C Programming Project

Customer Account Banking Management System

A Complete Console-Based Banking Application

Demonstrating Advanced C Programming Concepts

Console Application
Windows Platform

powered by sherif

Banking System
Admin & User Windows

PRESENTATION ROADMAP

Contents

INTRODUCTION

Project Overview

What is This Project?

A comprehensive **console-based banking management system** built in C that simulates real-world banking operations. The system provides separate interfaces for administrators and customers to manage accounts, perform transactions, and maintain banking records.

Key Features

- Admin Authentication
- Create New Accounts
- Money Transfers
- Deposits & Withdrawals
- Account Status Control
- Password Management

Platform & Interface

Platform	Windows Console
Language	C (Standard C99)
UI Type	Text-based with Colors
Max Customers	30 Accounts

Educational Value

This project demonstrates **advanced C programming concepts** including structures, arrays, pointers, modular programming, and console I/O operations. It serves as an excellent example of how to build real-world applications using fundamental programming principles.

SYSTEM DESIGN

System Architecture & File Structure

Project Files

main.c
Entry point & main menu

sys.c / sys.h
Core system functions

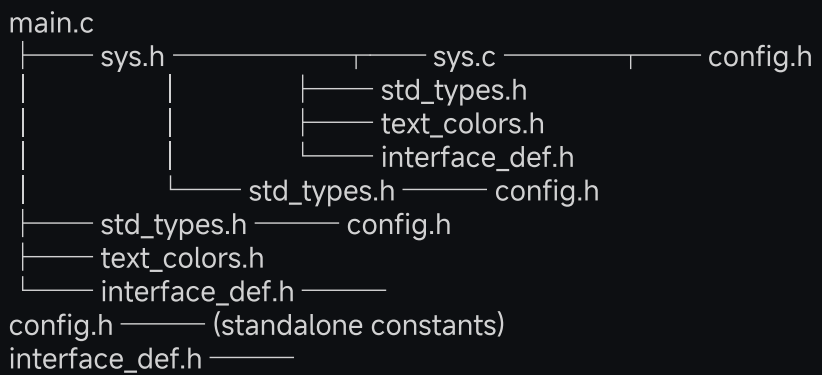
std_types.h
Custom type definitions

config.h
Configuration constants

interface_def.h
Interface definitions

text_colors.h
ANSI color codes

Module Dependencies



Architecture Pattern

Presentation Layer
Console UI & menus

Business Logic Layer
Core functions & operations

Data Layer
Structures & arrays

DATA MODELING

Data Structures: Customer Account

date_of_birth Structure

```
typedef struct {
    char day[3];
    char month[3];
    char year[5];
} date_of_birth;
```

Nested structure to store customer's date of birth. Uses character arrays to accommodate date formatting with null terminators.

Custom Type Definitions

```
typedefunsigned int       uint_32;
typedefunsigned short     uint_16;
typedefunsigned char      uint_8;
typedefunsigned long long uint_64;
typedeffloat              f_32;
typedefdouble             f_64;
```

custmer_account Structure

```
typedef struct {
 uint_8   Account_Status;
 uint_8   Age;
 f_64     Balance;
 uint_64  Bank_Account_ID;
 uint_32  Phone_Number;
 uint_64  National_ID;
 uint_8   Password[MAX_PASSWORD_SIZE];
 uint_8   Full_Name[MAX_LETTERS_NAME_NUMBER];
 uint_8   Address[MAX_LETTERS_ADDRESS_NUMBER];
 date_of_birth date;
} custmer_account;
```

Data Type Selection Rationale

uint_8: Small integers (age, status) - 1 byte

uint_16: Array indices, counters - 2 bytes

uint_32: Phone numbers - 4 bytes

uint_64: IDs, large numbers - 8 bytes

f_64: Currency/balance - 8 bytes precision

SYSTEM CONFIGURATION

Configuration & Constants

Admin Credentials

```
#define ADMIN_ID     100200
#define ADMIN_PASS   "hello@890"
```

Hardcoded admin credentials for system authentication. In production, these should be stored securely with encryption.

Default Values

```
#define ACCOUNT_DEFUALT_PASSWORD "1234"  #define DEFAULT_BALANCE
    0.0
```

Default password assigned to new accounts. Users should change this on first login for security.

Size Limits

```
#define MAX_PASSWORD_SIZE          20
#define MAX_LETTERS_NAME_NUMBER    50
#define MAX_LETTERS_ADDRESS_NUMBER 60
#define MAX_NUMBERS_PHONE_NUMBER   11
#define MAX_CUSTOMERS_NUMBER       30
```

Account ID Generation

```
#define DEFAULT_ACCOUNT_ID(NATIONAL_ID) \
    NATIONAL_ID * 3
```

Macro that generates unique bank account IDs by multiplying National ID by 3. Simple algorithm for demonstration purposes.

Example: National ID 123456 → Account ID 370368

ENTRY POINT

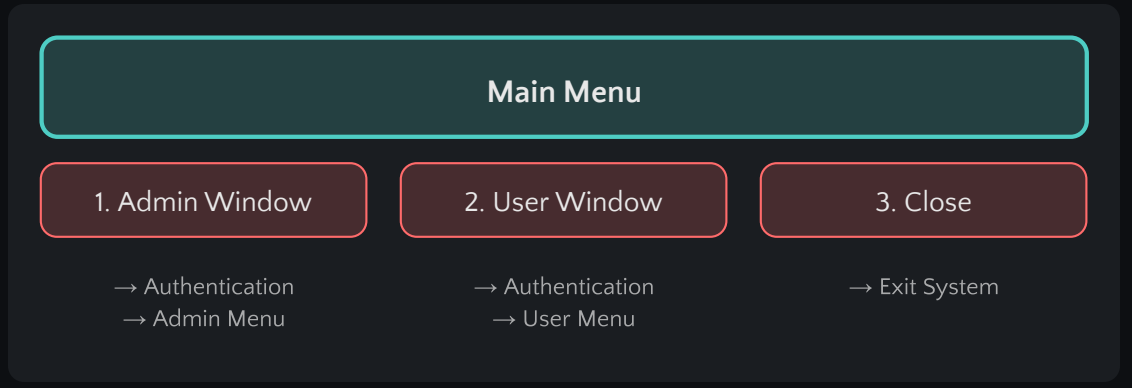
Main Menu System

Code Implementation

```
int main() {
 uint_8 choice;

 while(choice != _3rd_op) {
       printf(__yellow__"\n\n\t\t\t CUSTOMER ACCOUNT BANKING MA
NAGMENT");
       printf(__grey__"\n\t\t\t\tWELCOME TO THE MENUE\n");
       printf(__green__);
       printf("\n\n\n\t1.Admin window");
       printf("\n\t2.User window");
       printf("\n\t3.Close system");
       printf("\n\n\n\n\tEnter your choice:");
       fflush(stdin);
       scanf(" %c", &choice);
.
.
}
```

Menu Flow Diagram



Key Concepts

- **Infinite Loop:** while(1) keeps menu running until user chooses to exit
- **Switch-Case:** Clean branching based on user input
- **ANSI Colors:** text_colors.h macros for visual enhancement
- **fflush(stdin):** Clears input buffer before reading

ADMIN INTERFACE

Admin Window: Authentication & Menu

Secure Password Input

```
uint_8 i = 0;
while(i < MAX_PASSWORD_SIZE) {
   ch = getch();
 if(ch == 13) {  // Enter key
       PASS_CHECK[i] = '\0';
 break;
   } else if(ch == 8) {  // Backspaceif (i > 0) {
         i--;printf("\b \b");
       }
   } else {
       PASS_CHECK[i++] = ch;
       printf("*");
   }
}
```

Authentication Check

```
if(ID_CHECK == ADMIN_ID &&
   !strcmp(PASS_CHECK, ADMIN_PASS)) {
    // Access granted - show admin menu
} else {
    printf(__red__"\n\t\t\t id or password incorrect\n\n");
}
```

Admin Menu Options

1. **Create New Account**
   Register new customer with auto-generated ID
2. **Open Existing Account**
   Access account operations submenu
3. **Return to Main Menu**
   Exit admin window

Security Features

- **Hidden Input:** Password characters masked with asterisks
- **Backspace Support:** Users can correct typing errors
- **Dual Verification:** Both ID and password required
- **No Echo:** getch() prevents password display

ADMIN OPERATIONS

Admin Window: Create Account

Account Creation Flow

1. **Check Array Capacity**
   Verify space available
2. **Enter National ID**
   Check for duplicates
3. **Collect Customer Data**
   Name, address, DOB, phone, age
4. **Auto-Generate Fields**
   Account ID, default password, balance
5. **Display Confirmation**
   Show created account details

Duplicate Check Logic

```
for (Iteration_array = 0;
    Iteration_array < index_of_array;
    Iteration_array++) {
 if(customers[Iteration_array].National_ID
      == National_ID) {
       printf(__red__"SORRY THIS ACCOUNT IS REGISTERED");
       CHECK_CUSTOMER_FOUND_FLAG = FOUND;
 break;
```


Account Initialization

```
// Generate Bank Account ID
customers[index].Bank_Account_ID = DEFAULT_ACCOUNT_ID(customers[index].National_ID);
// Set default password
strcpy(customers[index].Password, ACCOUNT_DEFUALT_PASSWORD);
// Initialize balance
customers[index].Balance = DEFAULT_BALANCE;
// Set account status to active
customers[index].Account_Status = active;
// Increment array index
index_of_array++;
```

Key Functions Used

fgets()	Read string with spaces
strcpy()	Copy string to struct
scanf()	Read numeric values
getchar()	Clear input buffer

ADMIN OPERATIONS

Admin Window: Account Operations

Operations Submenu

1 Make Transaction
Transfer money between accounts

2 Make Withdraw
Withdraw cash from account

3 Make Deposit
Add money to account

4 Change Status
Active/Dormant/Delete

5 Back
Return to admin menu

Transfer Money Function

```
// Check if account is activeif(customers[ITS].Account_Status == active) {
// Validate amount & balanceif(customers[Iteration_array].Balance
        >= TRANS_CASH) {
         customers[Iteration_array].Balance -= TRANS_CASH;
         customers[ITS].Balance += TRANS_CASH;
         printf(__green__"MONEY TRANSFERED SUCCESSFULLY");
```

Withdraw Function

```
void make_withdrow(uint_16 idx) {
   if(customers[idx].Account_Status
      == active) {
       // Validate & deductif(customers[idx].Balance
>= cash) {
           customers[idx].Balance -= cash;
       }
   }
}
```

Deposit Function

```
void make_deposit(uint_16 idx) {
   if(customers[idx].Account_Status
      == active) {
       // Add to balance
       customers[idx].Balance += cash;
   }
}
```

USER INTERFACE

User Window Features

User Authentication

```
// Search for account by IDfor(iter_arr = 0; iter_arr < index_of_array; iter_arr++) {
 if(customers[iter_arr].Bank_Account_ID== ID_CHECK) {
       CHECK_CUSTOMER_FOUND_FLAG = FOUND;
 break;
   }
}
// Verify passwordif(!strcmp(customers[iter_arr].Password,
          PASS_CHECK)) {
 // Access granted
}
```

Users authenticate with their **Bank Account ID** and **password**. The system searches the array for matching credentials before granting access.

User Menu Options

1. **Make Transaction**
   Transfer money to another account
2. **Make Withdraw**
   Withdraw cash from account
3. **Make Deposit**
   Add money to account
4. **Change Password**
   Update account password
5. **Display Info**
   View account details
6. **Back to Main Menu**
   Exit user window

EDUCATIONAL VALUE

Key Programming Concepts Demonstrated

Data Structures

Structures
custmer_account with multiple fields

Nested Structures
date_of_birth inside custmer_account

Arrays of Structures
customers[MAX_CUSTOMERS_NUMBER]

Custom Types
typedef for uint_8, uint_16, f_64

C Language Features

Macros
#define for constants and functions

Enums
enum status {dormant, active}

Header Files
Modular organization with .h files

Include Guards
#ifndef/#define/#endif pattern

Console I/O

printf()/scanf()
Formatted output and input

getch()
Read single character without echo

fflush(stdin)
Clear input buffer

ANSI Colors
Escape sequences for colored text

Control Flow

Switch-Case
Menu navigation and option selection

While Loops
Infinite loops for menu systems

For Loops
Array iteration and searching

If-Else
Conditional logic and validation

String Handling

strcpy()
Copy strings between variables

strcmp()
Compare strings for equality

fgets()
Read strings with spaces safely

Character Arrays
Fixed-size buffers for strings

Best Practices

Input Validation
Check for negative amounts, duplicates

Error Handling
Colored error messages for clarity

Modular Design
Separate functions for each operation

Code Documentation
Comments and function headers

Thank You

Questions & Discussion

Explore the Code

Study the implementation details and understand how each component works together

Ask Questions

Don't hesitate to ask about any concepts, functions, or design decisions

Build Your Own

Use this project as a foundation to create your own banking system

Customer Account Banking Management System

A Complete C Programming Project