Lab Exercises CS304 - 2nd

Object Oriented Programming (Virtual University of Pakistan)

Programming Labs 1-11

CS304 – Object Oriented Programming

	How to use static data members.	
6	**Lab 06:** Ascertaining the concept of Operator overloading.	11
7	**Lab 07:** Implementation of overloading stream insertion operator (<<) and stream extraction operator (>>).	14
Mid Term		
8	**Lab 08:** Exploring the concept of Inheritance between different classes.	16
9	**Lab 09:** Determining the use of Access specifiers and understanding compile time errors.	19
10	**Lab 10:** Application of the concept of Polymorphism in a C++ program. **Part (1)** How to create object for static data type. **Part (2)** How to achieve polymorphism using virtual function.	23
11	**Lab 11:** Implementation of Partial specialization.	27s

Lab 01

Identify all objects in the following paragraph.

In Virtual University bookshop, the handouts of all courses are provided at reasonable price. The handouts are made in accordance with the video lectures recorded by famous professors. It is a step to facilitate students to easily digest the course contents.

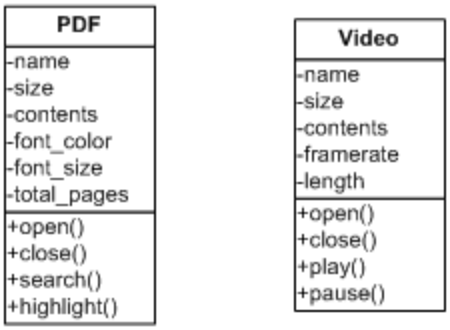
Solution:

We have identified the following classes according to the given scenario.

1. University
2. Bookshop
3. Handouts
4. Course
5. Video lecture
6. Professor
7. Student
8. Course content

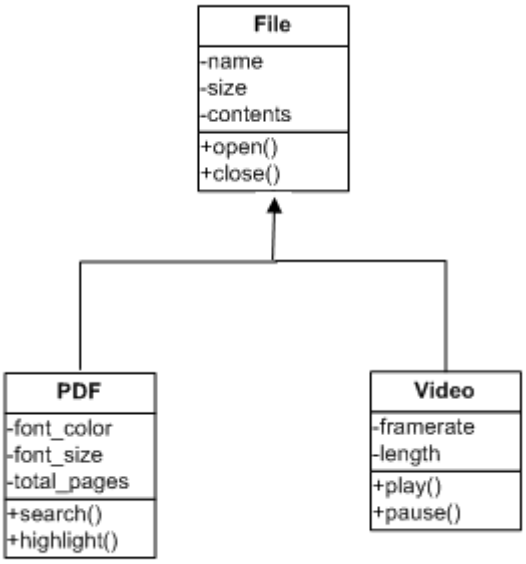
Lab 02

In the below diagram we have two different types of file classes (PDF and Video). You are required to apply the concept of generalization on the given classes.



Solution:

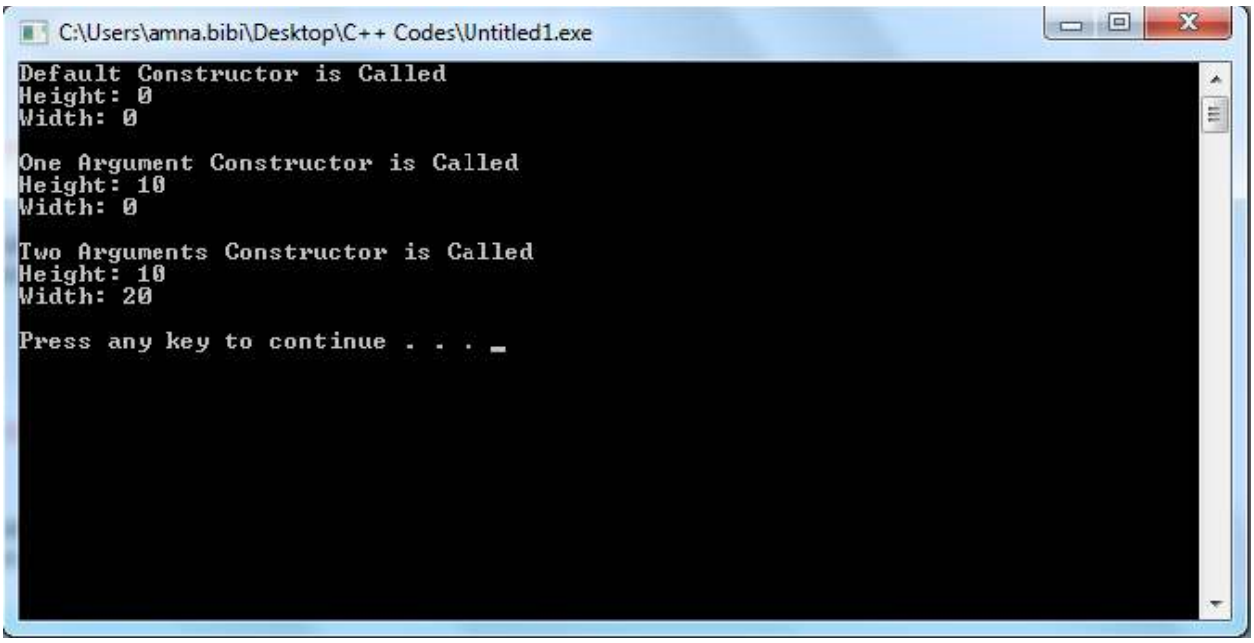
Both classes are kind of File, so we make a File class having common attributes of both PDF and Video classes.Then apply generalization using arrow symbol between File and Video Lecture classes, and File and PDF classes.



Lab 03

Write a program which consists of a class named **Room** having two data members **Height** and **Width**, the class should also consist of three constructors i.e. Default constructor, one argument constructor and two arguments constructor.

After running your program, the following screen should display.



Solution:

```
#include <iostream>

using namespace std;

class Room


{

        private:

        int height;

        int width;
```

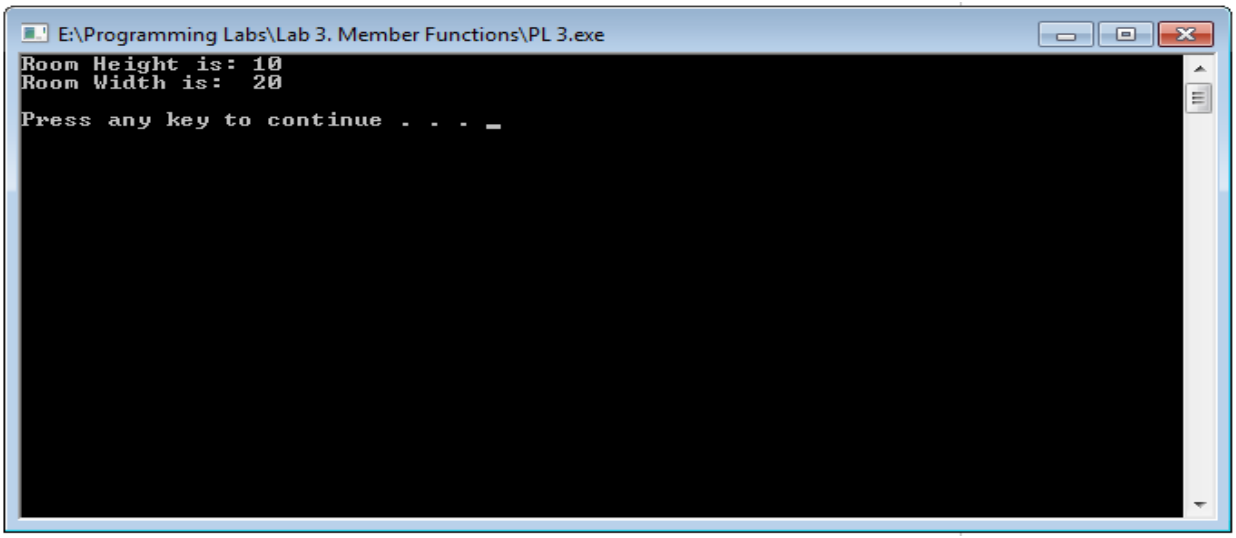
```
public:
        Room()
        {
cout<<"The default constructor is called"<<endl;
height=0;
cout<<"Height: "<<height<<endl;
width=0;
cout<<"Width: "<<width<<endl<<endl;
        }
 Room(int h)
 {
cout<<"1 Parameter constructor is called"<<endl;
                height=h;
cout<<"Height: "<<height<<endl;
width=0;
cout<<"Width: "<<width<<endl<<endl;
 }
 Room(int h, int w)
 {
cout<<"2 Parameter constructor is called"<<endl;
                height=h;
cout<<"Height: "<<height<<endl;
width=w;
cout<<"Width: "<<width<<endl<<endl;
```

```
        }


};

        int main(){
                Room x;
                Room y(20);
                Room z(20,30);
                //system("pause");
                cin.get();

        }
```

Lab 04

Write a program which consists of a class named **Room** having two data members **Height** and **Width**, the class should also consist of setter and getter functions for both data members. Use setter function to set values of data members as 10, 20 respectively for Height and Width. Display room size using getter function.

After running your program, the following screen should display.



Solution:

```
#include <iostream>
using namespace std;
class Room{
	private:
		float height;
		float width;
	public:
		Room(){};
		void setHeight(float);
		float getHeight() const;
		void setWidth(float);
		float getWidth() const;
		void displayMessage();
};
void Room::displayMessage(){
	cout << "Room Height is: "<< height << endl;
	cout << "Room Width is:  "<< width <<endl<<endl;
}
```

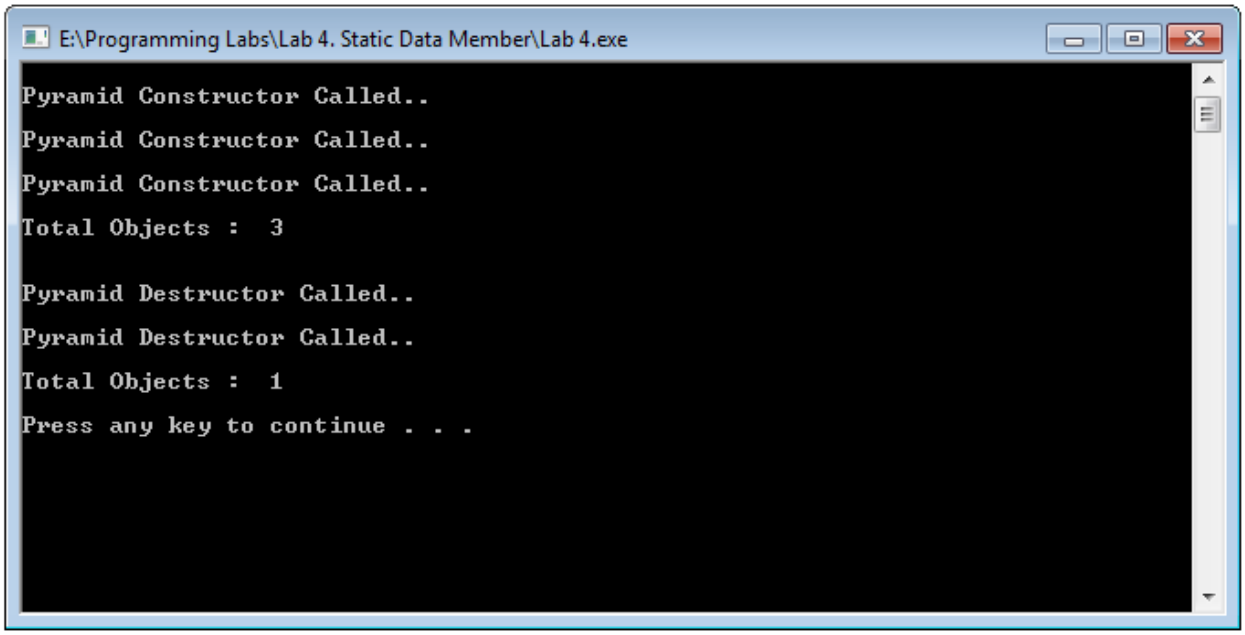
```
void Room::setHeight(float height){
	this->height = height;
}

float Room::getHeight() const{
	return height;
}
void Room::setWidth(float width){
	this->width = width;
}
float Room::getWidth() const{
	return width;
}
main(){
	Room room;
	room.setHeight(10);
	room.setWidth(20);

	room.displayMessage();
	system("pause");
}
```

Lab 05

Write C++ coding program that consists of a class Pyramid. The class should contain a static data member "Pcount" which stores the total number of Pyramid's objects.

In the main() function, dynamically create three objects of Pyramid class and print the value of "Pcount". Afterwards, delete any two objects of Pyramid and print the value of "Pcount" again.

Following is a sample output for above scenario:



Solution:

```
#include<iostream>
using namespace std;
class Pyramid
{
        private:
                static int Pcount;
        public:
                static int get_pcount();
                Pyramid();
                ~Pyramid();
};
main()
```

```
{
	Pyramid *p1 = new Pyramid;
	Pyramid *p2 = new Pyramid;
	Pyramid *p3 = new Pyramid;
	cout << endl << "Total Objects :  " << Pyramid::get_pcount() << endl << endl;

	delete p1;
	delete p2;

	cout << endl << "Total Objects :  " << Pyramid::get_pcount() << endl << endl;

	system("pause");
}
int Pyramid::Pcount = 0;

int ::Pyramid::get_pcount()
{
	return Pcount;
}
Pyramid::Pyramid()
{
	cout << endl << "Pyramid Constructor Called.." << endl;
	Pcount++;
}
Pyramid::~Pyramid()
{
	cout << endl << "Pyramid Destructor Called.." << endl;
	Pcount--;
}
```

Lab 06

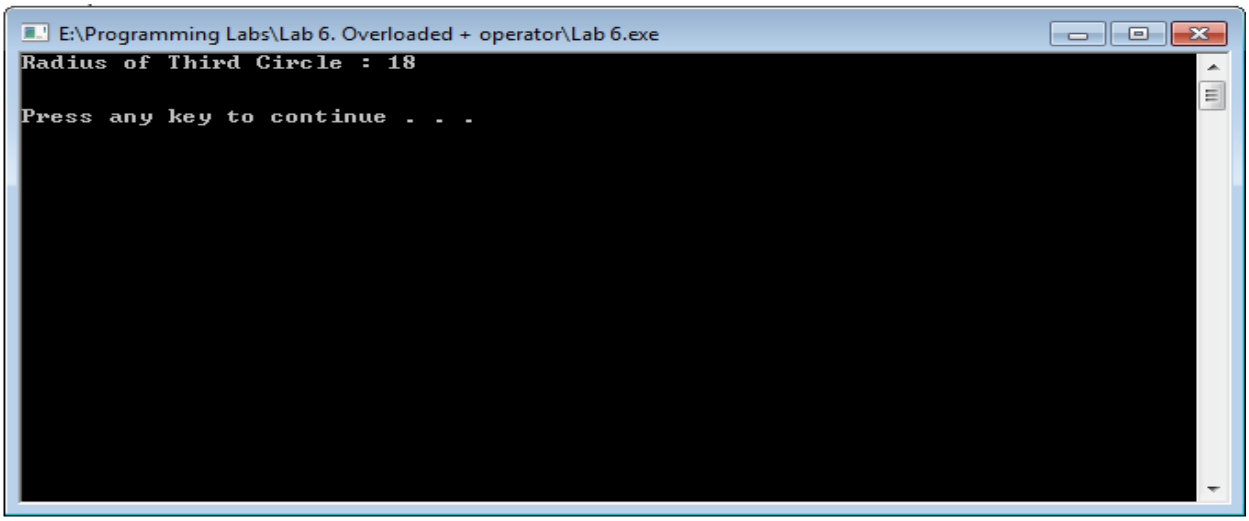
Write a C++ program which consists of a class named **Circle** which has one data member **radius** and **setter and getter functions**. Overload the **+ operator** for this class.

In main () function, create three objects c1, c2 and c3. Call setter function for objects c1 and c2. The program should be able to execute the following statement;

c3=c1+c2;

Call the getter function for c3.

After running your program, the following screen should display.



Solution:

```
#include <iostream>

using namespace std;

class Circle {

  public:

    void setRadius( double r ) {
```

```
      radius = r;
   }
  double getRadius( ) {
      return radius;
   }
   // Overload + operator to add two Box objects.
   Circle operator+(const Circle& c) {
      Circle cir;
      cir.radius = this->radius + c.radius;
      return cir;
   }
 private:
   double radius;     // radius of circle
};
// Main function for the program
int main( ) {
 Circle c1,c2,c3;
 double r;
 // box 1 specification
 c1.setRadius(6.0);


 // box 2 specification
```

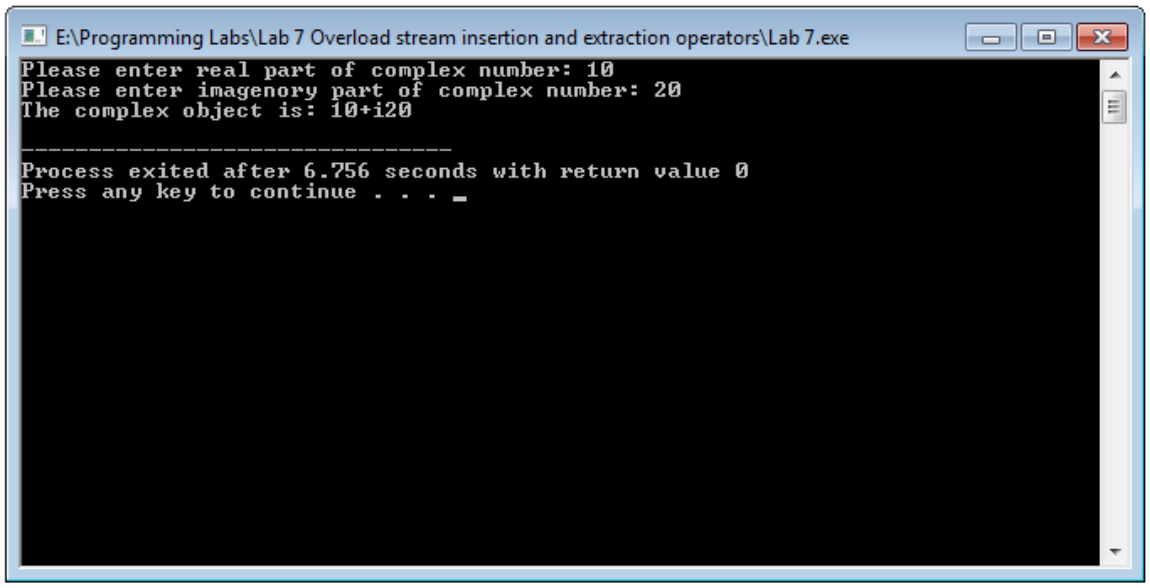
```
  c2.setRadius(12.0);


 // Add two object as follows:

  c3 = c1 + c2;


  // volume of box 3

  r = c3.getRadius();

  cout << "Radius of Third Circle : " << r <<endl<<endl<<endl;

        system("pause");

        return 0;

}
```

Lab 07

Write a C++ program which consists of a class named **Complex** which has data members **real** and **imaginary**. Overload the **stream insertion operator (<<) operator** and **stream extraction operator (>>)** for this class.

In main () function, create an object com1 and take input using stream extraction operator and print output using stream insertion operator

After running your program, the following screen should display.



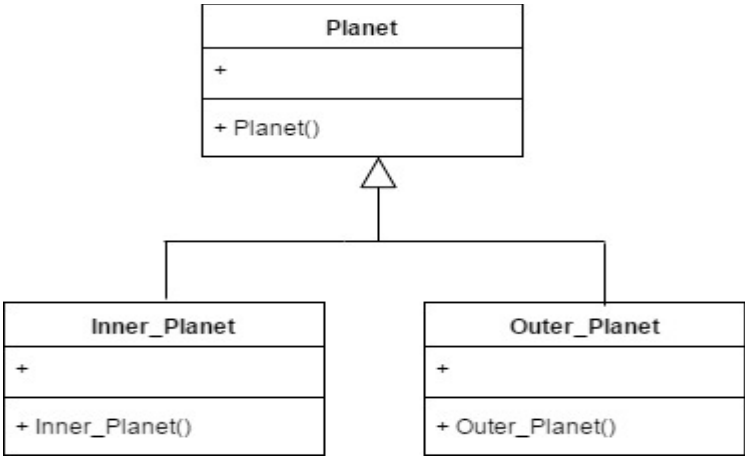
Solution:

```
#include<iostream>
#include<string.h>
#include <assert.h>
#include <iostream>
using namespace std;

class Complex
{
private:
        int real, imag;
public:
        Complex(int r = 0, int i =0)
        { real = r; imag = i; }
        friend ostream & operator << (ostream &outObj, const Complex &com); //stream insertion
operator
        friend istream & operator >> (istream &inObj, Complex &com); //stream extraction operator
```

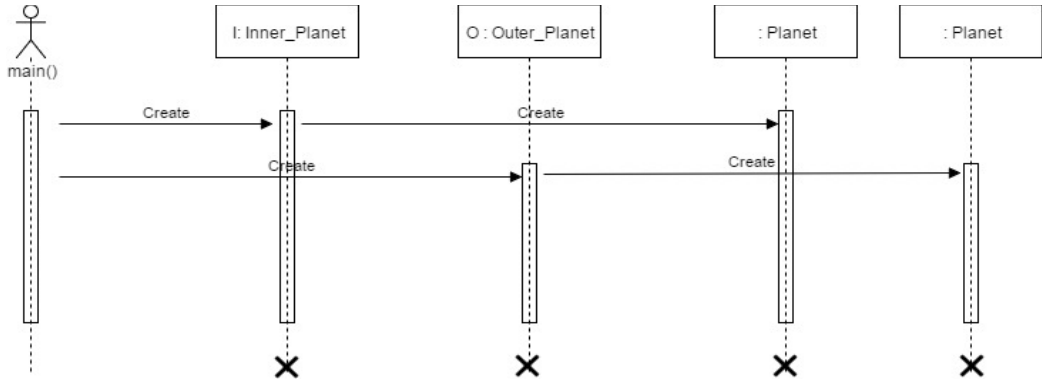
```
};
ostream & operator << (ostream &outObj, const Complex &com)
{
	outObj << com.real;
	outObj << "+i" << com.imag << endl;
	return outObj;
}
istream & operator >> (istream &inObj, Complex &com)
{
	cout << "Please enter real part of complex number: ";
	inObj >> com.real;
	cout << "Please enter imagenory part of complex number: ";
	inObj >> com.imag;
	return inObj;
}
int main()
{
Complex com1;
cin >> com1;
cout << "The complex object is: ";
cout << com1;
return 0;
}
```

Lab 08

Write a C++ program which consists of a class named **Planet** which has no data member and only contains a default constructor. Your program should contain two more classes **Inner_Planet** and **Outer_Planet**. Both of these classes should also contain default constructor. Your program should implement the concept of inheritance between **"Planet class, Inner_Planet class and Outer_Planet class".**

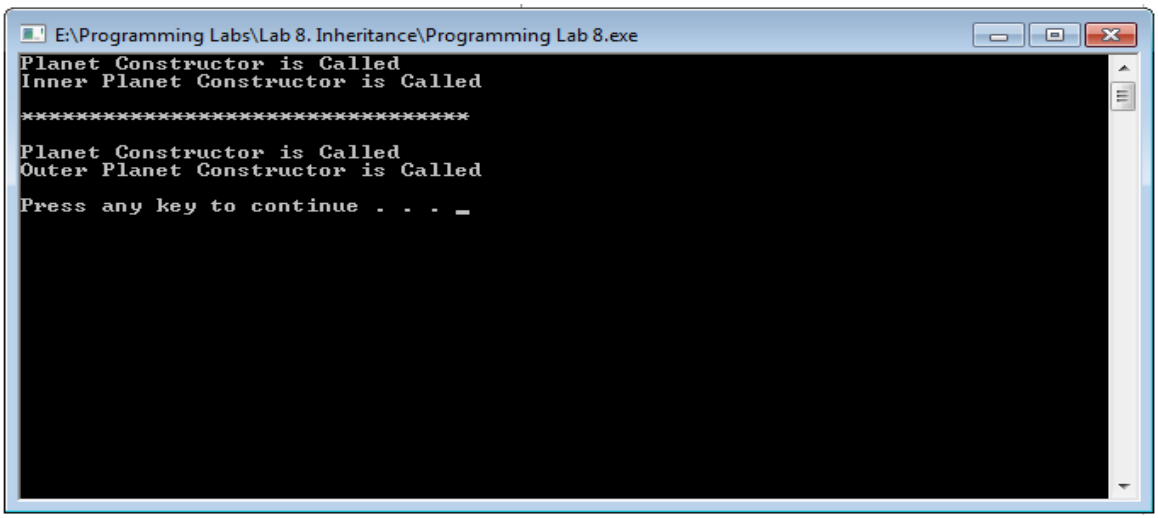


Sequence Diagram:



Sample Output:

After running your program, the following output screen should display.



Solution:

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

class Planet
{
    private:

        public:
        Planet()
        {
```

```
            cout<<"Planet Constructor is Called"<<endl;
        }

};

class Inner_Planet:Planet
{
    private:
     float distance;
        public:
        Inner_Planet()
        {
            cout<<"Inner Planet Constructor is Called"<<endl;
        }

};
class Outer_Planet:Planet
{
    private:
        float distance;
        public:
        Outer_Planet()
        {
            cout<<"Outer Planet Constructor is Called"<<endl;
        }

};
main()
{

    Inner_Planet I;
    cout<<"\n**********************************\n\n";
    Outer_Planet O;
    cout<<"\n";
    system("pause");

}
```

Lab 09

Write a C++ program which should consist of following four classes.

1. Base
2. Derived_Private
3. Derived_Protected
4. Derived_Public

“Base” class should consist of following data members with following type.

Data member	**Access Specifier**
secret	private
protect	protected
access	public

Further, “Base” class should contain default constructor to initialize its data members.

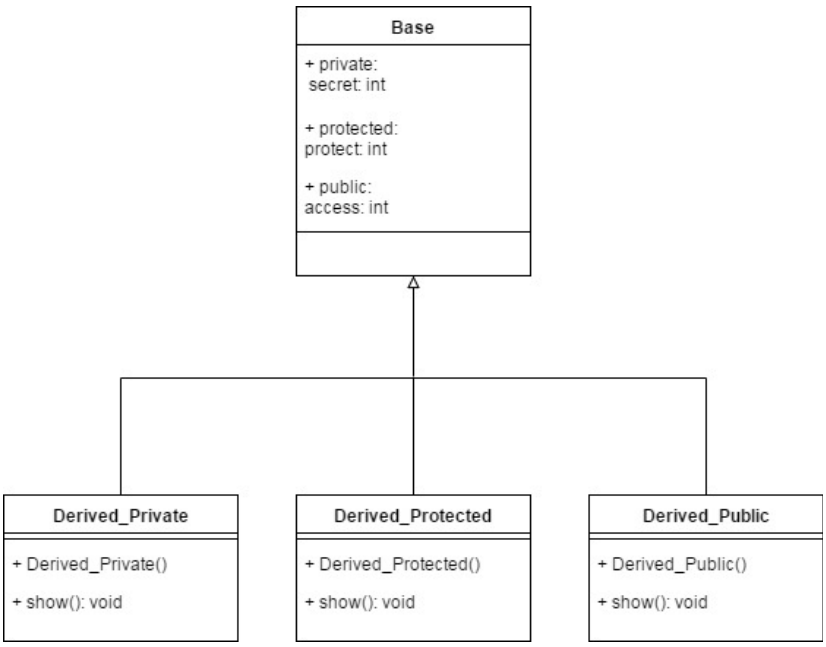
Other three classes should be inherited from “Base” class with respect to following type of inheritance.

Class	**Inheritance type**
Derived_Private	Private
Derived_Protected	Protected
Derived_Public	Public

All three derived classes should contain **show() function**, which should display the values of **secret, protect** and **access** data member of base class. On running this program, your compiler will generate

compile time errors. Carefully observe for which type of base class data member, compiler is generating error.

Class Diagram:



Sample Output:

Compiler (9) | Resources | Compile Log | Debug | Find Results | Close

Line	Col	File	Message
		E:\Programming Labs\Lab 9. Inheritance 2\Lab 9.cpp	**In member function 'void Derived_Private::show()':**
7	7	E:\Programming Labs\Lab 9. Inheritance 2\Lab 9.cpp	[Error] 'int Base::secret' is private
33	25	E:\Programming Labs\Lab 9. Inheritance 2\Lab 9.cpp	[Error] within this context
		E:\Programming Labs\Lab 9. Inheritance 2\Lab 9.cpp	**In member function 'void Derived_Protected::show()':**
7	7	E:\Programming Labs\Lab 9. Inheritance 2\Lab 9.cpp	[Error] 'int Base::secret' is private
44	25	E:\Programming Labs\Lab 9. Inheritance 2\Lab 9.cpp	[Error] within this context
		E:\Programming Labs\Lab 9. Inheritance 2\Lab 9.cpp	**In member function 'void Derived_Public::show()':**
7	7	E:\Programming Labs\Lab 9. Inheritance 2\Lab 9.cpp	[Error] 'int Base::secret' is private
55	25	E:\Programming Labs\Lab 9. Inheritance 2\Lab 9.cpp	[Error] within this context

Solution:

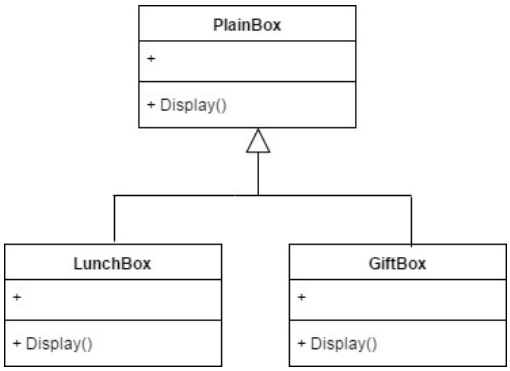
```
#include<iostream>
using namespace std;

class Base
{
```

```
        private:
                int secret;

                protected:
                int protect;

          public:
                int access;

        public:

Base()
   {
            access=0;
            protect=0;
            secret=0;
   }
};
class Derived_Private: private Base
{
    public:
    void show()
        {
            cout<<access;  // access is public, so it will be accessed from Derived_Public class.
            cout<<protect; // protect is proteced and will be accessed from Derived_Public class.
            cout<<secret;  // Error. Secret can't be accessed from Derived_Public class because it is private.
        }
};
class Derived_Protected: protected Base
{
    public:
     void show()
        {
            cout<<access;  // access is public, so it will be accessed from Derived_Public class.
            cout<<protect; // protect is proteced and will be accessed from Derived_Public class.
            cout<<secret;  // Error. Secret can't be accessed from Derived_Public class because it is private.
        }
};
class Derived_Public: public Base
{
    public:
        void show()
        {
            cout<<access;  // access is public, so it will be accessed from Derived_Public class.
            cout<<protect; // protect is proteced and will be accessed from Derived_Public class.
            cout<<secret;  // Error. Secret can't be accessed from Derived_Public class because it is private.
        }
```

```
};

int main()
{/*
   Derived_Public child1;
   cout<<"This is accessability of base data members in derived class in case of Public
Inheritance"<<endl;
   child1.show();
   cout<<endl<<endl;

   Derived_Protected child2;
   cout<<"This is accessability of base data members in derived class in case of Protected
Inheritance"<<endl;
   child2.show();
   cout<<endl<<endl;


   Derived_Private child3;
   cout<<"This is accessability of base data members in derived class in case of Private
Inheritance"<<endl;
   child3.show();
   cout<<endl<<endl;
   */
   return 0;
}
```

Lab 10

Class Diagram:



Part (1):

Write C++ program which should contain three classes as follows:

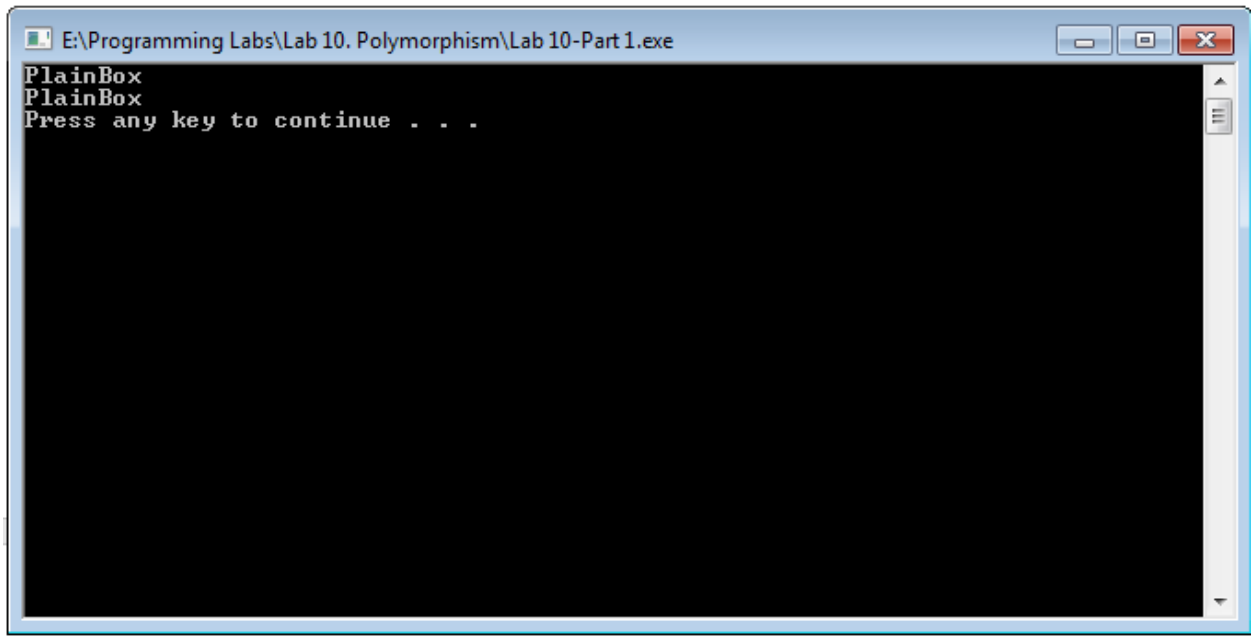
1. PlainBox
2. LunchBox
3. GiftBox

“LunchBox” and “GiftBox” classes should be publically inherited from PlainBox Class.

All three classes should contain Display() function, which should display respective class name.

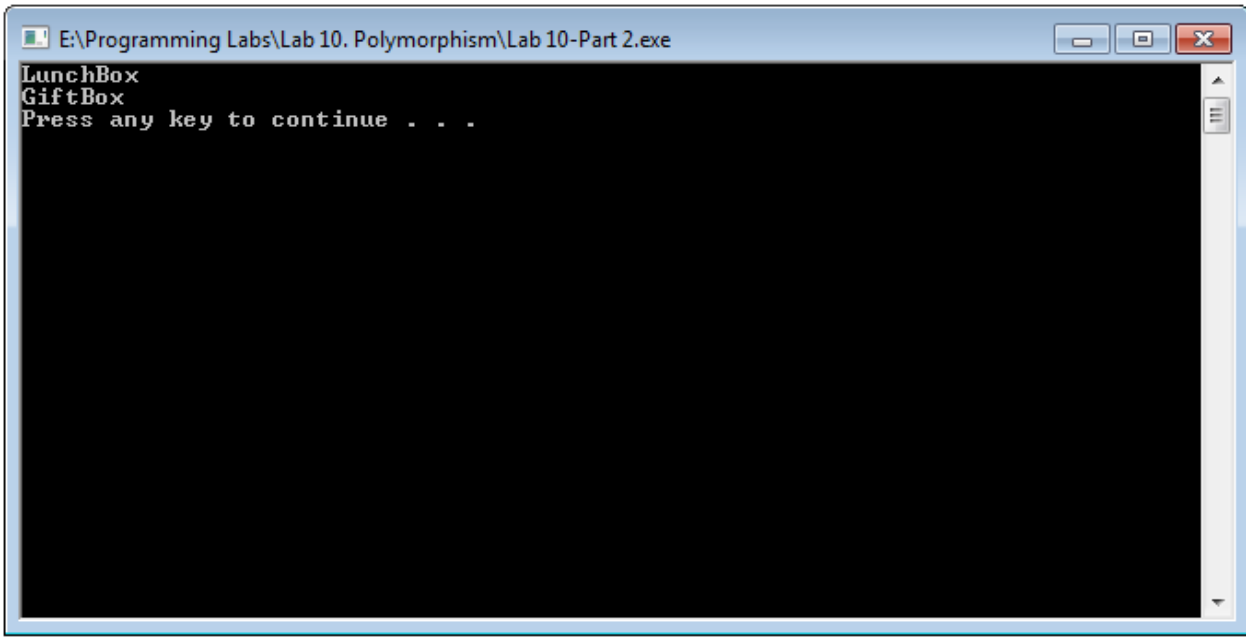
In main() function, create objects of LunchBox and GiftBox classes dynamically. Static type of these objects should be “PlianBox”. Using object of PlianBox, call Display() function.

Following is a sample output for the above scenario:



Part (2):

Make Display() function of “PlainBox” class virtual and then run the program. Your output should be as follows:



Solution:
//Lab 10 Part 1

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

class PlainBox
{

        public:
   void Display()
   {
        cout<<"PlainBox"<<endl;
   }

};
class LunchBox: public PlainBox
{
    public:
   void Display()
   {
        cout<<"LunchBox"<<endl;
   }

};
class GiftBox: public PlainBox
{
    public:
   void Display()
```

```
    {
        cout<<"GiftBox"<<endl;
    }

};

int main()
{
    PlainBox *P1= new LunchBox();
    PlainBox *P2= new GiftBox();

    P1->Display();
    P2->Display();
    system("pause");
    return 0;
}
```

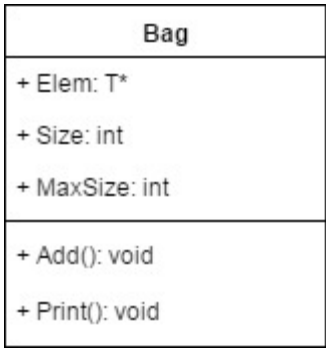
//Lab 10- Part 2

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

class PlainBox
{
        public:
    virtual void Display()
    {
        cout<<"PlainBox"<<endl;
    }

};
class LunchBox: public PlainBox
{
     public:
    void Display()
    {
        cout<<"LunchBox"<<endl;
    }

};
class GiftBox: public PlainBox
{
     public:
    void Display()
    {
        cout<<"GiftBox"<<endl;
    }
```

```
};


int main()
{
   PlainBox *P1= new LunchBox();
   PlainBox *P2= new GiftBox();

   P1->Display();
   P2->Display();
   system("pause");
   return 0;
}
```

Lab 11

Write C++ code for the following template Bag class.

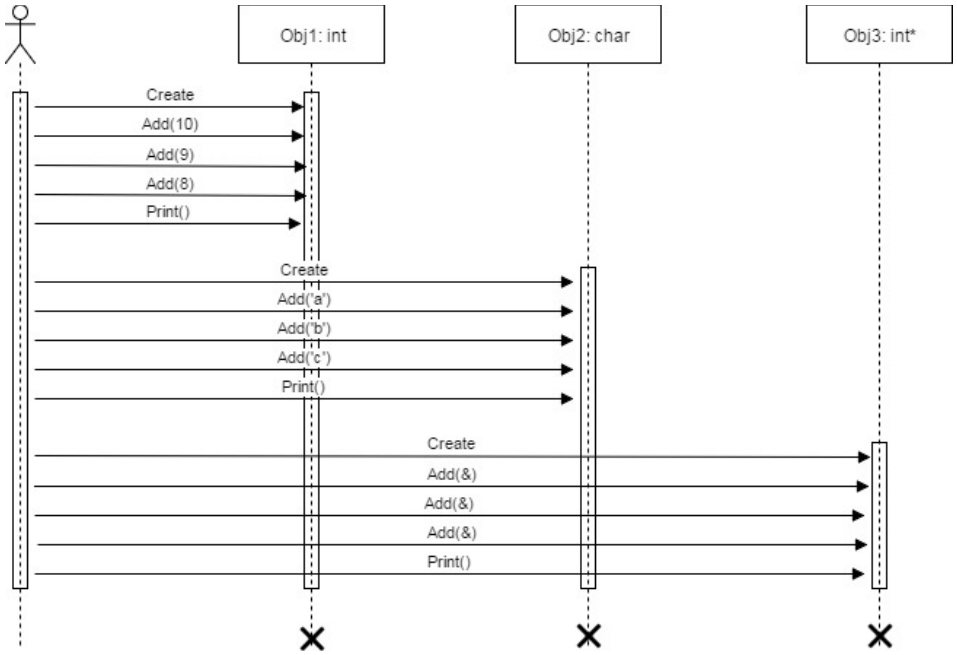


Element is an array which stores the elements present in the bag. Elements can store different types of elements. Size is the number of elements present in Bag and MaxSize hold the max number of element's a bag can have.

Add method is adding element present in bag into the array Elem and print will print the added elements on screen. Implement partial specialization for the given class to handle pointer to any datatype.

In main create 3 objects one for each integer, character and integer type pointer datatypes. Add at least three elements in each.

Sequence Diagram:



Sample Output:

After running your program, the following output screen should display