**Lab Exercises**

Class:  软件1801 Name: 徐应成

StuID: 201816040121

Date: 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( int ); // constructor initializes balance

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

void debit( int );//withdrawa an amount from the account balance

int getBalance(); // return the account balance

void juge( int );//juge the value of account balance

private:

int 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( int 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

//debit an amount from the account balance

void Account::debit(int amount)

{

balance = balance - amount;//withdrawa amount from balance

}//end function debit

// return the account balance

int Account::getBalance()

{

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

} // end function getBalance

//juge the account balance's value

void Account::juge(int value)

{

if(value < 0)

{

cout << "Debit amount exceeded account balance!\n" <<endl;

}

else

{

balance = value;

}

}

// 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 ); // create Account object

Account account2( 0 ); // create Account object

// display initial balance of each object

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

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

int 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";

account1.debit(withdrawalAmount);//withdraw money from account1

// 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";

account2.debit(withdrawalAmount);//withdraw money from account2

int account2balance = account2.getBalance();

account2.juge(account2balance);//juge the value of accoun2 balance

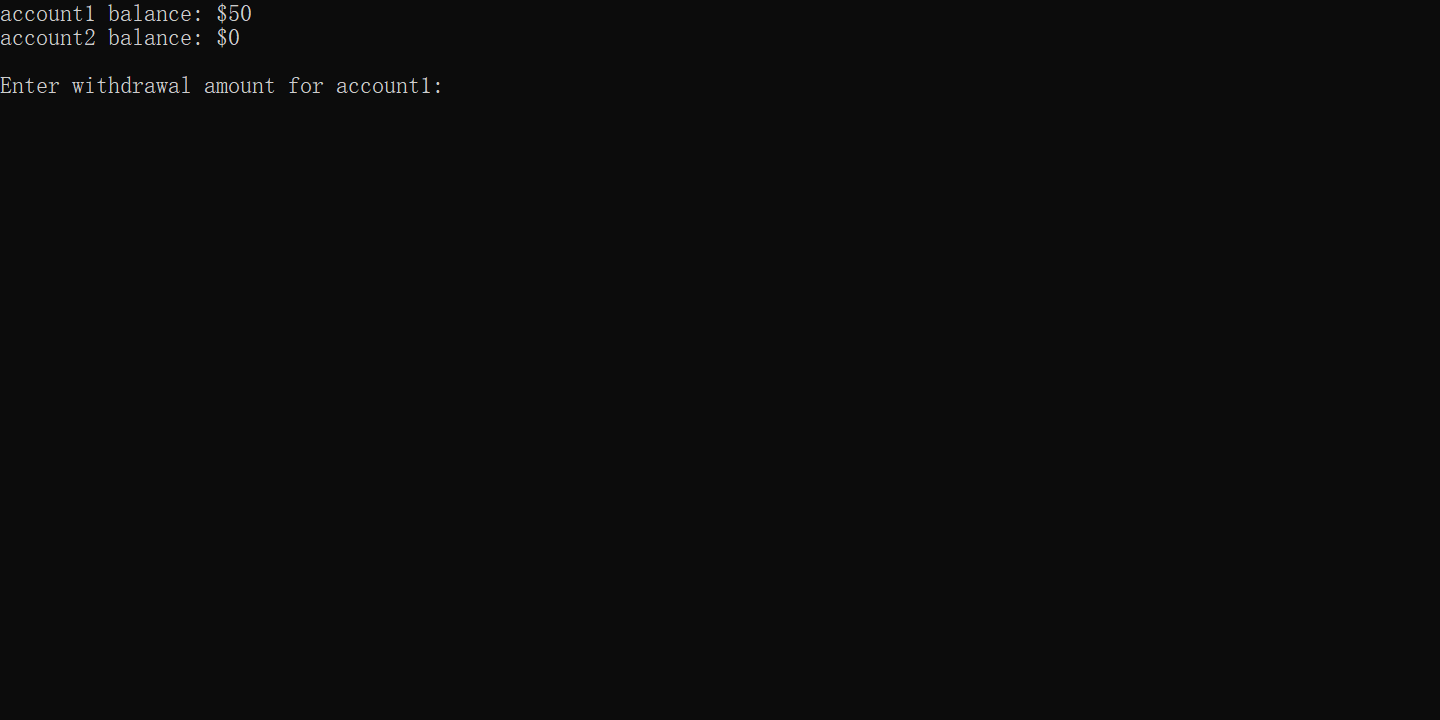
account2.credit(withdrawalAmount);//add amount to account2

// 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

void setInstructorName(string);//fuction to set the instructor name

string getInstructorName();//fuction to retrieve the instructor name

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

//fuction to set the instructor name

void GradeBook::setInstructorName(string name)

{

instructorName = name;

}//end fuction setInstructorname

//fuction to retrieve the course name

string GradeBook::getInstructorName()

{

return instructorName;

}//end fuction getInsturctiorNmae

// 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;

//display the instructor name

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

"CS101 Introduction to C++ Programming" ,

"Sam Smith");

// display welcome message and instructor's name

gradeBook.displayMessage();

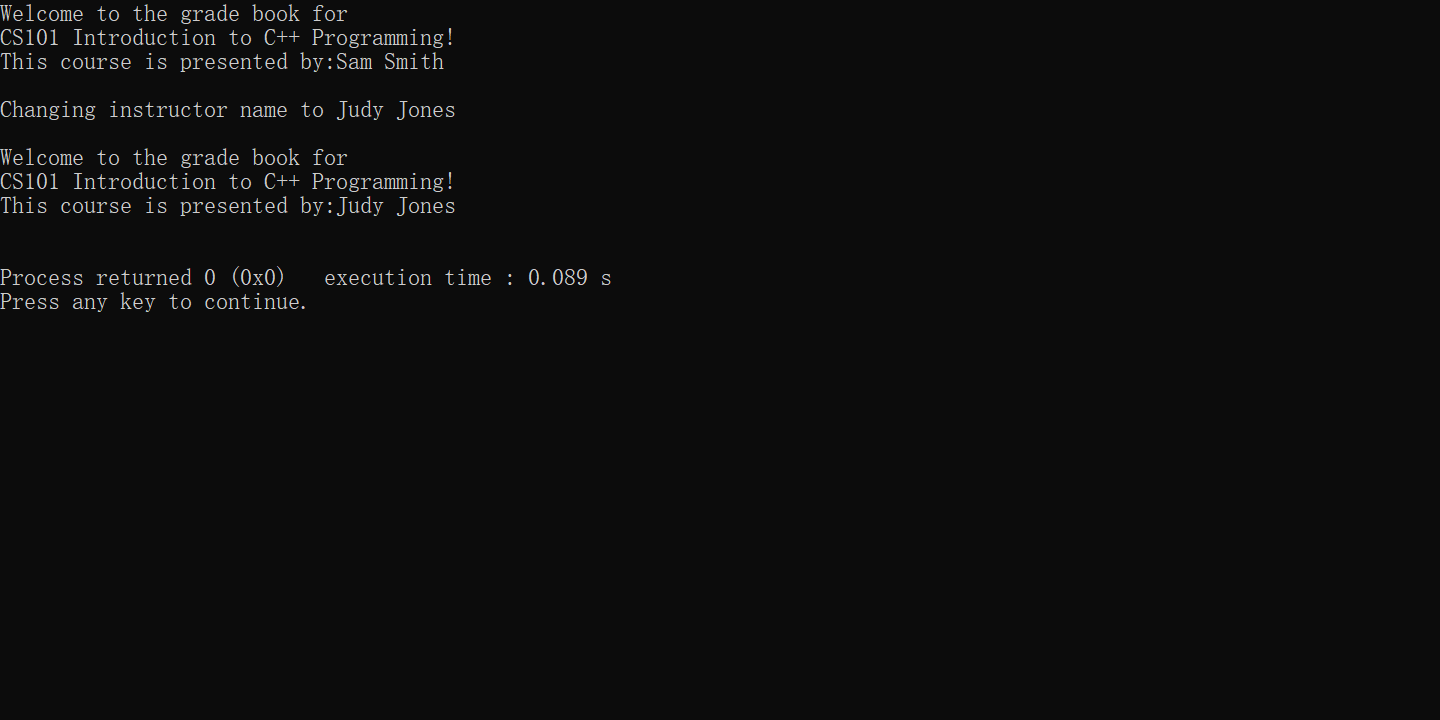
cout << "Changing instructor name to Judy Jones\n" << endl;

gradeBook.setInstructorName("Judy Jones");//change instructor name to Judy Jones

//display welcome message and instructor'sname

gradeBook.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:

//constructor initializes first name, last name, and monthly salary

Employee(string, string, int);

void setFirstName(string);//fuction to set the first name

string getFirstName();//fuction to retrieve the first name

void setLastName(string);//fuction to set the last name

string getLastName();//fuction to retrieve the last name

void setMonthlySalary(int);//fuction to set the employee's monthly salary

int getMonthlySalary();//fuction to get the employee's monthly salary

int calculationSalary();//fuction to calculation salary

private:

string firstName;//first name for this Employee

string lastName;//last name foor this Employee

int monthlySalary;//monthly salary for this Employee

}; // end class Employee

// Lab 3: Employee.cpp

// Employee class member-function definitions.

#include <iostream>

using namespace std;

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

//constructor initializes first name, last name, and monthly salary

Employee::Employee(string firstName, string lastName, int initializeSalary)

{

setFirstName(firstName);

setLastName(lastName);

if(initializeSalary < 0)

{

monthlySalary = 0;

}

else

{

monthlySalary = initializeSalary;

}

}//end Employee constructor

//fuction to set the first name

void Employee::setFirstName(string first)

{

firstName = first;

}//end fuction setFirstName

//fuction to get the first name

string Employee::getFirstName()

{

return firstName;

}//end fuction getfirstName

//fuction to set the last name

void Employee::setLastName(string last)

{

lastName = last;

}//end fuction setLastName

//fuction to get the last name

string Employee::getLastName()

{

return lastName;

}//end fuction getLastName

//fuction to get monthly salary

void Employee::setMonthlySalary(int value)

{

monthlySalary = value;

}//end fuction setMontlySalary

//fuction to get the montly salary

int Employee::getMonthlySalary()

{

return monthlySalary;

}//end fuction getMonthlySalary

//fuction to calculation MontlySalary

int Employee::calculationSalary()

{

int value = getMonthlySalary();

value\*=1.1;

return value;

}

// 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", 34500);

Employee employee2("Susan", "Baker", 37800);

//display initial data of each object

cout << "Employee 1:" << employee1.getFirstName() << " " << employee1.getLastName()

<< ";" << "Yearly Salary: " << employee1.getMonthlySalary() << endl;//display the message of employee1

cout << "Employee 2:" << employee2.getFirstName() << " " << employee2.getLastName()

<< ";" << "Yearly Salary: " << employee2.getMonthlySalary() << endl;//display the message of employee2

cout << "\n" <<endl;

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

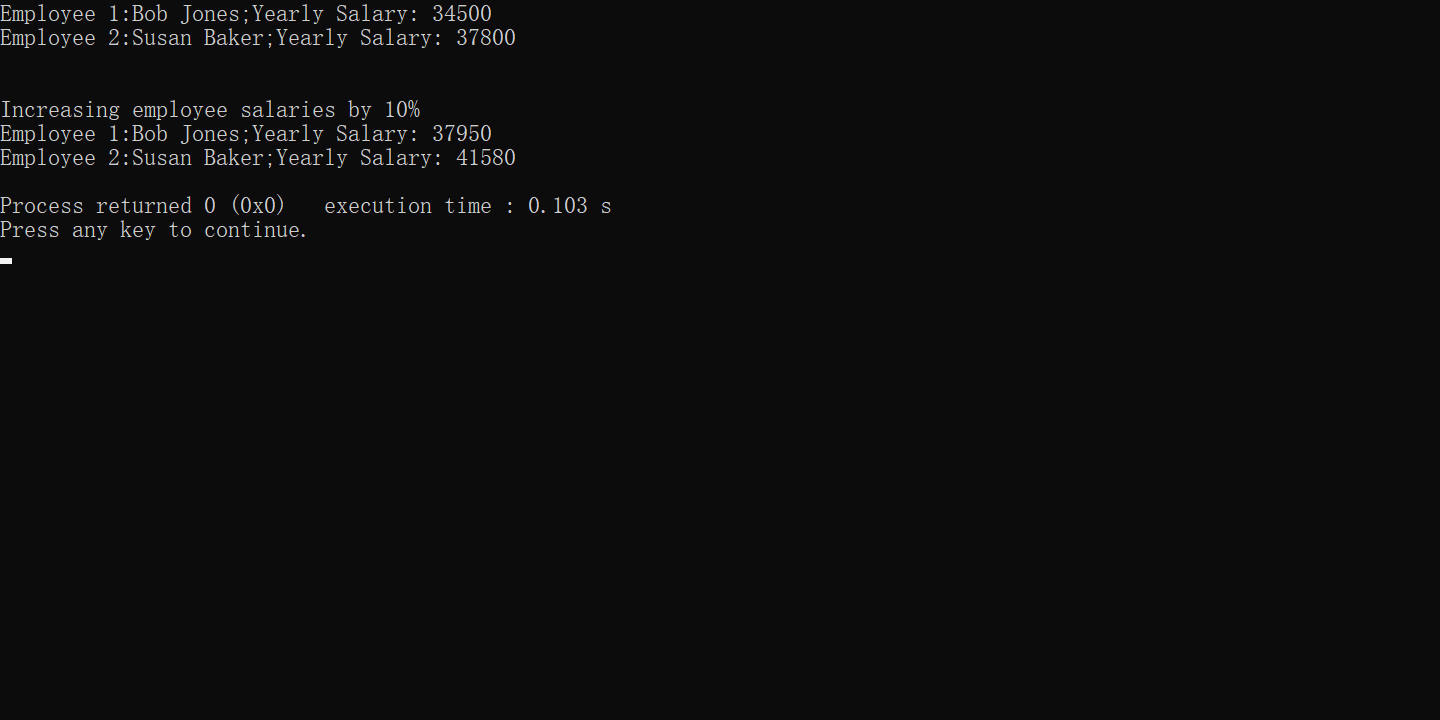
cout << "Employee 1:" << employee1.getFirstName() << " " << employee1.getLastName()

<< ";" << "Yearly Salary: " << employee1.calculationSalary() << endl;//display the message of employee1 after increasing salaries

cout << "Employee 2:" << employee2.getFirstName() << " " << employee2.getLastName()

<< ";" << "Yearly Salary: " << employee2.calculationSalary() << endl;//display the message of employee2 after increasing salaries

} // 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**

//Lab4:Complex.h

//Complex class definition

#include <string>

#include <iostream>

#ifndef COMPLEX\_H

#define COMPLEX\_H

using namespace std;

//Complex class defition

class Complex

{

public:

//constructor realpart, imaginarypart

Complex(double r = 0.0, double i = 0.0);

void setComplexNumber(double, double);//fuction set the number

Complex add(const Complex& n);//fuction to calculate the sum

Complex subtract(const Complex& n);//fuction to calculate the difference

void printComplex();//fuction to print the result

private:

double real;//real for this Complex

double imaginary;//imaginary for this Complex

};

#endif

//lab4:Complex.cpp

#include <iostream>

using namespace std;

#include "Complex.h"

//constructor realpart, imaginarypart

Complex::Complex(double r, double i)

{

setComplexNumber(r, i);

}

//fuction to set the number

void Complex::setComplexNumber(double r, double i)

{

real = r;

imaginary = i;

}

//fuction to calculate the sum

Complex Complex::add(const Complex& n)

{

Complex result;

result.real = real + n.real;

result.imaginary = imaginary + n.imaginary;

return result;

}

//fuction to calculate the difference

Complex Complex::subtract(const Complex& n)

{

Complex result;

result.real = real - n.real;

result.imaginary = imaginary - n.imaginary;

return result;

}

//fuction to print

void Complex::printComplex()

{

cout << '(' << real << ", " << imaginary << ')';

}

// 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

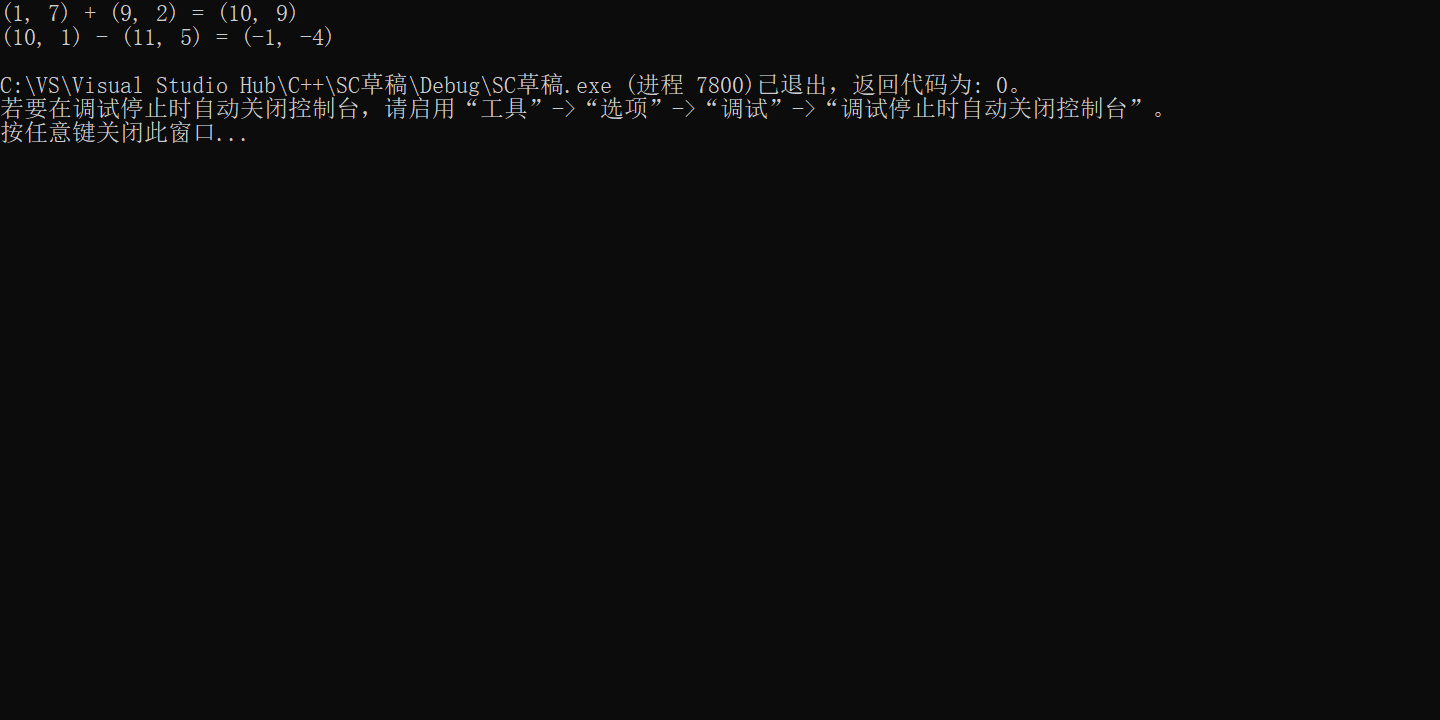
cout << " = ";

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?
2. 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.
3. Declare a Complex number, as follows, without passing any arguments to the constructor. What happens?Does the default constructor get called?

Complex a;

Answer：

1. 因为要用const关键字来指定传入加减法函数中的参数不可修改，防止影响计算结果。
2. //Lab4:Complex.h

//Complex class definition

#include <string>

#include <iostream>

#ifndef COMPLEX\_H

#define COMPLEX\_H

using namespace std;

//Complex class defition

class Complex

{

public:

//constructor realpart, imaginarypart

Complex(double r = 0.0, double i = 0.0);

void setComplexNumber(double, double);//fuction set the number

Complex add(const Complex n);//fuction to calculate the sum

Complex subtract(const Complex n);//fuction to calculate the difference

void printComplex();//fuction to print the result

private:

double real;//real for this Complex

double imaginary;//imaginary for this Complex

};

#endif

3）会发生编译错误，因为类的定义中没有定义默认构造函数，声明一个没有参数的对象a，编译器并不会自动生成一个默认构造函数来初始化它，从而编译出错。

**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

~Date();//destructor

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

void addDays();//fuction to add a day

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

Date::~Date()

{

}

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

void Date::addDays()

{

int d, m, y;

d = getDay();

m = getMonth();

y = getYear();

d += 1;

if(getMonth() == 12 && getDay() == 31)

{

m = 1;

y += 1;

}

setMonth(m);

setDay(d);

setYear(y);

}

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 )

{

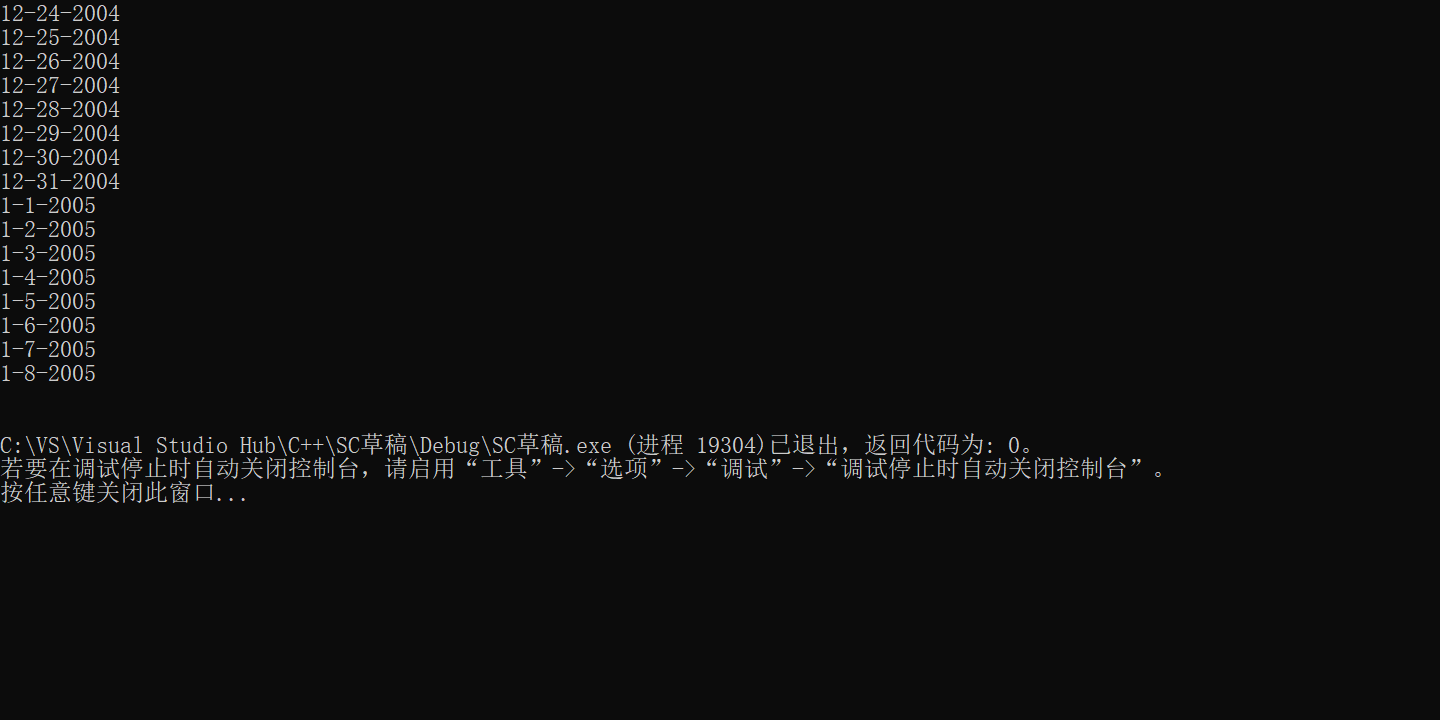
d.print(); // invokes function print

d.addDays();//fuction to add one day

} // 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?
2. What happens when a member function that takes no arguments is called without the parentheses (i.e.,dateObject.nextDay)?
3. Write a destructor for the Date class. The destructor should print text indicating that the destructor for the Date class was called successfully.
4. 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?

Answer：

1. 可以有多个构造函数，当定义了带有参数的构造函数时，还需要定义默认构造函数来初始化不含参数的对象。
2. 编译器会报错。
3. Date::~Date()

{

cout << “This object was destructor.” << endl;

}

4) 无法改变值，因为这条语句没有权限访问类的Year数据成员。

**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:

SimpleCalculator(double a = 0.0, double b = 0.0);//fuction to constructor

double add( double, double ) const;//fuction to add

double subtract( double, double ) const;//fuction to subtract

double multiply( double, double ) const;//fuction to multiply

double divide( double, double ) const;//fuction to divide

private:

double m;

double n;

}; // end class SimpleCalculator

// Lab Exercise 6: SimpleCalculator.cpp

#include "SimpleCalculator.h"

//fuction to constructor

SimpleCalculator::SimpleCalculator( double a, double b )

{

m = a;

n = b;

}

//fuction to add

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

//fuction to divide

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

{

return a / b;

}// end fuction divide

// Lab Exercise 6: CalcTest.cpp

#include <iostream>

using namespace std;

#include "SimpleCalculator.h"

int main()

{

double a = 10.0;

double b = 20.0;

SimpleCalculator sc;

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

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

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

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";

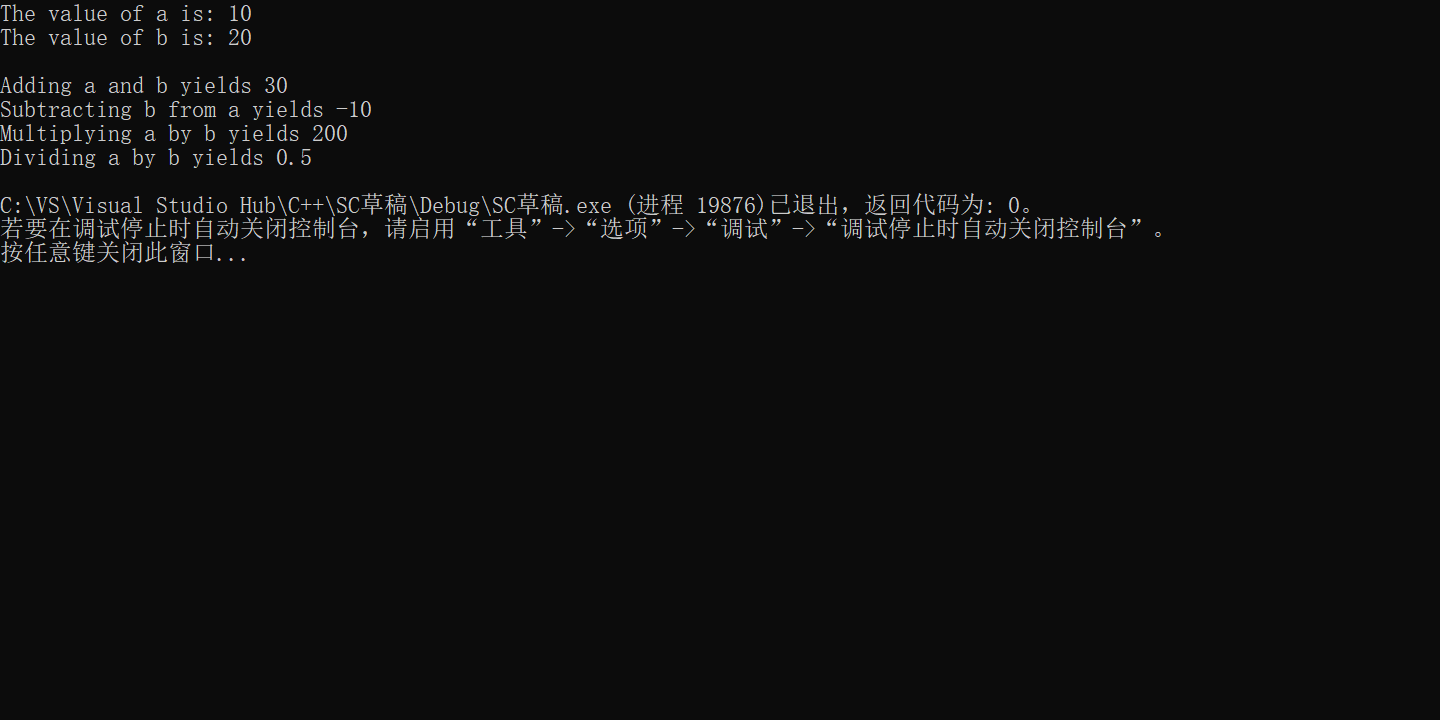
double multiplication = sc.multiply( a, b );

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

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?
2. Why are no private data members needed for class SimpleCalculator?
3. 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.
4. 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:



**Answer：**

1. 因为对象sc在定义时并没有参数，所以对于这种无参数的对象，编译器会自动提供一个默认构造函数来初始化对象，因此我们不需要再显式地定义构造函数。
2. 因为类所定义的对象希望它的参数能够被其他函数（main）访问，即将两个double类型的参数传入类的对象完成对象的初始化。

**3）**// Lab Exercise 6: SimpleCalculator.h

// class SimpleCalculator definition

class SimpleCalculator

{

public:

SimpleCalculator(double a = 0.0, double b = 0.0);//fuction to constructor

SimpleCalculator();//constructor answer

double add(double, double) const;//fuction to add

double subtract(double, double) const;//fuction to subtract

double multiply(double, double) const;//fuction to multiply

double divide(double, double) const;//fuction to divide

double getAnswer();//return result of the last calculator

private:

double m;

double n;

double answer;

}; // end class SimpleCalculator

// Lab Exercise 6: SimpleCalculator.cpp

#include "SimpleCalculator.h"

//fuction to constructor

SimpleCalculator::SimpleCalculator(double a, double b)

{

m = a;

n = b;

}

//fuction to constructor answer

SimpleCalculator::SimpleCalculator()

{

answer = 0.0;

}

//fuction to add

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

{

answer = a + b;

return answer;

}

4）// Lab Exercise 6: SimpleCalculator.h

// class SimpleCalculator definition

class SimpleCalculator

{

public:

SimpleCalculator(double a = 0.0, double b = 0.0);//fuction to constructor

SimpleCalculator(a= 0.0, b = 0.0);//default constructor

SimpleCalculator(double, double);

void input(double, double);//input two data to object

double add(double, double) const;//fuction to add

double subtract(double, double) const;//fuction to subtract

double multiply(double, double) const;//fuction to multiply

double divide(double, double) const;//fuction to divide

void printValues();//fuction to print values

private:

double a;

double b;

double m;

double n;

}; // end class SimpleCalculator

// Lab Exercise 6: SimpleCalculator.cpp

#include "SimpleCalculator.h"

//fuction to constructor

SimpleCalculator::SimpleCalculator(double a, double b)

{

m = a;

n = b;

}

SimpleCalculator::SimpleCalculator(double x, double y)

{

a = x;

b = y

}

//fuction to input data

void SimpleCalculator::input(double x, double y)

{

a = x;

b = y;

}

//fuction to add

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

//fuction to divide

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

{

return a / b;

}

//fuction to print values

void SimpleCalculator::printValues()

{

cout << a << endl;

cout << b << endl;

}

**\*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**

**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?