**Lab Exercises**

Class: 软件工程1802 Name: 刘雨萌 StuID:\_\_\_\_\_201816040225\_\_\_\_\_\_\_\_\_\_

Date: 2019.11.10 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.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

int Account::debit(int money)

{

if(balance>=money)//Judge the amount

{

balance=balance-money;//Do subtraction

return 1;

}

else

cout<<"the money is more than balance,try again";//stay

}

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

// return the account balance

int 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 ); // 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";

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

if(!account1.debit(withdrawalAmount))//Judge

{

cout<<"\nDebit amount exceeded amount balance."<<endl;

}

// 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 \*/

if(!account2.debit(withdrawalAmount))//Judge

{

cout<<"\nDebit amount exceeded amount balance."<<endl;

}

// display balances

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

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

} // end main

// 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

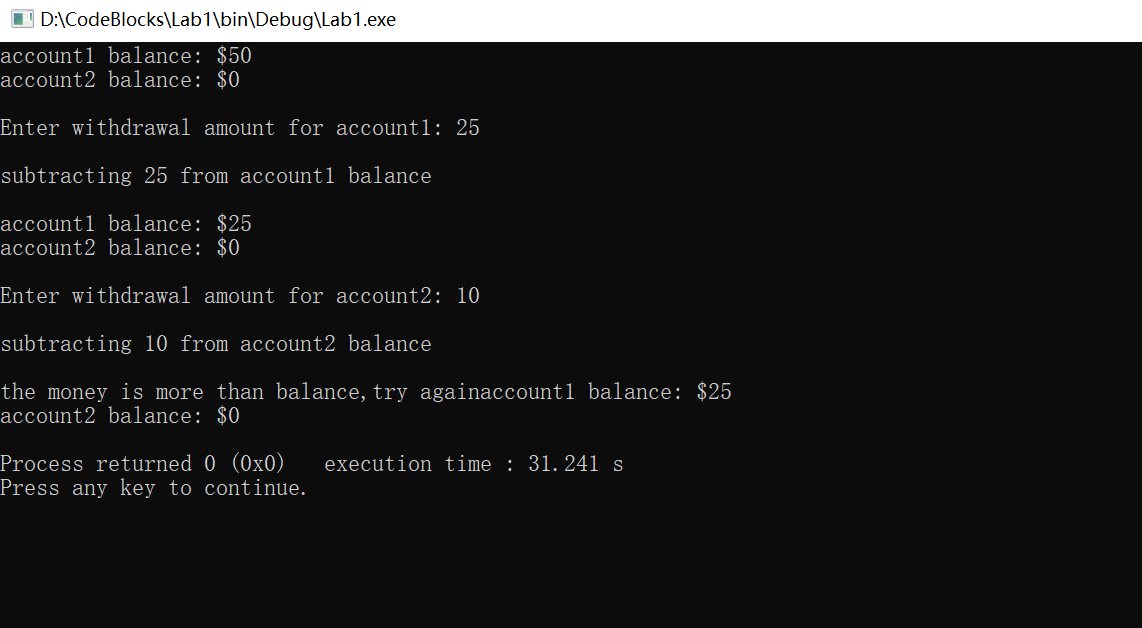
int debit( int );/\* write code to declare member function debit. \*/

int getBalance(); // return the account balance

private:

int balance; // data member that stores the balance

}; // end class Account



**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: 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()

{

string name;

// 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 << "\nChanging instructor name to ";

getline(cin, name);

gradeBook.setInstructorName(name);

cout << endl;

gradeBook.displayMessage();

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

} // end main

// 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 name)

{

instructorName=name;

}

// 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() << "!\n"

<< "This course is presented by: " << getInstructorName()

<< endl;

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

} // end function displayMessage

// 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 \*/

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

void setInstructorName(string);

string getInstructorName();

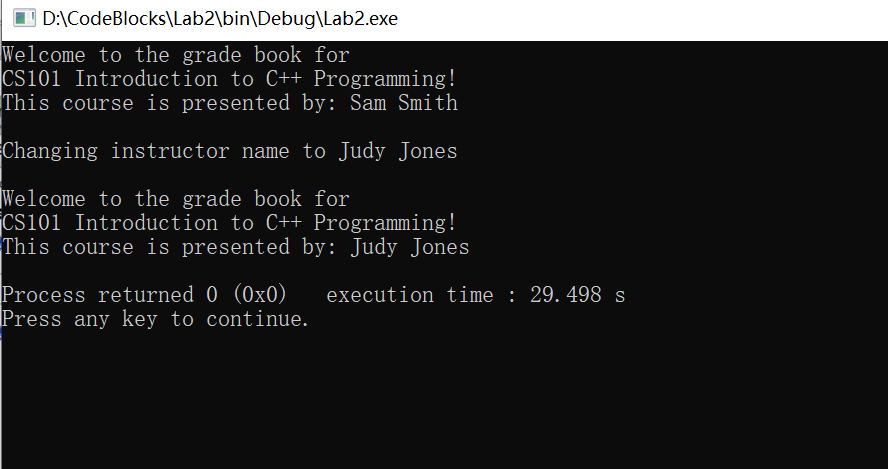
void displayMessage(); // display welcome message and instructor name

private:

string courseName; // course name for this GradeBook

string instructorName; // instructor name for this GradeBook

}; // end class GradeBook



**Lab 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 setEmployeefName( string );

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

string getEmployeefName();

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

void setEmployeelName( string );

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

string getEmployeelName();

/\* 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();

void finalsalary();

private:

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

string EmployeefName;

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

string EmployeelName;

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

int 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 name1,string name2,int salary1)

:EmployeefName(name1), EmployeelName(name2), salary(salary1)

{

}

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

void Employee::setEmployeefName(string name1)

{

EmployeefName = name1;

}

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

string Employee::getEmployeefName()

{

return EmployeefName;

}

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

void Employee::setEmployeelName(string name2)

{

EmployeelName = name2;

}

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

string Employee::getEmployeelName()

{

return EmployeelName;

}

/\* 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 salary1)

{

salary = salary1;

}

int Employee::getsalary()

{

return salary;

}

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

void Employee::finalsalary()

{

salary\*=1.1;

}

// 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 member1("Bob", "Jones", 34500);

Employee member2("Susan", "Baker", 37800);

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

cout<<"member1:"<<member1.getEmployeefName()<<" "<<member1.getEmployeelName()<<" "<<"monthly salary is: "<<member1.getsalary()<<endl;

cout<<"member2:"<<member2.getEmployeefName()<<" "<<member2.getEmployeelName()<<" "<<"monthly salary is: "<<member2.getsalary()<<endl;

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

member1.finalsalary();

member2.finalsalary();

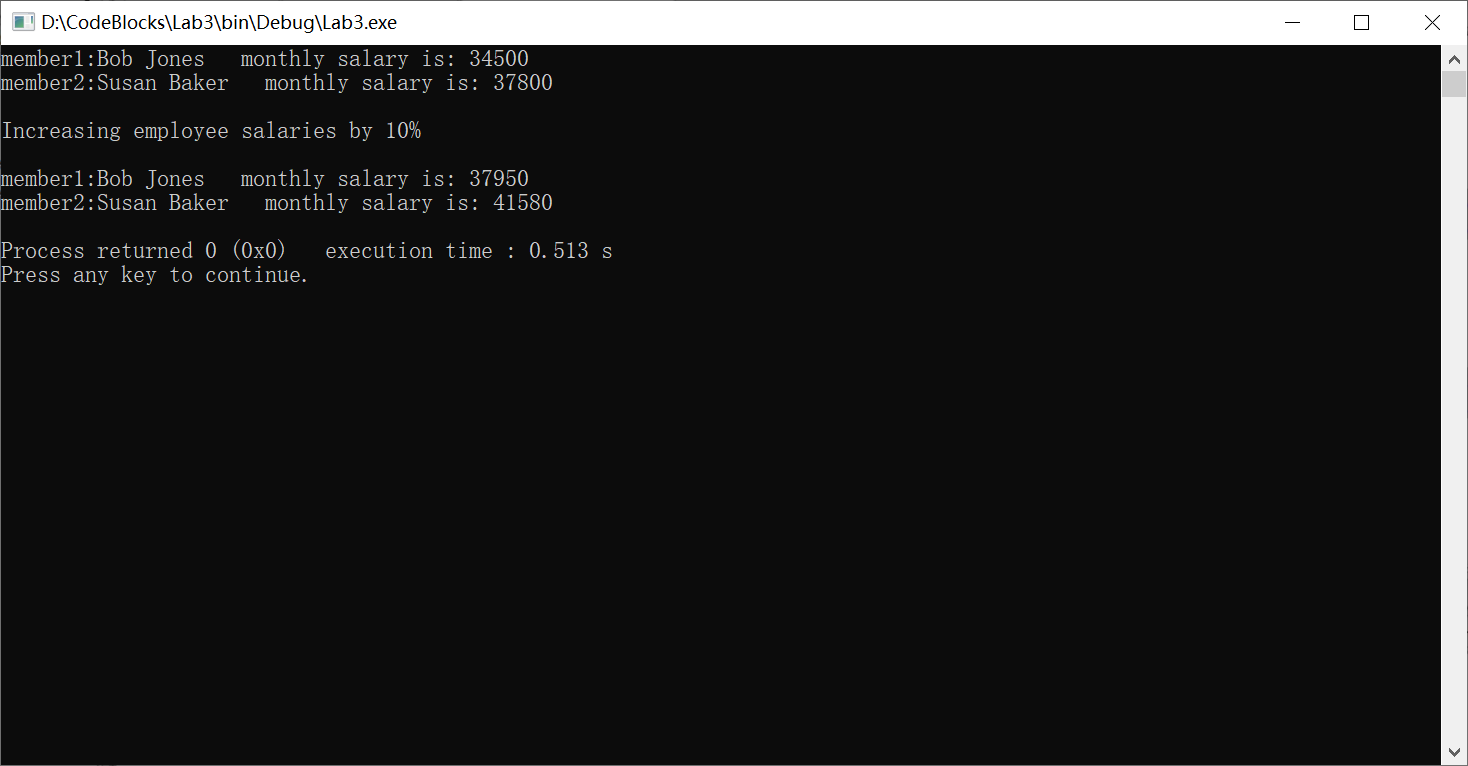
cout << "\nIncreasing employee salaries by 10%\n" << endl;

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

cout<<"member1:"<<member1.getEmployeefName()<<" "<<member1.getEmployeelName()<<" "<<"monthly salary is: "<<member1.getsalary()<<endl;

cout<<"member2:"<<member2.getEmployeefName()<<" "<<member2.getEmployeelName()<<" "<<"monthly salary is: "<<member2.getsalary()<<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

class Complex{

public:

Complex(double,double);//create constructor

Complex add( const Complex &right );//add function

Complex subtract( const Complex &right );//subtract function

void printComplex();//print function

void setComplexNumber( double,double ) ;//new complexnumber

private://data member

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 )//Complex constructor

{

setComplexNumber( real, imaginary );

}

Complex Complex::add( const Complex &right )//function add

{

realPart=realPart+right.realPart;

imaginaryPart= imaginaryPart+right.imaginaryPart;

return \*this;//get the result

}

Complex Complex::subtract( const Complex &right )//function subtract

{

realPart=realPart-right.realPart;

imaginaryPart=imaginaryPart-right.imaginaryPart;

return \*this;//get the result

}

void Complex::printComplex()//function printComplex

{

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

}

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(0,0); // three Complex objects

a.printComplex(); // output a

cout << " + ";

b.printComplex(); // output b

cout << " = ";

c= a.add( b ); // add to c

c.printComplex(); // output c

cout << '\n';

a.setComplexNumber( 10, 1 ); // new part

b.setComplexNumber( 11, 5 ); // new part

a.printComplex(); // output a

cout << " - ";

b.printComplex(); // output b

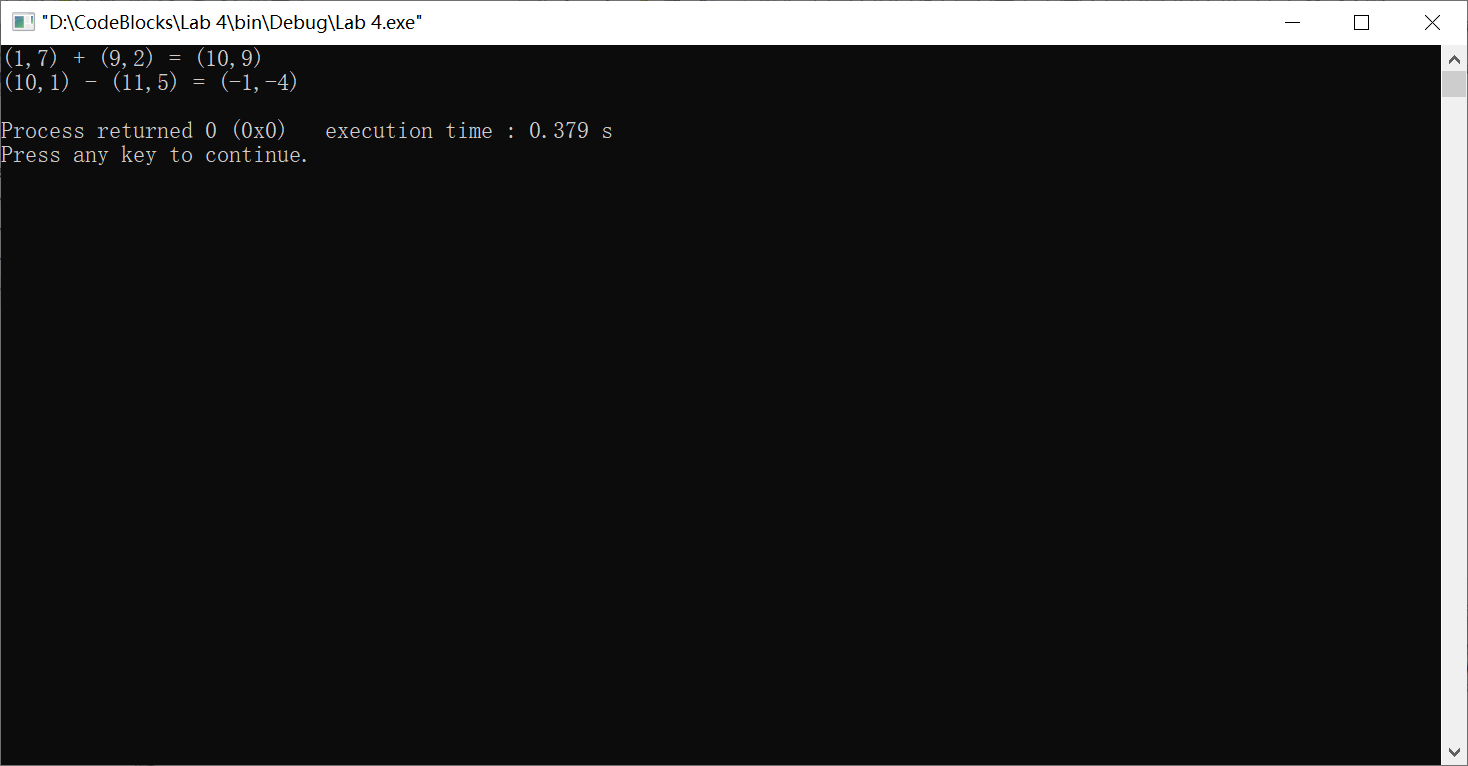
cout << " = ";

c = a.subtract( b ); //add function 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;

1. Only data members can be read，not modified.
2. Can.

#include<iostream>

#include<string>

using namespace std;

class A//A class

{

public:

void change(int c,int d)//function to change

{

c=3;

d=5;

cout<<"a= "<<a<<endl;

cout<<"b= "<<b<<endl;

}

private://data member

int a;

int b;

};

int main()

{

int a=1;int b=1;

A bbb;

bbb.change(a,b);//show the a, b

}

3)The constructor implicitly invokes the default constructor for each data member and initializes it as an empty string. It will call.

**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**

//Lab5:Date.h

#ifndef DATE\_H

#define DATE\_H

//define a Date class

class Date

{

public:

Date( int=1, int=1, int=2000);//Define the date type

void printDate();//print

void setDate( int,int,int);//a function to set

void setMonth( int );//a function to set

void setDay(int);//a function to set

void setYear(int);//a function to set

int getMonth();//a function to get

int getYear();//a function to get

int getDay();//a function to get

int nextDate();//a function to nextDate

private://data member

int Month;

int Day;

int Year;

bool leapYear();

int monthDays();

};

#endif // DATE\_H

//Lab5:Date.cpp

#include<iostream>

using namespace std;

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

Date::Date(int month1,int day1,int year1)//constructor data

{

setDate(month1,day1,year1);

}

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

{

setMonth(mo);

setDay(dy);

setYear(yr);

}

void Date::setMonth(int month1)//set month

{

Month=(month1>=1&& month1<=12)?month1:1;

}

int Date::getMonth()//get month

{

return Month;

}

void Date::setDay(int day1)//set day

{

if ( Month == 2&&leapYear())

Day = ( day1 <= 29&&day1 >= 1 ) ? day1 : 1;

else

Day = ( day1<= monthDays()&&day1 >= 1 ) ? day1 : 1;

}

int Date::getDay()//get day

{

return Day;

}

void Date::setYear(int year1)//set year

{

Year=year1>=1900?year1:1900;

}

int Date::getYear()//get year

{

return Year;

}

void Date::printDate()//function to print

{

cout << Month << '-' << Day << '-' << Year << '\n';

}

int Date::nextDate()//next date

{

if(Day<monthDays())//one month' date

{

Day=Day+1;

Month=Month;

Year=Year;

}

else//over one month to next month

{

Day-=monthDays()-1;

Month=Month+1;

if(Month>12)

{

Month=Month-12;

Year=Year+1;

}

}

}

bool Date::leapYear()//judge the year

{

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

return true;

else

return false;

}

int Date::monthDays()//12 months

{

const int days[12]=

{

31,28,31,30,31,30,31,31,30,31,30,31

};

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

}

//Lab5 DateTest.cpp

#include<iostream>

using namespace std;

#include"Date.h"

int main()

{

const int MAXDAYS=16;

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

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

{

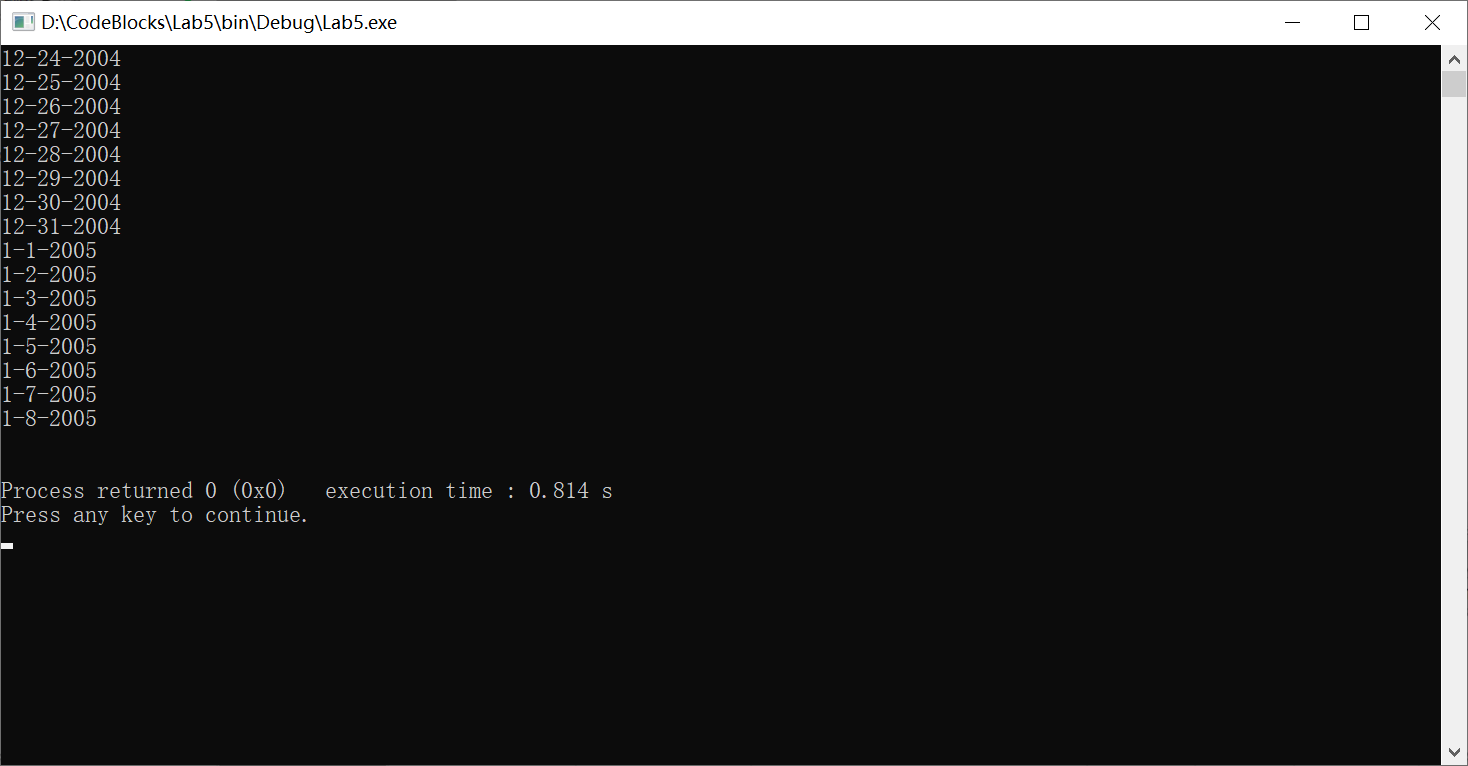
d.printDate();//print

d.nextDate();//next Date print

}

cout<<endl;

}



**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?

1):Each class can provide one or more constructors.

2):There is no way to pass parameters.

3): #include<iostream>

using namespace std;

class Date

{

public:

Date();

~Date();

};

Date::Date()

{

cout<<"use function"<<endl;

}

Date::~Date()

{

cout<<"use destructor"<<endl;

}

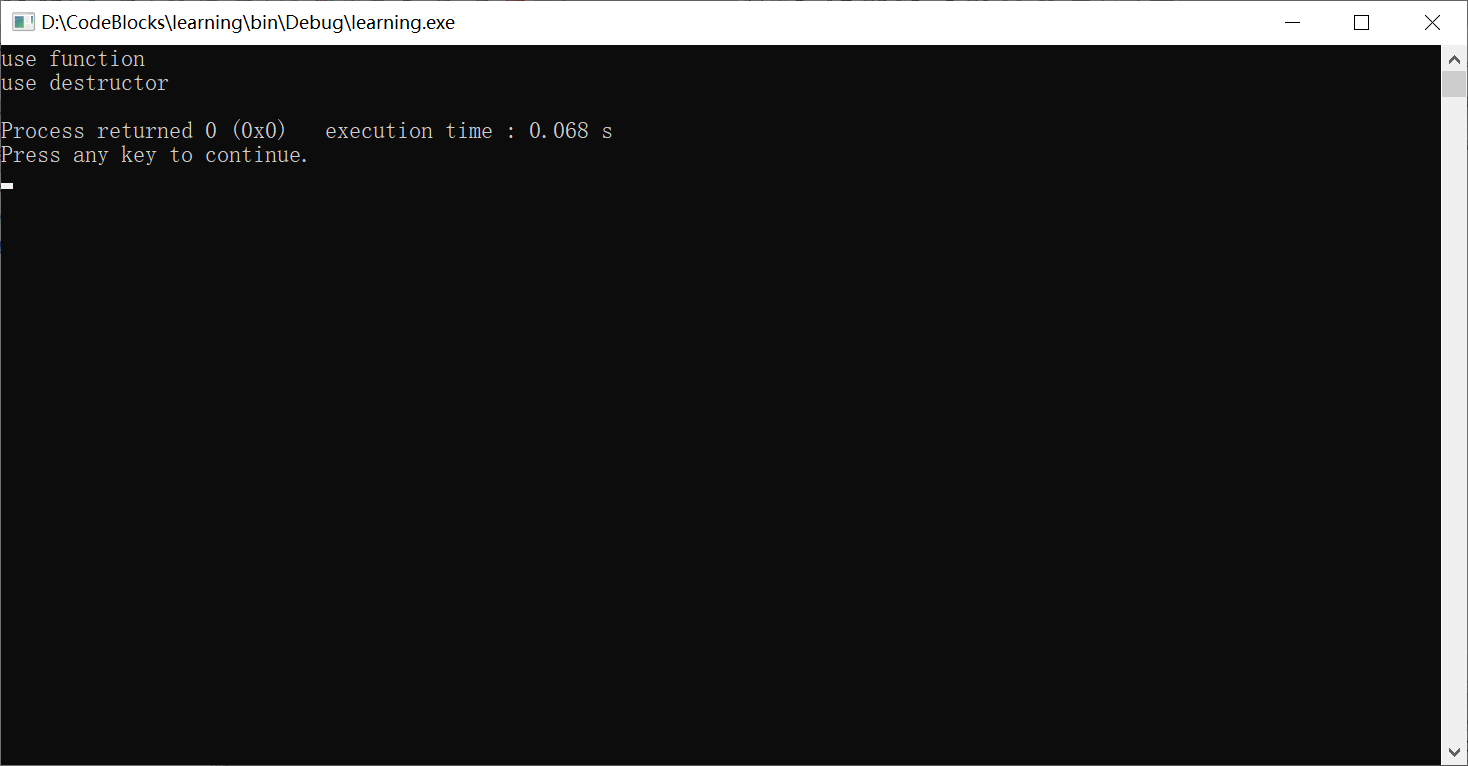
int main()

{

Date test;

return 0;

}



4)You cannot directly change the element value. Delete the old element before inserting the new one.

**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 6:SimpleCalculator.h

class SimpleCalculator

{

public:

double add(double,double)const;//add function

double subtract(double,double)const;//subtraction function

double multiply(double,double)const;//multiply function

double divide(double,double)const;//divide function

};

//Lab6:SimpleCalculator.cpp

#include"SimpleCalculator.h"

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

{

return a+b;

}

double SimpleCalculator::subtract(double a,double b)const//subtract function

{

return a-b;

}

double SimpleCalculator::multiply(double a,double b)const//multiply function

{

return a\*b;

}

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

{

return a/b;

}

//Lab6:SimpleCalculatorTest.cpp

#include<iostream>

using namespace std;

#include"SimpleCalculator.h"

//define the class

int main()

{

SimpleCalculator sc;//define the object

double a=10.0;//data

double b=20.0;//data

cout<<"The value of a is: "<<a<<"\n"//show meaning

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

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

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

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

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

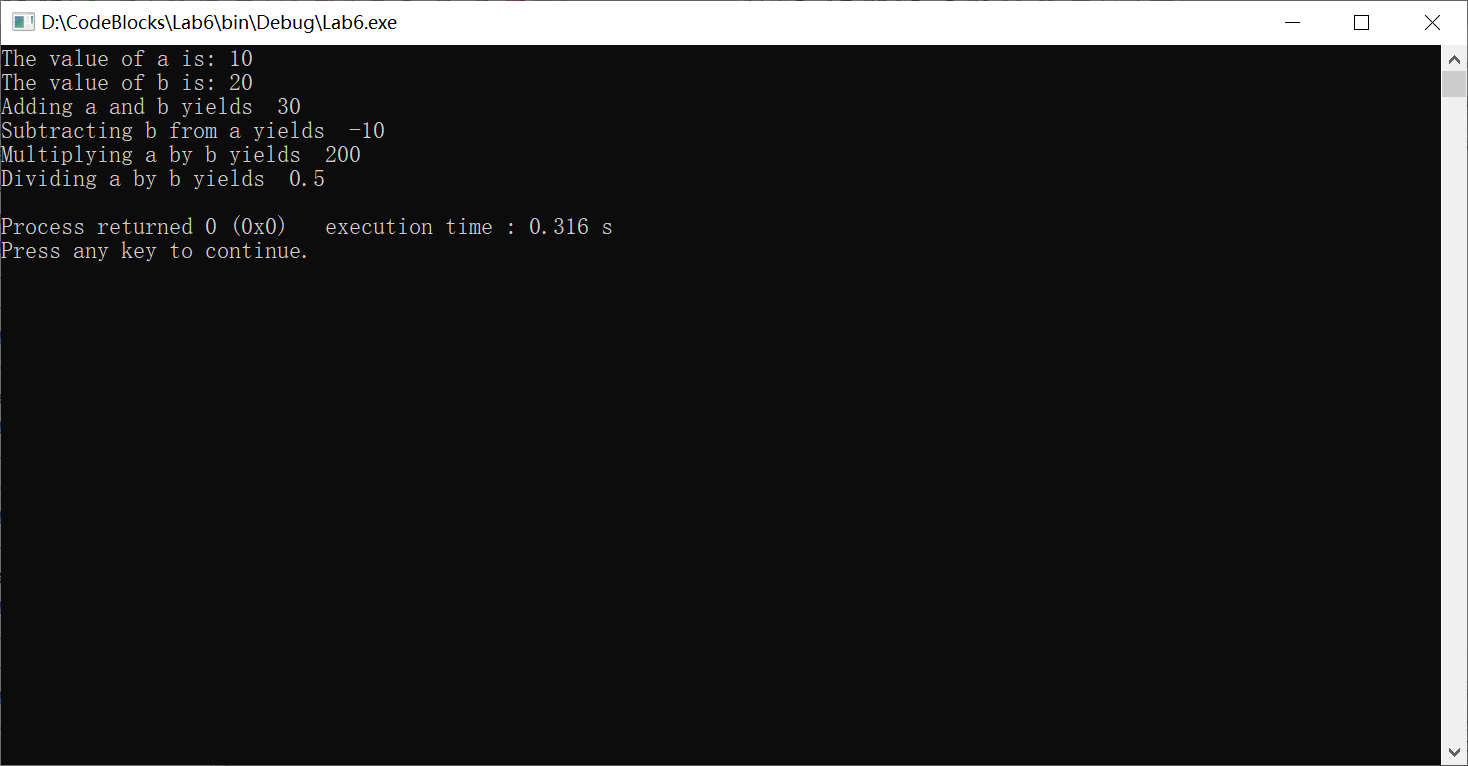
double multiplication=sc.multiply(a,b);

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

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

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

}



**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:



1):This class calls the system’s default constructor.

2):Member functions can get data directly.

3): //Lab 6:SimpleCalculator.h

class SimpleCalculator

{

public:

double add(double,double)const;//add function

double subtract(double,double)const;//subtraction function

double multiply(double,double)const;//multiply function

double divide(double,double)const;//divide function

double getAnswer(double)const;//get answer

//private://data member

//double number1;

// double number2;

private:

double answer;

};

//Lab6:SimpleCalculator.cpp

#include"SimpleCalculator.h"

explicit SimpleCalculator(double answer)

:Answer(answer)

{

}

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

{

return answer=a+b;

}

double SimpleCalculator::subtract(double a,double b)const//subtract function

{

return answer=a-b;

}

double SimpleCalculator::multiply(double a,double b)const//multiply function

{

return answer=a\*b;

}

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

{

return answer=a/b;

}

double SimpleCalculator::getAnswer(double a,double b )const//get answer function

{

return answer;

}

//Lab6:SimpleCalculatorTest.cpp

#include<iostream>

using namespace std;

#include"SimpleCalculator.h"

//define the class

int main()

{

SimpleCalculator sc;//define the object

double a=10.0;//data

double b=20.0;//data

cout<<"The value of a is: "<<a<<"\n"//show meaning

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

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

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

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

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

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

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

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

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

}

4): class SimpleCalculator//the SimpleCalculator class

{

public:

void setNumber(int a,int b);//set function

int addNumber();//add

int subNumber();//subtract

int mulNumber();//multiply

int divNumber();//divide function

private://data members

int number1;

int number2;

int sum1,sum2,sum3,sum4;

};

#include<iostream>

#include"SimpleCalculator.h"//definition

void SimpleCalculator::setNumber(int a,int b)//setNumber

{

number1=a;

number2=b;

}

int SimpleCalculator::addNumber()//add function

{

sum1=number1+number2;

return sum1;

}

int SimpleCalculator::subNumber()//sub function

{

sum2=number1-number2;

return sum2;

}

int SimpleCalculator::mulNumber()//multiply function

{

sum3=number1\*number2;

return sum3;

}

int SimpleCalculator::divNumber()//divide function

{

sum4=number1/number2;

return sum4;

}

#include<iostream>

using namespace std;

#include"SimpleCalculator.h"

int main()

{

SimpleCalculator sc;

int a;

int b;

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

cin<<a;

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

cin<<b;

cout<<"Adding a and b yields "<<sc.addNumber()<<"\n";//show addition

cout<<"Subtracting b from a yields "<<sc.subNumber()<<"\n";//show subtraction

cout<<"Multiplying a by b yields "<<sc.mulNumber()<<"\n";//show multiplication

cout<<"Dividing a by b yields "<<sc.divNumber()<<endl;//show division

}

**\*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?