1.I A phone number, such as (212) 767-8900, can be thought of as having three parts: the area code (212), the exchange (767), and the number (8900). Write a program that uses a structure to store these three parts of a phone number separately. Call the structure phone. Create two structure variables of type phone. Initialize one, and have the user input a number for the other one. Then display both numbers. The interchange might look like this:

**ANS**

**#include<iostream>**

**#include<stdlib.h>**

**using namespace std;**

**struct phone {**

**int area = 212;**

**int exchange = 767;**

**int numb = 8900;**

**};**

**int main()**

**{**

**long long int number;**

**struct phone b1,b2;**

**cout << "Enter your phone number:";**

**cin >> number;**

**b2.numb = number%10000;**

**number = number/10000;**

**b2.exchange = number%1000;**

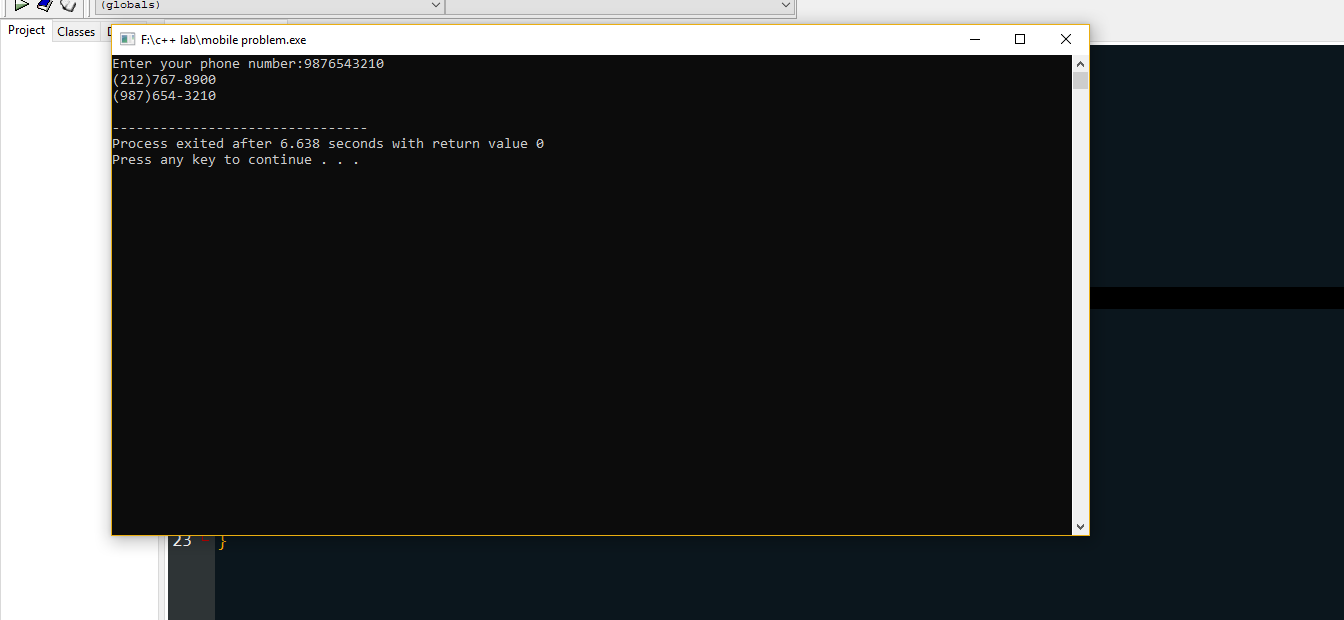
**number = number/1000;**

**b2.area = number%10000;**

**cout<<"(" << b1.area<< ")" << b1.exchange << "-" << b1.numb <<endl;**

**cout<<"(" << b2.area<< ")" << b2.exchange << "-" << b2.numb <<endl;**

**}**



1.II A point on the two-dimensional plane can be represented by two numbers: an X coordinate and Y coordinate. For example, (4,5) represents a point 4 units to the right of the origin along the X axis, and 5 units up the Y axis. The sum of two points can be defined as a new point whose X coordinate is the sum of the X coordinates of the two points, and whose Y coordinate is the sum of their Y coordinates. Write a program that uses a structure called point to model a point. Define three points, and have the user input values to two of them. Then set the third point equal to the sum of the other two, and display the value of the new point.

**ANS:**

**#include<iostream>**

**using namespace std;**

**class points**

**{**

**private:**

**float x, y;**

**public:**

**void toAdd(points a,points b);**

**void getInput();**

**void getOutput();**

**};**

**void points ::getOutput()**

**{**

**cout<<x<<y;**

**}**

**void points:: getInput()**

**{**

**cin>>x>>y;**

**}**

**void points::toAdd(points a, points b)**

**{**

**x = a.x +b.x;**

**y = a.y +b.y;**

**}**

**int main()**

**{**

**points a, b,c;**

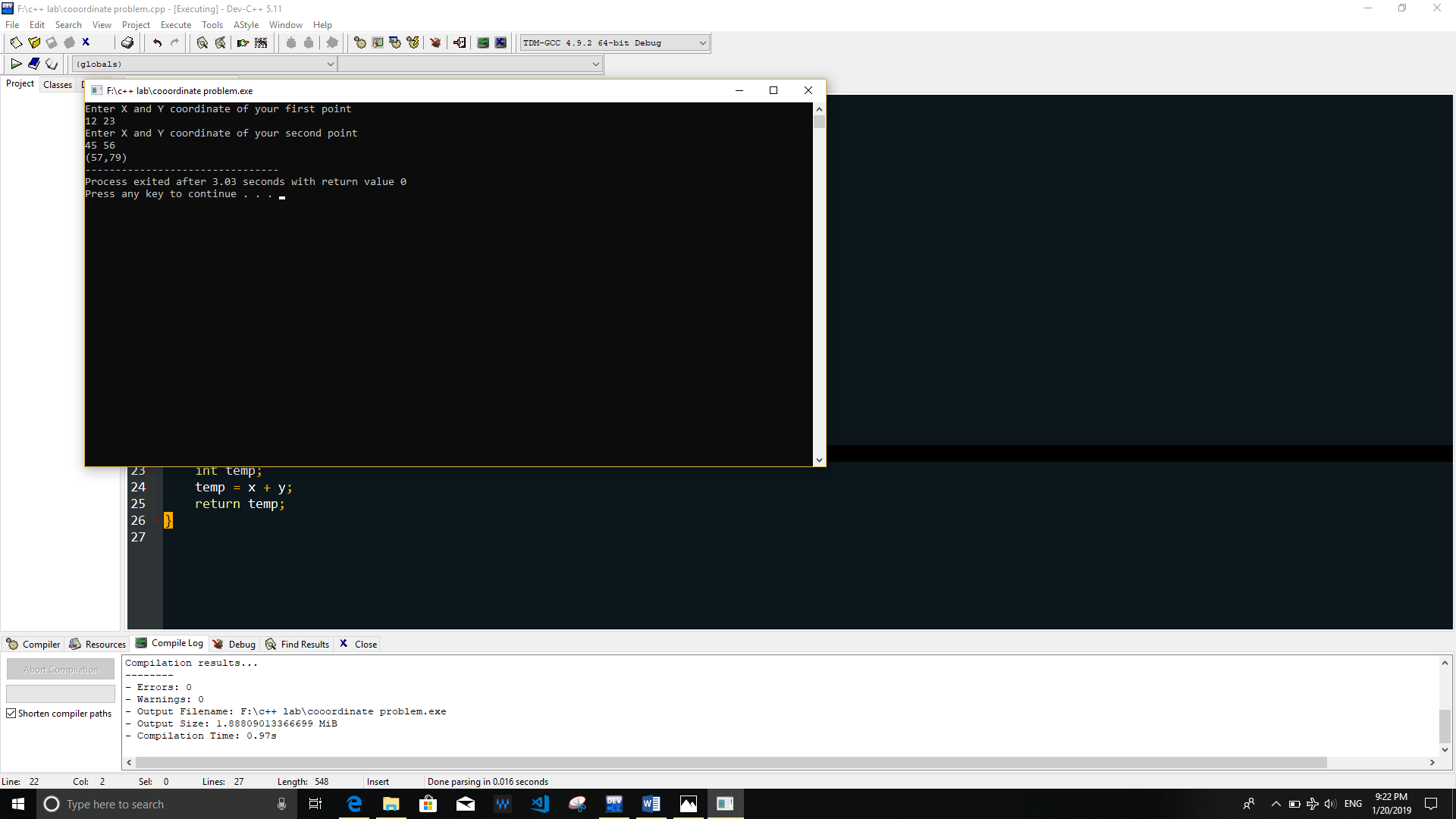
**a.getInput();**

**b.getInput();**

**c.toAdd(a,b);**

**c.getOutput();**

**}**



1.III Write a program to create a structure, Employee, having following attributesPrivate data: Employee\_name, Employee\_id, Employee\_salary, Employee\_address Write a main program to take employee’s information and print the same.

**ANS:**

**#include<iostream>**

**using namespace std;**

**class Employee**

**{**

**private:**

**string eName,eID,eNum;**

**public:**

**void getInput();**

**void getOutput();**

**};**

**void Employee::getInput()**

**{**

**getline(cin, eName);**

**getline(cin, eID);**

**getline(cin,eNum);**

**}**

**void Employee::getOutput()**

**{**

**cout<<eName<<endl<<eID<<endl<<eNum;**

**}**

**int main()**

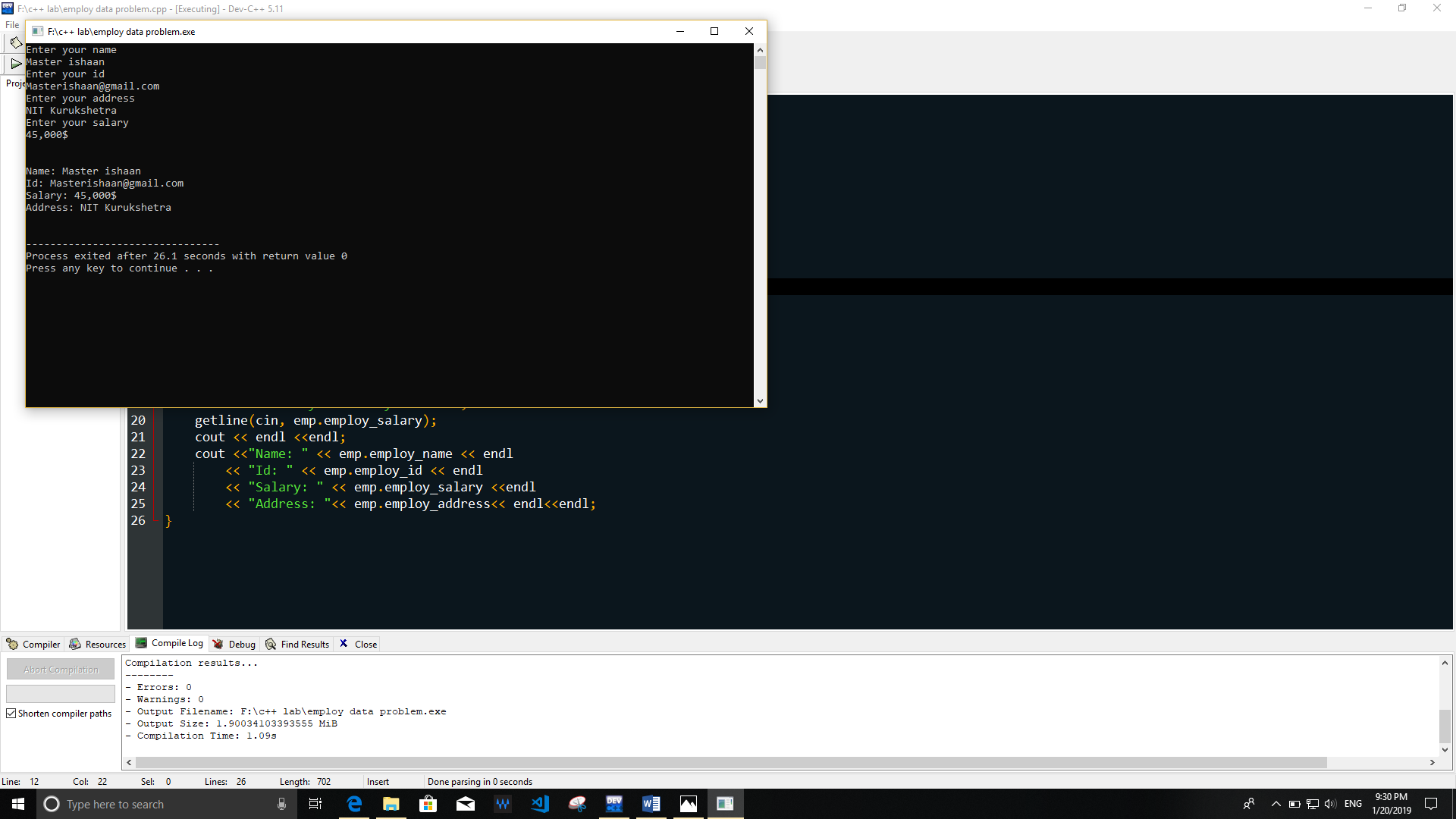
**{**

**Employee a,b;**

**a.getInput();**

**a.getOutput();b.getInput();b.getOutput();**

**}**

******

2.I Write a program to create a class, Employee having following attributesPrivate data: Employee\_name, Employee\_id, Employee\_salary, Employee\_address Public methods : getInput(); PrintOutput(); Write a main program to test it.

**ANS**

**#include<iostream>**

**#include<string>**

**using namespace std;**

**class employee**

**{**

**string name;**

**float salary;**

**int id;**

**string address;**

**public:**

**void get\_data()**

**{**

**cout << "Enter name of employee" << endl;**

**getline (cin, name);**

**cout << "Enter your id" << endl;**

**cin >> id;**

**cout << "Enter salary of employee" << endl;**

**fflush(stdin);**

**cin >> salary;**

**cout << "Enter your address" << endl;**

**fflush(stdin);**

**getline (cin, address);**

**}**

**void print\_data()**

**{**

**cout << "Name : " << name << endl;**

**cout << "ID : " << id << endl;**

**cout << "Salary : " << salary << endl;**

**cout << "Address : " << address << endl ;**

**}**

**};**

**int main()**

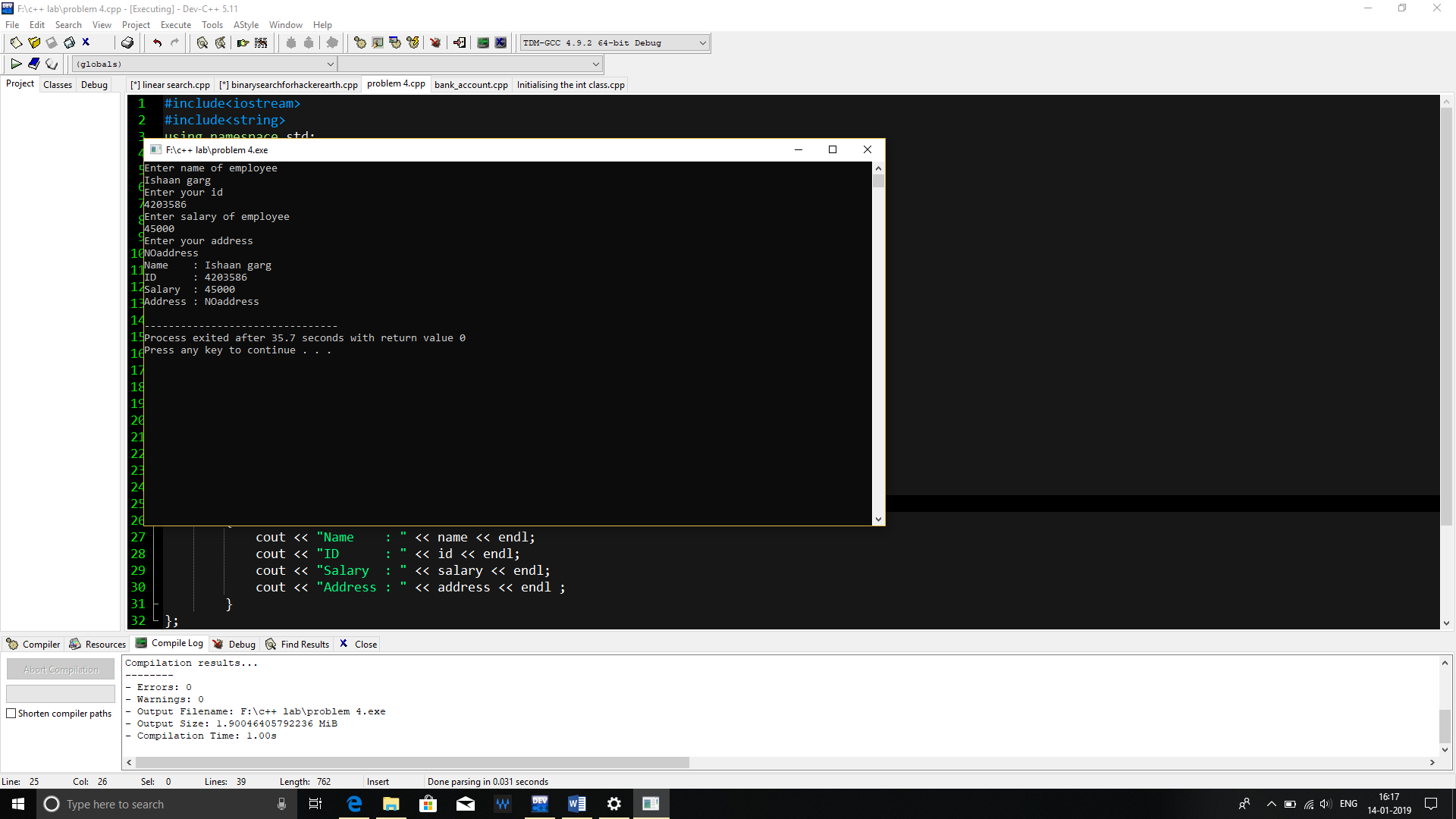
**{**

**employee emp;**

**emp.get\_data();**

**emp.print\_data();**

**}**

******

2.II Create a class that initiates part of functionality of the basic data type int. Call the class Int(note different spelling). The only data in this class is an int variable. Include member functions to initialize an Int to 0, to initialize it to an int value, to display it( looks just like an int), and to add two Int values. Write a program that exercises this class by creating two initialized and one uninitialized Int values, adding these two initialized values and placing the response in the uninitialized value, and then displaying this result.

**ANS:**

#include<iostream>

using namespace std;

class Int{

private:

int a;

public:

Int(void);//constructor

Int(int x);//constructor

void show();

};

Int:: Int(void)

{

a=0;

}

Int:: Int(int x)

{

a=x;

}

void Int::show()

{

cout<<a;

}

int main()

{

int x;

cin>>x;

Int obj1(x),obj2;

obj1.show();

obj2.show();

}

2.III Define a class to represent a bank account. Include the following members: Data Members: Name of the depositor, Account no. , Type of account, Balance amount. Member Functions: To assign initial values, To deposit an amount,To withdraw an amount after checking the balance, To display name and balance. Write a main program to test the program.

**ANS:**

**#include<iostream>**

**using namespace std;**

**class BankDetails**

**{**

**private:**

**int balance,wAmount,dAmout;**

**string bankAcNum;**

**public:**

**int getBalance();**

**int withdraw(int a);**

**int deposit (int b);**

**string acNum();**

**};**

**class HolderDetails**

**{**

**private:**

**string holderName,pNum,acNum;**

**public:**

**void inputHolderName();**

**void inputPNum();**

**string acNum();**

**};**

**int BankDetails::getBalance()**

**{**

**cout<<"Balance is: "<<balance<<endl;**

**}**

**int BankDetails::withdraw(int z)**

**{**

**if(balance>=z)**

**{**

**balance = balance-z;**

**cout<<"Updated balance is :"<<balance<<endl;**

**}**

**else**

**cout<<"Not Enough Money"<<endl;**

**}**

**int BankDetails::deposit(int z)**

**{**

**balance = balance+z;**

**cout<<"Updated balance is :"<<balance<<endl;**

**}**

**string BankDetails::acNum()**

**{**

**cin>>bankAcNum;**

**}**

2.IV Imagine a tollbooth at a bridge. Cars passing by the booth are expected to pay a fifty-cent toll. Mostly they do, but sometimes a car goes by without paying. The tollbooth keeps track of the number of cars that have gone by, and of the total amount of money collected. Model this tollbooth with a class called TollBooth. The two data items are a type unsigned int to hold the total number of cars, and a type double to hold the total number of cars, and a type double to hold the total amount of money collected. An initialization function sets both these values to 0. A member function called payingCar() increments the car total and adds 0.50 to the cash total. Another function, called nopayCar(), increments the car total but adds nothing to the cash total. Finally, a member function called display() desplays the two totals. Include a program to test this class. This program shold allow the user to push one key to count a paying car, and another to count a nonpaying car. Pusing the ESC key should cause the program to print out the total cars and total cash and then exit.

3.I Create a class called Employee that contains a name (an array of char) and an employee number (type long). Include a member function called getData() to get data from the user for insertion into the object, and another function called putData() to display the data. Assume the name has no embedded banks. Write a main() program to exercise this class. It should create an array of type employee, and then invite the user to input data for up to 100 employees. Finally, it should print out the data for all the employees.

3.II Create the database of students using Class, having the following attributes : roll\_no, student\_name, student\_address, student\_city, student\_pin, student\_sem, rank, and branch. Also write the program to enter the data for 500 students in any order and then display the list of students for a given branch and semester on display.

3.III Write a program that calculates the average of up to 100 English distances input by the user. Create an array of objects of the Distance class. To calculate the average, you can create dist\_add() method, that perform addition over 100 distance’s object. You will also need a member function that divides a Distance value by an integer. A brief description of class Distance is given below. class Distance { private: int feet; float inches; public: void getDist(); void showDist(); };

4.I Create a class called Time that has separate int member data for hours, minutes, and seconds. One constructor should initialize this data to 0, and another should initialize it to fixed values. A member function should add two objects of type time passed as arguments. A main() program should create two initialized time objects, and one that isn’t initialized. The it should add the two initialized values together, leaving the result in third Time variable. Finally it should display the value of this third value.

4.II Write a program to implement a binary member function to subtract one fraction from another. The function should simulate the subtract/assign operator (fr1 - =fr2) and should return void.

4.III Write a program to implement a binary member function to multiply two fractions. The function should simulate the multiply/assign operator (fr1 \* =fr2) and should return void.

4.VI Write a program to implement a binary friend function to divide two fractions. The function should simulate the divide/assign operator (fr1 / =fr2) and should return void.

**ANS:**

|  |
| --- |
| #include<iostream> |
|  |  |
|  | using namespace std; |
|  |  |
|  | class Fraction{ |
|  | float frac; |
|  | public: |
|  | Fraction(){ |
|  | ///normal constructor |
|  | frac = 0.0; |
|  | } |
|  | Fraction(int p, int q){ |
|  | ///fraction in p/q form |
|  | frac = float(p/q); |
|  | } |
|  | friend ostream &operator<<(ostream &output, const Fraction &D){ |
|  | output << D.frac; |
|  | return output; |
|  | } |
|  | void operator = (float f); |
|  | void operator \*=(int i); |
|  | void operator \*=(Fraction); |
|  | void operator \*=(float f); |
|  | void operator \*=(double db); |
|  |  |
|  | }; |
|  |  |
|  | void Fraction::operator =(float f){ |
|  | ///to assign value of fraction |
|  | frac = f; |
|  | } |
|  |  |
|  | void Fraction::operator \*= (int i){ |
|  | ///to compound multiply integer |
|  | frac = frac\*i; |
|  | } |
|  |  |
|  | void Fraction::operator \*=(Fraction target){ |
|  | ///to compound multiply Fraction object |
|  | frac = frac \* target.frac; |
|  | } |
|  |  |
|  | void Fraction::operator \*=(float f){ |
|  | ///to compound multiply float |
|  | frac = frac \* f; |
|  | } |
|  |  |
|  | void Fraction::operator \*=(double db){ |
|  | ///to compound multiply double |
|  | frac = frac\*db; |
|  | } |
|  |  |
|  | int main(){ |
|  | Fraction f1, f2, f3; |
|  | f1 = 2.2; |
|  | f2 = 3.9; |
|  | f3 = 4.2; |
|  | cout <<"fraction f1 : "<< f1 <<endl; |
|  | cout <<"fraction f2 : "<< f2 <<endl; |
|  | cout <<"fraction f3 : "<< f3 <<endl; |
|  | f1 \*= 3; |
|  | f2 \*= 6.4; |
|  | f3 \*= f1; |
|  | cout <<"fraction f1 : "<< f1 <<endl; |
|  | cout <<"fraction f2 : "<< f2 <<endl; |
|  | cout <<"fraction f3 : "<< f3 <<endl; |
|  | } |

4.IV Declare and define a class for complex number. A complex number is defined as x + y, where x defines the real part of the number and y is the imaginary part of the number. Write functions to implement the operations of Add, Subtract, Multiply and Divide. Use the following formulas: Add: x3 = x1 + x2 y3 = y1 + y2 Subtract: x3 = x1 - x2 y3 = y1 - y2 Multiply: x3 = x1 \* x2 – y1 \* y2 Y3 = x1 \*y2 + y1 \*x2 Divide: x3 = (x1 \* x2 + y1 +y2) / (x12 + y12 ) y3 = (y1 \* x2 - x1 \*y2) / (x22 y22 )

**ANS:**

|  |
| --- |
| #include<iostream> |
|  | using namespace std; |
|  |  |
|  | class Complex{ |
|  | float real; |
|  | float imag; |
|  | public: |
|  | Complex(float real, float imag); |
|  | Complex(); |
|  | //normal operator overloading |
|  | Complex operator+(Complex comp1); |
|  | Complex operator-(Complex comp1); |
|  | Complex operator\*(Complex comp1); |
|  | Complex operator/(Complex comp1); |
|  | //compound operator overloading |
|  | void operator+=(Complex comp1); |
|  | void operator-=(Complex comp1); |
|  | void operator\*=(Complex comp1); |
|  | void operator/=(Complex comp1); |
|  | friend ostream &operator<<( ostream &output, const Complex &comp); |
|  | void operator()(float real, float imag); |
|  | }; |
|  |  |
|  | Complex::Complex(){ |
|  | real = 0.0; |
|  | imag = 0.0; |
|  | } |
|  |  |
|  | Complex::Complex(float real, float imag){ |
|  | this->real = real; |
|  | this->imag = imag; |
|  | } |
|  |  |
|  | Complex Complex::operator+(Complex comp1){ |
|  | Complex ans; |
|  | ans.real = real+comp1.real; |
|  | ans.imag = imag+comp1.imag; |
|  | return ans; |
|  | } |
|  |  |
|  | void Complex::operator+=(Complex comp1){ |
|  | real = real+comp1.real; |
|  | imag = imag+comp1.imag; |
|  | } |
|  |  |
|  | Complex Complex::operator-(Complex comp1){ |
|  | Complex ans; |
|  | ans.real = real-comp1.real; |
|  | ans.imag = imag-comp1.imag; |
|  | return ans; |
|  | } |
|  |  |
|  | void Complex::operator-=(Complex comp1){ |
|  | real = real-comp1.real; |
|  | imag = imag-comp1.imag; |
|  | } |
|  |  |
|  | Complex Complex::operator\*(Complex comp1){ |
|  | Complex ans; |
|  | ans.real = (real\*comp1.real)-(imag\*comp1.imag); |
|  | ans.imag = (real\*comp1.imag)+(imag\*comp1.real); |
|  | return ans; |
|  | } |
|  |  |
|  | void Complex::operator\*=(Complex comp1){ |
|  | real = (real\*comp1.real)-(imag\*comp1.imag); |
|  | imag = (real\*comp1.imag)+(imag\*comp1.real); |
|  | } |
|  | Complex Complex::operator/(Complex comp1){ |
|  | Complex ans; |
|  | ans.real = ((real\*comp1.real)+imag+comp1.imag)/((real\*real) + (imag\*imag)); |
|  | ans.imag = ((imag\*comp1.real)-(real\*comp1.imag))/ |
|  | (comp1.real\*comp1.real\*comp1.imag\*comp1.imag); |
|  | return ans; |
|  | } |
|  | void Complex::operator/=(Complex comp1){ |
|  | real = ((real\*comp1.real)+imag+comp1.imag)/((real\*real) + (imag\*imag)); |
|  | imag = ((imag\*comp1.real)-(real\*comp1.imag))/ |
|  | (comp1.real\*comp1.real\*comp1.imag\*comp1.imag); |
|  | } |
|  |  |
|  | void Complex::operator()(float real, float imag){ |
|  | ///to update the value of complex number |
|  | this->real = real; |
|  | this->imag = imag; |
|  | } |
|  | ostream &operator<<( ostream &output, const Complex &comp){ |
|  | if(comp.imag >= 0){ |
|  | output << comp.real <<" +" <<comp.imag<<"i"; |
|  | } |
|  | else{ |
|  | output << comp.real <<" " <<comp.imag<<"i"; |
|  | } |
|  | return output; |
|  | } |
|  |  |
|  | int main(){ |
|  | Complex comp\_1(2.4, 4.1), comp\_2(2.2, -5.4), comp\_3; |
|  | cout << "comp 1 : "<<comp\_1<<endl; |
|  | cout << "comp 2 : "<<comp\_2<<endl; |
|  | cout << "comp 3 : "<<comp\_3<<endl; |
|  | comp\_3 = comp\_1+comp\_2; |
|  | cout << "addition comp 3 : "<<comp\_3<<endl; |
|  | comp\_3 = comp\_1-comp\_2; |
|  | cout << "subtraction comp 3 : "<<comp\_3<<endl; |
|  | comp\_3 = comp\_1\*comp\_2; |
|  | cout << "multiply comp 3 : "<<comp\_3<<endl; |
|  | comp\_3 = comp\_1/comp\_2; |
|  | cout << "divide comp 3 : "<<comp\_3<<endl; |
|  | comp\_3 += comp\_1; |
|  | cout << "compound addition comp 3 : "<<comp\_3<<endl; |
|  | comp\_3(4.5, 8.9); |
|  | cout << "parenthesis comp 3 : "<<comp\_3<<endl; |
|  | return 0; |
|  | } |

4.IV Write a program to find greatest of two numbers given in two different classes using a friend function.

4.V Write a program to find greatest of two numbers given in two different classes using a friend function.

4.VII Create a class called Employee that contains a name (an array of char) and an employee number (type long). Include a member function called getData() to get data from the user for insertion into the object, and another function called putData() to display the data. Assume the name has no embedded banks. Write a main() program to exercise this class. It should create an array of type employee, and then invite the user to input data for up to 100 employees. Finally, it should print out the data for the employee who has highest salary

5.I Create a class named Stack, take private data members according the requirements. Public methods are given below- (a) void push() (b) int pop() (c) stackfull() (d) stackempty() Write a main() program to exercise it.

**ANS**

|  |
| --- |
| #include<iostream> |
|  |
|  | #define MAX 50 |
|  | using namespace std; |
|  |  |
|  | class Stack{ |
|  | private: |
|  | int sizee; |
|  | int top; |
|  | int \*arr = NULL; |
|  | int elem; |
|  | public: |
|  | Stack(int len); |
|  | void push(int val); |
|  | int pop(); |
|  | bool isStackFull(); |
|  | bool isStackEmpty(); |
|  | }; |
|  | Stack::Stack(int len){ |
|  | sizee = len; |
|  | top = -1; |
|  | elem = 0; |
|  | arr = new int[len]; |
|  | if(arr == NULL){ |
|  | cout<<"ERROR: can't create stack"; |
|  | } |
|  | } |
|  |  |
|  | void Stack::push(int val){ |
|  | if(top < sizee-1 && !isStackFull()){ |
|  | arr[++top] = val; |
|  | } |
|  | else{ |
|  | cout << "stack is full"<<endl; |
|  | } |
|  | } |
|  |  |
|  | int Stack::pop(){ |
|  | if(top > -1 && !isStackEmpty()){ |
|  | return arr[top--]; |
|  | } |
|  | else{ |
|  | cout << "stack is empty"<<endl; |
|  | return 9999; |
|  | } |
|  | } |
|  |  |
|  | bool Stack::isStackEmpty(){ |
|  | if(top == -1){ |
|  | return true; |
|  | } |
|  | return false; |
|  | } |
|  |  |
|  | bool Stack::isStackFull(){ |
|  | if(top == sizee-1){ |
|  | return true; |
|  | } |
|  | return false; |
|  | } |
|  |  |
|  | int main(){ |
|  | int i, elem; |
|  | Stack mineStack(10); |
|  | for(i=0; i<13; i++){ |
|  | cout<<"enter value: "; |
|  | cin >> elem; |
|  | mineStack.push(elem); |
|  | } |
|  | for(i=0; i<13; i++){ |
|  | elem = mineStack.pop(); |
|  | cout <<"popped: "<<elem<<endl; |
|  | } |
|  | return 1; |
|  | } |