

CHAPTER 1

INTRODUCTION

1.1 GENERAL OVERVIEW

Transaction management defines how the work can be managed using transactions. Transaction is the logical unit of work that contains two or more C++ file structure statements. A Transaction Processing System (TPS) is a type of information system that collects, stores, modifies and retrieves the data transactions of an enterprise.

An automated teller machine (ATM) is an electronic telecommunications device that enables customers of financial institutions to perform financial transactions, such as cash withdrawals, deposits, transfer funds, or obtaining account information, at any time and without the need for direct interaction with bank staff. ATM is a system whereby each transaction is processed immediately, without the delay of accumulating transactions into the batch. It is kind of online transaction processing. Real time processing is where all details of the transactions are recorded and changed at the time as it occurs.

1.2 PROBLEM DEFINITION

The code is written in C++ language. Visual studio is used to compile the code. The code carries out all the functions that all standard ATM machines do. You can check amount present in your account, withdraw balance and deposit amount. Each card holder is assigned a pin code for their ATM cards.

When you run the compiled code it displays two buttons, ADD NEW ACCOUNT and NEW TRANSACTION. On clicking the former, the user can open new account by adding details like account no, name, account type, setting pin and initial account balance. On clicking the latter, lets you login with your new account no and PIN. There you can view your account information and check balance as well as can withdraw and deposit money.

1.3 OBJECTIVES

An ATM increases sales because it allows your customers to access all of their available cash from bankcards, credit cards, etc., from anywhere in the world. Besides the rebates earned for transactions on the ATM, the big profit is from the additional sales from the thousands of extra dollars available in your facility. An ATM transaction is guaranteed and eliminates charge backs, disputes, credit card fees and bad checks. An ATM provides more security for you, your employees and customers by providing less risk of robbery and employee theft. You reduce significantly or eliminate the credit card fees you are currently paying by directing your customers to the ATM. You will have customers stopping at your place of business because the competition cannot accommodate their needs.

40% of regular ATM users go to the ATM an average of 10 times per month, each withdrawal is on average \$75.00. An ATM saves employees time and customer embarrassment. When you provide new and unique services for your patrons, your image is improved. Your potential customers will no longer have to stop at your competitors to get cash (and spend it there). With an ATM, your customer makes electronic deposits directly into your bank account, which saves you both time and work. ATM users will spend 20% more than a non – ATM customer.

Store owners are seeing 20% to 40% of the cash dispensed stay in their stores. Nightclub owners report that 70% to 80% of the cash dispensed stay in their clubs. 65% to 90% of all ATM transactions are cash withdrawals, with non-banking ATM's being 80% to 95% cash withdrawals. 7-11 stores report that over half of all customers using ATM's transact a purchase and volumes continue to grow.

CHAPTER 2

HARDWARE AND SOFTWARE REQUIREMENTS

2.1 HARDWARE REQUIREMENTS

Processor	: Pentium IV
RAM	: 1GB RAM
Hard Disk	: 40 GB
Monitor	: 1024 * 768 Resolution Color Monitor Pentium IV

2.2 SOFTWARE REQUIREMENTS

1. OS : Windows XP,7,8,10
 2. Editor: Microsoft Visual Studio C++
- The hardware requirements specified are the hardware components/capacity of the system in which the application is developed and deployed.
 - The above software requirements are the necessary software's requirement to develop the application and the run the application.
 - Microsoft Visual Studio is used to develop the application on windows platform.
 - The project is developed in C++ language with concepts of File handling.

CHAPTER 3

SYSTEM DESIGN

3.1 CONTEXT-FLOW DIAGRAM

The context diagram is used to establish the context and boundaries of the system to be modeled which things are inside and outside of the system being modeled, and what is the relationship of the system with these external entities.

A context diagram, sometimes called a level 0 data-flow diagram, is drawn in order to define and clarify the boundaries of the software system. It identifies the flows of information between the system and external entities. The entire software system is shown as a single process.

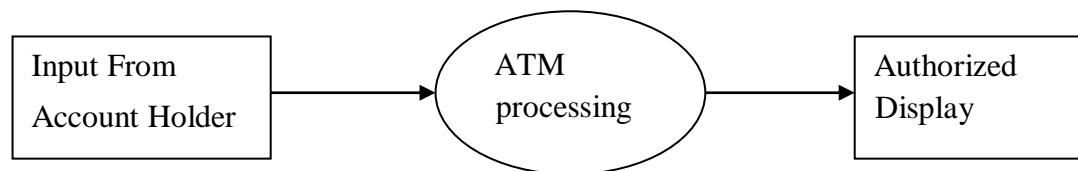


Figure 3.1: Context flow Diagram

3.2 SYSTEM DESIGN

System design is a solution, a “HOW TO” approach to the creation of a new system. It translates system requirements into ways by which they can be made operational. It is a translational from a user oriented document to a document oriented programmers. For that, it provides the understanding and procedural details necessary for the implementation.

Here we use Flowchart to supplement the working of the new system. The system thus made should be reliable, durable and above all should have least possible maintenance costs.

It should overcome all the drawbacks and should meet the user requirements.

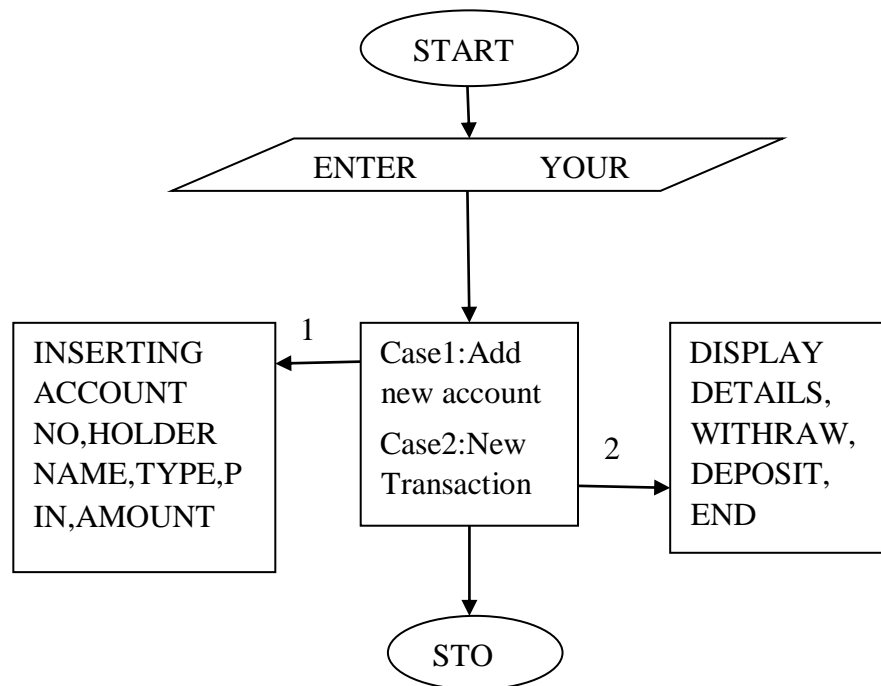


Figure 3.2: System design for ATM Transaction System

The user can perform five operations as you can see in the Figure 3.2. The user can do the following:

- Insert/open new account
- search an existing account
- display that existing account
- withdraw money
- deposit money

3.3 DATA-FLOW DIAGRAM

A data-flow diagram (DFD) is a way of representing a flow of a data of a process or a system. The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow, there are no decision rules and no loops. Specific operations based on the data can be represented by a flowchart. Data flow diagrams are used to graphically represent the flow of data in an information system. DFD describes the processes that are involved in a system to transfer data from the input to the file storage and reports generation.

3.3.1 DATA-FLOW DIAGRAM FOR INSERTION

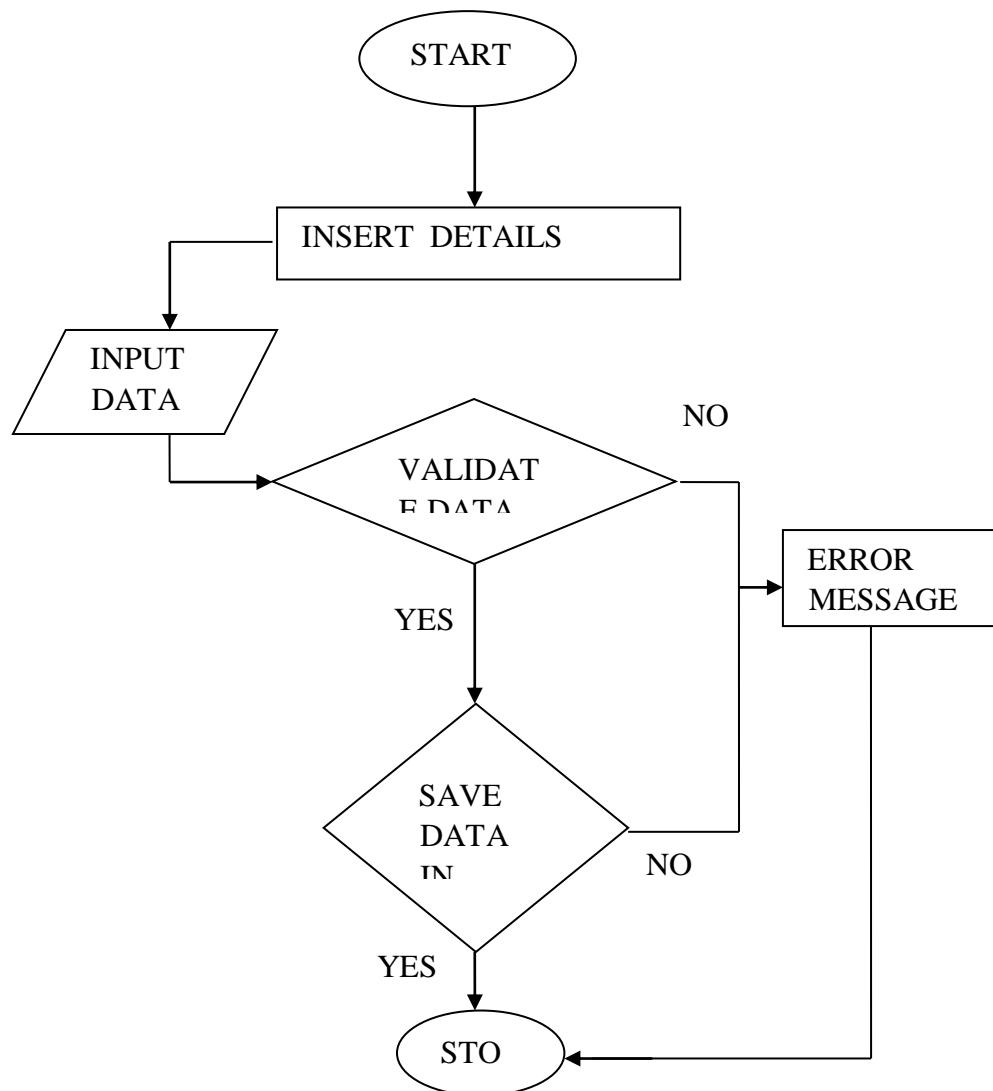


Figure3.3.1: Data-flow diagram for insertion

In the ATM transaction system, (Figure 3.3.1) user can add and insert the account details like account no, account holder's name, account type, ATM PIN, initial amount. Then the data is validated and checked for errors, if there are no errors then the data is saved to the file. The data that is saved in the file can be accessed by the user later.

3.3.2 DATA-FLOW DIAGRAM FOR SEARCHING

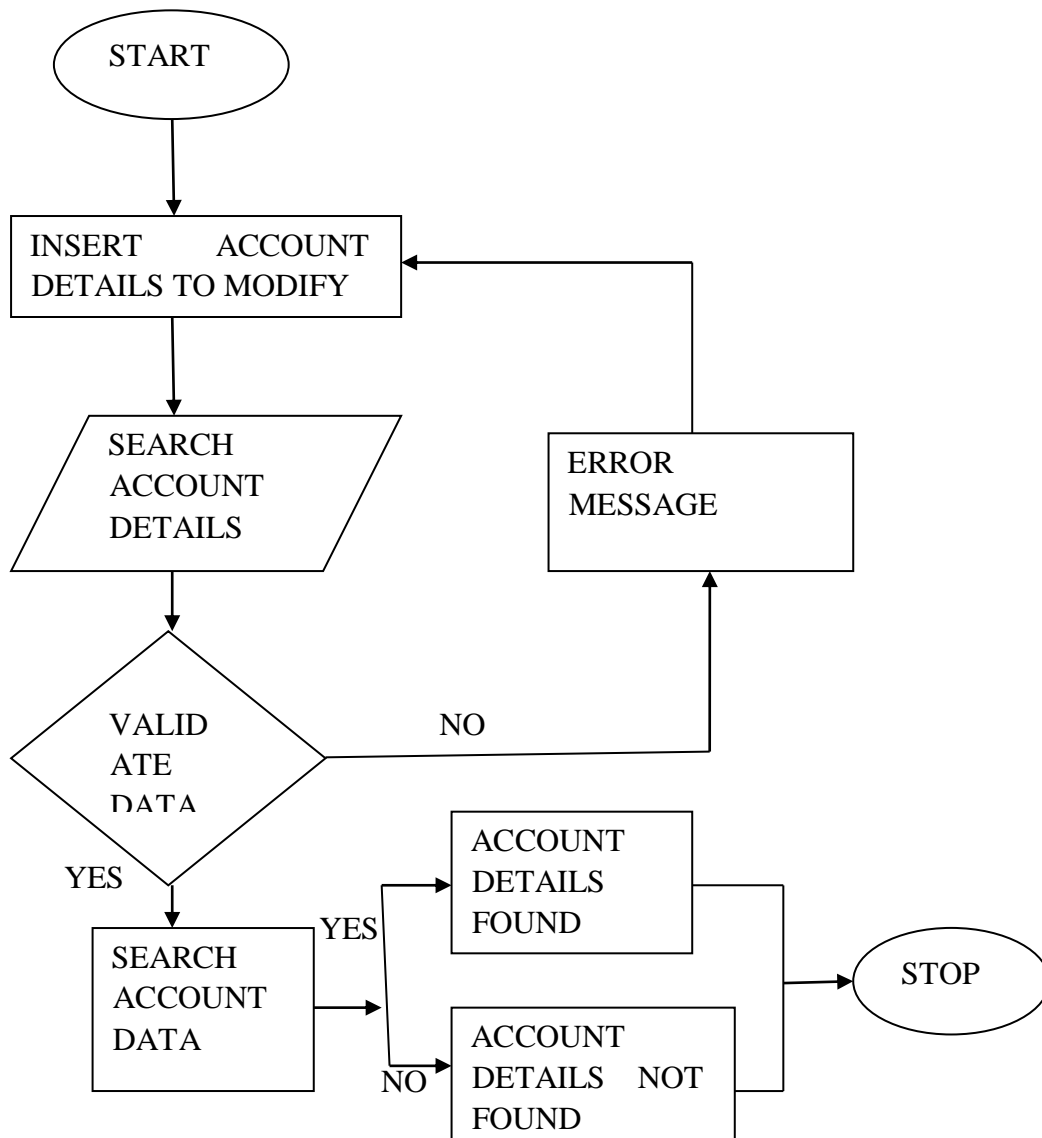


Figure 3.3.2: Data-flow diagram for searching during LOGIN

In the ATM transaction system, (Figure 3.3.2) the user can login to his account by entering valid account no and associated PIN. Then, the entered account no and PIN are searched and compared with each file in the record. If it matches with existing account, it displays co

3.3.3 DATA-FLOW DIAGRAM FOR MODIFICATION

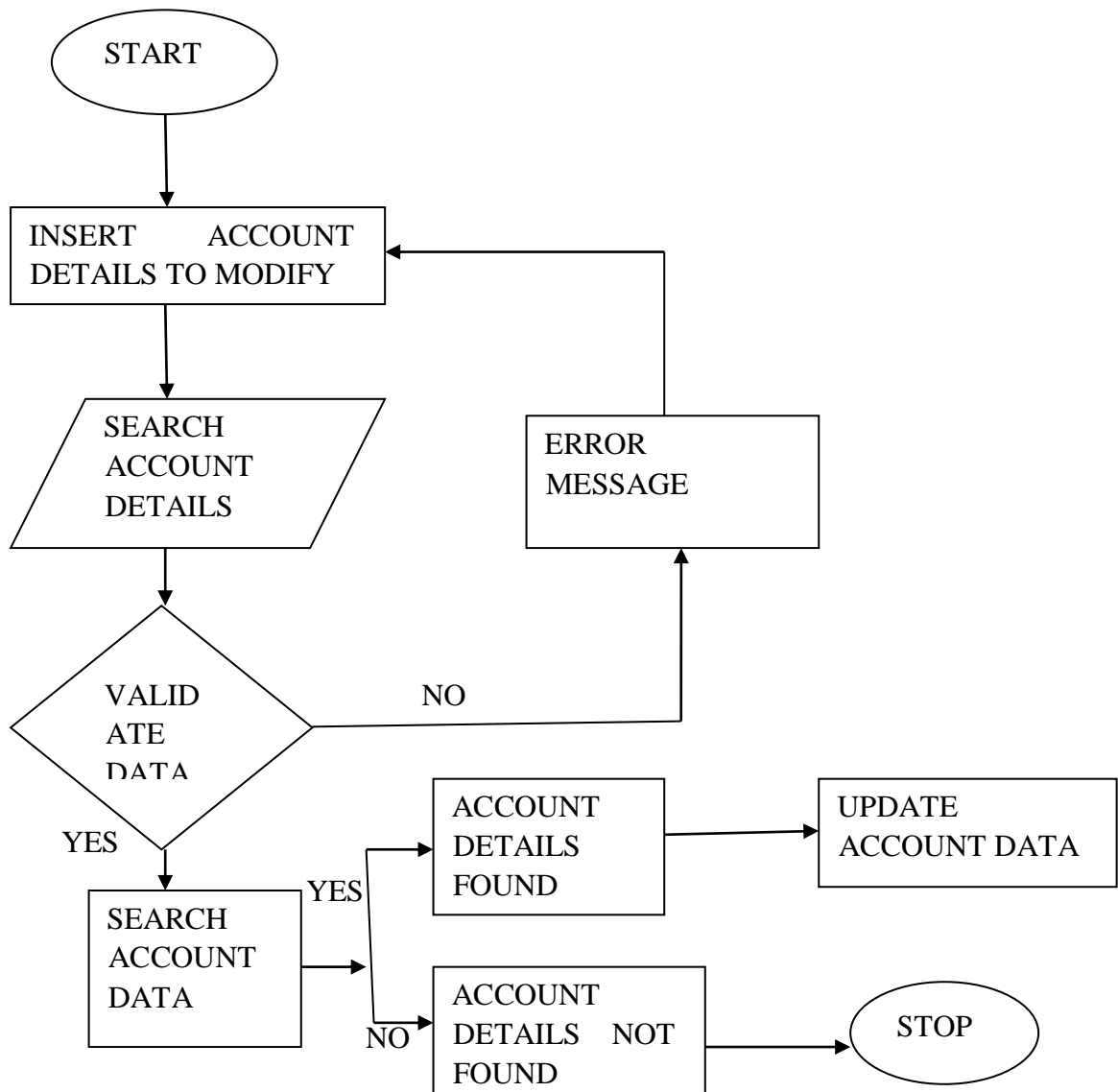


Fig 3.3.3 : Data-flow diagram for modification
(Withdraw, Deposit)

On clicking withdraw and deposit money, the account holder can update his account balance by withdrawing and depositing money.

3.3.4 DATA-FLOW DIAGRAM FOR DISPLAY

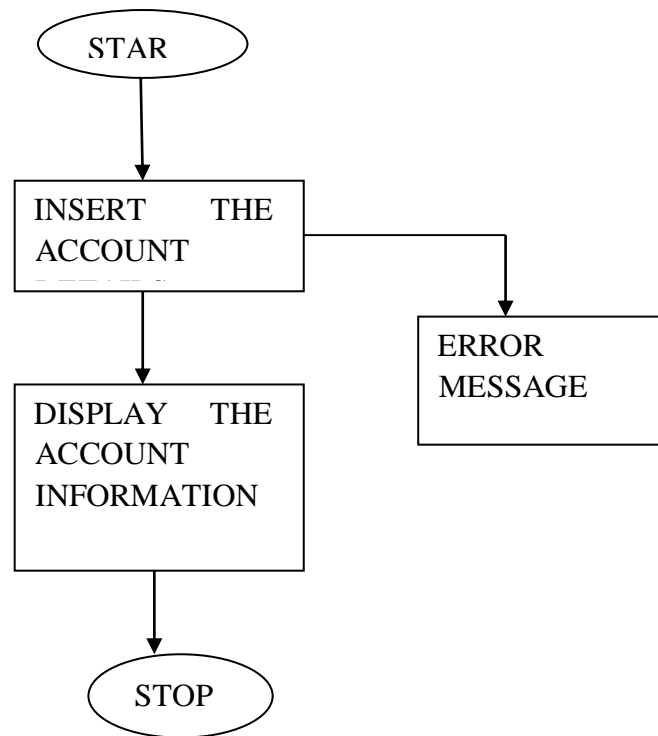


Figure3.3.4:Data-flow diagram for display

In the ATM transaction system, (Figure 3.3.4) the account holder can view all the details of the account.

CHAPTER 4

IMPLEMENTATION

4.1 TECHNIQUE USED- TRANSACTION MANAGEMENT

A transaction defines a logical unit of work that either completely succeeds or produces no result at all. A distributed transaction is simply a transaction that accesses and updates data on two or more networked resources, and therefore must be coordinated among those resources. Transactions provide an "all-or-nothing" proposition, stating that each work-unit performed in a application must either complete in its entirety or have no effect whatsoever. Further, the system must isolate each transaction from other transactions, results must conform to existing constraints and transactions that complete successfully must get written to durable storage. File systems that protect their file system structure consistency through "transactions" (log, modify, commit) so that any kind of soft failure, will still leave the structure of directory, files and metadata (like ACL or owner, group, etc.) in a sound state, as well as increase the chances of successful recovery in case of a hard failure. Unlike the old file systems (FAT, ext2) who required a manually triggered repair operation after a software failure to ensure their consistence, modern file systems (NTFS, ext3, JFS) automatically take care of that by using a mechanism similar to database log/recovery mechanism[1].

4.1.1 COMMIT:

Committed and Working States The file system has two distinct states—the committed (on-disk) state and the working state. The committed state is the state found on the media at initialization time. The committed state is also the state of the file system as written to the media at the last completed transaction point.

Automatic transactions are done in conjunction with prescribed file system operations. The developer can tell application to do a transaction point every time a file is closed, for example. Transaction points can be configured for virtually all standard file system operations, as desired by the system integrator or application programmer. Timed transactions can be used to force a transaction to occur regularly at a given frequency. Explicit transactions can be done under application control.

4.1.2 ROLLBACK:

Transactions provide an "all-or-nothing" proposition, stating that each work-unit performed in a application must either complete in its entirety or have no effect whatsoever. Further, the system must isolate each transaction from other transactions, results must conform to existing constraints and transactions that complete successfully must get written to durable storage.

Automatic transactions are done in conjunction with prescribed file system operations. The developer can tell application to do a transaction point every time a file is closed, for example. Transaction points can be configured for virtually all standard file system operations, as desired by the system integrator or application programmer. Timed transactions can be used to force a transaction to occur regularly at a given frequency. Explicit transactions can be done under application control.

4.1.3 ADVANTAGES OF TRANSACTION MANAGEMENT:

The ACID property describes the transaction management well. ACID stands for Atomicity, Consistency, isolation and durability.

Atomicity: means either all successful or none.

Consistency: ensures bringing the database from one consistent state to another consistent state.

Isolation: ensures that transaction is isolated from other transaction.

Durability: means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

CHAPTER 5**SOFTWARE TESTING****5.1 UNIT TESTING**

The software units in a system are modules and routines that are assembled and integrated to perform a specific function. Unit testing focuses first on modules, independently of one another, to locate errors. This enables, to detect errors in coding and logic that are contained within each module. This testing includes entering data and ascertaining if the value matches to the type and size supported by C++. The various controls are tested to ensure that each performs its action as required

Table 5.1 Unit testing

S.NO	INPUT	OUTPUT	REMARKS
1	Give an invalid account no or PIN during insertion	Invalid account no / PIN	Fail
2	Opening new account with existing account no	User with this account no already exists.	Pass
3	Insertion: if special characters are given for account holder's name.	Program accepts name.	Fail
4	During Login, wrong account no and PIN don't match	LOGIN failed	Pass
5	Withdrawing the money more than account balance.	The amount exceeds the current balance.	Pass

CHAPTER 6

RESULTS

The project is compiled and executed on Microsoft Visual C++. We have put in few screen shots in here to show the working of our Application.

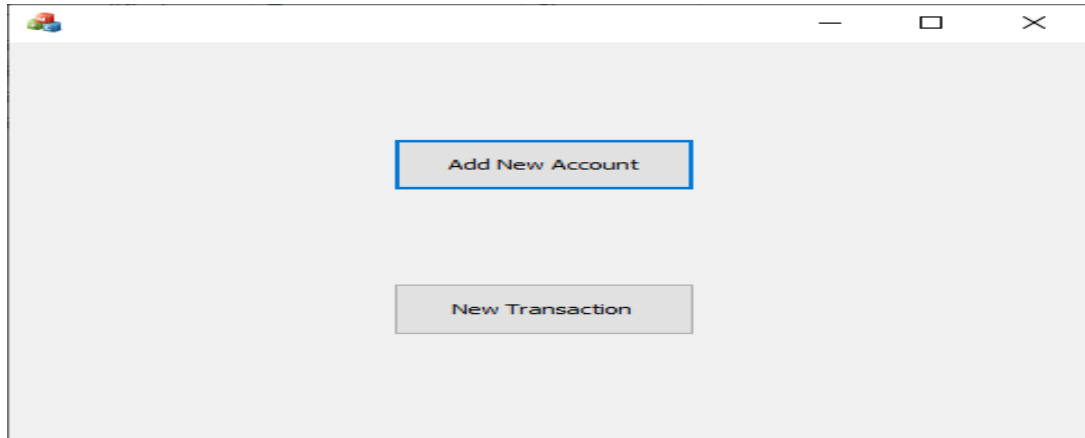
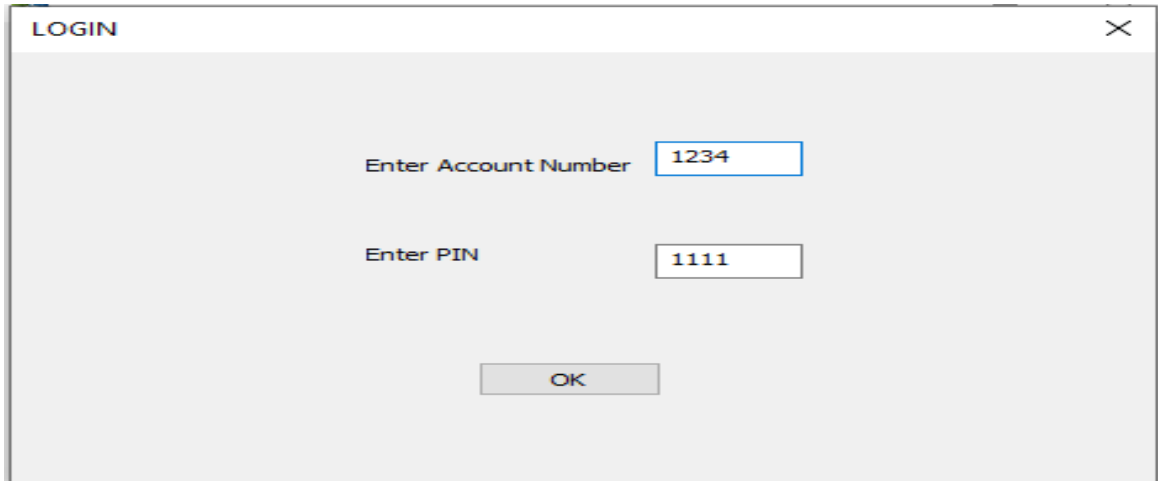


Figure 6.1: First Screen

The user when enter the ATM, the machine displays the first screen (Figure 6.1) It displays 2 buttons, add new account and new transaction.

Figure 6.2: Inserting account details

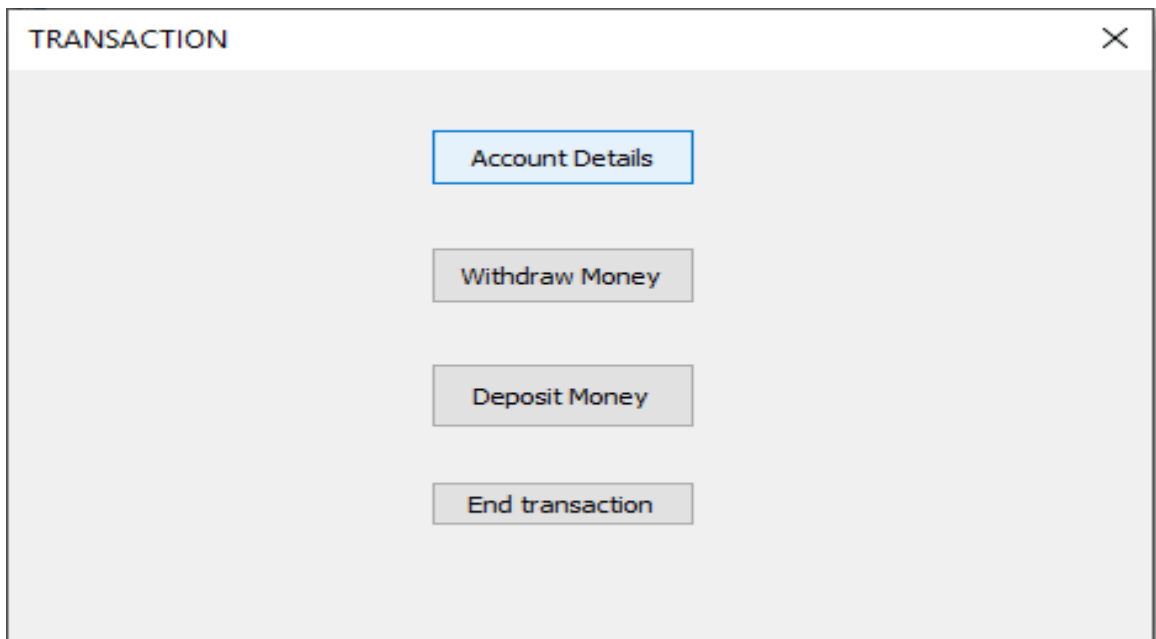
The user on clicking the ADD NEW ACCOUNT button, display the open account screen (Figure 6.2) where the user enters the account details for which he want to open account.



A screenshot of a web application window titled "LOGIN". The window has a light gray background and a close button (X) in the top right corner. It contains two input fields: "Enter Account Number" with the value "1234" and "Enter PIN" with the value "1111". Below these fields is a single "OK" button.

Figure 6.3: LOGIN

On clicking new transaction, the LOGIN screen is displayed (Fig 6.3), and enters account no and ATM PIN for validation. On clicking OK, it displays failed if invalid input is entered or successful if it is a authenticate account.



A screenshot of a web application window titled "TRANSACTION". The window has a light gray background and a close button (X) in the top right corner. It contains four buttons stacked vertically: "Account Details" (highlighted with a blue border), "Withdraw Money", "Deposit Money", and "End transaction".

Figure 6.4: Transaction Page

On successful login, the Transaction screen is displayed (Fig 6.4), in which 4 buttons are displayed.

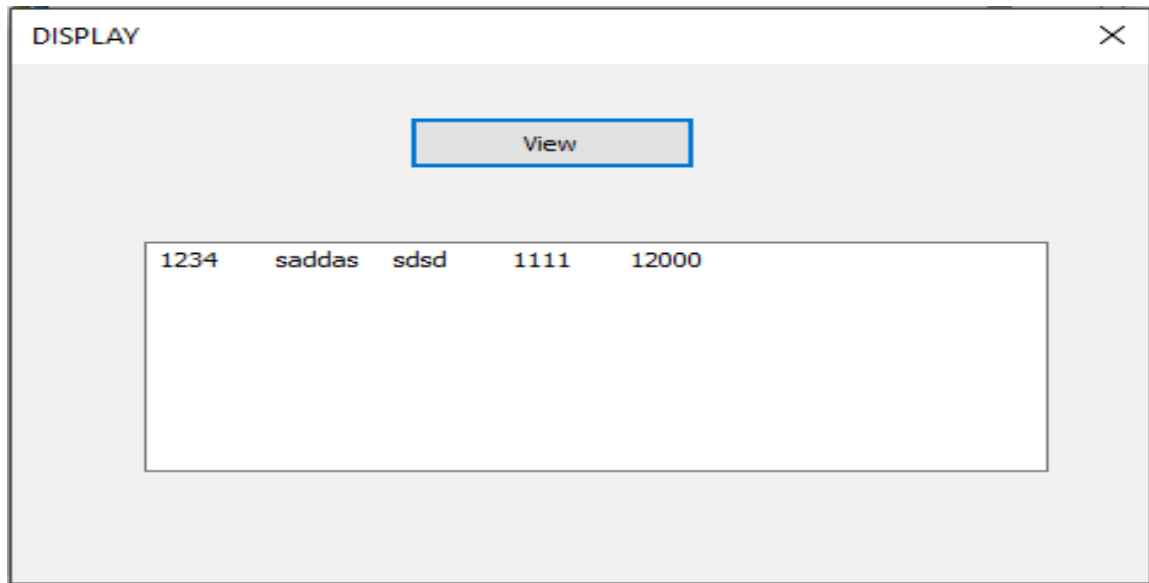


Figure 6.5: Display screen

On clicking the ACCOUNT DETAILS, the display screen is displayed (Fig 6.5) which displays the authenticate account holder details.

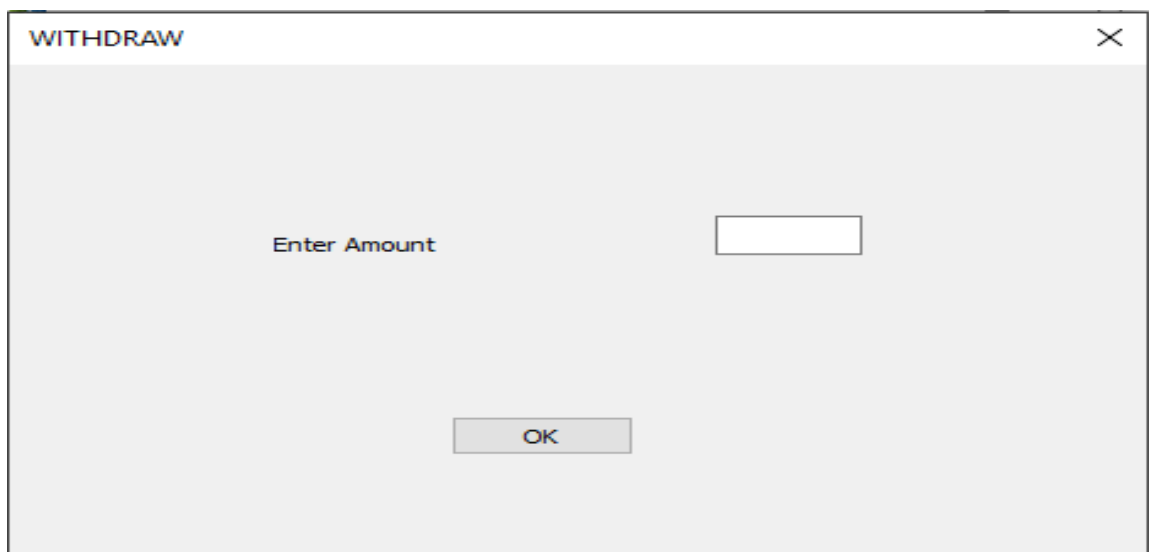


Figure 6.6: Withdraw and Deposit Screen

On clicking WITHDRAW, the withdraw screen is displayed (Fig 6.6), where the user enter amount which is deducted from his account and he gets his cash. On clicking DEPOSIT, the deposit screen displayed and he enters amount which is credited and added to his account.

CHAPTER 7

CONCLUSION

This software is efficient in maintaining account holder's account details and can easily perform transaction operations on account's records and also works to handle the information of the users account on the particular banks. This software also reduces the work load of visiting the bank again and again with our passbooks and have to wait in long queues to request and deposit money and to perform other transaction operations. This saves time and provide security too.

REFERENCES

- [1] Blog: Stack over flow <https://www.stackoverflow.com>
- [2] FILE STRUCTURES, Michael J Folk, Bill Zoellick, Greg Riccardi .
- [3] Google Search [https:// www.google.com](https://www.google.com)
- [4] Geeks for Geeks <https://www.geeksforgeeks.com>

APPENDIX A

FRONT END CODE

Insert.cpp

In the new record form the user can insert the new account details adding their details like account number, name, type, ATM PIN and initial account balance. If the details are not valid, on the login page the error message 'failed' is displayed. If the details are valid he can perform transaction.

```
#include "pch.h"
#include "final.h"
#include "INSERT.h"
#include "afxdialogex.h"
#include<string>
#include<fstream>
#include "Header.h"
// INSERT dialog
```

```
IMPLEMENT_DYNAMIC(INSERT, CDialogEx)
```

```
INSERT::INSERT(CWnd* pParent /*=nullptr*/)
    : CDialogEx(IDD_DIALOG1, pParent)
    , aname(_T(""))
    , aano(_T(""))
    , atype(_T(""))
    , aamt(_T(""))
    , apin(_T(""))
{
}

}
```

```
INSERT::~~INSERT()
{
}
```

```
void INSERT::DoDataExchange(CDataExchange* pDX)
{
    CDialogEx::DoDataExchange(pDX);
    // DDX_Text(pDX, txtAccountHolderName, acno);
    DDX_Text(pDX, txtAccountHolderName, aname);
    DDX_Text(pDX, txtAccountType, atype);
    DDX_Text(pDX, txtPIN, apin);
    DDX_Text(pDX, txtAmount, aamt);
    DDX_Text(pDX, txtAccountNumber, aano);
}
```

```
}

BEGIN_MESSAGE_MAP(INSERT, CDialogEx)
    ON_BN_CLICKED(btnOK1, &INSERT::OnBnClickedbtnok1)
END_MESSAGE_MAP()

// INSERT message handlers

void INSERT::OnBnClickedbtnok1()
{
    // TODO: Add your control notification handler code here
    UpdateData(TRUE);
    std::string values[5];
    values[0] = (CT2CA)aano;
    values[1] = (CT2CA)aname;
    values[2] = (CT2CA)atype;
    values[3] = (CT2CA)apin;
    values[4] = (CT2CA)aamt;
    addAccount(values);
    MessageBeep(1);
    this->OnOK();
}
```

LOGIN.cpp

```
void LOGIN::OnBnClickedbtnok2()
{
    UpdateData(1);
    bool b;
    CString c;
    std::string s1, s2;
    GetDlgItem(txtEnterAccountNumber)->GetWindowTextW(c);
    s1 = (CW2A)c.GetString();
    GetDlgItem(txtEnterPIN)->GetWindowTextW(c);
    s2 = (CW2A)c.GetString();
    b = unpack(s1, s2);
    if (b)
    {
        MessageBox(L"Sucessful");
        TRANSACTION dlg3;
        dlg3.DoModal();
    }
    else
        MessageBox(L"Failed");
}
```

APPENDIX B

BACK END CODE

Header.h

In this header file all the important methods like add, unpack, display, withdraw and deposits are present and implemented.

```
#pragma once
#include<conio.h>
#include<stdio.h>
#include<iostream>
#include<ios>
#include<limits>
#include<fstream>
#include<string.h>
using namespace std;

class Account
{
public:
    std::string anum, name, type, apin, bal;
} found[50];
int n = 0;

bool unpack(std::string key_acno, std::string key_pin)
{
    fstream f3("index.txt", ios::in);
    int flag = 0, k = 0;
    std::string anum, apin;
    char* str = new char[100];
    //char* result = new char[100];
    for (int j = 0; j < n; j++)
    {
        f3.getline(str, 20, '|');
        anum = str;
        f3.getline(str, 20, '\n');
        apin = str;
        if ((key_acno.compare(anum) == 0) && (key_pin.compare(apin)))
        {
            found[k].anum = anum;
            flag = 1;
            break;
        }
    }
    f3.close();
}
```

```
        if (flag == 1)
            return TRUE;
        else
            return FALSE;

    }

void addAccount(std::string values[])
{
    Account a[30];
    a[n].anum = values[0];
    a[n].name = values[1];
    a[n].type = values[2];
    a[n].apin = values[3];
    a[n].bal = values[4];
    n++;

    fstream f1("bank.txt", ios::out|ios::app);
    fstream f2("index.txt", ios::out|ios::app);

    //fstream f2(ifname, ios::out);
    for (int i = 0; i < n; i++)
    {
        f1 << a[i].anum.c_str() << '|' << a[i].name.c_str() << '|' <<
a[i].type.c_str() << '|' << a[i].apin.c_str() << '|' << a[i].bal.c_str() << endl;
        f2 << a[i].anum.c_str() << '|' << a[i].apin.c_str() << '|' << endl;

    }
    f1.close();
    f2.close();
}

char* display()
{
    Account acc[50];

    int k = 0;

    char* result = new char[100];
    strcpy_s(result, 20, "");
    for (int i = 0; i < n; i++)
    {
        std::fstream f4("bank.txt", std::ios::in);
        std::string anum, name, type, apin, bal;

        char* str = new char[100];
```

```
        f4.getline(str, 20, '|');
        anum = str;
        f4.getline(str, 20, '|');
        name = str;
        f4.getline(str, 20, '|');
        type = str;
        f4.getline(str, 20, '|');
        apin = str;
        f4.getline(str, 20, '\n');
        bal = str;
        if (anum.compare(found[k].anum) == 0)
        {
            strcpy_s(result, 100, anum.c_str());
            strcat_s(result, 100, "\t");
            strcat_s(result, 100, name.c_str());
            strcat_s(result, 100, "\t");
            strcat_s(result, 100, type.c_str());
            strcat_s(result, 100, "\t");
            strcat_s(result, 100, apin.c_str());
            strcat_s(result, 100, "\t");
            strcat_s(result, 100, bal.c_str());
            strcat_s(result, 100, "\r\t");
            f4.close();
            break;
            k++;
        }
    }

    return result;
}

char withdrawMoney(std::string key_amt) {

    fstream f1("bank.txt", ios::in);
    fstream f3("index.txt", ios::in);
    int flag = 0, k = 0;
    std::string anum, apin;
    char* str = new char[100];
    //char* result = new char[100];
    for (int j = 0; j < n; j++)
    {
        f3.getline(str, 20, '|');
        anum = str;
        f3.getline(str, 20, '\n');
        apin = str;
        if ((key_acno.compare(anum) == 0) && (key_pin.compare(apin)))
        {
            found[k].anum = anum;
            flag = 1;
            break;
        }
    }
}
```

```
        }

    }
    f3.close();
    if (flag == 1)
        return TRUE;
    else
        return FALSE;

}

void depositMoney(string num) {
    int m;
    cout << "Enter Amount:";
    cin >> m;
    if (m % 100 != 0) {
        cout << "Enter in Multiple of 100s." << endl;
        return;
    }
    Account a;
    fstream f1(fname, ios::in);
    fstream f2("temp.txt", ios::out);
    fstream f3("itemp.txt", ios::out);
    string line;
    while (getline(f1, line)) {
        a.unpack(line);
        if (isEqual(a.num, num)) {
            cout << "Money Deposited." << endl;
            a.bal += m;
            line = a.pack();
        }
        f3 << a.num << "|" << getSize("temp.txt") << endl;
        f2 << line << endl;
    }
    f1.close();
    f2.close();
    f3.close();
    remove(fname);
    rename("temp.txt", fname);
    remove(ifname);
    rename("itemp.txt", ifname);
}
```