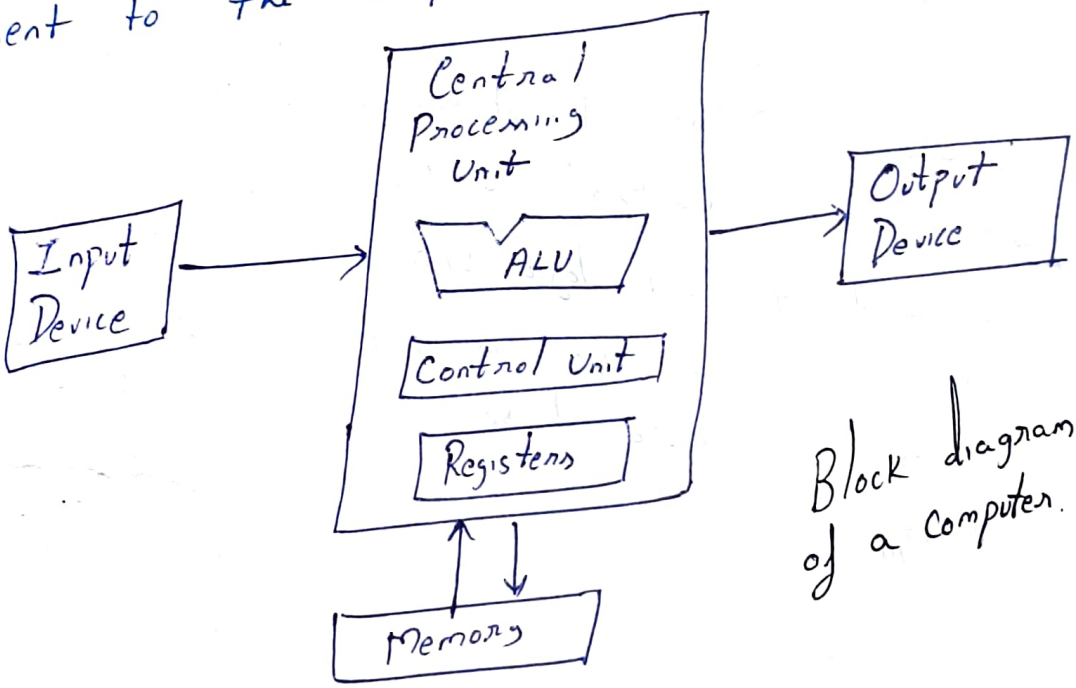
19ITT101 – Programming in C and Data Structures

Unit I – Introduction to C

Basic Blocks of Computers –

A Computer is a Programmable electronic Machine that accepts instructions and data through input devices, manipulating data according to the instructions and finally Providing result to the Output device. The result can be stored in memory or sent to the Output device.



Block diagram of a computer.

i) Input Device –

It is used to accept the data and instructions into the Computer & stored in Computer memory.

Stored instructions are further read by Computer from memory & manipulates the data according to the instructions.

I/p devices => Keyboards, Mouse, Micro phone, Pendrive, Floppy, CD - Compact Disc, Joystick, track ball, light pen, Analog to Digital Converters, Optical character Reader, Scanner, etc

3) CPU - Central Processing Unit

It is the Brain of the computer. It is Hardware device and called as Microprocessor chip. Chip produced from Silicon. Over which millions of transistors are mounted.

CPU is responsible to interpret and execute the instructions.

CPU Comprises of Arithmetic & logical unit, Registers and Control Unit.

ALU - Arithmetic and logic Unit

ALU performs various Arithmetic & logic operations on data based upon the instructions.

Control Unit -

It generates the timing and control signals for carrying out operations within the processor.

Registers -

Registers are used for holding the instructions & storing the results temporarily.

Instructions are stored in the memory and they are fetched one by one and are executed by the processor.

iii) Output Devices -

It is used to display the results on the screen or send the data to output device.

o/p Devices ⇒ Monitor, Printer, Speaker, Hard Disk, Floppy, CD, plotter, LED Display, D to A Converter, etc.,.

iv) Memory -

It is used to store the program, Data & Informations. It stores instructions & Data in form of 0's & 1's.

2 Types of Main Memory { → RAM (Random Access Memory) → ROM (Read Only Memory)

Memory → Primary / Main Memory → Main Memory (RAM & ROM), Cache Memory, Register Memory
→ Secondary Memory - eg optical Disk, Magnetic Disk, Tapes, etc.,.

Processor first read the instruction or Data from primary memory and Execute the instructions.

RAM - It is volatile, used to hold data & instructions that are currently in use.

ROM - It is non volatile & holds the contents even when the power is switched off.

↳ PROM (Programmable ROM) - Data can't be erased.

↳ EPROM (Erasable Programmable ROM) - Datas erased using electrical UV signals.

Volatile - The system loses the Data or content when power is lost.

Cache - Temporarily hold the current Executing Data

Registers - Hold next instruction, Data Address & Result during execution

Algorithm, Pseudocode, Flow chart.

These are the different ways of representing the logical steps for finding a solution of a given Problem.

Algorithm -

An Algorithm is an step by step procedure, Provided for solving the Problem.

Defined as "The finite set of steps, which Provide a chain of actions for solving a definite nature of Problem".

It is a sequential & well organized which will be later used for constructing a program.

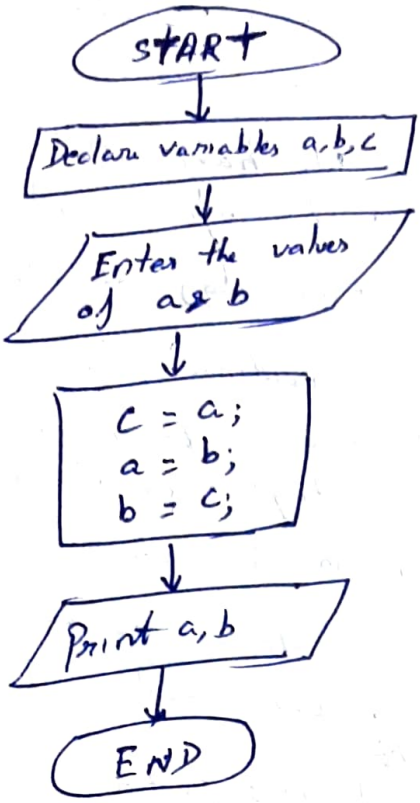
An algorithm is pre-arranged textual module, which receives some set of values as input and provide set of values as Output.

eg Swap 2 numbers using 3rd variable.

Algorithm

step 1 : start
step 2 : Declare variables a, b & c.
step 3 : Read values of a & b.
step 4 : Copy value of a to c; b to a; & c to b.
step 5 : Print Swapped values of a & b.
step 6 : Exit.

Flowchart

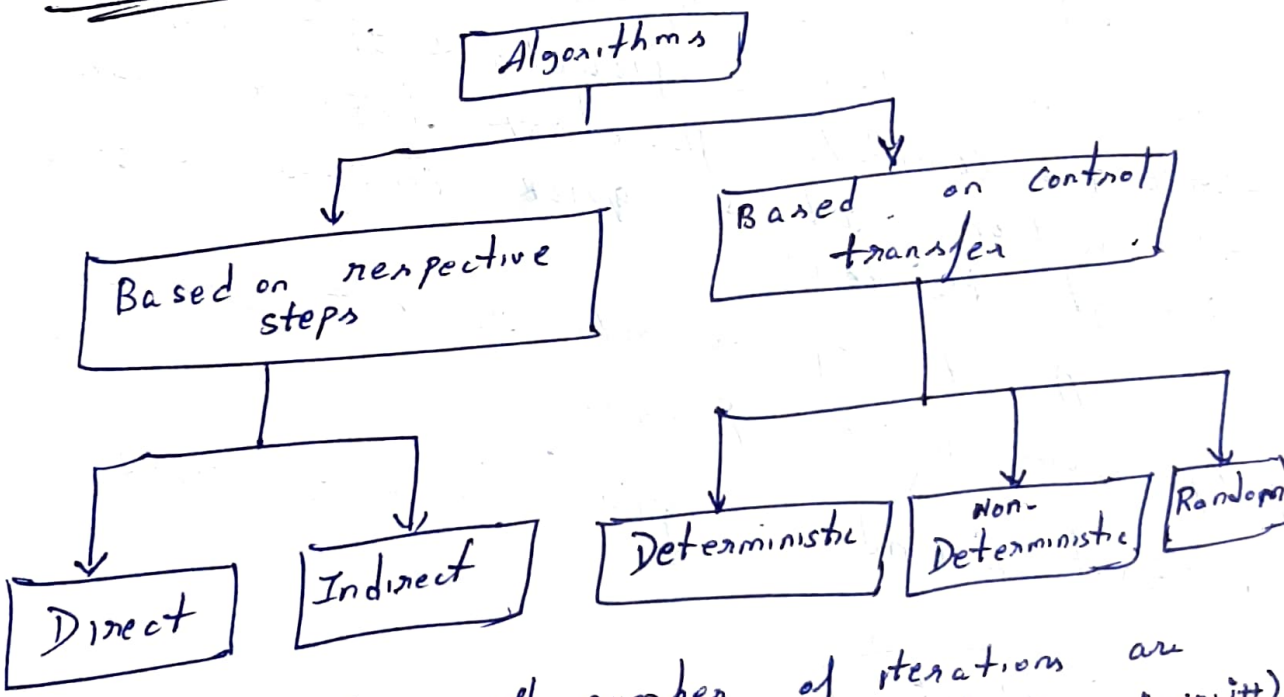


Program

```
# include <stdio.h>
# include <conio.h>
Void main()
{ int a,b,c;
  clrscr();
  Printf("\n Enter two numbers
                A & B :");
  Scanf("%d%d", &a, &b);

  c = a;
  a = b;
  b = c;
  Printf("After swapping values
        are : A = %d &
          B = %d ", a, b);
  getch();
}
```

Classification of Algorithms -



Direct Algorithm - The number of iterations are known in advance eg for(i= 0; i<10; i++)

InDirect Algorithm - Exactly number of repetitions are unknown. eg Palindrome, Armstrong No. eg 153

Deterministic Algorithm - Based on condition, either follow 'yes' or 'no' path
eg odd or even No.

Non-Deterministic Algorithm - We have multiple Paths for solution.
eg Find day of a week.

Random Algorithm - While executing, the control of the program transfers to another step randomly.
eg Random search.

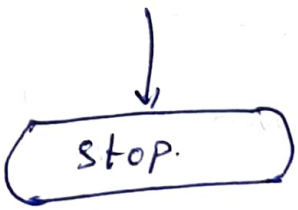
Infinite Algorithm - the no. of steps are not known in advance, continued till Best result.
eg Find shortest path.

Flowcharts -

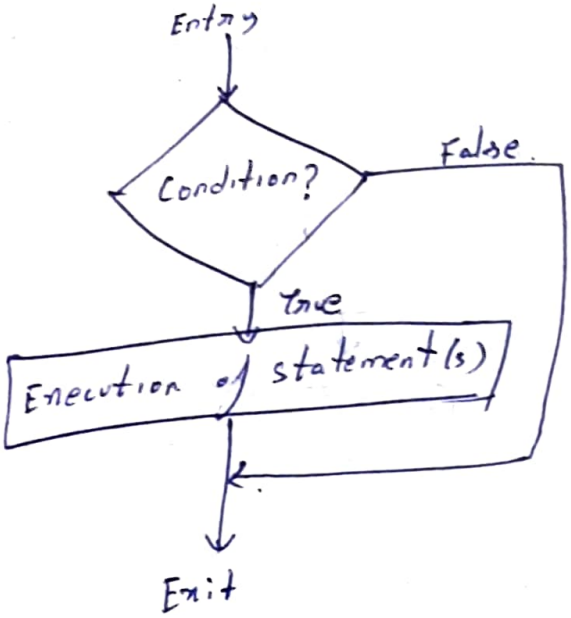
The pictorial representation of logical steps is called a flowchart. It shows operation Seq. -uence. The complete flowchart helps to plan the action. It is quick compared to going through the text.

A set of symbols indicate various operations in the program.

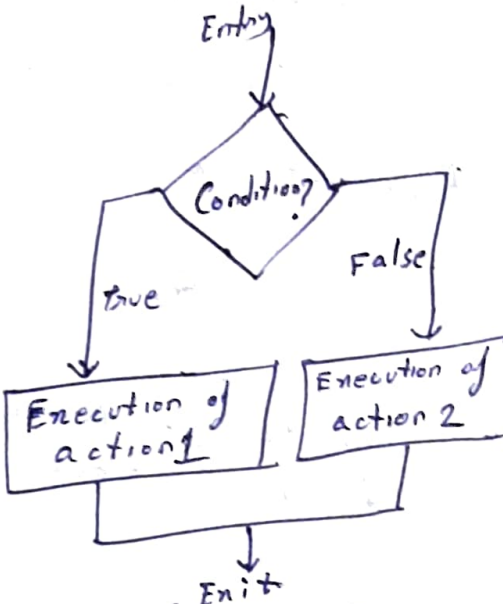
i) Start & End - Flat Ovel or egg shaped.



ii) Decision on test Symbol – Diamond shaped.

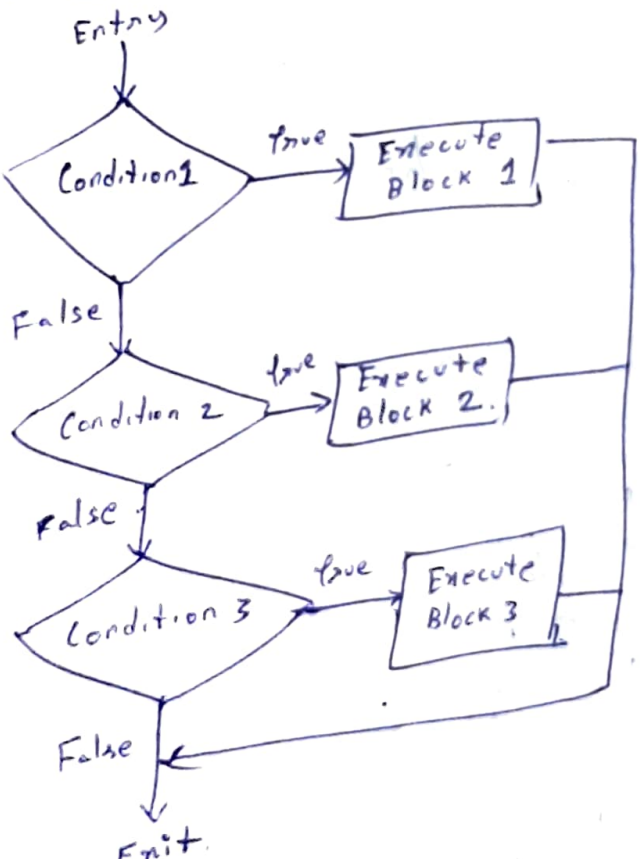


Single Alternative Decision



Two Alternative Decision

Multiple Alternative Decisions



If condition is satisfied execute appropriate statement block, otherwise next condition is verified

Connector Symbol - Circle.

For establishing connection b/w 2 pages.

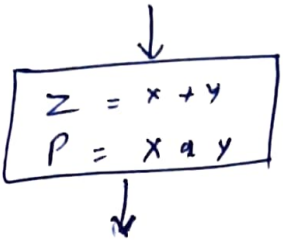
Alphabet or number is written inside symbol.

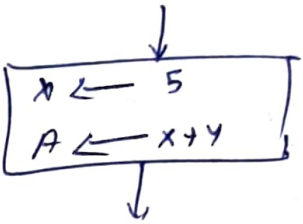
Connector for connecting to next block

Connector comes from Previous Block.

Process Symbol - Rectangle.

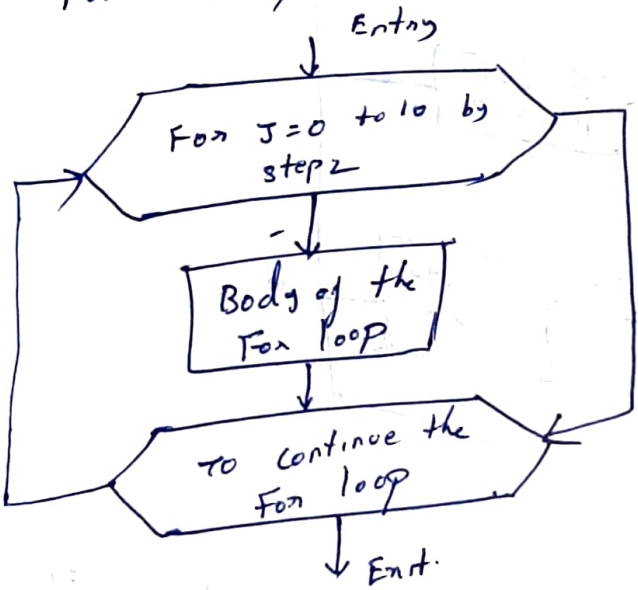
For Data handling & Assignment operations.



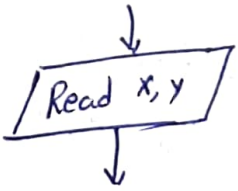


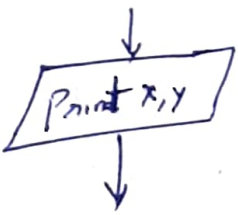
Loop Symbol - Hexagon.

For Implementation of loops.

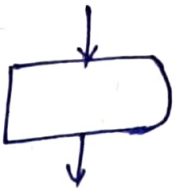


Input / Output Symbol - Parallelogram.

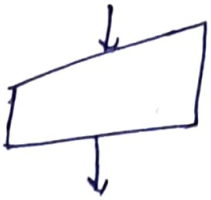




Delay Symbol - AND gate.



Manual Input Symbol -



Pseudo Code -

Pseudo Code is an informal high level language or simple version of programming Coding language to solve the problem.

It solve the problem using natural language and mathematical notations.

eg Find largest no. of a & b.

Algorithm	Pseudocode
Input a and b.	get numbers a & b
Is a > b.	Compare a & b
If yes a is larger than b.	if a is large max = a
If no b is larger than a.	if b is large max = b
Print the larger number.	Larger number is max.

Structure of C Program -

C language → developed by Dennis Ritchie at Bell Laboratories, USA in 1972.

C is most Popularly used general-Purpose Programming languages.

It is a middle-level Language.

ANSI C. (American National Standard Institute) Standard. C language - 6 yrs from 1990 came to usage.

* Machine Language ⇒ 0's & 1's used to write Program
* Assembly Language ⇒ English like words abbreviated as mnemonics.

  eg ADD, SUB, MUL, DIV, etc,.

  Assembles used to convert assembly language Pgm to machine language.
* High-Level Language ⇒ eg COBOL, FORTRAN, BASIC, etc...

Source Program ⇒ Written Program

Object Program ⇒ Machine language.

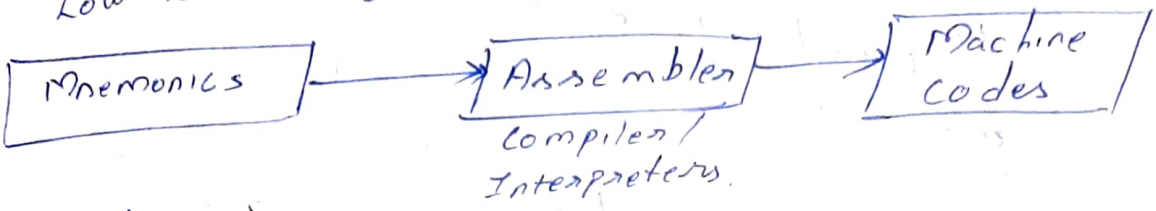
Source Pgm —to→ Obj Pgm using translators.

1) Assembler
2) Compiler
3) Interpreter.

Assembler -

It translates the symbolic codes of Programs of assembly language into machine language.

Low level lang → Machine code.



Compiler / Interpreters

Compiler -

Compiler translates all the instructions of the entire program into machine codes, which can be used again and again.

Interpreter -

Interpreter reads the program line by line, and generates the object codes. It executes the program from source code, so, every time it should be interpreted.

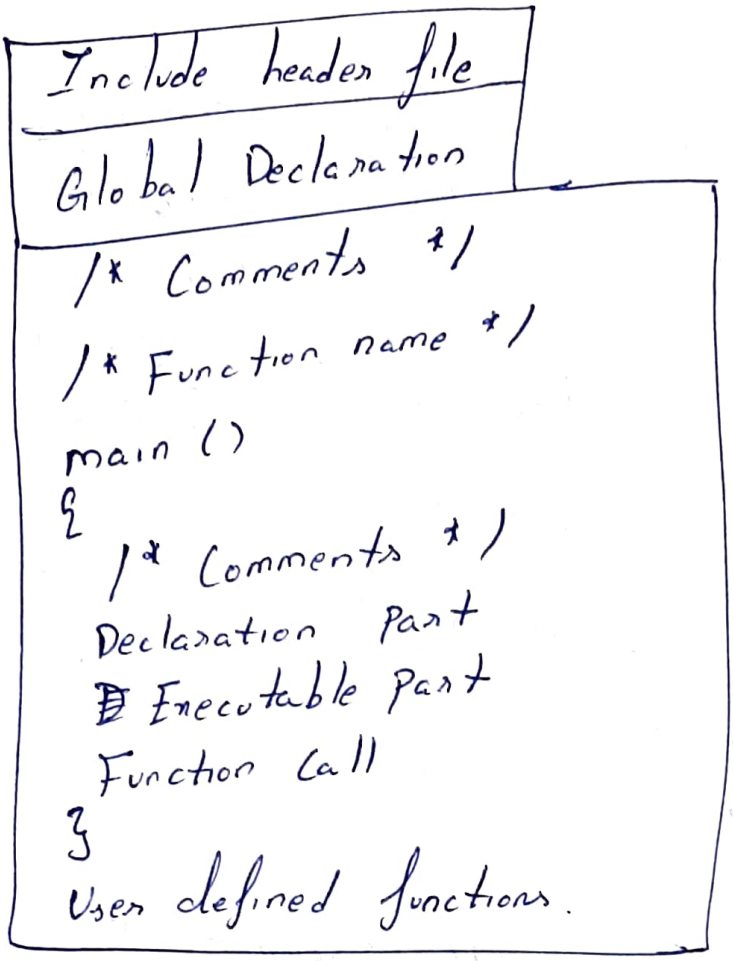
C program contains functions - building blocks & standard libraries.

Linking -

Linker is a program that combines source code and codes from the library.

Structure of C Program -

C program contains No. of building blocks known as functions. Each function performs a specific task independently



Structure of a C Program.

i) Include Header file section -

header file has extension '.h' & should include # include directive.

It should be included at the beginning of the C program.

eg # include <stdio.h>

Header File	Functions	Function Example
Stdio.h	I/P, o/p & file operations fln's	printf(), scanf()
Conio.h	Console I/P & o/p fln's	clrscr(), getche()
alloc.h	Memory allocation - related fln's	malloc(), realloc()
graphics.h	All graphic - related fln's	Circle(), bar3d()
math.h	Mathematical fln's	abs(), sqrt()
string.h	String manipulation fln's	strcpy(), strcat()
time.h	Contains date & time related fln's	asctime()

ii) Global Declaration -

This section is declared outside the function, these variables are used by more than one function. These variables are known as global variables.

iii) Function main() -

The program execution starts from the main function.

Returntype main(Arguments) ← syntax.

eg void main() - No arguments & no return statement.

The open curly brace specifies the start of the main f/n. the close curly brace specifies the end of code for main f/n.

B/w the { & } the program has its declaration & execution part.

iv) Declaration Part -

It declares the entire local variables used in the executable part.

The local variable scope is limited to the f/n where it is declared.

Initialization of variables can also be done here.

eg int a, b;

eg int a=5, b=10;

v) Executable part -

This part contain the statements following the variable declaration.

It contains the set of operations to be performed for solving the problem.

vi) Function Call -

A user defined f/n can be invoked from the main() as per need.

vii) User-defined f/n -

The f/n defined by the user outside the main() f/n.

viii) Body of the f/n -

The statements enclosed b/w { & } are called body of the f/n.

ix) Comments -

It is used to understand the flow of program by the programmer. Placed between /* and */

It is not necessary in the program, the compiler doesnot execute comments.

It can be inserted anywhere in the program.

Eg:

```
/* Single line comment */
/* comment /* nested comments */ */
/* the example of
   comments in
   multiple lines */
```

Programming Rules -

1) Every program should have main() f/n
2) Statements should be terminated by semi-colon.
3) All statements are written in lowercase.
4) Blank spaces are allowed b/w words in statement.
5) Can write more statements in one line seperating ;

Eg: a = b+c; d = b*c;

6) The opening & closing braces should be balanced.

Executing the C Program -

1) Creating of a Program.
2) Execution of Preprocessor Program.
3) Compilation and Linking of a program.
4) Executing the program.

Advantages of C -

1) Contains powerful data definition.
2) Supports operators.
3) Execution is faster.
4) Assembly code can be inserted.
5) Highly portable into any platforms
6) Has keywords, lib fln & header files.

C - Character Set -

↳ Alphabets (a-z, A-Z)
↳ Digits (0-9)
↳ White space & Escape sequences (\n - newline, \t - Horizontal space tab)
↳ Special Characters. (, ; & ! ! < > = % # * {} [])

Types of Tokens -

A smallest unit in a program/ statement is called a token.

Types i) Keywords - Keywords are reserved by the compiler (ANSI → 32 Keywords)
ii) Variables - User defined.
iii) Constants - Are assigned to variables.
iv) Operators - Used in expression.
v) Special characters - Declaration in C.
vi) Strings - A sequence of characters.

i) Keywords -

There are reserved words by the compiler. All has fixed meanings & can't be used as Variables.

eg

Void	if	Switch	Auto
int	do	Case	enum
char	else	break.	extern
const	for	Continue	register
double	goto	default	return
float	While	struct	size of
long		type def	static
short		Union	Volatile.
signed			
unsigned			

ii) Identifiers -

A symbolic name used to refer a variable, Pointer, function, array, structure, union, etc..

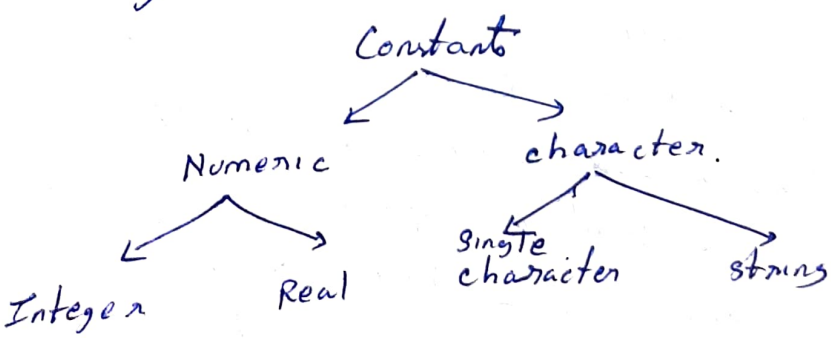
ie) There are the names of Variable, fln, array, etc..

It is combination of letters & numbers, no blank spaces, punctuations & symbolls are allowed. Only (_) underscore symbol is allowed.

eg a, sum, Area_of_Square.

iii) Constants -

Constants are the values do not change during execution of program.



Numerical Constants -

Integer - (2-4 byte)
- Represented with whole number.
- Decimal, Fraction & Symbol not permitted.
- either positive or negative or zero.
- Number without sign is assumed '+ve'.

eg 10, +30, -5

Real Constants / Floating Point Constants.

Represented in exponential or fractional form.
Decimal point permitted.
either +ve or -ve.

eg { like length, price, distance values } 3 mantissa exponent
2.5, 5.215, 3.14, etc... & $2.4561 \times e^{+3} =$
2456.123
Represented as 2.4561e+3

Character Constant -

Single character constants -

Single digit character enclosed within a pair of single quotation. eg 'a', '8', '-'

Printf("%c %d", 65, 'B') o/p ⇒ A 66

String Constants -

Sequence of character enclosed within double quote. eg "Hello", "444", "a"

iv) Variables -

A variable is used to store values. It is a data name used for storing data value. Value may be changed during Program execution.

It can be combination of letters & digits.

eg height, avg12

(X) Variable should not be an keyword.

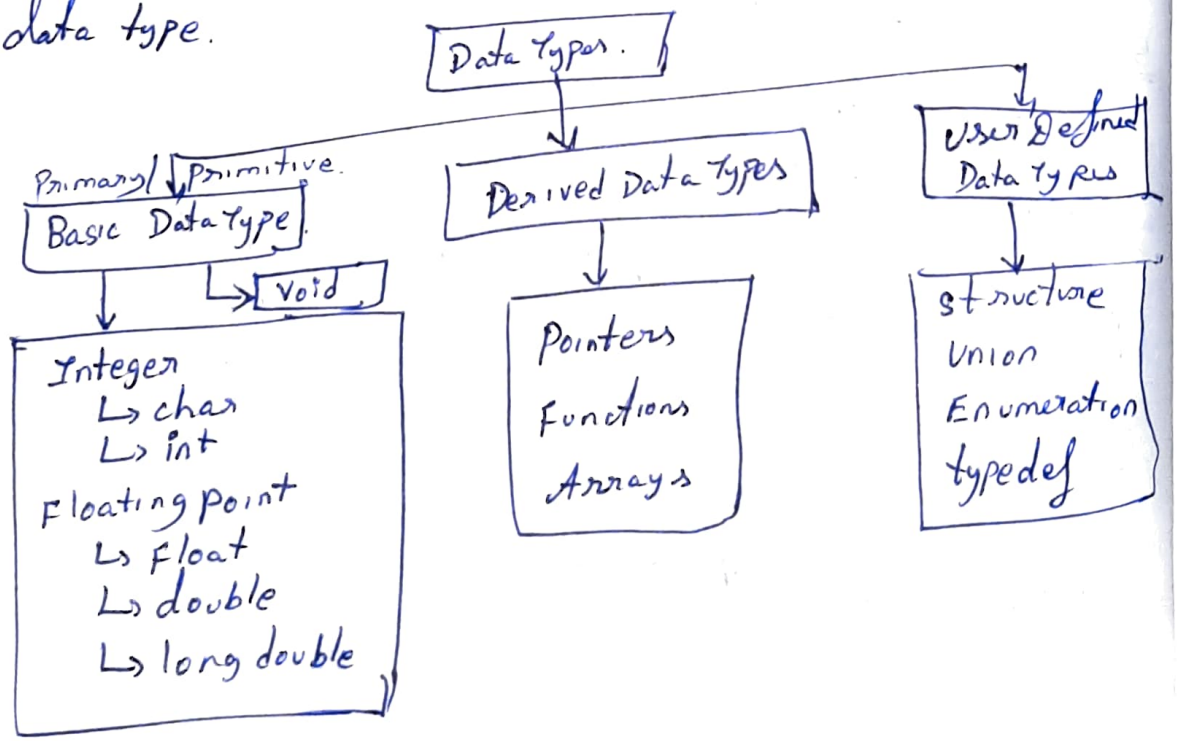
May be combination of upper case, lower case & numbers.

Variable should start with a character, should not start with a digit.

Blank & comma not permitted.

DATA TYPES -

Each variable is associated with a data type.



Type (Keywords)	Format Specifier	Size (Bits)	(Byte)	Range
Signed short int	%i %d	8	1	-128 to 127
Signed long int	%ld	32	4	-2,147,483,648 to 2,147,483,647
unsigned short int	%u	8	1	0 to 255
unsigned long int	%lu	32	4	0 to 4,294,267,295
int / Signed int	%d %i	16	2	-32,768 to 32,767
unsigned int	%u	16	2	0 to 65535.
Signed char	%c	8	1	-128 to 127
unsigned char	%c	8	1	0 to 255
Float	%f	32	4	3.4E-38 to 3.4E+38
double	%lf	64	8	1.7E-308 to 1.7E+308
long double	%Lf	80	10	3.4E-4932 to 1.1E+4932
enum	%d	2		-32768 to 32767

Declaring Variables -

1) Primary Data types.

Syntax - Data_type Variable_name;

eg
```
int sum;
int a, b, c=0;
double ratio;
char s;
float m;
```

Initializing Variables -

Syntax - Variable_name = Constant, (or)
Data_type Variable_name = Constant;

eg
```
sum = 5;
int a=5, b=3;
int sum, a=5, b=3;
```

2) User-defined data type

Syntax - typedef type identifier; (user defined Data type)

eg
```
typedef int mark;
```

Operators –

An Operator indicates an Operation to be Performed on data.

Used to link variables & Constants.

An Operand is a data item on which Operators perform operations.

Operators

i) Arithmetic
ii) Relational
iii) Logical
iv) Bitwise.

Type of Operator.	Symbolic Representation
Arithmetic Operators	+, -, *, / and %
Relational operators.	>, <, ==, >=, <= and !=
Logical operators	&&, || and !
Increment & Decrement Operators.	++ and --
Assignment Operator	=, *=, /=, %=, -=, +=, ^=, |=, <<=, >>=
Bitwise operators	&, |, ^, >>, << and ~
Comma operator	,
Conditional operators.	?:

Operator Precedence - (Priority)

Operations on Operands are carried out according to the priority of the operators.

* Higher priority 1st ← Lower priority next.

+ ← -

Arithmetic ⇒ * , / & %

eg 8 + 9 * 2 - 10

8 + 18 - 10

26 - 10

16 //

* Operators of same priority, evaluate from left.
* Inner parenthesis is solved first followed by outermost.

eg (16 / (2 - (2+2));

```
# include <stdio.h>
# include <conio.h>
main()
{
int a=1, b=2, c=3, j;
clrscr();
j = a+b * c;
Printf("\n j = %d", j);
j = (a+b) * c;
Printf("\n j = %d", j);
}
```

Output - j = 7

j = 9

Associativity -

If operators are with equal precedence, the associativity decides which operation to be performed first.

2 Types → i) Left to right
ii) Right to Left.

Comma & Conditional Operator.

Comma operator is used to seperate two or more expressions.

eg
```
Void main()
{ clrscr();
  Printf(" Addition = %d In Subtraction = %d",
                        2+3, 5-4);
}
```

o/p
Addition = 5
Subtraction = 1.

Conditional Operator (?:) / Ternary Operator.

If the Condition is true 1st statement is executed, otherwise 2nd statement is executed.

Syntax - Condition ? (expression1) : (expression2);

eg
```
void main()
{
clrscr();
3>2 ? Printf(" True") : Printf("False");
}
```

output true.

Arithmetic Operators -

Operators → Unary, Binary, Ternary.

Unary - Operators which require only one operand are called unary operators.

operator.	Description.
-	minus
++	Increment
--	Decrement
&	Address operator.
sizeof	Gives the size of operator.

eg

```
void main()
{
int a,z, x=10, y=20;
clrscr();
z = x * y++;
a = x * y;
Printf("\n %d %d", z, a);
}
```

o/p

200 210.

Binary - Operators which require 2 operands.

```
int main()
{ int a=9, b=4, c;
  c = a+b;
  Printf("a+b = %d \n", c);
  c = a-b;
  Printf("a-b = %d \n", c);
  c = a * b;
  Printf("a * b = %d \n", c);
```

```
c = a/b;
Printf("a/b = %d \n", c);
c = a % b;
Printf(" Remainder = %d \n", c);
return 0;
}
```

o/p

a+b = 13
a - b = 5
a * b = 36
a / b = 2
Remainder = 1.

```
void main()
{
int x = 2;
float y = 2;
clrscr();
Printf("\n sizeof(x) = %d bytes", sizeof(x));
Printf("\n size of (y) = %d bytes", sizeof(y));
Printf("\n Address of x = %u and y = %u", &x, &y);
}
```

o/p

size of (x) = 2
size of (y) = 4
Address of x = 4066 and y = 25096.

Relational Operators -

These operators provide the relationship b/w 2 expressions.

If the relation is true, it returns a value 1 otherwise 0 for false.

Operator	Description	Example	Return value
>	Greater than	5>4	1
<	Less than	10<9	0
<=	Less than or equal to	10<=10	1
>=	Greater than equal to	11>=5	1
==	Equal to	2==3	0
!=	Not equal to	3!=3	0

```
void main()
{ clrscr();
Printf("\n Condition : Return Values \n");
Printf("\n 10!=10 : %5d", 10!=10);
Printf("\n 10==10 : %5d", 10==10);
Printf("\n 10>=10 : %5d", 10>=10);
Printf("\n 10<=100 : %5d", 10<=100);
Printf("\n 10!=9 : %5d", 10!=9);
}
```

Output.

```
Condition : Return values.
10!=10  :  0
10==10  :  1
10>=10  :  1
10<=100 :  1
10!=9   :  1
```

Assignment Operators & Expressions -

operators - `=` `*=` `/=` `%=` `+=` `-=` `<<=` `>>=` `>>>=` `&=` `^=` `!=`

e.g. `x = x+3;` ⇒ `x += 3;`

`j = j * (x+1);` ⇒ `j *= x+1;`

```
void main ()
{
int a=5, b=3;
Printf ("value of a=%d and b=%d ", a, b);
a+=b;
Printf ("\nsum of a and b is %d", a);
}
```

o/p value of a = 5 and b = 3
sum of a and b is 8

Logical Operators -

The logical relationship between 2 expressions is tested. If true (1) or false (0) will be returned.

operator	Description	Example	Return value
&&	Logical AND	5>3 && 5<10	1
||	Logical OR	8>5 || 8<2	1
!	Logical NOT	8 != 8	0

(i) Logical AND (&&) ⇒ the o/p is true when both the expression is true.

(ii) Logical OR (||) ⇒ the o/p is true when one of the expression is true.

(iii) Logical NOT (!) ⇒ the o/p is 0 if it is true. otherwise 1.

```
void main()
{
clrscr();
Printf("\n Condition : Return value\n");
Printf("\n 5>3 && 5<10 : %d", 5>3 && 5<10);
Printf("\n 8>5 || 8<2 : %d", 8>5 || 8<2);
Printf("\n !(8==8) : %d", !(8==8));
}
```

o/p

Condition	: Return value
5>3 && 5<10	: 1
8>5 || 8<2	: 1
!(8==8)	: 0

eg Write a program to print logic 1 if input character is capital otherwise 0.

```
void main()
{
char x;
int y;
clrscr();
Printf("\n Enter a character :");
scanf("%c", &x);
y = (x>=65 && x<=90 ? 1 : 0);
Printf("y : %d", y);
}
```

output

Enter a character : A
y : 1

Enter a character : a
y : 0

Program to display 1 if i/p is b/w 1 and 100.

```
void main()
{
int x, z;
clrscr();
Printf(" Enter numbers : ");
Scanf("%d", &x);
z = (x>=1 && x<=100 ? 1 : 0);
Printf(" z = %d", z);
}
```

o/p

Enter numbers : 5
z = 1

(or)

Enter numbers : 10
z = 0.

Program to display 1 if i/p is 1 or 100.

```
void main()
{
int x, z;
clrscr();
Printf("Enter numbers : ");
Scanf("%d", &x);
z = (x==1 || x==100 ? 1 : 0);
Printf("z = %d", z);
}
```

o/p

Enter numbers : 1
z = 1
Enter numbers : 10
z = 1
Enter numbers : 101
z = 0

Program to display 1 if the i/p No. is except 100

```
void main()
{
int x, z;
clrscr();
Printf("Enter number :");
Scanf("%d", &x);
z = (x!=100 ? 1 : 0);
printf("d : %d", z);
}
```

o/p

Enter number : 100
d : 0

Enter number : 10
d : 1

Bitwise Operators -

Bitwise operators for bit manipulation, supports for int, char, short & long operands.

operators	Meaning
>>	Right shift
<<	Left shift
^	Bitwise XOR (exclusive OR)
~	One's complement.
&	Bitwise AND
|	Bitwise OR

Program to shift o/p data by 2 bits Right.

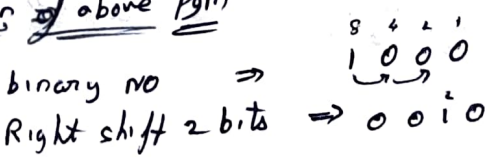
```
void main()
{
int  x, y;
clrscr();
Printf("Read the integer from keyboard(x):-");
scanf("%d", &x);
x>>=2;
y=x;
Printf("The Right shifted data is = %d", y);
}
```

o/p

Read the integer from Keyboard (x):-8

The Right shifted data is = 2.

Operation/Execution of above Pgm



8 ⇒ binary no ⇒ 1000

Right shift 2 bits ⇒ 0010

y = 2.

Program to shift i/p data by 3 bits left.

```
Void main()
{
int x, y;
clrscr();
Printf("Read i/p (x):-");
Scanf("%d", &x);
x <<= 3;
y = x;
Printf("Left shifted data is = %d", y);
}
```

o/p

Read i/p (x) :- 2

Left shifted data is :- 16

AND

i/p		o/p
x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

OR

i/p		o/p
x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

NOT

i/p	o/p
x	y
0	1
1	0

EX-OR

i/p		o/p
x	y	z
0	0	0
0	1	1
1	0	1
1	1	0

```
void main()
{
int a,b,c;
clrscr();
Printf("Read i/p (a & b):");
Scanf("%d %d", &a, &b);
c = a & b;
Printf("a & b = %d", c);
c = a | b;
Printf("a | b = %d", c);
c = a ^ b;
Printf("a ^ b = %d", c);
getch();
}
```

o/p

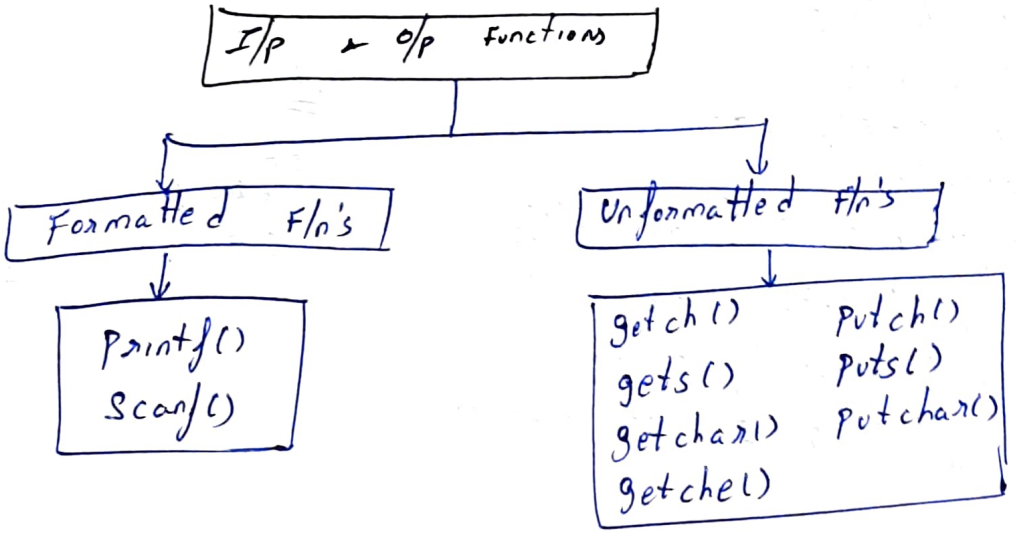
Read i/p (a & b): 8 4

a & b = 0

a | b = 12

a ^ b = 10

Input and Output statements -



Formatted Functions -

It read & write operation is performed on all data types.

printf() -

Syntax → printf ("Control string", var1, var2, ... varn);

{ %d, %s, %f, ... }

```
void main()
{
int x = 65;
float y = 2.2;
char z = 'C';
printf ("%d %f %c", x, y, z);
printf ("\n%d = %c", x, x);
}
```

o/p 65 2.2 C

65 = A

Width & Precision Specifier -

```
// width specifier. <= Use . or *.
Void main()
{ int x=55, y=33;
clrscr();
Printf("\n %.2s", "abcdef");
Printf("\n %.3s", "abcdef");
Printf("\n %.4s", "abcdef");
Printf("\n %*d", 15, x-y);
}
```

o/p

```
ab
abc
abcd.
```



```
// Precision Specifier. <= use .
void main()
{
float g=123.456789;
clrscr();
Printf("\n %.1f", g);
Printf("\n %.2f", g);
Printf("\n %.3f", g);
Printf("\n %.4f", g);
}
```

o/p

```
123.5
123.46
123.457
123.4568.
```

Scanf() -

syntax - Scanf ("Control string", address of var1, address of var2, ... address of varn);

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

```
void main()
{
int a;
clrscr();
Printf("Enter value of A = ");
scanf("%d", &a);
Printf("A = %d", a);
}
```

o/p

```
Enter value of A=8
A=8.
```

Escape Sequences

\n → new line
\b → backspace.
\t → Horizontal tab.
\v → Vertical tab.

Unformatted functions -

It is used for characters, and strings. data only.

getch() & getche() - Read I/P. putch() - Print character.

getchar() -

Syntax => variable name = getchar();

eg
```
char c;
c = getchar();
```

```
void main()
{
char c;
Printf(" Enter a char : ");
c = getchar();
Printf ("a = %c", c);
}
```

o/p
```
Enter a char : g
a = g
```

putchar() -

Syntax => putchar (variable name);

eg
```
char c = 'c';
putchar (c);
```

```
void main()
{
Char c = 'A';
Clrscr();
Putchar (c);
}
```

o/p
```
A
```