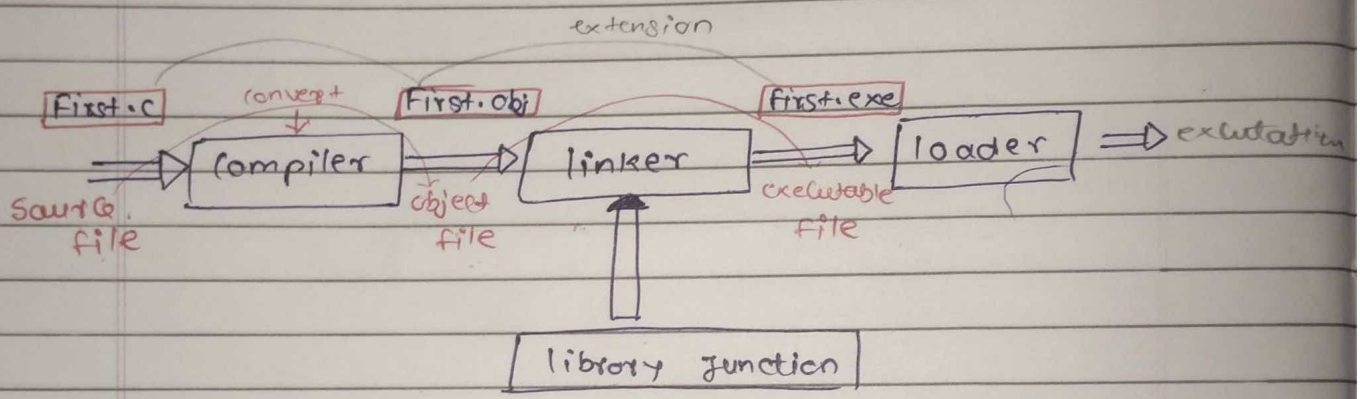
Module - 2 Fundamentals of C- Programming

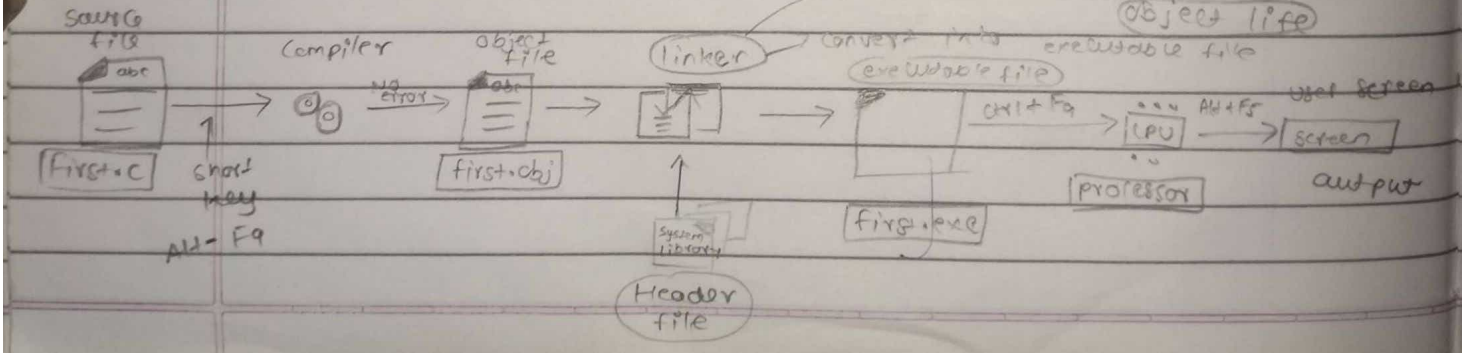
Execution of a program :-

- whenever C program file is complied and excuted
- Compiler generates some files with same name as that of the C program file and with different extensions.
- So, what are these files and How are they created?



or

1. Creating the program
2. Compiling the program
3. linking the program with function that are needed from the C library,
4. Executing the program



Character Set:

- As a every language contain a group of character to used construct word, statement etc.
- C language also feature of set of character which include : alphabets, digit and special symbols
- C language supports a total of 256 character
- Every C program contains statement

These statement are construct by using words and these words are construct using character from the C character list.

(1) digit (3) Alphabet
(2) special symbols

Identifers:

- The identifiers are user-defined words in the C language.
- These are used to name the variable, function, structure etc.

like char (A) = ABHISHEKH
(a) = Abhishekh

• Rule for defining Identifier

(1) It can have either → alphabet (Both upper case and lower case)
→ digit
→ underscore (_)

like int a, int _, int _a
int a_
int 5a X

(2) First letter must be either alphabet, underscore (_)

(3) It can not be use keyword name as identifer

(4) It is case sensitive. → int a = 10 } Both are different
→ int A = 10

(5) Length of identifiers is 31 character.

int a = 10

keyword → Predefined

Identifiers → User defined

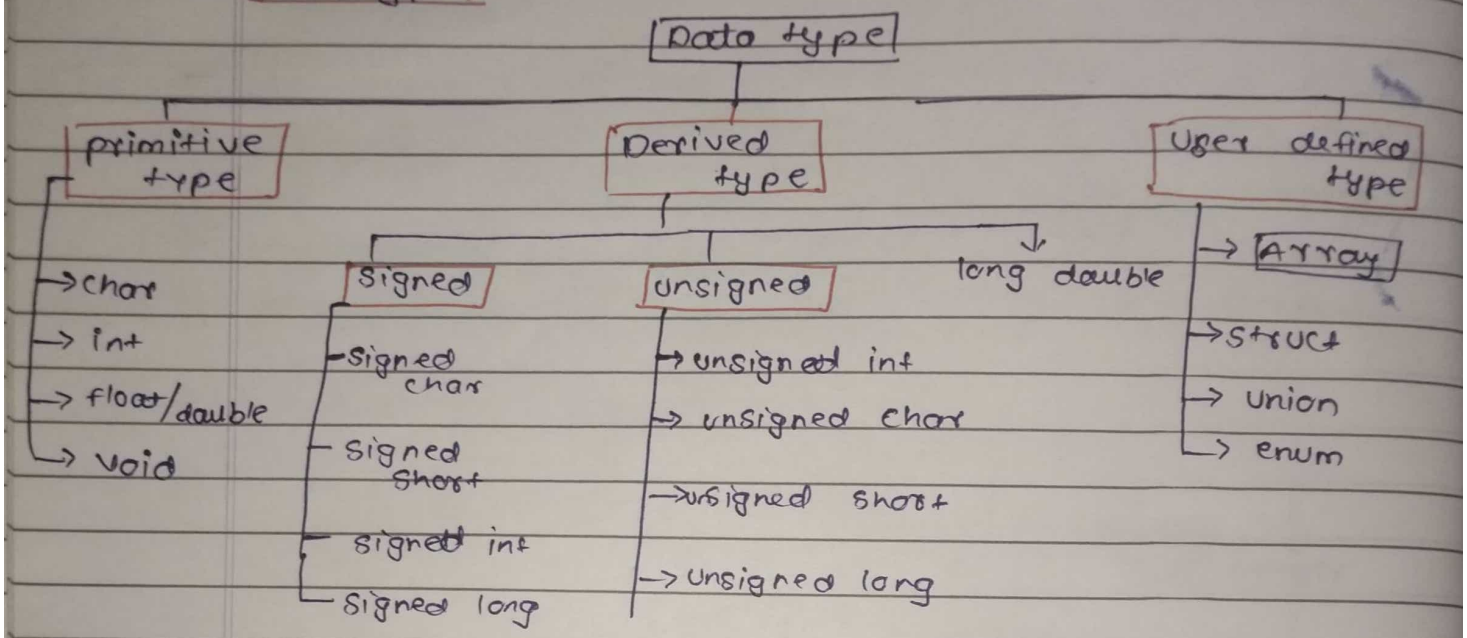
Keywords:

- keywords are predefined,
- reserved words utilized in programming that have special meaning to compiler.
- keywords are part of Syntax and they cannot be used as an identifiers.
- e.g int make;

  Here, int is a keyword that indicates 'make' is a variable of type integer
- C is a case sensitive language

  (All keywords be written in lowercase)
- In programming of c languages there are 32 keywords have their meaning which is already known to the compiler

data type :-



* data type classified into three type

- primitive/primary data type (basic data type OR Predifined data type)
- Derived data type (secondary data type OR Userdefined data type)
- User defined type

#include <stdio.h> ⟶ Header file
main() ⟶ Start compilation
{
printf("Hello world");
}
{ body }
format
end of program semicolon

Input Scanf( )
output printf( )

operator

int a = x + y ; ⟶ expression
operand
operator

(1) Arithematic operator :

— The arithematic operator are the symbol that are used to perform basic mathemical operator like addition, substraction, multiplication division and percentage modulo

operator	meaning	Example
+	Addition	10+5 = 15
-	Substraction	10-5 = 5
*	multiplication	10*5 = 50
/	Division	10/5 = (2)
%	remainder of the division	5%2 = 1

Quotient
2
2) 5
4
1 ⟶ remainder

Arithmetic operator on integers

```
int main()
{
  int x=10, y=5;
printf("sum = %d\n", x+y);
Printf("difference = %d\n", x-y);
printf("product = %d\n", x*y
printf("Quotient = %d\n", x/y
Printf("Remainder = %d\n", x%
```

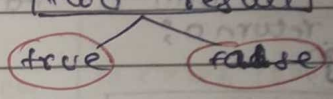
or

Arithmetic operator on Float

```
int main()
{
  float x=5.8, y=2.3;
Printf("Sum = %f\n", x+y);
Printf("diff = %f\n", x-y);
Printf("product = %f\n", x*y);
Printf("Quotient = %f\n", x/y);
return 0;
}
```

(2) Relational operator (<, >, <=, >=, ==, !=)

- The relational operator are the (Symbol) that are (used) to (Compare) two value.
- That means the relational operator are (used) to (check) the relationship b/w two value
- Every relational operator has two result



==	x == y	x is equal to y
!=	x != y	x is not equal to y
<	x < y	less than
<=	x <= y	less than equal to
>	x > y	greater than
>=	x >= y	greater than equal to

eg.

```
# include <stdio.h>
int main()
{
int a=5 , b=5 ;
if (a==b) {
        Printf ("True");
} else {
        print.f ("False");
}
return 0;
}
```

output = True

```
# include <stdio.h>
int main()
{
int a=6 , b=5 ;
if (a!=b)
  {
      Printf ("True");
} else {print.f ("False");
  }
return 0;
}
```

output = ~~False~~ True

```
#include<stdio.h>
int main()
{
int a=6 , b=5;
If (a>b)
{
printf ("True");
} else {
printf ("False"); }
return 0;
}
```

output = True

```
#include<stdio.h>
int main()
{
int a=6, b=5;
If (a<b)
{
printf("True");
} else {
printf("False"); }
return 0;
}
```

output = false

```
#include<stdio.h>
int main()
{
int a=5 , b=5;
If (a>=b)
{
printf ("True");
} else {
printf ("False");
}
return 0;
}
```

output = True

a=b

```
#include<stdio.h>
int main()
{
int a=6 , b=7 }
If (a<=b)
{
printf ("True");
} else {
printf ("False");
}
return 0;
}
```

output = True

a<b

(3) logical operator : (& , && , ||

- The logical operator are the
used to combine multip
one

Return True
If all condition are tru
otherwise return false

operator	meaning	Exomple
&& AND	return True when all the condition match	2>1 && 4<5 is True
	return False if any condition doesn't match	10<9 && 12>10 is False
|| OR	return True if anyone condition match return true if all condition are False otherwise return false	10<2 || 12>10 is False
	return False if anyone condition doesn't match	10<2 || 12<10 is True
! NOT	Return True if condition is false. and Return False if condition is True.	!(10<5 && 12<10) False !(False) is True

```
# Include
int main()
{
int a=5;
If (a==5)
 {
  Printf("True");
 }
 else
 {
  Print f("False");
 }
 return 0;
}
```

output = True

Not operator

```
# include
int main()
{
 int a=5;
 If (!(a==5))
 {
  Print f("True");
 }
  else
 {
  Print f("False")
 }
  return 0;
 }
```

output = False

```
#include <stdio.h>
int main()
{
 int a=15, b=30;
 if (a>=15 && b>15)
 {
  print f("True");
 } else
  {
   Print f("False");
  return 0;
}
```

output = True

```
# include <stdio.h>
 int main()
{
 int a=15, b=30;
 if (a==16 || b>=40)
 {
  Print f("True");
 } else
  {
   Print f("False");
  return a;
 }
```

output = False True ?

operand
x + y

(4) Increment and Decrement : (++ & --)

- The increment and decrement operator are called unary operator because both need only one operand
- The increment operator (++) add one to the existing value of the operand.
- The decreament operator (--) substract one to the existing value of the operand.

operator	meaning	example
++	increment - Adds one to existing value	int a=5 a++; ⇒ a=6
--	decrement - Subtracts one from existing value	int a=5 a--; ⇒ a=4

Pre increment and Pre decrement

```
#include<stdio.h>
int main()
{
  int a=5;
Printf("a=%d\n", a);
Printf("a=%d\n", ++a);
Printf("a=%d\n", a);
Printf("a=%d\n", --a);
printf("a=%d\n", a);
}
```

Post decrement and post decrement

```
#include<stdio.h>
int main()
{
  int a=5;
Printf("a=%d\n", a);
Printf("a=%d\n", a++);
Printf("a=%d\n", a);
Printf("a=%d\n", a--);
Printf("a=%d\n", a);
}
```

- The increment and decrement operators are used in → front of the operand (++a)
  → after the operand (a++)

If it is used in front of the operand { we call it as pre-increment and pre-decrement }

If it is used in after the operand { we call it as post-increment and post-decrement }

pre-increment and pre-decrement:

In the case pre-increment, the value of variable in increased by one before the expression evaluation (e.g. a = 4)

Pre-decrement, the value of variable in decrease by one before the expression evaluation

Program:

```
#include<stdio.h>
int main(){
int a=10, x;
   x= ++a;        // pre increment
   printf("%d", x);   or printf("%d", a);  → output x=11, a=11
                  or printf("%d", ++a);  → output a=12
   return 0;
}
```

because of this one more line

or

```
printf("%d, a=%d", x, a); return 0; }
```

output

x = 11

a = 11

Post - increment and post - decrement:

In the case

Post - increment, the value of variable increase by one after the expression evaluation

Post - decrement. the value of variable decrease by one after the expression evaluation

program :

```
#include<stdio.h>
int main()
{
int a=10, x;
x=a++;          // post- increament
printf("%d", x); or printf("%d", a);
}
```

format specifier

output x = 10

```
printf("%d", a)
or
printf("%d", a++);
```

output a = a++ = 11

or

```
printf("x = %d, a = %d", x, a,);
return 0;
}
```

output x = 10, a = 10++ = 11

```
#include<stdio.h>
int main()
{
int a=10;
printf("%d", a+=30);
printf("%d", a-=30);
printf("%d", a*=30);
return 0;
}
```

odd assignment operator

a = a+30 = 10+30

output = 40

a = a-30 = 10-20

output = -20

a = a*30 = 10*30

output = 300

(=, +=, -=, *=, /=, %=)

(5) Assignment operator :

- The assignment operator are used to assign the right-hand side value to the left-hand side variable
- The assignment operator is used in different variants along with arithmetic operators like =, +=, -=, *=, %=

operator	meaning	Example
=	assign the right-hand side value to left-hand side variable	A = 15
+=	add both left and right-hand value and store result into left-hand side variable	A += B ⇒ A = A+B
-=	Substract right-hand side value from left-side variable value and store result into left-hand side variable	A -= B ⇒ A = A-B
*=	multiply right-hand side value with left-hand side variable value and store result into left-hand side variable	A *= B ⇒ A = A*B
/=	divide left-hand side variable value with right hand side variable value and store result into left-hand side variable	A /= B ⇒ A = A/B
%=	divide left-hand side variable value with right-hand side variable value and store the remainder into left-hand side variable	A %= B ⇒ A = A%B

```
printf("%d", a/=30);
```

a = 4/30 = 19/30

output = 0.33

```
printf("%d", a%=30);
```

a = a%30

output = remainder

(c) **Bitwise operator** (&, |, ^, ~, <<, >>)

- Bitwise operator are used to perform individual bits of a number.
- It can be used only integer or character type value not float, double etc.

A = 25 (11001) B = 20 (10100)

Operator	meaning	Example
&	- The result of Bitwise AND is 1 if all bits are 1 otherwise it is 0	A & B = 16 = (10000)
|	- The result of Bitwise OR is 0, if all bits are 0 otherwise it is 1	A | B = 29 = (11101)
^	- the result of Bitwise XOR is 0 • if all the bit are same	= A^B => 13 = (01101)
~	the result of Bitwise one's complement is negation of the bit	~A = 6 (00110) A = 11001 ~A = 00110
<<	the Bitwise left shift operator → Shift all the bits to the left by the specified no. of opposition	A << 2 = 100 = (1100100)
>>	the Bitwise Right shift operator → Shift all the bit to the right by the specified no. of position	A >> 2 = 6 = 00110

The conditional operator behave like or If-else statement

e.g a+b+c

(7) Conditional operator (?:)

- The conditional operator is called a ternary operator.
- Because it requires three operands.
- This operator is used for decision making.
- 1st we verify a condition then we perform one operation out of two operation.
- If the condition is TRUE the 1st option is performed
- If the condition is FALSE the 2nd option is performed

Based on condition result

Syntax :- Condition ? True part : False part ;

ex :- ①

A = (10<15) ? 100 : 200 ; ⟹ A value is 100

e.g ②

B = (10>15) ? 100 : 200 ; ⟹ A value is 200

Program :

```
#include<stdio.h>
int main()
{
    int a, b, c, Max;
    printf("Enter Three Numbers:");
    Scanf("%d %d %d", &a, &b, &c); //a=23 b=78 c=45
    Max = (a>b) ? (a>c ? a : c) : (b>c ? b : c)
    printf("%d", max);
    Return 0;
}
```

Because operator 1 → 1 = 1

//23>78 false

//78>45 True

three numbers a= 23 b=78 , c=45

output = 78

(8) **Size of operator**: only perform single operand

- This operator is used to find the size of memory (in bytes) allocated for a variable. or datatype

eg **Syntax**: Size of (variable Name);

e.g
```
Size of (a) ;   → 2 or 4 bytes
Size of (int) ; → 2 or 4 bytes
```

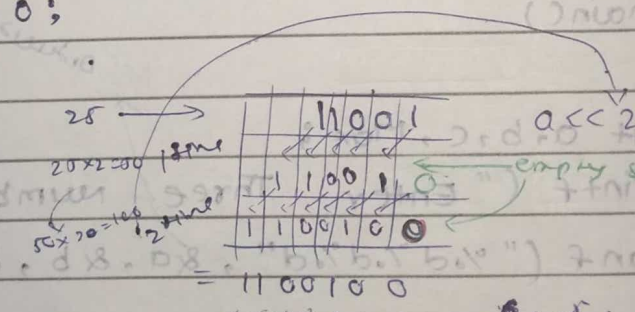
a&b ⇒ a = 11001, b = 10100 → 10000 = 16

```
#include <stdio.h>
int main()
{
int a ; char b ; float c ;
printf ("%d bytes", size of (a));
printf ("%d bytes", size of (b));
printf ("%d bytes", size of (b));
Return 0;
}
```

a|b → a = 11001, b = 10100 → 10101 = 21

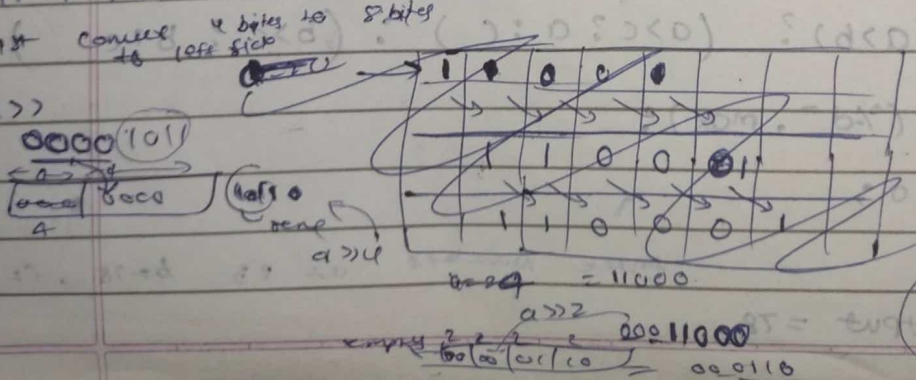
a^b ⇒ a = 11001, b = 10100 → 01101

a = 11001, ~a = 00110



a=11 → 1011

if connect 4 bits to 8 bits to left side



Bitwise AND

the result of Bitwise is one if all bit are (1) otherwise it is (0).

Bitwise operator :- It is used to perform individual bits of number

- It is used only character and integer datatype

a = 25 (11001) , b = 20 (10100)

operator	meaning	example
&	The result of Bitwise AND is 1 if all bits are 1 otherwise it is 0	a = 25 , b = 20 a & b = 10000 = 16
|	The result of Bitwise OR is 0 if all bits are 0 otherwise it is 1	a = 25 , b = 20 a | b = 11101 = 29
^	The result of Bitwise XOR is 0 if all bits are same	a = 25 , b = 20 a ^ b = 13 = 1101
~ (1 to 0 0 to 1)	The result of Bitwise complement is negation of Bit	a = 25 ~a = 00110 = 6
<<	The Bitwise left shift operator → shift all bits to the left multiplication of (2)	a = 25 a << 2 = 1100100 = 100
>>	The Bitwise Right shift operator → shift all bits to the Right divide by 2 $\frac{5}{2} = 2.5$ → take (2) negleat point (0.5)	a = 10 a >> 3 =

Preference

~	<<	>>	&	^	|

Type conversion (casting):

- Type conversion (casting) is nothing but the new way of converting a data type of one variable to some other datatype.

- Type conversion (casting) is of 2 types:

Implicit type conversion (casting)

- It is done by the compiler and there is no loss of information

The convert small datatype to large datatype

ex:-
let x is 8 bit Number
y is 16 bit Number
Now perform addition (x+y)

Explicit type conversion (casting)

- It is done by the programmer and there will be a loss of information

The convert large datatype to small datatype

ex:-
float x = 3.435;
int y = (int)x;

output = 3 (0.435)

Syntax:
int i = 10;
Float x = 15.5;
char ch = 'A';

i = x; =====> x value 15.5 is converted as 15 and assigned to variable i

x = i; =====> Here i value is 10 is converted as 10.000 and assigned to variable x

i = ch; =====> Here the ASCII value of A(65) is assigned to i

program :-

```
# include<stdio.h>
int main()
{
int i=10;
float x=15.5;
char ch='A';
  i=x;
printf("i value is %d\n", i);
        x=i;
printf("x value %f\n", x);

        i=ch;
        printf("i value %d\n", i);
}
```

float ka identity specifier

output :

i value is 15
x value is 10.000000
i value is 65

we convert the data from one data type to another datatype by explict type casting

Type casting :-

Type casting is also called an explicit type conversion

-> data not loss

example :-

```
int total mark = 450 ,  max Marks = 600 ;
float average ;

average = (float) total mark / max marks * 100 ;
```

- In above example code, both total marks and maxmarks are integer data type value.
- When perform total marks / maxmarks the result is float value
- But the destination (average) datatype a float.
- So we use type casting to convert total marks and max marks into float data type

```
# include<stdio.h>
int main()
{
  int a, b, c ;
  float avg ;
  Printf "("Enter any three integer value : \n");
   Sacnf(" %d %d %d", &a , &b , &c);

  avg = (a+b+c)/3 ;
Printf("avg before casting = %d \n", avg);
  avg = (float) (a+b+c)/3 ;
} Printf ("avg after casting = %d \n", avg);
  return 0;
```

and put
Enter any three integer value : 1 2 3
avg before casting = 2.000000
avg after casting = 2.000000

most Imp

Structure of C program

- The structure of C program can be mainly divided into six parts.
- Each part as have different purpose for executed the prog
- It makes the program → easy to read, easy to modify, easy to document

There are six sections → responsible for the proper execution of program

(1) Documentation
(2) Preprocessor section or link section
(3) Definition
(4) Global Declaration
(5) main() Function
(6) Sub programs

(1) Documentation → write a comment but not excuted by compiler
it is statement which is not executed by compiler

- In C Program

Single-line comments → can be written → using → two forward slashes

i.e //

we can create

multi-line comments → using /* */

e.g
```
/* File name : area of circle.c
   author : Elon musk
   Date : 22/6/20
   decription : programme to calculate
                area of circle */
```

```
1  /* File name : area o
   author : Elon mu
   Date : 22/6/20
   decription : Programme
2  #include <stdio.h>
3  # define Pi 9.14
4  float area (float) ;
5  int main ()
   {
6  float r, 0;
   printf (" Enter the rad
   scanf (" %d", &r);
   int a = area (r);
   printf (" Area of circle is
   %d\n", a
   }
   float area (float r)
   int A;
   A = pi*r*r;
   return A;
   }
```

(2) preprocessor section or link section : Include Header files whose function are to used in the program

e.g :- #include<stdio.h>
#include<conio.h>
#include<math.h>
#include<time.h>

- The "#include" is used to include the header file with ".h" extension in the program

(3) Definition

- All the Symbolic constants are written in the definition section (Pi 3.14, MAX 1000)
- Symbolic constant are known as macros
- "#define" directive is used to define macros

e.g
#define Pi 3.14
#define MAX 100

- The define statement does not end with a semicolon
- macro definition is not variable.

(4) Global Declaration :

- All Global variable are written in this section.
- Includes all global variable, function declaration, static variable.
- The global variable can be access anywhere in the program. thy for is called global variables

Syntax:
datatype Variablename;

ex:-
float r;
int result;

(5) main() Function :

- In the structure of C program this section contains the main function of the code.
  - In C program compiler start from main function
- It can be used → static variable, printf(), scanf(); in built function; global variable; user defined function → sum(), fact()
- return type of main function → void / int

```
int main()
{
float r,a;
printf()
scanf()  a=area(r);}
```

(6) Subprogram section :

- This includy the user-defined function called in the main() function
- user-defined function generally written after the main() function.
- this function is called in the main function . it return an integer to the main function

```
int a=area(r);
return area;
```

Following built-in function

(1) printf() :- The printf() function is used to

print → string / data value → combination of string and data value

on the output screen (user screen)

- The printf() function is a built-in function → defined in a Header file called "stdio.h"
- when we use printf() function in our program we need to include the respective header file (stdio.h). using # include statement

- Syntax :

```
Printf ("message to be display!!!");
```

Program :

```
#include<stdio.h>
int main(){
Printf("Hello world");
return 0;
}
```

output : Hello world

```
# include <stdio.h>
int main(){
  int a;
Printf (" integer value = %d", a);
```

Printf(" string format string , variable name);

output : integer value = 10

(2) Scanf() :- The Scanf() function is used to

Input function | read → multiple data value of different data type from the keyword.

- The Scanf() function is a built-in function → defined in a Header file called "stdio.h"
- when we use Scanf() function in our program we need to include the respective Header file (stdio.h) using # include statement

Syntax :

```
Scanf(" format string", & variable name);
```

e.g : Program

```
# include<stdio.h>
int main(){
  int i;
Printf("\n Enter any integer value:");
  Scanf("%d", &i);
Printf("\n integer value:%d\n", i);
  return 0;
  }
```

output

Enter any integer value: 5

integer value : 5

② getch() ③ getchar() } Both are same and Input function

- The getch()/getchar() is used → to → read → a character from the keyword and return it to program.
- This function is used to read a single character
- To read multiple character → we need to write → multiple times or use a looping statement

e.g: Program

```
#include<stdio.h>
int main()
{
  char ch;
  printf("\n Enter any character :");
  ch = getchar();
  or
  ch = getch();
  Printf("\n you have entered : %c\n", ch);
}
```

output Enter any character : A
you have entered : A

④ gets() Function - the gets() function is used to read line of string and store it into character array

e.g: Program:

```
#include<stdio.h>
int main(){
char name[30];
printf("\n Enter your website :");
gets(name);
Printf("%s", name);
return 0;
}
```

output:
Enter your website : ramstdio
ramstdio

⑤ fscanf() function
The fscanf() function is used with the concept of files
used to read data values from a files

output function

The printf() function is used to
printf → String
→ Data value
→ Both combination

(1) printf()

(2) putchar() or putch() : The putchar function, is used to print single character on the output screen

ex :-

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

output : A

(3) puts() : The puts() Function is used to display a string on the output screen

```
#include<stdio.h>
void main(){
    char name[30];
    printf("\n Enter your favourite website :");
    gets(name);   -> Scanf
    puts(name);   -> Printf
}
```