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

2012 年 2 月

目　　录

实验作业及课后习题

Lab1 Classes：A Deeper Look， PartI

OBJECTIVES :

1．掌握类的定义和操作（包括常成员函数）
2．掌握对常量对象的访问
3．掌握使用友元访问对象的私有数据成员
4．掌握静态数据成员和成员函数

Experiments

作业一（习题 9.5，复数类）

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.

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

2. 结果示例

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

作业二（习题 9.7，增强的 Time 类）

1. Description of the Problem

(Enhancing Class Time) Modify the ***Time*** class of Figs. 9.89.9 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:

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

注意以下额外要求：

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. 结果示例

```
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
```

作业三（习题 9.14，大整数类）

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*** 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. HugeInteger 类定义

```
#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. 结果示例

```
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
```

HOMEWORK:

9.3 9.4

Lab2 Classes:A DeeperLook,Part 2

OBJECTIVES :

In this chapter you willl learn:

- To specify const (constant) objects and constmember functions.
- To create objects composed of other objects.
- To use friend functions and friend classes.
- To use the this pointer.
- To create and destroy objects dynamically with operatorsnew and delete, respectively.
- To use static data members and member functions.
- The concept of a container class.
- The notion of iterator classes that walk through theelements of container classes.
- To use proxy classes to hide implementation details froma class’s clients.

Experiments

EX1:Simple Calculator

This problem is intended to be solved in a closed-lab session with a teaching assistant or instructor present. Theproblem is divided into six parts:

1. Lab Objectives
2. Description of the Problem
3. Sample Output
4. Program Template (Fig. L 10.1 – Fig. L 10.3)
5. Problem-Solving Tip
6. Follow-Up Questions and Activities

The program template represents a complete working C++ program, with one or more key lines of code replacedwith comments. Read the problem description and examine the sample output; then study the template code.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 10 of C++ How To Program: Sixth Edition.In this lab, you will practice:

- Using classes to create a data type SimpleCalculator capable of performing arithmetic operations.
- Creating const member functions to enforce the principle of least privilege.

The follow-up questions and activities also will give you practice:

- Using constructors to specify initial values for data members of a programmer-defined class.

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.

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
```

Template

```
// Lab Exercise 1: SimpleCalculator.h

// class SimpleCalculator definition
class SimpleCalculator
{
public:
   /* Write prototype for add member function */
   double subtract( double, double ) const;
   double multiply( double, double ) const;
   /* Write prototype for divide member function */

}; // end class SimpleCalculator
```

Fig. L 10.1 | Contents of `SimpleCalculator.h`.

```
// Lab Exercise 1: SimpleCalculator.cpp

#include "SimpleCalculator.h"

/* Write definition for add member function */

// function subtract definition
double SimpleCalculator::subtract( double a, double b ) const
{
   return a - b;

} // end function subtract

// function multiply definition
double SimpleCalculator::multiply( double a, double b ) const
{
   return a * b;

} // end function multiply

/* Write definition for divide member function */

```

Fig. L 10.2 | Contents of `SimpleCalculator.cpp`.

```
// Lab Exercise 1: CalcTest.cpp

#include <iostream>

using std::cout;
using std::endl;

#include "SimpleCalculator.h"

int main()
{
   double a = 10.0;
   double b = 20.0;
```

Fig. L 10.3 | Contents of `CalcTest.cpp`. (Part 1 of 2.)

```
14
15     /* Instantiate an object of type SimpleCalculator */
16     cout << "The value of a is: " << a << "\n"
17          << "The value of b is: " << b << "\n\n";
18
19     /* Write a line that adds a and b through your SimpleCalculator
20        object; assign the result to a variable named addition */
21     cout << "Adding a and b yields " << addition << "\n";
22
23     double subtraction = sc.subtract( a, b );
24     cout << "Subtracting b from a yields" << subtraction << "\n";
25
26     double multiplication = sc.multiply( a, b );
27     cout << "Multiplying a by b yields " << multiplication << "\n";
28
29     /* Write a line that divides a and b through the
30        SimpleCalculator object; assign the result to a
31        variable named division */
32     cout << "Dividing a by b yields " << division << endl;
33
34     return 0;
35
36  } // end main
```

Fig. L 10.3 | Contents of `CalcTest.cpp`. (Part 2 of 2.)

Problem-Solving Tip

1. All of `SimpleCalculator`'s member functions should have return type `double`.

```
//
```

EX2:Integer Set

This problem is intended to be solved in a closed-lab session with a teaching assistant or instructor present. Theproblem is divided into six parts:

1. Lab Objectives
2. Description of the Problem
3. Sample Output
4. Program Template (Fig. L 10.4 – Fig. L 10.6)
5. Problem-Solving Tips
6. Follow-Up Question and Activity

The program template represents a complete working C++ program, with one or more key lines of code replacedwith comments. Read the problem description and examine the sample output; then study the template code.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 10 of C++ How To Program: Sixth Edition.In this lab, you will practice:

- Using classes to create a data type, IntegerSet, capable of storing a set of integers.
- Using dynamic memory allocation with the new and delete operators.In the follow-up question and activity you also will practice:
- Using destructors to deallocate memory that was dynamically allocated.

Description of the Problem

Create class IntegerSet for which each object can hold integers in the range 0 through 100. A set is representedinternally 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 setwhose 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 bothof 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. Printonly those elements which are present in the set (i.e., their position in the array has a value of 1). Print --- foran 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 thearray to initialize a set object.Now write a driver program to test your IntegerSet class. Instantiate several IntegerSet objects. Test thatall your member functions work properly.

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   }
```

Template

```
1  // Lab 2: IntegerSet.h
2  // Header file for class IntegerSet
3  #ifndef INTEGER_SET_H
4  #define INTEGER_SET_H
5
6  class IntegerSet
7  {
8  public:
9     // default constructor
10    IntegerSet()
```

Fig. L 10.4 | Contents of `integerset.h`. (Part 1 of 2.)

```
11     {
12        /* Write call to emptySet */
13     } // end IntegerSet constructor
14
15     IntegerSet( int [], int ); // constructor that takes an initial set
16     IntegerSet unionOfSets( const IntegerSet& );
17     /* Write a member funcion prototype for intersectionOfSets */
18     void emptySet(); // set all elements of set to 0
19     void inputSet(); // read values from user
20     void insertElement( int );
21     /* Write a member function prototype for deleteElement */
22     void printSet() const
23     /* Write a member function prototype for isEqualTo */
24  private:
25     int set[ 101 ]; // range of 0 - 100
26
27     // determines a valid entry to the set
28     int validEntry( int x ) const
29     {
30        return ( x >= 0 && x <= 100 );
31     } // end function validEntry
32  }; // end class IntegerSet
33
34  #endif
```

Fig. L 10.4 | Contents of `integerset.h`. (Part 2 of 2.)

```
1   // Lab 2: IntegerSet.cpp
2   // Member-function definitions for class IntegerSet.
3   #include <iostream>
4   using std::cout;
5   using std::cin;
6   using std::cerr;
7
8   #include <iomanip>
9   using std::setw;
10
11  /* Write include directive for IntegerSet.h here */
12
13  // constructor creates a set from array of integers
14  IntegerSet::IntegerSet( int array[], int size)
15  {
16     emptySet();
17
18     for ( int i = 0; i < size; i++ )
19        insertElement( array[ i ] );
20  } // end IntegerSet constructor
21
22  /* Write a definition for emptySet */
23
24  // input a set from the user
25  void IntegerSet::inputSet()
26  {
27     int number;
28
```

Fig. L 10.5 | Contents of `integerset.cpp`. (Part 1 of 3.)

```
29     do
30     {
31        cout << "Enter an element (-1 to end): ";
32        cin >> number;
33
34        if ( validEntry( number ) )
35           set[ number ] = 1;
36        else if ( number != -1 )
37           cerr << "Invalid Element\n";
38     } while ( number != -1 ); // end do...while
39
40     cout << "Entry complete\n";
41  } // end function inputSet
42
43  // prints the set to the output stream
44  void IntegerSet::printSet() const
45  {
46     int x = 1;
47     bool empty = true; // assume set is empty
48
49     cout << '{';
50
51     for (int u = 0; u < 101; u++ )
52     {
53        if ( set[ u ] )
54        {
55           cout << setw( 4 ) << u << ( x % 10 == 0 ? "\n" : "" );
56           empty = false; // set is not empty
57           x++;
58        } // end if
59     } // end for
60
61     if ( empty )
62        cout << setw( 4 ) << "---"; // display an empty set
63
64     cout << setw( 4 ) << "}" << '\n';
65  } // end function printSet
```

```
66
67  // returns the union of two sets
68  IntegerSet IntegerSet::unionOfSets( const IntegerSet &r )
69  {
70     IntegerSet temp;
71
72     // if element is in either set, add to temporary set
73     for ( int n = 0; n < 101; n++ )
74        if ( set[ n ] == 1 || r.set[ n ] == 1 )
75           temp.set[ n ] = 1;
76
77     return temp;
78  } // end function unionOfSets
79
80  /* Write definition for intersectionOfSets */
81
82  // insert a new integer into this set
83  void IntegerSet::insertElement( int k )
84  {
```

Fig. L 10.5 | Contents of `integerset.cpp`. (Part 2 of 3.)

```
85     if ( validEntry( k ) )
86        set[ k ] = 1;
87     else
88        cerr << "Invalid insert attempted!\n";
89  } // end function insertElement
90
91  /* Write definition for deleteElement */
92
93  /* Write definition for isEqualTo */
94
95  // determines if two sets are equal
96  bool IntegerSet::isEqualTo( const IntegerSet &r ) const
97  {
98     for ( int v = 0; v < 101; v++ )
99        if ( set[ v ] != r.set[ v ] )
100          return false; // sets are not-equal
101
102    return true; // sets are equal
103 } // end function isEqualTo
```

Fig. L 10.5 | Contents of `integerset.cpp`. (Part 3 of 3.)

```
1  // Lab 2: SetTest.cpp
2  // Driver program for class IntegerSet.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include "IntegerSet.h" // IntegerSet class definition
8
9  int main()
10 {
11    IntegerSet a;
12    IntegerSet b;
13    IntegerSet c;
14    IntegerSet d;
15
16    cout << "Enter set A:\n";
17    a.inputSet();
18    cout << "\nEnter set B:\n";
19    b.inputSet();
20    /* Write call to unionOfSets for object a, passing
21       b as argument and assigning the result to c */
22    /* Write call to intersectionOfSets for object a,
23       passing b as argument and assigning the result to d */
24    cout << "\nUnion of A and B is:\n";
25    c.printSet();
26    cout << "Intersection of A and B is:\n";
27    d.printSet();
28
29    if ( a.isEqualTo( b ) )
30       cout << "Set A is equal to set B\n";
31    else
32       cout << "Set A is not equal to set B\n";
33
```

Fig. L 10.6 | Contents of `SetTest.cpp`. (Part 1 of 2.)

```
34      cout << "\nInserting 77 into set A...\n";
35      a.insertElement( 77 );
36      cout << "Set A is now:\n";
37      a.printSet();
38
39      cout << "\nDeleting 77 from set A...\n";
40      a.deleteElement( 77 );
41      cout << "Set A is now:\n";
42      a.printSet();
43
44      const int arraySize = 10;
45      int intArray[ arraySize ] = { 25, 67, 2, 9, 99, 105, 45, -5, 100, 1 };
46      IntegerSet e( intArray, arraySize );
47
48      cout << "\nSet e is:\n";
49      e.printSet();
50
51      cout << endl;
52
53      return 0;
54   } // end main
```

Fig. L 10.6 | Contents of SetTest.cpp. (Part 2 of 2.)

Problem-Solving Tips

1. Member function intersectionOfSets must return an IntegerSet object. The object that invokes thisfunction 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 addthose 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 functionvalidEntry. If so, the corresponding element in the set array should be set to 0; otherwise, display anerror 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.

Follow-Up Question and Activity

1. Why might it be advantageous for the set array to be allocated dynamically using new [], if the IntegerSetclass were to be used for more general sets?

Dynamically allocating the set array would allow IntegerSets to contain integers outside the 0 through100 range by specifying a specific valid range for each individual IntegerSet object. Dynamically allocating thearray would also save memory space if a particular IntegerSet will only contain a small range of values, say fromthe range 1 through 10.

HOMEWORK:

HW1: (SavingsAccount Class) Create a SavingsAccount class. Use a static data member

annual-InterestRate to store the annual interest rate for each of the savers. Each member of the class contains a private data member savingsBalance indicating the amount the saver currently has ondeposit. Provide member function calculateMonthlyInterest that calculates the monthly interestby multiplying the balance by annualInterestRate divided by 12; this interest should be added tosavingsBalance. Provide a static member function modifyInterestRate that sets the static annualInterestRate to a new value. Write a driver program to test class SavingsAccount. Instantiatetwo different objects of class SavingsAccount, saver1 and saver2, with balances of $2000.00 and$3000.00, respectively. Set the annualInterestRate to 3 percent. Then calculate the monthly interest and print the new balances for each of the savers. Then set the annualInterestRate to 4 percent, calculate the next month's interest and print the new balances for each of the savers.

HW2:(IntegerSet Class) Create class IntegerSet for which each object can hold integers in therange 0 through 100. Represent the set internally as a vector of bool values. Element a[i] is trueif integer iis in the set. Element a[j] is false if integer j is not in the set. The default constructorinitializes a set to the so-called "empty set," i.e., a set for which all elements contain false.Provide member functions for the common set operations. For example, provide a unionOf-Sets member function that creates a third set that is the set-theoretic union of two existing sets(i.e., an element of the result is set to true if that element is true in either or both of the existingsets, and an element of the result is set to false if that element is false in each of the existing sets).Provide an intersectionOfSets member function which creates a third set which is the set-theoretic intersection of two existing sets (i.e., an element of the result is set to false if that element is false in either or both of the existing sets, and an element of the result is set to true if thatelement is true in each of the existing sets).

Provide an insertElement member function that places a new integer k into a set by settinga[k] to true. Provide a deleteElement member function that deletes integer m by setting a[m] tofalse.

Provide a printSet member function that prints a set as a list of numbers separated by spaces.Print only those elements that are present in the set (i.e., their position in the vector has a value oftrue). 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 arrayand uses the array to initialize a set object.Now write a driver program to test your IntegerSet class.Instantiate several IntegerSetobjects. Test that all your member functions work properly.

Project : (required for every student**)**

Emergency Response Class) The North American emergency response service, 9-1-1,

connects callers to a local Public Service Answering Point (PSAP). Traditionally, the PSAP wouldask the caller for identification information—including the caller's address, phone number and thenature of the emergency, then dispatch the appropriate emergency responders (such as the police,an ambulance or the fire department). Enhanced 9-1-1 (or E9-1-1) uses computers and databases todetermine the caller's physical address, directs the call to the nearest PSAP, and displays the caller'sphone number and address to the call taker. Wireless Enhanced 9-1-1 provides call takers with identification information for wireless calls. Rolled out in two phases, the first phase required carriers toprovide the wireless phone number and the location of the cell site or base station transmitting thecall. Thesecondphaserequiredcarriers to provide the location of thecaller(using technologies suchas GPS). To learn more about 9-1-1, visit www.fcc.gov/pshs/services/911-services/Welcome.html and people.howstuffworks.com/9-1-1.htm.

An important part of creating a class is determining the class's attributes (instance variables).For this class design exercise, research 9-1-1 services on the Internet. Then, design a class calledEmergency that might be used in an object-oriented 9-1-1 emergency response system. List theattributes that an object of this class might use to represent the emergency. For example, the classmight include information on who reported the emergency (including their phone number), thelocation of the emergency, the time of the report, the nature of the emergency, the type of responseand the status of the response. The class attributes should completely describe the nature of theproblem and what's happening to resolve that problem.

Lab3 Operator Overloading; String and Array Objects

OBJECTIVES :

In this chapter you will learn:

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

Experiments

EX1: String Concatenation

Lab Objectives

In this lab, you will practice:

- Overloading the + operator to allow String objects to be concatenated.
- Writing function prototypes for overloaded operators.
- Using overloaded operators.

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.

Sample Output

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

Template

```
// Lab 1: String.h
```

Fig. L 11.3 | Contents of String.h.

```
// Header file for class String.
#ifndef STRING_H
#define STRING_H

#include <iostream>
using std::cout;
using std::ostream;

#include <cstring>
#include <cassert>

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 & );
   /* Write a prototype for the operator+ member function */
private:
   char *sPtr; // pointer to start of string
   int length; // string length
}; // end class String

#endif
```

```
// Lab 1: StringCat.cpp
// Demonstrating overloaded + operator that does not modify operands
#include <iostream>
using std::cout;
using std::endl;
```

```

#include "String.h"

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

   // test overloaded operators
   cout << "string1 = string2 + string3\n";
   /* Write a statement to concatenate string2 and string3,
      and assign the result to string1 */
   cout << "\"" << string1 << "\" = \"" << string2 << "\" + \""
      << string3 << "\"" << endl;
   return 0;
} // end main
```

EX2: Huge Integer

Lab Objectives

In this lab, you will practice:

- Overloading arithmetic and comparison operators to enhance a huge integer class, HugeInt.

- Writing function prototypes for overloaded operators.
- Calling overloaded operator functions.

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. 11.8－11.10. Study the class carefully, then overload the relational and equality operators.[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.]

Sample Output

```
n1 is 7654321
n2 is 7891234
n3 is 99999999999999999999999999999
n4 is 1
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

99999999999999999999999999999 + 1
= 100000000000000000000000000000

7654321 + 9 = 7654330
7891234 + 10000 = 7901234
12341234 - 7888 = 12333346
```

Template

```
// Lab 2: Hugeint.h
// HugeInt 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;

   /* Write prototypes for the six relational and equality operators */

   int getLength() const;
private:
   short integer[ 30 ];
}; // end class HugeInt

#endif
```

Fig. L 11.6 | Contents of HugeInt.h.

```
// Lab 2: HugeIntTest.cpp
// HugeInt test program.
#include <iostream>
using std::cout;
using std::endl;

#include "Hugeint.h"

int main()
{
   HugeInt n1( 7654321 );
   HugeInt n2( 7891234 );
   HugeInt n3( "99999999999999999999999999999" );
   HugeInt n4( "1" );
   HugeInt result;

   cout << "n1 is " << n1 << "\nn2 is " << n2
      << "\nn3 is " << n3 << "\nn4 is " << n4
      << "\nresult is " << result << "\n\n";

   // test relational and equality operators
   if ( n1 == n2 )
      cout << "n1 equals n2" << endl;
```

Fig. L 11.8 | Contents of HugeIntTest.cpp. (Part 1 of 2.)

```
24
25     if ( n1 != n2 )
26        cout << "n1 is not equal to n2" << endl;
27
28     if ( n1 < n2 )
29        cout << "n1 is less than n2" << endl;
30
31     if ( n1 <= n2 )
32        cout << "n1 is less than or equal to n2" << endl;
33
34     if ( n1 > n2 )
35        cout << "n1 is greater than n2" << endl;
36
37     if ( n1 >= n2 )
38        cout << "n1 is greater than or equal to n2" << endl;
39
40     result = n1 + n2;
41     cout << n1 << " + " << n2 << " = " << result << "\n\n";
42
43     cout << n3 << " + " << n4 << "\n= " << ( n3 + n4 ) << "\n\n";
44
45     result = n1 + 9;
46     cout << n1 << " + " << 9 << " = " << result << endl;
47
48     result = n2 + "10000";
49     cout << n2 << " + " << "10000" << " = " << result << endl;
50
51     return 0;
52  } // end main
```

Fig. L 11.8 | Contents of `HugeIntTest.cpp`. (Part 2 of 2.)

HOMEWORK:

HW1: (complex number)

Consider class `Complex` shown in Fig. 11.19–Fig. 11.20. The class enables operations on so-called *complex numbers*. These are numbers of the form `realPart` + `imaginaryPart` * *i*, where *i* has the value

$\sqrt{-1}$

a) Modify the class to enable input and output of complex numbers through the overloaded >> and << operators, respectively. (You should remove the `print` member function from the class.)

b) Overload the multiplication operator to enable multiplication of two complex numbers as in algebra. Complex number multiplication is performed as follows:

$$(a + bi) * (c + di) = (ac - bd) + (ad + bc)i$$

c) Overload the == and != operators to allow comparisons of complex numbers.

HW2:(Polynomial)

Develop class `Polynomial`. The internal representation of a `Polynomial` is an array of terms. Each term contains a coefficient and an exponent. The term

$$2x^4$$

has the coefficient 2 and the exponent 4. Develop a complete class containing proper constructor and destructor functions as well as *set* and *get* functions. The class should also provide the following overloaded operator capabilities:

a) Overload the addition operator (+) to add two `Polynomials`.

b) Overload the subtraction operator (-) to subtract two `Polynomials`.

c) Overload the assignment operator to assign one `Polynomial` to another.

d) Overload the addition assignment operator (+=) and subtraction assignment operator (-=).

Lab4 Object-Oriented Programming: Inheritance

Objectives

This lab was designed for learning inheritance mechanism to support OO programming in C++:

- To create classes by inheriting from exiting classes.
- The notations of base classes and derived classes and the relationships between them.
- The order in which objects were constructed and destructed in inheritance hierarchies.
- The initial in heritance
- The difference between *public*, *protected* and *private* member access specifier.
- The difference between *public*, *protected* and *private* inheritance.
- The inheritance, add and hide of class member functions.
- The translation between base class and derived classs.

Experiments

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;
}
```

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);
}
```

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();
}
```

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;
}
```

Homework:

12.3　12.7

Lab5 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.

Experiment

Ex 1：（习题 13.12， Employee 类继承层次）

1. Description of the Problem

英文：(Payroll System Modification) Modify the payroll system of Figs. 13.13~13.23 to include private data member birthDate in class Employee. Use class Date from Figs. 11.12~11.13 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.

中文：修改图 13.13~13.23 的工资系统，增加 private 数据成员 birthDate(Date 对象), 要求使用图 11.12~11.13 的 Date 作为生日类型。假设工资系统每月处理一次，创建一个 vector 存储 Employee 指针来存储不同的员工对象，用一个循环计算每个员工的工资时(多态)，遇到当月过生日的员工多发 100 美元奖金。

2. Problem-Solving Tips

1）取当前时间函数提示：

方法一：

```
#include <windows.h>
int    main()
{ SYSTEMTIME systm;
   GetLocalTime(&systm);
   cout<<systm.wYear<<"-"<<systm.wMonth<<"-"<<systm.wDay<<" "<<
        systm.wHour<<":"<<systm.wMinute<<":"<<systm.wSecond;
 return 0;
}
```

方法二：

```
#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）测试函数示例

参考教材的测试函数 13.23 和 13.25。

3. 结果示例

```
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：（习题 13.13，多态，Account 类继承层次）

1. Description of the Problem

(Shape Hierarchy) Implement the Shape hierarchy designed in Exercise 12.7 (which is based on the hierarchy in Fig. 12.3). 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. 实验要求

（1）画出类图，写够 5 个类定义，并且继承关系正确

（2）每个类的成员函数定义，且必须至少包含以下函数：

构造函数，析构函数，virtual getArea 函数

（4）main 函数：

vector 数组声明

使用类对象对 vector 数组的赋值

提供一个测试函数 test，用来测试 getArea，其参数是基类指针或引用变量

Ex 3：（习题 13.16，多态，Account 类继承层次）

1. Description of the Problem

对实验 8 中的习题 12.10 进行修改。

创建一个与银行账户相关的类继承层次。银行的所有账户都可以存款和取款。存款能够产生一定的利息，查询和取款交易要缴纳一定的手续费。

要求：基类：Account(参考实验提示)；

派生类：SavingAccount 和 CheckingAccount；

SavingAccount：继承 Account 的成员函数；构造函数接收两个参数：存款初始值（initialBalance）和利率（rate）；增加一个数据成员：利率(interestRate)，增加 public 类型的成员函数用于计算利率(calculateInterest())。

CheckingAccount：构造函数应接收到两个参数，一个是存款初始值（initialBalance），一个是手续费（fee）；增加一个数据成员：手续费（transactionFee）；重新定义成员函数 credit()和 debit()，以便能够从存款余额中减去手续费，要求成员函数通过调用基类的成员函数来更新存款数目，debit()函数应该在确定取款之后才能扣除手续费。

要求 1：创建一个 vector 存储一组 SavingAccount 和 CheckingAccount 对象（多态），处理每一个账户时，判断该账户的类型，如果是 SavingAccount，使用其成员函数 calculateInterest()计算利率并加入账户，处理完一个账户，调用基类的成员函数 getBalance() 打印其新的存款。

2. Problem-Solving Tips

1）类定义：

要求独立完成。

2）测试函数：

要求独立完成。

3. 结果示例

```
Account 1 balance: $25.00
Enter an amount to withdraw from Account 1: 10
Enter an amount to deposit into Account 1: 30
Adding $1.35 interest to Account 1 (a SavingsAccount)
Updated Account 1 balance: $46.35

Account 2 balance: $80.00
Enter an amount to withdraw from Account 2: 0
$1.00 transaction fee charged.
Enter an amount to deposit into Account 2: 10
$1.00 transaction fee charged.
Updated Account 2 balance: $88.00

Account 3 balance: $200.00
Enter an amount to withdraw from Account 3: 10
Enter an amount to deposit into Account 3: 0
Adding $2.85 interest to Account 3 (a SavingsAccount)
Updated Account 3 balance: $192.85

Account 4 balance: $400.00
Enter an amount to withdraw from Account 4: 0
$0.50 transaction fee charged.
Enter an amount to deposit into Account 4: 0
$0.50 transaction fee charged.
Updated Account 4 balance: $399.00
```

Lab6 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.

Experiment

- **Ex1**

Problem description

`quickSort`(快速排序) is a fast sorting algorithm, which is widely applied in practice. It is used on the principle of divide-and-conquer. **`quickSort`** works by partitioning a given array A[p . . r] into two non-empty sub-arrays A[p . . q] and A[q+1 . . r] such that every element in A[p . . q] is less than or equal to every element in A[q+1 . . r]. Then the two sub-arrays are sorted by recursive calls to **`quickSort`**. The details of **`quickSort`** are described as follows:

1. **Choose a pivot value**(基准)**.** You may take the value of the first element as **`pivot`** value, but it can be any value, which is in range of sorted values, even if it doesn't present in the array.
2. **Partition**(划分). Rearrange elements in such a way that all elements which are lesser than the **`pivot`** go to the **`left`** part of the array, and all elements greater than the **`pivot`** go to the **`right`** part of the array. Values equal to the **`pivot`** can stay in any position of the array.
3. **Sort both sub-arrays.** Apply **`quicksort`** algorithm recursively to the left and the right parts of the array.

Write a function template **`quickSort`** under the reference of the sort program of Fig.8.15, and also write a diver program that inputs, sorts and outputs an **`int`** array and a **`float`** array with 8 elements.

```
int data elements in original order
6 5 3 2 1 7 8 4
int data elements in ascending order
1 2 3 4 5 6 7 8

float point data elements in original order
12.3   77.2   36.9   28.4   9.7   50.5   21.9   43.6
float point data elements in ascending order
9.7   12.3   21.9   28.4   36.9   43.6   77.2   50.5
```

Templates

```
// Lab14: quickSort.cpp
```

```
// Sorts elements of an array in ascending order using template functions.
#include <iostream>
using std::cout;
using std::endl;

#include <iomanip>
using std::setw;

int main()
{
    const int SIZE = 8; // size of array
    int a[ SIZE ] = { 6, 5, 3, 2, 1, 7, 8, 4 };

    // display int array in original order
    cout << "int data elements in original order\n";
    printArray( a, SIZE ); // print int array
    quickSort( a, 0, SIZE-1 ); // sort int array

    // display int array in sorted order
    cout << "\nint data elements in ascending order\n";
    printArray( a, SIZE );
    cout << "\n\n";

    // initialize float array
    float b[ SIZE ] =
        { 12.3, 77.2, 36.9, 28.4, 9.7, 50.5, 21.9, 43.6 };

    // display float array in original order
    cout << "float point data elements in original order\n";
    printArray( b, SIZE ); // print float array
    quickSort( b, 0, SIZE-1 ); // sort float array

    // display sorted float array
    cout << "\nfloat point data elements in ascending order\n";
    printArray( b, SIZE );
```

```
    cout << endl;
    return 0;
} // end main
```

Problem-Solving Tips

1. **`quickSort`** is a recursive function. Its non-template function definition is provided as follows for reference, in which the subscripts **`left`** and **`right`** stands for the head and tail of the array, **`pivotpos`** stands for the position of the pivot value.

```
void quickSort ( const int * array, const int left, const int right )
{
    if (left<right) {    // when array length is >1
        int pivotpos=partition(array, left, right); // to partition array[left]~array[right]
        quicksort (array, left, pivotpos-1);     // recursively invocation on left part of array
        quicksort (array, pivotpos+1, right); // recursively invocation on right part of array
    } // end if
} // end function
```

2. In **`partition`** function, each element's value in the array will be compared to the **`pivot`** value with a **`for`** loop. If the element's value is lesser than **`pivot`** value, this element will be moved left, if the element's value is greater than **`pivot`** value, this element will be moved right. At last, **`pivot`** is deposited in place and its value is returned to function. The non-template function declaration of **`partition`** can be described as follows.

```
int partition ( const int * array, const int low, const int high );    // prototype
```

- **Ex2 (p.577-14.7)**

Problem Description

Use an **`int`** template nontype parameter **`numberOfElements`** and a type parameter **`elementType`** to help create a template for the **`Array`** class

(Figs.11.6-11.7) we developed in Chapter 11. This template will enable **`Array`** objects to be instantiated with a specified number of elements of a specified element type at compile time.

Experimental Results

```
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
```

Homework:

Ex 1:（习题 14.6）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?

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 use `auto_ptr` to prevent memory leaks.
8. To understand the standard exception hierarchy.

Experiment

Ex 1：（习题 16.25，异常处理的逻辑流程）

1． Description of the Problem

（英文）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.

（中文）假设一个程序抛出一个异常，而一个特定的异常处理程序将开始执行。现在假设一个异常处理程序本身又抛出了一个相同的异常，这会形成无限循环吗？编写一个程序来证明你的观点。

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 16.2，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． 结果示例

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

Ex 2：（习题 16.30 构造函数、析构函数和异常处理）

1. Description of the Problem

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

（中文）编写一个程序，证明只有在异常抛出之前创建的成员对象的析构函数才会被调用。

2. Problem-Solving Tips

a) 定义类 Item，并包含整形成员变量 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. 结果示例

```
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
```

Homework:

Ex 1：（习题 16.34 重新抛出异常）

1. Description of the Problem

（英文）Write a program that illustrates rethrowing an exception.

（中文）编写一个程序，描述重新抛出异常的情况。

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. 结果示例

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

Ex 2：（习题 16.34 堆栈展开）

1. Description of the Problem

（英文）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.

（中文）编写一个程序，从深层嵌套函数抛出异常，并且由含有调用链的 try 语句块后的 catch 处理器来捕获那个异常。

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. 结果示例

```
In main: Caught TestException
```

Lab8 File Processing

Objectives

1. To create, read, and write sequential-access files

Experiments

- **Ex1**

Description of the Problem

Suppose we wish to process survey results that are stored in a file. This exercise requires two separate programs. First, create a program that prompts the user for survey responses and outputs each response to a file. Use an *ofstream* to create a file called "numbers.txt". Then create a program to read the survey responses from "numbers.txt". The responses should be read from the file by using an *ifstream*. Input one integer at a time from the file. The program should continue to read responses until it reaches the end of file. The results should be output to the text file "output.txt".

Hint

• The second program will use both *ifstream* and *ofstream* objects, the first for reading responses from numbers.txt and the second for writing frequency counts to output.txt.

Contents of numbers.txt

5 3 7 2 8 6 9 5 4 2 1 2 8 10 4 5 2 7 10 4 9 8 2 1 3 7 5 6 8 4 3 8 2 1

Contents of output.txt

Number of 1 responses: 3 Number of 2 responses: 6 Number of 3 responses: 3 Number of 4 responses: 4 Number of 5 responses: 4 Number of 6 responses: 2 Number of 7 responses: 3 Number of 8 responses: 5 Number of 9 responses: 2 Number of 10 responses: 2

- Ex2

Description of the Problem

(1) Create a simple sequential-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 sequential file specified by the user (for example "file.dat").

Contents of file.dat

253 Bill Purple 88.9 632 Debbie Green 91.2 412 Steven Red 94.7 522 Mike Blue 83.8

(2) Create a simple sequential-access file-processing program to complement the program in (1). This program should open the file created by (1) and read and display the grade information for each student. The program should also display the class

average.

Output

```
Enter a file name: file.dat
253 Bill Purple 88.9
632 Debbie Green 91.2
412 Steven Red 94.7
522 Mike Blue 83.8
Class average: 89.65
```

Homework

Ex17.7, Ex17.8, Ex17.9 (p664)