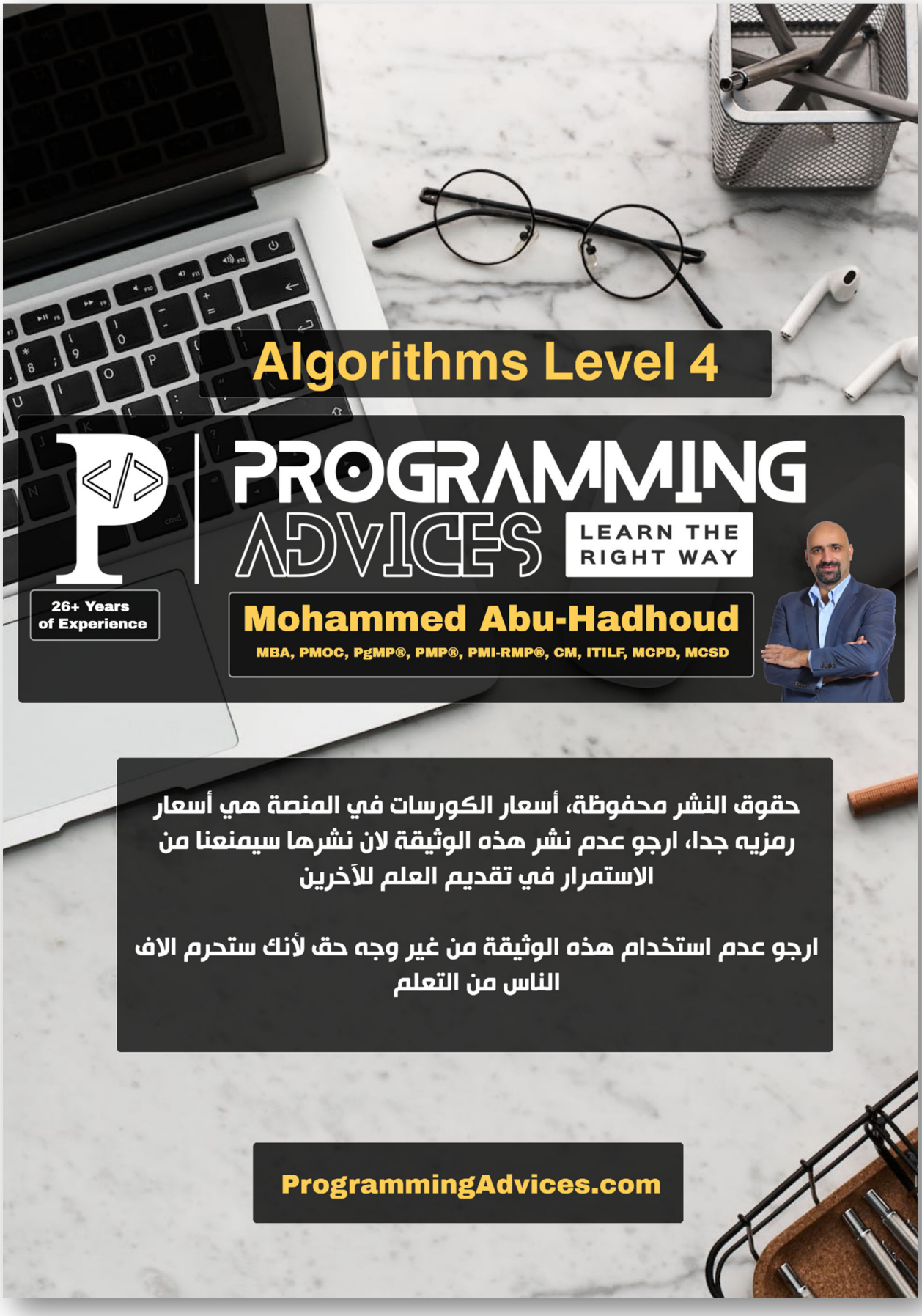
Algorithms Level 4
PROGRAMMING ADVICES
LEARN THE RIGHT WAY
26+ Years of Experience
Mohammed Abu-Hadhoud
MBA, PMOC, PgMP®, PMP®, PMI-RMP®, CM, ITILF, MCPD, MCSD
حقوق النشر محفوظة، أسعار الكورسات في المنصة هي أسعار رمزيه جدا، ارجو عدم نشر هذه الوثيقة لان نشرها سيمنعنا من الاستمرار في تقديم العلم للآخرين
ارجو عدم استخدام هذه الوثيقة من غير وجه حق لأنك ستحرم الاف الناس من التعلم
ProgrammingAdvices.com

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <iomanip>

using namespace std;

struct sClient
{
    string AccountNumber;
    string PinCode;
    string Name;
    string Phone;
    double AccountBalance;
    bool MarkForDelete = false;
};

enum enMainMenueOptions {
    eQucikWithdraw = 1, eNormalWithDraw = 2, eDeposit = 3,
    eCheckBalance = 4, eExit = 5
};

const string ClientsFileName = "Clients.txt";
sClient CurrentClient;

void ShowMainMenue();
void Login();
void ShowQuickWithdrawScreen();
void ShowNormalWithDrawScreen();
void ShowDepositScreen();
```

```
vector<string> SplitString(string S1, string Delim)
{

    vector<string> vString;

    short pos = 0;
    string sWord; // define a string variable

    // use find() function to get the position of the delimiters
    while ((pos = S1.find(Delim)) != std::string::npos)
    {
        sWord = S1.substr(0, pos); // store the word
        if (sWord != "")
        {
            vString.push_back(sWord);
        }

        S1.erase(0, pos + Delim.length());  /* erase() until
positon and move to next word. */
    }

    if (S1 != "")
    {
        vString.push_back(S1); // it adds last word of the string.
    }

    return vString;

}

sClient ConvertLinetoRecord(string Line, string Seperator =
"#//#")
{
    sClient Client;
    vector<string> vClientData;
    vClientData = SplitString(Line, Seperator);

    Client.AccountNumber = vClientData[0];
    Client.PinCode = vClientData[1];
    Client.Name = vClientData[2];
    Client.Phone = vClientData[3];
    Client.AccountBalance = stod(vClientData[4]);//cast string to
double

    return Client;
}
```

```
string ConvertRecordToLine(sClient Client, string Seperator =
"#//#")
{

    string stClientRecord = "";

    stClientRecord += Client.AccountNumber + Seperator;
    stClientRecord += Client.PinCode + Seperator;
    stClientRecord += Client.Name + Seperator;
    stClientRecord += Client.Phone + Seperator;
    stClientRecord += to_string(Client.AccountBalance);

    return stClientRecord;

}

vector <sClient> LoadCleintsDataFromFile(string FileName)
{

    vector <sClient> vClients;

    fstream MyFile;
    MyFile.open(FileName, ios::in);//read Mode

    if (MyFile.is_open())
    {
        string Line;
        sClient Client;

        while (getline(MyFile, Line))
        {

            Client = ConvertLinetoRecord(Line);

            vClients.push_back(Client);
        }

        MyFile.close();
    }

    return vClients;
}
```

```
bool FindClientByAccountNumberAndPinCode(string AccountNumber,
string PinCode, sClient& Client)
{

    vector <sClient> vClients =
LoadCleintsDataFromFile(ClientsFileName);

    for (sClient C : vClients)
    {

        if (C.AccountNumber == AccountNumber && C.PinCode ==
PinCode)
        {
            Client = C;
            return true;
        }
    }
    return false;

}

vector <sClient> SaveCleintsDataToFile(string FileName, vector
<sClient> vClients)
{
    fstream MyFile;
    MyFile.open(FileName, ios::out);//overwrite

    string DataLine;

    if (MyFile.is_open())
    {
        for (sClient C : vClients)
       {
           if (C.MarkForDelete == false)
            {
                DataLine = ConvertRecordToLine(C);
                MyFile << DataLine << endl;

            }

        }
        MyFile.close();
    }

    return vClients;

}
```

```
bool DepositBalanceToClientByAccountNumber(string AccountNumber,
double Amount, vector <sClient>& vClients)
{
    char Answer = 'n';

    cout << "\n\nAre you sure you want perfrom this transaction?
y/n ? ";
    cin >> Answer;
    if (Answer == 'y' || Answer == 'Y')
    {

        for (sClient& C : vClients)
        {
            if (C.AccountNumber == AccountNumber)
            {
                C.AccountBalance += Amount;
                SaveCleintsDataToFile(ClientsFileName, vClients);
                cout << "\n\nDone Successfully. New balance is: "
<< C.AccountBalance;

                return true;
            }

        }


        return false;
    }

}

short ReadQuickWithdrawOption()
{
    short Choice = 0;
    while (Choice < 1 || Choice>9)
    {
        cout << "\nChoose what to do from [1] to [9] ? ";
        cin >> Choice;
    }

    return Choice;
}
```

```
short getQuickWithDrawAmount(short QuickWithDrawOption)
{
    switch (QuickWithDrawOption)
    {
    case 1:
        return  20;
    case 2:
        return 50;
    case 3:
        return 100;
    case 4:
        return 200;
    case 5:
        return 400;
    case 6:
        return 600;
    case 7:
        return 800;
    case 8:
        return 1000;
    default:
        return 0;

    }

}
```

```
void PerfromQuickWithdrawOption(short QuickWithDrawOption)
{
    if (QuickWithDrawOption == 9)//exit
        return;

    short WithDrawBalance =
getQuickWithDrawAmount(QuickWithDrawOption);

    if (WithDrawBalance > CurrentClient.AccountBalance)
    {
        cout << "\nThe amount exceeds your balance, make another
choice.\n";
        cout << "Press Anykey to continue...";
        system("pause>0");
        ShowQuickWithdrawScreen();
        return;
    }

    vector <sClient> vClients =
LoadCleintsDataFromFile(ClientsFileName);

DepositBalanceToClientByAccountNumber(CurrentClient.AccountNumber,
WithDrawBalance * -1, vClients);
    CurrentClient.AccountBalance -= WithDrawBalance;


}

double ReadDepositAmount()
{
    double Amount;
    cout << "\nEnter a positive Deposit Amount? ";

    cin >> Amount;
    while (Amount <= 0)
    {
        cout << "\nEnter a positive Deposit Amount? ";
        cin >> Amount;
    }
    return Amount;
}
```

```
void PerfromDepositOption()
{

    double DepositAmount = ReadDepositAmount();

    vector <sClient> vClients =
LoadCleintsDataFromFile(ClientsFileName);

DepositBalanceToClientByAccountNumber(CurrentClient.AccountNumber,
DepositAmount, vClients);
    CurrentClient.AccountBalance += DepositAmount;
}

void ShowDepositScreen()
{
    system("cls");
    cout << "=========================================\n";
    cout << "\t\tDeposit Screen\n";
    cout << "=========================================\n";
    PerfromDepositOption();

}

void ShowCheckBalanceScreen()
{
    system("cls");
    cout << "=========================================\n";
    cout << "\t\tCheck Balance Screen\n";
    cout << "=========================================\n";
    cout << "Your Balance is " << CurrentClient.AccountBalance <<
"\n";
}

int ReadWithDrawAmont()
{
    int Amount;
    cout << "\nEnter an amount multiple of 5's ? ";

    cin >> Amount;

    while (Amount % 5 != 0)
    {
        cout << "\nEnter an amount multiple of 5's ? ";
        cin >> Amount;
    }
    return Amount;
}
```

```
void PerfromNormalWithdrawOption()
{
    int WithDrawBalance = ReadWithDrawAmont();

    if (WithDrawBalance > CurrentClient.AccountBalance)
    {
        cout << "\nThe amount exceeds your balance, make another
choice.\n";
        cout << "Press Anykey to continue...";
        system("pause>0");
        ShowNormalWithDrawScreen();
        return;
    }

    vector <sClient> vClients =
LoadCleintsDataFromFile(ClientsFileName);

DepositBalanceToClientByAccountNumber(CurrentClient.AccountNumber,
WithDrawBalance * -1, vClients);
    CurrentClient.AccountBalance -= WithDrawBalance;
}

void ShowNormalWithDrawScreen()
{
    system("cls");
    cout << "===========================================\n";
    cout << "\t\tNormal Withdraw Screen\n";
    cout << "===========================================\n";
    PerfromNormalWithdrawOption();
}

void ShowQuickWithdrawScreen()
{
    system("cls");
    cout << "===========================================\n";
    cout << "\t\tQucik Withdraw\n";
    cout << "===========================================\n";
    cout << "\t[1] 20\t\t[2] 50\n";
    cout << "\t[3] 100\t\t[4] 200\n";
    cout << "\t[5] 400\t\t[6] 600\n";
    cout << "\t[7] 800\t\t[8] 1000\n";
    cout << "\t[9] Exit\n";
    cout << "===========================================\n";
    cout << "Your Balance is " << CurrentClient.AccountBalance;

    PerfromQuickWithdrawOption(ReadQuickWithdrawOption());
}
```

```
void GoBackToMainMenue()
{
    cout << "\n\nPress any key to go back to Main Menue...";
    system("pause>0");
    ShowMainMenue();
}

short ReadMainMenueOption()
{
    cout << "Choose what do you want to do? [1 to 5]? ";
    short Choice = 0;
    cin >> Choice;

    return Choice;
}

void PerfromMainMenueOption(enMainMenueOptions MainMenueOption)
{
    switch (MainMenueOption)
    {
    case enMainMenueOptions::eQucikWithdraw:
    {
        system("cls");
        ShowQuickWithdrawScreen();
        GoBackToMainMenue();
        break;
    }
    case enMainMenueOptions::eNormalWithDraw:
        system("cls");
        ShowNormalWithDrawScreen();
        GoBackToMainMenue();
        break;

    case enMainMenueOptions::eDeposit:
        system("cls");
        ShowDepositScreen();
        GoBackToMainMenue();
        break;

    case enMainMenueOptions::eCheckBalance:
        system("cls");
        ShowCheckBalanceScreen();
        GoBackToMainMenue();
        break;
```

```
    case enMainMenueOptions::eExit:
         system("cls");
         Login();

         break;
     }

}

void ShowMainMenue()
{
     system("cls");
     cout << "===========================================\n";
     cout << "\t\tATM Main Menue Screen\n";
     cout << "===========================================\n";
     cout << "\t[1] Quick Withdraw.\n";
     cout << "\t[2] Normal Withdraw.\n";
     cout << "\t[3] Deposit\n";
     cout << "\t[4] Check Balance.\n";
     cout << "\t[5] Logout.\n";
     cout << "===========================================\n";

PerfromMainMenueOption((enMainMenueOptions)ReadMainMenueOption());
}

bool  LoadClientInfo(string AccountNumber, string PinCode)
{

     if (FindClientByAccountNumberAndPinCode(AccountNumber,
PinCode, CurrentClient))
         return true;
     else
         return false;
}
```

```
void Login()
{
    bool LoginFaild = false;
    string AccountNumber, PinCode;
    do
    {
        system("cls");
        cout << "\n-----------------------------------\n";
        cout << "\tLogin Screen";
        cout << "\n-----------------------------------\n";

        if (LoginFaild)
        {
            cout << "Invlaid Account Number/PinCode!\n";
        }

        cout << "Enter Account Number? ";
        cin >> AccountNumber;

        cout << "Enter Pin? ";
        cin >> PinCode;
        LoginFaild = !LoadClientInfo(AccountNumber, PinCode);

    } while (LoginFaild);

    ShowMainMenue();

}

int main()

{
    Login();

    system("pause>0");
    return 0;
}
```