Catalog

Catalog

عدد الساعات الاسبوعية				اسم المادة	
عدد الوحدات	م	ع	ن	باللغة الانكليزية	باللغة العربية
6	4	2	2	Computer Programming (II)	برمجة الحاسوب (2)

أهداف المادة: تعريف الطالب على البرمجة الشيئية واستخدام لغة c++ المتقدم كمثال للبرمجة الشيئية في حل مسائل ذات علاقة بالاختصاص.

Weeks	Syllabus
1st,2nd	C++ Review (Program structure, namespace, identifiers, variables, constants, enum, operators, typecasting, control structures and functions).
3rd	Introduction to Object-Oriented Programming in C++.
4th,5th ,6th,7th,8th	Objects and Classes (Basics of object and class in C++, Private and public members, static data and function members, constructors and their types, destructors and operator overloading)
9th,10th,11th,12th, 13th,14th	Inheritance (Concept of Inheritance, types of inheritance: single, multiple, multilevel, hierarchical, hybrid, protected members, overriding, virtual base class).
15th,16th,17th 18th,19 th	Polymorphism (Pointers in C++, Pointes and Objects, this pointer, virtual and pure virtual functions, Implementing polymorphism)
20th ,21th,22 th 23 th,24 th	I/O and File management (Concept of streams, cin and cout objects, C++ stream classes, Unformatted and formatted I/O, manipulators, File stream, C++ File stream classes, File management functions, File modes, Binary and random files).
25 th,26 th ,27 th 28 th,29 th,30 th	Templates, Exceptions and STL (What is template? function templates and class templates, Introduction to exception, try-catch-throw, multiple catch, catch all, rethrowing exception, implementing user defined exceptions, Overview and use of Standard Template Library).

Book:// C++.pdf

Computer Programming II

د.آن زكي / دكتوراه في علوم الحاسبات – جامعة الموصل

BASIC DATA TYPES IN C++

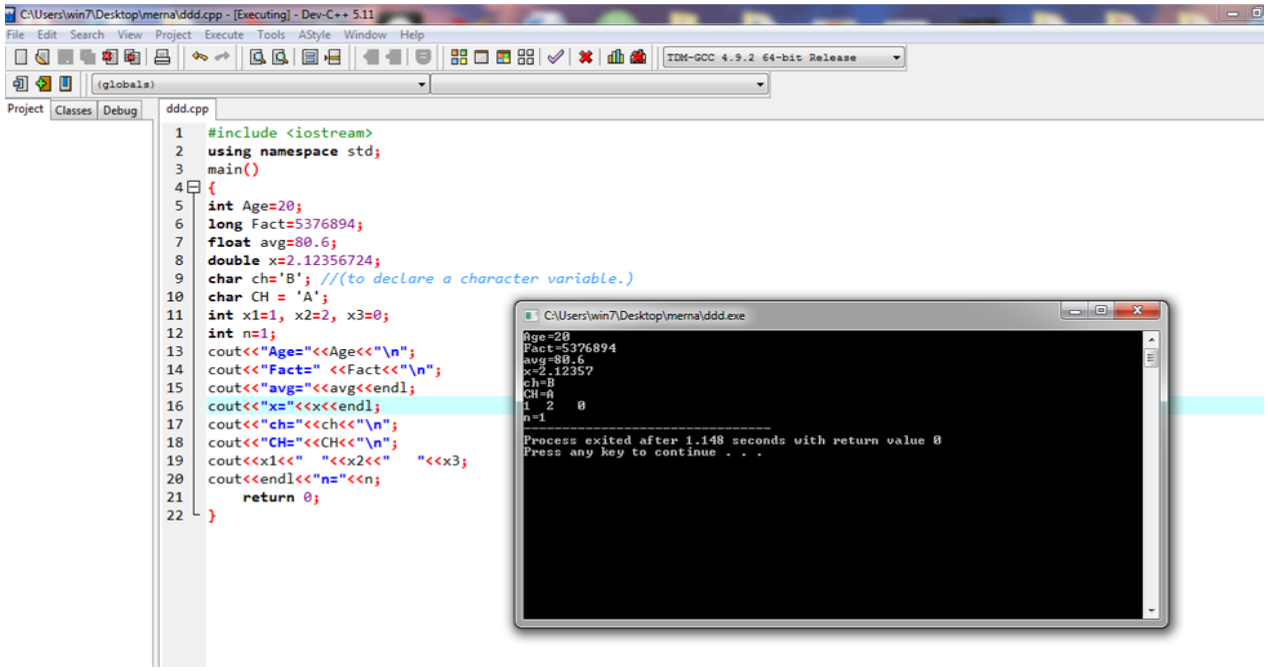
The basic data types available in C++ language are given in the following table:

Data Type	Range	Size	Usage
int	-32768 **to** 32767	2 bytes (16 bits)	For storing numbers without decimal.
long	-2147483648 **to** 2147483647	4 bytes (32 bits)	For storing integers. Has higher range than '**int**'.
char	0 **to** 255	1 byte (8 bits)	For storing characters.
float	$-3.4 * 10^{38}$ **to** $3.4 * 10^{38}$	4 bytes (32 bits)	For storing floating point numbers. It has seven digits of precision.
double	$\pm 1.7*10^{\pm 308}$ (15 digits)	8 bytes (64 bits)	It is used to store double precision floating point numbers.

Examples:

```
int my_Age;                 ( to declare an integer variable.)
long Fact=5376894;          (to declare a long integer variable with initial value.)
float AVERAGE2;             (to declare a real (floating point) variable.)
double x (2.12356724);      (to declare a double precision floating point variable
                             with initial value.)
char ch;                    (to declare a character variable.)
char CH = 'A';              (to declare a character variable with initial value.)
int x1, x2, x3=0;           (to declare more than one integers.)
```

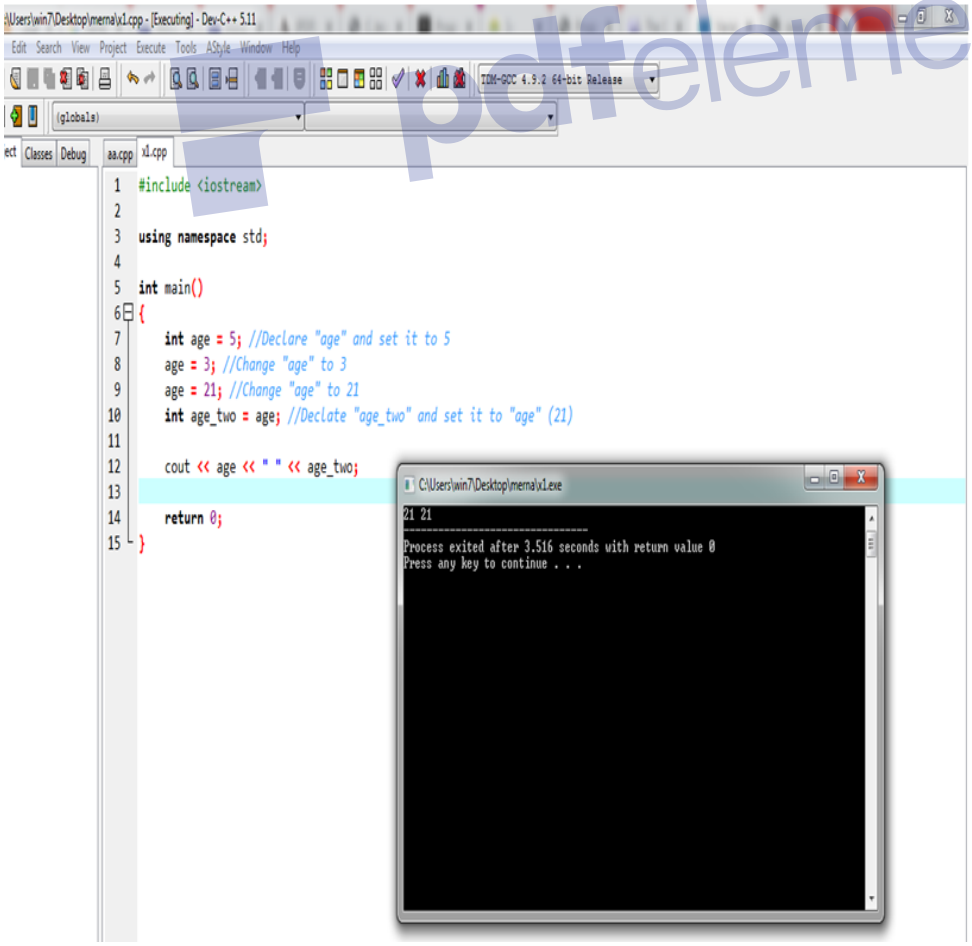
Example



```
#include <iostream>
using namespace std;
main()
{
int Age=20;
long Fact=5376894;
float avg=80.6;
double x=2.12356724;
char ch='B'; //(to declare a character variable.)
char CH = 'A';
int x1=1, x2=2, x3=0;
int n=1;
cout<<"Age="<<Age<<"\n";
cout<<"Fact=" <<Fact<<"\n";
cout<<"avg="<<avg<<endl;
cout<<"x="<<x<<endl;
cout<<"ch="<<ch<<"\n";
cout<<"CH="<<CH<<"\n";
cout<<x1<<"  "<<x2<<"   "<<x3;
cout<<endl<<"n="<<n;
    return 0;
}
```

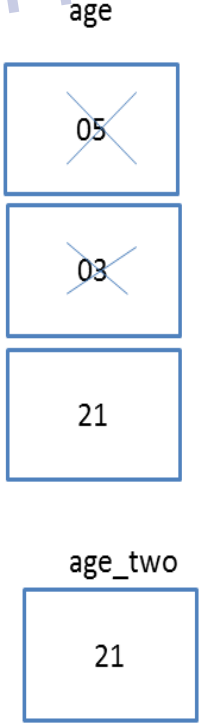
```
C:\Users\win7\Desktop\merna\ddd.exe
Age=20
Fact=5376894
avg=80.6
x=2.12357
ch=B
CH=A
1  2   0
n=1
--------------------------------
Process exited after 1.148 seconds with return value 0
Press any key to continue . . .
```

Example

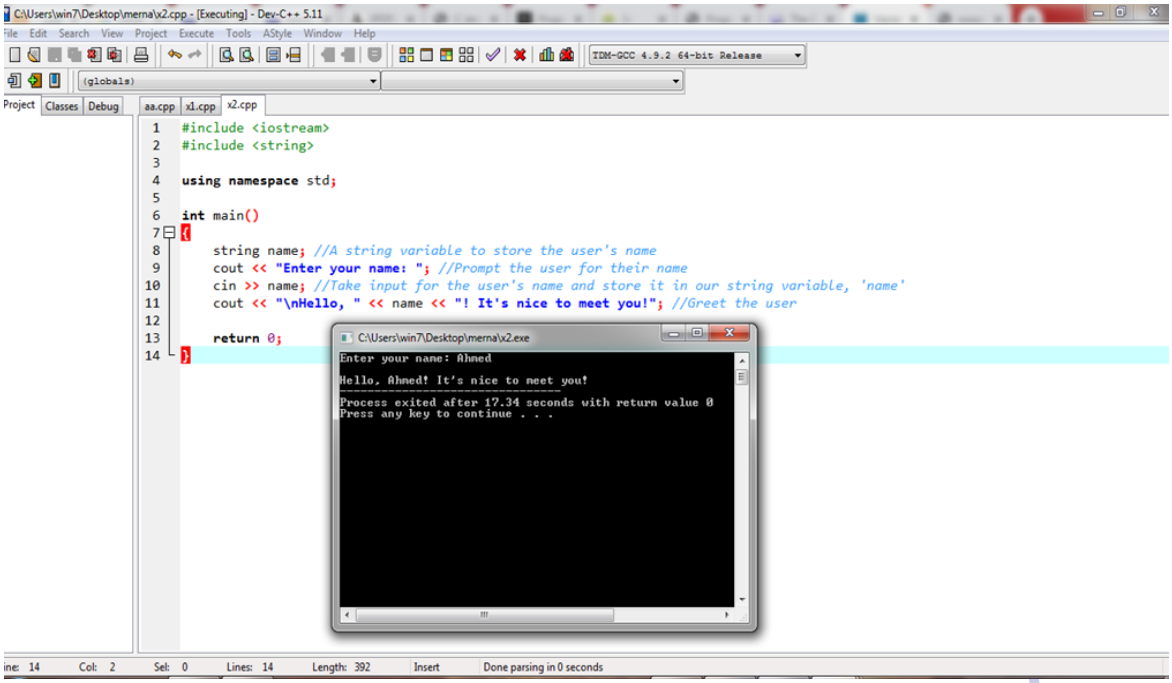


```
#include <iostream>

using namespace std;

int main()
{
    int age = 5; //Declare "age" and set it to 5
    age = 3; //Change "age" to 3
    age = 21; //Change "age" to 21
    int age_two = age; //Declare "age_two" and set it to "age" (21)

    cout << age << " " << age_two;

    return 0;
}
```

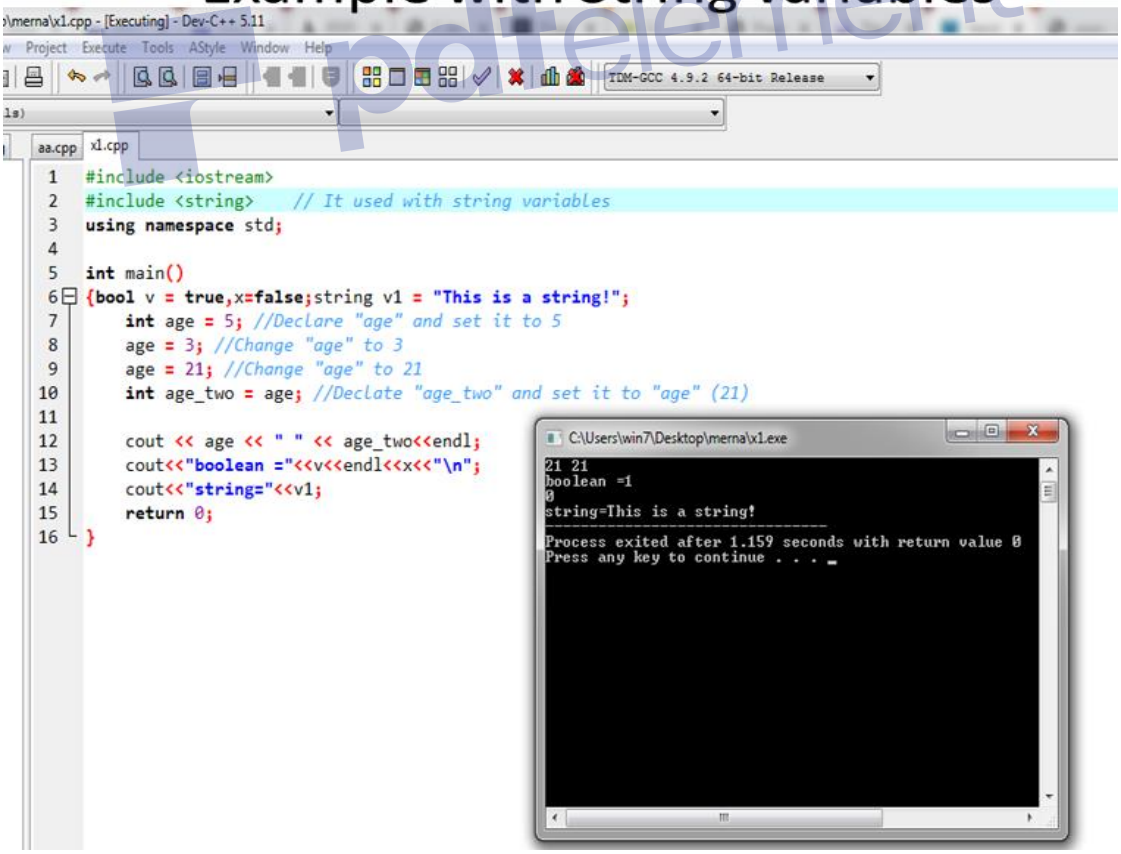
```
C:\Users\win7\Desktop\merna\x1.exe
21 21
--------------------------------
Process exited after 3.516 seconds with return value 0
Press any key to continue . . .
```



Example with String variables



Example with String variables



CONSTANTS

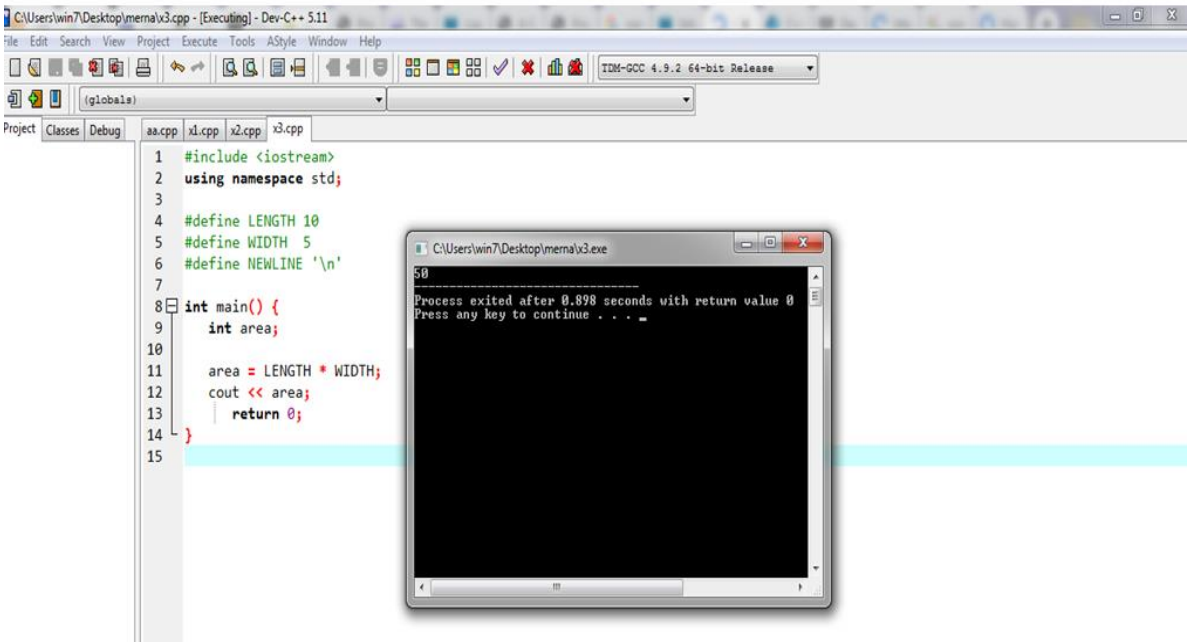
C++ allow for the programmer to define constants that represent decimal, hexadecimal octal, string and character constants. The **#define** directive can be used to define constants and it is placed after headers files.

```
#define PI 3.14156
#define MYNAME "JOHN DOE"
#define LIMIT 10
#define ESC 0x1B
```

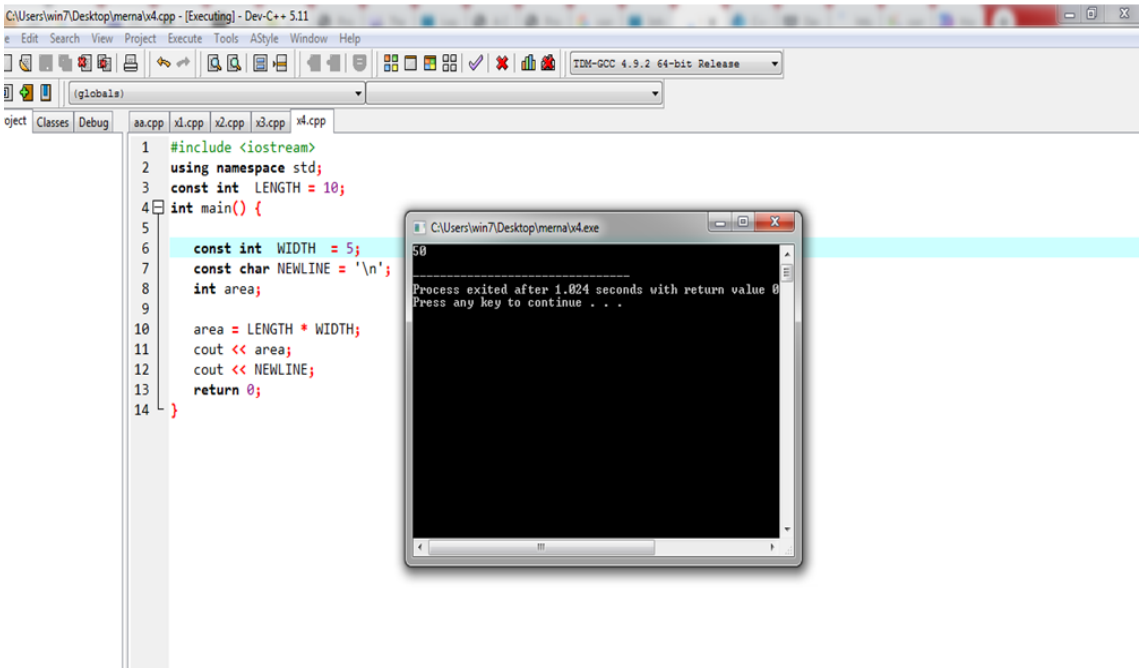
Also we can use **const** to define constants as follows :

```
const int diameter = 10 ;
const float PI = 3.14159;
const char ch= 'a';
```


Example with Constants



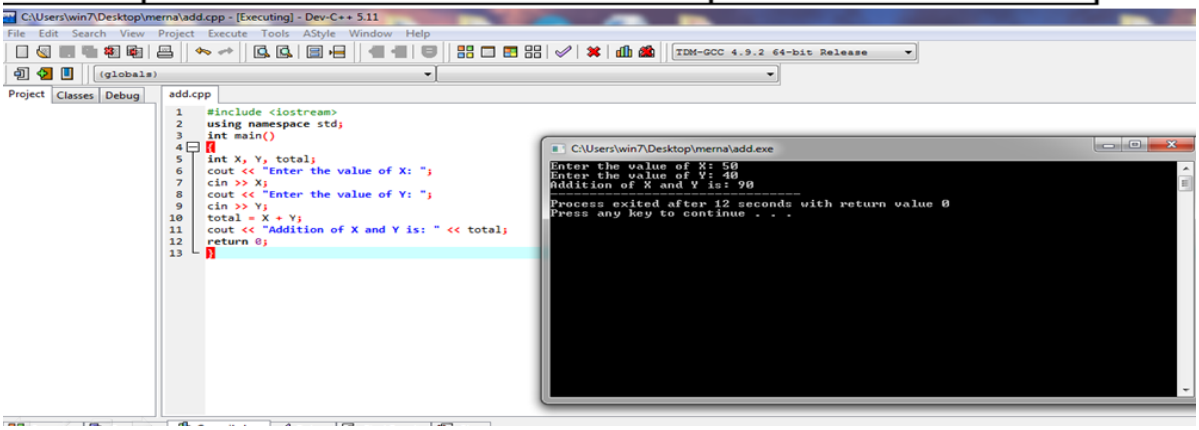
Example with Constants

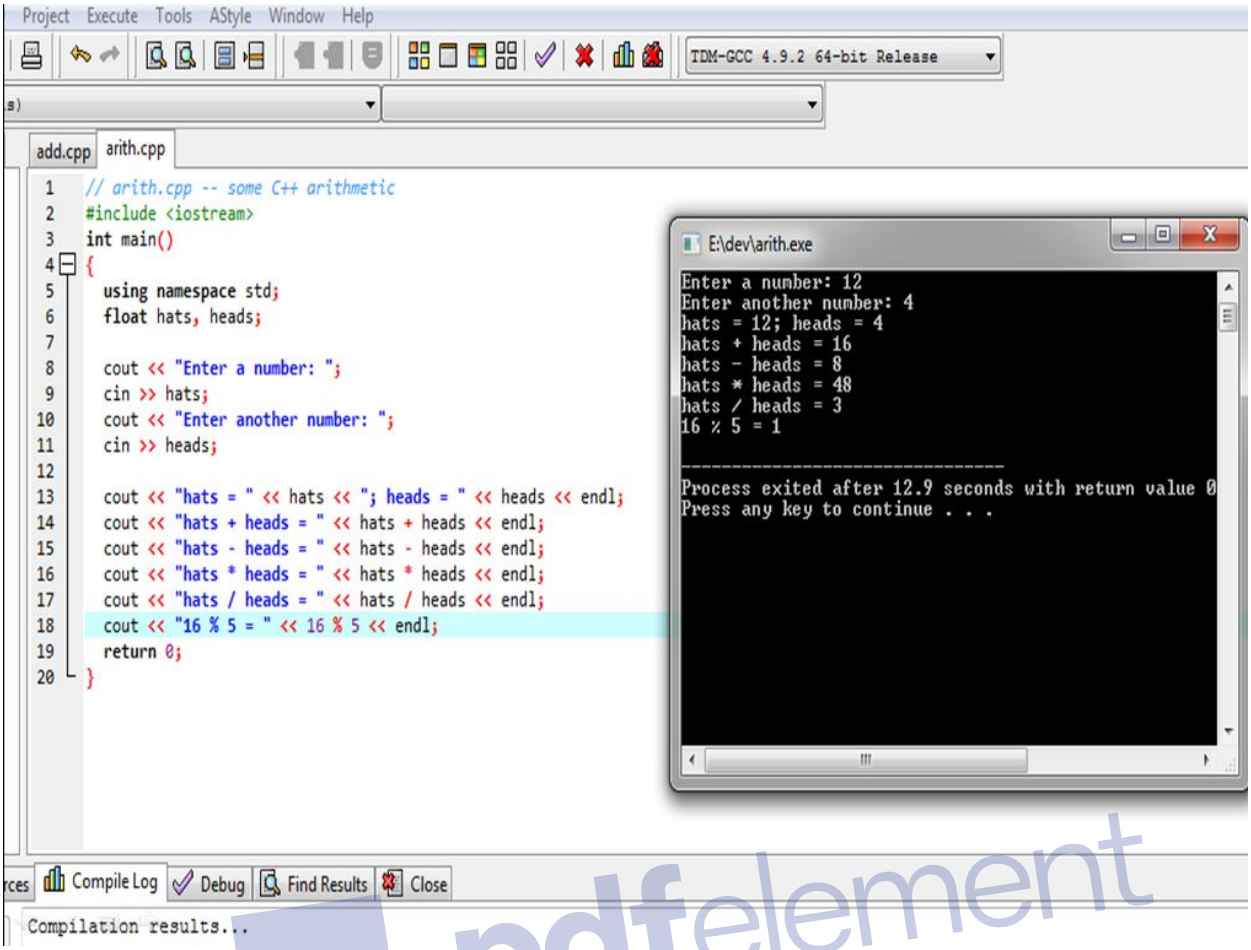


. Arithmetic Operators:

Arithmetic operators are used to perform the basic arithmetic operations. They are explained in the following table:

Operator	Usage	Examples
+	Used for addition	`Sum = a + b`
-	Used for subtraction	`Difference = a - b`
*	Used for multiplication	`Product = a * b`
/	Used for division	`Quotient = a / b`
%	This operator is called the remainder or the modulus operator. It is used to find the remainder after the division. This operator cannot be used with floating type variables.	`Remainder = a % b`





. Relational Operators:

The relational operators are explained in the following table:

Operator	Usage	Example	Explanation
<	Less than	A<B	A is less than B.
>	Greater than	A>B	A is greater than B.
<=	Less than or equal to	A<=B	A is less than or equal to B.
>=	Greater than or equal to	A>=B	A is greater than or equal to B.
==	Equality	A == B	A equal to B.
!=	Not equal to	A != B	A is not equal to B.

Form :

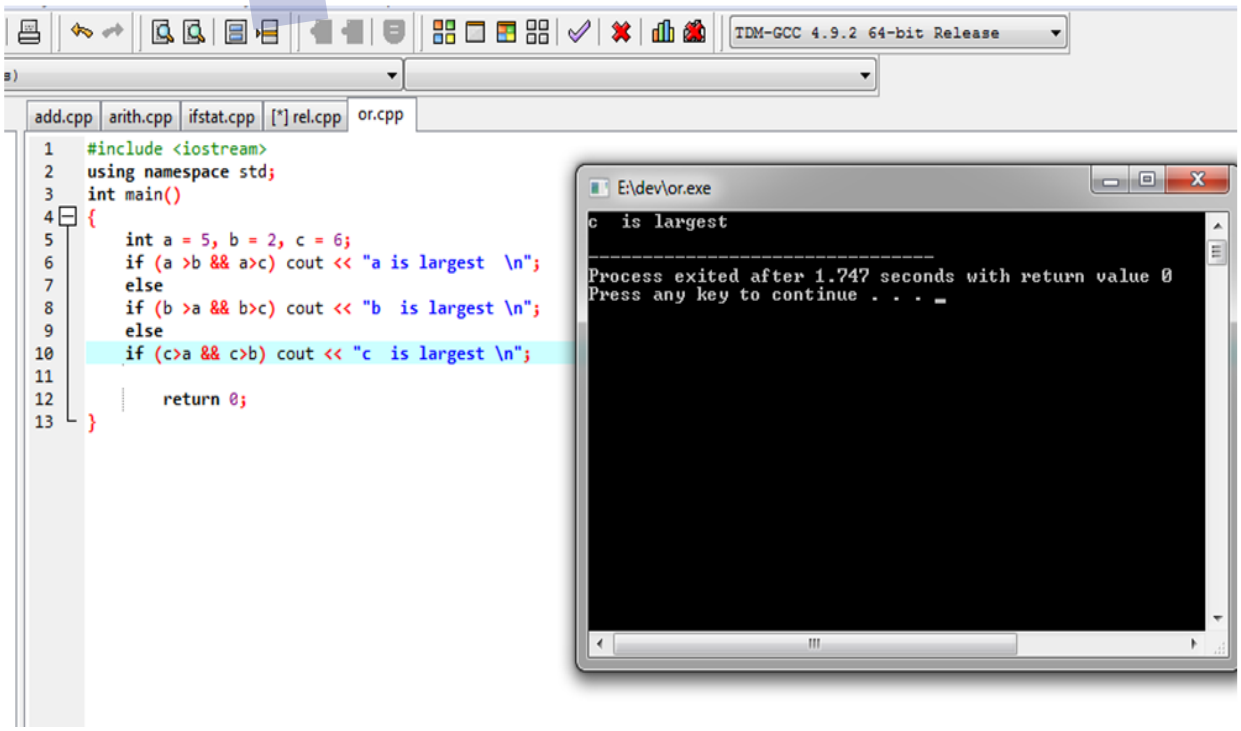
```
if (boolean-expression)
        statement-1;
else
        statement-2;
```

Logical Operators

The logical operators are used to combine multiple conditions (logical statements). Th following table describes the logical operators:

Operator	Usage	Example
&& (logical AND)	The compound condition is true, if both conditions are true.	((a>b) && (a>c))
|| (logical OR)	The compound statement is true, if any or both conditions are true.	((a>b) || (a>c))
! (logical NOT)	It negates the condition.	! (a>b)

Example : Find the largest number between 3 numbers.



HOMEWORK

Q1. WRITE C++ PROGRAM FOR CHECKING ANY NUMBER IF IT IS POSITIVE PRINT "POS" ELSE PRINT "NEG".

Q2. WRITE C++ PROGRAM FOR CHECKING ANY NUMBER IF IT IS ODD PRINT "ODD" ELSE PRINT "EVEN".

Q3. Write a C++ program that prompts the user to input three integer values and find the greatest value of the three values and smallest value.

Q4. Write a program that determines a student's grade. The program will read the average And determine the grade based on the following rules:

-if the average score =90% =>grade=A
-if the average score >= 70% and <90% => grade=B
-if the average score>=50% and <70% =>grade=C
-if the average score<50% =>grade=F

The switch Statement

When you want to create different outcomes depending on specific values of a variable int or char.

```
switch(expression_to_evaluate)
 { case value_to_check_against :
                              Statement 1;
                               break;
case value_to_check_against_two :
                              Statement 2;
                               break;
      .
      .
      .
default :
}
```

Example

sww.cpp

```
1  #include <iostream>
2  using namespace std;
3  int main(){
4     int i=2;
5     switch(i) {
6        case 1: cout<<"Case1 "<<endl;
7        case 2: cout<<"Case2 "<<endl;
8        case 3: cout<<"Case3 "<<endl;
9        case 4: cout<<"Case4 "<<endl;
10       default: cout<<"Default "<<endl;
11    }
12    return 0;
13 }
```

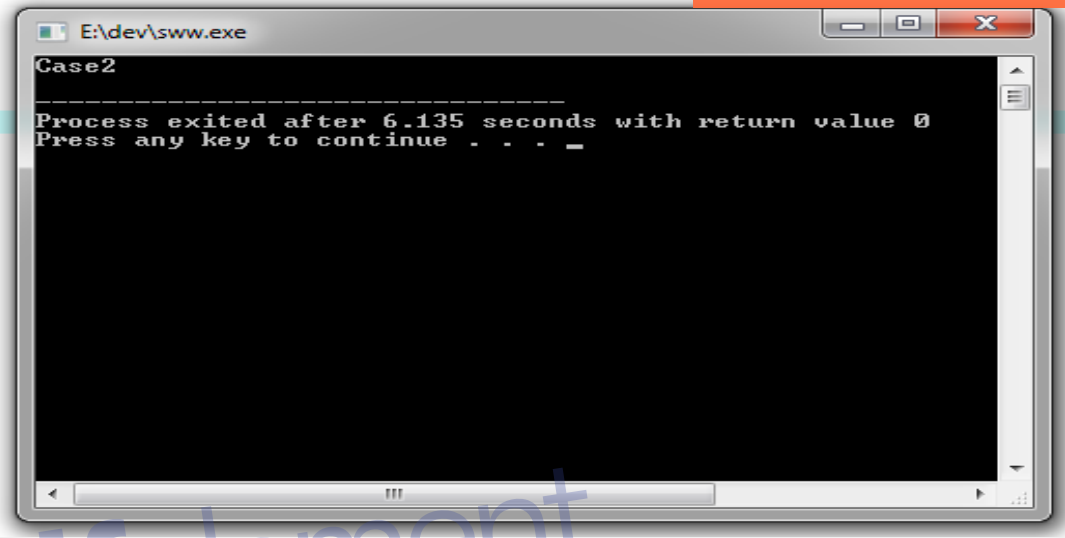
E:\dev\sww.exe

```
Case2
Case3
Case4
Default

--------------------------------
Process exited after 4.708 seconds with return value 0
Press any key to continue . . .
```

sww.cpp

```cpp
#include <iostream>
using namespace std;
int main(){
    int i=2;
    switch(i) {
        case 1: cout<<"Case1 "<<endl; break;
        case 2: cout<<"Case2 "<<endl;break;
        case 3: cout<<"Case3 "<<endl;break;
        case 4: cout<<"Case4 "<<endl;break;
        default: cout<<"Default "<<endl;
    }
    return 0;
}
```

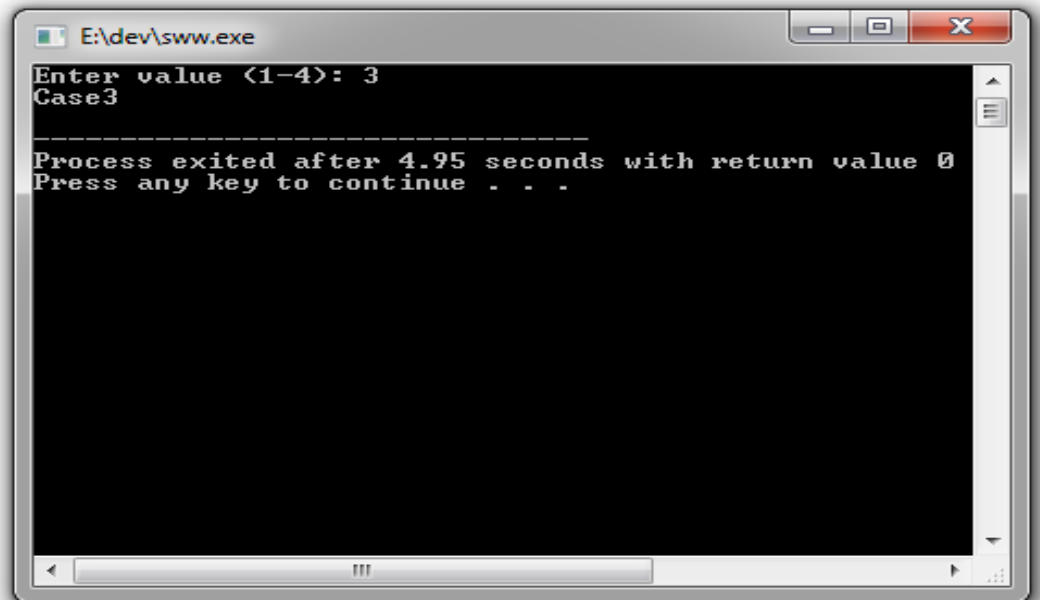
Example



sww.cpp

```cpp
#include <iostream>
using namespace std;
int main(){
    int i;cout<<"Enter value (1-4): ";
    cin>>i;
    switch(i) {
        case 1: cout<<"Case1 "<<endl; break;
        case 2: cout<<"Case2 "<<endl;break;
        case 3: cout<<"Case3 "<<endl;break;
        case 4: cout<<"Case4 "<<endl;break;
        default: cout<<"Default "<<endl;
    }
    return 0;
}
```

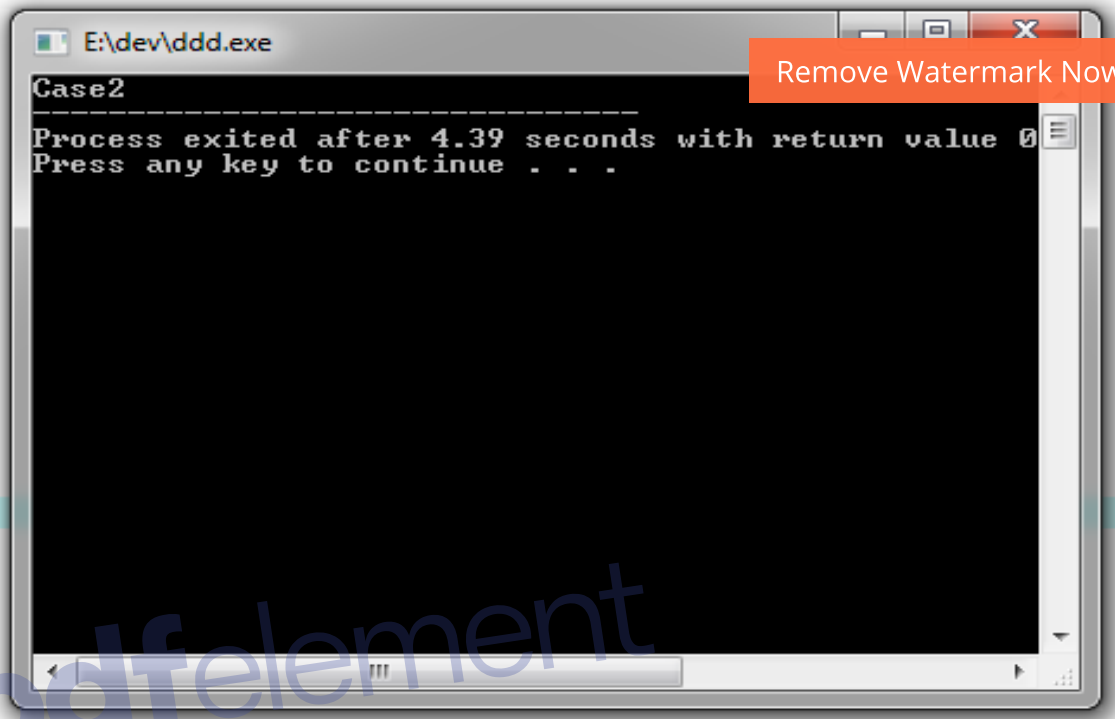
Example



sww.cpp ddd.cpp

Example

```cpp
#include <iostream>
using namespace std;
int main(){
    char ch='b';
    switch(ch) {
        case 'd': cout<<"Case1 ";
        break;
        case 'b': cout<<"Case2 ";
        break;
        case 'x': cout<<"Case3 ";
        break;
        case 'y': cout<<"Case4 ";
        break;
        default: cout<<"Default ";
    }
    return 0;
}
```



Example

```cpp
#include <iostream>
using namespace std;

int main () {
   // local variable declaration:
   char grade = 'D';

   switch(grade) {
      case 'A' :
         cout << "Excellent!" << endl;
         break;
      case 'B' :
      case 'C' :
         cout << "Well done" << endl;
         break;
      case 'D' :
         cout << "You passed" << endl;
         break;
      case 'F' :
         cout << "Better try again" << endl;
         break;
      default :
         cout << "Invalid grade" << endl;
   }
   cout << "Your grade is " << grade << endl;

   return 0;
}
```

Example


```
4  int main () {
5     // local variable declaration:
6     char grade = 'M';
7
8     switch(grade) {
9        case 'A' :
10          cout << "Excellent!" << endl;
11          break;
12       case 'B' :
13       case 'C' :
14          cout << "Well done" << endl;
15          break;
16       case 'D' :
17          cout << "You passed" << endl;
18          break;
19       case 'F' :
20          cout << "Better try again" << endl;
21          break;
22       default :
23          cout << "Invalid grade" << endl;
24    }
25    cout << "Your grade is " << grade << endl;
26
27    return 0;
28 }
```

```
E:\dev\qq.exe
Invalid grade
Your grade is M

--------------------------------
Process exited after 4.691 seconds with return value 0
Press any key to continue . . .
```


Loop instructions:

```
for(initialization; condition ; increment/decrement)
{
   C++ statement(s);
}
```

Examples

sww.cpp | ddd.cpp | qq.cpp | loop.cpp

```cpp
#include <iostream>
using namespace std;
int main(){
    for(int i=1; i<=6; i++){
        cout<<"Value of variable i is: "<<i<<endl;
    }
    return 0;
}
```

```
E:\dev\loop.exe
Value of variable i is: 1
Value of variable i is: 2
Value of variable i is: 3
Value of variable i is: 4
Value of variable i is: 5
Value of variable i is: 6

--------------------------------
Process exited after 2.626 seconds with re
Press any key to continue . . .
```



```cpp
#include <iostream>
using namespace std;
int main(){
    int arr[5]={21,9,56,99, 202};

    for(int i=0; i<5; i++){
        cout<<arr[i]<<endl;
    }
    return 0;
}
```

```
E:\dev\arr.exe
21
9
56
99
202

--------------------------------
Process exited after 6.153 seconds w
Press any key to continue . . .
```

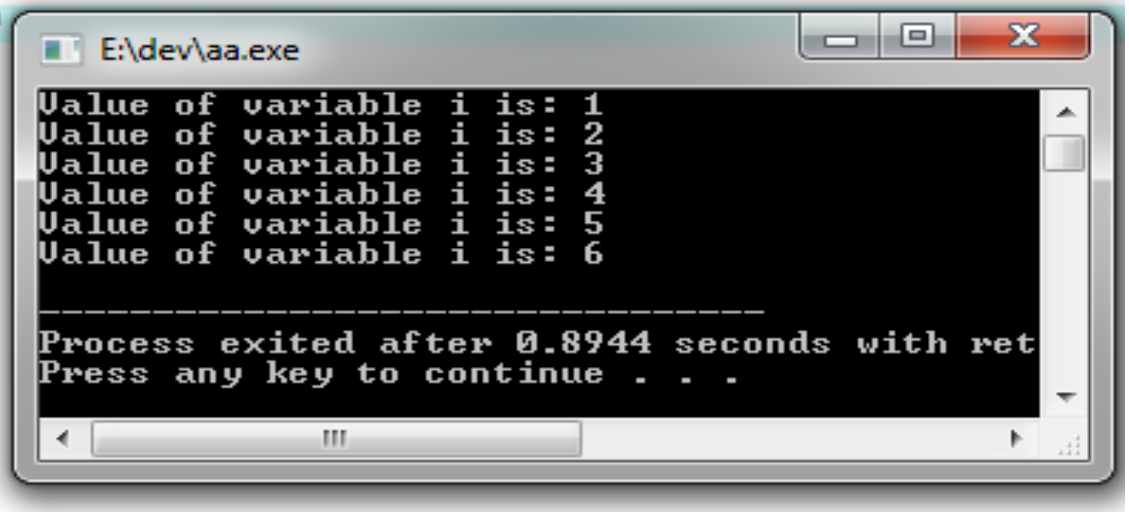
While Command


```
while(condition) {
    statement(s);
}
```

Example

aa.cpp

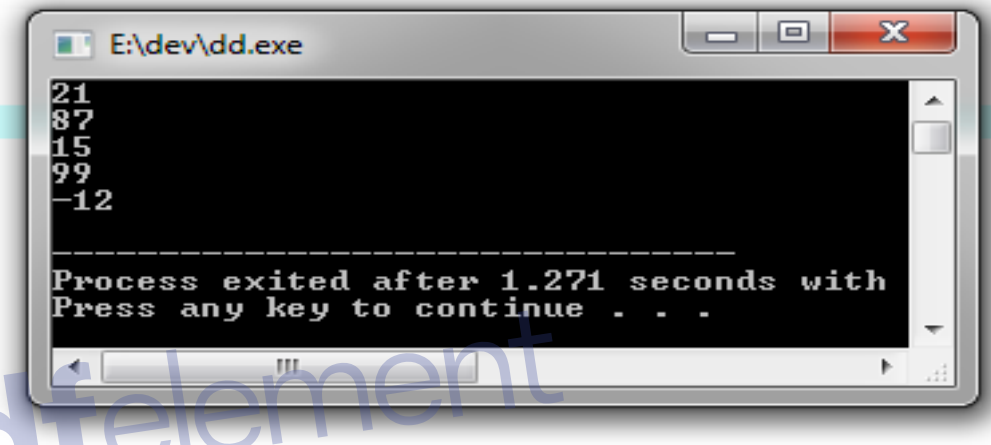
```
1  #include <iostream>
2  using namespace std;
3  int main(){
4      int i=1;
5          while(i<=6){
6          cout<<"Value of variable i is: "<<i<<endl; i++;
7      }
8  }
```

Example

aa.cpp | dd.cpp

```
1   #include <iostream>
2   using namespace std;
3   int main(){
4      int arr[]={21,87,15,99, -12};
5       int i=0;
6      while(i<5){
7         cout<<arr[i]<<endl;
8         i++;
9      }
10  }
```



Syntax of do-while loop

```
do
{
    statement(s);
} while(condition);
```

Example

aa.cpp | dd.cpp | do.cpp

```cpp
#include <iostream>
using namespace std;
int main(){
    int num=1;
    do{
        cout<<"Value of num: "<<num<<endl;
        num++;
    }while(num<=6);
    return 0;
}
```

E:\dev\do.exe

```
Value of num: 1
Value of num: 2
Value of num: 3
Value of num: 4
Value of num: 5
Value of num: 6

--------------------------------
Process exited after 1.914 seconds w
Press any key to continue . . .
```

aa.cpp | dd.cpp | do.cpp | dddd.cpp

```cpp
#include <iostream>
using namespace std;
int main(){
    int arr[]={21,99,15,109};
        int i=0;
    do{
        cout<<arr[i]<<endl;
        i++;
    }while(i<4);
    return 0;
}
```

E:\dev\dddd.exe

```
21
99
15
109

--------------------------------
Process exited after 4.419 seconds with re
Press any key to continue . . .
```

Arrays in C++

An **array** is a collection of elements of the same type placed in contiguous memory locations that can be individually referenced by using an index to a unique identifier. Five values of type int can be declared as an **array** without having to declare five different variables (each with its own identifier).

Single dimensional arrays

Syntax:
type arrayname[array size] = {};

Type: type can be any c++ data type
Array size: array size must be integer constant greater than zero
Array name: valid c++ identifier

Example: int x[1]={10}; ==> Note: x is an array of one integer with array size one.
Example: int x[10]; ==> Note: here x is an array of 10 integers
you can assign integers values to this example through For loop

```
for(int i =0 ; i< 10 ; i++)
{
   x[i]= i + 2;
}
```

arrays integers value will b 2,3,4,5,6,7,8,9,10,11

Initialization: Arrays can be initialize in two ways weather one by one as in above example or using a single statement.

Example using single statement:

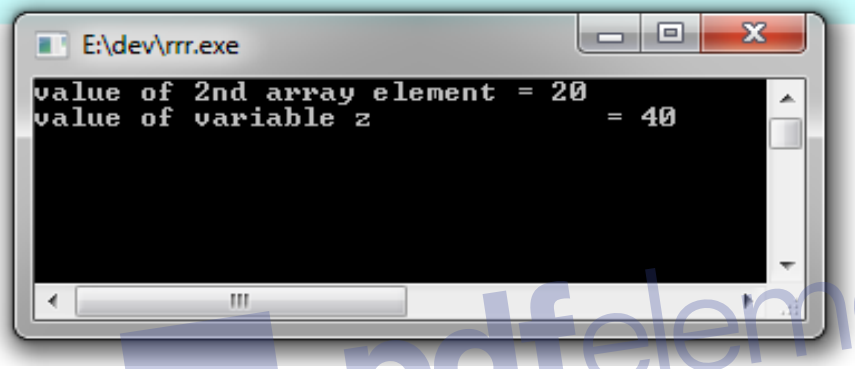
```
int x[5] = {1,2,3,4,5};
```

Example without Array size:

```
int x[ ] = {1,2,3,4,5};
```

==> Note: it is also correct but the difference is array size is undefined

```cpp
#include<conio.h>
using namespace std;

int main()
{
 int x[5]={10,20,30,40,50};          // initialization of array

 int z =  x[3];                                  // assigning integer variable z a value of 40 from array named x[]

 cout<<"value of 2nd array element = "<<x[1]<<endl;                    // it will display 20
 cout<<"value of variable z              = "      <<z;                                  // it will display 40

 getch();

}
```

Multi Dimensional Arrays

Syntax:

type array_name [size 1] , [size 2] , ... , [size n];

Example: for three dimensional array
int threedim[5][6][2];

Two Dimensional Arrays:
Two dimensional arrays are the simplest form of multi-dimensional arrays

Syntax:
type array_name [size 1] [size 2];

Array name : valid c++ identifier
Type : valid c++ data type
[size 1][size 2] : two dimensional array size

Two dimensional arrays can be easily understand if you consider it as a table which has **size 1** number of rows and **size 2** number of columns.Look at this example

Example:

int x [3][4]; ==> two dimensional array with three rows and four columns

Consider this two dimensional array as table which has three rows and four columns.

	Column 0	Column 1	Column 2	Column 3
Row 0	x [0][0]	x[0][1]	x [0][2]	x [0][3]
Row 1	x [1][0]	x [1][1]	x [1][2]	x [1][3]
Row 2	x [2][0]	x [2][1]	x [2][2]	x [2][3]

Thus every element in the array x can be identified by form x[i][j] , where i and j are can be considered as subscripts.

Initialization:

Multidimensional arrays can be initialized by specifying each row with braces separated by comma's.

Example:

```
int x[3][4] = { {10,20,30,40} , {50,60,70,80} , {90,100,110,120} };
```

It can also be initialized like this

```
int x[3][4]= { 10,20,30,40,50,60,70,80,90,100,110,120 };
```

Examples

aa.cpp | dd.cpp | do.cpp | dddd.cpp | rrr.cpp | ere.cpp

```cpp
#include<iostream>
#include<conio.h>
using namespace std;

int main()
{
 int x[3][4]={ {10,20,30,40} , {50,60,70,80} ,  {90,100,110,120}  };      // initializtion of 2 dim. array

 cout<<"x[2][1] elementof the array is = "  << x[2][1] <<endl;                  // it wil dislpay 100

 getch();

}
```

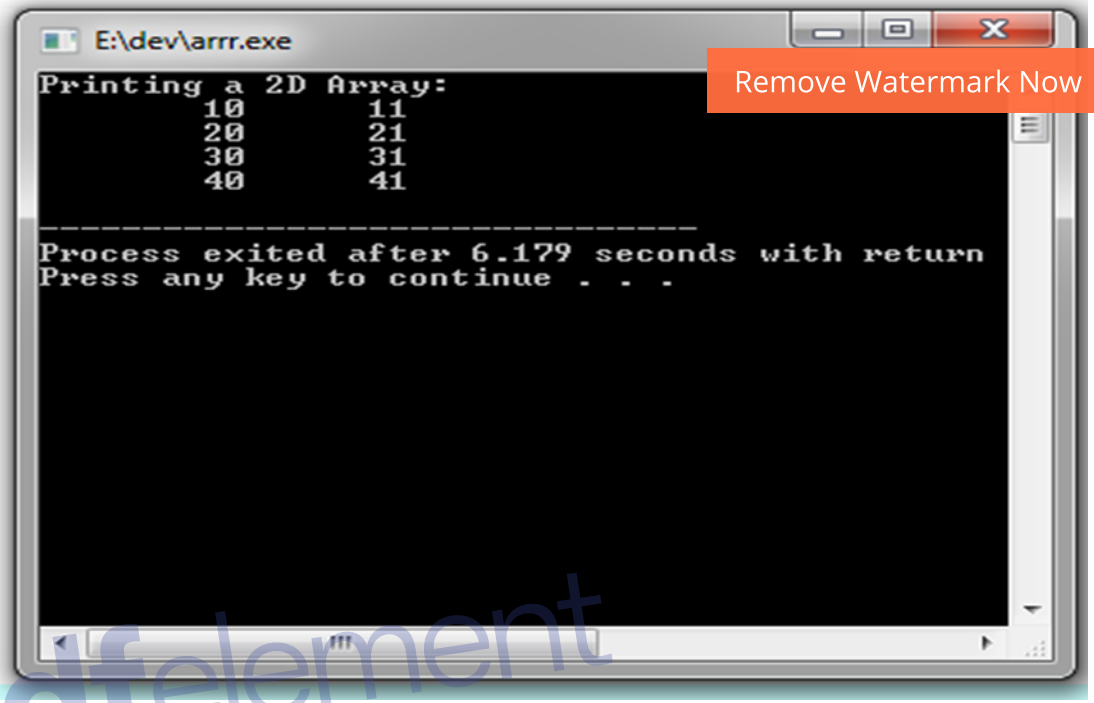
```
E:\dev\ere.exe
x[2][1] elementof the array is = 100
```

arrr.cpp | starray.cpp

```cpp
#include <iostream>
#include <string>
using namespace std;
int main()
{
   string  strArray[5] = {"one", "two", "three", "four", "five"};
    cout<<"String array is as follows:"<<endl;
    for(int i=0;i<5;i++)
    {
        cout<<"Element "<<i<<"= "<<strArray[i]<<endl;
    }

  return 0;
}
```

```
E:\dev\starray.exe
String array is as follows:
Element 0= one
Element 1= two
Element 2= three
Element 3= four
Element 4= five

--------------------------------
Process exited after 1.978 seconds with return v
Press any key to continue . . .
```

```cpp
#include<iostream>
using namespace std;
main( )
{
    int arr[4][2] = {
        { 10, 11 },
        { 20, 21 },
        { 30, 31 },
        { 40, 41 }
        } ;

    int i,j;

    cout<<"Printing a 2D Array:\n";
    for(i=0;i<4;i++)
    {
        for(j=0;j<2;j++)
        {
            cout<<"\t"<<arr[i][j];
        }
        cout<<endl;
    }
}
```

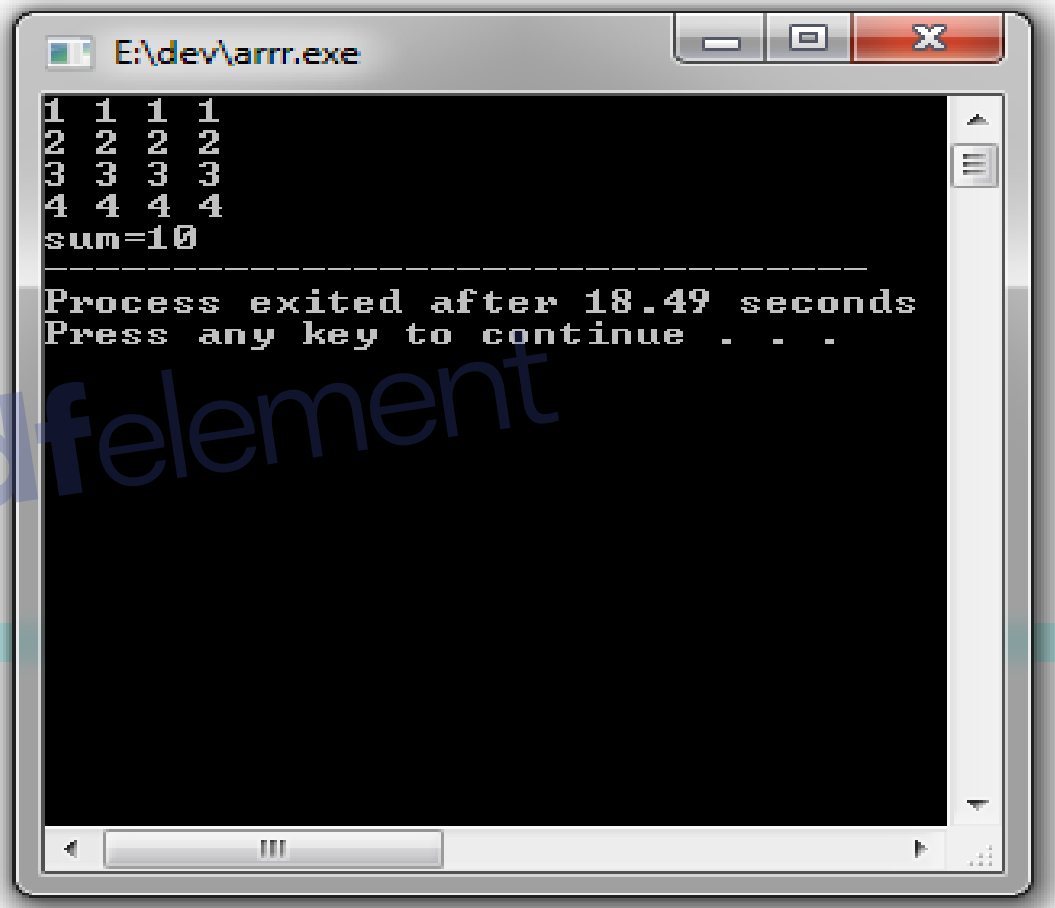


i	j
0	0
	1
	2
1	0
	1
	2
2	0
	1
	2
3	0
	1
	2

Example :

Write C++ program for adding the element of main diameter for an integer array a[4][4].

```
1   #include<iostream>
2   using namespace std;
3   main( )
4   {
5       int a[4][4];
6       int i,j,sum=0;
7       for(i=0;i<4;i++)
8       {
9           for(j=0;j<=3;j++)
10                  cin>>a[j][i];
11      }
12
13      for(i=0;i<4;i++)
14      sum=sum+a[i][i];
15          cout<<"sum="<<sum;
16
17  }
18
```



Homework

Q1. Write C++ program for checking any number if it is prime or not. If it is prime print “PR” else print “NPR”.

Q2. Write C++ program for sort 10 integer number ascending.

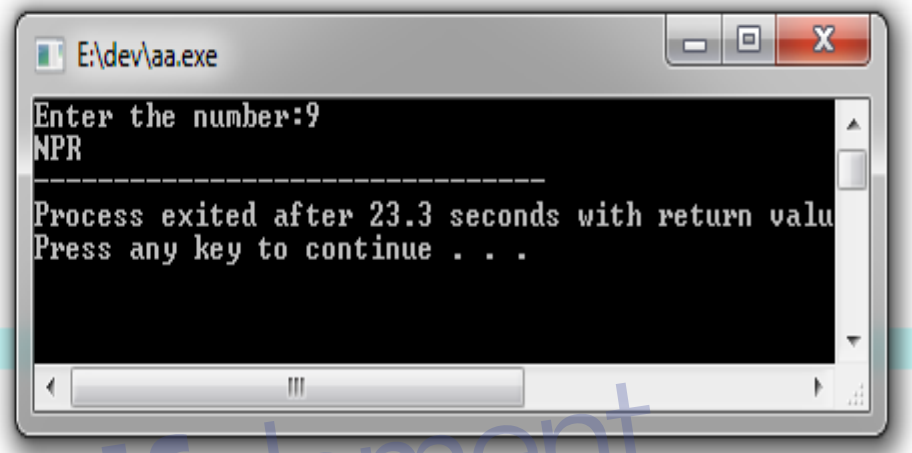
Q3. Write C++ program for sort 10 names descending . Hind : use array of string.

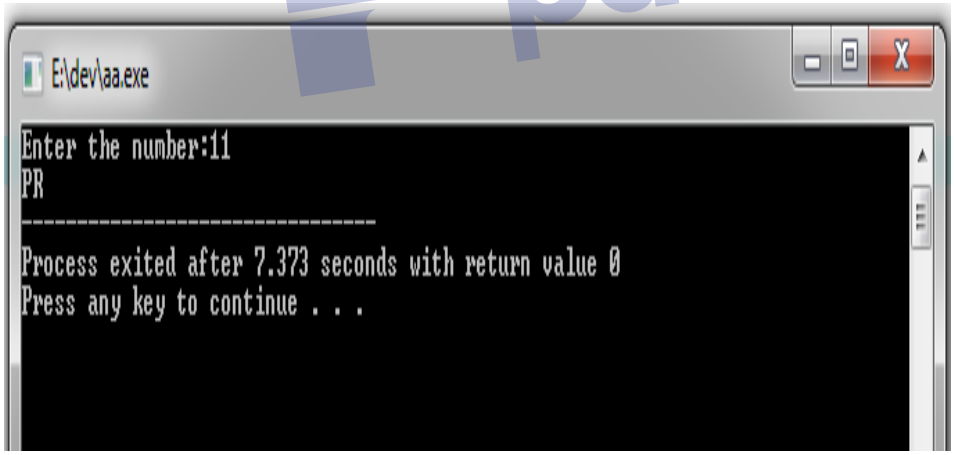
Q4. If you have an array ma of type character , write C++ program for :

1. Print all the elements up of main diameter .
2. Print all the elements under the main diameter.
3. Print first rows only.
4. Print last column only.

Q1. Write C++ program for checking any number if it is prime or not. If it is prime print “PR” else print “NPR”.


```cpp
#include <iostream>
using namespace std;
int main(){
    int i,num,f=0;
    cout<<"Enter the number:";cin>>num;
    for (i=2;i<num;i++)
      if (num%i==0) f=1;
      if (f==0) cout<<"PR";
      else cout<<"NPR";
    }
```





Functions :

1. Build in Function

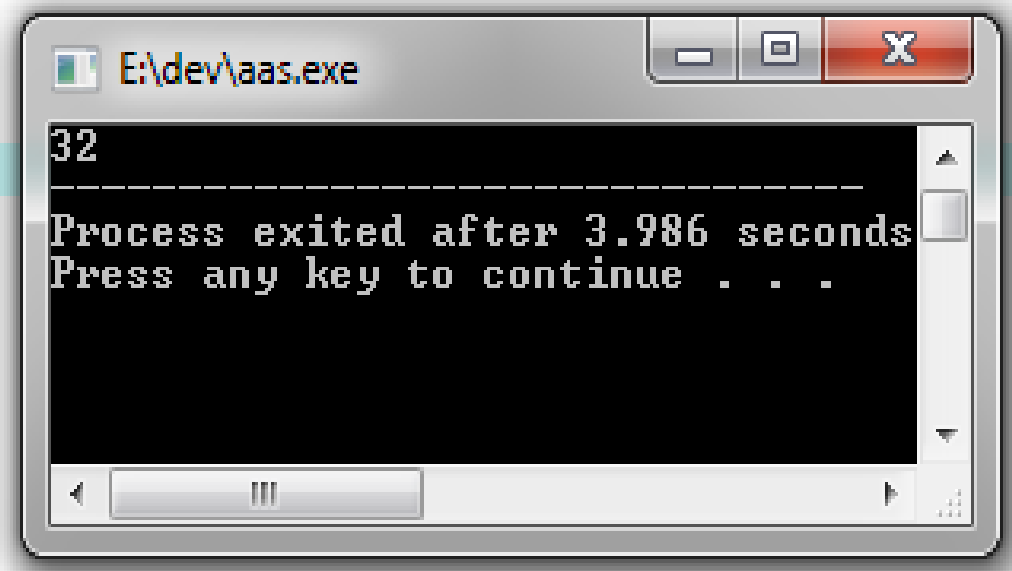
Built-in functions are also known as library functions. We need not to declare and define these functions as they are already written in the C++ libraries such as iostream, cmath etc. We can directly call them when we need.

Example: C++ built-in function example

Here we are using built-in function pow(x,y) which is x to the power y. This function is declared in cmath header file so we have included the file in our program using #include directive.

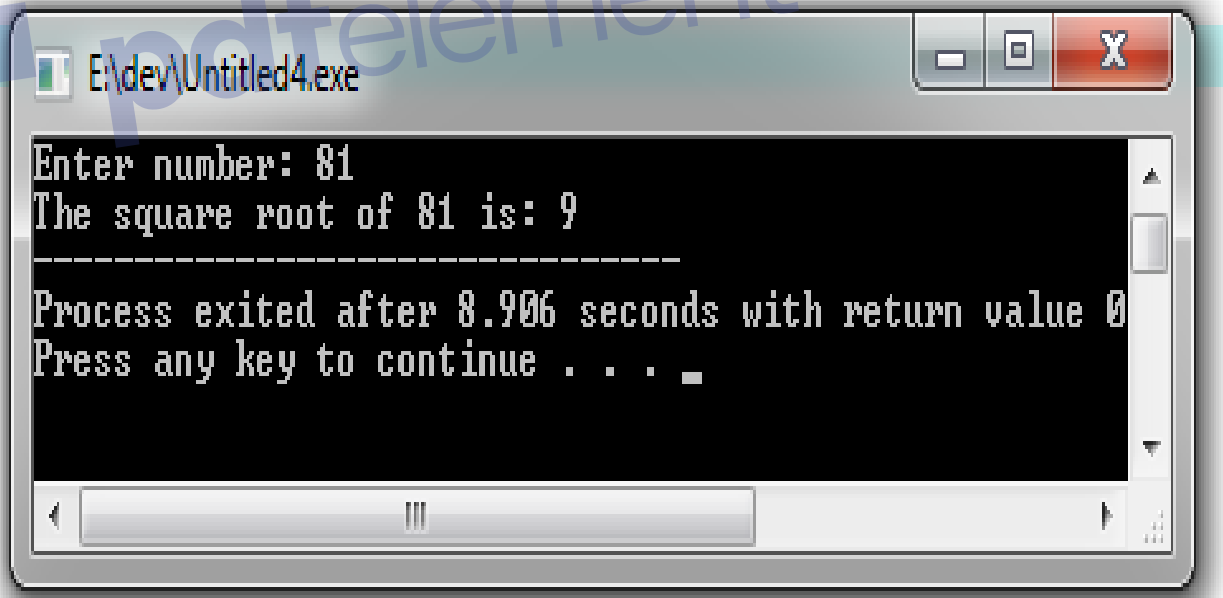
fun1.cpp | aaa.cpp | aas.cpp

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4  int main(){
5      cout<<pow(2,5);
6      return 0;
7  }
```



fun1.cpp | aaa.cpp | aas.cpp | Untitled4.cpp

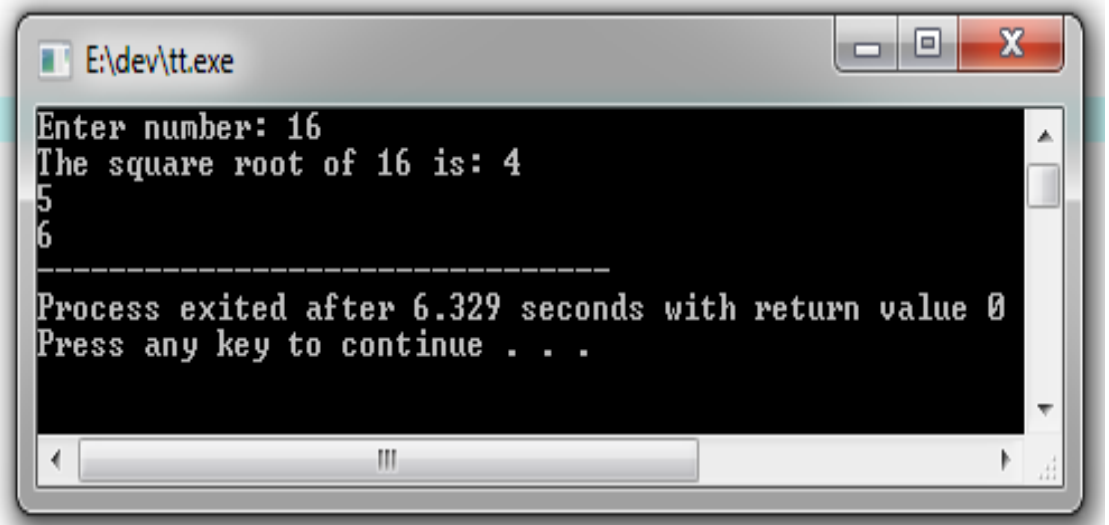

```cpp
#include <iostream>
#include <cmath>
using namespace std;
int main() {
    double num, squareRoot;
    cout << "Enter number: ";
    cin >> num;
    squareRoot = sqrt(num);
    cout << "The square root of " << num << " is: " << squareRoot;
    return 0;
}
```

Build in function :
sqrt(x) √x
fabs(x) |x|
floor(x) largest integer not greater than x; example: floor(5.768) = 5
ceil(x) smallest integer not less than x; example: ceil(5.768) = 6

tt.cpp

```cpp
#include <iostream>
#include <cmath>
using namespace std;
int main() {
    double num, squareRoot;
    cout << "Enter number: ";
    cin >> num;
    squareRoot = sqrt(num);
    cout << "The square root of " << num << " is: " << squareRoot<<"\n";
    cout <<floor(5.768)<<"\n";
    cout<<ceil(5.768) <<"    ";
    return 0;
}
```

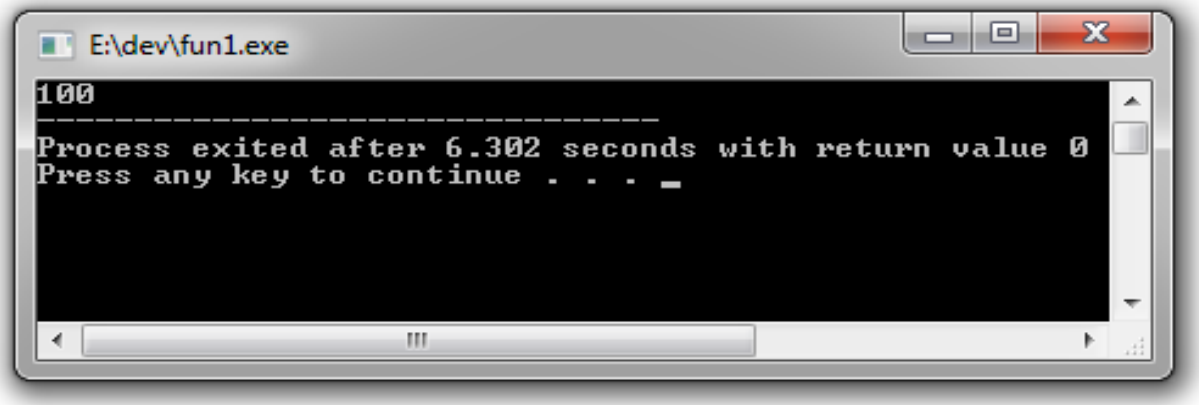
2. A function is a block of code which is used to perform a particular task, for example let's say you are writing a large C++ program and in that program you need to perform a particular task several number of times, like displaying value from 1 to 10, in order to do that you have to write few lines of code and you need to repeat these lines every time you display values. Another way of doing this is that you write these lines inside a function and call that function every time you want to display values. This would make you code simple, readable and reusable.

Syntax of Function

```
return_type function_name (parameter_list)
{
    //C++ Statements
}
```

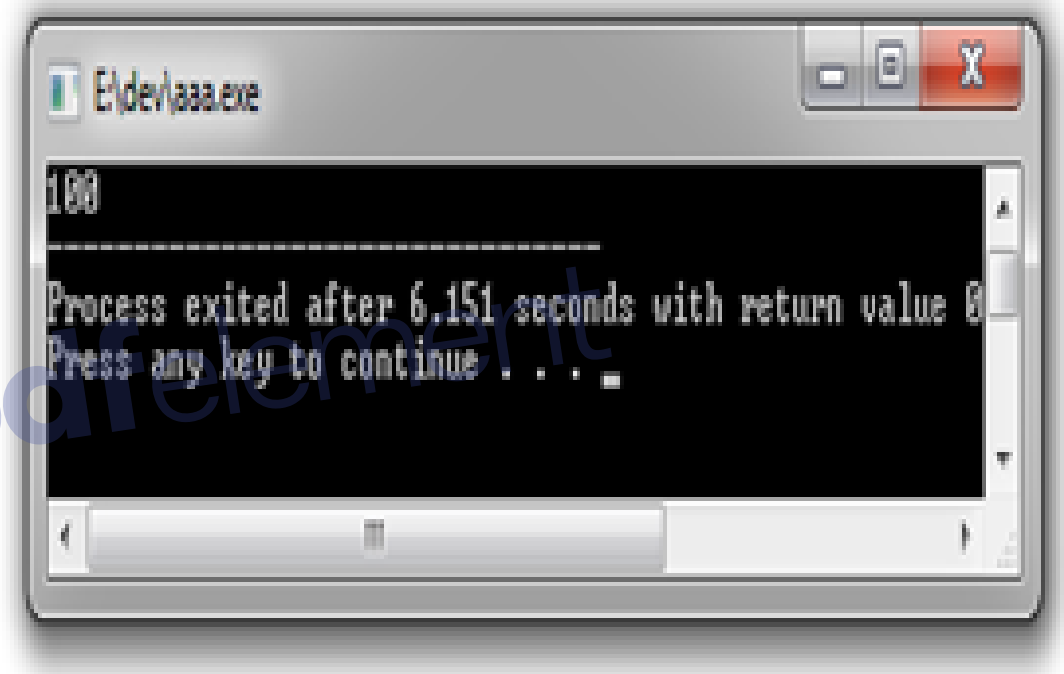


```
1   #include <iostream>
2   using namespace std;
3   /* This function adds two integer values
4    * and returns the result */
5    int sum(int num1, int num2){
6      int num3 = num1+num2; return num3;
7   }
8
9   int main(){
10     //Calling the function
11     cout<<sum(1,99);
12     return 0;
13  }
```



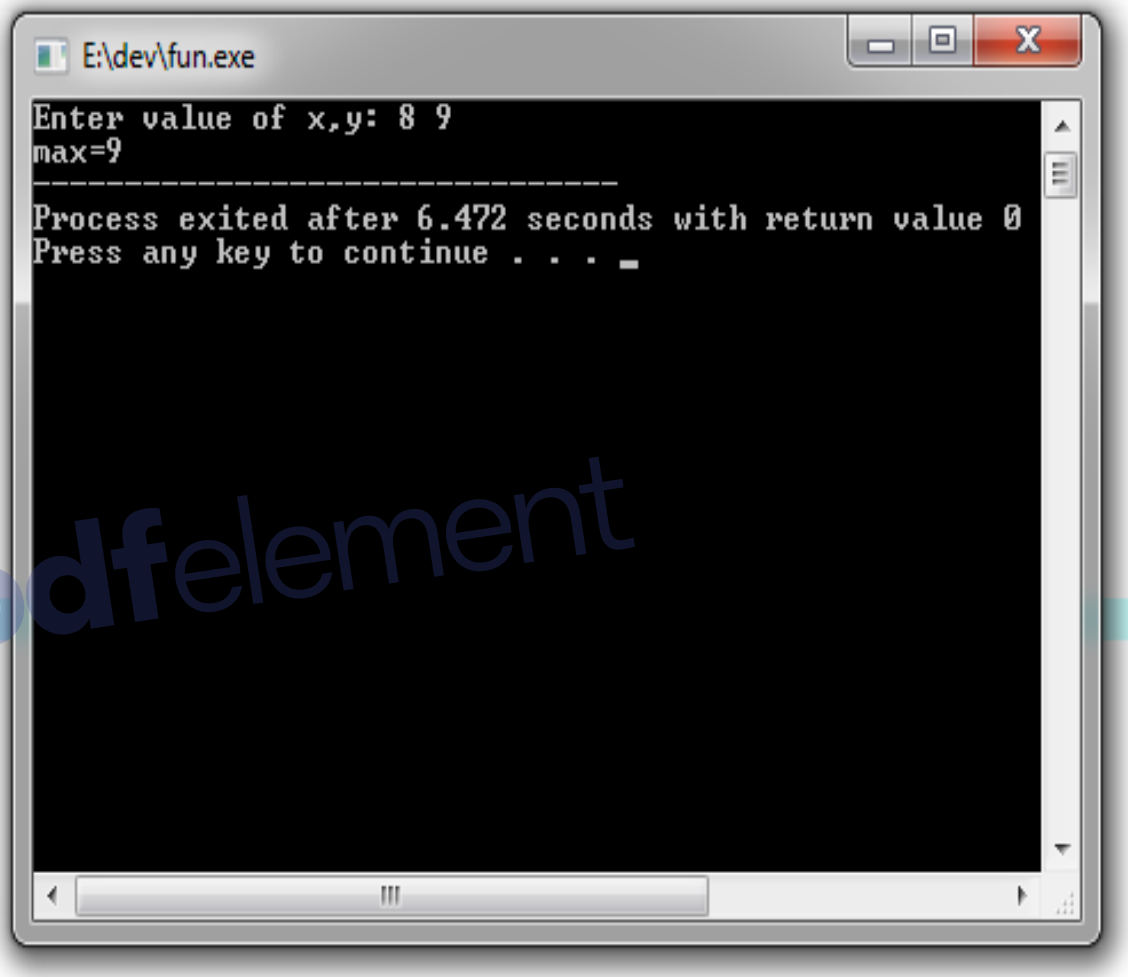
fun1.cpp | aaa.cpp

```cpp
#include <iostream>
using namespace std;
//Function declaration
int sum(int,int);

//Main function
int main(){
    //Calling the function
    cout<<sum(1,99);
    return 0;
}
/* Function is defined after the main method
 */
int sum(int num1, int num2){
    int num3 = num1+num2;
    return num3;
}
```

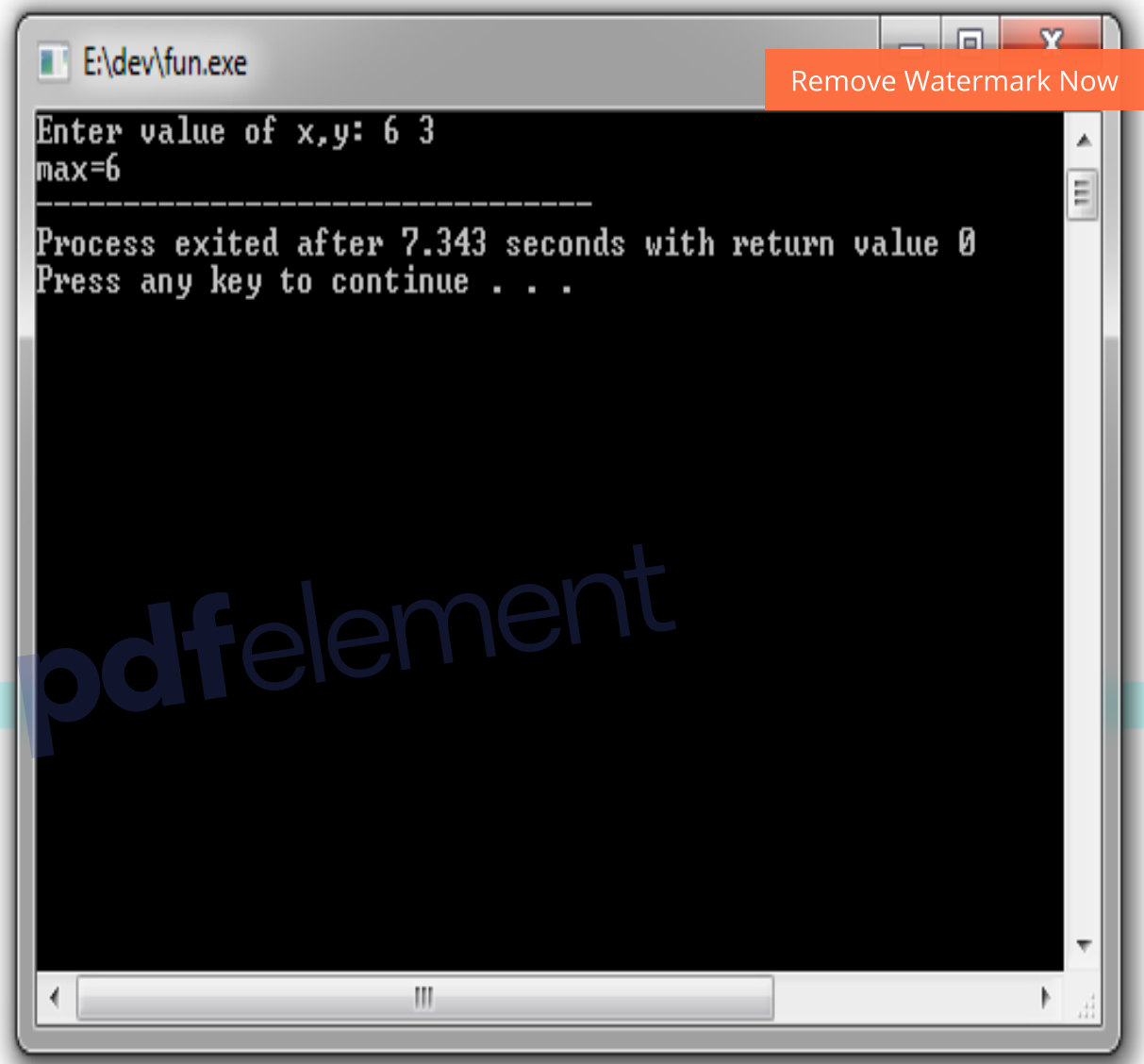


Example : Write a function for finding the biggest number between two numbers


```
1   #include <iostream>
2   using namespace std;
3   // prototype / declaratoin
4   int max(int a, int b)
5   {
6   if(a > b)
7   return a; // immediately stops max
8   else
9   return b; // immediately stops max
10  }
11  int main()
12  {
13  int x, y, mx;
14  cout<<"Enter value of x,y: ";
15  cin >> x >> y;
16  cout<<"max="<<max(x,y);
17
18  }
19
```

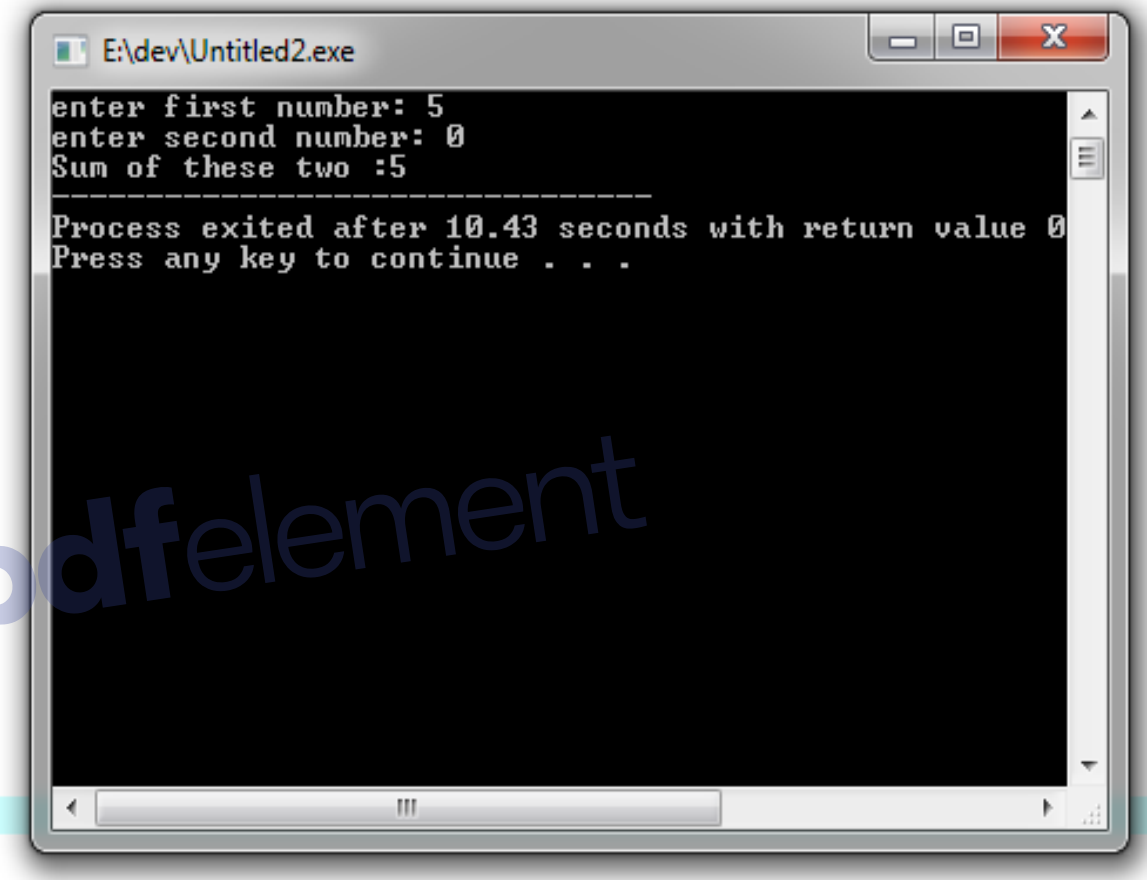


```cpp
#include <iostream>
using namespace std;
// prototype / declaratoin
void max(int a, int b)
{
if(a > b) cout <<a;
else
cout<<b;
}
int main()
{
int x, y, mx;
cout<<"Enter value of x,y: ";
cin >> x >> y;
cout<<"max=";
max(x,y);

}
```



Example


```
1   #include <iostream>
2   #include <cmath>
3   using namespace std;
4   //Declaring the function sum
5   int sum(int,int);
6
7   int main(){
8      int x, y;
9      cout<<"enter first number: ";
10     cin>> x;
11
12     cout<<"enter second number: ";
13     cin>>y;
14
15     cout<<"Sum of these two :"<<sum(x,y);
16     return 0;
17  }
18  //Defining the function sum
19  int sum(int a, int b) {
20     int c = a+b;
21     return c;
22  }
```



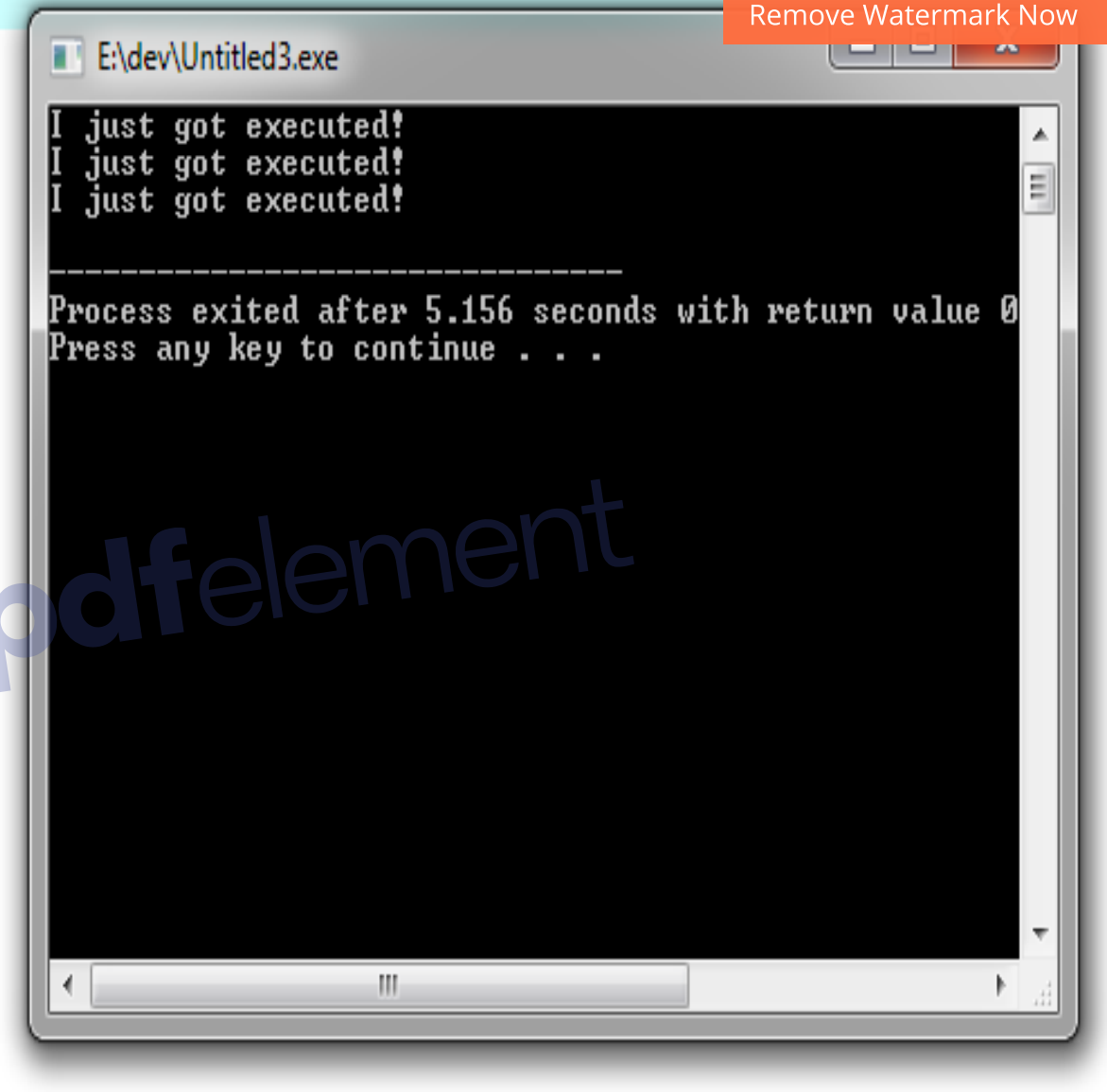
tt.cpp | ft.cpp

```cpp
#include <iostream>
using namespace std;

void sayHello() {
    cout << "Hello!";
}

int main() {

    sayHello();

    return 0;
}
```

E:\dev\ft.exe

```
Hello!
--------------------------------
Process exited after 1.621 seconds
Press any key to continue . . .
```

```cpp
#include <iostream>
using namespace std;
void myFunction() {
  cout << "I just got executed!\n";
}

int main() {
  myFunction();
  myFunction();
  myFunction();
  return 0;
}
```



Object-oriented programming

OOP stands for Object-Oriented Programming.

Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.

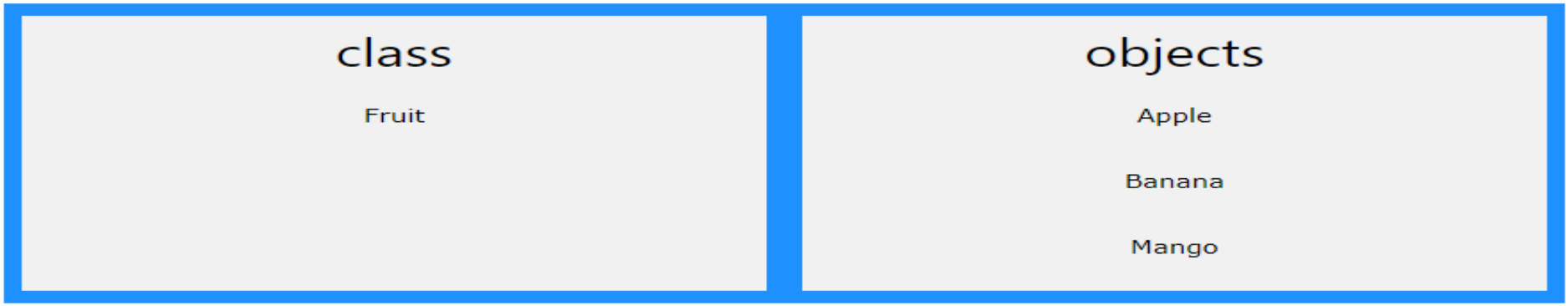
Object-oriented programming has several advantages over procedural programming:

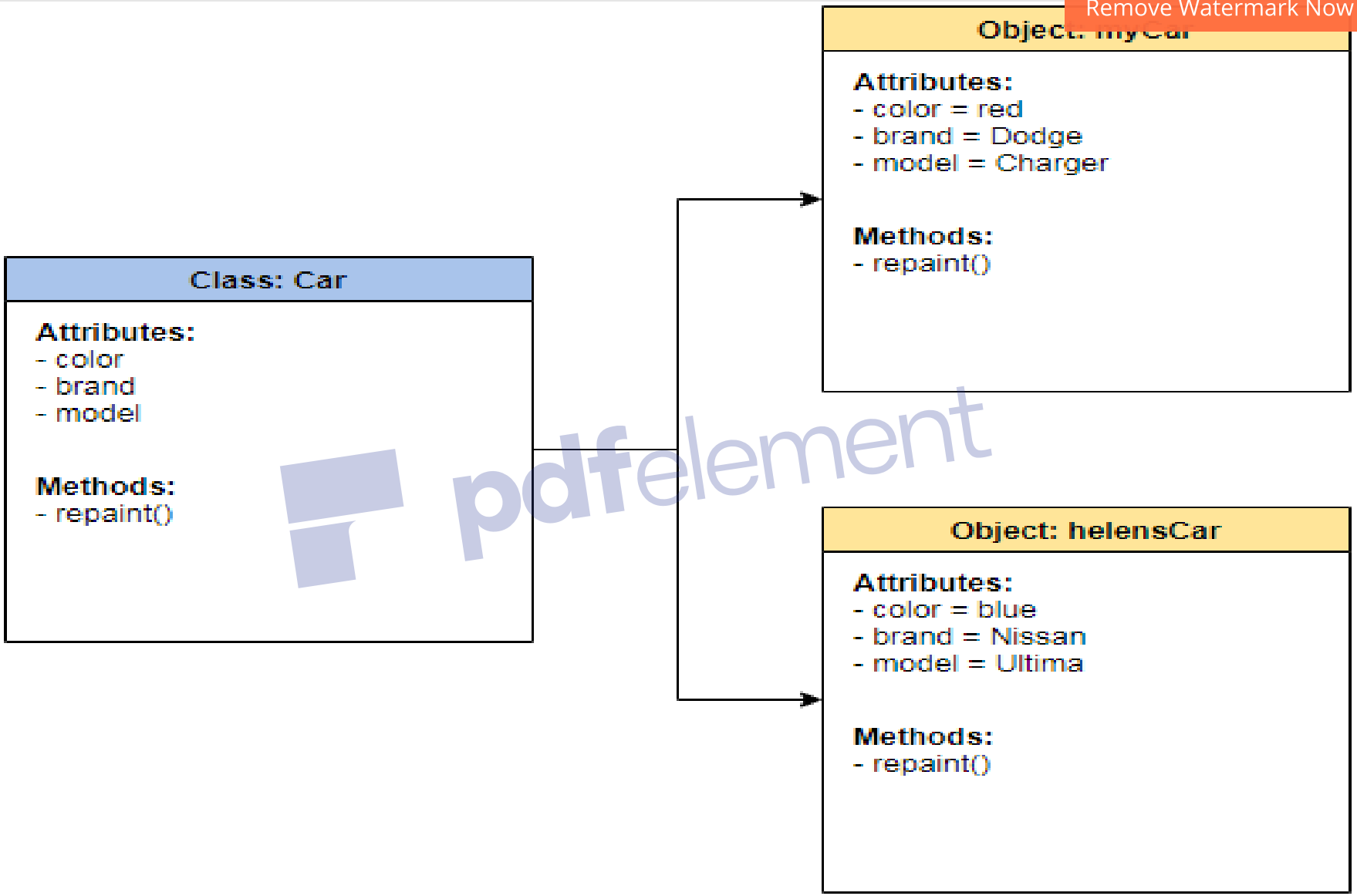
- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the C++ code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time.

C++ What are Classes and Objects?

Classes and objects are the two main aspects of object-oriented programming.

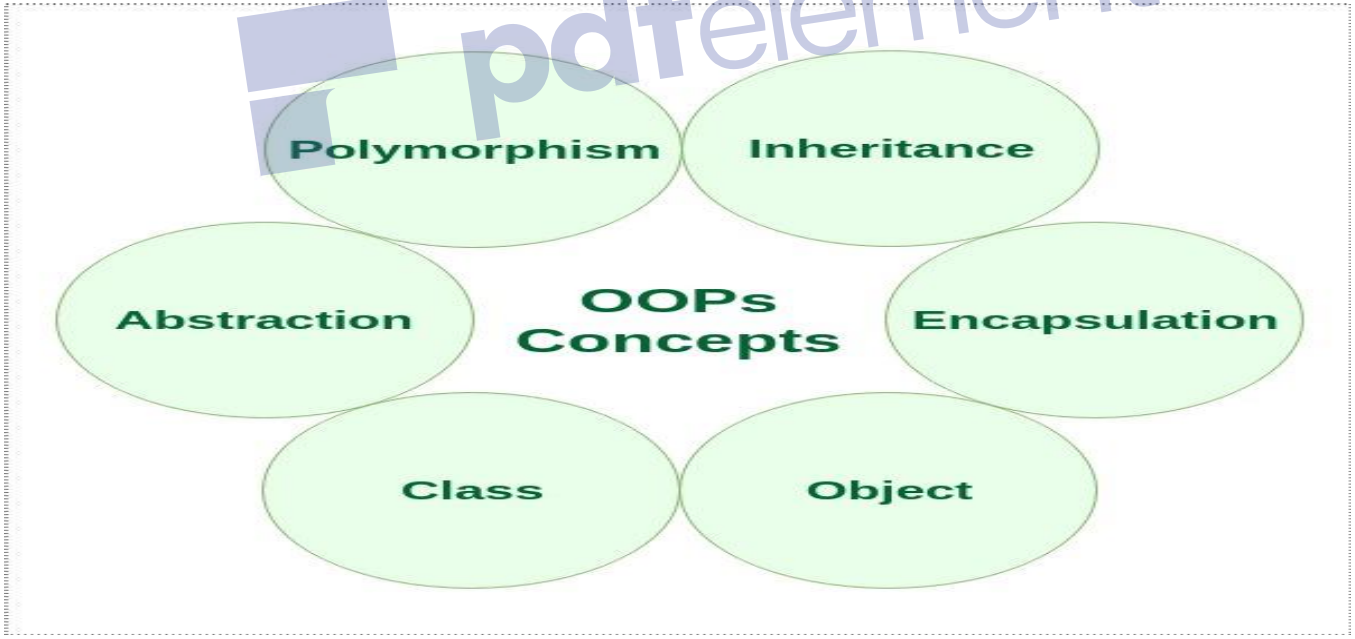
Look at the following illustration to see the difference between class and objects:





Object-oriented programming – As the name suggests uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc. in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

Characteristics of an Object Oriented Programming language



Class: The building block of C++ that leads to Object-Oriented programmin[illegible] a Class, it is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

For Example: Consider the Class of Cars. There may be many cars with different names and brand but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range etc. So here, Car is the class and wheels, speed limits, mileage are their properties.

- A Class is a user-defined data-type which has data members and member functions.
- Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions define the properties and behaviour of the objects in a Class.
- In the above example of class Car, the data member will be speed limit, mileage etc and member functions can apply brakes, increase speed etc.

Object: An Object is an identifiable entity with some characteristics and behaviour. An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.


```
#include <iostream>
class person
{
char name[20];
int id;
public:
void getdetails(){}
};

int main()
{
person p1; // p1 is a object
}
```

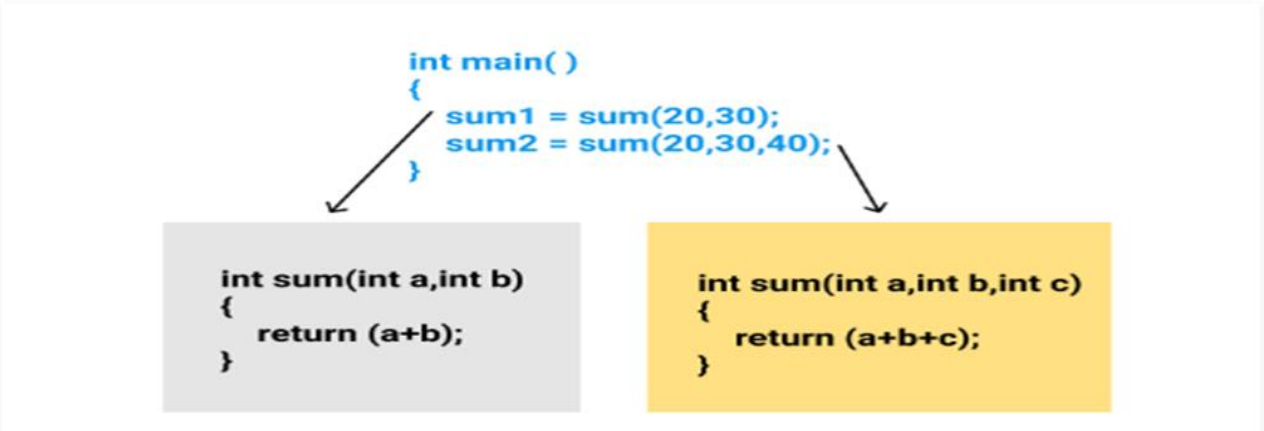

When a program is executed the objects interact by sending messages to one another. Each object contains data and code to manipulate the data. Objects can interact without having to know details of each other's data or code, it is sufficient to know the type of message accepted and type of response returned by the objects.

Encapsulation: In normal terms, Encapsulation is defined as wrapping up of data and information under a single unit. In Object-Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulate them.

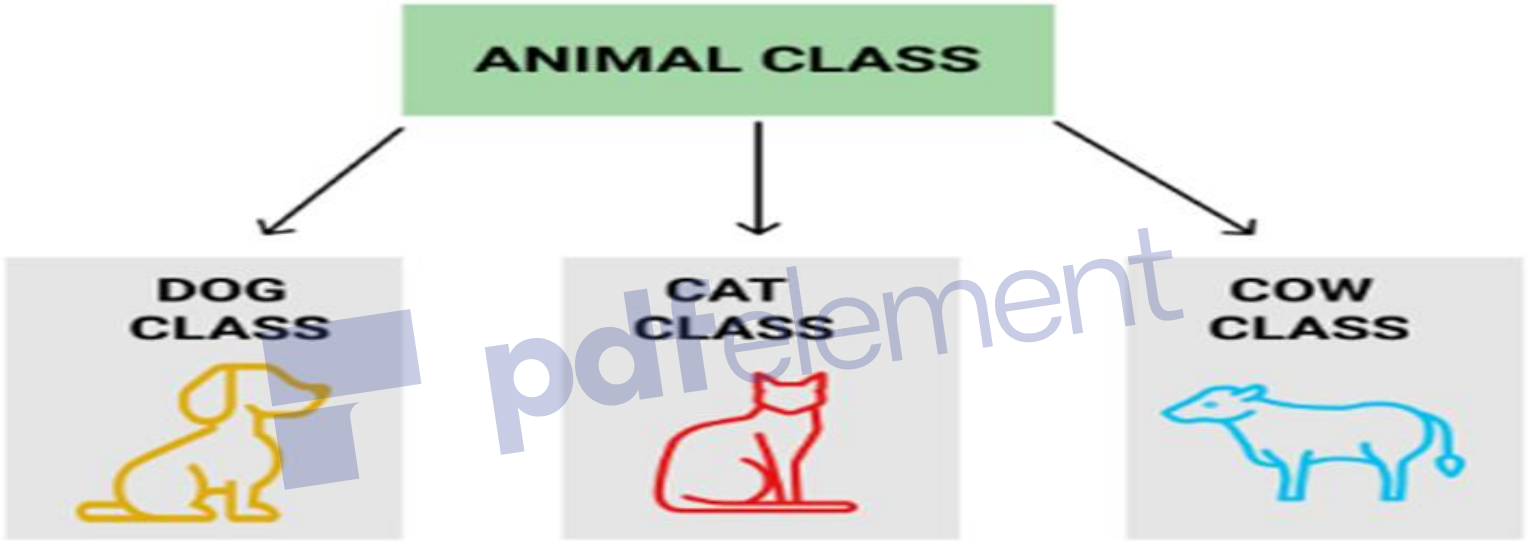
Abstraction: Data abstraction is one of the most essential and important features of object-oriented programming in C++. Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

Polymorphism: The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

Example: Suppose we have to write a function to add some integers, some times there are 2 integers, some times there are 3 integers. We can write the Addition Method with the same name having different parameters, the concerned method will be called according to parameters.



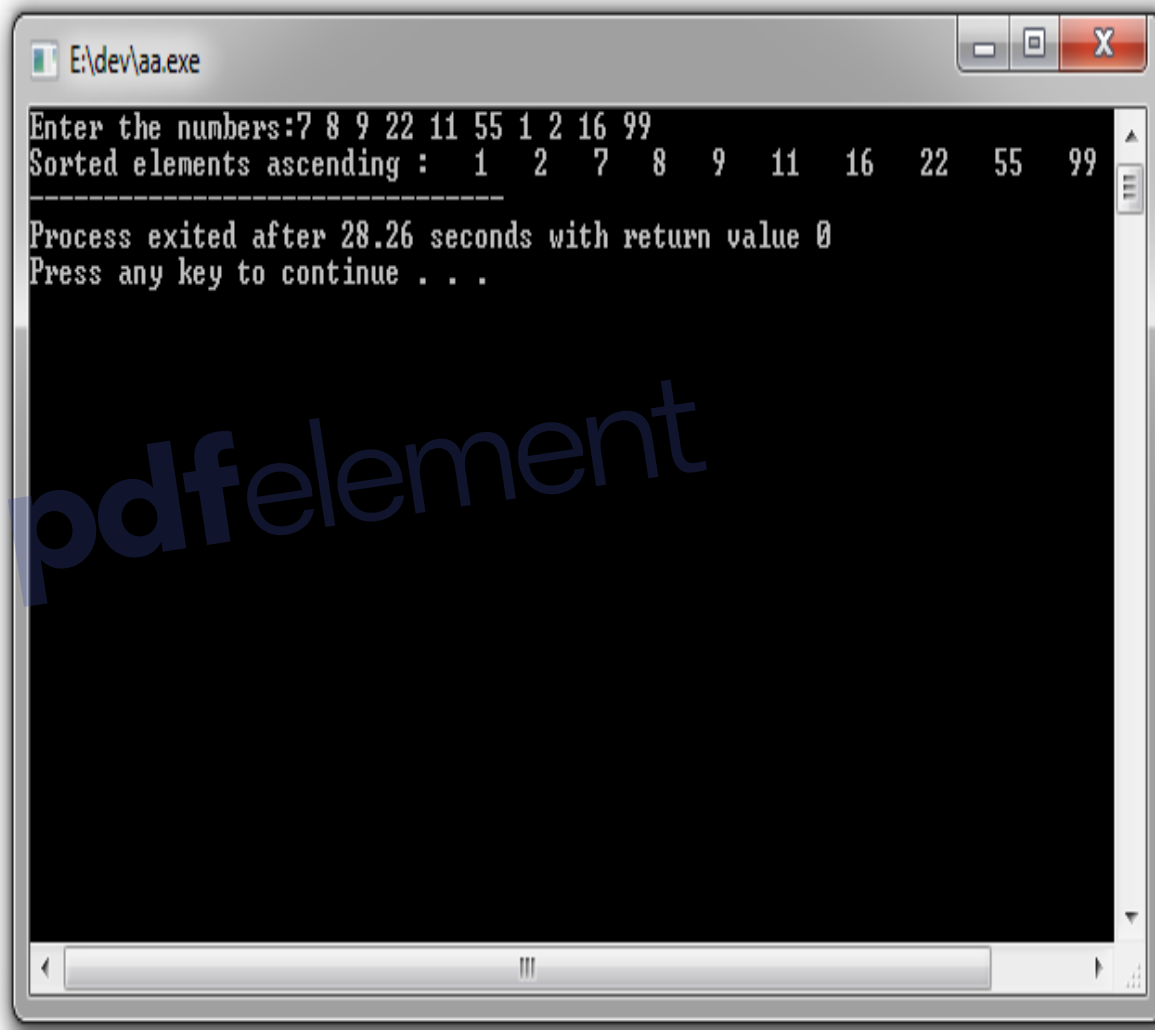
Inheritance: The capability of a class to derive properties and characteristics from another class is called Inheritance. Inheritance is one of the most important features of Object-Oriented Programming.



Dynamic Binding: In dynamic binding, the code to be executed in response to function call is decided at runtime. C++ has virtual functions to support this.

Message Passing: Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function and the information to be sent.

```cpp
//Write C++ program for sort 10 integer number ascending.

#include <iostream>
using namespace std;
int main(){
    int i,j,t,num[10];
    cout<<"Enter the numbers:";
    for(i=0;i<=9;i++)cin>>num[i];
    for(i=0;i<=9;i++)
    for (j=i+1;j<10;j++)
      if (num[i]>num[j]) {t=num[i];
                          num[i]=num[j];
                          num[j]=t; }
cout<<"Sorted elements ascending :";
    for(i=0;i<=9;i++)cout<<"   "<< num[i];



    }
```



```cpp
//Write C++ program for sort 10 integer number descending

#include <iostream>
using namespace std;
int main(){
    int i,j,t,num[10];
    cout<<"Enter the numbers:";
    for(i=0;i<=9;i++)cin>>num[i];
    for(i=0;i<=9;i++)
    for (j=i+1;j<10;j++)
      if (num[i]<num[j]) {t=num[i];
                          num[i]=num[j];
                          num[j]=t; }
cout<<"Sorted elements ascending :  ";
    for(i=0;i<=9;i++)cout<<"   "<< num[i];


    }
```



```
E:\dev\aa.exe
Enter the numbers:7 8 13 5 66 00 99 56 44 33
Sorted elements ascending :     99   66   56   44   33   13   8   7   5   0
--------------------------------

Process exited after 23.9 seconds with return value 0
Press any key to continue . . .
```

Homework:

Q1. Write a function for checking any number if it is prime and use it for checking elements of integer number of an array a[10]

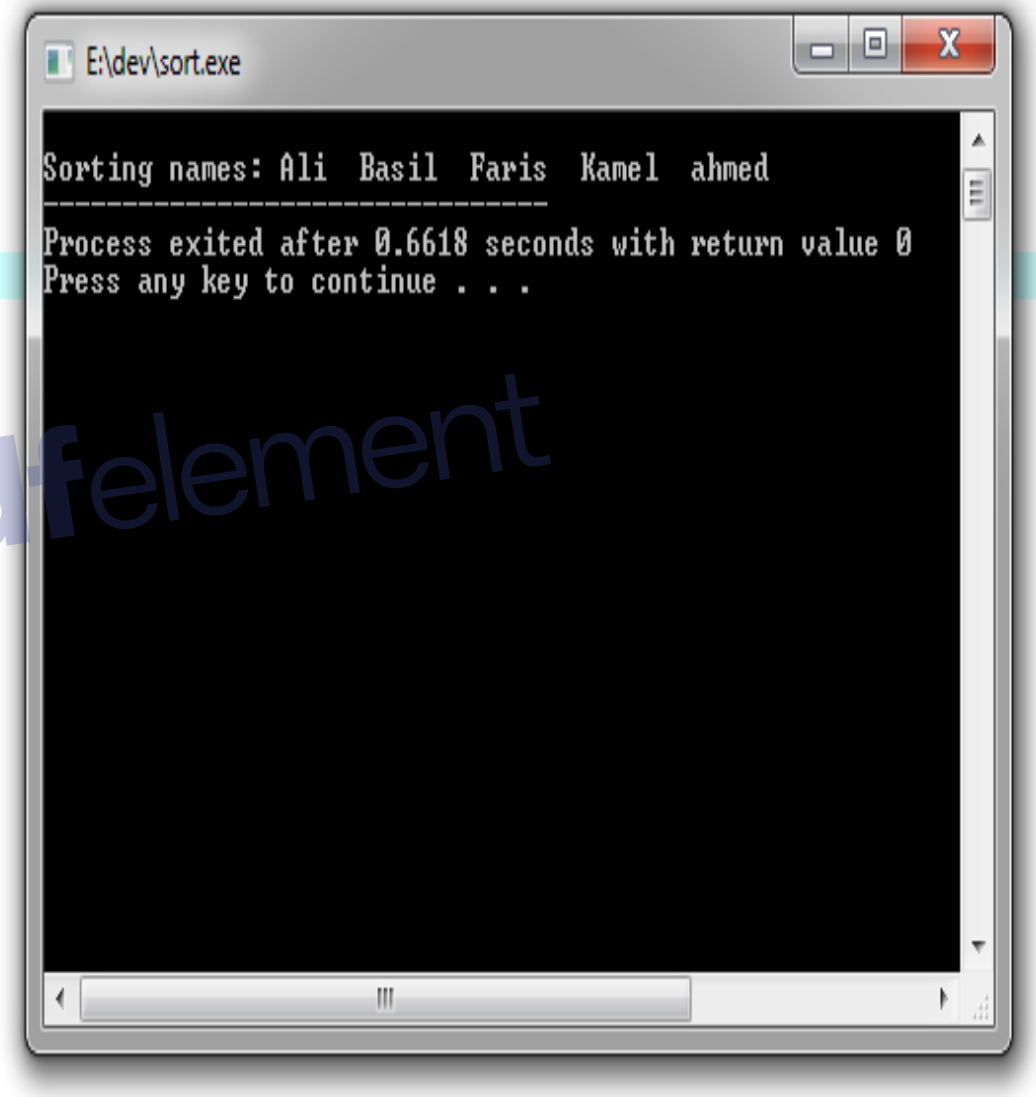
Q2. Write a function for finding the odd number and use it for finding the sum o f odd number of an integer array mark[10].

Q3.Write a function for sorting string array ascending and use it for sorting 5 names ascending

Q3.Write a function for sorting string array ascending and use it for sorting 5 names ascending

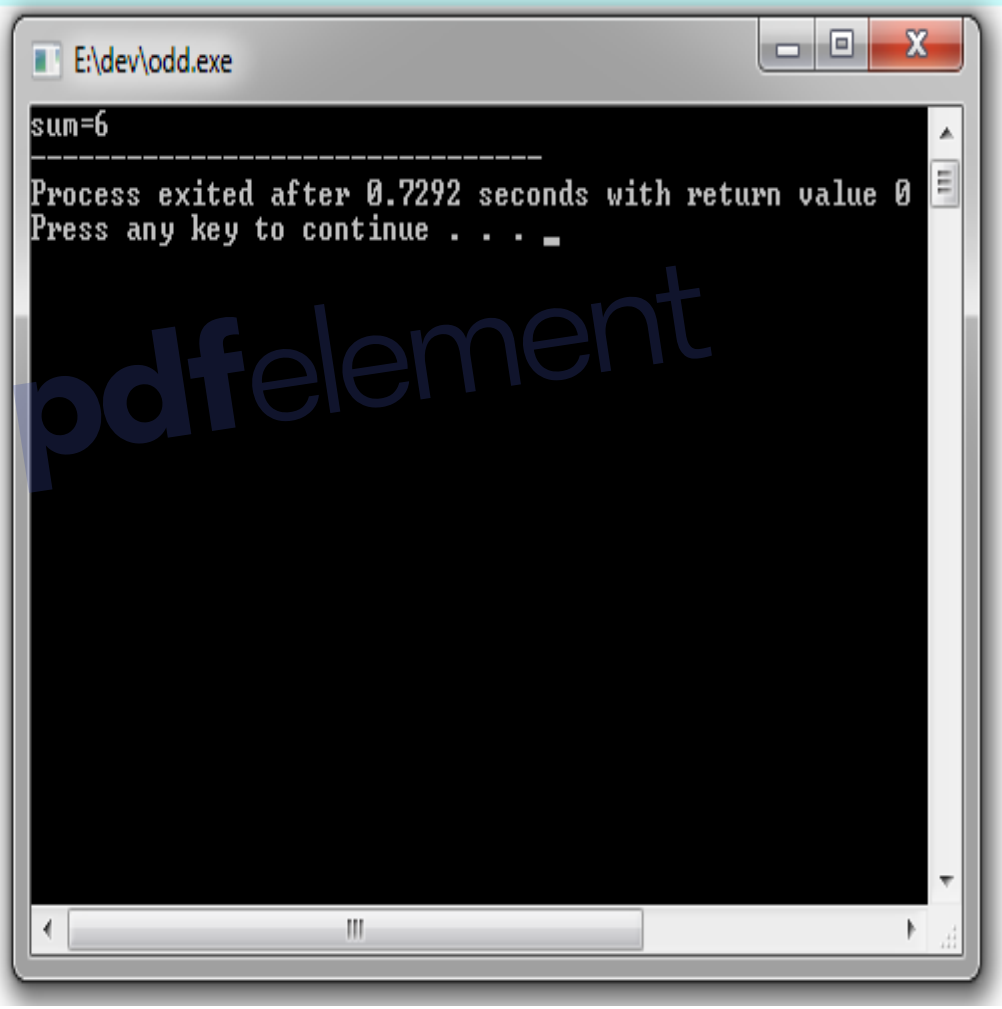
sort.cpp


```
1  // Q3.Write a function for sorting string  array ascending and use it for sorting 5 names ascending
2  #include <iostream>
3  using namespace std;
4  void xx(string name[5])
5  { int i,j;string x;
6     for (i=0;i<=3;i++)
7       for (j=i+1;j<=4;j++)
8       if (name[i]>name[j]) {
9                            x=name[i];
10                           name[i]=name[j];
11                           name[j]=x;}
12    cout<<"\n"<<"Sorting names: " ;
13     for (i=0;i<=4;i++) cout<< name[i]<<"  ";
14
15       }
16 int main()
17 { string na[5]={"Ali","ahmed","Basil","Faris","Kamel"};
18 xx(na);
19          }
```



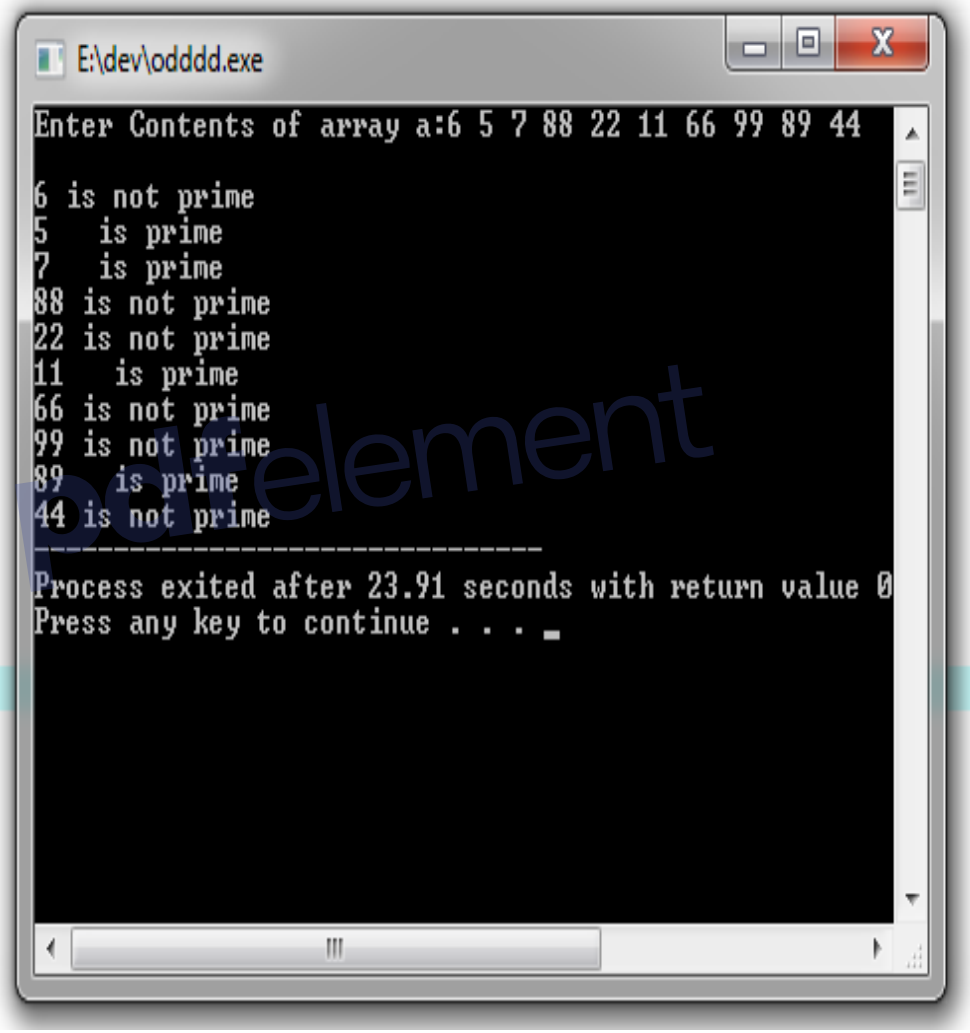
Q2. Write a function for finding the odd number and use it for finding the sum of odd number of an integer array mark[10].


```
// Q2. Write a function for finding the odd number and use it for finding   the number o f odd number of an integer array mark[10].
#include <iostream>
using namespace std;
int odd(int x)
{ int f;
if (x%2==1) f=1;
else f=0;
return f;
 }
 int main()
 { int sum=0, i,a[10]={2,3,4,5,7,13,8,9,33,10};
 for (i=0;i<=9;i++)
 sum=sum+odd(a[i]);
 cout <<"sum="<<sum;
 }
```



Q1. Write a function for checking any number if it is prime or not . And checking elements of integer number of an array a


```
1  //Q1. Write a function for checking any number if it is prime or not . And checking elements of integer number of an array a
2  #include <iostream>
3  using namespace std;
4  void prime(int x)
5  { int i,f=0;
6  for (i=2;i<x;i++)
7  if (x%i==0) f=1;
8  if (f==0) cout <<"\n"<<x<<"   is prime";
9  else cout<<"\n"<<x<<" is not prime";
10 }
11 int main()
12 {int i,a[10];
13 cout<<"Enter Contents of array a:";
14 for (i=0;i<=9;i++)
15 cin >>a[i];
16 for (i=0;i<=9;i++)
17 prime(a[i]);
18 return 0;
19 }
```



Object Oriented Programming

Class

A class is an expanded concept of a data structure: instead of holding only data, it can hold both data and functions.

An object is an instantiation of a class. In terms of variables, a class would be the type, and an object would be the variable. The format of the class:

```
class class_name {
access_specifier_1:
member1;
access_specifier_2:
member2;
...
} object_names;
```

Where class_name is a valid identifier for the class, object_names is an optional list of names for objects of this class. The body of the declaration can contain members, that can be either data or function declarations, and optionally access specifiers. All is very similar to the declaration on data structures, except that we can now include also functions and members, but also this new thing called *access specifier*. An access specifier is one of the following three keywords: private, public or protected. These specifiers modify the access rights that the members following them acquire:

- private members of a class are accessible only from within other members of the same class or from their *friends*.
- protected members are accessible from members of their same class and from their friends, but also from members of their derived classes.
- Finally, public members are accessible from anywhere where the object is visible.

By default, all members of a class declared with the class keyword have private access for all its members. Therefore, any member that is declared b

class specifier automatically has private access. For example:

```
class CRectangle {
        int x, y;
    public:
        void set_values (int,int);
        int area (void);
        } rect;
```

Declares a class (i.e., a type) called CRectangle and an object (i.e., a variable) of this class called rect. This class contains four members: two data members of type int (member x and member y) with private access (because private is the default access level) and two member functions with public access: set_values() and area(), of which for now we have only included their declaration, not their definition.

Notice the difference between the class name and the object name: In the previous example, CRectangle was the class name (i.e., the type), whereas rect was an object of type CRectangle. It is the same relationship int and a have in the following declaration:

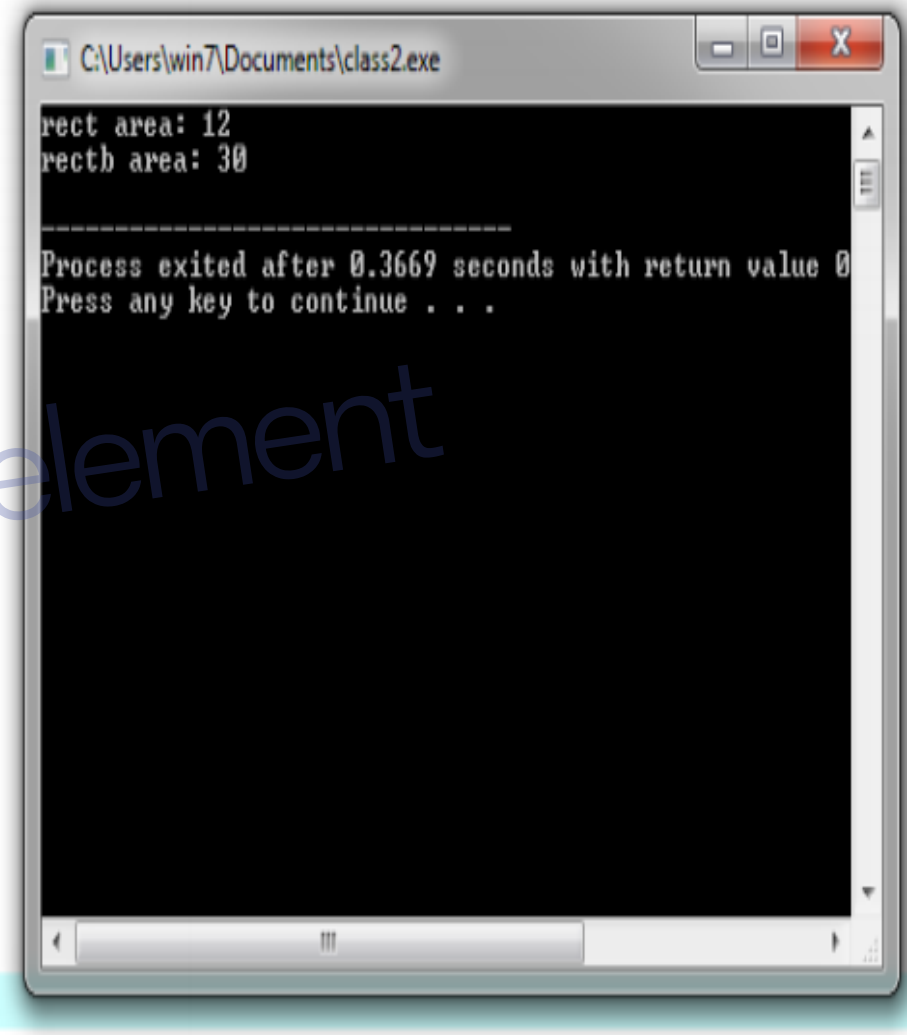
```
int a;
```

where int is the type name (the class) and a is the variable name (the object).

After the previous declarations of CRectangle and rect, we can refer within the body of the program to any of the public members of the object rect as if they were normal functions or normal variables, just by putting the object's name followed by a dot (.) and then the name of the member. All very similar to what we did with

class1.cpp | class2.cpp

```
2  #include <iostream>
3  using namespace std;
4  class CRectangle {
5  int x, y;
6  public:
7  void set_values (int,int);
8  int area () {return (x*y);}
9  };
10 void CRectangle::set_values (int a, int b) {
11 x = a;
12 y = b;
13 }
14 int main () {
15 CRectangle rect, rectb;
16 rect.set_values (3,4);
17 rectb.set_values (5,6);
18 cout << "rect area: " << rect.area() << endl;
19 cout << "rectb area: " << rectb.area() << endl;
20 return 0;
21 }
```



Q1: Write an O.O.P for finding the value of Y:

$$Y = \overset{1}{1/2} + \overset{2}{2/2} + \overset{3}{3/2} + \ldots + \overset{n}{n/2}$$

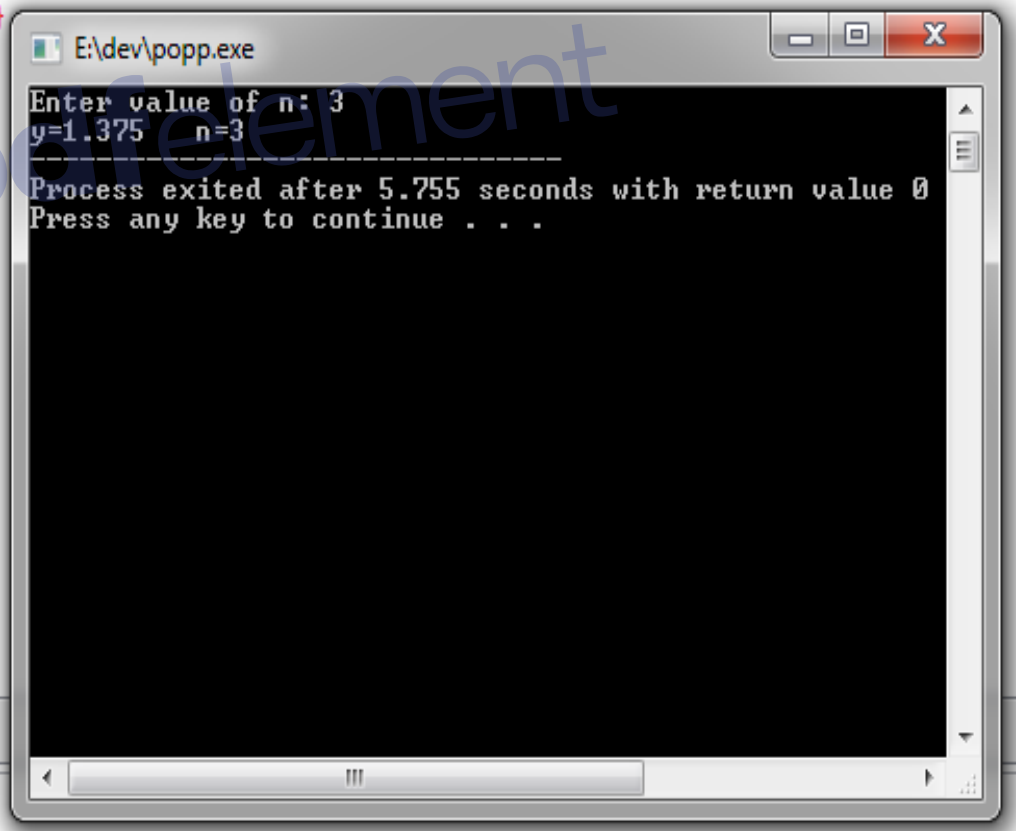
Q2 : If you have an int a[5][5] .Write an O.O.P for finding :

1. The sum of main diameter of a.
2. The sum of second diameter of a.
3. The sum of each column of a.
4. The sum of each row of a.
5. Print the elements of Up triangle of a.
6. Print the elements of Down triangle of a.

Q1: Write an O.O.P for finding the value of Y:

$$Y = 1/2^{1} + 2/2^{2} + 3/2^{3} + \ldots + n/2^{n}$$

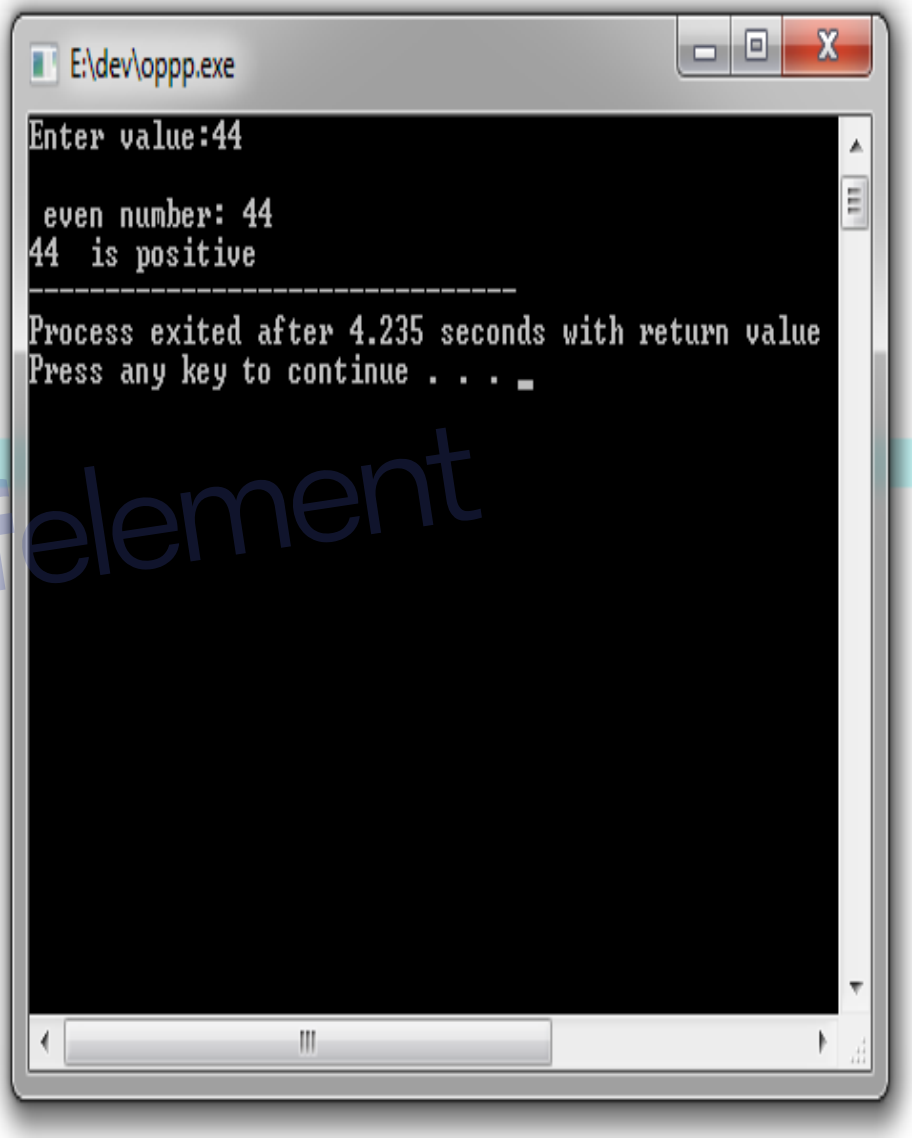

```
1   //                                1     2     3          n
2   // write an O.O.P for finding y=1/2 + 2/2 + 3/2 +.... + n/2
3   #include <iostream>
4   #include <cmath>
5   using namespace std;
6   class ser{ private:
7         int n;
8         public:
9       float get_n(){ cout<<"Enter value of n: "; cin>>n;return n; }
10       float po(int x){ int i=pow(2,x); return i;}
11        };
12  int main()
13  { ser A;float y=0,i;
14  int n;
15  n=A.get_n();
16   for (i=1;i<=n;i++)
17   y=y+i/A.po(i);
18   cout <<"y="<<y<<"    "<<"n="<<n;
19  return 0;
20  }
```



Compile Log | Debug | Find Results | Close

```
- Errors: 0
- Warnings: 0
```

```cpp
// Write an O.O.p for cheking any number if it is 1. positive or not 2. even or odd
#include <iostream>
using namespace std;
class check
{
 int n;
 public:
    int get_n() { cout<<"Enter value:";cin>>n; return n;}
     void ch_even()
{ if (n%2==0) cout <<endl<<" even number: "<<n ;
   else cout<<endl<<" odd number :"<<n; }
   void ch_neg(){ if (n<0) cout <<endl<<n<<"  is negative";
                  else cout<<endl<<n <<"  is positive";  }
 };
int main()
{ check x;
x.get_n();
x.ch_even();
x.ch_neg();
return 0;
 }
```



Note: error program for loosing the library <iostream>

Note : #include <iostream>

Includes the declaration of the basic standard input-output library in c++ and it is functionality is going to be used later in the program.

Using namespace std:

Namespaces are containers that contain the declarations of all the elements of the standard C++ library

```
1  //Example: C++ Structure
2  //C++ Program to assign data to members of a structure variable and display it.
3  //#include <iostream>
4  using namespace std;
5  struct Person
6  {
7      string  name;
8      int age;
9      float salary;
10 };
11 int main()
12 {     Person p1[5];int i;
13     for (i=0;i<=4;i++)
14   { cout << "Enter Full name: ";
15     cin>>p1[i].name;
16     cout << "Enter age: ";
17     cin >> p1[i].age;
18     cout << "Enter salary: ";
19     cin >> p1[i].salary;}
20        cout << "\nDisplaying Information." << endl;
21     for (i=0;i<=4;i++)
22    {
23     cout << "Name: " << p1[i].name << endl;
24     cout <<"Age: " << p1[i].age << endl;
25     cout << "Salary: " << p1[i].salary;}
26     return 0;
27 }
28
```


Compiler (8) | Resources | Compile Log | Debug | Find Results | Close

Line	Col	File	Message
7	5	E:\dev\struc.cpp	[Error] 'string' does not name a type
		E:\dev\struc.cpp	**In function 'int main()':**
14	5	E:\dev\struc.cpp	[Error] 'cout' was not declared in this scope
15	5	E:\dev\struc.cpp	[Error] 'cin' was not declared in this scope
15	16	E:\dev\struc.cpp	[Error] 'struct Person' has no member named 'name'

How to declare a structure in C++ programming?

The `struct` keyword defines a structure type followed by an identifier (name of the structure).

Then inside the curly braces, you can declare one or more members (declare variables inside curly braces) of that structure. For example:

```
struct Person
{
    char name[50];
    int age;
    float salary;
};
```


How to define a structure variable?

Once you declare a structure `person` as above. You can define a structure variable as:

```
Person bill;
```

Here, a structure variable `bill` is defined which is of type structure P`erson`.

When structure variable is defined, only then the required memory is allocated by the compiler.

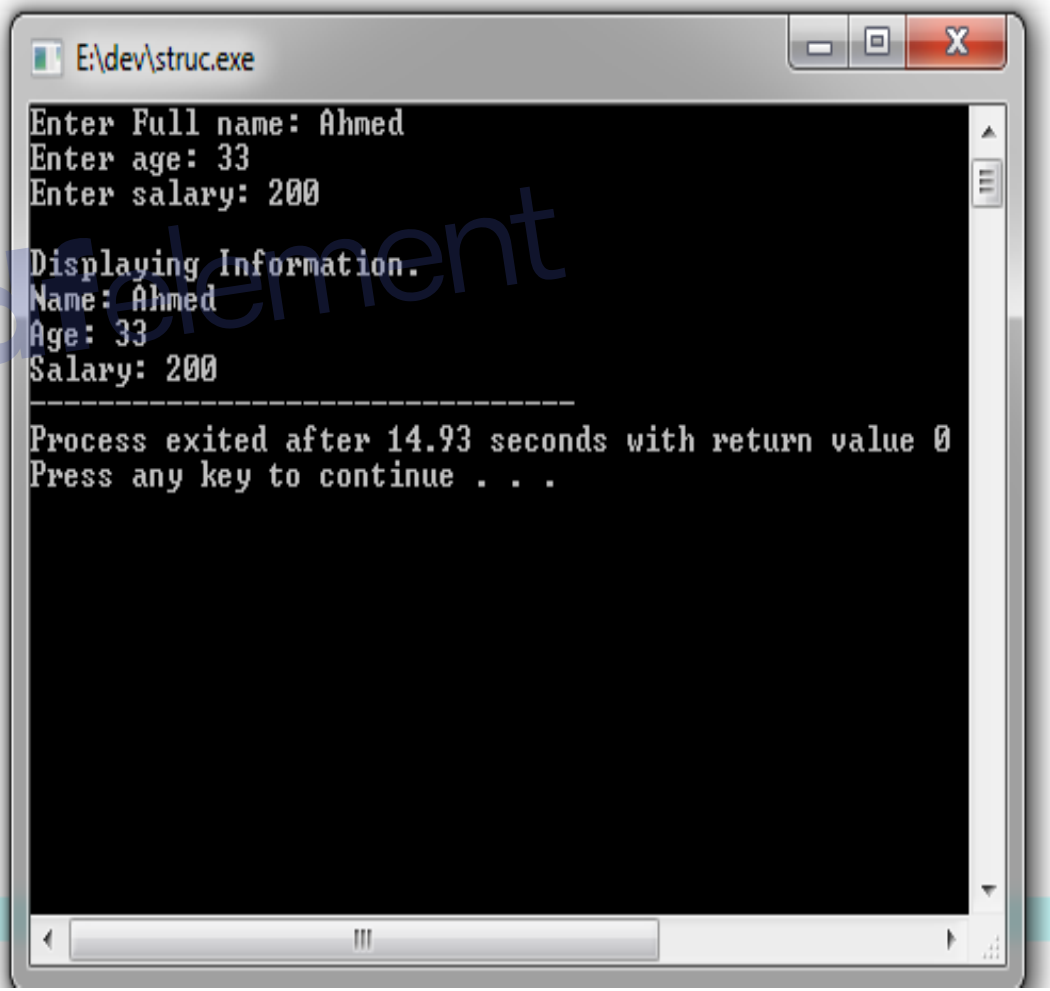
How to access members of a structure?

The members of structure variable is accessed using a **dot (.)** operator.

Suppose, you want to access `age` of structure variable `bill` and assign it 50 to it. You can perform this task by using following code below:

```
bill.age = 50;
```

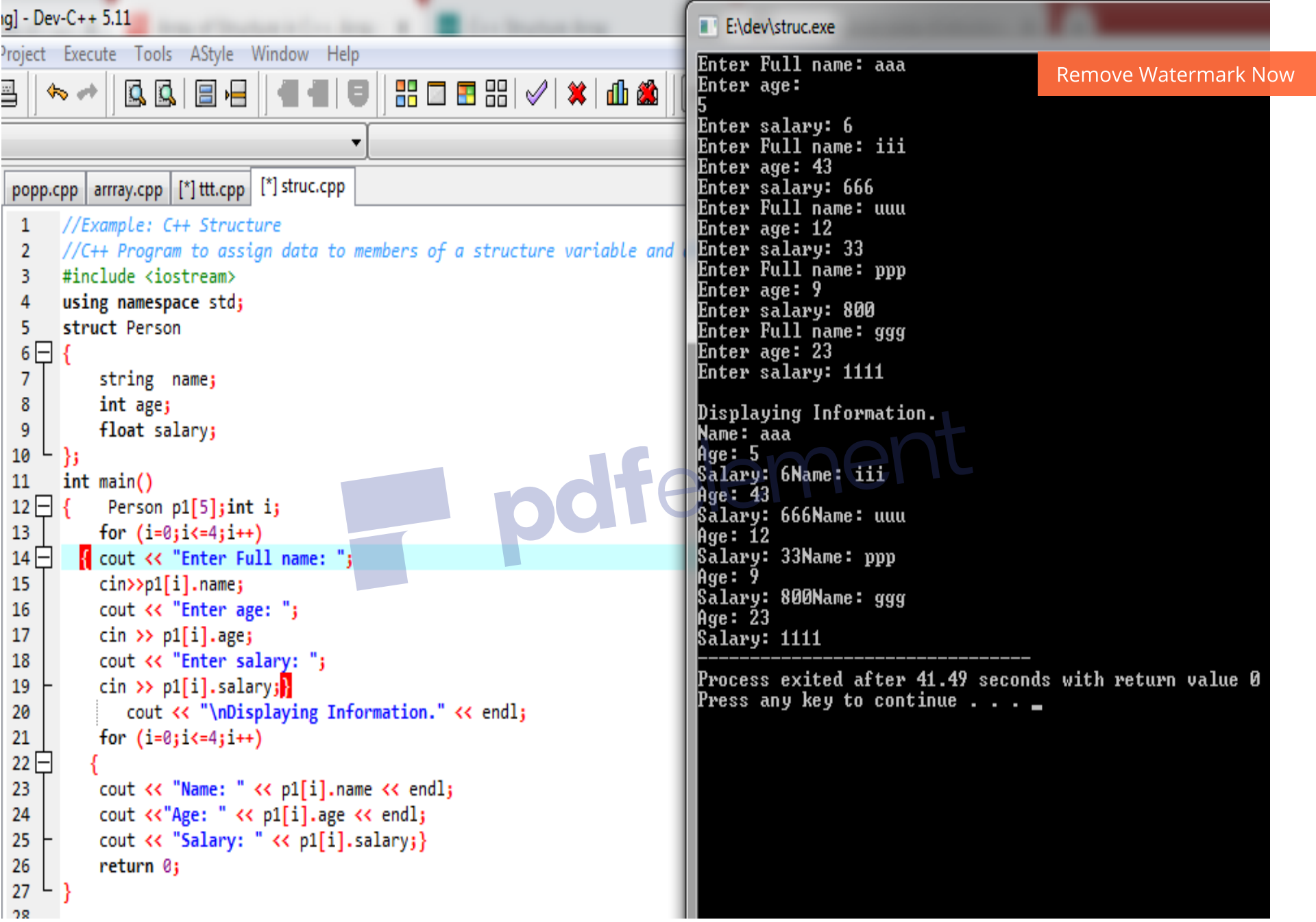
```
1  //Example: C++ Structure
2  //C++ Program to assign data to members of a structure variable and display it.
3  #include <iostream>
4  using namespace std;
5  struct Person
6  {
7      string  name;
8      int age;
9      float salary;
10 };
11 int main()
12 {
13     Person p1;
14     cout << "Enter Full name: ";
15     cin>>p1.name;
16     cout << "Enter age: ";
17     cin >> p1.age;
18     cout << "Enter salary: ";
19     cin >> p1.salary;
20     cout << "\nDisplaying Information." << endl;
21     cout << "Name: " << p1.name << endl;
22     cout <<"Age: " << p1.age << endl;
23     cout << "Salary: " << p1.salary;
24     return 0;
25 }
```



```cpp
//Example: C++ Structure
//C++ Program to assign data to members of a structure variable and display it.
#include <iostream>
using namespace std;

int main()
{struct Person
{
    string  name;
    int age;
    float salary;
};
    Person p1;
    cout << "Enter Full name: ";
    cin>>p1.name;
    cout << "Enter age: ";
    cin >> p1.age;
    cout << "Enter salary: ";
    cin >> p1.salary;
    cout << "\nDisplaying Information." << endl;
    cout << "Name: " << p1.name << endl;
    cout <<"Age: " << p1.age << endl;
    cout << "Salary: " << p1.salary;
    return 0;
}
```

```
E:\dev\struc.exe
Enter Full name: eeeee
Enter age: 55
Enter salary: 666

Displaying Information.
Name: eeeee
Age: 55
Salary: 666
--------------------------------
Process exited after 17.98 seconds with return value 0
Press any key to continue . . .
```


Note: structure can be written before or after main()



H.W.

Q1. Write a program to find the value of y from the following (using switch statement).

$$y = \begin{cases} \sqrt{(x+5)^3} & x = -1 \\ 2x^3 + 5x + 4 & x = 0 \\ x - \sin(x) & x = 1 \\ 5 & otherwise \end{cases}$$

Q2. **Write a program to find the sum of the following series.**

$$sum = \frac{2!}{x^2} - \frac{4!}{x^4} + \frac{6!}{x^6} - \frac{8!}{x^8} + \cdots \cdots \cdots \cdots \cdots \mp \frac{n!}{x^n}$$

Q3. Write a program to find the sum of the following series using function :

1 + 1/2! + 1/3! + … 1/n!

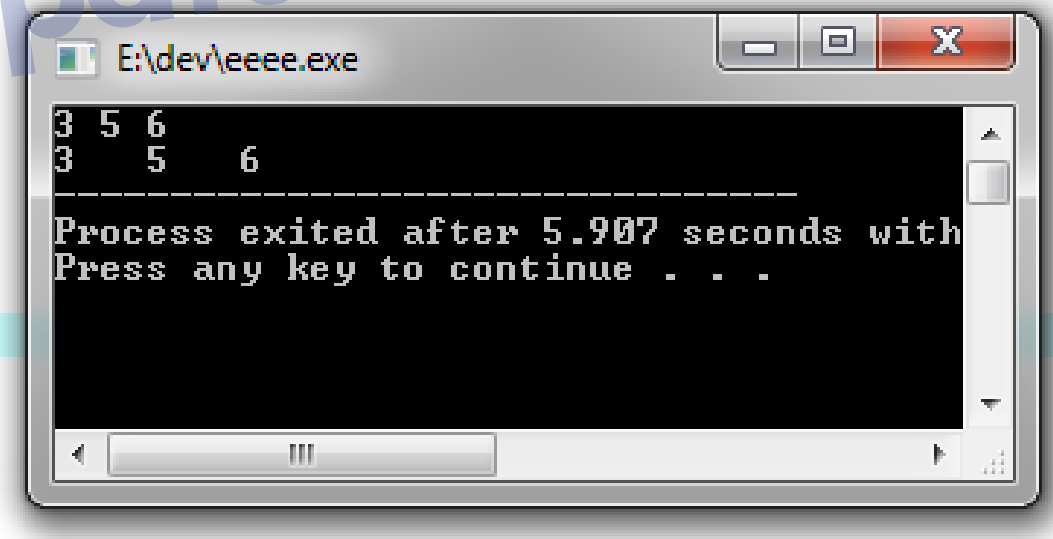
Q4. Write c++ program using structure for sorting information about 5 students according to avg. (Info= name , avg, age ,gender.)ascending .

```
// write a program  for findig sum
//        2     4              n
// sum=2!/x -4!/x  + ..... n!/x
#include <iostream>
#include <math.h>
using namespace std;
int fact(int k) { int i,f=1;for (i=k;i>=1;i--) f=f*i;return f; }
int po_k(int s,int w){ return pow(s,w); }
 int main()
 { float sum;int i,n,x=3;
  cout<<"Enter value of n: ";
  cin>>n;
  for (i=2;i<=n;i+2)
  sum=sum+fact(i)/po_k(x,i);
  cout<<sum;



  return 0;
  }
```

Array of Objects in C++

class_name array_name [size] ;

```
1   // array of objects
2   #include <iostream>
3   using namespace std;
4   class A
5   { int x;
6    public :int get_x() { cin>>x; return x;}                    // a[0]   x  get_x()
7           int pr_x() {return x;}                               //a[1]    x  get_x()
8                                                                //a[2]    x  get_x()
9   } ;
10  int main(){ int i;A a[3];
11
12      for(i=0;i<3;i++)
13      a[i].get_x();
14    for(i=0;i<3;i++)
15   cout<<a[i].pr_x()<<"    ";
16
17      return 0;
18  }
```



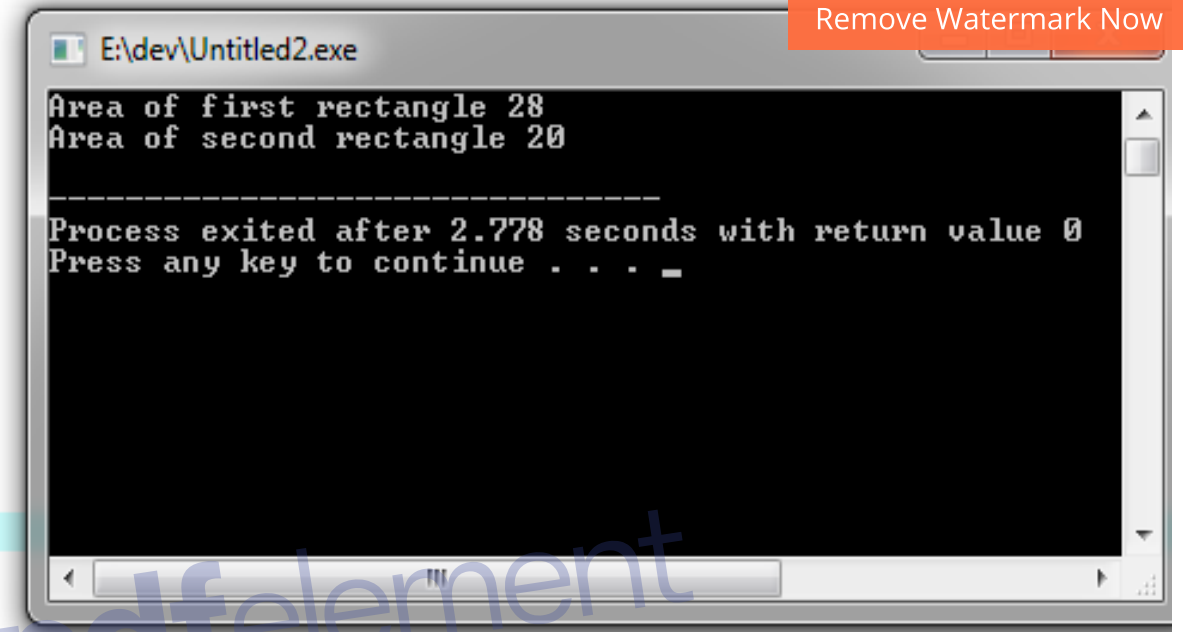
```cpp
// write O.O.P  program using structure for sorting information for 5 students descending acording to average.
#include <iostream>
using namespace std;
class stud
{ public :int no;
  string name;

   float avg;
  void get_info(){
                  cin>>no>>name>>avg; }
} a[5];
int main(){ int i,j;stud t;
cout <<"enter information of student:";
for (i=0;i<=4;i++)
a[i].get_info();
for (i=0;i<=3;i++)
for (j=i+1;j<=4;j++)
   if (a[i].avg<a[j].avg) { t=a[i];
                            a[i]=a[j];
                            a[j]=t; }
  for (i=0;i<=4;i++)
 cout<<a[i].no<<"   "<<a[i].name<<"     "<<a[i].avg<<endl;
   return 0;

   }
```

```
E:\dev\struccc.exe
enter information of 5 students:
7 rrr 90
enter information of 5 students:5 ttt 88
enter information of 5 students:3 sss 97
enter information of 5 students:2 yyy 22
enter information of 5 students:6 iii 77
3   sss     97
7   rrr     90
5   ttt     88
6   iii     77
2   yyy     22

--------------------------------
Process exited after 67.16 seconds with return value 0
Press any key to continue . . .
```


Compile Log | Debug | Find Results | Close

```
- Errors: 0
- Warnings: 0
```

```cpp
#include <iostream>
using namespace std;
class Rectangle
{   public:
        int length;
        int width;
        void get_lw( int l, int w )
        {   length = l;
            width = w;}
        int printArea()
        {
            return length * width;
        }
};
int main()
{
    Rectangle rt1,rt2;

    rt1.get_lw( 7, 4 );
    rt2.get_lw( 4, 5 );
    cout << "Area of first rectangle " << rt1.printArea() << endl;
    cout << "Area of second rectangle " << rt2.printArea() << endl;
    return 0;
}
```



Compile Log | Debug | Find Results | Close

```
- Errors: 0
- Warnings: 0
- Output Filename: E:\dev\Untitled2.exe
- Output Size: 1.83307838439941 MiB
```

Q2 : If you have an int a[5][5] .Write an O.O.P for finding :

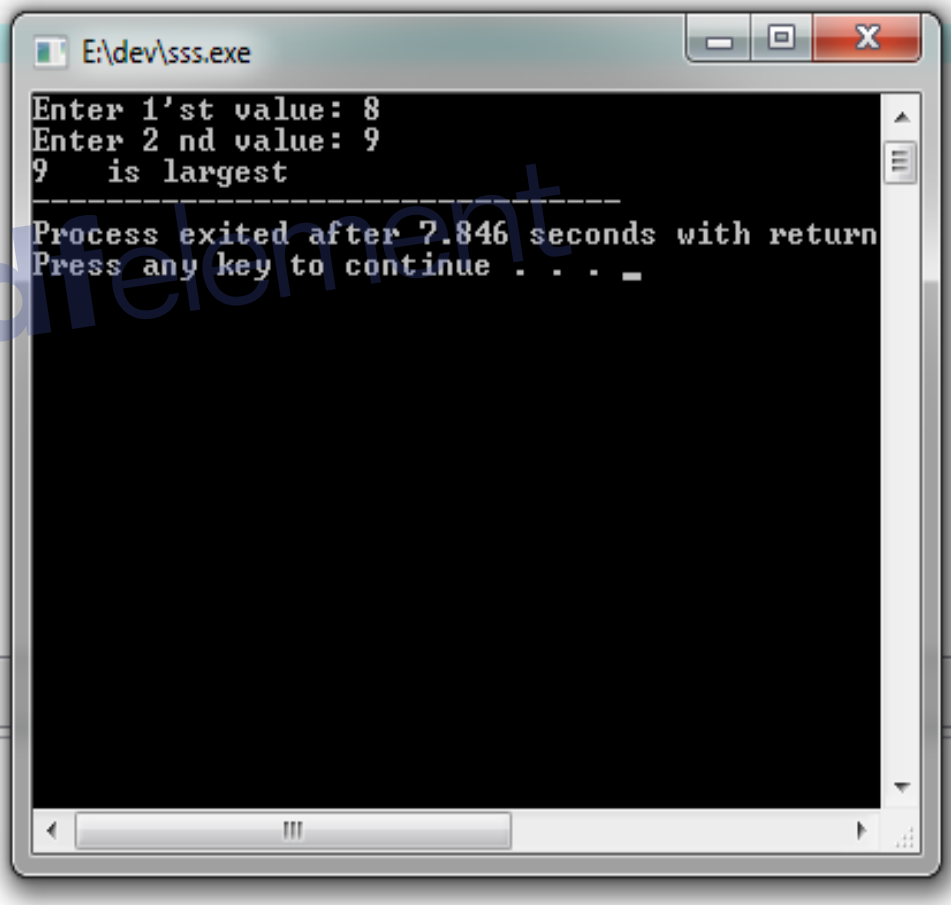
1. The sum of main diameter of a.
2. The sum of second diameter of a.
3. The sum of each column of a.
4. The sum of each row of a.
5. Print the elements of Up triangle of a.
6. Print the elements of Down triangle of a.


```
#include <iostream>
using namespace std;
class array_op
{int x[5][5];
public :
    void get_x(){for (int i=0;i<=4;i++)
                    for (int j=0;j<=4;j++)cin>>x[i][j];}
    void main_di(){
   int sum=0;
  for (int i=0;i<=4;i++) sum=sum+x[i][i];
  cout<<endl<<"The sum of main diameter= "<<sum;
  }
  void second_di()
{ int sum=0;for (int i=0;i<=4;i++)
                    for (int j=0;j<=4;j++)
                    if (i+j==4)sum=sum+x[i][j];
                    cout<<endl<<"The sum of second diameter= "<<sum;
}

void sum_row(){int sum=0;for (int i=0;i<=4;i++)
                    {for (int j=0;j<=4;j++)
                    sum=sum+x[i][j];
                    cout<<endl<<"sum of row:"<<i<<"    "<<sum;
                    sum=0;
                    }
                }
```

```
23          void sum_column(){int sum=0;for (int i=0;i<=4;i++)
24                  {for (int j=0;j<=4;j++)
25                  sum=sum+x[j][i];
26        cout<<endl<<"sum of column:"<<i<<"    "<<sum;
27                  sum=0;
28                  } }
29                void top_tri(){cout<<endl<<"up triangle"<<endl;
30                    for (int i=0;i<=4;i++) {cout<<"\n";
31                     for (int j=i;j<=4;j++)cout <<x[i][j];}
32      }
33  void down_tri(){cout<<endl<<"down triangle"<<endl;
34                    for (int i=0;i<=4;i++)
35                   {cout<<"\n";
36                  for (int j=0;j<=i;j++)cout <<x[i][j];}}
37  };
38  int main()
39  {array_op obj1;
40  obj1.get_x();
41  obj1.main_di();
42  obj1.second_di();
43  obj1.sum_column();
44  obj1.sum_row();
45  obj1.top_tri();
46  obj1.down_tri();
47  return 0;
48  }
49
```

```
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
5 5 5 5 5

The sum of main diameter= 15
The sum of second diameter= 15
sum of column:0    15
sum of column:1    15
sum of column:2    15
sum of column:3    15
sum of column:4    15
sum of row:0    5
sum of row:1    10
sum of row:2    15
sum of row:3    20
sum of row:4    25
up triangle

11111
2222
333
44
5
down triangle

1
22
333
4444
55555
--------------------------------
Process exited after 19.48 seconds with return value 0
Press any key to continue . . .
```

```cpp
//If you have two separated class each class one float value . Write an O.O.P for finding the largest one.
#include <iostream>
using namespace std;
class A
{ float x;
  public: int get_x() { cout<<"Enter 1'st value: ";cin>>x;return x; }
 } ;
class B
{ float y;
  public : int get_y() { cout<<"Enter 2 nd value: ";cin>>y;return y; }
 } ;
 int main()
 { A a;B b;int c1,c2;
 c1=a.get_x();
 c2=b.get_y();
  if (c1>c2 ) cout <<c1<<"  is largest";
  else cout<<c2<<"   is largest";
  return 0;
  }
```

Compile Log | Debug | Find Results | Close

```
- Errors: 0
- Warnings: 0
- Output Filename: E:\dev\sss.exe
- Output Size: 1.83280849456787 MiB
```

Home work :

Q1. If you have 3 separated classes each class has one value write an O.O.P program for combining these values to form number of 3 digits.

Q2. Write O.O.p for split number of 3 digits to it is digits
e.G 543 =3 40 500

Local and Global variables

Local variables can be used only by statements that are inside that function or block of code. Local variables are not known to functions on their own.

Example

```
#include <iostream>
using namespace std;
int main () {
   // Local variable declaration:
   int a, b;
   int c;

   // actual initialization
   a = 10;
   b = 20;
   c = a + b;

   cout << c;
   return 0;
}
```


Output

This will give the output –

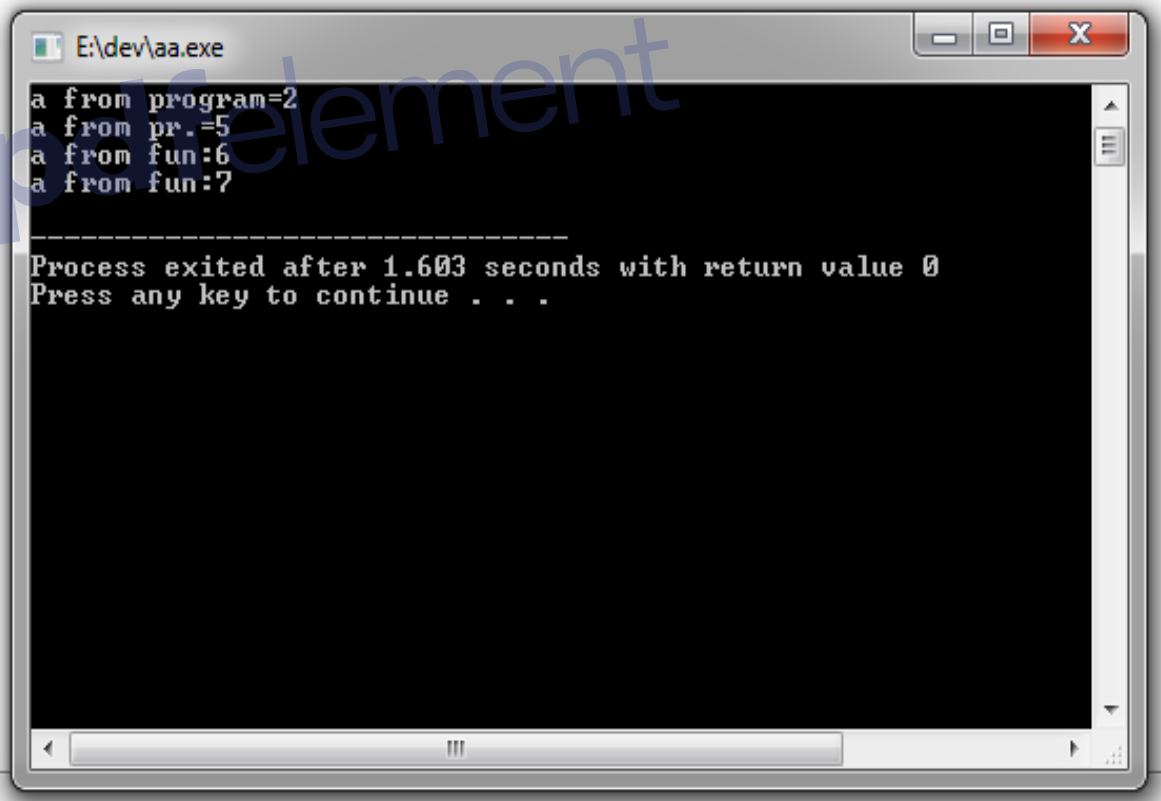
```
30
```

```cpp
#include <iostream>
using namespace std;

void local(){
        int a=0;
         a=a+1;
         cout<<"a from fun:"<<a<<endl;}
int main()
{float a=4; a=a+1;
cout<<"a from program="<<a<<endl;
a=a+3;cout<<"a from pr.="<<a<<endl;
local();
local();
   return 0;

    }
```

```
E:\dev\example1.exe
a from program=5
a from pr.=8
a from fun:1
a from fun:1

--------------------------------
Process exited after 1.383 seconds with return value 0
Press any key to continue . . .
```

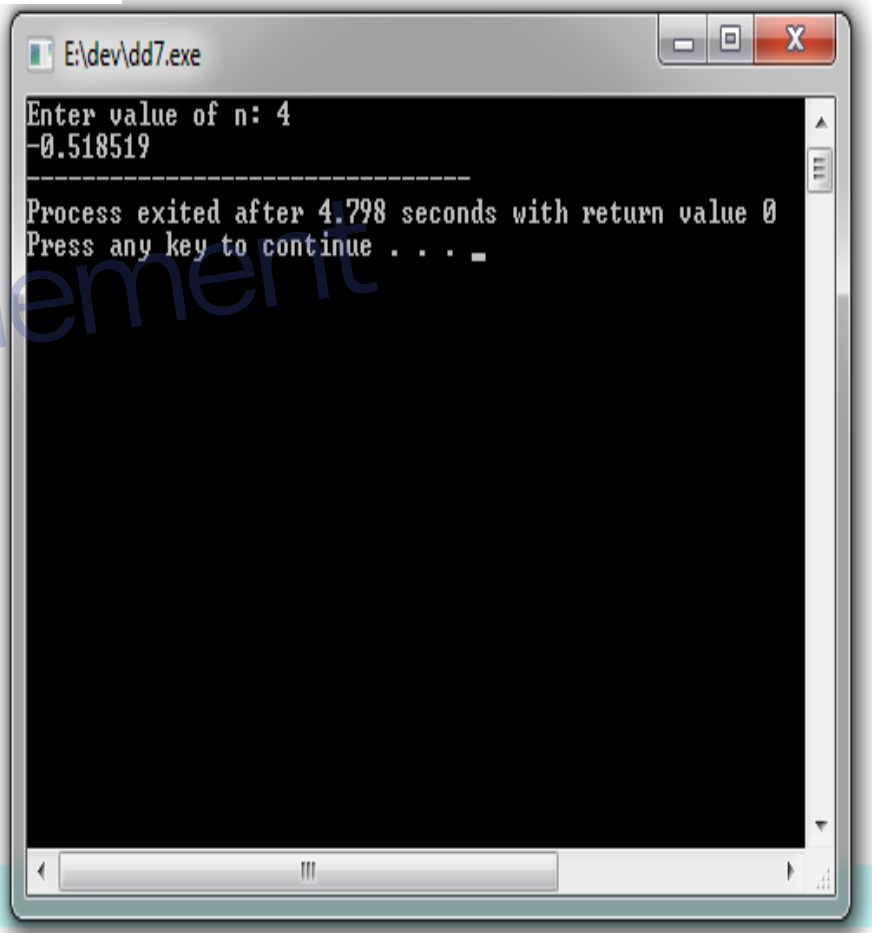
Global variables are defined outside of all the functions, usually on top of the program. The global variables will hold their value throughout the lifetime of your program. A global variable can be accessed by any function.


```
1  // write c++  program using structure for sorting information for 5 students descending acording to average.
2  #include <iostream>
3  using namespace std;
4  int a=1;
5  void glob(){ a=a+1;cout<<"a from fun:"<<a<<endl;}
6  int main()
7  { a=a+1;
8  cout<<"a from program="<<a<<endl;
9  a=a+3;cout<<"a from pr.="<<a<<endl;
10 glob();
11 glob();
12    return 0;
13
14    }
```



Compile Log | Debug | Find Results | Close

```
1  // write a program  for findig sum
2  //          2     4                  2n
3  // sum=2!/x -4!/x  + ..... 2n!/x
4  #include <iostream>
5  #include <math.h>
6  using namespace std;
7  float fact(int k)
8   { int i,f=1;for (i=1;i<=k;i++) f=f*i;return f; }
9  int po_k(int s,int w){
10  return pow(s,w); }
11  int main()
12  { float sum=0;int  i,n,x=3;int f=1;
13   cout<<"Enter value of n: ";  cin>>n;
14   for (i=2;i<=n;i++)
15   {if (i%2==0) sum=sum+(fact(i)/po_k(x,i));
16    sum=sum*(f*-1);}
17   cout<<sum;
18    return 0; }
```



dd7.cpp

```
1  // write O.O. program  for findig sum
2  //          2     4                 2n
3  // sum=2!/x -4!/x  + ..... 2n!/x
4  #include <iostream>
5  #include <math.h>
6  using namespace std;
7  class sum
8  { public : int n,x;
9
10  void get_nx(){ cout<<"Enter 2 value of n,x: "; cin>>n>>x;}
11 float fact(int k) { int i,f=1;for (i=1;i<=k;i++) f=f*i;return f; }
12 int po_k(int s,int w){ return pow(s,w); }
13 };
14  int main()
15  { sum A;float sum=0;int  i;int f=1;
16  A.get_nx();
17   for (i=2;i<=A.n;i++)
18   {if (i%2==0) sum=sum+(A.fact(i)/A.po_k(A.x,i));
19    sum=sum*(f*-1);}
20   cout<<sum;
21    return 0; }
```

E:\dev\dd7.exe

```
Enter 2 value of n,x: 3 1
2
--------------------------------

Process exited after 8.459 seconds with return value 0
Press any key to continue . . .
```

Friend Function

One of the important concepts of OOP is data hiding, i.e., a nonmember function cannot access an object's private or protected data. But, sometimes this restriction may force programmer to write long and complex codes. So, there is mechanism built in C++ programming to access private or protected data from non-member functions. This is done using a friend function or/and a friend class.

A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

A friend function can be friendly to 2 or more classes. The friend function does not belong to any class, so it can be used to access private data of two or more classes as in the following example.

1. Friend functions are not members of any class but they can access private data of the class to which they are a friend.
2. Because they are not members of any class, you should not call them using the dot operator.

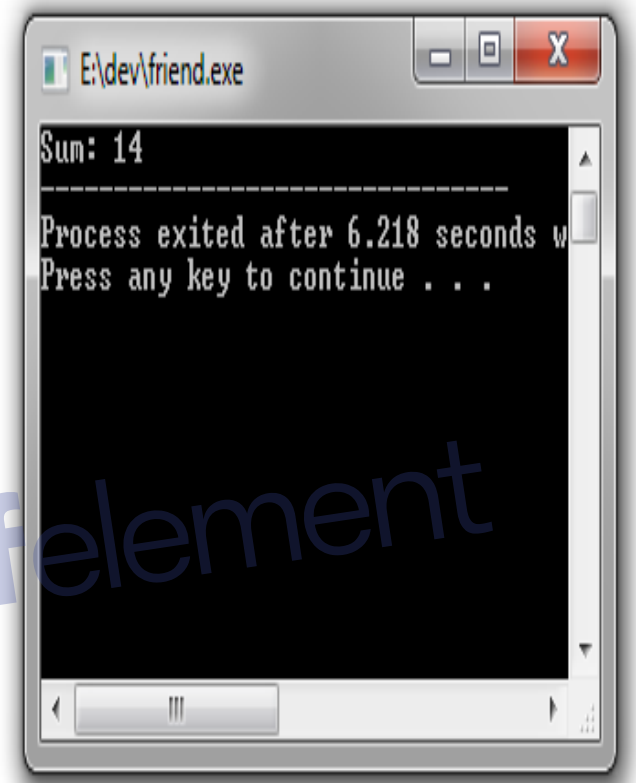
Syntax:

```
Class ABC
{
…
…..
public:
friend void xyz(object of class); /// declaration of friend function
};
. //definition of friend function
.
```

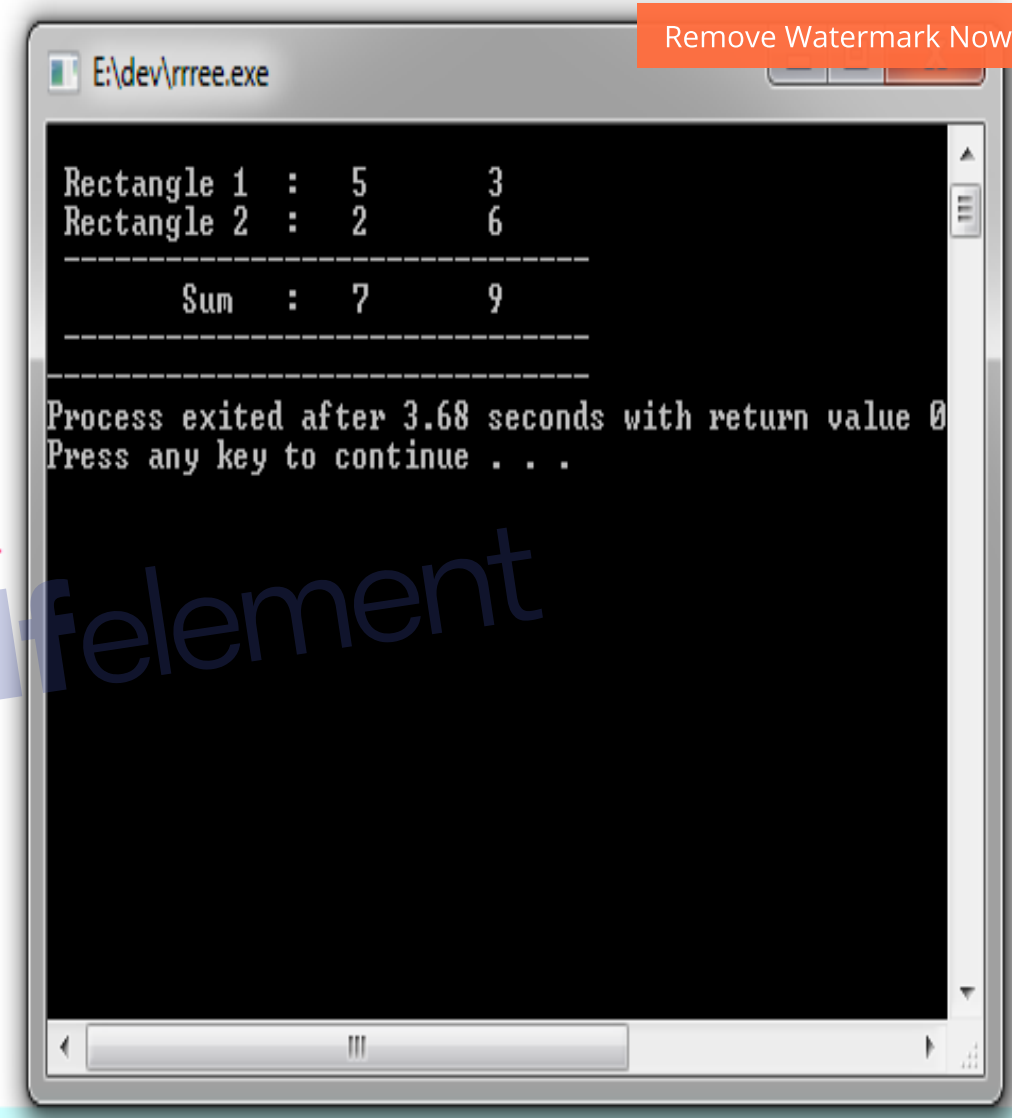

Friend function characteristics:

- It is not scope of class
- It cannot be called using object of that class.
- It can be invoked like a normal function.
- It should use a dot operator for accessing members.
- It can be public or private.
- It has objects as arguments.

```cpp
// Example of FRIEND Function for adding two values ,each value from one class #include <iostream>
#include <iostream>
using namespace std; // forward declaration class B;
class B;
class A { private: int numA;
public:
void get_numA(int b) {numA=b; } // friend function declaration
friend int add(A, B);
 };
class B { private:
            int numB;
public:
void get_numB(int a) {numB=a; } // friend function declaration
friend int add(A , B);
};

int add(A objectA,B objectB)
// Function add() is the friend function of classes A and B // that accesses the member variables numA and numB int add(A objectA, B objectB)
{
return (objectA.numA + objectB.numB); }
int main() {
A objectA;  objectA.get_numA(6);
B objectB; objectB.get_numB(8);
cout<<"Sum: "<< add(objectA, objectB);
return 0;
}
```



```cpp
// Finding area in two classes and Adding these  two area :
 #include <iostream>
 using namespace std;
class RectangleTwo;
class RectangleOne
{  int L,B;
   public:
       RectangleOne(int l,int b) {L = l; B = b; }
friend void Sum(RectangleOne, RectangleTwo);
};
class RectangleTwo {  int L,B;
   public:
          RectangleTwo(int l,int b) {L = l; B = b; }
friend void Sum(RectangleOne, RectangleTwo);
 };
void Sum(RectangleOne R1,RectangleTwo R2) {
cout<<"\n Rectangle 1  :   "<<R1.L<<"\t  "<<R1.B;
cout<<"\n Rectangle 2  :   "<<R2.L<<"\t  "<<R2.B;
cout<<"\n ------------------------------";
cout<<"\n\tSum   :   "<<R1.L+R2.L<<"\t  "<<R1.B+R2.B;
cout<<"\n ------------------------------"; }
int main()
{ RectangleOne Rec1(5,3);
RectangleTwo Rec2(2,6);
Sum(Rec1,Rec2);
return 0; }
```

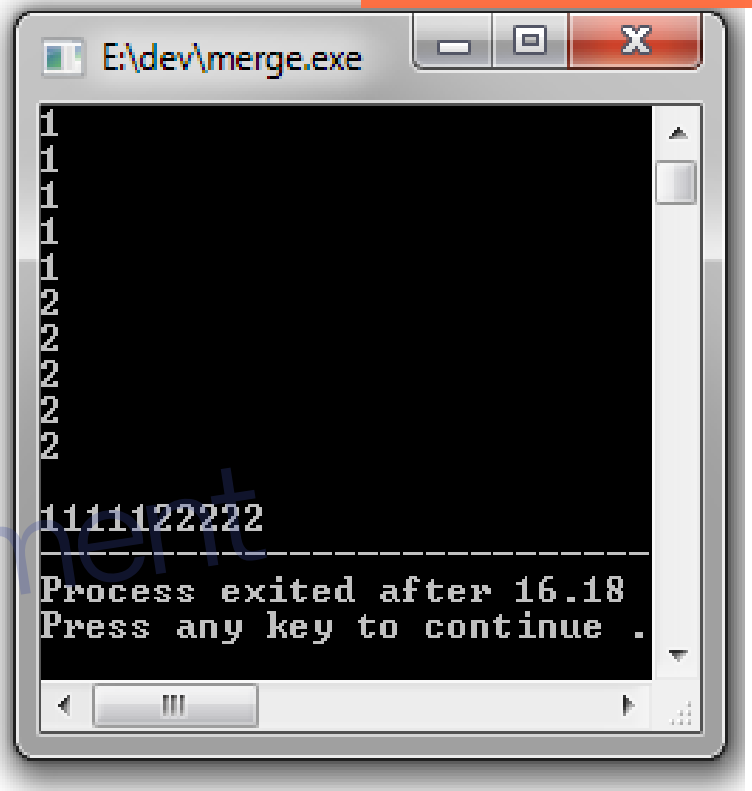


Q1. If you have two separated classes , each one has one string ,write an O.O.P for merging this string into one.

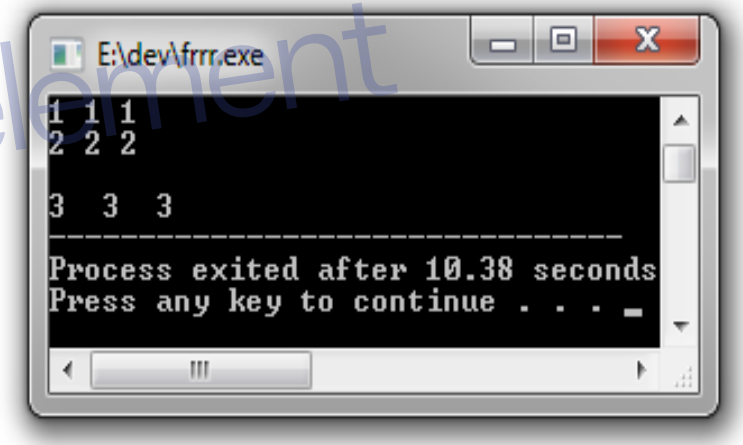
Q2. If you have 3 separated classes (A,B,C) each one has an array (a[3],b[3],c[3]) of type character. Write an O.O.P for combining these arrays into one d[3][3].

Q3/ If you have 2-classes A,B. Class A contains an array a[5]. Class B contains an array b[5]. Write an O.O.P using friend function for merge these arrays into one array c[10].


```
 #include <iostream.h>
class B;
class A
{ int a[5];
 public:
  void get_a(void){ for (int i=0;i<=4;i++) cin>>a[i];}
  friend void merge(A a1,B b1);
};
class B
{int b[5];
 public:
    void get_b(void){ for (int i=0;i<=4;i++) cin>>b[i];}
    friend void merge(A a1,B b1);
  };
  void merge(A a1,B b1)
  {int c[10],i;
   for (i=0;i<=4;i++)
    c[i]=a1.a[i];
    for (i=5;i<=9;i++)
    c[i]=b1.b[i-5];
    cout<<"\n";
    for ( i=0;i<=9;i++)
    cout << c[i];
    }
main()
{B b2;A a2;
a2.get_a();
b2.get_b();
merge(a2,b2);}
```



```cpp
// If you have two separated classes ,each class has one dimention array a[3],b[3] .Write oop for
//adding these arrays into c[3]  using friend function
#include <iostream>
using namespace std;
    class A;
    class B {       int L[3];
        public:
    void get_b(int l[3]){int i;  for (i=0;i<=2;i++)L[i]=l[i];}
            friend void add(A a, B b);};
    class A {   int s[3];
        public:void get_a(int m[3]){int i;     for (i=0;i<=2;i++)s[i]=m[i];}
            friend void add(A a, B b);  };
void  add(A a, B b) { int i,j;int c[3];
       for (i=0;i<=2;i++)      c[i]=a.s[i]+b.L[i];
           cout<<"\n";
          for (i=0;i<=2;i++)
           cout<<c[i]<<"  ";
}
    int main()
    {A aa;B bb;
       int a1[3],a2[3],i;
       for (i=0;i<=2;i++) cin>>a1[i];
        for (i=0;i<=2;i++)cin >>a2[i];
      aa.get_a(a1); bb.get_b(a2);
     add(aa,bb);
return 0;
    }
```



Constructor and Destructor

A constructor is a special member function of a class that is executed whenev objects of that class.

A constructor will have exact same name as the class and it does not have any return type at all, not even void. Constructors can be very useful for setting initial values for certain member variables. Following example explains the concept of constructor:

constructor.cpp

```
1  #include  <iostream>
2  using namespace std;
3  int a=1;
4  class Line
5  { public:
6  Line(); // This is the constructor
7  };
8  // Member functions definitions including constructor
9  Line::Line(void)
10 {cout << "Object is being created:   " <<a<<endl;
11  a++;
12 }
13 // Main function for the program
14 int main( )
15 {
16 Line line; //create object1
17 Line aa;   //create object 2
18 Line ff;    //create object3
19 return 0;
20 }
```

E:\dev\constructor.exe

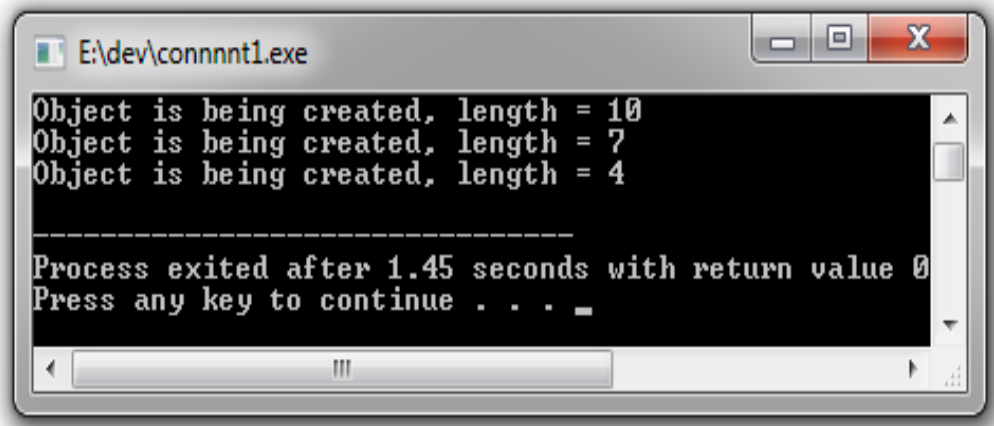
```
Object is being created:   1
Object is being created:   2
Object is being created:   3

--------------------------------
Process exited after 1.716 seconds with return value 0
Press any key to continue . . .
```

Parameterized Constructor

A default constructor does not have any parameter, but if you need a constructor can have parameters. This helps you to assign initial value to an object at the time of its creation as shown in the following example:

Example 1

```
2  using namespace std;
3  class Line
4  {public:
5  Line(double len); // This is the constructor
6  double length;
7  };
8  // Member functions definitions including constructor
9  Line::Line( double len)
10 {
11 cout << "Object is being created, length = " << len << endl;
12 length = len; }
13 // Main function for the program
14 int main( )
15 {
16 Line line1(10.0);//create object 1
17 Line line2(7.0);//create object 2
18 Line line3(4);//create object 3
19 return 0;
20 }
```



Example 2

constructor.cpp | connnnt1.cpp | [*] exa2.cpp

```
1   #include <iostream>
2   using namespace std;
3   class Line
4   {public:
5   void setLength( double len );
6   double getLength( void );
7   Line(double len); // This is the constructor
8   private:double length;
9   };
10  // Member functions definitions including constructor
11  Line::Line( double len)
12  {cout << "Object is being created, length = " << len << endl;
13  length = len;}
14  void Line::setLength( double len ){length = len;}
15  double Line::getLength( void )
16  {return length;}
17  // Main function for the program
18  int main( )
19  {
20  Line line(10.0);
21  // get initially set length.
22  cout << "Length of line : " << line.getLength() <<endl;
23  // set line length again
24  line.setLength(6.0);
25  cout << "Length of line : " << line.getLength() <<endl;
26  return 0;
27  }
28
```

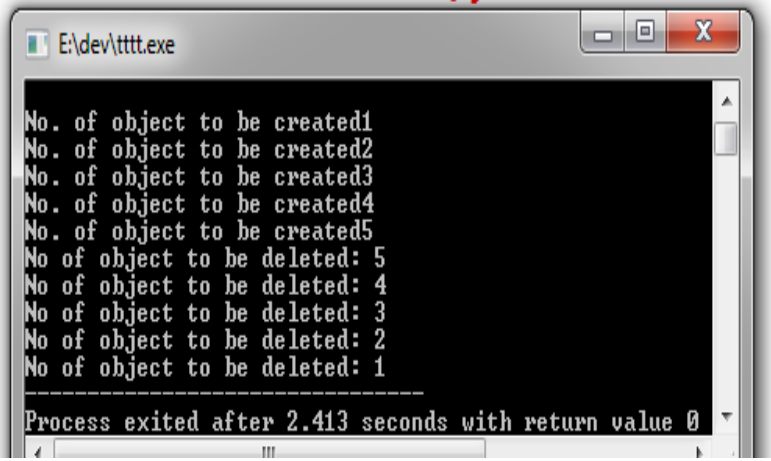
E:\dev\exa2.exe

```
Object is being created, length = 10
Length of line : 10
Length of line : 6

--------------------------------
Process exited after 1.747 seconds with return value 0
Press any key to continue . . .
```

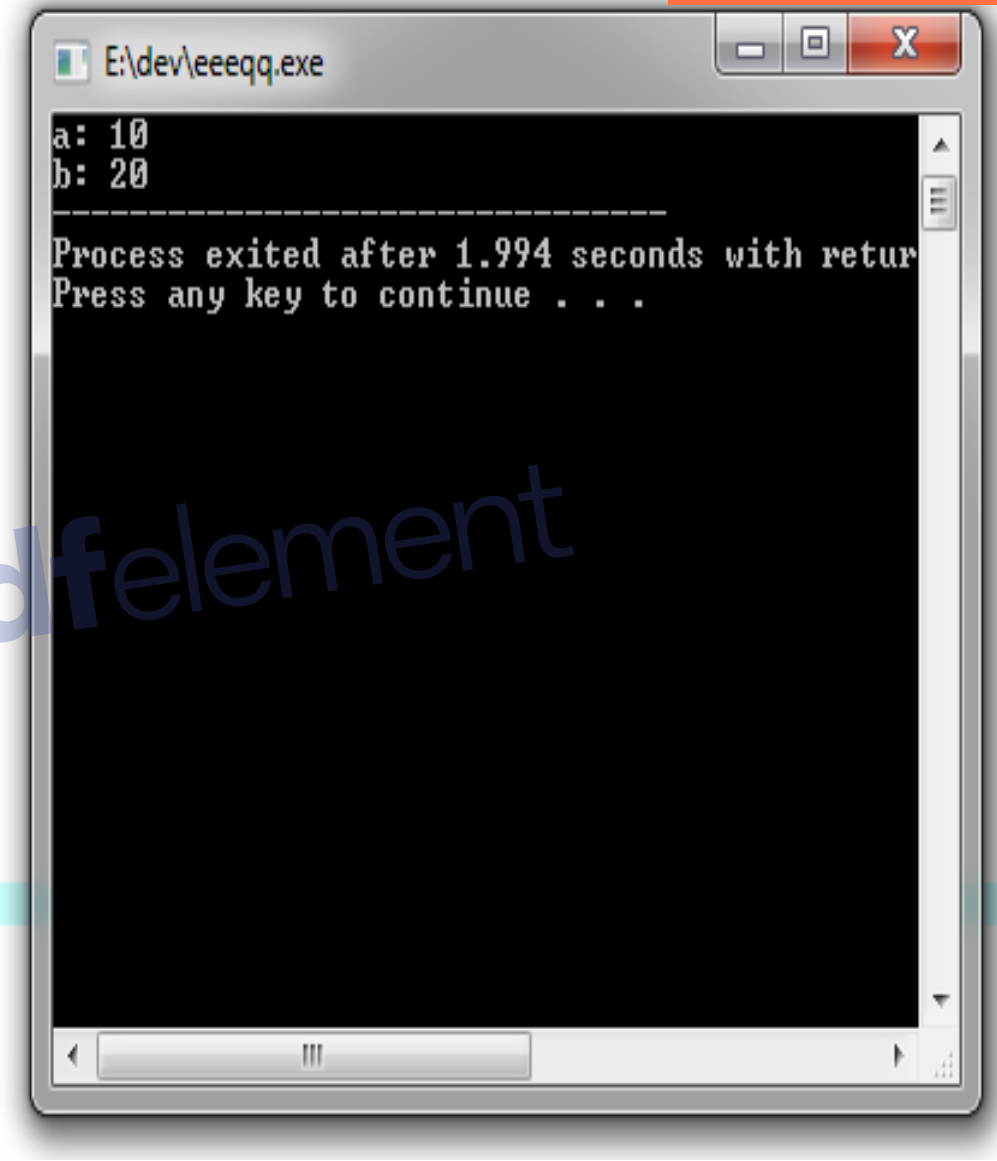
The Class Destructor

A **destructor** is a special member function of a class that is executed whenever an object of its class goes out of scope or whenever the delete expression is applied to a pointer to the object of that class. A destructor will have exact same name as the class prefixed with a tilde (~) and it can neither return a value nor can it take any parameters. Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories etc. Following example explains the concept of destructor:


```
1  #include <iostream>
2  using namespace std;
3  int a=1;
4  class Line
5  {
6  public:
7
8  Line(){cout<<endl<<"No. of object to be created"<<a;a=a+1;} // This is the constructor
9  ~Line(){a=a-1;cout<<endl<<"No of object to be deleted: "<<a;} // This is the destructor
10 };
11 int main( )
12 {
13 Line line1;Line line2; Line line3;
14 Line line4;Line line5;
15 return 0;
16 }
```

```cpp
// Cpp program to illustrate the
// concept of Constructors
#include <iostream>
using namespace std;
class construct {
public:
    int a, b;
    // Default Constructor
    construct()
    {
        a = 10;
        b = 20;
    }
};
int main()
{
    // Default constructor called automatically
    // when the object is created
    construct c;
    cout << "a: " << c.a << endl
         << "b: " << c.b;
    return 1;
}
```



Homework :

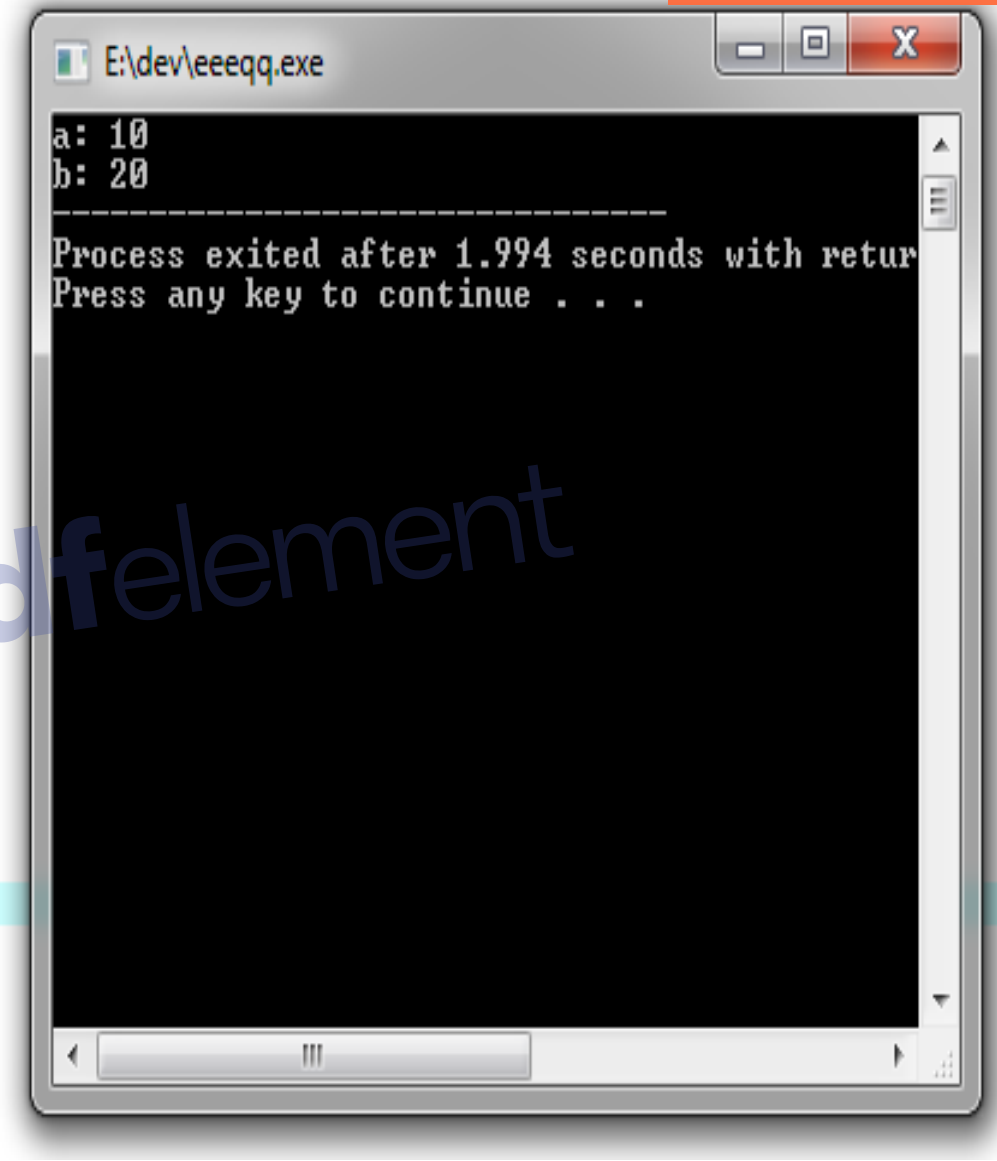
Q1 . Write an O.O.P for creating 10 objects and remove them from memory.

Q2. Write an O.O.P program for printing the following sequence:

1 1 2 3 5 8 13 21 34 55 89 ……..

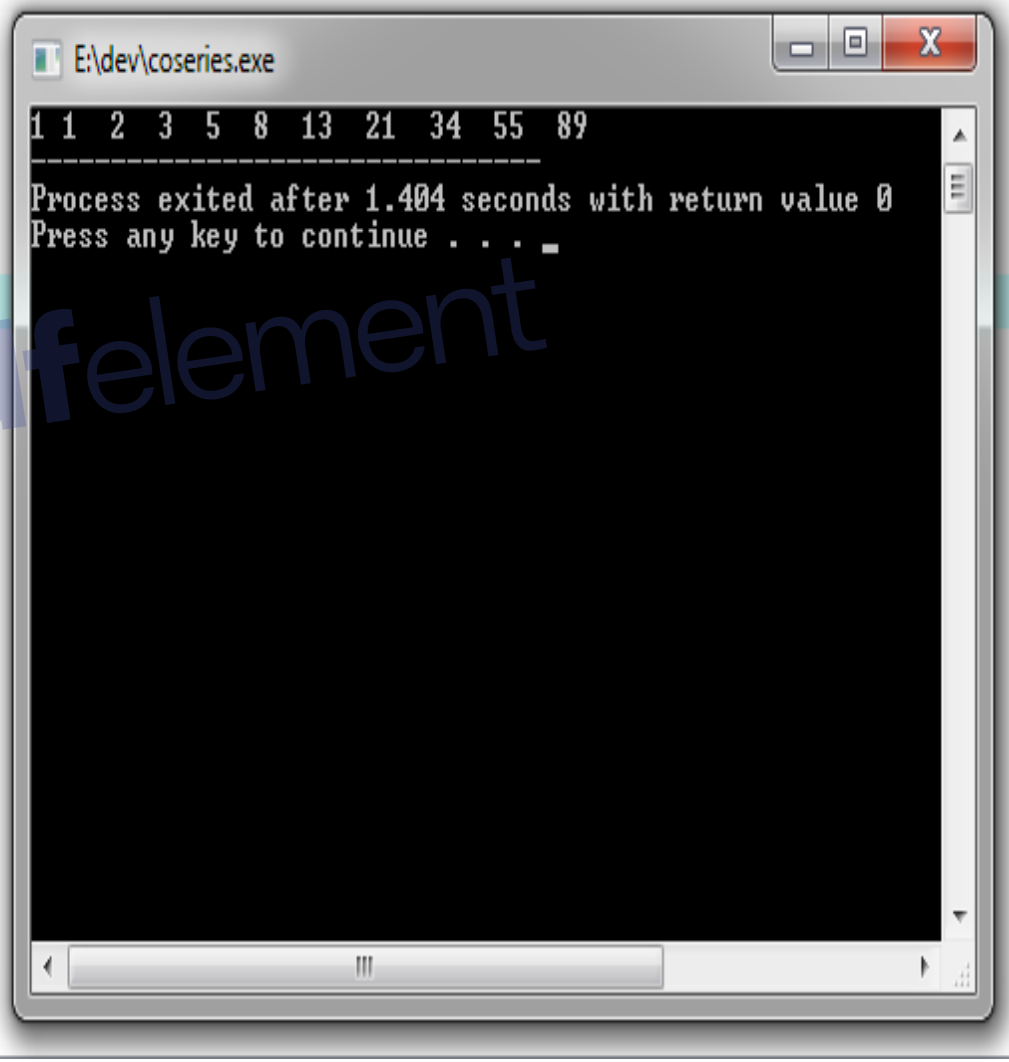
Q3.

```cpp
// Cpp program to illustrate the
// concept of Constructors
#include <iostream>
using namespace std;
class construct {
public:
    int a, b;
    // Default Constructor
    construct()
    {
        a = 10;
        b = 20;
    }
};
int main()
{
    // Default constructor called automatically
    // when the object is created
    construct c;
    cout << "a: " << c.a << endl
         << "b: " << c.b;
    return 1;
}
```



Q2. Write an O.O.P program for printing the following sequence:
1 1 2 3 5 8 13 21 34 55 89 .....

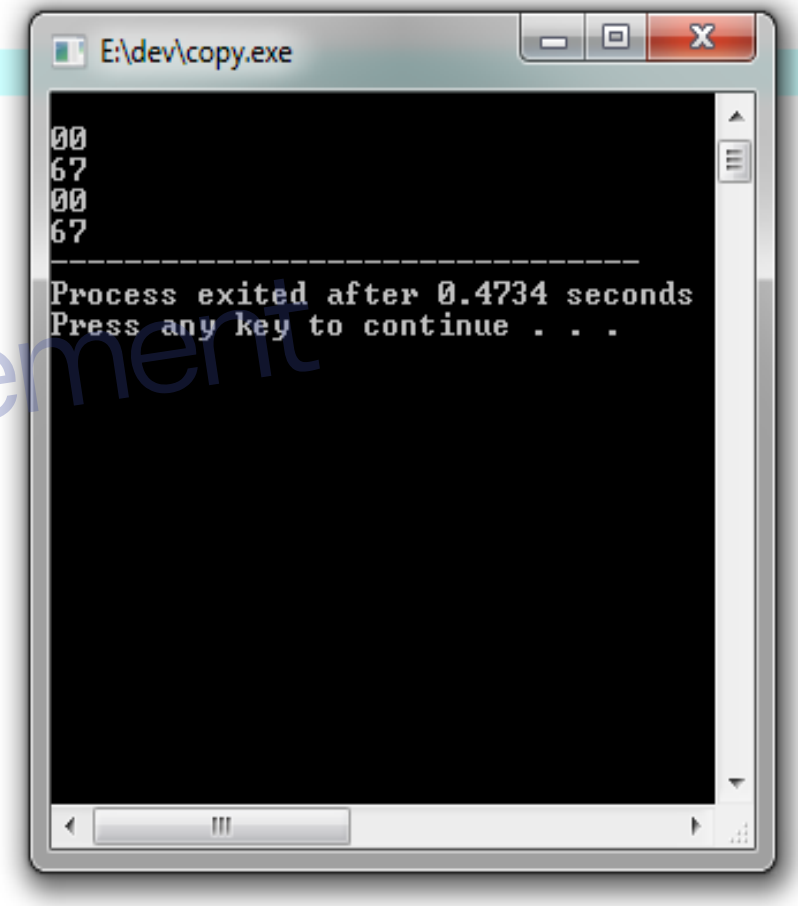

```
1  //Q2. Write an O.O.P program for printing the following sequence:
2  //                1 1 2 3 5 8 13 21 34 55 89 ……..
3  #include <iostream>
4  using namespace std;
5  class series
6  { int a,b;
7  public:
8   series(int x){ a=1;b=1;cout <<a<<" "<<b;int c;
9              do { c=a+b;cout<<"  "<<c;
10                a=b;
11                b=c;
12             } while (c!=x);}
13  };
14  int main()
15  { series a(89);
16
17 }
```



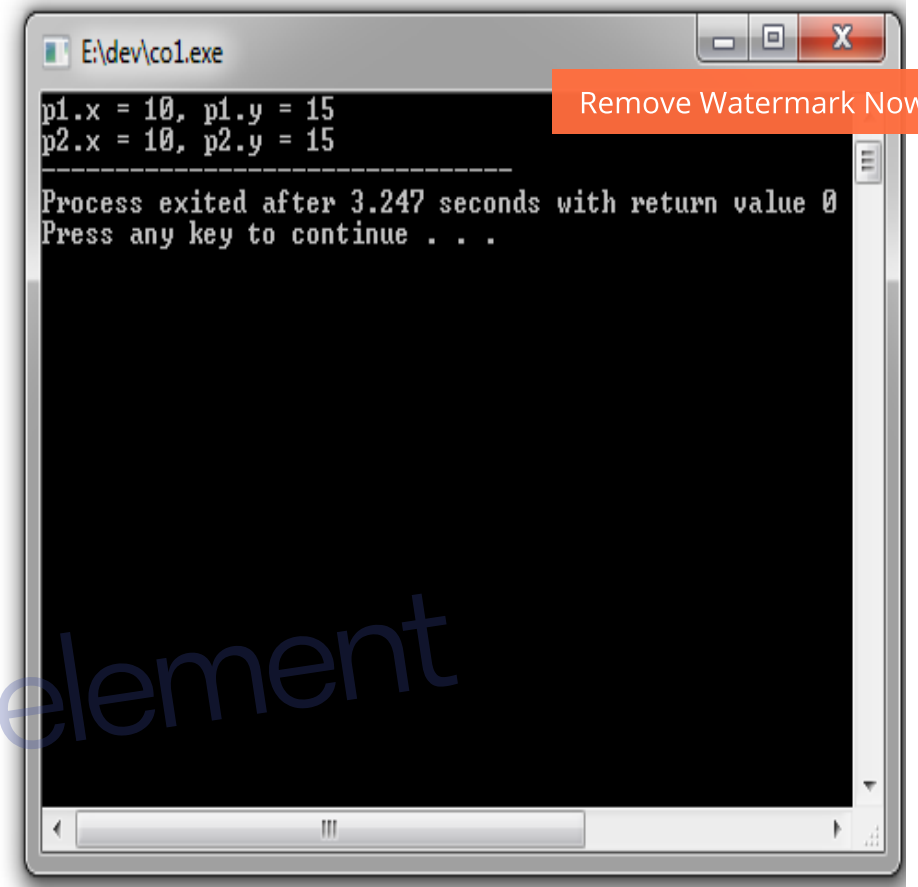
C++ copy constructor

The copy constructor is a constructor which creates an object by initializing it with an object of the same class, which has been created previously. Following an example of copy constructor.

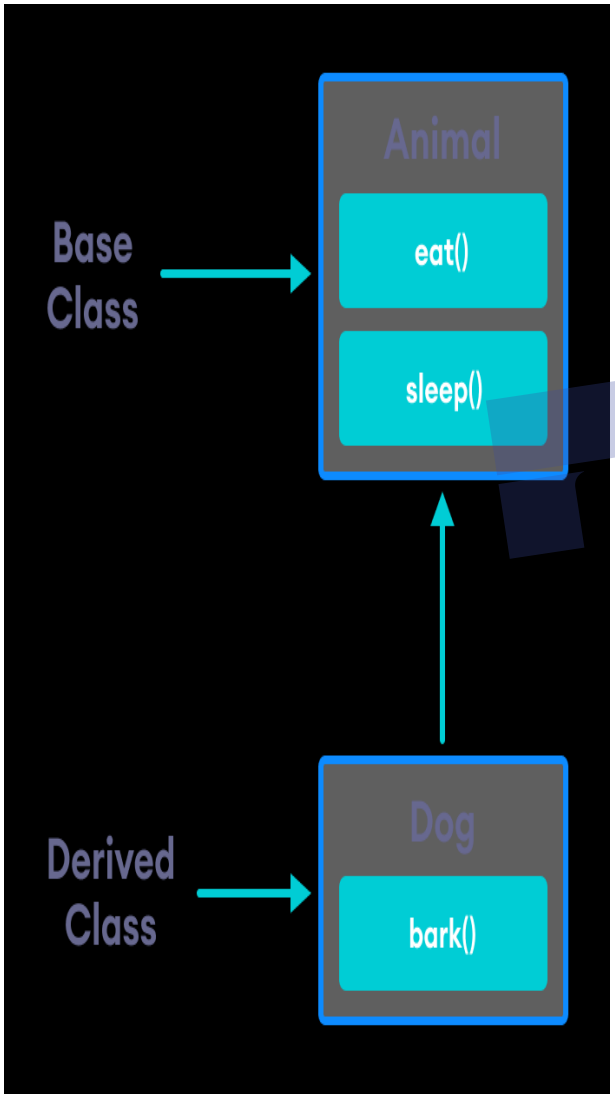
```
1  //C++ copy constructor
2  #include <iostream>
3  using namespace std;
4  class Line
5  {
6  int m,n;
7  public:
8  Line(){m=0;n=0;}; // constructor 1
9  Line(int a,int b) {m=a;n=b;}// constructor2
10 Line(Line & i) // copy constructor - constructor 3
11 { m=i.m;n=i.n;}
12 void print_mn(){ cout<<endl<<m<<n;}
13 };
14
15 // Main function for the program
16 int main( )
17 {
18 Line line;line.print_mn();
19 Line aa(6,7);aa.print_mn();
20 Line ff(line);ff.print_mn();
21 Line qq(aa);qq.print_mn();
22 return 0;
23 }
```



```cpp
#include<iostream>
using namespace std;

class Point
{
private:
    int x, y;
public:
    Point(int x1, int y1) { x = x1; y = y1; }

    // Copy constructor
    Point(const Point &p1) {x = p1.x; y = p1.y; }

    int getX()            {  return x; }
    int getY()            {  return y; }
};

int main()
{
    Point p1(10, 15); // Normal constructor is called here
    Point p2 = p1; // Copy constructor is called here

    // Let us access values assigned by constructors
    cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();
    cout << "\np2.x = " << p2.getX() << ", p2.y = " << p2.getY();

    return 0;
}
```



Inheritance is one of the key features of Object-oriented programming in C++. It allows us to create a new class (derived class) from an existing class (base class). The derived class **inherits** the features from the base class and can have additional features of its own.



Syntax:

```
class subclass_name : access_mode
base_class_name
{
  //body of subclass
};
```

Notice the use of the keyword public while inheriting Dog from Animal.

```
class Animal { }
class Dog : public Animal {...};
```

Notice the use of the keyword private while inheriting Dog from Animal.

```
class Animal { }
      class Dog : private Animal {...};
Or class Dog: Animal{.....}
```

Modes of Inheritance

Public mode: If we derive a sub class from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in derived class.

Protected mode: If we derive a sub class from a Protected base class. Then both public member and protected members of the base class will become protected in derived class.

Private mode: If we derive a sub class from a Private base class. Then both public member and protected members of the base class will become Private in derived class.

Note : The private members in the base class cannot be directly accessed in the derived class, while protected members can be directly accessed. For example, Classes B, C and D all contain the variables x, y and z in below example. It is just question of access.

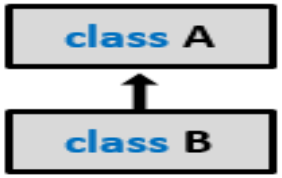
// C++ Implementation to show that a derived class doesn't inherit access to private data members. However, it does inherit a full parent object .

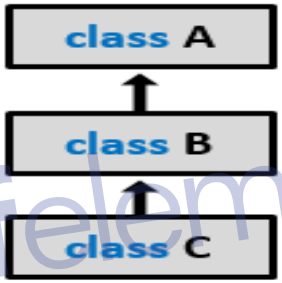
```
class A
{  public:     int x;
   protected : int y;
   private:   int z;
};
class B : public A
{
   // x is public
   // y is protected
   // z is not accessible from B
};
class C : protected A
{
   // x is protected
   // y is protected
   // z is not accessible from C
};
class D : private A    // 'private' is default for classes
 {
    // x is private      y is private
// z is not accessible from D  };
```

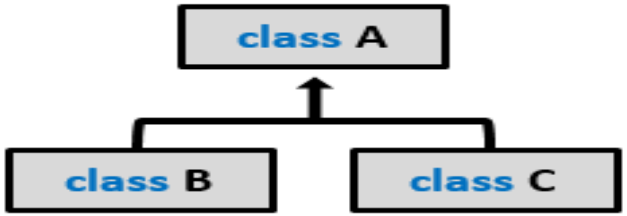
The below table summarizes the above three mo
access specifier of the members of base class in the sub class when derived in public, protected and private modes:

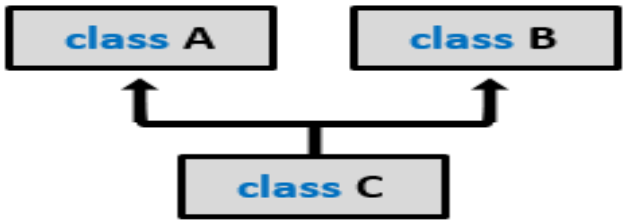
Base class member access specifier	Type of Inheritence		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not accessible (Hidden)	Not accessible (Hidden)	Not accessible (Hidden)

Types of Inheritance

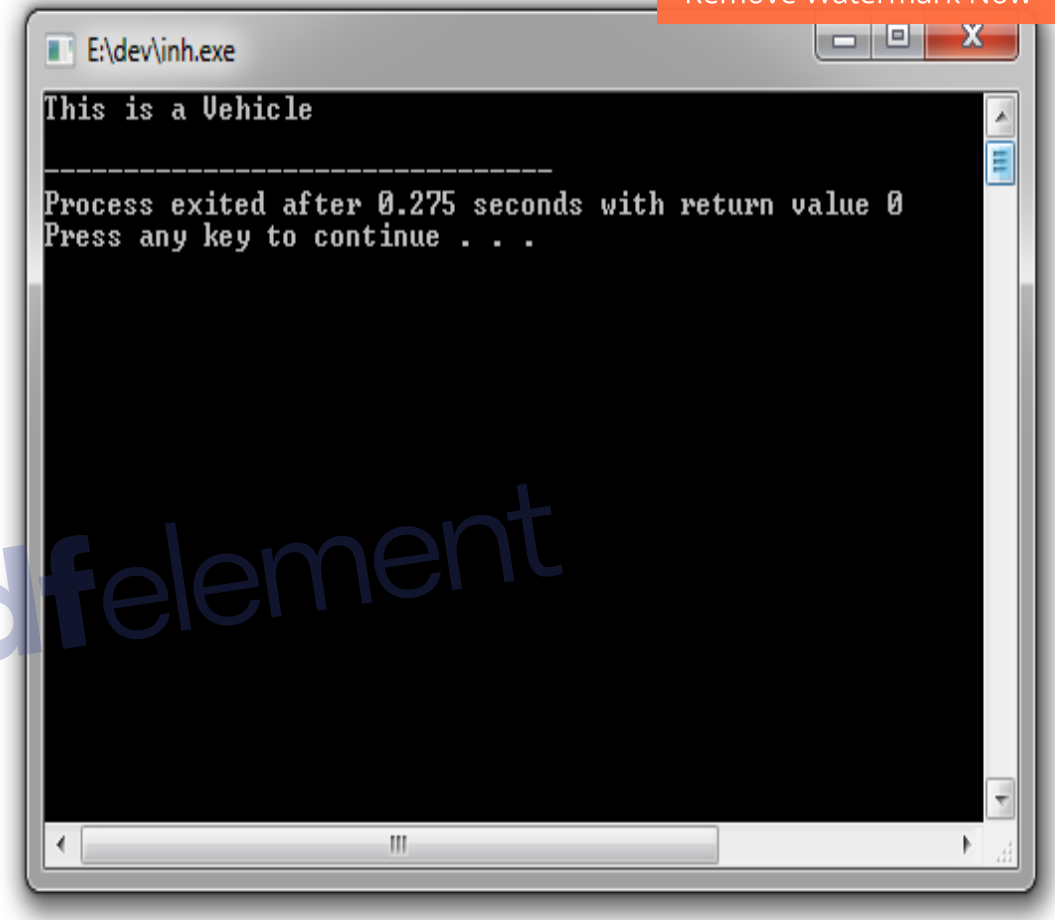
الكود	شكلها	إسمها
`class A { }` `class B : A { }`		Single inheritance وراثة فردية

`class A { }`
`class B : A { }`
`class C : B { }`
		Multi Level inheritance
		وراثة متتالية

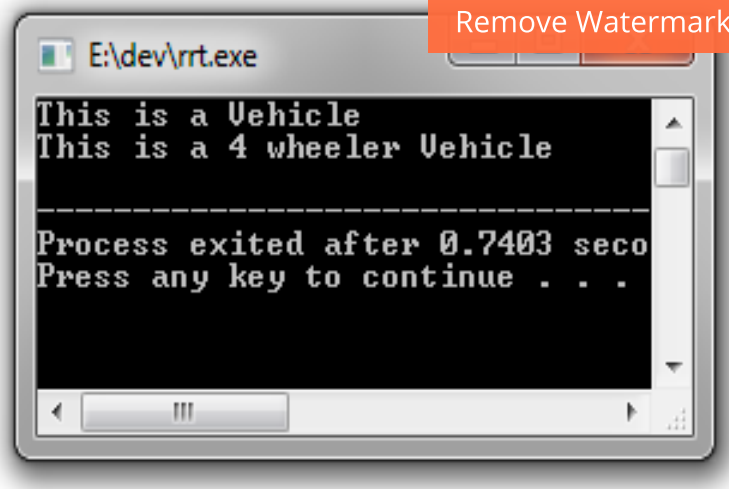
`class A { }`
`class B : A { }`
`class C : A { }`
		Hierarchical inheritance
		وراثة هرمية

`class A { }`
`class B { }`
`class C : A , B { }`
		Multiple inheritance
		وراثة متعددة

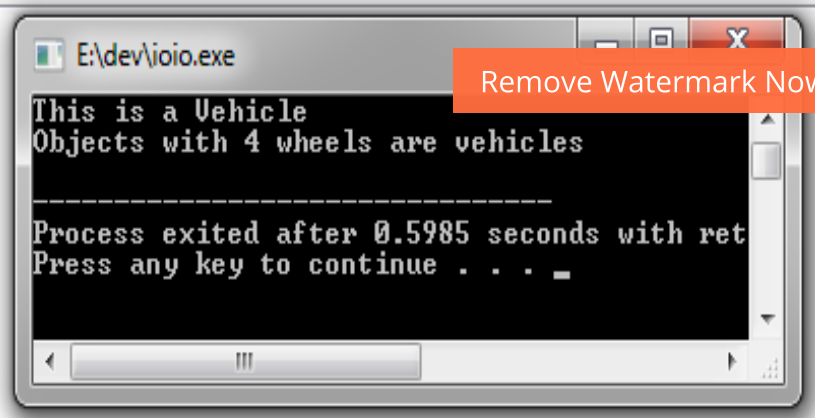
```cpp
// C++ program to explain
// Single inheritance
#include <iostream>
using namespace std;

// base class
class Vehicle {
  public:
    Vehicle()
    {
      cout << "This is a Vehicle" << endl;
    }
};

// sub class derived from two base classes
class Car: public Vehicle{

};

// main function
int main()
{
    // creating object of sub class will
    // invoke the constructor of base classes
    Car obj;
    return 0;
}
```



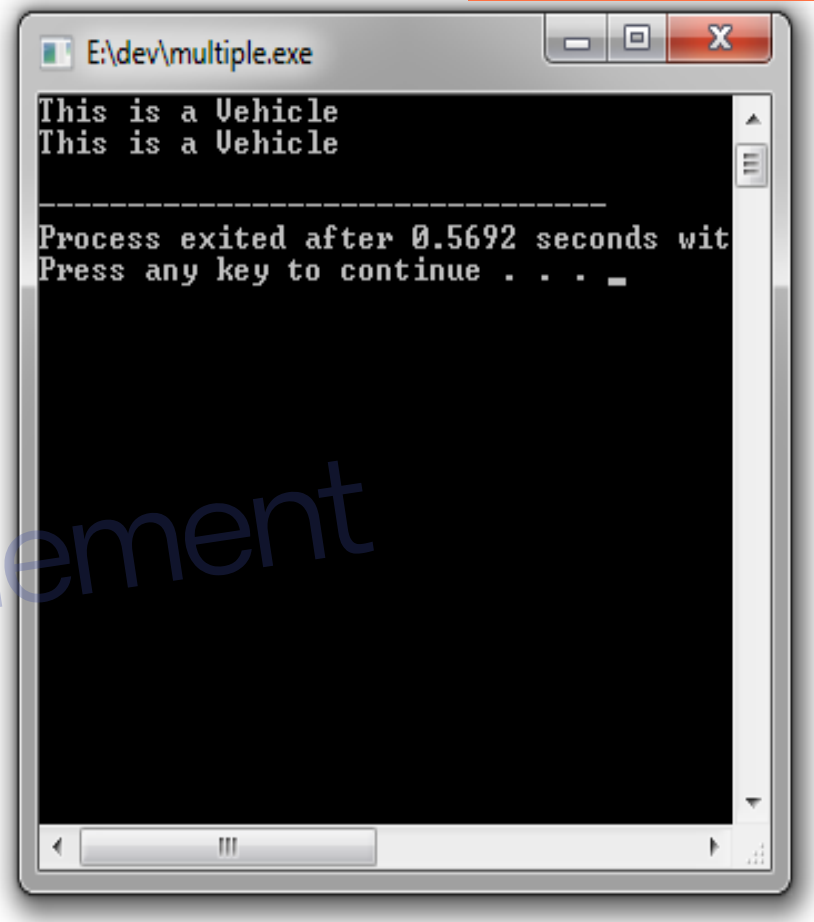
```cpp
// C++ program to explain
// multiple inheritance
#include <iostream>
using namespace std;
// first base class
class Vehicle {
  public:
    Vehicle()
    {
      cout << "This is a Vehicle" << endl; }
};
 // second base class
class FourWheeler {
  public:
    FourWheeler()
   {cout << "This is a 4 wheeler Vehicle" << endl;
        } };
// sub class derived from two base classes
class Car: public Vehicle, public FourWheeler { };
 // main function
int main()
{
// creating object of sub class will
// invoke the constructor of base classes
        Car obj;
    return 0;
}
```



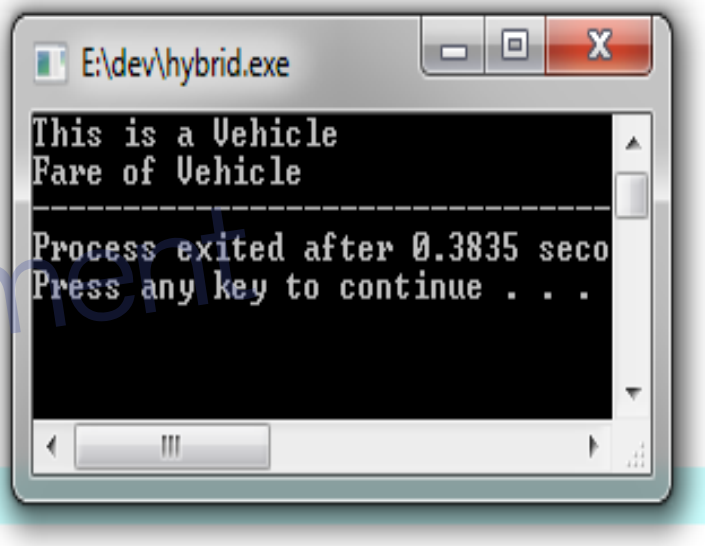
```cpp
// C++ program to implement   Multilevel Inheritance
#include <iostream>
using namespace std;
 // base class
class Vehicle
{   public:
    Vehicle()  {  cout << "This is a Vehicle" << endl;  }
};
class fourWheeler: public Vehicle
{  public:   fourWheeler()   {cout<<"Objects with 4 wheels are vehicles"<<endl; }
};
// sub class derived from two base classes
class Car: public fourWheeler{
   public:
     car()
     {
       cout<<"Car has 4 Wheels"<<endl;
     }
};

// main function
int main()
{
    //creating object of sub class will
    //invoke the constructor of base classes
    Car obj;
    return 0;
}
```



```cpp
// C++ program to implement
// Hierarchical Inheritance
#include <iostream>
using namespace std;

// base class
class Vehicle
{
  public:
    Vehicle()
    {
      cout << "This is a Vehicle" << endl;     } };


// first sub class
class Car: public Vehicle
{ };
  // second sub class
class Bus: public Vehicle
{ };
int main()
{
    // creating object of sub class will
    // invoke the constructor of base class
    Car obj1;
    Bus obj2;
    return 0;
}
```

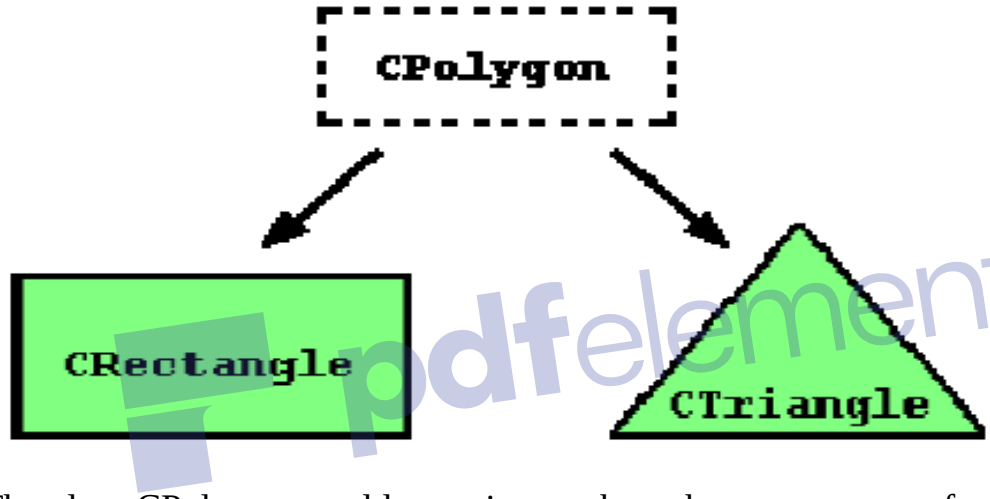


```cpp
// C++ program for Hybrid Inheritance
#include <iostream>
using namespace std;
// base class
class Vehicle
{   public:    Vehicle()    {  cout << "This is a Vehicle" << endl;  } };
 //base class
class Fare
{     public:   Fare()  {  cout<<"Fare of Vehicle";  } };
// first sub class
class Car: public Vehicle { };
  // second sub class
class Bus: public Vehicle, public Fare
{
 };
// main function
int main()
{
    // creating object of sub class will
    // invoke the constructor of base class
    Bus obj2;
    return 0;
}
```



Inheritance between classes

A key feature of C++ classes is inheritance. Inheritance allows to create classes which are derived from other classes, so that they automatically include some of its "parent's" members, plus its own. For example, we are going_to suppose that we want to declare a series of classes that describe polygons like our CRectangle, or like CTriangle. They have certain common properties, such as both can be described by means of only two sides: height and base.
This could be represented in the world of classes with a class CPolygon from which we would derive the two other ones: CRectangle and CTriangle.



The class CPolygon would contain members that are common for both types of polygon. In our case: width and height. And CRectangle and CTriangle would be its derived classes, with specific features that are different from one type of polygon to the other. Classes that are derived from others inherit all the accessible members of the base class. That means that if a base class includes a member A and we derive it to another class with another member called B, the derived class will contain both members A and B.
In order to derive a class from another, we use a colon (:) in the declaration of the derived class using the following format:

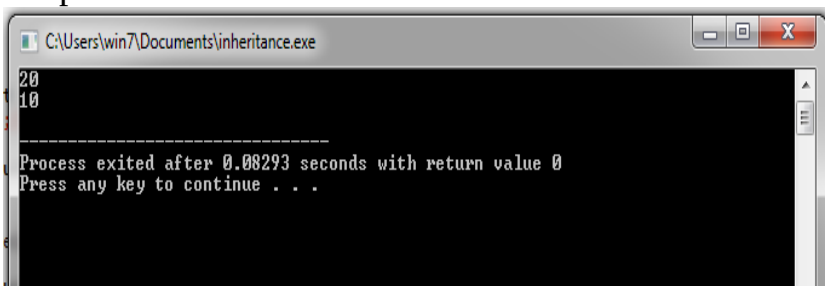
class derived_class_name: public base_class_name { /*...*/ };

Where deriveed_class_name is the name of the derived class_and base_class_name is the name of the class on which it is based. The public access specifier may be replaced by any one of the other access specifies protected and private. This access specifier describes the minimum access level for the members that are inherited from the base class.

EXAMPLE:

```
// derived classes
#include <iostream>
using namespace std;
class CPolygon {
protected:
int width, height;
public:
void set_values (int a, int b)
{ width=a; height=b;}
};
class CRectangle: public CPolygon {
public:
int area ()
{ return (width * height); }
};
class CTriangle: public CPolygon {
public:
int area ()
{ return (width * height / 2); }
};
int main () {
CRectangle rect;
CTriangle trgl;
rect.set_values (4,5);
trgl.set_values (4,5);
cout << rect.area() << endl;
cout << trgl.area() << endl;
return 0;
}
```

Output:



Private Inheritance

Consider the following classes:

```
class A { /*.....*/);
class C: private A
{ /*
 .
 .
.
.
 */
}
```

All the public parts of class A and all the protected parts of class A, become private members/parts of the derived class C in class C. No private member of class A can be accessed by class C. To do so, you need to write public or private functions in the Base class.
A public function can be accessed by any object, however, private function can be used only within the class hierarchy that is class A and class C and friends of these classes in the above cases.

Public Inheritance

Consider the following classes:
Consider the following classes:

```
class A{/*.......*/};
class E: public A
{ /*
 :
 :
 :
};
```

Now, all the public parts of class A become public in class E and protected part of A become protected in E.

Example:

Consider the following classes:

```
 class E: protected A
{ /*
 .
 .
 .
 */
};
```

Now, all the public and protected parts of class A become protected in class E.
No private member of class A can be accessed by class E.

We can summarize the different access types according to who can access them in the following way:

Access	public	protected	private
members of the same class	yes	yes	yes
members of derived classes	yes	yes	no
not members	yes	no	no

This public keyword after the colon (:) denotes the maximum access level for all the members inherited from the class that follows it (in this case CPolygon). Since public is the most accessible level, by specifying this keyword the derived class will inherit all the members with the same levels they had in the base class.
If we specify a more restrictive access level like protected, all public members of the base class are inherited as protected in the derived class. Whereas if we specify the most restricting of all access levels: private, all the base class members are inherited as private. For example, if daughter was a class derived from mother that we defined as:

class daughter: protected mother;

This would set protected as the maximum access level for the members of daughter that it inherited from mother. That is, all members that were public in mother would become protected in daughter. Of course, this would not restrict daughter to declare its own public members. That maximum access level is only set for the members inherited from mother.

If we do not explicitly specify any access level for the inheritance, the compiler assumes private for class**.**

What is inherited from the base class?

In principle, a derived class inherits every member of a base class **except:**

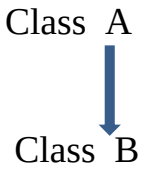
• Its constructor and its destructor.

• Its friends**.**

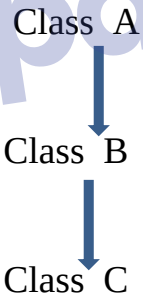
Although the constructors and destructors of the base class are not inherited themselves, its default constructor (i.e., its constructor with no parameters) and its destructor are always called when a new object of a derived class is created or destroyed.

Types of inheritance:

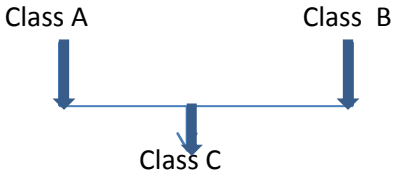
1-Single inheritance



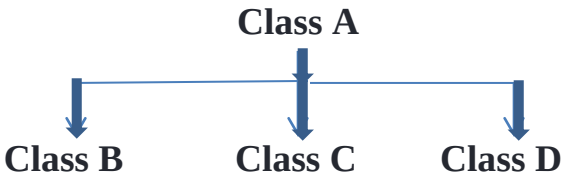
2- Multilevel inheritance



3- Multiple inheritances



4. Hierarchical inheritance



Example of Multilevel Inheritance

```
#include <iostream>
using namespace std;
class A
{    public: void display(){  cout<<"Base class content."; }
};
class B : public A{};
class C : public B{};
int main()
{   C obj; obj.display();
    return 0;
}
```


Multiple inheritances

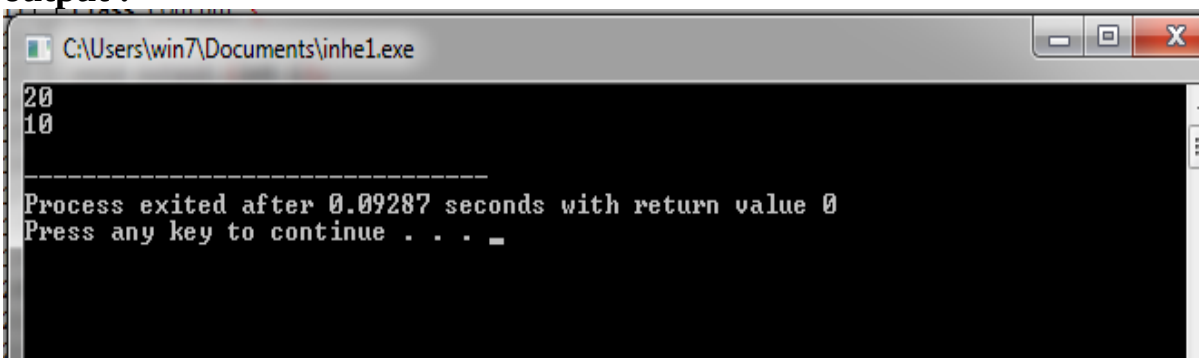
In C++ it is perfectly possible that a class inherits members from more than one class. This is done by simply separating the different base classes with commas in the derived class declaration. For example, if we had a specific class to print on screen (COutput) and we wanted our classes CRectangle and CTriangle to also inherit its members in addition to those of CPolygon we could write:

class CRectangle: public CPolygon, public COutput
class CTriangle: public CPolygon, public COutput;

Example:

```
// multiple inheritance
#include <iostream>
using namespace std;
class CPolygon {
protected:
int width, height;
public:
void set_values (int a, int b)
{ width=a; height=b;}
};
class COutput {
public:
void output (int i);
};
void COutput::output (int i) {
cout << i << endl;
}
class CRectangle: public CPolygon, public COutput {
public:
int area ()
{ return (width * height); }
};
class CTriangle: public CPolygon, public COutput {
public:
int area ()
{ return (width * height / 2); }
};
int main () {
CRectangle rect;
CTriangle trgl;
rect.set_values (4,5);
trgl.set_values (4,5);
rect.output (rect.area());
trgl.output (trgl.area());
return 0;}
```

output :



Questions:

Q1. If you have 2 inherited Classes A-> B .Class A contains (a1) string of 5 small letters , class B contains (b1) string of 5 small letters .Write an O.O.P for merge a1 with b1 into c1 ,then convert c1 to capital letters .

Q2. If you have 3 inherited classes A->B->C each class has an array of integer numbers a[3] ,b[3],c[3] .Write an O.O.P for combining all three digits into one number.

e.g. a[0]=8,b[0]=6,c[0]=2 the result=862

Q3. If you have 3-inherited classes A->B->C each one has an array a[4],b[4],c[4] respectively . Write an O.O.P for finding d [4], where each element of d[i] = the biggest no. of a[i], b[i], c[i], and so on.

More Examples of Inheritance:

Q. What is the result of following program and mention the type of inheritance and draw it?

```
#include <iostream>
        using namespace std;
        class student
        {
           public:
           int rno , m1 , m2 ;
           void get()
           {
              rno = 15, m1 = 10, m2 = 10;
           }
        };
        class sports
        {
           public:
           int sm;
           void getsm()
           {
              sm = 10;
           }
        };
        class statement:public student,public sports
        {
           int tot,avg;
           public:
           void display()
           {
              tot = (m1 + m2 + sm);
              avg = tot / 3;
              cout << tot <<"   ";
              cout << avg <<"    ";
           }
        };
        int main()
        {
           statement obj;
           obj.get();
           obj.getsm();
           obj.display();
}
```

Output : 30 10 type ? draw ?

Q. If you have four classes A-> B->C->D . Each class has one words ,write an o.o.p using inheritance to combine sentence. E.g Class A has the word "these", class B has the word "strings" ,class C has "are" and class D has "concatenated.

Q1. If you have 3 classes A, B, C .Class A contains a number in Decimal system, Class B contains a number in Binary system, class C contains a number in Octal system, Write an O.O.P program to convert the three values in different system to decimal system.

Q2. If you have two inherited classes A->B. Class A contains an array a [10] of type characters. Class B contains array b [10] of type character. Write an O.O.P program for printing the odd letter several times until print one letter of b[10] and print the even letter several times until print one letter of a[10] in between.

e.g a[10]=[A B C D E F G H I K],B[10]= [L M N O P Q R S T V]

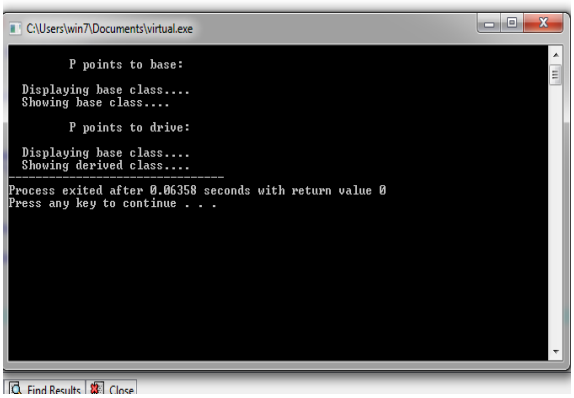
result= B L D N F O H Q S K V

Virtual Function

A virtual function is a function that is declared as virtual in a base class and redefined in one or more derived classes. A class that contains one or more virtual functions is called a polymorphic class. A virtual function is declared as virtual inside the base class by preceding its declaration with the keyword virtual. However, when a virtual function is redefined by a derived class, the keyword virtual need not be repeated.

A virtual function is a special form of member function that is declared within a base class and redefined by a derived class. The keyword virtual is used to create a virtual function, precede the function's declaration in the base class. If a class includes a virtual function and if it gets inherited, the virtual class redefines a virtual function to go with its own need. Example of virtual function:

```
#include <iostream>
using namespace std;
class b { public:
virtual void show()
{cout<<"\n  Showing base class....";}
void display(){cout<<"\n  Displaying base class...." ;}};
class d:public b { public:
void display(){ cout<<"\n  Displaying derived class....";}
void show(){cout<<"\n  Showing derived class....";}
};
int main()
{b B; b *ptr;
cout<<"\n\t P points to base:\n" ; ptr=&B; ptr->display();
ptr->show();
cout<<"\n\n\t P points to drive:\n"; d D; ptr=&D; ptr->display();
ptr->show();
}
```

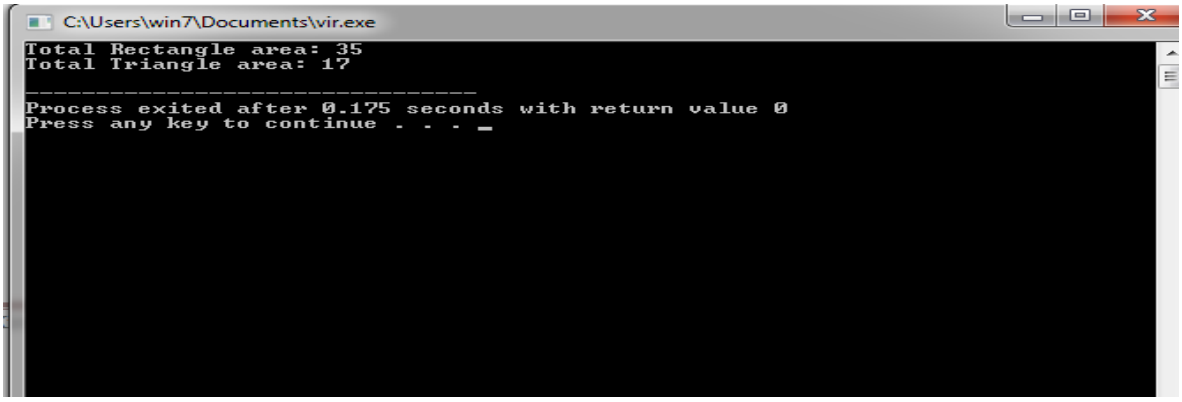


Example:

Finding the area of rectangle and triangle using virtual function

```
#include <iostream>
using namespace std;
// Base class
class Shape {   public:     // pure virtual function providing interface framework.
     virtual int getArea() = 0;
     void setWidth(int w) {  width = w;  }
       void setHeight(int h) { height = h;}
  protected:   int width;    int height; };
// Derived classes
class Rectangle: public Shape {
  public:   int getArea() {  return (width * height);  }};
class Triangle: public Shape {
  public:
    int getArea() { return (width * height)/2;  }};
int main(void) {   Rectangle Rect; Triangle  Tri;
  Rect.setWidth(5);
  Rect.setHeight(7);
  // Print the area of the object.
  cout << "Total Rectangle area: " << Rect.getArea() << endl;
  Tri.setWidth(5);  Tri.setHeight(7);
    // Print the area of the object.
  cout << "Total Triangle area: " << Tri.getArea() << endl;
  return 0;
}
```

Output:



Q1. Write an O.O.P for find the area of circle ,square ,and triangle using Virtual function.

Q2. Write an O.O.P for finding 2^3, 3^4, and 5^5 using virtual function.

Example:/ without virtual but with three pointer.

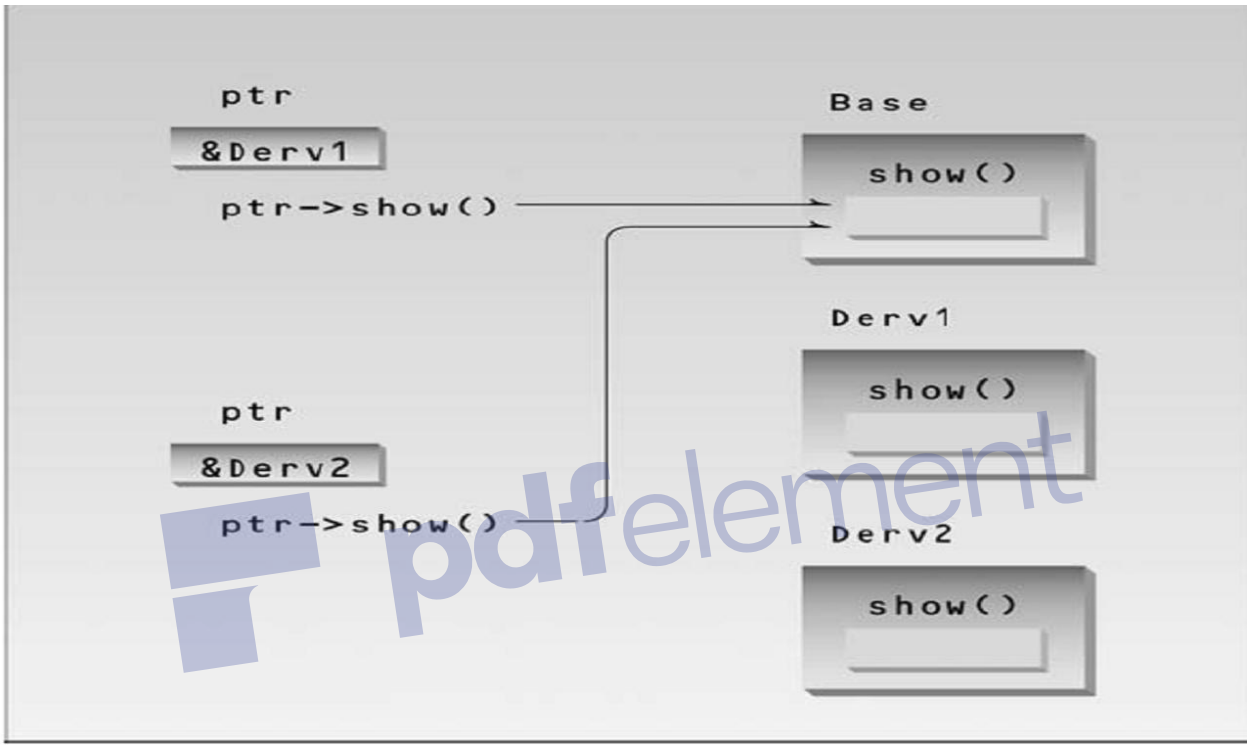
```
//without Virtual and with pointer
#include <iostream>
using namespace std;
class Base
{public :
        void show()//normal function
        {cout<<"Base \n"; }
};
//////////////////
class Derv1 :public Base //derived class
{
        public:
        void show(){ cout<<"Derv111 \n";}
        };
        ////////////////////////
 class Derv2 : public Base // derived class 2
 { public:
 void show() {cout<<"Derv2222 \n"; }
 };
 ///////////////
 int main()
 {Derv1 dv1;// object of class 1
 Derv2 dv2; // object of classs 2
 Base *ptr; // pointer to base class
 ptr= &dv1; //put address of dv1 in pointer
 ptr->show(); //execute show()
 ptr = &dv2; //put address of dv2 in pointer
 ptr->show(); //execute show()
 }
```

Result//

```
Base
Base
```

The Derv1 and Derv2 classes are derived from class Base. Each of these three classes has a member function show (). In main () we create objects of class Derv1 and Derv2, and a pointer to class Base. Then we put the address of a derived class object in the base class pointer in the line ptr = &dv1; // derived class address in

base class pointer .The rule is that pointers to objects of a derived class are type compatible with pointers to objects of the base class. Now the question is, when you execute the line ptr->show(); what function is called? Is it Base::show() or Derv1::show()? Again, in the last two lines of not virtual we put the address of an object of class Derv2 in the pointer, and again execute ptr->show(); Which of the show() functions is called here? The output from the program : Base Base
As follow:



Nonvirtual pointer access.

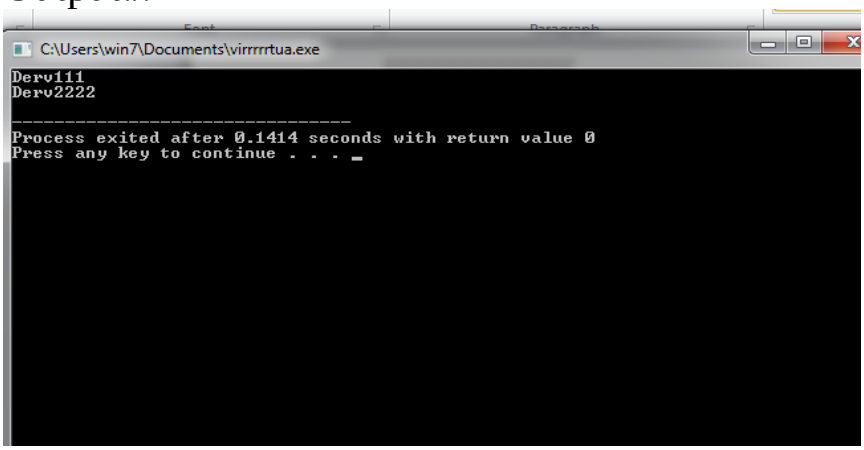
Virtual Member Functions Accessed with Pointers

Let's make a single change in our program: We'll place the keyword *virtual* in front of the declarator for the show() function in the base class. Here's the listing for the resulting program.

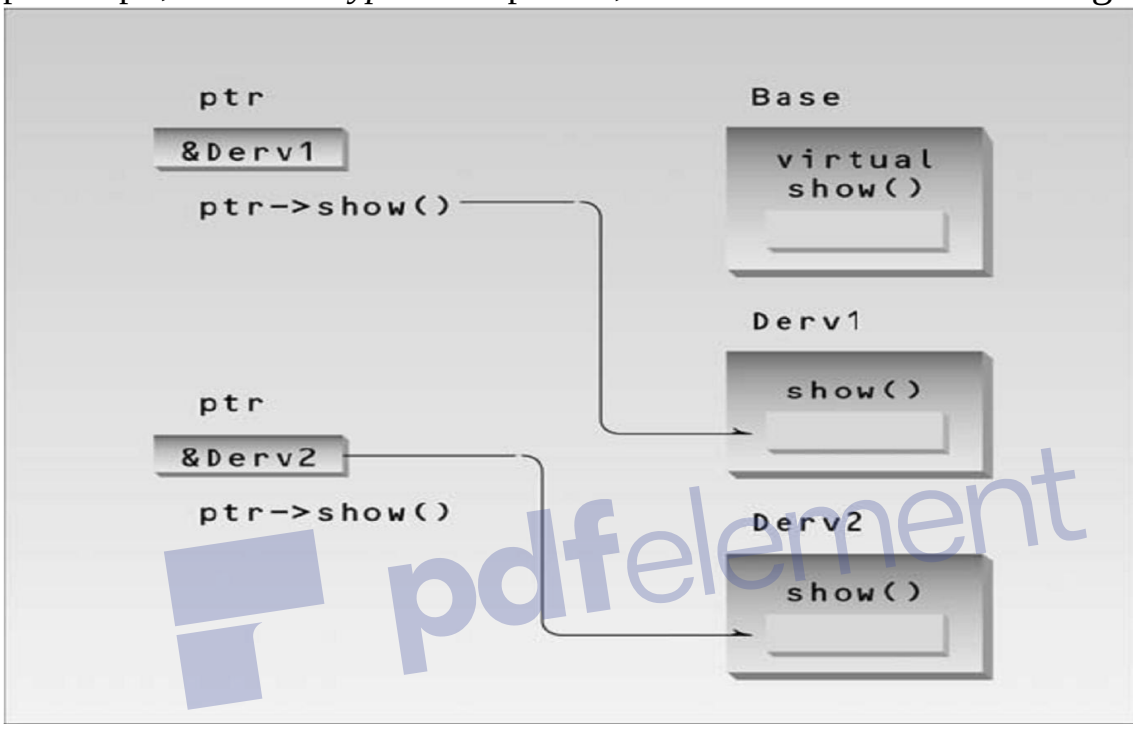
```
//with Virtual and with pointer
#include <iostream>
using namespace std;
class Base
{public :
  virtual    void show()//normal function
```

```
        {cout<<"Base \n";      }
};
//////////////////
class Derv1 :public Base //derived class
{
        public:
        void show(){ cout<<"Derv111 \n";}
        };
        ////////////////////////
 class Derv2 : public Base // derived class 2
 { public:
 void show() {cout<<"Derv2222 \n"; }
 };
 ///////////////
 int main()
 {Derv1 dv1;// object of class 1
 Derv2 dv2; // object of classs 2
 Base *ptr; // pointer to base class
 ptr= &dv1; //put address of dv1 in pointer
 ptr->show(); //execute show()
 ptr = &dv2; //put address of dv2 in pointer
 ptr->show(); //execute show()
 }
```

Output://



The member functions of the derived classes, not the base class, are executed. We change the contents of ptr from the address of Derv1 to that of Derv2, and the particular instance of show() that is executed also changes. So the same function call ptr->show(); executes different functions, depending on the contents of ptr. The rule is that the compiler selects the function based on the *contents* of the pointer ptr, not on the *type* of the pointer, as in not virtual. As following:



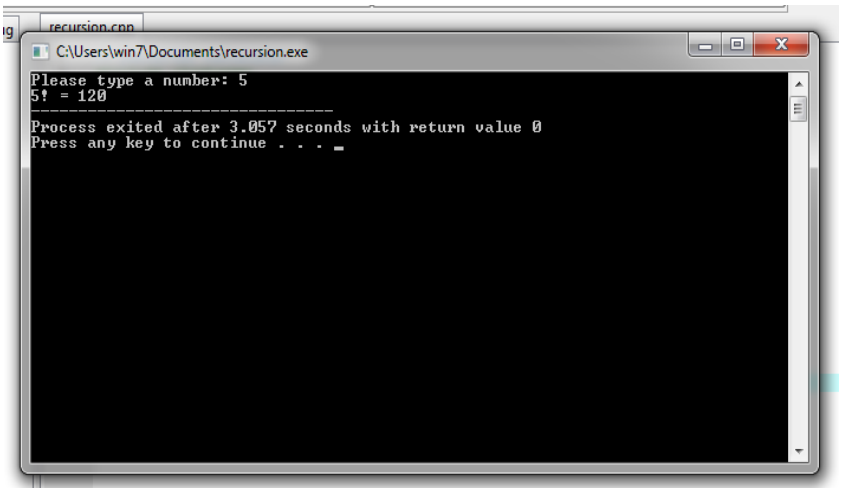
Recursion

A function that calls itself is said to be *recursive*. This process can also be performed indirectly if the function first calls another function or multiple functions before it is called once more. But a break criterion is always necessary to avoid having the function call itself infinitely.

The concept of local objects makes it possible to define recursive functions in C++. Recursion requires local objects to be created each time the function is called, and these objects must not have access to any other local objects from other function calls. What effectively happens is that the local objects are placed on the stack, and thus the object created last is destroyed first.

Recursion is much easier to understand with an example than with lengthy explanations.

```
// factorial calculator
#inlude <iostream>
using namespace std;
long factorial (long a)
{
 if (a > 1)
 return (a * factorial (a-1));
 else
 return (1);
}
int main ()
{
 long number;
 cout << "Please type a number: ";
 cin >> number;
 cout << number << "! = " << factorial (number);
 return 0;
}
```



The function factorial is another story. What's it doing? If n is greater than 1, the function calls itself. Notice that when it does this it uses an argument one less than the argument it was called with. Suppose it was called from main() with an argument of 5. It will call a second version of itself with an argument of 4. Then this function will call a third version with an argument of 3, and so on.
Notice that each version of the function stores its own value of n while it's busy calling another version of itself.
After factorial calls itself four times, the fifth version of the function is called with an argument of 1. It discovers this with the if statement, and instead of calling itself, as previous versions have, it returns 1 to the fourth version. The fourth version has stored a value of 2, so it multiplies the stored 2 by the returned 1, and returns 2 to the third version. The third version has stored 3, so it multiplies 3 by the returned 2, and returns 6 to the second version. The second version has stored 4, so it multiplies this by the returned 6 and returns 24 to the first version. The first version has stored 5, so it multiplies this by the returned 24 and returns 120 to main().Thus in this example we have five function calls followed by five function returns. Here's a summary of this process:

Version	*Action*	*Argument or Return Value*
1	Call	5
2	Call	4
3	Call	3
4	Call	2

Version	*Action*	*Argument or Return Value*
5	Call	1
5	Return	1
4	Return	2
3	Return	6
2	Return	24
1	Return	120

Every recursive function must be provided with a way to end the recursion. Otherwise it will call itself forever and crash the program. The if statement in factorial() plays this role, terminating the recursion when n is 1. Is it true that many versions of a recursive function are stored in memory while it's calling itself? Not really. Each version's variables are stored, but there's only one copy of the function's code. Even so, a deeply-nested recursion can create a great many stored variables, which can pose a problem to the system if it doesn't have enough space for them.

Q1. Write an O.O.P for finding the factorial of any number using recursion.

Q2. Write an O.O.P for finding $2^0 * 2^1 * 2^2 * \ldots * 2^n$.

Q3. Write an O.O.P for finding the sum of $3^1 + 3^2 + 3^3 + 3^4 + \ldots + 3^n$