BIT

5. Hiding the Implementation

Hu Sikang
skhu@163.com

School of Computer
Beijing Institute of Technology

B17

Contents

- Setting limits
- Friend

5.1 Setting limits

- There are two reasons for controlling access(private and public) to members:
  - The first is to keep the client programmer's hands off functions they should touch.
  - The second reason for access control is to allow the library designer to change the internal workings of the class without worrying about how it will affect the client programmer.

5.1.1 C++ access control

```
class Point {
 public:
    double GetX(); // retuarn the value of x
    double GetY(); // return the value of y
    void SetX(double valX) { x = valX; }  // set x
    void SetY(double valY) { y = valY; }  // set y
 private:
    double x,y;  // the coordinates
};  // semicolon:  don't forget it!
```

Access Control

Member functions

Member variables

5.1.2 Styles of creating a class

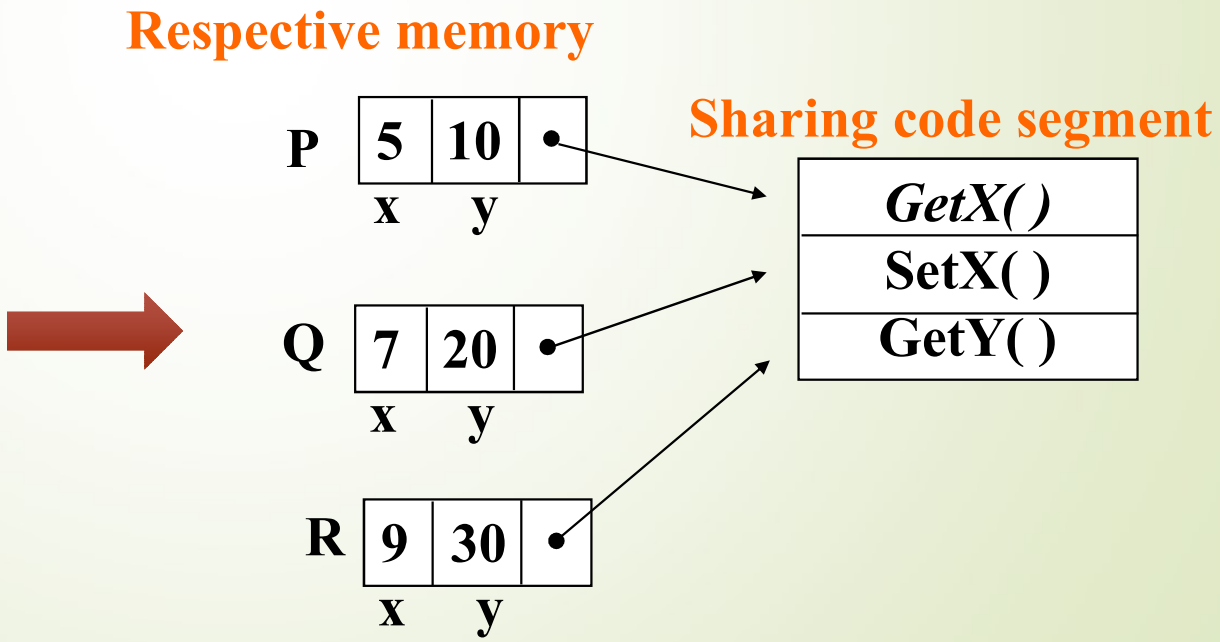
```
class Point {
 public:
     double GetX();
     double GetY();
     void SetX(double valX) { x = valX; }  // set x
     void SetY(double valY) { y = valY; }  // set y
 private:
     double x,y;
};

double  Point::GetX()   { return x;   }
double  Point::GetY()   {  return y;   }
double GetX()   {  cout << “Global Function!”; }
```

```
int main( )
 {
    //define two objects
    Point P, P2;

    //call member function
    P.SetX(300);

    P2 = P;    //object assigned

    cout << "x= "<< P.GetX();
    GetX();    //  Call Global Fun
    return 0;
}
```

5.1.3 Class Members

- A class object contains a copy of the data defined in the class.
  - The functions are shared.
  - The data belonged to themselves.

```
Point  P(5, 10);
Point  Q(7, 20);
Point  R(9, 30);
```



5.1.4 Controlling Member Access

```
class class_name{
public:
//public members
private:
//private members
};
```

817

Controlling Member Access

```
class class_name{
public:
 //public members
private:
 //private members
};
```

The part of public constitutes the public interface to objects of the class. If the *mem_fun()* is the public member function of class, you can wirte like these:

```
class_name obj;
obj.mem_fun( ); // OK
```

817

Controlling Member Access

```
class class_name{
public:
//public member
private:
//private member
};
```

The part of private can be used only by member function. If the *mem_fun()* is the private member function of class, you cann't write like these:

```
class_name obj;
obj.mem_fun( );      // Error
```

Again Point class

```
class Point {
public:
    double GetX();     // return the value of x
    double GetY();     // return the value of y
    void SetX(int valX) { x=valX; }  // set x
    void SetY(int valY) { y=valY; }  //set y
private:
     double x, y;          //the coordinates
};
```

```
int main()
{
        Point P;
        P.SetX(10);
        P.GetX();

        //error: private member
        P.x = 10;
        return 0;
}
```

5.1.5 Notes about defining a class

- Data members should be private.
- Public data members violate the principles of encapsulation.
- Function members can be public if it is for out services.

5.2 Friends

A *friend* function is a function which can obtain the private member of a class. But it *doesn't belong to* a class.

Syntax:

```
friend   data_type function_name(arguments);
```

5.2.1. a global function as a friend

```
#include <iostream>
using namespace std;
class Time
   int hours, minutes;
public:
   SetTime(int nhours, int nminutes)
   {  hours = nhours;  minutes = nminutes;  }
   friend void show (const Time&);     //friend functions
};
void show (const Time& showTime)   // no prefix "friend"
{
     //access private members
     cout << showTime.hours << ":" << showTime.minutes << endl;
}
```

```
int main()
{
      Time time;
      time.SetTime(20, 30)

      // calling friend-functions
      show (time);
      return 0;
}
```

5.2.2 a global function as a friend of two classes

```
#include <iostream>
using namespace std;
class Boat;  // forword references
class Car {
public:
        Car(int i) { weight = i; }
        friend int totalWeight(Car &c, Boat &b);
private:
        int weight;
};

class Boat {
  public:
        Boat(int i) { weight=i; }
        friend int totalWeight(Car &c,Boat &b);
  private:
        int weight;
  };
```

```
int totalWeight(Car &c, Boat &b)
{    return c.weight+b.weight;   }

 int main( )   {
        Car c(10);
        Boat b(8);
        cout<<"The total weight is "
                << totalWeight(c,b) << endl;
        return 0;
}
```

Summary

- Access control in C++ gives valuable control to the creator of a class. The users of the class can clearly see exactly what they can use and what to ignore.
- The *public* interface to a class is what the client programmer *does* see.
- The *private* interface to a class is what the client programmer doesn't see.