

# **C++ language**

**BY:ENG.VIAN ADNAN**

**AL NAHRAIN UNIVERSITY/COMPUTER CENTER**

**Computer** is a device capable of performing computations and making logical decisions at speeds millions and even billions of times faster than human beings.

Computers process data under the control of sets of instructions called **computer programs**.

**Programming** is the process of writing instructions for a computer in a certain order to solve a problem.

The computer programs that run on a computer are referred to as **software (SW)**. While the hard component of it is called **hardware (HW)**.

Developing new software requires written lists of instructions for a computer to execute. Programmers rarely write in the language directly understood by a computer

# History of C++

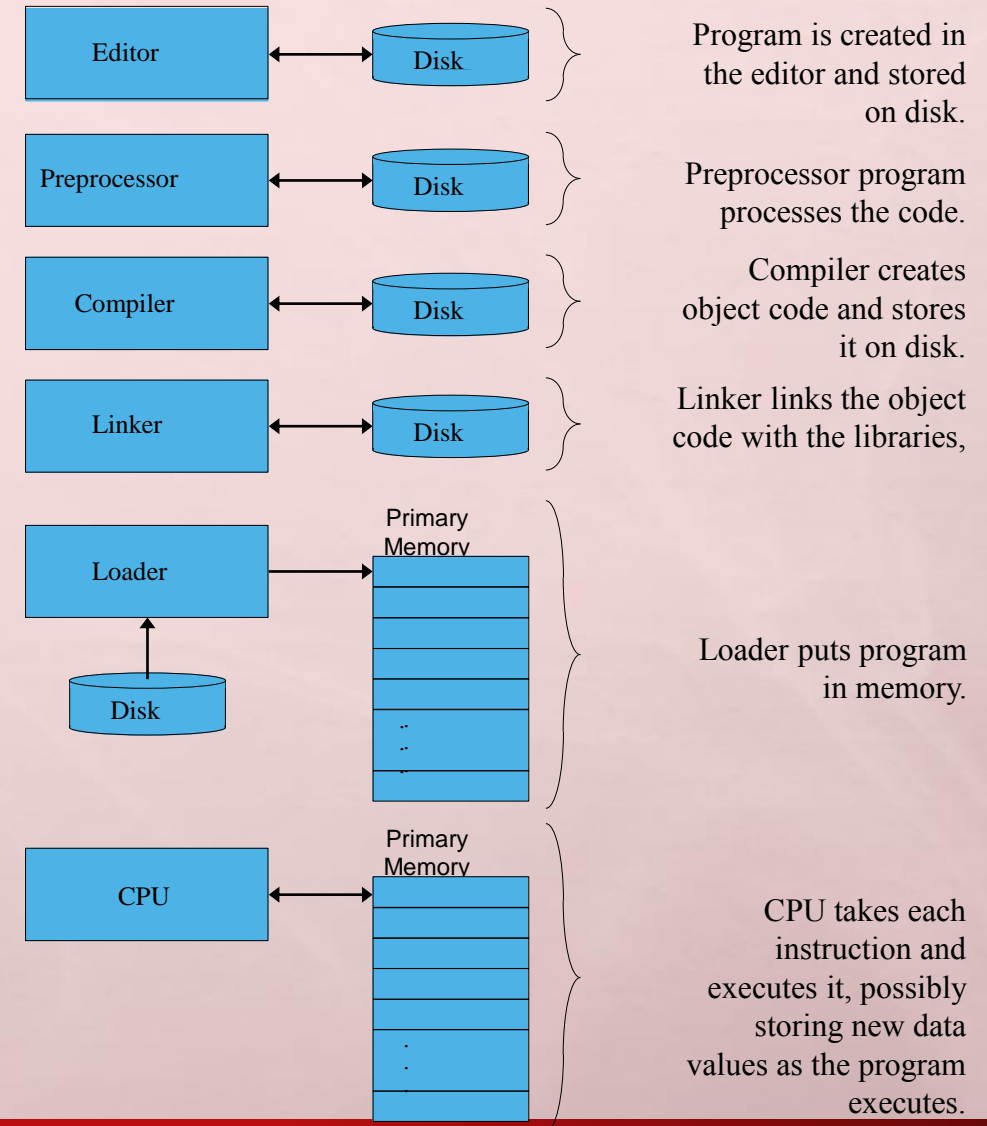
- Extension of C
- early 1980s: **bjarne stroustrup**
- For the last couple of decades, the C programming language has been widely accepted for all applications, and is perhaps the most powerful of structured programming languages. Now, C++ has the status of a structured programming language with object oriented programming (OOP). C++ has become quite popular due to the following reasons:
  1. It supports all features of both structured programming and OOP.
  2. C++ focuses on function and class templates for handling data types.



# 1.14 Basics of a Typical C++ Environment

## Phases of C++ Programs:



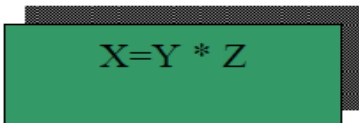
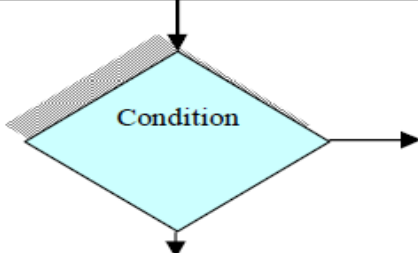
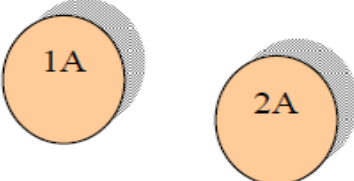
- Edit
- Preprocess
- Compile
- Link
- Load
- Execute



# Flowcharts

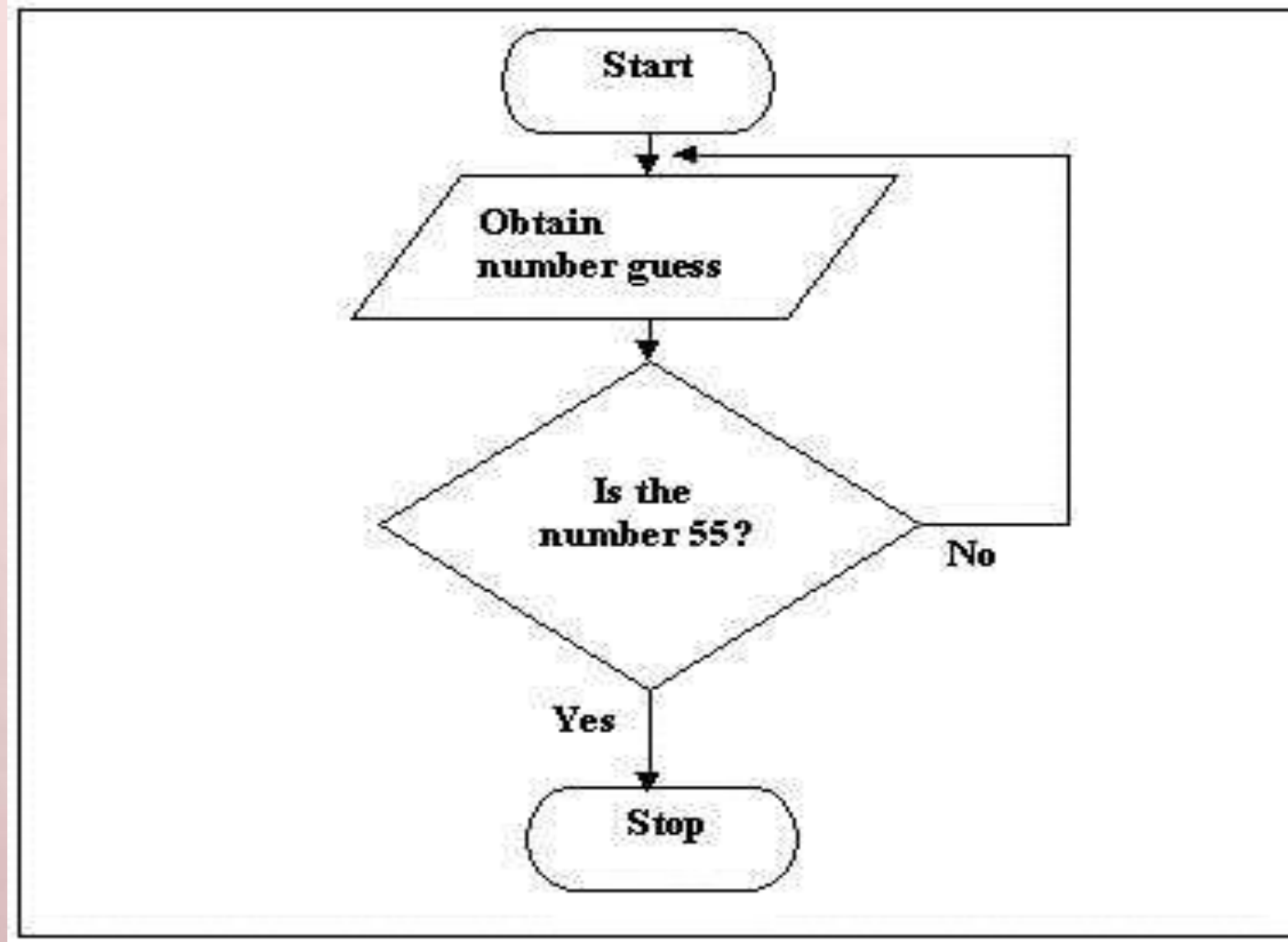
A flowchart is a graphical representation of an algorithm or of a portion of an algorithm .

Flowcharts are drawn using symbols.

	Start and Stop Symbols
	Input and Output Symbols
	Mathematical and logical processing symbol
	Decision making symbol
	Connector symbols



# Example:



## #Include preprocessor directive:

In the C++ programming language, the `#include` directive tells the preprocessor to insert the contents of another file into the source code at the point where the `#include` directive is found.

### Syntax

The syntax for the `#include` directive in the C language is:

`#include <header_file>`

OR

`#include "header_file"`

- `#include <iostream.h>` cin,cout,....
- `#include <math.h>` abs,sin,cos,tan,sqrt,...
- `#include <string.h>` strcat,strchr,strcpy,...
- `#include<graphics.h>` circle(xx,y,radius)
- `#include <stdio.h>` printf,scanf,gets(),....

## ***Variables :***

variable is a symbolic name for a memory location in which data can be stored and subsequently recalled.

### **Datatype id.1, id2, ...,idn;**

A variable defined by stating its type, followed by one or more spaces, followed by the one or more variable names separated by commas, then followed by semicolon. For example:

**Unsigned short int X;**

**Float Y;**

**char A, a, c;**

**int num=80;**

- variable can be declared anywhere in code .
- Variable names are case sensitive.

Type	Description
char	Typically a single octet(one byte). This is an integer type.
int	The most natural size of integer for the machine, two byte.
float	A single-precision floating point value , four bytes.
double	A double-precision floating point value, eight bytes.
void	Represents the absence of type.



# Keywords:

the keywords are also identifiers but cannot be user defined, since they are reserved words. All the keywords should be in lower case letters. Reserve words cannot be used as variable names or constant. The following words are reserved for use as keywords:

Break, case, char, cin, cout, delete, double ,else, enum, false, float, for, goto, if, int, long ,main, private, public, short, sizeof, switch, true, union, void.....

# Escape of sequences

<code>\n</code>	Newline. Position the screen cursor to the beginning of the next line.
<code>\t</code>	Horizontal tab. Move the screen cursor to the next tab stop.
<code>\r</code>	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.

**Endl : end line**

# *Comments :*

- A comment is a piece of descriptive text which explains some aspect of a program.
  - program comments are totally ignored by the compiler and are only intended for human readers.
- C++ provides two types of comment delimiters: -

\*Anything after // (until the end of the line on which it appears) is considered a comment.

**//This is first program**

\* Anything enclosed by the pair /\* and \*/ is considered a comment

**/\*this is first program**

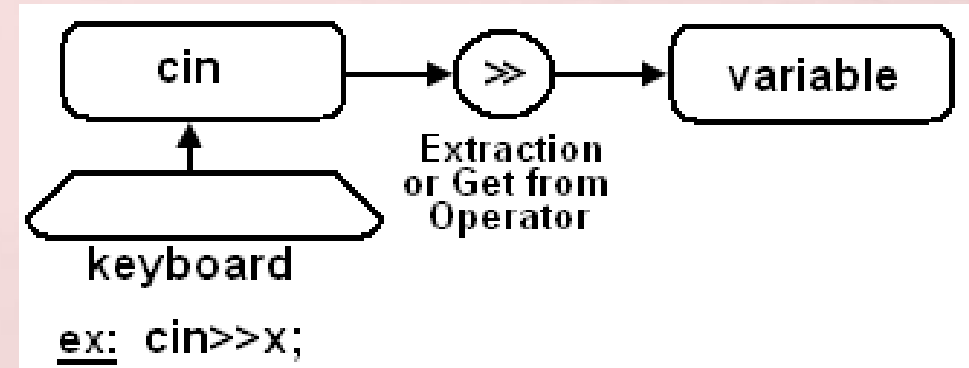
**Enjoy**

**\*/**

# Input/output:

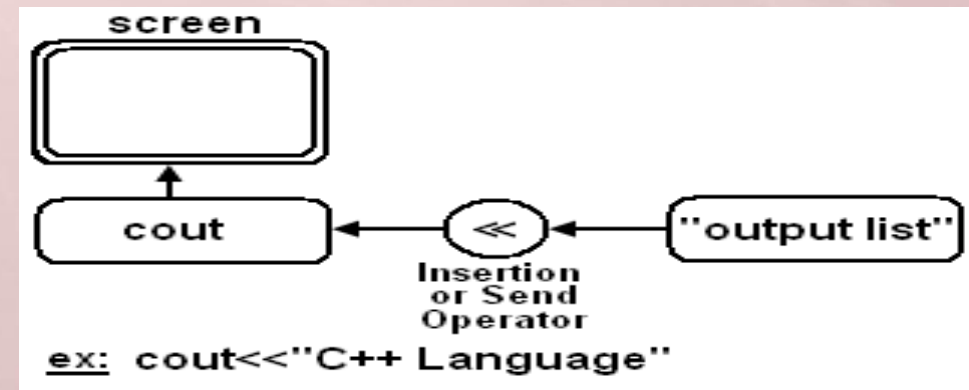
**cin:** standard input stream normally keyboard

**cin>>var.1>>var.2>>...>>var.N;**



**cout :**standard output stream normally computer screen

**cout<<var.1<<var2<<...<<var.N;**



# Example :

The following program reads three different inputs and outputs.

```
#Include<iostream.h>
```

```
Void main( )
```

```
{ int num=3;
```

```
Cout << "number="<<num<<"\n";
```

```
Char ch='a';
```

```
Cout << "character="<<ch<<"\n";
```

```
Float fa=-34.45;
```

```
{Cout<<"real number="<<fa<<"\n"; }
```

**Output:**

**Number=3**

**Character=a**

**Real number=34.45**



## Example :

Write a program that reads the radius of a circle, then computes and outputs its area.

```
#include<iostream.h>
void main( )
{
    const float pi = 3.14;
    int r; float c;
    cout << "enter the radius of circle:";
    cin>>r;
    cout<<endl;
    c = r * r * pi;
    cout << "the area of circle:" << c;
}
```

### Output:

```
enter the radius of circle: 5

the area of circle: 78.5
```

# C++ operators

## C++ operators

Arithmetic operators	
Assignment operators	
Comparison and logical operators	Relational, equality, logical
Bit wise logical operators	
Special operators	Unary, ternary, comma Scope, new&delete, other

# 1. Arithmetic operators : These operators require two variables to be evaluated:

+ addition      - subtraction      \* multiplication  
/ division      % modula (remainder of an integer division)

The division result are:

Integer / integer = integer ►  $39/7=5$

Integer / float = float ►  $39/7.0 =5.57$

float / integer = float ►  $39.0/7 =5.57$

float / float = float ►  $39.0/7.0=5.57$

## Example:

$X+y*X-Z$ , where  $X=5$ ,  $Y=6$ , and  $Z=8$ .

$5+(6*5)-8;$

$(5+30)-8;$

$35-8=27$

## 2. Assignment Operators: The operational assignment operator has the form:

**Variable = variable operator expression;**

The operational assignment operator can be written in the following form:

**Variable operator = expression**

**Ex:** `x=x+5; y=y*10;`

**Ex:** `x+=5; y*=10;`

<b>=</b>	Assign right hand side (RHS) value to the left hand side (LHS).
<b>+=</b>	Value of LHS var. will be added to the value of RHS and assign it back to the var. in LHS.
<b>-=</b>	Value of RHS var. will be subtracted to the value of LHS and assign it back to the var. in LHS.
<b>*=</b>	Value of LHS var. will be multiplied to the value of RHS and assign it back to the var. in LHS.
<b>/=</b>	Value of LHS var. will be divided to the value of RHS and assign it back to the var. in LHS.
<b>%=</b>	The remainder will be stored back to the LHS after integer division is carried out between the LHS var. and the RHS var.
<b>&gt;&gt;=</b>	Right shift and assign to the LHS.
<b>&lt;&lt;=</b>	Left shift and assign to the LHS.
<b>&amp;=</b>	Bitwise AND operation and assign to LHS
<b> =</b>	Bitwise OR operation and assign to LHS
<b>~=</b>	Bitwise complement operation and assign to LHS

**Example:** Rewrite the equivalent statements for the following examples, and find its results. Assume:  $X=2$  ,  $Y=3$  ,  $Z=4$  ,  $V=12$  ,  $C=8$ .

$$X += 5$$

$$X = X + 5$$

$$X = 7$$

$$Y -= 8$$

$$Y = Y - 8$$

$$Y = -5$$

$$Z *= 5$$

$$Z = ?$$

$$V /= 4$$

$$V = ?$$

$$C \% = 3$$

$$C = ?$$



**3. Comparision and logical operators:** It has three types relational operators, equality operators, and logical operators.

**(a) Relational operators:**

< less than,  
> greater than,  
<= less than or equal,  
>= greater than or equal,  
an expression that use relational operators return the value of **one** if the relational is **TRUE**, **ZERO** otherwise.

**Ex:**  $3 > 4 \rightarrow \text{false}$ ,  $6 <= 2 \rightarrow \text{false}$ ,  $10 > -32 \rightarrow \text{true}$ ,  $(23 * 7) >= (-67 + 89) \rightarrow \text{true}$

**(b) Equality operators:**

**==** equal to , **!=** not equal to

**Ex:**  $a=4$ ,  $b=6$ ,  $c=8$ .       $A==b \rightarrow \text{false}$ ,       $(a*b) != c \rightarrow \text{true}$ ,       $'s' == 'y' \rightarrow \text{false}$ .

**(C) logical operators:** the logical expression is constructed from relational expressions by the use of the logical operators **not(!)**, **And(&&)**, **or(||)**

AND (&&) Table:		
A	B	A && B
T	T	T
T	F	F
F	T	F
F	F	F

AND (&&) Table:		
A	B	A && B
1	1	1
1	0	0
0	1	0
0	0	0

OR (   ) Table:		
A	B	A     B
T	T	T
T	F	T
F	T	T
F	F	F

OR (   ) Table:		
A	B	A     B
1	1	1
1	0	1
0	1	1
0	0	0

NOT (!) Table:	
A	!A
T	F
F	T

NOT (!) Table:	
A	!A
1	0
0	1

## (d) Bitwise logical operator:

& bitwise AND,

^ bitwise exclusive OR(XOR),

| bitwise inclusive OR,

>> bitwise left shift,

<< bitwise right shift,

~ bitwise complement.

**Ex:**  $x=23$  (0001 0111)       $\sim x=132$  (1110 1000)

**Ex:**  $X=5, y=2 \rightarrow x\&y$  (0000) ,  $x|y$  (0111) ,  $x^y$  (0111)

## Example:

The following program computes different division operators.

```
#include<iostream.h>
void main( )
{
int x, y, z, r ;
x= 7 / 2;
cout << "x=" << x <<endl;
y=17/(-3);
cout << "y="<< y <<endl;
z=-17/3;
cout << "z="<< z <<endl;
r=-17/(-3);
cout << "r="<< r <<endl;
}
```

**Output:**

```
x= 3
y= -5
z= -5
r= 5
```

## Example:

```
#include<iostream.h>
void main( )
{
int y1, y2;
y1 = 8 % 3;
y2 = -17 % 3;
cout << "y1="<< y1 <<endl;
cout << "y2="<< y2 <<endl;
}
```

**Output:**

```
y1=2
y2=-2
```

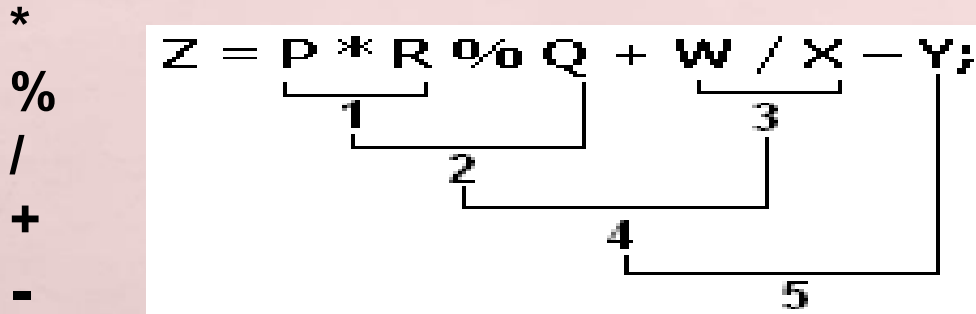


## Example :

State the order of evaluation for the following expression:

$Z = P * R \% Q + W / X - Y;$

*Solution:*



```
#include<iostream.h>
void main( )
{
    int Z, P, R, Q, W, X, Y;
    cout << "enter P:"; cin >> P;
    cout << "enter R:"; cin >> R;
    cout << "enter Q:"; cin >> Q;
    cout << "enter W:"; cin >> W;
    cout << "enter X:"; cin >> X;
    cout << "enter Y:"; cin >> Y;
    Z= P * R % Q + W / X - Y;
    cout << "the result="<< Z;
}
```

# The “math.h” Library:

The “math.h” library contains the common mathematical function used in the scientific equations.

Common function from math.h library:	
Mathematical Expression	C++ Expression
$e^n$	Exp(x)
Log(x)	Log10(x)
Ln(x)	Log(x)
Sin(x)	Sin(x)
$x^n$	Pow(x,n)
$\sqrt{x}$	Sqrt(x)

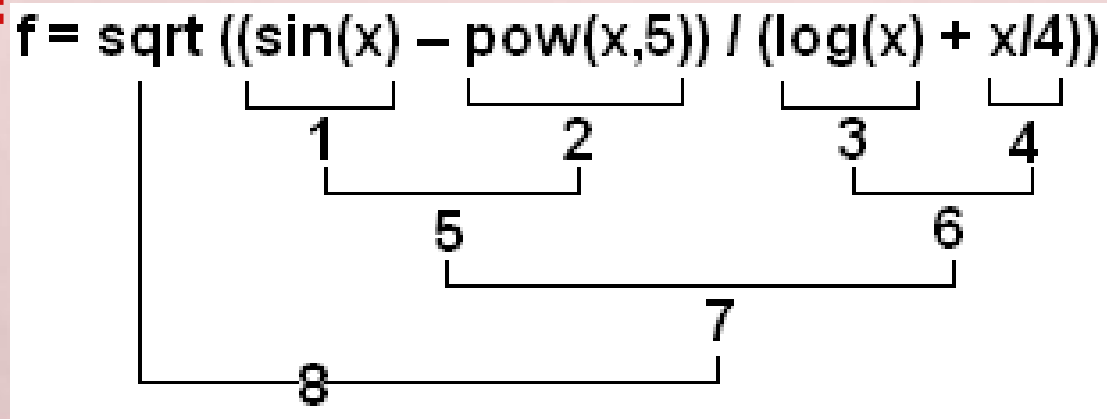
**Example:** Write the following equation as a C++ expression and state the order of evaluation of the binary operators:

$$f = \sqrt{\frac{\sin(x) - x^5}{\ln(x) + \frac{x}{4}}}$$

**Solution:**

`f = sqrt ((sin(x) – pow(x,5)) / (log(x) + x/4))`

**Order of evaluation:**



# Prefix & postfix

the ++ and - - operators can be written either before the variable (prefix notation) or after the variable (postfix notation) as in the following:

**prefix notation: ++ X** X is incremented before its value is taken or returned to current statement.

**Postfix notation: X ++** X is incremented after its value is taken or returned to current statement.

## The difference between the Prefix and Postfix notations:

### Prefix notation

```
int y;  
int x = 7;  
cout<< ++x <<endl;  
y=x;  
cout<<y;
```

### Output:

8  
8

### Postfix notation

```
int y;  
int x = 7;  
cout<< x++ <<endl;  
y=x;  
cout<<y;
```

### Output:

7  
8

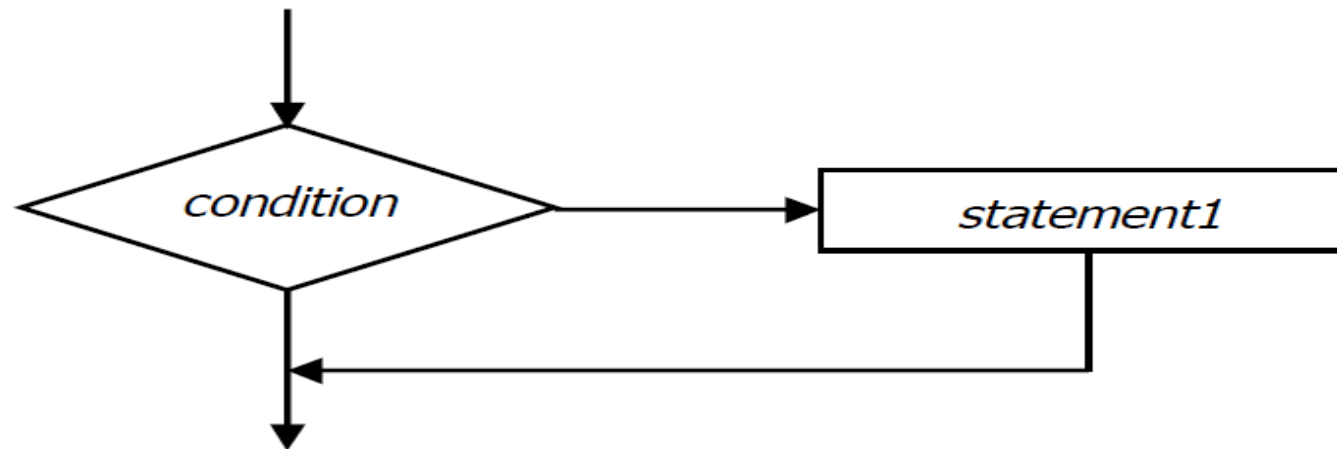
## Selection Statements:

Conditional expressions are mainly used for decision making. C++ provides multiple selection structures: if, if/else, else if, nested if and switch.

**The Single If Statement Structure:** The IF statement is used to express conditional expression. If the given condition is true then it will execute the statements; otherwise it will execute the optional statements.

### General Form of single-selection If statement:

```
if ( expression or condition ) statement1 ;
```





## Example :

Write a C++ program to read any two numbers and print the largest value of it:

```
#include<iostream.h>
void main( )
{
Float x,y;
Cout<<"Enter any two numbers\n";
Cin>>x>>y;
If (x>y)
Cout << "largest value is"<<x<<endl;
}
```

# The Single Block If Statement Structure :

The block IF statement are enclosed in ( { ) and ( } ) to group declaration and statements into a compound statement or a block. These blocks are always considered as a single statement. The structure is:

## General Form of single block selection If statement:

```
if ( expression or condition )  
{  
    statement1 ;  
    statement2 ;  
    statement3 ;  
}
```

## Example:

Write a C++ program to read a number and check if it's positive, if it's so print it, add it to a total, and decrement it by 2:

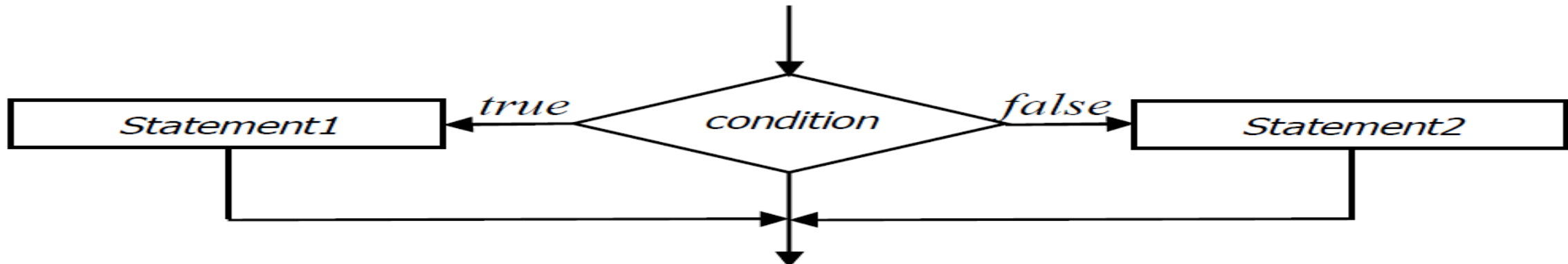
```
#include<iostream.h>
void main( )
{
    int num, total=0;
    cin >> num;
    if ( num >= 0 )
        { cout << num <<" is a positive";
          total += num; num = num - 2;
        }
}
```

## General Form of If/else statement:

```
if ( expression )  
    statement1 ;  
else statement2 ;
```

```
if ( expression )  
    { statements }  
else { statements }
```

## The If/else Statement Structure:



## Example :

Write a C++ program to read a student degree, and check if it's degree greater than or equal to 50, then print pass, otherwise print fail:

```
#include<iostream.h>
void main( )
{
int degree;
cin >> degree;
if (degree >= 50 )
cout << "pass";
else
{cout << "fail"; }
```

## Example:

Write a C++ program to read a number, and check if it's even or odd:

```
#include<iostream.h>
void main( )
{
int num;
cin >> num;
if ( num % 2 == 0 )
cout << "even";
else
cout << "odd";
}
```



## Else if Statements:

### General Form of else if statement:

```
if ( expression or condition 1 ) statement1 ;  
else if ( expression or condition 2 ) statement2 ;  
else if ( expression or condition 3 ) statement3 ;  
:  
else if ( expression or condition n ) statement-n ;  
else statement-e ;
```

## Example:

Write a C++ program to read a number, and print the day of the week:

```
#include<iostream.h>
void main( )
{
int day;
cin >> day;
if ( day == 1 ) cout << "Sunday";
else if (day == 2 ) cout << "Monday";
else if (day == 3 ) cout << "Tuesday";
else if (day == 4 ) cout << "Wednesday";
else if (day == 5 ) cout << "Thursday";
else if (day == 6 ) cout << "Friday";
else if (day == 7 ) cout << "Saturday";
else cout << "Invalid day number";
}
```

## Example:

Write C++ program to compute the value of Z according to the following equations:

$$Z = \begin{cases} x + 5 & : x < 0 \\ \cos(x) + 4 & : x = 0 \\ \sqrt{x} & : x > 0 \end{cases}$$

```
#include<iostream.h>
void main( )
{
int Z, x;
cout << "Enter X value \n";
cin >> x;
if ( x < 0 ) Z= x + 5;
else if ( x == 0 ) Z= cos(x) + 4;
else Z= sqrt(x);
cout << "Z is " << Z;
}
```

## Nested If Statements:

Some of the samples of **NESTED if-else** constructions are shown below:

```
If (exp.) { Statements }  
Else      { Statements }
```

```
If (exp.) {  
  If (exp.) {Statements}  
  Else      { Statements} }  
Else        {Statements}
```

```
If (exp.) {  
  If (exp.) {Statements}  
  Else      { Statements} }  
Else        {If (exp)  
              {Statements}  
              Else {Statement}  
              }  
}
```

**Example:** Write C++ program to find a largest value among three numbers:

```
#include<iostream.h>
void main( )
{
Float x,y,z;
Cout<<"Enter any two numbers\n";
Cin>>x>>y,z;
If (x>y)
{
    If (x>z)
        Cout << "largest value is"<<x<<endl;
    Else
        Cout << "largest value is"<<z<<endl;
}
Else If (y>z)
    Cout << "largest value is"<<y<<endl;
Else
    Cout << "largest value is"<<z<<endl;
}
```



# The Switch Selection Statement (Selector):

The switch statement is a special multi way decision maker that tests whether an expression matches one of the number of constant values, and braces accordingly.

## General Form of Switch Selection statement:

```
switch ( selector )  
{  
    case label1 : statement1 ; break;  
    case label2 : statement2 ; break;  
    case label3 : statement3 ; break;  
    :  
    case label-n : statement-n ; break;  
    default : statement-e ; break;  
}
```



# Example:

**Write C++ program to read integer number, and print the name of the day in a week:**

```
#include<iostream.h>
void main( )
{
int day;
cout << "Enter the number of the day \n";
cin >> day;
switch (day)
{
case 1: cout << "Sunday"; break;
case 2: cout << "Monday"; break;
case 3: cout << "Tuesday"; break;
case 4: cout << "Wednesday"; break;
case 5: cout << "Thursday"; break;
case 6: cout << "Friday"; break;
case 7: cout << "Saturday"; break;
default: cout << "Invalid day number"; break;
}}
```

## Example :

Write C++ program to read two integer numbers, and read the operation to perform on these numbers:

```
#include<iostream.h>
void main( )
{
int a, b;
char x;
cout << "Enter two numbers \n";
cin >> a >> b;
cout << "+ for addition \n";
cout << "- for subtraction \n";
cout << "* for multiplication \n";
cout << "/ for division \n";
cout << "enter your choice \n";
cin >> x;
```

```
switch ( x )
{
case '+': cout << a + b;
break;
case '-': cout << a - b;
break;
case '*': cout << a * b;
break;
case '/': cout << a / b;
break;
default: break;
}
}
```

## Loop Statements:

The loop statements are essential to construct systematic block styled programming. C++ provides three iteration structures: **while**, **do/while**, and **for**.

## While Repetition Structure:

### Example

```
i = 0;
while ( i < 10 )
{
    cout << i;
    i ++;
}
```

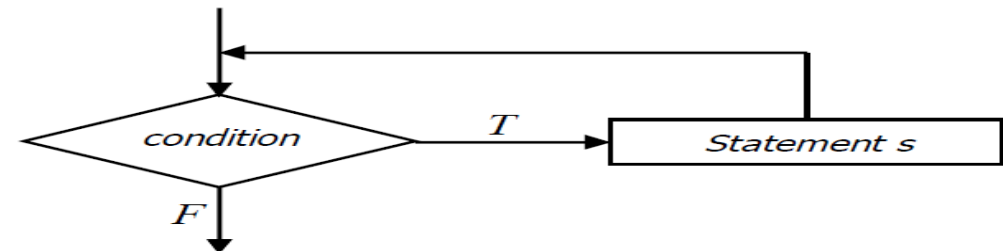
### Output:

**0123456789**

### General Form of While statement:

```
while ( condition )
    statement1 ;
```

```
while ( condition )
{
    statement1 ;
    statement2 ;
    :
    statement-n ;
}
```



## Example:

Write C++ program to find the summation of the following series: **sum = 1 + 3 + 5 + 7 + ... + 99**  
in other words: find the summation of the odd numbers, between 0 and 100

```
#include<iostream.h>
void main( )
{
int count = 1;
int sum = 0;
while ( count <= 99 )
{
sum = sum + count;
count = count + 2;
}
cout << "sum is: " << sum << endl;
}
```

## Example :

Write C++ program to find the summation of student's marks, and it's average, assume the student have 8 marks:

```
#include<iostream.h>
void main( )
{
int mark, i, sum = 0;
float av = 0;
i = 1;
while ( i <= 8 )
{
cout << "enter mark: ";
cin >> mark;
sum = sum + mark;
i++;
}
cout << "sum is: " << sum << endl;
av = sum / 8;
{cout << "average is: " << av; }
```

## Example :

Write C++ program that display the following board pattern:

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

```
#include<iostream.h>
void main( )
{
int row = 8, column;
while ( row-- > 0 )
{column = 8;
```

```
if ( row % 2 == 0 )
cout << " ";
while ( column-- > 0 )
cout << "*";
cout << '\n';
} }
```



# Do / While Statement:

## Example :

```
i = 0;  
do  
{  
cout << i;  
i ++;  
}  
while ( i < 10 )
```

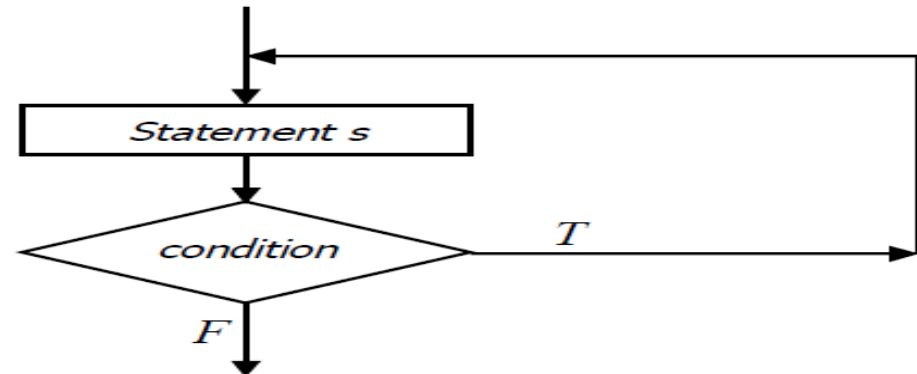
## Output:

0 1 2 3 4 5 6 7 8 9

### General Form of Do / While statement:

```
do  
    statement1 ;  
while ( condition );
```

```
do  
{  
    statement1 ;  
    statement2 ;  
    :  
    statement-n ;  
}  
while ( condition );
```



## Example:

Write C++ program to valid input checking, that accept the numbers between 50 ... 70 only:

```
#include<iostream.h>
```

```
void main( )
```

```
{
```

```
int accept = 1;
```

```
int x, low = 50, high = 70;
```

```
do
```

```
{
```

```
cout << "enter number: ";
```

```
cin >> x;
```

```
if ( x >= low && x <= high )
```


```
accept =1;
```

```
else
```

```
accept = 0;
```

```
}
```

```
while ( ! accept ); }
```



*while (accept == 1) or  
while (accept != 0)*

# Example:

Write C++ program to find the factorial of n:  $n! = n * n-1 * n-2 * n-3 * \dots * 2 * 1$

```
#include<iostream.h>
void main( )
{
int n, f = 1;
cout << "enter positive number: ";
cin >> n;
do
{
f = f * n;
n --;
}
while ( n > 1 );
cout << "factorial is: " << f;
}
```

## Example:

Write C++ program to find the summation of even numbers

```
#include<iostream.h>
void main( )
{
int max,sum,digit;
digit=2;
cout << "enter a number: ";
cin >> max;
sum=0;
do
{
Sum=sum+digit;
Digit+=2;
}
while ( digit<=max );
cout << "2+4+...="<<max<<"sum="<<sum<<endl; }
```

# For Statement:

## Example :

```
for ( i = 0; i < 10; i ++ )  
cout << i;
```

## Output:

0 1 2 3 4 5 6 7 8 9


### General Form of For statement:

```
for ( initialization ; continuation condition ; update )  
    statement1 ;
```

```
for ( initialization ; continuation condition ; update )  
{  
    statement1 ;  
    statement2 ;  
    :  
}
```

## Example:

Write C++ program to find the factorial of n (*using for statement*):  $n! = n * n-1 * n-2 * n-3 * \dots * 2 * 1$

```
#include<iostream.h>
void main( )
{
int n, f = 1;
cout << "enter positive number: ";
cin >> n;
for ( int i = 2; i <= n; i ++ )  for ( int i = n; i >=2 n; i -- )

f = f * i;
cout << "factorial is: " << f;
}
```



# Example:

Write C++ program to the result of the following:

$$\sum_{i=1}^{20} a_i^2$$

```
#include<iostream.h>
void main( )
{
int sum = 0;
for ( int i = 1; i <= 20; i ++ )
sum = sum + ( i * i );
cout << "The sum is: " << sum;
}
```

## Example:

Write C++ program to read 10 integer numbers, and find the sum of positive number only:

```
#include<iostream.h>
void main( )
{
int num, sum = 0;
for ( int i = 1; i <= 10; i ++ )
{
cout << "enter your number: ";
cin >> num;
if ( num > 0 )
sum = sum + num;
}
cout << "The sum is: " << sum;
}
```

## Example:

Write C++ program to print the following series: 1, 2, 4, 8, 16, 32, 64

```
#include<iostream.h>
void main( )
{
int x;
for ( x = 1; x < 65; x *= 2 )
cout << x <<" ";
}
```

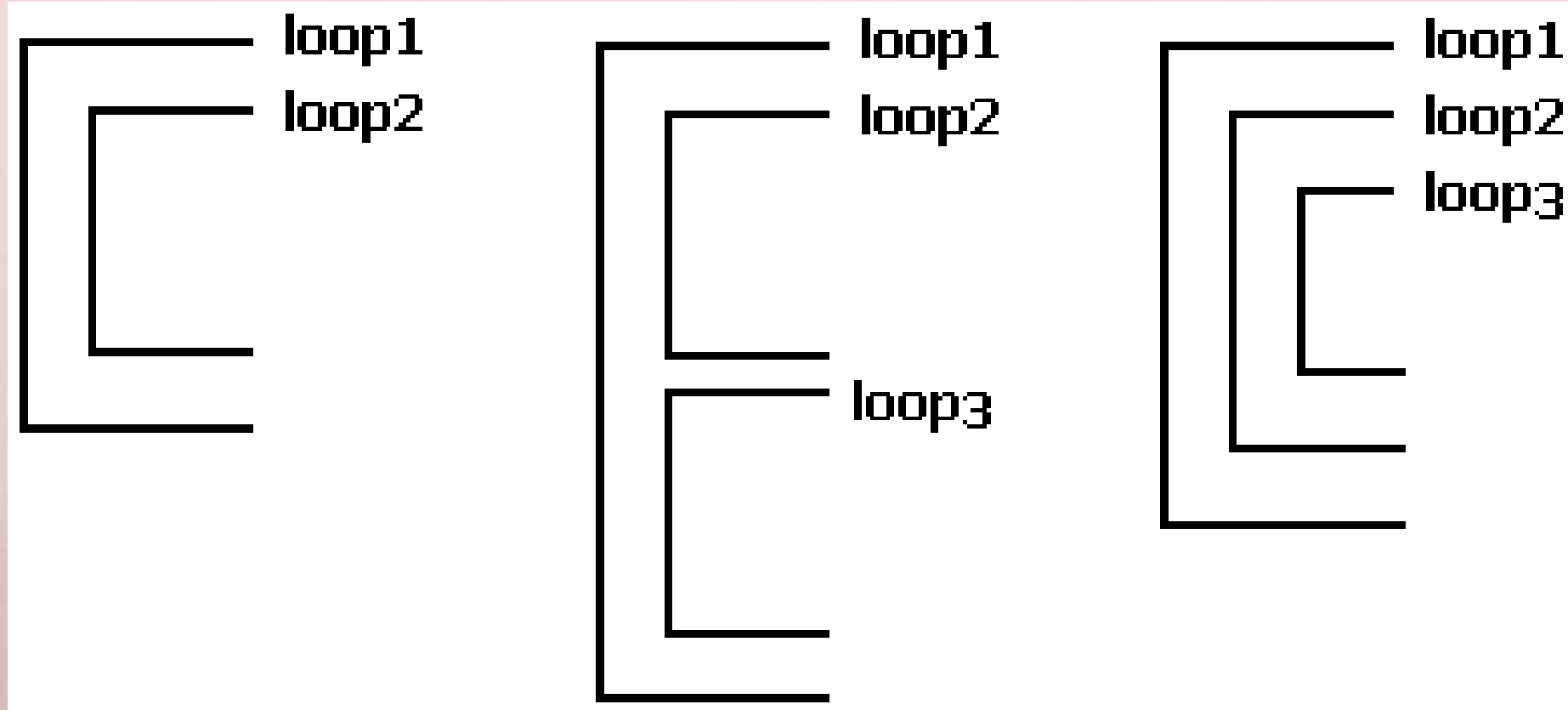
## Example: Write C++ program to print the following:

```
#include<iostream.h>
void main( )
{
int x;
for ( x = 1; x < 7; ++ x )
cout << x <<"\t" << 11 - x << endl;
}
```

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

# Nested Loops:

We can put loops one inside another to solve a certain programming problems. Loops may be nested as follows:



# Example:

+Write C++ program to print the following figure:

```
+ +
+ + +
+ + + +
+ + + + +
+ + + + + +
+ + + + + + +
+ + + + + + + +
+ + + + + + + + +
+ + + + + + + + + +
```

```
#Include<iostream.H>
```

```
Void main( )
```

```
{
```

```
Int i, j;
```

```
For ( i = 1; i <= 10; i ++ )
```

```
{ For ( j = 1; j <= i; j ++ )
```

```
Cout << " + ";
```

```
Cout << "\n";} }
```

## Example:

What is the output of the following C++ program??????????????

```
#include<iostream.h>
void main( )
{
int i, j, k;
for ( i = 1; i <= 2; i ++ )
{
for ( j = 1; j <= 3; j ++ )
{
for ( k = 1; k <= 4; k ++ )
cout << " + ";
cout << "\n";
}
cout << "\n";
}
}
```



## Using For Statement

## Using While Statement

## Using Do/While Statement

*Q1: Find the summation of the numbers between 1 and 100.*

```
for( i=1 ; i<=100 ; i++ )  
    s = s + i;
```

```
i = 1;  
while ( i <= 100)  
{  
    s = s + i;  
    i++;  
}
```

```
i = 1;  
do  
{  
    s = s + i;  
    i++;  
}  
while ( i <= 100 );
```

*Q2: Find the factorial of n.*

```
cin >> n;  
for( i=2 ; i<=n ; i++ )  
    f = f * i;
```

```
cin >> n;  
i = 2;  
while ( i <= n)  
{  
    f = f * i;  
    i++;  
}
```

```
cin >> n;  
i = 2;  
do  
{  
    f = f * i;  
    i++;  
}  
while ( i <= n);
```

# Functions:

A function is a set of statements designed to accomplish a particular task. Experience has shown that the best way to develop and maintain a large program is to construct it from smaller pieces or (modules). Modules in C++ are called functions. Functions are very useful to read, write, debug and modify complex programs. They can also be easily incorporated in the main program. In C++, the main() itself is a function that means the main function is invoking the other functions to perform various tasks.

## Defining a Function :

### General Form of Function:

```
return-type function-name ( parameters-list )  
{  
    (body of function)  
    statement1 ;  
    statement2 ;  
    :  
    statement-n ;  
    (return something)  
}
```

### Example:

```
void printmessage ( )  
{  
    cout << "University of Technology";  
}  
void main ( )  
{  
    printmessage( ); }  
}
```

## Example:

Write C++ program using function to calculate the average of two numbers entered by the user in the main program:

```
#include<iostream.h>
float aver (int x1, int x2)
{
float z;
z = ( x1 + x2) / 2.0;
return ( z);
}
void main( )
{
float x;
int num1,num2;
cout << "Enter 2 positive number \n";
cin >> num1 >> num2;
x = aver (num1, num2);
cout << x; }
```

# Example:

Write C++ program, using function, to find the summation of the following series:

$$\sum_{i=1}^n i^2 = 1^2 + 2^2 + 3^2 + \dots + n^2$$

```
#include<iostream.h>
int summation ( int x)
{
int i = 1, sum = 0;
while ( i <= x )
{
sum += i * i ;
i++;
}
return (sum);
}
```

```
void main ( )
{
int n ,s;
cout << "enter positive number";
cin >> n;
s = summation ( n );
cout << "sum is: " << s << endl;
}
```

## Example :

Write a function to find the largest integer among three integers entered by the user in the main function.

```
#include <iostream.h>
int max(int y1, int y2, int y3)
{
    int big;
    big=y1;
    if (y2>big) big=y2;
    if (y3>big) big=y3;
    return (big);
}
void main( )
{
    int largest,x1,x2,x3;
    cout<<"Enter 3 integer numbers:";
    cin>>x1>>x2>>x3;
    largest=max(x1,x2,x3);
    cout<<largest; }
```

# Arrays:

An array is a consecutive group of homogeneous memory locations. Each element (location) can be referred to using the array name along with an integer that denotes the relative position of that element within the array. The data items grouped in an array can be simple types like **int** or **float**, or can be user-defined types like **structures** and **objects**.

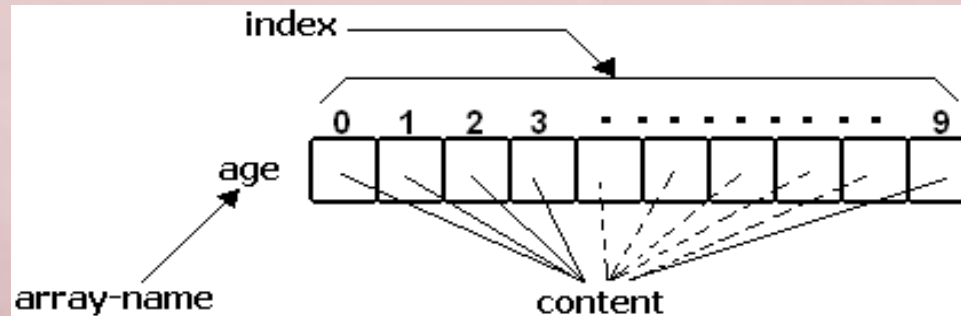
## Array of One Dimension:

It is a single variable specifies each array element. The declaration of one dimensional arrays is:

**data-type Array-name [ size ];**

### Examples:

```
int age [10];  
int num [30];  
float degree[5];  
char a [15];
```





# Initializing Array Elements:

- The first element of array age:

```
age [0] = 18;
```

- The last element of array age:

```
age [9] = 19;
```

- All elements of array age:

```
age [9] = { 18, 17, 18 ,18 ,19, 20 ,17, 18 ,19 };
```

## Accessing Array Elements:

We access each array element by written name of array, followed by brackets delimiting a variable (or constant) in the brackets which are called the array index.

- Accessing the first element of array num to variable x:

```
x = num [0];
```

- Accessing the last element of array num to variable y:

```
y = num [9];
```

- cout << num [0] + num [9];

- num [0] = num [1] + num[2];

- num [7] = num [7] + 3; num [7] += 3;

## Example:

Write C++ program to read 5 numbers and print it in reverse order:

```
#include<iostream.h>
void main( )
{
int a [5];
cout << "Enter 5 numbers \n";
for ( int i =0; i <5; i++ )
{
cout << i << ": ";
cin >> a [ i ];
cout << "\n";
}
cout << "The reverse order is: \n";
for ( i =4; i >=0; i-- )
cout << i << ": " << a [ i ] << endl;
}
```

# Example:

Write C++ program, to find the summation of array elements:

```
#include<iostream.h>
void main ( )
{
int const L = 10;
int a [L];
int sum = 0;
cout << "enter 10 numbers \n";
for ( int i =0; i <L; i++ )
{
cout << "enter value " << i << ": ";
cin >> a [ i ];
sum += a [ i ];
}
cout << "sum is: " << sum << endl;
}
```

## Example:

Write C++ program, to find the minimum value in array of 8 numbers:

```
#include<iostream.h>
void main ( )
{ int n = 8; int a [ ] = { 18, 25, 36, 44, 12, 60, 75, 89 };
int min = a [ 0 ];
for ( int i = 0; i < n; i++ )
if ( a [ i ] < min ) min = a [ i ];
cout << "The minimum number in array is: " << min;
}
```

## Example:

Write C++ program, using function, to find (search) X value in array, and return the index of it's location:

```
#include<iostream.h>
int search( int a[ ], int y)
{
    int i= 0;
    while ( a [ i ] != y )
        i++;
    return ( i );
}
void main ( )
{
    int X, f;
    int a [ 10 ] = { 18, 25, 36, 44, 12, 60, 75, 89, 10, 50 };
    cout << "enter value to find it: ";
    cin >> X;
    f= search (a, X);
    cout << "the value " << X << " is found in location "<< f; }
```

# Array of Two Dimension:

Arrays can have higher dimension. There can be arrays of two dimension which is array of arrays. It is accessed with two index. Also there can be arrays of dimension higher than two.

## General Form of 2D-Array:

*data-type* **Array-name** [ *Row-size* ] [ *Col-size* ];

### Example:

```
int a [10] [10];  
int num [3] [4];
```

A diagram illustrating a 2D array structure. It shows a 3x4 grid of cells. The columns are labeled 0, 1, 2, 3 at the top, with a bracket above them labeled 'col'. The rows are labeled 0, 1, 2 on the left, with a bracket to their left labeled 'row'. Each cell contains a pair of indices separated by a comma, representing its row and column coordinates. A diagonal line from the top-left to the bottom-right is labeled 'index' at its end.

	0	1	2	3
0	0,0	0,1	0,2	0,3
1	1,0	1,1	1,2	1,3
2	2,0	2,1	2,2	2,3



# Initializing 2D-Array Elements:

- The first element of array age:  
a [2] [3] = { {1, 2, 3} , {4, 5, 6} };

## Example:

Write C++ program, to read 15 numbers, 5 numbers per row, the print them:

```
#include<iostream.h>
```

```
void main ( )
```

```
{
```

```
int a [ 3 ] [ 5 ];
```

```
int i , j;
```

```
for ( i = 0 ; i < 3; i++ )
```

```
for ( j = 0 ; j < 5; j++ )
```

```
cin >> a [ i ] [ j ];
```

```
for ( i = 0 ; i < 3; i++ )
```

```
{
```

```
for ( j = 0 ; j < 5; j++ )
```

```
cout << a [ i ] [ j ]; cout << endl; } }
```

# Example:

Write C++ program, to read 4\*4 2D-array, then find the summation of the array elements, finally print these elements:

```
#include<iostream.h>
void main ( )
{
int a [ 4 ] [ 4 ];
int i , j, sum = 0;
for ( i = 0 ; i < 4; i++ )
for ( j = 0 ; j < 4; j++ )
cin >> a [ i ] [ j ];
for ( i = 0 ; i < 4; i++ )
for ( j = 0 ; j < 4; j++ )
sum += a [ i ] [ j ];
cout << "summation is: " << sum << endl;
for ( i = 0 ; i < 4; i++ )
{
for ( j = 0 ; j < 4; j++ )
cout << a [ i ] [ j ]; cout << endl; } }
```

## Example:

Write C++ program, to read 3\*4 2D-array, then find the summation of each row:

```
#include<iostream.h>
```

```
void main ( )
```

```
{
```

```
int a [ 3 ] [ 4 ];
```

```
int i , j, sum = 0;
```

```
for ( i = 0 ; i < 3; i++ )
```

```
for ( j = 0 ; j < 4; j++ )
```

```
cin >> a [ i ] [ j ];
```

```
for ( i = 0 ; i < 3; i++ )
```

```
{
```

```
sum = 0;
```

```
for ( j = 0 ; j < 4; j++ )
```

```
sum += a [ i ] [ j ];
```

```
cout << "summation of row " << i << " is: " << sum << endl;
```

```
}
```

```
}
```

## Example:

Write C++ program, to read 3\*4 2D-array, then replace each value equal 5 with 0:

```
#include<iostream.h>
void main ( )
{
int a [ 3 ] [ 4 ];
int i , j;
for ( i = 0 ; i < 3; i++ )
for ( j = 0 ; j < 4; j++ )
cin >> a [ i ] [ j ];
for ( i = 0 ; i < 3; i++ )
for ( j = 0 ; j < 4; j++ )
if ( a [ i ] [ j ] == 5 )      a [ i ] [ j ] = 0;
for ( i = 0 ; i < 3; i++ )
{
for ( j = 0 ; j < 4; j++ )
cout << a [ i ] [ j ];
cout << endl; } }
```

**Example:** Write C++ program, to convert 2D-array into 1D-array:

```
#Include<iostream.h>
```

```
Void main ( )
```

```
{
```

```
Int a [ 3 ] [ 4 ];
```

```
Int b [ 12 ];
```

```
Int i , j, k = 0;
```

```
For ( i = 0 ; i < 3; i++ )
```

```
For ( j = 0 ; j < 4; j++ )
```

```
Cin >> a [ i ] [ j ];
```

```
For ( i = 0 ; i < 3; i++ )
```

```
For ( j = 0 ; j < 4; j++ )
```

```
{
```

```
B [ k ] = a [ i ] [ j ];
```

```
K++;
```

```
}
```

```
For ( i = 0 ; i < k; i++ )
```

```
Cout << b [ i ]; } }
```

**Example:** Write C++ program, to replace each element in the main diameter (diagonal) with zero:

```
#include<iostream.h>
void main ( )
{
int a [ 3 ] [ 3 ];
int i , j;
for ( i = 0 ; i < 3; i++ )
for ( j = 0 ; j < 3; j++ )
cin >> a [ i ] [ j ];
for ( i = 0 ; i < 3; i++ )
for ( j = 0 ; j < 3; j++ )
if ( i == j )    a [ i ] [ j ] = 0;
for ( i = 0 ; i < 3; i++ )
{
for ( j = 0 ; j < 3; j++ )
cout << a [ i ] [ j ]; cout << endl;
} }

```

0,0		
	1,1	
		2,2
i = j		



# String:

In C++ strings of characters are implemented as an array of characters. In addition a special null character, represented by `\0`, is appended to the end of string to indicate the end of the string.

## General Form of String:

```
char String-name [ size ];
```

## Examples:

```
char name [10] = "Mazin Alaa";
```

```
'M', 'a', 'z', 'i', 'n', ' ', 'A', 'l', 'a', 'a', '\0'
```

```
char str [ ] = "ABCD";
```


```
'A', 'B', 'C', 'D', '\0'
```

```
str [0] : 'A'
```

```
str [1] : 'B'
```

```
str [2] : 'C'
```

```
str [3] : 'D'
```

```
str [4] : '\0'  null
```

## Example :

Write C++ program to print string, then print it character by character:

```
#include<iostream.h>
void main( )
{
char s [ ] = "ABCD";
cout << "Your String is: " << s << endl;
for ( int i =0; i < 5; i++ )
cout << "S[" << i << "] is: " << s [ i ] << endl;
}
```

### Output is:

Your String is: ABCD

S[0] is: A

S[1] is: B

S[2] is: C

S[3] is: D

S[4] is:

# Example:

Write C++ program to convert each lower case letter to upper case letter:

```
#include<iostream.h>
#include<ctype.h>
void main( )
{
char s [ ] = "abcd";
cout << s << endl;
for ( int i =0; i < 4; i++ )
s [i] = char(toupper (s[i] ));
cout << s;
}
```

# Member Function of String:

The string library has many member functions of string like:

Member Function	Functionality	Example
<b>strlen ( string )</b>	Return the length of the string	<pre>a [ ] = "abcd"; cout &lt;&lt; strlen ( a );</pre>
<b>strcpy ( string2, string1 )</b>	Copy the content of the 1 <sup>st</sup> string into the 2 <sup>nd</sup> string	<pre>char a[ ]= "abcd" , b[ ]="  "; strcpy ( b , a ); cout &lt;&lt; a &lt;&lt; b;</pre>
<b>strcat ( string1, string2 )</b>	Append the content of the 2 <sup>nd</sup> string into the end of the 1 <sup>st</sup> string	<pre>char a[ ]= "abcd" , b[ ]="1234"; strcat ( a , b ); cout &lt;&lt; a &lt;&lt; b; abcd1234 1234</pre>
<b>strcmp ( string1, string2 )</b>	<p>Return 0 if the 1<sup>st</sup> string is equal to the 2<sup>nd</sup> string.</p> <p>Return a Positive number if the 1<sup>st</sup> string is greater than the 2<sup>nd</sup> string.</p> <p>Return a Negative number if the 1<sup>st</sup> string is smaller than the 2<sup>nd</sup> string.</p>	<pre>char a[ ]= "abcd" , b[ ]="abcd"; cout &lt;&lt; strcmp ( a , b );</pre> <p><b>0</b>    if a == b <b>+</b>    if a &gt; b <b>-</b>    if a &lt; b</p>

# Data structure:

**data** : set of information & raw material.

**Structure** : any information in a well design manner.

Data structure can be defined in two parts:

**1) linear data structure:** accessing of data values are made in a sequential fashion

**EX:** Array ,link list, stack, queue.

**2) non- linear data structure:** accessing of data values are made in a non linear fashion or non continue fashion **EX:Tree, graph**

## **data structure applications:**

- 1) design & controlling os.
- 2) design & controlling the file management.
- 3) design & controlling process management.
- 4) design & controlling memory management.
- 5) design & controlling computer language.
- 6) animation & video game.



# Stack (data structure):

stack is a temporary abstract data type and data structure based on the principle of last in first out (LIFO). It has two operations:

1) push operation

2) pop operation

## Stack algorithms :

**A- push algorithm :** let int s[n] is array represent a stack

1- is stack over flow?

If top=n then cout<<"stack is full" and exit'

2- increment top pointer

Top =top+1

3- push new element and exit

S[top]=x and exit

**B- pop algorithm :**

1- is stack under flow?

If top=0 then cout<<"stack is empty" and exit'

2- delete the element

x =s[top]

3- decrement top pointer

Top=top-1



# Example:

```
#include <iostream.h>
#define max 3
int stack[max+1];
int top =0;
void push(int a)
{
    if (top==3)
        cout<<"stack is full";
    else
    {top=top+1;
    stack[top]=a;
    cout <<top<<"stack"<<stack[top]<<"\n";}
}
void pop()
{int x;
  if (top ==0)
    cout <<"stack is empty";
  else
  {
      x=stack[top];
      top--;
      cout<<"\n the element deleted"<<x;
  }}
void main()
{push(10);
push(20);
push(90);
push(40);
pop();
pop();
pop();
pop();
}
```

# queue (data structure):

queue is linear D.S. in which insertion and deletion operation performed on different ends. It's a FIFO (first in first out ) list . The first end called Rear and later is called Front . element insert from rear –end and delete from front end .

## Queue algorithms :

**A- insert algorithm :** let int Q[n] is array represent a queue

1- is queue over flow?

**If** Rear=n **then** cout<<“Queue is over flow” and exit'

2- increment Rear pointer

Rear =Rear+1

3- insert new element

Q[Rear]=x

4- is Front pointer properly set ?

**If** Front =0 **then** set Front to 1 and exit

**B- delete algorithm :**

1- is queue under flow?

**If** Front=0 **then** cout<<“queue is under flow” and exit

2- delete the first element

x =Q[Front]

3- is queue empty

**If** Front =Rear **then** Front =Rear =0 and exit

4- increment Front pointer

Front=Front+1 and exit

# Example:

```
#include <iostream.h>
#define max 3
int queue[max];
int front=0;int rear =0;
void insert()
{
    int a;
    cout<<"\nenter any element:";
    cin>>a;
    if (rear==max) cout<<"\nqueue is full";
    else
    {++rear;
    queue[rear]=a;}
    if (front==0)
        front++;
}
void del()
{int x;
    if (front ==0)
        cout <<"\nqueue is empty you can not delete elements";
    else
    {cout<<"deleted element is";
        x=queue[front];

        cout<<x;
        if (front ==rear){front=0;rear=0;}
        else
            front++;
    }
}
```

```
void main()
{int i ,choise;
for(i=0;i<=10;++i)
{    cout<<"\n\t\t***** main menue*****\n";
cout<<"\t[1] insert\n";
cout<<"\t[2] delete\n";
cout<<"\t[3] exit\n";
cout<<"\n enter your choise:";
cin>>choise;
switch(choise)
{
case 1 :insert();break;
case 2 :del();break;
case 3: break;
default:cout <<"\ninvalid choise";
}
//if (choise==3)
//break;
} //end for;
} //enf program;
```

# Structures:

Structures are typically used to group several data items together to form a single entity. It is a collection of variables used to group variables into a single record. Thus a structure (the keyword **struct** is used in C++) is used. Keyword struct is a data-type, like the following C++ data-types ( int, float, char, etc... ). This is unlike the array, which all the variables must be the same type. The data items in a structure are called the members of the structure.

## General Form of Structure:

```
struct struct-name
{
    variables ...
};
```

# The Three Ways for Declare the Structure:

#Include <iostream.h>

Struct data

{

Char name;

Int age;

};

Void main()

{

Struct data student;

.....

..... }

struct data

{

char name;

int age;

} student;

typedef struct

{

char name;

int age;

} student;

To access elements in a structure, use a record selector ( . ).

student . name="ahmed";

student . age=20;



This example uses parts inventory to demonstrate structures.

```
#include<iostream.h>
Struct part // specify a structure
{
int model_no;
int part_no;
Float cost;
};
Void main()
{
part p1; // define a structure variable.
p1.model_no=6244;
p1.part_no=373;
p1.cost=217.55;
cout<<"\n model"<<p1.model_no;
cout<<" part"<<p1.part_no;
cout<<" cost"<<p1.cost;
}
```



## Example:

Write C++ program to find the distance in English system. 1 foot=12 inch.

```
#include<iostream.h>
```

```
struct distance
```

```
{
```

```
int feet;
```

```
float inches;
```

```
};
```

```
Void main ()
```

```
{
```

```
distance d1,d3;
```

```
distance d2={11,6.25};
```

```
cout<<"\n Enter feet:";
```

```
cin>>d1.feet;
```

```
cout<<"\n Enter inches:";
```

```
cin>>d1.inches;
```

```
d3.inches=d1.inches+d2.inches;
```

```
d3.feet=0;
```

```
if (d3.inches >=12.0)
```

```
{
```

```
d3.inches -=12.0;
```

```
d3.feet ++;
```

```
}
```

```
d3.feet +=d1.feet + d2.feet;
```

```
cout<<d1.feet<<"\'-“<<d1.inches<<"\'"+";
```

```
cout<<d2.feet<<"\'-“<<d2.inches<<"\'"=";
```

```
cout<<d3.feet<<"\'-“<<d1.inches<<"\'"\n";
```

```
}
```

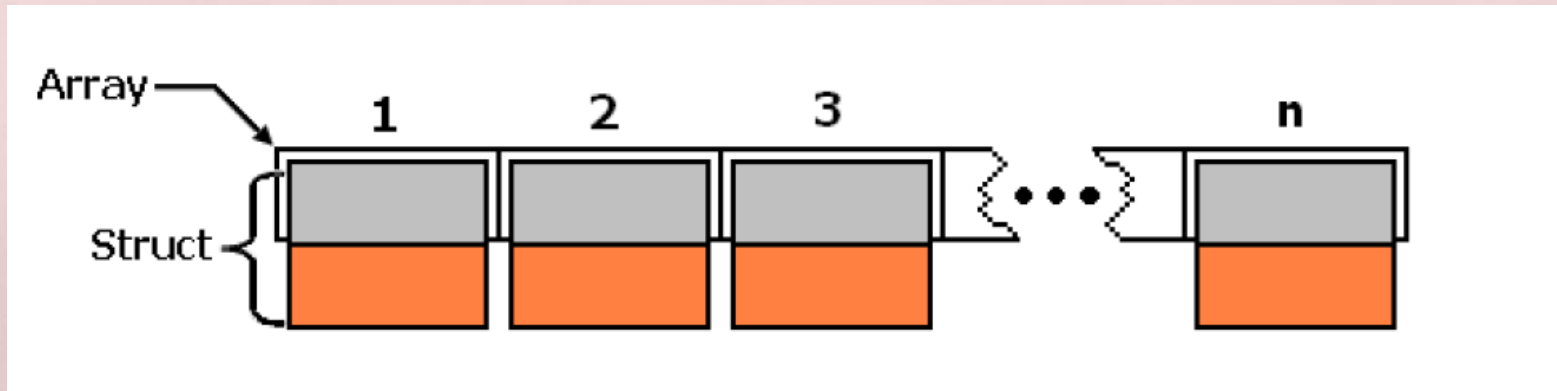
## Example:

Write C++ program to find the area of the room in English system.

```
#include<iostream.h>
struct distance
{
int feet;
float inches;
};
struct room
{
distance length;
distance width;
};
Void main ()
{
    room dining;
    dining.length.feet=13;
    dining.length.inches=6.5;
    dining.width.feet=10;
    dining.width.inches=0.0;
    float L=dining.length.feet+dining.length.inches/12;
    float W=dining.width.feet+dining.width.inches/12;
    cout<<"\n Dining room area is"<<L*W<<"Square feet";
}
```

# Array of Structures:

The **struct** is a data-type. So we can define an array as an array of struct, like define an array as an array of **int**, or of any other C++ data-types.



## Example:

Write a C++ Program, using structure type, to read name and age for ten students.

```
#include<iostream.h>
```

```
typedef struct
```

```
{  
char name[20]; //Or name[10]
```

```
int age;
```

```
} student;
```

```
void main ( )
```

```
{  
student array [10];  
for ( i = 0 ; i < 10 ; i++ )
```

```
{  
cin >> array [i] . name;  
cin >> array [i] . age;  
}
```

```
for ( i = 0 ; i < 10 ; i++ )
```

```
{  
cout << array[i] . name << endl;  
cout << array[i] . age;  
}  
}
```

# Functions and Structures:

A structure can be passed to a function as a single variable. The scope of a structure declaration should be an external storage class whenever a function in the main program is using a structure data types. The field or member data should be same throughout the program either in the main or in a function.

## Example :

Write C++ program to display the contents of a structure using function definition.

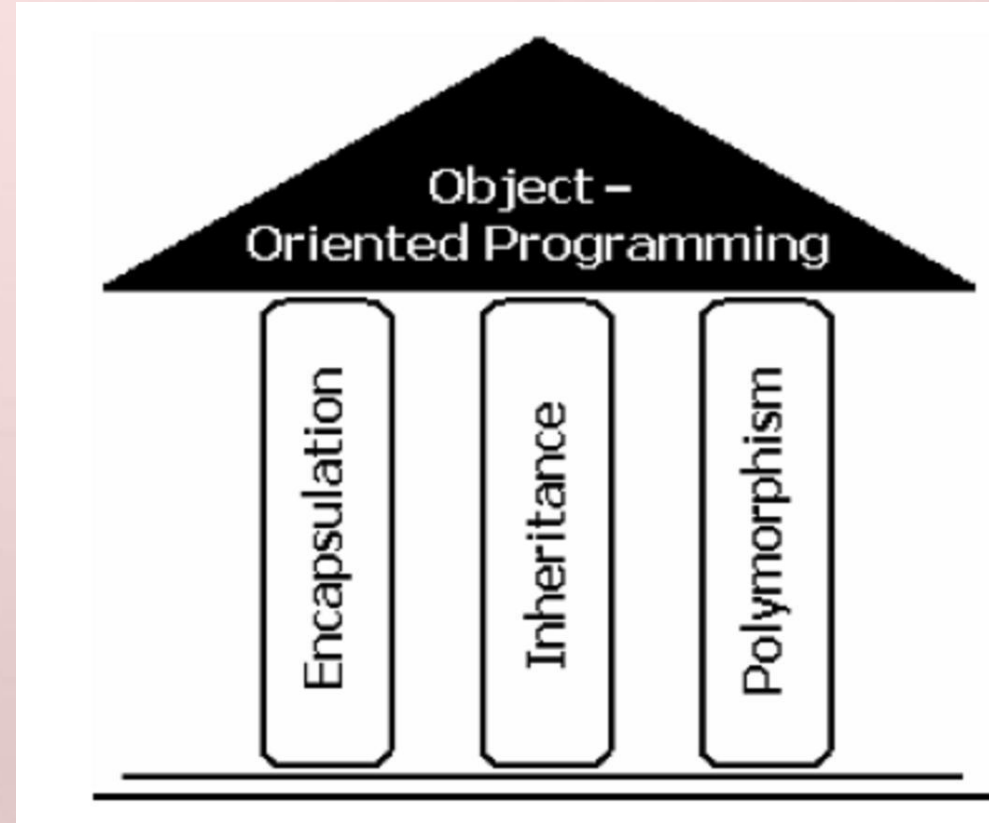
```
#include<iostream.h>
struct date
{
int day;
int month;
int year;
};
Void main(void)
{
date today;

void display (struct date one); // function declaration
today.day=18;
today.month=1;
today.year=2017;
display (today);
}
Void display (struct date one)
{   cout<<"Today's date is =" << one.day << "/";
cout<< one.month;
cout<<"/" << one.year << endl;
}
```

# Oop object oriented program

three importance OOP features:

- Encapsulation and Data Hiding.
- Inheritance and Reuse.
- Polymorphism.





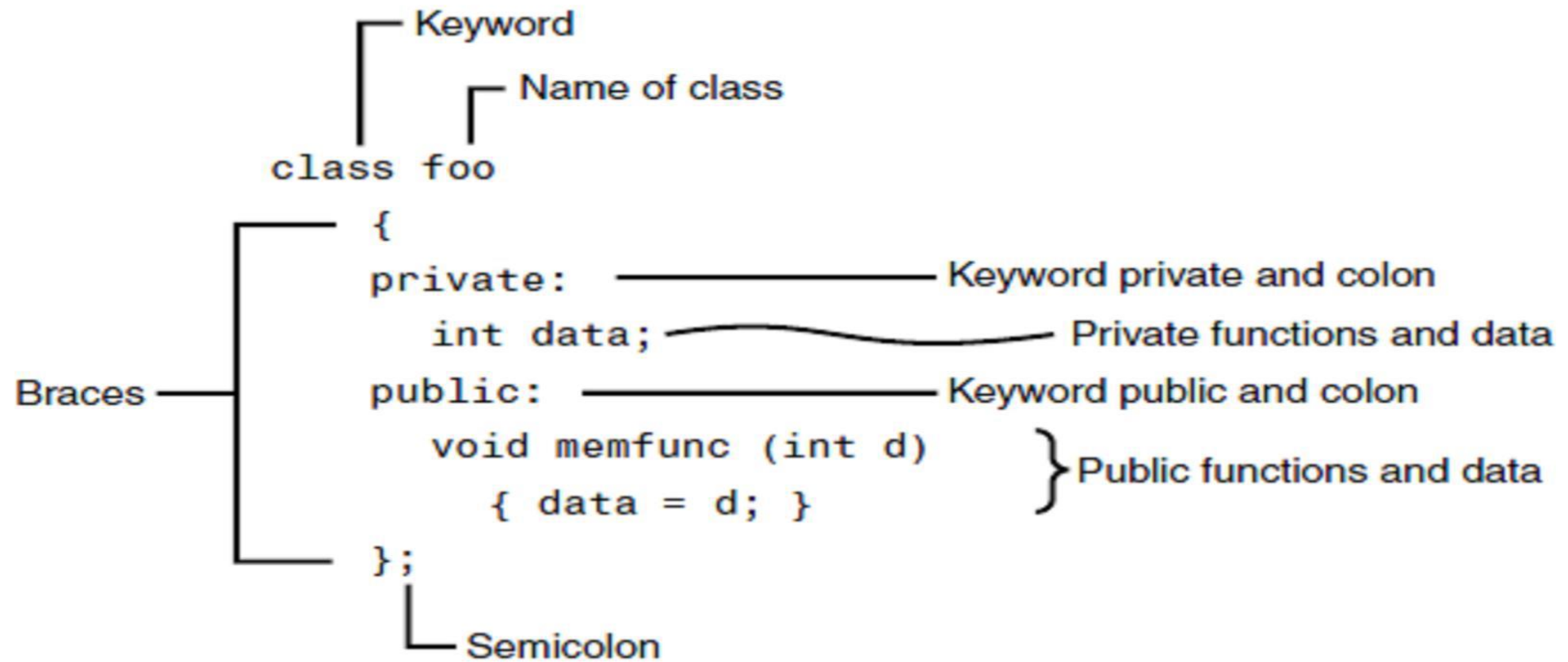
# Class Definition:

**Class** is a keyword, whose functionality is similar to that of the **struct** keyword, but with the possibility of including functions as members, instead of only data. Classes are collections of variables and functions that operate on those variables. The variables in a class definition are called **data members**, and the functions are called **member functions**.



Data + Functions = Object

Note: Class is a specification for number of objects.



**Figure2: Syntax of a class definition**

## Class members fall under one of three different access permission categories:

- ❖ **Public members** are accessible by all class users.
- ❖ **Private members** are only accessible by the class members.
- ❖ **Protected members** are only accessible by the class members and the members of a derived class.

### General form of class declaration:

```
class class-name
{
    public:
        public-data-members;
        public-functions;

    private:
        private-data-members;
        private -functions;

    protected:
        protected-data-members;
        protected -functions;
};
```

# Example:



Write a C++ program to modeling the Rectangle class.

```
#include <iostream.h>

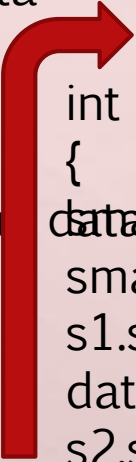
class Rectangle
{
    public:
        int length , width;
        int area( )
        {
            return length * width;
        }
};

int main( )
{
    Rectangle my_rectangle;
    my_rectangle.length = 6;
    my_rectangle.width = 7;
    cout<< my_rectangle.area( );
    return 0;
}
```

## Example 2: A Simple Class

```
#include <iostream>
class smallobj          //define a class
{
private:
int somedata;           //class data
public:
void setdata(int d)      //member function to set data
{
somedata = d;
}
void showdata()          //member function to display data
{ cout << "Data is " << somedata << endl; }
};

int main()
{
    smallobj s1, s2;      //define two objects of class
    s1.setdata(1066);     //call member function to set
                           data
    s2.setdata(1776);
    s1.showdata();        //call member function to display
                           data
    s2.showdata();
    return 0;
}
```





# Class Constructors and Destructors:

A class constructor is a function that is executed automatically whenever a new instance of a given class is declared.

The main purpose of a class constructor is to perform any initializations related to the class instances via passing of some parameter values as initial values and allocate proper memory locations for that object.

Note1: A class constructor must have the same name as that of the associated class.

Note2: A class constructor has not return type not even void.

Note3: A class constructor can be overloaded.



## Example:

Write an oop program to represent a rectangle constructor.

```
# include <iostream.h>

class Rectangle
{
    int length , width;
public:
    Rectangle(int x, int y)
    {
        length = x;
        width = y;
    }

    int area( )
    {
        return (length *width);
    }
};

void main( )
{
    Rectangle rect1(6,7);
    cout<< rect1.area( ) << endl;
}
```

# Destructor

Just as a constructor is used to initialize an object when it is created, a destructor is used to clean up the object just before it is destroyed. A destructor always has the same name as the class itself, but is preceded with a ~ symbol. Unlike constructors, a class may have at most one destructor. A destructor never takes any arguments and has no explicit return type.

Destructors are generally useful for classes which have pointer data members which point to memory blocks allocated by the class itself. In such cases it is important to release member-allocated memory before the object is destroyed. A destructor can do just that.

# Example:

Write an oop program to represent a simple destructor.

```
# include <iostream.h>

class Rectangle
{
    int length , width;
public:
    Rectangle(int x, int y)           //constructor
    {
        length = x;
        width = y;
    }
    ~ Rectangle()                     //destructor
    {
        cout<<"destructor delete data"<<endl;
    }

    int area( )
    {
        return (length *width);
    }
};

void main( )
{
    Rectangle rect1(6,7);
    cout<< rect1.area( ) << endl;
}
```

# Friend function

Occasionally we may need to grant a function access to the nonpublic members of a class. Such an access is obtained by declaring the function a **friend** of the class.

Example:

Write an oop program to find the summation of point using friend function.

```
#include <iostream.h>
class point
{
private:
int xval,yval;
public:
point()
{
xval=0;
yval=0;
cout<<"xval= "<<xval<<"yval= "<<yval<<endl;
}
point(int x,int y)
{
xval=x+2;
yval=y+3;
cout<<"xval= "<<xval<<" "<<"yval= "<<yval<<endl;
}

friend int sum(point p);           //friend function
};

int sum(point p)
{
int ss = p.xval + p.yval;
return (ss);
}

void main( )
{ point p2;
point p(2,2);
int dd;
dd=sum(p);
cout<<" the summation of coordinate x & y = "<<dd<<endl;
}
```

## Example:

Write an oop program to represent a friend function .

```
#include<iostream.h>
class xyz
{
    int x;
public:
    void setvalue(int i)
    {x=i;}
    friend void max (xyz,abc);
};
class abc
{
    int a;
public:
    void setvalue(int i)
    {a=i;}
    friend void max (xyz,abc);
};

void max (xyz m,abc n)
{
    if((m.x)>=(n.a))
        cout<<m.x;
    else
        cout<<n.a;
}

void main()
{
    abc vv;
    vv.setvalue(10);
    xyz aa;
    aa.setvalue(20);
    max(aa, vv);
}
```



# Example:



Write a C++ program under the concept of class constructor to modeling the Rectangle class, using scope resolution operator with the member function.

*Note: all data member are private.*

```
#include <iostream.h>
```

```
class Rectangle
```

```
{
```

```
    int length , width;
```

```
    public:
```

```
        Rectangle(int, int);
```

```
        int area( );
```

```
};
```

```
Rectangle :: Rectangle(int x, int y)
```

```
{
```

```
    length = x;
```

```
    width = y;
```

```
}
```

```
int Rectangle::area( )
```

```
{
```

```
    return length * width;
```

```
}
```

```
int main( )
```

```
{
```

```
    Rectangle my_rectangle(6,7);
```

```
    cout<< my_rectangle.area( );
```

```
    return 0;
```

```
}
```



# Static Members

## Example:

Write an oop program to represent a static member.

```
# include <iostream.h>
class test{

static int count;//count is static
int code;
public:
void setcode()
{
    cout<<"i am in set code"<<endl;
    code=++count;
}
void showcode()
{
    cout<<"i am in show code"<<endl;
    cout<<"object number"<<code<<"\n";
}
static void showcount()
{
    cout<<"i am in showcount"<<endl;
    cout<<"count: "<<count<<"\n";
}
};
int test::count;//count defined
void main ()
{
    test t1,t2;
    t1.setcode ();
    t2.setcode ();
    test::showcount ();
    test t3;
    t3.setcode ();
    test::showcount ();
    t1.showcode();
    t2.showcode();
    t3.showcode();
}
```

## output

```
i am in set code
i am in set code
i am in showcount
count : 2
i am in set code
i am in showcount
count : 3
i am in show code
object number1
i am in show code
object number2
i am in show code
object number3
```

# Arrays of Objects

## Example :

Write a simple program to represent array of class object point .

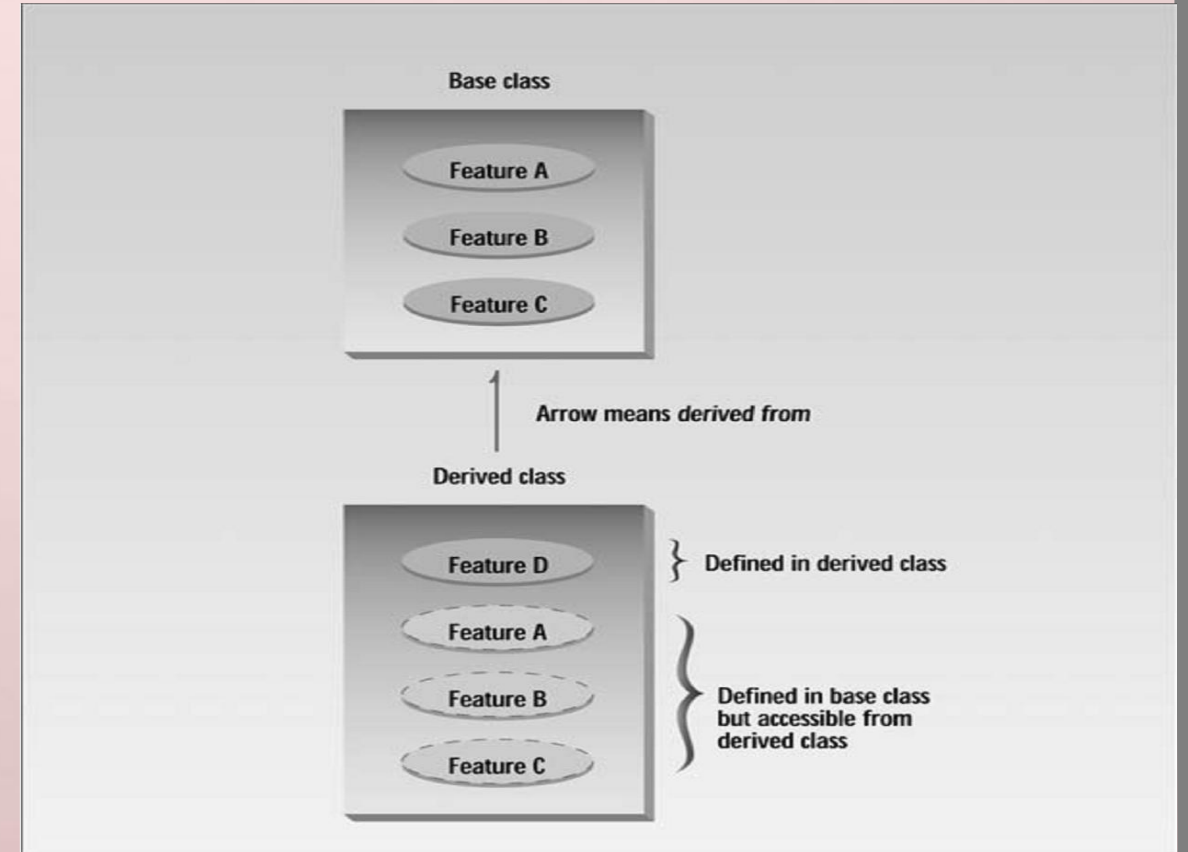
```
#include<iostream.h>
class point {
    int xval,yval;
public:
    void setpt(int x,int y)
    {
        xval=x;
        yval=y;
    }
    void offsetpt(int x,int y)
    {
        xval+=x;
        yval+=y;
        cout<<xval<<yval;
    }
};
void main()
{ int d,f;
  point pt[2];
  cout<<"enter the value of d & f";
  cin>>d>>f;
  pt[0].setpt(d,f);
  cout<<endl;
  pt[1].setpt(30,40);

  pt[0].offsetpt(2,2);
  pt[1].offsetpt (4,6);

}
```

# Inheritance

Inheritance is probably the most powerful feature of object-oriented programming, after classes themselves. Inheritance is the process of creating new classes, called **derived classes**, from existing or **base classes**. The derived class inherits all the capabilities of the base class but can add embellishments and refinements of its own. The base class is unchanged by this process. The inheritance relationship is shown in the Figure



# Defining Derived Class

- Whenever a derived class is defined, we have to specify its relationship with the base class. The general form of specifying derived class:

```
Class derived_class_name: visibility_mode base_class_name
{
    -----
    -----// members of derived class
    -----
};
```

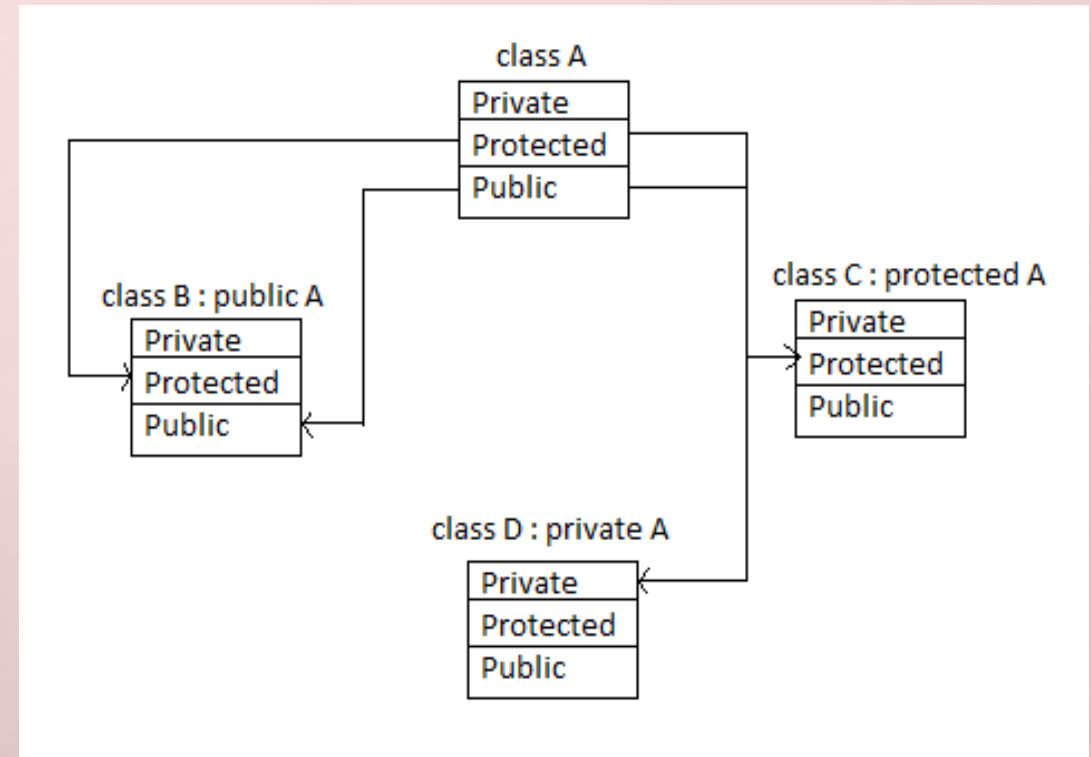
- The colon(:) symbol shows the relationship of a derived class to its base class. The visibility mode may be private or public.

**The default mode is private.**



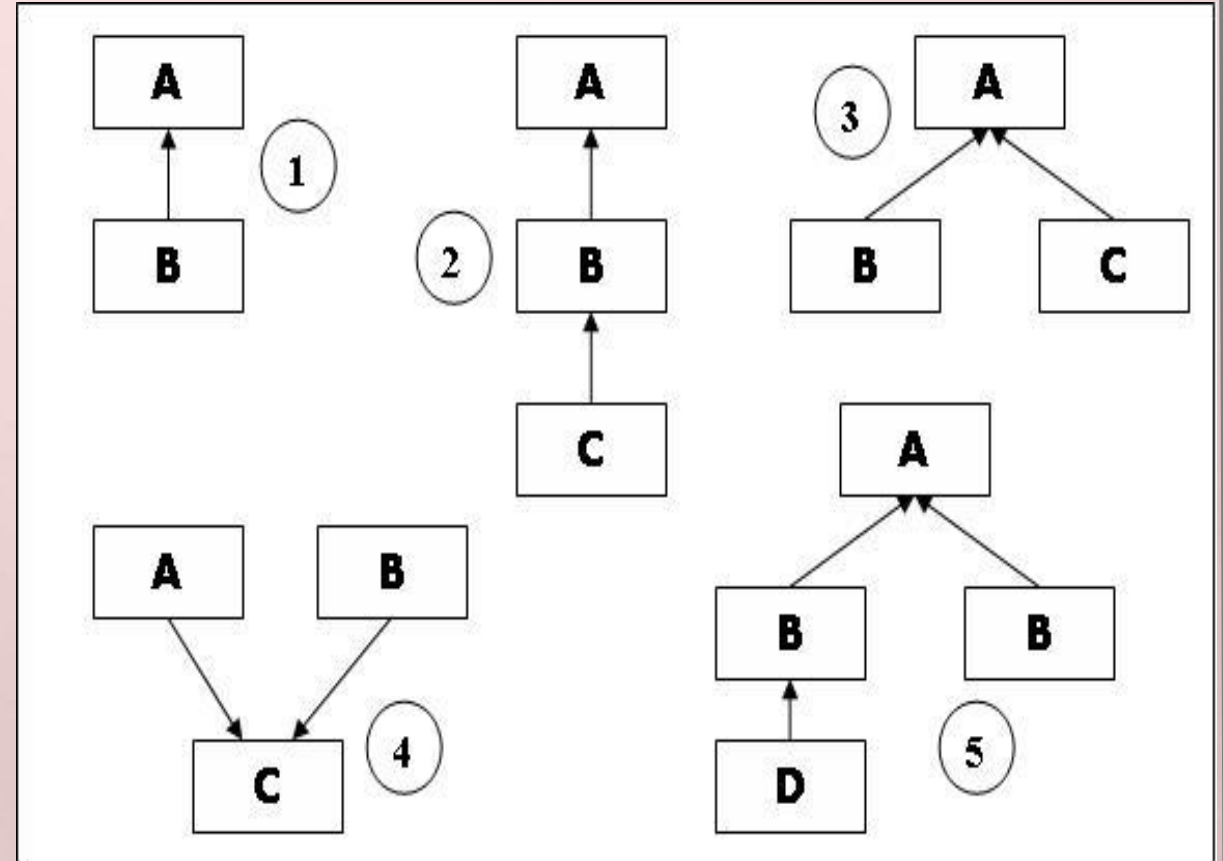
# Inheritance and Accessibility

Base class member access specifiers	Visibility of base class member in derived class		
	Public derivation	Protected derivation	Private derivation
Private	Invisible	Invisible	Invisible
Protected	Protected	Protected	Private
public	Public	Protected	Private



# forms of inheritance

1. single inheritance
2. Multilevel inheritance
3. Hierarchical inheritance
4. Multiple inheritance
5. Hybrid inheritance





# Example:

```
#include<iostream.h>

class person //base class
{
protected:
    int age;
    char name[20];
public:
    void readAge()
    {
        cout<<"Enter Age: ";
        cin>>age;
    }
    void readName(void)
    {
        cout<<"\nEnter Name: ";
        cin>>name;
    }
    void printPerInformation(void)
    {
        cout<<"Name - "<<name;
        cout<<"\nAge - "<<age;
    }
}; //derived class inherits base class
class student:public person
{
private:
    int Sno;
    int percentage;
public:
    void readSno()
    {
        cout<<"Enter Sno.: ";
        cin>>Sno;
    }
    void readpercentage()
    {
        cout<<"Enter percentage: ";
        cin>>percentage;
    }
};
```

```
void printStuInformation()
{
    cout<<"\nName - "<<name;
    cout<<"\nAge - "<<age;
    cout<<"\nS.no - "<<Sno<<endl;
    cout<<"Percentage- "<<percentage<<endl;
    cout<<"conclusion"<<endl;
    if(percentge>=80)
        cout<<"\nThe student is Outstanding"<<endl;
    else if(percentge>=70)
        cout<<"The student is Medium"<<endl;
    else cout<<"The student is Poor"<<endl;
}

};
int main()
{
    student st;
    st.readName();
    st.readAge();
    st.readSno();
    st.readpercentage();
    st.printStuInformation();
    return 0;
}
```