PROGRAMMING IN C

NOTES

By

Prof. Prabhu Kichadi

9880437187

UNIT – II

Operators & Expressions: Arithmetic operators; Relational operators; Logical operators; Assignment operators; Increment & Decrement operators; Bitwise operators; Conditional Operators; Special operators; Operator Precedence and Associatively; Evaluation of arithmetic expressions; Type conversion. Control Structures: Decision making Statements - Simple if, if else, nested if else, else if ladder, Switch Case, goto, break & continue statements; Looping. Statements - Entry controlled and exit controlled statements, while, do-while, for loops, Nested loops.

Operators & Expressions:

Expressions: ***"Expression is a valid combination of Operators, operands, and/or variables, constants which outputs a useful result".***

```
Ex :
a= 10;
c= a + b;
a = a + 1;
res = 4 * 5;
a= b=c=10;
```

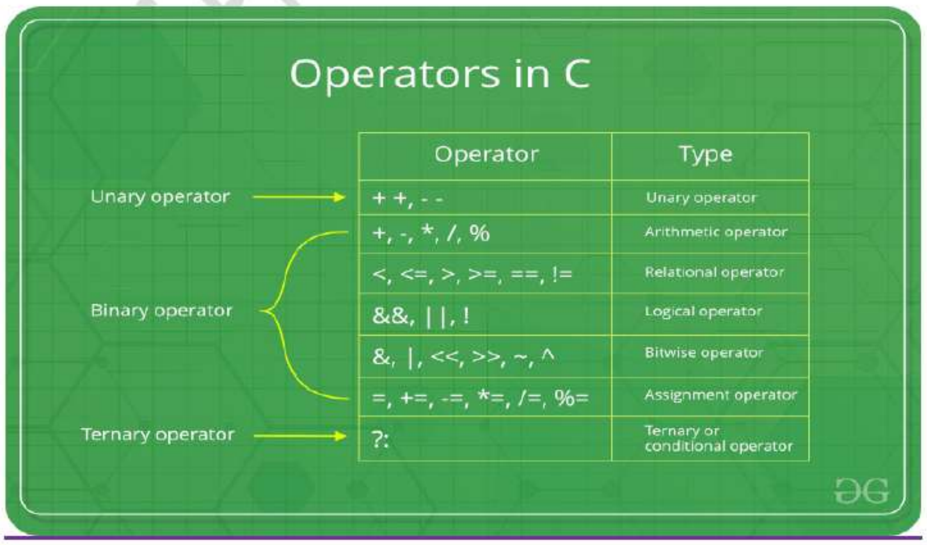
Operators: ***"Operator is a symbol that performs some action/operation, it can be mathematical or logical and produces a useful result".***

Operators are used to manipulate data and variables in C, these are part of expressions.

Ex :
+ -> Addition operator,->Addition operation
2+4, 6+x , sum = a + b, a = 3.142 * r * r etc.

Types of operators

1. **Assignment Operator: =**
2. **Arithmetic Operators: +,-,*,/,%**
3. **Relational operators: <,<=,>,>=, ==, !=**
4. **Increment and decrement operators: ++,--**
5. **Logical operators: &&, ||, !**
6. **Conditional operators: ?:**
7. **Bitwise operators: << ,>>, &,|,~**
8. **Special operators: ,(comma) .(dot) sizeof( ), &, ***



Operators in C

	Operator	Type
Unary operator	+ +, - -	Unary operator
Binary operator	+, -, *, /, %	Arithmetic operator
	<, <=, >, >=, ==, !=	Relational operator
	&&, | |, !	Logical operator
	&, |, <<, >>, ~, ^	Bitwise operator
	=, +=, -=, *=, /=, %=	Assignment operator
Ternary operator	?:	Ternary or conditional operator

1. Assignment Operator =

- It is used to store a value/data from right hand side to left hand variable.
- Assignment operator is used to assign the result of an expression to a variable.
- It is also called as **Initialization** operator.
- Assignment happens from right to left.
- Left hand side it must be a variable.

Syntax: **variable = expression/data;**

Ex1:

```
int a;
a = 100;
printf("Value of a = %d",a);
```

Output

```
Value of a = 100
```

2. Arithmetic operators: +, -, *, / ,%

- C supports arithmetic operators like **+** (addition), - (subtraction), * (Multiplication), / (division), % (modulo).
- Evaluation happens from right to left.

Ex1 :

```
int x, y, res;
x = 12;
y = 24;
res = x + y;
printf("\n Res = %d",res);
```

Output:

```
Res = 36
```

Note: Mod operator(%) always gives reminder, and it is applied only on integral types.

Ex1 :

```
int x, y, res;
x = 12;
y = 2;
res = x % y;
printf("\n Res = %d",res);
```

Output:

```
Res = 0
```

3. Relational operators: <, <=, >, >=, ==, !=

"To compare contents of two or more items on their relation, to take certain decisions on".

- Relational operators are used to compare values/variables values/expressions.
- **The result of relational operation is always coming in 0(False) or 1(True) value.**
- In C true values **means non zero numbers are true** values, while **zero value is false** value.
- Relational operators are,
  Less Than(<), Less Than or Equals to(<=), Greater Than(>),
  Greater Than or Equals to(>=), Equals to(= =), Not equals to (!=).

Ex 1:
```
int x, y, res;
 x = 22;
 y = 23;
 res = x > y;
printf("\n res = %d",res);
```
Output
```
res = 0
```

Ex 2:
```
int x, y, res;
 x = 22;
 y = 22;
 res = x == y;
printf("\n res = %d",res);
```
Output
```
res = 1
```

Ex 3:
```
int x, y, res;
 x = 22;
 y = 22;
 res = x <= y;
printf("\n res = %d",res);
```
Output
```
res = 1
```
----

4. Increment and decrement operators: ++,--

- C allows useful operators increment & decrement operators.
- Only applicable for variables.
- Increment (++) operator increases content of variable by 1.
- Decrement (--) operator decreases content of variable by 1.

Syntax:

```
var++;  or  ++var;

var--;  or  --var;
```

4.1 Increment Operator (++)

post Increment(var++): It is applied to a variable, if it is suffixed to a variable then it is post increment. Here the value of the variable is assigned first then it is incremented by 1.

```
Ex1:    int a,b;
        b = 10;
        a = b++;
printf("a = %d   b = %d",a,b);
Output:
        a = 10  b = 11
```

pre-Increment(++var) : It is applied to a variable, if it is prefixed to a variable then it is pre increment. Here the value of the variable is first increased by 1 then it is assigned.

```
Ex1: int a, b;
        b = 10;
     a = ++b;
     printf("a = %d   b = %d",a,b);
Output:
     a = 11  b = 11
```

4.2 Decrement operator(--):

post decrement(var--)

It is applied to a variable, if it is suffixed to a variable then it is post decrement. Here the value of the variable is assigned first then it is decremented by 1.

```
Ex1:    int a,b;
        b = 10;
       a = b--;
      printf("a = %d   b = %d",a,b);
Output:
      a = 10  b = 9
```

pre decrement(--var)

It is applied to a variable, if it is prefixed to a variable then it is pre decrement. Here the value of the variable is first decremented by 1 then value is assigned.

```
Ex:     int a, b;
         b = 10;
        a = --b;
printf("a = %d    b = %d",a,b);
```

Output:

```
a = 9  b = 9
```

5. Logical Operators: &&, ||, !

If we want to **combine/test two or more expressions**/conditions then we can use logical operators to take decisions.

Ex: For voting

1. Age must be greater than or equals to 18.

&&

2. Person should have a voter ID.

5.1 Logical AND &&

Logical AND result will be true only when **both conditions are true**.
The net result of Logical AND (&&) operator depends on the truth table of AND

Expression1	Expression2	Result (&&)
0	0	0
0	1(non-zero)	0
1(non-zero)	0	0
1(non-zero)	1(non-zero)	1

Ex1:

```
int x, y, res;
x = 100;
y = 20;
res = x>10 && y<x;
printf("\nRes = %d",res);
```

Output:

```
Res = 1
```

Ex2:

```
int x, y, res;
x = 10;
y = 20;
res = x<10 && y<88;
printf("\nRes = %d",res);
```

Ouput:

```
Res = 0
```

5.2 Logical OR ||

Logical OR result will be true if any one condition is true, in remaining cases result will be false.
The net result of Logical OR(||) operator depends on the truth table of OR

Expression1	Expression2	Result (||)
0	0	0
0	1(non-zero)	1
1(non-zero)	0	1
1(non-zero)	1(non-zero)	1

Ex1:

```
int x, y, res;
x = 10;
y = 20;
res = x>10 || y<=20;
printf("\nRes = %d",res);
```

Output:

```
Res = 1
```

5.3 Logical NOT !

Logical NOT(!) operators invert(reverse) the value from nonzero to zero, and zero to nonzero for the expression.

Expression1	Result(!)
0	1
1	0

Ex1:

```
int x = 100;
```

```
        x = !x;
        printf("\n x = %d",x);
```

Output:

```
        Res = 0
```

Ex2:

```
        int x = 0;
        x = !x;
        printf("\n x = %d",x);
```

Output:

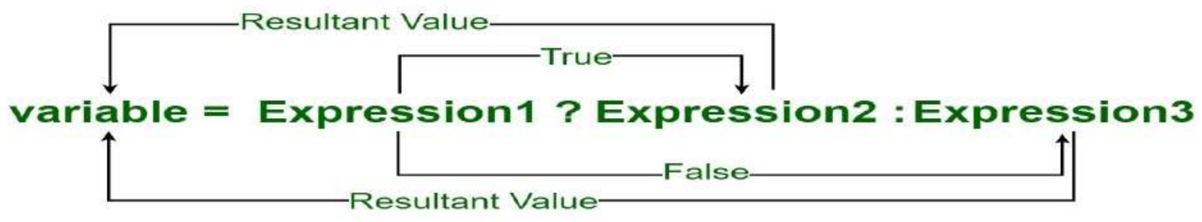
```
        Res = 1
```

6. Conditional Operator/Ternary Operator ? :

It is also called as ternary operator; it requires three operands for evaluation.

If the expression1 result is true, then Expression2 will be assigned to variable
If the Expression1 result is false, then Expression3 will be assigned to variable.

Syntax:



Ex1: C program for maximum of two numbers using conditional operator(IMP)

```
#include<stdio.h>
int main()
{
   int a,b,max;

   a = 100;
   b = 20;

   max = (a>b)? a : b;

printf("\nMaximum = %d",max);
   return 0;
}
```

Output:

```
Maximum = 100
```

Ex2: C Program for maximum of three numbers using ternary operator(IMP)

```
#include<stdio.h>
int main()
{
   int a,b,c,max;

   a = 100;
   b = 200;
   c = 444;

   max = (a>b)?((a>c)?a:c):((b>c)?b:c);

   printf("\nMaximum = %d",max);
  return 0;
}
```

Output:

```
Maximum = 444
```

Ex3: C program to check whether a given year is leap year or not using conditional operator. (IMP)

```
#include<stdio.h>
int main()
{
   int year;
   year = 2000;
   (year%4==0 && year%100!=0)? printf("\nLeap Year") :
      (year%400 == 0)?printf("\nLeap Year") : printf("\nNot Leap Year");

   return 0;
}
```

Output:

```
Leap Year
```

7. Bitwise Operators <<, >>, &, |, , ^, ~

Bitwise operator works on bit level of data.
Internally data is converted into binary form and bitwise operator operates on each bit of data, Produces result in converted form.

7.1 Bitwise AND (&) : Logical AND works on each bit of the data.

Ex 1:

```
        int  a,  b,  res;
        a = 10;                                    10 =>  1   0   1   0
        b = 6;                                     20 =>  0   1   1   0
        res  =  a & b;                             --------------------------
        printf("\nResult = %d",res);                      0   0   1   0  => 2
```

Output

```
        Result = 2
```

7.2 Bitwise OR (|) : Logical OR works on each bit of the data.

Ex 1:

```
        int  a,  b,  res;
        a = 10;                                    10 =>  1   0   1   0
        b = 6;                                     20 =>  0   1   1   0
        res  =  a & b;                             --------------------------
        printf("\nResult = %d",res);                      1   1   1   0  => 14
```

Output

```
        Result = 14
```

7.3 Bitwise Left Shift <<

Bitwise shift operator moves bits to left at specific positions.

Ex :

```
        int  a,  res;
        a  = 10;                                   10 =>   1   0   1   0
        res  =  a<<2;                          1   0   1   0   0   0
        printf("\nResult = %d",res);
```

Output

```
        Result = 40
```

7.4 Bitwise Right Shift >>

Bitwise shift operator moves bits to right at specific positions.

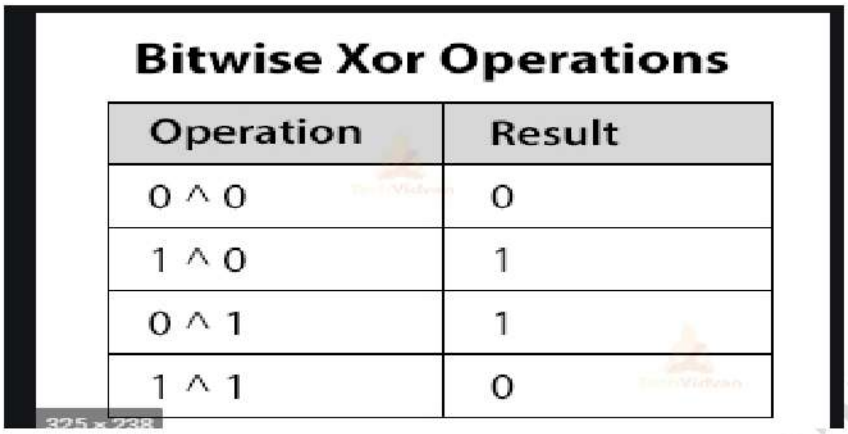
Ex :

```
        int  a,  res;
        a  = 10;                                   10 =>   1   0   1   0
        res  =  a>>2;                                              1   0   1  0
        printf("\nResult = %d",res);
```

Output

```
        Result = 2
```

7.5 Bitwise X-OR (^)



Bitwise Xor Operations

Operation	Result
0 ^ 0	0
1 ^ 0	1
0 ^ 1	1
1 ^ 1	0

Ex:

```
int x, y, res;
x = 4; y=6;
res = x ^ y;
printf("\n res = %d",res);
```

Output: res = 2

```
x -> 4 ->  0 1 0 0
y -> 6 ->  0 1 1 0        Logical XOR applies on every bit
----------------------
 0 0 1 0  ->  2
```

7.6 Bitwise NOT (~)

- It reverses all the bits, it produces 1's complement of data.

Ex :

```
int x;
x = 3;
x = ~x;
printf("\nx = %d",x);
```

Output:

```
 x = 0
```

8. Special Operators:

8.1- Comma(,) Operator
8.2 - sizeof(datatype) operator
8.3 - & Operator/Address of operator
8.4 - * Pointer Operator
8.5 - dot(.) Operator

8.1 Comma (,) Operator

It combines related multiple expressions/statements of similar types in a single line.

Ex 1:

```
int x;
int y;
int z;
```

Using comma,

```
int x,y,z;
```

Ex 2:

```
int x;
int y;
int z;
x = 10;
y = 20;
z = 30;
```

Using comma,

```
int x,y,z;
x=10, y=20,z=30;
```

8.2 sizeof(datatype) Operator

It returns the size of data type or variable in terms of number of bytes.

Syntax:

sizeof(datatype/variable);

Ex 1:

```
printf("int size = %d bytes", sizeof(int));
```

Ex 2:

```
float x;
int res;
res = sizeof(x);
printf("\nres = %d",res);
```

Output : res = 4

8.3 & Address of Operator/Reference Operator

This is used to get the address of the variable. Example : &a will give address of a.

- Generally used to scanf( ) statement to store the data/input to a variable. To manipulate pointers this operator is used as Direction operator.

Ex 1:

```
int  x;
```

```
float   y;

printf("\nEnter x & y Value : ");
scanf("%d%f", &x, &y);
```

Ex 2:

```
int x = 10;
float  y = 44.44;

printf("\nAddress of x = %d\nAddress of y = %d", &x, &y);
```

Output

```
Address of x = 21465623
Address of y = 33434354
```

8.3 * Pointer Operator

This Operator is used in pointer declarations, using as Dereference operator. To access the content of a variable hold by pointer we use * Pointer operator.

Ex 1:

```
int   x = 10;      //x is normal variable
int  *p;     //p is now pinter variable

p = &x;

printf("\n*p = %d", *p);
```

Output

```
*p = 10
```

Operators Hierarchy, Precedence and Associativity

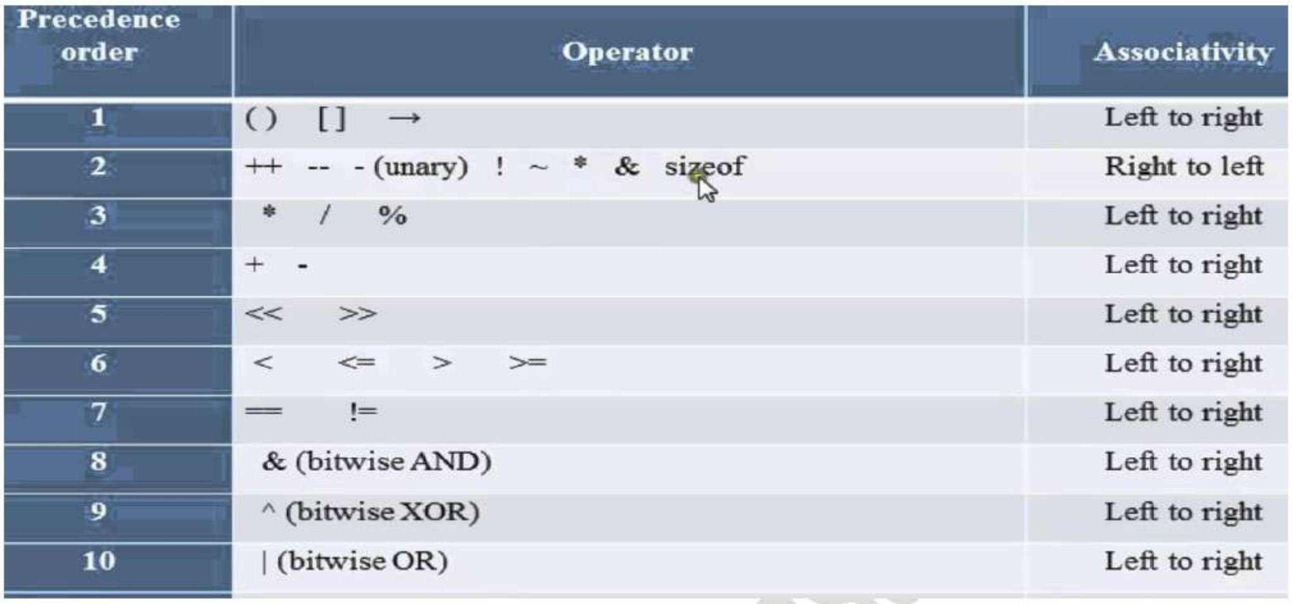
Operator Precedence

If an expression contains more than one operator, precedence determines which operator evaluates first and next.
It is determined by precedence of operators.
Priority/Precedence is from Highest to Lowest.

Operator Associativity

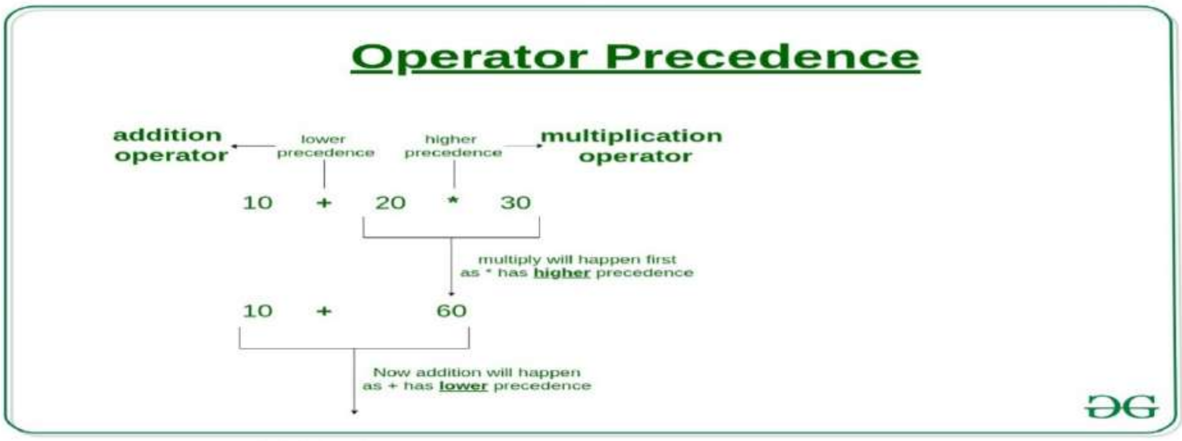
Operator Associativity indicates the direction/order in which operators are evaluated when same priority operators are present.

Operator Precedence and Associativity Table

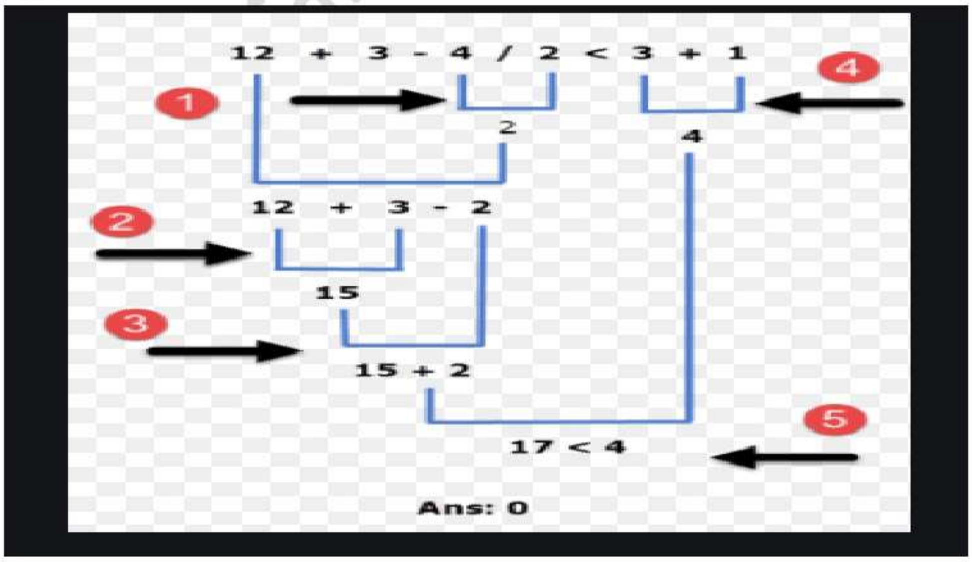


Precedence order	Operator	Associativity
1	( ) [ ] →	Left to right
2	++ -- - (unary) ! ~ * & sizeof	Right to left
3	* / %	Left to right
4	+ -	Left to right
5	<< >>	Left to right
6	< <= > >=	Left to right
7	== !=	Left to right
8	& (bitwise AND)	Left to right
9	^ (bitwise XOR)	Left to right
10	| (bitwise OR)	Left to right

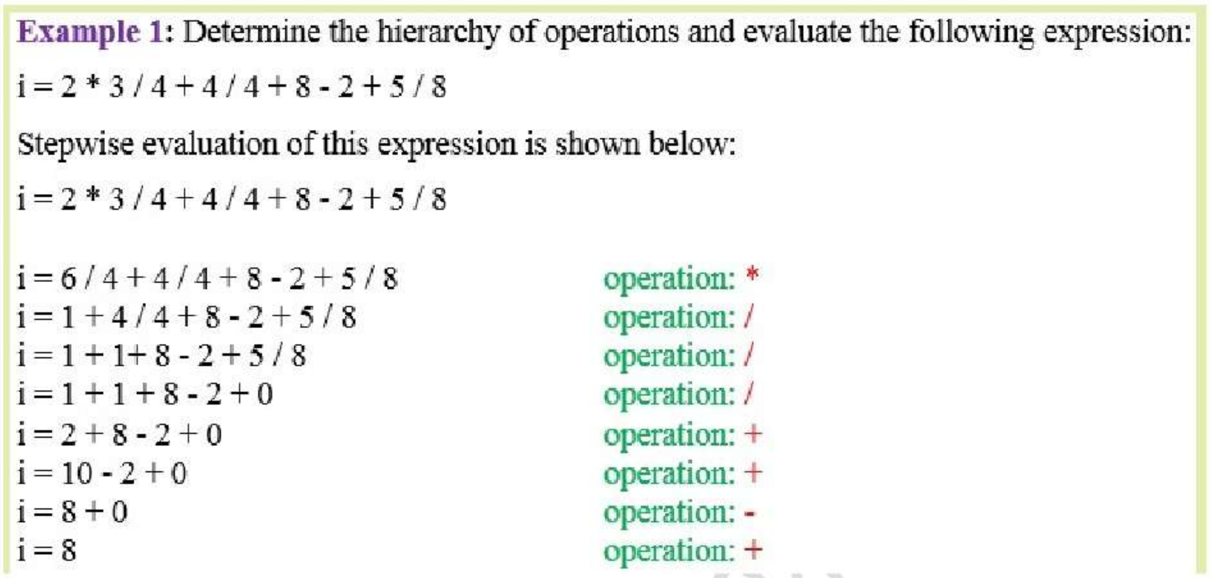
Ex1:



Ex2:



Ex3:



Example 1: Determine the hierarchy of operations and evaluate the following expression:

i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8

Stepwise evaluation of this expression is shown below:

i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8

i = 6 / 4 + 4 / 4 + 8 - 2 + 5 / 8	operation: *
i = 1 + 4 / 4 + 8 - 2 + 5 / 8	operation: /
i = 1 + 1+ 8 - 2 + 5 / 8	operation: /
i = 1 + 1 + 8 - 2 + 0	operation: /
i = 2 + 8 - 2 + 0	operation: +
i = 10 - 2 + 0	operation: +
i = 8 + 0	operation: -
i = 8	operation: +

Convert the following expressions into C Expressions.

Type Conversion/Casting

"Converting one type of data into another related type is called type-casting".
It is allowed in C that we can convert one form data into related another form.

- There are two types of casting in C.
- **1. Implicit type-casting**
- **2. Explicit type-casting**

1. Implicit type-casting:

- This type of casting happens from smaller size data types to bigger size data types; this is done by the compiler internally & automatically.
- Data accuracy is maintained and there is no loss of information.

Ex1:

```
float x = 22;
printf("\n x = %f", x);        //Output : x = 22.00000
```

- In this example compiler will automatically convert 22 value to 22.0000 internally and automatically.

2. Explicit type-casting:

- This type of casting happens from bigger size data types to smaller size data types; this is done by the programmer explicitly.
- There is a chances of data loss.

Syntax:

```
(datatype)variable;
```

Ex1:

```
float a;
int b;
a = 33.4444;
b = (int) a;
printf("\n b = %d", b);
```

Output:

```
b = 33
```

Control Structures/Control Statements

Control statements are used to control the flow of execution since C language is a sequential programming language.

If we want to **execute or skip** some parts/statements of the program as per requirement, then we can go for control statements.

1. Decision Making Statements: simple if, if-else, nested if-else, else-if ladder
2. Selection Statements: switch statement
3. Looping or Iterative Statements: for, while, do-while
4. Jump or Branching Statements: break, continue, return, goto

1. Decision Making statements:

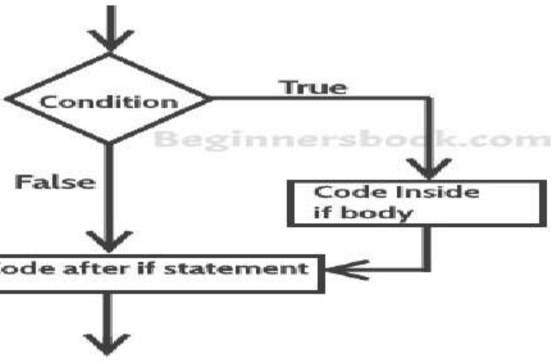
Based Condition/Expression Result we can take decision to execute parts of the program.

1.1 simple if ( ) statement: if the expression result is true then if body will execute, if the expression result is false then if body will not execute. If the condition returns false then the statements inside "if" are skipped.

Syntax:

```
if (Expression/Condition)
{
        statement1;
        statement2;
}
Next_Statement;
```

Flowchart:



Ex1:

```
int num;
num = 101;
if(num == 100)
{
        printf("Yes num is  equals to 100");
}
printf("\n I am out of if body…");
```

Ex 2:

```
int num;
num = 15;
if(num%3 == 0)
{
        printf("\nYes num is divisible by 3");
}
```

Ex 3:

```
int x;
x = 3;
if(x%2 == 0)
{
        printf("Yes Even");
}
```

Ex 4:

```
int x;
x = 14;
if(x%2 == 0)
{
        printf("\nEven Num");
        printf("\nYes i am inside if body");
}
printf("\nI am out of if body");
```

Ex5:

```
int x;
x = -6;

if(x >= 0)
{
        printf("\n Positive Number");
}
```

Ex6:

```
int x;
x = 12 ;
if(x%5 == 0)
{
        printf("\n x is divisible by 5");
        }
```

Ex7:

```
        int age;
        age = 44 ;
        if(age >= 18)
        {
                printf("\n Eligible for Voting");
        }
```

1.2 if else statement

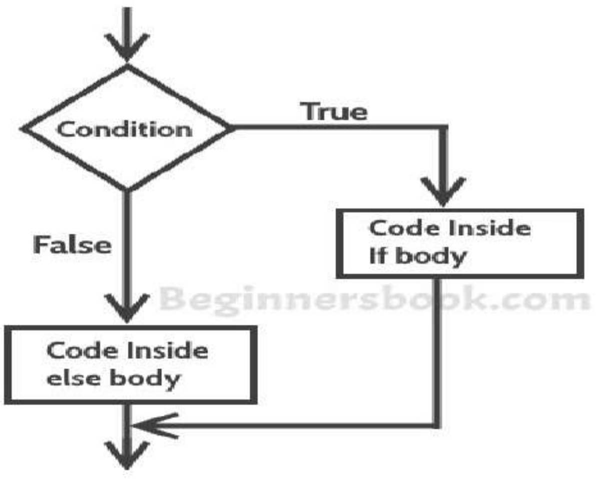
If condition returns true then the statements inside the body of "if" are executed and the statements inside body of "else" are skipped.
If condition returns false then the statements inside the body of "if" are skipped and the statements in "else" are executed.

Syntax:

```
if (Expression)
{
        statement1;
}
else
{
        statement2;
}

Next_Statement;
```

Flowchart:



Ex1: Program to check whether a given number is Even Or Odd.

```
        int num;
        num = 7 ;

        if(num%2 == 0)
        {
                printf("\n%d is Even",num);
        }
        else
        {
                printf("\n %d is Odd",num);
        }
        Printf("\nProgram Ends");
```

Output

```
        7 is Odd
```

Ex3: Program to check maximum of two numbers.

```
#include<stdio.h>
int main()
{
  int a,b;

  a = 10;
  b = 20;

  if(a>b)
  {
    printf("\nMaximum = %d",a);
  }
  else
  {
    printf("\nMaximum = %d",b);
  }

  return 0;
}
```

Output:

```
Maximum = 20
```

Ex2: Program to check whether a given number is Positive or Negative.

```
#include<stdio.h>
int main()
{
  int a;
  a = -10;
  if(a>=0)
  {
    printf("\n%d is Even Number",a);
  }
  else
  {
    printf("\n%d is Odd Number",a);
  }

  return 0;
}
```

Output:

```
10 is Odd number
```

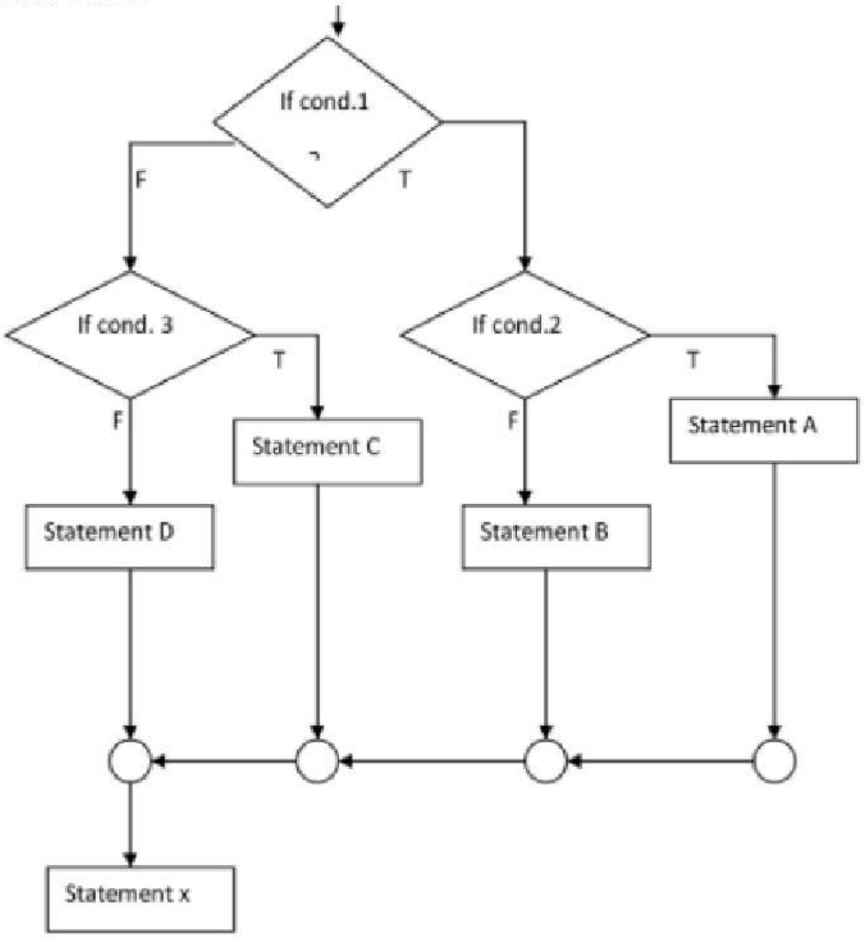
1.3 nested if-else statement

Nesting means defining if-else body into another if-else body, Execution of inner if-else body depends on outer expression result.

Syntax:

```
if (Expression1)
{
        if (Expression2)
        {
                Statement1;
        }
        else
        {
                Statement2;
        }
}
else
{
        Statement3;
}
Next_Statement;
```

Flowchart:



Ex:

```
int num;
num = -8 ;
if(num >= 0)
{
        if(num%2 == 0)
        {
                printf("\n %d is Positive & Even",num);
        }
        else
        {
                printf("\n %d is Positive & Odd",num);
        }
}
else
{
        printf("\n %d is Negative",num);
}
```

2.Write a C program to check maximum of three numbers

```
#include<stdio.h>
int main()
{
   int a,b,c;

   printf("\nEnter a, b, c values : ");
   scanf("%d %d %d",&a,&b,&c);

           if(a>b)
            {
                     if(a>c)
                     {
                              printf("\n%d is Greater",a);
                     }
                     else
                    {
                               printf("\n%d is Greater",c);
                     }
            }
           else
           {
                     if(b>c)
                     {
                                printf("\n%d is Greater",b);
                     }
                    else
                     {
                              printf("\n%d is Greater",c);
                    }
            }
   return 0;
}
```

1.4 else-if ladder/if-else-if ladder

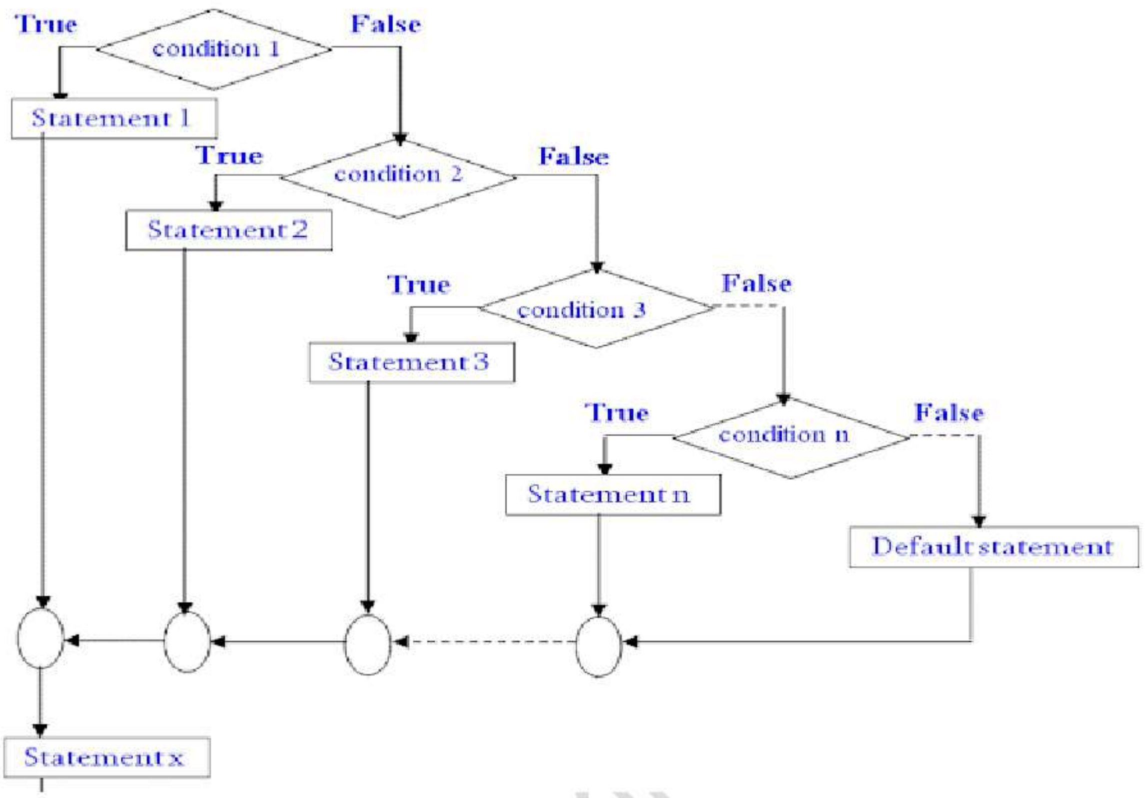
else-if ladder is used to execute a block of code that is decided from multiple options/Conditions.

If any condition becomes true then the statements associated with that if body is executed, and the rest of the C else-if ladder is bypassed(skipped).

Syntax:

```
if (Expression1)
{
        statement1;
}
else if (Expression2)
{
        statement2;
}
else if (Expression3)
{
        statement3;
}
else if (Expression4)
{
        statement4;
}
.........
else
{
        Statement n;
}
Next_Statement;
```

Flowchart



Ex1:

```
int x;
x = 4;
if(x>10)
{
        printf("\nx is Greater Than 10");
}
else if(x<10)
{
        printf("\nx is Less Than 10");
}
else
{
        printf("\n x is equals to 10");
}
```

Output

```
x is Less Than 10
```

2. Write a C program to display Grade of the students based on student percentage given below.

```
1. perc >= 85            -> DISTINCTION
2. perc >=60 && perc<85   ->FIRST CLASS
3. perc >=50 && perc<60   ->SECOND CLASS
4. perc >=35 && perc<50   ->THIRD CLASS
5. less than 35           ->FAIL
```

```
#include<stdio.h>
int main()
{
        float perc;
        printf("\nEnter your Percentage : ");
        scanf("%f",&perc);

        if(perc >= 85)
        {
            printf("\nYour Result :  DISTINCTION");
        }
        else if(perc>=60  && perc<85)
        {
            printf("\nYour Result :  FIRST CLASS");
        }
        else if(perc>=50 && perc<60)
        {
            printf("\nYour Result :  SECOND CLASS");
        }
        else if(perc>=35 && perc<50)
        {
            printf("\nYour Result :  THIRD CLASS");
        }
        else
        {
            printf("\nYour Result :  FAIL");
        }
  return 0;
}
```

Output:

2. Selection Statements: *switch statement*

Keywords: **switch, case, break, default.**

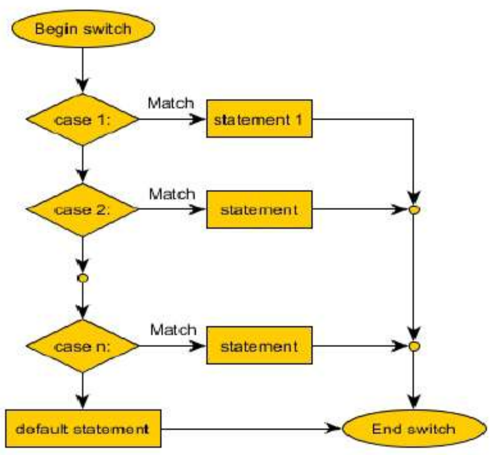
- switch statement is used to execute block of statements if any matching case is found in the switch body, with respect switch variable/value.
- If any matching case is found then default case will execute.
- Break statement is used to exit/come out of from switch body.
- Switch works only with integers and characters values. Hence variables must be of int or char type.

Syntax:

```
switch (Variable/value)
{
        case value1: statements;
                   break;
        case value2: statements;
                       break;
        case value2: statements;
                       break;
        .....
        .....
        case valueN: statements;
                       break;

         default: statements;
                 break;
}
```

Flowchart:



Ex1:

```
int x;
x = 3;
switch(x)
{
        case 1: printf("\n Result = 1");
                break;
        case 2: printf("\n Result = 2");
                break;
        case 3: printf("\n Result = 3");
                break;
        case 4: printf("\n Result = 4");
                break;
        default: printf("\n No Matching Value");
                break;
}
```

Output: Result = 3

Ex2: C Program to check whether a given character is Vowel or Consonant. (IMP)

```
#include<stdio.h>
int main()
{
  char ch;
  printf("\nEnter a Character : ");
  scanf("%c", &ch);
  switch(ch)
  {
    case 'a' :
    case 'A' :
    case 'e' :
    case 'E' :
    case 'i' :
    case 'I' :
    case 'o' :
    case 'O' :
    case 'u' :
    case 'U' :  printf("\n %c is Vowel",ch);
           break;
    default :   printf("\n %c is Consonant",ch);
           break;
  }
}
```

Output:

```
Enter a Character: A
A is Vowel
```

3. Looping statements OR Iterative statements

- If we want to execute some set of statements repeatedly until the conditions becomes false.

  Loops are classified into two categories

  1. Entry controlled loop: for, while
  2. Exit controlled loop: do-while

- Every loop has mainly 3 parts in C.
  1. **Initialization**
  2. **Test Condition**
  3. **Inc/Dec (Update)**

3.1 for loop:

Syntax:

```
for (Initialization; Condition; Inc/Dec/Update)
{
        statements;
}
```

for loop body will be executed repeatedly until the condition becomes false.
Iteration: Total number of for body execution.

Points to remember about for loop:

- The initialization statement is executed only once.
- Then, the test expression is evaluated. If the test expression is evaluated to false, the for loop is terminated.
- However, if the test expression is evaluated to true, statements inside the body of for loop are executed, and the update expression is updated.
- Again, the test expression is evaluated.

This process goes on until the test expression is false. When the test expression is false, the loop terminates.

Ex1:

```
#include<stdio.h>
int main()
{
        int i;
        for(i=1; i<=5; i++)
        {
                printf("\nHello BCA");
        }
        return 0;
}
```

Output

```
Hello BCA
Hello BCA
Hello BCA
Hello BCA
Hello BCA
```

Ex2: Print first 10 natural numbers.

```
#include<stdio.h>
int main()
{
            int i;
           for(i=1;   i<=10;   i++)
           {
                      printf("\n%d",i);
           }
         return 0;
}
```

Output

```
1
2
3
4
5
6
7
8
9
10
```

3.2 while loop: While loop works like for only, but change in the syntax, Until condition becomes false, the while body will execute repeatedly.

Syntax:

```
Initialization;
 while(Condition)
 {
          statements;
          Inc/Dec/Update;
 }
Next_Statement;
```

Ex1:

```
int i;
i = 1;
while(i<=3)
{
        printf("\n%d",i);
        i++;
}
```

Output:

```
1
2
3
```

3.3 do-while loop: do-while loop is also called as exit controlled loop,
Here irrespective condition result, do body will execute at least one time. Next iterations will execute based condition result.

- In do-while loop first do body will execute then it tests condition for next iterations.

Syntax:

```
Initialization;
 do
 {
        statements;
        inc/Dec/Update;
 }while(Condition);
```

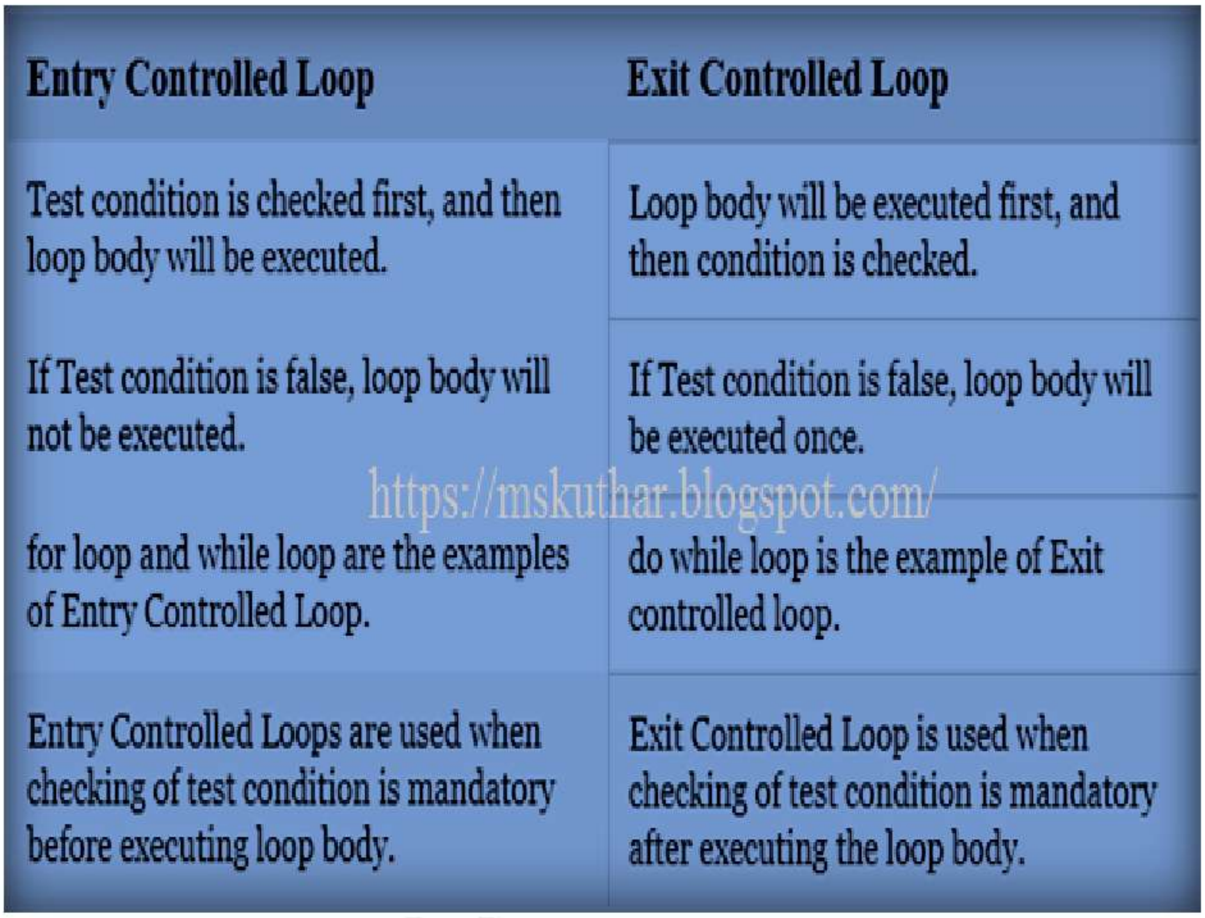
Ex:

```
int i;
i = 1;
do
{
        printf("\n%d",i);
        i++;
}while(i<=5);
```

Output:

```
1
2
3
4
5
```

Entry Controlled Loop and Exit Controlled Loop



Entry Controlled Loop	Exit Controlled Loop
Test condition is checked first, and then loop body will be executed.	Loop body will be executed first, and then condition is checked.
If Test condition is false, loop body will not be executed.	If Test condition is false, loop body will be executed once.
for loop and while loop are the examples of Entry Controlled Loop.	do while loop is the example of Exit controlled loop.
Entry Controlled Loops are used when checking of test condition is mandatory before executing loop body.	Exit Controlled Loop is used when checking of test condition is mandatory after executing the loop body.

4. Jump or Branching statements:

Jump statements are used to interrupt the normal flow of program. Jump statements are used to jump from one section of the program to another.

Types of jump statements in C.

1. **break;**
2. **continue;**
3. **goto label;**
4. **return;**

1. break;

- break statement is used to come out/break from looping statements and/or switch statements.
- it skips the rest of the iterations inside loops.
- break statement can be conditional or unconditional.

Ex1:

```
int i;
for(i=1; i<=10; i++)
{
        if(i==3)
        {
                break;
        }
        printf("\n %d",i);

}
printf("\n Out of for body..");
```

Output:

```
1
2
Out of for body.
```

2. continue: -

- It skips the current/present iteration in looping statements.
- It will not execute the current iteration but it executes remaining iterations.

Ex1:

```
int i;
for(i=1; i<=10; i++)
{
        if(i==5)
        {
                continue;
        }
        printf("\n %d", i);
}
printf("\n out of for body...");
```

Output:

```

1
2
3
4
6
7
8
9
10
out of for body...
```

Ex2:

```
int i;
for(i=1; i<=10; i++)
{
        continue;
        printf("\n %d", i);
}
printf("\n out of for body...");
```

Output:

```
out of for body...
```

3. goto statement: - to jump from one part of the program to another part, parts are nothing but labels.
- goto keyword is used to execute label block.
Syntax:

```
goto labelname;
```

labelname: It is a name of the part of the program, must be a valid identifier.
Label creation Syntax:

```
                labelname:
```

Ex1:

```
#include<stdio.h>
int main()
{
  int age;
  reenter:                          ->label creation
  printf("\nEnter Your Age : ");
  scanf("%d", &age);
  if(age < 0)
  {
    printf("\nEnter Positive Age... ");
    goto reenter;                   ->Label execution
  }
  else
  {
    goto yesgo;
  }
  yesgo:
  printf("\n Yes i am eligible for voting...");

}
```

4. return statement;

- it returns the control along with optional value back to the caller,
- it is used in Functions body.

A return statement ends the execution of a function, and returns control to the calling function.

Ex:

```
int add(int x, int y);  //Fn prototype OR Fn Declaration
int add(int x, int y)    //Fn Definition
{
        int res;
        res = x + y;
        return res;
}
int main()
{
        int res;
        res = add(10,20);   //Fn Call
        printf("\n Result = %d",res);
}
```

Nesting of Loops:

- Defining one loop inside another loop.
- Any number of loops can be defined inside another loop.
- there is no restriction for defining any number of loops. The nesting level can be defined at n times. You can define any type of loop inside another loop.

Syntax for nested loops:
Outer loop and Inner loop are the valid loops that can be a 'for' loop, 'while' loop or 'do-while' loop.

```
Outer Loop
{
        Inner Loop
        {
                Inner Loop statements:
        }

        Outer loops statements;

}
```

Ex1:

```
int i,j;

  for(i=1; i<=3; i++)
  {
    for(j=1; j<=3; j++)
    {
      printf("%d\t",j);
    }
    printf("\n\n");
  }
```

Output:

```
1       2       3
1       2       3
1       2       3
```

UNIT-WISE QUESTION ANK ANSWERS

UNIT-II

1. **Define operator. List types of operators in C.**
2. **What is the use assignment operator? Explain with an example.**
3. **List and explain different types of operators in C.**
4. **Specify the significance of increment & decrement operators in C.**
5. **What are Logical operators in C. Explain with examples.**
6. **Mention different types of relational operators in C.**
7. **List and explain bitwise operators in C.**
8. **What is Ternary/conditional operator, give the syntax and example.**
9. **What are special operators in C. Explain with examples.**
10. **What is operator precedence & associativity?**
11. **Convert the following expressions into C expressions.**
12. **What is Decision making statements in C? Explain various Decision-making statements in C with syntax and examples.**
13. **Explain various if-else statements in C.**
14. **Give the syntax of if-else statement with example.**
15. **Give the syntax of nested if else statement with example.**
16. **Give the syntax of else if ladder with an example.**
17. **Demonstrate nested if else with a program.**
18. **Evaluate the following expressions.**
19. **Classify unary, binary and ternary operators in C.**
20. **Mention Relational and logical operators in C.**
21. **State the syntax of if-else statement.**
22. **Differentiate entry-controlled loop and exit controlled loop.**
23. **Explain various decision-making statements in C.**
24. **Write a program to check whether an alphabet is vowel or consonant.**
25. **What is goto statement? Why it is not recommended to use.?**
26. **Write a C program to read three numbers and find the biggest of three.**
27. **Write a C program to demonstrate library functions in math.h.**
28. **Write a program to generate factorial of a given number.**
29. **Write a C program to generate n fibonacci sequence.**
30. **Write a program to read a number, find the sum of digits, reverse the number, and check it for palindrome.**
31. **Program to read numbers from keyboard continuously till the user presses 999 and to find the sum of only positive numbers.**
32. **Explain for, while, do-while loop with examples and syntax.**
33. **What is looping? Explain nested loops in detail.**
34. **Explain branching/Jump statements in C.**