MH1402 Algorithms & Computing II

Lecture 11 Classes

Wu Hongjun

Overview

- **Class**
  - **Declaration**
  - **Member functions**
    - **including Constructors & Destructors**
  - **Passing an instance of a class to function**
    - **by value**
    - **by reference**

What is a class?

- **Class can be considered as a souped-up data type with its own members:**
  - **Member constants**
  - **Member variables: Different types of variables may be included**
  - **Member functions, operators**
  - **Member classes …**
- **An object is a specific instance of a class**
  - **The object here is not related to "object file" (the .o file after compilation)**
- **Class is one of the defining ideas of object-oriented programming**
  - **We learn the basics of classes in this class.**

What is a class?

- **Classes we have used in this course (built-in C++ classes)**
  - **vector, string**

    Example: vector<int> foo;

    // foo is an object of the vector class
    - **elements of vector (of the same data type)**
    - **member functions: push_back(), pop_back(), insert(), erase(), …**
    - **operator [ ] used for accessing elements**
  - **ostream, istream**
    - **cout is an object of ostream (output stream),**
    - **cin is an object of istream (input stream)**
  - **ofstream, ifstream**
    - **Example: ofstream foo;**

      // foo is an object of the ofstream class (output file stream)

A simple example of class (here they are in one file)

```
#include <iostream>
using namespace std;
class Rectangle
{
  public:
    double width, height;
    void inc_size();
    int area();
};
```

Class declaration
(class interface)

```
void Rectangle::inc_size()
{
    width = width + 1;
    height = height + 1;
}
int Rectangle::area()
{
    return width*height;
}
```

Define member
functions of Rectangle
(class implementation)

```
int main ()
{
    Rectangle foo;
    foo.width  = 3;
    foo.height = 4;
    cout << "area: " << foo.area();
    foo.inc_size();
    cout << "area: " << foo.area();
    return 0;
}
```

Declare object foo of the Rectangle class

Access member variables
and functions of foo.

Class Declaration

- **Class declaration specifies class name, class members:**
  - **The key word is class**
  - **The member variables are declared**
  - **The member functions are declared**
    - **Note the parameter list of the member functions**
    - **The member variables of the class can be used inside the member functions directly without being passed into those member functions**

```
class Rectangle
{
  public:
    int width, height;    //declare member variables
    void inc_size();      //declare member function
    int area();           //declare member function
};
```

Class Declaration

- **Class declaration specifies class name, class members:**
  - **The key word public indicates that those members (variables, functions…) can be accessed outside the class**
    - **All the members of the class Rectangle in the previous slides are public**
  - **Another key word private indicates that those members (variables, functions …) can be accessed only within the class**
    - **Example: a private variable can only be used in member functions/classes in the same class**
  - **Class declaration is followed by semicolon ;**

```
class Rectangle
{
  public:
     int width,height;
     void inc_size();
     int area();
};
```

Class Declaration: private (example)

```
#include <iostream>
using namespace std;
class Rectangle
{
  private:
    int width, height;
  public:
    int area();
    void set_size(int,int);
};
int Rectangle::area()
{  return width*height; }
```

```
void Rectangle::set_size(int x, int y)
{
    width = x;
    height = y;
}
int main ()
{
    Rectangle foo;
    foo.width  = 3;
    foo.height = 4;
    foo.set_size(3,4);   // this statement is ok.
    cout << "area: " << foo.area();
    return 0;
}
```

Compilation error: cannot access private variables width and height

Class Declaration: private

- In the example given in the previous slide, width and height are two private variables of the class Rectangle
  - The main function cannot access those two variables since main is not a member function of the class Rectangle
  - The set_size and area are member functions of Rectangle, so they can access width and height
  - It prevents the values of width and height from being accidently changed by the rest of the program
  - It is common that all the variables of a class are in the private section
- Encapsulation
  - Placing variables and functions in the private section of a class is called encapsulation

Class Declaration

- **It is common that in class declaration, the member functions are only declared. Then**
  - **the declaration section is called the *class interface* ;**
  - **the section containing the member function definition is called the *implementation***
- **A class can only be used after its declaration (similar to function)**
  - **When we use the built-in class (such as string, vector), we must include the header file #include <string> #include <vector>**

    (In header files string and vector, string and vector classes are declared)
- **The member functions of a class can be defined only after the declaration of the class**

Class Member Functions

- **Class member functions are defined as usual except that:**
  - **The Class_Name:: is added before the function name**
  - **The member function can access all the member variables in the class without passing those variables as parameters into the function**

```
int Rectangle::area()
{
    return width*height;
}
```

```
void Rectangle::set_size(int x, int y)
{
    width = x;
    height = y;
}
```

Class Member Functions

- **Scope resolution operator ::**
  - **It helps to identify and specify the context to which an identifier refers**
    - **The scope resolution operator (::) in the previous slide specifies the class to which the member function being declared belongs**
  - **When we use cout, cin, vector, string, the scope resolution operator is used to specify that they are from the std (C++ standard library) namespace**
    - **std::out, std::cin, std::vector, std::string …**

Class Member Functions

- **The dot operator is used to specify a member variable or member function of an object**
  - **Example:**

```
vector<int> foo;
foo.push_back(3);
cout << foo.at(0);  // use member function “at” to access elements

cout << foo[0];     //  alternatively, use operator [ ] to access elements
```

Class Member Functions

- It is allowed to define class member functions within the class declaration.
  - Example:
- But it is better to define member functions separately from the class declaration.

```
class Rectangle
{
  private:
    int width, height;
  public:
    int area()
    {
       return width*height;
    }
    void set_size(int,int)
    {
       width = x;
       height = y;
    }
};
```

Where to declare a class and to define the member functions of a class?

- **When writing small programs, the class declaration and the member functions can all be implemented in the same source file (.cpp) together with the main function**
- **When developing large programs, it is better to declare classes in header files (.h), and to define the member functions of the classes in separate source files (.cpp)**
  - **Traditionally, the class declaration is given in a header file with the same name as the class, and the member functions defined outside of the class are given in a .cpp file of the same name as the class**
  - **Example given in the next page.**

Rectangle.h

```
class Rectangle
{
  private:
    int width, height;
  public:
    int area();
    void set_size(int,int);
};
```

Rectangle.cpp

```
#include "Rectangle.h"
int Rectangle::area()
{
    return width*height;
}
void Rectangle::set_size(int x, int y)
{
    width = x;
    height = y;
}
```

Continued on next slide …

compute_recantangle.cpp

```
#include <iostream>
#include "Rectangle.h"
using namespace std;
int main ()
{
   Rectangle foo,bar;
   foo.set_size(3,4);
   cout << "area: " << foo.area() << endl;
   bar.set_size(5,6);
   cout << "area: " << bar.area() << endl;
   return 0;
}
```

Passing an instance of class to function

- **We already learned how to pass a vector and string to function**
- **Pass an object (an instance of class) to function is similar**
  - **It is “passing by value”: changes to the member variables in an object in a function are not preserved after the function call**
  - **Passing by reference can be used if we want to retain the changes after the function call**
- **Examples are given in the following slides.**

Passing by Value v.s. Passing by Reference Example

```
#include <iostream>
using namespace std;
class Rectangle {
  private:
    int width, height;
  public:
    int area();
    void set_size(int,int);
};
```

```
int Rectangle::area()
{
    return width*height;
}

void Rectangle::set_size(int x, int y)
{
    width = x;
    height = y;
}
```

Continued on the next slide ….

Passing by Value Program

```
void bar(Rectangle qux, int w, int h)
{
    qux.set_size(w,h);
}
int main ()
{
   Rectangle foo;
   foo.set_size(3,4);
   bar(foo, 4, 5);
   cout << "area: " << foo.area();
                  // print 12
   return 0;
}
```

Passing by Reference Program

```
void bar(Rectangle& qux, int w, int h)
{
    qux.set_size(w,h);
}
int main ()
{
   Rectangle foo;
   foo.set_size(3,4);
   bar(foo, 4, 5);
   cout << "area: " << foo.area();
                  // print 20;
   return 0;
}
```

Constructors & Destructors

- **Constructor is a member function of a class**
- **Constructor never returns value, no return type (no even void)**
- **The name of constructor is same as the class name**
- **Constructor gets automatically invoked when an object gets declared (created)**
- **Constructor cannot be called explicitly as if it was regular member functions (a constructor of an object is only executed once, when a new object of that class is created)**
- **Constructor by default creates an empty object**
  - **User may add input parameters and statements to the constructor, such as initialization, allocation of memory space … (Example: the values of elements of vector are set to zero at the time of declaration)**
- **If constructor is not defined in a class, the system will provide default constructor automatically, i.e., create an empty object when an object gets declared.**

Constructors & Destructors

- Destructor is a **member function** of a class
- Destructor never returns value, **no return type (no void)**
- The name of destructor is the name of class with ~ being prepended
- Destructor gets **automatically invoked** when an object ceases to exist
  - Each object (or variable) has a scope. When a program executes outside of that scope, that object (or variable) ceases to exist
- Destructor cannot be called explicitly as if it was regular member functions (a destructor of an object is only executed once, when an object ceases to exist)
- Destructor by default may set the data members to zero (for clearing sensitive data from memory), or do nothing at all
  - User can add more statements in the destructor
- If destructor is not defined in a class, the system will provide default destructor automatically, i.e., set the data members to zero, or doing nothing at all when an object ceases to exist (depending on compiler).

```
#include <iostream>
using namespace std;
class Rectangle
{
  private:
    int width, height;
  public:
    Rectangle();   //declare constructor
    ~Rectangle(); //declare destructor
    int area();
};
int Rectangle::area()
{
    return width*height;
}
```

```
Rectangle::Rectangle()   //define constructor
{
    width  = 0;   // initialization of variables
    height = 0;
}
Rectangle::~Rectangle()  //define destructor
{
    width  = 0;
    height = 0;
}
int main ()
{
    Rectangle foo;
    cout << "area: " << foo.area();
    return 0;
}
```

Example 1

Constructor gets invoked when foo is declared

Destructor gets invoked when foo ceases to exist

```
#include <iostream>
using namespace std;
class Rectangle
{
  private:
    int width, height;
  public:
    Rectangle(int x, int y);
    ~Rectangle();
    int area();
};
int Rectangle::area()
{
    return width*height;
}
```

```
Rectangle::Rectangle(int x, int y)
{
    width  = x;
    height = y;
}
Rectangle::~Rectangle()
{
    width  = 0;
    height = 0;
}
int main ()
{
    Rectangle foo(2,3);
    cout << "area: " << foo.area();
    return 0;
}
```

Example 2 Constructor with Input Parameter(s)

Constructor gets invoked when foo is declared

Destructor gets invoked when foo ceases to exist