**Instituto Tecnológico y de Estudios Superiores de Occidente**

Departamento de Electrónica, Sistemas e Informática

Ingeniería en Sistemas Computacionales



PROGRAMACIÓN CON MEMORIA DINÁMICA

Tarea 2. Memoria Dinámica y Archivos

Autor: Flores Ruiz David

22 de junio de 2018. Tlaquepaque, Jalisco,

Presentación: 5 pts.

Funcionalidad: 40 pts.

Pruebas: 20 pts.

- Falta describir las pruebas (escenario, y resultados de la experimentación).

- Tienes algunos detalles en el tema de funcionalidad: borrado de datos en el archivo y modificación.

**Instrucciónes para entrega de tarea**

Esta tarea, como el resto, es ***IMPRESCINDIBLE*** entregar los entregables de esta actividad de la siguiente manera:

* **Reporte:** vía *moodle* en **un archivo PDF**.
* **Código:** vía su repositorio **Github**.

La evaluación de la tarea comprende:

* 10% para la presentación
* 60% para la funcionalidad
* 30% para las pruebas

Es necesario responder el apartado de conclusiones, pero no se trata de llenarlo con paja. Si no se aprendió nada al hacer la práctica, es preferible escribir eso. Si el apartado queda vacío, se restarán puntos al porcentaje de presentación.

**Objetivo de la actividad**

El objetivo de la tarea es que el alumno aplique los conocimientos y habilidades adquiridos en el tema de manejo de memoria dinámica y archivos utilizando el lenguaje ANSI C.

**Descripción del problema**

Ahora tienes los conocimientos para enfrentarte a un nuevo proyecto llamado **MyDB**. En este proyecto vas a recrear una parte de un sistema de transacciones bancarias. Para esto vas a requerir del uso de:

* Estructuras
* Funciones y paso de parámetros
* Apuntadores
* Memoria Dinámica
* Archivos binarios

El sistema **MyDB** al ser ejecutado deberá mostrar al usuario una interfaz con el siguiente menú principal:

**<< Sistema MyDB >>**

1. Clientes
2. Cuentas
3. Transacciones
4. Salir

El sistema **MyDB** debe realizar automáticamente, las siguientes operaciones:

1. Si el sistema **MyDB** se ejecutó por primera vez, este deberá crear tres archivos binarios: **clientes.dat**, **cuentas.dat** y **transacciones.dat**. Para esto el sistema debe solicitar al usuario indicar la **ruta de acceso** (por ejemplo, c:\\carpeta\\) en donde se desea crear los archivos (esta información deberá ser almacenada en un archivo de texto llamado **mydb.sys**).

**Clientes**

La opción **Clientes** debe mostrar un submenú con las siguientes opciones:

* **Nuevo** cliente Registra los datos de un nuevo cliente del banco
* **Buscar** cliente Permite consultar la información de un usuario a partir de su

id\_cliente.

* **Eliminar** cliente Si existe, elimina un usuario deseado del sistema. Esto implica que

deben Borrarse las cuentas registradas a nombre del usuario

(utilice id\_usuario para buscar).

* **Imprimir** clientes Imprime la información de todos los clientes registrados en el

sistema.

La información que el sistema requiere almacenar sobre cada cliente es la siguiente:

* Id\_usuario (es un número entero que se genera de manera consecutiva, clave única)
* Nombre
* Apellido materno
* Apellido paterno
* Fecha de nacimiento (tipo de dato estructurado: dd/mm/aaaa)

Para gestionar la información de los clientes, defina un tipo de dato estructurado llamado **Usuario**, utilice instancias de Usuario para capturar la información desde el teclado y posteriormente guardarlo en el archivo usuario.dat.

Un ejemplo del contenido que se estará almacenando en el archivo **usuario.dat** es el siguiente:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **id\_usuario** | **nombre** | **apellido\_paterno** | **apellido\_materno** | **fecha\_nacimiento** |
| 1 | Ricardo | Perez | Perez | {3,10, 2010} |
| 2 | Luis | Rodriguez | Mejía | {2,7, 2005} |
| 3 | Gabriela | Martínez | Aguilar | {7,11,2015} |

**Importante:** considere que no pueden existir datos **id\_usuario** repetidos y que es un valor autonúmerico. Adicionalmente, recuerde que al inicio el archivo no tendrá datos.

**Cuentas**

La opción **Cuentas** debe mostrar un submenú con las siguientes opciones:

* **Nueva** cuenta Registra una cuenta nueva a nombre de un usuario, utilice

**id\_cliente** para relacionar el usuario y la cuenta. Antes de

crear la nueva cuenta se debe verificar que el usuario exista en el

sistema. Adicionalmente, se debe indicar el saldo con el que se

abre la cuenta. Por ejemplo; $1000.

* **Buscar** cuenta Permite consultar en pantalla la información de una cuenta en el

sistema a partir de su **id\_cuenta**. En pantalla debe mostrarse:

**id\_cuenta**, **nombre de cliente**, **saldo de la cuenta**.

* **Eliminar** cuenta Si existe, elimina la cuenta deseada en el sistema.
* **Imprimir** cuentas Imprime la información de todas las cuentas registradas en el

sistema. En pantalla debe mostrarse un listado con la siguiente información de las cuentas: **id\_cuenta**, **nombre de cliente**, **saldo de la cuenta**.

La información que el sistema requiere almacenar sobre cada cuenta es la siguiente:

* id\_cuenta (es un número entero que se genera de manera consecutiva, clave única)
* id\_usuario (indica a quien pertenece la cuenta)
* Saldo
* Fecha de apertura (tipo de dato estructurado: dd/mm/aaaa)

Para gestionar la información de las cuentas, defina un tipo de dato estructurado llamado **Cuenta**, utilice instancias de **Cuenta** para capturar la información desde el teclado y posteriormente guardarlo en el archivo **cuenta.dat**.

Un ejemplo del contenido que se estará almacenando en el archivo **cuenta.dat** es el siguiente:

|  |  |  |  |
| --- | --- | --- | --- |
| **id\_cuenta** | **Id\_usuario** | **Saldo** | **fecha\_apertura** |
| 1 | 1 | Perez | {12,6, 2018} |
| 2 | 2 | Rodriguez | {2,7, 2018} |
| 3 | 1 | Martínez | {7,3,2018} |

**Importante:** considere que no pueden existir valores de **id\_cuenta** repetidos y que es un valor autonúmerico. Adicionalmente, observe que un usuario puede tener más de una cuenta.

**Transacciones**

La opción **Transacciones** debe mostrar un submenú con las siguientes opciones:

* **Depósito** Permite incrementar el saldo de la cuenta, para esto el sistema

requiere: **id\_cuenta**, **monto a depositar** (valide que la cuenta exista).

* **Retiro** Permite a un cliente disponer del dinero que tiene una cuenta

bancaria. Para esto el sistema requiere: **id\_cuenta**, **monto a retirar**

(valide que la cuenta existe y que tiene fondos suficientes).

* **Transferencia** Permite a un cliente transferir dinero de una cuenta origen a una

cuenta destino. Para esto el sistema requiere: **id\_cuenta origen**,

**id\_cuenta destino**, **monto a transferir** (valide que existan ambas

cuentas y que la cuenta origen tiene fondos suficientes).

La información que el sistema requiere almacenar sobre cada transacción es la siguiente:

* id\_transacción (es un número entero que se genera de manera consecutiva, no se puede repetir)
* Tipo de operación (depósito, retiro, transferencia)
* Cuenta origen
* Cuenta destino (se utiliza para las operaciones de transferencia, en otro caso, NULL)
* Fecha de la transacción
* Monto de la transacción

Para gestionar la información de las trasferencias, defina un tipo de dato estructurado llamado **Transferencia**, utilice instancias de Transferencia para capturar la información desde el teclado y posteriormente guardarlo en el archivo transferencia.dat.

Un ejemplo del contenido que se estará almacenando en el archivo **transferencia.dat** es el siguiente:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **id\_transaccion** | **tipo\_transaccion** | **Id\_cuenta\_origen** | **Id\_cuenta\_destino** | **fecha\_transaccion** | **monto\_transaccion** |
| 1 | Retiro | 1 | Null | {12,6, 2018} | $100 |
| 2 | Deposito | 2 | Null | {12,6, 2018} | $5000 |
| 3 | Transferencia | 2 | 1 | {12,6,2018} | $1500 |

**Importante:** considere que no pueden existir datos **id\_transaccion** repetidos y que es un valor autonúmerico. Adicionalmente, recuerde que al inicio el archivo no tendrá datos y que los saldos de las cuentas deberán afectarse por las transacciones realizadas.

Solución del alumno, pruebas y Conclusiones

Código fuente

<<Copie y pegue su código fuente aquí.>>

/\*

============================================================================

Name : NEW\_TAREA.c

Author :

Version :

Copyright : Your copyright notice

Description : Hello World in C, Ansi-style

============================================================================

\*/

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <string.h>

**#define** MAX 20

**typedef** **struct**{

**int** dia;

**int** mes;

**int** ano;

}fecha;

**typedef** **struct**{

**int** id;

**char** nombre[25];

**char** paterno[25];

**char** materno[25];

fecha nacimiento;

}Cliente;

**typedef** **struct**{

**int** id\_cuenta;

**int** id\_cliente;

**int** saldo;

fecha apertura;

}Cuenta;

**typedef** **struct**{

**int** id\_transaccion;

**char** operacion[15];

**int** cuenta\_origen;

**int** cuenta\_destino;

**int** monto;

fecha fch\_transaccion;

}Transaccion;

**enum** MenuPrincipal {*Cero*, *Clientes*, *Cuentas*, *Transacciones*, *Salir*};

**void** **submenuCliente**();

**void** **submenuCuentas**();

**void** **submenuTransacciones**();

**void** **altaCliente** (Cliente clientes[MAX], **int** posicion, Cliente \*arrApuntadores[]);

**void** **buscarCliente** (Cliente clientes[MAX], Cliente \*arrApuntadores[]);

**void** **eliminarCliente** (Cliente clientes[MAX], Cliente \*arrApuntadores[]);

**void** **imprimirClientes**(Cliente clientes[MAX], Cliente \*arrApuntadores[]);

**int** **altaCuenta** (Cuenta cuentas[MAX], **int** posiCuentas, Cuenta \*apCuentas[]);

**void** **buscarCuenta** (Cuenta cuentas[MAX], Cuenta \*apCuentas[]);

**void** **eliminarCuenta** (Cuenta cuentas[MAX], Cuenta \*apCuentas[], **int** valor);

**void** **imprimirCuentas**(Cuenta cuentas[MAX], Cuenta \*apCuentas[]);

**int** **deposito** (Transaccion transacciones[MAX], **int** posTransa, Transaccion \*apTransa[]);

**int** **retiro** (Transaccion transacciones[MAX], **int** posTransa, Transaccion \*apTransa[]);

**int** **transferencia** (Transaccion transacciones[MAX], **int** posTransa, Transaccion \*apTransa[]);

Cliente clientes[MAX];

**int** posicion = 0;

Cliente \*arrApuntadores[MAX];

Cuenta cuentas[MAX];

**int** posiCuentas = 0;

Cuenta \*apCuentas[MAX];

Transaccion transacciones[MAX];

**int** posTransa = 0;

Transaccion \*apTransa[MAX];

//GLOBALES para funcion Crear o abrir Archivos Binarios

FILE \*mydb\_sys; //Para guardar ruta de acceso de los otros 3 archivos binarios

FILE \*clientes\_dat = NULL, \*cuentas\_dat = NULL, \*transacciones\_dat = NULL;

**char** rutaClientes[35] = "";

**char** rutaCuentas[35] = "";

**char** rutaTransacciones[35] = "";

**void** **leerArchivo** (FILE \*archivo, **char** arr[]);

**void** **ArchivosBinarios** ();

**int** **main**(**void**) {

**setvbuf**(stdout, NULL, \_IONBF, 0);

ArchivosBinarios (); //Pide ruta y crea los archivos BINARIOS

**int** i;

**for**(i=0; i<MAX; i++){

arrApuntadores[i] = NULL; //Inicializar el Arreglo de apuntadores en posiciones NULL

}

**for**(i=0; i<MAX; i++){

apCuentas[i] = NULL; //Inicializar el Arreglo de apuntadores en posiciones NULL

}

**for**(i=0; i<MAX; i++){

apTransa[i] = NULL; //Inicializar arreglo de apuntadores

}

**int** opcion;

**do**{

**printf**("\n \t \_-\_-\_MENU PRINCIPAL\_-\_-\_");

**printf**("\n1. Clientes");

**printf**("\n2. Cuentas");

**printf**("\n3. Transacciones");

**printf**("\n4. Salir");

**printf**("\nEscoja su opción (1-4) el número válido:");

**scanf**("%d", &opcion);

**switch**(opcion)

{

**case** *Clientes*: submenuCliente();

**break**;

**case** *Cuentas*: submenuCuentas();

**break**;

**case** *Transacciones*: submenuTransacciones();

**break**;

**default**:

**if**(opcion != *Salir*)

**printf**( "\n-----Número NO válido, escoja de nuevo-----");

}

}**while**(opcion != *Salir*);

//Para leer su contenido CLIENTES.dat

clientes\_dat = **fopen**(rutaClientes,"rb");

**if** (clientes\_dat==NULL){

**printf**("Error al abrir archivo clientes\_dat\n");

}

**else**{

**printf**("\nTodos los datos almacenados en el archivo clientes\_dat.");

Cliente arrLeerClientes[250];

i = 0;

**fread**(&arrLeerClientes[i], **sizeof**(Cliente), 1, clientes\_dat);

//Cliente \*apunCliente = arrLeerClientes;

**while**( !feof(clientes\_dat) )

{

//if( (apunCliente + i)->id != -5 ){

**printf**("\n\*\*\*\*Cliente %d\*\*\*\*", i+1);

**printf**("\nEl ID es: %d", arrLeerClientes[i].id);

**printf**("\nEl nombre del cliente es: %s", arrLeerClientes[i].nombre);

**printf**("\nEl apellido Paterno del cliente es: %s", arrLeerClientes[i].paterno);

**printf**("\nEl apellido Materno del cliente es: %s", arrLeerClientes[i].materno);

**printf**("\nSu fecha de Nacimiento es: %d / %d / %d \n", arrLeerClientes[i].nacimiento.dia, arrLeerClientes[i].nacimiento.mes, arrLeerClientes[i].nacimiento.ano);

i++;//Leer la siguiewnte

**fread**(&arrLeerClientes[i], **sizeof**(Cliente), 1, clientes\_dat);

//}

}

**fclose**(clientes\_dat);

}

//Para leer su contenido CUENTAS.dat

cuentas\_dat = **fopen**(rutaCuentas,"rb");

**if** (cuentas\_dat==NULL){

**printf**("Error al abrir archivo cuentas\_dat\n");

}

**else**{

**printf**("\nTodos los datos almacenados en el archivo cuentas\_dat.");

Cuenta arrLeerCuentas[250];

i = 0;

**fread**(&arrLeerCuentas[i], **sizeof**(Cuenta), 1, cuentas\_dat);

//Cuenta \*apunCuenta = arrLeerCuentas;

**while**( !feof(cuentas\_dat) )

{

//if( (apunCuenta + i)->id != -5 ){

**printf**("\n\*\*\*\*Cuenta %d\*\*\*\*", i+1);

**printf**("\nEl ID cuenta es: %d", arrLeerCuentas[i].id\_cuenta);

**printf**("\nEl ID del cliente es: %d", arrLeerCuentas[i].id\_cliente);

**printf**("\nEl saldo en la ciuenta es: %d", arrLeerCuentas[i].saldo);

**printf**("\nSu fecha de Nacimiento es: %d / %d / %d \n", arrLeerCuentas[i].apertura.dia, arrLeerCuentas[i].apertura.mes, arrLeerCuentas[i].apertura.ano);

i++;//Leer la siguiewnte

**fread**(&arrLeerCuentas[i], **sizeof**(Cuenta), 1, cuentas\_dat);

//}

}

**fclose**(cuentas\_dat);

}

//Para leer su contenido TRANSACCIONES.dat

transacciones\_dat = **fopen**(rutaTransacciones,"rb");

**if** (transacciones\_dat==NULL){

**printf**("Error al abrir archivo transacciones\_dat\n");

}

**else**{

**printf**("\nTodos los datos almacenados en el archivo transacciones\_dat.");

Transaccion arrLeerTransacciones[250];

i = 0;

**fread**(&arrLeerTransacciones[i], **sizeof**(Transaccion), 1, transacciones\_dat);

//Transaccion \*apunTransaccion = arrLeerTransacciones;

**while**( !feof(transacciones\_dat) )

{

//if( (apunTransaccion + i)->id != -5 ){

**printf**("\n\*\*\*\*Transaccion %d\*\*\*\*", i+1);

**printf**("\nEl ID transaccion es: %d", arrLeerTransacciones[i].id\_transaccion);

**printf**("\nLa operación realizada es: %s", arrLeerTransacciones[i].operacion);

**printf**("\nLa cuenta de origen es: %d", arrLeerTransacciones[i].cuenta\_origen);

**printf**("\nLa cuenta de origen es: %d", arrLeerTransacciones[i].cuenta\_destino);

**printf**("\nEl monto fue de: %d", arrLeerTransacciones[i].monto);

**printf**("\nSu fecha de Nacimiento es: %d / %d / %d \n", arrLeerTransacciones[i].fch\_transaccion.dia, arrLeerTransacciones[i].fch\_transaccion.mes, arrLeerTransacciones[i].fch\_transaccion.ano);

i++;//Leer la siguiewnte

**fread**(&arrLeerTransacciones[i], **sizeof**(Transaccion), 1, transacciones\_dat);

//}

}

**fclose**(transacciones\_dat);

}

**return** EXIT\_SUCCESS;

}

**int** **deposito** (Transaccion transacciones[MAX], **int** posTransa, Transaccion \*apTransa[]){

Transaccion \*arrTransa = transacciones; //Declarar puntero local

**int** auxID = posTransa + 1;

**char** auxOperacion[15];

**int** cuentaOrigen;

//int cuentaDestino; Vale 0 no se utiliza

**int** auxMonto;

**int** dia, mes, ano;

**int** i=0, bandera = 0;

Cuenta \*apunArr = cuentas;

**printf**("\nPara hacer el depósito, introduzca su ID de cuenta: ");

**scanf**("%d", &cuentaOrigen);

**for**(i=0; i < MAX; i++){

**if**( (apunArr + i)->id\_cuenta == cuentaOrigen )

{

**printf**("Su ID de Transaccion es --- %d --- \n", auxID);

**printf**("Dame el monto del depósito es:\n");

**scanf**("%d", &auxMonto);

(apunArr + i)->saldo = (apunArr + i)->saldo + auxMonto; //DEPÓSITO

**setbuf**(stdin, 0);

**printf**("Dame la fecha de la transaccion dd/mm/aa:\n");

**scanf**("%d %d %d", &dia, &mes, &ano);

(arrTransa + posTransa)->id\_transaccion = auxID;

**strcpy**(((arrTransa + posTransa)->operacion),"");

**strcpy**( auxOperacion , "Deposito");

**strcpy**( ((arrTransa + posTransa)->operacion), auxOperacion );

(arrTransa + posTransa)->cuenta\_origen = cuentaOrigen;

(arrTransa + posTransa)->cuenta\_destino = 0;

(arrTransa + posTransa)->monto = auxMonto;

(arrTransa + posTransa)->fch\_transaccion.dia = dia;

(arrTransa + posTransa)->fch\_transaccion.mes = mes;

(arrTransa + posTransa)->fch\_transaccion.ano = ano;

apTransa[posTransa] = (arrTransa + posTransa); //Ya no es NULL el Arreglo del main

bandera = 1;

//Escribir en Archivo nuevas transacciones

transacciones\_dat = **fopen**(rutaTransacciones, "a+b");

**fwrite**(&arrTransa[posTransa], **sizeof**( Transaccion ), 1, transacciones\_dat);

**fclose**(transacciones\_dat);

//Actualizar el archivo Cuentas con lo mas nuevo

cuentas\_dat = **fopen**(rutaCuentas, "wb");//Borrar lo que habia antes en cuentas\_dat

**fclose**(cuentas\_dat);

cuentas\_dat = **fopen**(rutaCuentas, "a+b");//Borrar lo que habia antes en cuentas\_dat

**for**(i=0; i<posiCuentas; i++)

**fwrite**(&cuentas[i], **sizeof**( Cuenta ), 1, cuentas\_dat);

**fclose**(cuentas\_dat);

**return** 1;

}

}

**if**(bandera == 0){

**printf**("\nNo se pudo realizar el deposito, el ID de cuenta no existe\n");

**return** 0;

}

**return** -8;//Seria un error el -8

}

**int** **retiro** (Transaccion transacciones[MAX], **int** posTransa, Transaccion \*apTransa[]){

Transaccion \*arrTransa = transacciones; //Declarar puntero local

**int** auxID = posTransa + 1;

**char** auxOperacion[15];

**int** cuentaOrigen;

//int cuentaDestino; Vale 0 no se utiliza

**int** auxMonto;

**int** dia, mes, ano;

**int** i=0, bandera = 0;

Cuenta \*apunArr = cuentas;

**printf**("\nPara hacer el RETIRO, introduzca su ID de cuenta: ");

**scanf**("%d", &cuentaOrigen);

**for**(i=0; i < MAX; i++){

**if**( (apunArr + i)->id\_cuenta == cuentaOrigen )

{

**printf**("Su ID de Transaccion es --- %d --- \n", auxID);

**printf**("Dame el monto del retiro:\n");

**scanf**("%d", &auxMonto);

**while**((apunArr + i)->saldo < auxMonto){

**printf**("El monto excede su Saldo, dame el monto del retiro:\n");

**scanf**("%d", &auxMonto);

}

(apunArr + i)->saldo = (apunArr + i)->saldo - auxMonto; //RETIRO

**setbuf**(stdin, 0);

**printf**("Dame la fecha de la transaccion dd/mm/aa:\n");

**scanf**("%d %d %d", &dia, &mes, &ano);

(arrTransa + posTransa)->id\_transaccion = auxID;

**strcpy**(((arrTransa + posTransa)->operacion),"");

**strcpy**( auxOperacion , "Retiro");

**strcpy**( ((arrTransa + posTransa)->operacion), auxOperacion );

(arrTransa + posTransa)->cuenta\_origen = cuentaOrigen;

(arrTransa + posTransa)->cuenta\_destino = 0;

(arrTransa + posTransa)->monto = auxMonto;

(arrTransa + posTransa)->fch\_transaccion.dia = dia;

(arrTransa + posTransa)->fch\_transaccion.mes = mes;

(arrTransa + posTransa)->fch\_transaccion.ano = ano;

apTransa[posTransa] = (arrTransa + posTransa); //Ya no es NULL el Arreglo del main

bandera = 1;

//Escribir en Archivo nuevas transacciones

transacciones\_dat = **fopen**(rutaTransacciones, "a+b");

**fwrite**(&arrTransa[posTransa], **sizeof**( Transaccion ), 1, transacciones\_dat);

**fclose**(transacciones\_dat);

//Actualizar el archivo Cuentas con lo mas nuevo

cuentas\_dat = **fopen**(rutaCuentas, "wb");//Borrar lo que habia antes en cuentas\_dat

**fclose**(cuentas\_dat);

cuentas\_dat = **fopen**(rutaCuentas, "a+b");//Borrar lo que habia antes en cuentas\_dat

**for**(i=0; i<posiCuentas; i++)

**fwrite**(&cuentas[i], **sizeof**( Cuenta ), 1, cuentas\_dat);

**fclose**(cuentas\_dat);

**return** 1;

}

}

**if**(bandera == 0){

**printf**("\nNo se pudo realizar el retiro, el ID de cuenta no existe\n");

**return** 0;

}

**return** -8;//Seria un error el -8

}

**int** **transferencia** (Transaccion transacciones[MAX], **int** posTransa, Transaccion \*apTransa[]){

Transaccion \*arrTransa = transacciones; //Declarar puntero local

**int** auxID = posTransa + 1;

**char** auxOperacion[15];

**int** cuentaOrigen;

**int** cuentaDestino;

**int** auxMonto;

**int** dia, mes, ano;

**int** i=0, bandera = 0;

Cuenta \*apunArr = cuentas;

**printf**("\nPara hacer la transferencia, introduzca su ID de cuenta: ");

**scanf**("%d", &cuentaOrigen);

**printf**("\nPara hacer la transferencia, la cuenta de destino: ");

**scanf**("%d", &cuentaDestino);

**int** j;

**for**(i=0; i < MAX; i++)

**for**(j = 0; j < MAX; j++)

**if** ( ((apunArr + i)->id\_cuenta == cuentaOrigen) && (apunArr + j)->id\_cuenta == cuentaDestino)

{

**printf**("Su ID de Transaccion es --- %d --- \n", auxID);

**printf**("Dame el monto a transferir:\n");

**scanf**("%d", &auxMonto);

**while**((apunArr + i)->saldo < auxMonto){

**printf**("No tiene fondos, dame el monto:\n");

**scanf**("%d", &auxMonto);

}

(apunArr + i)->saldo = (apunArr + i)->saldo - auxMonto; //TRANSFERENCIA

(apunArr + j)->saldo = (apunArr + j)->saldo + auxMonto;

**setbuf**(stdin, 0);

**printf**("Dame la fecha de la transaccion dd/mm/aa:\n");

**scanf**("%d %d %d", &dia, &mes, &ano);

(arrTransa + posTransa)->id\_transaccion = auxID;

**strcpy**(((arrTransa + posTransa)->operacion),"");

**strcpy**( auxOperacion , "Retiro");

**strcpy**( ((arrTransa + posTransa)->operacion), auxOperacion );

(arrTransa + posTransa)->cuenta\_origen = cuentaOrigen;

(arrTransa + posTransa)->cuenta\_destino = 0;

(arrTransa + posTransa)->monto = auxMonto;

(arrTransa + posTransa)->fch\_transaccion.dia = dia;

(arrTransa + posTransa)->fch\_transaccion.mes = mes;

(arrTransa + posTransa)->fch\_transaccion.ano = ano;

apTransa[posTransa] = (arrTransa + posTransa); //Ya no es NULL el Arreglo del main

bandera = 1;

//Escribir en Archivo nuevas transacciones

transacciones\_dat = **fopen**(rutaTransacciones, "a+b");

**fwrite**(&arrTransa[posTransa], **sizeof**( Transaccion ), 1, transacciones\_dat);

**fclose**(transacciones\_dat);

//Actualizar el archivo Cuentas con lo mas nuevo

cuentas\_dat = **fopen**(rutaCuentas, "wb");//Borrar lo que habia antes en cuentas\_dat

**fclose**(cuentas\_dat);

cuentas\_dat = **fopen**(rutaCuentas, "a+b");//Borrar lo que habia antes en cuentas\_dat

**for**(i=0; i<posiCuentas; i++)

**fwrite**(&cuentas[i], **sizeof**( Cuenta ), 1, cuentas\_dat);

**fclose**(cuentas\_dat);

**return** 1;

}

**if**(bandera == 0){

**printf**("\nNo se pudo realizar la transferencia, el ID de cuenta no existe\n");

**return** 0;

}

**return** -8;//Seria un error el -8

}

**void** **imprimirCuentas**(Cuenta cuentas[MAX], Cuenta \*apCuentas[]){

Cuenta \*apunArr = cuentas;

**printf**("\nLas cuentas registradas al momento son:");

**for**(**int** i=0; i < MAX; i++){

**if** ( (apCuentas[i] != NULL) && ((apunArr + i)->id\_cuenta != -5) )//IMPRIMIR SOLO LOS QUE NO APUNTEN A NULL

{

**printf**("\n\*\*\*\*Cuenta %d\*\*\*\*", i+1);

**printf**("\nEl ID de cuenta es: %d", (apunArr + i)->id\_cuenta);

**printf**("\nEl ID de cliente es: %d", (apunArr + i)->id\_cliente);

**printf**("\nEl saldo de la cuenta es: %d", (apunArr + i)->saldo);

**printf**("\nSu fecha de Apertura de la cuenta es: %d / %d / %d \n", (apunArr + i)->apertura.dia, (apunArr + i)->apertura.mes, (apunArr + i)->apertura.ano);

}

}

}

**void** **eliminarCuenta** (Cuenta cuentas[MAX], Cuenta \*apCuentas[], **int** valor){

Cuenta \*apunArr = cuentas;

**int** idBorrar;

**int** bandera = 0;

**int** i;

Cliente \*arrApuntadores = clientes;

**if**(valor == 1){ //Llamado desde el MENU

**printf**("\nDame el ID de la Cuenta que desea borrar:");

**scanf**("%d", &idBorrar);

**for**(i= 0; i < MAX; i++){

**if** ( (apunArr + i)->id\_cuenta == idBorrar )

{

apCuentas[i] = NULL;

(apunArr + i)->id\_cuenta = -5; //BORRADO LÓGICO al ID = -5

**printf**("Cuenta eliminada\n");

bandera = 1;

}

}

**if**(bandera == 0)

**printf**("No existe ese no. de ID de cuenta\n");

}

**else** **if** (valor == 2){ //Llamado desde eliminar Cliente

**for**(i= 0; i < MAX; i++){

**if** ((arrApuntadores + i)->id == -5)

{

apCuentas[i] = NULL;

(apunArr + i)->id\_cuenta = -5; //BORRADO LÓGICO al ID = -5

**printf**("Cuenta eliminada\n");

bandera = 1;

}

}

**if**(bandera == 0)

**printf**("No existe ese no. de ID de cuenta\n");

}

}

**int** **altaCuenta** (Cuenta cuentas[MAX], **int** posiCuentas, Cuenta \*apCuentas[]){

Cuenta \*arrCuenta = cuentas; //Declarar puntero local

**int** auxID\_cuenta = posiCuentas + 1;

**int** auxID\_cliente;

**int** auxSaldo;

**int** dia, mes, ano;

**printf**("\nPara crear su cuenta, introduzca su ID de cliente: ");

**scanf**("%d", &auxID\_cliente);

Cliente \*apunArr = clientes;

**int** i, bandera = 0;

**for**(i=0; i < MAX; i++){

**if**( (apunArr + i)->id == auxID\_cliente )

{

**printf**("Su ID\_cuenta es --- %d ---\n", auxID\_cuenta);

(arrCuenta)->id\_cliente = (apunArr + i)->id;

**printf**("Dame el saldo inicial:\n");

**scanf**("%d", &auxSaldo);

**setbuf**(stdin, 0);

**printf**("Dame la fecha de apertura de la cuenta dd/mm/aa:\n");

**scanf**("%d %d %d", &dia, &mes, &ano);

(arrCuenta + posiCuentas)->id\_cuenta = auxID\_cuenta;

(arrCuenta + posiCuentas)->id\_cliente = auxID\_cliente;

(arrCuenta + posiCuentas)->saldo = auxSaldo;

(arrCuenta + posiCuentas)->apertura.dia = dia;

(arrCuenta + posiCuentas)->apertura.mes = mes;

(arrCuenta + posiCuentas)->apertura.ano = ano;

apCuentas[posiCuentas] = (arrCuenta + posiCuentas); //Ya no es NULL el Arreglo del main

bandera = 1;

cuentas\_dat = **fopen**(rutaCuentas, "a+b");

**fwrite**(&arrCuenta[posiCuentas], **sizeof**( Cuenta ), 1, cuentas\_dat);

**fclose**(cuentas\_dat);

**return** 1;

}

}

**if**(bandera == 0)

{

**printf**("\nNo se pudo crear la cuenta, el ID de cliente no existe\n");

**return** 0;

}

**return** -8;//Seria un error el -8

}

**void** **buscarCuenta** (Cuenta cuentas[MAX], Cuenta \*apCuentas[]){

Cuenta \*apunArr = cuentas;

**int** ID\_cuenta; //Para buscar por ID

**printf**("\nBuscar por ID\_Cuenta: \n");

**scanf**("%d", &ID\_cuenta);

**int** bandera = 0;

**for**(**int** i=0; i < MAX; i++){

**if**( ((apunArr + i)->id\_cuenta == ID\_cuenta) && (apCuentas[i] != NULL) )

{

**printf**("\n\*\*\*\*Cuenta %d\*\*\*\*", i+1);

**printf**("\nEl id\_cuenta es: %d", (apunArr + i)->id\_cuenta);

**printf**("\nEl id\_cliente es: %d", (apunArr + i)->id\_cliente);

**printf**("\nEl saldo de la cuenta es: %d", (apunArr + i)->saldo);

**printf**("\nSu fecha de Apertura de la cuenta es: %d / %d / %d \n", (apunArr + i)->apertura.dia, (apunArr + i)->apertura.mes, (apunArr + i)->apertura.ano);

bandera = 1;

}

}

**if**(ID\_cuenta != 0)

{

**if**(bandera == 0)

**printf**("\nNo se encontro ningun cliente en su búsqueda\n");

}

}

**void** **buscarCliente** (Cliente clientes[MAX], Cliente \*arrApuntadores[]){

Cliente \*apunArr = clientes;

**int** ID=0; //Para buscar por ID

**printf**("\nBuscar por ID: \n");

**scanf**("%d", &ID);

**int** bandera = 0;

**for**(**int** i=0; i < MAX; i++){

**if**( ((apunArr + i)->id == ID) && (arrApuntadores[i] != NULL) )

{

**printf**("\n\*\*\*\*Cliente %d\*\*\*\*", i+1);

**printf**("\nEl ID es: %d", (apunArr + i)->id);

**printf**("\nEl nombre del cliente es: %s", (\*(apunArr + i)).nombre);

**printf**("\nEl apellido Paterno del cliente es: %s", (\*(apunArr + i)).paterno);

**printf**("\nEl apellido Materno del cliente es: %s", (\*(apunArr + i)).materno);

**printf**("\nSu fecha de Nacimiento es: %d / %d / %d \n", (apunArr + i)->nacimiento.dia, (apunArr + i)->nacimiento.mes, (apunArr + i)->nacimiento.ano);

bandera = 1;

}

}

**if**(ID != 0)

{

**if**(bandera == 0)

**printf**("\nNo se encontro ningun cliente en su búsqueda\n");

}

}

**void** **altaCliente** (Cliente clientes[MAX], **int** posicion, Cliente \*arrApuntadores[]){

Cliente \*arrCliente = clientes; //Declarar puntero local

**int** auxID = posicion + 1;

**char** auxNombre[20];

**char** auxMaterno[25];

**char** auxPaterno[25];

**int** dia, mes, ano;

**printf**("Su ID de cliente es --- %d --- \n", auxID);

**setbuf**(stdin, 0);

**printf**("Dame el nombre:\n");

**gets**(auxNombre);

**setbuf**(stdin, 0);

**printf**("Dame el apellido paterno:\n");

**gets**(auxPaterno);

**setbuf**(stdin, 0);

**printf**("Dame el apellido materno:\n");

**gets**(auxMaterno);

**setbuf**(stdin, 0);

**printf**("Dame la fecha de nacimiento dd/mm/aa:\n");

**scanf**("%d %d %d", &dia, &mes, &ano);

(arrCliente + posicion)->id = auxID;

**strcpy**(((arrCliente + posicion)->nombre),"");

**strcpy**( ((arrCliente + posicion)->nombre), auxNombre );

**strcpy**(((arrCliente + posicion)->paterno),"");

**strcpy**( ((arrCliente + posicion)->paterno), auxPaterno );

**strcpy**(((arrCliente + posicion)->materno),"");

**strcpy**( ((arrCliente + posicion)->materno), auxMaterno );

(arrCliente + posicion)->nacimiento.dia = dia;

(arrCliente + posicion)->nacimiento.mes = mes;

(arrCliente + posicion)->nacimiento.ano = ano;

arrApuntadores[posicion] = (arrCliente + posicion); //Ya no es NULL el Arreglo del main

clientes\_dat = **fopen**(rutaClientes, "a+b");

**fwrite**(&arrCliente[posicion], **sizeof**( Cliente ), 1, clientes\_dat);

**fclose**(clientes\_dat);

}

**void** **eliminarCliente** (Cliente clientes[MAX], Cliente \*arrApuntadores[]){

Cliente \*apunArr = clientes;

**int** idBorrar;

**int** bandera = 0;

**printf**("\nDame el ID del Cliente que desea borrar:");

**scanf**("%d", &idBorrar);

**for**(**int** i=0; i < MAX; i++){

**if** ((apunArr + i)->id == idBorrar)

{

arrApuntadores[i] = NULL; //Por que no lo pude poner: (arrApuntadores + i) = NULL;

(apunArr + i)->id = -5; //BORRADO LÓGICO al ID = -5

**printf**("Cliente eliminado\n");

bandera = 1;

eliminarCuenta (cuentas, apCuentas, 2);

}

}

**if**(bandera == 0)

**printf**("No existe ese no. de ID\n");

// // A CONTINUACION PARA LEER UN ARCHIVO, modificar si es necesario

// clientes\_dat = fopen(rutaClientes,"r+b");

// if (clientes\_dat==NULL){

// printf("Error al abrir archivo clientes\_dat\n");

// }

// else{//Se abrió el archivo para leer y escribir

// rewind(clientes\_dat);

// Cliente arrLeerClientes[250];

// int i = 0;

// int borradoLogico = -5;

// fread(&arrLeerClientes[i], sizeof(Cliente), 1, clientes\_dat);

////Cliente \*apunCliente = arrLeerClientes;

// while( ! feof(clientes\_dat) )

// {

// if( (apunArr + i)->id == idBorrar){

// int retorno = fseek(clientes\_dat, 0, SEEK\_CUR);

//

// if(retorno == 0){

// fwrite(&borradoLogico, sizeof( int ), 1, clientes\_dat);

// printf("El cursor se movió correctamente y ya borramos");

// }

// i++;//Leer la siguiente

// fread(&arrLeerClientes[i], sizeof(Cliente), 1, clientes\_dat);

// }

// }

// fclose(clientes\_dat);

// }

}

**void** **imprimirClientes**(Cliente clientes[MAX], Cliente \*arrApuntadores[]){

Cliente \*apunArr = clientes;

**printf**("\nLos clientes registrados al momento son:");

**for**(**int** i=0; i < MAX; i++){

**if** ( (arrApuntadores[i] != NULL) && ((apunArr + i)->id != -5) )//IMPRIMIR SOLO LOS QUE NO APUNTEN A NULL

{

**printf**("\n\*\*\*\*Cliente %d\*\*\*\*", i+1);

**printf**("\nEl ID es: %d", (apunArr + i)->id);

**printf**("\nEl nombre del cliente es: %s", (\*(apunArr + i)).nombre);

**printf**("\nEl apellido Paterno del cliente es: %s", (\*(apunArr + i)).paterno);

**printf**("\nEl apellido Materno del cliente es: %s", (\*(apunArr + i)).materno);

**printf**("\nSu fecha de Nacimiento es: %d / %d / %d \n", (apunArr + i)->nacimiento.dia, (apunArr + i)->nacimiento.mes, (apunArr + i)->nacimiento.ano);

}

}

}

**void** **submenuCliente**(){

**int** opcion;

**do**{

**printf**("\n\t\t \t -> Submenu CLIENTES...");

**printf**("\n\t\t1. Nuevo cliente");

**printf**("\n\t\t2. Buscar cliente");

**printf**("\n\t\t3. Eliminar cliente");

**printf**("\n\t\t4. Imprimir clientes");

**printf**("\n\t\t5. Salir");

**printf**("\n\t\tOpción CLIENTES, escoja (1-5) un número válido:");

**scanf**("%d", &opcion);

**switch**(opcion)

{

**case** 1: altaCliente (clientes, posicion, arrApuntadores);

posicion++;

**break**;

**case** 2: buscarCliente (clientes, arrApuntadores);

**break**;

**case** 3: eliminarCliente (clientes, arrApuntadores);

**break**;

**case** 4: imprimirClientes (clientes, arrApuntadores);

**break**;

**default**:

**if**(opcion != 5)

**printf**( "\n\t\t -------Número NO válido, escoja de nuevo\n");

}

}**while**(opcion != 5);

}

**void** **submenuCuentas**(){

**int** valor, opcion;

**do**{

**printf**("\n\t\t \t -> Submenu CUENTAS...");

**printf**("\n\t\t1. Nueva cuenta");

**printf**("\n\t\t2. Buscar cuenta");

**printf**("\n\t\t3. Eliminar cuenta");

**printf**("\n\t\t4. Imprimir cuentas");

**printf**("\n\t\t5. Salir");

**printf**("\n\t\tOpción CUENTAS, escoja (1-5) un número válido:");

**scanf**("%d", &opcion);

**switch**(opcion)

{

**case** 1: valor = altaCuenta (cuentas, posiCuentas, apCuentas);

**if**(valor == 1){

posiCuentas++;

}

**break**;

**case** 2: buscarCuenta (cuentas , apCuentas);

**break**;

**case** 3: eliminarCuenta (cuentas, apCuentas, 1);

**break**;

**case** 4: imprimirCuentas(cuentas, apCuentas);

**break**;

**default**:

**if**(opcion != 5)

**printf**( "\n\t\t -------Número NO válido, escoja de nuevo\n");

}

}**while**(opcion != 5);

}

**void** **submenuTransacciones**(){

**int** opcion, valor;

**do**{

**printf**("\n\t\t \t -> Submenu TRANSACCIONES...");

**printf**("\n\t\t1. Depósito");

**printf**("\n\t\t2. Retiro");

**printf**("\n\t\t3. Transferencia");

**printf**("\n\t\t4. Salir");

**printf**("\n\t\tOpción TRANSACCIONES, escoja (1-4) un número válido:");

**scanf**("%d", &opcion);

**switch**(opcion)

{

**case** 1: valor = deposito (transacciones, posTransa, apTransa);

**if** (valor == 1){

posTransa++;

}

**break**;

**case** 2: valor = retiro (transacciones, posTransa, apTransa);

**if**(valor == 1){

posTransa++;

}

**break**;

**case** 3: valor = transferencia (transacciones, posTransa, apTransa);

**if**(valor == 1){

posTransa++;

}

**break**;

**default**:

**if**(opcion != 4)

**printf**( "\n\t\t -------Número NO válido, escoja de nuevo\n");

}

}**while**(opcion != 4);

}

**void** **ArchivosBinarios** (){

**char** rutaAcceso[35], verificarRuta[35], rutaBase[35];

**strcpy**(verificarRuta , "");

mydb\_sys = **fopen**("mydb.sys", "rt");//Validar: si no existe retorna NULL

//clientes\_dat = fopen("Carpeta1\\Yomero.dat", "w"); //Asi se crea correctamente en una carpeta dentro del directorio RAÍZ

**if**(mydb\_sys == NULL){ //Si no existe haz...

**printf**("Introduce Ruta de acceso (Relativa)... Ejemplo: Carpeta1\\\\ \n");

mydb\_sys = **fopen**("mydb.sys", "wt");//modo "w" : abrir un archivo para escritura, se crea si no existe o se sobreescribe si existe.

**scanf**("%s", rutaAcceso);

**fputs**( rutaAcceso , mydb\_sys);

**fclose**(mydb\_sys);

mydb\_sys = **fopen**("mydb.sys", "rt");

leerArchivo(mydb\_sys, verificarRuta);

**fclose**(mydb\_sys);

**printf**("\nLinea del IF: %s", verificarRuta);

}

**else**{ //Si ya existe o NO entonces verificar que la ruta este bien...

**fclose**(mydb\_sys);//Cerrar el de lectura

mydb\_sys = **fopen**("mydb.sys", "rt");

leerArchivo(mydb\_sys, verificarRuta);

**fclose**(mydb\_sys);

**printf**("\nLinea del ELSE: %s", verificarRuta);

}

**int** valor = **strcmp**(verificarRuta, "Carpeta1\\\\");

**printf**("\n comparacion:%d", valor);

**while**(valor != 0)

{

**printf**("\n\nError... (Ruta de acceso INCORRECTA)\n");

**printf**("Introduce Ruta de acceso (Relativa)... Ejemplo: Carpeta1\\\\ \n");

**scanf**("%s", rutaAcceso);

mydb\_sys = **fopen**("mydb.sys", "wt");//modo "w" : abrir un archivo para escritura, se crea si no existe o se sobreescribe si existe.

**fputs**( rutaAcceso , mydb\_sys);

**fclose**(mydb\_sys);

mydb\_sys = **fopen**("mydb.sys", "rt");

leerArchivo(mydb\_sys, verificarRuta);

**fclose**(mydb\_sys);

**printf**("\nLinea del WHILE: %s", verificarRuta);

valor = **strcmp**(verificarRuta, "Carpeta1\\\\");

}

//leer el archivo mydb.sys y recuperar la ruta de Acceso

**printf**("\nverificarRuta: %s", verificarRuta);

**printf**("\nrutaAcceso: %s", rutaAcceso);

**printf**("\nrutaBase: %s", rutaBase);

**strcpy**(rutaBase, verificarRuta); //verificarRuta llega BIEN y la respaldamos

**strcpy**(rutaClientes, rutaBase);

**strcat**(rutaClientes, "clientes.dat"); //Nombre de Primer Archivo a crear

clientes\_dat = **fopen**(rutaClientes, "a+b"); //Validar que la ruta este correcta

**strcpy**(rutaCuentas, rutaBase);

**strcat**(rutaCuentas, "cuentas.dat"); //Nombre de 2do Archivo a crear

cuentas\_dat = **fopen**(rutaCuentas, "a+b");

**strcpy**(rutaTransacciones, rutaBase);

**strcat**(rutaTransacciones, "transacciones.dat"); //Nombre de 3er Archivo a crear

transacciones\_dat = **fopen**(rutaTransacciones, "a+b");

**fclose**(clientes\_dat);

**fclose**(cuentas\_dat);

**fclose**(transacciones\_dat);

}

**void** **leerArchivo** (FILE \*archivo, **char** arr[35]){

**char** letra;

**int** i = 0;

**printf**("Funcion leerArchivo: ");

**while** ((letra = **fgetc**(archivo)) != EOF) {

arr[i] = letra;

**printf**( "%c",letra );

i++;

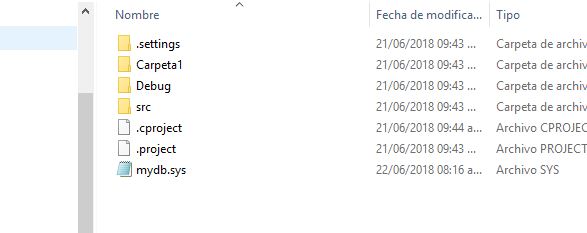
}

arr[i] = '\0';

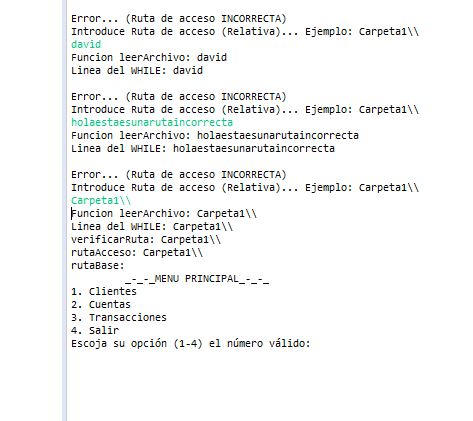
}

Ejecución

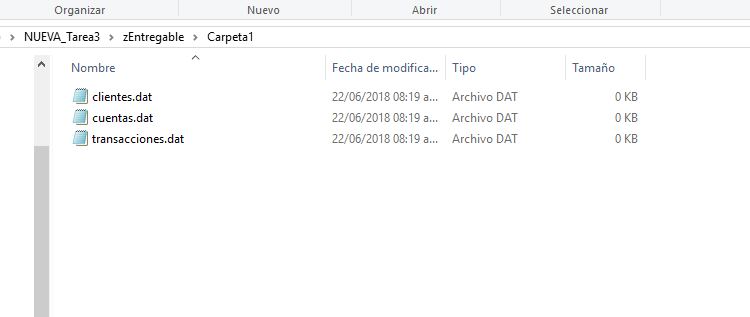
<<Inserte capturas de pantalla de una ejecución para todos los casos>>



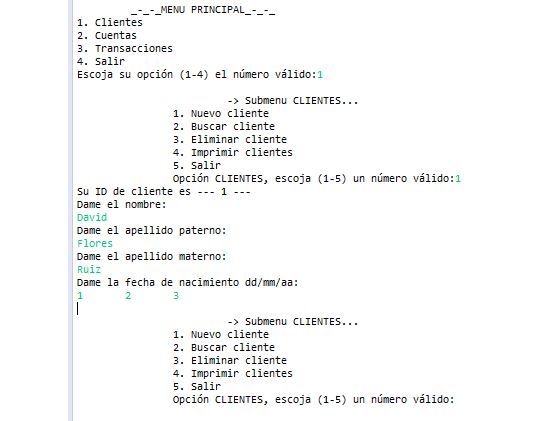
La Carpeta1 debe existir y la carpeta mydb.sys asi como los archivos Binarios dentro de Carpeta1\\ también pueden o no existir y todo funcionara bien ya que el programa te obliga a seguir los pasos para poder crearlos o abrirlos si ya existían.



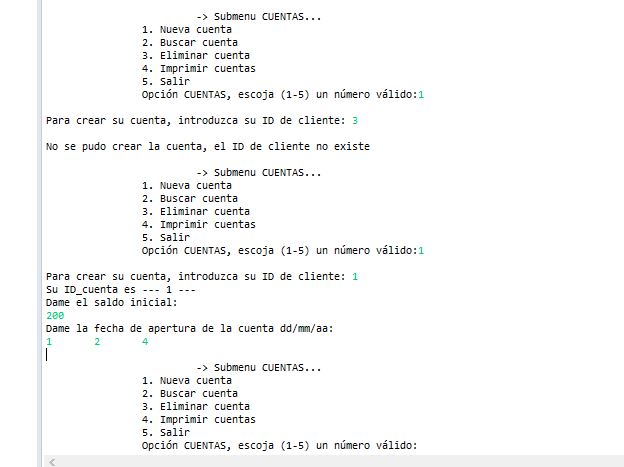
Validar la ruta correcta debe ser “Carpeta1\\” así lo debe introducir el usuario porque es una ruta relativa.



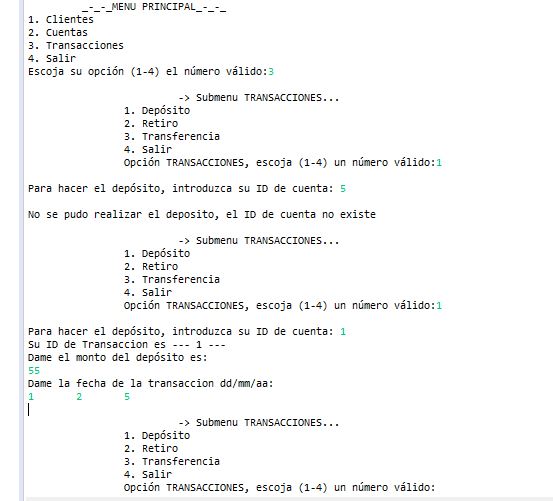
Se crean los archivos por primera y única vez dentro de la Carpeta1.



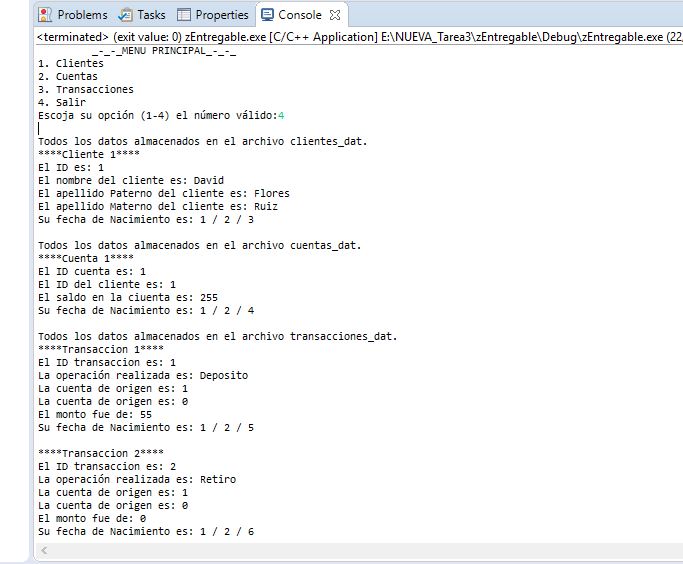
Entonces después de introducir la ruta muestra el menú y ya se da de alta los datos para el primer Nuevo Cliente con su ID aleatorio y que no se repita.



Dar de alta la primera Cuenta solo si el ID\_cliente ya existe.



Se da de alta la cuenta con un ID de cuenta válido (no de cliente porque un cliente porque las transacciones repercuten en el archivo cuentas y además porque un mismo cliente puede tener varias cuentas y se necesita especificar en cual se quiere realizar la transacción).



Impresión de lo todo lo que se guardó en los archivos.

Conclusiones (obligatorio):

* Lo que aprendí con esta práctica. Lo que ya sabía.

Logré abrir los archivos exactamente como lo describe la tarea y cualquier ruta no valida cuando no existen los archivos se crea por primera y única vez el archivo mydb\_sys y dentro de él se escribe la ruta correcta e incorrecta luego de que ya la escribió la lee una vez y revisa si tiene exactamente el formato de la carpeta ya existente que es Carpeta1\\ y mientras el formato de la comparación no sea correcto te sigue pidiendo la ruta la vuelve a escribir en el archivo y luego la vuelve a leer para compararla nuevamente y hasta que la comparación sea exitosa; entonces muestra el menú y quedando escrito en mydb\_sys solamente la ruta correcta.

Comprendí mucho más como es que se usan los archivos, entendí que cada vez que abro un archivo existente o por primera vez es importante guardar la referencia al archivo en una variable de tipo FILE porque mientras intente escribir o leer tengo que usar siempre la referencia al archivo, ya que no necesariamente es la misma que al crearlo, también me di cuenta que yo intentaba abrir archivos y trabajar en ellos y hasta el final cerrar todos pero de esta forma no podía visualizar los cambios que iba realizando; entonces hice muchos fopen y fclose a los archivos para así poder ver los cambios en él.

Otra cosa que fue muy importante es dominar que es lo que hace cada Modo de apertura del archivo; es diferente añadir a escribir ya que este último reemplaza lo que hay escrito y hacer un buen y correcto uso del Acceso Aleatorio para poner el cursor en el lugar exacto donde se desea escribir o borrar, etc.

También aprendí que al crear un archivo solo con el nombre y el modo; se crea en el directorio raíz es decir, en la carpeta donde se encuentra el actual proyecto; y si se desea crear en una carpeta más adentro entonces es necesario escribir en el lugar donde va la ruta el nombre de la carpeta ya existente y en seguida \\ o /, y después junto el nombre del archivo.

* Lo que me costó trabajo y como lo solucioné.

Me costó trabajo crear los archivos y también que validara la ruta correcta, y que cuando ya hubiera creado los archivos, no volviera a crear archivos en la Raíz sino que se abriera en los ya existente.

Fue también difícil repercutir los movimientos en las transferencias en el archivo cuentas\_dat y en transferencias\_dat; tuve que crear mi propia parte de código para leer estructuras dentro de un archivo con fread, porque no me salía usando las funciones existentes y no la pude hacer en función porque aunque podía mandar la ruta y el archivo, necesitaba también el tipo de dato estructurado del arreglo y no pude mandarlo a la función para recibirlo; por eso lo deje como una pequeña parte de código que reutilizaba algunas veces.

* Lo que no pude solucionar.

También otra cosa que hasta al momento se me dificulta luego de escribir uno por uno las altas en el archivo, fue el hecho de como reemplazar físicamente en el archivo el ID por -5 (borrado lógico); ya que en un arreglo virtual si lo hace pero no logro repercutirlo.

Fue difícil utilizar el acceso aleatorio moviendo el cursor del archivo con fseek, dado que yo antes no lo había utilizado nunca y resultaba casi obligado usarlo para modificar valores en los archivos.