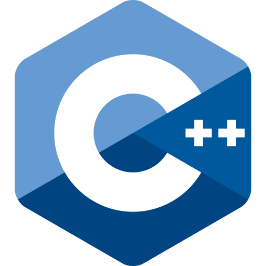
****

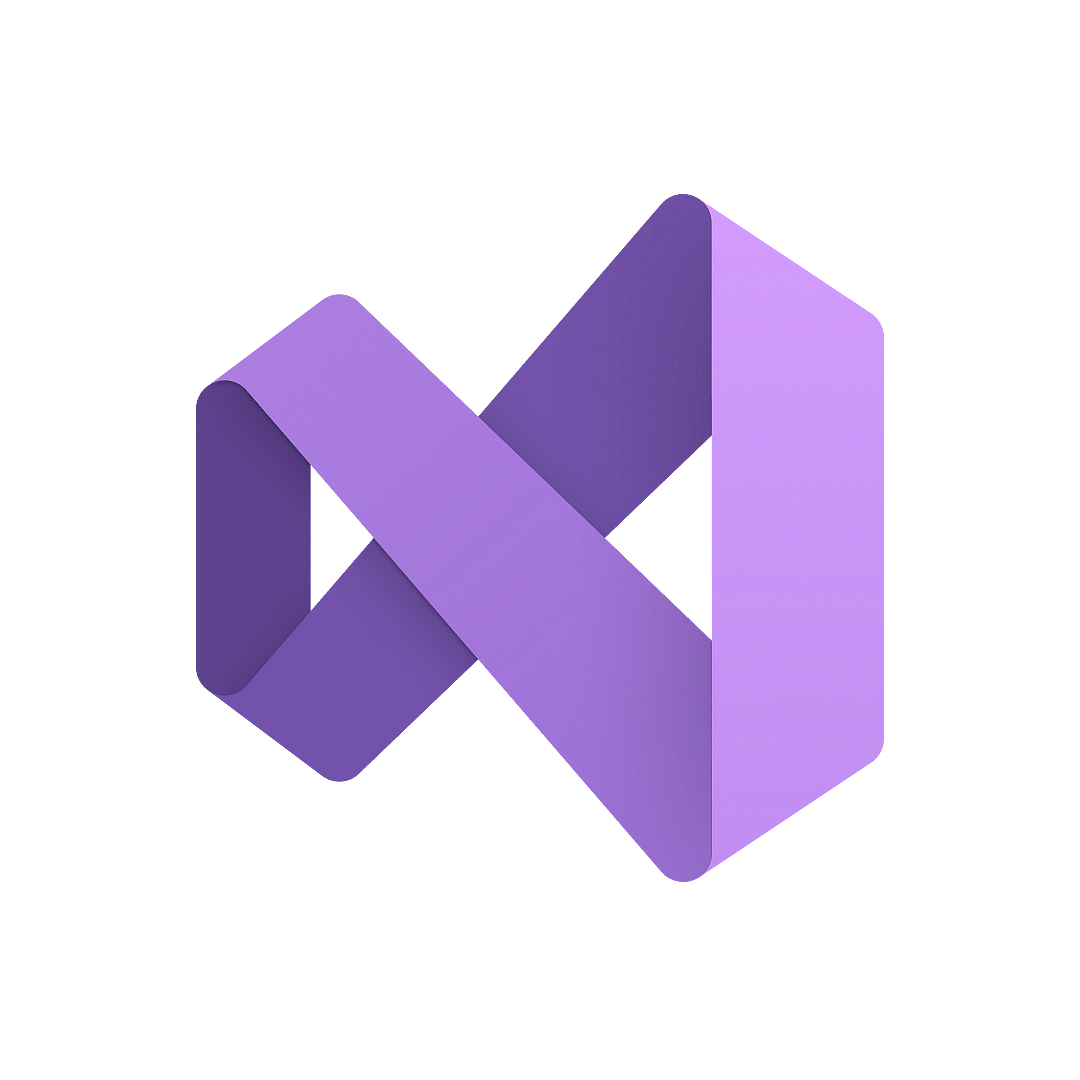
**<Computer Programming>**

***<Semester Project>***



**<Project By> </Muhammad Muneeb Zahid (01-134251-060)**

**Awwad Mehdi (01-134251-014)>**

****

**#include<Project Report>**

**Using namespace std:**

**int main(){Banking Management System }**

**char Project Title {**

Bank-Al-Bahria: A Procedural Banking Management System in C++ };

# char Objective {

To design and implement a file-based banking management system in C++ that allows users to manage bank accounts with features such as account creation, deposits, withdrawals, balance inquiry, transaction history, and inter-account money transfers };

# char System Design {

The system uses procedural programming, dynamic memory allocation, and file handling.

};

**char Architecture Overview {**

* Main Menu
* Account Handling
* Transaction Logic
* File Management

Each module is connected via function calls and data stored in dynamically allocated arrays. };

# char Features Implemented {

* Create Account with CNIC and PIN
* Deposit and Withdraw money
* Check Balance
* Send money to other accounts
* Maintain transaction history
* Save/load data using files

# };

**char array[Code Structure and Key Snippets]{**

# array[1] = void Data Storage{

/\* The system uses **dynamic arrays** to store data for up to 100 accounts \*/

int\* account\_numbers = new int[max\_accounts];

char\*\* names = new char\* [max\_accounts];

int\* pins = new int[max\_accounts];

double\* balances = new double[max\_accounts];

char\*\* cnics = new char\* [max\_accounts];

int total\_accounts = 0;

/\* Each index represents one account. Data is persisted via accounts.txt. \*/

}

# array[2] = void Account Creation{

|  |
| --- |
| void create\_account() {      if (total\_accounts >= max\_accounts) {          cout << "Account limit reached.\n";          return;      }      int acc\_no, pin;      double deposit;      char temp\_name[100];      char temp\_cnic[20];      cout << "Enter account number: ";      cin >> acc\_no;      if (find\_account\_index(acc\_no) != -1) {          cout << "Account already exists.\n";          return;      }      cout << "Enter account holder name: ";      cin.ignore();      cin.getline(temp\_name, 100);      cout << "Set 4-digit PIN: ";      cin >> pin;      if (pin < 1000 || pin > 9999) {          cout << "Invalid PIN. Must be 4 digits.\n";          return;      }      cout << "Enter CNIC (xxxxx-xxxxxxx-x): ";      cin.ignore();      cin.getline(temp\_cnic, 20);      if (strlen(temp\_cnic) != 15 || temp\_cnic[5] != '-' || temp\_cnic[13] != '-') {          cout << "Invalid CNIC format.\n";          return;      }      cout << "Enter initial deposit: ";      cin >> deposit;      // Store account info      account\_numbers[total\_accounts] = acc\_no;      names[total\_accounts] = new char[strlen(temp\_name) + 1];      strcpy\_s(names[total\_accounts], strlen(temp\_name) + 1, temp\_name);      pins[total\_accounts] = pin;      balances[total\_accounts] = deposit;      cnics[total\_accounts] = new char[strlen(temp\_cnic) + 1];      strcpy\_s(cnics[total\_accounts], strlen(temp\_cnic) + 1, temp\_cnic);      log\_transaction(acc\_no, "Account created with deposit", deposit, deposit);      total\_accounts++;      cout << "Account created successfully.\n";      save\_accounts();  } |

// Validates CNIC and PIN.

}

# array[3] = void Deposit and Withdraw {

void deposit() {

    int acc\_no;

    cout << "Enter account number: ";

    cin >> acc\_no;

    int index = find\_account\_index(acc\_no);

    if (index == -1) {

        cout << "Account not found.\n";

        return;

    }

    if (!verify\_pin(index)) return;

    double amount;

    cout << "Enter deposit amount: ";

    cin >> amount;

    balances[index] += amount;

    log\_transaction(account\_numbers[index], "Deposit", amount, balances[index]);

    cout << "Deposit successful. New balance: " << balances[index] << "\n";

    save\_accounts();

}

|  |
| --- |
| void withdraw() {      int acc\_no;      cout << "Enter account number: ";      cin >> acc\_no;      int index = find\_account\_index(acc\_no);      if (index == -1) {          cout << "Account not found.\n";          return;      }      if (!verify\_pin(index)) return;      double amount;      cout << "Enter withdrawal amount: ";      cin >> amount;      if (amount > balances[index]) {          cout << "Insufficient funds.\n";          return;      }      balances[index] -= amount;      log\_transaction(account\_numbers[index], "Withdrawal", amount, balances[index]);      cout << "Withdrawal successful. New balance: " << balances[index] << "\n";      save\_accounts();  } |

// Ensures security using PIN verification

// Updates balance and transaction history }

# array[4] = void Transaction Logging{

|  |
| --- |
| void log\_transaction(int acc\_no, const char\* type, double amount, double balance) {      string filename = "transactions\_" + to\_string(acc\_no) + ".txt";      ofstream fout(filename, ios::app);      fout << "[" << get\_timestamp() << "] " << type           << " of " << amount << " | Balance: " << balance << "\n";      fout.close();  } |

/\* Every transaction is logged with timestamp and account number. \*/

}

# array[5] = void send Money{

void send\_money() {

    int sender\_acc, receiver\_acc;

    cout << "Enter your account number (sender): ";

    cin >> sender\_acc;

    int sender\_index = find\_account\_index(sender\_acc);

    if (sender\_index == -1) {

        cout << "Sender account not found.\n";

        return;

    }

    if (!verify\_pin(sender\_index)) return;

    cout << "Enter receiver account number: ";

    cin >> receiver\_acc;

    int receiver\_index = find\_account\_index(receiver\_acc);

    if (receiver\_index == -1) {

        cout << "Receiver account not found.\n";

        return;

    }

    double amount;

    cout << "Enter amount to send: ";

    cin >> amount;

    if (amount <= 0) {

        cout << "Invalid amount.\n";

        return;

    }

    if (amount > balances[sender\_index]) {

        cout << "Insufficient funds.\n";

        return;

    }

    balances[sender\_index] -= amount;

    balances[receiver\_index] += amount;

    log\_transaction(sender\_acc, "Sent money", amount, balances[sender\_index]);

    log\_transaction(receiver\_acc, "Received money", amount, balances[receiver\_index]);

    cout << "Transfer successful.\n";

    cout << "Your new balance: " << balances[sender\_index] << "\n";

    save\_accounts();

}

// Transfers funds securely between two valid accounts }

**array [6] = void File Operations{**

# // Save Accounts

void save\_accounts() {

    ofstream fout("accounts.txt");

    for (int i = 0; i < total\_accounts; i++) {

        fout << account\_numbers[i] << "\n"

             << names[i] << "\n"

             << pins[i] << "\n"

             << balances[i] << "\n"

             << cnics[i] << "\n";

    }

    fout.close();

}

# // Load Accounts

void load\_accounts() {

    ifstream fin("accounts.txt");

    if (!fin) return;  // File not found, start fresh

    while (fin >> account\_numbers[total\_accounts]) {

        fin.ignore(); // Ignore newline after account number

        names[total\_accounts] = new char[100];

        fin.getline(names[total\_accounts], 100);  // Read account holder name

        fin >> pins[total\_accounts];  // Read PIN

        fin >> balances[total\_accounts];  // Read balance

        fin.ignore(); // Ignore newline after balance

        cnics[total\_accounts] = new char[20];

        fin.getline(cnics[total\_accounts], 20);  // Read CNIC

        total\_accounts++;

        if (total\_accounts >= max\_accounts) break;

    }

    fin.close();

}

// Ensures persistence across sessions.

}

## };

## char Sample Output Screens{

};

## char Limitations{

* No encryption or hashing of PINs
* Limited to 100 accounts
* No GUI

## };

Char functions{

// Includes custom functions and standard library functions.

* get\_timestamp
* log\_transaction
* load\_accounts
* save\_accounts
* find\_account\_index
* verify\_pin
* create\_account
* deposit
* withdraw
* check\_balance
* show\_history
* send\_money
* menu
* main
* strcpy\_s (string copy, safe version)
* strlen (string length)
* getline (for reading a line of input from a stream, here cin.getline and ifstream.getline)
* cin.ignore (ignores characters in input buffer)
* cin >> (input extraction operator)
* cout << (output operator)
* time (get current time)
* localtime / localtime\_s (convert time to local time structure)
* strftime (format date/time into string)
* ofstream.open (open a file stream)
* ofstream.close (close a file stream)
* ifstream.open (open a file stream for reading)
* ifstream.close (close a file stream)
* exit (program exit, although not explicitly used, sometimes in error handling)
* new and delete (dynamic memory allocation and deallocation)

};

## char Conclusion{

This project successfully demonstrates a basic **banking system** using procedural programming and file handling. Despite constraints like no OOP, vectors, or advanced DSA, the system achieves essential banking functionalities. It highlights strong fundamentals in logic design, file I/O, and memory management. **};**

**char Links{**

**Github:**  [**Bank Management System (**](https://github.com/Mayoneeeb/Bank_Management_System)**Github)**

**LinkedIn:** [**Project Video (LinkdIn)**](https://www.linkedin.com/posts/muneeb-zahid-192398352_banking-management-system-this-project-is-activity-7329832059168813056-IaAA?utm_source=share&utm_medium=member_desktop&rcm=ACoAAFf6U_4BUbmtxTDTnvcr8mxzXlBlWimvIiE) **};**

**if (References exist){**

**cout<< “Chatgpt ”<<endl;**

**cout<< “ Assistance in Functions like strcpy\_s,strlen,cin.ignore,time\_stamp,time,localtime,strftime were taken from chatgpt”<<endl;**

**cout<< “The code was organized and comments were also added by chatgpt”.**

**}**

**Note:** The headings used in this report, such as 'Functions', are included solely to match the Visual Studio theme. They serve only as a stylistic choice and do not represent actual Visual Studio functions or features.