CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

UNIVERSITY INSTITUTE OF COMPUTING

PROJECT REPORT

ON

BANKING MANAGEMENT SYSTEM

Program Name: BSC

Subject Name/Code: Data Structures(23CAT-201)

Submitted by:

Name: Franklin Francis
UID: 23BSC 10079
Section: BSC– 1 “A”

Submitted to:

Name: Parwan Singh
Designation:

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

Table of Contents

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

Aim

The aim of this project is to design and implement a simple yet efficient Banking Management System using the C++ programming language. The system is intended to perform essential operations such as creating, displaying, searching, deleting, depositing into, and withdrawing from bank accounts. The project focuses on building a miniature model of how real-world banking software manages customer records, applying core programming logic, and using the linked list data structure for dynamic data handling.

The goal is to combine fundamental C++ principles — including structures, functions, pointers, and dynamic memory management — into a single, functional system that allows a student to practically experience how data organization and user interaction work in a banking context.

Objective

- To apply data structure concepts, particularly linked lists, for managing dynamic customer records within a banking system.
- To design a menu-driven console application that allows users to interactively perform common banking operations like adding, deleting, and updating accounts.
- To practice modular programming by dividing the entire program into independent functions such as addAccount(), deleteAccount(), depositMoney(), and others.
- To develop a program that emphasizes real-world logic and usability, demonstrating how core C++ features translate into practical problem-solving.
- To understand memory allocation and deallocation through pointer-based structures, ensuring efficient use of system resources.
- To enhance error handling and debugging skills by testing operations under different conditions such as insufficient balance, invalid input, or empty data structures.
- To strengthen conceptual understanding of how data storage and retrieval occur in real-world financial software using linked data models.
- To encourage an appreciation for software design discipline, including planning, analysis, implementation, and testing phases.

Theory

The project is built upon fundamental principles of data structures, with an emphasis on linked lists.

In contrast to static arrays, linked lists provide dynamic memory allocation — enabling flexible addition and deletion of records during program execution.

Each account is represented by a node containing details and a pointer linking to the next node.

Advantages of Using Linked Lists

1. Dynamic Memory Allocation: Linked lists allocate memory at runtime.
2. Efficient Insertion and Deletion: Adding or removing nodes requires only pointer adjustments.
3. Flexible Size Management: The system grows or shrinks depending on account count.
4. Efficient Traversal and Better Memory Utilization.

Disadvantages of Linked Lists

1. Sequential Access only, no random indexing.
2. Memory Overhead due to pointer storage.
3. Complex reverse traversal and potential pointer errors.

Application of Linked Lists in the Banking System

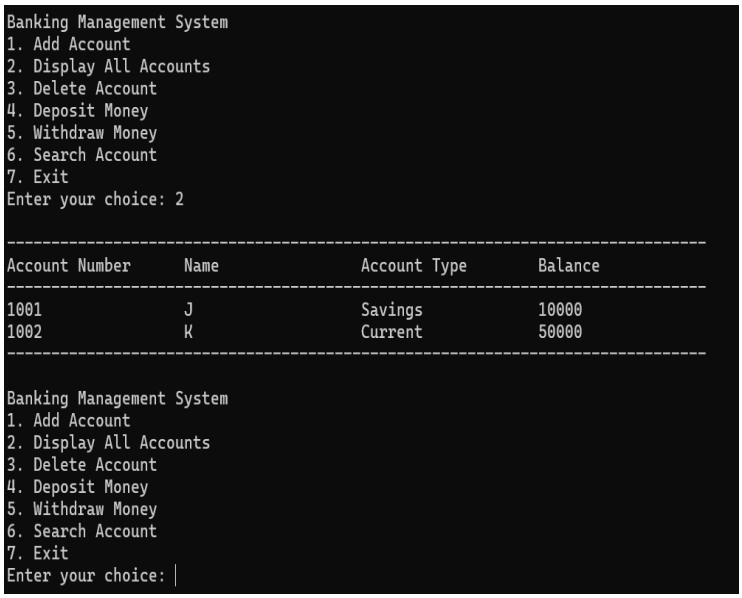
1. Account Storage and Dynamic Handling.
2. Efficient Updates for deposit/withdraw operations.
3. Searching and Displaying Accounts.
4. Memory management through node deallocation.
5. Simulating real-world relational data connections.

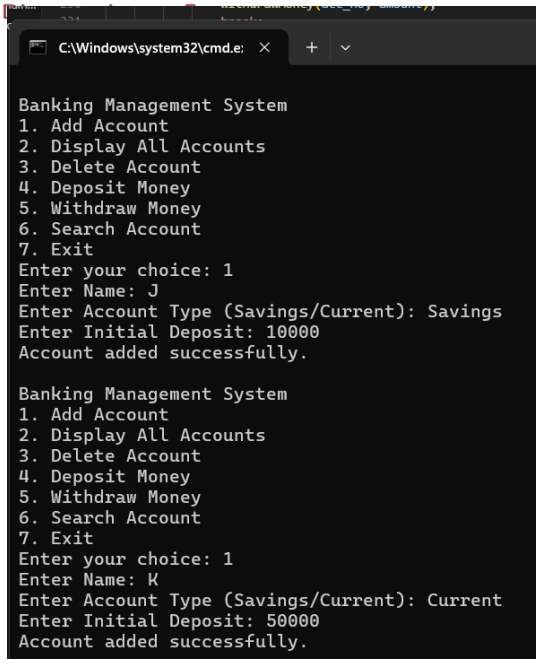
Methods

1. **Problem Identification**: Recognizing the need for essential banking operations such as account creation, deposit, withdrawal, and search.
2. **System Design**: Defining the structure Account with fields and pointers.
3. **Algorithm Design**: Preparing step-by-step logic for each operation.
4. **Implementation**: Translating algorithms into C++ using modular functions.
5. **Testing and Debugging**: Running test cases and resolving pointer issues.
6. **Validation**: Ensuring data accuracy in all transactions.
7. **Output Formatting**: Using iomanip for clean tabular display.
8. **Memory Deallocation**: Cleaning up memory before termination.

Result

The Banking Management System executes successfully and fulfills all objectives. Users can easily add, delete, search, and modify account details. All transactions occur dynamically within a linked list, showcasing efficient memory and data management. The output displays formatted tables of accounts, demonstrating professional presentation and logical consistency.





Code

```
#include <iostream>

#include <iomanip>

#include <string>

using namespace std;

struct Account {

   int account_number;

   string name;

   double balance;

   string account_type;

   Account* next;

};


Account* head = NULL;

int accountCounter = 1001;


void addAccount(string name, string account_type, double initial_deposit) {

   Account* newAccount = new Account{accountCounter++, name, initial_deposit,
account_type, nullptr};


   if (head == NULL) {
```

```
        head = newAccount;
    } else {
        Account* temp = head;
        while (temp->next) {
            temp = temp->next;
        }
        temp->next = newAccount;
    }
    cout << "Account added successfully.\n";
}

void displayAccounts() {
    if (!head) {
        cout << "No accounts to display.\n";
        return;
    }

    cout << "\n--------------------------------------------------------------------------------\n";
    cout << left << setw(20) << "Account Number"
         << setw(20) << "Name"
         << setw(20) << "Account Type"
         << setw(20) << "Balance"
         << "\n";
    cout << "--------------------------------------------------------------------------------\n";
```

```
    Account* temp = head;
    while (temp) {
        cout << left << setw(20) << temp->account_number
             << setw(20) << temp->name
             << setw(20) << temp->account_type
             << setw(20) << temp->balance
             << "\n";
        temp = temp->next;
    }
    cout << "--------------------------------------------------------------------------------\n";
}

void deleteAccount(int account_number) {
    if (!head) {
        cout << "No accounts to delete.\n";
        return;
    }

    if (head->account_number == account_number) {
        Account* toDelete = head;
        head = head->next;
        delete toDelete;
        cout << "Account deleted successfully.\n";
        return;
    }
```

```
    Account* temp = head;
    while (temp->next && temp->next->account_number != account_number) {
        temp = temp->next;
    }

    if (temp->next && temp->next->account_number == account_number) {
        Account* toDelete = temp->next;
        temp->next = temp->next->next;
        delete toDelete;
        cout << "Account deleted successfully.\n";
    } else {
        cout << "Account with number " << account_number << " not found.\n";
    }
}

void depositMoney(int account_number, double amount) {
    Account* temp = head;
    while (temp) {
        if (temp->account_number == account_number) {
            temp->balance += amount;
            cout << "Deposited successfully. New balance: " << temp->balance << "\n";
            return;
        }
        temp = temp->next;
```

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

void withdrawMoney(int account_number, double amount) {
    Account* temp = head;
    while (temp) {
        if (temp->account_number == account_number) {
            if (amount > temp->balance) {
                cout << "Insufficient balance!\n";
                return;
            }
            temp->balance -= amount;
            cout << "Withdrawal successful. New balance: " << temp->balance << "\n";
            return;
        }
        temp = temp->next;
    }
    cout << "Account not found.\n";
}

void searchAccount() {
    int searchChoice;
    cout << "Search Account By:\n";
    cout << "1. Account Number\n";
```

```
        cout << "2. Name\n";
    cout << "Enter choice: ";
    cin >> searchChoice;

    if (searchChoice == 1) {
        int account_number;
        cout << "Enter Account Number: ";
        cin >> account_number;

        Account* temp = head;
        while (temp) {
            if (temp->account_number == account_number) {
                cout << "\nAccount Number: " << temp->account_number
                     << "\nName: " << temp->name
                     << "\nAccount Type: " << temp->account_type
                     << "\nBalance: " << temp->balance << "\n";
                return;
            }
            temp = temp->next;
        }
        cout << "Account not found.\n";
    } else if (searchChoice == 2) {
        string name;
        cout << "Enter Account Holder Name: ";
        cin.ignore();
```

```
getline(cin, name);

    Account* temp = head;
    bool found = false;
    while (temp) {
      if (temp->name == name) {
        cout << "\nAccount Number: " << temp->account_number
          << "\nName: " << temp->name
          << "\nAccount Type: " << temp->account_type
          << "\nBalance: " << temp->balance << "\n";
        found = true;
      }
      temp = temp->next;
    }
    if (!found)
      cout << "Account with name \"" << name << "\" not found.\n";
  } else {
    cout << "Invalid choice. Returning to main menu.\n";
  }
}

void freeMemory() {
  Account* temp = head;
  while (temp) {
    Account* next = temp->next;
```

```
        delete temp;
        temp = next;
    }
    head = NULL;
}

int main() {
    int choice;
    string name, account_type;
    double amount;

    for (;;) {
        cout << "\nBanking Management System\n";
        cout << "1. Add Account\n";
        cout << "2. Display All Accounts\n";
        cout << "3. Delete Account\n";
        cout << "4. Deposit Money\n";
        cout << "5. Withdraw Money\n";
        cout << "6. Search Account\n";
        cout << "7. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
```

```
                cout << "Enter Name: ";
                cin.ignore();
                getline(cin, name);
                cout << "Enter Account Type (Savings/Current): ";
                getline(cin, account_type);
                cout << "Enter Initial Deposit: ";
                cin >> amount;
                addAccount(name, account_type, amount);
                break;

            case 2:
                displayAccounts();
                break;

            case 3:
                int acc_no;
                cout << "Enter Account Number to delete: ";
                cin >> acc_no;
                deleteAccount(acc_no);
                break;

            case 4:
                cout << "Enter Account Number: ";
                cin >> acc_no;
                cout << "Enter Amount to Deposit: ";
```

```
            cin >> amount;
            depositMoney(acc_no, amount);
            break;

        case 5:
            cout << "Enter Account Number: ";
            cin >> acc_no;
            cout << "Enter Amount to Withdraw: ";
            cin >> amount;
            withdrawMoney(acc_no, amount);
            break;

        case 6:
            searchAccount();
            break;

        case 7:
            cout << "Exiting system.\n";
            freeMemory();
            return 0;

        default:
            cout << "Invalid choice. Please try again.\n";
    }
}
```

Learning Outcomes

Through this project,

- A strong understanding of linked lists and dynamic memory management was achieved.
- Students learned pointer manipulation, modular programming, algorithm design, and debugging skills while bridging theoretical learning with practical implementation.

Areas of Improvement and Future Aspects

- File Handling: Add data persistence through file I/O.
- User Authentication: Implement secure login systems.
- Error Handling: Introduce exceptions for invalid inputs.
- GUI: Develop a graphical interface for better user experience.
- Database Connectivity: Use MySQL or SQLite for scalable data storage.
- Networking Features: Enable online transaction simulation.
- Interest and Loan Calculation Modules.
- Transaction Logs for history tracking.
- Data Validation and Scalability Improvements.

Conclusion

The Banking Management System using C++ effectively demonstrates modular programming, dynamic memory, and linked list implementation. It transforms theoretical knowledge into a functioning system and serves as a solid foundation for more advanced software projects, reinforcing key principles of problem-solving and structured design.