程序设计基础及语言 II

实验指导手册

东南大学

计算机科学与工程学院

软件学院

人工智能学院

网络空间安全学院

2020 年 2 月

目　　录

Lab1 Classes：A Deeper Look

Objectives:

1. Use and include guard and access class members
2. To specify const (constant) objects and const member functions.
3. Learn the order of constructor and destructor calls.
4. To create objects composed of other objects.
5. To use friend functions and friend classes.
6. To use the pointer.
7. To use static data members and member functions.

Experiments

EX1:(9.5 Complex Class)

1. Description of the Problem

(Complex Class) Create 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*

where *i* is $\sqrt{-1}$

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

(a) ***Adding*** two Complex numbers: The real parts (实部) are added together and the imaginary parts (虚部) are added together.

(b) ***Subtracting*** 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)***Printing*** Complex numbers in the form (a, b), where a is the real part and b is the imaginary part.

2. Sample Output

```
(1, 7) + (9, 2) = (10, 9)
(10, 1) - (11, 5) = (-1, -4)
```

EX2: (9.7, Enhancing Class Time)

1. Description of the Problem

(Enhancing Class Time) Modify the Time class of Figs. 9.4-9.5 to include a tick member function that increments the time stored in a Time object by one second. The Time object should always remain in a consistent state. Write a program that tests the tick member function in a loop that prints the time in standard format during each iteration of the loop to illustrate that the tick member function works correctly. Be sure to test the following cases:

(a) Incrementing into the next minute.

(b) Incrementing into the next hour.

(c) Incrementing into the next day (i.e., 11:59:59 PM to 12:00:00 AM).

Note：

Then change the tick member function to a friend function of class Time, which will access the private data member of Time directly. You should get the same output as above.

friend void tick(Time &t); // increment one second

2. Sample Output

```
11:59:57 PM
11:59:58 PM
11:59:59 PM
12:00:00 AM
12:00:01 AM
12:00:02 AM
12:00:03 AM
12:00:04 AM
12:00:05 AM
12:00:06 AM
12:00:07 AM
12:00:08 AM
12:00:09 AM
12:00:10 AM
12:00:11 AM
```

EX3: (9.14 HugeInteger Class)

1. Description of the Problem

(HugeInteger Class) Create a class ***HugeInteger*** that uses a 40-element array of digits to store integers as large as 40 digits each. Provide member functions: (a) Constructor, destructor, (b) ***input***, ***output***, ***add*** and ***substract,*** (c) For comparing HugeInteger objects, provide functions ***isEqualTo***, ***isNotEqualTo***, ***isGreaterThan, isLessThan isGreaterThanOrEqualTo*** and ***isLessThanOrEqualToeach,*** each of these is a "predicate" function that simply returns true if the relationship holds between the two HugeIntegers and returns false if the relationship does not hold. Also, provide a predicate function ***isZero***.

If you feel ambitious, provide member functions ***multiply***, ***divide*** and ***modulus***.

注：不考虑负数情况，即hugeintA-hugeintB确保hugeintA大于hugeintB；而hugeintA+hugeintB，确保不溢出

2.Sample Class Definition

```
#ifndef HUGEINTEGER_H
#define HUGEINTEGER_H
class HugeInteger
{
public:
HugeInteger( int = 0 ); // conversion/default constructor
HugeInteger( const char * ); // conversion constructor
// addition operator; HugeInteger + HugeInteger
HugeInteger add( const HugeInteger & );
// addition operator; HugeInteger + int
HugeInteger add( int );
// addition operator;
// HugeInteger + string that represents large integer value
HugeInteger add( const char * );
// subtraction operator; HugeInteger - HugeInteger
HugeInteger subtract( const HugeInteger & );
// subtraction operator; HugeInteger - int
HugeInteger subtract( int );
// subtraction operator;
// HugeInteger - string that represents large integer value
HugeInteger subtract( const char * );
bool isEqualTo( HugeInteger & ); // is equal to
bool isNotEqualTo( HugeInteger & ); // not equal to
bool isGreaterThan(HugeInteger & ); // greater than
bool isLessThan( HugeInteger & ); // less than
bool isGreaterThanOrEqualTo( HugeInteger & ); // greater than
// or equal to
bool isLessThanOrEqualTo( HugeInteger & ); // less than or equal
bool isZero(); // is zero
void input( const char * ); // input
void output(); // output
private:
int integer[ 40 ]; // 40 element array
}; // end class HugeInteger
#endif
```

3.Sample Output

```
7654321 + 7891234 = 15545555

7891234 - 5 = 7891229

7654321 is equal 7654321

7654321 is not equal to 7891234

7891234 is greater than 7654321

5 is less than 7891234

5 is less than or equal to 5

0 is greater than or equal to 0

n3 contains value 0
```

EX4: Simple Calculator

1.Description of the Problem

Write a SimpleCalculator class that has public methods for adding, subtracting, multiplying and dividing twodoubles. A sample call is as follows:

double answer = sc.add(a, b);

Object sc is of type SimpleCalculator. Member function add returns the result of adding its two arguments.

2.Sample Output

```
The value of a is: 10
The value of b is: 20

Adding a and b yields 30
Subtracting b from a yields -10
Multiplying a by b yields 200
Dividing a by b yields 0.5
```

EX5: Integer Set

1.Description of the Problem

Create class IntegerSet for which each object can hold integers in the range 0 through 100. A set is represented internally as an array of ones and zeros. Array element a[i] is 1 if integer i is in the set. Array element a[j]is 0 if integer j is not in the set. The default constructor initializes a set to the so-called “empty-set,” i.e., a set whose array representation contains all zeros.

Provide member functions for the common set operations. For example, a unionOfSets member function(already provided) creates a third set that is the set-theoretic union of two existing sets (i.e., an element of thethird array’s is set to 1 if

that element is 1 in either or both of the existing sets, and an element of the third set'sarray is set to 0 if that element is 0 in each of the existing sets).

Provide an intersectionOfSets member function which creates a third set which is the set-theoretic intersectionof two existing sets (i.e., an element of the third set's array is set to 0 if that element is 0 in either or both of the existing sets, and an element of the third set's array is set to 1 if that element is 1 in each of the existing sets).

An insertElement member function (already provided) inserts a new integer k into a set (by setting a[k]to 1).

Provide a deleteElement member function that deletes integer m (by setting a[m] to 0).

A printSet member function (already provided) prints a set as a list of numbers separated by spaces. Print only those elements which are present in the set (i.e., their position in the array has a value of 1). Print --- for an empty set.

Provide an isEqualTo member function that determines whether two sets are equal.

Provide an additional constructor that receives an array of integers and the size of that array and uses the array to initialize a set object.

Now write a driver program to test your IntegerSet class. Instantiate several IntegerSet objects. Test that all your member functions work properly.

2. Sample Output

```
Enter set A:
Enter an element (-1 to end): 45
Enter an element (-1 to end): 76
Enter an element (-1 to end): 34
Enter an element (-1 to end): 6
Enter an element (-1 to end): -1
Entry complete

Enter set B:
Enter an element (-1 to end): 34
Enter an element (-1 to end): 8
Enter an element (-1 to end): 93
Enter an element (-1 to end): 45
Enter an element (-1 to end): -1
Entry complete

Union of A and B is:
{   6   8  34  45  76  93   }
Intersection of A and B is:
{  34  45   }
Set A is not equal to set B

Inserting 77 into set A...
Set A is now:
{   6  34  45  76  77   }

Deleting 77 from set A...
Set A is now:
{   6  34  45  76   }
Invalid insert attempted!
Invalid insert attempted!

Set e is:
{   1   2   9  25  45  67  99 100   }
```

3. Problem-Solving Tips

1)Member function intersectionOfSets must return an IntegerSet object. The object that invokes this function and the argument passed to the member function should not be modified by the operation .intersectionOfSets should iterate over all integers an IntegerSet could contain (1–100) and add those integers that both IntegerSets contain to a temporary IntegerSet that will be returned.
2) Member function deleteElement should first verify that its argument is valid by calling utility function validEntry. If so, the corresponding element in the set array should be set to 0; otherwise, display an error message.
3) Member function isEqualTo should iterate over all integers an IntegerSet could contain and (1–100).If any integer is found that is in one set but not the other, return false; otherwise return true.

```
// Lab 2: SetTest.cpp
// Driver program for class IntegerSet.
#include <iostream>
using std::cout;
using std::endl;

#include "IntegerSet.h" // IntegerSet class definition

int main()
{
   IntegerSet a;
   IntegerSet b;
   IntegerSet c;
   IntegerSet d;

   cout << "Enter set A:\n";
   a.inputSet();
   cout << "\nEnter set B:\n";
   b.inputSet();
   /* Write call to unionOfSets for object a, passing
      b as argument and assigning the result to c */
   /* Write call to intersectionOfSets for object a,
      passing b as argument and assigning the result to d */
   cout << "\nUnion of A and B is:\n";
   c.printSet();
   cout << "Intersection of A and B is:\n";
   d.printSet();

   if ( a.isEqualTo( b ) )
      cout << "Set A is equal to set B\n";
   else
      cout << "Set A is not equal to set B\n";

   cout << "\nInserting 77 into set A...\n";
   a.insertElement( 77 );
   cout << "Set A is now:\n";
   a.printSet();

   cout << "\nDeleting 77 from set A...\n";
   a.deleteElement( 77 );
   cout << "Set A is now:\n";
   a.printSet();

   const int arraySize = 10;
   int intArray[ arraySize ] = { 25, 67, 2, 9, 99, 105, 45, -5, 100, 1 };
   IntegerSet e( intArray, arraySize );

   cout << "\nSet e is:\n";
   e.printSet();

   cout << endl;

   return 0;
} // end main
```

Lab2 Operator Overloading; String and Array Objects

Objectives:

1. What operator overloading is and how it can make programs more readable and programming more convenient.
2. To redefine (overload) operators to work with objects of user-defined classes.
3. The differences between overloading unary and binary operators.
4. To convert objects from one class to another class.
5. When to, and when not to, overload operators.
6. To create PhoneNumber, Array, String and Date classes that demonstrate operator overloading.
7. To use overloaded operators and other member functions of standard library class string.

Experiments

EX1: Complex Class

2. Description of the Problem

Consider class Complex shown in Figs. 10.14-10.16. The class enables operations on so-called complex numbers. These are numbers of the form realPart + imaginaryPart * I, where i has the value √-1.

a) Modify the class to enable input and output of complex numbers via overloaded >> and << operators, respectively (you should remove the print function from the class).
b) Overload the multiplication operator to enable multiplication of two complex numbers as in algebra.
c) Overload the == and != operators to allow comparisons of complex numbers.

3. Sample Main Function

```
int main()
{
    Complex x, y( 4.3, 8.2 ), z( 3.3, 1.1 ), k;

    cout << "Enter a complex number in the form: (a, b)\n? ";
    cin >> k; // demonstrating overloaded >>
    cout << "x: " << x << "\ny: " << y << "\nz: " << z << "\nk: "
         << k << '\n'; // demonstrating overloaded <<
    x = y + z; // demonstrating overloaded + and =
```

```
    cout << "\nx = y + z:\n" << x << " = " << y << " + " << z << '\n';
    x = y - z; // demonstrating overloaded - and =
    cout << "\nx = y - z:\n" << x << " = " << y << " - " << z << '\n';
    x = y * z; // demonstrating overloaded * and =
    cout << "\nx = y * z:\n" << x << " = " << y << " * " << z << "\n\n";
    if ( x != k ) // demonstrating overloaded !=
        cout << x << " != " << k << '\n';
    cout << '\n';
    x = k;
    if ( x == k ) // demonstrating overloaded ==
        cout << x << " == " << k << '\n';
    return 0;
} // end main
```

4. Sample Output

```
Enter a complex number in the form: (a, b)
? (2.5, 2)
x: (0, 0)
y: (4.3, 8.2)
z: (3.3, 1.1)
k: (2.5, 2)

x = y + z:
(7.6, 9.3) = (4.3, 8.2) + (3.3, 1.1)

x = y - z:
(1, 7.1) = (4.3, 8.2) - (3.3, 1.1)

x = y * z:
(23.21, 31.79) = (4.3, 8.2) * (3.3, 1.1)

(23.21, 31.79) != (2.5, 2)

(2.5, 2) == (2.5, 2)
```

EX2: Huge Integer Class

4. Description of the Problem

A machine with 32-bit integers can represent integers in the range of approximately -2 billion to +2 billion. This fixed-size restriction is rarely troublesome, but there are applications in which we would like to be able to use a much wider range of integers. This is what C++ was built to do, namely, create powerful new data types. Consider class HugeInt of Figs. 10.17 – 10.19. Study the class carefully, then overload the relational operators, the * multiplication operation, and the / division operator.

[Note: We do not show an assignment operator or copy constructor for class HugeInt, because the assignment operator and copy constructor provided by the compiler are capable of copying the entire array data member properly.]

5. Sample Class Definition

```
#ifndef HUGEINT_H
#define HUGEINT_H
#include <iostream>
using std::ostream;
class HugeInt
{
    friend ostream &operator<<( ostream &, const HugeInt & );
public:
    HugeInt( long = 0 ); // conversion/default constructor
    HugeInt( const char * ); // conversion constructor
    // addition operator; HugeInt + HugeInt
    HugeInt operator+( const HugeInt & ) const;
    // addition operator; HugeInt + int
    HugeInt operator+( int ) const;
    // addition operator;
    // HugeInt + string that represents large integer value
    HugeInt operator+( const char * ) const;
    bool operator==( const HugeInt & ) const; // equality operator
    bool operator!=( const HugeInt & ) const; // inequality operator
    bool operator<( const HugeInt & ) const; // less than operator
    // less than or equal to operator
    bool operator<=( const HugeInt & ) const;
    bool operator>( const HugeInt & ) const; // greater than operator

    // greater than or equal to operator
    bool operator>=( const HugeInt & ) const;
    HugeInt operator-( const HugeInt & ) const; // subtraction operator
    HugeInt operator*( const HugeInt & ) const; // multiply two HugeInts
    HugeInt operator/( const HugeInt & ) const; // divide two HugeInts
    int getLength() const;
private:
    short integer[ 30 ];
}; // end class HugeInt
#endif
```

6. Sample Output

```
n1 is 7654321
n2 is 7891234
n3 is 999999999999999999999999999
n4 is 1
n5 is 12341234
n6 is 7888
result is 0

n1 is not equal to n2
n1 is less than n2
n1 is less than or equal to n2
7654321 + 7891234 = 15545555

999999999999999999999999999 + 1
= 1000000000000000000000000000

7654321 + 9 = 7654330
7891234 + 10000 = 7901234
12341234 * 7888 = 97347653792
12341234 - 7888 = 12333346
12341234 / 7888 = 1564
```

EX3: Rational Number Class

1. Description of the Problem

Create a class RationalNumer(fractions) with these capabilities:

a) Create a constructor that prevents a 0 denominator in a fraction. Reduces or simplifies fractions that are not in reduced form and avoids negative denominators.

b) Overload the addition, subtraction, multiplication and division operators for this class.

c) Overload the relational and equality operators for this class.

2.Sample Class Definition

```
// Exercise 11.15 Solution: RationalNumber.h
// RationalNumber class definition.
#ifndef RATIONAL_NUMBER_H
#define RATIONAL_NUMBER_H

class RationalNumber
{
public:
   RationalNumber( int = 0, int = 1 ); // default constructor
   RationalNumber operator+( const RationalNumber& ); // addition
   RationalNumber operator-( const RationalNumber& ); // subtraction
   RationalNumber operator*( const RationalNumber& ); // multiplication
   RationalNumber operator/( RationalNumber& ); // division
```

```
    // relational operators
    bool operator>( const RationalNumber& ) const;
    bool operator<( const RationalNumber& ) const;
    bool operator>=( const RationalNumber& ) const;
    bool operator<=( const RationalNumber& ) const;

    // equality operators
    bool operator==( const RationalNumber& ) const;
    bool operator!=( const RationalNumber& ) const;
    void printRational() const; // display rational number
private:
    int numerator; // private variable numerator
    int denominator; // private variable denominator
    void reduction(); // function for fraction reduction
}; // end class RationalNumber

#endif
```

3.Sample main function

```
#include <iostream>
using std::cout;
using std::endl;

#include "RationalNumber.h"

int main()
{
    RationalNumber c( 7, 3 ), d( 3, 9 ), x;
    c.printRational();
    cout << " + " ;
    d.printRational();
    cout << " = ";
    x = c + d; // test overloaded operators + and =
    x.printRational();
    cout << '\n';
    c.printRational();
    cout << " - " ;
    d.printRational();
    cout << " = ";
    x = c - d; // test overloaded operators - and =
    x.printRational();
```

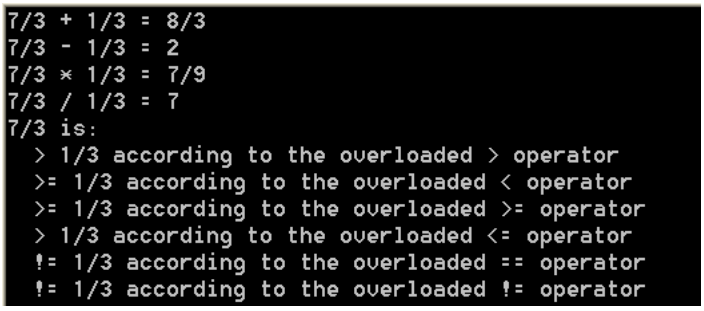
```
cout << '\n';
c.printRational();
cout << " * " ;
d.printRational();
cout << " = ";
x = c * d; // test overloaded operators * and =
x.printRational();

cout << '\n';
c.printRational();
cout << " / " ;
d.printRational();
cout << " = ";
x = c / d; // test overloaded operators / and =
x.printRational();

cout << '\n';
c.printRational();
cout << " is:\n";
// test overloaded greater than operator
cout << ( ( c > d ) ? "    > " : "    <= " );
d.printRational();
cout << " according to the overloaded > operator\n";
// test overloaded less than operator
cout << ( ( c < d ) ? "    < " : "    >= " );
d.printRational();
cout << " according to the overloaded < operator\n";
// test overloaded greater than or equal to operator
cout << ( ( c >= d ) ? "    >= " : "    < " );
d.printRational();
cout << " according to the overloaded >= operator\n";
// test overloaded less than or equal to operator
cout << ( ( c <= d ) ? "    <= " : "    > " );
d.printRational();
cout << " according to the overloaded <= operator\n";
// test overloaded equality operator
cout << ( ( c == d ) ? "    == " : "    != " );
d.printRational();
cout << " according to the overloaded == operator\n";
// test overloaded inequality operator
cout << ( ( c != d ) ? "    != " : "    == " );
d.printRational();
cout << " according to the overloaded != operator" << endl;
return 0;
```

} // end main

4.Sample Output



EX4: String Concatenation

1.Description of the Problem

String concatenation requires two operands—the two strings that are to be concatenated. In the text, we showed how to implement an overloaded concatenation operator that concatenates the second String object to the right of the first String object, thus modifying the first String object. In some applications, it is desirable to produce a concatenated String object without modifying the String arguments. Implement operator+ to allow operations
such as

string1 = string2 + string3; in which neither operand is modified.

2.Sample Class Definition

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

class String
{
    friend ostream &operator<<(ostream &output, const String &s);
public:
    String(const char * const = ""); // conversion constructor
    String(const String &); // copy constructor
    ~String(); // destructor
    const String &operator=(const String &);
    String operator+(const String &);
private:
    char *sPtr; // pointer to start of string
    int length; // string length
}; // end class String
```

```
#endif
```

7. Sample Main Function

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

int main()
{
    String string1, string2("The date is");
    String string3(" August 1, 1993");

    // test overloaded operators
    cout << "string1 = string2 + string3\n";
    string1 = string2 + string3; // tests overloaded = and + operator
    cout << "\"" << string1 << "\" = \"" << string2 << "\" + \""
        << string3 << "\"" << endl;
    return 0;

}
```

5.Sample Output

```
string1 = string2 + string3
"The date is August 1, 1993" = "The date is" + " August 1, 1993"
```

Lab3 Object-Oriented Programming: Inheritance

Objectives

1. To create classes by inheriting from exiting classes.
2. The notations of base classes and derived classes and the relationships between them.
3. The order in which objects were constructed and destructed in inheritance hierarchies.
4. The initial in heritance.
5. The difference between public, protected and private member access specifier.
6. The difference between public, protected and private inheritance.
7. The inheritance, add and hide of class member functions.
8. The translation between base class and derived class.

Experiments

Ex 1:(The construction and destroying of objects in heritance)

1) To create a base class as following:

```
class MyBase1 {
public:
    MyBase1(){ cout << "…BaseClass1 Object is created!"<< end; }
    ~MyBase1(){ cout << "…BaseClass1 Object is destroyed!"<< end; }
}
```

2) To create a derived class from MyBase1 with public inheritance and analyze the result.

```
class Myderived1 : public MyBase1 {
public:
    MyDerived1()
    { cout << "…First layer derived Object is created!"<< end; }
    ~MyDerived1()
    { cout << "…First layer derived Object is Destroyed!"<< end; }
}
class Myderived11 : public MyDerived1 {
public:
    MyDerived11()
```

```
    { cout << "…Second layer derived Object is created!"<< end; }
    ~MyDerived11()
    { cout << "…Second layer derived Object is destroyed!"<< end; }
}
int main()
{
    MyBase1 a;
    MyDerived1 b;
    MyDerived11 c;
}
```

3) To create a base class as following:

```
class MyBase2 {
    MyBase1 a1;
public:
    MyBase2()
    { cout << "…BaseClass2 Object is created!"<< end; }
    ~ MyBase2()
    { cout << "…BaseClass2 Object is destroyed!"<< end; }
}
```

4) To create a derived class from MyBase2 with public inheritance and analyze the result.

```
class Myderived1 : public MyBase2 {
    MyBase1 a1;
public:
    MyDerived1()
    { cout << "…First layer derived Object is created!"<< end; }
    ~MyDerived1()
    { cout << "…First layer derived Object is Destroyed!"<< end; }
}
class Myderived11 : public MyDerived1 {
public:
    MyDerived11()
    { cout << "…Second layer derived Object is created!"<< end; }
    ~MyDerived11()
    { cout << "…Second layer derived Object is destroyed!"<< end; }
}
int main()
{
    MyBase2 a;
    MyDerived1 b;
    MyDerived11 c;
}
```

Ex 2: The initial of objects in heritance

1) To create two classes as following and analyze the result

```
class MyBase31 {
    int a, b, c;
public:
    MyBase31(int x, int y, int z) :a(x), b(y), c(z)
    {
        cout << "…BaseClass31 Object is created!"<< end;
        cout << a << " " << b << " " << c << endl;
    }
    ~ MyBase31(){ cout << "…BaseClass31 Object is destroyed!"<< end; }
}
class MyBase32 {
    int a, b, c;
public:
    MyBase32(int x, int y, int z)
    {
        cout << "…BaseClass32 Object is created!"<< end;
        cout << a << " " << b << " " << c << endl;
        a=x, b=y, c=z;
        cout << a << " " << b << " " << c << endl;
    }
    ~ MyBase32(){ cout << "…BaseClass32 Object is destroyed!"<< end; }
}
int main()
{
    MyBase31 a(1,2,3);
    MyBase32 b(4,5,6);
}
```

2) To create some derived classes as following and analyze the result

```
class MyDerived1 : public MyBase31 {
    MyBase31 a(5,6,7);
    int c;
public:
    MyDerived1(int x) : c(x), MyBase31(x,8,9)
    {
        cout << "…Base Object has been created!" << endl;
        cout << "…Member Object has been created! " << a.x << " " << a.y << " "
<< a.z << endl;
```

```
        cout << "…Derived Object is created! "<< c << endl;
    }
}
int main()
{
    MyDerived1 b(88);
}
```

Ex 3: The access properties in inheritance

1) To create a base class as following:

```
class MyBase3 {
    int x;
    fun1() { cout << "MyBase3---fun1()" << endl; }
protcted:
    int y;
    fun2() { cout << "MyBase3---fun2()" << endl; }
public:
    int z;
    MyBase(int a, int b, int c) {x=a; y=b; z=c;}
    int getX(){cout << "MyBase3---x:" << endl; return x;}
    int getY(){cout << "MyBase3---y:" << endl; return y;}
    int getZ(){cout << "MyBase3---z:" << endl; return z;}
    fun3() { cout << "MyBase3---fun3()" << endl; }
}
```

2) To create a derived classes from MyBase3 with public inheritance and analyze the result.

```
class MyDerived1 : public MyBase3 {
    int p;
public:
    MyDerived1(int a) : p(a)
        int getP(){cout << "MyDerived---p:" << endl; return p;}
    int disply()
    {
        cout << p << " " << x << " " << y << " " << z << " " << endl
            << fun1( ) << endl << fun2() << endl << fun3() << endl;
    }
}
int main()
{
    MyDerived1 a(3);
    a.disply();
```

```
    cout << a.x << " " << a.p << " " << a.y << " " << a.z << endl;
    cout << a.getX() << " " << a.getP() << " " << a.getY() << " " << a.getZ() <<
endl;
}
```

3) To create a derived classes from MyBase3 with private inheritance and analyze the result.

```
class MyDerived2 : private MyBase3 {
    int p;
public:
    MyDerived2(int a) : p(a)
        int getP(){cout << "MyDerived---p:" << endl; return p;}
    int disply()
    {
        cout << p << " " << x << " " << y << " " << z << " " << endl
            << fun1( ) << endl << fun2() << endl << fun3() << endl;
    }
}
class MyDerived21 : public MyDerived3 {
    int p;
public:
    MyDerived21(int a) : p(a)
        int getP(){cout << "MyDerived21---p:" << endl; return p;}
    int disply1()
    {
        cout << p << " " << x << " " << y << " " << z << " " << endl;
    }

}
int main()
{
    MyDerived2 a(3);
    MyDerived21 b(6);
    a.disply();
    cout << a.x << " " << a.p << " " << a.y << " " << a.z << endl;
    cout << a.getX() << " " << a.getP() << " " << a.getY() << " " << a.getZ() <<
endl;
    b.disply1();
}
```

4) To create a derived classes from MyBase3 with protected inheritance and analyze the result.

```
class MyDerived3 : protected MyBase3 {
    int p;
```

```
public:
    MyDerived3(int a) : p(a)
        int getP(){cout << "MyDerived---p:" << endl; return p;}
    int disply()
    {
        cout << p << " " << x << " " << y << " " << z << " " << endl
            << fun1( ) << endl << fun2() << endl << fun3() << endl;
    }
}
class MyDerived31 : public MyDerived3 {
    int p;
public:
    MyDerived31(int a) : p(a)
        int getP(){cout << "MyDerived31---p:" << endl; return p;}
    int disply1()
    {
        cout << p << " " << x << " " << y << " " << z << " " << endl;
    }
}
int main()
{
    MyDerived3 a(3);
    MyDerived31 b(6);
    a.disply();
    cout << a.x << " " << a.p << " " << a.y << " " << a.z << endl;
    cout << a.getX() << " " << a.getP() << " " << a.getY() << " " << a.getZ() <<
endl;
    b.disply1();
}
```

5) To analyze the result

```
class MyBase {
public:
    void f1(){ cout << "…MyBase f1-----!" << endl; }
    void f2(){ cout << "…MyBase f2-----!" << endl; }
}
class MyDerived : public MyBase {
public:
    void f2(){ cout << "…MyDerived f2-----!" << endl; }
    void f22(){ MyBase::f2(); cout << "…MyDerived f2-----!" << endl; }
    void f3(){ cout << "…MyDerived f3-----!" << endl; }
}
int main()
```

```
{
    MyDerived a;
    a.f1(); a.f2(); a.f3(); a.f22();
}
```

Ex 4: The translation between base class and derived class.

1)To create a base class as following:

```
class MyBase {
int x;
public:
MyBase(int a):x(a);
int getX(){ cout << “” << endl; return x; }
}
```

2)To create a derived class as following:

```
class MyDerived : public MyBase {
int y;
public:
MyDerived(int a):y(a),MyBase(a+4);
int getY(){ cout << “” << endl; return Y; }
}
```

3)To create a test program as following and analyze the result.

```
int main()
{
MyBase a(2), *p = a,;
MyDerived b(4), *q=b;
MyBase &c = a;
MyBase &d = b;
cout << a.getX() << “ ” << p-> getX() << endl;
cout << b.getY() << “ ” << q-> getY() << b.getX() << “ ” << q-> getX() << endl;
a = b;
cout << a.getX() << “ “ << a.getY() << endl;
p = q;
cout << p->getX() << “ “ << p->getY() << endl;
cout << c.getX() << “ “ << d.getX() << “ “ << d.getY() << endl;
b = a;
cout << b.getX() << “ ” << b.getY() << endl;
}
```

Ex 5: Construction and Composition

Implement Date class and FinalTest class, and make the main function output correctly. All data members should be private.

Tips:

(1) Data validation is not required.
(2) It is only required to implement the necessary member functions.
(3) Interface and implementation are not necessarily separated.

int main()
{
FinalTest item1("C++ Test", Date(2014,6,2));
item1.print();
FinalTest item2("Java");
item2.print();
item2.setDue(Date(2014,6,10));
item2.print();
}
	Title: C++ Test
	Test Date: 2014-6-2
	Title: Java
	Test Date: 2014-1-1
	Title: Java
	Test Date: 2014-6-10

Ex 6: Inheritance, Constructor and Initializer

Design and implement a hierarchical class structure, according to the following requirements.

- Shape is a base class.
- Classes Circle, Triangle and Rectangle are directly inherited from shape.
- Square is directly inherited from Rectangle.
- Provide constructors and destructor for each class.
- Each object must include at least one data member named id (string).
- Objects of derived classes should contain some necessary data members to determine their position and area, such as centerofcircle(circle), lefttop(circle), rightbottom(circle), radius etc.
- Objects of class Square have one special method named incircle. This method can create and return the inscribed circle obejcet(circle) of the corresponding Square object.
- Each object provides area()function to calculate the area of an shape object and print()function to display all information of an object such as radius, width, length, area and incircle.
- Use Initializers to initialize data members of base class and composition objects.

Lab4 Object-Oriented Programming: Polymorphism

Objectives

1. What polymorphism is, how it makes programming more convenient, and how it makes systems more extensible and maintainable.
2. To declare and use virtual functions to effect polymorphism.
3. The distinction between abstract and concrete classes.
4. To declare pure virtual functions to create abstract classes.
5. How C++ implements virtual functions and dynamic binding “under the hood.”
6. How to use virtual destructors to ensure that all appropriate destructors run on an object.

Experiments

Ex 1:(12.12, Payroll System Modification)

1. Description of the Problem

(Payroll System Modification) Modify the payroll system of Figs. 12.9-12.17 to include private data member birthDate in class Employee. Use class Date from Figs. 10.6-10.7 to represent an employee's birthday. Assume that payroll is processed once per month. Create a vector of Employee references to store the various employee objects. In a loop, calculate the payroll for each Employee (polymorphically), and add a $100.00 bonus to the person's payroll amount if the current month is the month in which the Employee's birthday occurs.

2. Problem-Solving Tips

1) Tips: how to get system time

Method 1:

```
#include <windows.h>
int main()
{
  SYSTEMTIME systm;
  GetLocalTime(&systm);
```

```
 cout<<systm.wYear<<"-"<<systm.wMonth<<"-"<<systm.wDay<<" "<<
systm.wHour<<":"<<systm.wMinute<<":"<<systm.wSecond;
 return 0;
}
```

Method 2:

```
#include <iostream>
#include <ctime>
using namespace std;
int main()
{
   time_t nowtime;
   struct tm* ptm;
   time(&nowtime);
   ptm = localtime(&nowtime);
   cout<<ptm->tm_year + 1900<<"-"<<ptm->tm_mon + 1
   <<"-"<<ptm->tm_mday<<" "<<ptm->tm_hour<<":"
   <<ptm->tm_min<<":"<<ptm->tm_sec;
   return 0;
}
```

2) main function

Modify the main functions of Figs 12.17, 12.19.

3. Results example

```
Employees processed polymorphically via dynamic binding:

salaried employee: John Smith
birthday: June 15, 1944
social security number: 111-11-1111
weekly salary: 800.00
earned $800.00

hourly employee: Karen Price
birthday: April 29, 1960
social security number: 222-22-2222
hourly wage: 16.75; hours worked: 40.00
HAPPY BIRTHDAY!
earned $770.00

commission employee: Sue Jones
birthday: September 8, 1954
social security number: 333-33-3333
gross sales: 10000.00; commission rate: 0.06
earned $600.00

base-salaried commission employee: Bob Lewis
birthday: March 2, 1965
social security number: 444-44-4444
gross sales: 5000.00; commission rate: 0.04; base salary: 300.00
earned $500.00

deleting object of class SalariedEmployee
deleting object of class HourlyEmployee
deleting object of class CommissionEmployee
```

Ex 2:

1. Description of the Problem

(Shape Hierarchy) Implement the Shape hierarchy designed in Exercise 11.7 (which is based on the hierarchy in Fig. 11.3).

Each TwoDimensionalShape should contain function getArea to calculate the area of the two-dimensional shape. Each ThreeDimensionalShape should have member functions getArea and getVolume to calculate the surface area and volume of the three-dimensional shape, respectively.

Create a program that uses a vector of Shape pointers to objects of each concrete class in the hierarchy. The program should print the object to which each vector element points. Also, in the loop that processes all the shapes in the vector, determine whether each shape is a TwoDimensionalShape or a ThreeDimensionalShape. If a shape is a TwoDimensionalShape, display its area. If a shape is a ThreeDimensionalShape, display its area and volume.

2. Requirements

(1) Define at least 5 classes.

(2) Define member functions for each class, the following functions must be included: constructor, destructor, virtual getArea function

(3) main function:

vector declaration

test the getArea function

Ex 3:(12.14 Polymorphic Banking Program Using Account Hierarchy)

(Polymorphic Banking Program Using Account Hierarchy) Develop a polymorphic banking program using the Account hierarchy created in Exercise 11.10.

Create a vector of Account pointers to SavingsAccount and CheckingAccount objects. For each Account in the vector, allow the user to specify an amount of money to withdraw from the Account using member function debit and an amount of money to deposit into the Account using member function credit.

As you process each Account, determine its type. If an Account is a SavingsAccount, calculate the amount of interest owed to the Account using member function calculateInterest, then add the interest to the account balance using member function credit.

After processing an Account, print the updated account balance obtained by invoking baseclass member function getBalance.

Lab5 File processing

Objectives

To create, read and write sequential/random access files

Experiments

Ex 1:

Suppose we want to simulate accessing a website from a variety of clients. This problem consists of two parts.

1)Part I: Randomly generate an IPv4 address (0.0.0.0 ~ 255.255.255.255) and a time-stamp (YYYY-MM-DD HH:MM:SS). Print all records in a file with each individual record occupying a single line. For example:

Contents of record.txt

```
192.168.2.1 2020-02-02 19:11:20
202.182.22.23 2020-02-12 20:22:13
...
```

Note that your time-stamp should be in ascending order. Generate at least 100 records.

2)Part II: Read the record.txt file you created in part I, take the statistics and print on the screen how many addresses each class has:

```
Number of class A addresses: 12
Number of class B addresses: 15
Number of class C addresses: 23
Number of class D addresses: 28
Number of class E addresses: 22
```

Roughly, you can classify each address according to the following rules:

Class A: 0.0.0.0 – 127.255.255.255
Class B: 128.0.0.0 – 191.255.255.255
Class C: 192.0.0.0 – 223.255.255.255
Class D: 224.0.0.0- 239.255.255.255
Class E: 240.0.0.0 – 247.255.255.255

Ex 2:

Create a simple random-access file-processing program that might be used by professors to help manage their student records. For each student, the program should obtain an ID number, the student's first name, the student's last name and the student's grade. The data obtained for each student constitutes a record for the student and should be stored in an object of a class called Student. The program should save the records in a binary file specified by the user (for example "file.dat").

The program should also be able to:

(1) Display all records (together with students' average score)

(2) Add/delete the record

(3) Edit each record (i.e. change the ID, name and/or grade of each record)

Lab6 Standard Library Containers and Iterators

Objectives：

1. Use the vector, list and deque sequence containers.
2. Use the set, multiset, map and multimap associative containers.
3. Use the stack, queue and priority_queue container adapters.
4. Use iterators to access container elements.
5. Use the copy algorithm and ostream_iterators to output a container.
6. Use the bitset “near container” to implement the Sieve of Eratosthenes for determining prime numbers.

Experiments

EX1:（15.23 palindrome）

1.Description of the Problem

Write a function template palindrome that takes a vector parameter and returns true or false according to whether the vector does or does not read the same forward as backward (e.g., a vector containing 1, 2, 3, 2, 1 is a palindrome, but a vector containing 1, 2, 3, 4 is not).

2.Experimental Results

```
75 74 73 72 71 70 69 68 67 66 65  is not a palindrome
K J I H G F G H I J K  is a palindrome
请按任意键继续. . .
```

EX2: (15.25-15.26 Sieve of Eratosthenes)

(The Sieve of Eratosthenes) A prime integer is any integer that is evenly divisible only by itself and 1. The Sieve of Eratosthenes is a method of finding prime numbers. It operates as follows:

a. Create an array with all elements initialized to 1 (true). Array elements with prime subscripts will remain 1. All other array elements will eventually be set to zero. You will ignore elements 0 and 1 in this exercise.
b. Starting with array subscript 2, every time an array element is found whose value is 1, loop through the remainder of the array and set to zero every element whose subscript is a multiple of the subscript for the element with value 1. For array subscript 2, all elements beyond 2 in the array that are multiples of 2 will be set to zero (subscripts 4, 6, 8, 10, etc.); for array

subscript 3, all elements beyond 3 in the array that are multiples of 3 will be set to zero (subscripts 6, 9, 12, 15, etc.); and so on.

When this process is complete, the array elements that are still set to one indicate that the subscript is a prime number. These subscripts can then be printed. Write a program that uses an array of 1000 elements to determine and print the prime numbers between 2 and 999. Ignore element 0 of the array.

1. Description of the Problem

15.25 (Sieve of Eratosthenes) Modify Exercise 15.24, the Sieve of Eratosthenes, so that, if the number the user inputs into the program is not prime, the program displays the prime factors of the number. Remember that a prime number's factors are only 1 and the prime number itself. Every nonprime number has a unique prime factorization. For example, the factors of 54 are 2, 3, 3 and 3. When these values are multiplied together, the result is 54. For the number 54, the prime factors output should be 2 and 3.

15.26 (Prime Factors) Modify Exercise 15.25 so that, if the number the user inputs into the program is not prime, the program displays the prime factors of the number and the number of times each prime factor appears in the unique prime factorization. For example, the output for the number 54 should be：

The unique prime factorization of 54 is: 2 * 3 * 3 * 3

2.Experimental Results

```
The prime numbers in the range 2 to 1023 are:
    2    3    5    7   11   13   17   19   23   29   31   37
   41   43   47   53   59   61   67   71   73   79   83   89
   97  101  103  107  109  113  127  131  137  139  149  151
  157  163  167  173  179  181  191  193  197  199  211  223
  227  229  233  239  241  251  257  263  269  271  277  281
  283  293  307  311  313  317  331  337  347  349  353  359
  367  373  379  383  389  397  401  409  419  421  431  433
  439  443  449  457  461  463  467  479  487  491  499  503
  509  521  523  541  547  557  563  569  571  577  587  593
  599  601  607  613  617  619  631  641  643  647  653  659
  661  673  677  683  691  701  709  719  727  733  739  743
  751  757  761  769  773  787  797  809  811  821  823  827
  829  839  853  857  859  863  877  881  883  887  907  911
  919  929  937  941  947  953  967  971  977  983  991  997
 1009 1013 1019 1021

Enter a value from 1 to 1023 (-1 to end): 331
331 is a prime number

Enter a value from 2 to 1023 (-1 to end): 659
659 is a prime number

Enter a value from 2 to 1023 (-1 to end): 688
688 is not a prime number
The unique prime factorization of 688 is: 2 * 2 * 2 * 2 * 43

Enter a value from 2 to 1023 (-1 to end):
```

Lab7 Exception Handling

Objectives

1. What exceptions are and when to use them.
2. To use try, catch and throw to detect, handle and indicate exceptions, respectively.
3. To process uncaught and unexpected exceptions.
4. To declare new exception classes.
5. How stack unwinding enables exceptions not caught in one scope to be caught in another scope.
6. To handle new failures.
7. To understand the standard exception hierarchy.

Experiments

Ex 1：（习题 17.21，异常处理的逻辑流程）

1. Description of the Problem

Suppose a program throws an exception and the appropriate exception handler begins executing. Now suppose that the exception handler itself `throws` the same type of exception. Does this create infinite recursion? Write a program to check your observation.

2. Problem-Solving Tips

a) 定义一个 runtime_error 派生类

class TestException : public runtime_error{}

b) main 函数

参考教材的 main 函数 17.2，在 try 语句块中抛出异常，并且在异常处理部分重新抛出该异常。

3. Results example

```
This is a test
abnormal program termination
```

Ex 2：（习题 17.25 构造函数、析构函数和异常处理）

1. Description of the Problem

Write a program illustrating that member object destructors are called for only those member objects that were constructed before an exception occurred.

2. Problem-Solving Tips

a) 定义类 Item，并包含整型成员变量 value，并在 Item 的构造函数中定义条件判断语句以抛出异常，例如：

if (value == 3) throw runtime_error("An exception was thrown");

b) main 函数

main 函数中构建若干 Item 对象，并在合适位置打印测试语句。

3. Results example

```
Constructing an object of class ItemGroup
Item 1 constructor called
Item 2 constructor called
Item 3 constructor called
Item 2 destructor called
Item 1 destructor called
An exception was thrown
```

Ex 3：（习题 17.29 重新抛出异常）

1. Description of the Problem

Write a program that illustrates rethrowing an exception.

2. Problem-Solving Tips

a) 定义 runtime_error 的派生类 TestException

class TestException : public runtime_error{…};

b) 定义一个函数 g()，其中 try 语句块中抛出 TestException 异常，在**可以处理任何类型异常**的 catch 语句块部分打印并重新抛出异常。

c) main 函数

在 main 函数中的 try 语句块部分调用 g()函数，并在 catch 语句块中打印。

3. Results example

```
Exception caught in function g(). Rethrowing...
Exception caught in function main()
```

Ex 4：（习题 17.31 堆栈展开）

1. Description of the Problem

Write a program that throws an exception from a deeply nested function and still has the catch handler following the try block enclosing the call chain catch the exception.

2. Problem-Solving Tips

a) 定义 runtime_error 的派生类 TestException

class TestException : public runtime_error{…};

b) 定义三个函数 f()，g()，h()，并设计相应的嵌套包含关系。

c) main 函数

try 语句块中调用某函数，并在 catch 语句块中调用异常类基类的 what 函数进行打印

3. Results example

```
In main: Caught TestException
```

Lab8 Templates

Objectives

1. To use function templates to conveniently create a group of related.
2. To distinguish between function templates and function – template specializations, class templates and class-template specializations.

Experiments

Ex 1:

1. Description of the Problem

Use an int template nontype parameter numberOfElements and a type parameter elementType to help create a template for the Array class. This template will enable Array objects to be instantiated with a specified number of elements of a specified element type at compile time.

2. Problem-Solving Tips

```
Enter 5 integer values:
1 2 3 4 5
The values in the intArray are:
1 2 3 4 5
Enter 7 one-word string values:
red blue yellow black pink purple green
The values in the stringArray are:
red blue yellow black pink purple green
```

Ex 2:

Write a simple function template for predict function isEqualTo that compares its two arguments of the same type with the equality operator (==) and returns true if they are equal and false if they are not equal.

Use this function template in a program that calls isEqualTo only with a variety of built-in types.

Now write a separate version of the program that calls isEqualTo with a user-defined class type Complex, but does not overload the equality operator. What happens when you attempt to run this program?

Now overload the equality operator (with the operator function) operator ==. Now what happens when you attempt to run this program?

Ex 3:

Define a class template called Vector(a single-column- Matrix). The templates can instantiate a vector of any element type. Overloaded >> and << operators: to enable input and output of a vector, respectively.

Lab9 Comprehensive Program Design

Objectives

1. Each team of up to 4 students is required to complete a project using the knowledge and techniques covered in lectures of this course.
2. This is to provide each student with an opportunity to apply the knowledge, skills and techniques learned in the course to the real-world problems.

The report should contain the following:

Cover Sheet (project title, and student name and contribution)
b) Summary (a brief description of the project and the outcome)
c) Procedures (steps and methods used in completing the project)
d) Results (detailed lab outcome such as screenshots and calculated results)
e) Discussions (discussing any issues and problems in completing the project or any suggestions)
f) Conclusion (any lesson you learned from this project)
g) Resources (any references or sources of code or data used in completing the project)

Project Grade:

Activity	Points%
Topic	10
Demo	30
Individual Contribution	30
Presentation	30

Topic selection reference: (The following topic is for reference only, you can choose your own topic.)

Problem 1：Bank Account Management System
Please use the object-oriented programming techniques learned in this course, such as inheritance, derivation, polymorphism, file processing, etc., to develop a bank account management system.
The bank account management system includes two types of accounts: users and bank employees.
Users and bank employees log in through a simple text menu interface and perform corresponding operations.
中文：
请综合利用本课程中学习的面向对象编程相关技术，如继承、派生、多态、文件处理、容器等。实现一个银行账户管理系统。银行账户管理系统中包括用户及银行职员两类人员。用户及银行职员通过简易的文字菜单式系统界面登录后进行相应操作。

Problem 2: Hospital information system

Construct a hospital information system to store the information of hospital, department and doctors. The system can add, edit, search and delete doctors.

医院信息系统

建立一个医院信息系统，存储医院、科室、医生等信息。可以实现增加、编辑、查询与删除医生信息

Problem 3： Poker game

Construct a deck of poker cards. The cards can be shuffled any number of times, and can be distributed to 4 people in sequence.

Finding the patterns of the cards, including flush, sister-pairs and etc.

You can refer to the existing rules of poker game or personal designed rules for the playing strategy.

中文：

扑克牌游戏

创建一副扑克牌，可以任意次数的洗牌，依次分发给 4 个人并显示。

查找牌的模式：如，同花顺、姊妹对等

可参照已有的玩牌游戏规则（或自定义规则），给出出牌策略等

Problem 4：Keywords Processing

Find keywords from the English documents and obtain the frequency. We can also replace the keyword with other word.

中文：

英文文档中关键字词频统计、查找、替换

Problem 5： Puzzle Game

中文：

填单词游戏 puzzle

Problem 6：Document Comparison Tool

(1)Compare the contents of the two files, display the differences in a high brightness way, and quickly locate the differences so that users can quickly find out.

(2)For the differences between the two files, bidirectional and selective content coverage is supported.

(3)Support the comparison of files in two folders, including file name, size, etc.

The function of the software is similar to WinMerge, which mainly involves file / folder management, file reading and writing programming, and requires consideration of user interface design friendliness, software fault tolerance and other issues.

中文：

1） 将两个文件的内容做比对，在相异之处以高亮度的方式显示，并能快速定位

相异之处，让使用者可以很快的查知；
2） 对于两个文件的相异之处，支持双向的、选择性的内容覆盖；
3） 支持两个文件夹中文件的比较，包括文件名、大小等。
该软件功能类似于 WinMerge，主要涉及文件/文件夹管理、文件读写方面的编程，要求考虑用户界面设计的友好性、软件容错性等问题。

Problem 7：Maze game

When the program starts running, it displays a maze map with a mouse in the center of the maze and a granary at the bottom right of the maze. The task of the game is to use the direction keys on the keyboard to manipulate the mouse to walk to the granary within the stipulated time.

迷宫问题

程序开始运行时显示一个迷宫地图，迷宫中央有一只老鼠，迷宫的右下方有一个粮仓。游戏的任务是使用键盘上的方向键操纵老鼠在规定的时间内走到粮仓处。