PCS – 307 OOP With C++ Lab Term work

Name : Bharat Upadhyay

Section : AI & DS

University Roll NO : 2017641

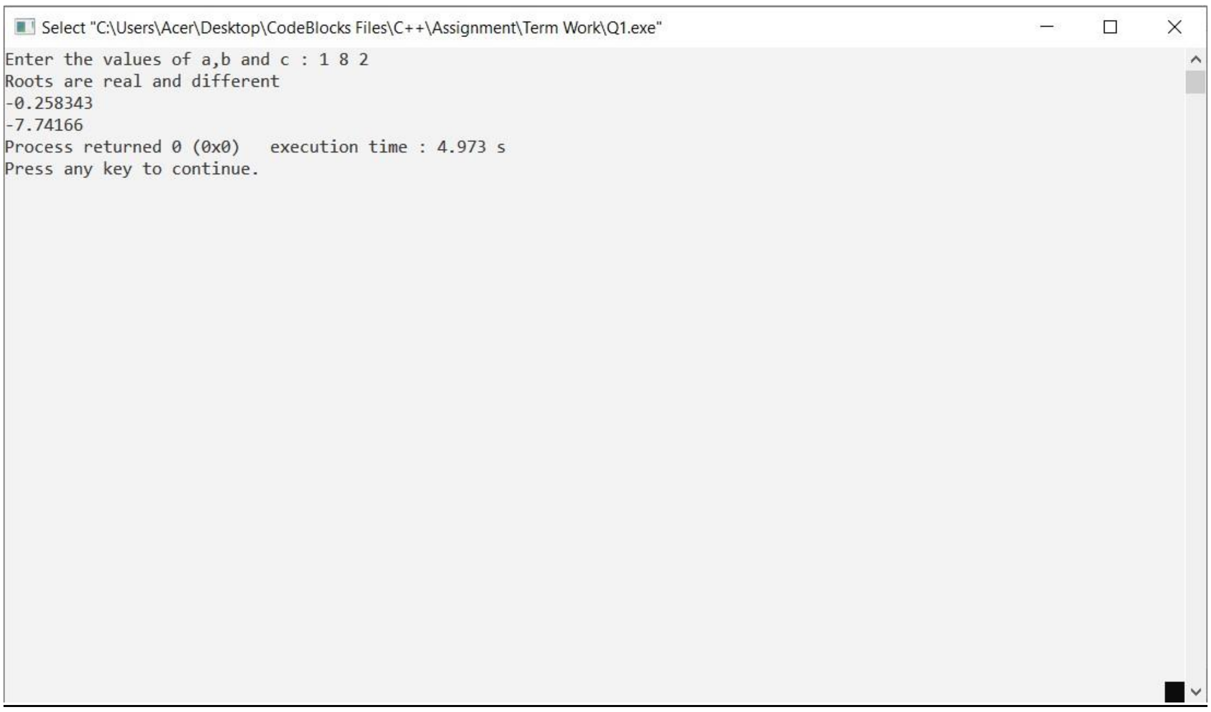
Q1. Given the coefficients of the quadratic polynomial (float variables), write a C++ program to determine whether the roots are real or complex (imaginary). If the roots are real, find them otherwise write the message "No real roots.

ALGORITHM:

```
1. Start
2. Input the coefficients of the equation, a, b and c.
3. Calculate d(discriminant) = (b * b) – (4 * a * c)
4. If d > 0:
        1: Calculate r1 = ( -b + sqrt(d)) / (2 * a)
        2: Calculate r2 = ( -b - sqrt(d)) / (2 * a)
        3: Display "Roots are real and different"
        4: Display r1 and r2
5: Else if d = 0:
        1: Calculate r1 = -b / (2 *a)
        2: r2 = r1
        3: Display "Root are real and equal"
        4: Display r1 and r2
6. Else:
        1: Calculate real = -b / (2 * a)
        2: Calculate imaginary = sqrt(-d) / (2 * a)
        3: Display "Roots are imaginary"
        4: Display real, "±" , imaginary, "i"
7. Stop
```

PROGRAM:

```
#include <bits/stdc++.h>
using namespace std;
void findRoots(int a, int b, int c)
{
 if (a == 0) {
 cout << "Invalid";
 return;
 }
 int d = b * b - 4 * a * c;
 double sqrt_val = sqrt(abs(d));
 if (d > 0) {
 cout << "Roots are real and different \n";
 cout << (double)(-b + sqrt_val) / (2 * a) << "\n"
 << (double)(-b - sqrt_val) / (2 * a);
 }
 else if (d == 0) {
 cout << "Roots are real and same \n";
 cout << -(double)b / (2 * a);
 }
 else // d < 0
 {
 cout << "Roots are complex \n";
 cout << -(double)b / (2 * a) << " + i" << sqrt_val
 << "\n"
 << -(double)b / (2 * a) << " - i" << sqrt_val;
 }
}
int main()
{
     int a,b,c;
    cout<<"Enter the values of a,b and c : ";
    cin>>a>>b>>c;
    findRoots(a, b, c);
    return 0;
}
```



Select "C:\Users\Acer\Desktop\CodeBlocks Files\C++\Assignment\Term Work\Q1.exe"

```
Enter the values of a,b and c : 1 8 2
Roots are real and different
-0.258343
-7.74166
Process returned 0 (0x0)   execution time : 4.973 s
Press any key to continue.
```

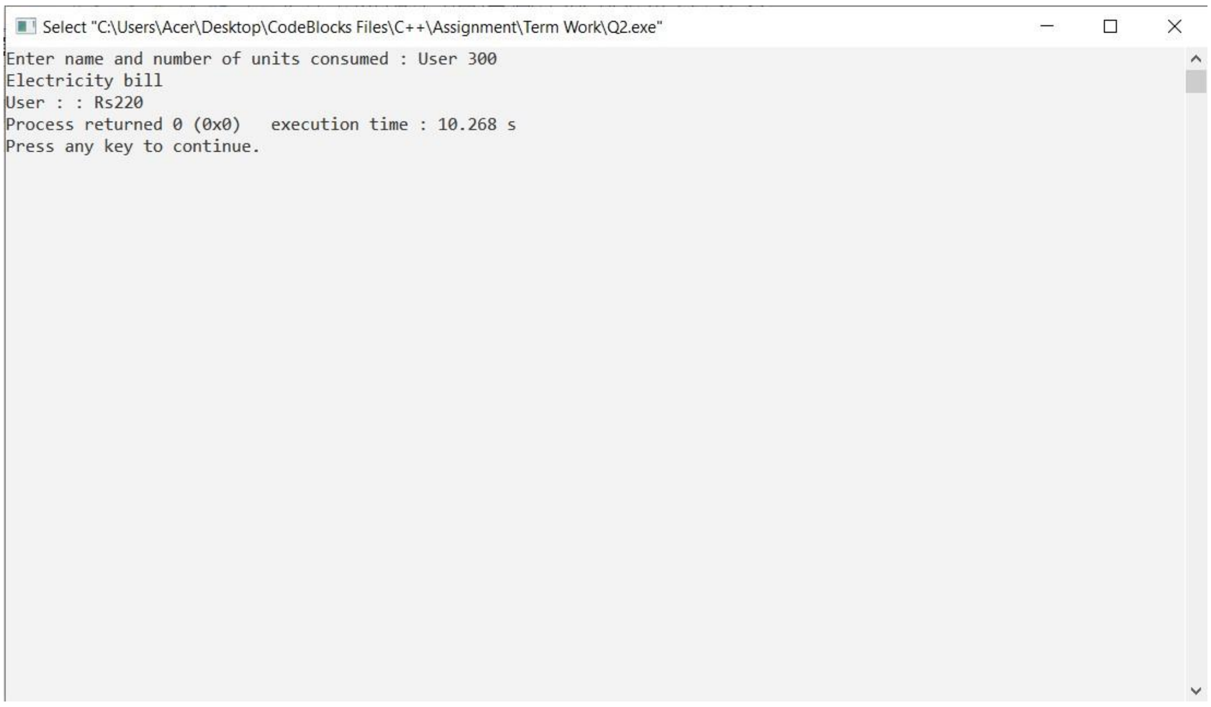
Q2. An electricity board charges the following rates to domestic users to discourage large consumption of energy For the first 100 units:- 60 P per unit For the next 200 units:-80 P per unit Beyond 300 units:-90 P per unit All users are charged a minimum of Rs 50 if the total amount is more than Rs 300 then an additional surcharge of 15% is added. WAP to read the names of users and number of units consumed and display the charges with names.

ALGORITHM:

```
1.  Start
2.  Input the name and number of units consumed
3.  If no_units <100:
        charge = (no_units*0.60)
4.  Else if no_units >100  and no_units <300:
        charge = (100*0.60)+((no_unit-100)*0.80)
5.  Else if no_units >300:
        charge = (100*0.60)+(200*0.80)((no_unit-300)*0.90)
6.  If charge <50
        charge = 50
7.  If charge >300
        charge=charge+(charge*0.15)
8.  Display charge
9.  Stop
```

PROGRAM:

```
#include<iostream>
using namespace std;
int main()
{
int no_unit;
float charge,scharge;
char name[20];
cout<< "Enter name and number of units consumed : ";
cin>>name;
cin>>no_unit;
if(no_unit<=100)
charge=(0.60*no_unit);
else if(no_unit>100&&no_unit<=300)
{
charge=(100*0.60);
charge+=((no_unit-100)*0.80);
}
else if(no_unit>=300)
{
charge=(100*0.60);
charge+=(200*0.80);
charge+=((no_unit-300)*0.90);
}
if(charge<50)
charge=50;
if(charge>300)
{
scharge=(0.15*charge);
charge+=scharge;
}
cout<< "Electricity bill\n";
cout<<name<<" : : Rs"<<charge;
return(0);
}
```



Select "C:\Users\Acer\Desktop\CodeBlocks Files\C++\Assignment\Term Work\Q2.exe"

```
Enter name and number of units consumed : User 300
Electricity bill
User : : Rs220
Process returned 0 (0x0)   execution time : 10.268 s
Press any key to continue.
```

Q3. W.A.P in C++ by defining a class to represent a bank account.Include the following -

Data Members

- **Name of the depositor**
- **Account number**
- **Type of account (Saving, Current etc.)**
- **Balance amount in the account**

Member Functions

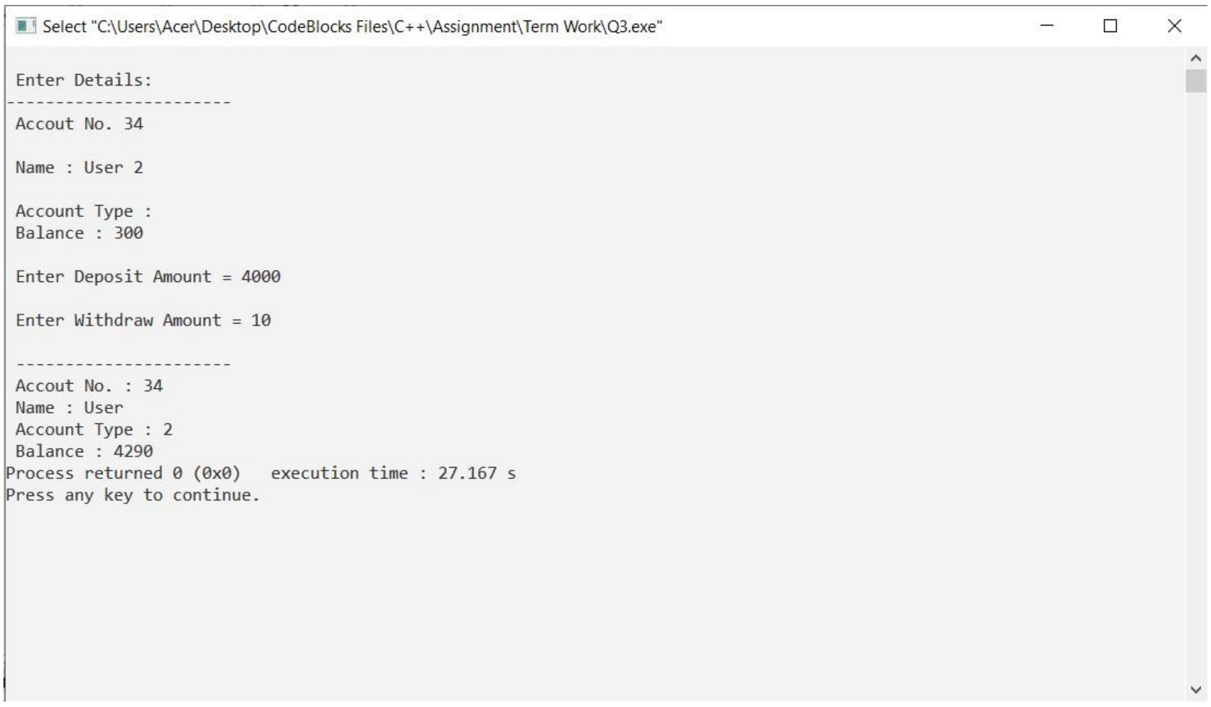
- **To assign initial values**
- **To deposit an amount**
- **To withdraw an amount after checking the balance**
- **To display name and balance**

ALGORITHM:

```
1.      Start
2.      Input name, account no., type of account, balance amount
3.      Deposit:
        Input amount to deposit
        Balance=balance+amount
4.      Withdraw:
        Input amount to withdraw
        If amount >balance
        print Insuficient balance
        Else
        Balance=balance-amount
        display “withdraw successful”
5.      display:
        display “name,account no,account type,balance”
6.      Stop
```

PROGRAM:

```
#include<iostream>
#include<stdio.h>
#include<string.h>
using namespace std;
class bank
{
 int acno;
 char nm[100], acctype[100];
 float bal;
 public:
 bank(int acc_no, char *name, char *acc_type, float balance)
 {
 acno=acc_no;
 strcpy(nm, name);
 strcpy(acctype, acc_type);
 bal=balance;
 }
 void deposit();
 void withdraw();
 void display();
};
void bank::deposit()
{
 int damt1;
 cout<<"\n Enter Deposit Amount = ";
 cin>>damt1;
 bal+=damt1;
}
void bank::withdraw()
{
 int wamt1;
 cout<<"\n Enter Withdraw Amount = ";
 cin>>wamt1;
 if(wamt1>bal)
 cout<<"\n Cannot Withdraw Amount";
 bal-=wamt1;
}
void bank::display()
{
 cout<<"\n ----------------------";
 cout<<"\n Accout No. : "<<acno;
 cout<<"\n Name : "<<nm;
 cout<<"\n Account Type : "<<acctype;
 cout<<"\n Balance : "<<bal;
}
int main()
{
 int acc_no;
 char name[100], acc_type[100];
 float balance;
 cout<<"\n Enter Details: \n";
 cout<<"-----------------------";
 cout<<"\n Accout No. ";
 cin>>acc_no;
 cout<<"\n Name : ";
 cin>>name;
 cout<<"\n Account Type : ";
 cin>>acc_type;
 cout<<"\n Balance : ";
 cin>>balance;

 bank b1(acc_no, name, acc_type, balance);
 b1.deposit();
 b1.withdraw();
 b1.display();
 return 0;
}
```



Select "C:\Users\Acer\Desktop\CodeBlocks Files\C++\Assignment\Term Work\Q3.exe"

```

 Enter Details:
-----------------------
 Accout No. 34

 Name : User 2

 Account Type :
 Balance : 300

 Enter Deposit Amount = 4000

 Enter Withdraw Amount = 10

 ----------------------
 Accout No. : 34
 Name : User
 Account Type : 2
 Balance : 4290
Process returned 0 (0x0)   execution time : 27.167 s
Press any key to continue.
```

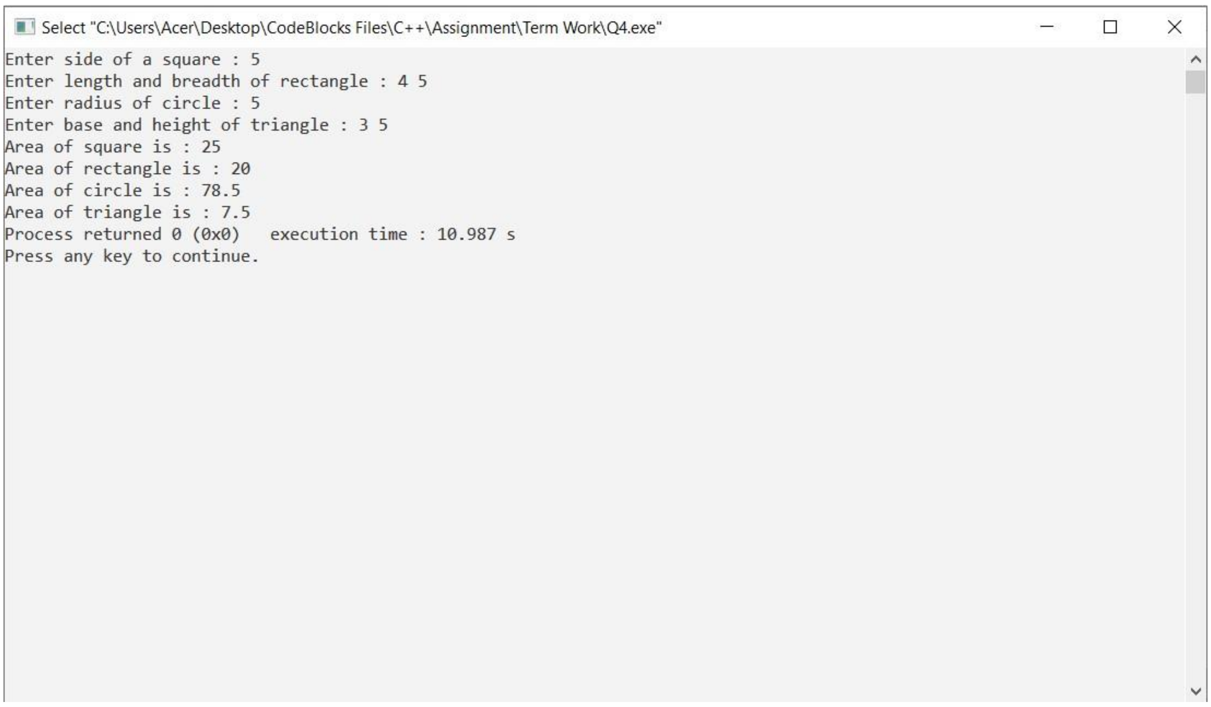
Q4. W.A.P in C++ to show the working of function overloading by using a function named calculateArea() to calculate area of square, rectangle and triangle using different signatures as required.

ALGORITHM:

- START
- Declare variables (s,l,b) of type int and (r,bs,ht) of float type.
- Initialize values of variables.
- Declare a function area() of type int which return(s*s).
- Declare a function area() of type int which return (l*b).
- Declare a function area() of type float which return (3.14*r*r).
- Declare a function area() of type float which return ((bs*ht)/2).
- Display the values calculated by area().
- STOP.

PROGRAM:

```
#include<iostream>
using namespace std;
int area(int);
int area(int,int);
float area(float);
float area(float,float);
int main()
{
 int s,l,b;
 float r,bs,ht;
 cout<<"Enter side of a square : ";
 cin>>s;
 cout<<"Enter length and breadth of rectangle : ";
 cin>>l>>b;
 cout<<"Enter radius of circle : ";
 cin>>r;
 cout<<"Enter base and height of triangle : ";
 cin>>bs>>ht;
 cout<<"Area of square is : "<<area(s);
 cout<<"\nArea of rectangle is : "<<area(l,b);
 cout<<"\nArea of circle is : "<<area(r);
 cout<<"\nArea of triangle is : "<<area(bs,ht);
}
int area(int s)
{
 return(s*s);
}
int area(int l,int b)
{
 return(l*b);
}
float area(float r)
{
 return(3.14*r*r);
}
float area(float bs,float ht)
{
 return((bs*ht)/2);
}
```



Select "C:\Users\Acer\Desktop\CodeBlocks Files\C++\Assignment\Term Work\Q4.exe"

```
Enter side of a square : 5
Enter length and breadth of rectangle : 4 5
Enter radius of circle : 5
Enter base and height of triangle : 3 5
Area of square is : 25
Area of rectangle is : 20
Area of circle is : 78.5
Area of triangle is : 7.5
Process returned 0 (0x0)   execution time : 10.987 s
Press any key to continue.
```

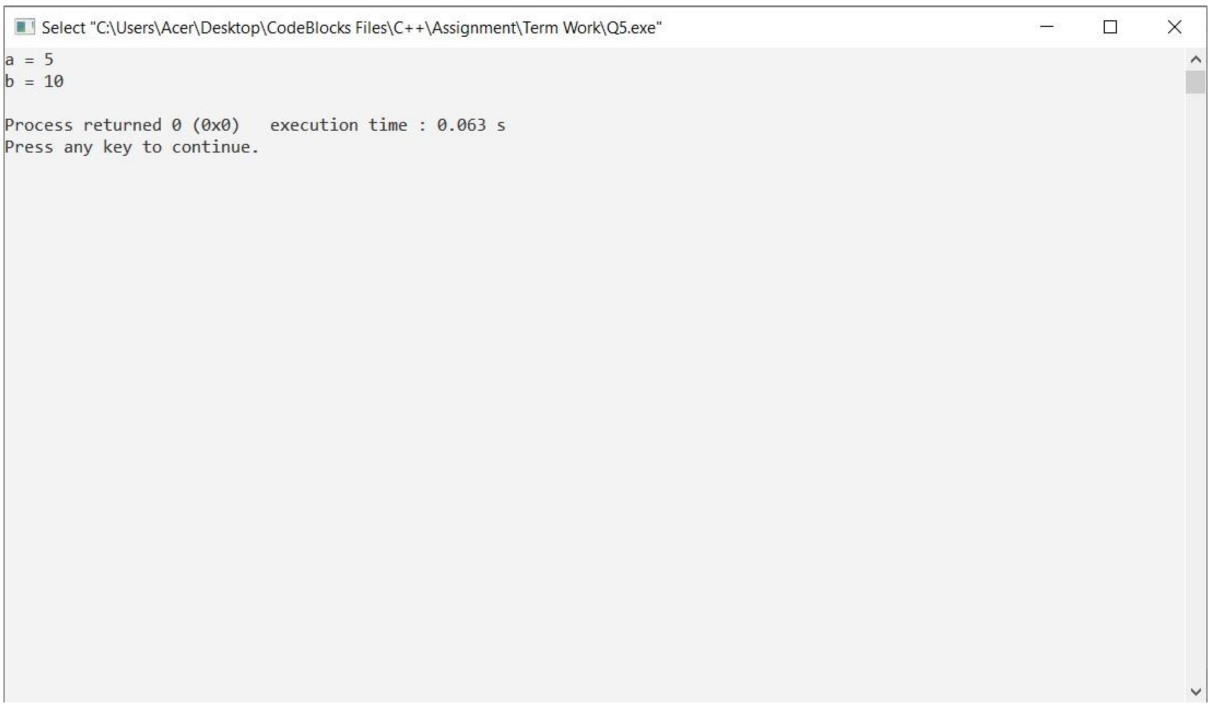
Q5. Write a Program in C++ to demosntrate the concept of data abstraction using the concept of Class and Objects.

ALGORITHM:

- STOP
- Create a class called implementAbstraction.
- Initialize private data members of class implementAbstraction named a and b.
- Create a data method called setvalue() to assign values to a and b, and display() to display the values.
- Close the class.
- In main(),create an object of class implementAbstraction and call the data methods.
- STOP

PROGRAM:

```
#include <iostream>
using namespace std;
class implementAbstraction
{
private:
int a,b;
public:
void set(int x, int y)
{
a = x;
b = y;
}
void display()
{
cout<<"a = " <<a << endl;
cout<<"b = " << b << endl;
}
};
int main()
{
implementAbstraction obj;
obj.set(5,10);
obj.display();
return 0;
}
```


Select "C:\Users\Acer\Desktop\CodeBlocks Files\C++\Assignment\Term Work\Q5.exe"

```
a = 5
b = 10

Process returned 0 (0x0)   execution time : 0.063 s
Press any key to continue.
```

Q6. Create a class called Invoice that a hardware store might use to represent an invoice for an item sold at the store. An Invoice should include four pieces of information as instance Data Members -

• partNumber (type String)

• partDescription (type String)

• quantity of the item being purchased (type int) price_per_item (type double)

Your class should have a constructor that initializes the four instance variables. Provide a set and a get method for each instance variable. In addition, provide a method named getInvoiceAmount() that calculates the invoice amount (i.e., multiplies the quantity by the price per item), then returns the amount as a double value. If the quantity is not positive, it should be set to 0. If the price per item is not positive, it should be set to 0.0. Write a test application named invoiceTest that demonstrates class Invoice's capabilities.

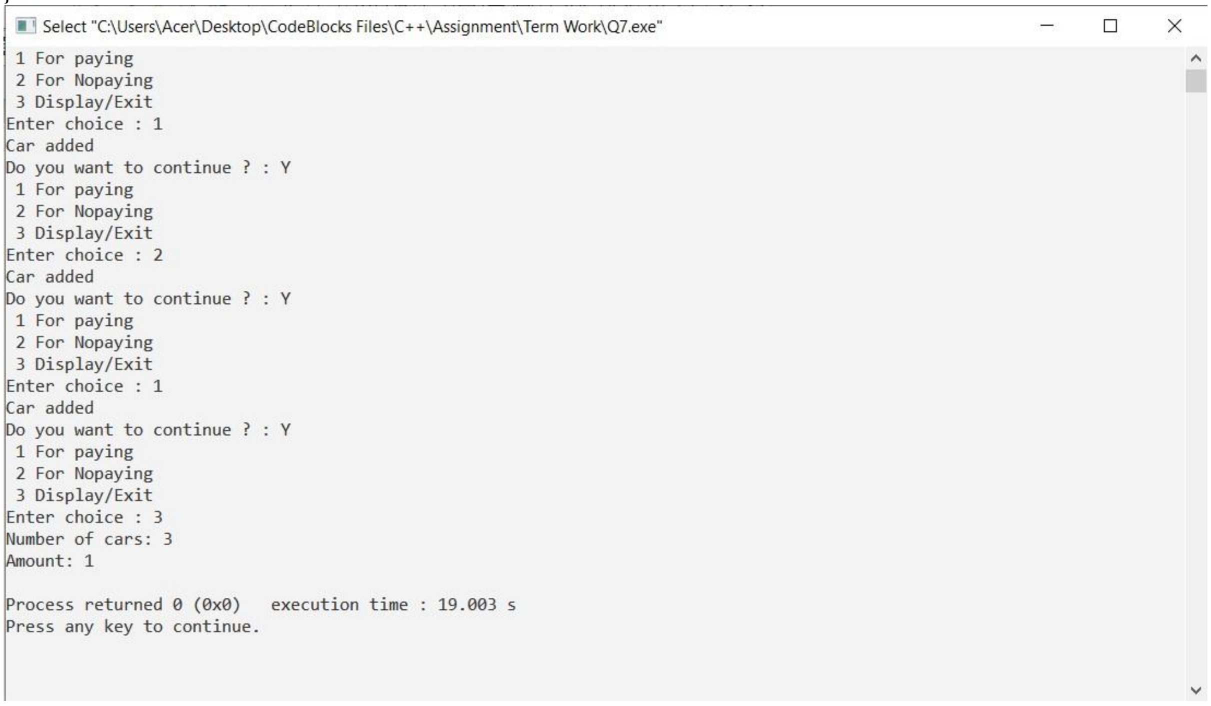
ALGORITHM:

- START
- declare a class invoice
- declare the variables and the member functions of the class
- create a parameterized construct to initialize variables
- create a member function PartNumber(),to input and return part number
- create a member function PartDescription(), to store and return the description of the part.
- create a member functions PartQuantity(), to store and return the quantity of the parts.
- create a member function ItemPrice(), to store and return the price.
- to create a member function InvoiceAmount(), to perform quantity *price and return amount
- create a main to print PartNumber , PartDescription , PartQuantity , PartPrice , Amount.
- STOP

PROGRAM:

```
#include<iostream>
#include <string>
using namespace std;
class Invoice
{
public:
 Invoice( string, string, int, int );
 void setPartNumber( string );
 string getPartNumber();
 void setPartDescription(string);
 string getPartDescription();
 void setItemQuantity(int);
 int getItemQuantity();
 void setItemPrice(int);
 int getItemPrice();
 int getInvoiceAmount();
private:
 string partNumber;
 string partDescription;
 int itemQuantity;
 int itemPrice;
};
Invoice::Invoice( string number, string description, int quantity, int price )
{
partNumber=number;
partDescription=description;
if(quantity>0)
 itemQuantity=quantity;
else
{
 itemQuantity=0;
 cout<<"Initial quantity was invalid."<<endl;
}
if(price>0)
 itemPrice=price;
else
{
 itemPrice=0;
 cout<<"Initial price was invalid."<<endl;
}
}
void Invoice::setPartNumber( string number)
{
if ( number.length() <= 25 )
 partNumber = number;
if ( number.length() > 25 )
{
 partNumber = number.substr( 0, 25 );
 cout << "Name \"" << number <<"\" exceeds maximum length (25).\n"<< "Limiting partNumber to first 25
characters.\n" << endl;
}
}
void Invoice::setPartDescription(string description )
{
if ( description.length() <= 25 )
 partDescription = description;
if ( description.length() > 25 )
{
 partDescription = description.substr( 0, 25 );
 cout << "Name \"" << description <<"\" exceeds maximum length (25).\n"<< "Limiting partDescription to first
25 characters.\n" << endl;
}
}
void Invoice::setItemQuantity(int quantity )
```

```
{
if(quantity>0)
 itemQuantity=quantity;
else
{
 itemQuantity=0;
 cout<<"Initial quantity was invalid."<<endl;
}
}
void Invoice::setItemPrice(int price )
{
if(price>0)
 itemPrice=price;
else
{
 itemPrice=0;
 cout<<"Initial price was invalid."<<endl;
}
}
string Invoice::getPartNumber()
{
return partNumber;
}
string Invoice::getPartDescription()
{
return partDescription;
}
int Invoice::getItemQuantity()
{
return itemQuantity;
}
int Invoice::getItemPrice()
{
return itemPrice;
}
int Invoice::getInvoiceAmount()
{
return itemQuantity*itemPrice;
}
int main()
{
Invoice Invoice1("User1","Screw Guage",5,40);
Invoice Invoice2("User2","Screws",5,10);
cout << "Invoice1's initial part number is: "<< Invoice1.getPartNumber()<< "\nand part description is: "<<
Invoice1.getPartDescription()<<endl;
cout<< "quantity per item is: "<< Invoice1.getItemQuantity()<< "\nprice per item is: "<<Invoice1.getItemPrice()<< endl;
cout<<"Invoice1's total amount is: "<<Invoice1.getInvoiceAmount()<<endl<<endl;
cout << "Invoice2's initial part number is: "<< Invoice2.getPartNumber()<< "\nand part description is: "<< Invoice2.getPartDescription()<<endl;
cout<< "quantity per item is: "<< Invoice2.getItemQuantity()<< "\nprice per item is: "<<Invoice2.getItemPrice()<< endl;
cout<<"Invoice2's total amount is: "<<Invoice2.getInvoiceAmount()<<endl;
}
```



Select "C:\Users\Acer\Desktop\CodeBlocks Files\C++\Assignment\Term Work\Q7.exe"

```
 1 For paying
 2 For Nopaying
 3 Display/Exit
Enter choice : 1
Car added
Do you want to continue ? : Y
 1 For paying
 2 For Nopaying
 3 Display/Exit
Enter choice : 2
Car added
Do you want to continue ? : Y
 1 For paying
 2 For Nopaying
 3 Display/Exit
Enter choice : 1
Car added
Do you want to continue ? : Y
 1 For paying
 2 For Nopaying
 3 Display/Exit
Enter choice : 3
Number of cars: 3
Amount: 1

Process returned 0 (0x0)   execution time : 19.003 s
Press any key to continue.
```

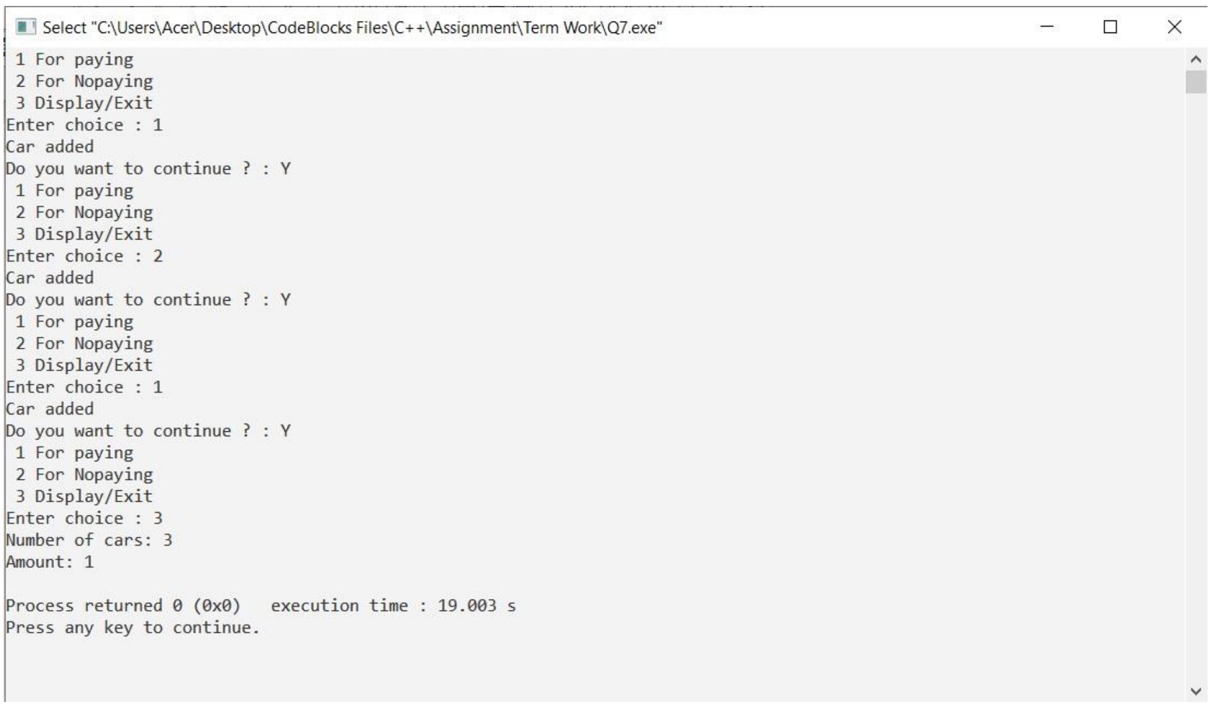
Q7. Imagine a tollbooth with a class called TollBooth. The two data items are of type unsigned int and double to hold the total number of cars and total amount of money collected. A constructor initializes both of these data members to 0. A member function called payingCar( ) increments the car total and adds 0.5 to the cash total. Another function called nonPayCar( ) increments the car total but adds nothing to the cash total. Finally a member function called display( ) shows the two totals. Include a program to test this class. This program should allow the user to push one key to count a paying car , and another to count a non paying car. Pushing the ESC key should cause the program to print out the total number of cars and total cash and then exit.

ALGORITHM:

- START
- create a class called tollbooth
- Initialize private data memebers of class tollbooth named car and amt
- create a default constructor where car and amt are initialized to 0
- create three memeber functions display (), payingcar() and nonpayingcar(). In payingcar() increment car by 1 and amt by 0.5. In nonpayingcar() increment car by 1. In display () print the car and amt.
- Close the tollbooth
- In main(), create an object t of class tollbooth.
- Using switch case call the functions payingcar(), nonpayingcar () for the paying cars and non paying cars. Assign the default value of switch case with display() to display the total
- STOP

PROGRAM:

```
#include<iostream>
using namespace std;
class tollbooth
{
 unsigned int car;
 double amt;
 public:
 tollbooth()
 {
 this->car = 0;
 this->amt = 0;
 }
 void payingcar()
 {
 this->car++;
 this->amt+=0.50;
 }
 void nonpayingcar()
 {
 this->car++;
 }
 void display()
 {
 cout<<"Number of cars: "<<car<<endl;
 cout<<"Amount: "<<amt<<endl;
 }
};
int main()
{
 char c='y';
 int ch;
 tollbooth t;
 do{
 cout<<" 1 For paying \n 2 For Nopaying \n 3 Display/Exit \n";
 cout<<"Enter choice : ";
 cin>>ch;
 switch(ch)
 {
 case 1:
 t.payingcar();
 cout<<"Car added";
 break;
 case 2:
 t.nonpayingcar();
 cout<<"Car added";
 break;
 case 3:
 t.display();
 c='n';
 break;
 }
 if(c=='y'||c=='Y'){
 cout<<"\nDo you want to continue ? : ";
 cin>>c;
 }
 }
 while(c=='y'||c=='Y');
 return 0;
}
```


Select "C:\Users\Acer\Desktop\CodeBlocks Files\C++\Assignment\Term Work\Q7.exe"

```
 1 For paying
 2 For Nopaying
 3 Display/Exit
Enter choice : 1
Car added
Do you want to continue ? : Y
 1 For paying
 2 For Nopaying
 3 Display/Exit
Enter choice : 2
Car added
Do you want to continue ? : Y
 1 For paying
 2 For Nopaying
 3 Display/Exit
Enter choice : 1
Car added
Do you want to continue ? : Y
 1 For paying
 2 For Nopaying
 3 Display/Exit
Enter choice : 3
Number of cars: 3
Amount: 1

Process returned 0 (0x0)   execution time : 19.003 s
Press any key to continue.
```

Q8. 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 display it in 11:59:59 format. A member function named add() should add two objects of type time passed as arguments. A main ( ) program should create two initialized values together, leaving the result in the third time variable. Finally it should display the value of this third variable.

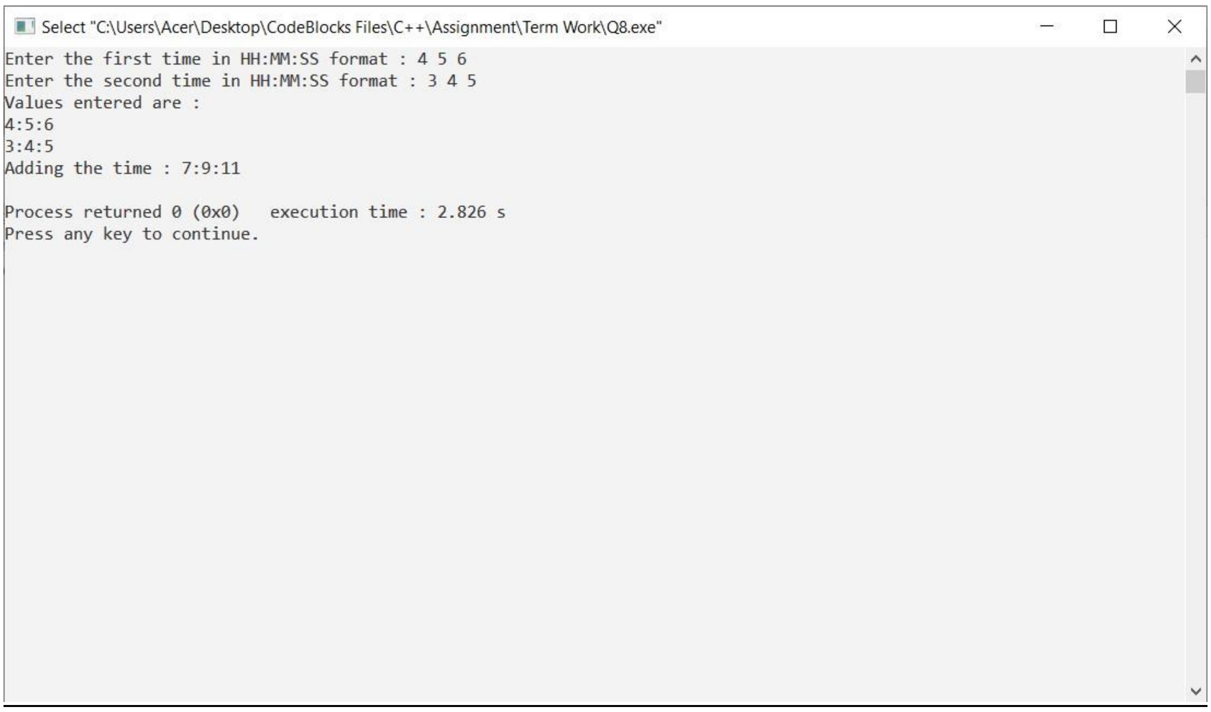
ALGORITHM:

```
1. START.
2. Declare the class time.
3. Declare the variables hour, min, sec and member function dispdata.
4. Define the default constructor to initialise the value of variables to 0 and parametrised constructor to initialise the variables to fixed values.
5. Define the operator +() to add two objects of class time and return to third object of class time.
6. In main, declare two objects of class time using parameters.
7. Now, declare a new object of class time and initialise it to sum of previous objects using + operator.
8. Call dispdata function for all three objects to display the data.
9. STOP.
```

PROGRAM:

```
#include <iostream>
using namespace std;
class user_time
{
  int h,m,s,sum;
public:
  user_time()
  {
    h=0;
    m=0;
    s=0;
  }
  void getdata()
  {
    cin>>h>>m>>s;
  }
  void disp()
  {
    cout<<h<<":"<<m<<":"<<s<<endl;
  }
  void add(user_time c1,user_time c2)
  {
    h=c1.h+c2.h;
    m=c1.m+c2.m;
    s=c1.s+c2.s;
    if(m>60)
    {
      h=h+1;
      m=m-60;
    }
```

```
        if(s>60)
        {
            m=m+1;
            s=s-60;
        }
        cout<<"Adding the time : "<<h<<":"<<m<<":"<<s<<endl;
    }
};
main()
{
    user_time c1,c2,c3;
    cout<<"Enter the first time in HH:MM:SS format : ";
    c1.getdata();
    cout<<"Enter the second time in HH:MM:SS format : ";
    c2.getdata();
    cout<<"Values entered are : \n" ;
    c1.disp();
    c2.disp();
    c3.add(c1,c2);
}
```



Select "C:\Users\Acer\Desktop\CodeBlocks Files\C++\Assignment\Term Work\Q8.exe"

```
Enter the first time in HH:MM:SS format : 4 5 6
Enter the second time in HH:MM:SS format : 3 4 5
Values entered are :
4:5:6
3:4:5
Adding the time : 7:9:11

Process returned 0 (0x0)   execution time : 2.826 s
Press any key to continue.
```

Q9. *Create class SavingsAccount. Use a static variable annualInterestRate to store the annual interest rate for all account holders. Each object of the class contains a private instance variable savingsBalance indicating the amount the saver currently has on deposit. Provide method calculateMonthlyInterest to calculate the monthly interest by multiplying the savingsBalance by annualInterestRate divided by 12.This interest should be added to savingsBalance. Provide a static method modifyInterestRate that sets the annualInterestRate to a new value. Write a program to test class SavingsAccount. Instantiate two savingsAccount objects, saver1 and saver2, with balances of $2000.00 and $3000.00, respectively. Set annualInterestRate to 4%, then calculate the monthly interest and print the new balances for both savers. Then set the annualInterestRate to 5%, calculate the next month's interest and print the new balances for both savers.*

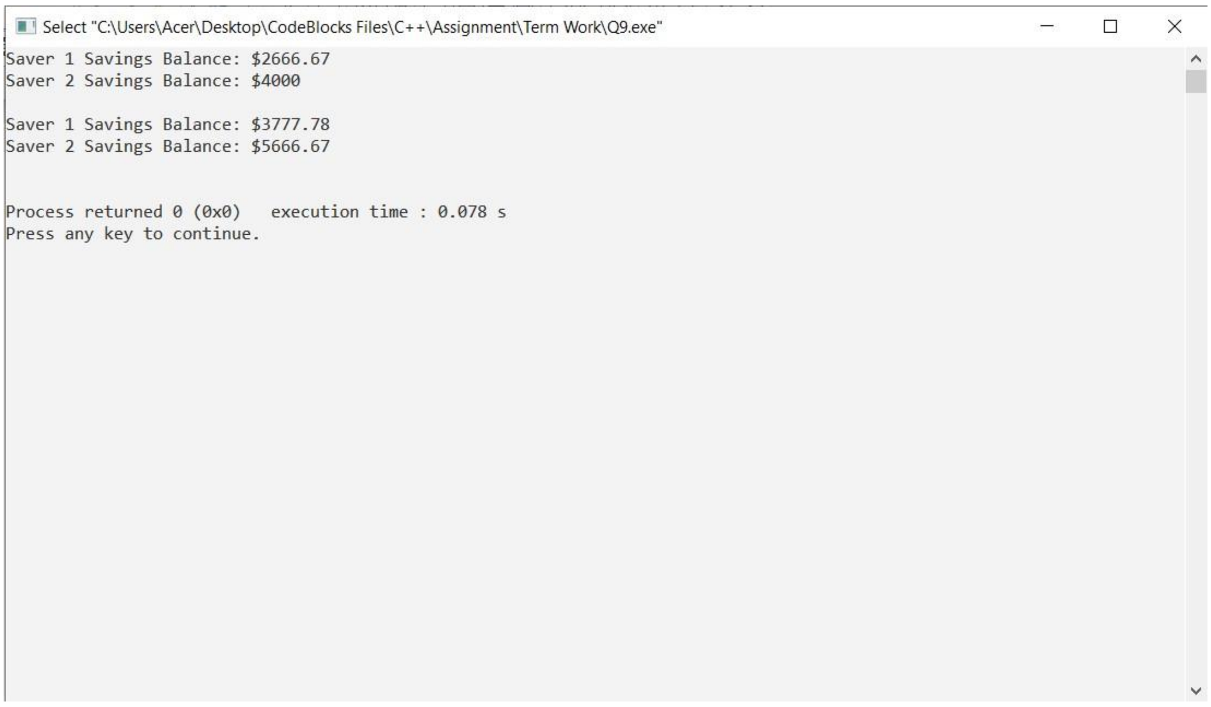
ALGORITHM:

```
1. START
2.Declare the class named savings account
3. declare the variable savings balance of float type under private
4. use  a parametrized constructor to get value of base salary .
5. declare a static variable named annual interest rate and initizlie it with 0.
6. declare a member function calculate monthly interest which will calculate the required interest rate by using the formulae savingsBalance = (savingsBalance+(savingsBalance * annualInterestRate) / 12);
7. declare the member function named modify interest rate that will take the new interest rate from the user.
8. declare class savings account and declare the objects named saver 1 and saver 2 with parameters as basic salary.
9. modify the interest rate by setting it to as desired by user.
10. calculate monthly interest by calling the function calculate monthly interest().
11. display the monthly interest calculated .
12.STOP
```

PROGRAM:

```cpp
#include <iostream>
using namespace std;
class SavingsAccount
{
public:
   SavingsAccount(){}
   SavingsAccount(int value);
   static float annualInterestRate;
   void calculateMonthlyInterest();
   static void modifyIntererestRate(float value);
   float GetBalance() const { return savingsBalance; }
   private:
   float savingsBalance;
};
SavingsAccount::SavingsAccount(int value)
{
   savingsBalance = value;
}
float SavingsAccount::annualInterestRate = 0;
void SavingsAccount::calculateMonthlyInterest()
{
   savingsBalance = (savingsBalance+(savingsBalance * annualInterestRate) / 12);
}
void SavingsAccount::modifyIntererestRate(float value)
{
   annualInterestRate = value;
}
int main()
{
   SavingsAccount saver1(2000.00);
```

```
    SavingsAccount saver2(3000.00);
    SavingsAccount::modifyIntererestRate(4);
    saver1.calculateMonthlyInterest();
    cout << "Saver 1 Savings Balance: $" << saver1.GetBalance() << endl;
    saver2.calculateMonthlyInterest();
    cout << "Saver 2 Savings Balance: $" << saver2.GetBalance() << endl;
    cout << endl;
    SavingsAccount::modifyIntererestRate(5);
    saver1.calculateMonthlyInterest();
    cout << "Saver 1 Savings Balance: $" << saver1.GetBalance() << endl;
    saver2.calculateMonthlyInterest();
    cout << "Saver 2 Savings Balance: $" << saver2.GetBalance() << endl;
    cout << endl;
    return 0;
}
```



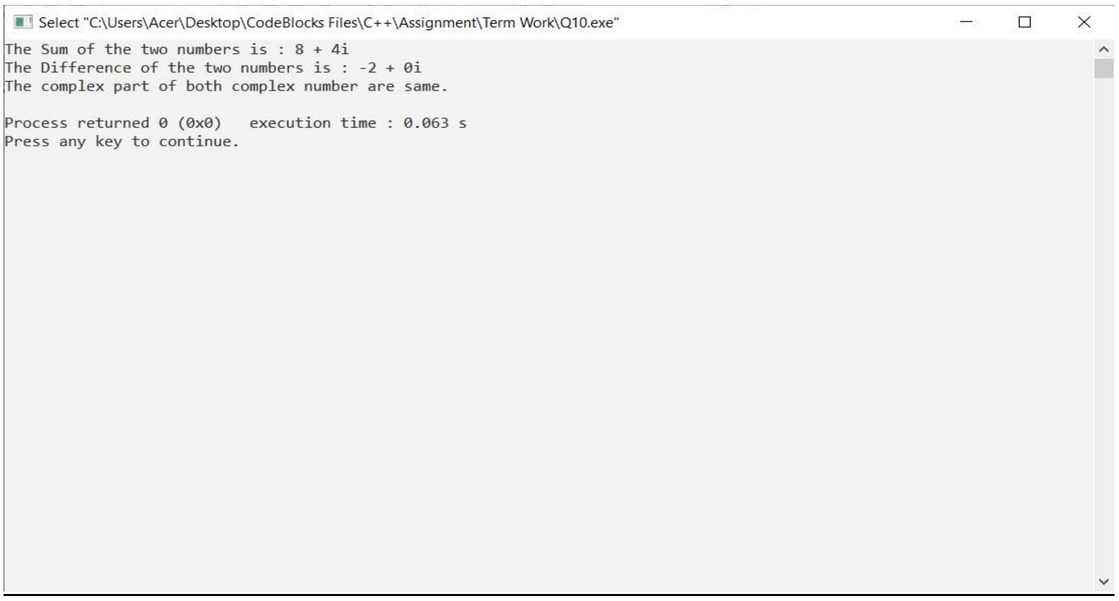
Select "C:\Users\Acer\Desktop\CodeBlocks Files\C++\Assignment\Term Work\Q9.exe"

```
Saver 1 Savings Balance: $2666.67
Saver 2 Savings Balance: $4000

Saver 1 Savings Balance: $3777.78
Saver 2 Savings Balance: $5666.67


Process returned 0 (0x0)   execution time : 0.078 s
Press any key to continue.
```

Q10. Create a class Complex having two int type variable named real & img denoting real and imaginary part respectively of a complex number. Overload + , - , == operator to add, to subtract and to compare two complex numbers being denoted by the two complex type objects.

ALGORITHM:
1. START
2. Declare the complex class.
3. Declare the variables(real , complex) and its member functions.
4. Use a parametrised constructor to get the value of the two variables.
5. Define the display function.
6. Declare the three functions as friend function(==(),+(),-()).
7. Define the function operator +() to add two complex numbers.
8. Define the function operator -() to subtract two complex numbers.
9. Define the function operator ==() to compare two complex numbers.
10. Declare the complex class objects c1,c2 , result.
11. Calculate the value for object result by calling function operator + and -.
12. Call the display function for object result to display the value of object result.
13. Call the function operator == for c1 and c2 to compare the two complex numbers.
14. STOP

PROGRAM:
```
#include<iostream>
using namespace std;

class complex{
   private:
   int real;
   int imag;
   public:
   complex(int r = 0, int i = 0){
      real = r;
      imag = i;
   }
   void display(){
      cout<<real<<" + "<<imag<<"i"<<endl;
   }
   friend complex operator+(complex c1,complex c2);
   friend complex operator-(complex c1,complex c2);
   friend void operator==(complex c1,complex c2);
};

complex operator+(complex c1,complex c2){
   complex t;
   t.real = c1.real + c2.real;
   t.imag = c1.imag + c2.imag;
   return t;

}
complex operator-(complex c1,complex c2){
   complex t;
   t.real = c1.real - c2.real;
   t.imag = c1.imag - c2.imag;
   return t;
}
void operator==(complex c1,complex c2){
   if(c1.real ==c2.real && c1.imag ==c2.imag){
      cout<<"Both complex numbers are same"<<endl;
   }
   if(c1.real == c2.real){
      cout<<"The real part of both the complex number are same."<<endl;
   }
   else if(c1.imag ==c2.imag){
      cout<<"The complex part of both complex number are same."<<endl;
   }
   else{
      cout<<"Both numbers are different."<<endl;
   }
}
int main(){
   complex c1(3,2);
   complex c2(5,2);
   complex result;
   cout<<"The Sum of the two numbers is : ";
   result = c1 + c2;
   result.display();
   cout<<"The Difference of the two numbers is : ";
   result = c1 - c2;
   result.display();
   c1 == c2;
}
```



Select "C:\Users\Acer\Desktop\CodeBlocks Files\C++\Assignment\Term Work\Q10.exe"

```
The Sum of the two numbers is : 8 + 4i
The Difference of the two numbers is : -2 + 0i
The complex part of both complex number are same.

Process returned 0 (0x0)   execution time : 0.063 s
Press any key to continue.
```

Q11. Using the concept of operator overloading.Write a program to overload using with and without friend Function.
a. Unary –
b. Unary ++ preincrement, postincrement
c. Unary -- predecrement, postdecrement

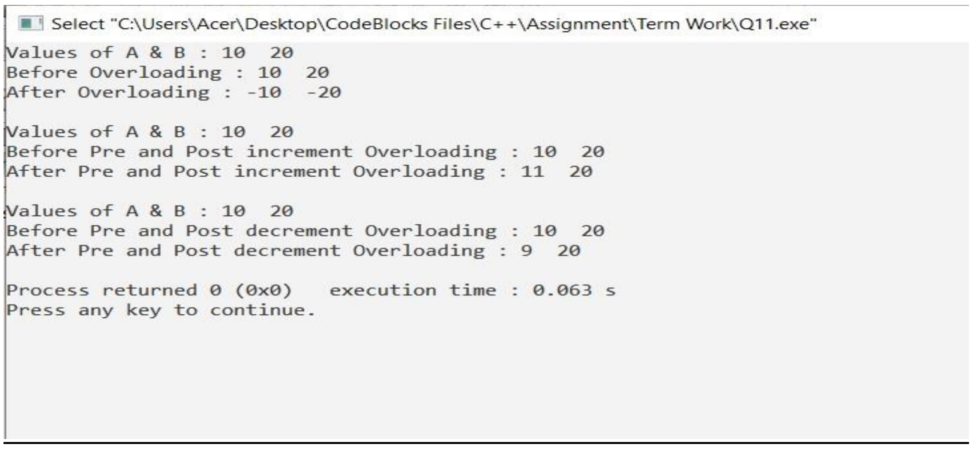
ALGORITHM:

- START
- Declare the class.
- Declare the variables and its member function.
- Using the function getvalues() to get the two numbers.
- Define the friend function operator +() for two numbers.
- Define the friend function operator –-() for both preincrement and postincrement of two numbers.
- Define the friend function operator ++()for both preincrement and postincrement of two numbers.
- Define the show() function.
- Declare the class objects x1, x2 and x3.
- Call the function getvalues using x1 and x2.
- Calculate the value for the object x3 by calling the function operator + and operator-- and operator ++.
- Call the show function using x1, x2 and x3.
- STOP

PROGRAM:

```
#include<iostream>
using namespace std;
class UnaryFriend
{
 int a=10;
 int b=20;
 public:
 void getvalues()
 {
 cout<<"Values of A & B : ";
 cout<<a<<"  "<<b<<"  "<<endl;
 }
 void show()
 {
 cout<<a<<"  "<<b<<"  "<<endl;
 }
 void friend operator-(UnaryFriend &x);
 void friend operator ++(UnaryFriend &x);
 void friend operator --(UnaryFriend &x);
};
void operator-(UnaryFriend &x)
{
 x.a = -x.a;
 x.b = -x.b;
}
 void operator++(UnaryFriend &x)
 {
```

```
 x.a=++x.a;
 x.b=x.b++;
 }
 void operator--(UnaryFriend &x)
 {
 x.a=--x.a;
 x.b=x.b--;
 }
int main()
{
 UnaryFriend x1;
 UnaryFriend x2;
 UnaryFriend x3;
 x1.getvalues();
 cout<<"Before Overloading : ";
 x1.show();
 cout<<"After Overloading : ";
 -x1;
 x1.show();
 cout<<endl;
 x2.getvalues();
 cout<<"Before Pre and Post increment Overloading : ";
 x2.show();
 cout<<"After Pre and Post increment Overloading : ";
 ++x2;
 x2.show();
 cout<<endl;
 x3.getvalues();
 cout<<"Before Pre and Post decrement Overloading : " ;
 x3.show();
 cout<<"After Pre and Post decrement Overloading : ";
 --x3;
 x3.show();
 return 0;
}
```



Select "C:\Users\Acer\Desktop\CodeBlocks Files\C++\Assignment\Term Work\Q11.exe"

```
Values of A & B : 10  20
Before Overloading : 10  20
After Overloading : -10  -20

Values of A & B : 10  20
Before Pre and Post increment Overloading : 10  20
After Pre and Post increment Overloading : 11  20

Values of A & B : 10  20
Before Pre and Post decrement Overloading : 10  20
After Pre and Post decrement Overloading : 9  20

Process returned 0 (0x0)   execution time : 0.063 s
Press any key to continue.
```

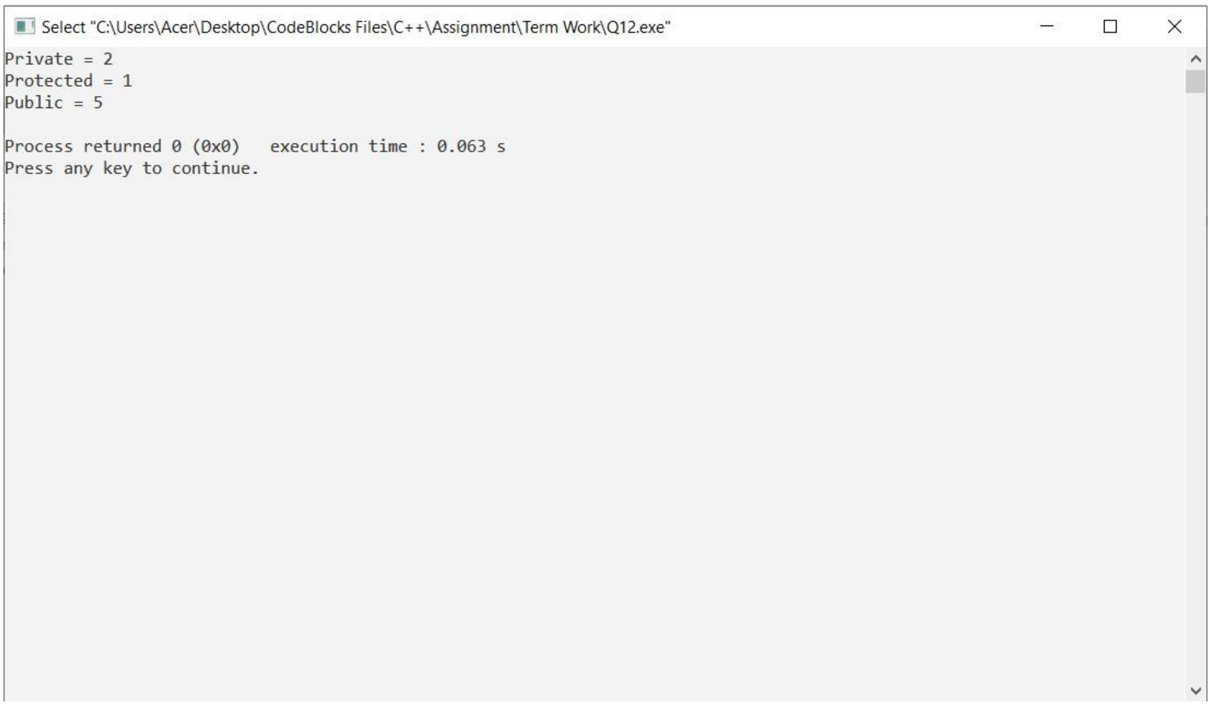
Q12. Create a Base class that consists of private, protected and public data members and member functions. Try using different access modifiers for inheriting Base class to the Derived class and create a table that summarizes the above three modes (when derived in public, protected and private modes) and shows the access specifier of the members of base class in the Derived class.

ALGORITHM:

```
1. Declare a base class BASE.
2. Declare its private a, public b and protected c data members.
3. create a void function get() to initialize variables.
4. Declare a derived class DER using public inheritance specifier.
5. Create a function void print() to print all inherited data members.
6. Repeat steps 4 to 5 for class DER2 and DER 3 using protected and private inheritance respectively.
7. Create main function and initialize one object for each class.
8. Call function print() for each object.
9. STOP
```

PROGRAM:

```
#include <iostream>
using namespace std;
class Base {
 private:
 int pvt = 2;
 protected:
 int prot = 1;
 public:
 int pub = 5;
 int getPVT() {
 return pvt;
 }
};
class PublicDerived : public Base {
 public:
 int getProt() {
 return prot;
 }
};
int main() {
 PublicDerived object1;
 cout << "Private = " << object1.getPVT() << endl;
 cout << "Protected = " << object1.getProt() << endl;
 cout << "Public = " << object1.pub << endl;
 return 0;
}
```


Select "C:\Users\Acer\Desktop\CodeBlocks Files\C++\Assignment\Term Work\Q12.exe"

```
Private = 2
Protected = 1
Public = 5

Process returned 0 (0x0)   execution time : 0.063 s
Press any key to continue.
```

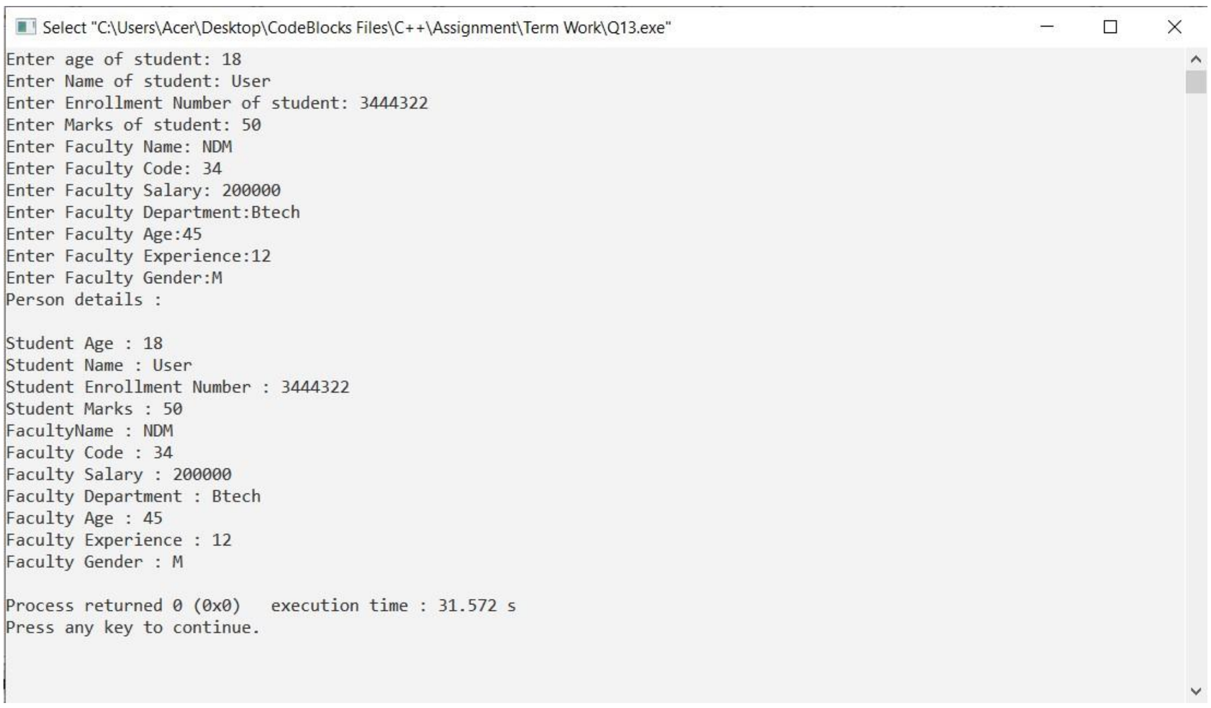
Q13. Create a class called Student that contains the data members like age, name, enroll_no, marks. Create another class called Faculty that contains data members like facultyName, facultyCode, salary, deptt, age, experience, gender. Create the function display() in both the classes to display the respective information. The derived Class Person demonstrates multiple inheritance. The program should be able to call both the base classes and displays their information. Remove the ambiguity (When we have exactly same variables or same methods in both the base classes, which one will be called?) by proper mechanism.

ALGORITHM:

- START.
- Create a class called student.
- Initialize private data members of class student name, enroll_no, age, marks.
- Create a data method to set the values of data members and display the values.
- Close the class.
- Create another class name faculty.
- Initialize private data members of class faculty faculty_name, salary, age, experience, dept, faculty code.
- Create a data method to set the values of data members and display the values.
- Create another class person .
- In main() create an object of class student, faculty and person and call the data methods,
- STOP.
-

PROGRAM:

```
#include<iostream>
using namespace std;
class Student
{
public:
 int age;
 string name;
 string enroll_no;
 float marks;
void getStudent()
 {
 cout << "Enter age of student: ";
cin >> age;
 cout << "Enter Name of student: ";
cin >> name;
cout << "Enter Enrollment Number of student: ";
cin >> enroll_no;
cout << "Enter Marks of student: "; cin >> marks;
 }
};
class Faculty
{
public:
 string facultyName;
 string facultyCode;
 float salary;
 string deptt;
 int age;
 float experience;
 string gender;
void getFaculty ()
{
 cout << "Enter Faculty Name: ";
 cin >> facultyName;
 cout << "Enter Faculty Code: ";
 cin >> facultyCode;
 cout << "Enter Faculty Salary: ";
 cin >> salary;
 cout << "Enter Faculty Department:";
 cin >> deptt;
 cout << "Enter Faculty Age:";
 cin >> age;
 cout << "Enter Faculty Experience:";
 cin >> experience;
 cout << "Enter Faculty Gender:";
 cin >> gender;
}
};
class Person : public Student , public Faculty
{
public:
void display()
{
cout<<"Person details : \n" << "\n"<< "Student Age : " << Student::age << "\n" << "Student Name : " <<
Student::name << "\n"<< "Student Enrollment Number : " <<Student::enroll_no << "\n" << "Student Marks : " <<
Student::marks << "\n" <<"FacultyName : " << Faculty::facultyName << "\n" << "Faculty Code : " <<
Faculty::facultyCode<< "\n" << "Faculty Salary : " << Faculty::salary << "\n" << "Faculty Department : "
<<Faculty::deptt << "\n" << "Faculty Age : " << Faculty::age << "\n" << "Faculty Experience : " <<
Faculty::experience <<"\n" << "Faculty Gender : " << Faculty::gender <<"\n" ;
}
};
int main()
{
Person obj1;
obj1.getStudent();
obj1.getFaculty();
 obj1.display();
return 0;
}
```



Select "C:\Users\Acer\Desktop\CodeBlocks Files\C++\Assignment\Term Work\Q13.exe"

```
Enter age of student: 18
Enter Name of student: User
Enter Enrollment Number of student: 3444322
Enter Marks of student: 50
Enter Faculty Name: NDM
Enter Faculty Code: 34
Enter Faculty Salary: 200000
Enter Faculty Department:Btech
Enter Faculty Age:45
Enter Faculty Experience:12
Enter Faculty Gender:M
Person details :

Student Age : 18
Student Name : User
Student Enrollment Number : 3444322
Student Marks : 50
FacultyName : NDM
Faculty Code : 34
Faculty Salary : 200000
Faculty Department : Btech
Faculty Age : 45
Faculty Experience : 12
Faculty Gender : M

Process returned 0 (0x0)   execution time : 31.572 s
Press any key to continue.
```

Q14. Create a base class called shape. Use this class to store two double type values that could be used to compute the area of figures. Derive two specific classes called triangle and rectangle from base shape. Add to the base class , a member function get_data() to initialize base class data members and another member function display_area() to compute and display the area of figures. Make display_area() as a virtual function and redefine this function in the derived class to suit their requirements. Using these three classes, design a program that will accept dimensions of a triangle or a rectangle interactively and display the area.
Remember the two values given as input will be treated as lengths of two sides in the case of rectangles and as base and height in the case of triangle and used as follows:
Area of rectangle = x * y
Area of triangle = ½ *x*y

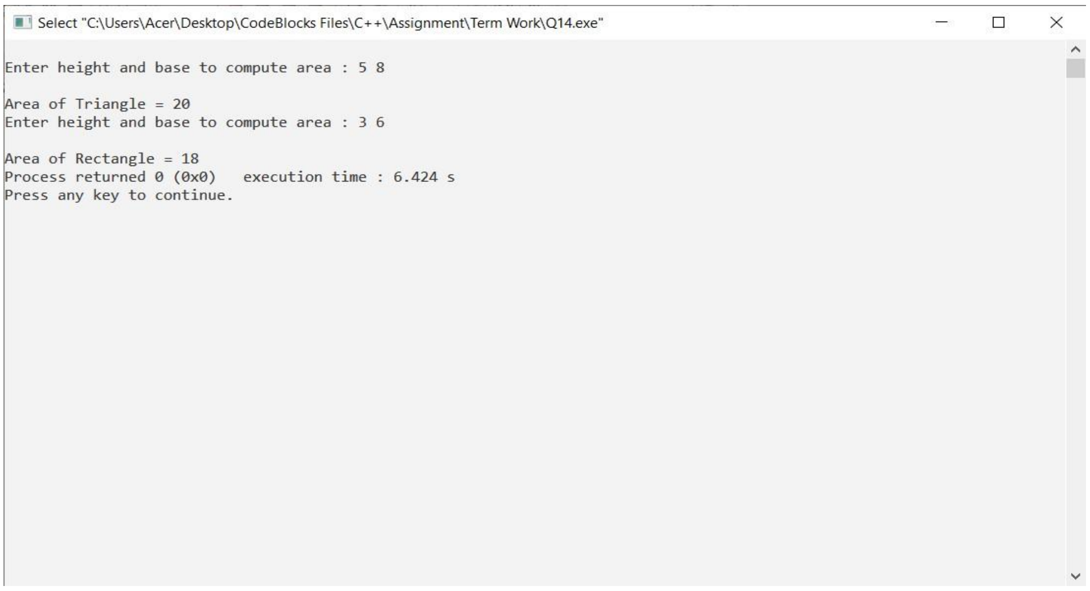
ALGORITHM:

- START
- declare a class shape
- declare data members and member functions of the class shape
- create a default constructor to initialize variables
- create a function void get_data() to enter height and base to compute area
- declare a virtual function void display_area()
- create a derived class triangle under class shape
- define virtual function display_area() and print the area of triangle
- create another derived class rectangle under class shape
- redefine virtual function void display_area() and calculate and print area of rectangle
- create a main function to create object and call the functions.
- STOP

PROGRAM:

```
#include<iostream>
using namespace std;
//class shape
class Shape
{
public:
double height,base;
Shape()
{
height=0;
base=0;
}
void get_data()
{
 cout<<"\nEnter height and base to compute area : ";
 cin>>height>>base;
}
virtual void display_area()
{
}
};
```

```
class Triangle : public Shape
{
public:
void display_area()
{
cout<<"\nArea of Triangle = "<<(height*base)/2;
}
};
class Rectangle : public Shape
{
public:
void display_area()
{
cout<<"\nArea of Rectangle = "<<height*base;
}
};
int main()
{
Shape *s;
Triangle t;
t.get_data();
s=&t;
s->display_area();
Rectangle r;
r.get_data();
s=&r;
s->display_area();
return 0;
}
```

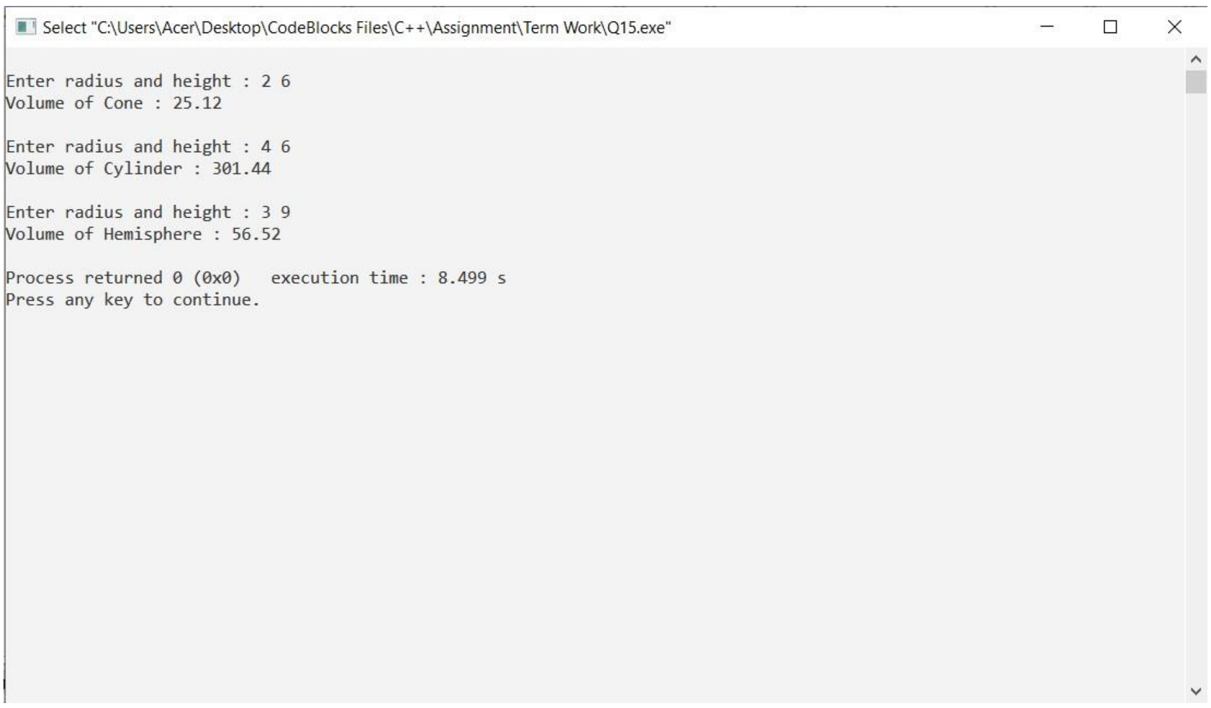

Select "C:\Users\Acer\Desktop\CodeBlocks Files\C++\Assignment\Term Work\Q14.exe"

```
Enter height and base to compute area : 5 8

Area of Triangle = 20
Enter height and base to compute area : 3 6

Area of Rectangle = 18
Process returned 0 (0x0)   execution time : 6.424 s
Press any key to continue.
```

Q15. Create a base class called CAL_AREA(Abstract). Use this class to store float type values that could be used to compute the volume of figures. Derive two specific classes called cone, hemisphere and cylinder from the base CAL_AREA. Add to the base class, a member function getdata ( ) to initialize base class data members and another member function display volume( ) to compute and display the volume of figures. Make display volume ( ) as a pure virtual function and redefine this function in the derived classes to suit their requirements. Using these three classes, design a program that will accept dimensions of a cone, cylinder and hemisphere interactively and display the volumes. Remember values given as input will be and used as follows:
Volume of cone = (1/3)πr2 h
Volume of hemisphere = (2/3)πr3
Volume of cylinder = πr2 h

ALGORITHM:
```
1. START
2. declare a class calc_area
3. declare data members and member functions of the class calc_area
4.create a default constructor to initialize variables
5.create a function void get_data() to enter height and base to compute area
6.declare a virtual function void display_volume()
7.create a derived class cone under class calc_area
8.define virtual function display_volume() and print the volume of cone
9.create another derived class hemisphere under class calc_area
10.redefine virtual function void display_volume() and calculate and print volume of hemisphere
11.create another derived class cylinder under class calc_area
12.redefine virtual function void display_volume() and calculate and print volume of cylinder
13.create a main function to create object and call the functions.
14.STOP.
```
PROGRAM:
```
#include<iostream>
#include <cmath>
#include <iomanip>
const float PI = 3.14;
using namespace std;
class Cal_area
{
 public:
 float r,h;
 Cal_Volume()
 {
 r=0;
 h=0;
 }
 void getdata ()
 {
 cout<<endl<<"Enter radius and height : ";
 cin>>r>>h;
 }
 virtual void display_volume() =0;
};
```

```
class Cone :public Cal_area
{
 public: void display_volume()
 {
 cout<<"Volume of Cone : "<<(1/(float)3)*PI*r*r*h<<endl ;
 }
};
class Hemisphere:public Cal_area
{
 public: void display_volume()
 {
 cout<<"Volume of Hemisphere : "<<(2/(float)3)*PI*r*r*r<<endl;
 }
};
class Cylinder: public Cal_area
{
 public: void display_volume ()
 {
 cout<<"Volume of Cylinder : "<<PI*r*r*h<<endl;
 }
};
int main()
{
 Cal_area *ca;
 Cone co;
 ca= &co;
 ca->getdata();
 ca->display_volume();
 Cal_area *cb;
 Cylinder cy;
 cb= &cy;
 cb->getdata();
 cb->display_volume();
 Cal_area *cc;
 Hemisphere he;
 cc= &he;
 cc->getdata();
 cc->display_volume();
 return 0;
}
```


Select "C:\Users\Acer\Desktop\CodeBlocks Files\C++\Assignment\Term Work\Q15.exe"

```
Enter radius and height : 2 6
Volume of Cone : 25.12

Enter radius and height : 4 6
Volume of Cylinder : 301.44

Enter radius and height : 3 9
Volume of Hemisphere : 56.52

Process returned 0 (0x0)   execution time : 8.499 s
Press any key to continue.
```

Q16. Implement a C++ program to demonstrate and understand Diamond problem.

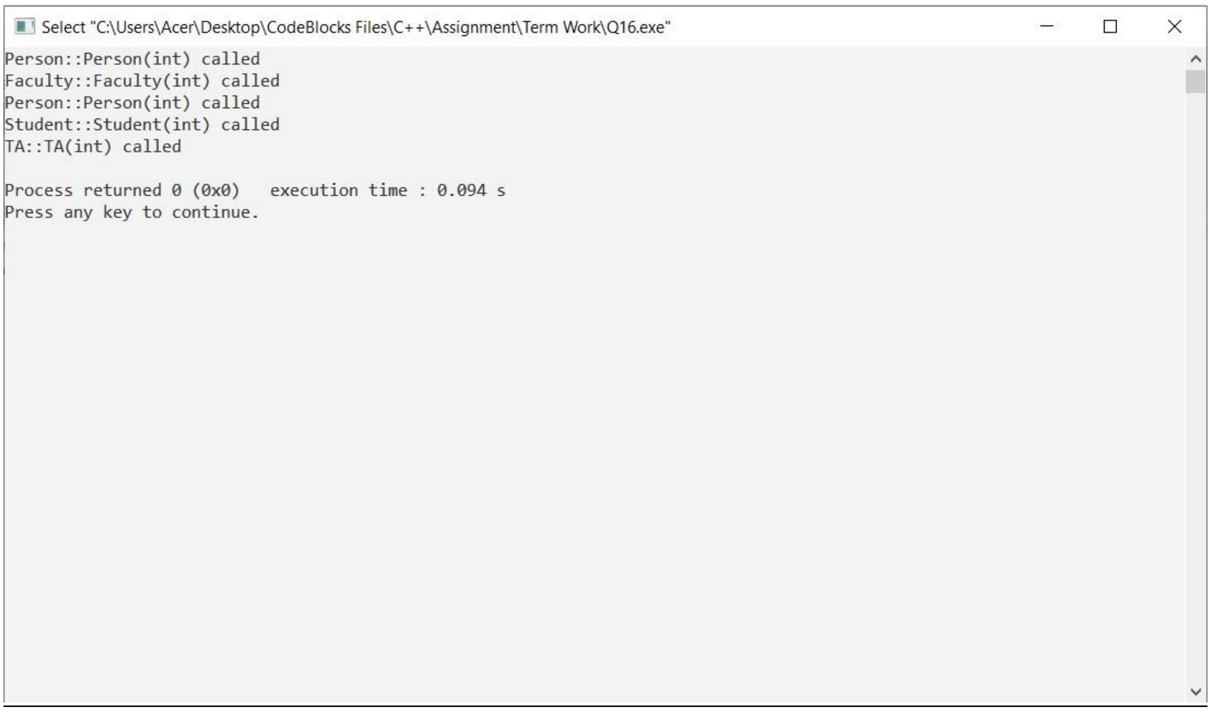
ALGORITHM:

- START
- Create a class Person.
- Create a parameterized constructor Person(int x) in class Person, where we print "Person::Person(int) called".
- Close the class Person.
- Create a class Faculty that publicly derives from class Person.
- Create a parameterized constructor Faculty(int x) in class Faculty that derives from the constructor of class Person, where we print "Faculty::Faculty(int) called".
- Close the class Faculty.
- Create a class Student that publicly derives from class Person.
- Create a parameterized constructor Student(int x) in class Student that derives from the parameterized constructor of class Person, where we print "Student::Student(int) called".
- Close the class Student.
- Create a class TA that publicly derives from class Faculty and class Student.
- Create a parameterized constructor TA(int x) in class TA that derives from the parameterized constructors of class Faculty and Student, where we print "TA::TA(int) called".
- Close the class TA.
- In main(), create an object tal of class TA that passes value 30.
- STOP

PROGRAM:

```
#include<iostream>
using namespace std;
class Person {
public:
Person(int x)
{
cout << "Person::Person(int ) called" << endl;
}
};
class Faculty : public Person {
public:
Faculty(int x):Person(x)
{
cout<<"Faculty::Faculty(int ) called"<< endl;
}
};
class Student : public Person {
public:
Student(int x):Person(x)
{
cout<<"Student::Student(int ) called"<< endl;
}
};
class TA : public Faculty, public Student {
public:
TA(int x):Student(x), Faculty(x)
{
cout<<"TA::TA(int ) called"<< endl;
}
};
int main()
```

```
{
TA ta1(30);
}
```



Select "C:\Users\Acer\Desktop\CodeBlocks Files\C++\Assignment\Term Work\Q16.exe"

```
Person::Person(int) called
Faculty::Faculty(int) called
Person::Person(int) called
Student::Student(int) called
TA::TA(int) called

Process returned 0 (0x0)   execution time : 0.094 s
Press any key to continue.
```