**Lab Exercises**

Class: 软件1801 Name: 杨龙 StuID:\_201816040108\_\_

Date: 2019/11/6 Score:

**Lab Exercise 1 — Modifying Class Account**

**I Lab Objectives**

In this lab, you will practice:

1. Creating member functions.
2. Invoking functions and receiving return values from functions.
3. Testing a condition using an if statement.
4. Outputting variables with stream insertion and the cout object.

**II Description of the Problem**

Modify class Account to provide a member function called debit that withdraws money from an Account. Ensure that the debit amount does not exceed the Account’s balance. If it does, the balance should be left unchanged and the function should print a message indicating "Debit amount exceeded account balance." Modify class AccountTest to test member function debit.

**III Sample Output**



**IV Your Solution**

// Lab 1: Account.h

// Definition of Account class.

class Account

{

public:

Account( double ); // constructor initializes balance

void credit( int ); // add an amount to the account balance

/\* write code to declare member function debit. \*/

void debit(double);

double getBalance(); // return the account balance

private:

double balance; // data member that stores the balance

}; // end class Account

// Lab 1: Account.cpp

// Member-function definitions for class Account.

#include <iostream>

using namespace std;

#include "Account.h" // include definition of class Account

// Account constructor initializes data member balance

Account::Account( double initialBalance )

{

balance = 0; // assume that the balance begins at 0

// if initialBalance is greater than 0, set this value as the

// balance of the Account; otherwise, balance remains 0

if ( initialBalance > 0 )

balance = initialBalance;

// if initialBalance is negative, print error message

if ( initialBalance < 0 )

cout << "Error: Initial balance cannot be negative.\n" << endl;

} // end Account constructor

// credit (add) an amount to the account balance

void Account::credit( int amount )

{

balance = balance + amount; // add amount to balance

} // end function credit

/\* write code to define member function debit. \*/

void Account::debit(double money)

{

if(balance>money)

balance=balance-money;

else

cout<<"Debit amount exceeded account balance."<<endl;

}

// return the account balance

double Account::getBalance()

{

return balance; // gives the value of balance to the calling function

} // end function getBalance

// Lab 1: AccountTest.cpp

// Create and manipulate Account objects.

#include <iostream>

using namespace std;

// include definition of class Account from Account.h

#include "Account.h"

// function main begins program execution

int main()

{

Account account1( 50.00); // create Account object

Account account2( 0.00 ); // create Account object

// display initial balance of each object

cout << "account1 balance: $" << account1.getBalance() << endl;

cout << "account2 balance: $" << account2.getBalance() << endl;

double withdrawalAmount; // stores withdrawal amount read from user

cout << "\nEnter withdrawal amount for account1: "; // prompt

cin >> withdrawalAmount; // obtain user input

cout << "\nsubtracting " << withdrawalAmount

<< " from account1 balance\n\n";

/\* write code to withdraw money from account1 \*/

account1.debit(withdrawalAmount);

// display balances

cout << "account1 balance: $" << account1.getBalance() << endl;

cout << "account2 balance: $" << account2.getBalance() << endl;

cout << "\nEnter withdrawal amount for account2: "; // prompt

cin >> withdrawalAmount; // obtain user input

cout << "\nsubtracting " << withdrawalAmount

<< " from account2 balance\n\n";

/\* write code to withdraw money from account2 \*/

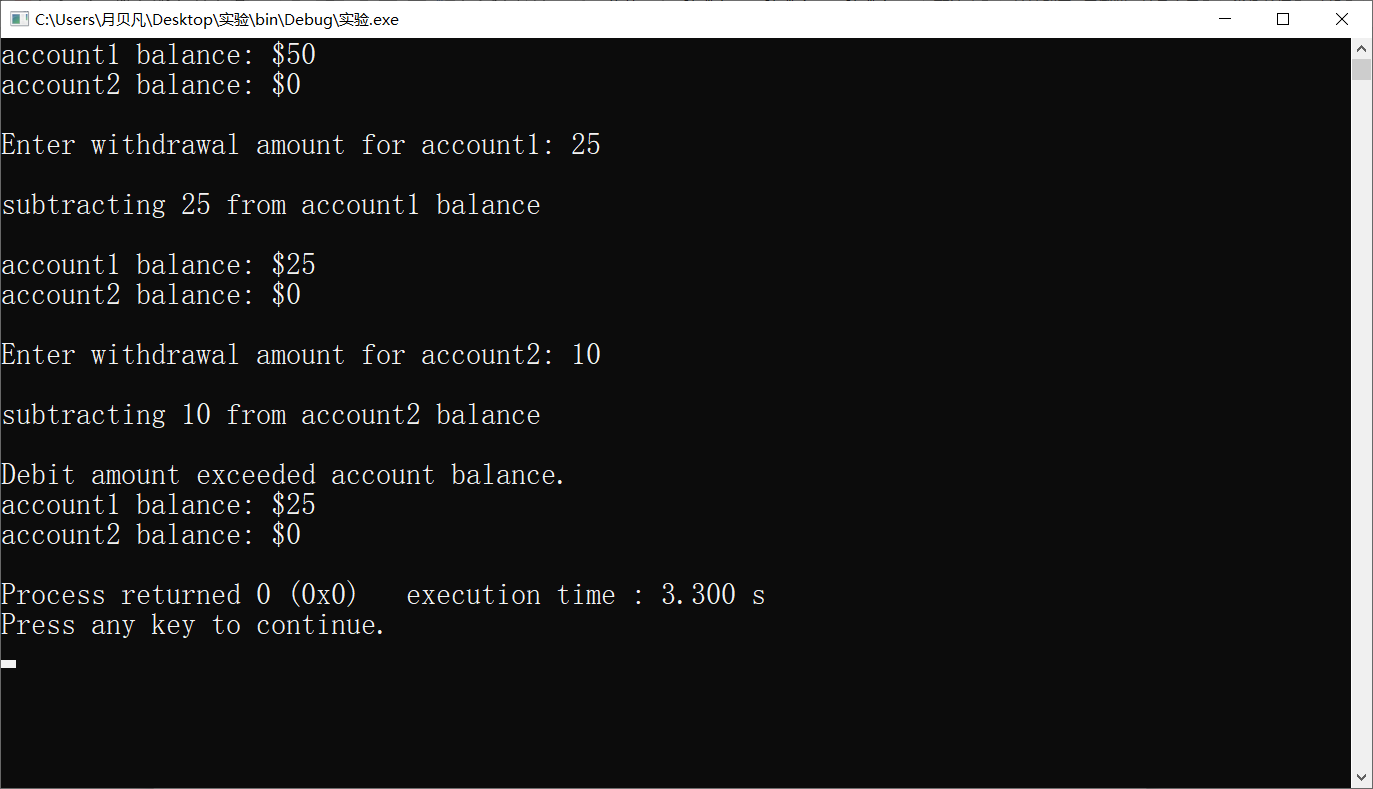
account2.debit(withdrawalAmount);

// display balances

cout << "account1 balance: $" << account1.getBalance() << endl;

cout << "account2 balance: $" << account2.getBalance() << endl;

} // end main



**Lab Exercise 2 — Modifying class GradeBook**

**I Lab Objectives**

In this lab, you will practice:

1. Declaring a data member.
2. Providing *set* and *get* functions to manipulate a data member’s value.
3. Declaring member functions with parameters.

**II Description of the Problem**

Modify class GradeBook. Include a second string data member that represents the name of the course’s instructor. Provide a *set* function to change the instructor’s name and a *get* function to retrieve it. Modify the constructor to specify *two* parameters—one for the course name and one for the instructor’s name. Modify member function displayMessage such that it first outputs the welcome message and course name, then outputs "This course is presented by: " followed by the instructor’s name. Modify the test application to demonstrate the class’s new capabilities.

**III Sample Output**



**IV Your Solution**

// Lab 2: GradeBook.h

// Definition of GradeBook class that stores an instructor's name.

#include <string> // program uses C++ standard string class

using namespace std;

// GradeBook class definition

class GradeBook

{

public:

// constructor initializes course name and instructor name

GradeBook( string, string );

void setCourseName( string ); // function to set the course name

string getCourseName(); // function to retrieve the course name

/\* write code to declare a get function for the instructor's name \*/

string getInstructorName();

/\* write code to declare a set function for the instructor's name \*/

void setInstructorName(string);

void displayMessage(); // display welcome message and instructor name

private:

string courseName; // course name for this GradeBook

string instructorName; // instructor name for this GradeBook

}; // end class GradeBook

// Lab 2: GradeBook.cpp

// Member-function definitions for class GradeBook.

#include <iostream>

using namespace std;

// include definition of class GradeBook from GradeBook.h

#include "GradeBook.h"

// constructor initializes courseName and instructorName

// with strings supplied as arguments

GradeBook::GradeBook( string course, string instructor )

{

setCourseName( course ); // initializes courseName

setInstructorName( instructor ); // initialiZes instructorName

} // end GradeBook constructor

// function to set the course name

void GradeBook::setCourseName( string name )

{

courseName = name; // store the course name

} // end function setCourseName

// function to retrieve the course name

string GradeBook::getCourseName()

{

return courseName;

} // end function getCourseName

/\* write code to define a get member function for the instructor's name \*/

string GradeBook::getInstructorName()

{

return instructorName;

}

/\* write code to define a set member function for the instructor's name \*/

void GradeBook::setInstructorName(string names)

{

instructorName=names;

}

// display a welcome message and the instructor's name

void GradeBook::displayMessage()

{

// display a welcome message containing the course name

cout << "Welcome to the grade book for\n" << getCourseName() << "!"

<< endl;

/\* write code to output the instructor's name \*/

cout << "This course is presented by:\n" << getInstructorName() << "!"

<< endl;

} // end function displayMessage

// Lab 2: GradeBookTest.cpp

// Test program for modified GradeBook class.

#include <iostream>

using namespace std;

// include definition of class GradeBook from GradeBook.h

#include "GradeBook.h"

// function main begins program execution

int main()

{

// create a GradeBook object; pass a course name and instructor name

GradeBook gradeBook1(

"CS101 Introduction to C++ Programming" ,"Sam Smith");

// display welcome message and instructor's name

gradeBook1.displayMessage();

string names;

cout<<"Changing instructor name to "<<names;

getline(cin,names);

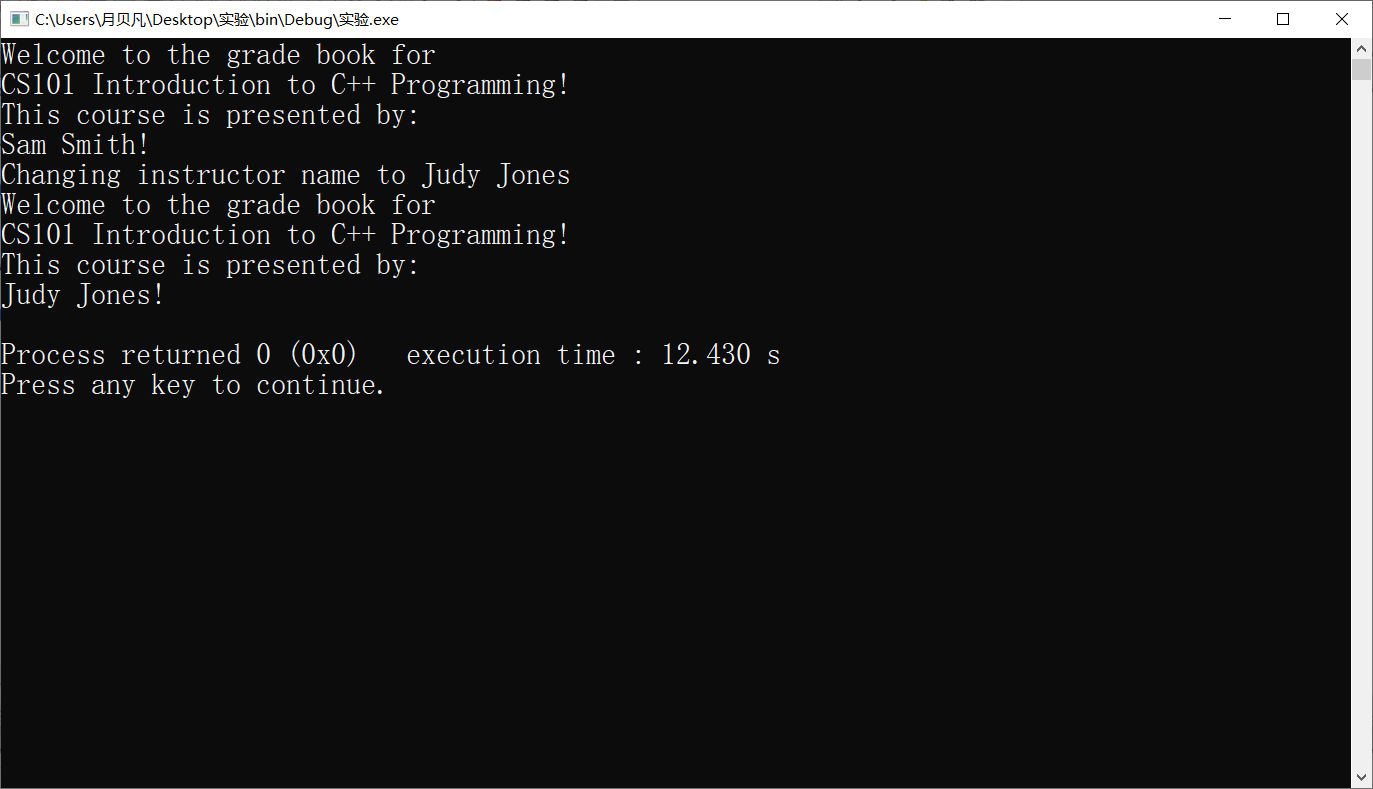
GradeBook gradeBook2(

"CS101 Introduction to C++ Programming" ,names);

/\* write code to change instructor's name and output changes \*/

gradeBook2.displayMessage();

} // end main



**Lab Exercise 3 — Creating an Employee Class**

**I Lab Objectives**

In this lab, you will practice:

1. Creating a class definition.
2. Declaring data members.
3. Defining a constructor.
4. Defining set and get functions.
5. Writing a test application to demonstrate the capabilities of another class.

**II Description of the Problem**

Create a class called Employee that includes three pieces of information as data members—a first name (type string), a last name (type string) and a monthly salary (type int). Your class should have a constructor that initializes the three data members. Provide a *set* and a *get* function for each data member. If the monthly salary is not positive, set it to 0. Write a test program that demonstrates class Employee’s capabilities. Create two Employee objects and display each object’s yearly salary. Then give each Employee a 10 percent raise and display each Employee’s yearly salary again.

**III Sample Output**



**IV Your Solution**

// Lab 3: Employee.h

// Employee class definition.

#include <string> // program uses C++ standard string class

using namespace std;

// Employee class definition

class Employee

{

public:

/\* Declare a constructor that has one parameter for each data member \*/

Employee(string,string,int);

/\* Declare a set method for the employee's first name \*/

void setfirstname(string);

/\* Declare a get method for the employee's first name \*/

string getfirstname();

/\* Declare a set method for the employee's last name \*/

void setlastname(string);

/\* Declare a get method for the employee's last name \*/

string getlastname();

/\* Declare a set method for the employee's monthly salary \*/

void setsalary(int);

/\* Declare a get method for the employee's monthly salary \*/

int getsalary();

private:

/\* Declare a string data member for the employee's first name \*/

string firstname;

string lastname;

int salary;

/\* Declare a string data member for the employee's last name \*/

/\* Declare an int data member for the employee's monthly salary \*/

}; // end class Employee

// Lab 3: Employee.cpp

// Employee class member-function definitions.

#include <iostream>

using namespace std;

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

/\* Define the constructor. Assign each parameter value to the appropriate data

member. Write code that validates the value of salary to ensure that it is

not negative. \*/

Employee::Employee(string firstname,string lastname,int salary)

{

setfirstname(firstname);

setlastname(lastname);

setsalary(salary);

}

/\* Define a set function for the first name data member. \*/

void Employee::setfirstname(string name)

{

firstname=name;

}

/\* Define a get function for the first name data member. \*/

string Employee::getfirstname()

{

return firstname;

}

/\* Define a set function for the last name data member. \*/

void Employee::setlastname(string name)

{

lastname=name;

}

/\* Define a get function for the last name data member. \*/

string Employee::getlastname()

{

return lastname;

}

/\* Define a set function for the monthly salary data member. Write code

that validates the salary to ensure that it is not negative. \*/

void Employee::setsalary(int money)

{

if(money>0)

salary=money;

else

salary=0;

}

/\* Define a get function for the monthly salary data member. \*/

int Employee::getsalary()

{

return salary\*12;

}

// Lab 3: EmployeeTest.cpp

// Create and manipulate two Employee objects.

#include <iostream>

using namespace std;

#include "Employee.h" // include definition of class Employee

// function main begins program execution

int main()

{

/\* Create two Employee objects and assign them to Employee variables. \*/

Employee employee1("Bob","Jones",2875);

Employee employee2("Susan","Baker",3150);

/\* Output the first name, last name and salary for each Employee. \*/

cout<<"Employee 1:"<<employee1.getfirstname()<<" "<<employee1.getlastname()<<";";

cout<<"Yealy Salary:"<<employee1.getsalary()<<endl;

cout<<"Employee 2:"<<employee2.getfirstname()<<" "<<employee2.getlastname()<<";";

cout<<"Yealy Salary:"<<employee2.getsalary()<<endl;

/\* Give each Employee a 10% raise. \*/

cout<<"Increasing employee salaries by 10%"<<endl;

/\* Output the first name, last name and salary of each Employee again. \*/

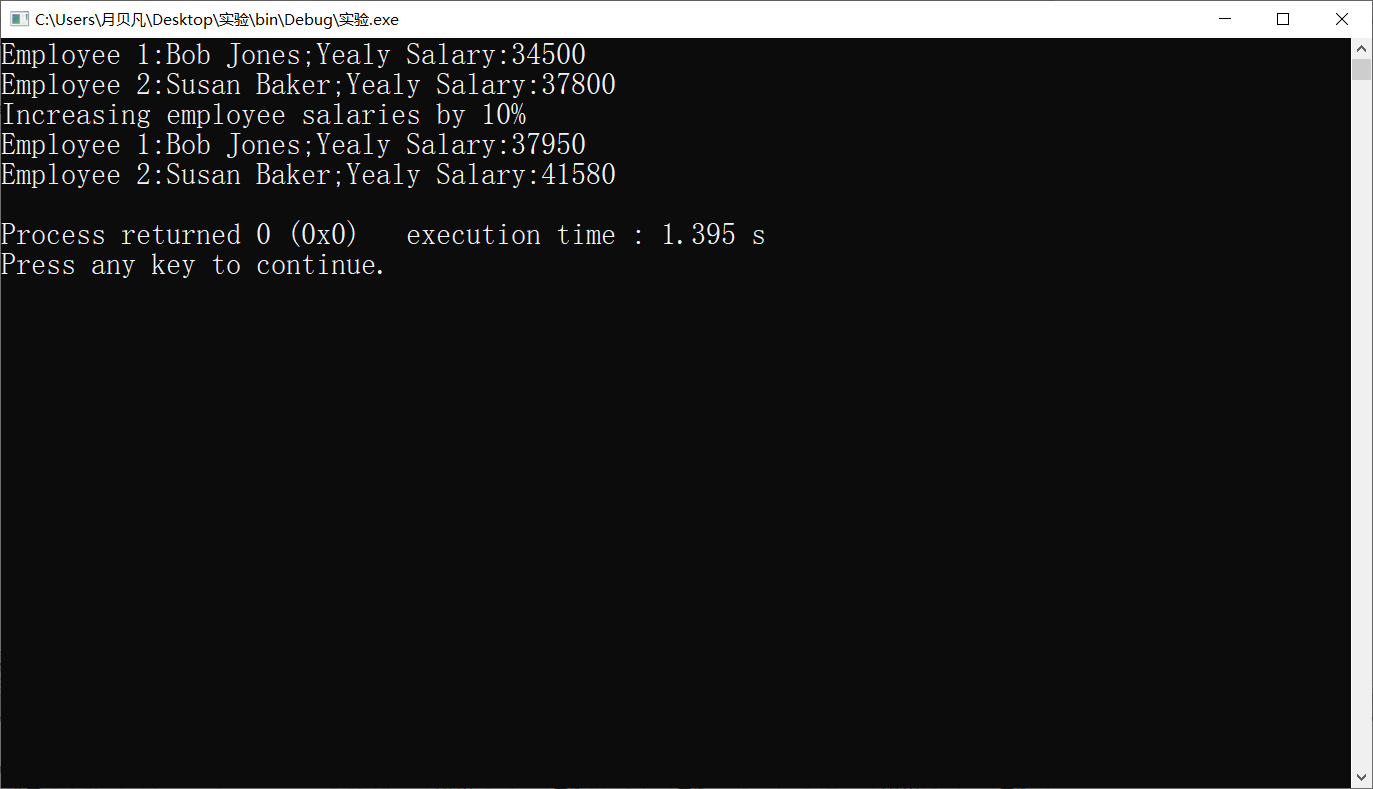
cout<<"Employee 1:"<<employee1.getfirstname()<<" "<<employee1.getlastname()<<";";

cout<<"Yealy Salary:"<<employee1.getsalary()\*1.1<<endl;

cout<<"Employee 2:"<<employee2.getfirstname()<<" "<<employee2.getlastname()<<";";

cout<<"Yealy Salary:"<<employee2.getsalary()\*1.1<<endl;

} // end main



**Lab Exercise 4 — Complex Numbers**

**I Lab Objectives**

In this lab, you will practice:

1. Creating new data types by writing class definitions.
2. Defining member functions of programmer-defined classes.
3. Instantiating objects from programmer-defined classes.
4. Calling member functions of programmer-defined classes.

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

1. Initializing programmer-defined class data members with class constructors.

**II Description of the Problem**

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

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:

1) Adding two Complex numbers: The real parts are added together and the imaginary parts are added together.

2) 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.

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

**III Sample Output**



**IV Your Solution**

// Lab 4: Complex.h

#ifndef COMPLEX\_H

#define COMPLEX\_H

/\* Write class definition for Complex \*/

class Complex

{

public:

Complex(double=0.0,double=0.0);

void setComplexNumber(double,double);

void printComplex();

Complex add(const Complex &);

Complex subtract(const Complex &);

private:

double realPart;

double imaginaryPart;

};

#endif

// Lab 4: Complex.cpp

// Member-function definitions for class Complex.

#include <iostream>

using namespace std;

#include "Complex.h"

Complex::Complex( double real, double imaginary )

{

setComplexNumber( real, imaginary );

} // end Complex constructor

Complex Complex::add( const Complex &right )

{

/\* Write a statement to return a Complex object. Add

the realPart of right to the realPart of this Complex

object and add the imaginaryPart of right to the

imaginaryPart of this Complex object \*/

realPart+=right.realPart;

imaginaryPart+=right.imaginaryPart;

return \*this;

} // end function add

Complex Complex::subtract( const Complex &right )

{

/\* Write a statement to return a Complex object. Subtract

the realPart of right from the realPart of this Complex

object and subtract the imaginaryPart of right from

the imaginaryPart of this Complex object \*/

realPart-=right.realPart;

imaginaryPart-=right.imaginaryPart;

return \*this;

} // end function subtract

void Complex::printComplex()

{

cout << '(' << realPart << ", " << imaginaryPart << ')';

} // end function printComplex

void Complex::setComplexNumber( double rp, double ip )

{

realPart = rp;

imaginaryPart = ip;

} // end function setComplexNumber

// Lab 4: ComplexTest.cpp

#include <iostream>

using namespace std;

#include "Complex.h"

int main()

{

Complex a( 1, 7 ), b( 9, 2 ), c; // create three Complex objects

a.printComplex(); // output object a

cout << " + ";

b.printComplex(); // output object b

cout << " = ";

c = a.add( b ); // invoke add function and assign to object c

c.printComplex(); // output object c

cout << '\n';

a.setComplexNumber( 10, 1 ); // reset realPart and

b.setComplexNumber( 11, 5 ); // and imaginaryPart

a.printComplex(); // output object a

cout << " - ";

b.printComplex(); // output object b

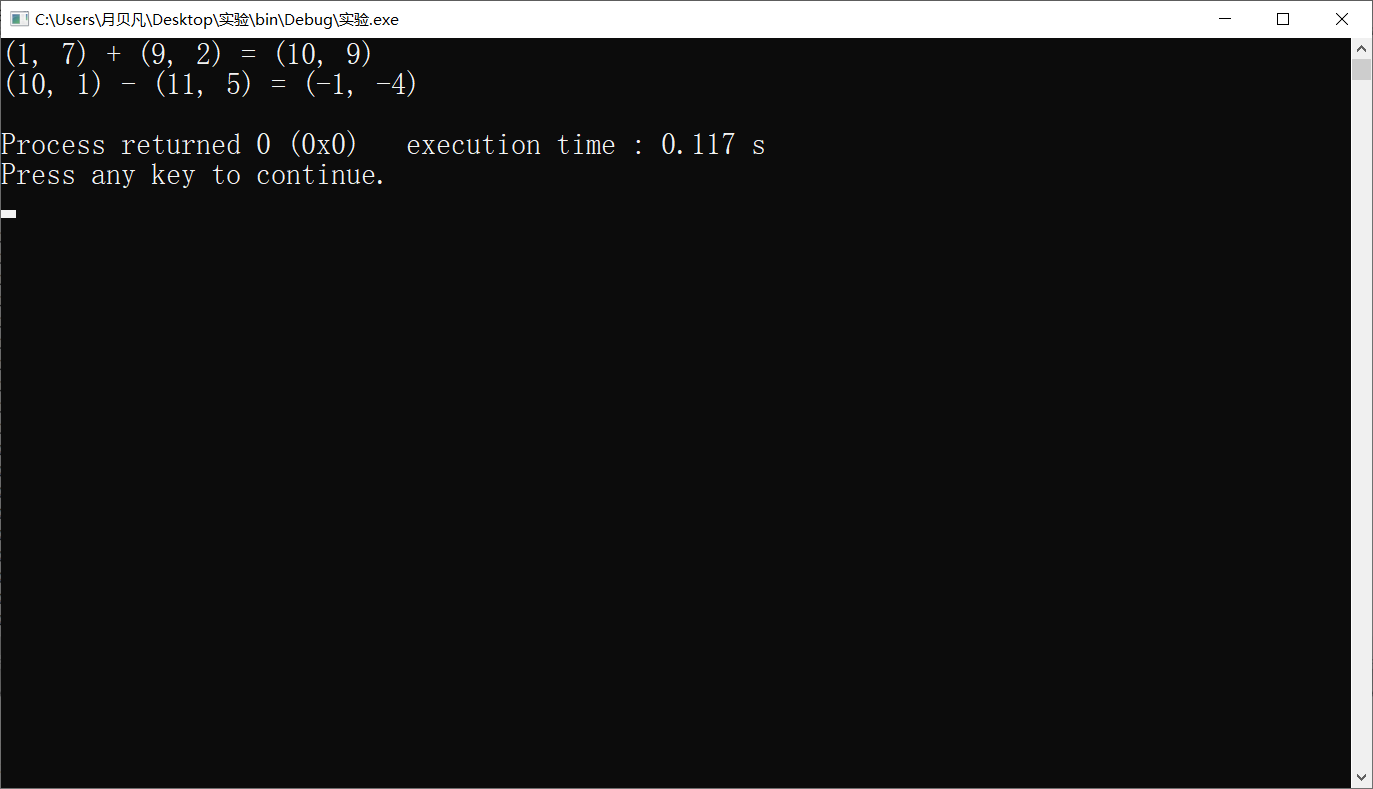
cout << " = ";

c = a.subtract( b ); // invoke add function and assign to object c

c.printComplex(); // output object c

cout << endl;

} // end main



**V Follow-Up Questions and Activities**

1. Why do you think const was used in the parameter list of add and subtract?

防止被作为参数对象的数据发生改变，提高效率。

1. Can add and subtract’s parameters be passed by value instead of by reference? How might this affect the design of class Complex? Write a new class definition that illustrates how the parameters would be passed by value.

可以。

1. Declare a Complex number, as follows, without passing any arguments to the constructor. What happens?Does the default constructor get called?

Complex a;

没有在内存中创建对象，只是创建了一个“对象名”。

# **Lab Exercise 5 — Dates**

**I Lab Objectives**

In this lab, you will practice:

1. Using access functions and utility functions so that it is not necessary for non-member functions to be able to access a class’ data members.

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

1. Overloading constructors and using default arguments with constructors.
2. Defining a destructor.

**II Description of the Problem**

Modify the Date class to provide a member function nextDay to increment the day by one. The Date object should always remain in a consistent state. Write a program that tests function nextDay in a loop that prints the date during each iteration to illustrate that the nextDay function works correctly. Be sure to test the following cases:

1. Incrementing into the next month.
2. Incrementing into the next year.

**III Sample Output**



**IV Your Solution**

// Lab 5: Date.h

#ifndef DATE\_H

#define DATE\_H

class Date

{

public:

Date( int = 1, int = 1, int = 2000 ); // default constructor

void print(); // print function

void setDate( int, int, int ); // set month, day, year

void setMonth( int ); // set month

void setDay( int ); // set day

void setYear( int ); // set year

int getMonth(); // get month

int getDay(); // get day

int getYear(); // get year

/\* Write a member function prototype for nextDay,

which will increment the Date by one day \*/

void nextDay();

private:

int month; // 1-12

int day; // 1-31 (except February(leap year), April, June, Sept, Nov)

int year; // 1900+

bool leapYear(); // leap year

int monthDays(); // days in month

}; // end class Date

#endif

// Lab 5: Date.cpp

// Member-function definitions for class Date.

#include <iostream>

using namespace std;

#include "Date.h" // include definition of class Date

Date::Date( int m, int d, int y )

{

setDate( m, d, y ); // sets date

} // end Date constructor

void Date::setDate( int mo, int dy, int yr )

{

setMonth( mo ); // invokes function setMonth

setDay( dy ); // invokes function setDay

setYear( yr ); // invokes function setYear

} // end function setDate

void Date::setDay( int d )

{

if ( month == 2 && leapYear() )

day = ( d <= 29 && d >= 1 ) ? d : 1;

else

day = ( d <= monthDays() && d >= 1 ) ? d : 1;

} // end function setDay

void Date::setMonth( int m )

{

month = m <= 12 && m >= 1 ? m : 1; // sets month

} // end function setMonth

void Date::setYear( int y )

{

year = y >= 1900 ? y : 1900; // sets year

} // end function setYear

int Date::getDay()

{

return day;

} // end function getDay

int Date::getMonth()

{

return month;

} // end function getMonth

int Date::getYear()

{

return year;

} // end function getYear

void Date::print()

{

cout << month << '-' << day << '-' << year << '\n'; // outputs date

} // end function print

/\* Write code to define member function nextDay;

make sure to check if the new day is the start of

a new month or a new year \*/

void Date::nextDay()

{

day++;

if(leapYear()&&month==2)

{

if(day>29)

{

day=1;

month++;

}

}

else if(day>monthDays())

{

day=1;

month++;

if(month>12)

{

month=1;

year++;

}

}

}

bool Date::leapYear()

{

if ( getYear() % 400 == 0 || ( getYear() % 4 == 0 && getYear() % 100 != 0 ) )

return true; // is a leap year

else

return false; // is not a leap year

} // end function leapYear

int Date::monthDays()

{

const int days[ 12 ] =

{ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

return getMonth() == 2 && leapYear() ? 29 : days[ getMonth() - 1 ];

} // end function monthDays

// Lab 5: DateTest.cpp

#include <iostream>

using namespace std;

#include "Date.h" // include definitions of class Date

int main()

{

const int MAXDAYS = 16;

Date d( 12, 24, 2004 ); // instantiate object d of class Date

// output Date object d's value

for ( int loop = 1; loop <= MAXDAYS; ++loop )

{

d.print(); // invokes function print

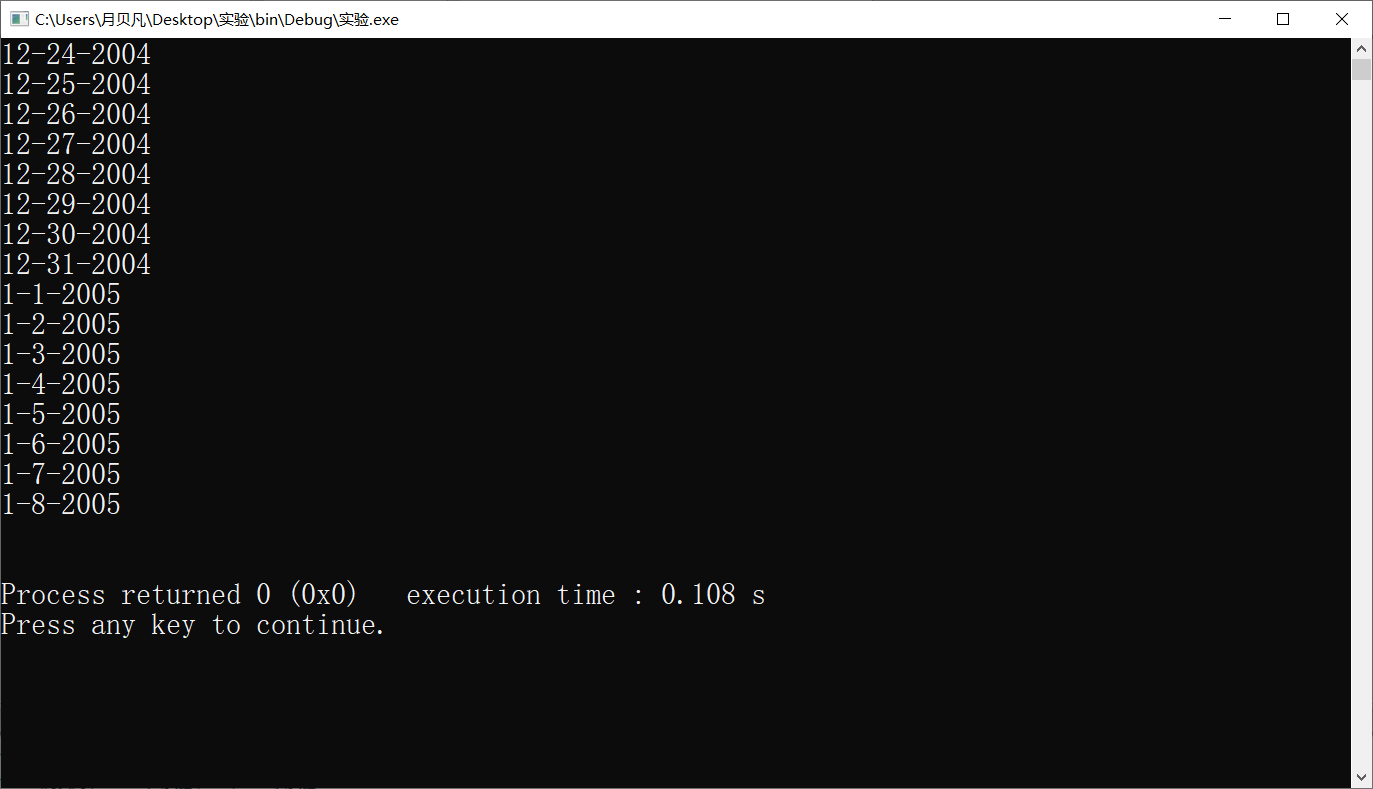
/\* Write call to nextDay \*/

d.nextDay();

} // end for

cout << endl;

} // end main



**V Follow-Up Questions and Activities**

1. The Date class has only one constructor. Is it possible to have more than one constructor?

可以有多个构造函数。

1. What happens when a member function that takes no arguments is called without the parentheses (i.e.,dateObject.nextDay)?

编译会报错。

1. Write a destructor for the Date class. The destructor should print text indicating that the destructor for the Date class was called successfully.

Date::Date()

{

Cout<<”The destructor for the Date class was called successfully.”<<endl;

}

1. In main, try to change d’s year to 2003 using an assignment statement. Do not call function setYear. What happens? Are you able to change the value?

不能。

**Lab Exercise 6 — Simple Calculator**

**I Lab Objectives**

In this lab, you will practice:

1. Using classes to create a data type Simple Calculator capable of performing arithmetic operations.
2. Creating const member functions to enforce the principle of least privilege.

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

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

**II Description of the Problem**

Write a SimpleCalculator class that has public methods for adding, subtracting, multiplying and dividing two doubles. 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.

**III Sample Output**



**IV Your Solution**

// Lab Exercise 6: SimpleCalculator.h

// class SimpleCalculator definition

class SimpleCalculator

{

public:

/\* Write prototype for add member function \*/

double add(double,double) const;

double subtract( double, double ) const;

double multiply( double, double ) const;

/\* Write prototype for divide member function \*/

double divide(double,double) const;

}; // end class SimpleCalculator

// Lab Exercise 6: SimpleCalculator.cpp

#include "SimpleCalculator.h"

/\* Write definition for add member function \*/

double SimpleCalculator::add(double a ,double b) const

{

return a+b;

}

// 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 \*/

double SimpleCalculator::divide(double a,double b) const

{

return a/b;

}

// Lab Exercise 6: CalcTest.cpp

#include <iostream>

using namespace std;

#include "SimpleCalculator.h"

int main()

{

double a = 10.0;

double b = 20.0;

/\* Instantiate an object of type Simplecalculator \*/

SimpleCalculator sc;

cout << "The value of a is: " << a << "\n";

cout << "The value of b is: " << b << "\n\n";

/\* Write a line that adds a and b through your SimpleCalculator

object; assign the result to a variable named addition \*/

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

cout << "Adding a and b yields " << addition << "\n";

double subtraction = sc.subtract( a, b );

cout << "Subtracting b from a yields " << subtraction << "\n";

double multiplication = sc.multiply( a, b );

cout << "Multiplying a by b yields " << multiplication << "\n";

/\* Write a line that divides a and b through the

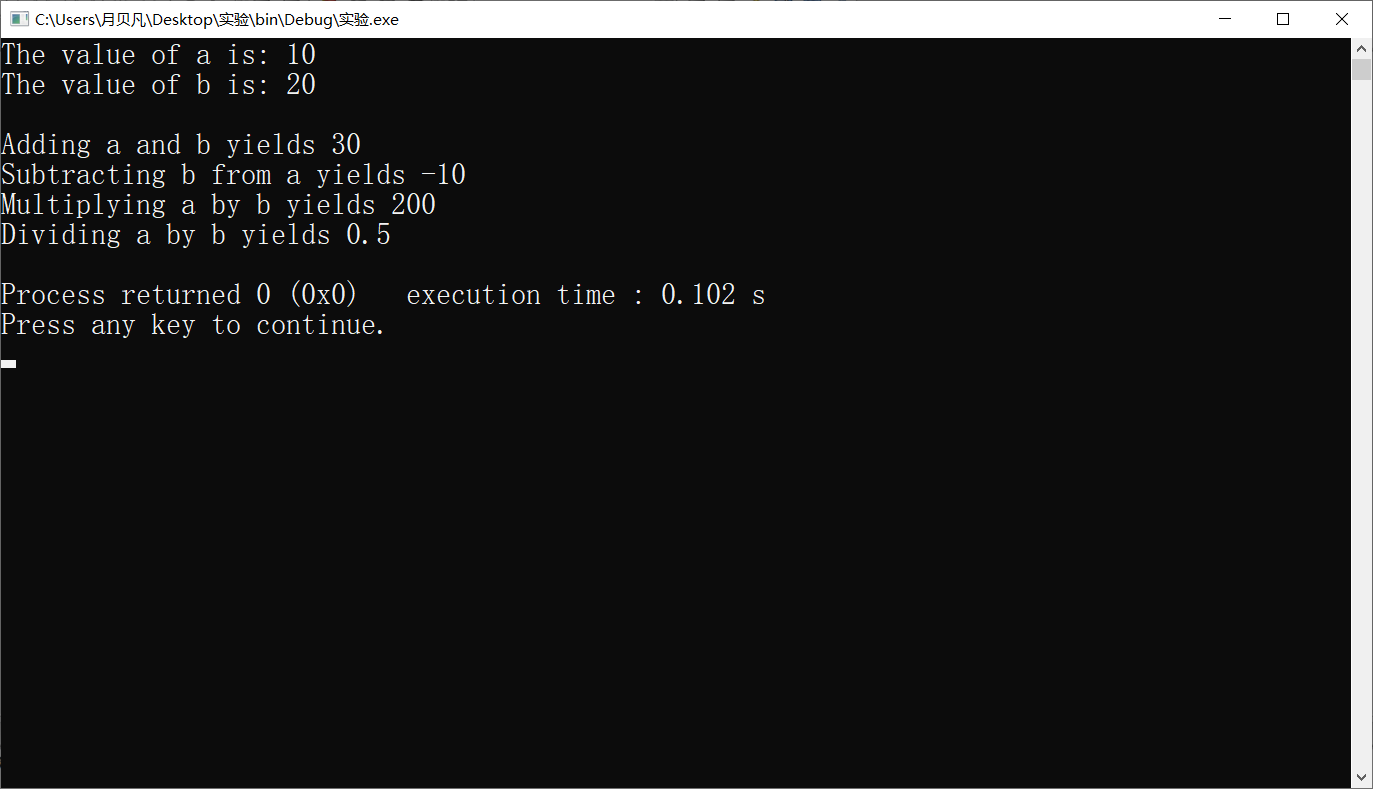
SimpleCalculator object; assign the result to a

variable named division \*/

double division = sc.divide(a,b);

cout << "Dividing a by b yields " << division << endl;

}



**V Follow-Up Questions and Activities**

1. Why doesn’t the SimpleCalculator class have a constructor?

该类不需要初始化，系统将分配默认参数。

1. Why are no private data members needed for class SimpleCalculator?

该类不需要维护任何私有数据来执行其算术计算。

1. Modify your class so that SimpleCalculator has a private data member called answer. After performing an operation, assign the result to answer. Add a member function named getAnswer to retrieve the result of the last arithmetic operation performed by the object. Also, add a constructor for class SimpleCalculator that initializes the value of answer to 0.

class SimpleCaculator

{

public:

SimpleCaculator()

{

answer = 0;

}

double add(double a, double b)

{

answer = a + b;

return a + b;

}

double subtract(double a, double b)

{

answer = a - b;

return a - b;

}

double multiply(double a, double b)

{

answer = a \* b;

return a \* b;

}

double divide(double a, double b)

{

answer = a / b;

return a / b;

}

double getAnswer()

{

return answer;

}

private:

double answer;

};

1. Modify the program so that the SimpleCalculator class has an input member function that allows the user to input two doubles. The function should then store the values that were input in private data members. Use these two values for each of the arithmetic calculations. Create two constructors for this class, one that takes no arguments and initializes a and b to 0 and another that takes two doubles and initializes a and b to those values. Finally, create a member function printValues that displays the values of a and b. A segment of the driver program might now look like this:



class SimpleCaculator

{

public:

SimpleCaculator()

{

answer = 0;

a=0;

b=0;

}

void input()

{

cin>>a>>b;

}

void privntValues()

{

cout<<"a is "<<a<<" and b is "<<b<<endl;

}

double add(double a, double b)

{

answer = a + b;

return a + b;

}

double subtract(double a, double b)

{

answer = a - b;

return a - b;

}

double multiply(double a, double b)

{

answer = a \* b;

return a \* b;

}

double divide(double a, double b)

{

answer = a / b;

return a / b;

}

double getAnswer()

{

return answer;

}

private:

double answer;

double a,b;

};

**\*Lab Exercise 7 — Integer Set**

**I Lab Objectives**

In this lab, you will practice:

1. Using classes to create a data type, IntegerSet, capable of storing a set of integers
2. Using dynamic memory allocation with the new and delete operators

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

1. Using destructors to deallocate memory that was dynamically allocated.

**II 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, aunionOfSets member function (already provided) creates a third set that is the set-theoretic union of two existing sets (i.e., an element of the third 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’s array is set to 0 if that element is 0 in each of the existing sets).

Provide an intersectionOfSetsmember function which creates a third set which is the set-theoretic intersection of 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 anisEqualTo 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.

**III Sample Output**



**IV Your Solution**

// Lab 7: IntegerSet.cpp

// Member-function definitions for class IntegerSet.

#include <iostream>

#include <iomanip>

using namespace std;

/\* Write include directive for IntegerSet.h here \*/

#include"IntegerSet.h"

// constructor creates a set from array of integers

IntegerSet::IntegerSet( int array[], int size)

{

emptySet();

for ( int i = 0; i < size; i++ )

insertElement( array[ i ] );

} // end IntegerSet constructor

/\* Write a definition for emptySet \*/

void IntegerSet::emptySet()

{

fill(set,set+101,0);

}

// input a set from the user

void IntegerSet::inputSet()

{

int number;

do

{

cout << "Enter an element (-1 to end): ";

cin >> number;

if ( validEntry( number ) )

set[ number ] = 1;

else if ( number != -1 )

cerr << "Invalid Element\n";

} while ( number != -1 ); // end do...while

cout << "Entry complete\n";

} // end function inputSet

// prints the set to the output stream

void IntegerSet::printSet() const

{

int x = 1;

bool empty = true; // assume set is empty

cout << '{';

for (int u = 0; u < 101; u++ )

{

if ( set[ u ] )

{

cout << setw( 4 ) << u << ( x % 10 == 0 ? "\n" : "" );

empty = false; // set is not empty

x++;

} // end if

} // end for

if ( empty )

cout << setw( 4 ) << "---"; // display an empty set

cout << setw( 4 ) << "}" << '\n';

} // end function printSet

// returns the union of two sets

IntegerSet IntegerSet::unionOfSets( const IntegerSet &r )

{

IntegerSet temp;

// if element is in either set, add to temporary set

for ( int n = 0; n < 101; n++ )

if ( set[ n ] == 1 || r.set[ n ] == 1 )

temp.set[ n ] = 1;

return temp;

} // end function unionOfSets

/\* Write definition for intersectionOfSets \*/

IntegerSet IntegerSet::intersectionOfSets( const IntegerSet &r )

{

IntegerSet temp;

// if element is in either set, add to temporary set

for ( int n = 0; n < 101; n++ )

if ( set[ n ] == 1 && r.set[ n ] == 1 )

temp.set[ n ] = 1;

return temp;

}

// insert a new integer into this set

void IntegerSet::insertElement( int k )

{

if ( validEntry( k ) )

set[ k ] = 1;

else

cerr << "Invalid insert attempted!\n";

} // end function insertElement

/\* Write definition for deleteElement \*/

void IntegerSet::deleteElement(int k)

{

if ( validEntry( k ) )

set[ k ] = 0;

else

cerr << "Invalid insert attempted!\n";

}

// determines if two sets are equal

bool IntegerSet::isEqualTo( const IntegerSet &r ) const

{

for ( int v = 0; v < 101; v++ )

if ( set[ v ] != r.set[ v ] )

return false; // sets are not-equal

return true; // sets are equal

} // end function isEqualTo

// Lab 7: IntegerSet.h

// Header file for class IntegerSet

#ifndef INTEGER\_SET\_H

#define INTEGER\_SET\_H

class IntegerSet

{

public:

// default constructor

IntegerSet()

{

/\* Write call to emptySet \*/

emptySet();

} // end IntegerSet constructor

IntegerSet( int [], int ); // constructor that takes an initial set

IntegerSet unionOfSets( const IntegerSet& );

/\* Write a member funcion prototype for intersectionOfSets \*/

IntegerSet intersectionOfSets( const IntegerSet& );

void emptySet(); // set all elements of set to 0

void inputSet(); // read values from user

void insertElement( int );

/\* Write a member function prototype for deleteElement \*/

void deleteElement(int);

void printSet() const;

/\* Write a member function prototype for isEqualTo \*/

bool isEqualTo(const IntegerSet &) const;

private:

int set[ 101 ]; // range of 0 - 100

// determines a valid entry to the set

int validEntry( int x ) const

{

return ( x >= 0 && x <= 100 );

} // end function validEntry

}; // end class IntegerSet

#endif

// Lab 7: SetTest.cpp

// Driver program for class IntegerSet.

#include <iostream>

using namespace std;

#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 \*/

c=a.unionOfSets(b);

/\* Write call to intersectionOfSets for object a,

passing b as argument and assigning the result to d \*/

d=a.intersectionOfSets(b);

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 };

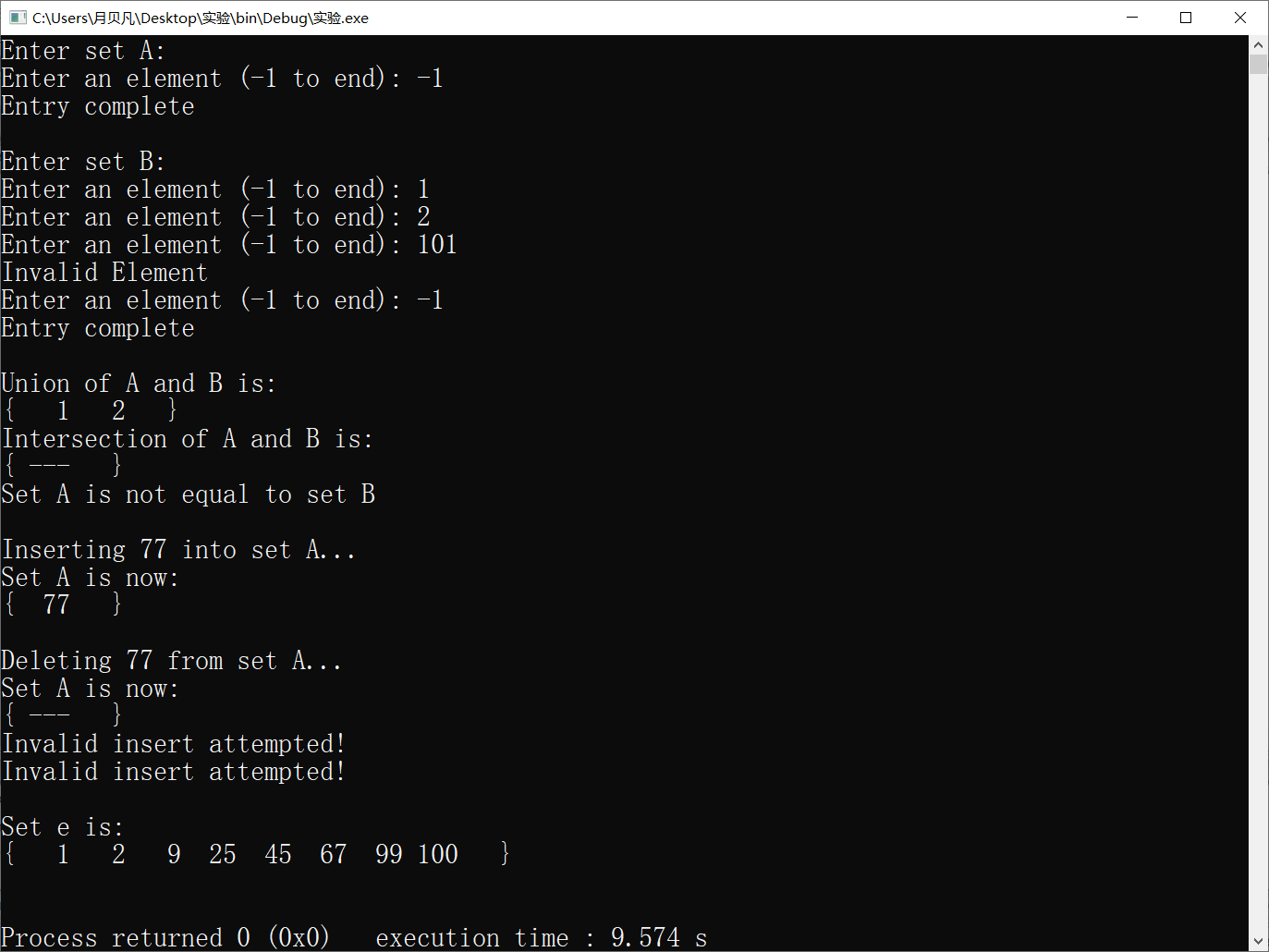
IntegerSet e( intArray, arraySize );

cout << "\nSet e is:\n";

e.printSet();

cout << endl;

} // end main



**V Follow-Up Questions and Activities**

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

如果声明一个类的一个const实例，那么它就只能调用有const修饰的函数。在加减法的参数列表中使用const，即为不想对该参数列表数据成员做任何改动。