I - UNIT PPS

1. Define algorithm? Explain properties of algorithm & example?
2. Define flow chart? what are symbols in flow chart & example?
3. Explain about constants?
4. Define variable? Rules for creating variable?
5. Explain about data types?
6. Explain about operators?
7. Explain about control statements (control structures)?
8. Explain about type casting?
9. Explain about storage classes?
10. Explain operator precedence & associativity?
11. Structure of a C program?
12. Explain about command line arguments?

(1) Define algorithm & Explain properties of algorithm & example?

Algorithm :- It is a finite no. of steps to complete a task in a given problem, in a finite amount of time.

Properties :- Every algorithm should follow the following properties.

1) Finiteness :- An algorithm should be terminated in some finite no. of steps.

2) Definiteness :- Each statement in an algorithm should be stated clearly for better understanding

3) Effectiveness :- It is how easily we are converting an algorithm statement into program statement.

4) Generality :- Once an algorithm is written that should work for other data of same kind.

5) Input & Output :- An algorithm can take min of 0 & more input & produce 1 & more outputs.

Example :- Write an algorithm for adding of 2 numbers,

Step-1 :- Start

Step-2 : Declare variables a, b, c.

Step-3 :- Enter values in a & b

Step-4 :- Calculate

step-5:- Display 'c'

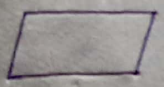
step-6 :- Stop.

(2) (a) Define flowchart? What are the symbols in flowchart & Example?

Flowchart is a graphical (or) Diagramatic representation of an algorithm is called flowchart.

By the symbols used in flow chart are :-

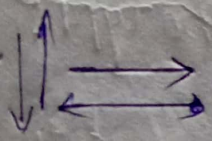
1.  oval (or) ellipse → start & stop.

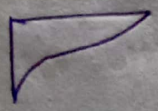
2.  parallelogram → input & output

3.  Rectangle → processing/calculations

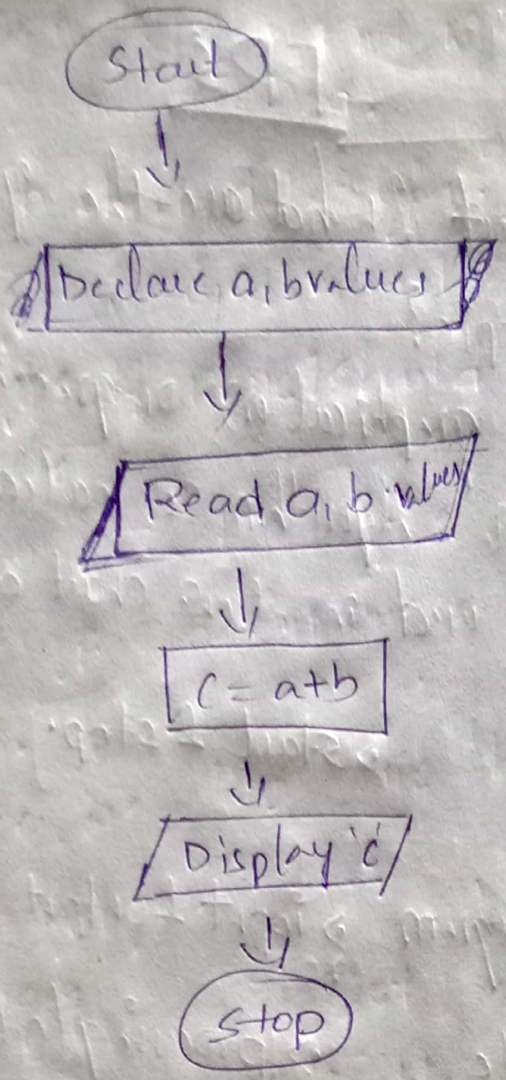
4.  Rhombus → condition/selection/Decision.

5.  circle → connector or continuation

6.  Arrow → flow lines.

7.  → Documentation of comments.

Examples:- (a) Draw a flow chart for addition of two numbers.



3. Explain about Constants?

Constants:- The values for these variables are fixed cannot be modified.

* constants are categorized into two types:

i) Numeric constants (or) Number constants

ii) Characters constants.

i) Number (or) Numeric constants:-

a) Integer constants (or) Decimal: +10, -15, 25, 35

b) Real constant Ex:- +10.2, 5.3, -1.5

ii) Character constants :-

a) Single character constants : In this a character is enclosed by a single quotation.

Ex: 'A', 'a', 'b', 't'

b) String constant :- A string is a group of characters enclosed by double quotations.

Ex: "cse", "A", "b", "123".

4 Define variable & Rules for creating a variable?

Variable :- variables are used to store a value in a memory the value of the variable may be changed during the execution of a program.

Rules for creating variable :-

* All variable names must be start (or) begin with a letter of the alphabet or an underscore (_).
* After the first initial letter, variable names can also contain letters & numbers.
* Upper case characters are distinct from lower case characters.
* You cannot use a C++ keyword (reserved word) as a variable name.
* variable names should not be same as keywords.
* length of the variable name should not exceed 32 characters.

* Variable name should not with digit.
* spaces are not allowed.

5. Explain about data types?

Datatype is used to indicate,

i) What type of value should be stored by a variable

ii) The amount of memory allocated for a variable.

iii) Type of operations are allowed to perform on a variable.

iv) There are four basic datatypes available in 'c' language.

1. Integer 3. Float

2. Character 4. Double

Integer :- This datatype is used to store, whole numbers.

* Keyword used is "int" to declare a variable of integer type.
* format specifier is %d
* The amount of memory allocated to integer datatype is 2 Bytes.
* Range is -32,768 to +32,767.
* Various forms of integer datatypes.

(a) Unsigned integer :- Which is used to store only +ve numbers.

- keyword - Unsigned int
- format specifier - %d
- Memory - 2 bytes total 16 bits are used to store the value.
- Range - 0 - 65,535

(b) Signed integer : Which is used to store both +ve & -ve numbers.
- keyword - signed int
- format specifier - %d
- Memory - 2 Bytes (out of 16 bits 1 bit is used to store the sign, & 15 bits is used to store the value.
- Range - -32,768 to +32,767.

(c) Short Integer : This is used to store the small values.
- keyword - short int
- format specifier - %d
- Memory - 1 byte
- Range - -128 to +127

(d) Long integer : This is used to store the large values.
- keyword - long int
- format specifier - %ld
- Memory - 4 Bytes
- Range - -2147483648 to +2147483647.

Character: this is used to store a character

By initialising a character to a variable, characters must be enclosed by a single quotations.

- keyword - char
- Format specifier - %c
- Memory - 1 byte
- Range - -128 to +127.

⇒ character Datatypes is of two types.

a) signed character:

keyword - signed char

format specifier - %c

Memory - 1 byte

Range - -128 to +127

b) Unsigned character:

- keyword - Unsigned char
- Memory - 1 byte
- format specifier - %uc
- Range - 0-255

Float: which is used to store, real numbers (upto eight digits after the point)

- keyword - float
- format specifier - %f
- Memory - 4 bytes
- Range - -3.4E+38 to 3.4E+38

Long float :- which is later more than [illegible] point.

- keyword - long float
- format specifies - %lf
- Memory - 8 bytes
- Range - -1.7E+808 to +1.7E+808.

Double :- which is same as long float

keyword - double

format specifieq - %lf

Memory - 8 Bytes

Range - -1.7E+308 to +1.7E+308.

Long double :-

Memory - 10 bytes

keywords - long double

format specifies - %Lf

(6) Explain about Operators?

Operators are used to perform Operations on operands.

Operands

c = a+b

Operators

Operators & Operands will form Expressions

Ex:- c = a + b

operands

operators.

⇒ Various Types of Operators available in C language.

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Increment / Decrement Operators
6. conditional (or) Ternary operators
7. Bitwise Operators
8. Special Operators.

1. **Arithmetic Operators:** these Operators are used to perform arithmetic Operations.

Operator	Meaning
+	Addition
−	Subtraction
*	Multiplication
/	Division
%	Modulo Division

Note: '/' Operator gives Quotient as a result whereas '%' Operator gives remainder as a result

Ex: $10/3 = 3$

$10\%3 = 1$

$$\begin{array}{r|l} 3\,)\,10 & 3 \\ \underline{9} & 1 \\ 1 & \end{array}$$

=> Arithmetic Operators can be implemented in three forms.

1. Integer Arithmetic
2. Real arithmetic
3. Mixed mode arithmetic

i) Integer Arithmetic :-

In this all the input variables are of integer type & the result is also of integer type.

ii) Real Arithmetic :-

In this all the input variable are of float type & the result is of type float.

iii) Mixed mode Arithmetic :-

In this some input variables are of type integer & some are of type float & the result is of the type float.

2. Relational Operations :-

These Operators are used to compare two variable.

Operator	Meaning
<	less than
>	greater than
<=	less than (or) equal to
>=	greater than (or) equal to
==	Equal to operator
!=	Not Equal to

The relational operators returns a value 1 if the expression is true & returns '0' if it is false.

Ex: a=5, b=6.

i) d = a<b iii) d = (a==b)

ii) d = a>b iv) d = (a!=b)

3. Logical Operators :-

These operators are used to combine two or more relational expressions.

Operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT.

AND

a	b	AND
0	0	0
0	1	0
1	0	0
1	1	1

OR

a	b	OR
0	0	0
0	1	1
1	0	1
1	1	1

NOT

a	NOT a
0	1
1	0

Ex :- a=5, b=6, c=7

1) d = (b<c) && (a>c) - 0

2) d = (b<c) || (a>c) - 1

3) d = !(b>a) - 0

4. Assignment Operators :-

These Operators are used to assign a value to a variable (or) copy a value from one variable to another variable.

Operator	Meaning
=	Assignment

Ex:- a = 10;
b = a; /* 'a' value '10' is copied to 'b' */

Short hand Assignment :-

a = 5 ; b = 6 ;

p /= q; a = a + b; a = a - b; c = c * d;

p = p/q; a + = b; a - = b c * = d;

5. Increment - Decrement Operators :-

These Operators are used to increment a value by one (++) & decrement a value by one

→ Increment (++) :- Available in two forms.

a = 6 , b = ++a , a = 7 b = 7.

1) Pre-Increment : ++a ++a

A value is incremented first & then assigment

2) Post-Increment : a++ a++

→ A value is assigned first & then incremented.

→ Decrement (--) :- Available in two forms.

1) Pre-Decrement : ~~--a~~ --a --a

→ A value is Decremented first & then assigned

2) Post-Decrement : a--

→ A value is assigned first & then decremented.

6. Conditional (or) Ternary Operator :- (?)

This operator is used to execute one set of statement when the expression is true & executes the another set of statement when the expression is false.

Syntax :- Expression 1 ? Expression 2 : Expression 3

false

true

Ex :- a = 5 ; b = 4

1) d = a > b ? a : b ⇒ d = 5

2) d = a < b ? a : b ⇒ d = 4

If the exp 1 is true & exp 2 is executed and

If the exp 1 is false exp is executed.

7. Bitwise Operators :-

These operators are used to perform on Binary numbers.

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Exclusive OR
~	1's complement
<<	left shift
>>	Right shift

① Bitwise AND (&): This will return '1' when both the inputs are '1' otherwise it will return '0'.

Ex: a=5 b=10

c = a&b

↓

0

a=5 → 0101

b=10 → 1010

0000 → 0

② Bitwise (OR) (|): This will return '1' when any one of the input is '1' otherwise return '0'.

Ex: a=5 b=10

a|b

a=5 → 0101

b=10 → 1010

1111 → 15 (or) -1

③ Exclusive (OR) EX-OR: This will return '1' when both the inputs are different & return '0' when inputs are same.

a	b	EX-OR
0	0	0
0	1	1
1	0	1
1	1	0

Ex: c = a^b

↓

-1 (or) 15

a=5 → 0101

b=10 → 1010

1111

15 (or) -1

④ 1's complement: In this 1's are changed into 0's & 0's are changed to 1's.

Ex: c = ~a

a=5 → 0101

~a 1010 → 10

⑤ Left shift (<<): This is used to shift the bits in a given binary number towards left to the required no. of positions.

Syntax: x<<y

$x \times 2^y$

'y' no. of bits are shifted from 'x' towards left and trailed bits positions are filled with zero.

Ex:- 1) a=5; a<<1

00000101<<1

00001010 = 10.

2) a=5; a<<2

00000101<<2

00010100 = 20

3) a=7; a<<2

a=0111; 0111<<2

4) b=13; b<<3

b=1101; 1101<<3

00001101 = 01101000

= 104.

(6) Right shift (>>): this is used to shift the bits in a given binary number towards right to the required no. of positions.

Syntax: x>>y $\left(\frac{x}{2^y}\right)$.

'y' no. of bits are shifted from 'x' towards right.

Ex:- 1) a=5; a>>1

a=0101; 0101>>1

00000010 = 00000010 = $2\left(\frac{5}{2}\right)$.

2.5.

2) a=10; a>>2.

a=1010; 1010>>2

00000010 = 00000010 = $2\left(\frac{10}{2^2}\right)$

2.5.

(8) Special Operators:- there are two operators under special operators.

i) size of Operator

ii) Member selection

i) Size of Operator: this operator is used to find the size of a variable (or) a size of a datatype.

Syntax :- size of (variables);
(or)
size of (datatype);

Ex:- int a; x;
float b;
x = size of (a);
x = size of (float);

ii) Member Selection Operator :-
dot (.) is used to select the members in structures & unions.

(7) :- Explain about control statements? (control structures)

Conditional statements (or) Selection statements (or) Branching (or) Decision (or) control statements :-

In a 'c' program the instruction are executed in a sequential fashion.
If we want to alter the execution sequence depending on the condition (or) the situation than these conditional (or) selection statements is used.

⇒ Selection statements are available in 4 forms :- (if statements)

i) Simple if
ii) if - else
iii) Nested if else

i) Simple if :-

Syntax :- if (condition)

{

statement1;

}

statement2;

(if block)

→ 'if' condition is Evaluated first, if condition is true statement 1 is Executed then continous with the next immediate statement i.e, statement 2.

→ if the condition is false there is no alternative then the Execution is from statement 2.

Note :- if statements will not Ends with a semicondut

Q) Write a 'c' program to display the value of 'a' if a is greater '10'.

```
# include <stdio.h>
# include <conio.h>
void main ( )
{
  int a;
  clrscr();
printf ("Enter a value");
Scanf ("%d", &a);
if (a>10)
```

```
{
printf (" the value a is greater than 10 and
                a is %d", a);
}
getch();
}
```

ii) if else :-

Syntax :- if (condition)

```
/*if block*/ {
{
statement 1;
}
else
{
statement 2;
}
statement 3;
```

/*else block*/

→ if condition is true statement 1 is executed then continous with the stmt 3.

→ if condition if false stat 2 is executed then continous with stmt 3.

Q: write a c-program to check whether the given value is even (or) odd;

```
# include <stdio.h>
# include <conio.h>
void main()
{
int a;
clrscr();
```

```
Printf ("Enter a value");
Scanf ("%d", &a);
 if (a%2==0)
 {
 printf ("the value 'a' is odd and a is %d", a)

 getch ();
 }
```

iii Nested if-else :-

Syntax :-

```
if (condition 1)
{
  if (condition 2)
  {
   stmt 1;
  }
 else
  {
  stmt 2;
  }

}
else
{
  if (condition 3)
  {
  stmt 3;
  }
  else
```

```
{
 stat u;
 }

}
statement 5;
```

→ Writing of if-else in another if-else is known as nested if else.

→ if condition 1 is true, condition 2 is evaluate
cond^n 2 - true, stat 1 is executed
cond^n 2 - false, statut 2 is Executed

→ if condition 1 is false, condition 3 in else block is Evaluated.
cond^n 3 - true, stat is Executed
cond^n 3 - false, stat u is Executed

After the if-else the controller continues with the Execution of stat 5.

(a) Write a c-program to find the biggest of three numbers, by using nested if 'else.

```
# include <stdio.h>
#include <conio.h>
void main()
{
int a,b,c;
clrscr();
printf ("Enter three values");
```

```
Scanf("%d\t%d\t%d", &a, &b, &c);
 if (a>b)
 {
     if (a>c)
     {
     printf("%d is greater then of three
              numbers ;a);
      }
      else
      {
      printf("%d is greater of three numbers", c);
      }

  }
  else
  {
  printf  if (b>c)
          {
           printf("%d is greater of three
                    numbers", b);
          }
          else
          {
           Printf("%d is greater of three
                    numbers", c);
          }
  }
  getch();
  }
```

iv) else-if ladder:-

Syntax:- if (condn 1)
{
stmt 1;
}
elseif (condn 2)
{
stmt 2;
}
elseif (condn 3)
{
stmt 3;
}
else
{
stmt 4;
}
stmt 5;

-> condition 1 is evaluated first
if con-1 is true - stmt 1 is Executed.
con-1 is false - condn-2 is Executed.
if con-2 is true - stmt 2 is Executed
con-2 is false - condn-3 is executed
if con-3 is true - stmt 3 is executed
cond-3 is false - stmt 4 which is in the else block will be executed then
Continously with. stmt 5.

Q. Write a C-program to read '6' subject marks of a student and find out the average and find the class of marks (1class, 2class, 3class, fail).

distinclinction - >= 75

1 class - ≥ 60 & ≤ 74

2 class - ≥ 50 & ≤ 59

3 class - ≥ 40 & ≤ 49

fail - < 40

By using if else if ladder

Ans:-

```
#include <stdio.h>
#include <conio.h>
void main()
{
int s1, s2, s3, s4, s5, s6, avg;
clrscr();
printf("Enter the 6 subject marks")
scanf("%d\t%d\t%d\t%d\t%d\t%d", &s1, &s2, &s3, &s4, &s5, &s6);
avg = (s1+s2+s3+s4+s5+s6)/6;
printf("Average of the student is %d", avg);
if (avg >= 75)
{
Printf("The student passed in distinction");
```

```
else if (avg >= 60 && avg <= 74)
  {
  printf("the student passed in 1st class");
  }
else if (avg >= 50 && avg <= 59)
  {
  printf("the student passed in 2nd class");
  }
else if (avg >= 40 && avg <= 49)
  {
  else
  {
  printf("the student failed");
  }
  getch();
  }
```

(8) Explain about type casting? (or) type conversion?

The process of converting a data (or) value from one datatype of another datatype is called type casting (or) conversion.

Type conversion is done in two types ways:

i) Implict Type conversion (or) widening

ii) Explict Type conversion (or) Narrowing (or) type casting

Implicit type conversion:

In this the smallest datatype value is copied to the largest datatype. During this conversion no data will be lost.

Syntax:- destination-vas = source-variable;

Ex:- int a=5;
float f;
f=a;
5.0.5

Q. Write a 'c' Program to impliment the concept of implict type conversion.

```
# include <stdio.h>
#include <conio.h>
void main()
{
int a=5;
float f;
clrscr();
f=a;
Printf("Value of a after implict conversion is %f,-f");
getch();
}.
```

→ Explicit type conversion:

In this the largest datatype value is converted into the smallest datatype value. During this process some value maybe lost. This conversion is also known as type casting (or) narrowing.

Syntax:- destination-var = (destination-datatype) source

Ex:- float f = 3.4;
int a;
a = (int)f;
3

Q:- Write a 'c' program to impliment the concept of Explicit type conversion.

```
#include <stdio.h>
#include <conio.h>
void main()
{
float f = 3.4;
int a;
clrscr();
a=(int)f;
printf("value of f after Explicit type conversion is %d",a);
getch();
}
```

(9) Explain about storage classes?

Storage classes specifies about

1. where a variable is stored in a memory.
2. if a variable is declared and not initial what is the initial value of a variable
3. scope of a variable
4. Life time of a variable.

* There are four types of storage classes a in c-langnag.

i) Automatic storage class

ii) ~~Internal~~ storage class

iii) static storage class

iv) Register storage class

i) Automatic storage classes:- (local variables)

• All the local variable declarations comes under the automatic storage class by default.

• If we want to declare automatic variable "auto" is a keyword used.

• All the automatic variables are stored in a memory location in a memory

• The intial values for automatic variables is "Garbage value" if not initialised.

→ scope of an automatic variable is within the block.
→ life time of the automatic variable is within the block (starts & ends to the declaration).

Ex:-
```
void main()
{
auto int a;
printf("%d", a);
}
```

o/p:- Garbage values.

ii) Internal storage classes (global variables):-
→ All the global variables are external variables by default.
→ Extern is a keyword used to declare an external variable.
→ External variables are stored in a memory locations of a memory.
→ The initial value of the external variable if it is throughout the declared and not initialised is zero ("0").
→ scope of the external variable is throughout the program.
→ Life time of the external variable is also throughout the programm.

Ex:-
```
int x, y;
void main()
```

```
{
Printf("%d %d", x,y);
}
```

O/p:- 0 0

```
void main()
scanf in x,y;
Pf("%d %d", x,y);
}
```

O/p:- 0 0

iii) <u>Static storage class:-</u>

- The static variables are initialised only once in it life time.
- Static is a keyword used to declare a static variable.
- static variables are stored in the memory location of a memory.
- The initial value of the static variable, if it is declared & not initialised is zero("0").
- Scope of the static variable is within the block.
- Life time of the static variable is within the block.

Ex:

```
Void main()
{
static int a, b=3;
b=u; /* It will not initized again
bcoz it is static*/
/* It is only initialized once in a
life time*/
Printf("%d %d", a,b);
}
```

O/P:- 0 3
a b

iv) Register storage class:-

- Register is a smallest memory location which is very nearer to the CPU.
- If we store any values in the register then the CPU performs calculations in a faster way
- Register is a keyword used to declare a variable of register type.
- Register variables are stored in Registers instead of memory allocation.
- The initial value for register variable, if it is not initialised is "Garbage value".
- Scope of register variable is within the block
- Lifetime - within the block

Ex:-

```
void main()
{
register int a;
pf("%d", a);
}
```

O/P:- Garbage value.

(10) Explain Operator precedence & associativity?

The Operator precedence tells to the compiler that which operation should be evaluated.

List:

Operator	Description	Associativity
( )	Parenthesis	L-R
[ ]	Square braces	L-R
-, ++, --, !	Unary operators	R-L
*, /, %	Arithemetic multiplication, Div, modulus	L-R
+, -	Add & sub	L-R
<, <=, >, >=	Relational Operators	L-R
==, !=	Equality Operators	L-R
&	Bitwise AND	L-R
^	Bitwise EX-OR	L-R
|	Bitwise OR	L-R

&&	Logical AND	L-R
||	Logical OR	L-R
?:	conditional Operators	R-L
=, +=, -=, *=, /=, %=	Assigment Operator	R-L

Expression Evaluation:-

1) int a=10, b=16, c=5;

x = a-b/u+c*2-3

x = 10-16/u+5*2-3

= 10-4+5*2-3

= 10-4+10-3

= 6+10-3

= 16-3

= 13.

3) int i=10, c=15;

float f=2.5

x = (i+f)<12

= 10+2.5<12

= 12.5<12 - (F)

x = 0.

2) int a=10, b=18, c=5

x = a-b/(u+c)*(3-2)

x = 10-18/(u+5)*(1)

x = 10-18/9 *1

x = 10-2*1

x = 10-2

x = 8.

4) int i=10, c=15,

f=2.5

+i c<=f*i*10;

15+10c<= 2.5*10*10;

15+10c<= 250

25c<=250 - True

=>1.

5) int i=10; float f=2.5
char c='A'

(i>5) || (c=='a') && (i<f)

(10>5) || (A=='a') && (10<2.5)

0 && 0

0 || 0

1 - True

6) i=5, J=10, K=15

K+=(i>10&&i==10)?++i:J

15+=(5>10&&5<++5;5<10)

0?6:5

15+=5

15=15+5=20

7) x=8, y=12, z=-3

a= ++x - z-- + (x+y)

= ++8 - z-- + (8+12)

= ++8 - (-3-- + 20)

= 9 - -4 + 20

= 13 + 20

= 33

(11) structure of a c program?

structure of c-program.

1. Preprocessor Directives

#include <stdio.h> < printf() scanf() >

#include <conio.h> < clrscr() getche() >

2. Global variable declaration (optional)

3. Main function

void main()

{

4. Local variable declaration

5. Executable statement ; c=a+b
}
6. User defined functions (optional)
{
}

1. Preprocessor Directives:-
These statements are executed before the actual program execution begins.
Preprocessor Directives are starts with # symbol.
These Directives are involved by the compiler for processing of some files.

Ex: #include <stdio.h>
#include<conio.h>.

⇒ stdio.h → standard input output header file.
It provides the functions printf(), scanf(), for o/p & i/p.

⇒ conio.h - console input output headerfile

⇒ printf():- used as output function
Syntax:
Printf ("Message");
Printf ("Message or format specifier", variable);

⇒ Scanf():-
This function used for reading the data (or) taking the input.

Syntax:-

Scanf ("format specified", & variable);

⇒ conio.h - console input output header files.

It provides the functions clrscr() & getch()

2. Global variable declaration

The variables defined under this can be used throughout the program.

3. Main function:- void main()
{
}

This is the place where the actual execution of a program begins & it is sourrounded by a pair of curly braces.

4. Local variable declaration:-

Whatever the variables which we are going to use in a program should be declared under this

5. Executable Statements:-

Making use of Printf(), Scanf() functions & any Expressions are called as executable statements. All the statements should end with a semicolon

Note:- Semicolon (;) indicates end of a statement.

6. User Defined functions:-

This section is used when the user wants to define his (or) her own functions.

→ Example:-

Write a 'c' program to display a statement "Welcome to CSE-B".

```
# include <stdio.h>
# include <conio.h>
void main()
{
printf ("Welcome to CSE-B");
getch();
}
```

(12) Explain about command line arguments?

* Common line argument is a parameter supplied to the program when it is invoked.
* Common line argument is an important concept in C programming
* It is mostly used when you need to control your program from outside.
* Command line arguments are passed to the main() method.

Syntax:-

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

Here argc counts the no. of arguments on the command line and argv[] is a pointer array

which holds pointer of type char which points to the arguments passed to the program.

Example:- command Line Argument

```
#include <stdio.h>
#include <conio.h>
int main(int argc, char *argv[])
{
    int i;
    if (argc >=2)
    {
        printf(" The arguments supplied are
                                \n");
        for (i=1; i<argc; i++)
        {
            printf("%s\t", argv[i]);
        }
    }
    else
    {
        printf(" argument list is Empty.\n");
    }
    return 0;
}
```

Rembes Remembes that aggv[0] holds the name of the program & argv [1] points to the first command line argument and argv[n] gives the last argument. If no argument is supplied, argc will be 1.