Deepak Sir. UNIT - II P.P.S

ELEMENTS OF 'C' Language.

→ There are various elements of 'C' language as:

1) Preprocessor directives
2) functions
3) Keywords
4) identifiers.
5) Variables
6) Constant

1) Preprocessor Directives:-

→ Preprocessor Directive is a Software which is used to insert the content of other file into Source file.

→ It begin with a Special character #.

→ for Example:
```
#include <stdio.h>
#include <conio.h>
```

2 Function :-

→ A function is a Collection of statement that perform a task.

→ In 'c' language there should be atleast one function i.e main() function.

→ main() func is the entry point in 'c' language.

→ for Example:

```
# include <stdio.h>
main()
{
}
```

3 keywords :-

→ In 'c' language keywords are reserved words. with a sequence of characters having fixed meaning.

→ In 'c' lanuage there are total 32 keywords.

→ for Example: auto, break, char, int, float, double, for, while, goto, if etc.

4 identifiers :-

→ identifiers are used to identify data & other

Objects in the program.

→ identifiers are basically program elements such as Variable, array & function.

→ identifier consist of alphabet, digit or an underscore.

→ keywords cannot be used as identifiers.

→ Examples :- roll_number, marks, name etc.

5 Variables :-

→ Variables is the name which indicate some physical address in the memory where data will be stored in the form of bits of string.

→ The value of variable can be changed at different time of execution.

→ Syntax of variable :- Data type Variable name ;

→ Example :- int V1, V2, V3 ;

6 Constant :- It is a quantity that remains unchang during the execution of program.

→ integer Constant ⇒ 18, -11, 8752

→ Real Constant ⇒ 0.045, 0.0

→ Character Constant ⇒ 'a', '5', '$'

Data Type :-

→ 'C' data types are used to :

* identify the type of a variable when it declared.
* identify the type of the return value of a function.
* identify the type of a parameter expected by a function.

→ In 'C' there are 3 type of data type :

1) Primary (Built-in) data type
2) Derived data type
3) user defined data type

1) Primary data type :-

a) integer :- keyword ⇒ int

Size in bytes ⇒ 2 byte

Range ⇒ -32768 to +32767

Use ⇒ to store integer number.

Example :- int age;

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

b) Character :- keyword ⇒ char

Size ⇒ 1 byte

Range ⇒ −128 to +127

Use ⇒ to store characters.

Example :- char a;

```
#include <stdio.h>
main()
{
char a;
}.
```

c) float :- keyword ⇒ float

Size ⇒ 4 byte

Range ⇒ 3.4E−38 to 3.4E+38

Use ⇒ to store floating numbers.

Example :- float height;

```
#include <stdio.h>
main()
{
float height;
}.
```

d) Double :- keywords ⇒ Double

Size ⇒ 8 byte

Range ⇒ 1.7E-308 to 1.7E+308

Use ⇒ Store big floating point number.

Example :- double a;

```
#Include < Stdio.h>
main()
{
double a;
}.
```

c) Void :- keyword ⇒ Void

Size ⇒ 0

Range ⇒ Valueless.

Use ⇒ —

Example :- ⊗ Void main()

```
# Include < Stdio.h>
Void main()
{
}.
```

2 Derived data type :-

a) Array :- It is a sequence of data item having homogenous value.

Example :- int a[50];

b) pointers :- It is used to access the memory & deal with their address.

Example :- int *ptr;

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

3 User defined data type :- It allows programmer to define their identifier that represent existing data type.

a Structure :- It is a composite structure that contain variable of different data type.

Ex :-
```
struct student
{
char name[25];
int id;
} s1;
```

b UNION :- In this we can store different data type in the same memory location.

For Example:-

```
UNION Student
{
char name[20];
int id;
} S1;
```

- Enumeration :- It enhance the readability of the code.

For Example:- enum weekdays;

STORAGE CLASSES In 'C'

→ Storage class represents the visibility & location of a variable.

→ It tells from what part of code we can access a variable.

→ A storage class is used to describe the following things :

1) The variable scope
2) The location where the variable will be stored.
3) The initialized value of a variable
4) A life time of a variable.
5) who can access the variable?

'C' supports four storage class:

1) automatic
2) register
3) external
4) static

→ The general syntax for specifying the storage class as:

< storage class > < data type > < variable name >

1) Auto storage class:- It is a default storage class for variables declared inside a block. for Example:

auto int x;

→ The scope of 'x' variable is with in a block where 'x' is declared.

→ All local variable belong to auto storage class by default.

→ memory for auto variable is allocated automatic when enter into block & free automatic when exit from block.

→ The auto variable are stored in the primary memory of the computer.

→ If auto variable are not initialized at the time of declaration, then they contain garbage value.

Example:-

```
#include <stdio.h>
main()
{
auto int i = 5;
printf("%d", i);
}
```

2. Register storage class:-

→ Register storage class variable store in CPU Register instead of RAM.

→ one drawback of using register variable is that they cannot be operated using the unary '&' operator because it does not have a memory location.

→ It is declared as:

```
register int x;
```

→ Similar to auto storage class, register variable scop is with in a block where it declared.

→ when regisle variable is not intialized then by default it will take garbage value.

Example:-

```
#include<stdio.h>
main()
{
Register int i=5;
printf("%d", i);
}
```

3. Static Storage class:-

→ As auto is default storage class for all the local variables, static is the default storage class for all global variables.

→ static variable have a lifetime over the entire program.

→ To declare an integer 'x' as static, we write:

```
Static int x = 10;
```

→ If static variable is not intialized, then by default Zero(0) will be assigned.

```
main()                    o/p
{
static int i;      ⇒      0
pf("%d", i);
}
```

→ Static Variables exist even after the block in which they are defined.

Example:-

```
#include <stdio.h>
main()
{
f1();
f1();
}
f1()
{
static int x;
int y=0;
printf("%d", x);
printf("%d", y);
x++; y++;
}
```

output:-

on first func() call:

x=0
y=0

on second f1() call:

x=1
y=0

4. extern Storage class:-

→ This storage class is used when we have global function or variable which are shared between two or more files.

→ Key word extern is used to declare global variable.

→ The variable which are define with key word

extern are called global variable. These variables (7
are accesible throughout the program.

Example:- First file: main.c

```
#include <stdio.h>
# include " original.c"

extern i;

main( )
{
Printf(" value of external integer is = %d", i);
}.
```

Second file: original.c

```
#include <stdio.h>
i= 48;
```

output:- Value of external integer is = 48.

	Storage class	Storage	Default value	Scope	Lifetime.
1.	auto	RAM	Garbage value.	with in the block	with in the block
2.	register	Register	Garbage value	with in the block	with in the block
3.	Static	RAM	Zero	with in the block	Program runtime
4.	extern	RAM	Zero	Entire the file & other file where variable is declared	Program runtime

OPERATORS IN 'C' LANGUAGE

→ An operator is a symbol that tells the compiler to perform specific mathematical or logical function.

→ C language supports a lot of operators to be used in expression.

→ The operator can be categorized according to their priority order in following way:

1) Unary operators
2) Arithmatic operators
3) Bitwise operators
4) Relational operators
5) Logical operators
6) Conditional operators
7) Assignment operators
8) Post increment operators / Decrement operator.

→ A unary operator having highest priority among all other operators & post increment / Decrement operator have least priority.

1) Unary operator :- Unary operator act on a single operand. for Example ⇒ -2, ++2, --5 etc.

→ C language support four unary operator as:

* Unary minus (-)
* Unary increment (++)
* Unary decrement (--)
* Size of ()

→ Unary minus (-) operator is different from binary arithmatic operator.

→ operand preceded by minus sign, unary operator negates its value.

→ for Example: int a, b = 10;
a = -(b);

output: a = -10

→ increment operator is unary operator that increase the value of its operand by 1.

→ Similarly decrement operator decrease the value of an operand by 1.

→ for Example, --x is equivalent to x = x-1;

→ The increment and decrement operator comes in two variants as:

- prefix (++x)
- postfix (x++)

→ In prefix increment / decrement operator, first value of an operand is increased / decreased by 1 then value is assigned. for example:

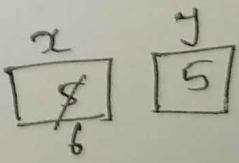
```
main()
{
int x=5, y;
  y = ++x;
printf("%d", y);
}
```

output: y = 6

→ In postfix increment / decrement operator, first value is assigned to other variable then value is increased / decreased by 1 of an operand

for Example:-

```
main()
{
int x=5; y;
  y= x++;
printf("%d", y);
}
```



output: y=5

```
#include <stdio.h>
main()
{
int num = 3;
printf("%d", num);
printf("%d", ++num);
printf("%d", --num);
printf("%d", num);
}
```

output:-

3
4
3
3

→ Sizeof is a unary operator used to calculate the size of data type.

→ When using this operator, the keyword sizeof is followed by a type name, variable or expression.

→ The operator return the size of the variable, data type or expression in bytes.

→ for Example:-

```
main()
{
int a = 10;
printf("%d", sizeof(a));
}
```

output: 2

→ Since 'a' is a integer, it require 2 byte of storage space.

2 Arithmetic operator :-

→ Arithmetic operator can be applied to any integer or floating points number.

→ following table shows the arithmetic operators :

int a = 9, b = 3, result;

Operation	operator	Syntax	Comment	Result
Multiply	*	a*b	result = a*b	27
Divide	/	a/b	result = a/b	3
Addition	+	a+b	result = a+b	12
Subtraction	−	a−b	result = a−b	6
modulus	%	a%b	result = a%b	0

for Example :-

```
#include <stdio.h>
main()
{
int a, b;
int sum, sub, mult, div, mod;
Scanf("%d%d", &a, &b);
sum = a+b;
printf("%d", sum);
sub = a-b;
Printf("%d", sub);
mult = a*b;
Printf("%d", mult);
div = a/b;
Printf("%d", div);
mod = a%b;
Printf("%d", mod);
```

3 Bitwise operator :-

→ bitwise operator are those operators that perform operation at bit level.

→ There operator include:

* bitwise AND (&)
* bitwise OR (|)
* bitwise XOR (^)
* Shift operator (<<, >>)

- 1) bitwise AND :- In this the bits in the first operand is ANDed with the corresponding bit in the Second operand.

→ If both the bits are 1, then the result bit is 1 & 0 otherwise. for Example:

10101010 & 01010101 = 00000000

→ In 'C' program, & operator is used as:

```
main()
{
int a=10, b=20, c=0;
c= a & b;
}
```

2 bitwise OR (|) :- In this if one or both bits are 1, (11) then corresponding but result is 1 & 0 otherwise.

→ for Example :- 10101010 | 01010101 = 11111111

→ In 'c' program (|) operator is used as :

```
int a = 10 , b = 20, c = 0;
c = a | b;
```

3 bitwise XOR (^) :- In this if one of the bits is 1 then result bit is 1, otherwise 0.

→ for Example :- 10101010 ^ 01010101 = 11111111

→ In 'c' program '^' is represent as :

```
int a = 10, b = 20, c = 0;
c = a^b;
```

4 Bitwise NOT :- Bitwise NOT (~) operator sets the bits to 1 if it was initially '0' & set it to '0' if it was intialy 1.

for Example :- ~10101010 = 01010101

Shift operator :- C support two bitwise shift operator

(1) Shift left (<<)

(2) Shift Right (>>)

Syntax :- operand op num

for Example :- x = 0001 1101 , then

x<<1 produce 0011 1010

→ If we apply left shift, every bit in 'x' is shifted to the left by one place. So the MSB of x is lost, the LSB of x is set to 0.

for Example :- x = 0001 1101 then

x<<4 produce 1010 0000.

→ If we apply Right shift, every bit in 'x' is shifted to right by one place.

for Example :- x = 0001 1101 then

x >> 1 produce = 0000 1110

4 Relational operator :- A relational operator also known as comparision operator is an operato that Compares Two values.

→ Relational operators returns true or false.

→ for Example, if x is Less than y then relations operator '<' is used as x < y. This will returns TRUE if 'x' is Less than 'y' otherwise return FALSE

operator	Meaning	Example.
<	Less than	3<5 given 1
>	greater than	7>9 given 0
>=	greater ~~Less~~ than or equal	100>=100 given 1
<=	Less than or equal	50>=100 given 0

→ The relational operator are evaluated from left to right

→ Characters are considered Valid operands since they are represented by numeric value in the Computer system. If we say 'A' < 'B', where 'A' is 65 & 'B' is 66 then result would be '1' as 65 < 66.

Example:-

```
#include <stdio.h>
main()
{
int x=10;
int y=20;
printf("%d<%d=%d", x, y, x<y);
printf("%d==%d=%d", x, y, x==y);
printf("%d!=%d=%d", x, y, x!=y);
printf("%d>%d=%d", x, y, x>y);
printf("%d>=%d=%d", x, y, x>=y);
}
```

Output:-

```
10<20=0
10==20=1
10!=20=1
10>20=0
10>=20=0
```

5. Logical operators:- 'C' language support three logical operators are:

- Logical AND (&&)
- Logical OR (||)
- Logical NOT (!)

→ These operator evaluated from left to right.

Example of logical operators:-

```
#include <stdio.h>
main()
{
int num1 = 30;
int num2 = 40;
if (num1 >= 40 || num2 >= 40)
printf("OR if block gets executed");
if (num1 >= 20 && num2 >= 20)
printf("AND if block executed");
if(!(num1 >= 40))
printf("NOT if block executed");
}.
```

output:- OR if block executed
AND if block executed
NOT if block executed.

logical operator chart:-

operator applied	A	B	output
AND	0	0	0
	0	1	0
	1	0	0
	1	1	1
OR	0	0	0
	0	1	1
	1	0	1
	1	1	1
NOT	0	-	1

Conditional operator :-

→ The conditional operator or the ternary (? :) operator is just like an if-else statement.

→ The syntax of conditional operator is :

```
exp1 ? exp2 : exp3
```

→ exp1 evaluated first. if it is true, then exp2 is evaluated & becomes the result of the expression, otherwise exp3 is evaluated.

→ for example :-

```
large = (a>b) ? a : b
```

→ above expression is used to find largest of two given numbers. first (a>b) is evaluated, if 'a' is greater than b then large = a, else large = b.

Example :- find largest of two number using ternary operator.

```
#include <stdio.h>
main()
{ int a, b, large;
  scanf("%d%d%d", &a, &b, &large);
  large = a>b ? a : b;
  printf("%d", large);
}
```

Assignment operator :-

→ This operator is responsible for assigning value to the variables.

→ The equal sign (=) is the fundamental assignment operator.

→ When equal sign (=) is encountered in an Expression, the compiler process the statment on the right side of the sign and assign the result to the variable on the left side.

→ for example: 1) int x;
x = 10;

2) int x = 2, y = 3, sum = 0;
sum = x + y;
then sum = 5.

→ Assignment operator has right to left associativity

∴ Expression a = b = c = 10; is evaluated a

(a = (b = (c = 10)));

Example of Assignment operator:-

```
#include <stdio.h>
main()
{
int a, b, c;
scanf("%d %d", &a, &b);
c = a+b;
printf("%d", c);
}
```

Output:- a = 5
b = 6
c = 5+6 = 11

OPERATOR PRECEDENCE AND ASSOCIATIVITY

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

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left

Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left

EXAMPLE

```
include <stdio.h>

main() {

   int a = 20;
   int b = 10;
   int c = 15;
   int d = 5;
   int e;

   e = (a + b) * c / d;       // ( 30 * 15 ) / 5
   printf("Value of (a + b) * c / d is : %d\n",  e );

   e = ((a + b) * c) / d;     // (30 * 15 ) / 5
   printf("Value of ((a + b) * c) / d is  : %d\n" ,  e );

   e = (a + b) * (c / d);    // (30) * (15/5)
   printf("Value of (a + b) * (c / d) is  : %d\n",  e );

   e = a + (b * c) / d;      //  20 + (150/5)
   printf("Value of a + (b * c) / d is  : %d\n" ,  e );

   return 0;
```

out put:

```
Value of (a + b) * c / d is : 90
Value of ((a + b) * c) / d is  : 90
Value of (a + b) * (c / d) is  : 90
Value of a + (b * c) / d is  : 50
```

formatted & unformatted I/O functions

→ C language provide various console input/output functions that allow us to -

- Read the input from the keyword by the user.
- Display the output to the user at console.

→ There are two types of console I/O functions:

1) formatted input/output function
2) unformatted input/output function.

1) **formatted I/O function**:- C language used two formatted I/O function as:

1) Printf ( )
2) Scanf ( )

1) **Printf( )**:- It is used to display information required by the user and also print the values of the variables.

→ printf function takes data value, convert them to text stream using formatting functions.

→ Syntax:- printf ("control string", variable list);

→ The control characters include \n, \t, \r, \a etc.

→ The formate can be %s, %d, %c, %f to print or read string, integer, character or float.

Example:-

```
Printf ("%d%c%f", 12, 'a', 2.3);
```

Output:- 12a2.3

2 Scanf():- It is used to read formatted data from keyboard.

Syntax:- Scanf ("control string", &arg1, &arg2 --&argn);

→ Control string may contain formate like %d, %f, %c, %s to read integer, float, character, string data from the keyboard.

example:-

```
float salary;
Scanf ("%f", &Salary);
```

Example of Printf / scanf :-

```
#include <stdio.h>
# include <conio.h>
 main()
{
int a;
clrscr();
Printf("Enter value of a");
Scanf ("%d", &a);
Printf ("%d", a);
}
```

2 Unformatted I/O functions:-

Types of unformatted I/O functions

1) getchar()
2) putchar()
3) getch()
4) putch()
5) gets()
6) puts()

1) getchar() :- This function read a character-type data from standard input

→ It reads one character at a time till the user presses the enter key.

Syntax:- Variable_name = getchar();

Example:-
```
char c;
c = getchar();
```

Program:-
```
#include <stdio.h>
void main()
{
char c;
Printf("Enter a character");
c = getchar();
Printf("%c", c);
}
```

output:-
Enter a character K
c=K

2 Putchar(); This function prints one character on the screen at a time which is ready by standard input.

Syntax:- putchar(variable_name);

Example:-
```
char c = 'c';
putchar(c);
```

output:
Enter character: r
r

Program:-
```
#include <stdio.h>
main()
{
char ch;
Printf("Enter Character:");
Scanf("%c", ch);
Putchar(ch);
}
```

3 getch() & getche();-

~~→ These functions read only~~

→ getch() is a non-standard function & it is present in conio.h header file.

→ It reads single character from keyboard & it doesnot use any buffer, so the entered character is immediately returned without waiting for the enter key.

Syntax:- int getch();

Example:-
```
#include <stdio.h>
#include <conio.h>
int main()
{
printf("%c", getch());
return 0;
}
```

→ 4) getche():- like getch(), this is also a non-standard function present in conio.h.

→ It reads a single character from the keyboard and display immediately on output screen without waiting for the enter key.

Syntax:- int getche(void);

Example:-
```
#include <stdio.h>
#include <conio.h>
main()
{
printf("%c", getche());
return 0;
}
```

:) gets() :- This function enable the user to enter some character followed by the enter key.

→ All the character entered by the user get stored in character array.

→ The null character is added to the array to make it a string.

→ gets() allow user to enter the space-separated strings & it return the string entered by user.

Syntax :- gets (char_array_Variable);

Example :-

```
#include <stdio.h>
main()
{
char s[30];
printf("Enter the string");
gets(s);
printf("you enter o/p s", s);
}
```

6) Puts(); - This function is very similar to Printf() function.

→ The Puts() function is used to print the string on the console which is previously read by using gets() or Scanf() function.

→ The puts() function returns an integer value representing the number of character being printed on the console.

Sytax:- puts(char Array_Variable / string);

Example:-

```
#include <stdio.h>
#include <string.h>
main()
{
char name[50];
printf("Enter your name");
gets(name);
printf("your name is:");
puts(name);
return 0;
}
```

gets VS Puts

gets

1. A 'C' library function that read a line from stdin & store it in the pointed string

2. Declaration is:
char *gets

3. Help to scan a line of text from a standard input device

4. Return string on success

5.
```
#include<stdio.h>
main()
{
char s[50];
gets(s);
Printf("%c",s);
}
```

Puts

1. A 'C' library function that writes a string to stdout or standard output

2. Declaration is:
int puts

3. Help to display a string on a standard output device.

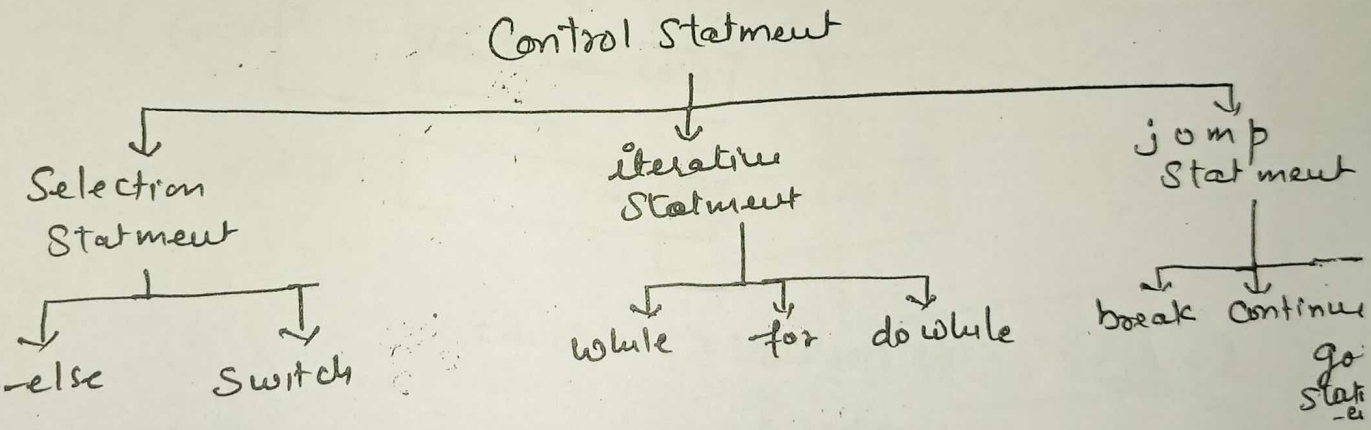
4. Return non-negative value if successfull.

5.
```
#include<stdio.h>
#include<conio.h>
main()
{
char s[50];
gets(s);
Printf("output is");
puts(s);
}
```

Control Statments in C

→ using decision Control statement we can control the flow of program in a way so that it execute certain statements based on the outcome of a condition (i.e true or false)

→ In C Language we have different type of Control statement as:



If-else In 'C'

→ It can be used to select a block for execution based on a condition.

→ If-else is classified into different types as:

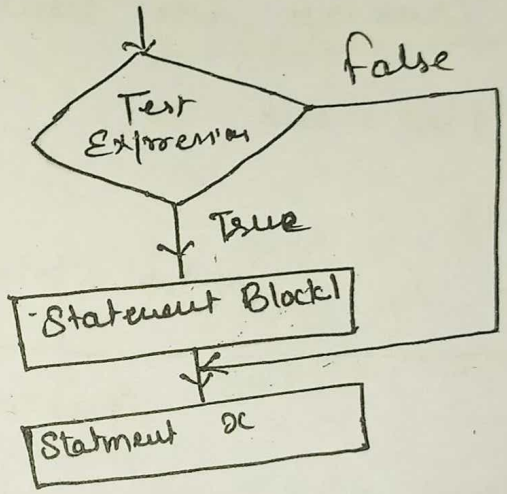
- If statment
- if-else statment
- nested if-else

- **If Statment :-** It is the simplest form of decision control statment

→ **Syntax :-**

```
if ( test expression)
{
  statment 1;
  - - - -
  - - - -
}
statment x;
```

flow of if statment :-



Example :-

```
#include<stdio.h>
main()
{
if (4>3)
{
printf ("Hello");
}
printf ("world");
}
```

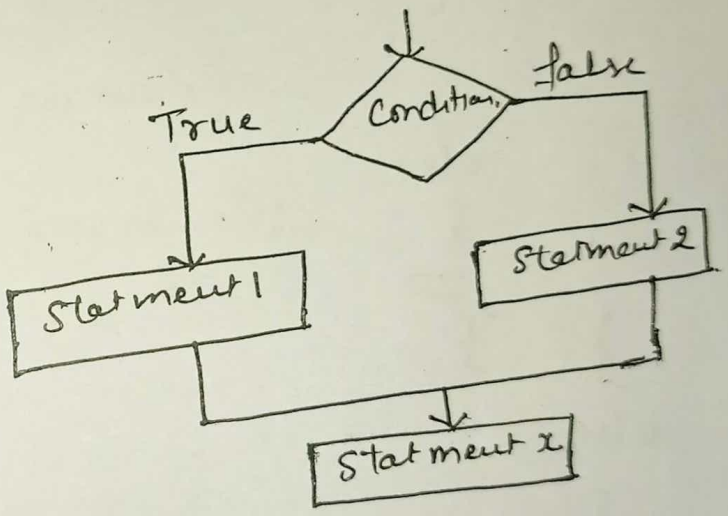
- **If-else statement :-** According to if-else, first the test expression is evaluated. If the expression is true, then if block statments executes & else block statments are skipped.

→ otherwise if the expression is false then else block statments execute & if block statments are skipped.

Syntax:-

```
if ( Condition )
{
 Statment 1 ;
}
else
{
Statment 2 ;
}.
Statment x;
```

flow control:



Example:-

```
#include <stdio.h>
main()
{
if ( 3>4)
{
Printf ("Hello");
}
else
{
Printf ("world");
}
```

- nested if-else :- Sometimes we need to test more than one condition, then we use nested if-else statement.

Syntax:

```
if (cond1)
{
    if (cond2)
    {
    ==
    }
    else
    {
    ==
    }
}
else
{
==
}
```

```
if (cond1)
{
==
}
else
{
    if (cond2)
    {
    ==
    }
    else
    {
    ==
    }
}
```

Example:- find whether the given number is zero or negative or positive.

```
#include <stdio.h>
main()
{
int a;
scanf("%d", &a);
if (a==0)
printf("Number is zero");
else if (a>0)
printf("number is positive");
else
printf("number is negative");
}
```

Switch Statment :-

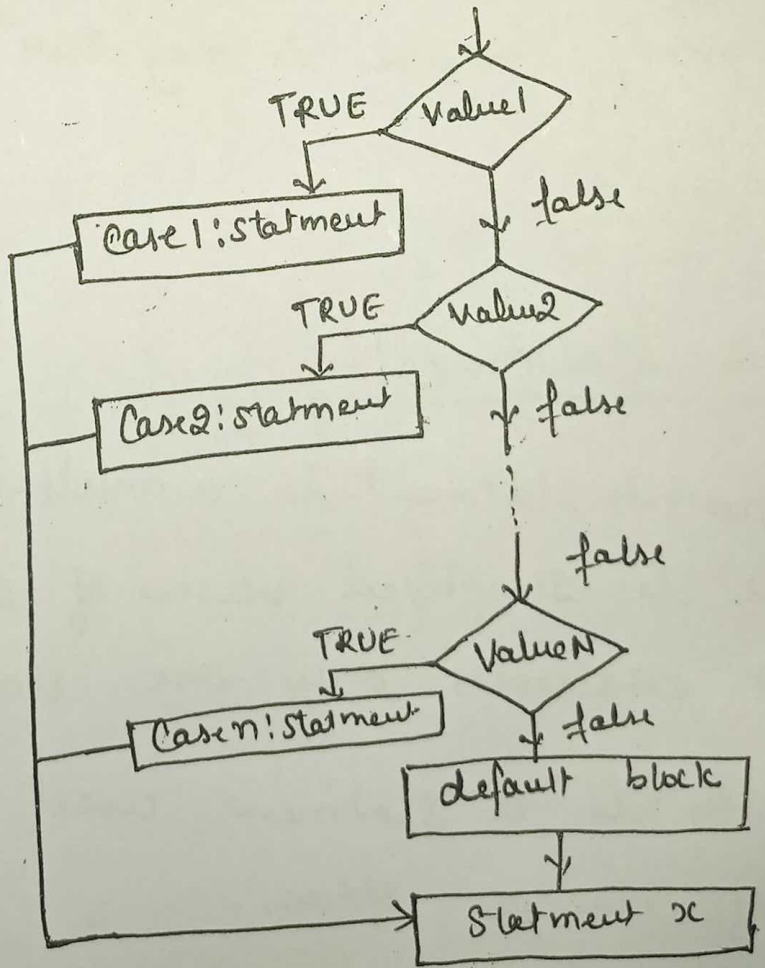
→ A switch statment is a multi-way decision statme that is simplified version of an if-else block that evaluates only one variable.

→ Switch is a statment used to select one out of multiple options.

Syntax:

```
switch (expression)
{
Case label1 : ------
 break;
Case label2 : ------
break;
Case label3 : ---
break;
-----
-----
default : ------
}
Statment x;
```

flow control :-



Example:-

```
#include <stdio.h>
main()
{
int a=1;
Switch (a)
{
Case 1: Printf (" Hello ");
    break;
Case 2: printf (" world ");
    break;
default: printf (" wrong choice ");
}
}
```

→ Generally a break statment is used at end of each case block.

→ If break is not provided, the system automatically enter into the next case block & executes the statments there.

Limitations of switch Statment:-

1) Good for equality comparision but not for range comparision
2) Works good for int data type only not for others.
3) Works good with constants but not with Variables.

Iterative statment :-

+ iterative statments are used to repeat the execution of a list of statments, depending on the value of an integer expression.

→ C Language support three type of iterative statments also known as looping statments

Ans:

a) while loop
b) do-while loop
c) for loop.

a) **while loop** :- while loop provids the mechanism to repeat one or more statment while a particular Condition is true.

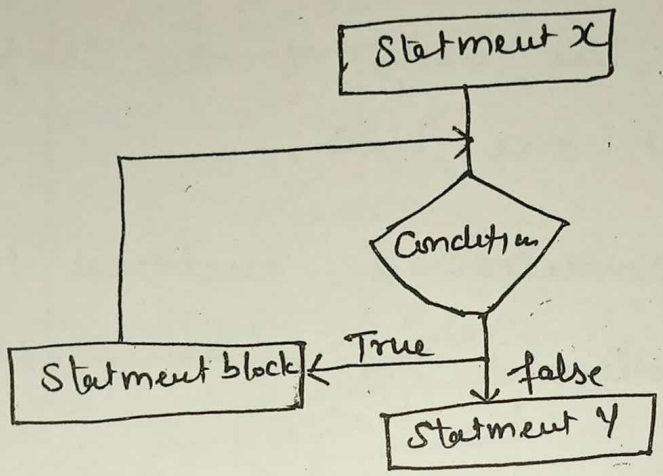
Syntax :-

```
Statment x;
while ( Condition)
{
Statment block;
}
Statment Y;
```

→ In while loop, the condition is tested, if the condition is true then statment under

while loop block is executed other wise Control will jump to Statment y.

flow chart of while loop:-



Example:- print first 10 numbers using while loop.

```
#include<stdio.h>
main()
{
int i=0;
while(i<=10)
{
printf("%d",i);
i++;
}
getche();
}
```

b) Do-while loop :-

→ It is similar to while loop. The only difference is that in a do-while loop, the condition is tested at the end of the loop.

→ In do-while loop, the body of the loops gets executed at least one time.

→ In do-while, test condition is enclosed in paranthesis and followed by a semicolon.

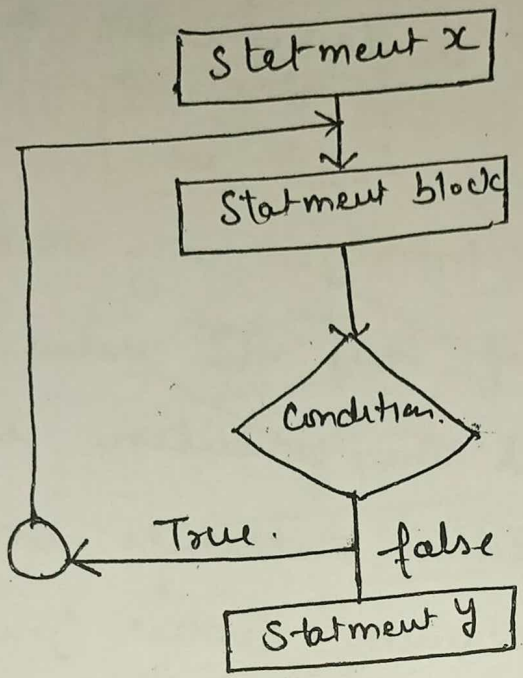
Syntax:

```
Statment x;
do
{
Statment block;
} while (Condition);
Statment y;
```

→ The major disadvantage of using do-while loop is that it always executes at least once, even if the user enter invalid data

→ However, do-while loops are widely used to print a list of option for a menu-driven programs.

flowchart of do-while loop :-



Example :- print first '10' number using do-while.

```
#include < stdio.h >
main()
{
int i=1;
do
{
printf("%d", i);
i++;
} while (i ≤ 10);
getche);
}
```

C for loop :-

→ like while & do-while loop, the for loop provides a mechanism to repeat a task untill a particular condition is true.

→ In for loop variable is intialized only once.

→ with every iteration of loop the value of a variable is updated & the condition is checked.

→ if condition is true then statments under for loop will executed, otherwise control jump to the immediate statment following the for loop.

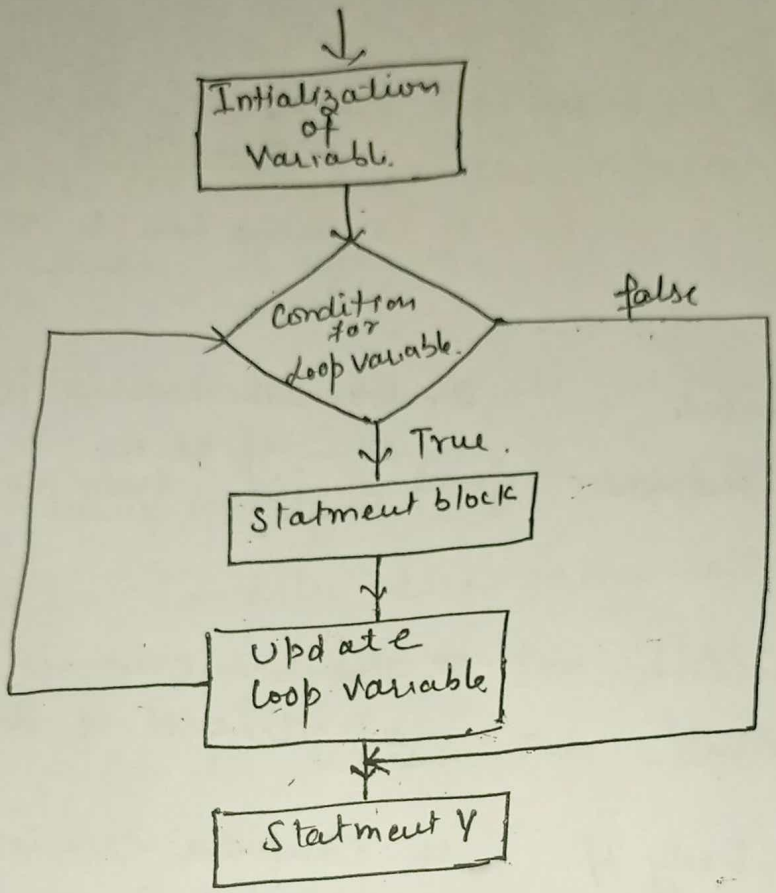
→ Every section of the for loop is seprated from the other with a semicolon ; .

→ for loop, is used to execute a single or a group of statment under limited time.

Syntax :-

```
for (intialization ; Condition ; increment/Decrement)
{
statment block ;
}
statment y ;
```

Flow chart of for loop:-



Example: print first '10' number using for loop.

```
#include<stdio.h>
main()
{
int i;
for(i=1; i<=10; i++)
{
printf("%d", i);
}
getche();
}
```

Difference between while & do-while loop.

while	do-while
• Condition check at the top	1. Condition check at the bottom.
2. no necessity of brackets if there is single statment in body	2. Brackets are compulsory even if there is a single statment.
3. There is no semicolon at the end of while	3. The semicolon is must at the end of do-while.
4. Computer execute body if and only if condition true	4. Computer executes the body at least once if the condition is false.
5. This is used when the condition is more important	5. This is used when the process is important
6. This is also called entry controlled loop	6. This is also called exit controlled loop.

7. while (n<10)
{
printf("%d", n);
}
	7. do
	{
	printf("%d", n);
	} while (n<=10);

JUMP STATMENTS

1) Break :- break statment is used to terminate the execution of the nearest enclosing loop in which it appears.

→ The break statment is widely used with for loop, while loop, do-while loop.

→ when compiler encounters a break statment the control passes to the statment that follow the loop in which the break statment appears.

→ Syntax :- break ;

→ In switch statment of the break statment is missing then every case from the matched case label till the end of the switch, including the default is executed.

→ When a break statment encountered inside a loop the loop is immediately terminated & program control is passed to the next statment following the loop.

1)

Example:-

```
#include<stdio.h>
main()
{
int i=0;
while(i<=10)
 {
    if(i==5)
    break;
    printf("%d", i);
    i++;
 }
getche();
}.
```

output:

0, 1, 2, 3, 4

2)

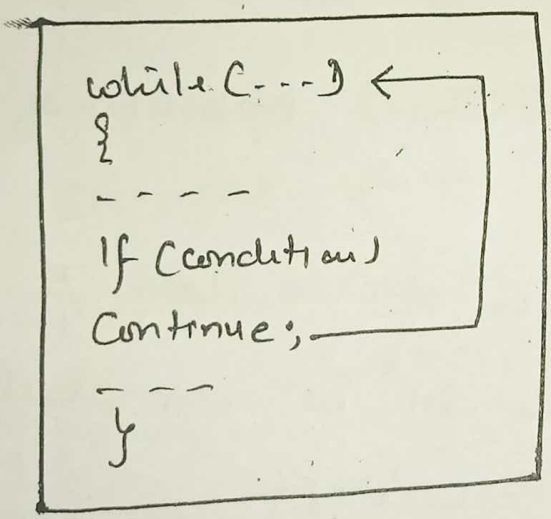
```
#includ <stdio.h>
main()
{
int i ;
Switch(i)
{
Case 1: Printf("Hellow");
 break;
Case 2: Printf("world");
breake;
default: Printf("Hi");
getche();
}.
```

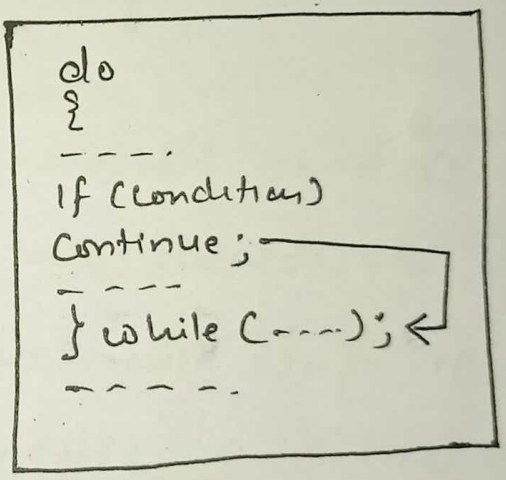
2. Continue :-

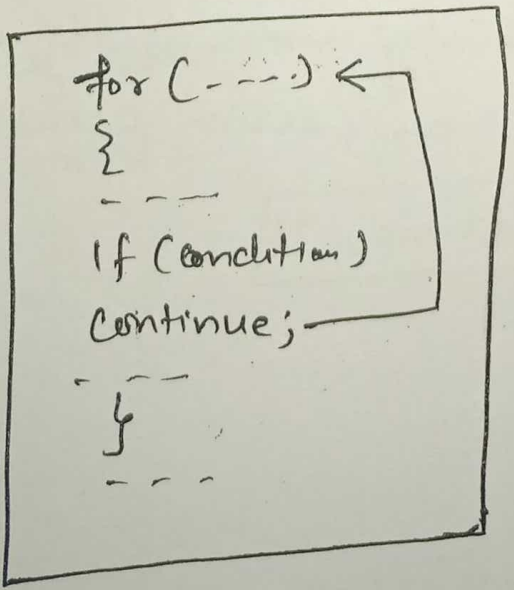
→ like break statment, continue statment can only appear in the body of loop.

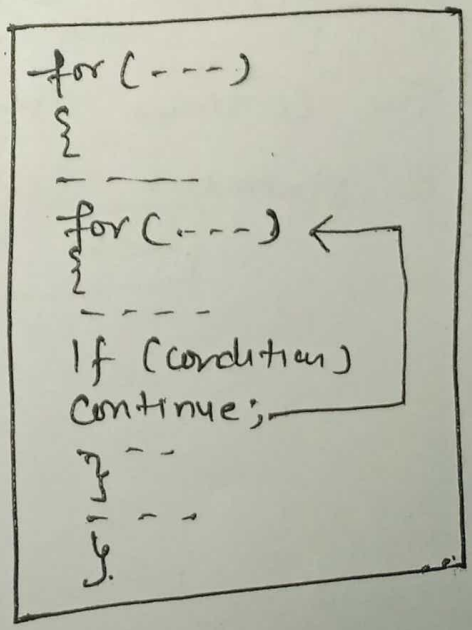
→ When the compiler encounters a continue statment then the rest of the statments in the loop are skipped and the control is unconditionaly transfe to the loop-continuation portion of the nearest enclosing loop.

→ Syntax :- continue ;









Example:-

```
#include<stdio.h>
main()
{
int i;
for(i=0; i<=10; i++)
{
        if(i==5)
        continue;
        printf("%d", i);
        i++;
}
getche();
}.
```

→ Continue Statment is somewhere opposite of the break statment.

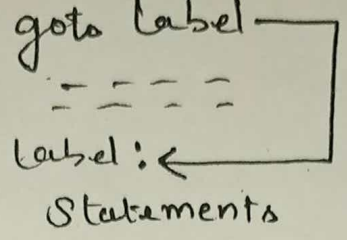
→ It force the next iteration of the loop to take place, skipping any code in between itself and the test condition of the loop.

→ The continue statment is usually used to restart a statment sequence when an error occurs.

3 goto Statment :-

→ The goto statment is used to transfer control to a specified label.

→ Syntax :-



→ Label is an identifier that specifies the place where the branch is to be made.

→ Label can be any valid variable name that is followed by a colon (:).

→ The label is placed immediately before the statment where the control has to be transf

→ The label can be placed anywhere in the progra either before or after the goto statment.

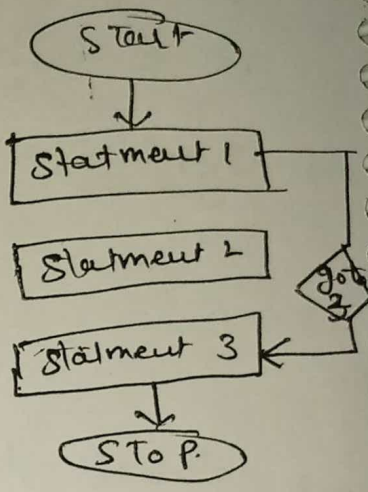
→ The goto statment is often combined with the if statment to cause a conditional transfe of control.

If Condition then goto Label

Example:-

```
#include <stdio.h>
main()
{
int i=4, j=3;
if (i>j)
goto there;
else
printf("Hello");
there:
printf("world");
getch();
}
```

Flow chart.



Disadvantage:-

1) goto statment can make code harder to understand.