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: Maris Mora, Michel

Presentación: 5 pts.
Funcionalidad: 50 pts.
Pruebas: 20 pts.

12 de junio de 2018. Tlaquepaque, Jalisco,

- Falta describir las pruebas (escenario, y resultados de la experimentación).
- Tienes algunos detalles en el tema de funcionalidad, verifica lo que se debe imprimir cuando consultas una cuenta.

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:

A) 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

```
//Directivas al preprocesador
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
//Variables Globales y nuevos tipos de datos

FILE * fcliente;
FILE * fcuentas;
FILE * ftransacciones;
int TF;
typedef enum {
	False = 0, True
} bool;
typedef struct {
	int dia;
	int mes;
	int anio;
} Fecha;
typedef struct {
	long ID_Usuario;
	char Nombre[50];
	char ApellidoP[50];
	char ApellidoM[50];
	Fecha FechaNacimiento;
} Usuario;
typedef struct {
	long ID_Cuenta, ID_Usuario, Saldo;
	Fecha FechaApertura;
} Cuenta;
typedef struct {
	long ID_Transaccion, ID_CuentaOrigen, ID_Cuenta_Destino,
MontoTransaccion;
	char TipoTransaccion[15];
	Fecha FechaTransaccion;
} Transaccion;
//prototipos de funciones
char* RutaAcceso(FILE *);
void NuevoCliente(char *);
void ImprimirCliente(char *);
long BuscarCliente(char*, long);
bool ComprobarCliente(char*, long);
void EliminarCliente(char*, long, char *);
void NuevaCuenta(char *, char *);
void BuscarCuenta(char *, long);
```

```
void ImprimirCuentas(char *);
void EliminarCuenta(char*, long);
bool Deposito(char *, char *, long);
bool ModificarCuenta(char*, long, long);
bool Retiro(char *, char *, long);
bool Transferencia(char *, char *, long, long);
//Función principal
int main() {
	//Configurando el buffer de salida
	setvbuf(stderr, NULL, _IONBF, 0);
	setvbuf(stdout, NULL, _IONBF, 0);
	//char CL[]={"//Clientes.dat"};
	//char CU[]={"//Cuentas.dat"};
	//char TR[]={"//Transacciones.dat"};

	FILE *Comprobar;
	Comprobar = fopen("mydb.sys", "r");
	int O, Case;
	long B, I;
	char *Ruta = (char *) malloc(sizeof(char *) * 260);
	Ruta = RutaAcceso(Comprobar);

	char * DireccionCliente = (char *) malloc(sizeof(char *) * 260);
	char * DireccionCuentas = (char *) malloc(sizeof(char *) * 260);
	char * DireccionTransacciones = (char *) malloc(sizeof(char *) *
260);
	strcpy(DireccionCliente, Ruta);
	strcpy(DireccionCuentas, Ruta);
	strcpy(DireccionTransacciones, Ruta);
	strcat(DireccionCliente, "\\Clientes.dat");
	strcat(DireccionCuentas, "\\Cuentas.dat");
	strcat(DireccionTransacciones, "\\Transacciones.dat");
	long x = 0, y = 0, z = 0;
	if (TF == False) {
		fcliente = fopen(DireccionCliente, "wb");
		fwrite(&x, sizeof(long), 1, fcliente);
		fclose(fcliente);
		fcuentas = fopen(DireccionCuentas, "wb");
		fwrite(&y, sizeof(long), 1, fcuentas);
		fclose(fcuentas);
		ftransacciones = fopen(DireccionTransacciones, "wb");
		fwrite(&z, sizeof(long), 1, ftransacciones);
		fclose(ftransacciones);
	}

	//printf("%s\n",Ruta);

	do {
		printf(
```

```
                        "<<Sistema
MyBD>>\n[1]Clientes\n[2]Cuentas\n[3]Transacciones\n[4]Salir\n");
            scanf("%d", &O);
            switch (O) {
            case 1:
                  printf(
                              "Clientes\n[1]Nuevo Cliente\n[2]Buscar
Cliente\n[3]Eliminar Cliente\n[4]Imprimir Clientes\n");
                  scanf("%d", &Case);
                  switch (Case) {
                  case 1:
                        NuevoCliente(DireccionCliente);
                        break;
                  case 2:
                        printf("Ingrese el ID del usuario que desea
buscar: ");
                        scanf("%li", &B);
                        I = BuscarCliente(DireccionCliente, B);
                        break;
                  case 3:
                        printf(
                                    "Ingrese el ID del usuario que desea
eliminar sus datos:");
                        scanf("%li", &B);
                        EliminarCliente(DireccionCliente, B,
DireccionCuentas);
                        break;
                  case 4:
                        ImprimirCliente(DireccionCliente);
                        break;
                  default:
                        printf("Error, favor de escoger una opción
valida\n");
                  }
                  break;
            case 2:
                  printf(
                              "Cuentas\n[1]Nueva Cuenta\n[2]Buscar
Cuenta\n[3]Eliminar Cuenta\n[4]Imprimir Cuenta\n");
                  scanf("%d", &Case);
                  switch (Case) {
                  case 1:
                        NuevaCuenta(DireccionCuentas, DireccionCliente);
                        break;
                  case 2:
                        printf("Ingrese el ID de la cuenta que desea
buscar: ");
                        scanf("%li", &B);
                        BuscarCuenta(DireccionCuentas, B);
                        break;
```

```
                  case 3:
                        printf("Ingrese el ID de la cuenta que desea
eliminar: ");
                        scanf("%li", &B);
                        EliminarCuenta(DireccionCuentas, B);
                        break;
                  case 4:
                        ImprimirCuentas(DireccionCuentas);
                        break;
                  default:
                        printf("Error, favor de escoger una opción
valida\n");
                  }
                  break;
            case 3:
                  printf(

      "Transacciones\n[1]Depósitos\n[2]Retiro\n[3]Transferencia\n");
                  scanf("%d", &Case);
                  switch (Case) {
                  case 1:
                        printf(
                                    "Ingrese el ID de la cuenta en la que
realizará el depósito: ");
                        scanf("%li", &B);
                        Deposito(DireccionTransacciones,
DireccionCuentas, B);
                        break;
                  case 2:
                        printf(
                                    "Ingrese el ID de la cuenta en la que
realizará el retiro: ");
                        scanf("%li", &B);
                        Retiro(DireccionTransacciones, DireccionCuentas,
B);
                        break;
                  case 3:
                        printf("Ingrese el ID de la cuenta origen: ");
                        scanf("%li", &B);
                        printf("Ingrese el ID de la cuenta destino");
                        scanf("%li", &I);
                        Transferencia(DireccionTransacciones,
DireccionCuentas, B, I);
                        break;
                  default:
                        printf("Error, favor de escoger una opción
valida\n");
                  }
                  break;
            default:
```

```
                if (O != 4)
                      printf("Error, favor de escoger una opción
valida\n");
            }
      } while (O != 4);

      return 0;
}
//Desarrollo de funciones
char* RutaAcceso(FILE *Comprobar) {
      char *Ruta = (char *) malloc(sizeof(char *) * 260);
      if (Comprobar != NULL) {
            fgets(Ruta, 260, Comprobar);
            fclose(Comprobar);
            TF = True;
            return Ruta;
      } else {
            fclose(Comprobar);
            Comprobar = fopen("mydb.sys", "w");
            printf(
                        "Error al detectar el archivo, favor de ingresar
la direccion en la que se encontraran sus archivos\n");
            //C:\Users\Michel\eclipse-workspace\Tarea-03PMD
            fflush(stdin);
            gets(Ruta);
            fputs(Ruta, Comprobar);
            fclose(Comprobar);
            TF = False;
            return Ruta;
      }
}
void NuevoCliente(char*DireccionCliente) {
      Usuario Cliente;
      long i;
      fcliente = fopen(DireccionCliente, "rb");
      fread(&i, sizeof(long), 1, fcliente);
      fclose(fcliente);
      fcliente = fopen(DireccionCliente, "a+b");
      printf("Nombre: ");
      fflush(stdin);
      gets(Cliente.Nombre);
      printf("Apellido Paterno: ");
      fflush(stdin);
      gets(Cliente.ApellidoP);
      printf("Apellido Materno: ");
      fflush(stdin);
      gets(Cliente.ApellidoM);
      printf("Fecha de nacimiento [dd/mm/aaaa]: ");
      scanf("%d/%d/%d", &Cliente.FechaNacimiento.dia,
```

```
                  &Cliente.FechaNacimiento.mes,
&Cliente.FechaNacimiento.anio);
      i++;
      Cliente.ID_Usuario = i;
      fwrite(&Cliente, sizeof(Usuario), 1, fcliente);
      fclose(fcliente);
      fcliente = fopen(DireccionCliente, "r+b");
      fwrite(&i, sizeof(long), 1, fcliente);
      fclose(fcliente);
}
long BuscarCliente(char*DireccionCliente, long ID) {
      long i;
      Usuario Buscar;
      fcliente = fopen(DireccionCliente, "rb");
      fread(&i, sizeof(long), 1, fcliente);
      printf("%li\n", i);
      do {
            fread(&Buscar, sizeof(Usuario), 1, fcliente);
            //printf("%li\n",Buscar.ID_Usuario);
            if (Buscar.ID_Usuario >= i) {
                  break;
            }

      } while (Buscar.ID_Usuario != ID);
      //printf("%li",Buscar.ID_Usuario);
      if (ID <= i && ID > 0) {
            printf("%li %s %s %s {%d/%d/%d}\n", Buscar.ID_Usuario,
Buscar.Nombre,
                        Buscar.ApellidoP, Buscar.ApellidoM,
Buscar.FechaNacimiento.dia,
                        Buscar.FechaNacimiento.mes,
Buscar.FechaNacimiento.anio);
            fclose(fcliente);
            return ID;
      } else {
            printf("Error, el id buscado no existe\n");
            fclose(fcliente);
            return 0;
      }
}
bool ComprobarCliente(char* DireccionCliente, long ID) {
      long i;
      Usuario Comprobar;
      fcliente = fopen(DireccionCliente, "rb");
      fread(&i, sizeof(long), 1, fcliente);
      //printf("%li\n",i);
      //printf("%li \n",i);
      do {
            fread(&Comprobar, sizeof(Usuario), 1, fcliente);
```

```
            //printf("%li %s %s %s {%d/%d/%d}\n", Comprobar.ID_Usuario,
Comprobar.Nombre,Comprobar.ApellidoP,
Comprobar.ApellidoM,Comprobar.FechaNacimiento.dia,
Comprobar.FechaNacimiento.mes,Comprobar.FechaNacimiento.anio);
            //printf("%li\n",Comprobar.ID_Usuario);
            if (Comprobar.ID_Usuario >= i)
                  break;
      } while (Comprobar.ID_Usuario != ID);
      fclose(fcliente);
      if (ID <= i && ID > 0) {
            //printf("%li \n",I);
            return True;
      } else {
            //printf("%li \n",I);
            return False;
      }
}
void EliminarCliente(char* DireccionCliente, long ID, char *
DireccionCuentas) {
      long i, j = 0, icuentas = 0;
      fcuentas = fopen(DireccionCuentas, "rb");
      fread(&i, sizeof(long), 1, fcuentas);
      Cuenta cuentas[i];
      fclose(fcuentas);
      fcliente = fopen(DireccionCliente, "rb");
      fread(&i, sizeof(long), 1, fcliente);
      //printf("%li\n",i);
      //Usuario *Clientes=(Usuario *)malloc(sizeof(Usuario)*i);
      Usuario Clientes[i];
      Usuario Cliente;
      Cuenta cuenta;
      while (fread(&Cliente, sizeof(Usuario), 1, fcliente)) {
            Clientes[j].ID_Usuario = Cliente.ID_Usuario;
            strcpy(Clientes[j].Nombre, Cliente.Nombre);
            strcpy(Clientes[j].ApellidoP, Cliente.ApellidoP);
            strcpy(Clientes[j].ApellidoM, Cliente.ApellidoM);
            Clientes[j].FechaNacimiento.dia =
Cliente.FechaNacimiento.dia;
            Clientes[j].FechaNacimiento.mes =
Cliente.FechaNacimiento.mes;
            Clientes[j].FechaNacimiento.anio =
Cliente.FechaNacimiento.anio;
            j++;
      }
      for (int k = 0; k < j; k++) {
            if (Clientes[k].ID_Usuario == ID) {
                  Clientes[k].ID_Usuario = 0;
            }
      }
      fclose(fcliente);
```

```
	fcuentas = fopen(DireccionCuentas, "rb");
	fread(&icuentas, sizeof(long), 1, fcuentas);
	long l = 0;
	while (fread(&cuenta, sizeof(Cuenta), 1, fcuentas)) {
		//printf("ID: %li\n", cuenta.Saldo);
		if (cuenta.ID_Usuario == ID) {
			//printf("%li %li\n", cuenta.ID_Cuenta,l);
			cuentas[l].ID_Cuenta = cuenta.ID_Cuenta;
			l++;
		}
		//EliminarCuenta(DireccionCuentas, cuentas.ID_Usuario);
	}
	//printf("%li",j);
	fclose(fcuentas);
	for (int k = 0; k < l; k++) {
		printf("Hola\n");
		EliminarCuenta(DireccionCuentas, cuentas[k].ID_Cuenta);
	}
	fcliente = fopen(DireccionCliente, "wb");
	fwrite(&i, sizeof(long), 1, fcliente);
	for (int k = 0; k < j; k++) {
		Cliente.ID_Usuario = Clientes[k].ID_Usuario;
		strcpy(Cliente.Nombre, Clientes[k].Nombre);
		strcpy(Cliente.ApellidoP, Clientes[k].ApellidoP);
		strcpy(Cliente.ApellidoM, Clientes[k].ApellidoM);
		Cliente.FechaNacimiento.dia =
Clientes[k].FechaNacimiento.dia;
		Cliente.FechaNacimiento.mes =
Clientes[k].FechaNacimiento.mes;
		Cliente.FechaNacimiento.anio =
Clientes[k].FechaNacimiento.anio;
		//printf("%li\n", Clientes[k].ID_Usuario);
		if (Clientes[k].ID_Usuario != 0)
			fwrite(&Cliente, sizeof(Usuario), 1, fcliente);
	}
	fclose(fcliente);
}
void ImprimirCliente(char*DireccionCliente) {
	fcliente = fopen(DireccionCliente, "rb");
	Usuario Cliente;
	long i;
	fread(&i, sizeof(long), 1, fcliente);
	printf("%li\n", i);
	printf("ID Usuario-Nombre-Fecha Nacimiento\n");
	/*while(!feof(fcliente))*/while (fread(&Cliente, sizeof(Usuario),
1,
			fcliente)) {
		//fread(&Cliente,sizeof(Usuario),1,fcliente);
		printf("%li-%s %s %s-{%d/%d/%d}\n", Cliente.ID_Usuario,
Cliente.Nombre,
```

```
                        Cliente.ApellidoP, Cliente.ApellidoM,
                        Cliente.FechaNacimiento.dia,
Cliente.FechaNacimiento.mes,
                        Cliente.FechaNacimiento.anio);
      }
      fclose(fcliente);
}
void NuevaCuenta(char *DireccionCuentas, char *DireccionCliente) {
      Cuenta cuenta;
      long i, IDU = 0;
      fcuentas = fopen(DireccionCuentas, "rb");
      fread(&i, sizeof(long), 1, fcuentas);
      fclose(fcuentas);
      while (!IDU) {
            printf("ID del usuario: ");
            scanf("%li", &cuenta.ID_Usuario);
            IDU = cuenta.ID_Usuario;
            IDU = ComprobarCliente(DireccionCliente, IDU);
            if (IDU == False) {
                  printf("Error, Cliente no existente\n");
            }
      }
      printf("Saldo: ");
      scanf("%li", &cuenta.Saldo);
      printf("Fecha de apertura [dd/mm/aaaa]: ");
      scanf("%d/%d/%d", &cuenta.FechaApertura.dia,
&cuenta.FechaApertura.mes,
                  &cuenta.FechaApertura.anio);
      i++;

      cuenta.ID_Cuenta = i;
      fcuentas = fopen(DireccionCuentas, "a+b");
      fwrite(&cuenta, sizeof(Cuenta), 1, fcuentas);
      fclose(fcuentas);
      fcuentas = fopen(DireccionCuentas, "r+b");
      fwrite(&i, sizeof(long), 1, fcuentas);
      fclose(fcuentas);

}
void BuscarCuenta(char *DireccionCuentas, long ID) {
      long i;
      Cuenta Buscar;
      fcuentas = fopen(DireccionCuentas, "rb");
      fread(&i, sizeof(long), 1, fcuentas);
      //printf("%li\n",i);
      do {
            fread(&Buscar, sizeof(Cuenta), 1, fcuentas);
            if (Buscar.ID_Cuenta >= i)
                  break;
      } while (Buscar.ID_Cuenta != ID);
```

```
	//printf("%li",Buscar.ID_Usuario);
	if (ID <= i && ID >= 0) {
		printf("%li %li $ %li {%d/%d/%d}\n", Buscar.ID_Cuenta,
				Buscar.ID_Usuario, Buscar.Saldo,
Buscar.FechaApertura.dia,
				Buscar.FechaApertura.mes,
Buscar.FechaApertura.anio);
		fclose(fcuentas);
	} else {
		printf("Error, el id buscado no existe\n");
		fclose(fcuentas);
	}
}
void ImprimirCuentas(char*DireccionCuentas) {
	fcuentas = fopen(DireccionCuentas, "rb");
	Cuenta Imprimir;
	long i;
	fread(&i, sizeof(long), 1, fcuentas);
	printf("%li\n", i);
	printf("ID Cuenta-ID Usuario-ID Saldo-Fecha Apertura\n");
	/*while(!feof(fcliente))*/while (fread(&Imprimir, sizeof(Cuenta),
1,
			fcuentas)) {
		//fread(&Cliente,sizeof(Usuario),1,fcliente);
		printf("%li-%li-$%li-{%d/%d/%d}\n", Imprimir.ID_Cuenta,
				Imprimir.ID_Usuario, Imprimir.Saldo,
Imprimir.FechaApertura.dia,
				Imprimir.FechaApertura.mes,
Imprimir.FechaApertura.anio);
	}
	fclose(fcuentas);
}
void EliminarCuenta(char* DireccionesCuentas, long ID) {
	long i, j = 0;
	fcuentas = fopen(DireccionesCuentas, "rb");
	fread(&i, sizeof(long), 1, fcuentas);
	//printf("%li\n",i);
	//Usuario *Clientes=(Usuario *)malloc(sizeof(Usuario)*i);
	Cuenta cuentas[i];
	Cuenta cuenta;
	while (fread(&cuenta, sizeof(Cuenta), 1, fcuentas)) {
		cuentas[j].ID_Cuenta = cuenta.ID_Cuenta;
		cuentas[j].ID_Usuario = cuenta.ID_Usuario;
		cuentas[j].Saldo = cuenta.Saldo;
		cuentas[j].FechaApertura.dia = cuenta.FechaApertura.dia;
		cuentas[j].FechaApertura.mes = cuenta.FechaApertura.mes;
		cuentas[j].FechaApertura.anio = cuenta.FechaApertura.anio;
		j++;
	}
	for (int k = 0; k < j; k++) {
```

```
            if (cuentas[k].ID_Cuenta == ID) {
                  cuentas[k].ID_Cuenta = 0;
            }
      }
      //printf("%li",j);
      fclose(fcuentas);
      fcuentas = fopen(DireccionesCuentas, "wb");
      fwrite(&i, sizeof(long), 1, fcuentas);
      for (int k = 0; k < j; k++) {
            cuenta.ID_Cuenta = cuentas[k].ID_Cuenta;
            cuenta.ID_Usuario = cuentas[k].ID_Usuario;
            cuenta.Saldo = cuentas[k].Saldo;
            cuenta.FechaApertura.dia = cuentas[k].FechaApertura.dia;
            cuenta.FechaApertura.mes = cuentas[k].FechaApertura.mes;
            cuenta.FechaApertura.anio = cuentas[k].FechaApertura.anio;
            //printf("%li\n", Clientes[k].ID_Usuario);
            if (cuentas[k].ID_Cuenta != 0)
                  fwrite(&cuenta, sizeof(Cuenta), 1, fcuentas);
      }
      fclose(fcuentas);
}
bool Deposito(char *DireccionTransacciones, char *DireccionCuentas, long
ID) {
      Cuenta Verifica;
      Transaccion Tr;
      long i,j,V=0;
      ftransacciones = fopen(DireccionTransacciones, "rb");
      fread(&i, sizeof(long), 1, ftransacciones);
      fclose(fcliente);
      fcuentas=fopen(DireccionCuentas,"rb");
      fread(&j,sizeof(long),1,fcuentas);
      while(fread(&Verifica,sizeof(Cuenta),1,fcuentas)){
            if(Verifica.ID_Cuenta==ID)
                  V++;
      }
      if(V==0){
            printf("El ID de cuenta ingresado es erroneo\n");
            return False;
      }
      fclose(fcuentas);
      strcpy(Tr.TipoTransaccion, "Depósito");
      Tr.ID_CuentaOrigen = ID;
      Tr.ID_Cuenta_Destino = (long) NULL;
      printf("Ingrese el monto de la transacción: ");
      scanf("%li", &Tr.MontoTransaccion);
      printf("Ingrese la fecha en la que se realizó el depósito
[dd/mm/aaaa]: ");
      scanf("%d/%d/%d", &Tr.FechaTransaccion.dia,
&Tr.FechaTransaccion.mes,
                  &Tr.FechaTransaccion.anio);
```

```
	i++;
	Tr.ID_Transaccion = i;
	ftransacciones = fopen(DireccionTransacciones, "a+b");
	fwrite(&Tr, sizeof(Transaccion), 1, ftransacciones);
	fclose(ftransacciones);
	ftransacciones = fopen(DireccionTransacciones, "r+b");
	fwrite(&i, sizeof(long), 1, ftransacciones);
	fclose(ftransacciones);
	ModificarCuenta(DireccionCuentas, ID, Tr.MontoTransaccion);
	return True;
}
bool ModificarCuenta(char* DireccionCuentas, long ID, long Monto) {
	long i, j = 0;
	fcuentas = fopen(DireccionCuentas, "rb");
	fread(&i, sizeof(long), 1, fcuentas);
	//printf("%li\n",i);
	//Usuario *Clientes=(Usuario *)malloc(sizeof(Usuario)*i);
	Cuenta cuentas[i];
	Cuenta cuenta;
	while (fread(&cuenta, sizeof(Cuenta), 1, fcuentas)) {
		cuentas[j].ID_Cuenta = cuenta.ID_Cuenta;
		cuentas[j].ID_Usuario = cuenta.ID_Usuario;
		cuentas[j].Saldo = cuenta.Saldo;
		cuentas[j].FechaApertura.dia = cuenta.FechaApertura.dia;
		cuentas[j].FechaApertura.mes = cuenta.FechaApertura.mes;
		cuentas[j].FechaApertura.anio = cuenta.FechaApertura.anio;
		j++;
	}
	for (int k = 0; k < j; k++) {
		if (cuentas[k].ID_Cuenta == ID) {
			if (cuentas[k].Saldo + Monto < 0)
				return False;
			else
				cuentas[k].Saldo += Monto;
		}
	}
	//printf("%li",j);
	fclose(fcuentas);
	fcuentas = fopen(DireccionCuentas, "wb");
	fwrite(&i, sizeof(long), 1, fcuentas);
	for (int k = 0; k < j; k++) {
		cuenta.ID_Cuenta = cuentas[k].ID_Cuenta;
		cuenta.ID_Usuario = cuentas[k].ID_Usuario;
		cuenta.Saldo = cuentas[k].Saldo;
		cuenta.FechaApertura.dia = cuentas[k].FechaApertura.dia;
		cuenta.FechaApertura.mes = cuentas[k].FechaApertura.mes;
		cuenta.FechaApertura.anio = cuentas[k].FechaApertura.anio;
		//printf("%li\n", Clientes[k].ID_Usuario);
		fwrite(&cuenta, sizeof(Cuenta), 1, fcuentas);
	}
```

```
	fclose(fcuentas);
	return True;
}
bool Retiro(char *DireccionTransacciones, char *DireccionCuentas, long
ID) {
	Transaccion Tr;
	Cuenta Verifica;
	long i,j,V=0;
	int TrFl = 1;
	ftransacciones = fopen(DireccionTransacciones, "rb");
	fread(&i, sizeof(long), 1, ftransacciones);
	fclose(fcliente);
	fcuentas=fopen(DireccionCuentas,"rb");
		fread(&j,sizeof(long),1,fcuentas);
		while(fread(&Verifica,sizeof(Cuenta),1,fcuentas)){
			if(Verifica.ID_Cuenta==ID)
				V++;
		}
		if(V==0){
			printf("El ID de cuenta ingresado es erroneo\n");
			return False;
		}

	strcpy(Tr.TipoTransaccion, "Retiro");
	Tr.ID_CuentaOrigen = ID;
	Tr.ID_Cuenta_Destino = (long) NULL;
	CR: printf("Ingrese el monto a retirar: ");
	scanf("%li", &Tr.MontoTransaccion);
	Tr.MontoTransaccion *= -1;
	TrFl = ModificarCuenta(DireccionCuentas, ID, Tr.MontoTransaccion);
	if (TrFl == False) {
		printf(
				"Error, el monto seleccionado sobrepasa el retiro
que se puede realizar");
		goto CR;
	}
	printf("Ingrese la fecha en la que se realizó el Retiro
[dd/mm/aaaa]: ");
	scanf("%d/%d/%d", &Tr.FechaTransaccion.dia,
&Tr.FechaTransaccion.mes,
			&Tr.FechaTransaccion.anio);
	i++;
	Tr.ID_Transaccion = i;
	ftransacciones = fopen(DireccionTransacciones, "a+b");
	fwrite(&Tr, sizeof(Transaccion), 1, ftransacciones);
	fclose(ftransacciones);
	ftransacciones = fopen(DireccionTransacciones, "r+b");
	fwrite(&i, sizeof(long), 1, ftransacciones);
	fclose(ftransacciones);
	return True;
```

```
}
bool Transferencia(char *DireccionTransacciones, char *DireccionCuentas,
            long IDOrigen, long IDDestino) {
      Cuenta Verifica;
      Transaccion Tr;
      long i,j,V=0;
      int TrFl = 1;
      ftransacciones = fopen(DireccionTransacciones, "rb");
      fread(&i, sizeof(long), 1, ftransacciones);
      fclose(fcliente);
      fcuentas=fopen(DireccionCuentas,"rb");
            fread(&j,sizeof(long),1,fcuentas);
            while(fread(&Verifica,sizeof(Cuenta),1,fcuentas)){
                  if(Verifica.ID_Cuenta==IDOrigen ||
Verifica.ID_Cuenta==IDDestino)
                        V++;
            }
            if(V==0){
                  printf("El ID de cuenta ingresado es erroneo\n");
                  return False;
            }


      strcpy(Tr.TipoTransaccion, "Transferencia");
      Tr.ID_CuentaOrigen = IDOrigen;
      Tr.ID_Cuenta_Destino = IDDestino;
      CR: printf("Ingrese el monto a que será transferido: ");
      scanf("%li", &Tr.MontoTransaccion);
      Tr.MontoTransaccion *= -1;
      TrFl = ModificarCuenta(DireccionCuentas, IDOrigen,
Tr.MontoTransaccion);
      if (TrFl == False) {
            printf(
                        "Error, el monto seleccionado sobrepasa el retiro
que se puede realizar");
            goto CR;
      }
      Tr.MontoTransaccion *= -1;
      TrFl = ModificarCuenta(DireccionCuentas, IDDestino,
Tr.MontoTransaccion);
      printf("Ingrese la fecha en la que se realizó el Retiro
[dd/mm/aaaa]: ");
      scanf("%d/%d/%d", &Tr.FechaTransaccion.dia,
&Tr.FechaTransaccion.mes,
                  &Tr.FechaTransaccion.anio);
      i++;
      Tr.ID_Transaccion = i;
      ftransacciones = fopen(DireccionTransacciones, "a+b");
      fwrite(&Tr, sizeof(Transaccion), 1, ftransacciones);
      fclose(ftransacciones);
```

```
	ftransacciones = fopen(DireccionTransacciones, "r+b");
	fwrite(&i, sizeof(long), 1, ftransacciones);
	fclose(ftransacciones);
	return True;
}
```

Ejecución

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

```
Error al detectar el archivo, favor de ingresar la direccion en la que se encontraran sus archivos
C:\Users\Michel\eclipse-workspace\Tarea-03PMD
<<Sistema MyBD>>
[1]Clientes
[2]Cuentas
[3]Transacciones
[4]Salir
<<Sistema MyBD>>
[1]Clientes
[2]Cuentas
[3]Transacciones
[4]Salir
1
Clientes
[1]Nuevo Cliente
[2]Buscar Cliente
[3]Eliminar Cliente
[4]Imprimir Clientes
1
Nombre: Roberto
Apellido Paterno: Zepeda
Apellido Materno: Hipólito
Fecha de nacimiento [dd/mm/aaaa]: 4/9/1998
Clientes
[1]Nuevo Cliente
[2]Buscar Cliente
[3]Eliminar Cliente
[4]Imprimir Clientes
2
Ingrese el ID del usuario que desea buscar: 3
7
3 Andrea Anahí Santana Hernandez {22/9/1997}
<<Sistema MyBD>>
[1]Clientes
[2]Cuentas
[3]Transacciones
[4]Salir
```

```
Clientes
[1]Nuevo Cliente
[2]Buscar Cliente
[3]Eliminar Cliente
[4]Imprimir Clientes
3
Ingrese el ID del usuario que desea eliminar sus datos:3
```

```
<<Sistema MyBD>>
[1]Clientes
[2]Cuentas
[3]Transacciones
[4]Salir
1
Clientes
[1]Nuevo Cliente
[2]Buscar Cliente
[3]Eliminar Cliente
[4]Imprimir Clientes
4
7
ID Usuario-Nombre-Fecha Nacimiento
1-Martin Casillas Ramos-{18/6/1998}
2-Michel Maris Mora-{9/2/1999}
4-Edgar Francisco Medina Rifas-{4/11/1999}
5-Mauricio Durán Padilla-{21/10/1998}
6-Ignacio Vazquez Martines-{5/5/1999}
7-Roberto Zepeda Hipólito-{4/9/1998}
```

```
<<Sistema MyBD>>
[1]Clientes
[2]Cuentas
[3]Transacciones
[4]Salir
2
Cuentas
[1]Nueva Cuenta
[2]Buscar Cuenta
[3]Eliminar Cuenta
[4]Imprimir Cuenta
1
ID del usuario: 5
Saldo: 4550
Fecha de apertura [dd/mm/aaaa]: 3/5/2011
<<Sistema MyBD>>
[1]Clientes
[2]Cuentas
[3]Transacciones
[4]Salir
2
Cuentas
[1]Nueva Cuenta
[2]Buscar Cuenta
[3]Eliminar Cuenta
[4]Imprimir Cuenta
2
Ingrese el ID de la cuenta que desea buscar: 1
1 1 $ 100 {22/6/2018}
```

```
Cuentas
[1]Nueva Cuenta
[2]Buscar Cuenta
[3]Eliminar Cuenta
[4]Imprimir Cuenta
3
Ingrese el ID de la cuenta que desea eliminar: 1
<<Sistema MyBD>>
[1]Clientes
[2]Cuentas
[3]Transacciones
[4]Salir
2
Cuentas
[1]Nueva Cuenta
[2]Buscar Cuenta
[3]Eliminar Cuenta
[4]Imprimir Cuenta
4
8
ID Cuenta-ID Usuario-ID Saldo-Fecha Apertura
2-2-$2000-{2/3/2001}
3-1-$3000-{10/10/2012}
4-4-$2500-{30/12/2004}
6-6-$3500-{9/2/2000}
7-5-$500-{22/4/2007}
8-5-$4550-{3/5/2011}
<<Sistema MyBD>>
[1]Clientes
[2]Cuentas
[3]Transacciones
[4]Salir
1
Clientes
[1]Nuevo Cliente
[2]Buscar Cliente
[3]Eliminar Cliente
[4]Imprimir Clientes
3
Ingrese el ID del usuario que desea eliminar sus datos:5
```

```
<<Sistema MyBD>>
[1]Clientes
[2]Cuentas
[3]Transacciones
[4]Salir
2
Cuentas
[1]Nueva Cuenta
[2]Buscar Cuenta
[3]Eliminar Cuenta
[4]Imprimir Cuenta
4
8
ID Cuenta-ID Usuario-ID Saldo-Fecha Apertura
2-2-$2000-{2/3/2001}
3-1-$3000-{10/10/2012}
4-4-$2500-{30/12/2004}
6-6-$3500-{9/2/2000}
```

```
<<Sistema MyBD>>
[1]Clientes
[2]Cuentas
[3]Transacciones
[4]Salir
3
Transacciones
[1]Depósitos
[2]Retiro
[3]Transferencia
1
Ingrese el ID de la cuenta en la que realizará el depósito: 2
Ingrese el monto de la transacción: 2000
Ingrese la fecha en la que se realizó el depósito [dd/mm/aaaa]: 22/06/2018
<<Sistema MyBD>>
[1]Clientes
[2]Cuentas
[3]Transacciones
[4]Salir
2
Cuentas
[1]Nueva Cuenta
[2]Buscar Cuenta
[3]Eliminar Cuenta
[4]Imprimir Cuenta
4
8
ID Cuenta-ID Usuario-ID Saldo-Fecha Apertura
2-2-$4000-{2/3/2001}
3-1-$3000-{10/10/2012}
4-4-$2500-{30/12/2004}
6-6-$3500-{9/2/2000}
```

```
Ingrese el ID de la cuenta en la que realizará el retiro: 3
Ingrese el monto a retirar: 1500
Ingrese la fecha en la que se realizó el Retiro [dd/mm/aaaa]: 22/06/2018
<<Sistema MyBD>>
[1]Clientes
[2]Cuentas
[3]Transacciones
[4]Salir
3
Transacciones
[1]Depósitos
[2]Retiro
[3]Transferencia
3
Ingrese el ID de la cuenta origen: 2
Ingrese el ID de la cuenta destino6
Ingrese el monto a que será transferido: 2000
Ingrese la fecha en la que se realizó el Retiro [dd/mm/aaaa]: 21/5/2018
ID Cuenta-ID Usuario-ID Saldo-Fecha Apertura
2-2-$2000-{2/3/2001}
3-1-$1500-{10/10/2012}
4-4-$2500-{30/12/2004}
6-6-$5500-{9/2/2000}
```

Conclusiones (obligatorio):

- ✓ Lo que aprendí con esta práctica. Lo que ya sabía.
  - Con la realización de esta práctica, aprendí a implementar las funciones que son utilizadas en el manejo de archivos y mediante esto generar mis propias funciones que por medio de lógica permiten realizar distintas acciones, antes de realizar la práctica tenía un mínimo conocimiento sobre apertura e ingreso de datos en archivos, pero ahora se me facilita utilizarlos y moverme dentro de ellos.
- ✓ Lo que me costó trabajo y cómo lo solucioné.
  - Lo que más me costó trabajo solucionar fue el identificar pequeños errores durante la implementación, debido a que aunque 1 sola variable fuese mal utilizada, toda la implementación llegaba a fallar, también se me dificultó un poco la implementación de lógica pero con observar bien los llegué a solucionar.
- ✓ Lo que no pude solucionar.