Object Oriented Programming

Lab Manual

Presented By

Asad
BSF2201587

Table Of Contents

Object Orientation in C++

Data Types, Objects, Classes and Instances

C++ lets you create variables which can be any of a range of basic data types: int, long, double and so on. However, the variables of the basic types don't allow you to model real-world objects (or even imaginary objects) adequately. It's hard to model a box in terms of an int, for example; what we need is something that allows us to collect together the various attributes of an object. In C++, we can do this very easily using classes. You could define variables, length, breadth and height to represent the dimensions of the box and bind them together as members of a Box class, as follows:

```
class Box
{
public:
double length;
double breadth;
double height;
};
```

This code actually creates a new type, called Box. The keyword class defines Box as such, and the elements that make up this class are defined within the curly braces. Note that each line defining an element of the class is terminated by a semicolon, and that a semicolon also appears after the closing brace. The elements length, breadth and height are referred to as data members. At the top of the class definition, you can see we have put the keyword public - this just means that the data members are generally accessible from outside the class. You may, however, place restrictions on the accessibility of class members, such as private or protected.

Class

The notion of class was invented by an Englishman to keep the general population happy. It derives from the theory that people who knew their place and function in society would be much more secure and comfortable in life than those who did not. The famous Dane, Bjarne Stroustrup, who invented C++, undoubtedly acquired a deep knowledge of class concepts while at Cambridge University in England, and appropriated the idea very successfully for use in his

new language. The idea of a class in C++ is similar to the English concept, in that each class usually has a very precise role and a permitted set of actions. However, it differs from the English idea, because class in C++ has largely socialist overtones, concentrating on the importance of working classes. Indeed, in some ways it is the reverse of the English ideal, because, as working classes in C++ often live on the backs of classes that do nothing at all.

Operations on Classes

In C++ you can create new data types as classes to represent whatever kinds of objects you like. As you'll come to see, classes aren't limited to just holding data; you can also define member functions that act on your objects, or even operations that act between objects of your classes using the standard C++ operators. You can define the class Box, for example, so that the following statements work and have the meanings you want them to have:

```
Box Box1;
Box Box2;
if(Box1 > Box2) // Fill the larger box
Box1.Fill();
else
Box2.Fill();
```

You could also implement operations as part of the Box class for adding, subtracting or even multiplying boxes in fact, almost any operation to which you could ascribe a sensible meaning in the context of boxes. We're talking about incredibly powerful medicine here and it constitutes a major change in the approach that we can take to programming. Program design now starts with deciding what new application-specific data types you need to solve the problem in hand and writing the program in terms of operations on the specifics that the problem is concerned with, be it Coffins or Cowpokes.

Terminology

Let's summarize some of the terminology that will be using when discussing classes in C++:

• A class is a user-defined data type

• Object-oriented programming is the programming style based on the idea of defining your own data types as classes

• Declaring an object of a class is sometimes referred to as instantiation because

you are creating an instance of a class
• Instances of a class are referred to as objects
• The idea of an object containing the data implicit in its definition, together with the functions that operate on that data, is referred to as **encapsulation.**

Defining a Class

Let's look again at the class we started talking about at the start of the chapter - a class of boxes. We defined the Box data type using the keyword class as follows:

```
class Box
{
public:
double length;
double breadth;
double height;
};
```

The name that we've given to our class appears following the keyword and the three data members are defined between the curly braces. The data members are defined for the class using the declaration statements that we already know and love, and the whole class definition is terminated with a semicolon. The names of all the members of a class are local to a class. You can therefore use the same names elsewhere in a program without causing any problems.

Access Control in a Class

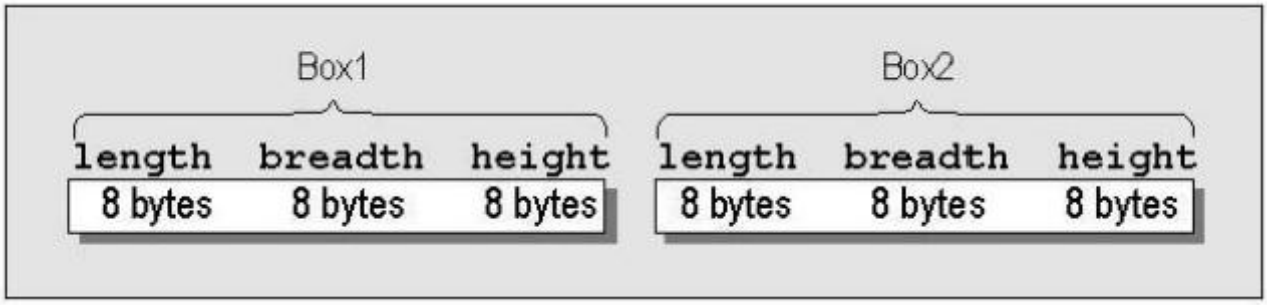
The keyword public looks a bit like a label, but in fact it is more than that. It determines the access attributes of the members of the class that follow it. Specifying the data members as public means that these members of an object of the class can be accessed anywhere within the scope of the class object. You can also specify the members of a class as private or protected. In fact, if you omit the access specification altogether, the members have the default attribute, private. We shall look into the effect of these keywords in a class definition a bit later. Remember that all we have defined so far is a class, which is a data type. We haven't declared any objects of the class. When we talk about accessing a class member, say height, we're talking about accessing the data member of a particular object, and that object needs to be declared somewhere.

Declaring Objects of a Class

We declare objects of a class with exactly the same sort of declaration that we use to declare objects of basic types. We saw this at the beginning of the chapter. So, we could declare objects of our class, Box, with these statements:

```
Box Box1;        // Declare Box1 of type Box
Box Box2;        // Declare Box2 of type Box
```

Both of the objects Box1 and Box2 will, of course, have their own data members. This is illustrated here:



The object name Box1 embodies the whole object, including its three data members. They are not initialized to anything, however - the data members of each object will simply contain junk values, so we need to look at how we can access them for the purpose of setting them to some specific values.

Accessing the Data Members of a Class

The data members of objects of a class can be referred to using the direct member access operator (.). So, to set the value of the data member height of the object Box2 to, say, 18.0, we could write this assignment statement: Box2.height = 18.0; // Setting the value of a data member We can only access the data member in this way, in a function outside the class, because the member height was specified as having public access. If it wasn't defined as public, this statement would not compile.

```
// Creating and using boxes
#include <iostream>
using namespace std;

  class Box // Class definition at global scope
  {
  public:
  double length;  // Length of a box in inches
  double breadth; // Breadth of a box in inches
  double height;  // Height of a box in inches
  };
  int main(void)
  {
  Box Box1;   // Declare Box1 of type Box
  Box Box2;   // Declare Box2 of type Box
  double volume = 0.0;  // Store the volume of a box here
  Box1.height = 18.0; // Define the values
  Box1.length = 78.0; // of the members of
  Box1.breadth = 24.0;  // the object Box1
  Box2.height = Box1.height - 10; // Define Box2
  Box2.length = Box1.length/2.0;  // members in
  Box2.breadth = 0.25*Box1.length; // terms of Box1
// Calculate volume of Box1

volume = Box1.height * Box1.length * Box1.breadth; cout << endl

<<  "Volume of Box1 = " << volume; cout << endl
<<  "Box2 has sides which total "
<<  Box2.height + Box2.length + Box2.breadth
<<  " inches.";
cout << endl  // Display the size of a box in memory
<<  "A Box object occupies "

<<  sizeof Box1 << " bytes."; cout <<endl;

return 0;

}
```

How It Works

Everything here works as we would have expected from our experience with structures. The definition of the class appears outside of the function main() and, therefore, has global scope. This enables objects to be declared in any function in the program and causes the class to show up in the ClassView once the program has been compiled.

Encapsulation

Encapsulation in C++ is defined as the wrapping up of data and information in a single unit. In Object Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulate them. Consider a real-life example of encapsulation, in a company, there are different sections like the accounts section, finance section, sales section, etc. Now,

- The finance section handles all the financial transactions and keeps records of all the data related to finance.
- Similarly, the sales section handles all the sales-related activities and keeps records of all the sales.

Exercise 01

Write a C++ program to find the sum of two numbers entered by the user.
Answer:

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

// Defining Class
class Calculator{
      public:
            long double num1,num2;
      Calculator(){
            cout<<"\n\tEnter First Number: ";cin>>num1;
            cout<<"\n\tEnter First Number: ";cin>>num2;
      }
      long double sum(){
            return(num1+num2);
      }
};
```

```
// Main Function
int main(){
cout<<"\n\n\t\tObject Oriented Programming\n\t\tAssignment# 01: Write a C++
program to find the sum of two numbers entered by the user.\n";
// Creating calculator object
Calculator my_cal;
cout<<"\n\tThe sum of these two numbers is: "<<my_cal.sum();
return 0;
}
```

Exercise 02

Write a C++ program that calculates the factorial of a given positive integer.

Answer:

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

// Defining Class
class Factorial{
      public:
      unsigned int x;
      Factorial(){
            cout<<"\n\tEnter your number: ";cin>>x;
      }
      long double fact(){
            int y=1;
            for(int i=x;i>0;i--){
                  y=y*i;
            }
            return y;
      }
};
int main(){
cout<<"\n\n\t\tObject Oriented Programming\n\t\tAssignment# 01: Write a C++
program that calculates the factorial of a given positive integer.\n";
// Creating Factorial Object
Factorial cal_fact;
```

```
cout<<"\n\tThe factorial of number is: "<<cal_fact.fact();

return 0;
}
```

Exercise 03

Write a C++ program to check if a number is even or odd.

Answer:

```
#include<iostream>
using namespace std;
// Defining Class
class Even_Odd{
	public:
	int x;
	Even_Odd(){
		cout<<"\n\tEnter your number: ";cin>>x;
	}
	string check(){
		string y;
		if(x%2==0){
			y="The given Number is Even.";
		}
		else{
			y="The given Number is Odd.";
		}
		return y;
	}
};
int main(){
cout<<"\n\n\t\tObject Oriented Programming\n\t\tAssignment# 01: Write a C++ program to check if a number is even or odd.\n";
// Creating Factorial Object
Even_Odd even_odd;
cout<<"\n\t"<<even_odd.check();
return 0;
}
```

Exercise 04

Write a C++ program to find the largest among three numbers entered by the user.

Answer:

```
#include<iostream>
using namespace std;
// Defining Class
class Sort{
      public:
      long double x,y,z,arr[3];
      Sort(){
            for(int i=0;i<3;i++){
                  cout<<"\n\tEnter your "<< i <<" number: ";cin>>arr[i];
            }
      }
      long double check(){
            for(int i=0;i<3;i++){
                  for(int j=i+1;j<3;j++){
                        if(arr[i]>arr[j]){
                              int temp=arr[i];
                              arr[i]=arr[j];
                              arr[j]=temp;}
                  }
            }
            return(arr[0],arr[2]);
      }
      long double large(){
            int a,b;
            a,b=check();
            return(b);
      }
};
int main(){
// Creating Factorial Object
Sort sort;
cout<<"\n\tLargest Number entered is : "<<sort.large();
return 0;
}
```

Exercise 05

Write a C++ program to reverse a string entered by the user.

Answer:

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

// Defining Class
class Reverse{
	public:
	string x;
	Reverse(){
		cout<<"\n\tEnter your String ";cin>>x;
	}

	void reverse(){
		for(int i=x.length();i>=0;i--){
			cout<<x[i];
		}
	}
};
int main(){
cout<<"\n\n\t\tObject Oriented Programming\n\t\tAssignment# 01: Write a C++ program to reverse a string entered by the user.";
// Creating Factorial Object
Reverse rev;
cout<<"\n\tReversed String is: ";rev.reverse();

return 0;
}
```

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.
Consider a ***real-life example of a man driving a car***. The man only knows that pressing the accelerator will increase the speed of the car or applying brakes will stop the car but he does not know how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of the accelerator, brakes, etc in the car. This is what abstraction is.

Abstraction using classes

We can implement Abstraction in C++ using classes. The class helps us to group data members and member functions using available access specifiers. A Class can decide which data member will be visible to the outside world and which is not.

Exercise 06

Write a C++ program that converts temperature from Fahrenheit to Celsius.
Answer:

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

// Defining Class
class Convertor{
      public:
      long double x;
      Convertor(){
            cout<<"\n\tEnter Temperature: ";cin>>x;
```

```
	}

	long double fah_to_cel(){
		long double temp=(x-32)*(5.0/9.0);
		return temp;
	}
};
int main(){
cout<<"\n\n\t\tObject Oriented Programming\n\t\tAssignment# 01: Write a C++ program that converts temperature from Fahrenheit to Celsius.";
// Creating Factorial Object
Convertor cnv;
cout<<"\n\tTemperature in Celcius is: "<<cnv.fah_to_cel();

return 0;
}
```

Exercise 07

Write a C++ program to implement a simple calculator with addition, subtraction, multiplication,and division operations.

Answer:

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

// Defining Class
class Calculator{
	public:
	long double x,y;
	Calculator(){
		cout<<"\n\tEnter First value: ";cin>>x;
		cout<<"\n\tEnter Second value: ";cin>>y;
	}

	long double sum(){
		return x+y;
```

```
	}
	long double sub(){
		return x-y;
	}
	long double mul(){
		return x*y;
	}
	long double div(){
		return x/y;
	}
};
int main(){
cout<<"\n\n\t\tObject Oriented Programming\n\t\tAssignment# 01: Write a C++ program to implement a simple calculator with addition, subtraction, multiplication,and division operations.";
// Creating Factorial Object
Calculator cal;
cout<<"\n\tEnter Operation (+,-,*,/) : ";char op;cin>>op;
switch(op) {
	case '+':
		cout<<"\n\tSum of these numbers is: "<<cal.sum();
		break;
	case '-':
		cout<<"\n\tSum of these numbers is: "<<cal.sub();
		break;
	case '*':
		cout<<"\n\tSum of these numbers is: "<<cal.mul();
		break;
	case '/':
		cout<<"\n\tSum of these numbers is: "<<cal.div();
		break;
	default:
		cout<<"\n\tERROR";
}


return 0;
}
```

Exercise 08

Write a C++ program to generate a Fibonacci sequence of a given length.

Answer:

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

// Defining Class
class Fibonacci{
	public:
	int x;
	Fibonacci(){
		cout<<"\n\tEnter length of series: ";cin>>x;
	}
	void generate(){
		long double n0=0,n1=1;
		cout<<n0<<" ";
		for(int i=0;i<x-1;i++){
			long double n2=n0+n1;
			cout<<n2<<" ";
			n0=n1;n1=n2;
		}
	}
};
int main(){
cout<<"\n\n\t\tObject Oriented Programming\n\t\tAssignment# 01: Write a C++ program to generate a Fibonacci sequence of a given length.";
// Creating Factorial Object
Fibonacci fib;
cout<<"\n\t";fib.generate();


return 0;
}
```

Exercise 09

Write a C++ program to find the prime factors of a number.

Answer:

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

// Defining Class
class Prime{
	public:
	int x;
	Prime(){
		cout<<"\n\tEnter your number: ";cin>>x;
	}
	void factors(){
		for(int i=x;i>0;i--){
			if(x%i==0){
				cout<<i<<", ";
			}
		}
	}
};
int main(){
cout<<"\n\n\t\tObject Oriented Programming\n\t\tAssignment# 01: Write a C++ program to find the prime factors of a number.";
// Creating Factorial Object
Prime prime;
cout<<"\n\tPrime factors of the given numbers are: ";prime.factors();


return 0;
}
```

Exercise 10

Write a C++ program to determine whether a given year is a leap year or not.

Answer:

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

// Defining Class
class Year{
	public:
	int x;
	Year(){
		cout<<"\n\tEnter Year: ";cin>>x;
	}
	bool is_leap(){
		bool y;
		if(x%4==0){
			y=true;
		}
		else{
			y=false;
		}
		return y;
	}
};
int main(){
cout<<"\n\n\t\tObject Oriented Programming\n\t\tAssignment# 01: Write a C++ program to determine whether a given year is a leap year or not.";
// Creating Factorial Object
Year year;
if(year.is_leap()==true)
cout<<"\n\tGiven Year is Leap Year.";
else
cout<<"\n\tGiven Year is not Leap Year.";


return 0;
}
```

Exercise 11

Define a class to represent a bank account. Include the following members:

Data members:

1) Name of the depositor

2) Account number

3) Type of account

4) Balance amount in the account.

Member functions:

1) To assign initial values

2) To deposit an amount

3) To withdraw an amount after checking the balance

4) To display name and balance.

Write a main program to test the program.

Answer:

In this program I've defined a simple class with some data members and their functions the code looks like this

```
#include<iostream>
using namespace std;
class Bank_account{
     private:
          string acc_holder_name,acc_num,acc_type;
          int acc_pin;
          long double balance_amount;
     public:
          void create_acc(){
               cout<<"\t\tEnter Account Holder Name:
";cin>>acc_holder_name;
               cout<<"\n\t\tEnter Account Number:
";cin>>acc_num;
               cout<<"\n\t\tEnter Account Type:
";cin>>acc_type;
```

```
                cout<<"\n\t\tSet Account Pin (format:XXXX):
";cin>>acc_pin;
                cout<<"\n\t\tEnter Account's Balance Amount:
";cin>>balance_amount;
          }
          void deposit(){
                long double n;
                int pin;
                cout<<"\n\t\tEnter Account Pin: ";cin>>pin;
                if(pin==acc_pin){
                     cout<<"\n\t\tEnter Amount to Deposit:
";cin>>n;
                     balance_amount+=n;
                     cout<<"\n\t\tThank you for using our bank.
";
                }
                else{
                     cout<<"\n\t\tEntered Wrong pin, Please
contact the department. ";
                }
          }
          void withdraw(){
                long double n;
                int pin;
                cout<<"\n\t\tEnter Account Pin: ";cin>>pin;
                if(pin==acc_pin){
                     cout<<"\n\t\tEnter Amount to Withdraw:
";cin>>n;
                     if(n>balance_amount){
                          cout<<"\n\t\tSorry, Insufficient
Balance.";
                     }
                     else{
                          balance_amount-=n;
                          cout<<"\n\t\tThanks for using our
```

```
bank. ";
                         cout<<"\n\t\tBalance Available:
"<<balance_amount;
                    }
               }
               else{
                    cout<<"\n\t\tEntered Wrong pin, Please
contact the department. ";
               }
          }
          void check(){
               int pin;
               cout<<"\n\t\tEnter Account Pin: ";cin>>pin;
               if(pin==acc_pin){
                    cout<<"\n\t\tAccount Holder Name:
"<<acc_holder_name;
                    cout<<"\n\t\tAccount Number: "<<acc_num;
                    cout<<"\n\t\tAccount Type: "<<acc_type;
                    cout<<"\n\t\tAccount Balance:
"<<balance_amount;
               }
               else{
                    cout<<"\n\t\tEntered Wrong pin, Please
contact the department. ";
               }
          }
};
int main(){
     system("color F0");
     cout<<"\n\n\t\t";cout<<" Object Oriented Programming ";
     cout<<"\n\n\t\t";cout<<" Assignment 02 ";
     cout<<"\n\t\t";cout<<" Presented by: Asad\n\n";
     // creating object
     Bank_account acc;
     // creating account
```

```
     acc.create_acc();
     // depositing amount
     acc.deposit();
//   // withdrawing amount
     acc.withdraw();
//   // checking details
     acc.check();
return 0;
}
```

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.

Inheritance is a feature or a process in which new classes are created from the existing classes. The new class created is called “derived class” or “child class” and the existing class is known as the “base class” or “parent class”. The derived class now is said to be inherited from the base class.

Exercise 12

Exhibit inheritance by creating a Manager and Engineer class from Employee class with appropriate data members and functions.

```
#include<iostream>
using namespace std;
class Employee{
     public:
          string empid;int salary;
     Employee(string empid,int salary){
          empid=empid;salary=salary;
     }
};
class Manager: public Employee{
     public:
          string post;
     Manager(string empid,int salary,string post):
Employee(empid, salary){
          post=post;
     }
};
class Engineer: public Employee{
     public:
```

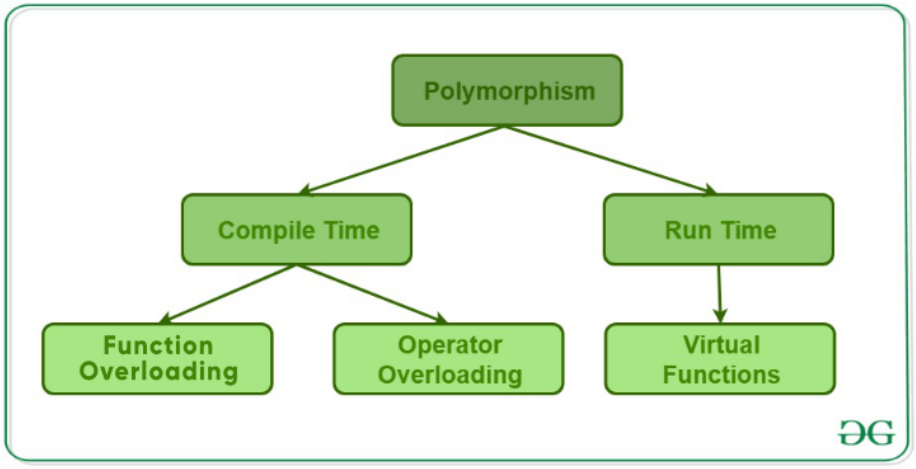
```
          string site;
     Engineer(string empid,int salary,string site):
Employee(empid, salary){
          site=site;
     }
};

int main(){

Employee emp("Ali",55000);

Manager mng("Asghar",280000,"Executive");

Engineer eng("Akram",25000,"Urban");

return 0;
}
```

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. A real-life example of polymorphism is a person who at the same time can have different characteristics. A man at the same time is a father, a husband, and an employee. So the same person exhibits different behavior in different situations. This is called polymorphism. Polymorphism is considered one of the important features of Object-Oriented Programming.

Types of Polymorphism

- Compile-time Polymorphism
- Runtime Polymorphism



Function Overloading: When there are multiple functions with the same name but different parameters, then the functions are said to be **overloaded,** hence this is known as Function Overloading. Functions can be overloaded by **changing the number of arguments** or/and **changing the type of arguments**. In simple terms, it is a feature of object-oriented programming providing many functions that have the same name but distinct parameters when numerous tasks are listed under one function name. There are certain Rules of Function Overloading that should be followed while overloading a function.

Example

```
#include <iostream>
using namespace std;
class Temp {
public:
      void func(int x)
      {
            cout << "value of x is " << x << endl;
      }
      void func(double x)
      {
            cout << "value of x is " << x << endl;
      }
      void func(int x, int y)
      {
            cout << "value of x and y is " << x << ", " << y
                  << endl;
      }
};

int main()
{
      Temp obj1;

      // func() is called with int value
      obj1.func(7);

      // func() is called with double value
      obj1.func(9.132);

      // func() is called with 2 int values
      obj1.func(85, 64);
      return 0;
}
```

Exercise 13

Create two classes inherited from a class to show function overloading.

```
#include<iostream>
using namespace std;
// Base Class
class Shape{
      public:
      string name;
      void draw(){
            cout<<"The shape is: Unknown";
      }
};
// Derived Class
class Circle: public Shape{
      public:
      float rad,cir;
      void draw(){
            cout<<"The shape is: Circle";
      }
      void details(){
            cout<<endl<<"Radius of circle: "<<rad<<endl<<"Circumference of
circle: "<<cir;
      }
      float area(int r){
            float a;
            a=(3.14159265)*(r*r);
            return a;
      }
};

// Derived Class
class Triangle: public Shape{
      public:
      float hei,wid,hyp;
      void details(){
            cout<<"Height of Triangle: "<<hei<<endl<<"Width of Triangle:
"<<wid<<endl<<"Hyp of Triangle: "<<hyp;
      }
      float area(float x, float y){
            float a;
            a=x*y;
            return a;
      }
```

```
};
// Main function
int main(){
// creating object
Circle c;
c.name="my circle";
c.rad=2.0;
c.cir=5.5;
cout<<"Area of circle is: "<<c.area(2.2);

// creating object
Triangle t;
t.hei=12;
t.wid=10;
t.hyp=13;
cout<<"\nArea of triangle is: "<<t.area(12.2,10.5);
return 0;
}
```