algorithm
Problem: Find even or odd

```
Step 1:   Start
Step 2:   Read N
Step 3:   Is (N%2=0) then
                Print "Even"
          else
                Print "Odd"
Step 4:   Stop
```

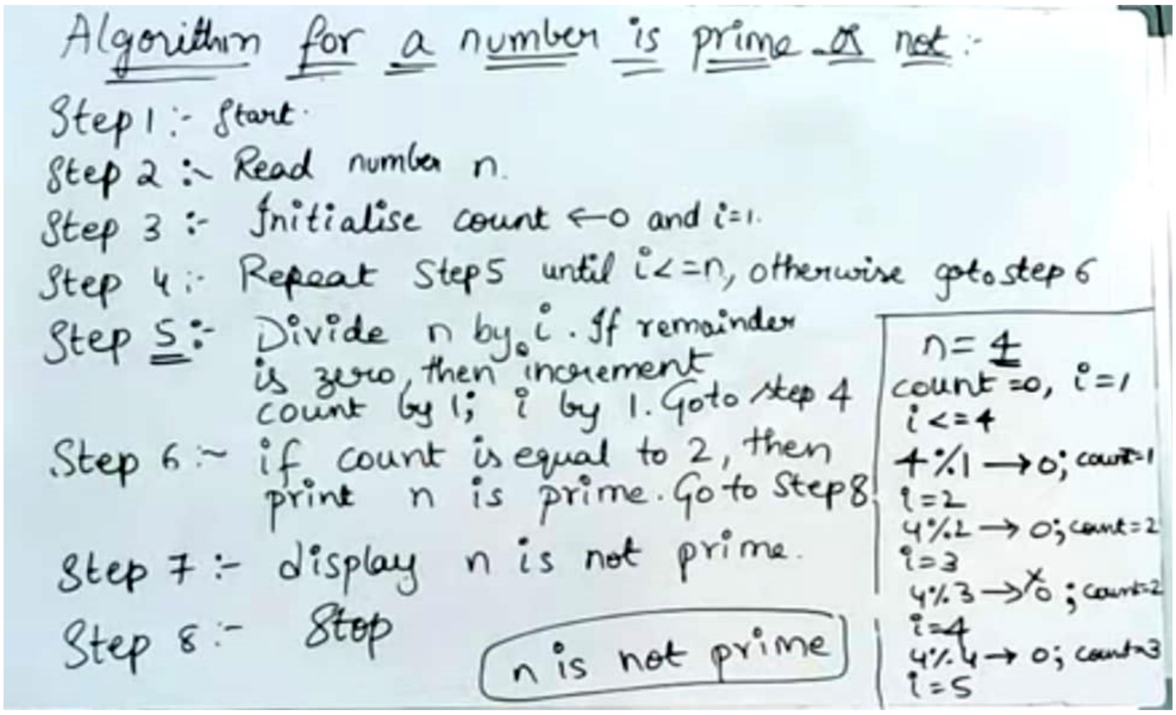
Algorithm to check number is prime or not:

A number that is divisible only by 1 and itself is called a prime number.. For example −

7 = 1 × 7

Few prime number are − 1, 2, 3, 5 , 7, 11 etc.

Algorithm:



Algorithm for a number is prime ~~or~~ not :-

```
Step 1 :- Start.
Step 2 :- Read number n.
Step 3 :- Initialise count ← 0 and i=1.
Step 4 :- Repeat Step5 until i<=n, otherwise goto step 6
Step 5 :- Divide n by i. If remainder
          is zero, then increment
          count by 1; i by 1. Goto step 4
Step 6 :- if count is equal to 2, then
          print n is prime. Goto Step8
Step 7 :- display n is not prime.
Step 8 :- Stop
```

$n = 4$
count $=0$, $i=1$
$i<=4$
$4\%1 \rightarrow 0$; count=1
$i=2$
$4\%2 \rightarrow 0$; count=2
$i=3$
$4\%3 \rightarrow \cancel{0}$; count=2
$i=4$
$4\%4 \rightarrow 0$; count=3
$i=5$

$\boxed{\text{n is not prime}}$

Algorithm, flowchart and C program to find the roots of a quadratic equation:

Analysis

Input − a,b,c values

Output − r1, r2 values

Procedure

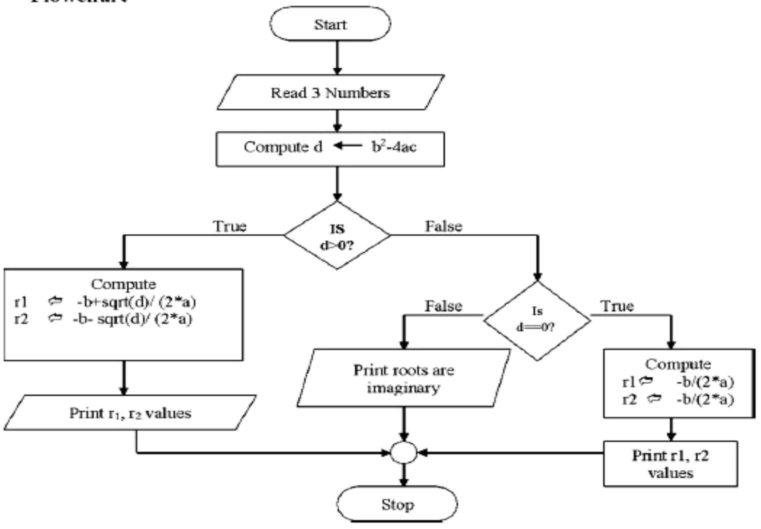
$$r_1 = \frac{-b+\sqrt{b^2-4ac}}{2a}$$

$$r_2 = \frac{-b-\sqrt{b^2-4ac}}{2a}$$

Algorithm:

- Start
- Read a, b, c values
- Compute d = b*b-4ac
- if d > 0 then
    - r1 = -b+ sqrt(d)/(2*a)
    - r2 = -b-sqrt(d)/(2*a)
- Otherwise if d = 0 then
    - compute r1 = -b/2a, r2=-b/2a
    - print r1,r2 values
- Otherwise if d < 0 then print roots are imaginary
- Stop

Flowchart



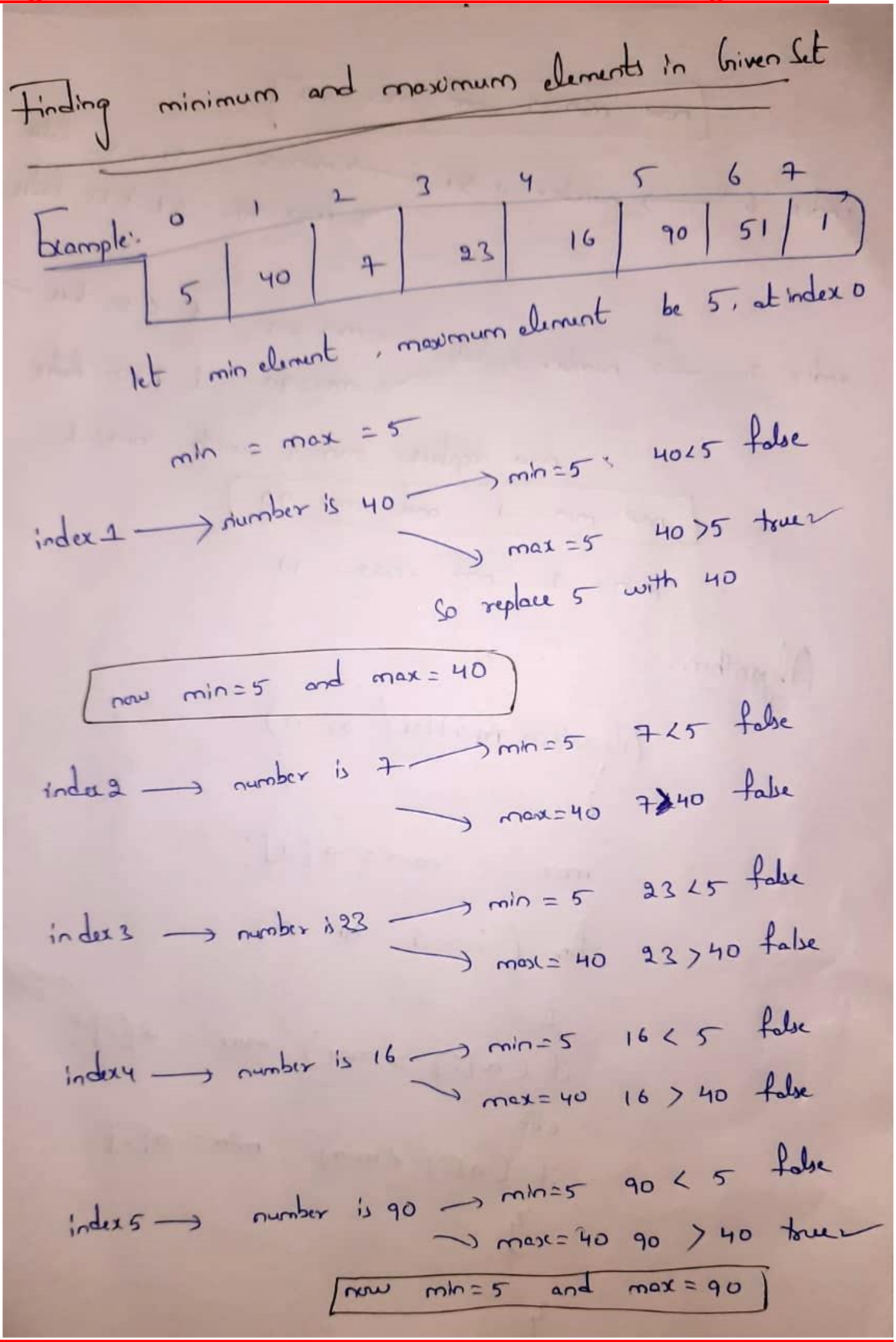
Program:

```
# include<stdio.h>
# include<conio.h>
# include<math.h>
main (){
  float a,b,c,r1,r2,d;
  printf (“enter the values of a b c”);
  scanf (“ %f %f %f”, &a, &b, &c);
  d= b*b – 4*a*c;
  if (d>0)
{
    r1 = -b+sqrt (d) / (2*a);
    r2 = -b-sqrt (d) / (2*a);
    printf (“The real roots = %f %f”, r1, r2);
  }
  else if (d= =0)
{
    r1 = -b/(2*a);
    r2 = -b/(2*a);
    printf (“roots are equal =%f %f”, r1, r2);
  }
  else
    printf(“Roots are imaginary”);
  getch ();
}
```

Testing:

```
Case 1: enter the values of a b c: 1 4 3
  r1 = -1
  r2 = -3
```

Algorithm for find minimum and maximum element in given list



Finding minimum and maximum elements in Given Set

Example:

0	1	2	3	4	5	6	7
5	40	7	23	16	90	51	1

let min element, maximum element be 5, at index 0

min = max = 5

index 1 → number is 40 → min = 5 : 40 < 5 false
→ max = 5   40 > 5 true
So replace 5 with 40

now min = 5 and max = 40

index 2 → number is 7 → min = 5   7 < 5 false
→ max = 40   7 > 40 false

index 3 → number is 23 → min = 5   23 < 5 false
→ max = 40   23 > 40 false

index 4 → number is 16 → min = 5   16 < 5 false
→ max = 40   16 > 40 false

index 5 → number is 90 → min = 5   90 < 5 false
→ max = 40   90 > 40 true

now min = 5 and max = 90

$\boxed{\text{now } min = 5 \text{ and } max = 90}$

$$\text{index } 6 \longrightarrow \text{number is } 51 \begin{cases} \longrightarrow min = 5 & 51 < 5 \text{ false} \\ \searrow max = 90 & 51 > 90 \text{ false} \end{cases}$$

$$\text{index } 7 \longrightarrow \text{number is } \underline{1} \begin{cases} \longrightarrow min = 5 & \underline{1} < 5 \text{ true } ✓ \\ \searrow max = 90 & \underline{1} > 90 \text{ false} \end{cases}$$

now replace $min = 5$ with $min = \underline{1.}$

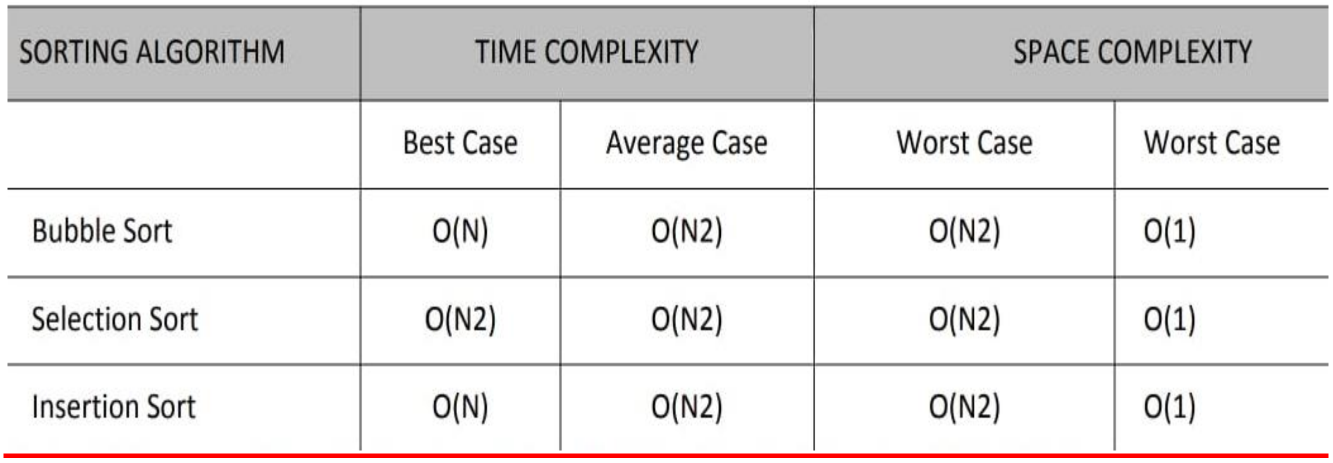
$\boxed{\text{now } min = 1 \text{ and } max = 90}$

$\therefore$ $min = 1$ and $max = 90$

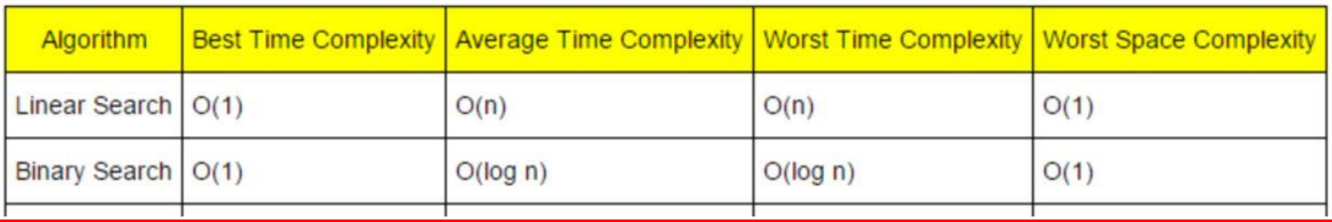
<u>Algorithm:</u>

```
Algorithm minMax( a, n)
  {
  max = min = a[0];
  for ( i = 1 to n)
    {
    if (a[i] > max)  max = a[i];
    else
    if (a[i] < min)  min = a[i];
    }
  }
```

Various complexities



SORTING ALGORITHM	TIME COMPLEXITY		SPACE COMPLEXITY	
	Best Case	Average Case	Worst Case	Worst Case
Bubble Sort	O(N)	O(N2)	O(N2)	O(1)
Selection Sort	O(N2)	O(N2)	O(N2)	O(1)
Insertion Sort	O(N)	O(N2)	O(N2)	O(1)



Algorithm	Best Time Complexity	Average Time Complexity	Worst Time Complexity	Worst Space Complexity
Linear Search	O(1)	O(n)	O(n)	O(1)
Binary Search	O(1)	O(log n)	O(log n)	O(1)

Asymptotic notations

Asymptotic Notations

By using Asymptotic Notations we can calculate time Complexity of an algorithm.
By using Asymptotic Notations we can calculate Best case time Complexity, Average Case time Complexity, worst Case time Complexity of an algorithm.

There are 5 types of Asymptotic notations.

1. Big oh Notation (O)
2. Big Omega Notations (Ω)
3. theta Notation (Θ)
4. little oh Notation (o)
5. little omega Notation (ω)

1. Big oh Notation: (O)

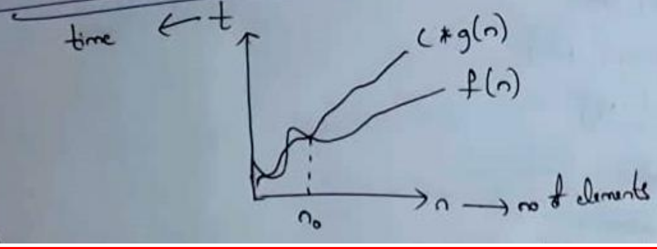
Big oh Notation: (O): Upper Bound of Algorithms run time.

→ It mainly represents upper Bound of Algorithms run time.

→ ~~It ma~~ By using Big oh Notation we can calculate maximum amount of time ie, worst Case Time Complexity of an algorithm that an algorithm requires for its execution.

Definition: let $f(n)$, $g(n)$ be two non negative function then $f(n) = O \cdot g(n)$ if there exist two positive Constants C, n_0 such that $f(n) \leq C * g(n)$, $\forall\ n > n_0$

Graphical Representation of Big oh Notation:



Big Omega Notations (Ω) :

⇒ It is mainly represents lower bound of Algorithm run time

⇒ By using Big omega Notations, we can calculate minimum amount of time that an algorithm requires for its execution i.e Best case time complexity of an algorithm.

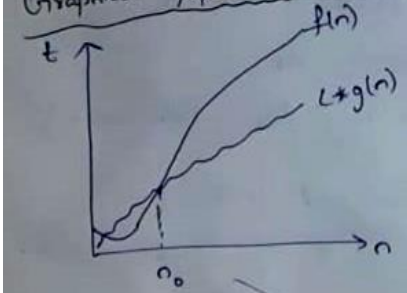
[redacted]

[redacted]

Definition: Let $f(n)$, $g(n)$ be two non negative functions, then $f(n) = \Omega(g(n))$ if there exist two positive constants C, n_0 such that $f(n) \geq C * g(n)$, $\forall\ n > n_0$

Graphical Representation of Big omega Notation:

[redacted]



[redacted]

Theta Notation (Θ):

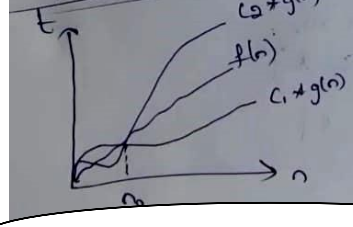
It mainly represents Average Bound of Algorithm's Run time.

→ By using theta notation we can calculate Average amount of time that an algorithm requires for its Execution i.e, Average Case time Complexity of an algorith

Definition:

Let $f(n)$, $g(n)$ be two non negative functions then $f(n) = \Theta(g(n))$ if there exist 3 +ve constants C_1, C_2, n_0 Such that $\boxed{C_1 * g(n) \le f(n) \le C_2 * g(n)}$ $\forall\ n \ge n_0$.

Graphical Representation of theta notation:



④ little oh (o):

Definition: Let $f(n)$, $g(n)$ be two non negative functions then $f(n) = o(g(n))$ Such that

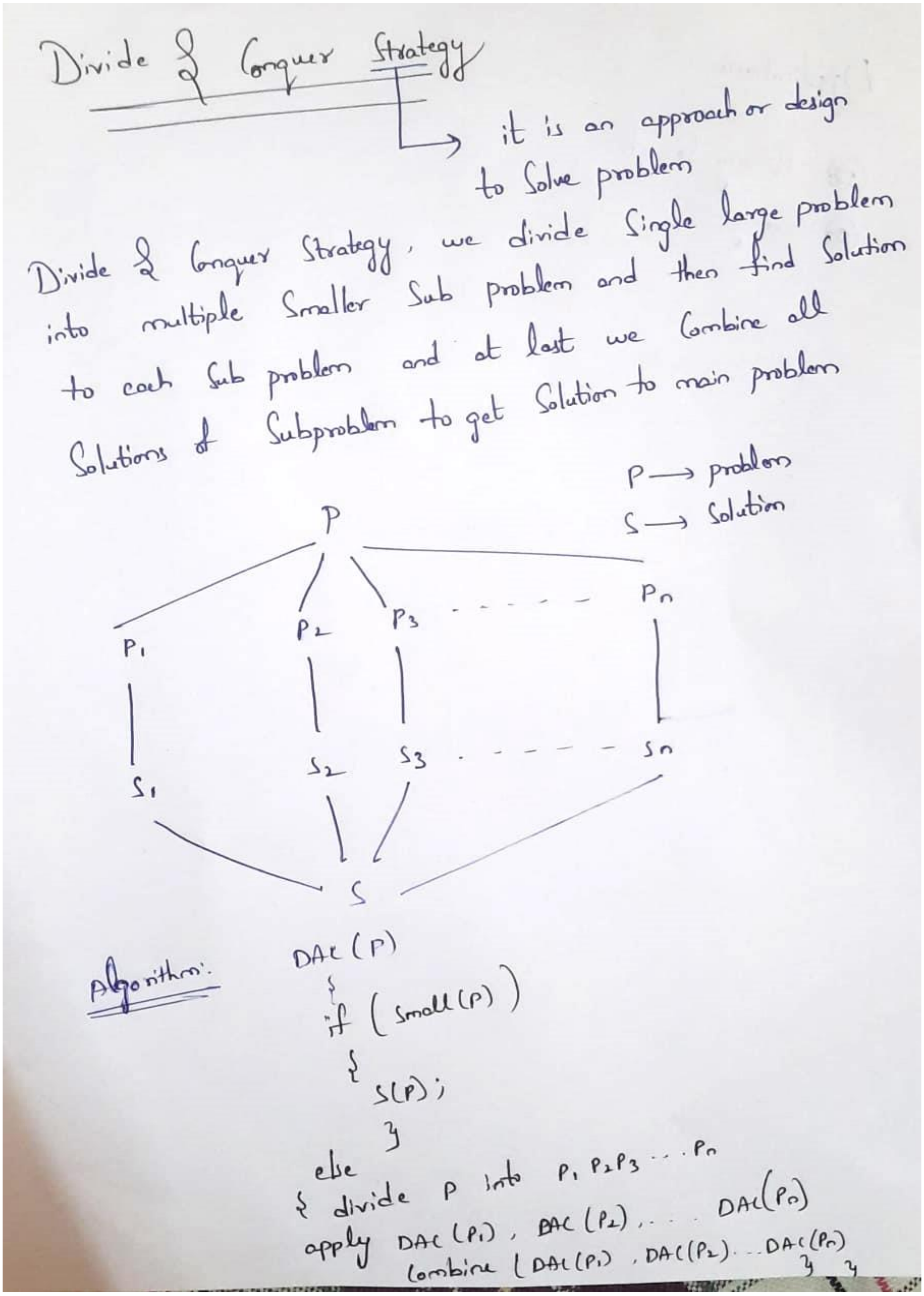
$$\boxed{\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0}$$

⑤ little omega (ω)

Definition: Let $f(n)$, $g(n)$ be two non negative functions then $f(n) = \omega(g(n))$ Such that

$$\boxed{\lim_{n \to \infty} \frac{g(n)}{f(n)} = 0}$$

DIVIDE AND CONQUER STRATEGY



Divide & Conquer Strategy

↳ it is an approach or design to Solve problem

Divide & Conquer Strategy, we divide Single large problem into multiple Smaller Sub problem and then find Solution to each Sub problem and at last we Combine all Solutions of Subproblem to get Solution to main problem

P ⟶ problem

S ⟶ Solution

$$\begin{tikzcd}
 & & P \arrow[dll, dash] \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & & \\
P_1 \arrow[d, dash] & P_2 \arrow[d, dash] & P_3 \arrow[d, dash] & \cdots & P_n \arrow[d, dash] \\
S_1 \arrow[drr, dash] & S_2 \arrow[dr, dash] & S_3 \arrow[d, dash] & \cdots & S_n \arrow[dll, dash] \\
 & & S & &
\end{tikzcd}$$

Algorithm:

```
DAC (P)
{
 if ( small(P) )
  {
   S(P);
  }
 else
 { divide P into P1 P2 P3 ... Pn
   apply DAC (P1), DAC (P2), ...  DAC(Pn)
   combine ( DAC(P1) , DAC(P2) ... DAC(Pn)
 } }
```

Searching techniques

Searching refers to the process of finding a desired element in set of items. The desired element is called "target".

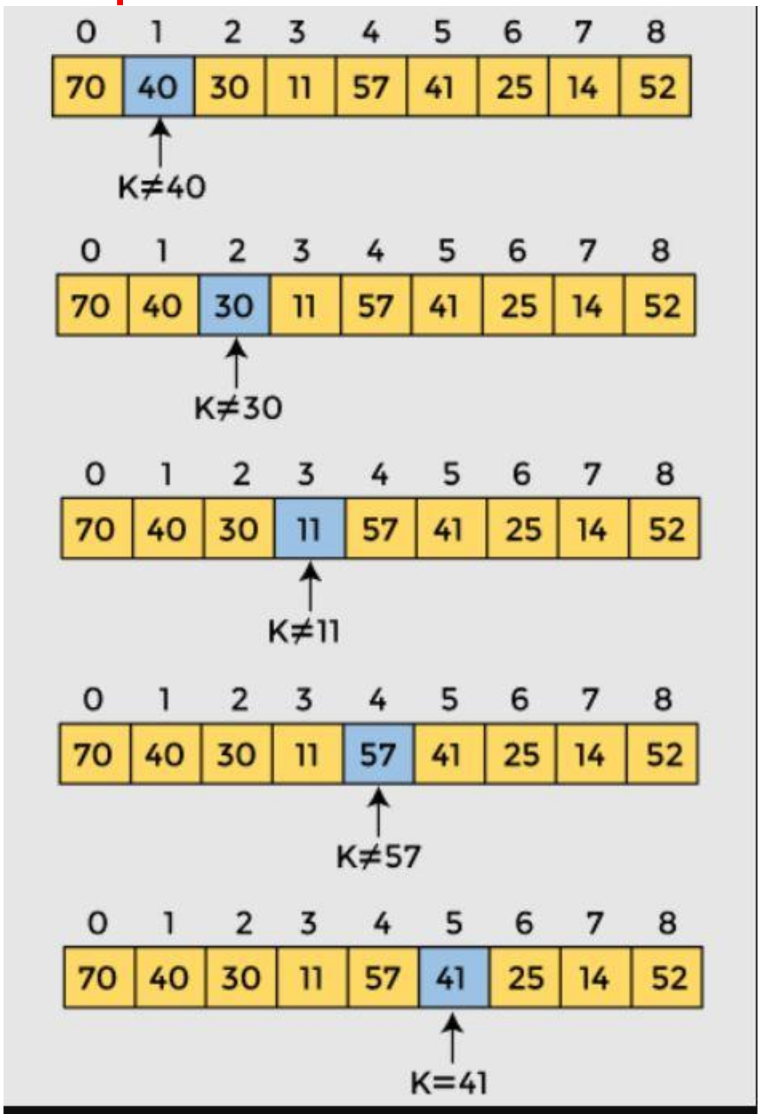
Searching Algorithms :

1. Linear Search
2. Binary Search

Linear search

Linear search is a very simple search algorithm. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection.

Example:



Algorithm:

```
Linear Search ( Array Arr, Value a ) // Arr is the name of the array, and a
is the searched element.

Step 1: Set i to 0 // i is the index of an array which starts from 0

Step 2: if i > n then go to step 7 // n is the number of elements in array

Step 3: if Arr[i] = a then go to step 6

Step 4: Set i to i + 1

Step 5: Goto step 2

Step 6: Print element a found at index i and go to step 8

Step 7: Print element not found

Step 8: Exit
```

Binary search

Binary Search:

⇒ To overcome the limitations of linear Search, new Searching technique is developed i.e binary Search to Save time

⇒ Binary Search is a fast Searching algorithm with runtime Complexity of $O(\log n)$

⇒ It is based on divide and Conquer Strategy

⇒ In Binary Search technique, we Search an element in a Sorted array

1	2	3	4	5	6

$\longrightarrow$ Sorted Array

Example: 11 24 55 60 71 77 80

find Value 71 ?

$$\begin{array}{ccccccc} 11 & 24 & 55 & 60 & 71 & 77 & 80 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \uparrow & & & \uparrow & & & \uparrow \\ l & & & m & & & h \end{array}$$

$$\text{mid} = \frac{l+h}{2} = \frac{1+7}{2} = \frac{8}{2} = 4$$

$\boxed{\text{mid} = 4}$

value = mid

$71 \neq 60 \Rightarrow 71 > 60$ So Convert mid to lower

$$\begin{array}{ccccccc} 11 & 24 & 55 & 60 & 71 & 77 & 80 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ & & & \uparrow & & & | \\ & & & l & & & h \end{array}$$

$$mid = \frac{l+h}{2} = \frac{4+7}{2} = \overset{5}{\cancel{11}}/\cancel{2} = 5$$

$$Value = mid \Rightarrow 71 = 71$$

$$\begin{array}{ccccccc} 11 & 24 & 55 & 60 & \boxed{71} & 77 & 80 \\ 1 & 2 & 3 & 4 & \boxed{5} & 6 & 7 \\ & & & | & \downarrow & & | \\ & & & l & mid & & h \end{array}$$

$$\therefore Value = mid = 71$$

<u>Example - 2</u> find 24 ?

$$\begin{array}{ccccccc} 11 & 24 & 55 & 60 & 71 & 77 & 8 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \uparrow & \uparrow & & \uparrow & & & \uparrow \\ l & & & m & & & h \end{array}$$

$$mid = \frac{l+h}{2} = \frac{1+7}{2} = \frac{8}{2} = 4$$

$24 \neq 60$ $24 < 60 \rightarrow$ change mid to high

$$\begin{array}{ccccccc} 11 & 24 & 55 & 60 & 71 & 77 & 8 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \uparrow & \uparrow & & \uparrow & & & \\ l & mid & & h & & & \end{array}$$

$$mid = \frac{l+h}{2} = \frac{1+4}{2} = 5/2 = 2.$$

$Value = 24$ $mid = 24$

$\therefore value = mid . = 24$

Example-3   find 23 ?

```
11   24   55   60   71   77   80
1    2    3    4    5    6    7
↑              ↑              ↑
l              mid            h
```

$mid = \frac{l+h}{2} = \frac{1+7}{2} = \frac{8}{2} = 4$

Value = mid
23   60

23 < 60

~~23<~~ if value is less than mid, change mid to higher

```
11   24   55   60   71   77   80
1    2    3    4    5    6    7
↑              ↑
l              h
```

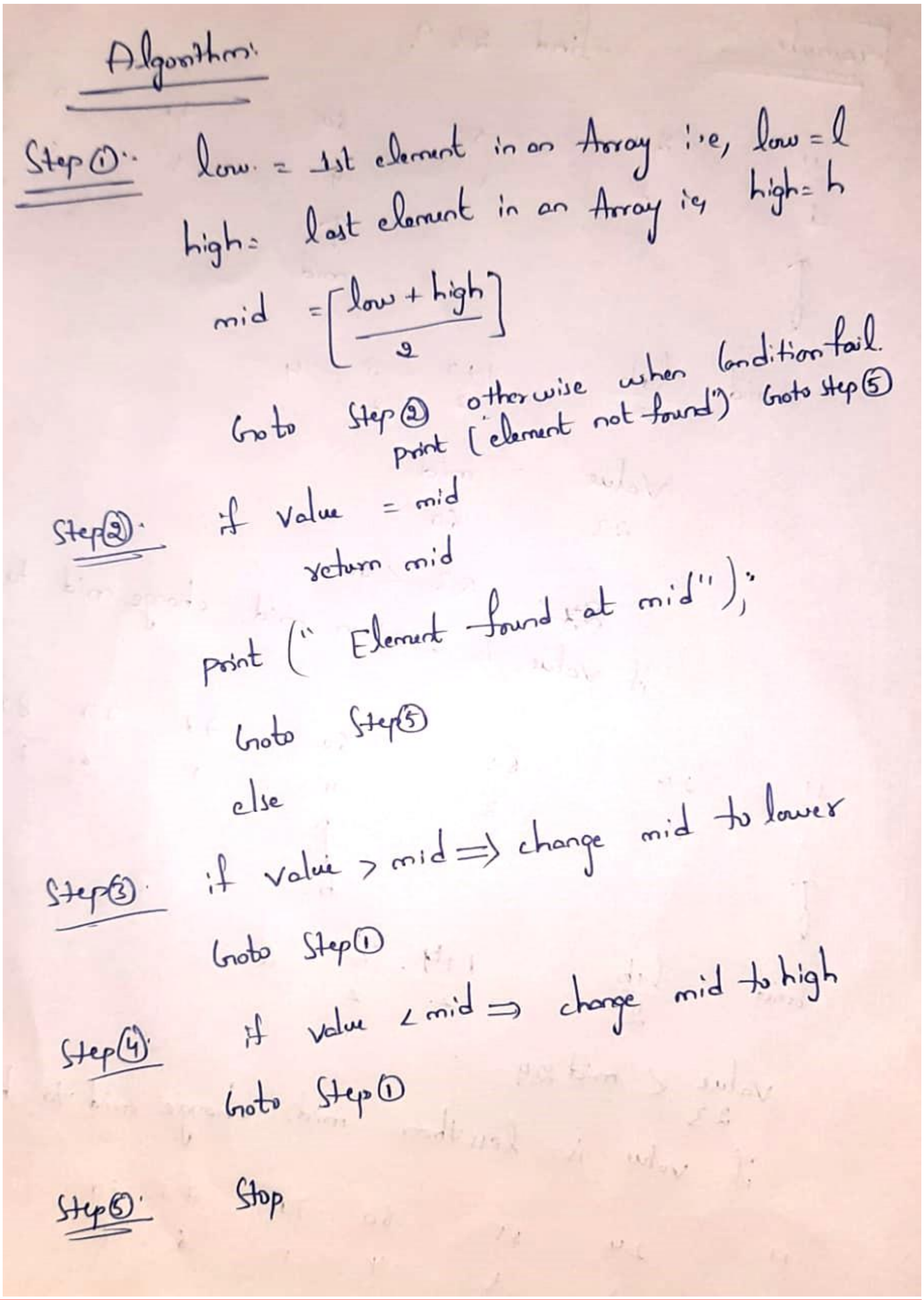
$mid = \frac{l+h}{2} = \frac{1+4}{2} = \overset{2.5}{\frac{5}{2}} = 2.$

Value < mid 24
23

if value is less than mid, change mid to higher

```
11   24   55   60   71   77   80
1    2    3    4    5    6    7
↑    ↑
l    h
```

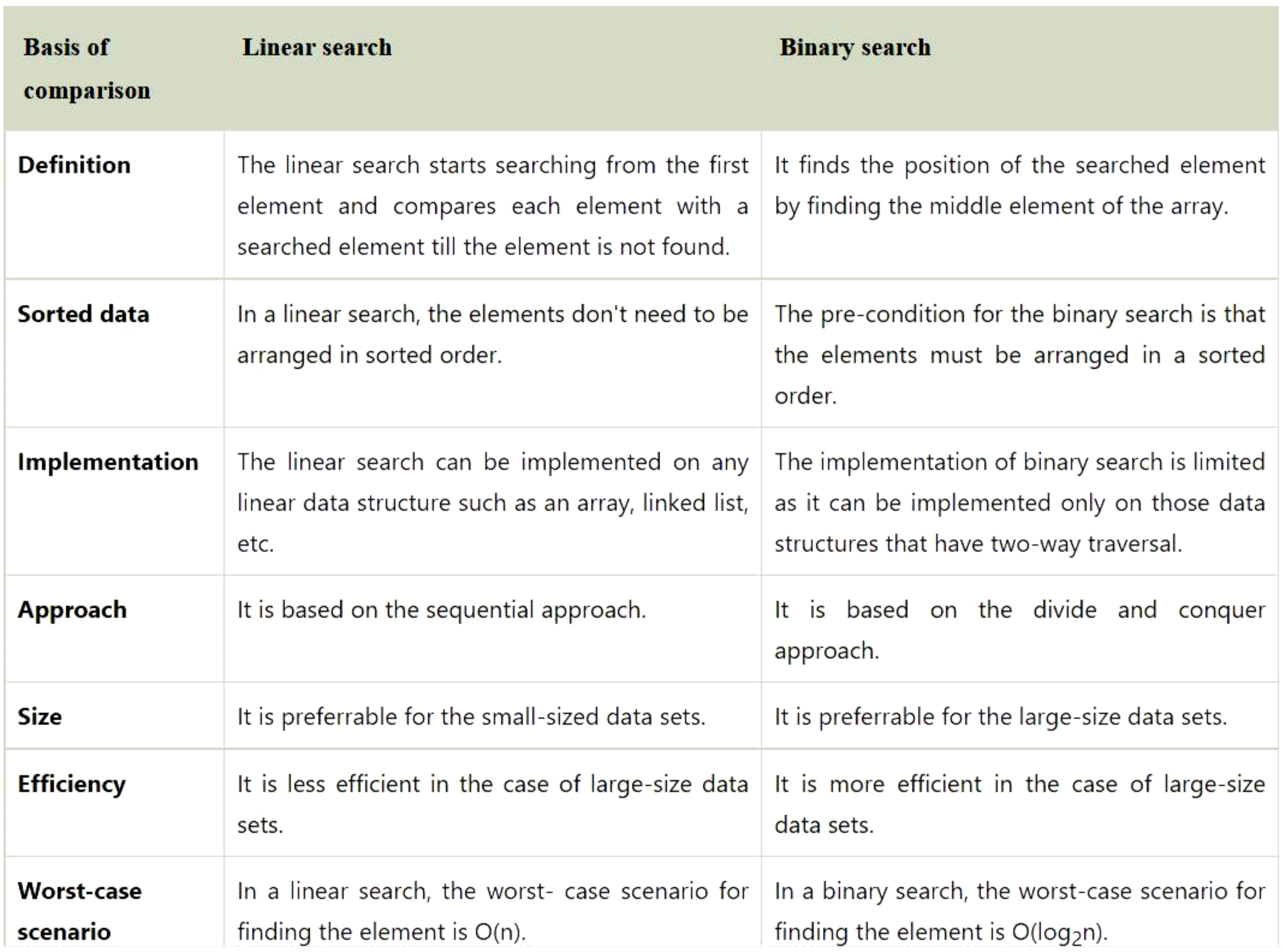
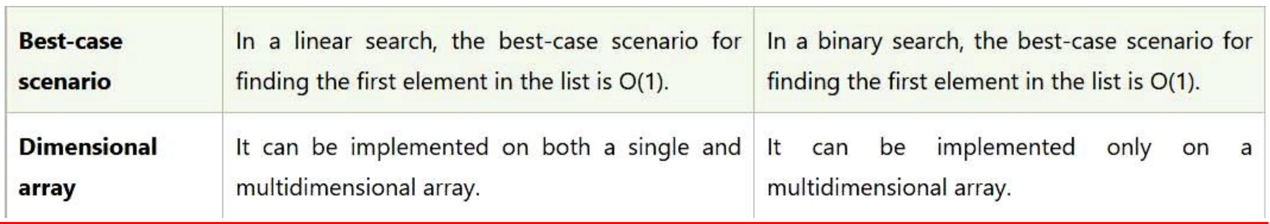
lower element & higher element both are side by side
∴ 23 element not found



Algorithm:

```
Step ①: low = 1st element in an Array i.e, low = l
        high = last element in an Array i.e, high = h
        mid = [ (low + high) / 2 ]
        Goto Step ② otherwise when condition fail.
                  print ('element not found') Goto step ⑤

Step ②: if value = mid
            return mid
        print (" Element found at mid");
        Goto Step ⑤
        else

Step ③: if value > mid ⇒ change mid to lower
        Goto Step ①

Step ④: if value < mid ⇒ change mid to high
        Goto Step ①

Step ⑤: Stop
```

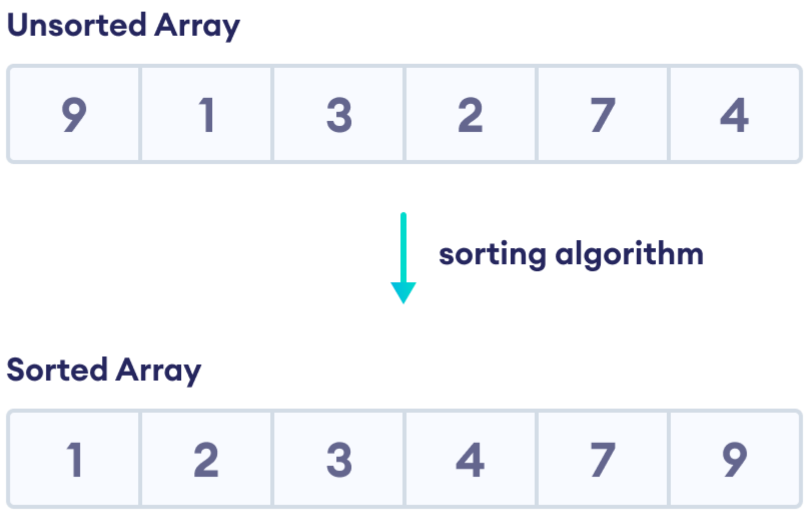
Difference between linear and binary search

Basis of comparison	Linear search	Binary search
Definition	The linear search starts searching from the first element and compares each element with a searched element till the element is not found.	It finds the position of the searched element by finding the middle element of the array.
Sorted data	In a linear search, the elements don't need to be arranged in sorted order.	The pre-condition for the binary search is that the elements must be arranged in a sorted order.
Implementation	The linear search can be implemented on any linear data structure such as an array, linked list, etc.	The implementation of binary search is limited as it can be implemented only on those data structures that have two-way traversal.
Approach	It is based on the sequential approach.	It is based on the divide and conquer approach.
Size	It is preferrable for the small-sized data sets.	It is preferrable for the large-size data sets.
Efficiency	It is less efficient in the case of large-size data sets.	It is more efficient in the case of large-size data sets.
Worst-case scenario	In a linear search, the worst- case scenario for finding the element is O(n).	In a binary search, the worst-case scenario for finding the element is O($\log_2 n$).
Best-case scenario	In a linear search, the best-case scenario for finding the first element in the list is O(1).	In a binary search, the best-case scenario for finding the first element in the list is O(1).
Dimensional array	It can be implemented on both a single and multidimensional array.	It can be implemented only on a multidimensional array.

Sorting

A sorting algorithm is used to arrange elements of an array/list in a specific order. For example,



Sorting an array

Here, we are sorting the array in ascending order.

There are various sorting algorithms that can be used to complete this operation. And, we can use any algorithm based on the requirement.

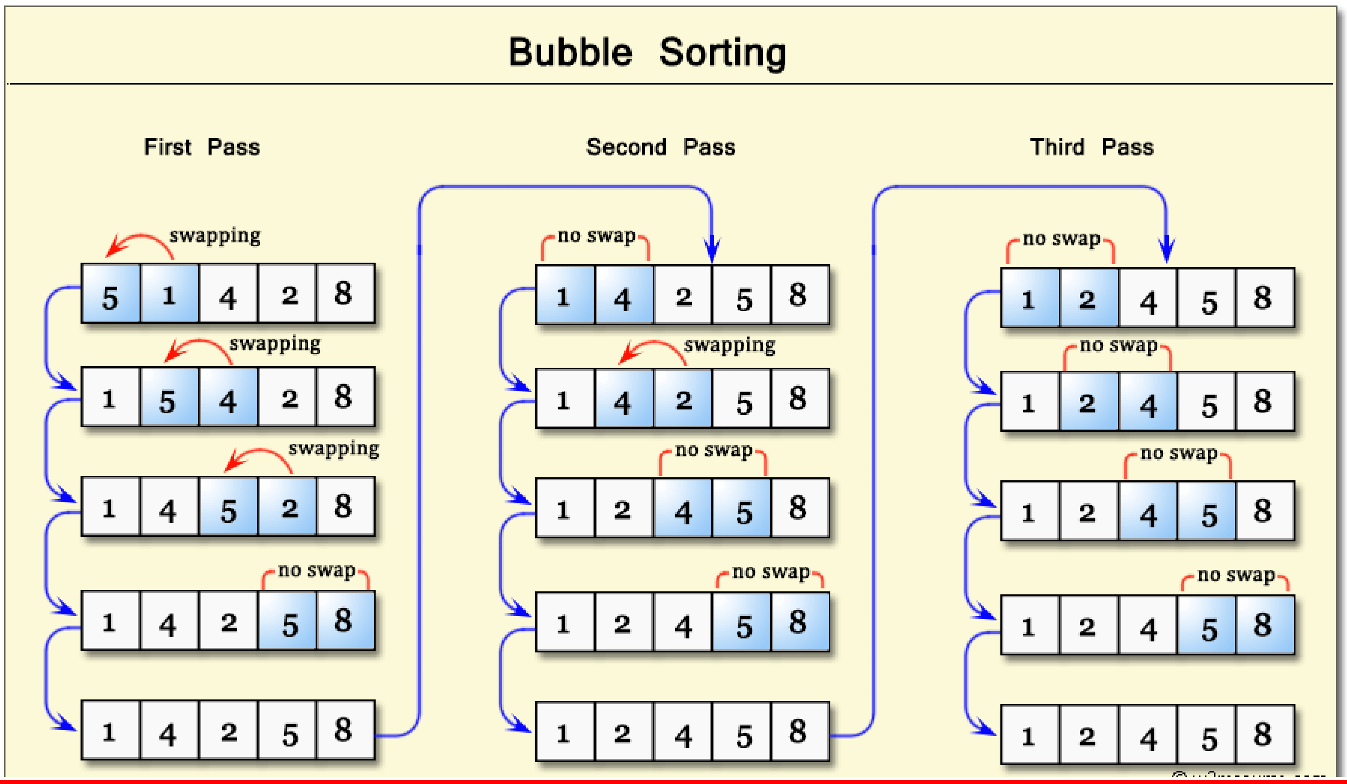
Bubble Sort Algorithm:

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

Algorithm:

In the algorithm given below, suppose **arr** is an array of **n** elements. The assumed **swap** function in the algorithm will swap the values of given array elements.

```
1. begin BubbleSort(arr)
2.    for all array elements
3.       if arr[i] > arr[i+1]
4.          swap(arr[i], arr[i+1])
5.       end if
6.    end for
7.    return arr
8. end BubbleSort
```



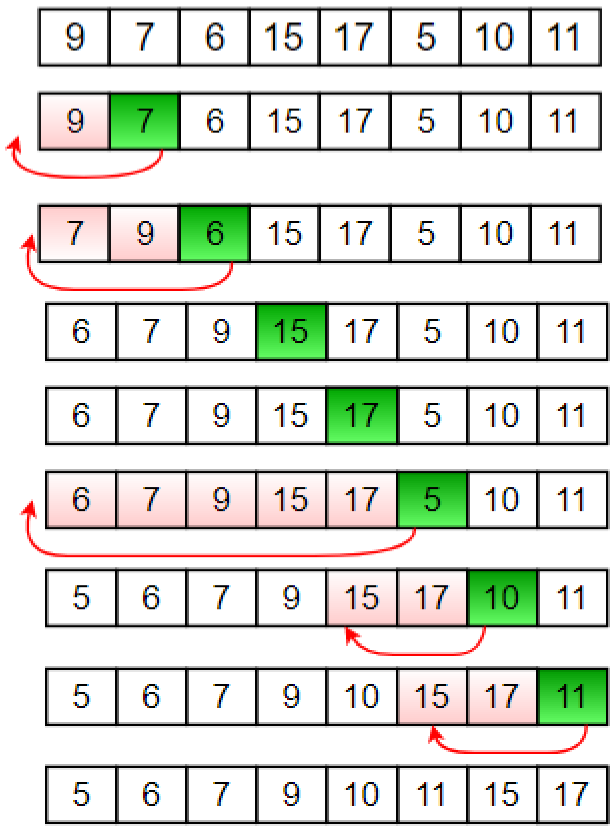
Insertion sort :

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

Characteristics of Insertion Sort:

- This algorithm is one of the simplest algorithm with simple implementation
- Basically, Insertion sort is efficient for small data values
- Insertion sort is adaptive in nature, i.e. it is appropriate for data sets which are already partially sorted.
  - **Algorithm**
  - // Sort an arr[] of size n
  - insertionSort(arr, n)
  - Loop from i = 1 to n-1.
  - a) Pick element arr[i] and insert it into sorted sequence arr[0 1 2 ..i-1]

- **Example:**



SELECTION SORT:

The **selection sort algorithm** sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two sub arrays in a given array.

- The subarray which is already sorted.
- Remaining subarray which is unsorted.

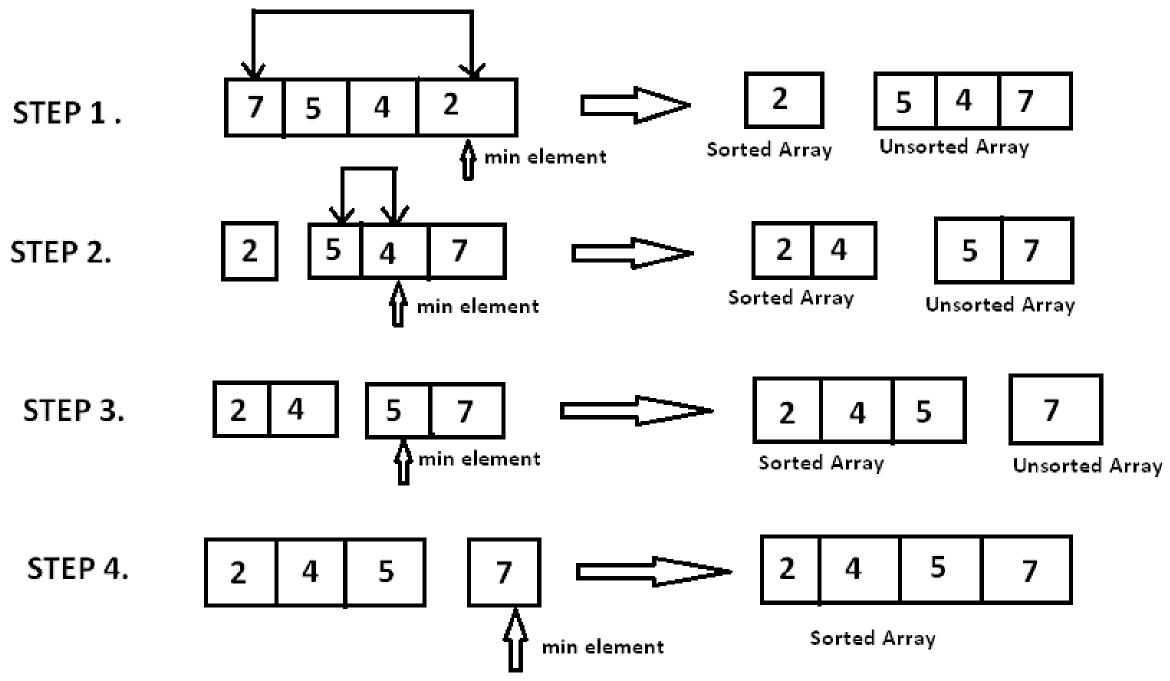
In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

Selection sort is generally used when -

- A small array is to be sorted
- Swapping cost doesn't matter
- It is compulsory to check all elements
-
- **ALGORITHM:**

- **Step 1** − Set MIN to location 0
- **Step 2** − Search the minimum element in the list
- **Step 3** − Swap with value at location MIN
- **Step 4** − Increment MIN to point to next element
- **Step 5** − Repeat until list is sorted

EXAMPLE :



DIFFERENCE BETWEEN INSERTION SORT AND SELECTION SORT

	Insertion Sort	Selection Sort
1.	Inserts the value in the presorted array to sort the set of values in the array.	Finds the minimum / maximum number from the list and sort it in ascending / descending order.
2.	It is a stable sorting algorithm.	It is an unstable sorting algorithm.
3.	The best-case time complexity is Ω(N) when the array is already in ascending order. It have $\Theta(N^2)$ in worst case and average case.	For best case, worst case and average selection sort have complexity $\Theta(N^2)$.
4.	The number of comparison operations performed in this sorting algorithm is less than the swapping performed.	The number of comparison operations performed in this sorting algorithm is more than

	Insertion Sort	Selection Sort
		the swapping performed.
5.	It is more efficient than the Selection sort.	It is less efficient than the Insertion sort.
6.	Here the element is known beforehand, and we search for the correct position to place them.	The location where to put the element is previously known we search for the element to insert at that position.
7.	The insertion sort is used when: • The array is has a small number of elements • There are only a few elements left to be sorted	The selection sort is used when • A small list is to be sorted • The cost of swapping does not matter • Checking of all the elements is compulsory • Cost of writing to memory matters like in flash memory (number of Swaps is O(n) as compared to O(n2) of bubble sort)
8.	The insertion sort is Adaptive, i.e., efficient for data sets that are already substantially sorted: the time complexity is **O(kn)** when each element in the input is no more than **k** places away from its sorted position	Selection sort is an in-place comparison sorting algorithm