Name : Aritra Das

Roll No: 75

Assignment 5

1.

/\*

There are number of students. For every student roll (unique), name is to be

stored. For each subject, subject code and name is to be stored. A student can

opt for number of subjects. System should be able to maintain student list,

subject list and will be able to answer: i) which student has selected which

subjects and ii) for a subjects who are the students.

Design the classes and implement. For list consider memory data structure.

\*/

#include<iostream>

#include<string>

#include<vector>

using namespace std;

class Student{

    public:

    int roll;

    string name;

    Student\* next;

    vector<int> sb1;

};

class StuList{

    private:

    Student\* head;

    public:

    StuList(){

        head = NULL;

    }

    bool checkUnique(int x){

        Student \*temp1;

        temp1= head;

        while(temp1!=NULL){

            if(temp1->roll==x)

            return false;

            temp1=temp1->next;

        }

        return true;

    }

    void push(int x,string y){

        Student \*temp = new Student;

        temp->roll = x;

        temp->name = y;

        temp ->next = NULL;

        if(!checkUnique(x))

        {

            cout<<"Roll Number not unique\n";

            return;

        }

        else{

            if(head == NULL)

            head = temp;

            else{

                Student \*temp1;

                temp1= head;

                while(temp1->next!=NULL)

                temp1=temp1->next;

                temp1->next = temp;

            }

        }

    }

    void display(){

        Student \*temp1;

        temp1= head;

        while(temp1!=NULL){

            cout<<"Roll No : "<<temp1->roll<<" Name: "<<temp1->name<<endl;

            temp1=temp1->next;

        }

    }

    void add(int rll,int code){

        Student \*temp1;

        temp1= head;

        while(temp1!=NULL){

            if(temp1->roll==rll)

            break;

            temp1=temp1->next;

        }

        temp1->sb1.push\_back(code);

    }

    vector<int> disSub(int rll){

        Student \*temp1;

        temp1= head;

        while(temp1!=NULL){

            if(temp1->roll==rll)

            break;

            temp1=temp1->next;

        }

        return temp1->sb1;

    }

    void display(int rll){

        Student \*temp1;

        temp1= head;

        while(temp1!=NULL){

            if(temp1->roll==rll){

                cout<<"Roll No : "<<temp1->roll<<" Name: "<<temp1->name<<endl;

                break;

            }

            temp1=temp1->next;

        }

    }

};

class Subject{

    public:

    int code;

    string name;

    Subject\* next;

    vector<int> sl1;

};

class SubList{

    private:

    Subject\* head;

    public:

    SubList(){

        head = NULL;

    }

    void push(int x,string y){

        Subject \*temp = new Subject;

        temp->code = x;

        temp->name = y;

        temp ->next = NULL;

        if(head == NULL)

        head = temp;

        else{

            Subject \*temp1;

            temp1= head;

            while(temp1->next!=NULL)

            temp1=temp1->next;

            temp1->next = temp;

        }

    }

    void display(){

        Subject \*temp1;

        temp1= head;

        while(temp1!=NULL){

            cout<<"Code No : "<<temp1->code<<" Name: "<<temp1->name<<endl;

            temp1=temp1->next;

        }

    }

    void display(int cde){

        Subject \*temp1;

        temp1= head;

        while(temp1!=NULL){

            if(temp1->code==cde){

                cout<<"Code No : "<<temp1->code<<" Name: "<<temp1->name<<endl;

                break;

            }

            temp1=temp1->next;

        }

    }

    void add(int cde, int roll){

        Subject \*temp1;

        temp1= head;

        while(temp1!=NULL){

            if(temp1->code==cde)

            break;

            temp1=temp1->next;

        }

        temp1->sl1.push\_back(roll);

    }

    vector<int> disStu(int cde){

        Subject \*temp1;

        temp1= head;

        while(temp1!=NULL){

            if(temp1->code==cde)

            break;

            temp1=temp1->next;

        }

        return temp1->sl1;

    }

};

int main(){

    StuList a;

    SubList b;

    for(;;){

        cout<<"Enter : \n1. To add subject\n2. To add student\n3. To assign subject to a particular student\n4. To show all students for a particular subject\n5. To show all subjects of a particular student\n6. Exit\n";

        int c;

        cin>>c;

        switch(c){

            case 1:{

                cout<<"Enter subject code and name\n";

                int x;

                string y;

                cin>>x>>y;

                b.push(x,y);

                break;

            }

            case 2:{

                cout<<"Enter student roll and name\n";

                int x;

                string y;

                cin>>x>>y;

                a.push(x,y);

                break;

            }

            case 3:{

                cout<<"Available subjects: \n";

                b.display();

                cout<<"Enter student roll and subject code \n";

                int x,y;

                cin>>x>>y;

                a.add(x,y);

                b.add(y,x);

                break;

            }

            case 4:{

                cout<<"Enter subject code \n";

                int x;

                cin>>x;

                vector<int> l(b.disStu(x));

                cout<<"Student list is: \n";

                for (auto i = l.begin(); i != l.end(); ++i)

                a.display(\*i);

                break;

            }

            case 5:{

                cout<<"Enter student roll \n";

                int x;

                cin>>x;

                vector<int> p(a.disSub(x));

                cout<<"Subject list is: \n";

                for (auto j = p.begin(); j != p.end(); ++j)

                b.display(\*j);

                break;

            }

            case 6:{

                cout<<"Exiting ...";

                exit(0);

                break;

            }

            default:{

                cout<<"Wrong Input";

                break;

            }

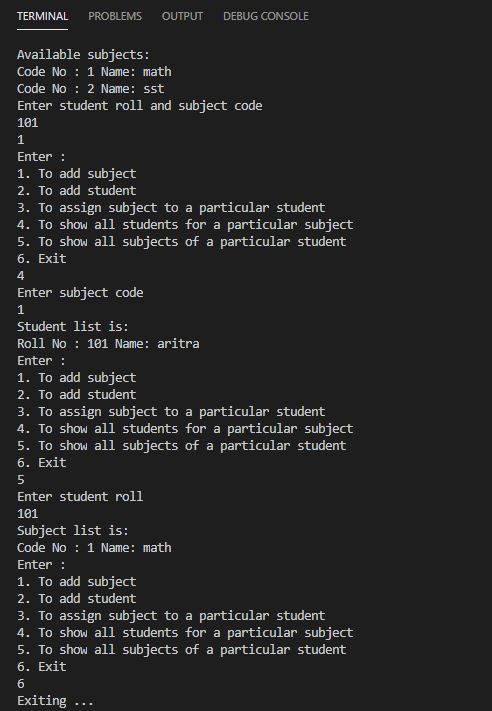
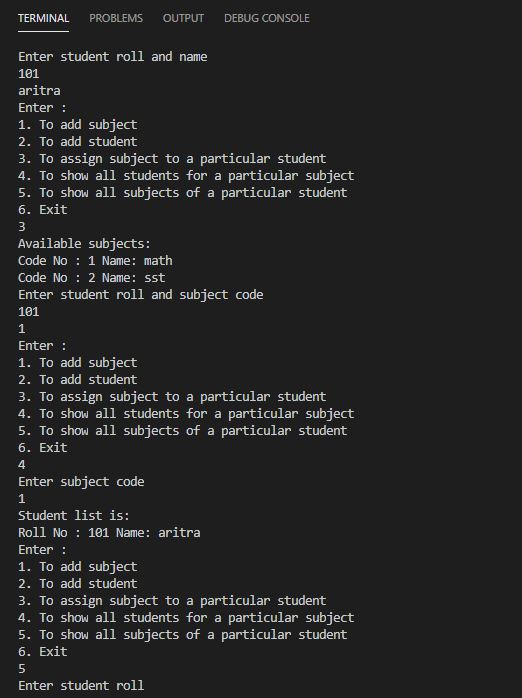
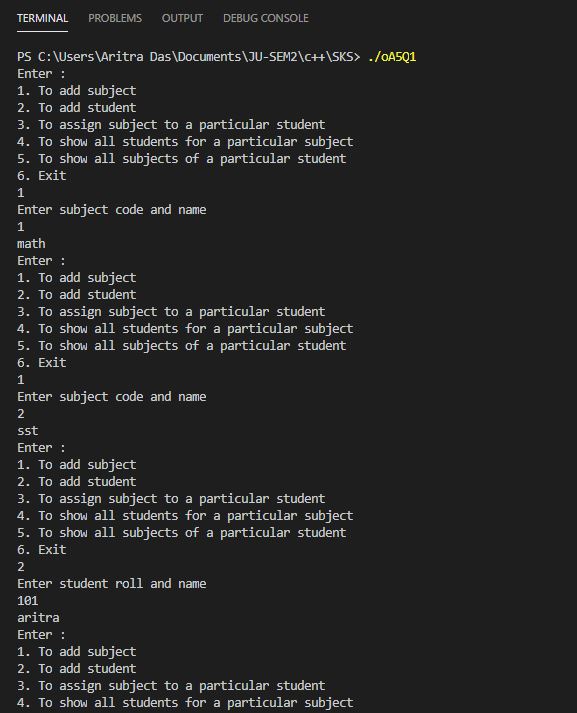
        }

    }

    return 0;

}

Output:



2.

/\*

In a library, for each book book-id, serial number (denotes copy number of a

book), title, author, publisher and price are stored. Book-id and serial number

together will be unique identifier for a book. Members are either student or

faculty. Each member has unique member-id. Name, e-mail, address are also to

be stored. For any transaction (book issue or return), members are supposed to

place transactions slip. User will submit member-id, book-id, and serial number

(only for book return). While processing a transaction, check the validity of the

member. While issuing, availability of a copy of the book is to be checked. While

returning a book, it is to be checked whether this copy was issued to the member

or not. A student member can have 2 books issued at a point of time. For faculty

members it is 10. Transaction information is to be stored like date of transaction,

member-id, book-id, serial number, returned or not. An entry is made when book

is issued and updated when the book is returned.

Design the classes and implement. For list consider memory data structure.

\*/

#include<iostream>

#include<string>

#include<vector>

#include<algorithm>

using namespace std;

class Book{

    public:

    int b\_id,s\_no,price;

    string title,author,pub;

    vector<int> m\_list;

    Book \*next;

};

class Member{

    public:

    int m\_id;

    string name,email;

    int count;

    char type;

    vector<int> b\_list;

    Member \*next;

};

class BookList{

    Book \*head;

    public:

    BookList(){

        head = NULL;

    }

    void push(int x, int t,string y,string z, string p){

        Book \*temp = new Book;

        temp->b\_id = x;

        temp->s\_no = 1;

        temp->title = y;

        temp->author = z;

        temp->pub = p;

        temp->price = t;

        temp ->next = NULL;

        if(head == NULL)

        head = temp;

        else{

            Book\* temp1;

            temp1 = head;

            while(temp1->next=NULL){

                if(!checkUnique(x)){

                    cout<<"Book ID already exists, increasing the Serial Count\n";

                    temp1->s\_no++;

                    return;

                }

                temp1=temp1->next;

            }

            temp1->next = temp;

        }

    }

    void display(){

        Book \*temp1;

        temp1= head;

        while(temp1!=NULL){

            cout<<"Book ID : "<<temp1->b\_id<<" Title: "<<temp1->title<<" Author: "<<temp1->author<<" Publisher: "<<temp1->pub<<" Serial No: "<<temp1->s\_no<<" Price: "<<temp1->price<<endl;

            temp1=temp1->next;

        }

    }

    bool checkUnique(int acNo1){

        Book\* temp;

        temp = head;

        if(head==NULL)

        return true;

        while(temp!=NULL){

            if(temp->b\_id == acNo1)

            return false;

            temp = temp->next;

        }

        return true;

    }

    void add\_M\_List(int x,int y){

        Book\* temp;

        temp = head;

        while(temp!=NULL&&temp->b\_id != y){

            temp = temp->next;

        }

        temp->m\_list.push\_back(x);

    }

    void remove\_M\_List(int x,int y){

        Book\* temp;

        temp = head;

        while(temp!=NULL&&temp->b\_id != y){

            temp = temp->next;

        }

        for (auto i =temp->m\_list.begin();  i != temp->m\_list.end(); ++i){

            if(\*i==x){

                temp->m\_list.erase(i);

                break;

            }

        }

    }

    void inc\_count(int x){

        Book\* temp;

        temp = head;

        while(temp!=NULL&&temp->b\_id != x){

            temp = temp->next;

        }

        temp->s\_no++;

    }

    void dec\_count(int x){

         Book\* temp;

        temp = head;

        while(temp!=NULL&&temp->b\_id != x){

            temp = temp->next;

        }

        if(temp->s\_no>0)

        temp->s\_no--;

        else

        cout<<"Insufficient Books\n";

    }

};

class MemList{

    Member \*head;

    public:

    MemList(){

        head = NULL;

    }

    void push(int x,string y,string z,char t){

        Member \*temp = new Member;

        temp->m\_id = x;

        temp->email = z;

        temp->name = y;

        temp->type = t;

        temp->count = 0;

        temp ->next = NULL;

        if(head == NULL)

        head = temp;

        else{

            Member\* temp1;

            temp1 = head;

            while(temp1->next=NULL){

                if(!checkUnique(x)){

                    cout<<"Member already exists\n";

                    return;

                }

                temp1=temp1->next;

            }

            temp1->next = temp;

        }

    }

    void display(){

        Member \*temp1;

        temp1= head;

        while(temp1!=NULL){

            cout<<"Member ID : "<<temp1->m\_id<<" Email: "<<temp1->email<<" Name: "<<temp1->name<<endl;

            temp1=temp1->next;

        }

    }

    bool checkUnique(int acNo1){

        Member\* temp;

        temp = head;

        if(head==NULL)

        return true;

        while(temp!=NULL){

            if(temp->m\_id == acNo1)

            return false;

            temp = temp->next;

        }

        return true;

    }

    void add\_B\_List(int y,int x){

        Member\* temp;

        temp = head;

        while(temp!=NULL&&temp->m\_id != y){

            temp = temp->next;

        }

        if(temp->type == 's' && temp->count < 3)

        temp->b\_list.push\_back(x);

        else if(temp->type == 'f' && temp->count < 8)

        temp->b\_list.push\_back(x);

        else

        cout<<"Limit exceeded\n";

    }

    void remove\_B\_List(int y,int x){

        Member\* temp;

        temp = head;

        while(temp!=NULL&&temp->m\_id != y){

            temp = temp->next;

        }

        for (auto i =temp->b\_list.begin();  i != temp->b\_list.end(); ++i){

            if(\*i==x){

                temp->b\_list.erase(i);

                break;

            }

        }

    }

};

class Transaction{

    public:

    BookList a;

    MemList a1;

    void addB(){

        int b,c;

        string d,e,f;

        cout<<"Enter Book ID, Title, Author, Publisher, Price:\n";

        cin>>b>>d>>f>>e>>c;

        a.push(b,c,d,e,f);

    }

    void viewB(){

        cout<<"The list of books are:\n";

        a.display();

    }

    void addS(){

        int b;

        string d,e;

        cout<<"Enter Member ID, Name, Email:\n";

        cin>>b>>d>>e;

        a1.push(b,d,e,'s');

    }

    void addM(){

        int b;

        string d,e;

        cout<<"Enter Member ID, Name, Email:\n";

        cin>>b>>d>>e;

        a1.push(b,d,e,'f');

    }

    void viewM(){

        cout<<"The list of Members are:\n";

        a1.display();

    }

    void issue(){

        cout<<"Available Books:\n";

        a.display();

        int k,l;

        cout<<"Enter Book ID and Member ID\n";

        cin>>k>>l;

        a1.add\_B\_List(k,l);

        a.add\_M\_List(l,k);

        a.dec\_count(k);

        cout<<"Book Issued\n";

    }

    void retur(){

        int k,l;

        cout<<"Enter Book ID and Member ID\n";

        cin>>k>>l;

        a1.remove\_B\_List(k,l);

        a.remove\_M\_List(l,k);

        a.inc\_count(k);

        cout<<"Book Returned\n";

    }

};

int main(){

    Transaction t;

    for(;;){

        cout<<"Enter:\n1. Add a Book\n2. Add a Student\n3. Add a Faculty\n4. Issue a Book\n5. Return a Book\n6. Show all Books\n7. Show all Members\n8. Exit\n";

        int c;

        cin>>c;

        switch(c){

            case 1:{

                t.addB();

                break;

            }

            case 2:{

                t.addS();

                break;

            }

            case 3:{

                t.addM();

                break;

            }

            case 4:{

                t.issue();

                break;

            }

            case 5:{

                t.retur();

                break;

            }

            case 6:{

                t.viewB();

                break;

            }

            case 7:{

                t.viewM();

                break;

            }

            case 8:{

                cout<<"Exiting ... \n";

                exit(0);

            }

            default:{

                cout<<"Wrong choice\n";

                break;

            }

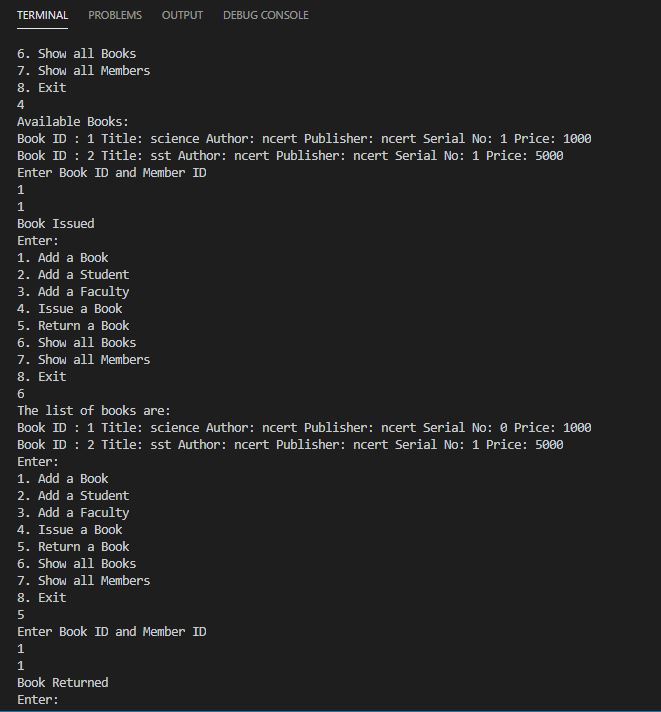
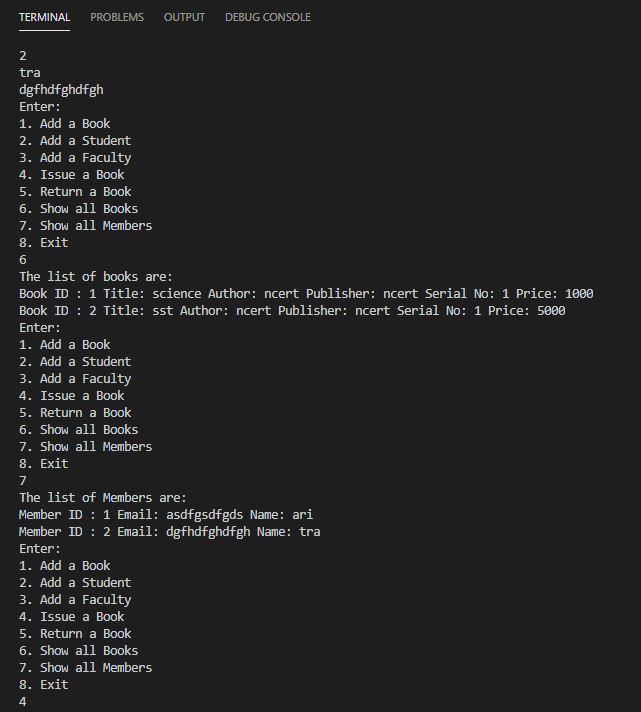
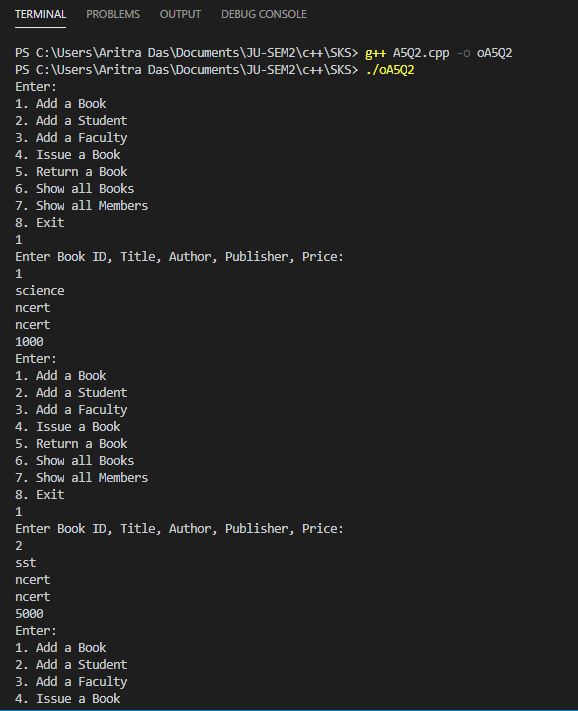
        }

    }

    return 0;

}

Output:



3.

/\*

Employee has unique emp-id, name, designation and basic pay. An employee is

either a permanent one or contractual. For permanent employee salary is

computed as basic pay+ hra (30% of basic pay) + da (80% of basic pay). For

contractual employee it is basic pay + allowance (it is different for different

contractual employee). An employee pointer may point to either of the two

categories and accordingly the salary has to be created.

Design the classes and implement.

\*/

#include<iostream>

#include<string>

#include<vector>

using namespace std;

class Employee{

    // int empId;

    string name,desig;

    public:

    int empId;

    int basicPay,finalPay;

    void show(){

        cout<<"Employee ID: "<<empId<<" | Employee Name: "<<name<<" | Employee Designation: "<<desig<<" | Employee Basic-Pay: "<<basicPay<<" | Employee Final-Pay "<<finalPay<<endl;

    }

    void get(){

        cout<<"Enter Employee ID, Name, Designation, Basic-Pay\n";

        cin>>empId>>name>>desig>>basicPay;

    }

    // int getId(){return empId;}

};

class Permanent: public Employee{

    public:

    Permanent\* next;

    Permanent(){

        get();

    }

    void genFinal(){

        finalPay = basicPay + 0.3\*basicPay +0.8\* basicPay;

    }

};

class Temporary: public Employee{

    public:

    Temporary\* next;

    Temporary(){

        get();

    }

    int allowance;

    void genFinal(){

        cout<<"Enter allowance: \n";

        cin>>allowance;

        finalPay = basicPay + allowance;

    }

};

class PermList{

    Permanent\* head;

    public:

    PermList(){

        head = NULL;

    }

    bool checkUnique(int x){

        //cout<<"Checking\n";

        Permanent\* temp;

        temp = head;

        if(head==NULL)

        return true;

        while(temp!=NULL){

            if(temp->empId==x)

            return false;

            temp = temp->next;

        }

        return true;

    }

    void push(){

        Permanent\* temp = new Permanent;

        temp->next = NULL;

        temp->genFinal();

        cout<<"Account Created\n";

        if(head == NULL){

            head = temp;

        }

        else{

            Permanent\* temp1;

            temp1 = head;

            while(temp1->next!=NULL){

                if(!checkUnique(temp->empId)){

                    cout<<"Account already exists\n";

                    return;

                }

                temp1=temp1->next;

            }

            if(!checkUnique(temp->empId)){

                cout<<"Account already exists\n";

                return;

            }

            temp1->next = temp;

        }

    }

    void display(){

        Permanent\* temp ;

        temp = head;

        while(temp!=NULL){

            temp->show();

            temp = temp->next;

        }

    }

};

class TempList{

    Temporary\* head;

    public:

    TempList(){

        head = NULL;

    }

    bool checkUnique(int x){

        //cout<<"Checking\n";

        Temporary\* temp;

        temp = head;

        if(head==NULL)

        return true;

        while(temp!=NULL){

            if(temp->empId==x)

            return false;

            temp = temp->next;

        }

        return true;

    }

    void push(){

        Temporary\* temp = new Temporary;

        temp->next = NULL;

        temp->genFinal();

        cout<<"Account Created\n";

        if(head == NULL){

            head = temp;

        }

        else{

            Temporary\* temp1;

            temp1 = head;

            while(temp1->next!=NULL){

                if(!checkUnique(temp->empId)){

                    cout<<"Account already exists\n";

                    return;

                }

                temp1=temp1->next;

            }

            if(!checkUnique(temp->empId)){

                cout<<"Account already exists\n";

                return;

            }

            temp1->next = temp;

        }

    }

    void display(){

        Temporary\* temp ;

        temp = head;

        while(temp!=NULL){

            temp->show();

            temp = temp->next;

        }

    }

};

int main(){

    PermList v1;

    TempList v2;

    for(;;){

        cout<<"Enter :\n1. To create new Permanent Employee\n2. to create new Temporary Employee\n3. To show all Employees\n4. To Exit\n";

        int c;

        cin>>c;

        switch(c){

            case 1:{

                v1.push();

                break;

            }

            case 2:{

                v2.push();

                break;

            }

            case 3:{

                cout<<"Permanent Employees:\n";

                v1.display();

                cout<<"Temporary Employees:\n";

                v2.display();

                break;

            }

            case 4:{

                cout<<"Exiting ... ";

                exit(0);

            }

            default:{

                cout<<"Wrong Input\n";

                break;

            }

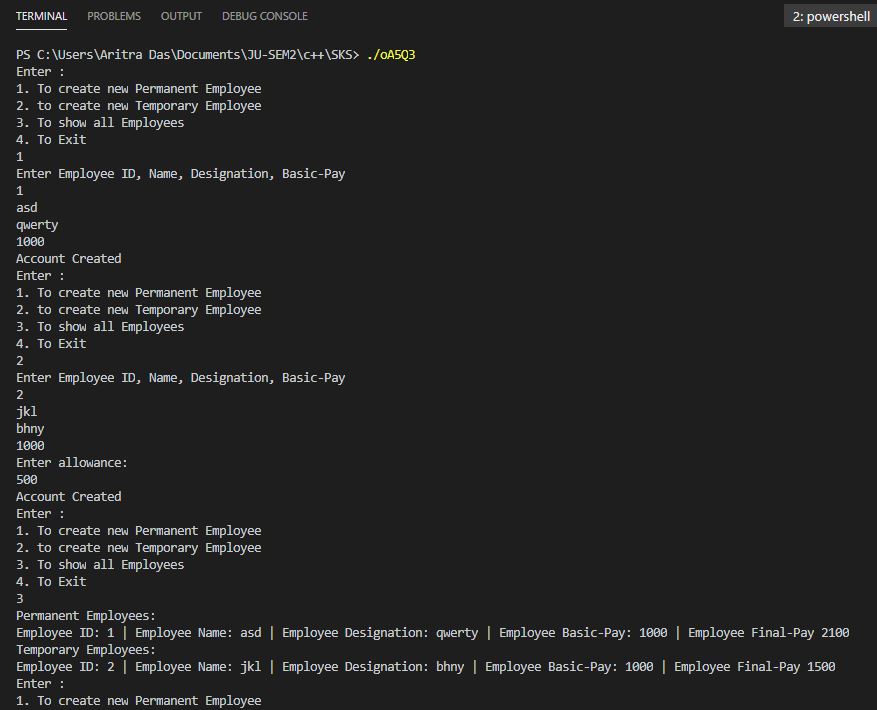
        }

    }

    return 0;

}

Output:



4.

/\*

Each cricketer has name, date of birth and matches played. Cricketer may be a

bowler or batsman. For a bowler, number of wickets taken, average economy is

stored. For a batsman, total runs scored, average score is stored. A double

wicket pair is formed taking a bowler and a batsman. An all-rounder is both a

bowler and batsman. Support must be there to show the details of a cricketer,

bowler, batsmen, all-rounder and the pair.

Design the classes and implement.

\*/

#include<iostream>

#include<string>

using namespace std;

class Cricketer{

    private:

    int dob;

    string name;

    public:

    int match;

    void setDeet(){

        int x,z;

        string y;

        cout<<"Enter Name, DOB and matches played\n";

        cin>>y>>x>>z;

        dob = x;

        match = z;

        name = y;

    }

    void getDeet(){

        cout<<"Name of player: "<<name<<"  Total no of matches played: "<<match<<"  Date of birth: "<<dob<<endl;

    }

};

class Bowler: public Cricketer{

    private:

    int wickets,avgEco;

    public:

    Bowler(){

        setDeet();

    }

    void setBolwer(){

        int x, y;

        cout<<"Enter Total Wickets and Total runs given by Bowler\n";

        cin>>x>>y;

        wickets = x;

        avgEco = y/match;

    }

    void getBowler(){

        getDeet();

        cout<<"Total wickets taken: "<<wickets<<" .Average economy: "<<avgEco<<endl;

    }

};

class Batsman: public Cricketer{

    private:

    int total,avgRun;

    public:

    Batsman(){

        setDeet();

    }

    void setBatsman(){

        int x;

        cout<<"Enter Total Runs of Batsman\n";

        cin>>x;

        total = x;

        avgRun = total/match;

    }

    void getBatsman(){

        getDeet();

        cout<<"Total runs scored: "<<total<<" .Average runs scored: "<<avgRun<<endl;

    }

};

class Pair{

    Batsman a;

    Bowler b;

    public:

    Pair(){

        a.setBatsman();

        b.setBolwer();

    }

    void showPair(){

        a.getBatsman();

        cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

        b.getBowler();

    }

};

class AllRounder:public Batsman,public Bowler{

    public:

    AllRounder(){

        setBatsman();

        setBolwer();

    }

    void getAllrounder(){

        getBatsman();

        getBowler();

    }

};

int main(){

    for(;;){

        cout<<"Enter :\n1. For Batsman \n2. For Bowler\n3. For All Rounder\n4. For Pair\n5. Exit\n";

        int c;

        cin>>c;

        switch(c){

            case 1:{

                Batsman t;

                t.setBatsman();

                cout<<"Details of Batsman:\n";

                t.getBatsman();

                break;

            }

            case 2:{

                Bowler t;

                t.setBolwer();

                cout<<"Details of Bowler:\n";

                t.getBowler();

                break;

            }

            case 3:{

                AllRounder t;

                cout<<"Details of All Rounder:\n";

                t.getAllrounder();

                break;

            }

            case 4:{

                Pair t;

                cout<<"Details of Pair:\n";

                t.showPair();

                break;

            }

            case 5:{

                cout<<"Exiting ...";

                exit(0);

            }

        }

    }

    return 0;

}

Output:

