NODE_3

BANK-CUSTOMER-MANAGEMENT-SYSTEM

OUR TEAM

Name : Mohammed El-Ahmady
role: Project Structure & Create Customer

Name : Hatem Ayman
role: Edit & Display Customer

Name :Mayar Mohamed

role: Delete & Transfer money

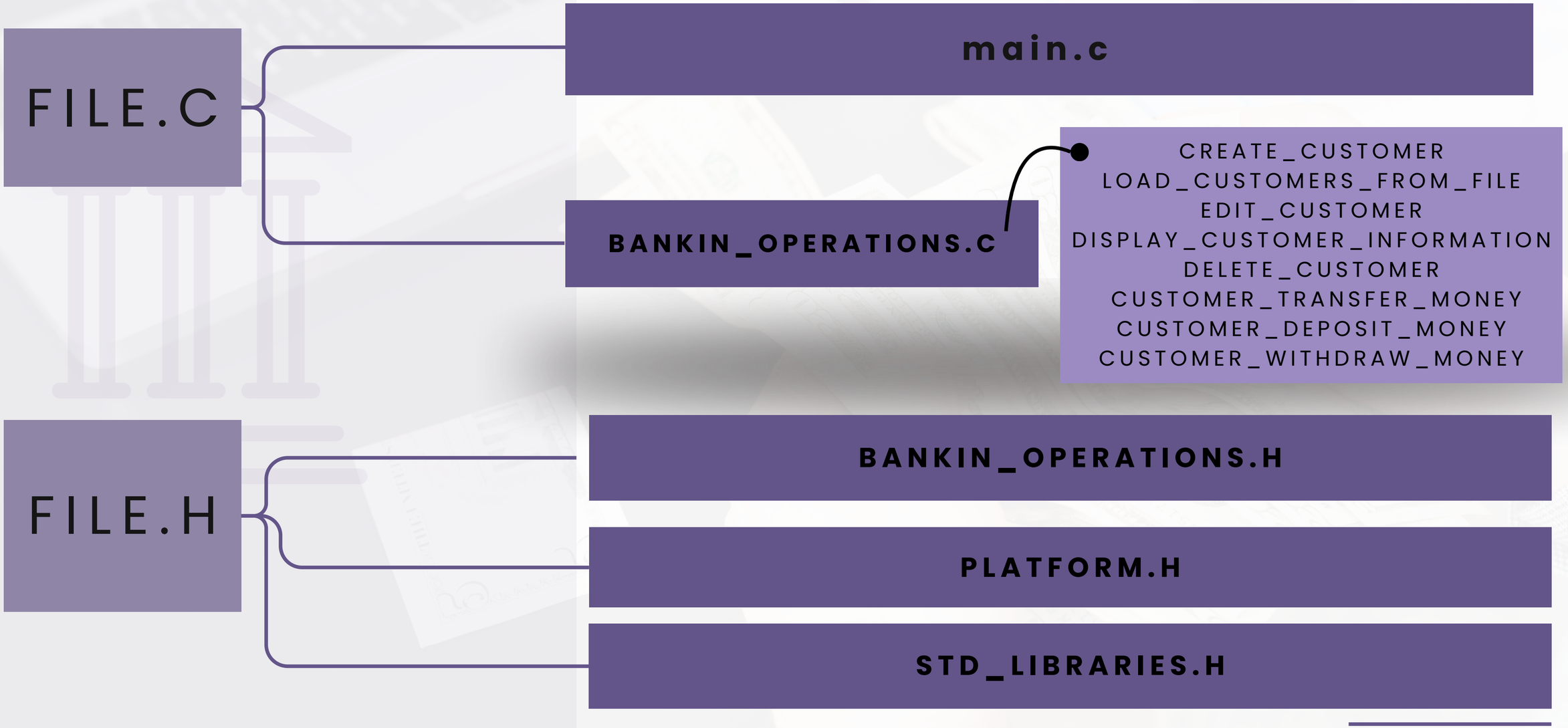
Name : A'laa Ahmed

role: Deposit & Withdraw functionalities

SYSTEM OVERVIEW

- A C-based console application simulating a bank system.
- Manages multiple customers with support for:
- Creating accounts
- Editing & deleting customers
- Deposits, withdrawals, transfers
- Data persistence through file I/O
- Simple and safe user interface using menus and input validation.

project files



PROJECT STRUCTURE

Platform File

- Standard Data Types for all project.

Error States

- All core functions return ErrorState to signal operation result.
- Facilitates error reporting in both terminal and GUI.

Customer Struct

- for store all the data about the customer
  - Name
  - ID
  - Phone Number
  - Cash Amount

System Struct

- to make a new System Includes
  - Customers array
  - Customer Validation Pointer

GUI Widgets Struct

- Adding a struct from GTK Library for
  - Window
  - Stack
  - Buttons
  - Data Entry
  - Data Labels

PLATFORM.H

```
1   typedef bool boolean;
2   typedef unsigned char uint8;
3   typedef unsigned short uint16;
4   typedef unsigned long uint32;
5   typedef unsigned long long uint64;
6
7   typedef char sint8;
8   typedef short sint16;
9   typedef long sint32;
10  typedef long long sint64;
11
12  typedef float float32;
13  typedef double float64;
14  typedef long double float128;
15
```

- Prevents redefinition using header guards.
- Defines custom type aliases for better portability.
- Improves readability of data types (e.g., uint32 instead of unsigned long).
- Centralized place to define constants like EXIT_SUCCESS.
- Prepares the project for scalability and multi-platform support.

```c
typedef enum {
    OP_Failed  = 0,
    OP_Success
} Error_States;

typedef struct {
    sint8 name[Name_Size];
    uint32 ID;
    uint32 Phone_Number;
    uint32 Cash_Amount;
} Customer;

typedef struct {
    Customer Customers[Max_Customers];
    uint32 Customer_pointer;
} System;
```

Purpose:

- Error_States : enum
  - OP_Success
  - OP_Failed

Purpose:

Customer : struct
- Name
- ID
- Phone Number
- Cash Amount

Purpose:

System : struct
- Customers
- Customer Pointer

```c
typedef struct {
    GtkWidget *window;
    GtkWidget *stack;

    // Main menu
    GtkWidget *btn_add_customer;
    GtkWidget *btn_edit_customer;
    GtkWidget *btn_display_customer;
    GtkWidget *btn_delete_customer;
    GtkWidget *btn_transfer_money;
    GtkWidget *btn_deposit;
    GtkWidget *btn_withdraw;

    // Add customer
    GtkWidget *add_name_entry;
    GtkWidget *add_id_entry;
    GtkWidget *add_phone_entry;
    GtkWidget *add_cash_entry;
    GtkWidget *add_status_label;

    // Edit customer
    GtkWidget *edit_id_entry;
    GtkWidget *edit_name_entry;
    GtkWidget *edit_phone_entry;
    GtkWidget *edit_status_label;

    // Display customer
    GtkWidget *display_id_entry;
    GtkWidget *display_info_text;

    // Delete customer
    GtkWidget *delete_id_entry;
    GtkWidget *delete_status_label;

    // Transfer money
    GtkWidget *transfer_sender_entry;
    GtkWidget *transfer_receiver_entry;
    GtkWidget *transfer_amount_entry;
    GtkWidget *transfer_status_label;

    // Deposit
    GtkWidget *deposit_id_entry;
    GtkWidget *deposit_amount_entry;
    GtkWidget *deposit_status_label;

    // Withdraw
    GtkWidget *withdraw_id_entry;
    GtkWidget *withdraw_amount_entry;
    GtkWidget *withdraw_status_label;
} AppWidgets;
```

GUI STRUCTURE :

- **Purpose:**
  - **Struct Provide a declaration for all gui Features and Elements**
    - **Windows**
    - **Stacks**
    - **Buttons**
    - **Labels**
    - **Data Entries**

```c
System *System_init(Error_States *state)
{
    System *Bank;
    *state = OP_Success;
    if (NULL == state)
    {
        *state = OP_Failed;
    }
    else
    {
        Bank = (System *)malloc(sizeof(System));
        if (NULL == Bank)
        {
            *state = OP_Failed;
        }
        else
        {
            (Bank->Customer_pointer) = System_Empty;
        }
    }
    return Bank;
}
```

Purpose:

- Initialize our Bank System with An Array of Customer and Pointer

Arguments:

- Error_States *state

Return:

- Bank : System {System Struct}

```c
Error_States Load_Customers_From_File(System *Bank, FILE *data_base)
{
    Error_States state = OP_Success;
    if (NULL == Bank || NULL == data_base)
    {
        state = OP_Failed;
    }

    rewind(data_base);

    sint8 line[1024];
    uint32 index = 0;

    while (fgets(line, sizeof(line), data_base) != NULL)
    {
        if (strncmp(line, "Customer [", 10) == 0)
        {
            // READ NAME
            if (fgets(line, sizeof(line), data_base) == NULL)
                break;
            sscanf(line, "Customer's Name : %[^\n]", Bank->Customers[index].name);

            // READ ID
            if (fgets(line, sizeof(line), data_base) == NULL)
                break;
            sscanf(line, "Customer's ID : %lu", &Bank->Customers[index].ID);

            // READ PHONE NUMBER
            if (fgets(line, sizeof(line), data_base) == NULL)
                break;
            sscanf(line, "Customer's Phone Number : %lu", &Bank->Customers[index].Phone_Number);

            // READ CASH
            if (fgets(line, sizeof(line), data_base) == NULL)
                break;
            sscanf(line, "Customer's Cash Amount : %lu", &Bank->Customers[index].Cash_Amount);

            // IGNORE LINE
            fgets(line, sizeof(line), data_base);

            index++;
        }
    }

    Bank->Customer_pointer = index;

    return (state);
}
```

Purpose:

- Load Data From The Data Base to Bank Struct that you initialize in the start of program

Arguments:

- System *bank
- FILE * database

Return:

- Error_States: enum {OP_Success, OP_Failed}

FUNCTION PURPOSE

Create customer:

- This function allows the user to create a new customer by entering name, ID, phone number, and initial cash.
- It checks for validity and uniqueness, then stores the data in memory and saves it to the database file.

Edit customer data :

- This function loads all customer data from the file data_base.txt and stores it in the system's memory structure (System *Bank) when the program starts.

Delete customer:

- This function removes a customer from the system based on their ID, updates the internal array, and rewrites the updated list into the database file.

Deposit:

- This function allows a customer to deposit a specific amount of money into their account by providing their unique ID.

Load customer from file:

- This function loads all customer data from the file data_base.txt and stores it in the system's memory structure (System *Bank) when the program starts.

display customer information:

- This function allows the user to display full information about a specific customer by entering their ID.

Transfer money:

- This function transfers money between two customers using their unique IDs.
- It updates their balances in memory and saves the result back to the file.

Withdraw:

- This function allows a customer to withdraw a specific amount from their account using their ID. It updates the balance in memory and in the file.

```
    }
    else
    {
        print_line();
        print_centered("Creating a New Customer Account");
        print_line();

        // 1. Name
        print_centered("Please enter the customer's full name:");
        printf("|%*s", (MENU_WIDTH - 1) / 2, "");
        getchar();
        fgets(Bank->Customers[Bank->Customer_pointer].name,
              sizeof(Bank->Customers[Bank->Customer_pointer].name), stdin);

        uint32 len = strlen(Bank->Customers[Bank->Customer_pointer].name);
        if (len > 0 && Bank->Customers[Bank->Customer_pointer].name[len - 1] == '\n')
        {
            Bank->Customers[Bank->Customer_pointer].name[len - 1] = '\0';
        }

        // 2. ID
        print_centered("Please enter the customer's unique ID:");
        printf("|%*s", (MENU_WIDTH - 1) / 2, "");
        scanf("%lu", &(Bank->Customers[Bank->Customer_pointer].ID));

        if ((Bank->Customers[Bank->Customer_pointer].ID) >= Max_Customers ||
            !Check_Repeated_Customer_ID(Bank, Bank->Customers[Bank->Customer_pointer].ID))
        {
            print_centered("The entered ID is either invalid or already exists.");
            state = OP_Failed;
        }

        // 3. Phone Number
        print_centered("Please enter the customer's phone number:");
        printf("|%*s", (MENU_WIDTH - 1) / 2, "");
        scanf("%lu", &(Bank->Customers[Bank->Customer_pointer].Phone_Number));

        if (!valid_phone_number(Bank->Customers[Bank->Customer_pointer].Phone_Number) ||
            !Check_Repeated_Customer_Phone_Number(Bank, Bank->Customers[Bank-
>Customer_pointer].Phone_Number))
        {
            print_centered("The phone number entered is invalid or already exists.");
            state = OP_Failed;
        }

        // 4. Cash Amount
        print_centered("Please enter the initial cash amount:");
        printf("|%*s", (MENU_WIDTH - 1) / 2, "");
        scanf("%lu", &(Bank->Customers[Bank->Customer_pointer].Cash_Amount));
        getchar();

        if (state == OP_Success)
        {
            // Write to database
            fprintf(data_base, "Customer [%lu]\n", (Bank->Customer_pointer) + 1);
            fprintf(data_base, "Customer's Name : %s\n", Bank->Customers[Bank->Customer_pointer].name);
            fprintf(data_base, "Customer's ID : %06lu\n", Bank->Customers[Bank->Customer_pointer].ID);
            fprintf(data_base, "Customer's Phone Number : 0%010lu\n", Bank->Customers[Bank-
>Customer_pointer].Phone_Number);
            fprintf(data_base, "Customer's Cash Amount : %lu\n", Bank->Customers[Bank-
>Customer_pointer].Cash_Amount);
            fprintf(data_base, "--------------------------------------------------------------------------
```

Create customer:

- **Purpose:**

  Enables the customer to update either their name or phone number through ID

Arguments:

- System *bank
-FILE * database
-uint32 id

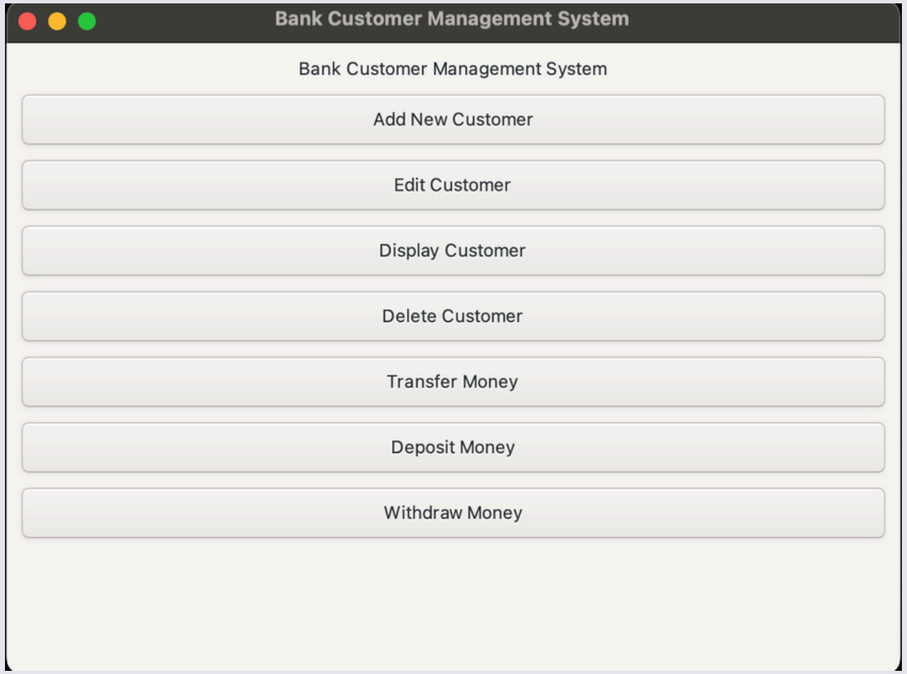
Return:

Error_States: enum {OP_Success, OP_Failed}

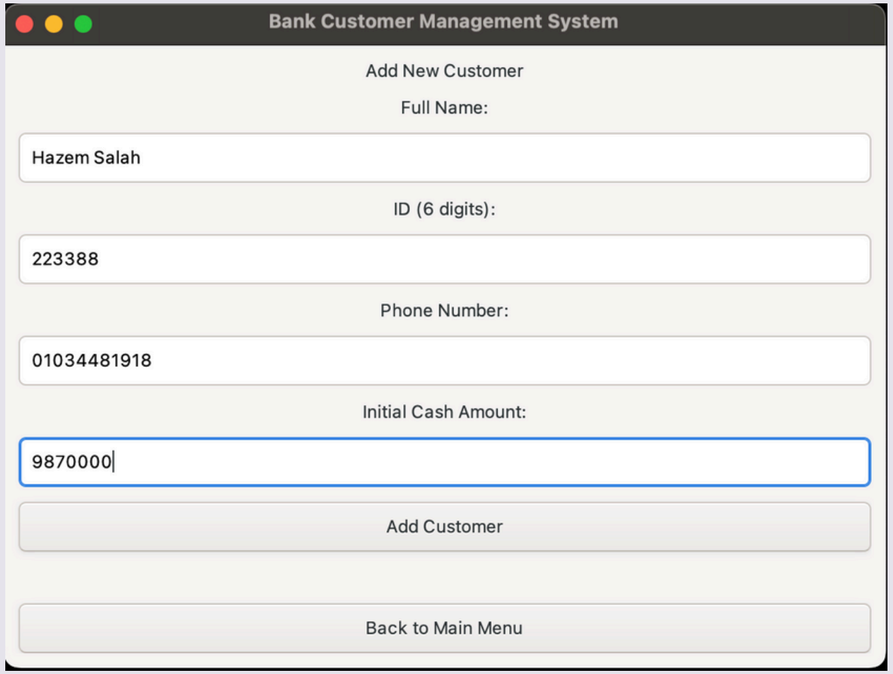
OUTPUT

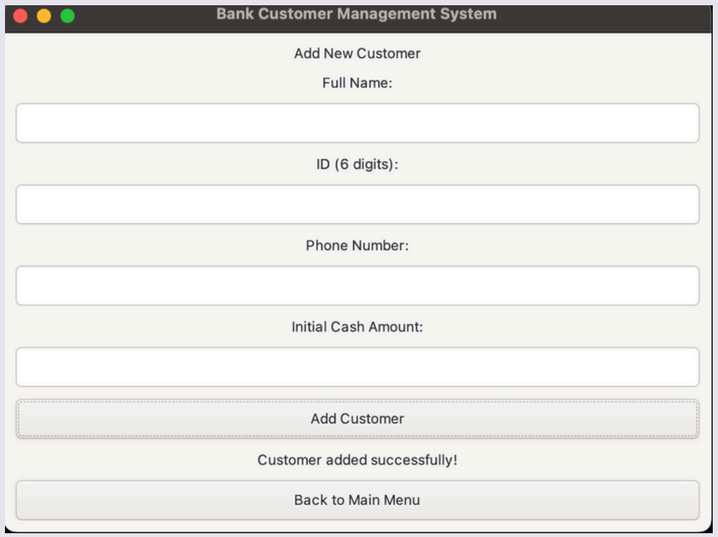
```
Please select the desired operation:
To add a new customer, press '1'
To edit customer information, press '2'
To display customer information, press '3'
To delete customer information, press '4'
To transfer funds between customers, press '5'
To deposit money, press '6'
To withdraw money, press '7'
To exit the system, press '8'
1
Creating new customer...
Creating a New Customer Account
Please enter the customer's full name:
Hatem Ayman
Please enter the customer's unique ID:
986234
Please enter the customer's phone number:
01098273456
Please enter the initial cash amount:
9870000
Customer created successfully.
```

```
Customer [11]
Customer's Name : Hatem Ayman
Customer's ID : 986234
Customer's Phone Number : 01098273456
Customer's Cash Amount : 9870000
```



Deep Dive with GUI






```
Customer [103]
Customer's Name : Hazem Salah
Customer's ID : 223388
Customer's Phone Number : 01034481918
Customer's Cash Amount : 9870000
--------------------------------------------------------------------
```

```
Error_States Edit_Customer(System *Bank, FILE *data_base, uint32 id) {

    Error_States state = OP_Success;

    if (NULL == Bank || NULL == data_base) {
        state = OP_Failed;
    }
    else {
        sint32 index = find_customer_id(Bank, id);

        print_line();
        print_centered("Edit Customer Information");
        print_line();

        if (index == -1) {
            char msg[100];
            snprintf(msg, sizeof(msg), "Customer with ID %lu not found.", id);
            print_centered(msg);
            state = OP_Failed;
        } else {
            print_centered("Please update the customer's name:");
            printf("|%*s", (MENU_WIDTH - 1) / 2, "");
            getchar();
            fgets(Bank->Customers[index].name, sizeof(Bank->Customers[index].name), stdin);

            uint32 len = strlen(Bank->Customers[index].name);
            if (len > 0 && Bank->Customers[index].name[len - 1] == '\n') {
                Bank->Customers[index].name[len - 1] = '\0';
            }

            print_centered("Please update the customer's phone number:");
            printf("|%*s", (MENU_WIDTH - 1) / 2, "");
            scanf("%lu", &(Bank->Customers[index].Phone_Number));

            if (!valid_phone_number(Bank->Customers[index].Phone_Number) ||
                !Check_Repeated_Customer_Phone_Number(Bank, Bank->Customers[Bank-
>Customer_pointer].Phone_Number)) {
                print_centered("The phone number entered is invalid or already exists.");
                state = OP_Failed;
            } else {
                state = Data_Base_Overwrite_data(Bank, data_base);
                if (state == OP_Success) {
                    print_centered("Customer data has been successfully updated.");
                } else {
                    print_centered("Failed to update customer data.");
                }
            }
        }

        print_line();
    }

    return state;
}
```

Edit customer data :

Purpose:

Enables the customer to update either their name or phone number through ID

Arguments:

- System *bank
- FILE * database
- uint32 id

Return:

Error_States: enum {OP_Success, OP_Failed}

Before

```
--------------------------------------------------------------------------------
Customer [11]
Customer's Name : Hatem Ayman
Customer's ID : 986234
Customer's Phone Number : 01098273456
Customer's Cash Amount : 9870000
--------------------------------------------------------------------------------
```

```
Please select the desired operation:
To add a new customer, press '1'
To edit customer information, press '2'
To display customer information, press '3'
To delete customer information, press '4'
To transfer funds between customers, press '5'
To deposit money, press '6'
To withdraw money, press '7'
To exit the system, press '8'
2
Enter the customer ID to modify their data:
986234
Edit Customer Information
Please update the customer's name:
Mahmoud Ibrahim
Please update the customer's phone number:
01031249845
Customer data has been successfully updated.
```

```
--------------------------------------------------------------------------------
Customer [11]
Customer's Name : Mahmoud Ibrahim
Customer's ID : 986234
Customer's Phone Number : 01031249845
Customer's Cash Amount : 9870000
--------------------------------------------------------------------------------
```

After

```
Error_States Edit_Customer(System *Bank, FILE *data_base, uint32 id) {

    Error_States state = OP_Success;

    if (NULL == Bank || NULL == data_base) {
        state = OP_Failed;
    }
    else {
```

Error handling for null pointer to avoid unexpected behavior

```
    else {
        sint32 index = find_customer_id(Bank, id);

        print_line();
        print_centered("Edit Customer Information");
        print_line();

        if (index == -1) {
            char msg[100];
            snprintf(msg, sizeof(msg), "Customer with ID %lu not found.", id);
            print_centered(msg);
            state = OP_Failed;
        } else {
```

```
static sint32 find_customer_id(System *Bank, uint32 target_id)
{
    sint32 index = 0;
    for (uint32 i = 0; i <= Bank->Customer_pointer; i++)
    {
        if (target_id == Bank->Customers[i].ID)
        {
            index = i;
            break;
        }
        else
        {
            index = -1;
        }
    }
    return (index);
}
```

In proceed, the edit function calls find_customer helper function to check if the user already exist in the System array

```
        } else {
            print_centered("Please update the customer's name:");
            printf("|%*s", (MENU_WIDTH - 1) / 2, "");
            getchar();
            fgets(Bank->Customers[index].name, sizeof(Bank->Customers[index].name), stdin);

            uint32 len = strlen(Bank->Customers[index].name);
            if (len > 0 && Bank->Customers[index].name[len - 1] == '\n') {
                Bank->Customers[index].name[len - 1] = '\0';
            }

            print_centered("Please update the customer's phone number:");
            printf("|%*s", (MENU_WIDTH - 1) / 2, "");
            scanf("%lu", &(Bank->Customers[index].Phone_Number));

            if (!valid_phone_number(Bank->Customers[index].Phone_Number) ||
                !Check_Repeated_Customer_Phone_Number(Bank, Bank->Customers[Bank-
>Customer_pointer].Phone_Number)) {
                print_centered("The phone number entered is invalid or already exists.");
                state = OP_Failed;
            } else {
```

```
static bool valid_phone_number(uint32 num)
{
    bool flag = true;
    for (uint32 i = 0; i < 8; i++)
    {
        num /= 10;
    }
    if (num == 10 || num == 11 || num == 12 || num == 15)
    {
        flag = true;
    }
    else
    {
        flag = false;
    }
    return (flag);
}
```

User Input Validation

if the user is already a customer in the bank system, the program prompts the user to enter new name and new phone number

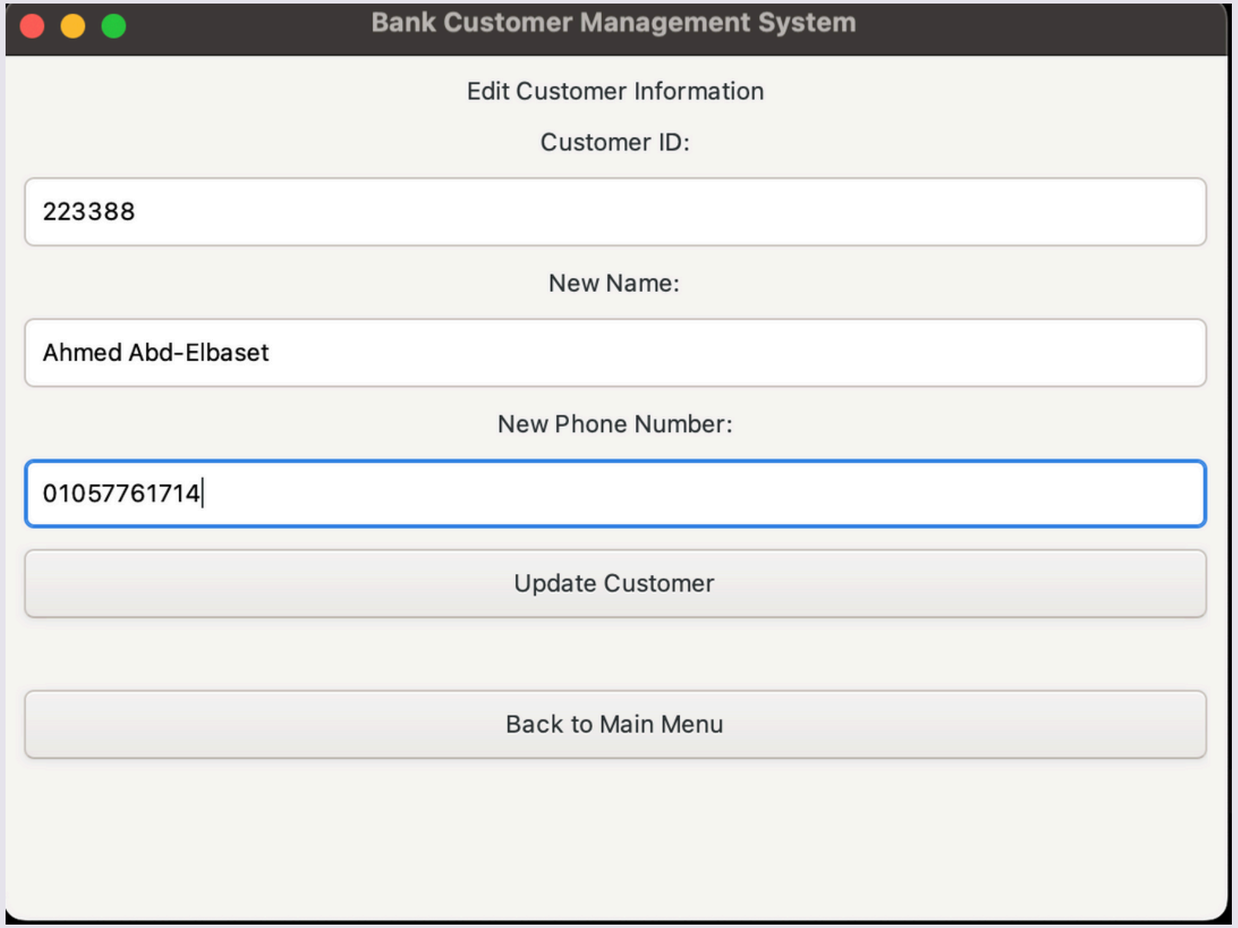
```
static bool Check_Repeated_Customer_Phone_Number(System *Bank, uint32 phone)
{
    bool flag = true;
    if (NULL == Bank)
    {
        flag = false;
    }
    else
    {
        for (uint32 i = 0; i < Bank->Customer_pointer; i++)
        {
            if (phone == Bank->Customers[i].Phone_Number)
            {
                flag = false;
                break;
            }
        }
    }
    return (flag);
}
```

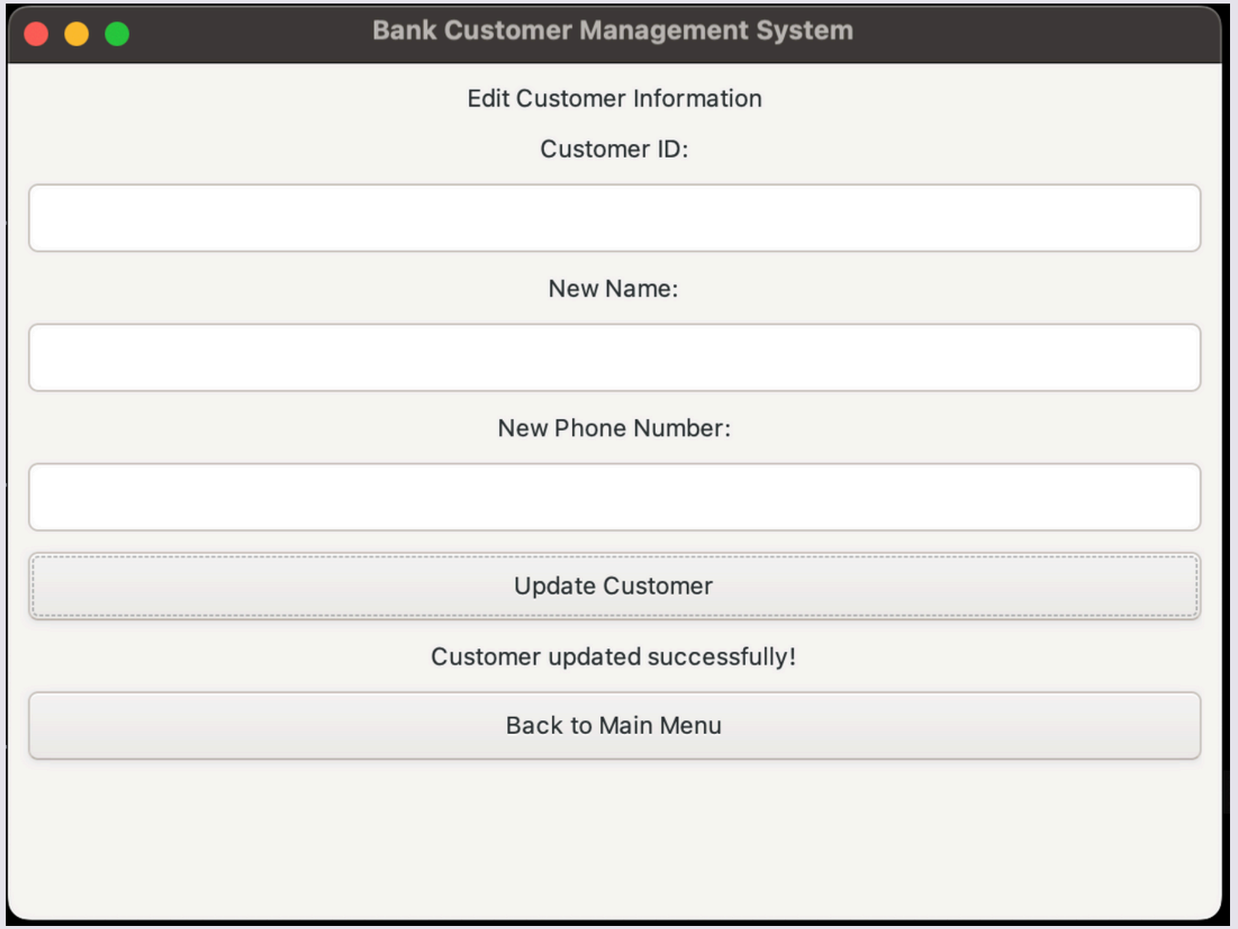
```
            } else {
                state = Data_Base_Overwrite_data(Bank, data_base);
                if (state == OP_Success) {
                    print_centered("Customer data has been successfully updated.");
                } else {
                    print_centered("Failed to update customer data.");
                }
            }
        }

        print_line();
    }

    return state;
}
```

```
static Error_States Data_Base_Overwrite_data(System *Bank, FILE *data_base)
{
    Error_States state = OP_Success;
    if (NULL == Bank || NULL == data_base)
    {
        state = OP_Failed;
    }
    else
    {
        freopen(FILE_PATH, "w", data_base);
        for (uint32 i = 0; i < Bank->Customer_pointer; i++)
        {
            fprintf(data_base, "Customer [%lu]\n", i + 1);
            fprintf(data_base, "Customer's Name : %s\n", Bank->Customers[i].name);
            fprintf(data_base, "Customer's ID : %06lu\n", Bank->Customers[i].ID);
            fprintf(data_base, "Customer's Phone Number : 0%010lu\n", Bank->Customers[i].Phone_Number);
            fprintf(data_base, "Customer's Cash Amount : %lu\n", Bank->Customers[i].Cash_Amount);
            fprintf(data_base, "--------------------------------------------------------------------------
-\n");
            fflush(data_base);
        }
    }
    return (state);
}
```

At last the edit function calls another helper function to overwrite the new data to the database file to save the new preset

Deep Dive with GUI





```c
Error_States Display_Customer_information(System *Bank, uint32 id)
{
    Error_States state = OP_Success;

    if (NULL == Bank)
    {
        state = OP_Failed;
    }
    else
    {
        sint32 index = find_customer_id(Bank, id);

        print_line();
        print_centered("Customer Information");
        print_line();

        if (index == -1)
        {
            char msg[100];
            snprintf(msg, sizeof(msg), "Customer with ID %lu not found.", id);
            print_centered(msg);
            state = OP_Failed;
        }
        else
        {
            char buffer[256];

            snprintf(buffer, sizeof(buffer),
            "Customer's Name : %s", Bank->Customers[index].name);
            print_centered(buffer);

            snprintf(buffer, sizeof(buffer),
            "Customer's ID : %06lu", Bank->Customers[index].ID);
            print_centered(buffer);

            snprintf(buffer, sizeof(buffer),
            "Phone Number : 0%010lu", Bank->Customers[index].Phone_Number);
            print_centered(buffer);

            snprintf(buffer, sizeof(buffer),
            "Cash Amount : %lu", Bank->Customers[index].Cash_Amount);
            print_centered(buffer);
        }

        print_line();
    }

    return (state);
}
```

Purpose:

Enables the customer to view a detailed report for their data: name, ID, phone number, cash amount

Arguments:

- System *bank
- FILE * database
- uint32 id

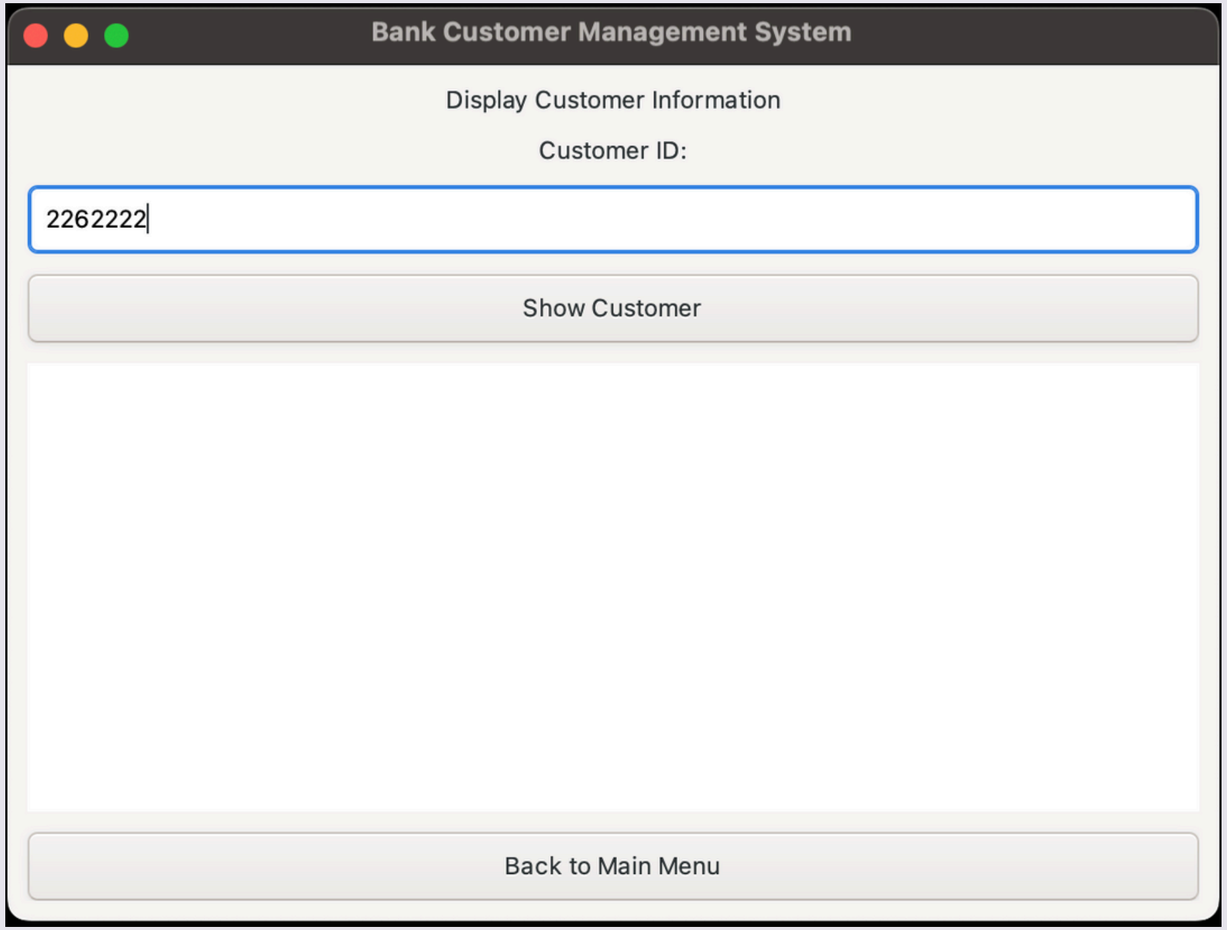
Return:

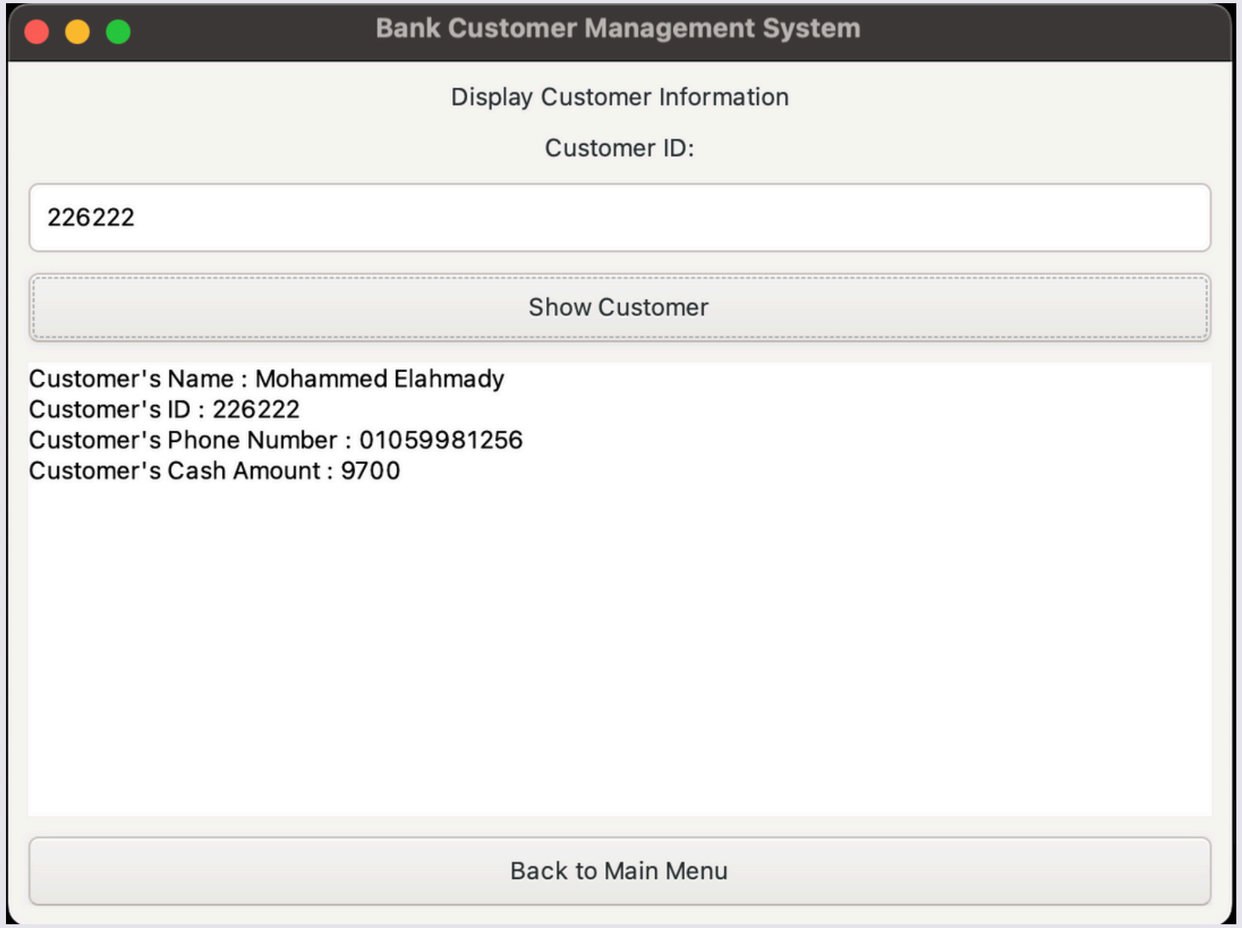
Error_States: enum {OP_Success, OP_Failed}

OUTPUT

```
Please select the desired operation:
To add a new customer, press '1'
To edit customer information, press '2'
To display customer information, press '3'
To delete customer information, press '4'
To transfer funds between customers, press '5'
To deposit money, press '6'
To withdraw money, press '7'
To exit the system, press '8'
3
Enter the customer ID to view their information:
986234
Customer Information
Customer's Name : Mahmoud Ibrahim
Customer's ID : 986234
Phone Number : 01031249845
Cash Amount : 9870000
```

Deep Dive with GUI





```c
Error_States Delete_Customer(System *Bank, FILE *data_base, uint32 id)
{
    Error_States state = OP_Success;

    if (NULL == Bank || NULL == data_base)
    {
        state = OP_Failed;
    }
    else
    {
        sint32 index = find_customer_id(Bank, id);

        print_line();
        print_centered("Delete Customer");
        print_line();

        if (index == -1)
        {
            char msg[100];
            snprintf(msg, sizeof(msg), "Customer with ID %06lu not found.", id);
            print_centered(msg);
            state = OP_Failed;
        }
        else
        {
            state = Bank_Shift_Customers(Bank, index);
            if (state == OP_Success)
            {
                Bank->Customer_pointer--;

                state = Data_Base_Overwrite_data(Bank, data_base);
                if (state == OP_Success)
                {

                    print_centered("The customer has been successfully removed from the system.");
                }
                else
                {
                    print_centered("An error occurred while removing the customer's data.");
                }
            }
            else
            {
                print_centered("An error occurred while shifting customer records.");
            }
        }

        print_line();
    }

    return (state);
}
```

Purpose:

Delete Customer from Data Base

Arguments:

- System *bank
- FILE * database
- uint32 id

Return:

Error_States: enum {OP_Success, OP_Failed}

```
Error_States Delete_Customer(System *Bank, FILE *data_base, uint32 id)
```

1. Validation Checks:

```
if (NULL == Bank || NULL == data_base)
```

Makes sure the pointers are valid.
If not, the function returns a failure.

2. Find Customer by ID:

```
sint32 index = find_customer_id(Bank, id);
```

Searches the customer list using a helper function.
Returns index if found, or -1 if not.

3. If Not Found:

```
if (index == -1)
```

Displays a centered message that the ID was not found.
Returns OP_Failed.

4. If Found → Shift and Remove:

```
state = Bank_Shift_Customers(Bank, index);
if (state == OP_Success)
```

Bank_Shift_Customers() shifts all customers after the deleted one to the left.
Decreases the pointer to reflect deletion.

5. Save Updated Data:

```
state = Data_Base_Overwrite_data(Bank, data_base);
```

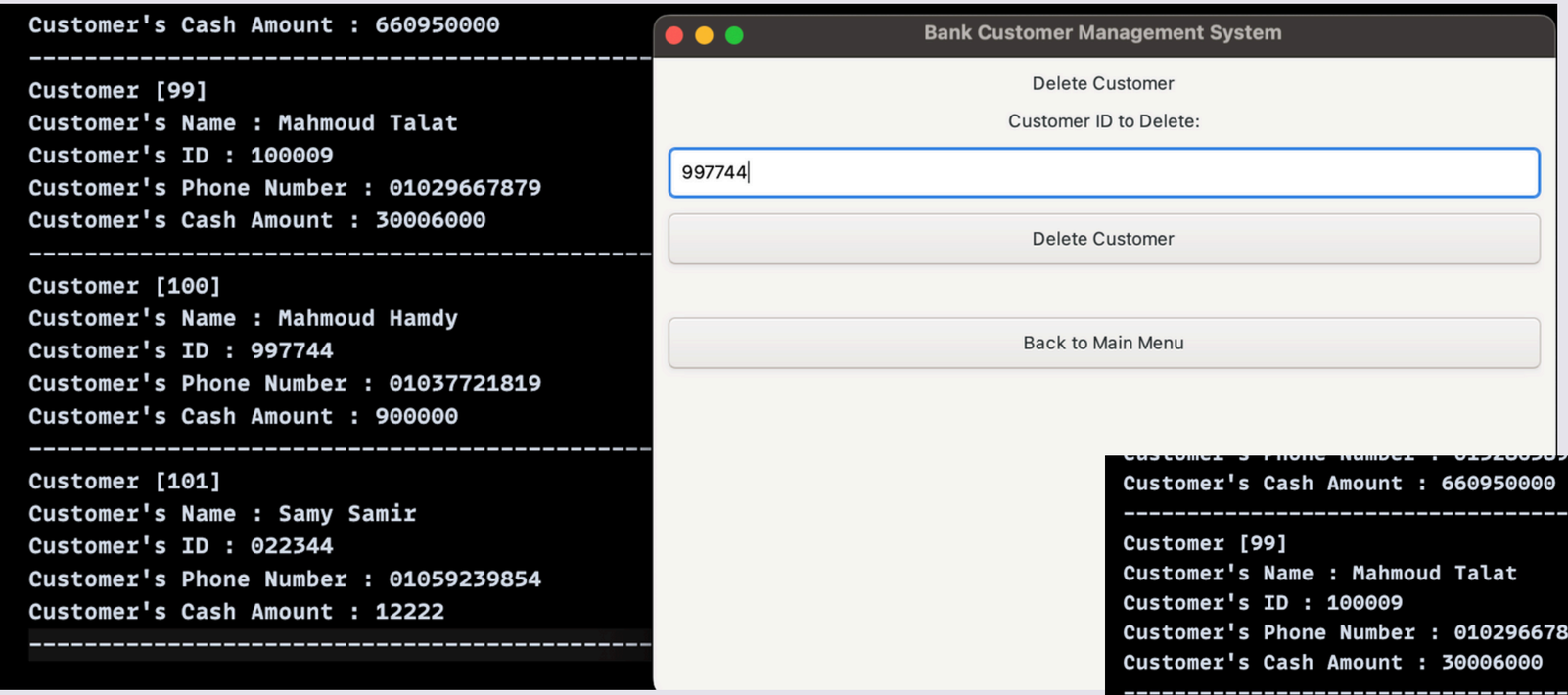
Rewrites all current customer data back to the text file using the same readable format.
If successful, prints confirmation message.

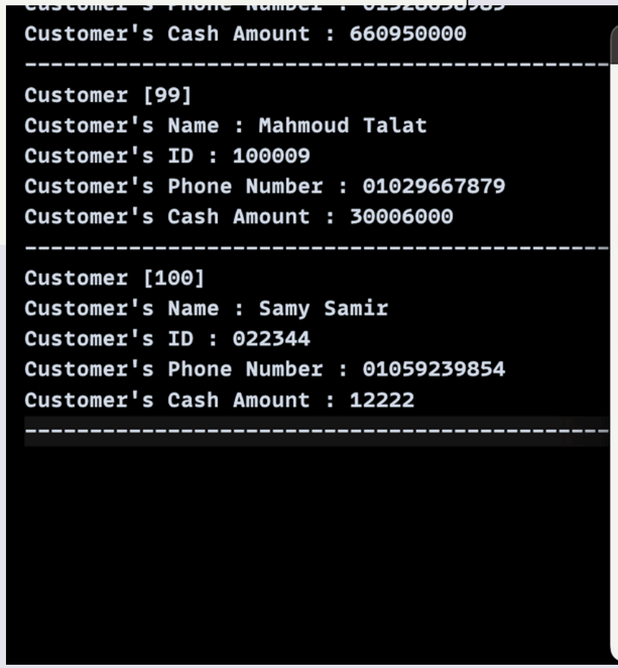
OUTPUT

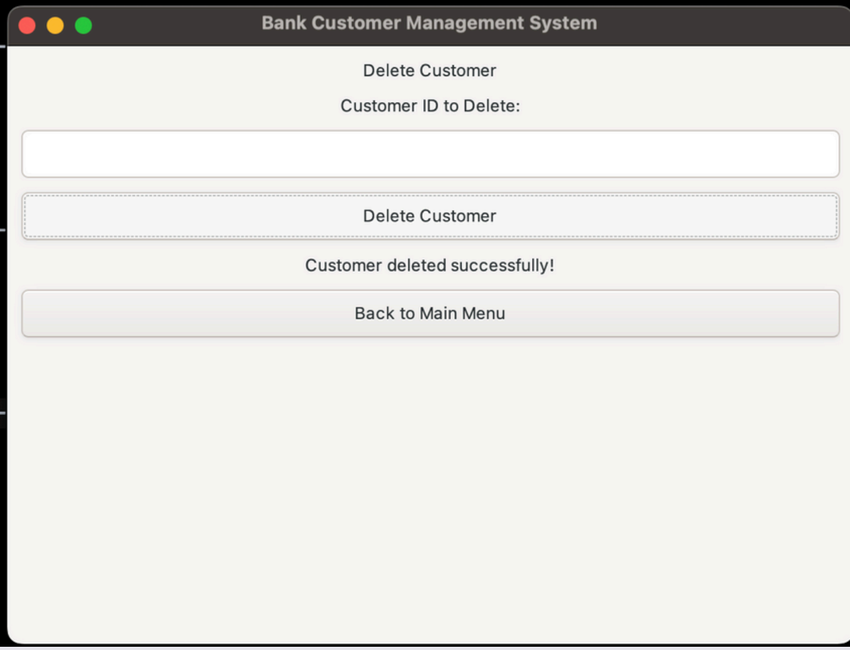
```
Please select the desired operation:
To add a new customer, press '1'
To edit customer information, press '2'
To display customer information, press '3'
To delete customer information, press '4'
To transfer funds between customers, press '5'
To deposit money, press '6'
To withdraw money, press '7'
To exit the system, press '8'
4
Enter the customer ID to remove them from the system:
986234
Delete Customer
The customer has been successfully removed from the system.
```

```
Customer [9]
Customer's Name : Saeed Samy
Customer's ID : 246123
Customer's Phone Number : 01049873322
Customer's Cash Amount : 62464
--------------------------------------------------------------------
Customer [10]
Customer's Name : Ahmed Zaki
Customer's ID : 889233
Customer's Phone Number : 01098762345
Customer's Cash Amount : 760000
--------------------------------------------------------------------
```

Deep Dive with GUI







```c
Error_States Customer_Transfer_Money(System *Bank, FILE *data_base, uint32 C_id, uint32 D_id, uint32
T_money)
{
    Error_States state = OP_Success;

    if (NULL == Bank || NULL == data_base)
    {
        state = OP_Failed;
    }
    else
    {
        sint32 C1 = find_customer_id(Bank, C_id);
        sint32 C2 = find_customer_id(Bank, D_id);

        print_line();
        print_centered("Transfer Funds Between Customers");
        print_line();

        if (C1 == -1 || C2 == -1)
        {
            print_centered("One or both customer IDs not found.");
            state = OP_Failed;
        }
        else
        {
            if (Bank->Customers[C1].Cash_Amount >= T_money)
            {
                char buffer[256];

                Bank->Customers[C1].Cash_Amount -= T_money;

                snprintf(buffer, sizeof(buffer),
                         "Funds transferred from %s (ID: %06lu): %lu",
                         Bank->Customers[C1].name,
                         Bank->Customers[C1].ID,
                         T_money);
                print_centered(buffer);

                snprintf(buffer, sizeof(buffer),
                         "Updated balance for sender: %lu",
                         Bank->Customers[C1].Cash_Amount);
                print_centered(buffer);

                Bank->Customers[C2].Cash_Amount += T_money;

                snprintf(buffer, sizeof(buffer),
                         "Funds received by %s (ID: %06lu) from %s (ID: %06lu)",
                         Bank->Customers[C2].name,
                         Bank->Customers[C2].ID,
                         Bank->Customers[C1].name,
                         Bank->Customers[C1].ID);
                print_centered(buffer);

                snprintf(buffer, sizeof(buffer),
                         "Updated balance for receiver: %lu",
                         Bank->Customers[C2].Cash_Amount);
                print_centered(buffer);

                state = Data_Base_Overwrite_data(Bank, data_base);
                if (state == OP_Success)
                {

                    print_centered("Transfer completed successfully.");
                }
                else
                {
                    print_centered("Transfer failed during data update.");
                }
            }
            else
            {
                print_centered("Insufficient balance to complete the transfer.");
                state = OP_Failed;
            }
        }

        print_line();
    }

    return (state);
}
```

- **Purpose:**
  - Transfer Money Between 2 Customers

- **Arguments:**
  - System *bank
  - FILE * database
  - uint32 C_id,D_id,T_money

- **Return:**
  - Error_States: enum {OP_Success, OP_Failed}

```
Error_States Customer_Transfer_Money(System *Bank, FILE *data_base, uint32 C_id, uint32 D_id, uint32 T_money)
```

1. Validation Checks:

```
if (NULL == Bank || NULL == data_base)
```

Makes sure the pointers are valid.
If not, the function returns a failure.

2. Find Sender and Receiver:

```
sint32 C1 = find_customer_id(Bank, C_id);
sint32 C2 = find_customer_id(Bank, D_id);
```

Uses a helper function to get the array index of both customers.

3. Check Existence

```
if (C1 == -1 || C2 == -1)
```

If either customer is not found → show error message, return failure.

4. Check Balance and Transfer

```
if (Bank->Customers[C1].Cash_Amount >= T_money)
```

Verifies if the sender has enough money.
If valid:
Deducts amount from sender. Adds amount to receiver.
Prints detailed confirmations.

5. Update File

```
state = Data_Base_Overwrite_data(Bank, data_base);
```

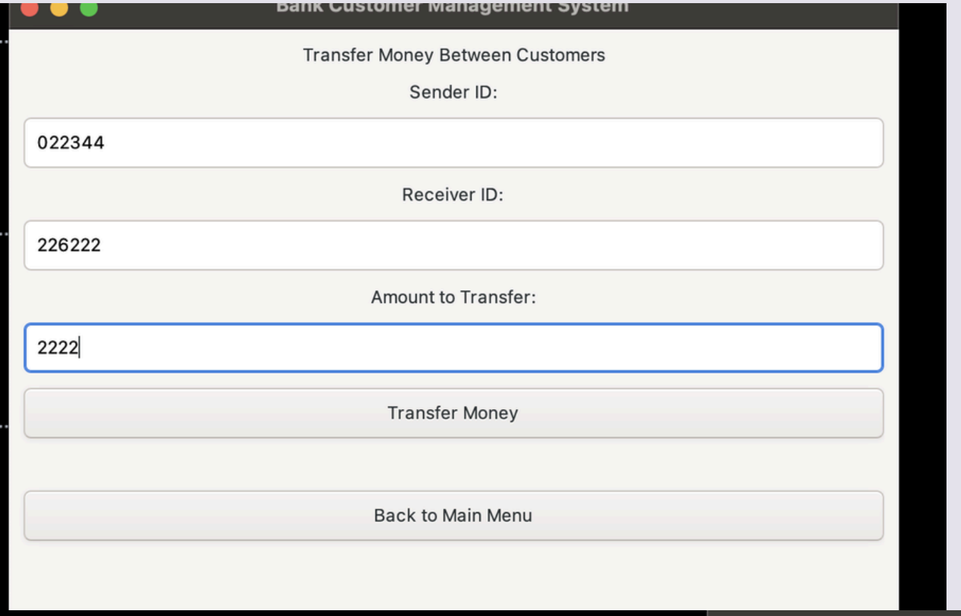
Rewrites the updated customer data to the text file.
If file write succeeds → show success message.
Else → show file error message.

OUTPUT

```
Please select the desired operation:
To add a new customer, press '1'
To edit customer information, press '2'
To display customer information, press '3'
To delete customer information, press '4'
To transfer funds between customers, press '5'
To deposit money, press '6'
To withdraw money, press '7'
To exit the system, press '8'
5
Enter the sender's customer ID:
246123
Enter the recipient's customer ID:
889233
Enter the amount you wish to transfer:
8000
Transfer Funds Between Customers
Funds transferred from Saeed Samy (ID: 246123): 8000
Updated balance for sender: 54464
Funds received by Ahmed Zaki (ID: 889233) from Saeed Samy (ID: 246123)
Updated balance for receiver: 768000
Transfer completed successfully.
```

Deep Dive with GUI

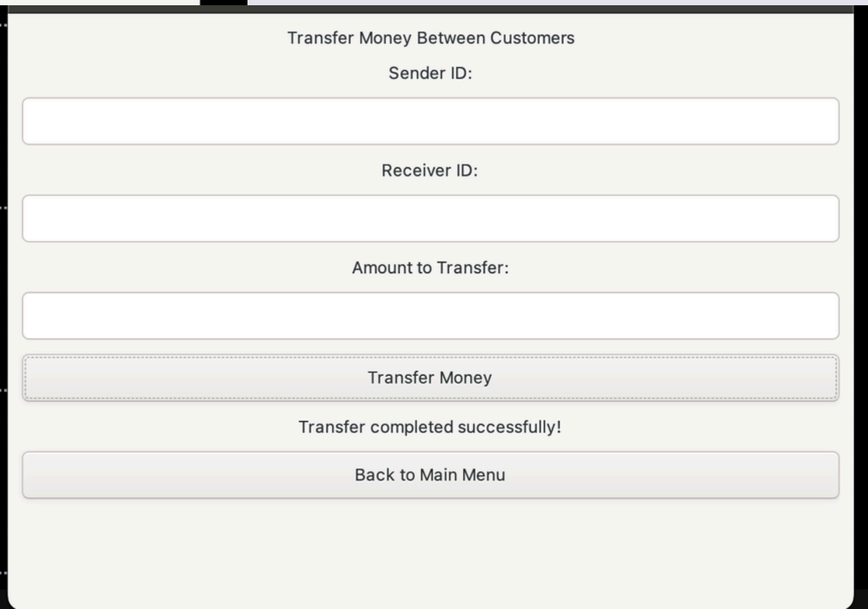
```
Customer's Cash Amount : 900000
--------------------------------------------------------------------------------
Customer [101]
Customer's Name : Samy Samir
Customer's ID : 022344
Customer's Phone Number : 01059239854
Customer's Cash Amount : 12222
--------------------------------------------------------------------------------
Customer [102]
Customer's Name : Mohammed Elahmady
Customer's ID : 226222
Customer's Phone Number : 01059981256
Customer's Cash Amount : 9700
--------------------------------------------------------------------------------
Customer [103]
Customer's Name : Ahmed Abd-Elbaset
Customer's ID : 223388
Customer's Phone Number : 01057761714
Customer's Cash Amount : 9870000
```



before

after

```
--------------------------------------------------------------------------------
Customer [101]
Customer's Name : Samy Samir
Customer's ID : 022344
Customer's Phone Number : 01059239854
Customer's Cash Amount : 10000
--------------------------------------------------------------------------------
Customer [102]
Customer's Name : Mohammed Elahmady
Customer's ID : 226222
Customer's Phone Number : 01059981256
Customer's Cash Amount : 11922
--------------------------------------------------------------------------------
Customer [103]
Customer's Name : Ahmed Abd-Elbaset
Customer's ID : 223388
Customer's Phone Number : 01057761714
Customer's Cash Amount : 9870000
--------------------------------------------------------------------------------
```



Functionality Overview

Implemented Features

Customer_Deposit_Money()

- Takes: System *Bank, FILE *data_base, uint32 id, uint32 money
- Finds the customer by ID
- Adds money to their balance
- Updates the database file

Customer_Withdraw_Money()

- Takes: Same inputs
- Checks for sufficient funds
- Deducts amount if balance allows
- Updates the file after withdrawal

System Logic (Code Flow)

Deposit / Withdraw Logic Flow

- 1. Get Customer ID → search using find_customer_index()
- 2. If not found → print error
- 3. If found:
- • Deposit: Add money
- • Withdraw: Check balance → subtract money
- 4. Call Data_Base_Overwrite_data() to save changes

Error Handling:

- Invalid ID → return OP_Failed
- Insufficient balance → return OP_Failed with message

Deposit Function

```
sint32 index = find_customer_id(Bank, id)
```

Locate the customer in the system by ID;

```
if (index == -1)
```

If customer not found → print error, return OP_Failed.

```
Bank->Customers[index].Cash_Amount += money;
```

Add the deposited money to customer's balance.

```
        state = Data_Base_Overwrite_data(Bank, data_base);
    }
}
return (state);
```

Save the updated data to file.

Withdraw Function

```
sint32 index = find_customer_id(Bank, id);
```

Find customer index using helper.

```
if (index == -1)
```

If ID is invalid → print error, return failure.

```
if (Bank->Customers[index].Cash_Amount < money)
{
    state = OP_Failed;
```

Insufficient balance → show error, return failure.

```
Bank->Customers[index].Cash_Amount -= money;
```

Subtract amount from balance.

```
            state = Data_Base_Overwrite_data(Bank, data_base);
        }
    }
}
return (state);
```

Save updated data to the file.

Sample Terminal Output

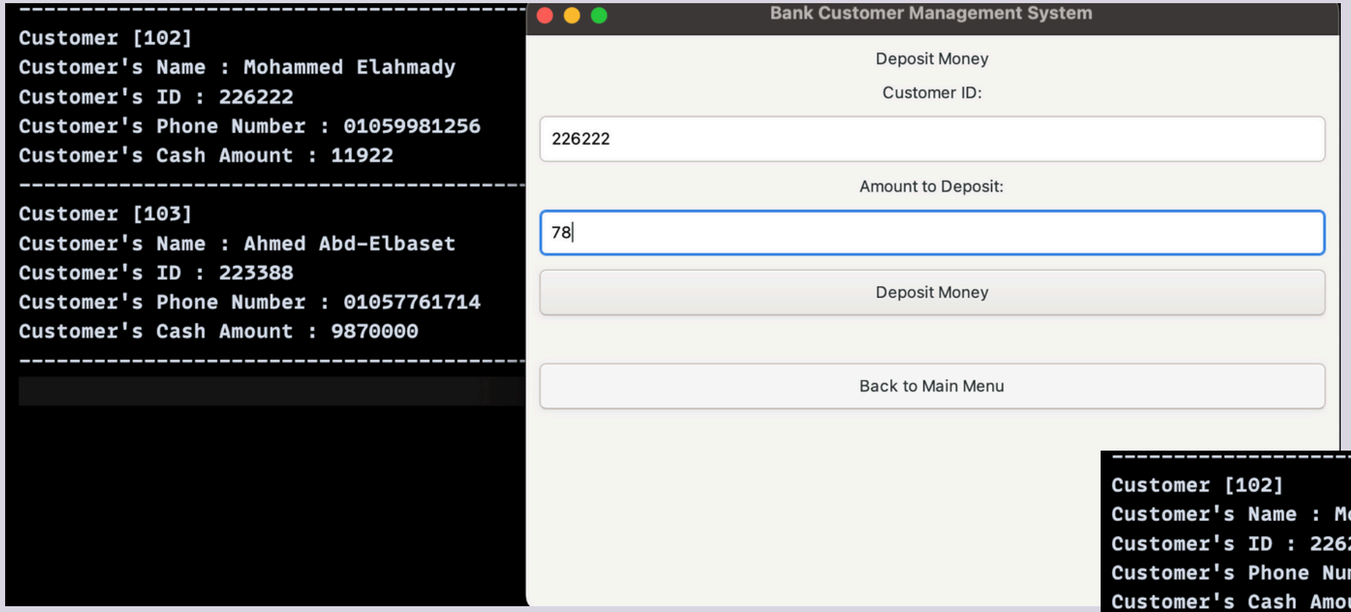
Deposit

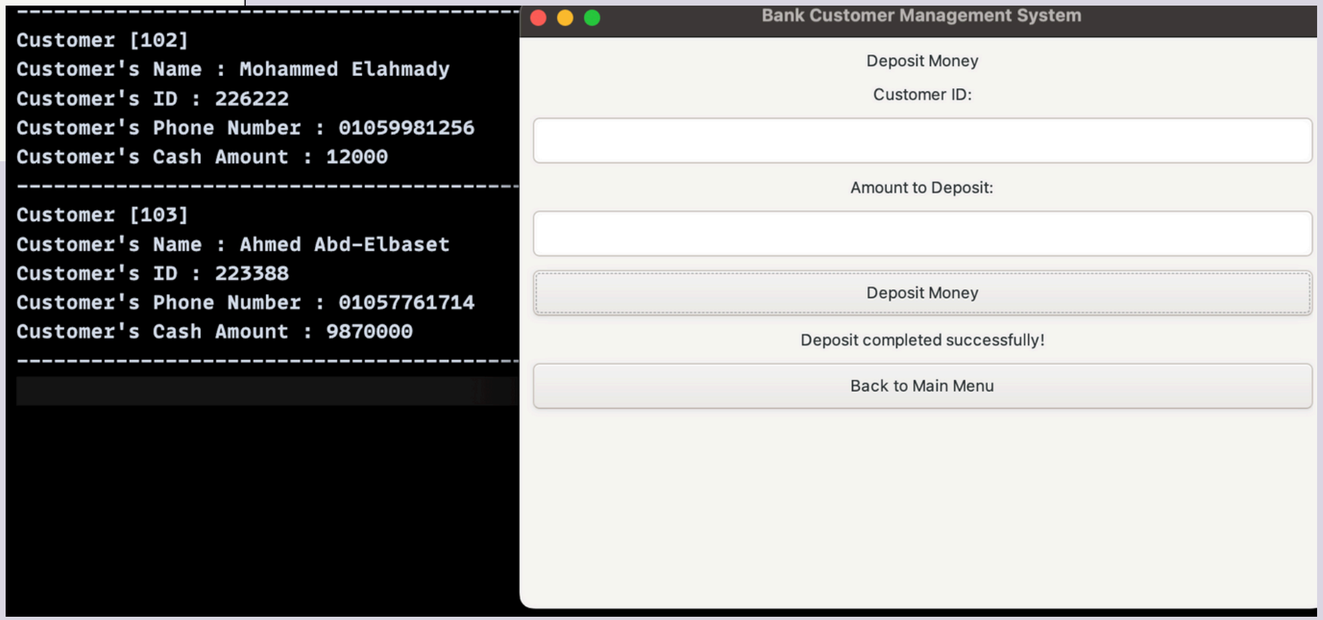
```
Please select the desired operation:
To add a new customer, press '1'
To edit customer information, press '2'
To display customer information, press '3'
To delete customer information, press '4'
To transfer funds between customers, press '5'
To deposit money, press '6'
To withdraw money, press '7'
To exit the system, press '8'
------------------------------------------------------------
7
------------------------------------------------------------
Enter the customer's ID to proceed with the withdrawal:
889233
Enter the amount to withdraw:
10000
------------------------------------------------------------
Withdraw Money
------------------------------------------------------------
Withdrawal completed successfully. New balance: 760000
------------------------------------------------------------
```

Withdraw

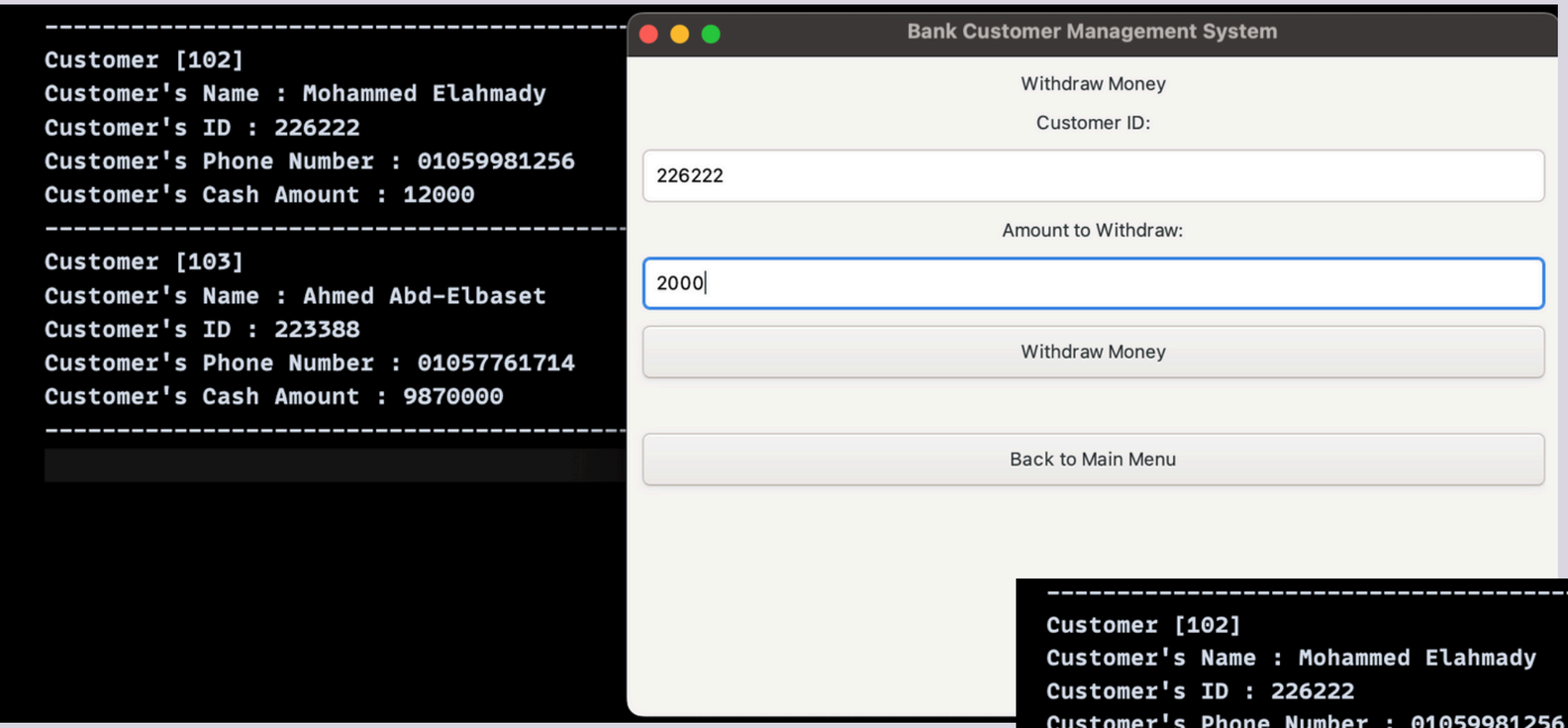
```
Please select the desired operation:
To add a new customer, press '1'
To edit customer information, press '2'
To display customer information, press '3'
To delete customer information, press '4'
To transfer funds between customers, press '5'
To deposit money, press '6'
To withdraw money, press '7'
To exit the system, press '8'
------------------------------------------------------------
6
------------------------------------------------------------
Enter the customer's ID to proceed with the deposit:
889233
Enter the amount to deposit:
2000
------------------------------------------------------------
Deposit Money
------------------------------------------------------------
Deposit completed successfully. New balance: 770000
------------------------------------------------------------
```

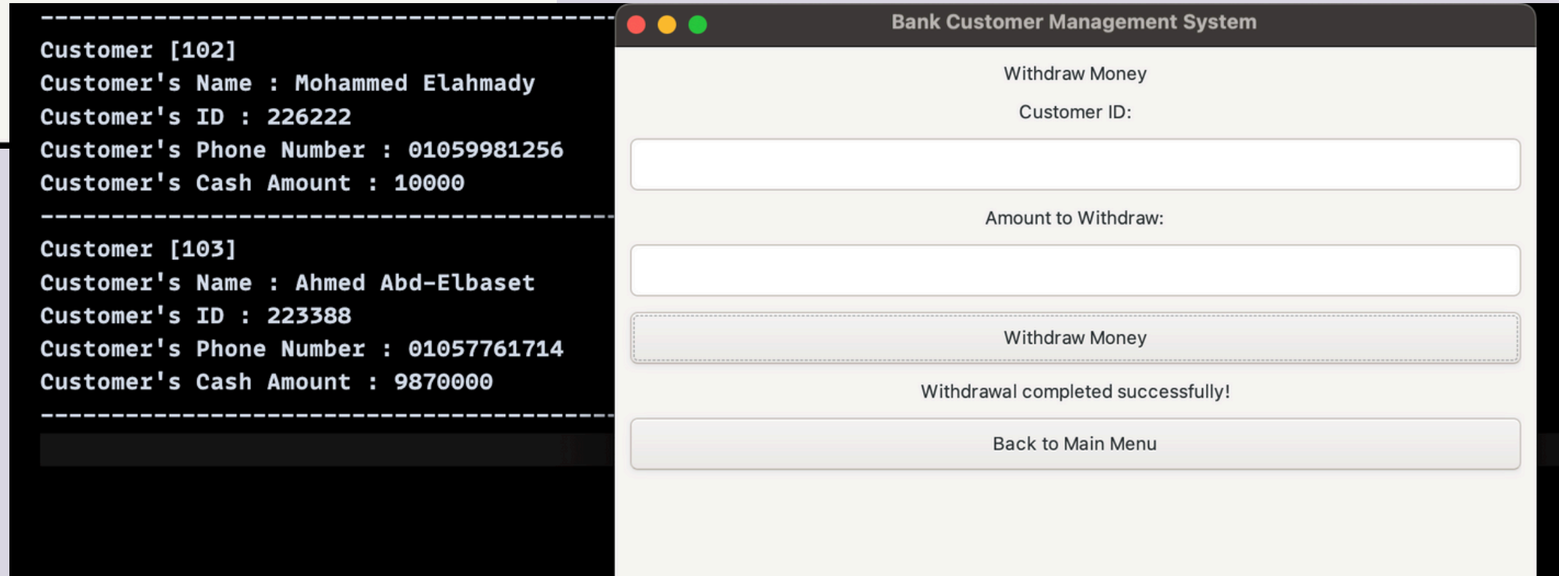
Deep Dive with GUI





Deep Dive with GUI





Thank You