Classes

Contents

- Introduced class definitions and objects
- Public versus private access into class.
- Syntax for member functions
- Syntax data members
- Get and Set functions
- Constructors & Destructors
- Placing classes in separate files
- Separating interface from implementation

Limitations of Procedural Programming using structures

- If the data structures change, many functions must also be changed

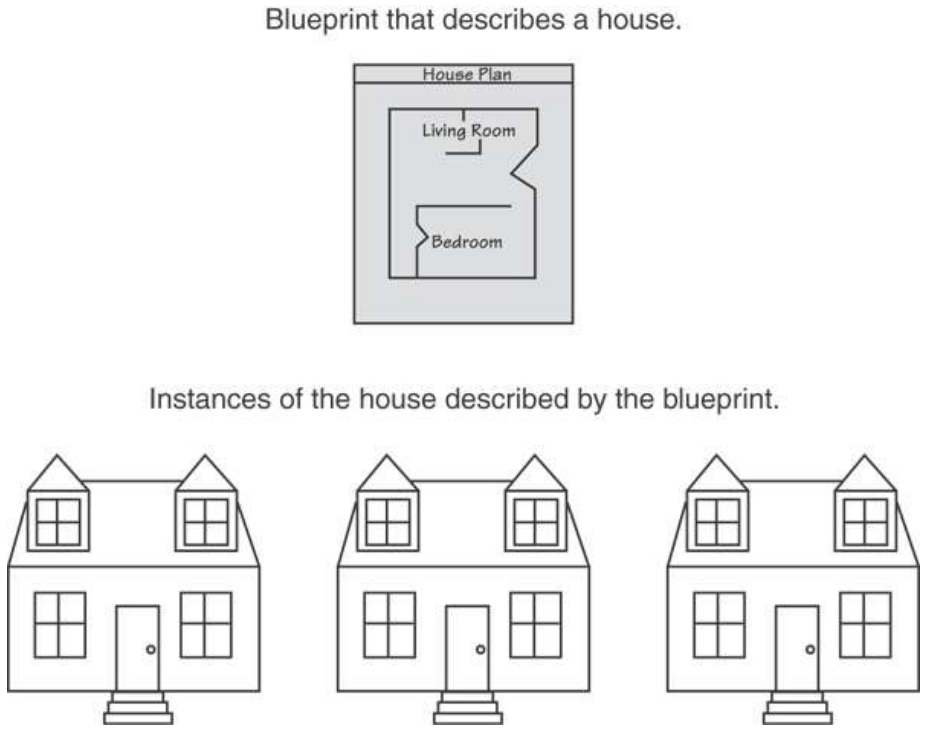
- Programs that are based on complex function hierarchies are:
  - difficult to understand and maintain
  - difficult to modify and extend
  - easy to break

Key Points

- **Data encapsulation**
- Encapsulation is a programming mechanism that binds together code and the data it manipulates, and that keeps both safe from outside interference.
- **Classes as Object-Oriented programming**
- It is based on the data and the functions that operate on it. In an object-oriented language, code and data may be bound together in such a way that a self-contained black box is created. Within the box are all necessary data and code. When code and data are linked together in this fashion, an object is created.
- In other words, an object is the device that supports encapsulation. Objects of classes are similar to *structs* but contain functions, as well.
- Variables and functions in the class can have different properties than in a `struct`

Classes and Objects

- A Class is like a blueprint and objects are like houses built from the blueprint



Declration a Class

- Format:

```
class ClassName
{  private:
   declaration of private variables and functions;
   public:
   declaration of public variables and functions;
                 };
```

Objects are created from a `class`

- C++ Objects
- When class is defined, only the specification for the object is defined; no memory or storage is allocated.

Same as struct

- To use the data and access functions defined in the class, you need to create objects.

Class Example

```
// This creates the class Rectangle
class Rectangle { private:
                    float width;
                    float length;
                  public:
                     void setwidth( float);
                     void setlength( float);
                     float getwidth( );
                     float getlength( );
                     float getarea();
                                    };
```

Private Members

Public Members

Access members of a class

Everything before public is private

- Can be listed in any order in a class
- Can appear multiple times in a class
- If not specified, the default is **`private`**
- **As a rule, data members should be declared private**
- **Member functions should be declared public, except member functions that are accessed only by other member functions of the class.**

Private Members

- Making data members `private` provides data protection, they are visible only by class.
- Data can be accessed only through `public` functions
- Public functions define the class's public interface
- **`public`**: visible everywhere

Defining a Member Function of class

- A member function can be defined by either of two methods:

- 1- Can be defined within class declaration 1. within class

```
Class Rectangle{private:…….
                public:
                void setWidth(double w)   {width = w;    }
```

2- Put prototype in class declaration 2. in the gloabal

```
class Rectangle{     public:   void setWidth(double w);……};
```

Define function using class name and scope resolution operator (: :)

```
void Rectangle::setWidth(double w)
  {
       width = w;
  }
```

fun tyep + class name : : fun name (){}

int student::grades(){}

Defining an Instance of a Class

- An object is an instance of a class
- Defined like structure variables:

```
Rectangle r;
```

- Access members using dot (.) operator:
- For the class Rectangle before,

```
r.setWidth(5.2);
cout << r.getWidth();
cout<<r.width;    //error
```

- There will be a compiler error if you attempt to access a **`private`** member using dot operator

Differences between classes and structures

- By default, the members of a class are private, while the members of a structure are public.
- For most programmers will use a class to define the form of an object that contains member functions, and a structure to create objects that contain only data members.

Example 1:

```
class test
{    private:
       int data1;       float data2;

   public:
      int b;
      void function1()
      {   data1 = 2;
            b=10;  cout<<" b  in function1   "<<b<<endl; }

            float function2()
      {           data2 = 3.5;
          function1();
           cout<<"  data1 after calling function1   "<<data1<<endl;
              b=20; cout<<" b in function2   "<<b<<endl;
          return data2;       }     };
```

Here, data1 and data2 are private members, where as variable b, function1() and function2() are public members. Member functions are declared within the class.

```
int main()
{    test t1, t2;  }
```

Here, two objects t1 and t2 of the class test are created.
You can access the data members and member functions by using a . (dot) perator.

For example,
t2.function1();
This will call the function1() function inside the test class for objects t2.

Similarly, the data member can be accessed as:

t1.b = 5.5;
t1.data1=10 //error in the main
It is important to note that, the private members can be accessed only from inside the class.

So, you can use t2.function1(); from any function or class in the above example.

Main program that uses the above class:

```
main( )
    1- {   test t1;   // create object t1 of class test
    2-   //data1=10;    //error data1 is not declared in main
    3-   // t1.data1=10;    //error data1 is private data
    4-    float h=t1.function2( ); // calling function2() as member of object t1
    5-   cout<<"  data2 after calling function2   "<<h<<endl;
    6-  cout<<"  b after calling function2  "<<t1.b<<endl;
    7-   // b=80;              //error b is not declared in main
    8-   t1.b=80;                //  calling b as member of t1
    9-   cout<<"  b in main   "<<t1.b;          }
```

Output

```
b  in function1   10
 data1 after calling function1   2
b in function2   20
 data2 after calling function2   3.5
 b after calling function2  20
 b in main   80
```

Member functions of class test can also be written as follows :

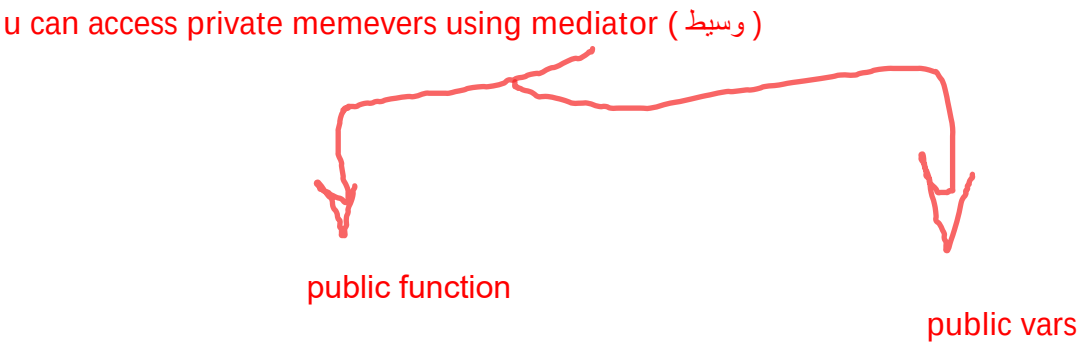
```
class test
{    private:
      int data1;
      float data2;
            public:
             int b;
            void function1();
            float function2();};
```

```
void test::function1()
 {   data1 = 2;
        b=10; cout<<" b  in function1   "<<b<<endl; }

float test::function2()
 {          data2 = 3.5;
     function1();
      cout<<"  data1 after calling function1   "<<data1<<endl;
        b=20; cout<<" b in function2   "<<b<<endl;
     return data2;       }
```

Gloabl

Example:2

```
// This creates the class test2 that contains integer array and uses functions to initialize
private data, insert and get values of an array of integer +ve numbers.
class test2 { int q[100];
        int sloc, rloc, size;
        public:
        void init();
        void put(int i);
        int get();  };
// Initialize the class test2.
        void test2::init()
        {rloc = -1;  sloc = 0; size=100; }

// Put an integer into the object.
        void test2::put(int i)
        { if(sloc< size) {  q[sloc] = i; sloc++; }
        else {cout << " object is full.\n";  return; }  }
```



Example 2: cont.

```
// Get an integer from the object.
        int test2:: get()
        { if(rloc == sloc) { cout << “object underflow.\n"; return -1; }
        rloc++;  return q[rloc];}

int main()
1.      test2 a, b; // create two test objects
2.      a.init();  b.init();
3.      a.put(10);  a.put(19);      a.put(20);  a.put(-1);
4.      b.put(1);  b.put(2);        b.put(3);  b.put(4);
5.      cout << "Contents of object a: ";
6.      cout << a.get() << " "<<a.get( )<<"    " << a.get() <<"    "<<a.get()<< "\n";
7.      cout << "Contents of object b: ";
8.      cout << b.get() <<"  "<<b.get()<<"   " << b.get() <<"   "<<b.get( )<< "\n";  return 0; }
```

Output:

Contents of object a: 10 19 20 -1

Contents of object b: 1 2   3   4

Comments on example 2

- a and b are two objects of class test2
- data1 and data2 are two private members of the class and used only by class members, i.e. a.data1, b.data2 are error statements

  init(); put(int i); get(); are three public member functions of class
- The three functions are specified after the declaration of the class
- Calling any function must be related to its object using dot operator, i.e. a. init( )
- In the main program we create two objects of the class, use function put( ) to construct an array from the entered data and get( ) function to get data from the array.

Pointers to Objects of class

same as struct

- We can define a pointer to an object of class:

```
Rectangle *rPtr;
```
pointer decleration

```
Rectangle otherR;
```
member decleration

- Can access public members via pointer:

```
rPtr = &otherR;
rPtr->setLength(12.5);//otherR.setLength(12.5);
cout << rPtr->getLength() << endl;
```

- To access member of class through its pointer, use arrow -> operator

-> have class name
-> NO return types
-> always public

Constructors and Destructors

Constructor

A constructor is a special type of member function that initializes an object automatically when it is created.

Compiler identifies a given member function is a constructor by its name and the return type.

Constructor has the same name as that of the class and it does not have any return type. Also, the constructor is always public.

Example:

```
class temp
{  private:
            int x;
            float y;
public:
            // Constructor
            temp( )      {// Body of constructor
             x=5;   y=5.5;
                                }
            … .. …   };
int main( )
{           temp t1;
            … .. …   }
```

The above program shows a constructor that is defined to initialize the values of x and y. It has no return type , it has the same name as the class.

How constructor works?

In the above code, temp() is a constructor.

When an object of class temp is created, the constructor is called automatically, and x is initialized to 5 and y is initialized to 5.5.

Use of Constructor in C++

Suppose you are working on 100's of Person objects and the default value of a data member age is 0. Initializing all objects manually will be a very tedious task.

Instead, you can define a constructor that initializes age to 0. Then, all you have to do is create a Person object and the constructor will automatically initialize the age.

These situations arise frequently while handling array of objects.

Also, if you want to execute some code immediately after an object is created, you can place the code inside the body of the constructor.

Example 3: Constructor in C++
Calculate the area of a rectangle and display it.

```
class Area
{    private:
     int length,      int width;
   public:
     // Constructor
     Area(){ length=5; width=2;  }
     void Get()
     {         cout << "Enter length and width respectively: ";
       cin >> length >> width;          }
     int AreaCalculation() {  return (length * width);  }
     void DisplayArea( )
     {          cout << "Area: " << AreaCalculation() ;        }  };
```

```
int main()
{
   Area A1, A2;
   int temp;
   A1.Get();
   temp = A1.AreaCalculation();

   A1.DisplayArea( );
   cout << endl << "Default Area
 when value is not taken from user" << endl;

temp = A2.AreaCalculation();

A2.DisplayArea();        return 0;    }
```

- In this program, class Area is created to handle area related functionalities. It has two data members length and width.
- A constructor is defined which initializes length to 5 and width to 2.

- We also have three additional member functions GetLength(), AreaCalculation() and DisplayArea() to get length from the user, calculate the area and display the area respectively.

When, objects A1 and A2 are created, the length and width of both objects are initialized to 5 and 2 respectively, because of the constructor.

Then, the member function GetLength() is invoked which takes the value of length and width from the user for object A1. This changes the length and width of the object A1.

Then, the area for the object A1 is calculated and stored in variable temp by calling AreaCalculation() function and finally, it is displayed.

For object A2, no data is asked from the user. So, the length and width remains 5 and 2 respectively.

Then, the area for A2 is calculated and displayed which is 10.

Output

Enter length and breadth respectively:
6
7
Area: 42
Default Area when value is not taken from user
Area: 10

Default Copy Constructor

An object can be initialized with another object of same type. This is same as copying the contents of a class to another class.

In the above program, if you want to initialize an object A3 so that it contains same values as A2, this can be performed as:

```
int main()
{    Area A1, A2;

 // Copies the content of A2 to A3
 Area A3(A2);
  OR,
 Area A3 = A2;   }
```

A2 = A3 true
A2 == A3 false

You might think, you need to create a new constructor to perform this task. But, no additional constructor is needed. This is because the copy constructor is already built into all classes by default.

Example 4

```
int main( )
1.   {     Area A1, A2, A3 ;
2.        int temp;
3.        A1.GetLength();
4.        temp = A1.AreaCalculation();
5.        A1.DisplayArea();
6.        cout << endl << "Default Area when value is not taken from user" << endl;
7.        temp = A2.AreaCalculation();
8.        A2.DisplayArea();
9.            A3 = A2;  // copy object to another object
10.       temp = A3.AreaCalculation();
11.          cout << endl << "Default Area when value is not taken from user" << endl;
12.          A3.DisplayArea();
13.           Area A4=A1;
14.          temp = A4.AreaCalculation();
15.                        A4.DisplayArea();      return 0;    }
```

Output

Enter length and width respectively:
7
8
Area: 56 // area of object A1

Default Area when value is not taken from user
Area: 10 // area of object A2

Default Area when value is not taken from user
Area: 10 // area of object A3
Area: 56 // area of object A4

Parameterized Constructors

A constructor can have parameters. This allows you to give member variables program defined initial values when an object is created.

You do this by passing arguments to an object's constructor. The next example will enhance the class to accept an argument.

```
class sample {  int a;
public:
sample(int n) { a = n; }
int get_a() { return a; } };
int main()
{ sample samp_1( -1 );
sample samp_2  ( 10 );
 cout << samp_1.get_a() <<“       “;
 cout << samp_2.get_a() <<endl;    return 0;  }
```

Output: -1 10

Destructor

- A destructor is a special member function that is called when the lifetime of an object ends. The purpose of the destructor is to free the resources that the object may have acquired during its lifetime.
- Local objects are created when their block is entered, and destroyed when the block is left.
- Global objects are destroyed when the program terminates. There are many reasons why a destructor may be needed. For example, an object may need to deallocate memory that it had previously allocated. In C++, it is the destructor that handles deactivation.
- The destructor has the same name as the constructor, but the destructor's name is
- preceded by a ~. Like constructors, destructors do not have return types.

Syntax:

- ~ class_name ();

The destructor may also be called directly, e.g. to destroy an object that was constructed.

If no user-declared destructor is provided for a class type, the compiler will always declare a destructor as an inline public member of its class.

```
class Area
{    private:
      int length;
      int width;
   public:
     // Constructor
     Area(){ length=5; width=2;  }
   // destructor
   ~ Area(  ){  cout<< "  object of class Area is destroyed  " ;}
```

```
int main()
{     Area A1, A2 ;
   int temp;
   A1.GetLength();
   temp = A1.AreaCalculation();
   A1.DisplayArea();
   cout << endl << "Default Area when value is not taken from user" << endl;
   temp = A2.AreaCalculation();
   A2.DisplayArea();              return 0;    }
```

When the above program uses the class Area that has been declared before the output will be

```
Enter length and width respectively:
7
 8
Area: 56
Default Area when value is not taken from user
Area: 10
  object of class Area is destroyed
  object of class Area is destroyed
```

Example 5: Passing an object of class to function

Program that use class to insert values in certain array, print them and add two arrays

```
1.  class Vector {      int  a[100], size;
2.  public:                 Vector ( );
3.          void insert (int index, int val);
4.          void print ( );
5.          void add (Vector b); // argument of the function is object of class   };
6.           Vector ::Vector( ) {cout<<"  create new data  "<<endl; size=0;}    // constructor
7.          void Vector ::insert(int index, int val) {   a[index] = val; size++;}
8.          void Vector :: print ( ){  cout<<"  values  of the vector  "<<endl;
9.          for(int i=0; i<size ; i++)cout<<a[i]<<"  "; cout<<endl; }
10.     void  Vector :: add (Vector b) { for(int i=0; i< size ; i++) a[i]+= b. a[i];}
11.         main( )
12.         { Vector L1, L2;    L1. insert(0,10); L1.insert(1,12);  L1.insert(2,40);  L1.insert(3,20);
13.    L2.insert (0,-5); L2.insert(1,30);    L2.insert(2,50); L2. insert(3, 15);
14.     L1.print( );  L2.print( );   L1. add(L2);  // add array of object L2 to the array of object L1
15.         cout<<“  vector  L1 after addition “<<endl<<L1. print ( ) ;}
```

Output

create new data
create new data
values of the vector
10 12 40 20
values of the vector
-5 30 50 15
vector L1 after addition
values of the vector
5 42 90 35

Notes:

1. Pass classes by reference if they need to be modified ***// void add(Vector &b);***

2. What will be the modification in the program, if we want to add the two arrays in another new array

Example of using pointer with classes

- Write the class distance that has the dimension of any distance as private data.
- The class contains the following functions:
- read_dist( ) : to input the dimension of any distance
- Print_dist( ) : to print the dimension of any distance
- Add( ) : to add any two distances
- In the main program, create two objects of the class, read the two distances and add them to get a new distance, you must send the two distances to the function add( ) as pointers and print the new distance.

Example 6 //use pointers with classes

```
1.  class dist_ex {private: int km,m,cm;
2.              public:
3.                      //costructor
4.                      dist_ex(){  km=0;m=0; cm=0;              }
5.                          void read_dist( );
6.                      void print_dist( );
7.                      void  add(dist_ex *a,dist_ex *b);
8.          // destructor          ~ dist_ex( ); };
9.     void dist_ex::read_dist( )
10.    {cout<<"  enter distance in cm, m and km  "<<endl;
11.          cin>>cm>>m>>km; }
12.
13. void dist_ex::print_dist()
14.          { cout<< "  dimension of distance   "<<endl;
15. cout<<" km = "<<km<<" m=  "<<m<<"   cm= "<<cm<<endl;}
16.
17. void dist_ex::add(dist_ex *a, dist_ex *b)
18.  { cm=a->cm+b->cm;    m=a->m+b->m;
19.  if(cm>=100)   {m++;  cm-=100;}
20.    km=a->km+b->km; if(m>=1000){ km++; m-=1000;  }       }
```

```
int main( )
{ dist_ex  d1, d2,d3;
   d1.read_dist(); d2.read_dist();
   d1.print_dist(); d2.print_dist();
   d3.add(&d1,&d2);
   d3.print_dist();  }
```

Output for the given input

```
 enter distance in cm, m and km
60 600 9
  enter distance in cm, m and km
77 800 5
  dimension of distance
 km = 9 m=  600   cm= 60
  dimension of distance
 km = 5 m=  800   cm= 77
  dimension of distance
 km = 15 m=  401   cm= 37
```

Array of Objects

- Objects can be the elements of an array:
  class Rectangle{……
  ……};

```
Rectangle rooms[8];
```

- Default constructor for the array of objects is used when array is defined

Array of Objects

- It isn't necessary to call the same constructor for each object in an array:

Accessing Objects in an Array

Creating an array of objects of class Area

```
Area rArea[3];
```

- Objects in an array are referenced using subscripts

- Member functions are referenced using dot notation:

```
rArea[0].GetLength( );
```

- The complete program will be shown below.

Array of Objects

- We can use initializer list to invoke constructor that takes arguments:

```
class Area
{   private:
    int length;        int width;
  public:
    // Constructor
     Area (int n1=5, int n2=2){  length=n1; width=n2;  }
       Area rArea[3];
        rArea[1]={2,3};
```

Here, Array elements of object rArea[1] will take the initial given values, not the default values of constructor

Example 7 //array of objects of class Area

```
class Area  {    private:       int length;       int width;
                public:
     // Constructor
     Area (int n1=5, int n2=2){  length=n1; width=n2;  }
   void GetLength()
     {         cout << "Enter length and width respectively: ";
       cin >> length >> width;         }
     int AreaCalculation() {  return (length * width);  }
     void DisplayArea()
      {           cout << "Area: " << AreaCalculation( )<<endl;;        }  };
                 int main()
  {   // array of objects with initial values
1.          Area rArea[3]={Area(2,3),  Area(4, 9), Area(11, 31)};
2.           rArea[0].GetLength();
3.          int temp = rArea[0].AreaCalculation();
4.          rArea[0].DisplayArea( );
5.          cout << endl << "Default Area when values are not taken from user" << endl;
6.          rArea[1].DisplayArea( );
7.          rArea[2].DisplayArea( );          return 0;    }
```

Here user will enter length and width for the first object rArea[0] using Getlength( ) function

The other two objects of the array will take the initial given values
Statement 1 can be written: ***Area rArea[3]={(2,3), (4, 9), (11, 31)};***

Output

Enter length and width respectively: 10 20
Area: 200
Default Area when values are not taken from user
Area: 36
Area: 341

Allocating Class Instances using **new**

- **new** can also be used to allocate a class instance

```
class Point {
                public:
                int x, y;
        Point() {  x = 0; y = 0; cout << "default constructor" << endl; } };
 int main() {  //create an object of class Point through pointer
                Point *p = new Point;
        // delete can be used to deallocate the object
                delete p; }
```

Using new and delete with classes
Example: 8

```
// pointer to classes example
1.  class Rectangle {     int width, height;
2.  public:
3.   Rectangle(int x, int y ){ width =x;  height=y;}
4.   int area( ) { return width * height; }  };
5.  int main( ) {     Rectangle obj (3, 4);
6.    Rectangle * RP1, * RP2, * RP3;      RP1 = &obj;
7.    RP2 = new Rectangle (5, 6);
8.     RP3 = new Rectangle[2] { {2,5}, {3,6} };
9.   cout << "obj's area: " << obj.area() << '\n';
10.  cout << "*RP1's area: " << RP1->area() << '\n';
11.  cout << "*RP2's area: " << RP2->area() << '\n';
12.  cout << "RP3[0]'s area:" << RP3[0].area() << '\n'; //array is a pointer by its name
13.  cout << "RP3[1]'s area:" << RP3[1].area() << '\n';
14.  delete RP2;       delete[] RP3;      return 0;  }
```

Using new and delete with classes
Example: 8 (cont.)

Statements 3, 8,12, 13 can be written as follows:

// you must put initial values in the constructor function to have statement 8 correct

```
3. Rectangle(int x=3, int y=6 ){ width =x;  height=y;}
8. RP3 = new Rectangle[2] ;
```

We can send values to the constructor as follows:

```
8. RP3[0]= Rectangle (4,5);     RP3[1]=  Rectangle(10,20);
cout << "RP3[0]'s area:" << RP3[0].area() << '\n'; //array is a pointer by its name
  cout << "RP3[1]'s area:" << RP3[1].area() << '\n';
```

The output statements can be written as follows, but not preferred

```
 13. cout << "RP3[0]'s area:" << RP3->area() << endl;
 RP3++;
  14.  cout << "RP3[1]'s area:" << RP3->area() << endl;
```

Objects can also be pointed to by pointers: Once declared, a class becomes a valid type, so it can be used as the type pointed to by a pointer.

For example: Rectangle * ptr;
Ptr is a pointer to the class Rectangle
Similarly, as with objects, members are accessed using the arrow operator ->

RP1 is pointer to the object **obj**. the members of the object **obj** can be accessed through its name or its pointer RP1. RP2 and RP3 are pointers to the created objects using **new**.
The two created objects that have the pointers RP2 and RP3 are deleted using the **delete** keyword.

Output

obj's area: 12

*RP1's area: 12

*RP2's area: 30 // constructor has initial values 5, 6

RP3[0]'s area: 10 // constructor has initial values 2, 5

RP3[1]'s area: 18 // constructor has initial values 3, 6

Example_class using another class

- Create class first with data members book no, book name and member function getdata() and putdata(). Create a class second with data members author name, publisher and members getdata() and showdata().
- Derive a class third that includes the classes first and second with data members no. of pages and year of publication. Display all these information using array of objects of third class.

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

class first //this is the base class I
{ char name[20]; int bookno;
public:
void getdata()
{ cout<<"\nEnter the name of book : ";
cin>>name;
cout<<"\nEnter Book No. : ";
cin>>bookno;}
void putdata()
{cout<<"\nName of Book : "<<name;
cout<<"\nBookNo. :"<<bookno;} };
```

```
class second //this is the base class II
{char author[20],publisher[20];

public:
void getdata()
{cout<<"\nEnter Author's Name : ";
cin>>author;
cout<<"\nEnterPublisher : ";
cin>>publisher; }

void showdata()
{cout<<"\nAuthor'sName : "<<author;
cout<<"\nPublisher : "<<publisher;} };
```

```
class  third
public first, public second //this is the derived class
{ int no_pages, year;
   public:
void get()
{ first::getdata();
   second::getdata();
cout<<"\nEnter No. of Pages : ";
cin>>no_pages;
cout<<"\nEnter the year of publication : ";
cin>>year; }
void display()
{ putdata(); showdata();
cout<<"\nNo. of Pages : "<<no_pages;
cout<<"\nYear of Publication : "<<year; }};
```

```
int main()
{ third book[5];
int num;
cout<<"\nEnter the number of books : ";
cin>>num;
for(int i=0;i<num;i++)
{book[i].get();
cout<<endl;}
for(int i=0;i<num;i++)
{book[i].display();
cout<<endl;} }
```

Another way to use class inside another class

```
class third  {
class first //this is the base class I
{ char name[20];   int bookno;
public:   void getdata() { cout<<"\nEnter the name of book : ";
cin>>name;cout<<"\nEnter Book No. : ";cin>>bookno;}
void putdata(){cout<<"\nName of Book : "<<name;
cout<<"\nBookNo. :"<<bookno;} };  //end of class first

class second //this is the base class II
{char author[20],publisher[20];
public:void getdata(){cout<<"\nEnter Author's Name:";
cin>>author;cout<<"\nEnterPublisher : "; cin>>publisher; }
void showdata()  {cout<<"\nAuthor'sName : "<<author;cout<<"\nPublisher :"
<<publisher;} };   //end of class second
```

Another way to use class inside another class (cont.)

```
first A; second B;
int no_pages, year;
public:
void get(){ A.getdata();   B.getdata();cout<<"\nEnter No. of Pages :  “;
cin>>no_pages;   cout<<"\nEnter the year of publication : ";cin>>year;}
void display()   {  A. putdata(); B.showdata();
cout<<"\nNo. of Pages :” <<no_pages;   cout<<"\nYear of Publication : "<<year;}
};  //end of class third
```

Main is the same as previous example

Class in a Separate Header File for Reusability

Header files

Separate files in which class definitions are placed.

Allow compiler to recognize the classes when used elsewhere.

Generally have .h filename extensions

The advantages of storing class definition in separate file are

1. The class is reusable

2. The clients of the class know what member functions the class provides, how to call them and what return types to expect

3. The clients do not know how the class's member functions are implemented.

Implementation

- Main program can use class by including it as follows: ***# include " classname.h"***

.cpp files for source-code implemenations

Class implementations

Main programs

Example of using header files

We create a file called “good.h”

```
int add(int x, int y)
{
   return x + y;
}
```

We create another .cpp file that uses the header file “good.h”

```
# include <iostream>
#include "good.h" // Insert contents of add.h at this point.
 using namespace std;
main()
{int a=6; int b=10;
  cout<<"call function    "<< add(a,b); }
```

Another example

1. File Test2.h contains class declaration (header file)
2. File Test2.cpp consists of the class's public member functions
3. File any_name.cpp contains the header file #include "Test2.h" and all other statements of the main program
4. Class Declaration (Test2.h) & Implementation (Test2.cpp)

Or we can have a .h file that contains the declaration and implementation of class.

A class implementation usually consists of 2 files. First we'll look at the header file Test2.h

Example 9

// header file Test2.h

```
class test
{    private:
       int data1;
       float data2;
           public:
            int b;
            test( )  {data1=0; data2=0;}
            ~test( );
          void function1();
            float function2();};
```

Example 9 (cont.)

```
void test::function1()
      {   data1 = 2;
          b=10; cout<<" b  in function1   "<<b<<endl; }

    float test::function2()
     {          data2 = 3.5;
         function1();
         cout<<"  data1 after calling function1   "<<data1<<endl;
         b=20; cout<<" b in function2   "<<b<<endl;
        return data2;       }
```

// the main program will be in file called for example “class_test.cpp”
// main program must include header file of class

```
#include<iostream>
// class header file
# include “Test2.h”
main( )
      {   test t1;   // create object t1 of class test
      float h=t1.function2( );   // calling function2() as member of object t1
         cout<<"  data2 after calling function2   "<<h<<endl;
     cout<<"  b after calling function2  "<<t1.b<<endl;

       cout<<"  b in main   "<<t1.b;                    }
```

The code for the Test2 example is in two files:
• The header file, Test2.h, contains the class declaration.
• main (class_test.cpp) contains the code outside the class. Test2.h is included.

The two above files can be used by any program including the header file Test2.h in the same folder
i.e, if we create a main program has the file name “main2_test.cpp” , then include the header file in it as shown:

```
// main_test .cpp file
# include< iostream>
# include” Test2.h”
```

If the header file is in another drive, we must include it by its correct pass

Example of class Time

The following is another example of header, implementation and a main program that uses the class header file.

Class contains the data of certain time as hour, minute and second. There will be some member functions that dealing with time as setting certain time, print time and so on.

You can try it.

// Ex_10: test_Time.h, header file that includes declaration of class

```
class Time
{
    private :
        int hour;
        int minute;
        int second;
    public :
        //with default value
        Time(const int h = 0, const int m  = 0, const int s = 0);
        //    setter function
        void setTime(const int h, const int m, const int s);
        // Print a description of object in " hh:mm:ss"
        void print() const;
        //compare two time object
        bool equals(const Time&);           // call object by reference
};
```

The member function definitions for a class are stored in a separate .cpp file, which is called the class implementation file. The file usually has the same name as the class, with the .cpp extension. For example the Time class member functions would be defined in the file test_Time.cpp.

// Implementation of class in the file test_Time.cpp

```
#include "test_Time.h"


Time :: Time(const int h, const int m, const int s)
  { hour=h; minute=m; second =s;}
 void Time :: setTime(const int h, const int m, const int s)
{     hour = h;
    minute = m;
    second = s;       }
```

// Implementation of class (cont.)

```
void Time :: print() const
{      cout << setw(10) <<  hour << ":"
              << setw(10) << minute << ":"
              << setw(10) <<  second << "\n";   }


bool Time :: equals(const Time &otherTime)
{      if(hour == otherTime.hour
         && minute == otherTime.minute
         && second == otherTime.second)
         return true;
     else
         return false;   }
```

Client Code

Client code, is the one that includes the main function. This file should be stored by any name as time_test.cpp

```
#include <iostream>
using namespace std;
#include “test_Time.h"
int main()
{     Time t1(10, 50, 59);
    t1.print();   // 10:50:59
    Time t2;
    t2.print(); // 06:39:09
    t2.setTime(6, 39, 9);
    t2.print();  // 06:39:09
      if(t1.equals(t2))
        cout << "Two objects are equal\n";
    else          cout << "Two objects are not equal\n";              return 0;  }
```

Output of Ex_10

- 10: 50: 59
- 0: 0: 0
- 6: 39: 9
- Two objects are not equal

If the main will be

```
int main()
{     Time t1;
    t1.print();   // 10:50:59
    Time t2;
    t2.print(); // 06:39:09
    t2.setTime(6, 39, 9);
    t2.print ( );
    t1=t2;
    t2.print();  // 06:39:09
      if(t1.equals(t2))
        cout << "Two objects are equal\n";
    else          cout << "Two objects are not equal\n";     return 0;  }
```

Output will be

0: 0: 0

0: 0: 0

6: 39: 9

6: 39: 9

Two objects are equal