UNIT-1

Computer:=

→ the term computer is derived from the word compute

A computer is an electronic device that takes data and instructions as an input from the user and process the data and provides useful information known as output

Data → Input → Process → output (Information)
↓
(Instruction)

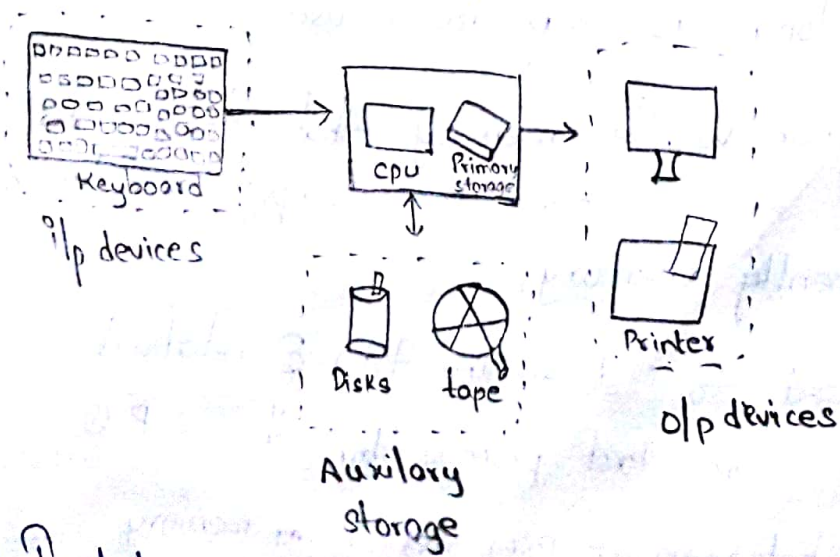
Introduction to components of a computer system:=

→ Computer system is made up of two major components

* Hardware
* Software

Hardware:=

hardware refers to the physical components of the computer system. It consist of 5 major parts



1. Input devices 2. Output devices 3. CPU 4. Primary storage 5. Secondary storage

Input devices:=

The input device is usually a Keyboard where programs and data are entered into the computer

Ex:= Keyboard, mouse, touchscreen, trogball

Processer (C.P.U):=

It is responsible for exiculing instructions such as airthmathic calcutions, comparisons among data & moment of data inside the system

Note: Todays computer have one or more c.p.u

Primary storage:= (in bits)

It is also Known as main memory It is a place the programs and data store temperarly during processing the data in primary storage

The data in primary storage is arized when we turn off the personal computer

Ex:=RAM

RAM:= RAM is called Random Acess memory. It is Volatile memory that temperaily stores data & Applications as long as they are in use

ROM:= ROM is non-volatile memory that stores data and instructions even when the computer is turned off. It is Permenty memory.

cache:= It is used to store the data & related Applications that was last process by the C.P.U It is placed between C.P.U and main memory

Note:= When the processer performs processing it first Searchs the cache memory and then RAM for an instruction

Auxilary storage device:= It is also known as secondary memory. It is a place where the programs and data are stored permnently. when we turnoff computer our programs remain in computer

Hard disk:=

A magnetic based on which we can store Computer data. Hard disk holds more data and also faster than Flopy disc. A harddisk can store anywhere from 10 to more than 100 giga bytes

Flopy disk:=

Where as more flopies have a maximum storage capacity of 1.4 megabites.

Output devices:=

The output device is usually a monitor or a Printer to show output.

if the output is shown on the monitor it is a softcopy and if the output is printed on the printer it is hardcopy.

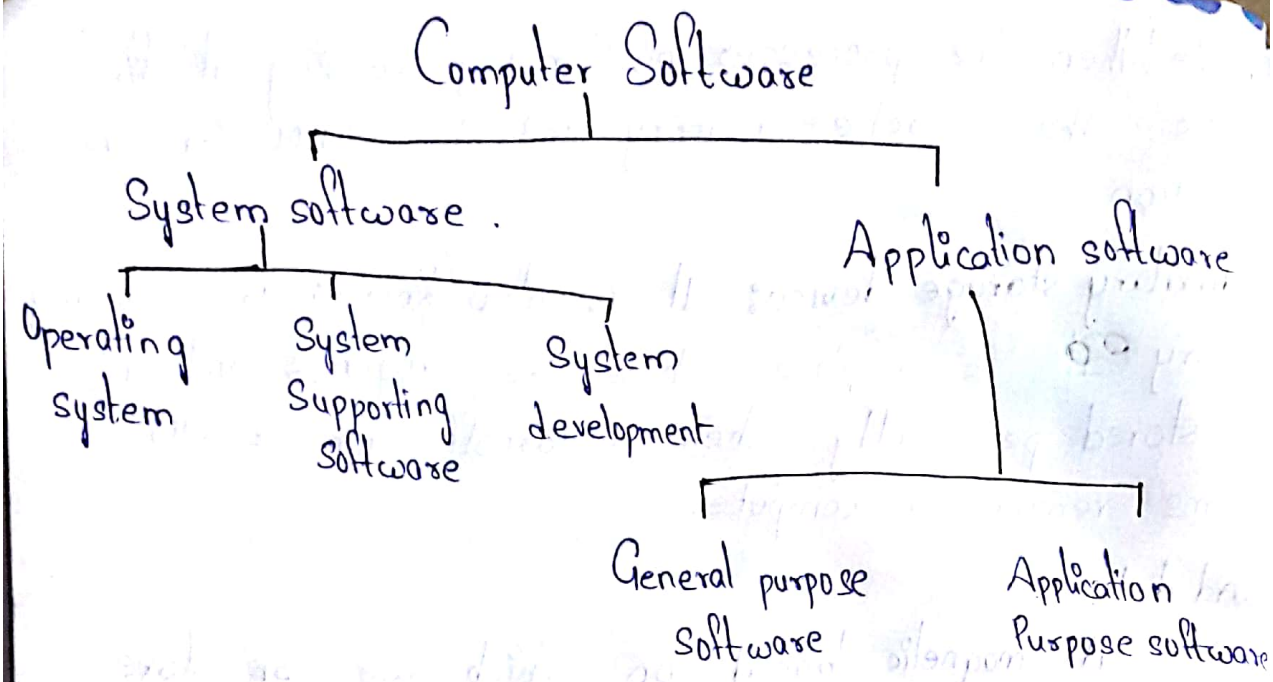
Computer softwaare:=

Software is the collection of programs or instruction that allow hardwaare to do its job.

Software divided into 2 types:

System softwaare.

Application Softwaare.



System Software:-

It consists of programs that manage the hardware resources of a computer and perform required information processing task.

These programs are divided into 3 types:-

1. OS
2. System supporting software.
3. System development

Operating system:-

The O.S provides the interface between an user and System to communicate.

It provides services such as user interface, file and database acess and interfaces to communication System such as internet protocall.

The primary purpose of this software is to Keep the system operating in ~~mobil~~ an efficient manner by allowing the user acess to the System

System support software:-

It provides system utilities and other operating services. the system utilities are sort programs and desk formate programs. the operating services include Providing security monitors to protect the system and data.

System development software:-

It includes the language translater that converts Programs into machine language for excution debaging tools to insure that programs are error free.

Application Software:-

It divided into two types:

1. General purpose software
2. Application specific Software

General purpose software:-

It is purchased from a software developer and can be used for more than one Application.

Example of Generalpurpose software include. workprocess databais / managment system.

Application purpose software:-

It can be used only for its intented purpose. Thes softwares are used for specific applications for which they are designed they cannot be used for other generalised task

System development tools:-

1. language translaters.
2. linkers.
3. Debuggers
4. Editors.

Language translators:=

It divided into 3 types:=

Assembler

Compiler - decompiles

Interpreter.

Assembler

It converts middle level language to machinery language

Compiler

It converts high level language to machinery language.

decompiler

It converts machinery language to high level language

Interpreter

It converts highlevel statement to machinery according to step by step.

Linkers:=

All headerfiles in computer are called linkers

Ex:- <stdio.h>

<conio.h>

<Math.h>

Debuggers:=

It shows errors in programs

It is divided into 2 types:-

1. Machinary level debuggers 2. high level debuggers

editior:=

It is special program that allow the user to work text in a computer system.

Computer Languages:= (to communicate with computer)

Low level language (understood by computer).

binary language

* It is 1st generation languge.
* It consists of '0' and '1'
  Ex:- 010110

Assembly language / symbolic language (1950

* It consists of symbols.
  Ex:- ADD, SUB (memonics)
* It is 2nd generation language.

Machine language by using compiler converts to high level

Assembly language by using Assembler converts to machinary then again using compiler converts to high level.

Ex:- 010101 ⟶ complied ⟶ highlevel

ADD, SUB ⟶ Assemble ⟶ 010101

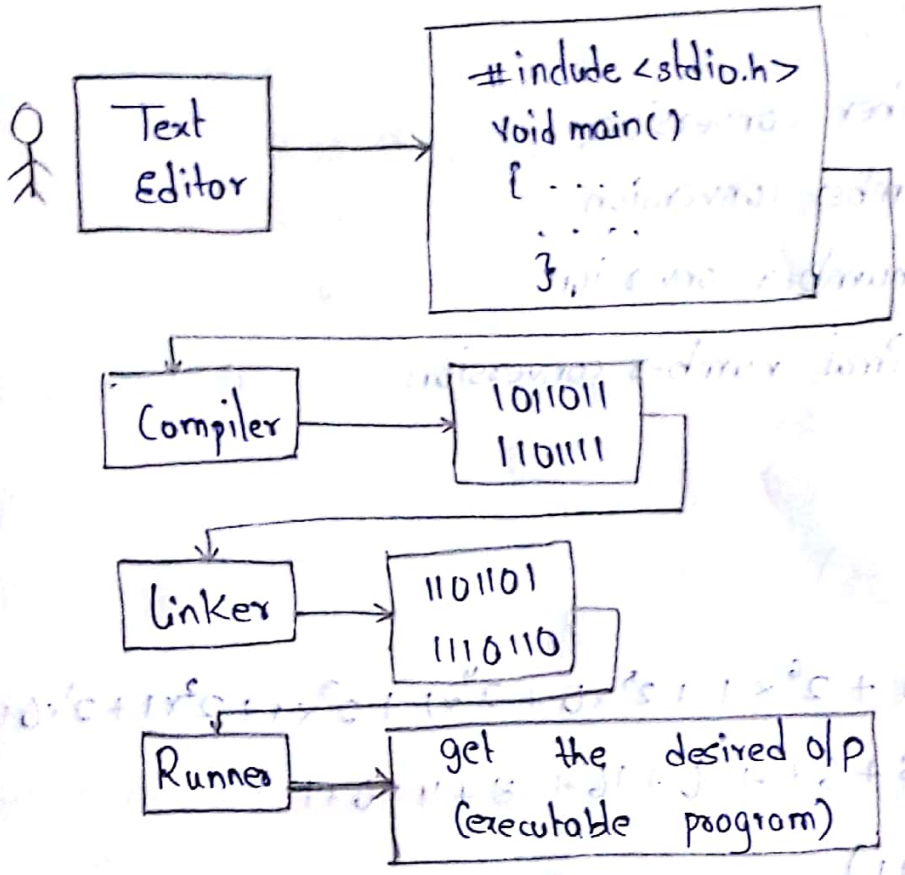
* low level language is machine dependent language.

high level language

* user can understand this language.
* It consists of english words.
* It is machine independent language

Creating And Running Program

* Writing and Editing programs
** Compiling programs
*** Linking programs
**** Runner [executing the programs]



Data measurement ($2^{10} = 1024$)

8 bits = 1 byte
1024 bytes = 1 Kilo byte
1024 KB = 1 MB (mega)
1024 MB = 1 Giga Byte
1024 GB = 1 Tera Byte.

Computing environments :=

* Personal computing environment desktop
** Time sharing environment
*** client - server environment
**** Distributed environment

Number systems :=

*Binary number conversion : {0,1} base = 2
**Octal number conversion {0,1,..7} base = 8 size - 3
***Decimal number conversion {0,1,...9} base - 10
****hexa-decimal number conversion {0,1..9 & A-F} base-16 size

Conversion of Binary to decimal.

$(11011101)_2 = (\ )_{10}$

$= 2^7 \times 1 + 2^6 \times 1 + 2^5 \times 0 + 2^4 \times 1 + 2^3 \times 1 + 2^2 \times 1 + 2^1 \times 0 + 2^0 \times$

$= 128 + 64 + 0 + 16 + 8 + 4 + 0 + 1$

$= (221)_{10}$

$(101111101)_2 = (\ )_{10}$

$= 2^8 \times 1 + 2^7 \times 0 + 2^6 \times 1 + 2^5 \times 1 + 2^4 \times 1 + 2^3 \times 1 + 2^2 \times 1 + 2^1 \times 0 + 2^0 \times 1$

$= 256 + 0 + 64 + 32 + 16 + 8 + 4 + 0 + 1$

$= (381)_{10}$

$(11101011)_2 = (\quad)_{10}$

$= 2^8 \times 1 + 2^7 \times 1 + 2^6 \times 1 + 2^5 \times 0 + 2^4 \times 1 + 2^3 \times 0 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 1$

$= 256 + 128 + 64 + 0 + 16 + 0 + 4 + 2 + 1$

$= (471)_{10}$

$(110101010) = (\quad)$

$= 2^8 \times 1 + 2^7 \times 1 + 2^6 \times 0 + 2^5 \times 1 + 2^4 \times 0 + 2^3 \times 1 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 0$

$= 256 + 128 + 0 + 32 + 0 + 8 + 0 + 2 + 0$

$= (426)_{10}$

Conversion of binary to octal

$(11011101)_2 = (335)_8$

011 | 011 | 101
$2^2 2^1 2^0$ | $2^2 2^1 2^0$ | $2^2 2^1 2^0$

3 | 3 | 5

$(101111101)_2 = (575)_8$

101 | 111 | 101
$2^2 2^1 2^0$ | $2^2 2^1 2^0$ | $2^2 2^1 2^0$

5 | 7 | 5

Computer languages.

These languages are used to communicate with the user and computer to implement his idea and logics.

These are classified into two types:-

low level language

high level language.

The low level language :- These are again divided into 2 types.

1. Binary or machine level language
2. Symbolic or assembly language.

Binary or machine level language:-

It is a first generation programming language it contains a sequence of binary digits. such as zero's and one's. These developed in 1940's.

Advantages:-

Computer understands binary program or binary language directly.

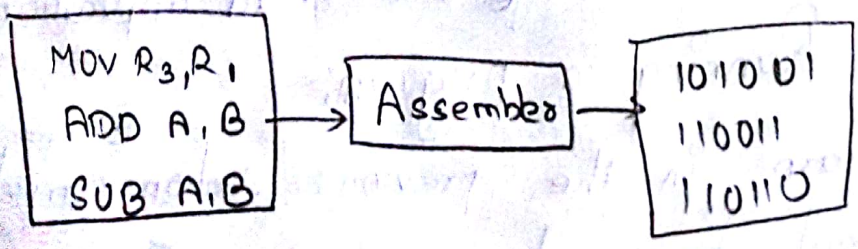
Disadvantages:

It is very difficult to remember the operational codes and address of the memory locations. user cannot modify the program easily it is suitable for only simple applications.

Assembly or Symbolic languages:-

Computer engneers development a new language such as Assembly or Symbolic in 1950's It overcome the drawbacks of machine language. This is a second generation Computer programming language. This language contains symbolic code to represent instructions.

Ex:- IN, OUT, ADD, SUB, MUL



Advantages:-

User can remember the symbol codes rather than the sequence of binary code. It is quite Easy to understand and develop the Program. user can modify the program very easily.

Disadvantages:-

It is machine dependent language.

Highlevel language.

Engneers develop another kind of programming languages know as high level languages. These are like english languages.

These are developed to overcome the difficulties of low level languages. These are machine independent languages. the elements of these languages are alpha.bets, digits and other special symbols. It is not directly understandable by the computer.

Ex:- Java, C++,

Advantages:-

Easy to modify and write the programs. It is suitable for large and small applications

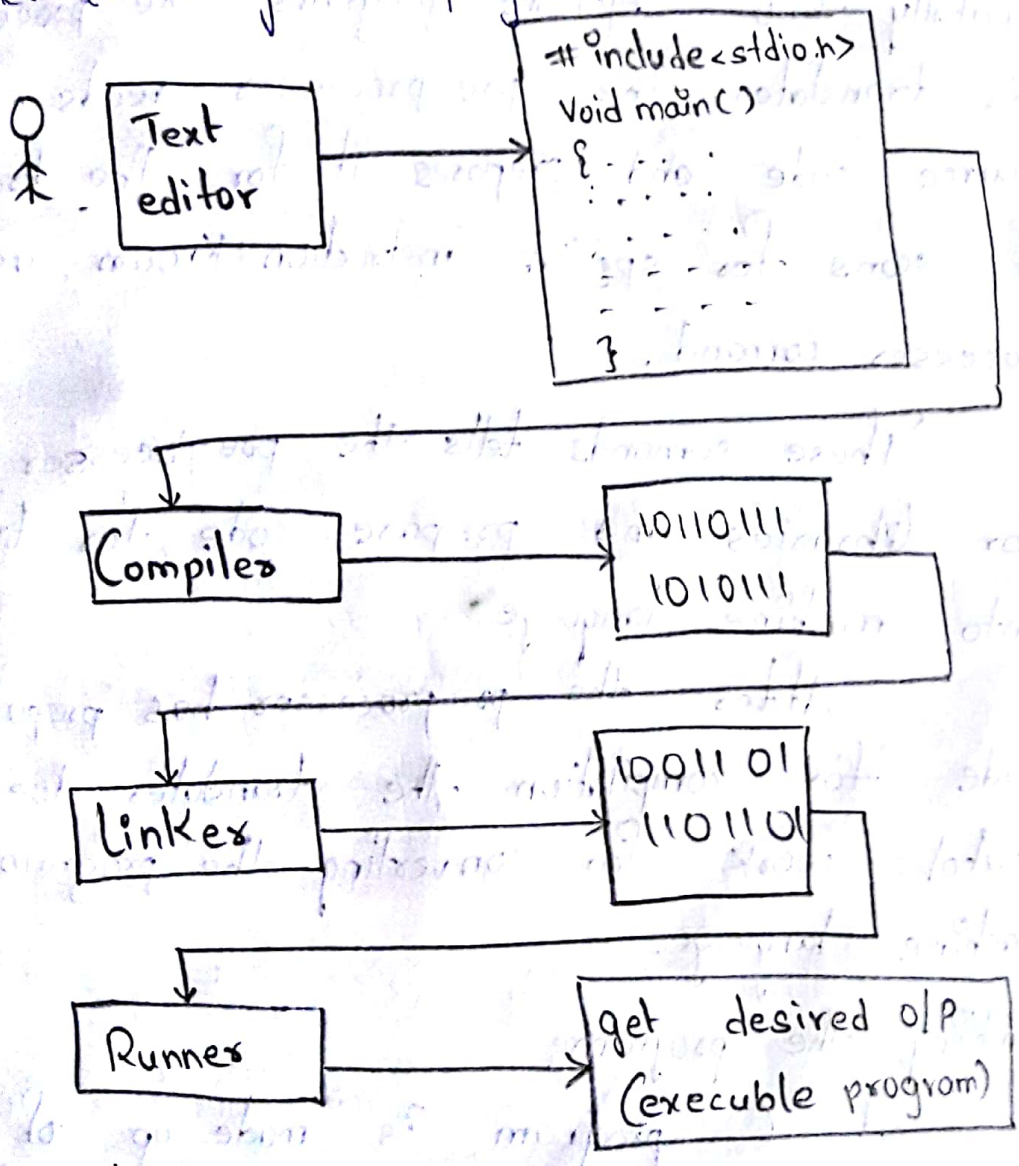
Creating and Running a program:-

As we learn in the previous section computer doesnot understand human languages computer understand a program if it is coded in its machine

language. It is a job of a programmer to write and test the program

there are 4 steps in the process.

1. writting and editing programs.
2. Compiling programs
3. linking programs.
4. Runner (executing the program)



Writing & edditing program:

Creat a text file using any word processing software with its name ending with .c excute and type a C program in that file

we are writting the source program the turbo c text editor

Compiling program:

The code in a source file store on the desk must be translated into machine language This is the job of Compiler. The C compiler is acutally two separate programs. The processer and the translater. the pre-processer reads the Source code and prepares it for the translater. It scans for special instruction Known as pre-Processer comand.

These comands tells the pre-pocesser to look for libraries and prepare code for translation into machine language.

After the pre processer has prepare the code for compilation the translater does the acutal work for converting the program into machine language.

linking the program:

A C program is made up of many functions we write some of these functions and they are part of our source program

Input, output functions are included in <stdio.h> headerfile. and we will be using mathematical library functions

The linker assembles all these functions with the system and creats a code called as exicutable code or exicutable program.

Exicuting the program:

to exicute a program we need an operating system comand such as run. to load the program into primary memory and exicute it.

Getting the program into memory is the function of an operating system program known as loader. loader locates the exicutable program and reads into the meomory when everything is loaded the program. takes control and begin exicution.

Computing Environments:

It contains 4 types of environments. They are

1. personal computer environment.
2. Time-sharing environment
3. Clint server environment
4. distributed Computing environment.

1. personal computing environment:

In this all of the computer hardware components are grouped together in our personal computer.

In this situation we have the whole computer for our self.

2. Time-sharing environment.

In this the central computer has many duties it must control share resources.

It must manage the shared data and Printing.

In the time-sharing environment many users are connected to one or more computers.

Ex:- printer

3. Client server environment:-

A client server computing environment divides the computing functions between the central Computer

In this the micro computer are called the clients

The central computer, which may control main frames is known as servers because the work is known shared by the using and central computers.

4. Distributed Computing environment

The internet provides conectivity to different Servers to the world

NUMBER SYSTEM:

A number can be denoted using different symbols a system of symbols used to represent numbers is called number system.

These number system are classified according to the values of base of no: system.

In digital system following number systems are frequently used.

1. Binary number conversion
2. Octal number conversion
3. Decimal number Conversion
4. hexa-decimal number conversion.

Algorithm

Algorithm is a method of representing step by step logical procedure for solving a problem. An algorithm must follow the following properties

Input, Output:-

Each algorithm must take one or more quantities as input data and procedure one or more output values

Finitness:-

An algorithm must terminate in a finite no: of ~~System~~ steps

Definitness:-

Each step of the algorithm must be clear and ambigious. (no confusion)

Effectiveness:-

each step must be effective and produces the correct results and can be perform exactly in a finite amount of time

Ex:- write an algorithm for finding 3 ave num (a,b,c)

Step1: Input values a,b,c.
Step2: Add a,b,c
Step3: Divide by 3
Step4: The result is stored in d
Step5: print the result.

* write an algorithm biggest of two numbers

Step 1: Take input values a,b
Step 2: Compare a<b, if true b else a
Step 3: result will be stored in c
Step4: print the c value.

* biggest of 3 numbers

step1: Take input values a,b,c
step2: Compare a,b and go to step4
step3: other wise compare b,c check b>c
then print b is biggest else c is big
step4: Compare a,c, check a>c then
Print a is big else c is big

Step 5: the result will be stored in c

Step 6: Print c value.

Flowchart :=

It is a diagramatic representation of an algorithm.

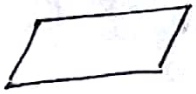
It is constructed using different types of boxes.

#Symbols

All symbols are connected by arrows to indicate the flow of information and processing

~~(vowel or terminal is)~~

Oval or terminal  to start and end.

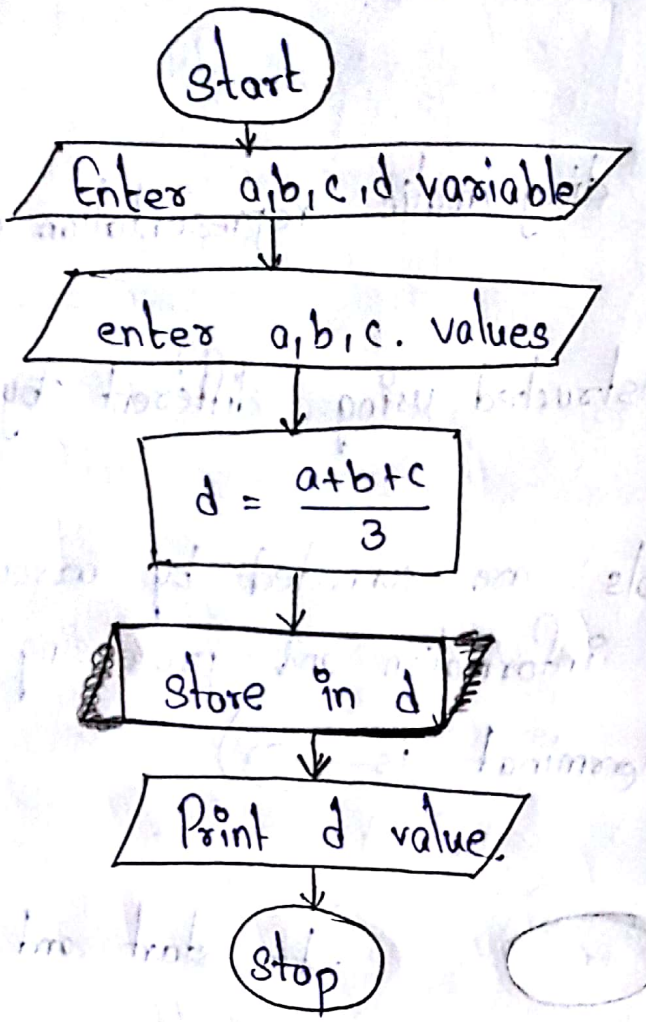
Input or output  making data availble for processing of the informal

Rectangle or Process box  Any process

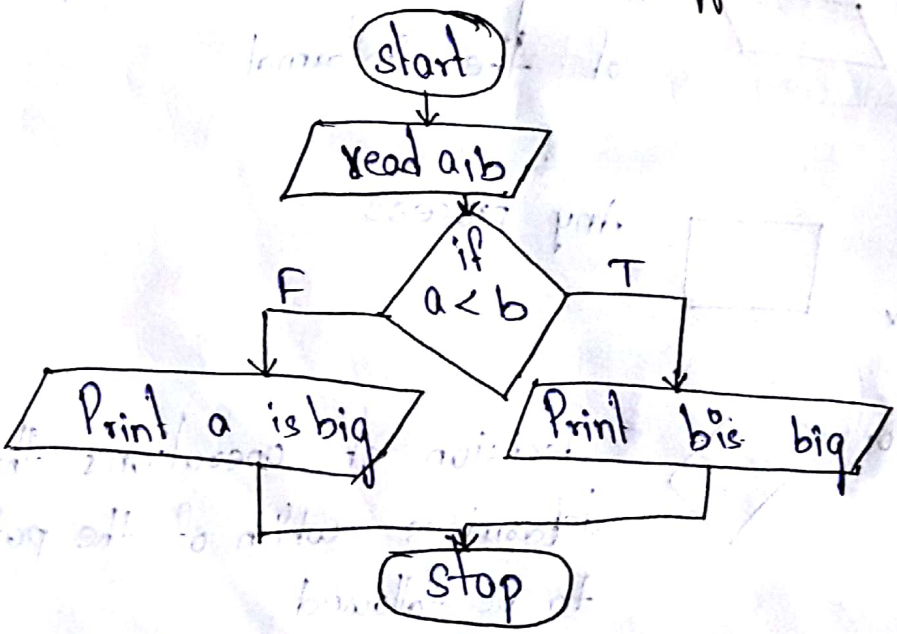
Rombus or decision  decision of operations that determines which of the path is to be followed

Arrow → used to connect different parts of flow chart

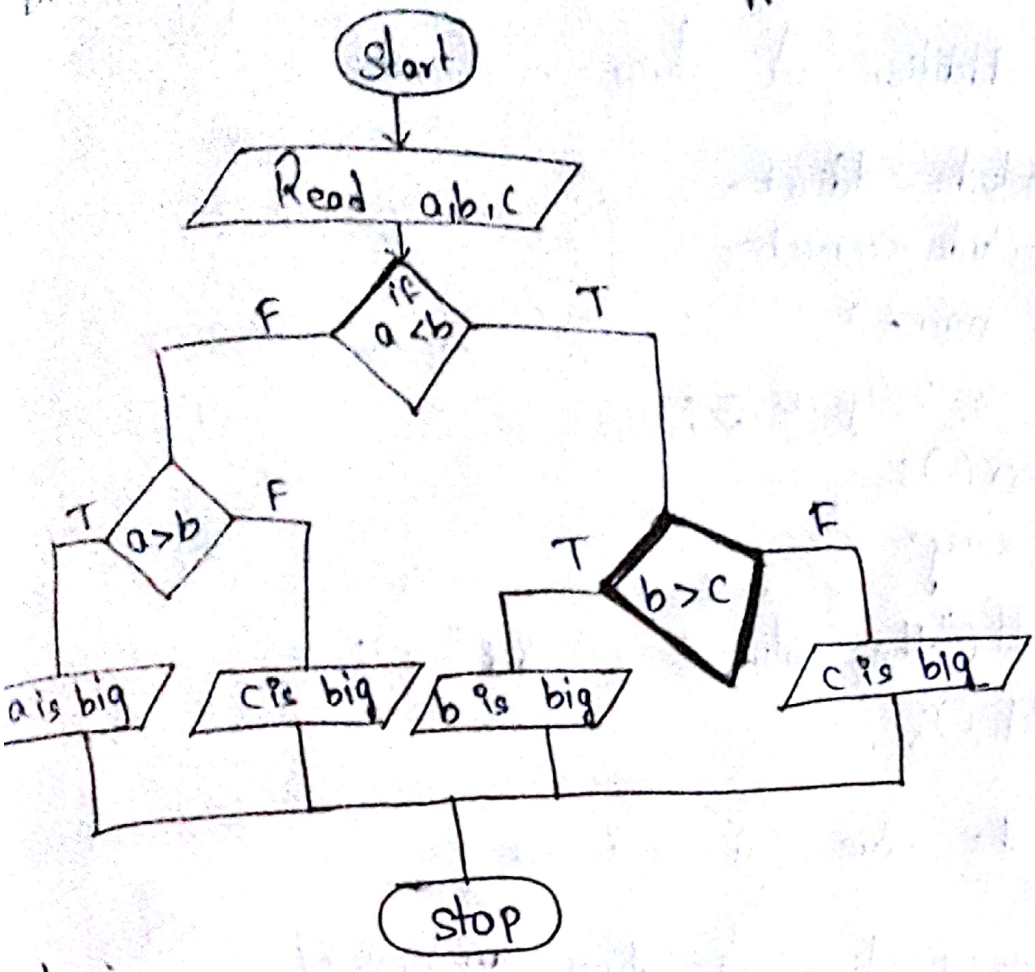
Draw the flowchart for avege of 3 numbers



Draw the flowchart for biggest of a numbers



Draw the flowchart for biggest of 3 numbers



Structure of C Programme.

Document Section	→ /* Addition of two numbers */
Link Section	→ headerfiles. [Printf(), scanf(), sqrt()]
Definite Section	→ #π=3.14. (constants, values)
Global declaration	→ giving variable before main() so that we can use anywhere in the program.

main()
{
declaration
execution
}	

functions	

Programme: 1

```
/* Addition of two numbers */

#include <stdio.h>
#include <conio.h>
void main()
{
int x=10, y=20, z;
clrscr();
z=x+y;
Printf("the value of z is %d ", z);
getch();
}
```

output: the value of z is 30

Programme 2:

```
/* Subtraction of two numbers */

#include < stdio.h>
#include <conio.h>
Void main()
{
int x=10, y=20, z;
clrscr();
z=y-x;
Printf (" the value of z is %d ", z);
getch();
}.
```

Output: the value of z is 10.

Programme : 3

```
/* multiplication of two numbers */
#include <stdio.h>
#include <conio.h>
Void main()
{
int x=5, y=4, z;
clrscr();
z=x*y;
Printf("the value of z is %d", z);
getch();
}
```

Output: Value of z is 200

Programme: 4

```
/* division of two numbers */
#include <stdio.h>
#include <conio.h>
Void main()
{
int x=15, y=3, z;
clrscr();
z=x/y;
Printf("the value of z is %d", z);
getch();
}
```

Output: the value of z is 2

Programme: 5
/* Addition of two numbers */

```
#include <stdio.h>
#include <conio.h>
Void main()
 {
  int a,b,c;
  clrscr();
  Printf ("Enter the value of a");
  Scanf ("%d", &a);
  Printf ("Enter the value of b");
  Scanf("%d", &b);
  c=a+b;
  Printf("the value of c is %d", c);
  getch();
  }
```

Programme 6
/* Subtraction of two numbers */

```
#include <stdio.h>
#include <conio.h>
Void main()
{
int a,b,c;
clrscr();
Printf ("enter the value of a");
Scanf ("%d", &a);
Printf ("enter the value of b");
Scanf ("%d", &b);
c=a-b;
Printf("the value of c is %d", c);
getch();
}
```

output: enter value of a,b
a
2
value of c = 2

/* Multiplication of two numbers */ Programme 7

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a,b,c;
clrscr();
Printf ("enter the value of a,b");
Scanf ("%d %d", &a, &b);
c = a*b;
Printf ("the value of c is %d", c);
getch();
}
```

output: enter the value of a,b
2
4
value of c is 8

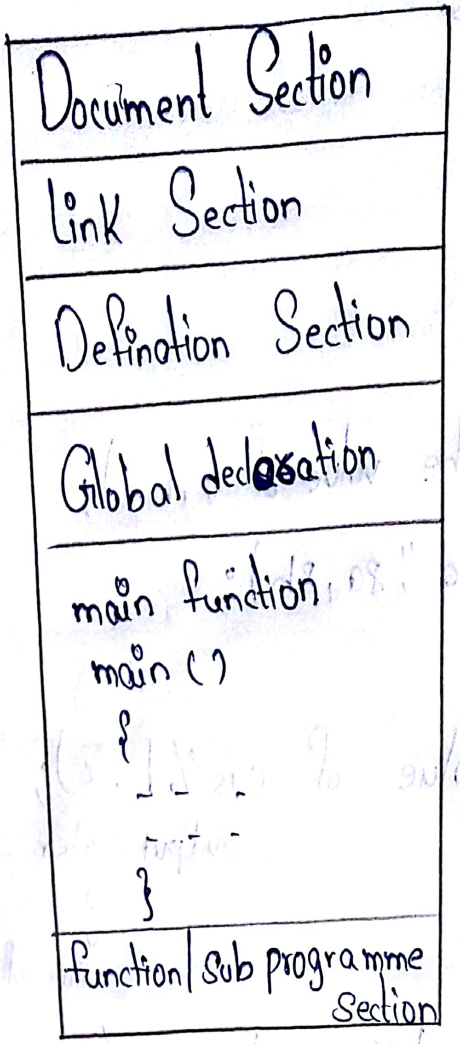
Programme 8

/* division of two numbers */

```
#include <stdio.h>
#include <conio.h>
Void main()
{
int a,b,c;
clrscr();
Printf ("enter the value of a,b");
Scanf ("%d %d", &a, &b);
c=a/b;
Printf ("the value of c is %d", c);
getch();
}
```

output: enter the value of a,b
100 (enter)
20 (enter)
the value of c=5

Structure of C programme.



Documentation Section:-

This section contains a set of comment lines use to specify the name of programme, comments include blw the delimiters.

/* ..... */

→ These statements are not excutable rather they are ignored by compiler.

→ Comments can be used in two different formats

* block comments
* line Comments

→ A block comment is nothing but a set of comment lines enclosed b/w opening and closing comment

→ Example /* This is a program to print even or odd */

→ The line comment uses two slashes (//) to identify a comment

→ Example :- Int a; //declaration of variable a.

Link Section :-

→ It is also called as preprocesser, headerfile.

→ This is used to link system library files, for defining the conditional instructions.

→ A headerfile has an extention of '.h'

→ These files are included in the programme, using the preprocesser directive #include.

Defination Section :-

→ In this section we are going to define few constant values like $\pi = 3.14$

Global declaration :-

→ This section is used for declaring global variable

→ There are some variables that are used in one or more functions, Such variables are called global variables.

→ These variables are declarated outside of main function.

Main programme section :-

→ Every C programme must have one function which specify starting of C programme the group of statements in main are exicuted sequencially

→ The main function consists of declaration and executable statement.

declaration part:

→ Is used to declare all variables, that are used in the function and these are called local variable

execution part:

→ This Section contains all variables used in execution

→ The execution of a program begins with opening brace ({) and closing brace (})

Sub programme Section:-

It is userdefine.

C langange Provides the facility to define their own function

This Section is also called as userdefine function section,

Variables:-

A variable is the name given to the momery location, where the data value of the variable is stored

A variable is an entity that may vary during program execution.

Syntax:- datatype variable name; datatype variable name = constant;

Ex:- int a;
float a = 3.1;
char ch = 'y';

Variable declaration:

A declaration is used to name an object such as variable.

Definations are used to create an object

A variable type can be any of the datatypes such as character, integer (or) real.

We donot use void data type to declare variables.

Syntax: data type variable;

Variable initialization:=

Initialization of variable can be using assigment operator (=)

Syntax:= datatype variable = constant;

Ex:- int x=50;

Scope of Variables:=

The scope of variables implies availability of variables with in the programme.

There are two scopes.

1. Local variables
2. Global variable

Local Variables:

The variables that are definied with in a function is called local variables

Ex:- fun()
{
int a;
}

a is local variable.

Global Variables:-

These are the variable defined before the main(), these variables can be used in any number of functions.

Global variables are also called as external variables

```
Ex:- int x,y;
     main()
     {
     int a,b
     }
```

x,y are global variables

Rules for creating variable names:-

→ Variable names consists of letters, digits and underscore

→ Variable names must begin with an alphabet and (or) underscore.

→ The length of the variable name can be upto 31 characters

→ No commas, blank spaces are allowed with in a variable name.

Ex:- My college (Invalid)

Mycollege (valid)

→ Key words shouldnot be used as variable names

→ Variable names are Case sensitive

SUM & sum are not equivalent

27/8/18

Data types:-

C language is rich in datatypes. The different datatypes are available and allows the programmer to select the type apropriate to the needs of application.

Datatype is used for representation of data ANSIC supports 3 classes of datatypes. Theyare

* primary datatypes
* derived datatypes
* userdefinied datatypes

Primary datatypes:-

There are ~~four~~ types of primary datatypes

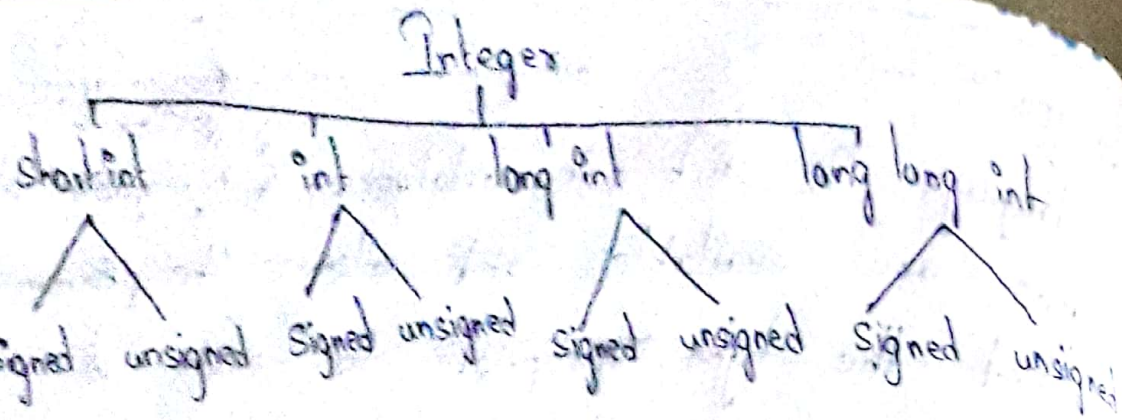
* integer
* character
* float
* Void

Integer (Int)

integers are whole numbers with a range of values supported by a particular machine

C supports four different sizes of the int datatype

a* Short int
b* int
c* long int
d* long long int



There are two categories in integer they are

Signed and unsigned

If the integer is signed then one bit must be used for Sign (+ or -)

If we need to know the size of any datatype C provides an operator called 'sizeof' which will tell us exactly size in bits

Signed values (integers) store both negative and positive values

Unsigned integers have only positive values

Type	Size	Range	format specifier
Signed short int	2 bytes (or) 16 bits	-32768 to +32767	%d
Signed int	4 bytes (or) 32 bits	-2^{31} to $2^{31}-1$ -2147483648 to 2147483647	%d
Signed long int	4 bytes (or) 32 bits	-2^{31} to $2^{31}-1$	%ld
Signed long long int	8 bytes or 64 bits	-2^{63} to $2^{63}-1$	%lld

Type	Size	Range	Format specifier
unsigned short int	2 bytes (or) 16 bits	0 to $2^{16}-1$	%u
unsigned int	2 bytes (or) 16 bits	0 - $2^{16}-1$	%u
unsigned long int	4 bytes (or) 32 bytes	0 to $2^{32}-1$	%lu
unsigned long long int	8bytes (or) 64 bits	0 to $2^{64}-1$	%llu

Syntax:-

datatype Identifier/variable;

Ex:-

```
short a;
Signed short a;
short int a;
Signed short int a;
unsigned short a;
unsigned short int a;
```

① programm to implment datatype

```
#include<stdio.h>
#include<conio.h>
void main()
{
short int a=20;
clrscr();
Printf("%d", a);
getch();
}
```

output - 20

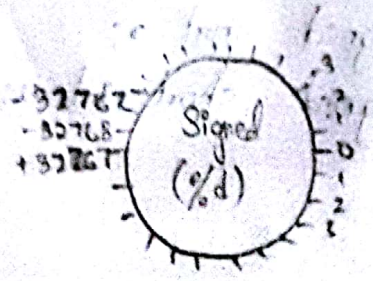
Signed datatype (short) with in a range

```
#include<stdio.h>
#include<conio.h>
void main()
{
signed short int x=32766;
clrscr();
Printf("%d", x);
getch();
}
```

output - 32766

datatype short beyond range

```
#include<stdio.h>
#include<conio.h>
void main()
{
Short in x=3269;
clrscr();
Printf("%d", x);
getch();
}
```

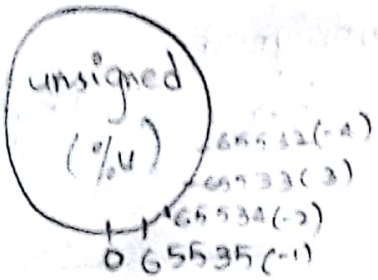


output → -32767

unsigned datatype short beyond range

```
#include<stdio.h>
#include<conio.h>
void main()
{
unsigned short int x=-4;
clrscr();
Printf("%u",x);
getch();
}
```

output = 65532



Character:-

→ character occupies one byte of memory

→ C standard provides 2 character types i.e 'char' and 'wchar_t'

→ wchar_t is wide character type.

→ most computers use the ASCII (America standard code for information interchange) called as ASCII alphabet for converting characters into numbers.

ASCII

A-65	a-97	0-48	#-35
B-66	b-98	1-47	
⋮	⋮	⋮	⋮
Z=90	Z=122	9-57	15
26	26	10	

26 + 26 + 10 + 15 < 256(2^8) one bytes

→ character must be enclosed in sigle codes (' ').

Syntax:- datatype Variable;

```
char x;
(or)
char ch='x';
```

Type	Size	Range	formate specifier
char (or) signed char	1byte	-2^7 to 2^7-1 -128 to 127	%c
unsigned char	1byte	0 to 2^8-1 (or) 0 to 255	%c

Programme to print character

```
#include<stdio.h>
#include <conio.h>
Void main()
{
char ch='x';
clrscr();
Printf("%c",ch);
getch();
}
```

Output – x

Printing character value interms of integer

```
#include <stdio.h>
#include <conio.h>
Void main()
{
char ch='A';
clrscr();
Printf("%c", ch);
Printf("%d", ch);
getch();
}
```

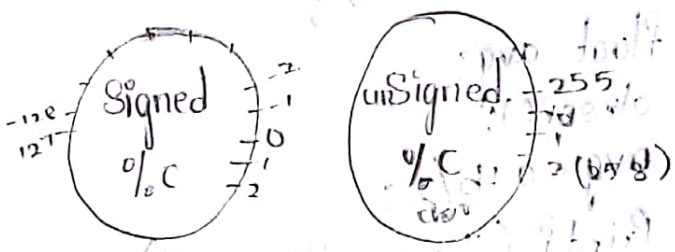
output – A
65

28/8/18

Programme for checking the range of a character

```
#include<stdio.h>
#include<conio.h>
void main()
{
char ch = 258;
clrscr();
Printf("%d\n", ch);
Printf("%c", ch);
getch();
}
```

o/p := 2
Garbage value



Float

floating points are stored in 32 bits or 4 bytes

Type	size	range	formate specifier
float	4bytes	$3.4e^{-38}$ to 3.4E+38	%f
double	8 bytes	1.7E-308 to 1.7E+308	%lf
long double	10 bytes	3.4E-4932 to 3.4E+4932	%Lf

Programe 1 :=

Programme to calculate average of two numbers

```
#include<stdio.h>
#include<conio.h>
Void main()
{
int a=20 ,b=5 ;
float avg;
clrscr();
avg = a+b/2;
Printf (" avg= %f ,avg");
getch();
}
```

output⇒ avg = 12.5

Constant:

Constants are data values that cannot be changed during execution of programme

Constant representation:

we use symbols to represent the constant we have different types of constant representation such as

1. Boolean constants
2. Character constants
3. Integer constants
4. Real / floating point constants - 3.14 , 3-4 e-4
5. String constants .

Boolean Constants:=

A boolean data type can take only two values ('1'T' or 'F'(0)

we use constant T or F in our programm

character constant:-

character constants are enclosed between two single quotes. In addition to character we can also use backslash ('\') and it is known as escape character.

Some of escape characters are listen below

Ascii character	symbolic name.
null character	'\0'
alert	('\a')
backspace	('\b')
horizontal tap	'\t'
Vertical tap	'\v'
new line	'\n'
backslash	'\\'
Single Quote	'\''
double quote	'\"'

Integer Constant:

These constants doesnot contain a decimal point

It contain digits, must be minimum of one digit

Blank spaces and commas are not allowed within the integer constants the integers can be either positive or negative.

Ex:- +123, -378 etc

Real constants:-

Real constants are those which accept decimal values (fractional and exponential)

Ex:- 3.1416 , +342E-4

String constants

It is a sequence of 0 or more chara enclosed in double codes

Ex:- [10] = "xyz"

or

"Helloworld"

Operators:-

C operators are classified into following catagi

arthimatic operator
logical operator
Relational operator
Assignment operator
increment or decrement operator
Bitwise operator
Conditional operator
Special operator.

Arthimatic operator:-

This is divided into 3 types

* Integer
* Real
* mixed mode

operator / symbol	operation
+	Add
-	sub
*	Mul
/	div
%	Modulos

here (%) modulos prodeices reminder as a value.

A

Integer Airthematic.

These arthimatic operations are done between two integer values

Ex:- int a = 10 b = 10;

a+b = 10+10=20

a-b = 10-10=0

a*b = 10*10=100

a/b = 10/10=1

a%b = 10%10=0

Real Arthimatic:-

These arthimatic operations are done between two real values

Ex:- float a=10.5, b=11.6

a+b = 10.5 + 11.6

a/b = 10.5 /11.6

mixed mode arthimatic:

The arthimatic operation done between integer value and real value

Ex:- int a = 5;

float b = 2.5;

a+b = 5 + 2.5;

a - b = 5 - 2.5;

Relational operators:-

Relational operators are used to compare the two quantities or two values

Relational operatior	meaning
<	lessthan
<=	lessthan or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal to

Ex:- int a=6, b=10;

a>b false

a==b false

a<b True

a!=b True

Logical operators

There are 3 types of logical operators.

logical operator	meaning
&&	LOGICAL AND
||	LOGICAL OR
!	LOGICAL NOT

LOGICAL AND (&&)

The result of logical and expression is true when both expressions are true

Syntax:-

exp1 && exp2

exp1	exp2	result
T	T	T
T	F	F
F	T	F
F	F	F

LOGICAL OR (||)

The result of logical or expression is false when both expressions are false.

Syntax:-

exp1 || exp2

exp1	exp2	result
T	T	T
T	F	T
F	T	T
F	F	F

Ex1- a=10, b=5, c=15

y = (a>b) || (b>c)

= T || F

y = 1 (T)

LOGICAL NOT (!)

The result of logical not expression is true if the expression is false.

Syntax :- !exp

Exp1	!Exp2
T	F
F	T

Assignment Operators:-

These are used assign the result of an expression to a variable

There are two forms of assignment operators

* simple * compound

Syntax

1. Variable = expression; // simple assignment

Ex:- a=10 b=5

c=a+b

c=15

2. Variable = variable operator expression;

(or)

Variable operator = expression; // compound

Ex:- a = a + (c*d);

or

a+=b;

Compound expression	simple expression
a+=b	a=a+b
a-=b	a=a-b
a*=c	a=a*c
a %=b	a=a%b
a-=10	a=a-10

29/8/18

Increment and decrement operators:-

These operators are represented as '++' & '--'

'++' means increments the value by 1

'--' decrements the value by 1

exp	meaning
a++	Post increment
++a	Pre increment
a--	Post decrement
--a	Pre decrement

Example:- int a=6;

b = ++a; // first increment

value of b is 7

Note: a++ is same a=a+1 or a+=1

Bitwise operators:-

These are applied on bits. bitwise operators does calculation on bits and stores in the memory

Bitwise operator	meaning
&	bitwise AND
|	bitwise OR
^	Exclusive OR (XOR)
!	bitwise NOT
<<	left shift
>>	right shift

Bitwise AND

a	b	a&b
1	1	1
1	0	0
0	1	0
0	0	0

Bitwise OR

a	b	a|b
1	1	1
1	0	1
0	1	1
0	0	0

Bitwise NOT

a	!a
1	0
0	1

XOR

a	b	a^b
1	1	0
1	0	1
0	1	1
0	0	0

Conditional or ~~terminary~~ ternary operators

These are also known as ternary operators. These take three operands and it is used to check the condition between two statements.

Syntax:-

```
exp1 ? exp2 : exp3;
```

Ex:- `int a=5, b=10, c=15;`

```
y = (a>b) ? b : c;
```

The operator (?) works as follows

Exp1 is evaluated first, if it is true then the exp2 is evaluated and becomes the value of expression. (either exp2 or exp3) is evaluted

Special Operators:-

C supports some special operators such as Comma operator, size of operator, member or selection operator

Comma operator

It has lowest preority among all operators this operator is used to separate the expresions.

Ex:- int a=5, b=10, c=20;

Size of operators:

It returns the number of bytes of an operant an operant may be a variable, constant or datatype

Syntax:

size of (operator);

Ex: x=size of (int);

x = 2 (∵ size of int is 2 bytes)

member selection operators:

These are used to acces members of Pointers, structures, unions

Ex:- Variable member -1

Unary operator

A unary operator is an operation with only one operand i.e, (++, --)

The unary operators include, increment(++), decrement (--), address (&), logical not (!), size of operator.

Binary operator:-
These requires two operands and one operator between them.

Ex:- a*b , a/b

Progamme to implement Arithmatic operators 3|8|18

```
#include<stdio.h>
#include <conio.h>
void main()
{
 int a=10; b=15;
 float c;
 clrscr();
 c= a+b;
 Printf ("the value of c is %f", c);
 getch();
 }
```

Programme to implement logical operator

```
#include<stdio.h>
#include <conio.h>
void main()
{
 int num1=20 , num2=30 ;
 clrscr();
 if (num1>20 && num2>20)
 Printf ("This logical AND");
 if (num1>20 || num2>20)
 Printf ("This is logical OR");
 if (num1<20)
 Printf ("This is logical not");
 getch();
 }
```

Programme to implement Assignment operator.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a=5 b=2;
float d;
clrscr();
d = a/b;
Printf ("the value of d is %f", d);
getch();
}
```

Programme to implement all the Airthmatic operators

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a,b,x,y,z;
clrscr();
Printf ("enter the value of a and b");
Scanf ("%d %d", &a, &b);
x=a+b;
y=a-b;
z = a*b;
Printf ("the values of x = %d y = %d z = %d", x,y,z);
getch();
}
```

Programme to implement conditional operator.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a=10, b=50;
clrscr();
if (a>b)
Printf ("a is greater");
else
Printf ("b is greater");
getch();
}
```

Programe to implement increment and decrement operator

```
#include <stdio.h>
#include <conio.h>
void main()
{
int y, a=10;
clrscr();
y=a++;
Printf ("the value of y=%d a%d", y, a);
y=++a;
Printf ("the value of y=% a=%d", y, a);
y=--a
Printf ("the value of y=%d a=%d", y, a);
getch();
}
```

4-9-18

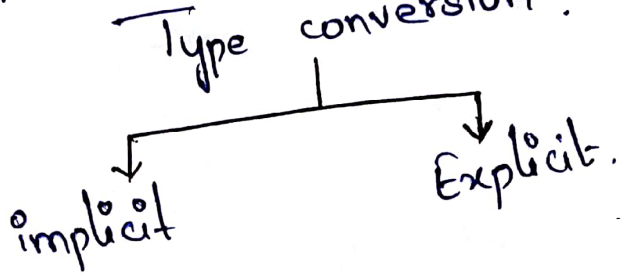
Programe to impliment bitwise operators

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a,b;
clrscr();
Printf("Enter the values of a&b");
Scanf("%d %d", &a, &b);
Printf("AND OPERATION : %d \n", a&b);
Printf("OR OPERATION: %d \n", a|b);
Printf("NOT OPERATION : %d \n", ~b);
getch();
}
```

Type Conversion:

Converting data from one form to anothe from is called type conversion.

* They are divided into 2 types.
  * Implicit
  * Explicit



Implicit

If the data is converted from one form to another automatically it is called Implicit type conversion.

```
Ex:- int m=15.
     float x=3.1, y;
     y=m*x;
```

Explicit

If the data is converted from one form to another by us:

If we want to convert data from one form to another we use following syntax.

```
Syntax:-
var1 = (datatype) var2

Ex:- int x=5;
     float y;
     y = x/2;
     y = (float) 5/2;
       = 2.5.
```

Program to impliment type conversion:

```
#include <stdio.h>
#include <conio.h>
Void main()
{
int a,b;
float c,d;
a = 1;
c = 3.14;
b = (int)c;
d = (float)a | (float) b;
Printf ("%f",d);
getch();
};
```

Expression and precidence and expression evaluation

Ex:= x=a*(b-c)+d

a=10, b=15, c=8, d=6

x = 10 * (15-8) + 6

= 10 * 7 + 6

= 70+6

=76

1st preference is given to bracket and evaluated

2nd preference is given to *

3rd preference is given to +

inorder to evaluate the above example according to the rules.

Highest priority of arthimatic operators.

/

%

*

lowest priority + , -

Rules of evaluation:

* perenthesized sub expression is evaluated from left to right.
* if the perenthesis is nexted the evalution begin with innermost sub expression.
* Determining the order the application of operator and evaluating sub expression.
* Associative rule is applied.
* Airthmatic operations are evaluted from left to right
* when perenthesis is used the expression within Priority assumes highest priority

Programme to implement relational operator:

```
#include<stdio.h>
#include<conio.h>
Void main()
{
int a,b,c,d,e,f,g,h;
clrscr();
Print("enter values of a,b");
scanf("%d %d",&a,&b);
```

```
c=a==b;
d=a!=b;
e =a>b;
f =a<b;
g =a>=b;
h=a<=b;
Printf ("It checks equal to %d ", c);
Printf ("It checks not equal to %d", d);
Printf ("It checks greater than %d", e);
Printf ("It checks less than %d", e);
Printf ("It checks greaterthan (or) equal to %d", g);
Printf ("It checks less than (or) Equal to %d", h);
getch();
}
```

Program to impliment all types of operators.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a,b;
clrscr();
Printf ("enter the values of a,b");
Scanf ("%d%d ", &a ,&b);
Printf ("The addition of two numbers %d \n", a+b);
Printf ("subtraction of 2 numbers %d \n", a-b);
Printf ("multiplication of 2 numbers %d \n", a*b);
Printf ("division of 2 numbers %d \n", a/b);
Printf ("modulus of 2 numbers %d \n", a%b);
Printf ("greater than %d \n", a>b);
```

```
Printf ("less than %d \n", a<b);
Printf ("greater than (or) equal to %d \n", a>=b);
Printf ("less than (or) equal to %d \n", a<=b);
Printf ("equal to %d \n", a==b);
Printf ("not equal to %d \n", a!=b);
Printf ("the post increment of a is %d \n", a++);
Printf ("the pre increment of a is %d \n", ++a);
Printf ("post decrement of b is %d \n", b--);
Printf ("pre decrement of b is %d \n", --b);
Printf ("logical AND is %d \n", a&&b);
Printf ("logical OR is %d \n", a||b);
Printf ("logical NOT is %d \n", a!b);
getch();
}
```

Storage classes:

There are 4 different types of storage classes

* Automatic
* Register
* static
* external

They help us to understand about the memory allocation, default value, scope, lifetime of variables

if we donot sapecify any type it is atomatically considered as auto

Syntax:=

storageclass datatype variablename;

Ex:- Auto int a;

Auto variables:=

* Auto variables are always local variables.
* they occupy space in the primary memory.
* They are alive only untill the block in which they are defined is in control.
* default of Value of Autuo variable is Garbage.

Programme to impliment Auto storage class:

```
#include<stdio.h>
#include<conio.h>
auto int j; ⟶ error
Void main()
{
auto int a=10; → method scope
clrscr();
{
auto int a; ⟶ block scope.
Printf ("a=%d\n", a); ⟶ Garbage value.
}
Printf ("a=%d\n", a); ⟶ 10
getch();
}
```

Register :=

Register type storage class is used to axcess the variables much faster and it can be used in a case where a variable is repeatedly used.

* The scope of register variables is confined to the block in which they are defined.
* Variables are allocated space in the C.P.U registers then default value is garbage value.
* The Key word used is register
* Lifetime of register variables is till the control remains in the block in which it is defined

Syntax:-

register int a;

Static:-

Syntax:- static int x;

* Static variables are allocated space in the Primary memory.

* Default value is zero

* Scope of static variables is confined to the block in which they are defined

* The value of variable remains as long as the completion of the programme

Programme to implement static variables

```
#include <stdio.h>
#include <conio.h>
void main ()
{
increment ();
increment ();
increment ();
getch();
}
intrement ()
{
static int a=0;
a =a+1;
Printf ("a = %d", a);
getch() return 0;
}
```

external:-

Keyword used is extern

Extern variables are allocated memory in main memory

default value is zero

Extern variables are accessable throughout the Programme (global variables)

The value of extern variables remains as long as the completion of the programme.

Syntax:-

Extern int x;

Programme to implment extern variables.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int x=3;
extern int y;
Printf ("x = %d", x); →3
Printf ("y = %d", y); →10
getch();
}
y=10;
```

Command line argument :=

6/9/22

To pass command line arguments, we typically define main() with two arguments.

1st argument is the no: of command line arguments

2nd is list of command line arguments.

int main(int argc, char *argv[])

Argc: Argument count

It is int and stores no: of Command line arguments passed by the user including the name of the programme.

Value of Argc should non-negative

Argv: Argument vector

It is array of character pointers listing the arguments. if argc is greater than 0, the array elements from Argv[0] to Argv[Argc-1] will contain pointers to strings.

argv[0] is the name of the programme, after that till argv[Argc-1] every element is command line arguments.

Properties of command line arguments:

→ They are passed to main().

→ They are parameters/arguments supplied to the programme when it is invoked

→ They are used to control programme from outside instead of hard coding those values inside the code.

→ Argv[Argc] is a null pointer

→ argv[0] holds the name of the programme

→ argv[1] points to the first command line argument and argv[n] points last arguments.

cmd > hello <> hello 10 c r 6

argv[0] argv[1] argv[2] argv[3] argv[4]

Programme to impliment command line arguments.

```
#include <stdio.h>
#include <conio.h>
Void main (int argc , char * argv[])
{
Printf (" Programm name is: %s", argv[0]);
if (argc == 1)
Printf ("No extra command line arguments passed
        except program name");
  if (argc>=2)
  {
   Printf ("no: of arguments passed %d", argc);
  }
  getch();
  }
```

instructions to be given in command prompt

```
cd \
cd tc
cd bin
tcc programme.c
Programme arg1 arg2 arg3 ....
```

Conditional branching and looping

There are different conditional statemen in C. They are.

* if
* if else.
* else if ladder
* switch.

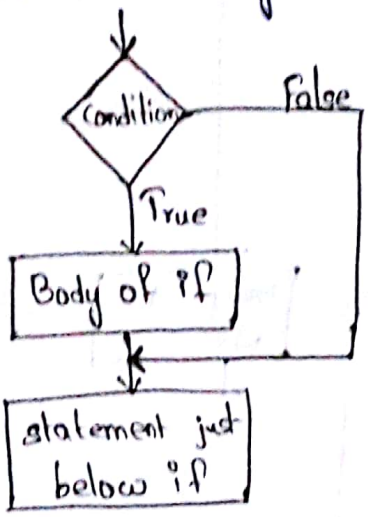
- If

Syntax:-

```
if (condition / test expression)
{
 statements;
}
 statements
```

The if statements evaluates the test expression / Condition inside the peranthesis. If the test expression is evaluated to true (non-zero), statemen inside the body of if is exicuted

if the test expression is evaluated to false statement inside the body of if is skiped.



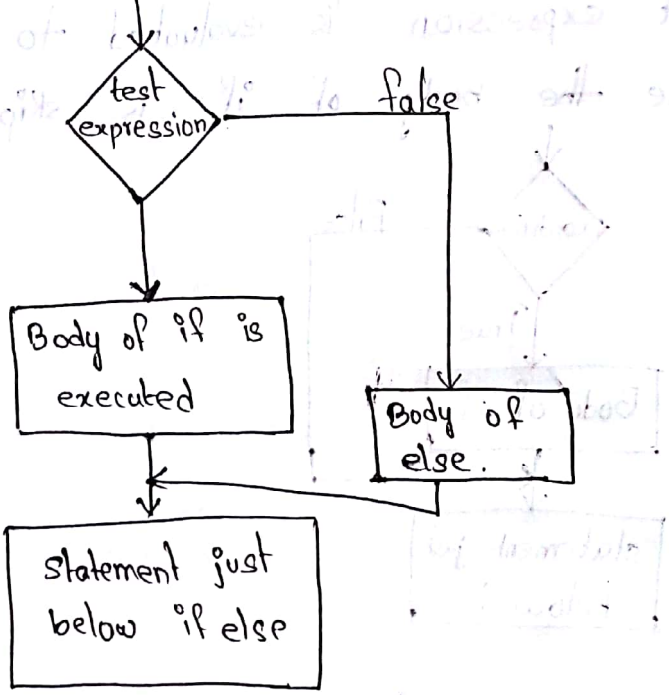
* if else

Syntax:

```
if (test expression)
{
statements / body of if ;
}
else.
{
Statements / Body of else ;
}
statements;
```

the if else statements evaluates the test expression inside the paranthesis. if the test expression is evaluted to True statement inside the body of if is exciuted. and statements inside the body of else is skiped

if the test expression is evaluted to false Statement inside the body of if is skiped and Statement inside the body of else is exciuted.



Programme to (impliment) print even or odd.

```
#Include <stdio.h>
# include <conio.h>
void main()
{
int n;
Printf ("Enter n value");
Scanf ("%d", &n);
if (n%2==0)
{
Printf ("n=%d is even", n);
}
else
{
Printf ("n=%d is odd", n);
}
getche();
}
```

To print a number is +ve

```
#include <stdio.h>
#include <conio.h>
void main()
{
int n;
Printf ("enter the n value");
Scanf ("%d ", &n);
if(n>0)
 {
 Printf ("n =%d is positive ", n);
 }
 getch();
 }
```

Else if ladder:

```
if (test exp1)
{
statement ;
}
else if (test exp 2)
{
 statement ;
}
else if (test exp 3)
{
 statement;
}
else
{
 statement;
} —
```

goto:

goto statement is used to transfer or jump the control from one statement to the other in the programm

Syntax:

```
goto label;
- - -
- - -
statements;
label:
```

forward jump

```
label:
statements;
- - -
- - -
- - -
goto label;
```

Backward jump

Programme to implement goto.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i=1;
xyz:
i++;
if (i<=10)
{
Printf ("%d",i);
goto xyz;
}
```

Switch

Switch case statements are a substitute for long if statements that compare a variable to several integral values.

→ The switch statement is a multiway branch statement

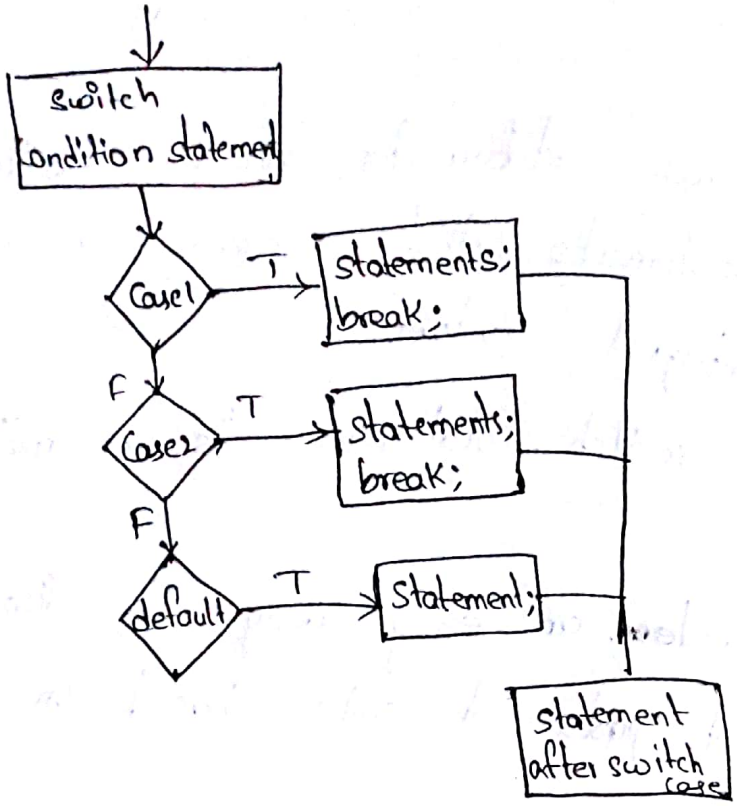
→ It provides an easy way to dispach execution to different parts of code based on the value of expression.

→ Switch is a control statement that allows a value to change control of execution

Syntax :=

```
Switch (n)
    {
    Case 1 : // code to be executed if n=1
     statements;
     break;
    Case : // codes to be executed if n=2
     statements;
     break;
    default : // if n doesnot match any case
    }
```

Flowchart



Progro

Constant expressions are allowed in switch case

Ex :- Switch (1+2+3)
Switch (1*2+3%.4)

Switch (ab+cd) } not valid
Switch (a+b+c)

Variable expressions are not valid in switch

Simple programme to implement switch case:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int x=2;
switch (2)
{
Case 1: Printf ("choice is 1");
break;
```

```
case 2: Printf ("choice is 2");
break;
default:
  printf ("out of range");
  }
  getch();
  }
```

output: choice is 2.

write a C program for all airthmatic operators using switch.

```
#include <stdio.h>
#Include <conio.h>
Void main()
 {
 int a,b,c;
 char oper;
 Printf (" enter the operator");
 Scanf ("%c", &oper);
 Printf ("enter the value of a,b");
 scanf ("%d %d", &a, &b);
 Switch (oper)
  {
  Case '+' : c=a+b;
   Printf (" sum is %d\n", c);
   break;
  Case '-' : c=a-b;
   Printf (" difference is %d\n", c);
   break;
   Case '*' : c=a*b;
    Printf ("multiplication is %d\n", c);
```

```
break;
Case '/': c = a/b;
        Printf ("division is %d\n", c);
        break;
    default;
        Printf ("operator is not valid");
        break;
        }
        getch();
        }
```

formated io:

It refores to the conversion data to and form a stream of characters for printing in plane text formate

Output with printf:

Printf ("format string", variable name);

Conversion specifier:

It is a symbol that is used as a place holder in a formating string

%d → Int (signed decimal integer)

%u → unsigned int

%f → float, double

%c → character

%s → string

Scanf

Syntax:-

Scanf ("format string", list of variables & address);

Standard input and output stream:-

→ i/o is essentially done one character or bit at a time.

Stream:

A sequence of characters flowing from one point to another.

when the main function of ur program is invoked it already has predefined streams open and available for use. These represents stdio.h channels that have been established for a Process.

These stream files are declared in the header stdio.h.

Variable : file *stdin

—the std input stream which is the normal source of input for the programme (default is Keyboard)

Variable: file *stdout

The stdoutput stream which is used for normal output from the programme (default is monitor)

Variable: file * stderr

The stderr stream which is used for digonasis and displaying of error messages.

Write a C programme to find the given is leap year or not

```
#Include <stdio.h>
#include <conio.h>
void main()
{
Int year;
Printf ("Enter the year");
Scanf ("%d", & year);
if (year % 4 ==0).
{
Printf (" the given year is leap year %d", year);
}
else
{Printf (" It is not leap year");
}
getch();
}
```

To find the ~~given~~ largest among two numbers.

```
#include<stdio.h>
#include <conio.h>
void main()
{
int num1, num2;
Printf ("enter numbers");
Scanf ("%d %d", & num1, & num2);
if (num1 > num2)
{
Printf (" num1 is largest %d", num1);
}
else
{
Printf (num 2 is largest %d", num2);
}
getch();
```

write a C programme to calculate total of marks, Print average & grade.

```
#include <stdio.h>
#include<conio.h>
Void main()
{
int  s1,s2,s3,s4;
int tot;
float avg;
char grade;
clrscr();
Printf("enter the marks in s1,s2,s3,s4");
Scanf("%d%d%d%d", &s1, &s2, &s3, &s4);
tot = s1+s2+s3+s4;
Avg. = tot/4;
if (avg >= 60)
{
grade = 'A'
}
else if (avg>=50 && avg<60)
{
grade = 'B';
}
else if (avg>=40 && avg<50)
{
grade = 'c';
}
else
{
grade = 'D';
}
Printf("total marks%d", tot);
Printf("Average %f", Avg);
Printf("The grade is %C", grade);
getch();
}
```

for

the general form of for loop is

syntax:-

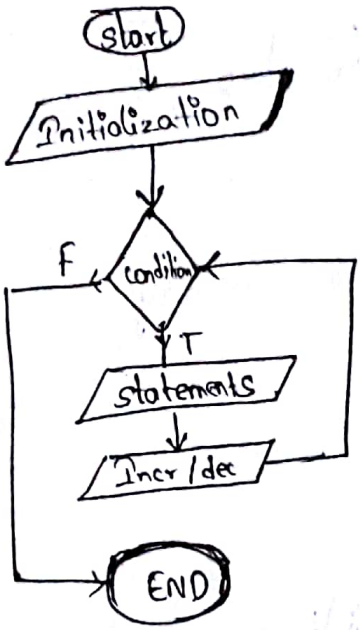
```
for (initialization ; condition ; modify)
{
body of loop / statements ;
}
```

In for loop first it initialises the values and later checks the condition if it is true. and execute the body of the loop. if the condition is false the loop is terminated. the control variable is incremented or decremented the new value is again tested if the condition is true. the Process continues and stops when condition becomes false.

Programme to calculate factorial of given number.

```
#include<stdio.h>
#include<conio.h>
void main( )
{
int n, i, fact=1;
clrscr();
Printf ("enter the value of n");
Scanf ("%d", &n);
for (i=1; i<=n; i++)
{
fact = fact *i;
}
Printf (" the value of factorial of %d is %d ", n, fact);
getch();
}
```

Flowchart:-



While:

The while is an entry control statement if the test control is true the body of the loop is executed.

write a Cprogramme to calculate sum of n natural numbers using while.

```
#Include<stdio.h>
#include<conio.h>
void main()
{
int i=0, sum=0, n;
clrscr();
Printf("enter the value of n");
Scanf("%d", &n);
while (i<=n)
{
Sum = sum +i;
i++;
}
Printf ("sum is %d", sum);
getch();
}
```

write a c programme to calculate sum of the squares of all integers.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int i=0, n, sum=0, a;
clrscr();
Printf("enter the value of n");
Scanf("%d", &n);
while (i<=n)
{
Sum+=i*i;
i++;
}
Printf("The sum is %d\n", sum);
getch();
}
```

do while:

It is similar to that of while except that it always executes the statements atleast once. in do while after executing statement the condition will be checked.

Syntax:-

```
do
{
statements;
incre / decrement;
}
while (condition);
```

Programme to impliment to while.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i=1;
clrscr();
do
{
Printf("%d\n", i);
i++;
}
while(i<=10)
Printf("out of loop");
getch();
}
```

10/9/18

ARRAYS UNIT-II

An array is a fixed size of sequence or collection of elements of same datatype. It is simply a grouping like data types / like type data.

Some examples where the concept of array can be used

→ list of temp recorded every hour in a day/moth/year.
→ list of employees in an organisation.
→ test score of a class of students.
→ list of customers and tellphone numbers.

Syntax:=

datatype Indentity [size];

(or)

datatype Variablename [size];

we can use arrays to represent not only simple list of values but also tables of data in two, three or more dimensions.

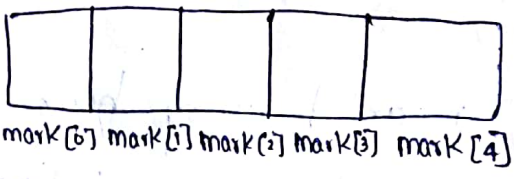
The following are the types of arrays.

(i) One dimensional array
(ii) 2 - dimensional array
(iii) multi- dimensiond array.

Accessing an array:-

You can access elements of an array by indices. Suppose you declared any array "mark" as given below → mark [5].

first element is mark of [0]



Few Key notes:

→ Array have 'zero' as the first index not '1'

→ If the size of an array is n then to access the last element [n-1] index is used

→ Suppose the starting address of mark[0] is 2120d then the next address a[1] 2124d ...

→ So it is of type float as it allocated 4 bytes each

→ Initalization of values into array:-

→ Int mark[5]= {10, 30, 40, 60, 80};

Programe to (impliment) calculate average of ~~two~~ numbers using arrays.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int marks[10], i, n, sum=0;
float average;
Printf ("enter the number of numbers");
Scanf ("%d", &n);
```

```
for (i=0 ; i<n ; i++)
{
Printf ("enter number %d ", i+1) ;
Scanf ("%d" , &marks [i]);
Sum = Sum + mark [i];
}
average = sum/n;
Printf (" Average = %d", Average);
getch();
}
```

11/9/18

Two-dimensional Array :=

In C programming you can create a array of arrays known as multi-dimensional arrays.

→ int a [3][4]

here a is a two dimensional array and it can hold 12 elements. you can think the array has table with 3 rows and 4 coloumns.

→

	C_1	C_2	C_3	C_4
R_1	a[0][0]	a[0][1]	a[0][2]	a[0][3]
R_2	a[1][0]	a[1][1]	a[1][2]	a[1][3]
R_3	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Intialization of two-deminsional arrays:-

There are different ways for intializations.

```
int c [2][3] = { {1, 3, 0} , {-1, 5, 9} }
int c[][3] = {{1,3,0}, {-1,5,9}}
int c[2][3] = {1,3,0,-1,5,9}
```

Programme to add two matrices.

```
#include<stdio.h>
#include<conio.h>
Void main()
{
int r,c, a[100][100], b[100][100], sum[100][100],i,j;
clrscr();
Printf("Enter No: of rows and columns (between 1 and 100):");
Scanf("%d%d", &r, &c);
Printf("\n Enter elements of 1st matrix:\n");
for(i=0; i<r; i++)
{
for(j=0; j<c; j++)
{
Printf("Enter element a %d %d", i+1, j+1);
Scanf("%d", &a[i][j]);
}
}
Printf("\n Enter elements of 2nd matrix:\n");
for(i=0; i<r; i++)
{
for(j=0; j<c; j++)
{
Printf("Enter element b %d %d", i+1, j+1);
Scanf("%d", &b[i][j]);
}
}
for(i=0; i<r; i++)
{
for(j=0; j<c; j++)
{
```

```
sum[i][j] = a[i][j] + b[i][j];
}
}
Printf ("sum of two matrix is :\n");
for (i=0; i<r; i++)
{
for (j=0; j<c; j++)
 {
 Printf ("%d \n", sum[i][j]);
  if (j==c-1)
     {
     Printf ("\n\n");
     }
     }
     }getch();
      }
```

Manupulating on elements of an array:-

Interchaning the position of elements in an array is called manupulating an array

write the c programme to find the transpose of a matrix

```
#include<stdio.h>
#include<conio.h>
Void main()
{
int a[10][10], transpose[10][10], r, c, i, j;
Printf ("Enter no: of rows and columns");
Scanf ("%d %d", &r, &c);
Printf ("\n enter elements of matrix :\n");
for (i=0; i<r; i++)
```

```
{
for (j=0 ; j<c ; j++)
 {
 Printf ("Enter element a %d %d", i+1 , j+1);
 Scanf ("%d ", &a[i][j]);
 }
 }
 Printf ("Entered matrix : \n");
 for (i=0; i<r ; i++)
  {
  for (j=0; j<c ; j++)
   {
    Printf ("%d ", a[i][j]);
    if (j==c-1)
    Printf("\n\n");
    }
    }
    Printf ("Transpose of matrix : \n");
    for (i=0; i<r ; i++)
     {
      for(j=0; j<c; j++)
       {
        transpose[j][i] = a[j][i];
        }
        }
         for (i=0; i<c; i++)
          {
           for (j=0; j<r; j++)
           {
            Printf ("%d " ,transpose [i][j]);
```

```
if (j==r-1);
Printf ("\n\n");
}
}
getch();
}
```

String :=

Strings are defined as an array of characters the difference b/w a character array and string is that the string is terminated with a special character '\0'

Declaration of a string.

Declaring a string is as simple as declaring a one dimensional array

Syntax

```
char str_name [size];
```

Please not that there is an extra terminating character which is the null character [\0] use to indicate termination of string which differs strings from normal character arrays.

Initialization:

```
char str[] = "Hello";
char str[50] = "Hello";
char str[] = {'H','E','L','L','O'};
char str[14] = {'H','E','L','L','O'};
```

The c language doesnot provide inbuilt data types for strings but it has an access specifier "%s" which can be used to directly print and read strings.

Ex:-

```
#include <stdio.h>
#include <conio.h>
void main()
{
char str[50];
Scanf ("%s", str);
Printf ("%s", str);
getch ();
}
```

we Know that the '&' is used to provide the address of the variable to the scanf function to store the value read in the memory as str[] is a character array so using str without braces with give the base address of string that is the reason we have not used '&' in this case as we are already providing the base address of the string to scanf.

String manupulation:-

you need to often manupulate string acc to the need of a problem. most, if not all, of the string manupulation can be done manually but this makes programming complex and large to solve this, C supports a large no. of string handling functions in the standerd library "string.h"

* strlen() ⟶ Calculates length of the string.
* strcpy() ⟶ Copies a string to another string
* strcat() ⟶ Concatenates (joins) two string

function gets and puts() are two string functions to take string input from the user and display it respectively

It is present in stdio.h

ex:-
```
#include <stdio.h>
#include <conio.h>
{
char name[30];
clrscr();
Printf ("enter name");
gets (name); // to read string from user
Printf ("Name");
Puts (name); // function to display string
getch();
}
```

Strlen()

The string length function calculates the length of the string. The function takes a single argument i.e the string variable whose length is used to find and returns the length of string fast. The strlen() is defined in <string.h>

Ex:- programe to find length of a string.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
Void main()
{
char a[20] = "Program";
char b[20] = {'P','R','O','G','R','A','M','\0'};
char c[20];
Printf ("Enter string");
gets(c);
Printf("length of string a = %d \n", strlen(a));
Printf ("length of string b= %d \n", strlen(b));
Printf ("length of string c= %d \n", strlen(c));
getch();
}
```

strcpy()

This function copies the string to the another character array

Syntax:-

```
char *strcpy (char * destination, const char*source)
```

The strcpy() copies the string pointed by source (including the null character) to the character array destination. the function returns character array didination. ~~the~~ ~~Printf~~

→ strcpy() is defined in <string.h>

Ex:- Program to implement strcpy();

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
Void main().
{
char str1[10] = "awesome";
char str2[10];
char str3[10];
 Strcpy (str2, str1);
 Strcpy (str3, "well");
 Puts (str2);
 Puts (str3);
 getch();
}
```

strcmp()

It compares two strings and returns(0) if both strings are identical.

Syntax:-

int strcmp(const char *str1, const char *str2);

* the strcmp functions takes two strings and returns an Integer.

The strcmp function compares two strings char by char. If the first char of two strings are equal, next char of two strings are compared this continous untill the corresponding characters of two strings are different or a null character '\0' is reached. It is defined in <string.h>

return value	Remark
0	if both strings are equal.
Negative	if ascii value of first unmatche char is less than second.
Positive	if ascii value of first unmatche char is greater than second

Programme to compare given two strings

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main()
{
char str1[10] = "abcd", str2[10] = "aBcd", str3[10] = "abcd";
int result;
clrscr();
result = strcmp(str1, str2);
Printf("strcmp(str1, str2) = %d \n", result);
result = strcmp(str1, str3);
Printf("strcmp(str1, str3) = %d \n", result);
getch();
}
```

Strcat()

The function strcat() concatinates (joines) two strings.

It is defined in <string.h>

Syntax:=

```
char* strcat(char* dest, const char* src);
```

It takes two arguments i.e, two strings or char arrays, and stores the resultant concatinated string in the first string specified in the argument.

The pointer to the resultant string is passed as returned value

Program to concatinate given two strings.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main()
{
char str1[10] = "This is", str2[10] = "ECE-C";
strcat (str1, str2);
Puts (str1);
Puts (str2);
getch();
}
```

Arrays of strings :-

A string is a 1-D array of characters, so an array of strings is 2-D dimension array of characters

Syntax :-

```
char ch-arr[3][10] = {{'S','P','I','K','E',\0},{T,O,M,\0},
                      {'J','E','R','R','Y',\0}};
```

No: of string

length of each string

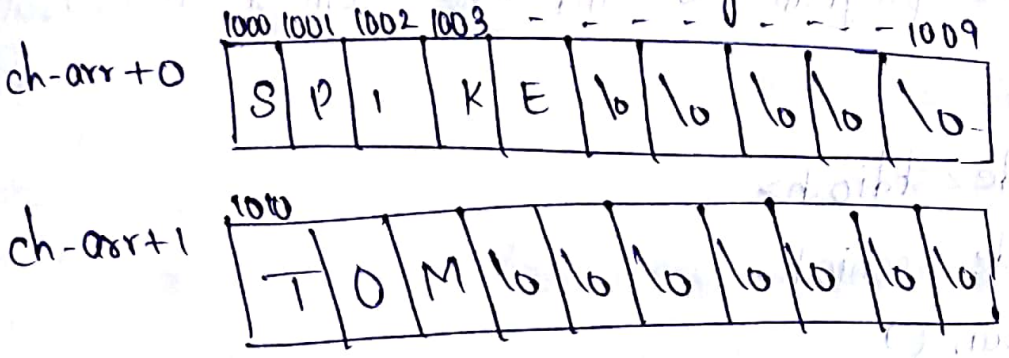
(or)

```
Char ch-arr[3][10] = {"SPIKE", "TOM", "JERRY"};
```

The first subscript of array i.e, 3 denotes the no: of strings in the array and second subscript holding value 10 denotes the maximum length of string.

Each character occupies one byte of data so when the compiler sees the above statement it

allocates [3×10] bytes of memory



Program to print array of strings and their addresses

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
 void main()
{
char ch[3][10], i;
clrscr();
Printf ("Enter 3 strings");
for (i=0; i<3 ; i++)
 {
 Scanf ("%s", ch[i]);
 }
 Printf ("strings are");
 for (i=0, i<3 , i++)
  {
   Printf ("%s address = %u \n", ch[i] , ch[i]);
  }
  getch();
 }
```

Write a program to find the sum of n natural numbers.

```
#include<stdio.h>
#include<conio.h>
Void main()
{
 int n, i, sum=0;
 clrscr();
 Printf("Enter a positive integer");
 Scanf("%d", &n);
 for(i=1; i<=n; i++)
 {
  sum = sum + i;
 }
 Printf("sum = %d", sum);
 getch();
 }
```

Structure

It is a collection of variables of different types under a single name.

Ex:- If you want to store some information about a person, i.e. the name, citizenship no, salary you can easily create different variables, name, salary, CITNO to store these information separately with the help of structures.

If you want to store information about mulliple Persons you need to creat different variables. for each information Per person.

name1, citno1, salary1, name2, citno:2, salary2

this collection of all related information under a single name person is a structure.

→ Keyword "struct" is used for creating a structure

```
struct structure name.
{
datatype member1;
datatype member2;
   .
   .
   .
  datatype member n;
};
```

Structure variable declaration and acessing

when a struct is defined, it creats a userdefined type but no storage or memory is alocated.

Ex:- 1.

```
struct emp
  {
  int eno;
  char ename [20];
  float esalary;
  };
void main()
{
struct emp emp1, emp2;
getch();
}
```

Ex:- 2;

```
struct emp
{
 int eno;
 char e nam [20];
 float esalary;
} emp1, emp2;
```

In the both cases two variables emp1, emp2 of type struct emp are created.

member operation (.) is used for accessing members of a structure.

Initializing structure variables.

we can directly initialize the structure variables as shown below.

Ex:- struct emp emp1 = {1000, "RAM", 63047};

Basic program to impliment structures.

```
#include<stdio.h>
#include<conio.h>
void main()
struct emp
{
int eno;
char ename[20];
float esalary;
};
void main()
{
struct emp e={1001,"RAM.",5000};
Printf ("Emp details:\n");
Printf ("Eno %d\n", e.eno);
Printf ("ename %s\n", e.name);
Printf ("esalary %f\n", e.salary);
getch();
}
```

Program to find size of structure.

```
#include <stdio.h>
#include <conio.h>
struct emp
  {
   int eno;
   char ename[20];
   float esalary;
  };
  void main()
   {
    struct emp e;
    Printf("%d", size of(e));
    getch();
   }
```

Array of structures:

Declaring an array of structure is same as declaring an array of fundamental type.

Since an array is collection of elements of the same type.

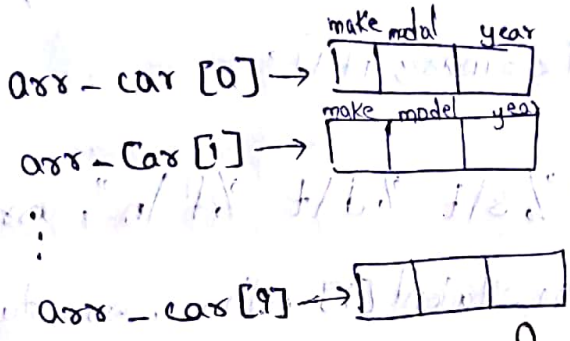
Then array of structures, each element of an array is of the structure type.

Syntax:-

struct car arr=car[10]

here, arr_car is an array of 10 elements where each element is of type struct car. we can use arr_car to store 10 structure variables of type struct car. to Access individual elements we will use subscript notation ([]) to acces the members of each element we will use (.) operator as usual.

struct car arr_car[10]



Program to impliment array of structure.

```
#include<stdio.h>
#include <conio.h>
#include <string.h>
# define max 2
 struct student
 {
  char name[20];
  int roll no;
  float marks;
  };
  void main()
  {
   struct student arr_student [max];
   int i;
   for (i=0; i<max; i++)
   {
   Printf ("\n Enter details of student %d \n\n", i++);
   Printf (" Enter name"):
```

```
Scanf ("%s", arr_student[i].name);
Printf ("Enter roll no");
Scanf ("%d", &arr_student[i].rollno);
Printf ("Enter marks");
Scanf ("%f", &arr_student[i].maks);
}
Printf ("\n\n");
Printf ("name\t rollno\t marks\n");
for(i=0; i<=max; i++)
{
Printf ("%s\t %d\t %f\n", arr_student[i].name,
        arr_student[i].rollno, arr_student[i].marks);
}
getch();
}
```

Unions:-

Unions are quite similar to structures. like structures unions are also derived types.

Syntax:

```
union car
{
char name[50];
int price;
};
```

defining a union is as easy as replacing the Keyword struct with the Keyword union.

union variables can be created in similar manner as structure variables

Ex:-
```
union car
{
char name [50];
int price;
}car1, car2, *car3;
```

or

```
union car
{
char name [50];
int price;
};
void main()
{
union car car1, car2, *car3;
getch();
}
```

In both cases union variables car1, car2 and union pointer variable car3 of type union car is created

Accessing members of a union:-

Again, the member of unions can be accessed in similar manner as structure.

In the above example suppose you want to access price for union variable car1. It can be accessed as car1.prize

Difference between union and structure.

The amount of meomery required to store a structure variable is the sum of meomory size of all membess.

But, the memory required to store a union variable is the memory required for the largest element of an union.

In the case of structures all of its membes can be accessed at time but in case of unions only one of its membess can be accesse at a time and all other membess will contain garboge values.

Program to implement unions:

```
#include<stdio.h>
#include<conio.h>
union un
{
  short a;
  Short b;
}
void main()
{
  union un var;
  var.a = 10
  Printf ("%d", var.b);
  var.b = 20;
  Printf ("%d ", var.a);
  getch();
}
```

Program to demonstrate primary difference between structure & union.

```
#include<stdio.h>
#include<conio.h>
union union job
{
char name[32];
float salary;
int workerno;
}u job;
struct struct job
{
char name[32];
float salary;
int workerno;
}sjob;
void main()
{
Printf("size of union = %d", size of(ujob));
Printf("size of structure = %d", size of(sjob));
getch();
}
```

Pointers

Pointers in c language is a variable that stores / points the address of another variable.

A pointer in c is used to allocate memory dynamically i.e, at run time. the pointer variable might be belonging to any datatype such as int, float, char, double short, e.t.c.

Syntax:-

datatype * variable;

(or)

datatype *variable;

Eg:- int *p;

where * is used to denote that p is a pointer variable and and not a not a normal variable.

Use of pointers:

→ Pointers are used to acess memory and manuplate the address.

→ To create dynamic datastructures

→ To pass and handle variable parameters passed to functions.

→ To acess information stored in arrays

Program to implement pointers:

```
#include <stdio.h>
#include <conio.h>
void main()
{
int i=100;
int *ptr;
Ptr = &i;
Printf ("%d", i);
Printf ("%d", ptr);
Printf ("%d", &i);
Printf ("%d", &ptr);
Printf ("%d", *ptr);
Printf ("%d", *(&i));
getch();
}
```

Types of pointers:=

There are two types of pointers they are typed and untyped pointers.

Typed pointers:

Points to specific type of data.

int * → int data

struct emp * → Employee data.

Untyped pointers:

Points to any type of data.

(Generic pointer) void * → Any data

&, * are the symbols to be focused in the case of pointers.

& → stores address of a variable

* → stores value of variable.

Pointers to pointer:

A pointer to pointer is a form of multiple in direction. or a chain of pointers.

Normally A pointer contains the address of a variable. when we define a pointer to pointer the first pointer contains the address of the second pointer, which points to the address of location that contains the acutally value as shown below.


```
int **a;
```

A variable that is a pointer to pointer must be declared as astrick. this is done by placing an additional asterick

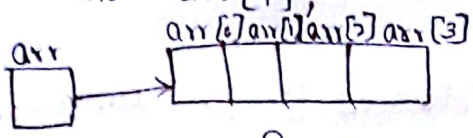
Pointer to arrays.

Arrays are closily related to pointer in C but the important deffirence blw them is that a pointer variable takes different addresses

as values where as in the case of array it is fixed.

Relation between Arrays and pointers:

Consider an array int arr[4];



In C programming name of the array always points to address of first element of array.

In the above example arr and arr[0] and &arr[0] points to the address of the first element

&arr[0] is equivalent to arr.

Since the address of both are same the values of arr and &arr[0] are also the same.

arr[0] is equivalent to *arr
[value of an address of pointer]

Similarly

arr = &arr[0]	*arr = arr[0]
arr+1 = &arr[1]	*(arr+1) = arr[1]
arr+2 = &arr[2]	*(arr+2) = arr[2]
⋮	⋮
n	n

In C you can declare any array and can use Pointer to alter the data of any array.

```
/* Program to find sum of n numbers with arrays and pointers */
#include <stdio.h>
#include <conio.h>
void main()
{
int i, classes [20], sum = 0, n = 15;
Printf ("enter numbers: \n")
for (i=0; i<n; i++)
 {
  Scanf ("%d ", (classes +i));
  Sum= Sum .+ * (classes +i)};
  }
  Printf ("sum = %d ", sum);
  getch();
  }
```

* In this program the elements are stored in the integer array data then using the for loop each element is traversed and print using the pointer method.

Pointer to Structures :=

Structures can be created and exessed using pointers a pointer variable of a struct can be created as below.

```
struct name
 {
  member 1;
  member 2;
  member 3;
 };
 void main()
  {
   struct name * ptr;
  }
```

here the pointer variable of type struct name is created. ~~Axis~~

Accessing structure members through pointer:

A structures member can be accessed through pointer in two ways.

1. Referencing pointer to another address to acess memory
2. Using dyramic memory allocation.

Program to acess structure members using pointers.

```
#include<stdio.h>
#include<conio.h>
void main()
 struct Person
  {
   int age;
   float weight;
  };
  void main()
  {
   struct person *PersonPtr, Person1;
```

```
Person Ptr = & person1;
Printf ("Enter integer");
Scanf ("%d", &(*person Ptr).age);
Printf ("Enter number");
Scanf ("%f", & (*Person Ptr).weight);
Printf ("Display");
Printf ("%d %f", (*person Ptr).age, (*personptr).weight);
getch();
}
```

In this example. the pointer variable of type struct person is referenced to the address of Person1. Then only the structure member through Pointer can be acessed.

Structure pointer member can also be acessed using arrow (→) operator

(*personptr.age) is same as PersonPtr → age

(*personPtr.weight) is same as PersonPtr → weig

Self referencial structures in C:-

~~Structures can be~~ A self referencial structures are those structures that have one or more Pointers which points to the same type of Structure as their members. In other words structures pointing to same type of structures are called self refericial structures

Ex:=

```
struct node
{
int data1;
char data2;
struct node *link;
};
void main()
{
struct node obj;
getch();
}
```

In the above example link is a pointer to a structure of type node. hence this structure node is a self referencial structure with link as a referencing pointer.

Types of self referencial structures:

* Self referencial structures with single link.
* Self referencial structures with multiple link.

Self referencial structure with single link.

These structures can have only one self pointer as their member.

the following example will show us how to connect the objects of a self referencial structure with single link and access the corresponding data member. the connection formed is shown in following fig.

Program to implement self referencial structures with single link.

```
#include<stdio.h>
#include<conio.h>
struct node
{
int data1;
char data2;
struct node *link;
};
void main()
{
struct node obj1;
obj1.link = NULL
obj1.data1 = 10;
obj1.data2 = 20;
struct node obj2;
obj2.link = NULL
obj2.data1 = 30;
obj2.data2 = 40;
obj1.link = &obj2;
Printf("%d", obj1.link -> data1);
Printf("%d", obj1.link -> data2);
getch();
}
```

Enumeration in C (enum):-

Enumeration is a userdefined datatype in C It is mainly used to assign names to integral constants, the names make a program easy to read and main

Syntax:- enum

```
{working=1, failed=0;
};
```

The Keyword enum is used to declare new enumeration in C.

following is an example of enum declaration.

enum flag { constant 1, constant 2 ... };

The name of enumeration is flag and the constant are the values of the flag.

By default the values of the constant are as follows.

constant 1 = 0
constant 2 = 1
constant 3 = 2 .....

Variables of type enum can also be defined in two ways

enum week {mon, Tue, wed};
enum week day;

(or)

enum week {mon, Tue, wed} day;

An example program to demostrate working of enum in C

```
#include<stdio.h>
#include<conio.h>
enum week{mon, tue, wed, thu, fri, sat, sun};
void main()
{
 enum week day;
 day = wed;
 Printf ("%d", day);
 getch();
}
```

output ⇒ 2

In the above example. we declared "day" as the variable and the value of "wed" is allocated to day, which is 2. so, as a result 2 is printed

Interesting facts about initialization of enum.

→ Two enum names can have same value.

forexample in the following C program both "failed" and "freezed" have same value zero

```
#include<stdio.h>
#include<conio.h>
void main()
enum state {working=1, failed=0, freezed=0};
Void main()
{
Printf("%d %d %d", working, failed, freezed);
getch();
}
```

output:=

100

→ If we donot explicitly assign values to enum name, the compiler by default assign value. Staring from zero.

→ we can assign value to some name in any order all unassigned names gets value as value of previous name + 1

```
#include<stdio.h>
#include<conio.h>
enum day {sun=1, mon, tue = 5, wed, thur = 10, fri, sat};
void main()
{
Printf("%d, %d, %d %d, %d, %d, %d", sun, mon, tue, wed,
getch();                                thur, fri, sat);
}
```

Output = 1, 2, 5, 6, 10, 11, 12.

→ The value assigned to enum names must be some integeral constant i.e, the value must be in range from min + integer value to max + integer value

→ All enum constants must be unique in their scope

for example the following program fails in compilation.

```
enum state {woking, failed};
enum result {failed, passed};
void main()
{
 getch();
}
```

Output

Complitation error: 'failed' has a previous declaration as state failed.

Strstr

strstr() is a predefined function used for String handling string.h is the headerfile. used for string functions.

This function takes two strings s_1 & s_2 as an argument and finds the first occurance of the substring s_2 in the string s_1

The process of matching doesnot include the terminating null characters '\0' but function stops their

Syntax:-

```
strstr (s1 , s2);
```

Program to implement strstr

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
Void main ()
{
char str[20] = "this is ECE";
 char * sub;
 clrscr ();
 Printf ("enter sample");
 Scanf ("%s", sample);
 Sub = strstr (str, sample);
 Printf ("%s", sub);
 getch ();
 }
```

UNIT - III

PREPROCESSER COMANDS

The C preprocesser is the macro processer that is used automatically by the C compiler. to transform the our program before actual compilation it is called a macro processer because it allows you to define macros which are brief abbrivations for longer

A macro is a segment of code which is replaced by the value of macro. macro is defined by #define directive.

Preprocessing directives are lines in your program that start with '#'. '#' is followed by an identifier that is the directive name. for example #define is the directive, that defines a macro.

All preprocesser directives starts with '#' symbol.

List of preprocesser directives.

* #include.
* #define
* #undef
* #ifdef
* #ifndef
* #if
* #else
* #elif
* #end if

include.

The #include preprocesses directive is used to paste code of given file into current file. it is used by include system - defined and user defined headerfiles if included file is not found compiler renders error. It has 3 variants.

include< file>

This varient is used for system headerfiles It searches for a file name file. in a list of directries specified by you. then in a standard list of system directries.

include " file "

This varient is used for headerfiles of your own program. It searches for a file named file first in the current directory then in the same directories used for system headerfiles. The current directory is the directory of the current input file.

macros (# define)

macro is defined by # define directives.

Syntax.

define token value.

Files:

In C programming file is a place on your physical disk. where information is stored.

Why files are needed:

When a program is terminated the entire data is lost, storing in a fille will preserve your data even if the program terminates. If you have to enter a large number of data it will take a lot of time to enter they all however if you have file containing all the data you can easily acess the contents of the file using few commands in c you can easily move your data froma one computer to another computer without any changes

Types of files:

When dealing with files there are two types of files you should know about

1. Text files
2. Binary files.

Text files:

Text files are the normal "txt" files that you can easily create using notepad or any simple text editors. when you open those files you will see all the contents within the file as plan texts you can easily edit acelerate the contents. They take minimum effort to maintaining are easily readable and Provide least security and takes bigger storage space

Binary files:

Binary files are "bin" files in your computer Instead of storing data in plain text they stores it in the binary form (0's & 1's)

They can hold higher amount of data are not readable easily and provides better Security than text files

files operations:-

In C you can performs four major operations on the file either text or binary.

1. Creating a newfile
2. Opening an existing file
3. closing a file
4. Reading from writing information to a file.

WorKing with files

when working with files you need to declare a pointer of type files. This declaration is needed for communication btw file and program.

```
FILE *fptr;
```

Opening a file - for creation and edit

Opening a file is performed using the library function in the stdio.h. headerfile i.e, fopen()

The syntax for opening a file in standard I/O is

```
fptr = fopen ("file open", "mode");
```

Ex:=

fopen ("e:\\c program\\new program.text", "w");

fopen ("e:\\c program\\old program.bin", "rb");

let us suppose the file new program.txt doesn't exist in the location e:\\c program,

The first function creates a new file named new program.txt and opens it for writing as per mode 'w'

The writing mode allows you to create & edit (over write) the contents of file. Now let suppose the second binary file old program exists in the location e:\\c program the second function opens an existing file reading in binary mode 'rb' the reading mode only allows you to read the file you cannot write into the file.

Closing file

The file both text and binary should be closed after reading or writing closing a file is performed using a library function fclose()

fclose(fptr);

Opening modes in standard input/output

filemode	meaning	during inexistence of file
r	open for reading	if file doesnot exists fopen() returns NULL
rb	open for reading in binary mode	if file doesn't exist fopen() returns NULL
w	open for writting	if file exist its contents are over written if file doesn't exist it will be created.
wb	open for writting in binary mode	if files "
a	open for aponding i.e, data is added to end of the file	if file doesnot exist it will be created.
ab	open for aponding in binary mode	"
r+	open for both reading and writting	if file doesn't exist fopen() returns null
rb+	open for both read and write in binary mode	"
w+	open for both read and write	if file exist the contents are over written if file doesn't exist creats new file.
wb+	" in binary mode	"
a+	open for both reading and aponding	if file doesn't exist it will create a file.
ab+	"	"

Reading and writting for text file.

For reading & writting to the text file we using the function fptr

They are just the file version of printf & scanf The only difference is that fprintf & fscanf expects the pointer to the structure file.

Example:1

Write to a text file using fprintf.

```
#include <stdio.h>
#include <conio.h>
Void main ()
{
 int num;
  FILE *fptr;
   fptr:=fopen("c:\\ece\\program.txt","w");
   if (fptr==NULL)
   {
   Printf ("Error!");
   }
   Printf ("Enter number");
   Scanf ("%d", &num);
   fprintf (fptr, "%d ", num);
     fclose (fptr);
     getch();
     }
```

This program takes a number from user and stores in the file program.txt. After you compile and run this program you can a txt file program.txt created in C drive, ece folder in your computer. when you open the file you can see the integer you entered.

Example 2:

Reading from the text file:

Read from a text file using fScanf.

```
#include<stdio.h>
#include<conio.h>
Void main()
 {
 int num;
  FILE *fptr;
  fptr = fopen ("C:\\ece\\program.txt","r");
  if (fptr == NULL)
  {
  Printf ("Error!");
  }
  fscanf (fptr, "%d", &num);
  Printf ("value of n=%d", num);
  fclose (fptr);
  getch();
  }
```

This program reads the integer present in the program.txt file and prints it on to the screen.

If you successfully created a file from ex:-1 running will get you the integer you entered.

Other functions like fgetchar(), fputc() e.t.c. can be used in similar way.

Example 3.

To read multiple characters.

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE *fp;
 int num;
 char ch;
 fp = fopen ("c:\\ece\\program.txt", "r");
 if (fp == NULL)
 {
 Printf ("Error!");
 }
 While ((ch=fget c.(fp)) != EOF)
 {
 Printf ("%d", ch);
 }
 fclose (fp);
 getch();
 }
```

Reading and writting to a binary file.

functions fread() and fwrite() are used for reading and writting to a file on the disc respectively in case of binary file.

Writing to a binary file.

To write into binary function you need to use fwrite().

The function takes four arguments Address of data to be written in disc, size of data to be written in disc(), No: of such type of data and pointer to the file where you want to write.

fwrite:-

Syntax:

fwrite (address-data, size-data, numbers-data, Pointer-to-file)

Example 3.

Writing to a binary file using fwrite().

```
#include<stdio.h>
#include<conio.h>
struct threenum
{
  int n1,n2,n3;
}
void main()
{
int n ;
struct threenum num;
FILE * fptr;
if((fptr=fopen("C:\\ece-c\\program1.bin","wb"))==NULL)
{
 Printf("Error");
}
```

```
for (n=1 ; n<5 ; n++)
{
  num . n1 = n ;
  num  n2 = 5*n ;
  num  n3 = 5*n+1 ;
  fwrite (&num , size of (struct threenum) , 1 , fptr) ;
  }
   fclose (fptr) ;
   getch() ;
   }.
```

In this program, you create a new file program1.bin in the C drive.

We declare a structure threenum with 3 numbers n_1, n_2, n_3 and define it in the main function as num.

Now inside the for loop we store the value into the file using fwrite.

The first parameter takes the address of num and the second parameter takes the size of the structure threenum.

Since we are only inserting one instance of num the third parameter is one. and the last parameter *fptr points to the file we are storing the data.

Finally we close the file.

Reading from a binary file.

function fread also takes four arguments. similar to fwrite() function as above

Syntax:

fread (address-data, size-data, numbers-data, Pointer-to-file);

Example 4 :- reading from a binaryfile using fread.

```
#include <stdio.h>
#include <conio.h>
struct threenum
{
int n1,n2,n3;
};
Void main
{
int n;
struct threenum num;
FILE *fptr;
if ((fptr =fopen ("c:\\ece-c\\program1.bin","rb"))==NULL)
{
Printf ("Error");
}
for (n=1; n<5; n++)
{
fread (&num, sizeof (struct threenum), 1, fptr);
Printf ("n1= %d \t n2 = %d \t n3= %d ", num.n1,
                                        num.n2, num.n3);
}
fclose (fptr);
getch();
```

In this program you read the same file programming and loop the through the records, one by one in simple terms you read one the enum record. of the enum size from the file pointed by *fptr into the structure num.

You will get the same records you inserted in example 3.

Getting data using fseek()

If you have many records inside file and needs to acess a record at a specific position you need to loop through all the records before it to get the record. this will waste a lot of memory and operation time. an easier way to get to the required data can be acheived using fseek function.

This function is used for seeking the pointer position in the file at the specified bite.

fseek (file pointer, displacement, pointer position);

where, file pointer is the pointer which points to the file.

displacement → It is positive or negative and this is the no. of bite which are skiped (back ward if -ve or forwards if positive from current position)

This is attached with L because it is a long integer.

Pointer position → This sets the pointer position in the file. Value pointer position.

Value	pointer position.
0	begining of file
1	Current position
2	end of file.

Example 1.

fseek (P, 10L, 0)

'0' means pointer position is on the begining of the file from this statement pointer position is skipped 10 bits from the begining of the file.

Example 2.

fseek (P, -5L, 1);

from this statement pointer position is skipped 5 bit backword from the current position.

ftell ( ) :=

This function returns the value of the current pointer position in the file. The value is count from the begin of the file.

Syntax:- ftell (file pointer);

rewind ( ) :=

This function is used to move the file pointer to the begining of the given file.

Syntax:- rewind (file pointer);

Program to impliment fseek function, ftell() and rewind()

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE *P;
char ch;
int n,c,b;
clrscr();
fp = fopen("f:\\ece\\ file1.txt","r");
if(P==NULL)
{
Printf("file is not available");
}
 else
{
 Printf("Enter the value of n");
  Scanf("%d", &n);
        c = ftell(P);
  Printf("%d",c);
  fseek(P, n, 0);
   c = ftell(P);
   Printf("%d\n", c);
   Printf("Enter no: of bytes");
   Scanf("%d", &b);
   fseek(P, b, 1);
  while((ch = fgetc(P)) != EOF)
  {
   Printf("%c", ch);
  }
  Printf("\n");
   c = ftell(P);
```

```
Printf ("%d", c);
rewind (p);
c = ftell (fp);
Printf ("%d\n", c);
fclose (fp);
getch();
}
```

UNIT - IV

Function:

Function is a named part of a program which can be called when ever needed or function is a program which performs specific task and some times return a value.

There are two types of functions.

1. built-in(or) library function
2. Userdefined function.

Syntax:

```
return_type function_name(Parameter)
{
 body of function
}
```

Example:-

```
int max(int a, int b)
{ int result;
  if (a > b)
     result = a;
  else
     result = b;
  return result;
}
```

A function declaration tells the compiler about the function name and how to call function

function declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

Calling a function:

When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.

To call a function, you simply need to pass the required parameters along with the function name. If the function returns a value, then you can store the returned value.

Example:-

```
#include<stdio.h>
#include<conio.h>
int max (int num1, int num2);
void main()
{
  int a = 100;
  int b = 200;
  int ret;
  ret = max(a,b)
  printf ("Max value : %d \n",ret);
  return 0;
}
```

```
int max(int num1, int num2)
{
int result;
if (num1 > num2)
    result = num1;
else
    result = num2;
return result;
}
```

1. Call by value
2. Call by reference.

These are two call types in function in which arguments can be passed to a function.

Call by value:

The call by value method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the fuction have no effect on the argument.

By default, C programming uses call by value to pass arguments

Example:-

```
#include <stdio.h>
#include <conio.h>
void swap(int x, int y);
int main()
```

```
{
int a = 100
int b = 200
Printf ("Before swap, value of a : %d \n", a);
Printf ("Before swap, value of b: %d \n", b);
Swap(a,b);
Printf ("After swap , value of a : %d \n", a);
Printf ("After swap , value of b : %d \n", b);
return 0;
}
void swap (int x , int y)
{
int temp;
temp = x;
x = y;
y = temp;
return;
}
```

Call by reference :

The call by reference is method of passing arguments to a function copies the address of an argument into the formal parameters. Inside the function, the adress is used to acess the actual argument used in the call. It means the changes made to the parameter affect the passed argument.

Call by reference / passing pointers as arguments

```
#include <stdio.h>
#include <conio.h>
void swap (int *, int *);
void main()
{
int a,b;
clrscr();
Printf ("Enter a & b");
scanf ("%d %d", &a, &b);
Swap (&a, &b);
Printf ("%d %d",a,b);
getch();
}
void swap (int *x, int *y)
{
int t;
t = *x;
*x = *y;
*y = t;
Printf ("%d %d", *x, *y);
}
```

Passing arrays as arguments in function:

```
#include <stdio.h>
#include <conio.h>
void function1(int[], int);
void main()
{
int arr[5], i;
clrscr();
Printf("Enter values in array");
for (i=0; i<5; i++)
{
scanf ("%d", &arr[i]);
}
function1(arr, i);
getch();
}
void function1(int arr[], int 5)
{
int i, sum=0;
Printf("Elements are");
for(i=0; i<5; i++)
{
        %d", arr[i]
Printf("Elements are");
Sum = sum + arr[i];
}
Printf("total = %d", sum);
}
```

RECURSION

Program to find factorial of given number using recursion.

```
#include <stdio.h>
#include <conio.h>
int fact(int);
void main()
{
int n,res;
clrscr();
Printf ("Enter the value of n");
Scanf ("%d ", &n);
res = fact(n);
Printf ("factorial of %d = %d ", n, res);
getch();
}
 int fact (int x)
 {
  int res;
  if (x==0)
  {
  res = 1;
  }
  else
  {
  res = x * fact (x-1);
  }
  return res;
  }.
  }
```

Program to print fibonaki series using recursion.

```
#include <stdio.h>
#include <conio.h>
int fib(int);
void main()
{
int n,i;
clrscr();
Printf ("Enter n value");
Scanf ("%d", &n);
Printf ("the fibonacci series");
for (i=0; i<n; i++)
{
Printf ("%d.\t", fib(i));
}
getch();
}
int fib (int x)
{
int res;
if (x==0 || x==1)
{
return x;
}
else
{
res=fib (x-2) + fib(x-1);
return res;
}
}
```

DMA

Dynamic Memory Allocation

Program to print the sum of n elements by user using dynamic memory allocation.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
{
int num, i, *ptr, sum=0;
Printf ("Enter no: of elements");
Scanf("%d",&sum);
Ptr = (int *) malloc(num * size of (int));
if (ptr==NULL)
{
Printf ("Error!, memory not allocated");
exit (0);
}
Printf ("Enter elements of array");
for (i=0; i<num; i++)
{
Scanf ("%d", Ptr+i);
Sum = Sum + *(Ptr+i);
}
Printf ("Sum=%d", sum);
free (ptr);
getch();
}
```

Program to print sum of n elements using Calloc.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
{
int num, i, *ptr, sum=0;
Printf ("Enter no: of elements");
Scanf ("%d", &sum);
Ptr = (int *) Calloc (num, size of (int));
if (Ptr==NULL)
{
Printf ("Error! memory not allocated");
exit (0);
}
Printf ("Enter elements of array");
for(i=0; i<num; i++)
{
Scanf ("%d", ptr+i);
Sum= Sum + *(Ptr+i);
}
Printf ("Sum = %d", sum);
free (ptr);
getch();
}
```

Dynamic Memory Allocation.

Dynamic memory management refers to manual memory management. This allows you to obtain more memory when required and release it when not necessary.

There are 4 libary functions defined under <stdlib.h> for dynamic memory allocation.

malloc()

The name malloc() stands for "memory allocation"

The function malloc() reserves a block of memory of specified size and return a pointer of type void which can be casted into pointer of any form

Syntax:

```
Ptr = (cast-type*) malloc (byte-size);
```

The malloc() returns a pointer to an area of memory with size of byte size. If space is insufficient, allocation fails and returns NULL pointer

Calloc()

The name calloc() stands for "contiguous allocation"

The only difference between malloc() and calloc() is that malloc() allocates single block of memory where as calloc() allocates multiple blocks of memory each of same size and sets all bytes

to zero

Syntax:
ptr = (cast-type*) calloc (n, element-size)

This statement allocates contiguous space in memory for an array of n elements.

free()

Dynamically allocated memory created with either calloc() or malloc() doesn't get freed on its own. You must explicitly use free() to release the space.

Syntax of free()

free(ptr);

This statement frees the space allocated in memory pointed by ptr.

realloc()

If the previously allocated memory is insufficient or more than required, you can change the previously allocated memory size using realloc()

Syntax:
Ptr = realloc (ptr, newsize);

here, ptr is reallocated with size of newsize

Program to implement realloc().

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
Void main()
{
unsigned int *Ptr;
int i, n1, n2;
Printf("enter size of an array");
Scanf("%d", &n1);
 Ptr = (int *) malloc(n1 * size of (int));
 Printf("Address of previously allocated memory");
for(i=0; i<n; i++)
{
 Printf("%u\t", Ptr+i);
}
Printf("\n Enter new size of array");
Scanf("%d", &n2);
 Ptr = realloc (Ptr, n2 * size of (int));
for(i=0; i<n2; i++)
{
 Printf("%u\t", Ptr+i);
}
getch();
}
```

Allocating memory for arrays of character data type dinamically.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void main()
{
int n=5;
char *Pvowels = (char*) malloc (n* size of (char));
int i;
clrscr();
Pvowels [0]='A';
Pvowels [1]='E';
*(Pvowels +2) = 'I';
Pvowels [3] = 'O';
*(Pvowels + 4) = 'U';
for (i=0; i<n; i++)
{
Printf ("%c", Pvowel[i]);
}
Printf ("\n");
free (Pvowels);
getch();
}
```

memory leak
if we won't free the memory these will be memory leak and when we free further efficiency increase

Program to allocate memory dinamically for array of double datatype.

```
#include <stdio.h>
#include <conio.h>
# include <stdlib.h>
void main()
int main (int argc , char *argv[])
{
int i;
double *p;
P= calloc (10, size of (double));
for (i=0; i<10; i++)
  *(P+i) =i;
for (i=0 ; i<10; i++)
Printf ("*(P+%d) = %lf \n", i, *(P+i));
free(p);
Putchar ('\n');
P = calloc (4, size of (double));
for (i=0; i<4; i++)
  *(P+i) = i*i;
for (i=0; i<4; i++)
Printf ("*(P+%d) = %lf \n", i, *(P+i));
free(P);
}
```

UNIT-V

INTRODUCTION TO ALGORITHMS

In programing algorithm is a set of well defined instructions in sequence to solve the problem

Qualities of good algorithm:-

* Input and Output should be defined precisely
* Each step in algorithm should be clear and unambiguous.
* Algorithm should be most effective among many different ways to solve a problem.
* An Algorithm shouldn't have computer code. Instead the Algorithm should be written in such a way that it can be used in similar programing langauges.

Algorithm for finding roots of quadratic equation

Step 1: Start
Step 2: Read a, b, c
Step 3: find discriminant = b*b - 4*a*c
Step 4: if discriminant is greater than 0
    4.1: root 1 = (-b + sqrt (discriminant)/2*a))
    4.2: root 2 = (-b + sqrt (discriminant)/(2*a))
    4.3: Print root 1 and root 2 values and goto 7
Step 5: else if discriminant is equal to 0
    5.1: root 1 = root 2 = -b/(2*a)
    5.2: Print roots are equal and Print root 1 and root 2 and goto Step 7

Step 6: else roots are imaginary and goto step 7

Step 7: Stop

Algorithm to find the maximum and minimum from a given set.

Step 1: Start

Step 2: declare arr[max_size], i, max, min, size

Step 3: Read size

Step 4: Input elements in array and store in arr using for loop i.e, for(i=0; i<size; i++)

Step 5: Declare two variables max & min to store maximum and minimum. Assume first array element as max and min both

max = arr[0]

min. = arr[0]

Step 6: Inside loop for each array element check for maximum and minimum assign current array element to max if (arr[i] > max) and assign current array element to min if (arr[i] < min)

Step 7: Print max

Step 8: Print min

Step 9: Stop.

a number is prime.

step 1 :- start

step 2 :- Declare num, i, count = 0

step 3 :- read num

step 4 :- for (i = 0 ; i <= num ; i++)

4.1 :- if num % i == 0 then

4.2 :- increement count

4.3 :- goto step 4

step 5 :- Repeat 4.1 step 4.2 and 4.3 untill the condition is false.

step 6 :- if count == 2 then Print num is prime

step 7 :- otherwise Print num is not Prime

step 8 :- stop

Basic Searching techniques

Introduction to searching:

Searching means to find whether a particular value is present in an array or not.

If the value is present in the array then searching is said to be successfull and searching process gives the location of that value in the array.

However, if the value is not present in the array the searching process displaces an appropriate massage and this in case seaching is said to be unsuccessfull

There are two popular methods for searching the array elements.

1. Linear Search 2. Binary search

The algorithm that should be used depends entirely on how the values are organised in the array. linear

Search:-

Linear search also called as sequencial search is a very simple method used for searching an array for a particular value. It works by comparing the value to be searched with every element of array one by one in a sequence until a match is found

linear search is mostly used to search an unordered list of elements.

for example if an array a[5] is declared and intialised as.

int a[5] = {10, 40, 50, 30, 20}

if Key = 30 Pos = 3

Linear search Algorithm:

Linear Search(a, n, key)

step1: [initialize] set POS = -1

Step2: [initialize] set i = 0

Step3: Repeat 4 while i<=n

step4: if a[i] = key

set pos = i

Print Pos

break

Goto step 6

[end of if]

set i = i+1

[end of loop]

Step 5: If POS = -1

Print Key is not present in array [end of if]

Step 6: Exit

Program to impliment linear search.

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int a[5] = {40,30, 80,60,90};
 int Pos = -1, i=0, key;
 clrscr();
 Printf ("Enter key");
  Scanf ("%d", &key);
  while(i<=5)
  {
  if (a[i]==key)
  {
   Pos=i;
  Printf("Key is present in %d Position", Pos);
  break;
  }
   i= i+1;
  }
  if (Pos == -1)
  {
   Printf ("Key Is not found");
  }
  getch();
  }
```

Binary Search :=

It is a searching algorithm that works effeciently with a sorted list.

The mechanism of binary search can be better understood by an enalogy of a telephone directary. when we are searching for a particular name in directary we first open the directary from the middle and then decide whether to look for the name in the first part of directary or in second part. again we open some page in the middle and the hole Process is repeated untill we finally find the write name.

The same mechanism is applied in the binary search.

Now let us consider how this mechanism is applied to search for a value in a sorted array with the help of the algorithm

Binary Search algorithm.

Binary_search (a, lower_bound, upper_bound, key)

Step1: [Intialize] set low = lower_bound.
high = upper_bound
flag = 0

Step 2: Repeat step 3 and 4 while low <= high

step 3: If (key = a[mid])
set flag=1
break
goto step 6

Step 4: else If (key < a[mid])

set high = mid - 1

else set low = mid + 1

(End of if)

(End of loop)

step 5: If flag = 0

Print "value is not Present"

(end of if)

step 6: Exit

Program to implement binary search.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[5] = {10,20,30,40,50};
int low = 0, high = 4, mid, key, flag = 0;
clrscr();
Printf ("Enter key");
Scanf ("%d", &key);
while (low <= high)
{
 mid = (low + high)/2;
 if (key == a[mid])
   {
    flag = 1;
    break;
   }
 else if (key < a[mid])
 {
 high = mid - 1;
 }
 else
 {
```

```
low = mid + 1;
}
}
if (flag == 1)
{
Printf ("Key is found");
}
else
{
Printf ("Key is not found);
}
getch();
}
```

Sorting

1. Bubble sort.

C programming code for bubble sort is used to sort numbers or arrange them in ascending order. you can modify it to print numbers in desending order. you can also sort strings using bubble sort, it is less efficient as its average and worst case complexcity is high. there are many other fast sorting algorithms like quick sort, heap sort e.t.c.

Sorting simplifies problem solving in computer Programing

Algorithm:-

let 'a' be a linear array of 'n' numbers swap is a temperary variable for swaping (or) interchange the position of the numbers.

Algorithm for bubble sort.

step 1:= Input n number of an array a

step 2:= Intialize i=0 and repeat through step 4 if (i<n)

Step 3:= Intialize j=0 and repeat through step 4 if (j<n-1)

Step 4:= If (a[j] > a[j+1])

step 4.1:= swap = a[j]

step 4.2:= a[j] = a[j+1]

setp 4.3:= a[j+1] = swap

step 5:= Display the sorted number of array a

step 6:= exit.

Program to impliment bubble sort.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[100], n,i,j, swap;
clrscr();
Printf("Enter no: of elements");
Scanf("%d", &n);
for(i=0;i<n;i++)
 {
 Scanf ("%d", &a[i]);
 }
 for (i=0 ; i<n-1 ; i++)
  {
  for (j=0 ; j<n-1 ; j++)
  {
   if (a[j] > a[j+1])
   {
    swap = a[j];
    a[j] = a[j+1];
```

```
        a[j+1] = swap;
      }
    }
  }
  Printf ("Sorted list");
  for (i=0; i<n; i++)
  {
    Printf ("%d \n", a[i]);
  }
  getch();
  }
```

Insertion Sort:

C program for insertion sort is used to sort given numbers or elements in an array this code impliments insertion sort algorithm to arrang nos of an array in ascending order with a little modification it will arrange nambers in desending order best case complexity of insertion sort is O(n) ; average and worst case complexity is $O(n^2)$

Program to impliment insertion sort.

```
#include <stdio.h>
#include <conio.h>
Void main()
{
int n, a[100], i, j, t;
Printf ("Enter no: of elements");
Scanf ("%d", &n);
Printf ("Enter %d integers", n);
```

```
for (i=0; i<n; i++)
{
scanf ("%d ", &a[i])
}
for (i=1 ; i<=n ; i++)
{
 j=i;
 while (j>0 && a[j-1] >a[j])
 {
  t=a[j];
   a[j] = a[j-1];
    a[j-1] =t;
    j--;
   }
  }
Printf ("Sorted list");
for (i=0, i<n, i++)
 {
  Printf ("%d\n", a[i]);
  };
  getch();
  }
```

Algorithm

Let arr is an array of n elements.

Step 1: Read arr

Step 2: Repeat step 3 to 8 for i=1 to n-1

Step 3: Set temp = arr[i]

Step 4: Step j=i-1

Step 5: Repeat step 6 and 7 while temp<arr[j] and j>=0

Step 6: Set arr[j+1] = arr[i] (moves element forward)

Step 7: set j = j-1

(end of step 5 inner loop)

Step 8: Set arr[j+1] = Temp (insert element in proper place)

(end of step 2 outer loop)

Step 9: Exit

Selection Sort:

One of the simplest technique is the selection sort as the name suggest selection sort is the selection of an element and arragning it in selected order.

In this the strategy is to find the smallest number in the array and exchange it with the value in the first position of array now find the second smallest element in the remaining array of elements. and exchange it with a value in the Second position, carry on till you have reached the end of the array. Now all the element have been sorted in ascending order.

Algorithm for Selection sort.

Let arr is an array having n elements.

Step 1: Read arr

2: Repeat step 3 to 6 for i=0 to n-1

3: Set MIN = Arr[i] and set LOC = i

4: Repeat step 5 for j = 1+i to n

5: If MIN = arr[j], then

(a) Set MIN = arr[j]

(b) Set LOC = j

[End of if]

[End of step 4 loop]

6. Interchange arr[i] and arr[loc] using temporary variable

[End of step 2 outer loop]

7. Exit.

Program to impliment selection sort.

```
#Include <stdio.h>
#include <conio.h>
Void main()
{
 int i,j,n,loc,temp,min,a[30];
 Printf("Enter no: of elements");
  Scanf("%d",&n);
  Printf("\n Enter the elements");
  for(i=0; i<n;i++)
   {
    Scanf("%d ",& a[i]);
   }
  for(i=0; i<n-1;i++)
   {
    min = a[i];
     loc = i;
    for(j=i+1; j<n; j++)
     {
      if (min > a[j])
        {
         min =a[j];
          loc = j;
        }
     }
```

```
temp = a[i];
a[i] = a[loc];
a[loc] = temp;
}
Printf (" sorted list ");
for (i=0 ; i<n ; i++)
{
Printf (" %d", a[i]);
}
getch();
}
```

Order of Complexity:

Generally an algorithm has an asymtotic computational complexity. Assuming the input is of size N we can say that the algorithm will finish at $O(N)$, $O(N^2)$, $O(N^3)$, $O(N*\log(N))$ e.t.c. This means that it is a certain mathematical expression of the size of the input and the algorithm finishes between two factors of it

Generally the smaller the order of complexity of programs underlying algorithm the faster it will run and the better it will scale as the input gets larger. Thus, we should often seek more efficient algorithms inorder to reduce the order of complexity.

We will go through some of basic and most common time complexity such as

1) Constant time complexity O(1): constant runing time.
2) Linear time complexity O(n): linear running time.
3) Logarithmic time complexity O(log n): logarithm running time.
4) log linear time complexity O(nlogn): log linear running time.
5) Polynomial time complexity O(n^c): Polynomial running time (c is a constant)
6) Exponential time complexity O(c^n): Exponential running time (c is a constant being raised to a power based on size of input)

Constant time Complexity:

The code that runs in fixed amount of time or has fixed no: of steps of execution no matter what is the size of input has constant time complexity.

For instances lets try and derive a time complexity for following code.

```
def sum(a,b);
    return a+b;
```

If we call this function by sum(2,5). it will return 7 in one del. That single step of computation is summing a and b. No matter how large is the size of input i.e. a & b is. It will always return sum in one step so time complexity of above code is O(1) or constant time complexity

Linear time complexity:

The code whose time complexity is order of growth increases linearly as the size of the input is increased has linear time complexity

Example def my_list_sum(a);

```
    result = 0
    for i in a:
        result = result + i;
    return result;
```

here we are providing a list to the function if I Pass the list of size 10 the no: of steps would be considering the size of list is 10

One step for initializing the result

second step loop over the result and statement in the loop is excited 10 times.

One step for return statements.

for list of size [N] time complexity would be 1+N+1 = N+2 we can safely neglect the adittional 2 and say that. Over all time complexity is O(N) because we saw that as N becomes large the steps with constant time have negligable effect on running time of the code

Logarithmic time complexity:

when the size of input is N but the no: of steps to exicute the code is log of N such code is said to be exicuting in logarithmic time This defination is quite confusing but it can be understood clearly with an example code.

lets, say we have a very large numbers which is Power of 2 that is we have 2^x we want to find x. for example $64 = 2^6$

```
def Power_of_2(a):
    x = 0
    while a>1:
        a = a/2
        x = x+1
    return x
```

Now if I call power_of_2(16) the while loop will run 4 times because we keep dividing a/2. in first iteration a will become 8 In second iteration a = 4 In 3rd iteration a = 2 and In 4th iteration a = 1 (thats when the loop with break.

Now there are two instructions inside loop So total no: of steps would.

1 for x = 0

2×4 for statements inside the loop

1 for return statement; intotal 2×4 + 2

If we increase the size of a to 1024 it will take 2×10+2 steps.

The pattern behind this is $\log 16 = \log_2 2^4 = 4$

$\log 1024 = \log_2 2^{10} = 10$

Soverall time complexity of the above (considering $\log_2$) code $2 \times \log N + 2$

for every large values of capital N and multiplied by additional 2 can be neglected.

Hence it is $\log N$ to base 2

log linear time complexity:

when we call a logarithm time algorithm inside a loop it would result into a log linear time complexity.

for example let us say I have long sorted list of size [N] and I have Q numbers for each of those Q numbers I have to find the index of it in the given list.

```
for i in Q list
    Point binary search(x, search, list) // this statement is executed Q times.
```

Analysing above code we know that the call to binary search function takes $O(\log N)$ time we are calling it Q times. hence the overall time complexity is $Q \times O(\log N)$. Simplyfing we have $O(Q \log N)$

Polynomial.

When the computation time increases as function of n raised to some power N being the size of the input. Such a code has polynomial time complexity. For example let say we have a list of size N and we have nested loops on that list.

```
for i in N:
    for j in N:
```

In the above code the processing part is executed N*N times that is N^2 times Such a code has O(N^2) time complexity

Exponential time complexity

When the computation time of a code increases as function of x^N, N being the size of the input. Such a code has exponential time complexity

```
Example: def f(n):
             if n==0:
                 return 0
             else if n==1;
                 return 1
             else
                 return f(n-1)+f(n-2)//every call to f, we make 2 more calls to f itself.
```

In the above code for every call to 'f' we make 2 more calls to function 'f' in the else part. So if I call f(4) the three of calls to function f would look like this.