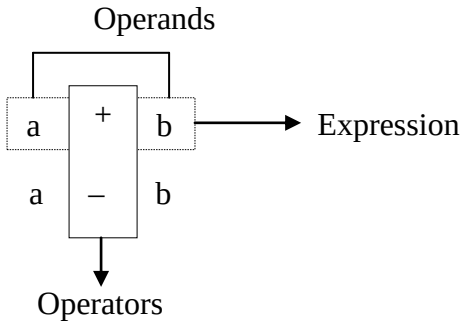
MODULE 2

Operators and Expressions

- ✓ **Operator: Operator is a symbol (or token) that specifies the operation to be performed on various types of data.**
- ✓ **Operand: A variable or a value on which the operation is performed is an operand.**
- ✓ **Expression: A sequence of operands and operators that reduces to a single value is an expression.**

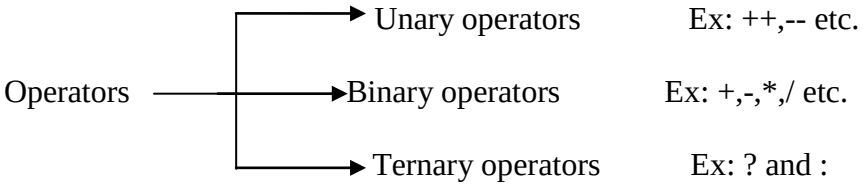


Classification of operators

The operators in C can be classified based on:

- The number of operands an operator has.
- The type of operation being performed.

1. Classification of operators based on the number of operands

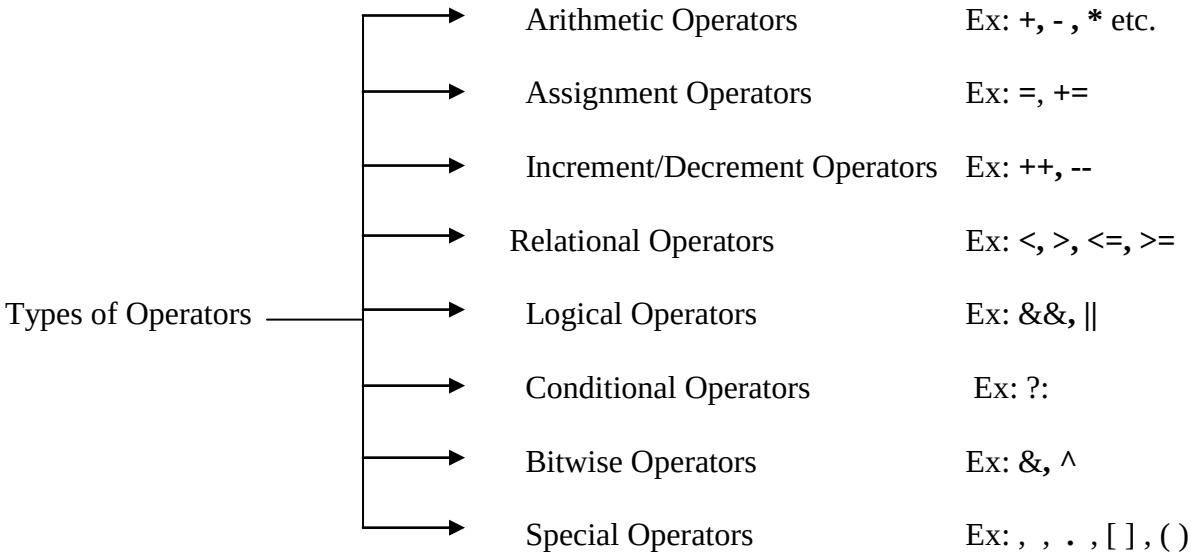


i) **Unary operator:** An operator which acts on only one operand to produce the result is called Unary operator.
Ex: -10,-a,*b,++a,a++,b-- etc.

ii) **Binary operator:** An operator which acts on two operands to produce the result is called Binary operator.
Ex: a+b, a*b, 10/5 etc

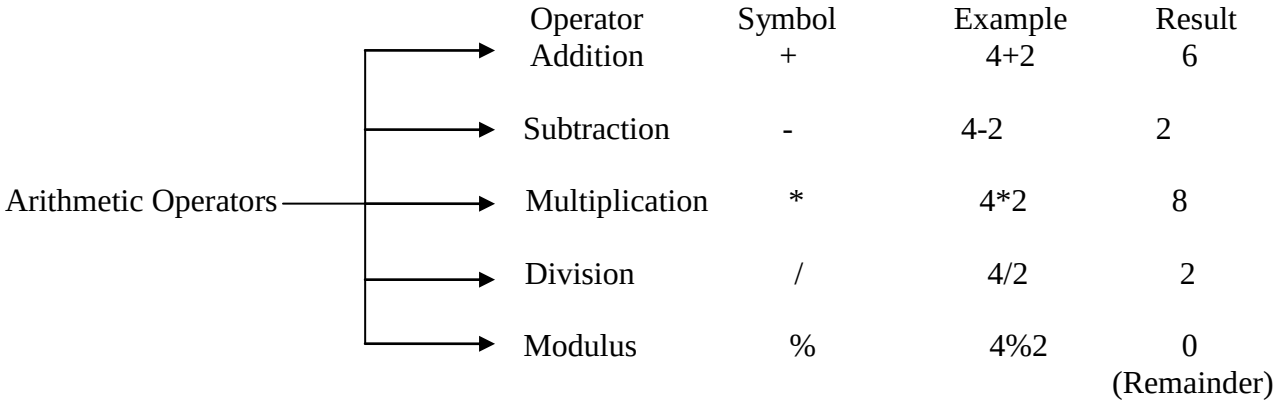
iii) **Ternary operator:** An operator which acts on three operands to produce the result is called Ternary operator.
Ex: a ? b : c;

Classification of operators based on type of operation



i) Arithmetic Operators

- ✓ **The operators that are used to perform arithmetic operations such as addition, subtraction, multiplication, division and modulus are called arithmetic operators.**
- ✓ These operators perform operations on two operands and hence they are called **binary operators.**



Operator	Symbol	Example	Result
Addition	+	4+2	6
Subtraction	-	4-2	2
Multiplication	*	4*2	8
Division	/	4/2	2
Modulus	%	4%2	0 (Remainder)

- ✓ **% (modulus operator)** divides the first operand by second and returns the **remainder.**
- ✓ % (modulus operator) cannot be applied to floating or double.
- ✓ % operator returns **remaining value (remainder)** of an integer division.

ii) Assignment Operators

- ✓ **An operator which is used to assign the data or result of an expression into a variable (also called memory location) is called an assignment operator.**
- ✓ Assignment operator is denoted by '=' sign.
  Ex: a=b; //value of b is copied into variable a

Types of Assignment statements

a). Simple Assignment Statement

```
variable= expression;
```

Ex: a=10;
a=b;
a=a+b;
Area= l*b; //Result of Expression l*b is copied into variable area.

b). Shorthand Assignment Statement

- ✓ **The operators such as +=, -=, *=, /= and %= are called shorthand assignment operators.**

Ex 1: a=a+10; // simple assignment
a+=10; // shorthand assignment
Ex 2: i=i+2;
i+=2;

- ✓ If $expr_1$ and $expr_2$ are expressions then

$expr_1$ op = $expr_2$;

is equivalent to

$expr_1$ = ($expr_1$) op ($expr_2$) ;

i.e., x*=y+1 means x=x*(y+1) ;

Simplify the expression: x *= y + 3
when $x = 10$ and $y = 5$

Solution: The given expression is:

x *= y + 3

can be written as

x *= (y + 3)

which in turn can be interpreted as:

x = x * (y + 3)
x = 10 * (5 + 3) (after substituting for x and y)
x = 10 * 8 = 80

So, **Result = 80**

c). Multiple Assignment Statement

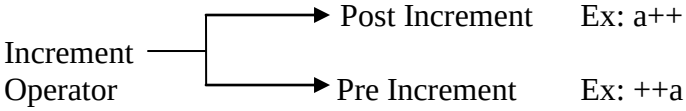
- ✓ Assigning a value or a set of values to different variables in one statement is called multiple assignment statement.
- ✓ The multiple assignments are used whenever same value has to be copied into various memory locations.

Ex: int i=10; //simple assignment
int j=10;
int k=10;
int i=j=k=10; //multiple assignment

iii) Increment and Decrement Operators

Increment Operator

✓ **'++' is an increment operator. This is unary operator. It increments the value of a variable by one.**



1. Post Increment

✓ **It increments the value after (post) the operand value is used. i.e., operand value is used first and then the operand value is incremented by 1.**

```
Ex: void main()
    {
     int a=20,b;          // a=20, b?
     b=a++;               // b=a=20, a=a+1= 20+1
     printf("%d",a);      // a=21
     printf("%d",b);      // b=20
    }
```

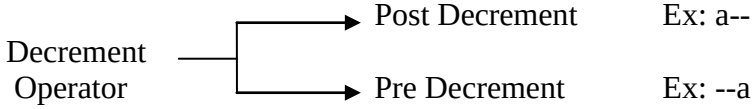
2. Pre Increment

✓ **It increments before (pre) the operand value is used. The operand value is incremented by 1 and this incremented value is used.**

```
Ex:  void main()
     {
      int a=20,b;          // a=20, b?
      b=++a;               // a=++a=a+1= 20+1, b=a=21
      printf("%d",a);      // a=21
      printf("%d",b);      // b=21
     }
```

Decrement Operator

✓ **'--' is a decrement operator. This is a unary operator. It decrements the value of a variable by one.**



1. Post Decrement

✓ **It decrements the value after (post) the operand value is used. i.e., operand value is used first and then the operand value is decremented by 1.**

```
Ex: void main()
    {
     int a=20,b;          // a=20, b?
     b=a--;               // b=a=20, a=a-1= 20-1
     printf("%d",a);      // a=19
     printf("%d",b);      // b=20
    }
```

2. Pre Decrement

✓ **It decrements before (pre) the operand value is used. The operand value is decremented by 1 and this decremented value is used.**

```
Ex:  void main()
```

```
{
  int a=20,b;          // a=20, b?
  b=--a;               // a=--a=a-1= 20-1, b=a=19
  printf("%d",a);      // a=19
  printf("%d",b);      // b=19
}
```

Eg:

Initially a=10, b=5, c=20 simplify the following expression

i. **W = a++ + ++b + c++ - ++a**

= 10 + 6 + 20 - 12

a=11 b=6 c=21 a=12

w = 24

a=12 , b=6, c=21, w=24

ii. **X = --a + b-- + ++c – a++ - --b**

= 9 + 5 + 21 - 9 - 5

a=9 b=4 c=21 a=10 b=5

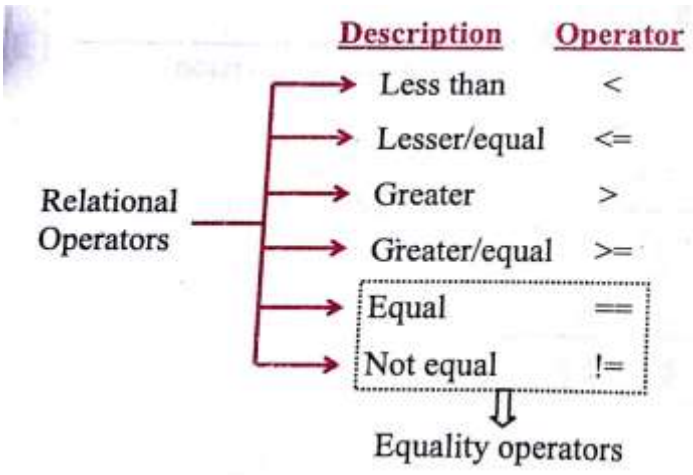
= **21**

a= 10 b= 5 c= 21 x= 21

iv) Relational operators

- ✓ **The operators that are used to find the relationship between two operands are called relational operators.**
- ✓ **The relationship between the two operand values results in true (always 1) or false (always 0).**

Relational operators available in C are:



- ✓ All the relational operators are having a same priority and left to right associativity.
- ✓ The relational operators have lower precedence than arithmetic operators.

Equality operators

- ✓ C supports two kinds of equality operators to compare their operands for strict equality or inequality.

equal = =

not equal ! =

- ✓ Equality operators have lower precedence than the relational operators.

Eg: a=10, b=6

i. C=a>b
C=10>6 → 10 is greater than 6 so the operator returns value 1, indicating true
C=1

ii. C=a<b
C=10<6 → 10 is not less than 6, so the operator returns value 0, indicating false
C=0

iii. C=a>=b
C=10>=6
C=1

iv. C=a<=b
C=10<=6
C=0

v. C=a==b
C=10==6 → ==checks whether 10 is equal to 6, if equal return 1, else returns 0
C=0

vi. C=a!=b
C=10!=6 → != checks whether 10 is not equal to 6, if not equal returns 1, else returns 0
C=1

v) Logical operators

- ✓ **The operators that are used to combine two or more relational expressions are called logical operators.**
- ✓ **The output of relational expression is true or false, the output of logical expression is also true or false.**

Logical operators available in C:

	Operators
not	!
and	&&
or	||

Operand1	**Operand2**	**AND(&&)**	**OR(||)**	**NOT(!) (Operand1)**
True(1)	True(1)	True(1)	True(1)	False(0)
True(1)	False(0)	False(0)	True(1)	False(0)
False(0)	True(1)	False(0)	True(1)	True(1)
False(0)	False(0)	False(0)	False(0)	True(1)

Logical NOT: The logical NOT operator is denoted by '!'. The output of not operator can be true or false. The result is true if the operand value is false and the result is false if the operand is true.

Logical AND: The logical AND operator is denoted by '&&'. The output of and operator is true if both the operands are evaluated to true. If one of the operand is evaluated false, the result is false.

Logical OR: The logical OR operator is denoted by '||'. The output of or operator is true if and only if at least one of the operands is evaluated to true. If both the operands are evaluated to false, the result is false.

Ex:

i. Given A=10, Evaluate C=!A
A=10 → ***other than value 0, any value is treated as 1. So, A=1***
C = !A
C = !1
C = 0

ii. Given A=10, B=15, Evaluate C = A&&B
A=10 → treated as A=1 (***Because other than value 0, any value is treated as 1)***
B=15 → treated as B=1 (***Because other than value 0, any value is treated as 1)***
C = A&&B
= 1 && 1
C= 1

iii. Given A=20, B=25, Evaluate C = A || B
A = 20 → treated as A=1 (***Because other than value 0, any value is treated as 1)***
B = 25 → treated as B=1 (***Because other than value 0, any value is treated as 1)***
C = A || B
= 1 || 1
C = 1

vi) Conditional Operator

- ✓ **The conditional operator is also called as 'ternary operator'.**
- ✓ **An operator that operates on the three operands is called ternary operator.**

Syntax:

```
(expr1)? expr2: expr3;
```

Where,

- expr1 is evaluated first.
- If expr1 is evaluated to true, then expr2 is evaluated.
- If expr1 is evaluated to false, then expr3 is evaluated.

Example:

1. Write a C program to find the biggest of two numbers using conditional operator.

```
#include<stdio.h>
void main()
{
        int a,b,big;
        printf("Enter the values of a and b:");
        scanf("%d%d",&a,&b);
        big=(a>b)?a:b;
        printf("Big=%d",big);
        getch();
}
```

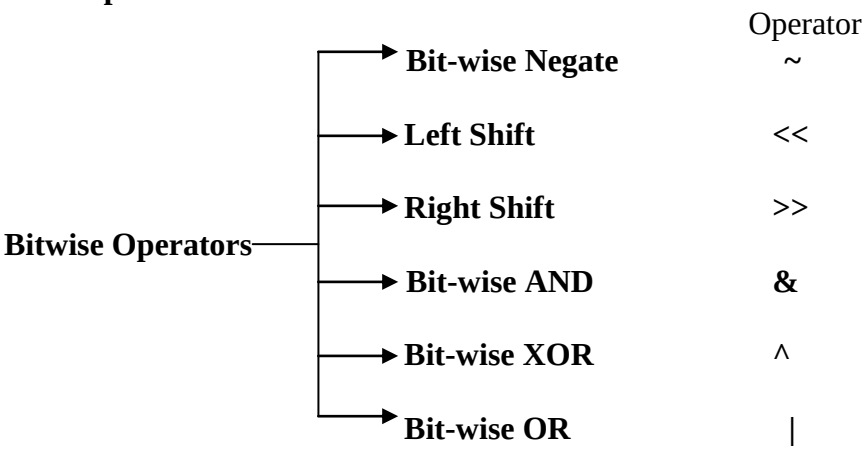
2. Write a C program to find the smallest of two numbers using conditional operator.

```
#include<stdio.h>
void main()
{
        int a,b,small;
        printf("Enter the values of a and b:");
        scanf("%d%d",&a,&b);
        small=(a<b)?a:b;
        printf("Small=%d",big);
        getch();
}
```

vii) Bitwise Operators

- ✓ **The operators that are used to manipulate the bits of given data are called bitwise operators.**

Bitwise operators available in C are:



- ✓ These may only be applied to integral operand. i.e., char, short, int and long whether signed or unsigned.

a. Bit-wise Negate or One's Complement(~)

- ✓ **The operator that is used to change every bit from 0 to 1 and 1 to 0 in the specified operand is called One's complement operator.**

Truth table of One's Complement(~):

Op1	~Op1
0	1
1	0

- ✓ One's complement operator is denoted by '~(tilde)' symbol.

Ex:

i. Let A= 15, Evaluate B = ~A
Convert A = 15 to binary representation and apply Bitwise Negate to binary values

Binary Representation (8 bit representation)
15 = 0 0 0 0 1 1 1 1

240 = 1 1 1 1 0 0 0 0

Write a C program to show the usage of Bitwise Negate operator.

```
#include<stdio.h>
void main()
{
        int a=10,b;
        b=~a;
        printf("~%d=%d",a,b);
}
```

Output: ~10=245

Binary Representation(8 bit representation):

10 = 0 0 0 0 1 0 1 0

245= 1 1 1 1 0 1 0 1

b. Left shift operator (<<)

- ✓ **The operator that is used to shift the data by a specified number of bit positions towards left is called 'left shift operator'.**

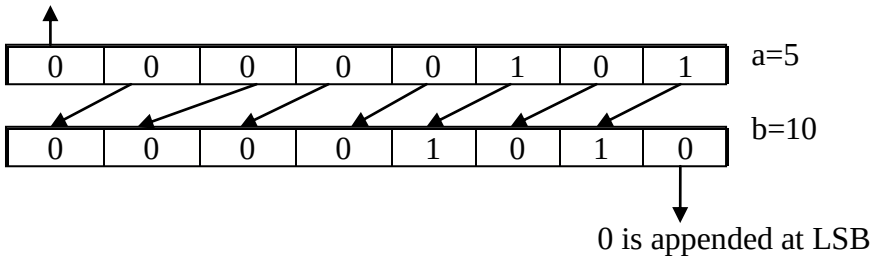
Syntax:

```
b=a<<num;
```

Ex:

i. Let a=5, Evaluate b = a<<1

MSB is discarded



b = 10

Write a C program to show the usage of Left Shift operator.

```
#include<stdio.h>
void main()
{
        int a=5,b;
        b=a<<1;
        printf("%d<<1=%d",a,b);
}
```

Output: 5<<1 = 10

c. Right shift operator (>>)

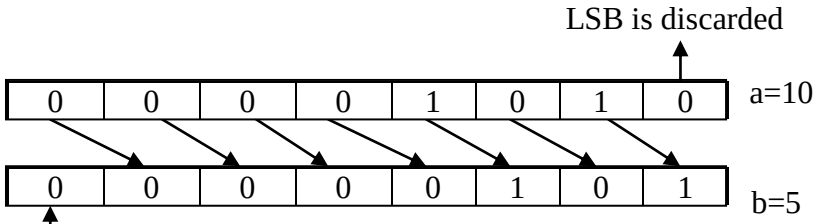
- ✓ **The operator that is used to shift the data by a specified number of bit positions towards right is called 'right shift operator'.**

Syntax:

```
b=a>>num;
```

Ex:

i. Let a=10, Evaluate b = a>>1



b = 5

Write a C program to show the usage of Right Shift operator.

```
#include<stdio.h>
void main()
{
        int a=10,b;
        b=a>>1;
        printf("%d>>1=%d",a,b);
}
```

Output: 10>>1=5

d. Bit-wise AND (&)

- ✓ **If the corresponding bit positions in both the operands are 1, then AND operation results in 1, otherwise AND operation results in 0.**

Truth table of Bit-wise AND (&):

Op1	Op2	Op1&Op2
0	0	0
0	1	0
1	0	0
1	1	1

Ex:

Let a=10 and b=6. Evaluate c = a & b

Binary Representation: (8 bit representation)

```
10 = 0 0 0 0 1 0 1 0
06 = 0 0 0 0 0 1 1 0
-----------------
02 = 0 0 0 0 0 0 1 0
```

C = 2

Write a C program to show the usage of '&' operator.

```
#include<stdio.h>
void main()
{
        int a=10,b=6;
```

```
        c=a&b;
        printf("%d&%d=%d",a,b,c);
}
Output: 10&6=2
```

e. Bit-wise OR (|)

- ✓ **If the corresponding bit positions in both the operands are 0, then OR operation results in 0, otherwise OR operation results in 1.**

Truth table of Bit-wise OR (|):

Op1	Op2	Op1|Op2
0	0	0
0	1	1
1	0	1
1	1	1

Ex:

Let a=10, b=6. Evaluate c = a | b

Binary Representation:

```
10 = 0 0 0 0 1 0 1 0
06 = 0 0 0 0 0 1 1 0
-----------------
14 = 0 0 0 0 1 1 1 0
```

C=14

Write a C program to show the usage of '|' operator.

```
#include<stdio.h>
void main()
{
        int a=10,b=6;
        c=a|b;
        printf("%d|%d=%d",a,b,c);
}
```

Output: 10|6=14

f. Bit-wise XOR (^)

- ✓ **If the corresponding bit positions in both the operands are different, then XOR operation results in 1, otherwise XOR operation results in 0.**

```
0^0=0
0^1=1
1^0=1
1^1=0
```

Ex:

Let a = 10 and b = 6. Evaluate c = a^b

Binary Representation:

```
10 = 0 0 0 0 1 0 1 0
06 = 0 0 0 0 0 1 1 0
```

12 = 0 0 0 0 1 1 0 0

C= 12

Write a C program to show the usage of '^' operator.

```
#include<stdio.h>
void main()
{
        int a=10,b=6;
        c=a^b;
        printf("%d^%d=%d",a,b,c);
        getch();
}
```

Output: 10^6=12

viii) Special Operators

- ✓ Comma operator, size of operator and [ ], ->, Indirection operator, *, dot operator etc.

i) Comma operator

- ✓ Comma Operator has the least precedence among all the operators and it is left associative operator.
- ✓ Comma Operator is used in the declaration to separate the variables.
  Ex: int a,b,c;
- ✓ It can be used to separate the items in the list.
  Ex: a=12,345,678;
- ✓ It can be used to combine two or more statements into a single statement.
  Ex: sum=a+b,sub=a-b,mul=a*b,div=a/b,mod=a%b;

ii) sizeof()

- ✓ 'sizeof()' operator is used to determine the number of bytes occupied by a variable or a constant in the memory.
- ✓ Ex: sizeof(char) 1 byte
  sizeof(int) 2 bytes
  sizeof(float) 4 bytes

Arithmetic Expressions

- ✓ **The expression consisting of only arithmetic operators such as +, -, *, / and % are called arithmetic expressions.**

Example: Write the equivalent C expression for the Mathematical expressions.

Mathematical Expression	C Equivalent Expression
$S = \frac{a+b+c}{2}$	S=(a + b +c) /2
$area = \sqrt{s(s-a)(s-b)(s-c)}$	area=sqrt(s*(s-a)*(s-b)*(s-c))

$x = \sqrt{2\pi n}$	x=sqrt(2*3.142*n)
$\frac{a}{b}$	a/b
$x = -\frac{b}{2a}$	x=-b/(2*a)
$ax^2 + bx + c$	a*x*x+b*x+c

$\sin\left(\frac{b}{\sqrt{a^2+b^2}}\right)$	sin(b / sqrt(a*a + b*b))
$\tau_1 = \sqrt{\left\{\frac{\sigma_x - \sigma_y}{2}\right\} + \tau xy^2}$	tow1 = sqrt((rowx – rowy)/2 + tow*x*y*y)
$\tau_1 = \sqrt{\left\{\frac{\sigma_x - \sigma_y}{2}\right\}^2 + \tau xy^2}$	tow1 = sqrt(pow((rowx – rowy)/2,2) + tow*x*y*y)

$x1 = \frac{-b+\sqrt{b^2-4ac}}{2a}$	x1 = (-b + sqrt(b*b – 4*a*c)) / (2*a)
$\frac{e^{\lvert a\rvert+b}}{x+y}(2x+3)$	exp(abs(a) + b) / (x + y) * (2*x + 3)
$y = \frac{\alpha+\beta}{Sin\theta} + \lvert x\rvert$	y = (alpha + beta) / sin(theta*3.1416/180) + abs(x)

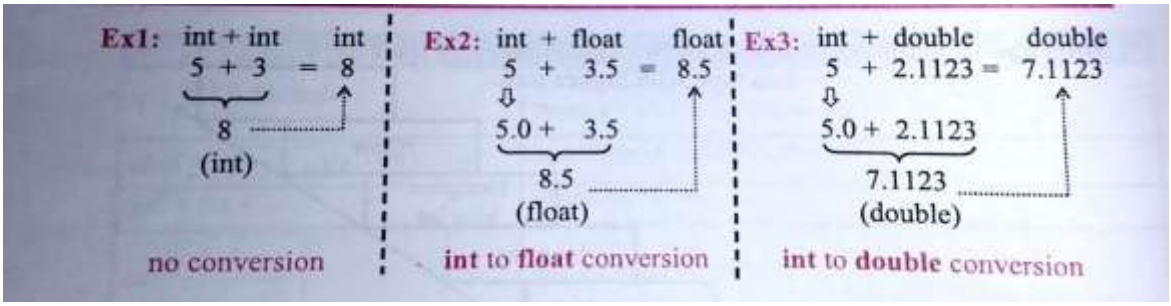
Type Conversion

- ✓ **The process of converting the data from one data type to another data type is called Type Conversion.**

Type Conversion	Implicit Conversion	1/2.0	1.0/2.0=0.5000
	Explicit Conversion	1/(float)2	1.0/2.0=0.5000

- ✓ **C compiler converts the data type with lower rank to the data type with higher rank. This process of conversion of data from lower rank to higher rank automatically by the C compiler is called "Implicit type Conversion".**

char → short int → int → unsigned int → long int → float → double Lower Rank
Higher Rank
Data types → Data types



- ✓ If one operand type is same as that of other operand type, no conversion takes place.
  Ex: int + int = int, float + float = float
- ✓ If one operand type is 'int' and other operand type is 'float', then the operand with type int is promoted to 'float' (because float is up in ladder compared with int).
- ✓ **The programmer can instruct the compiler to change the type of the operand from one data type to another data type. This forcible conversion from one data type to another data type is called "Explicit type Conversion" (Type Casting).**

Syntax:

(type) Expression

Ex: (int) 9.43

i= (int) 5.99 / (int) 2.4, now it becomes 5/2 = 2, i=2

(float) (3/10) = 3.0 / 10.0 = 0.3

BRANCHING AND LOOPING

Branching Statements

- ✓ The statements that transfer the control from one place to other place in the program with or without any condition are called branch statements or selection statements.
- ✓ The branching statements are classified into two types:
  i. Conditional branch statements
  ii. Unconditional branch statements

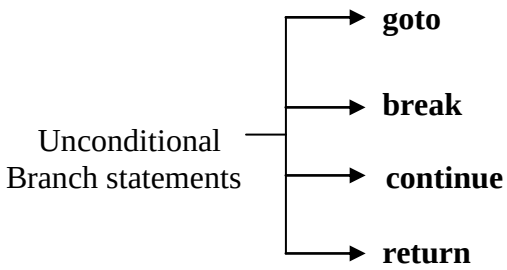
Conditional branch statements

- ✓ **The statements that transfer the control from one place to another place in the program based on some conditions are called Conditional branch statements.**

Conditional Branch statements
- **Simple- if** (One-way selection)
- **if-else** (Two-way selection)
- **Nested-if** (Multi-way selection)
- **Cascaded if-else or else-if ladder** (Multi-way selection)
- **switch** (Multi-way selection)

Unconditional branch statements

- ✓ **The statements that transfer the control from one place to another place in the program without any condition are called Unconditional branch statements.**

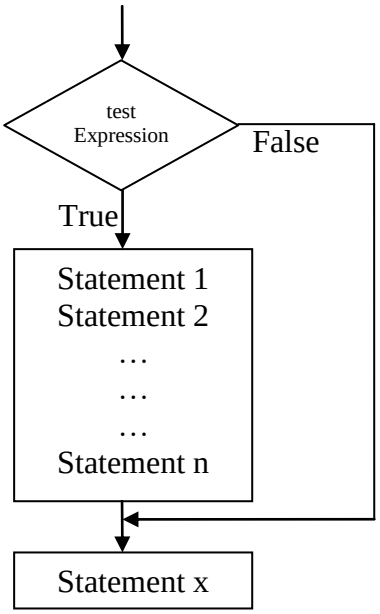


Conditional branching
if Statement

- ✓ The 'if' statement is the simplest form of decision control statement.
- ✓ When a set of statements have to be executed when an expression (condition) is evaluated to true or skipped when an expression (condition) is evaluated to false, then if statement is used.
- ✓ It is used whenever there is only one choice (alternative). Hence it is also called as **"One-way decision or selection statement".**

Syntax of if statement

```
if(test Expression)
{
   Statement 1;
   Statement 2;
        ...
        …
        …
   Statement n;
}
Statement x;
```



- ✓ The 'if' structure may include one statement or 'n' statements enclosed within curly brackets.

Working Principle

- ✓ First the test expression is evaluated.
- ✓ If the test expression is true, then the statements of 'if' block (statement 1 to n) are executed.
- ✓ Otherwise these statements will be skipped and the execution will jump to statement x.

Rules for if-statement

a) 'if' must be followed by an expression and the expression must be enclosed within parenthesis.

b) If multiple statements have to be executed when the expression is true, then all those statements must be enclosed within braces.

c) No semicolon is required for an if-statement.
if(a<b); // statement indicates a NULL statement. It will 'do nothing'.

Example Programs for 'if' statement

1. Write a C program to determine whether a person is eligible to vote or not.

```
#include<stdio.h>
void main()
{
    int age;
    printf("Enter the age:");
    scanf("%d",&age);
    if(age>=18)
        printf("\nThe person is eligible to vote");
    if(age<18)
        printf("\nThe person is not eligible to vote");
}
```

2. Write a C program to print the largest of two numbers.

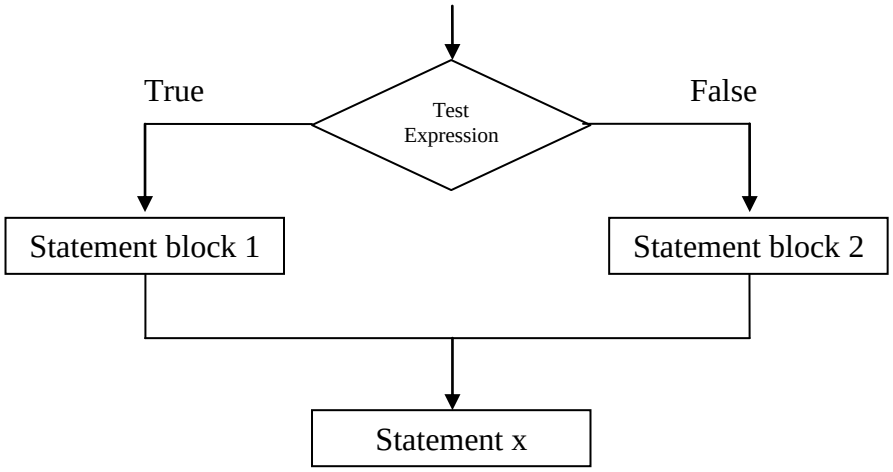
```
#include<stdio.h>
void main()
{
    int a,b;
    printf("Enter the two numbers:");
    scanf("%d%d",&a,&b);
    if(a>b)
        printf("\n a is greater than b");
    if(b>a)
        printf("\n b is greater than a");
}
```

2 if-else statement

- ✓ If one set of activities have to be performed when an expression is evaluated to true and another set of activities have to be performed when an expression is evaluated to false, then if-else statement is used.

Syntax of if-else statement:

```
if(test Expression)
{
   Statement block 1;

 }

else

{

   Statement block 2;

 }

Statement x;
```



- ✓ The is-else statement is used when we must choose between two choices (alternatives). Hence is also called as **"Two-way Decision or Selection Statement".**

Working Principle

- ✓ According to the if-else construct, first the 'test expression' is evaluated.
- ✓ If the expression is true then Statement block 1 is executed and Statement block 2 is skipped.
- ✓ If the expression is false the Statement block 2 is executed and Statement block 1 is ignored.
- ✓ Now in any case after the Statement block 1 or 2 gets executed the control will pass to Statement x. It is executed in every case.

Example Programs for 'if-else' statement

1. Write a C program to determine whether a person is eligible to vote or not.

```
#include<stdio.h>
 void main()
{
      int age;
       printf("Enter the age:");
      scanf("%d",&age);
      if(age>=18)
          printf("\nThe person is eligible to vote");
      else
          printf("\nThe person is not eligible to vote");
  }
```

2. Write a C program to check whether the number is even or odd and print the appropriate message.

```
#include<stdio.h>
void main()
{
    int n,rem;
    printf("Enter a number:");
    scanf("%d",&n);
    if(n%2==0)
        printf("Number is even");
    else
        printf("Number is odd");
}
```

3. Write a C program to enter a character and then determine whether it is a vowel or not.

```
#include<stdio.h>
void main()
{
    char ch;
    printf("Enter any character:");
    scanf("%c",&ch);
    if(ch=='a'||ch=='e'||ch=='i'||ch=='o'||ch=='u'||ch=='A'||ch=='E'||ch=='I'||ch=='O'||ch=='U')
        printf("\n Character is a vowel");
    else
        printf("\n Character is a consonant");
}
```

4. Write a C program to find whether a given year is leap year or not.

```
#include<stdio.h>
void main()
{
    int year;
    printf("Enter any year:");
    scanf("%d",&year);
    if(((year%4==0)&&(year%100!=0))||(year%400==0))
        printf("%d is a leap year",year);
    else
        printf("%d is not a leap year",year);
}
```

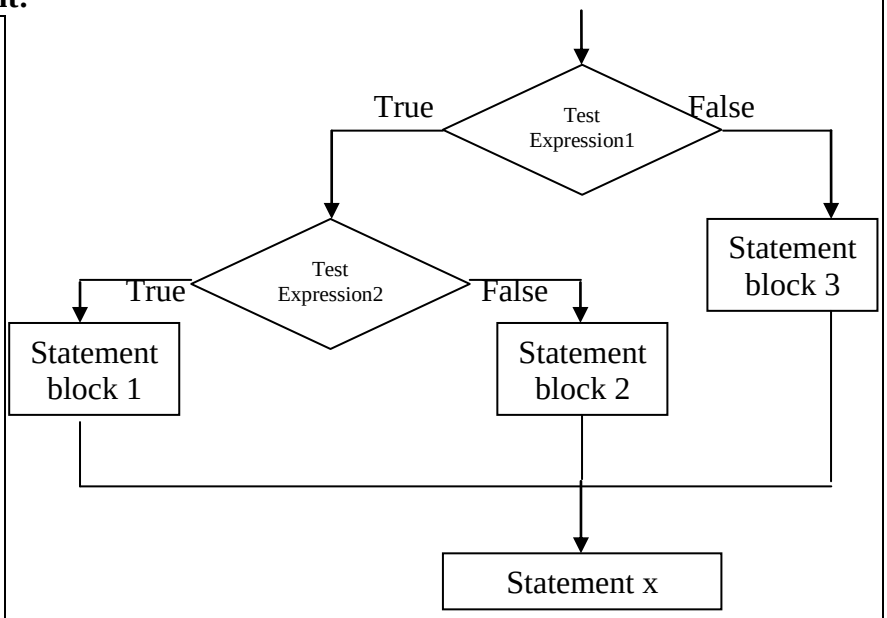
3 Nested if-else statement

- ✓ **An if-else statement within if statement or an if-else statement within else statement is called "nested if or if-else statement".**

- ✓ When an action has to be performed based on many decisions involving various types of expressions and variables, then this statement is used. So it is called as **"Multi-way decision statement".**

Syntax for nested if-else statement:

```
if(test Expression1)
{
   if(test Expression2)
  {
     Statement block 1;
   }
   else
   {
     Statement block 2;
    }
}
else

{

   Statement block 3;

 }

Statement x;
```



Working Principle

- ✓ If the test expression 1 is evaluated to true, then the test expression 2 is checked for true or false. If the test Expression2 is evaluated to true, then the statements in block 1 are executed, otherwise the statements in block 2 are executed. After executing the inner if-else the control comes out and the statement x is executed.
- ✓ If the test expression1 itself is false, then the statements in block 3 are executed. After executing these statements, the statement x is executed.

Example Programs for 'nested if-else' statement

1. Write a C program to find the largest of three numbers.

```
#include<stdio.h>
void main()
{
      int a,b,c;
      printf("Enter the three numbers:");
      scanf("%d%d%d",&a,&b,&c);
      if(a>b)
      {
         if(a>c)
            printf("Max=%d",a);
         else
            printf("Max=%d",c);
      }
      else
      {
         if(b>c)
```

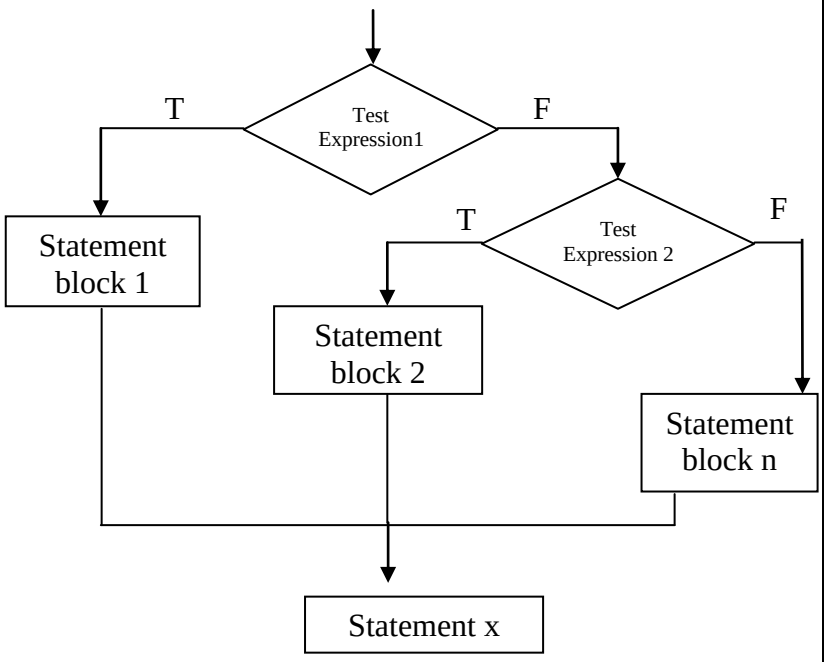
```
            printf("Max=%d",b);
        else
            printf("Max=%d",c);
    }
}
```

4 Cascaded if-else or if-else-if or else-if-ladder Statement

- ✓ **It is a special case of nested-if statement where nesting takes place only in the else part.**
- ✓ **The orderly nesting of if-else statement only in the else part is called 'else-if-ladder'.**
- ✓ When an action has to be selected based on the range of values, then this statement is used. So it is called **"Multi-way Decision or Selection Statement".**

Syntax :

```
if(test expression1)
{
   Statement block 1;
}
else if(test expression2)
{
   Statement block 2;
}
          ….

else

{

   Statement block n;

}

Statement x;
```



Working Principle

- ✓ The 'Expressions' are evaluated in order. If any expression is true then the statement associated with it is executed and this terminates the whole chain and statement x is executed.
- ✓ The last 'else' part is executed when all the test expression are false.

Example Programs for 'cascaded if-else' statement

1. Write a C program to demonstrate the use of cascaded if structure.

```
#include<stdio.h>

void main()

{

    int x,y;

    printf("Enter the values of x and y:");
```

```
    scanf("%d%d",&x,&y);
    if(x==y)
        printf("\n The two numbers are equal");
    else if(x>y)
        printf("\n %d is greater than %d",x,y);
    else
        printf("\n %d is less than %d",x,y);
}
```

2. Write a C program to print whether the number is positive, negative or zero.

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

3. Write a C program to input 3 numbers and then find the largest of them using '&&' operator.

```
#include<stdio.h>
void main()
{
    int a,b,c;
    printf("Enter the three numbers:");
    scanf("%d%d%d",&a,&b,&c);
    if(a>b && a>c)
        printf("\n %d is the largest number",a);
    else if(b>a && b>c)
        printf("\n %d is the largest number",b);
    else
        printf("\n %d is the largest number",c);
}
```

4. Write a C program to display the examination results.

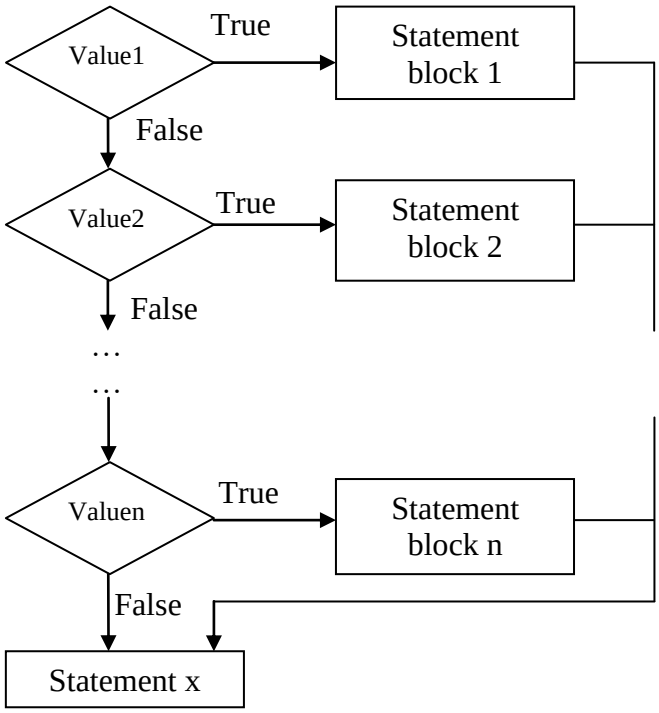
```
#include<stdio.h>
void main()
{
    int marks;
    printf("Enter the marks:");
    scanf("%d",&marks);
    if(marks>=75)
        printf("\n First Class with Distinction");
    else if(marks>=60&&marks<75)
        printf("\n First Class");
    else if(marks>=50&&marks<60)
        printf("\n Second Class");
    else if(marks>=40&&marks<50)
        printf("\n Third Class");
    else
        printf("\n Fail");
}
```

4 switch statement

- ✓ The 'switch' statement is a control statement used to make a select one alternative among several alternatives.
- ✓ It is a **"multi-way decision statement"** that tests whether an expression matches one of the case values and branches accordingly.

switch statement syntax

```
switch(expression)
{
 case value-1: Statement block 1;
               break;
 case value-2: Statement block 2;
               break;
                 …..
                 …..

 case value-n: Statement block n;
               break;
 default: Statements;

}
Statement x;
```



Working Principle

- ✓ First the expression within switch is evaluated.
- ✓ The value of an expression is compared with all the case values.
- ✓ The value of an expression within switch is compared with the case value-1. If it matches then the statements associated with that case are executed. If not then the case value-2 is compared, if it matches then the associated statements are executed and so on.
- ✓ The "default" statements are executed when no match is found.
- ✓ A default is optional.
- ✓ The 'break' statement causes an exit from the switch. 'break' indicates end of a particular case and causes the control to come out of the switch. **[Significance of break within switch statement]**

Example Programs for 'switch' statement

1. Write a C program to display the grade.

```
#include<stdio.h>
void main()
{  char grade;
   printf("Enter the grade:");
   scanf("%c",&grade);
   switch(grade)
   {
      case 'O': printf("Outstanding");
                break;
      case 'A': printf("Excellent");
                break;
      case 'B': printf("Good");
```

```
                break;
        case 'C': printf("Fair");
                break;
        case 'F': printf("Fail");
                break;
        default : printf("Invalid grade");
      }
     }
```

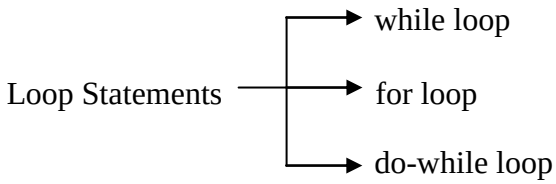
2. Write a C program to enter a number from 1 to 7 and display the corresponding day of the week using switch.

```
#include<stdio.h>
void main()
{   int day;
    printf("Enter any number:");
    scanf("%d",&day);
    switch(day)
    {
      case 1: printf("Sunday");
              break;
      case 2: printf("Monday");
              break;
      case 3: printf("Tuesday");
              break;
      case 4: printf("Wednseday");
              break;
      case 5: printf("Thursday");
              break;
      case 6: printf("Friday");
              break;
      case 7: printf("Saturday");
              break;
      default : printf("invalid input");
    }
}
```

LOOPS

- ✓ A set of statements may have to be repeatedly executed for a specified number of times or till a condition is satisfied.
- ✓ **The statements that help us to execute a set of statements repeatedly for a specified number of times or till a condition is satisfied are called as looping constructs or loop control statements.**

- ✓ These statements are also called as Repetitive or Iterative Statements.

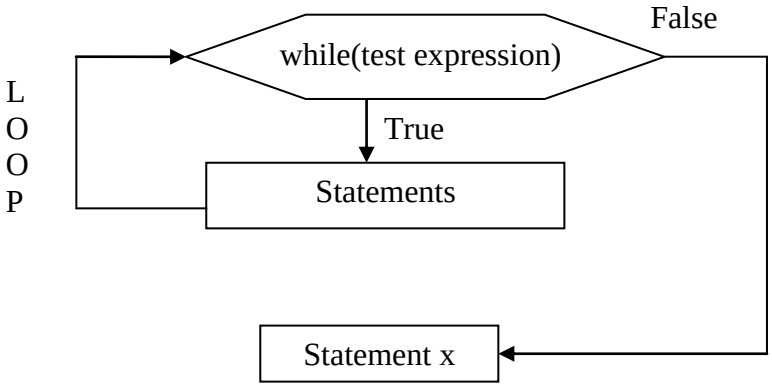


1 while Loop

- ✓ **It is a control statement using which the programmer can give instructions to the computer to execute a set of statements repeatedly as long as specified condition is satisfied.**
- ✓ The expression is evaluated to TRUE or FALSE in the beginning of the while loop. Hence it is called as **"Entry-Controlled Loop".**

Syntax of while loop

```
while(test expression)
{
   Statements;
 }
Statement x;
```



Working principle

- ✓ The test expression is evaluated first, if it is TRUE then the set of statements within the body of the loop are executed repeatedly as long as specified test expression is TRUE.
- ✓ If the test expression is false, then the control comes out of the loop by skipping the execution of the statements within the body of the loop, by transferring the control to the Statement x.

Example programs of 'while' loop

1. Write a C program to print the natural numbers from 1 to 10.

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

2. Write a C program to print the natural numbers from 1 to N.

```
#include<stdio.h>
void main()
{
    int i=0,n;
    printf("Enter the value of n:");
    scanf("%d",&n);
    while(i<=n)
    {
            printf("%d\n",i);
            i++;
    }
 }
```

3. Write a C program to print the natural numbers from N to 1.

```
#include<stdio.h>
void main()
{
    int n,i=n;
    printf("Enter the value of n:");
    scanf("%d",&n);
    while(i>=1)
    {
            printf("%d\n",i);
            i--;
    }
  }
```

4. Write a C program to calculate the sum of the first 10 natural numbers.

```
#include<stdio.h>
void main()
{
    int i=1,sum=0;
    while(i<=10)
    {
            sum=sum+i;
            i++;
    }
     printf("\nSum=%d",sum);
 }
```

5. Write a C program to print the characters from A to Z.

```
#include<stdio.h>
void main()
{
        char ch='A';
        while(ch<= 'Z')
        {
                printf("%c\t",ch);
                ch++;
        }
}
```

6. Write a C program to find the factorial of a given number using while statement.

```
#include<stdio.h>
void main()
{
        int fact=1,i=1,n;
       printf("Enter the value of n:\n");
       scanf("%d",&n);
       while(i<=n)
       {
                fact=fact*i;
                i++;
        }
       printf("Factorial of a given number is=%d",fact);
 }
```

7. Write a C program to read a number and determine whether it is palindrome or not.

```
#include<stdio.h>
void main()
{       int digit,rev=0,num,n;
        printf("Enter the number:\n");
        scanf("%d",&num);
        n=num;
        while(num!=0)
        {
                digit=num%10;
                rev=rev*10+digit;
                num=num/10;
        }
        printf("The reversed number is=%d\n",rev);
        if(n==rev)
                printf(" The number is a palindrome");
        else
                printf("The number is not a palindrome");
}
```

8. Program to find GCD and LCM of 2 numbers.

```
#include<stdio.h>
void main()
{       int a,b,m,n,gcd,lcm,rem;
        printf("Enter the value for m and n :\n");
        scanf("%d %d", &a, &b);
```

```
        m=a;
        n=b;
        while(n!=0)
        {
                rem=m%n;
                m = n;
                n = rem;
        }
        gcd = m;
        lcm = ( a * b ) / gcd;
        printf("The GCD of  %d %d  numbers is=%d\n", a, b, gcd );
        printf("The LCM of  %d %d  numbers is=%d\n", a, b, lcm );
}
```

OUT PUT:

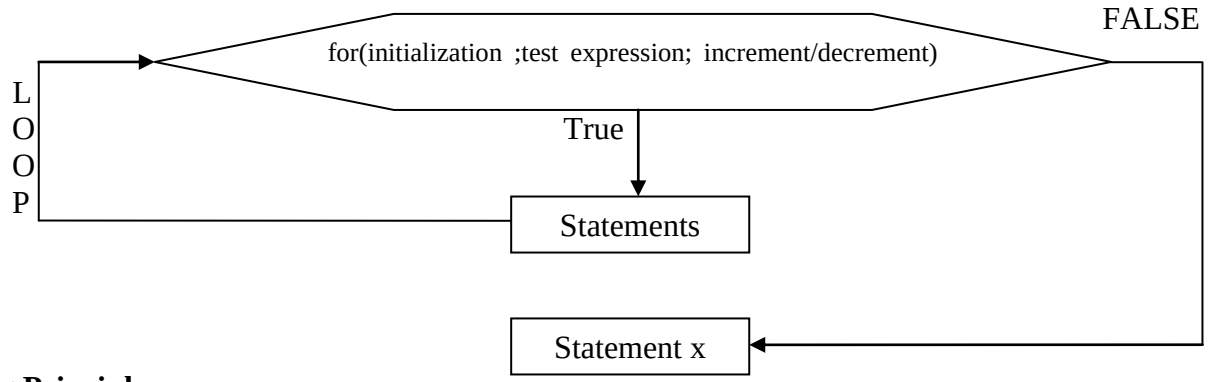
```
Enter the value for m and n:
6
12
The GCD of 6 12 numbers is= 6
The LCM of 6 12 numbers is= 12
```

2 for Loop (Counter- Controlled Loop)

- ✓ **It is a control statement using which the programmer can give instructions to the computer to execute a set of statements repeatedly for a specific number of times.**
- ✓ It is also called **as entry controlled loop** or **pretest loop**.

Syntax of for loop

```
for(initialization ;test expression; increment/decrement)
{
   Statements;
 }
Statement x;
```



- ✓ **Working Principle**
- ✓ **initialization:** In this section loop variable is initialized, like i=0, n=0, i=1,n=1. (i and n are loop variables).
- ✓ **test expression:** The test expression may be a relational expression or logical expression or both, which is evaluated to TRUE or FALSE. Depending on the value of the test expression, the body of the loop is executed. If the test expression is TRUE, then the body of the loop is executed. This process of execution of body of the loop is continued as long as the expression is TRUE. When the test expression becomes FALSE, execution of the statements contained

in the body of the loop are skipped, thereby transferring the control to the Statement x, which immediately follows the for loop.

- ✓ **increment/decrement:** This section increments or decrements the loop variables after executing the body of the loop.

Infinite Loop

- ✓ A for loop without a test expression is an Infinite loop.

```
Ex: for(i=0;    ;i++)
    {
            ……
    }
```

is an "infinite" loop.

Example programs of 'for' loop

1. Write a C program to print the natural numbers from 1 to 10.

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

2. Write a C program to print the natural numbers from 1 to n.

```
#include<stdio.h>
void main()
{
    int i,n;
    printf("Enter the value of n:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
          printf("%d\n",i);
    }
 }
```

3. Write a C program to compute the sum of the series 1+2+3+……+n.

```
#include<stdio.h>
void main()
{
        int i,sum=0,n;
        clrscr();
        printf("Enter the number of terms:");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                sum=sum+i;
        }
         printf("\nSum of the series=%d",sum);

  }
```

4. Write a C program to generate 'n' Fibonacci numbers.

```
#include<stdio.h>
void main()
{
         int n,f1,f2,f3,i;
```

```
        f1=0;
        f2=1;
        printf("Enter the value of n:");
        scanf("%d",&n);
        if(n==1)
                printf("%d",f1);
        else
        {
                printf("%d\t%d\t",f1,f2);
                for(i=3;i<=n;i++)
                {
                        f3=f1+f2;
                        printf("%d\t",f3);
                        f1=f2;
                        f2=f3;
                }
        }
  }
```

5. Write a C program to print the characters from A to Z.

```
#include<stdio.h>
void main()
{
        char ch;

        for(ch='A';ch<='Z';ch++)
        {

                printf("%c\t",ch);
        }

}
```

Output: A B C D E F ----------Z

6. Write a C program to find the factorial of a given number using for statement.

```
 #include<stdio.h>
void main()
{
        int fact=1,i,n;

        printf("Enter an integer:\n");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                fact=fact*i;
        }
        printf("Factorial of a given number is=%d",n,fact);

 }
```

Nested Loops

- ✓ The loops that can be placed inside other loops.

- ✓ It will work with any loops such as while, do-while and for.
- ✓ Nested loop is commonly used with the for loop because this is easiest to control.
- ✓ The 'inner for loop' can be used to control the number of times that a particular set of statements will be executed.
- ✓ The 'outer for loop' can be used to control the number of times that a whole loop is repeated.

Example Program: Write a C program to print the following output.

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4
```

```
#include<stdio.h>
void main( )
{
        int i,j;
        for(i=1;i<=5;i++)
        {
                printf("\n");
                for(j=1;j<=i;j++)
                {
                        printf("%d",j);
                }
        }
}
```

Outer for loop

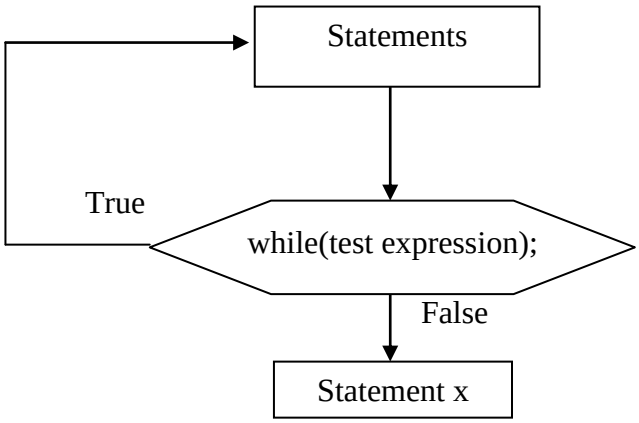
Inner for loop

3 do-while Loop

- ✓ **do- while loop is used when a set of statements have to be repeatedly executed at least once.**
- ✓ Since the test expression is evaluated to TRUE or FALSE at the end of do-while loop, the do-while loop is called as **exit-controlled loop**.
- ✓ The while and for loop test the expression at the top.
- ✓ The do-while tests the bottom after making each passes through the loop body.

Syntax of do-while loop

```
do{
        Statements;
   }while(test expression);
Statement x;
```



Working Principle

- ✓ It is a **post-test or bottom-testing loop** and hence the statements contained within the body of the loop are executed first and then the expression is evaluated to TRUE or FALSE. If it is TRUE, then the statements contained within the body of the loop are executed once again and the expression is evaluated. This is repeated until the expression is evaluated to FALSE.
- ✓ If the expression is FALSE, then the control comes out of the loop by skipping the execution of the statements within the body of the loop, by transferring the control to the Statement x.

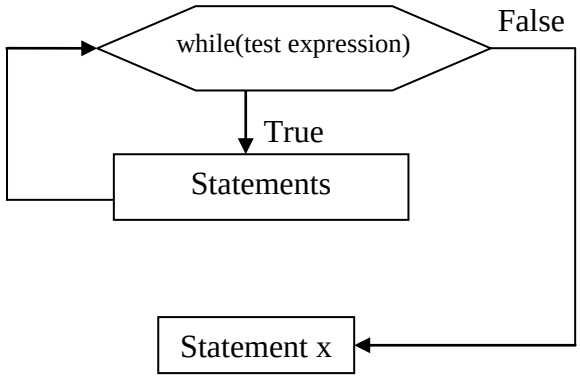
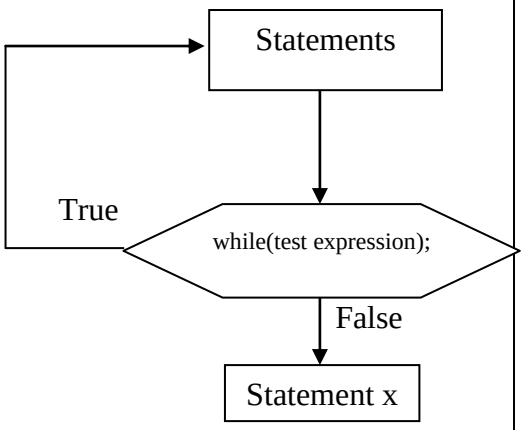
Difference between while loop and do-while loop

	while loop	**do-while loop**
1	It is **entry controlled loop. (top-testing)**	It is **exit controlled loop**. (bottom-testing)
2	It is pre-test loop.	It is post-test loop.

3	Syntax:
	while(expression)
	{
	Statements;
	}
		Syntax:
		do {
		Statements;
		} while(expression);

4	**Working Principle:** If the expression is evaluated to true, then the statements within the body of loop are executed. If the expression is false, then the statements within the body of the loop are not executed.	**Working Principle:** On reaching do statement, it proceeds to execute the statements within the body of the loop irrespective of the test expression. If the expression is evaluated to true, then the statements within the body of loop are executed once again. If the expression is false, then the statements within the body of the loop are not executed.
5	In while loop, while statement does not ends with semicolon (;).	In do-while loop, while statement ends with semicolon (;).

6	Ex:
	int i=0,sum=0;
	while(i<=n)
	{
	sum=sum+i;
	i=i+1;
	}
		Ex:
		int i=0,sum=0;
		do{
		sum=sum+i;
		i=i+1;
		}while(i<=n);

7	Flowchart:  	Flowchart: 

Example programs of 'do-while' loop

1. Write a C program to print the natural numbers from 1 to n.

```
#include<stdio.h>
void main()
{
    int i=1,n;
    printf("Enter the value of n:");
    scanf("%d",&n);
    do
    {
            printf("%d\n",i);
            i++;
    }while(i<=n);

}
```

2. Write a C program to calculate the sum and average of first 'n' numbers.

```
    #include<stdio.h>
    void main()
    {
             int n,i=0,sum=0;
             float avg=0.0;
             printf("Enter the value of n:");
             scanf("%d",&n);
            do
            {
                    sum=sum+i;
                    i++;

            }while(i<=n);
            avg=sum/n;
            printf("The sum of numbers=%d",sum);
            printf("The average of numbers=%f",avg);
    }
```

Unconditional Branch statements

1 break statement

- ✓ It is 'jump' statement which can be used in **switch statement and loops**.
- ✓ **The break statement in switch statement causes the control to come out of the switch statement and transfers the control to the statement which follows the switch statement.**

Ex:

```
switch(ch)
{
 case 1: printf("1st statement");
          break;
 case 2: printf("2nd  statement");
          break;
 default: printf("nth statement");
}
```

If 'case 1' is selected by the programmer then the output will be 1st statement only. It will directly come out of the switch.

- ✓ **If break is executed in a loop (for/while/do-while) then the control comes out of the loop and the statement following the loop will be executed.**

Syntax

1. while(…)
   {
   ……
   if(condition)
   break;
   ……
   }
   ……

   Transfers the control out of the while loop

2. do
   {
   ……
   if(condition)
   break;
   ……
   }while(condition);
   …

   Transfers the control out of the do-while loop

3. for(…)
   {
   ……
   if(condition)
   break;
   ……
   }
   ……

   Transfers the control out of the for loop

Example programs of 'break' statement

1. Write a C program to print the numbers (0 to 4) using break.

```
#include<stdio.h>
void main()
{
        int i=0;
        while(i<=10)
        {
                if(i==5)
                    break;
                printf("%d\n",i);
                i=i+1;
        }
}
```

2.6.2 continue statement

- ✓ During execution of a loop, it may be necessary to skip a part of the loop based on some condition. In such cases, we use continue statement.
- ✓ **The continue statement is used only in the loops to terminate the current iteration and continue with the remaining iterations.**

Syntax:

1. while(…)
   {
   if(condition) — Transfers the control to the expression of the while loop
   continue;
   …….
   }
   …….

2. do
   {
   ……
   if(condition)
   continue;
   …… — Transfers the control to the expression of the do-while loop
   }while(condition);
   ……

3. for(…)
   {
   ……
   if(condition) — Transfers the control to the expression of the for loop
   continue;
   ……
   }
   ……

Example programs of 'continue' statement

1. Write a C program to display the output in the following form 1 3 4 5

```
#include<stdio.h>
void main()
{
        int i;
        for(i=1;i<=5;i++)
        {
                if(i==2)
                    continue;
                printf("%d",i);
         }

}
```

Output: 1 3 4 5 [if i==2 then the continue statement is executed and the statements following continue are skipped]

3 goto statement

- ✓ **The goto statement is a 'jump' statement that transfers the control to the specified statement (Label) in a program unconditionally.**
- ✓ The specified statement is identified by 'label'(symbolic name). Label can be any valid variable name that is followed by a colon (:).

Syntax

```
goto label;
……
…….
label:
Statements
```

Forward Jump

```
label:
  Statements
  ……
  ……
goto label;
```

Backward Jump

Example programs of 'goto' statement

1. Write a C program to calculate the sum of all numbers entered by the user.

```
#include<stdio.h>
void main()
{
        int n,i,sum=0;
        printf("Enter the number:\n");
        scanf("%d",&n);
        sum=i=0;
        top:    sum=sum+n;
                i=i+1;
       if(i<=n)         goto top;
                printf("Sum of Series=%d",sum);

 }
```

Finding Root of Quadratic Equations

- ✓ Quadratic Equations $ax^2+bx+c=0$.
- ✓ Formula for calculating Roots of Quadratic equations. $\boldsymbol{x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}}$

```
#include<stdio.h>
#include<math.h>
void main()
{
        float a,b,c,disc,root1,root2,realp,imgp;
        printf("enter the co-efficients\n");
        scanf("%f%f%f",&a,&b,&c);
        if(a==0||b==0||c==0)
        {
                printf("invaled input");
                exit(0);
        }
        disc=b*b-4*a*c;
                printf("discriminate=%f\n",disc);
        if(disc==0)
        {
                root1=root2=-b/(2*a);
                printf("roots are equal\n");
                printf("root1=%f\n root2=%f\n",root1,root2);
        }
        else if(disc>0)
        {
                root1=(-b+sqrt(d))/(2*a);
                root2=(-b-sqrt(d))/(2*a);
                printf("roots are distinct\n");
                printf("root1=%f\n root2=%f\n",root1,root2);
        }
        else
        {
                printf("roots are imaginary\n");
                realp=-b/(2*a);
                imgp=sqrt(fabs(d))/(2*a);
                printf("root1=%f+i%f\n",realp,imgp);
                printf("root2=%f-i%f\n",realp,imgp);
        }
    }
}
```

Pascal Triangle

- ✓ Pascal Triangle is a Triangle form which, each number is the sum of immediate top row near by numbers. The Value of edge is always 1.
- ✓ The first thing one needs to know about Pascal's triangle is that all the numbers outside the triangle are "0"s. To build the triangle, start with a "1" at the top, the continue putting numbers below in a triangular pattern so as to form a triangular array. So, each new number added below the top "1" is just the sum of the two numbers above, except for the edge which are all "1"s.
- ✓ This can be summarized as:

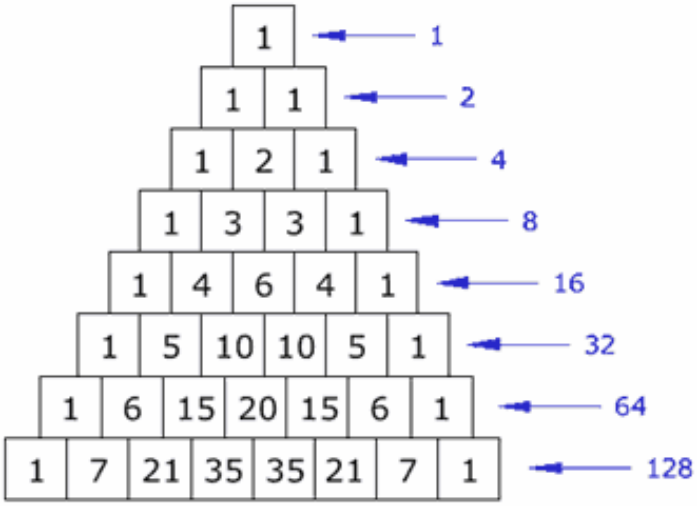
  0 row =1

  1 row = (0+1), (1+0) = 1, 1

  2 row = (0+1), (1+1), (1+0) = 1, 2, 1

  3 row = (0+1), (1+2), (2+1), (1+0) = 1, 3, 3, 1

  4 row = (0+1), (1+3), (3+3), (3+1), (1+0) = 1, 4, 6, 4, 1



- The sum of all the elements of a row is twice the sum of all the elements of its preceding row. For example, sum of second row is 1+1= 2, and that of first is 1. Again, the sum of third row is 1+2+1 =4, and that of second row is 1+1 =2, and so on. This major property is utilized to write the code in C program for Pascal's triangle.
- The sequence of the product of each element is related to the base of the natural logarithm, *e*.
- The left and the right edges of Pascal's triangle are filled with "1"s only.
- All the numbers outside the triangle are "0"s.
- The diagonals next to the edge diagonals contain natural numbers (1, 2, 3, 4, ….) in order.
- The sum of the squares of the numbers of row "n" equals the middle number of row "2n".

Example: C Program to print Pascal Triangle.

```
#include<stdio.h>
void main()
{
   int row,count=1,space,i,j;
     printf("Enter the Number of Rows: ");
     scanf("%d",&row);
   for(i=0; i<=row; i++)
   {
          for(space=1; space<=row-i; space++)
        printf ("  ");
           for(j=0; j<=i; j++)
     {
           if(j==0||i==0)
                  num=1;
           else
                count=(count*(i-j)/(j+1));
                printf("%4d ",count);
     }
     printf("\n");
   }
 }
```

Computation of Binomial Coefficients

✓ The binomial coefficient C(n,r) or nCr is the number of ways of picking ‘r’ unordered outcomes from ‘n’ possibilities, also known as a combination or combinatorial number. The number of ways that r objects can be chosen from among n objects;

$$^{n}C_{r} = n ! /(n - r)! * r !$$

Ex: C Program to Calculate Binomial coefficient.

```
#include<stdio.h>
Void main()
{
int i,n,r;
long int x,y,z,nCr;
printf(“enter the value of n and r\n”);
scanf(“%d %d”, &n, &r);
```

```
x = y = z = 1;
for(i = n; i >=1; i--)
{
        x = x * i;
}
for(i = n-r; i >=1; i--)
{
        y = y * i;
}
for(i = r; i >=1; i--)
{
        z = z * i;
}
nCr = x / (y * z);
printf("%d C %d= %d", n ,r, nCr);
}
```

Output:

Enter the value of n and r:
6
2
6C2 = 15

ASSIGNMENT QUESTIONS MODULE 2

1. Explain various Operators used in C language.
2. Explain Logical operators and Relational operators.
3. Explain the one way selection statement (if) in C language with syntax, flowchart and example.
4. Explain the two way selection statements (if-else, nested if-else, cascaded if else) in C language with syntax, flowchart and example.
5. Explain the switch statement with syntax and example.
6. Design and develop a C program to read a *year* as an input and find whether it is *leap year* or not. Also consider end of the centuries.
7. Explain the different types of loops in C with syntax and example.
8. Explain the use of break and continue statement in loops with example.
9. Design and develop a C program to r***everse*** of an integer number **NUM** and check whether it is PALINDROME or NOT.
10. Write a c program that takes three coefficients (*a*, *b*, and *c*) of a Quadratic equation ($ax^2+bx+c=0$) as input and compute all possible roots for a given set of coefficients with appropriate messages.
11. List the Difference between while loop and do-while loop.
12. Write a c program to find GCD and LCM of two numbers.
13. Write a C Program to compute **Sin(x)** using Taylor series approximation given by **Sin(x) = x - (x^3/3!) + (x^5/5!) - (x^7/7!) + …….**Compare your result with the built- in Library function. Print both the results with appropriate messages.
14. Write a C Program to print Pascal Triangle.
15. Write a C program to find binomial coefficient.