**LAB 08**

**QUESTION 1:**

**Create two classes, Speed and Time, that store speed (in km/h) and time (in hours),respectively. Declare a friend function named calculateDistance() that computes the distance traveled using both classes' private data.**

**Requirements:**

** Use a friend function to access private members from both classes.**

** Include a main function that instantiates objects with sample values (e.g., 60 km/h and 2 hours) and prints the calculated distance.**

**PROGRAM**:

#include<iostream>

using namespace std;

class Time;

class Speed{

 private:

 int speed;

 public:

 Speed(int *speed*):speed(speed){}

friend  float calcDistance(Time *t*,class Speed *s*);

};

class Time{

      private:

      int time;

      public:

      Time(int *time*): time(time){}

friend  float calcDistance(Time *t*,class Speed *s*);

};

float calcDistance(Time *t*,Speed *s*){

    int distance=0;

    distance=t.time \* s.speed;

    return distance;

}

int main(){

Time t(2);

Speed s(3);

float answer=0;

answer=calcDistance(t,s);

cout<<"distance is :"<<answer<<endl;

}

**RESULT:**

****

**QUESTION#2**

**Design a Car class that holds details such as the model name and price, and a Mechanic (or Technician) class that can view and modify these details. Make the Mechanic a friend class of Car so that it can apply discounts.**

**Requirements:**

** Implement a friend class declaration in Car.**

** In Mechanic, add functions to display car details and apply a discount (modify the price).**

** In main(), create a Car object, display its details, apply a discount, and display the updated details.**

**PROGRAM**:

#include<iostream>

using namespace std;

class car{

   private:

   string name;

   string model;

   int price;

    public:

  car(string *name*,string *model*,int *price*): name(*name*),model(*model*),price(*price*){}

    friend class Mechanic;

};

class Mechanic{

    public:

  void display(car *c*){

    cout<<"name :"<<*c*.name<<endl;

    cout<<"model :"<<*c*.model<<endl;

    cout<<"price :"<<*c*.price<<endl;

  }

  void discount(car *c*){

      if(*c*.price > 2500){

          cout<<"Congrats! you got discount"<<endl;

*c*.price=2500-*c*.price\*0.10;

          cout<<"new price of the car is :"<<*c*.price<<endl;

      }

      else{

        cout<<"you got no discount"<<endl;

      }

  }

};

int main(){

car c("Corolla","toyota",2800);

Mechanic m;

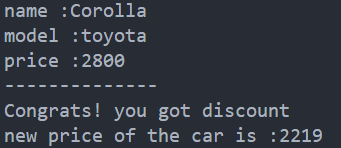
m.display(c);

cout<<"--------------"<<endl;

m.discount(c);

}

**RESULT:**

****

**QUESTION#3**

**Implement an Employee class that stores an employee's name and an array of three salaries (for three months). Also, create a Manager class (declared as a friend in Employee) that can update and view these salaries. Additionally, include a friend function to compute the average salary.**

**Requirements:**

** Use a friend class declaration so Manager can access and modify Employee's salaries.**

** Define a friend function calculateAverageSalary() to compute the average.**

** In main(), simulate a scenario by displaying salaries, updating one salary, and showing the new average.**

**PROGRAM**:

#include<iostream>

using namespace std;

class employee{

    private:

    string name;

    int salary[3];

    public:

    employee(string *name*,int *salary1*,int *salary2*,int *salary3*){

        this->name=*name*;

        salary[0]=*salary1*;

        salary[1]=*salary2*;

        salary[2]=*salary3*;

    }

    friend class Manager;

    friend int calculateAverageSalary(employee *e*);

};

int calculateAverageSalary(employee *e*){

    float average=(*e*.salary[0]+*e*.salary[1]+*e*.salary[2])/3;

    return average;

}

class Manager{

  public:

 void display(employee *e*){

    for(int i=0;i<3;i++){

        cout<<"salary "<<i+1<<":"<<*e*.salary[i]<<endl;

    }

 }

 void updateSalary(employee *e*){

    int index; int newSalary;

    cout<<"Enter which month salary do you want to change :";

    cin>>index;

     if(index >= 1 && index <=3){

        cout<<"Enter new salary :";

        cin>>newSalary;

*e*.salary[index-1]=newSalary;

        cout<<"new salary for month "<<index<<" is :"<<*e*.salary[index-1]<<endl;

     }

 }

};

int main(){

employee e("Umais",2500,2800,2600);

Manager m;

m.display(e);

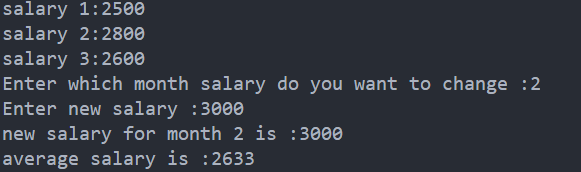
m.updateSalary(e);

float average= calculateAverageSalary(e);

cout<<"average salary is :"<<average<<endl;

}

**RESULT:**



**QUESTION#4**

**Design a BankAccount class that stores a customer's name, account balance, and an array of three recent transactions (in amounts). Additionally, create a BankManager class (declared as a friend in BankAccount) that can view and update these details. Include two friend functions:**

** calculateAverageTransaction() to compute the average of the last three transactions.**

** checkBalance() to return the current account balance.**

**Requirements:**

** Use a friend class declaration so BankManager can access and modify BankAccount's balance and transactions.**

** Define two friend functions: calculateAverageTransaction() to compute the average of the transactions, and checkBalance() to check the account balance.**

** In main(), simulate a banking scenario by displaying the recent transactions, updating one of them, checking the balance, and showing the average of the transactions.**

**PROGRAM**:

#include<iostream>

using namespace std;

class BankAccount{

    private:

    string name;

    float balance;

    float transaction[3];

    public:

    BankAccount(string *name*,float *balance*,int *transaction1*,int *transaction2*,int *transaction3*){

        this->name=*name*;

        this->balance=*balance*;

        transaction[0]=*transaction1*;

        transaction[1]=*transaction2*;

        transaction[2]=*transaction3*;

    }

    friend  class BankManager;

    friend float calculateAverageTransaction(BankAccount *a*);

    friend float checkBalance(BankAccount *a*);

};

float calculateAverageTransaction(BankAccount *a*){

   float average=(*a*.transaction[0]+*a*.transaction[1]+*a*.transaction[2])/3;

    return average;

}

float checkBalance(BankAccount *a*){

    cout<<"your current balance is "<<*a*.balance<<endl;

}

class BankManager{

   public:

   void display(BankAccount *a*){

    cout<<"name :"<<*a*.name<<endl;

    cout<<"name :"<<*a*.balance<<endl;

    cout<<"name :"<<*a*.name<<endl;

   }

   void updateBalance(BankAccount *a*){

     float Balance;

     cout<<"Enter new balance :"<<endl;

     cin>>Balance;

*a*.balance=Balance;

   }

    void updateTransaction(BankAccount *a*){

      int index; float newTrans;

    cout<<"Enter which month transaction do you want to change :";

    cin>>index;

     if(index >= 1 && index <=3){

        cout<<"Enter new transaction :";

        cin>>newTrans;

*a*.transaction[index-1]=newTrans;

        cout<<"new transaction for month "<<index<<" is :"<<*a*.transaction[index-1]<<endl;

     }

   }

};

int main(){

BankAccount a("Umais",30000,2500,2800,2600);

BankManager bm;

bm.display(a);

bm.updateBalance(a);

bm.updateTransaction(a);

float average= calculateAverageTransaction(a);

cout<<"average salary is :"<<average<<endl;

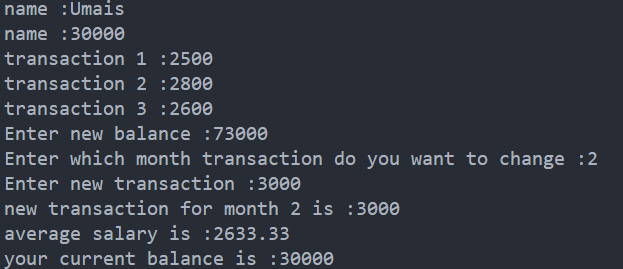
float balance;

balance=checkBalance(a);

cout<<"your current balance is :"<<balance<<endl;

}

**RESULT:**



**QUESTION#5**

**5. A C++ developer is building a ComplexNumber class to represent and perform operations on complex numbers. A complex number consists of a real part and an imaginary part, represented as a + bi, where a is the real part, and b is the imaginary part.The developer wants to support the following operations using operator overloading:**

** Addition (+): Allow two complex numbers to be added using the + operator. For example, adding 3 + 4i and 1 + 2i should result in 4 + 6i.**

** Subtraction (-): Allow two complex numbers to be subtracted using the - operator. For example,subtracting 5 + 6i from 8 + 3i should result in 3 - 3i.**

** Multiplication (\*): Allow two complex numbers to be multiplied using the \* operator. For example, multiplying 2 + 3i with 1 + 4i should result in -10 + 11i (since (2 + 3i) \* (1 + 4i) = -10 + 11i).**

** Equality Check (==): Implement the == operator to compare two complex numbers for equality.For example, 3 + 5i should be equal to 3 + 5i but not equal to 4 + 5i.Write a C++ class named ComplexNumber that implements these operations using operator overloading.**

**PROGRAM**:

#include<iostream>

using namespace std;

class ComplexNumber{

   private:

    int a;

    int b;

    public:

ComplexNumber(){ }

ComplexNumber(int *a*,int *b*){

        this->a=*a*;

        this->b=*b*;

}

ComplexNumber operator+(ComplexNumber &*c*){

       ComplexNumber temp;

       temp.a=a+*c*.a;

       temp.b=b + *c*.b;

       return temp;

}

ComplexNumber operator-(ComplexNumber &*c*){

    ComplexNumber temp;

    temp.a=a - *c*.a;

    temp.b=b - *c*.b;

    return temp;

}

ComplexNumber operator\*(ComplexNumber &*c*){

    ComplexNumber temp;

    temp.a=a\**c*.a;

    temp.b=b \* *c*.b;

    return temp;

}

ComplexNumber operator==(ComplexNumber &*c*){

    ComplexNumber temp;

    if(a==*c*.a & b==*c*.b){

        cout<<"both complex number are equal "<<endl;

    }

    else{

        cout<<"both complex number are not equal"<<endl;

    }

}

void display(){

    cout<<a<<" "<<b<<"i"<<endl;

}

};

int main(){

  ComplexNumber c(3,4);

  ComplexNumber c1(5,6);

  ComplexNumber c2=c+c1;

  c2.display();

  ComplexNumber c3=c-c1;

  c3.display();

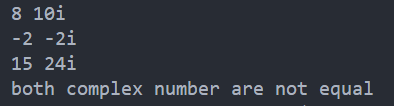
  ComplexNumber c4=c\*c1;

  c4.display();

  c==c1;

}

**RESULT:**

****