南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

C/C++ Program Design

Lab 9, Class

廖琪梅，王大兴

Class

- Class constructor and destructor
- Static member variables and static member functions
- Const member variables and const member functions
- **this** pointer

Class Constructors

A class constructor is a special member function:

1. Has exact same name as the class
2. No return value
3. It is a public member function of the class
4. Invoked whenever you create objects of that class

Constructors can be very useful for setting initial values for certain member variables.
If you do not provide a constructor, C++ compiler **generates** a **default constructor** (has no parameters and an empty body) for you.

Note:The constructor can be overloaded, be careful the constructor with default arguments.

```
#pragma once

class Box
{
private:
    double length;      //Length of a box
    double breadth;     // Breadth of a box
    double height;      // Height of a box

public:
    // Default Constructor
    Box();
    // Parameterized Constructor
    Box(double, double, double);

    // Member functions declaration
    double getVolume(void);
    void setLength(double len);
    void setBreadth(double bre);
    void setHeight(double hei);
};
```

Implementation of functions using the **scope resolution operator(::)**

```
Box::Box() {
    length = 3.0;
    breadth = 4.0;
    height = 5.0;
}
Box::Box(double length, double breadth, double height) {
    this->length = length;
    this->breadth = breadth;
    this->height = height;
}
```

"this" is a pointer points to the object itself

```h
// complex.h -- Complex class without operator overloading
#ifndef _MYCOMPLEX_H
#define _MYCOMPLEX_H

class Complex {
private:
    double real;
    double imag;

public:
    Complex();
    Complex(double re, double im);
    Complex Add(const Complex& data);
    void Show() const;
};

#endif //_MYCOMPLEX_H
```

```cpp
//complex.cpp --- implementing Complex methods
#include <iostream>
#include "complex.h"

Complex::Complex() : real(0), imag(0)
{
}
Complex::Complex(double re, double im) : real(re), imag(im)
{
}
```

initialization list

The member **initialization list** is set off from the parameter list by a colon. It is a comma-separated list in which the value to be assigned the member is placed in parentheses following the member's name.

The member initialization list is used primarily to pass arguments to **member class object** constructors.

Default constructor

A **default constructor** is a constructor that is used to create an object when you don't provide explicit initialization values. If you don not provide any constructors, the compiler will automatically supplies a default constructor. It's an implicit version of a default constructor, and it does nothing. There is **only one default constructor** in a class.

If you define any constructor for a class, the compiler will not provide default constructor. **You must define your own default constructor that takes no arguments.**

You can define a default constructor **two ways**. **One** is to provide default values for all the arguments to the existing constructor:

```
Box(double len = 1.0, double bre = 1.0, double hei = 1.0);
```

The **second** is to use function overloading to define a second constructor that has no arguments:

```
Box( );
```

NOTE: You can have **only one default constructor**, so be sure that you don't do both.

Creating objects

When you create an object, the constructor will be invoked automatically. If you define default constructor, you can declare object variables without initializing them explicitly.

```
Box box1;                // call the default constructor implicitly
Box box2 = Box();        // call the default constructor explicitly
Box *pbox = new Box;     // call the default constructor implicitly
```

You shouldn't be misled by implicit form of the non-default constructor.

```
Box first(5.0, 6.0, 9.0);  // call the nondefault constructor
Box second();              // declare a function not call the default constructor
Box third;                 // call the default constructor implicitly
```

second() is a function that returns a Box object.

When you implicitly call the default constructor, do not use parentheses.

Class Destructors

A class destructor is also a special member function:

1.A destructor name is the same as the classname but begins with tilde(~) sign.
2. Destructor has no return value.
3. A destructor has no arguments.
4. There can be only one destructor in a class.
5. The compiler always creates a default destructor if you fail to provide one for a class.
6. Invoke when an object goes out of scope or the delete is applied to a pointer to the object.

Destructor can be very useful for releasing resource before coming out of the program like closing files, releasing memories etc.

Note:The destructor can not be overloaded.

Static member variables

The static member variables in a class are shared by all the class objects as there is only one copy of them in the memory, regardless of the number of objects of the class.You must provide an explicit definition of that instance within a program text file. The definition looks like the global definition of an object except that its name is qualified with the class scope operator ::

```
#pragma once

class StaticTest
{
private:
    static int m_value;     //declare a static variable

public:

    static int getValue()   //define a static member function
    {
        return m_value;
    }
};

int StaticTest::m_value = 12; //define and initialize the static variable outside the class definition
```

static member variable can not be initialized when it is defined

static member variable must be initialized outside the class definition

Static member functions

A static member function can be invoked independently of a class object in exactly this way. A member function can be declared as static only if it does not access any non-static class members. We make it static by prefacing its declaration within the class definition with the keyword **static**

```
#pragma once

class StaticTest
{
private:
    static int m_value;      //declare a static variable

public:

    static int getValue()    //define a static member function
    {
        return m_value;
    }
};

int StaticTest::m_value = 12; //define and initialize the static variable outside the class definition
```

Define a static member function

You can use object or class to access the static members.

```
#include <iostream>
#include "static_variable.h"

using namespace std;

int main()
{
    StaticTest t;

    cout << "Invoke static member function by object:\n";

    cout << "The value of the static member variable is:"
        << t.getValue() << endl;

    cout << "Invoke static member function by class:\n";
    cout << "The value of the static member variable is:"
        << StaticTest::getValue() << endl;


    return 0;
}
```

Const member variables

The **const** keyword specifies that a variable's value is constant and tells the compiler to prevent the programmer from modifying it. If some member variables need not be modified, these variables can be defined as **const**. These const member variables can be initialized by **initialization list** in the constructor.

```
#include <iostream>
using namespace std;
class MyClass
{
private:
    const int x;
public:
    MyClass(int a) : x(a)
    {
        //constructor
    }
    void show_x()
    {
        cout << "Value of constant x: " << x;
    }
};
```

const member variable

Initialize the const member variable by initialization list

```
int main()
{
    MyClass ob1(40);
    ob1.show_x();

    return 0;
}
```

Create an object of MyClass, initialize the value of x 40.

Const member variables

```
class Person {
private:
    static const int Len = 30;
    char name[Len];

public:
    Person():name{"LiXia"}
    {}

    void display();
};
```

static const member variable can be initialized when it is defined

```
class Person {
private:
    enum {Len = 30};
    char name[Len];

public:
    Person():name{"LiXia"}
    {}

    void display();
};
```

Define an enumeration

Const member functions

The **const** modifier follows the parameter list of the function, such function is called **const member function**. const indicates that the data of the class would not be modified by the function.

A const member function defined outside the class body must specify the **const** modifier in **both its declaration and definition**.

Const member functions

```
// complex.h -- Complex class without operator overloading
#ifndef _MYCOMPLEX_H
#define _MYCOMPLEX_H

class Complex {
private:
    double real;
    double imag;

public:
    Complex();
    Complex(double re, double im);
    Complex Add(const Complex& data);
    void Show() const;
};

#endif //_MYCOMPLEX_H
```

```
void Complex::Show() const
{
    std::cout << real << " + " << imag << "i" << std::endl;
}
```

this pointer

There's only one copy of each class's functionality, but there can be many objects of a class. Every object has access to its own address through a pointer called ***this***. The ***this*** pointer is passed(by the compiler) as an implicit argument to each of the object's **non-static** member functions.

```
Box& Box::copy(const Box& rhs)
{
    this->length = rhs.length;
    this->breadth = rhs.breadth;
    this->height = rhs.height;

    return *this;
}
```

Within a member function, the ***this*** pointer addresses the class object that invokes the member function. In our example, **box1** is addressed by the this pointer.

Inside a class member function, the ***this*** pointer provides access to the class object through which the member function is invoked. To return **box1** from within copy(), we simply **dereference the this pointer**.

```
Box box1;                  // Declare box1 invoking default constructor implicitly
Box box2(5.0, 6.0, 9.0);   // Declare box2 invoking nondefualt constructor explicitly

box1.copy(box2);           // Copy box2 to box1
```

Exercises

1. Can the program below be run successfully? Why? How to modify it? Can the **display()** function be invoked by the Demo class instead of an object of Demo?

You need to explain the reason to a SA to pass the test.

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

class Demo
{
  public :
  static int num;
  void display()
  {
    cout << "The value of the static member variable num is: " << num <<endl;
  }
};

int main()
{
  Demo obj;
  obj.display();
  return 0;
}
```

Exercises

2. What is the result of the program below? What happens if you uncomment the commented line in function main()? Why? You need to explain the reason to a SA to pass the test.

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

class ConstMember
{
private:
  const int m_a;
public:
  ConstMember(int a) : m_a(a) {}

  void display() const
  {
    cout << "The value of the const member variable m_a is: "  << m_a << endl;
  }
};
```

```
int main()
{
  ConstMember o1{666};
  ConstMember o2{42};

  o1.display();
  o2.display();

//  o1 = o2;

  return 0;
}
```

Exercises

3.Define a class called **Complex** for performing arithmetic with complex numbers. Write a program to test your class. Complex numbers have the form

realPart + imaginaryPart * i

- Use two member variables to represent the private data of the class. Provide a constructor that enables an object of this class to be initialized when it's declared. The constructor should contain default values in case no initializers are provided. Provide public member functions that perform the following tasks:

- a) add—Adds two Complex numbers: The real parts are added together and the imaginary parts are added together.
- b) subtract—Subtracts two Complex numbers: The real part of the right operand is subtracted from the real part of the left operand, and the imaginary part of the right operand is subtracted from the imaginary part of the left operand.
- c) display—Displays a Complex number in the form of **a + bi** or **a - bi**, where **a** is the real part and **b** is the imaginary part.

Tip: If a member function does not modify the member variables, define it as const member function.