C++ Class and Objects

Yaping Jing

CS270 – Computer Science II

Terminologies and Concepts of Object Oriented Programming

- Class
- Object/Instance
- Attributes/State/Data/Member Variables.
- Member Functions/Methods
- Constructors What's the purpose?
- Access specifiers/Modifiers: `public private protected`
- `this` object

Member Functions

- Are declared inside the class body, in the same way as declaring a function.
- Their definition can be placed inside the class body, or outside the class body
- Can access both public and private member variables of the class
- Can be referred to using dot or arrow operator

More About Default Constructors

```
Rectangle();  //default constructor
```

- If all of a constructor's parameters have default arguments, then it is a default constructor. For example:

```
Rectangle (double w=1, double l=1);
```

- Creating an object and passing no arguments will cause this constructor to execute:

```
Rectangle r;
```

Only one default constructor is needed.

Default Constructor Example

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

class  Rectangle {
  private:
    double width, length;

  public:
     Rectangle (double w=1, double l=1);
     void set_values (double w, double l);
     double area ();
};

#endif
```

Implementation Example

```
#include "Rectangle.h"

 Rectangle :: Rectangle(double  w, double  l){
    width = w;
    length = l;
}
void Rectangle :: set_values (double  w, double  l){
    width = w;
    length = l;
}
double Rectangle :: area (){
    return width * length;
}
```

Client/Test Program – test.cpp

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

int main(){

    Rectangle r2;
    cout << r2.area() << endl;

    Rectangle r3(10);
    cout << r3.area() << endl;

    Rectangle r4(2, 8);
    cout << r3.area() << endl;

   return 0;
}
```

More on this Pointer

```
#include "Rectangle.h"

 Rectangle :: Rectangle(double  w, double  l){
   this->width = w;    // this pointer is made explicit
   this->length = l;
}

void Rectangle :: set_values (double  w, double  l){
   width = w;
   length = l;
}

double Rectangle :: area (){
   return width * length;
}
```

Pointer to an Object

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

int main(){
    Rectangle r2(3.0, 5.0);
    cout << r2.area() << endl;

     Rectangle *rPtr = NULL;

     rPtr = &r2;
     cout << rPtr->area() << endl;

     rPtr = new Rectangle(2, 8);
     cout << rPtr->area() << endl;

     delete rPtr;
     rPtr = NULL;
     return 0;
}
```

Arrays of Objects

```
Rectangle r[20];

Rectangle *r4=new Rectangle[40];  // dynamic array

delete [] r4;
```

Default constructor for object is used when array is defined

More Concepts of C++ Class and Objects

- destructors
- Friend functions
- Copy constructors vs. assignment operator
- Operator overloading
- Nonmember functions

Destructors

- A function used to clean up an object of a class prior to deleting that object
- Destructor name: `~classname`
- No parameters, no return type. Compiler provides one if you do not!
- Called automatically
- If constructor allocates dynamic memory, destructor should release it

Destructor Example

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

#include<iostream>
using namespace std;

class Rectangle{
   private: double width, length;

   public:
      Rectangle();
      Rectangle(width, length);
      int area();
      ~Rectangle(){ cout << "End of story, period."<<endl;  }
};
#endif
```

Dynamic Array and Destructor Example

```
// Vect.h

#ifndef VECT_H
#define VECT_H

class Vect {
   private:
      int* data;
      int size;

   public:
       Vect(int n);
       Vect();
       int getSize();
       ~Vect();
};
#endif
```

```
// Vect.cpp

Vect::Vect(int n){
  size = n;
  data = new int[n];
}

Vect::~Vect(){
  delete [] data;
}

int Vect::getSize(){
   return size;
}

Vect::Vect(){
  size = 0;
  data = NULL;
}
```

Testing Vect Class

```
// test.cpp

#include<iostream>
#include "Vect.h"
using namespace std;

int main(void) {
    Vect v(5);
    int size = v.getSize();
    cout << "vector size is " << size << endl;
}
```

Copy Constructor vs. Assignment Operator

Copy Constructor vs. Assignment Operator Example

```
Vect a(10);    // line 1,
Vect b = a;    // line 2, initialize b from a

Vect c;        // line 4
c = a;         // line 5, assigns a to c
```

Question: Does line 2 invoke the same behavior as lines 4-5?

Motivation Example

- Since there is no **copy constructor** in `Vest` class, the systems uses its default, which is a **shallow copy**:

```
/* the statement b = a
   merely copies the pointer to the array's initial element
 */
b.data = a.data ; // no contents is copied over.
```

- Since there is no **assignment operator** provided in `Vest` class the systems also uses its default, which does a **shallow copy**:

```
b.data = a.data ; // no contents is copied over.
```

Copy Constructor vs. Assignment Operator Example

```
Vect a(10);
Vect b = a;    // need copy constructor to do a deep copy

Vect c;
c = a;         // need assignment operator to do a deep copy
```

Copy Constructor Declaration

```
// Vect.h

#ifndef VECT_H
#define VECT_H

class Vect {
   private:
     int* data;
     int size;

   public:
      Vect(int n);
      Vect();
      int getSize();
      ~Vect();
      Vect(const Vect& v); // copy constructor
};
#endif
```

Copy Constructor Implementation Example

```
Vect::Vect(const Vect& a){
    size = a.size;
    data = new int[size];

    for (int i = 0; i < size; i++){
      data[i] = a.data[i];
    }
}

Vect b = a;    // b is initialized with the contents of a
```

Operator Overloading

Operator Overloading

- Operators such as `=` `+` `<=` `>=` and others can be redefined when used with objects of a class.
- Prototype for the overloaded operator goes in the declaration of the class.
- Declaration Syntax:

```
type operator= (const someClass& o);
```

Overloaded Operator Declaration and Calling Example

```
// assignment operator declaration example
Vect operator=(const Vect& v);

// addition operator declaration example
void operator+(const Rectangle& r);

// testing example for the assignment operator:
Vect c;
Vect a(10);
c = a;   // invokes  c.operator=(a)


// testing example for the + operator:
Rectangle r1, r2;
r1 = r1+r2;   // invokes  r1.operator+(r2)
```

friend function

friend Function

- Declared using the keyword friend in the **class definition**.
- Decalred as a **non-member** function without using scope "::" operator.
- Called **without** using the '.' operator. (why?)
- Has access to all non-public member variables of the class.
- The major use of friends is to provide more efficient access to data members than the function call. (why?)

friend Function Example

```
// declaration example
class  Foo{
  private: int secrete;

  public:
    friend ostream& operator<<(ostream& out, const Foo& x);
};

// implementation example
ostream& operator<<(ostream& out, const Foo& x){
   out << x.secret << endl;
   return out;
}

// testing example
Foo x;
Foo y;
cout << x << y << endl;
```

Nonmember Function

- Declared outside of the class.
- Invoked without object.

See HW4 Specification and Testing file for example.