INDEX

INTRODUCTION

A complete software programme called a bank application is made to make running a bank or other financial institution easier and more efficient. It acts as a central platform for integrating several features, enabling the effective management of financial transactions, client accounts, and other crucial banking tasks.

Bank applications have recently integrated digital channels and online banking features in response to technological improvements. Customers may now effortlessly access their accounts, conduct transactions, and carry out other banking tasks using online portals or mobile applications.

WHAT IS THE NEED OF A BANK APPLICATION?

An effective bank application is required to increase operational effectiveness, provide security, and assure compliance, enhance client service, make data-driven decisions possible, and adjust to the world of digital banking. It gives banks the ability to satisfy client demands, remain competitive, and have a firm control on the financial sector.

FEATURES OF THE PROJECT

The Application consists of the following functions:

Account Class:

This class is an abstract class and consists of the common features to both current and savings accounts.

The Account Class encompasses the following methods or features:

New User:

This feature is used to open a new bank account i.e. either current or savings.

Deposit Money:

This feature is used to deposit money from the respective bank account.

Withdraw Money:

This feature is used to withdraw money from the respective bank account.

View Bank Details:

This feature is used to view the details of the respective existing bank account.

Modify Details:

This feature is used to modify the details of the respective bank account.

Fund Transfer:

This feature is used to make transactions or transfer money from one bank account to another.

View Transaction History:

This feature is used to view transactions made by the account holder.

Current Class:

This class is derived from account class and offers some extra features to the current account holders.

The Current Class consists of the following methods or features:

Login:

This feature is used to login into the existing current account using the Customer ID and Password.

Issue Cheque Book:

This feature is used issue a new cheque book to the respective current account holder.

View Charges Imposed:

This feature is used to view the penalties or the charges imposed on the respective current account holder.

Logout:

This feature is used to securely LogOut from the bank account.

Savings Class:

This class is derived from account class and offers some extra features to the saving account holders.

The Savings Class consists of the following methods or features:

Login:

This feature is used to login into the existing savings account using the Customer ID and Password.

Issue Debit Card:

This feature is used issue a Debit Card to the respective savings account holder (if not previously issued).

Get A Loan:

This feature is used to Get a Loan and encompasses the following types of loans:

- *House Loan*
- *Personal Loan*
- *Education Loan*
- *Agricultural Loan*
- *Vehicle Loan*
- *Gold Loan*

Each type of loan has its own rate of interest which will be displayed to the user.

Logout:

This feature is used to securely LogOut from the bank account.

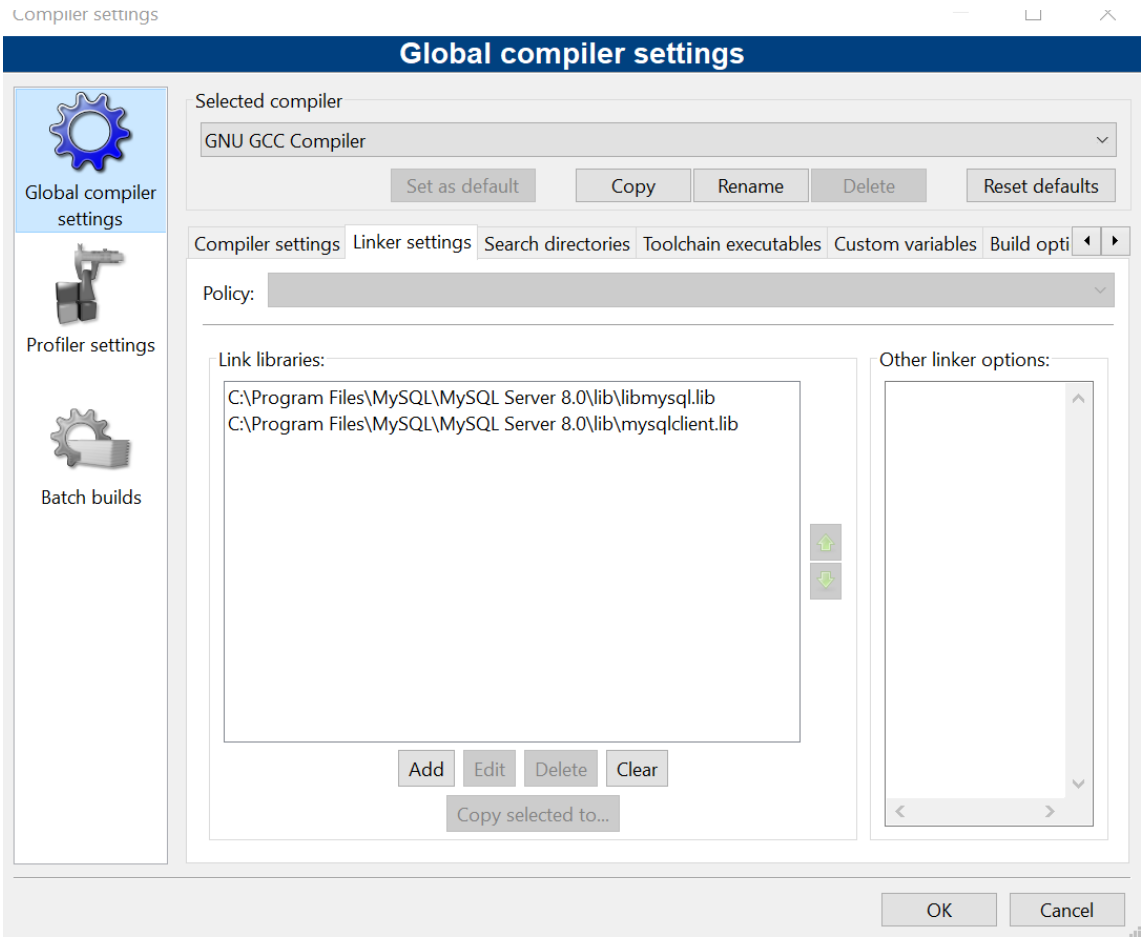
Find Nearest Bank Branch:

This feature is used to find the nearest bank branch w.r.t. the entered locality.

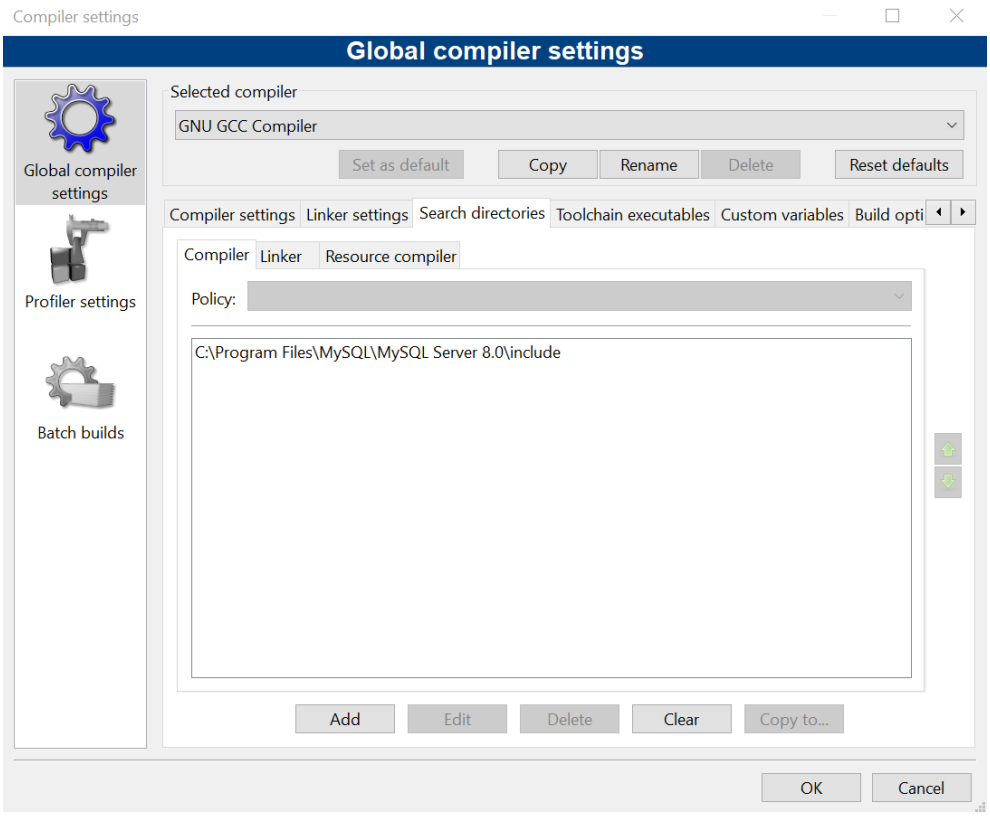
PRE-REQUISITES

The settings required in CODE::BLOCKS to run the program successfully are:

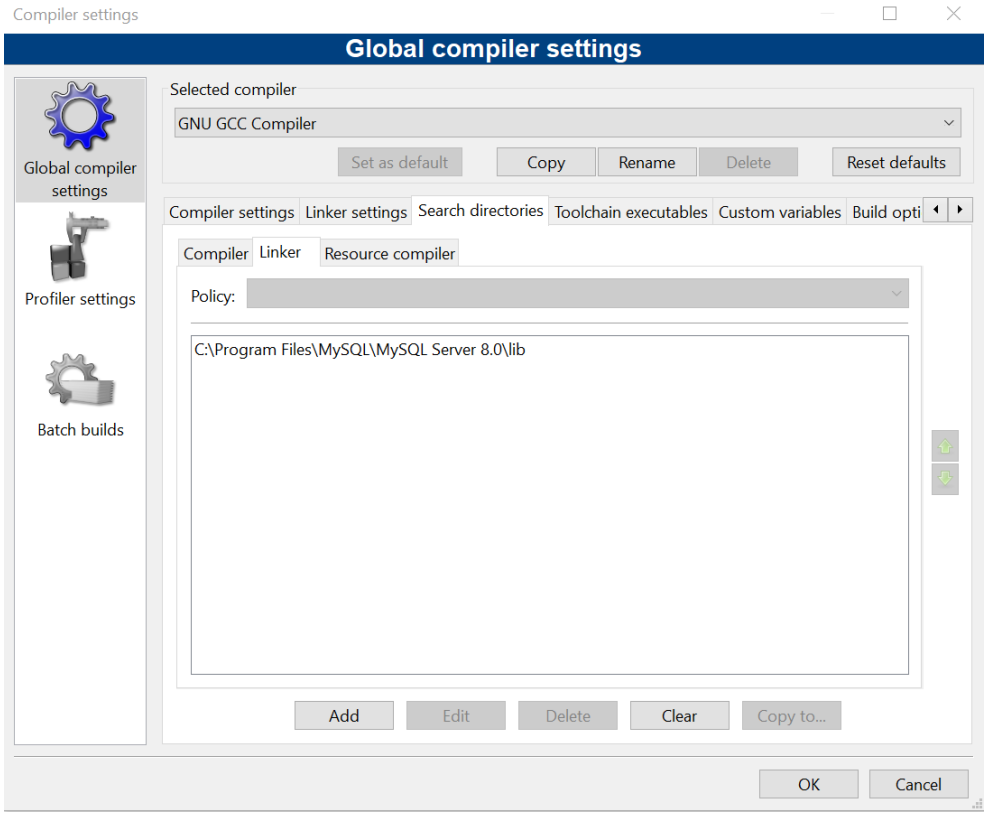
- Go to Compiler Settings in CODE::BLOCKS and **ADD** the following libraries as shown:



- Now go to Search Directories and **ADD** the following folder in Compiler Tab:



- Now go to Search Directories and **ADD** the following folder in Linker Tab:



The settings required in MySQL to run the program successfully are:

- Open MySQL Command Line Client and execute the following commands:
  - create database bank;
  - create table current (CustomerID char(8) primary key, password varchar(16) unique, FirstName varchar(20), Lastname varchar(20), DOB date, Email varchar(50), MobileNo char(10), Address varchar(50), IDType varchar(50), IDNum varchar(10) unique, balance decimal(10,2) default 0, ChequeBook varchar(50) default ‘Not Issued’);
  - create table savings (CustomerID char(8) primary key, password varchar(16) unique, FirstName varchar(20), Lastname varchar(20), DOB date, Email varchar(50), MobileNo char(10), Address varchar(50), IDType varchar(50), IDNum varchar(10) unique, balance decimal(10,2) default 0, DebitCard varchar(50) default ‘Not Issued’);
  - create table thistory (CustomerID char(8), Type varchar(10), Amount decimal(10,2), Date_Time datetime default NOW());

DATA STRUCTURES IMPLEMENTED

AVL TREE:

This Data Structure is used for fast data retrieval by assigning every account to a node in the AVL Tree.

```
struct node
{
    string custid;
    float balance;
    struct node *l;
    struct node *r;
    int h;
}*root;
int height(struct node *root)
{
struct node *rightr(struct node *root)
{
struct node *leftr(struct node *root)
{
struct node *lr(struct node *root)
{
struct node *rl(struct node *root)
{
struct node *ins(struct node *root,struct node *k)
{
    if(root==NULL)
        return k;
    if(k->custid<root->custid)
    {
    else
    {
    root->h=height(root);
    int bal=height(root->l)-height(root->r);
    if(bal>1&&k->custid<root->l->custid)
        root=rightr(root);
    if(bal<-1&&k->custid>root->r->custid)
        root=leftr(root);
    if(bal>1&&k->custid>root->l->custid)
        root=lr(root);
    if(bal<-1&&k->custid<root->r->custid)
        root=rl(root);
    return root;
}
```

STACK:

This Data Structure is used for recording the Transaction History of the customer.

```
struct THistory
{
    public:
    string custid;
    string type;
    float amount;
    string datetime;
};

template <typename t>
class stack
{
    public:
    int f=-1;
    t a[1000];
    void push(THistory s)
    {
        f++;
        a[f]=s;
    }
    void pop()
    {
        f--;
    }
    bool empty()
    {
        if(f==-1)
            return 1;
        else
            return 0;
    }
    t top()
    {
        return a[f];
    }
};
stack <THistory> th1;
stack <THistory> th2;
```

GRAPH:

This Data Structure is used for find the nearest bank branch with respect to the locality entered by the user.

```
struct vertex
{
    string city;
    string place;
    int d;
    struct vertex *next;
}**a,*temp;
struct vertex *newnode(string c,string v,int d)
{
struct vertex *inse(struct vertex *head,string c,string e,int d)
{
class graph
{
    public:
    int n;
    struct vertex **a;
    graph(int x)
    {
    int getin(string c,string v)
    {
    void addedge(string c,string v,string e,int d)
    {
    void nearestnode(string c,string v)
    {
};
```

GREEDY ALGORITHM:

This Algorithm is used find the nearest node of a vertex in a graph.

CODE SNIPPETS

```
#include <iostream>
#include <windows.h>
#include <mysql.h>
#include <string>
#include <cmath>
#include <iomanip>
#include <ctime>
using namespace std;

MYSQL* con;
MYSQL_RES *res;
MYSQL_ROW row;

class account
{
    public:
    string pass;
    string custid;
    string fname;
    string lname;
    string dob;
    string email;
    string mob;
    string add;
    string id;
    string idnum;
    string debitcard;
    float balance;
    public:
    virtual void mainMenu()=0;
    virtual void logout()=0;
    void newUser()
    {
    void deposit(float amn)
    {
    void withdraw(float amo)
    {
    void viewBalance()
    {
    void viewDetails()
    {
    void modifyDetails()
    {
```

fetchalldata():

This function is retrieving data from the database and inserting into AVL Tree by forming nodes.

```
struct node *fetchalldata(struct node *root,string t)
{
    mysql_query(con,("select CustomerID,balance from "+t+"").c_str());
    res=mysql_use_result(con);
    while(row=mysql_fetch_row(res))
    {
        struct node *s=new struct node;
        s->custid=row[0];
        s->balance=atoi(row[1]);
        s->l=NULL;
        s->r=NULL;
        s->h=0;
        root=ins(root,s);
    }
    mysql_free_result(res);
    return root;
}
```

sear():

This function is searching for the required node by regular AVL Tree searching using Customer ID of the account holder.

```
struct node *sear(struct node *root,string k)
{
    if(root==NULL)
        return NULL;
    if(root->custid==k)
        return root;
    else if(k<root->custid)
        return sear(root->l,k);
    else if(k>root->custid)
        return sear(root->r,k);
}
```

fundtransfer():

This function uses the fetchalldata() & sear() functions to quickly search for the beneficiary and transfers the amount.

```
void fundtransfer()
{
    string c,t;
    float amt;
    cout<<"\n------------------------Fund Transfer Portal--------
    cout<<"Enter Beneficiary Customer ID : ";
    cin>>c;
    cout<<"Enter Beneficiary Account Type(current/savings) : ";
    cin>>t;
    cout<<"Enter Amount : ";
    cin>>amt;
    root=fetchalldata(root,t);
    balance-=amt;
    struct THistory s1;
    s1.custid=custid;
    s1.type="Debit";
    s1.amount=amt;
    s1.datetime=now();
    th2.push(s1);
    root=sear(root,c);
    root->balance+=amt;
    struct THistory s2;
    s2.custid=c;
    s2.type="Credit";
    s2.amount=amt;
    mysql_query(con,("insert into thistory(CustomerID, Type, Amount
    cout<<"\n--------------------Fund Transferred Successfully-----
    int g;
    l:
    cout<<"1. To go Back To MAIN MENU\n";
    cout<<"2. LOGOUT\n";
    cout<<"Enter Your Choice: ";
    cin>>g;
    switch(g)
    {
        case 1:
            mainMenu();
            break;
        case 2:
            logout();
            break;
        default:
            cout<<"!!Wrong Value Entered!!";
            goto l;
    }
}
```

fetchallhistory():

This function is used to retrieve all previous transaction history from the database into a STACK.

```
stack <THistory> fetchallhistory(stack <THistory> th1,string cid)
{
    mysql_query(con,("select * from thistory where CustomerID='"+cid+"'").c_str());
    res=mysql_use_result(con);
    while(row=mysql_fetch_row(res))
    {
        struct THistory s;
        s.custid=row[0];
        s.type=row[1];
        s.amount=atoi(row[2]);
        s.datetime=row[3];
        th1.push(s);
    }
    mysql_free_result(res);
    return th1;
}
```

viewthistory():

This function uses the stack previously created using the fetchallhistory() function to display all the transaction history of the requested customer.

```
    void viewthistory()
    {
        stack <THistory> t1=th1;
        stack <THistory> t2=th2;
        cout<<"\n-------------------------Transaction History-------------------------\n\n";
        cout<<"+----------------------+------------+--------------------+\n";
        cout<<"| Type of Transaction  |   Amount   |    Date & Time     |\n";
        cout<<"+----------------------+------------+--------------------+\n";
        while(!t2.empty())
        {
            struct THistory s=t2.top();
            cout<<"| "<<left<<setw(21)<<s.type<<"|"<<setw(12)
                <<s.amount<<"|"<<setw(19)<<s.datetime<<"|"<<endl;
            t2.pop();
        }
        while(!t1.empty())
        {
            struct THistory s=t1.top();
            cout<<"| "<<left<<setw(21)<<s.type<<"|"<<setw(12)
                <<s.amount<<"|"<<setw(19)<<s.datetime<<"|"<<endl;
            t1.pop();
        }
        cout<<"+----------------------+------------+--------------------+\n\n";
```

updatealldata():

This function is used to update the balance of all the customers in the databases by using data in the AVL tree in the current instance.

```
void updatealldata(struct node *root,string t)
{
    if(root!=NULL)
    {
        mysql_query(con,("update "+t+" set balance='"+to_string
        updatealldata(root->l,t);
        updatealldata(root->r,t);
    }
}
```

updateallhistory():

This function is used to insert the history of the transactions made by the customer during current instance which were pushed in the stack temporarily.

```
void updateallhistory(stack <THistory> th2)
{
    while(!th2.empty())
    {
        struct THistory st=th2.top();
        mysql_query(con,("insert into thistory(CustomerID, Type, Amount,
        th2.pop();
    }
}
```

logout():

This function uses the updatealldata() & updateallhistory() functions to finally update the balances and transactions histories into the database after the user logouts.

```
class current: public account
{
    public:
    void issueCHQ()
    {
    void chargesImp()
    {
    void logout()
    {
        cout<<"\n\n--------------------------------THANK YOU--------
        updatealldata(root,"current");
        updateallhistory(th2);
        mysql_query(con,("update current set FirstName='"+fname+"'
        exit(1);
    }
     void mainMenu()
     {
         int n;
         float amn,amo;
         u:
         cout<<"\n1. Deposit Money\n";
         cout<<"2. Withdraw Money\n";
         cout<<"3. View Account Balance\n";
         cout<<"4. Issue Cheque Book\n";
         cout<<"5. Fund Transfer\n";
         cout<<"6. View Transaction History\n";
         cout<<"7. View Charges Imposed\n";
         cout<<"8. View Account Details\n";
         cout<<"9. Modify Account Details\n";
         cout<<"10. Logout\n";
         cout<<"\nEnter your Choice: ";
         cin>>n;
```

login():

This function is used to login into customers bank account using the given Customer ID and Password.

```
    void login()
    {
        string uname,pas;
        j:
        cout<<"\nEnter Login Credentials: \n\n";
        cout<<"Enter Customer ID: ";
        cin>>uname;
        cout<<"Enter Password: ";
        cin>>pas;
        mysql_query(con,("select * from current where CustomerID='"+uname+"'
        res=mysql_use_result(con);
        row=mysql_fetch_row(res);
        if(row!=NULL)
        {
        else
        {
            cout<<"\n                    !!INCORRECT LOGIN CREDENTIALS!!\n";
            cout<<"Enter Correct Login Credentials.\n\n";
            goto j;
        }
        mysql_free_result(res);
        th1=fetchallhistory(th1,custid);
    }
};
```

```
class savings: public account
{
    public:
    void loan()
    {
    void debitCard()
    {
    void logout()
    {
    void mainMenu()
    {
        int n;
        float amn,amo;
        u:
        cout<<"\n1. Deposit Money\n";
        cout<<"2. Withdraw Money\n";
        cout<<"3. View Account Balance\n";
        cout<<"4. Fund Transfer\n";
        cout<<"5. Get a Loan\n";
        cout<<"6. View Transaction History\n";
        cout<<"7. Issue Debit Card\n";
        cout<<"8. View Account Details\n";
        cout<<"9. Modify Account Details\n";
        cout<<"10. Logout\n";
        cout<<"\nEnter your Choice: ";
```

nearestnode():

This function is used to find the nearest vertex with respect to another vertex.

```
    void nearestnode(string c,string v)
    {
        int t=getin(c,v);
        int o=0;
        string f;
        for(temp=a[t];temp!=NULL;temp=temp->next)
        {
            if(temp->d>o)
            {
                o=temp->d;
                f=temp->place;
            }
        }
        cout<<"\nNearest Branch is: "<<f;
        cout<<"\nDistance from origin is: "<<o<<endl<<endl;
    }
};
```

findnearestbranch():

This function is uses the nearestnode() function to find the nearest bank branch.

```
void findnearestbranch()
{
    string c,p;
    cout<<"\n-----------------------Find Nearest Bank Branch--
    cout<<"Enter your City: ";
    cin>>c;
    cout<<"Enter your Locality: ";
    cin>>p;
    graph g(3);
    g=initializegraph();
    g.nearestnode(c,p);
}
```

```
int main()
{
    con=mysql_init(0);
    con=mysql_real_connect(con,"localhost","root","root","bank",0,NULL,0);
    if(!con)
        cout<<"!!ERROR!! Database not connected.";
    qw:
    int n;
    cout<<"                  DATA STRUCTURES & ALGORITHMS PROJECT\n\n";
    cout<<"                        WELCOME TO BANK OF INDIA  \n\n";
    cout<<"                               Home Page\n\n";
    cout<<"1. New User...Register\n";
    cout<<"2. Existing User...Login\n";
    cout<<"3. Find Nearest Bank Branch...\n";
    p:
    cout<<"\nEnter Your Choice: ";
    cin>>n;
    switch(n)
    {
}
```

CONCLUSION

In conclusion, the Bank Application successfully integrates crucial banking functionalities such as deposit, withdrawal, fund transfer, and transaction history. The implementation of advanced data structures, including stacks for transaction management, AVL trees for efficient account retrieval, and graphs for comprehensive relationship mapping, enhances the system's performance and scalability. By employing these structures, the project achieves a balance between user-friendly interactions and robust data management. The system not only caters to current banking needs but also lays a foundation for adaptability and innovation in the ever-evolving financial landscape.

BIBLIOGRAPHY

To develop this project the following references were used:

- 'C++: THE COMPLETE REFRENCE'
- https://www.google.com
- https://www.codeblocks.org.in
- https://www.geekofgeeks.org