Bank Management System

Prepared by: Karray Mahdi

GENERAL IDEA OF THE PROJECT:

In this project, I tried to show the working of a banking system and cover the basic functionality of a Bank Account Management System. It’s an application for maintaining a person's account in a bank.
It gives access to staff of the bank for adding accounts, deleting accounts, modifying accounts, withdraw and deposit. each Person has his/her own username and account number.

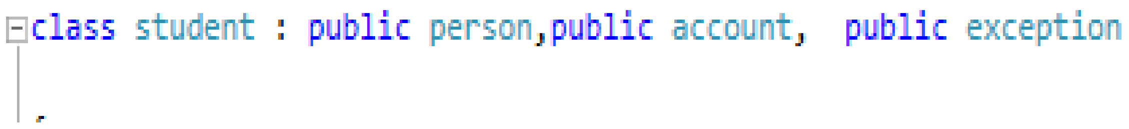
For this application, I needed to implement four C++ techniques to make these functions possible. So, the techniques which I implemented are:

- Inheritance
- Polymorphism
- File management
- Exception management

Inheritance:

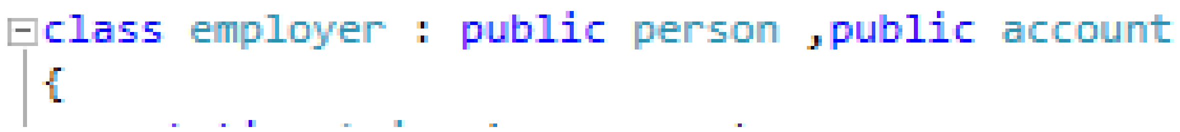
- I implement inheritance for my project because I wanted to have 2 types of accounts one for the students and the other for the employers.
- I have two parent classes:
- 1 / Person
- 2/ Account
- And 2 child classes:
- 1/student
- 2/employer

- for the child classes: I used multiple inheritance:
- for the student class:



```
class student : public person,public account,  public exception
```

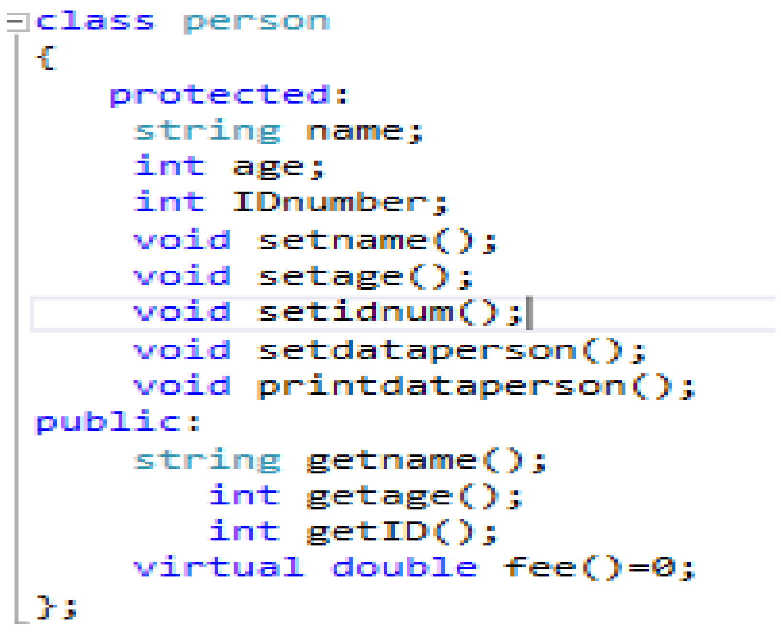
- for the employer class :



```
class employer : public person ,public account
{
```

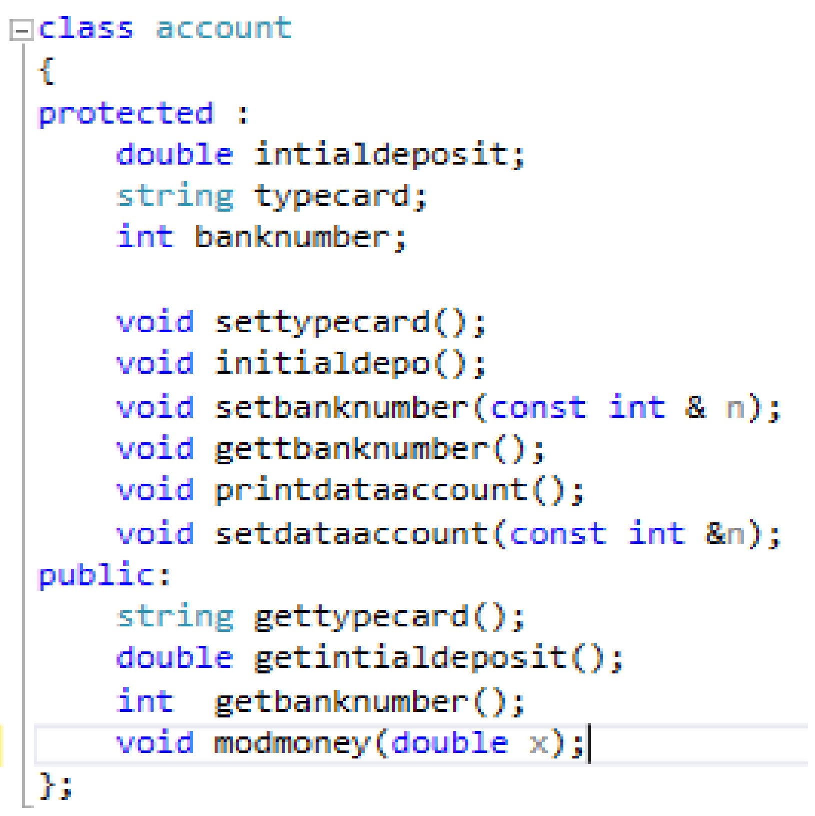
Student class and employer class inherent from person class and account class, because each account should contain a specific account number, name, age , ID number , type account (junior/ adult).

So here is the person class :



```
class person
{
    protected:
     string name;
     int  age;
     int  IDnumber;
     void setname();
     void setage();
     void setidnum();
     void setdataperson();
     void printdataperson();
public:
     string getname();
         int getage();
         int getID();
     virtual double fee()=0;
};
```

And for account:



```
class account
{
protected :
    double intialdeposit;
    string typecard;
    int banknumber;

    void settypecard();
    void initialdepo();
    void setbanknumber(const int & n);
    void gettbanknumber();
    void printdataaccount();
    void setdataaccount(const int &n);
public:
    string gettypecard();
    double getintialdeposit();
    int  getbanknumber();
    void modmoney(double x);
};
```

- Polymorphism
  I use polymorphism to specify the fee for all accounts, it depends on the type of account if it junior (student) so there is no fee(fee=0) but if it’s employer the fee is (300 + (deposit*0.01) FT).

- File management

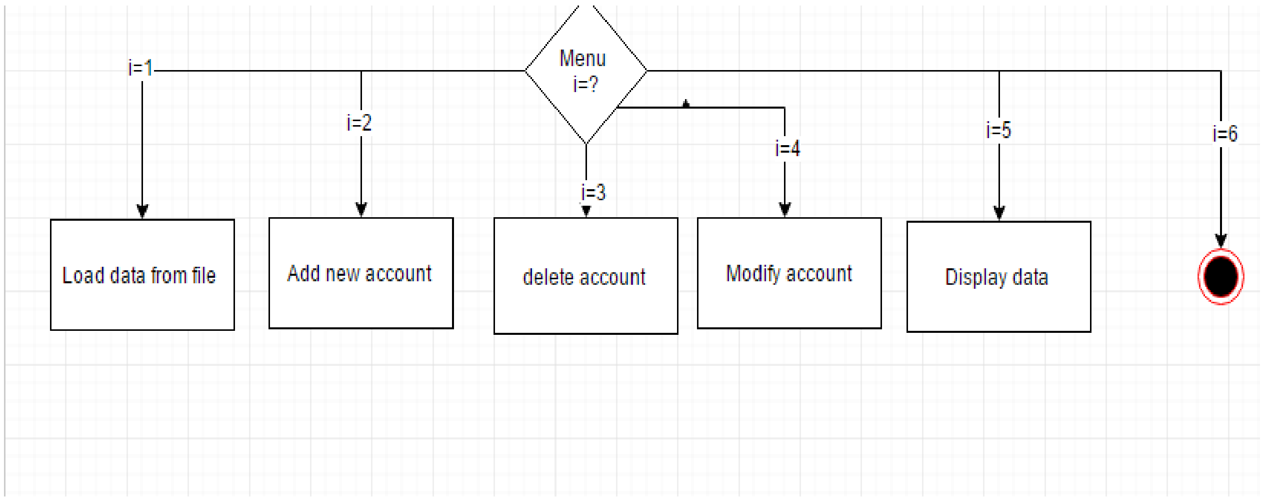
I needed to use files for different parts of my project. I used files for to save all accounts in the file.

I saved all accounts including the data of each account(Name of person, ID number , Bank number …) I used two file: one for employer’s accounts and the second for students account and we can access to modify them.

These are the sources which i got help from (for using files):

https://www.youtube.com/watch?v=gUFJW9Bmu-k&t=3s

Architecture:

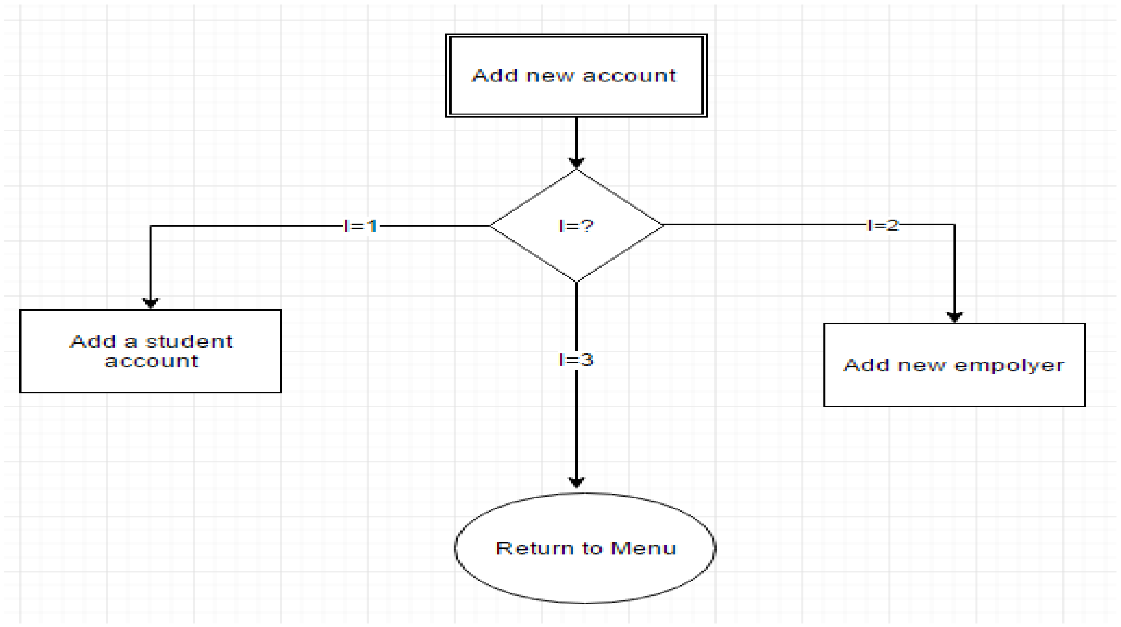


1/ Load data from files:

If the files exists, we load and put all the data from them

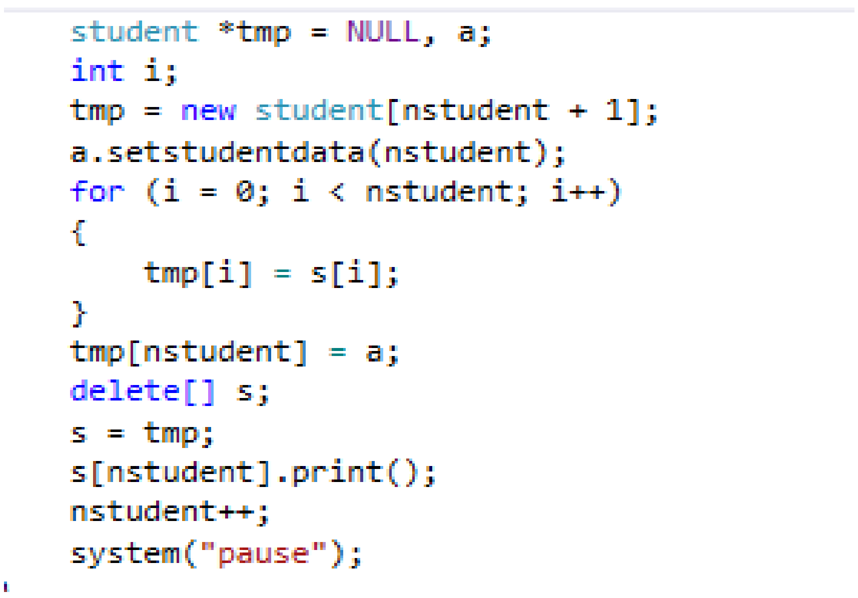
To two dynamical arrays (student[Number of students account], employer[Number of employers account])

2/ Add new account

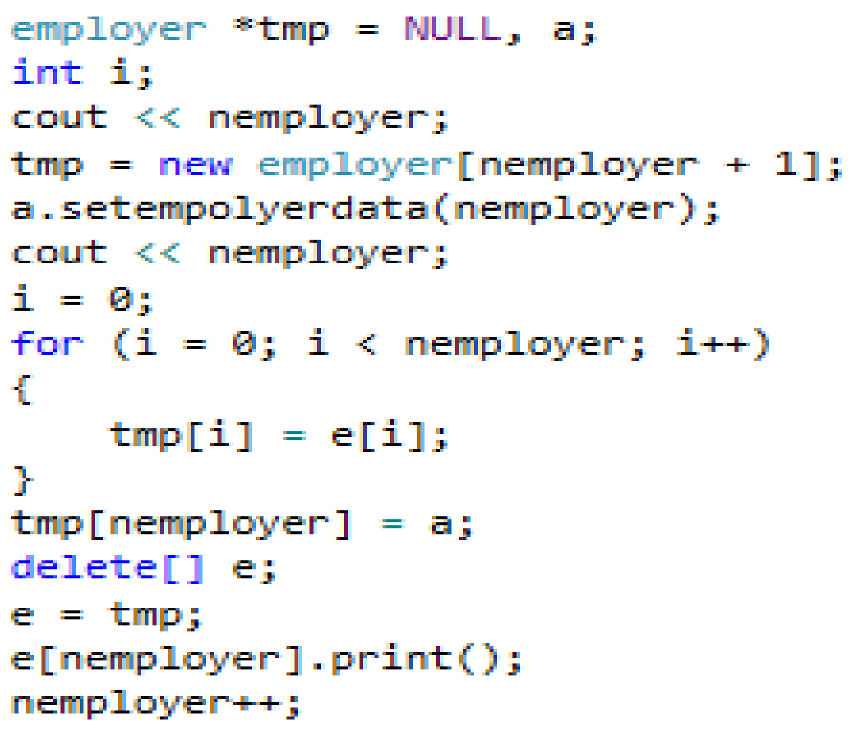


*Add new student → Number of student ++

*Add new employer -> Number of employer ++

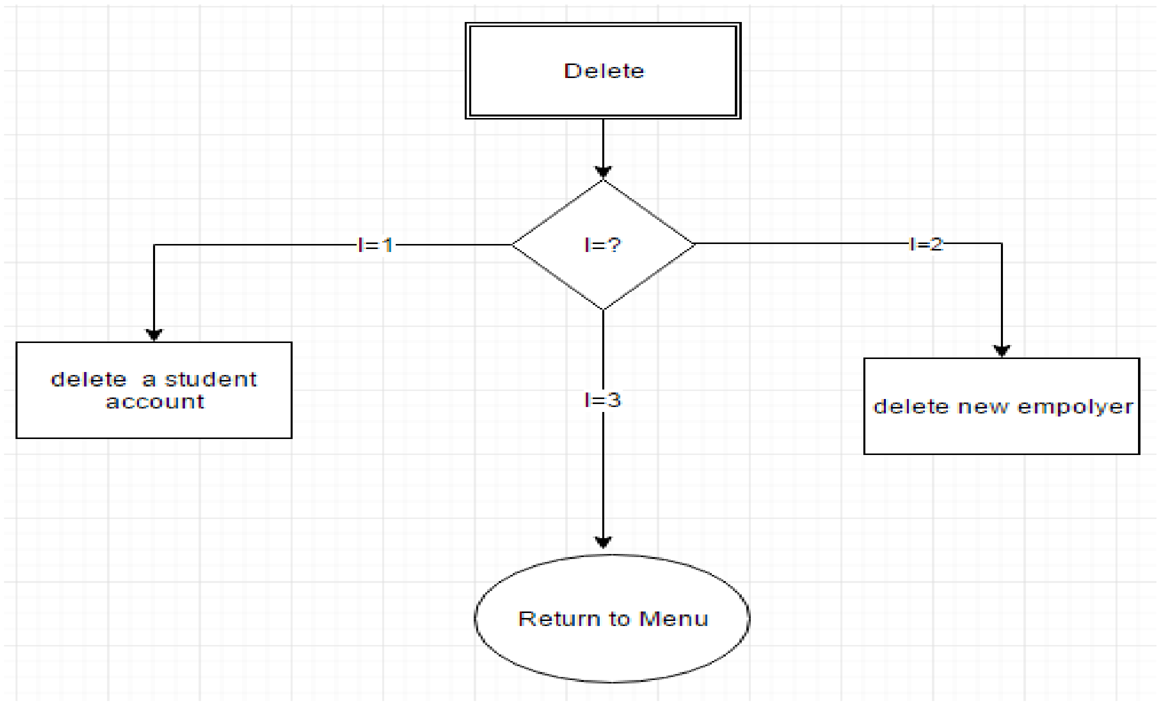


```
student *tmp = NULL, a;
int i;
tmp = new student[nstudent + 1];
a.setstudentdata(nstudent);
for (i = 0; i < nstudent; i++)
{
    tmp[i] = s[i];
}
tmp[nstudent] = a;
delete[] s;
s = tmp;
s[nstudent].print();
nstudent++;
system("pause");
```



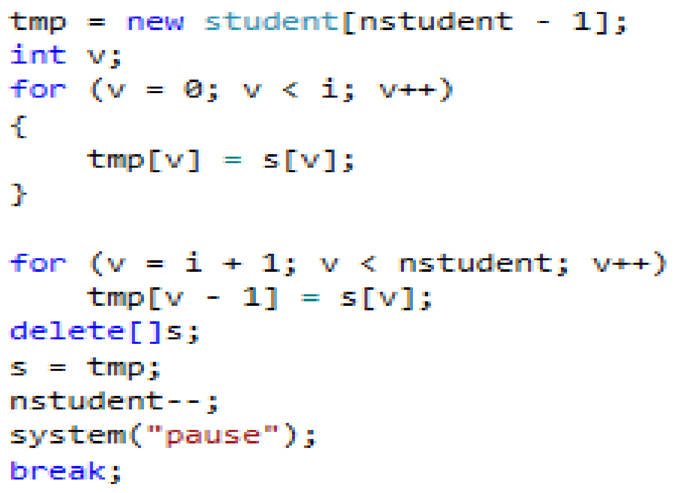
```
employer *tmp = NULL, a;
int i;
cout << nemployer;
tmp = new employer[nemployer + 1];
a.setempolyerdata(nemployer);
cout << nemployer;
i = 0;
for (i = 0; i < nemployer; i++)
{
    tmp[i] = e[i];
}
tmp[nemployer] = a;
delete[] e;
e = tmp;
e[nemployer].print();
nemployer++;
```

3/Delete account :



*delete student account → Number of student --

* delete employer account -> Number of employer –



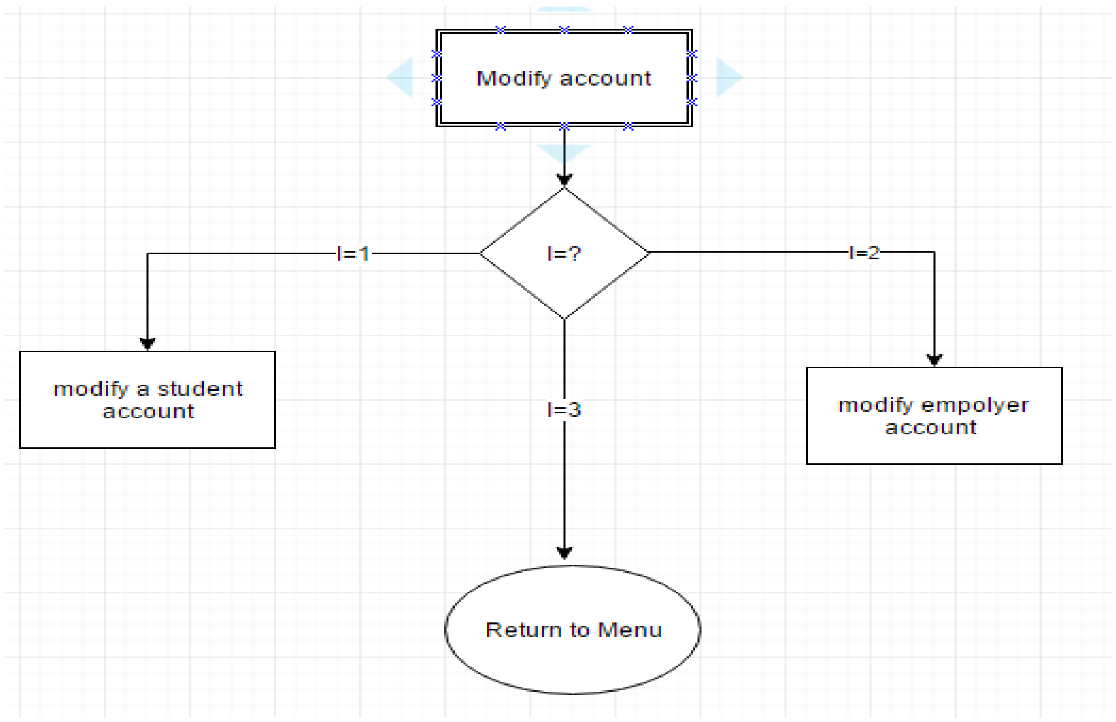
```
tmp = new student[nstudent - 1];
int v;
for (v = 0; v < i; v++)
{
    tmp[v] = s[v];
}

for (v = i + 1; v < nstudent; v++)
    tmp[v - 1] = s[v];
delete[]s;
s = tmp;
nstudent--;
system("pause");
break;
```

```
tmp = new employer[nemployer - 1];
int v;
for (v = 0; v < i; v++)
{
    tmp[v] = e[v];
}

for (v = i + 1; v < nemployer; v++)
    tmp[v - 1] = e[v];
delete[]e;
e = tmp;
nemployer--;
system("pause");
break;
```

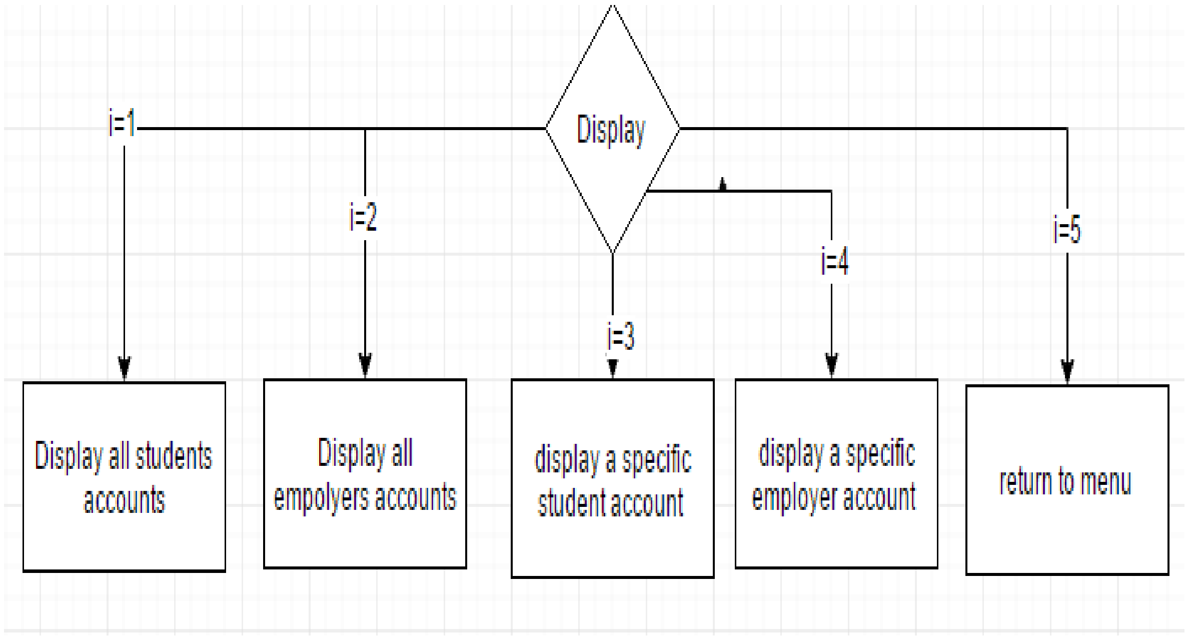
4/Modify account



After choosing which type account we will modify it, we read Bank number from user and display data of this account, and if it's the given account we decide to draw

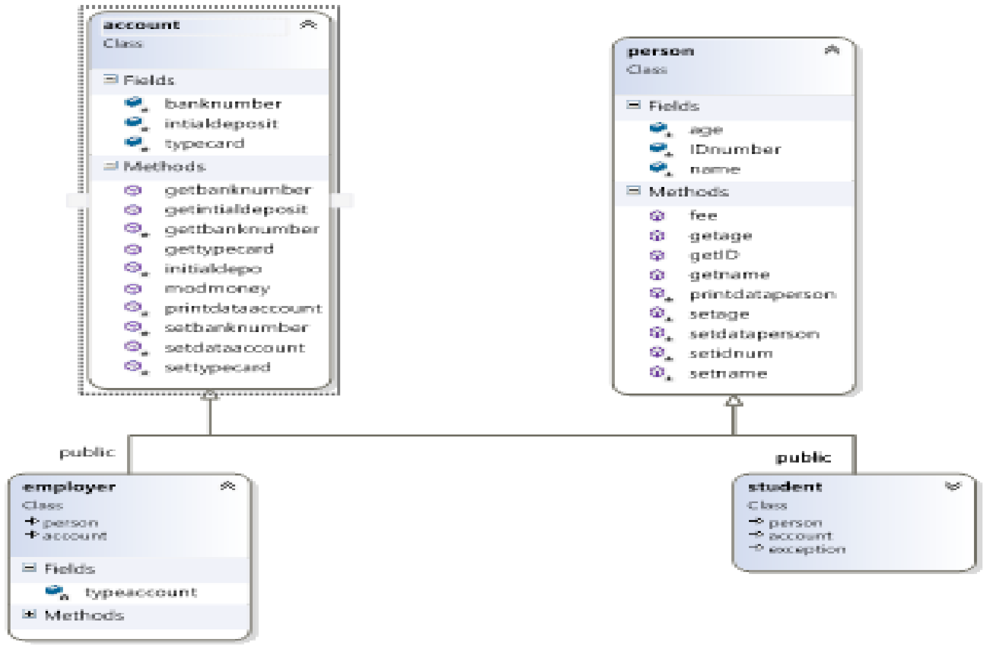
Or deposit and if we want to draw, we check if the account has money or not.

5/Display Data:


Display
i=1
i=2
i=3
i=4
i=5
Display all students accounts
Display all empolyers accounts
display a specific student account
display a specific employer account
return to menu

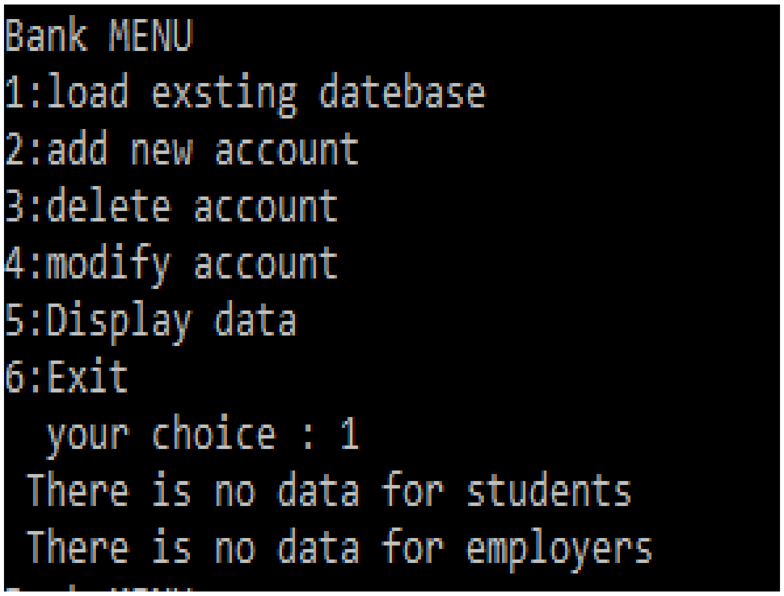
For specific student account and employer account we take the bank number from user and display all data .

Architecture



Employer and student classes are inherited from Account and person classes.

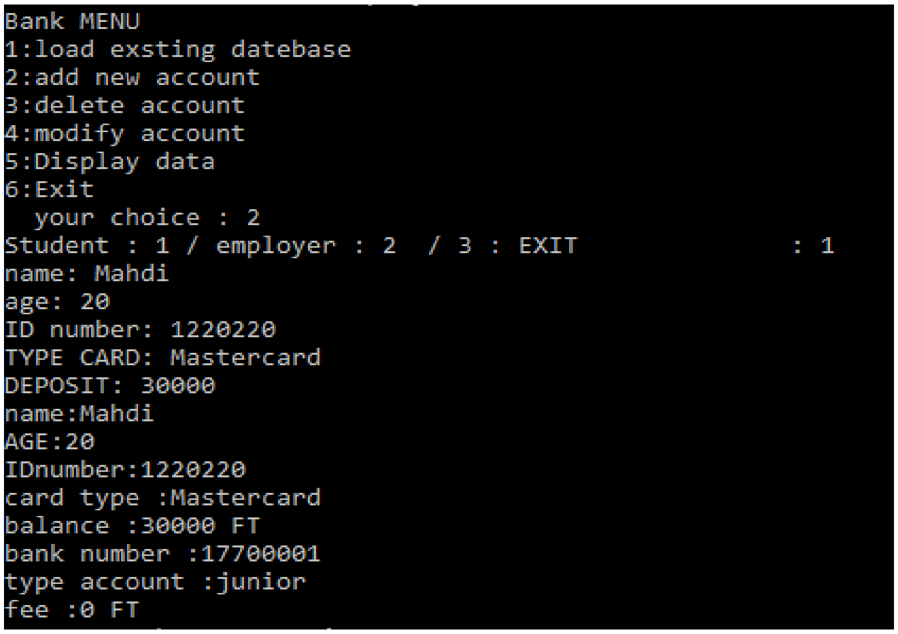
Testing :



```
Bank MENU
1:load exsting datebase
2:add new account
3:delete account
4:modify account
5:Display data
6:Exit
  your choice : 1
 There is no data for students
 There is no data for employers
```

When the result is there no data for student means that when we found the number of students=0 and for the same for employers.

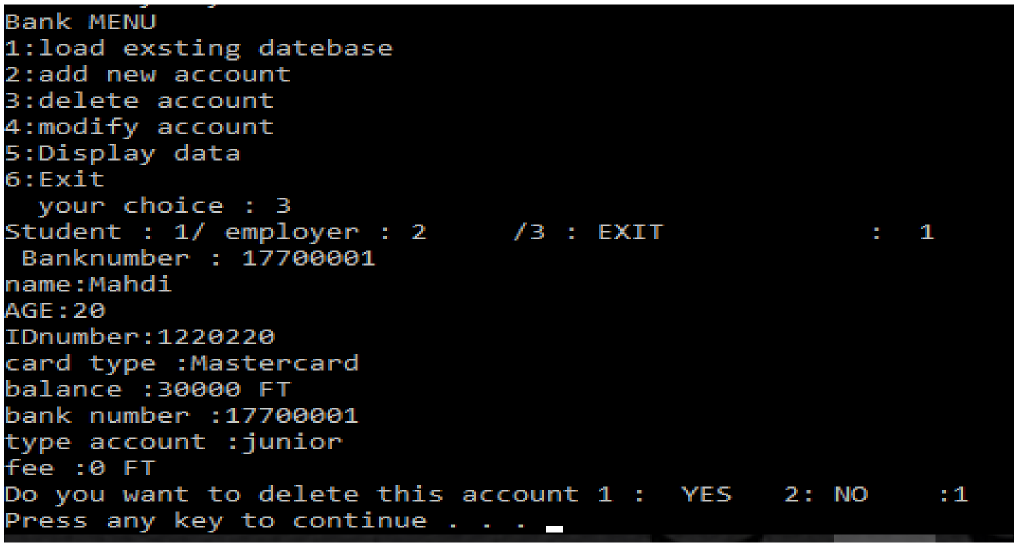
2/add new account



```
Bank MENU
1:load exsting datebase
2:add new account
3:delete account
4:modify account
5:Display data
6:Exit
  your choice : 2
Student : 1 / employer : 2  / 3 : EXIT              : 1
name: Mahdi
age: 20
ID number: 1220220
TYPE CARD: Mastercard
DEPOSIT: 30000
name:Mahdi
AGE:20
IDnumber:1220220
card type :Mastercard
balance :30000 FT
bank number :17700001
type account :junior
fee :0 FT
```

First we choose student or employer and we enter the data.

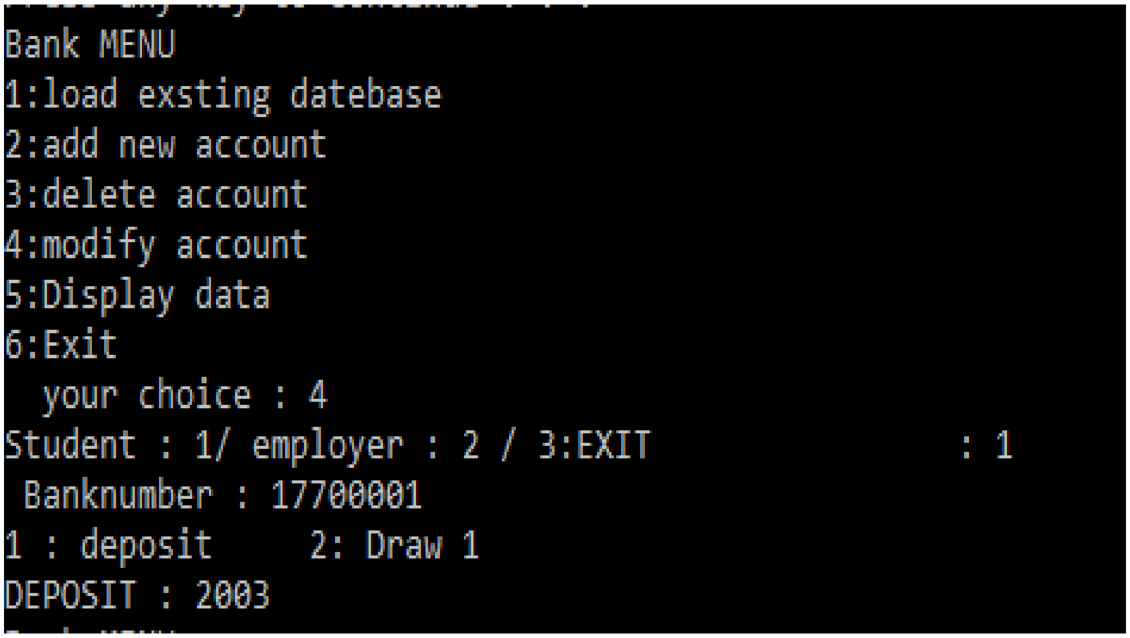
3/delete account:



```
Bank MENU
1:load exsting datebase
2:add new account
3:delete account
4:modify account
5:Display data
6:Exit
  your choice : 3
Student : 1/ employer : 2      /3 : EXIT              :   1
 Banknumber : 17700001
name:Mahdi
AGE:20
IDnumber:1220220
card type :Mastercard
balance :30000 FT
bank number :17700001
type account :junior
fee :0 FT
Do you want to delete this account 1 :   YES    2: NO     :1
Press any key to continue . . .
```

We choose student or employer account after that we enter the Bank number, we display the data of the given account and we decide to delete or not.

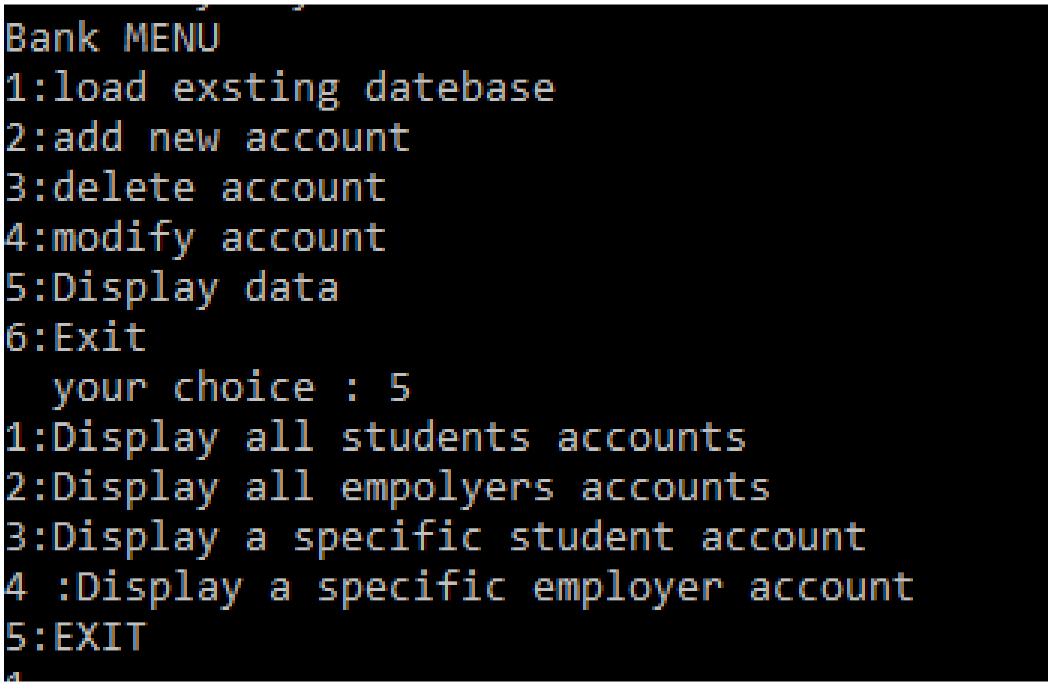
4/modify account:



```
Bank MENU
1:load exsting datebase
2:add new account
3:delete account
4:modify account
5:Display data
6:Exit
  your choice : 4
Student : 1/ employer : 2 / 3:EXIT               : 1
 Banknumber : 17700001
1 : deposit     2: Draw 1
DEPOSIT : 2003
```

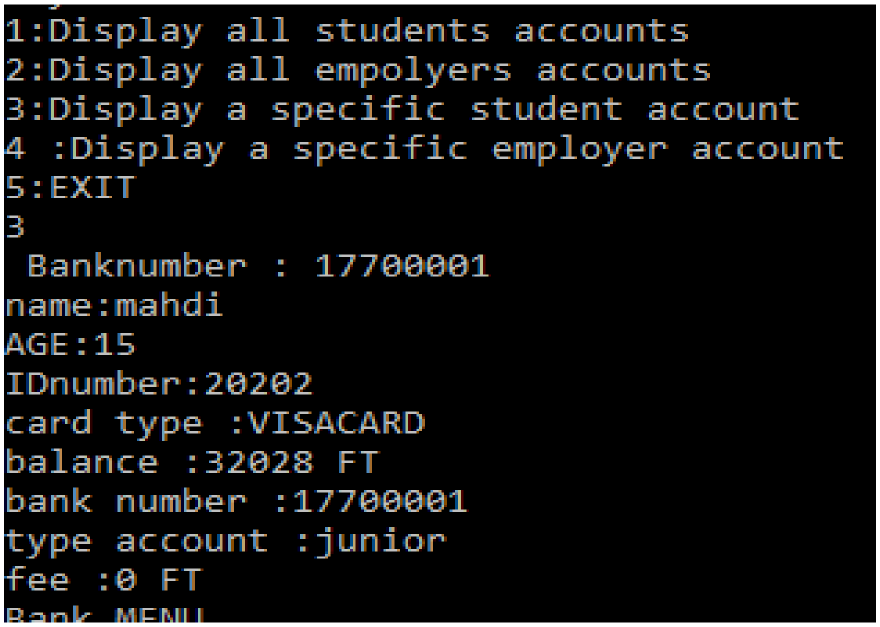
First, we choose student or employer and we enter the bank number and after we choose Deposit or draw and we modify the data of the given account.

5/Display data :



```
Bank MENU
1:load exsting datebase
2:add new account
3:delete account
4:modify account
5:Display data
6:Exit
  your choice : 5
1:Display all students accounts
2:Display all empolyers accounts
3:Display a specific student account
4 :Display a specific employer account
5:EXIT
```

If we choose the 3rd or 4th choice ,we should enter the bank number and display the data :

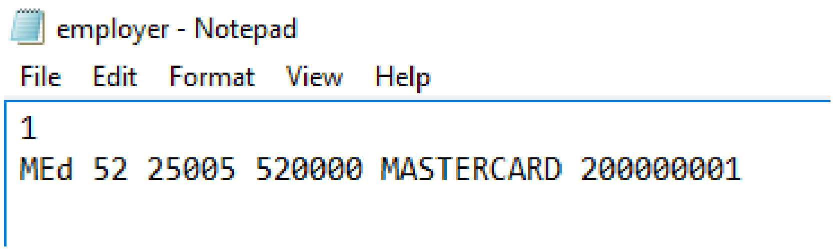


```
1:Display all students accounts
2:Display all empolyers accounts
3:Display a specific student account
4 :Display a specific employer account
5:EXIT
3
 Banknumber : 17700001
name:mahdi
AGE:15
IDnumber:20202
card type :VISACARD
balance :32028 FT
bank number :17700001
type account :junior
fee :0 FT
Bank MENU
```

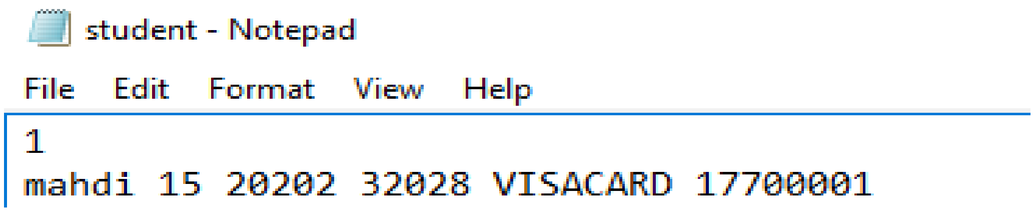
6/ Exit

If we exit from the program, all the data will be saved automatically.

as we see here :



```
1
MEd 52 25005 520000 MASTERCARD 200000001
```

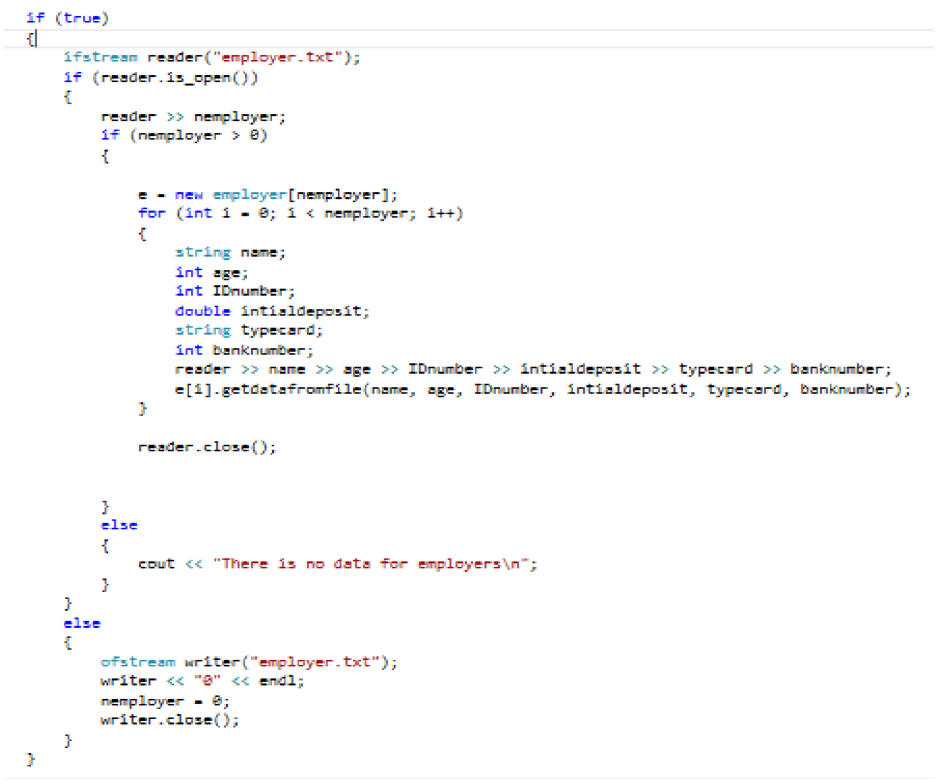


```
1
mahdi 15 20202 32028 VISACARD 17700001
```

For the first line we save there the numbers of accounts.

For the implementation of this part, as I said , I used tutorial to learn how the read from file and write to it.

And this a part from my code:



```
if (true)
{
    ifstream reader("employer.txt");
    if (reader.is_open())
    {
        reader >> nemployer;
        if (nemployer > 0)
        {

            e = new employer[nemployer];
            for (int i = 0; i < nemployer; i++)
            {
                string name;
                int age;
                int IDnumber;
                double intialdeposit;
                string typecard;
                int banknumber;
                reader >> name >> age >> IDnumber >> intialdeposit >> typecard >> banknumber;
                e[i].getdatafromfile(name, age, IDnumber, intialdeposit, typecard, banknumber);
            }

            reader.close();


        }
        else
        {
            cout << "There is no data for employers\n";
        }
    }
    else
    {
        ofstream writer("employer.txt");
        writer << "0" << endl;
        nemployer = 0;
        writer.close();
    }
}
```

If the files exist, we load data from it , first we load Number of account if >0 ➔ we load N accounts, if =0 we display there is no data for …

If there is no files → we create 2 files one for students accounts and one for employers accounts and we put

Nstudent=Nempolyer=0