

Name: Kirolos Sawiris

Neptun code: Y8B5KD

Project Name: Mobile Phone system Management

Documentation outline:

- Introduction to the program
- Used libraries
- C++ used techniques
- Explaining menus and how the program works
- Explaining the Code (Classes, function, etc.)
- Summary

Introduction to the program:

Mobile phone management project it is an object-oriented programming project which has a login interface to deal with multiple user. First user has to register by entering its name, password, phone number, and e-wallet value (digital or electronic wallet). After registering user can login to its account with no credits. user can get credits when he/she exchange some money from the e-wallet. each user has number of credits which can transfer to another user, use it to make a call, or use it to send a message. The program saves all the calls and messages done by users so that each user can see incoming and outgoing calls and messages that he/she has.

Used Libraries:

```
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <string>
#include <string.h>
#include <stdlib.h>
#include <algorithm>
#include <new>
```

C++ used techniques

Object oriented programming

Dynamic memory management

Inheritance (base and derived classes)

Polymorphism: abstract classes and virtual functions.

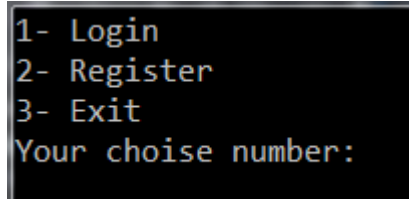
Operator overloading with local and global operations.

File management: saving data to file, loading data from file.

Exception management (try/catch).

Explaining menus and how the program works:

First menu it's the login and registration menu, which enable users to create an account and login to their accounts.



```
1- Login
2- Register
3- Exit
Your choise number:
```

When the user chooses register, he/she has to enter his/her username, password, phone number, and e-wallet value.

```
Enter your Username: user4
Enter your password: 123456
Enter your phonenumber: +3613456
Enter your Ewalletvalue: 100
```

After that the user can login if he entered his/her username and password correctly.

```
Enter your Username: user4
Enter your password: 123456
```

After login there will be many options to the user to choose from like view current information, edit current information transfer credits, calls and message settings, and logout.

```
Welcome user4
1- View your current information
2- Edit your current information
3- Transfare Credit
4- Top up credits
5- Calls and messages sittings
6- Logout
Enter your choice number:
```

Edit your current information menu enable the user to change some of its information such as, password, e-wallet value.

```
1-password
2-E-wallet Value
3-go back
Enter your choice to edit:
```

Calls and messages sittings menu enable the user to make new call, new messages, viewing the calls and messages that he/she made, or go back.

```
1-Make a Call
2-Send a Message
3-View Calls and Messages
4-go back
Choose:
```

Explaining the Code:

The code has four Classer, User class which contain 5 privet attributes username, password, phone number, credit value, and e-wallet value, and publicly it has constructors, some setters and getters, top up credit function which take from the user e-wallet value and exchange it with credits, transfer function which take from the user credit and transfer it to the other users, and finally user edit which can edit the user information.

First topup_credits() member function:

This boolean function asks the user for a value which want to exchange (from e-wallet to credits) if the e-wallet value was smaller than the value that the user enter then the function returns false, else the function subtract this value from the user e-wallet, and add it to the credits value after multiplying it by 10 since 1\$ from e-wallet equal 10 credits.

```
bool User::topup_credits()
{
    system("CLS");
    cout << "\n 1$ give 10 credits, Enter number of dollers that you want to exchange: ";
    int n;
    cin >> n;
    if (EwalletValue < n)
    {
        system("CLS");
        cout << "\nsorry, you don't have that ammount\n";
        return false;
    }
    else
    {
        CreditValue = CreditValue + n * 10;
        EwalletValue = EwalletValue - n;
        return true;
    }
}
```

print_user function:

```
void User::print_User() const
{
    cout << "User name : " << username << endl;
    cout << "password : " << Password << endl;
    cout << "phone number : " << PhoneNumber << endl;
    cout << "Credit Value : " << CreditValue << endl;
    cout << "E wallet Value : " << EwalletValue << endl << endl;
}
```

This void constant function prints the users data to the output screen.

transfer_credits member function:

This function asks the user for number of credits that he/she wants to transfer to another user, then check if the user has these number of credits if not it return false, if true, then it asks him/her to enter the phone number of the user that he/she wants to transfer that number of credits to. Then it searches if this number already exists if so then it takes from the credits he/she has and add it to the other user. If no, then the function returns false.

```
bool User::transfer_credits(User* Userarr, int N)
{
    system("CLS");
    cout << "Enter number of credits that you want to transfer: ";
    int n;
    cin >> n;
    if (CreditValue < n)
    {
        system("CLS");
        cout << "\nsorry, you don't have enough credits\n";
        return false;
    }
    cout << "\nEnter the phone number you want to transfer this ammount to: ";
    string t_Num;
    cin >> t_Num;
    for (int i = 0; i < N; i++)
    {
        if (t_Num == Userarr[i].get_phoneNumber())
        {
            CreditValue = CreditValue - n;
            Userarr[i].CreditValue = Userarr[i].CreditValue + n;
            system("CLS");
            cout << "\ntransaction succeeded\n";
            return true;
        }
    }
    cout << "\nNot found number, Try agein!\n";
    return false;
}
```

reading and writing operators overloading functions:

```
istream& operator>> (istream& is, User& A)
{
    is >> A.username;
    is >> A.Password;
    is >> A.PhoneNumber;
    is >> A.CreditValue;
    is >> A.EwalletValue;
    return is;
}
```

```
ostream& operator<< (ostream& os, User& A)
{
    os << A.username;
    os << endl;
    os << A.Password;
    os << endl;
    os << A.PhoneNumber;
    os << endl;
    os << A.CreditValue;
    os << endl;
    os << A.EwalletValue;
    os << endl;
    return os;
}
```

User_edit member function

This function asks the user if he want to change password or e-wallet if he wants to change the password then the function change the object password and the same with e-wallet value.

```
void User::user_edit()
{
    cout << "\n1-password\n2-E-wallet Value\n3-go back\n Enter your choice to edit: ";
    int n;
    cin >> n;
    string pass = Password;
    int v = EwalletValue;
    switch (n)
    {
    case 1:
        cout << "\nEnter new password: ";
        cin >> pass;
        this->set_password(pass);
        break;
    case 2:
        cout << "\nEnter new E-wallet Value: ";
        cin >> v;
        this->set_EwalletValue(v);
        break;
    default:
        break;
    }
}
```

The code also contain one abstract class which is connection class and two derived class which are Call class and Message Class, Connection class has two protected attributes Called number and Caller number, publicly it has some setters and getters in addition to one pure virtual function which is print function

Call class has two attributes inherited from connection class in addition to one privet attribute which is duration which store the duration of the call, publicly it has constructors and some setters and getters, in addition to get_Call function which get the call from user and is_Call_Valid function to check if the call is valid or not.

get_Call member function simply asks and stores the call data from a user.

```
void Call::get_Call(User& A)
{
    string Called_Num;
    cout << "Enter the number you want to call: ";
    cin >> Called_Num;
    cout << "Enter the Duration (in minute): ";
    int dur;
    cin >> dur;
    CallerNumber = A.get_phoneNumber();
    set_CalledPhoneNumber(Called_Num);
    set_Duration(dur);
}
```

is_Call_Valid member function:

this function gets a user and the user array and the number of elements in the user array, to check if the user can do this call or not, first it check if the user credit value is less that the duration of the call multiplied by 2 (2 is the cost of each minute) if not , if return false , if true it checks if the number that the user want to call exist in the user array or not if yes then it subtract the cost of the call and return true, if the called number doesn't exist it return false.

```
bool Call::is_Call_Valid(User& A, User* Userarr, int NUM)
{
    if (A.get_CreaditVlaue() < Duration * 2)
    {
        cout << "\nyou don't have enough credits!\n";
        return false;
    }
    if (CallerNumber == CalledNumber)
    {
        cout << "\nyou can't call your self!\n";
        return false;
    }
    for (int i = 0; i < NUM; i++)
    {
        if (CalledNumber == Userarr[i].get_phoneNumber())
        {
            int NewValue = A.get_CreaditVlaue() - Duration * 2;
            A.set_CreaditVlaue(NewValue);
            return true;
        }
    }
    cout << "\nNot found number, Try agein!\n";
    return false;
}
```

Message class has two attributes inherited from connection class (called number, and caller number) in addition to one privet attribute which is Message_content which stores the content of the message, publicly has:

get Message member function: which asks and stores the class data.

```
void Message::get_Message(User& A)
{
    string Called_Num;
    string content;
    cout << "Enter the number you want to Message: ";
    cin >> Called_Num;
    cout << "Enter the Content: ";
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    getline(cin, content);
    replace(content.begin(), content.end(), ' ', '+');
    CallerNumber = A.get_phoneNumber();
    CalledNumber = Called_Num;
    Message_content = content;
}
```

is_Message_valid member function:

this function gets a user and the user array and the number of elements in the user array, to check if the user can send the message or not, first it check if the user credit value is less that the message const which is 10 if not , if return false , if true it checks if the number that the user want to send the message exist in the user array or not if yes then it subtract the cost of the message and return true, if the called number doesn't exist it return false.

```
bool Message::is_Message_valid(User& A, User* Userarr, int NUM)
{
    if (A.get_CreaditVlaue() < 10)
    {
        cout << "\nyou don't have enough credits!\n";
        return false;
    }
    if (CalledNumber == CallerNumber)
    {
        cout << "\nyou can't Message your self!\n";
        return false;
    }
    for (int i = 0; i < NUM; i++)
    {
        if (CalledNumber == Userarr[i].get_phoneNumber())
        {
            int NewValue = A.get_CreaditVlaue() - 10;
            A.set_CreaditVlaue(NewValue);
            return true;
        }
    }
    cout << "\nNot found number, Try agein!\n";
    return false;
}
```

print

function:

print is a pure virtual function inherited from connection class and we override it in both derived classes (Call class and message class)

first the override in the call class:

it checks if the user phone number equal to the caller number if yes then the user was the caller so is print the call as outgoing call if the user phone number was equal to the called number then it prints it as coming call.

```
void Call::print(User& A)
{
    if (A.get_phoneNumber() == CallerNumber)
    {
        cout << "Outgoing Call to: " << CalledNumber << " Duration: " << Duration << endl;
    }
    if (A.get_phoneNumber() == CalledNumber)
    {
        cout << "Coming Call from: " << CallerNumber << " Duration: " << Duration << endl;
    }
}
```


Second the override in the Message class:

First it replaces all '+' character with space character as spaces was written in the file as '+' character, then it checks if the user phone number equal to the caller number if yes then the user was the sender so is print the Message as outgoing Message, if the user phone number was equal to the called number then it prints it as coming Message.

```
void Message::print(User& A)
{
    replace(Message_content.begin(), Message_content.end(), '+', ' ');
    if (A.get_phoneNumber() == CallerNumber)
    {
        cout << "Outgoing Message to: " << CalledNumber << " Content: " << Message_content << endl;
    }
    if (A.get_phoneNumber() == CalledNumber)
    {
        cout << "Coming Message from: " << CallerNumber << " Content: " << Message_content << endl;
    }
    replace(Message_content.begin(), Message_content.end(), ' ', '+');
}
```

In the main file which is " MobileMangmentProject.cpp " we have:

Count_element function which count the number of lines in the Calls or messages files since both store 3 lines per object then we count the number of line in the file (calls file or message file only) then it divided it by 3 (number of line per object) to return the number of elements.

```
int Count_element(string fileName)
{
    string line;
    fstream file;
    file.open(fileName.c_str(), ios::in);
    getline(file, line);
    int i;
    for (i = 0; !line.empty(); i++)
    {
        getline(file, line);
    }
    return i / 3;
}
```

User_count_element function it counts the number of lines in the users file and it divided it by 5 since one user objects it saved in 5. After dividing over five we get number of users in the file.

```
int User_count_element()
{
    string line;
    fstream users_file;
    users_file.open("UsersList.txt", ios::in);
    getline(users_file, line);
    int i;
    for (i = 0; !line.empty(); i++)
    {
        getline(users_file, line);
    }
    users_file.close();
    return i / 5;
}
```

Register function:

This function asks the user for a username, password, phone number, and e-wallet value and save it to the users file first it get the users from a file and save to the users array check if the user name already exist in the file if yes then it return false which is mean unsuccessful registration, if yes then it asks for a password then the phone number if the phone number was already exist then it return false if no then it save to the file. Note: the credit value should be zero after the registration.

```
bool Registerf()
{
    int Users_number = User_count_element();
    User* UserNumber = new User[Users_number];
    fstream users_file;
    users_file.open("UsersList.txt", ios::in);
    if (users_file.good())
    {
        for (int i = 0; i < Users_number; i++)
        {
            users_file >> UserNumber[i];
        }
    }
    users_file.close();

    string username, password, phonenumber;
    int creditValue, EwalletValue;

    users_file.open("UsersList.txt", ios::app);
    cout << "Enter your Username: ";
    cin >> username;
    for (int i = 0; i < Users_number; i++)
    {
        if (UserNumber[i].username == username)
        {
            system("CLS");
            cout << "username already exist. Try another one!\n";
            return false;
        }
    }
}
```

```
    cout << "Enter your password: ";
    cin >> password;
    cout << "Enter your phonenumber: ";
    cin >> phonenumber;
    for (int i = 0; i < Users_number; i++)
    {
        if (UserNumber[i].PhoneNumber == phonenumber)
        {
            system("CLS");
            cout << "phone number already exist. Try another one!\n";
            return false;
        }
    }
    cout << "Enter your Ewalletvalue: ";
    cin >> EwalletValue;
    creditValue = 0;

    users_file << username << endl << password << endl << phonenumber << endl;
    users_file << creditValue << endl << EwalletValue << endl;
    users_file.close();
    system("CLS");
    cout << "Registration was sucessfull\n";
    return true;
}
```

In the main function we have:

```
int current_User; // number of the current user int the array
int CallsNum = Count_element("CallsList.txt"); // number of objects in the calls file
int Users_number = User_count_element(); // number of objects in the users file
int MessageNum = Count_element("MessageList.txt"); // number of objects in the Messages file
User* UserNumber = NULL;
Call* Callsarr = NULL;
Message* Messagearr = NULL;
try
{
    UserNumber = new User[Users_number]; // Users Dynamic array
    Callsarr = new Call[CallsNum]; // Calls Dynamic array
    Messagearr = new Message[MessageNum]; // Messages Dynamic array
}
catch (bad_alloc& bad)
{
    cerr << "bad allocation caught: " << bad.what();
}
```

3 dynamically allocated arrays, and we use exception handling to handle bad allocation.

```
users_file.open("UsersList.txt", ios::in);
for (int i = 0; i < Users_number; i++) // reading from the file to users array
{
    users_file >> UserNumber[i];
}
calls_file.open("CallsList.txt", ios::in);
for (int i = 0; i < CallsNum; i++) // reading from the file to calss array
{
    calls_file >> Callsarr[i];
}
Message_file.open("MessageList.txt", ios::in);
for (int i = 0; i < CallsNum; i++) // reading from the file to Message array
{
    Message_file >> Messagearr[i];
}
Message_file.close();
calls_file.close();
users_file.close();
```

Then we load all the files (users file, calls file, message file) to the 3 arrays.

Then we ask the user for a username and looking for it in the array if we found it we check if the password is right if yes then we set login state to true to make the user log in.

```
for (int i = 0; i < Users_number; i++)
{
    if (UserNumber[i].get_username() == username)
    {
        cout << "Enter your password: ";
        cin >> password;
        if (password == UserNumber[i].get_password())
        {
            User_notfound = false;
            system("CLS");
            cout << "Welcome " << username << endl;
            current_User = i;
            LoginState = true;
        }
        else
        {
            User_notfound = false;
            system("CLS");
            cout << "wrong password. Try again!\n";
            LoginState = false;
        }
    }
}
```

```

cout << "1- View your current information\n2- Edit your current information\n3- Transfare Credit\n4- Top up credits";
cout << "\n5- Calls and messages sittings\n6- Logout\nEnter your choice number: ";
cin >> n;
bool ST = true;
switch (n)
{
case 1:
    system("CLS");
    UserNumber[current_User].print_User();
    break;
case 2:
    system("CLS");
    UserNumber[current_User].user_edit();
    break;
case 3:
    system("CLS");
    UserNumber[current_User].transfer_credits(UserNumber, Users_number);
    break;
case 4:
    system("CLS");
    UserNumber[current_User].topup_credits();
    break;
}

```

Then in the menu we use the classes member function which was described above.

```

cout << "1-Make a Call\n2-Send a Message\n3-View Calls and Messages\n4-go back\nChoose: ";
int O;
string Called_Num; // the Called number which we will get from the user to Look for in the array
Call NewCall;
Message NewMessage;
string content; // the contern which we will get from the user to store it in the message array
Call* tempC = NULL; // temp pointer for Calls
Message* tempM = NULL; // temp pointer for Messages
cin >> O;
switch (O)
{
case 1:
    system("CLS");
    NewCall.get_Call(UserNumber[current_User]);
    if (NewCall.is_Call_Valid(UserNumber[current_User], UserNumber, Users_number))
    {
        CallsNum++;
        Call* tempC = new Call[CallsNum];
        for (int i = 0; i < CallsNum-1; i++)
        {
            tempC[i] = Callsarr[i];
        }
        tempC[CallsNum - 1] = NewCall;
        delete[] Callsarr;
        Callsarr = tempC;
        calls_file.open("CallsList.txt", ios::out);
        for (int i = 0; i < CallsNum; i++)
        {
            calls_file << Callsarr[i];
        }
        calls_file.close();
    }
}

```

Here we have another menu which is calls and message setting menu which enable the user to make a new call or message or see the message and call that he/she has. First, we get the call and check if it's valid by the two-member function described above then if it is valid, we increase the number of calls to store new call. After saving the new call in the dynamic array we save it back to the file.

```

system("CLS");
Connectionarr = new Connection * [MessageNum + CallsNum]; // Dynamic memory of the abstract Calss
calls_file.open("CallsList.txt", ios::in);
Message_file.open("MessageList.txt", ios::in);
for (int i = 0; i < CallsNum; i++) // storing calls in it
{
    calls_file >> Callsarr[i];
    Connectionarr[i] = &Callsarr[i];
}
for (int i =CallsNum; i < MessageNum+CallsNum; i++) // storing Messages in it
{
    Message_file >> Messagearr[i-CallsNum];
    Connectionarr[i] = &Messagearr[i-CallsNum];
}
for (int i = 0; i < MessageNum + CallsNum; i++) // printing Calls and Messages
{
    Connectionarr[i]->print(UserNumber[current_User]);
}
calls_file.close();
Message_file.close();

break;

```

Then we create an array of abstract class pointers, to store the pointer of the calls a messages object so that we can print it in one time by the pure virtual function print.

```

users_file.open("UsersList.txt", ios::out); // saving the users after changes in the file
for (int i = 0; i < Users_number; i++)
{
    users_file << UserNumber[i];
}
users_file.close();
cout << "You have successfully logged out! \n"; LoginState = false; break;

```

After that we write the object back to the file.

Summary:

This project has a login interface to handle multiple users each time the program load the objects arrays from the files and use them to do some function like transferring credits making a call or sending a message....etc.