Application of Information & Communication Technologies

Final Project Report

Group – 2

Introduction

Purpose of the Program

The purpose of this project is to develop a simple banking system using C++ that simulates basic banking functionalities, including account creation, login, deposits, withdrawals, and transfers. The system allows users to interact with their accounts in a secure, text-based interface.

Scope of the Program

This program is a simple, text-based banking application. It is designed to handle up to 100 users, storing their usernames, passwords, and balances in memory. The program includes basic operations such as account creation, money deposits, withdrawals, and transfers. Data is stored temporarily and is lost when the program terminates.

Features

- **Account Creation:**
  Users can create an account by providing a unique username and password. A default balance of 100 units is assigned upon account creation.
- **Login System:**
  Users can log in using their credentials. The system checks for the existence of the username and validates the password.
- **Transactions:**
  - **Deposit:** Users can deposit money into their account.
  - **Withdraw:** Users can withdraw money from their account, provided they have sufficient funds.
  - **Transfer:** Users can transfer money to another user's account, after confirming the password for security.
- **User Search:**
  Users can search for other users by username to check if they exist in the system.
- **Session Management:**
  Users can log in, perform transactions, and log out. A confirmation prompt is displayed before logging out.

Code Structure

Global Variables:

- **USERNAMES:** A 2D array to store the usernames of all users.

- **PASSWORDS:** A 2D array to store the passwords of users.
- **BALANCES:** An array to store the balance of each user.
- **TOTAL_USERS:** A counter to track the number of registered users.

Functions:

- **Utility Functions:**
  - clearScreen(): Clears the console screen for a better user experience.
  - copyString(): Copies a string from the input to the array at a specific index.
  - compareStrings(): Compares two strings character by character.
  - findUser(): Searches for a user by their username.
- **Input Handling Functions:**
  - inputUsername(): Prompts the user to enter their username.
  - inputPassword(): Prompts the user to enter their password.
- **Core Operations:**
  - createAccount(): Registers a new user.
  - loginUser(): Authenticates an existing user.
  - deposit(), withdraw(), transfer(): Handles financial transactions between users.
  - searchUser(): Allows searching for a user by their username.
- **User Interaction:**
  - Menus for logged-in and logged-out states, prompting users to choose from available actions.
  - Confirmation prompts before sensitive actions (e.g., logging out, transferring funds).

Flow of the Program

1. **Initial Menu:**
   - Users are presented with options to create an account, log in, search for users, or exit.
2. **Logged-in Menu:**
   - After a successful login, users can deposit, withdraw, transfer funds, or log out.
3. **Exit:**

    - A confirmation prompt asks users to confirm if they want to exit the program.

Design Decisions

Choice of Data Structures:

- Static arrays were used to store user data, as they provide simplicity for this small-scale application.

Password Security:

- Passwords are stored in plain text for simplicity, but in a real-world application, password encryption (e.g., using hashing algorithms) would be recommended for better security.

Error Handling:

- Basic error handling is implemented for invalid user input and transaction limits, but it could be expanded for better robustness.

Limitations

- **No Data Persistence:**
  Data is lost when the program is closed. For persistent storage, a database or file-based solution should be implemented.
- **Scalability Issues:**
  The program is limited to 100 users due to the fixed size of arrays. This could be improved by using dynamic data structures.
- **Security:**
  The program does not encrypt passwords, and user data is stored in plain text. This would be a major security issue in a real-world application.

Improvements & Future Work

Potential Improvements:

- Implement password encryption for better security (e.g., hashing passwords).
- Introduce persistent storage to retain user data between program executions.
- Use dynamic data structures such as vectors or linked lists to handle more users efficiently.

- Improve input validation to handle edge cases (e.g., invalid characters, empty inputs).

Testing

Test Cases:

- **Account Creation:**
  Verify that accounts can be created with unique usernames.
- **Login System:**
  Ensure that users can log in with valid credentials and are rejected with invalid ones.
- **Transactions:**
  - Verify that deposits and withdrawals update the balance correctly.
  - Ensure that transfers between users are handled properly, including checking for sufficient funds.
- **User Search:**
  Confirm that the user search feature returns correct results or indicates when a user doesn't exist.

Conclusion

This banking system provides a simple yet functional demonstration of a basic banking system in C++. It allows users to manage their accounts, perform financial transactions, and interact with other users. While it meets the basic requirements, there are areas for improvement in terms of security and scalability.

Code:

```
#include<iostream>
#include<conio.h>

using namespace std;

char USERNAMES[100][256];
char PASSWORDS[100][256];
int BALANCES[100];
int TOTAL_USERS = -1;


void clearScreen(){
  #ifdef _WIN32
```

```
        system("cls");
    #else
        system("clear");
    #endif
}

// String Related Functions
void copyString(char array[][256], int index, const char* input) {
    int i = 0;
    while(input[i] != '\0' && i < 256){
        array[index][i] = input[i];
        i++;
    }
    array[index][i] = '\0';
}

bool compareStrings(const char* str1, const char* str2) {
    int i = 0;
    while (str1[i] != '\0' && str2[i] != '\0') {
        if (str1[i] != str2[i]) {
            return false;
        }
        i++;
    }
    return str1[i] == '\0' && str2[i] == '\0';
}

int findUser(const char* enteredUsername) {
    for (int i = 0; i <= TOTAL_USERS; i++) {
        if (compareStrings(USERNAMES[i], enteredUsername)) {
            return i;
        }
    }
    return -1;
}

// Input Handling Functions
char* inputUsername() {
    static char username[256];
    cout<<"Username: ";
    cin.ignore();
    cin.get(username, 256);
    return username;
}

char* inputPassword() {
    static char password[256];
    cout<<"Password: ";
    cin.ignore();
    cin.get(password, 256);
    return password;
```

```
}

bool sure(){
  bool choice = false;
  cout<<"\e[1m"<<"Are you sure!"<<"\e[0m"<<endl;
  cout<<"1. Yes"<<endl;
  cout<<"2. No"<<endl;
  cout<<"Choice: ";
  cin>>choice;

  if(choice == 1) choice = true;

  return choice;
}


// Controller Functions
int createAccount(){
  char* username = inputUsername();
  char* password = inputPassword();

  if(findUser(username) != -1){
    cout<<"Username is already in use"<<endl;
    cout<<"Press any key to continue";
    getch();
    return -1;
  }

  TOTAL_USERS++;
  int id = TOTAL_USERS;

  copyString(USERNAMES, id, username);
  copyString(PASSWORDS, id, password);
  BALANCES[id] = 0;

  return id;
}

int loginUser(){
  char* username = inputUsername();
  char* password = inputPassword();

  int id = findUser(username);
  if(id == -1){
    cout<<"User does not exists"<<endl;
    cout<<"Press any key to continue";
    getch();
    return -1;
  }

  if(compareStrings(PASSWORDS[id], password)){
    return id;
  }

  cout<<"Wrong Password"<<endl;
  cout<<"Press any key to continue";
  getch();
  return -1;
}

void searchUser(){
  cout<<"\e[1m"<<"Enter the username you want to search"<<"\e[0m"<<endl;
  char* username = inputUsername();

  int userId = findUser(username);

  if(userId == -1){
    cout<<"User doesn't exists"<<endl;
    cout<<"Press any key to continue";
    getch();
    return;
  }

  cout<<"User exists with id "<<userId<<endl;
  cout<<"Press any key to continue";
  getch();
}

void deposit(int id){
  int amount;
  cout<<"Amount: ";
  cin>>amount;

  int newBalance = BALANCES[id] + amount;
  BALANCES[id] = newBalance;
}

void withdraw(int id){
  int amount;
  cout<<"Amount: ";
  cin>>amount;

  int balance = BALANCES[id];

  if(amount > balance){
    cout<<"Amount is greater than the balance"<<endl;
    cout<<"Press any key to continue";
    getch();
    return;
  }
  int newBalance = balance - amount;
  BALANCES[id] = newBalance;
}

void transfer(int loggedInUser){
  char* username = inputUsername();
  int recipientId = findUser(username);

  if(recipientId == loggedInUser){
    cout<<"Cannot transfer to yourself"<<endl;
    cout<<"Press any key to continue";
    getch();
    return;
  }

  if(recipientId == -1){
    cout<<"User doesn't exists"<<endl;
    cout<<"Press any key to continue";
    getch();
    return;
  }

  int amount;
  cout<<"Amount: ";
  cin>>amount;

  if(amount > BALANCES[loggedInUser]){
    cout<<"Amount is greater than the balance"<<endl;
    cout<<"Press any key to continue";
    getch();
    return;
  }

  char* password = inputPassword();
  if(!compareStrings(PASSWORDS[loggedInUser], password)){
    cout<<"Wrong Password"<<endl;
    cout<<"Press any key to continue";
    getch();
  }

  BALANCES[loggedInUser] -= amount;
  BALANCES[recipientId] += amount;


  cout<<"Transfer Successful"<<endl;
  cout<<"Press any key to continue";
  getch();
}


int main(){
```

```
clearScreen();

int loggedInUser = -1;
bool exit = false;

while (!exit) {
  if(loggedInUser == -1){
    clearScreen();

    int choice;

    cout<<"\e[1m"<<"Select one option"<<"\e[0m"<<endl;
    cout<<"1. Create an account"<<endl;
    cout<<"2. Login"<<endl;
    cout<<"3. Search"<<endl;
    cout<<"4. Exit"<<endl;
    cout<<"Choice: ";
    cin>>choice;

    clearScreen();
    switch (choice)
    {
      case 1:
        loggedInUser = createAccount();
        break;
      case 2:
        loggedInUser = loginUser();
        break;
      case 3:
        searchUser();
        break;
      case 4:
        if(sure()) exit = true;
        break;

      default:
        break;
    }
  } else {

    clearScreen();

    int choice;

    cout<<"\e[1m"<<"Username: "<<"\e[0m"<<USERNAMES[loggedInUser]<<endl;
    cout<<"\e[1m"<<"Balance: "<<"\e[0m"<<BALANCES[loggedInUser]<<endl;
    cout<<"1. Deposit"<<endl;
    cout<<"2. Withdraw"<<endl;
    cout<<"3. Transfer"<<endl;
    cout<<"4. Logout"<<endl;
    cout<<"Choice: ";
```

```
            cin>>choice;

            clearScreen();
            switch (choice)
            {
               case 1:
                  deposit(loggedInUser);
                  break;
               case 2:
                  withdraw(loggedInUser);
                  break;
               case 3:
                  transfer(loggedInUser);
                  break;
               case 4:
                  if(sure()) loggedInUser = -1;
                  break;

               default:
                  break;
            }
         }

      }

      return 0;
   }
```