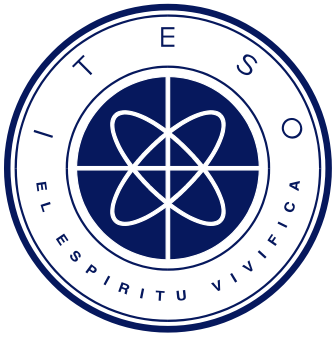
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Departamento de Electrónica, Sistemas e Informática

INGENIERÍA EN SISTEMAS COMPUTACIONALES



PROGRAMACIÓN CON MEMORIA DINÁMICA

TAREA 2. MEMORIA DINÁMICA Y ARCHIVOS

Autor: Galeana Torres, Fernanda

Presentación: 5 pts.
Funcionalidad: 45 pts.
Pruebas: 20 pts.

12 de junio de 2018. Tlaquepaque, Jalisco,

- Falta describir las pruebas (escenario, y resultados de la experimentación).
- Sobre la funcionalidad: verificar la operación de borrado y modificación de datos en los archivos.

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

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef enum{false,true,other} bool;


typedef struct{
	int d,m,a;
}Fecha;

typedef struct{
	int id_usuario;
	char nombre[20];
	char apellido_m[20];
	char apellido_p[20];
```

```
	Fecha f_nac;
}Cliente;

typedef struct{
	int id_cuenta;
	int id_usuario;
	int saldo;
	Fecha f_apertura;
}Cuenta;

typedef struct{
	int id_tr;
	char tipo[15];
	int id_c_origen;
	int id_c_destino;
	Fecha f_tr;
	int monto;
}Transaccion;

void Clientes(char ruta[], int*);
void NuevoCliente(char r[]);
bool BuscarCliente(char r[], int,bool print);
bool EliminarCliente(char r[], char r_cuentas[],int*);
bool ImprimirClientes(char r[]);
void Cuentas(char ruta[], int*);
bool NuevaCuenta(char r[],char r_clientes[], int*);
bool BuscarCuenta(char r[], char r_cl[], int, bool print);
bool EliminarCuenta(char r[]);
bool ImprimirCuentas(char r[], char r_cl[]);
void Transacciones(char ruta[]);
bool NewTr(char r[], char ruta_cuenta[], int tipo);
Fecha GetDate();
void PrintDate(Fecha f);
bool VerifSaldo(Transaccion, char r[]);
void ModifSaldo(char r[], Transaccion tr, int tipo);
bool CuentasCliente(char r_cu[], int id, int*);

int main(){
	setvbuf(stderr, NULL, _IONBF, 0);
	setvbuf(stdout, NULL, _IONBF, 0);
	FILE *mydb;
	int* total;
	int opc=0;
	char ruta[100];
	mydb=fopen("mydb.sys","r+");
	if(mydb==NULL){
		fclose(mydb);
		printf("Bienvenido al sistema MyDB. \nIngrese la ruta de acceso
donde desea crear los archivos: ");
		gets(ruta);
		mydb=fopen("mydb.sys","w");
		fputs(ruta,mydb);
		fclose(mydb);
		mydb=fopen("mydb.sys","r+");
	}
```

```
	fscanf(mydb,"%s",ruta);
	//fscanf(mydb,"%d",*total);
	//puts(ruta);
	while(opc!=4){
		//mydb=fopen(ruta,"r+");
		//printf("Total de cuentas: %d\n", *total);
		printf("<<Sistema
MyDB>>\n[1]Clientes\n[2]Cuentas\n[3]Transacciones\n[4]Salir\n");
		scanf("%d",&opc);
		switch(opc){
		case 1:
			Clientes(ruta,total);
			break;
		case 2:
			Cuentas(ruta,total);
			break;
		case 3:
			Transacciones(ruta);
		}
	}
	//fprintf(mydb,"%d",*total);
	fclose(mydb);
	return 0;
}
//Manejo de Clientes
void Clientes(char ruta[],int* t){
	int s=0;
	int id;
	char ruta_new[100];
	strcpy(ruta_new,ruta);
	strcat(ruta_new,"clientes.dat");
	char ruta_cuentas[100];
	strcpy(ruta_cuentas,ruta);
	strcat(ruta_cuentas,"cuentas.dat");

	while(s!=5){
		puts(ruta_new);
		printf("CLIENTES\n[1]Nuevo cliente\n[2]Buscar cliente\n[3]Eliminar
cliente\n[4]Imprimir clientes\n[5]Salir\n");
		scanf("%d",&s);
		switch(s){
		case 1:
			NuevoCliente(ruta_new);
			break;
		case 2:
			printf("Introduzca el ID del usuario que desea buscar: ");
			scanf("%d",&id);
			BuscarCliente(ruta_new,id,true);
			break;
		case 3:
			EliminarCliente(ruta_new,ruta_cuentas,t);
			break;
		case 4:
			ImprimirClientes(ruta_new);
			break;
```

```
			default:
				break;
			}
		}
}
void NuevoCliente(char r[]){
	FILE *cl;
	Cliente new,temp;
	int pos;
	int c=1;
	cl=fopen(r,"r+b");
	if(cl==NULL){
		fclose(cl);
		cl=fopen(r,"ab");
		Cliente cliente={NULL};
		fwrite(&cliente,sizeof(Cliente),1,cl);
		fclose(cl);
		cl=fopen(r,"r+b");
		c=1;
		}
		while(!feof(cl)){
		fread(&temp,sizeof(Cliente),1,cl);
		if(temp.id_usuario==0){
			pos=ftell(cl)-sizeof(Cliente);
			fseek(cl,pos,SEEK_SET);
			break;
		}
		c=temp.id_usuario+1;
		}
	new.id_usuario=c;
	printf("ID de usuario: %d\n",new.id_usuario);
	fflush(stdin);
	printf("Nombre: ");
	gets(new.nombre);
	printf("Apellido paterno: ");
	gets(new.apellido_p);
	printf("Apellido materno: ");
	gets(new.apellido_m);
	printf("Fecha de nacimiento ");
	new.f_nac=GetDate();
	fwrite(&new,sizeof(Cliente),1,cl);
	fclose(cl);
}
bool BuscarCliente(char r[], int usuario, bool print){
	FILE *cl;
	bool found=false;
	Cliente cliente;
	cl=fopen(r,"rb");
	if(cl==NULL){
		printf("Aún no existe un registro de clientes o se encuentra en una
ruta distinta.\n");
		fclose(cl);
		return false;
	}
	while(!feof(cl)){
```

```
			fread(&cliente,sizeof(Cliente),1,cl);
			if(cliente.id_usuario==usuario){
				found=true;
				break;
			}
		}
		if(print==false)
			return found;

		if(found==false){
			printf("Usuario no registrado.\n");
			return found;
		}
		else{
			if(print==other)
				printf("%s %s
%s",cliente.nombre,cliente.apellido_p,cliente.apellido_m);
			else{
			printf("ID de usuario: %d\n",cliente.id_usuario);
			printf("Nombre: %s\n",cliente.nombre);
			printf("Apellido paterno: %s\n",cliente.apellido_p);
			printf("Apellido materno: %s\n",cliente.apellido_m);
			printf("Fecha de nacimiento: ");
			PrintDate(cliente.f_nac);
			}
		}
		fclose(cl);
		return found;

}
bool EliminarCliente(char r[], char r_cuentas[], int* t){
		FILE *cl;
		Cliente cliente,temp;
		printf("Eliminar cliente.\n");
		int opc;
		cl=fopen(r,"rb+");
		if(cl==NULL){
			printf("No existe un registro de clientes o se encuentra en una
ruta distinta.\n");
			fclose(cl);
			return false;
		}
		printf("ID del usuario que desea eliminar: ");
		scanf("%d",&cliente.id_usuario);
		if(BuscarCliente(r,cliente.id_usuario,true)==false){
			fclose(cl);
			return false;
		}

		printf("¿Seguro que desea eliminar este cliente?\n[1]Sí\n[2]No\n");
		scanf("%d",&opc);
		if(opc==2){
			fclose(cl);
			return false;
		}
```

```
	int pos;
	while(!feof(cl)){
		fread(&temp,sizeof(Cliente),1,cl);
		if(temp.id_usuario==cliente.id_usuario){
			Cliente cliente={NULL};
			pos=ftell(cl)-sizeof(Cliente);
			fseek(cl,pos,SEEK_SET);
			fwrite(&cliente,sizeof(Cliente),1,cl);
			break;
		}
	}
	//CuentasCliente(r_cuentas, temp.id_usuario, t);

	printf("Se ha eliminado el cliente.\n\n");
	fclose(cl);
	return true;
}
bool ImprimirClientes(char r[]){
	FILE *cl;
	Cliente cl_temp;
	int c=0;
	cl=fopen(r,"rb");
	if(cl==NULL){
		printf("No existe un registro de clientes o se encuentra en una
ruta distinta.\n");
		fclose(cl);
		return false;
	}

	printf("ID usuario\tNombre\t   Apellidos\t   Fecha de nacimiento\n");
	while(!feof(cl)){
		fread(&cl_temp,sizeof(Cliente),1,cl);
		c++;
	}
	rewind(cl);
	int i;
	for(i=0;i<c-1;i++){
		fread(&cl_temp,sizeof(Cliente),1,cl);
		if(cl_temp.id_usuario!=0){
		printf("\t%d\t%s\t  %s %s\t
",cl_temp.id_usuario,cl_temp.nombre,cl_temp.apellido_p,cl_temp.apellido_m);
		PrintDate(cl_temp.f_nac);
		}
	}
	fclose(cl);
	return true;
}
//Manejo de Cuentas
void Cuentas(char ruta[], int* t){
	int s=0;
	char ruta_new[100];
	strcpy(ruta_new,ruta);
	strcat(ruta_new,"cuentas.dat");
	char ruta_clientes[100];
	strcpy(ruta_clientes,ruta);
```

```
	strcat(ruta_clientes,"clientes.dat");

	while(s!=5){
		puts(ruta_new);
		printf("CUENTAS\n[1]Nueva cuenta\n[2]Buscar cuenta\n[3]Eliminar
cuenta\n[4]Imprimir cuentas\n[5]Salir\n");
		scanf("%d",&s);
		int id;
		switch(s){
		case 1:
			NuevaCuenta(ruta_new,ruta_clientes,t);
			break;
		case 2:
			printf("Introduzca el ID de cuenta que desea buscar: ");
			scanf("%d",&id);
			BuscarCuenta(ruta_new,ruta_clientes,id,true);
			break;
		case 3:
			EliminarCuenta(ruta_new);
			break;
		case 4:
			ImprimirCuentas(ruta_new,ruta_clientes);
			break;
		default:
			break;
		}
	}
}
bool NuevaCuenta(char r[],char r_clientes[],int* t){

	FILE *cu;
	Cuenta new,temp;
	int c=1,pos;
	printf("ID de usuario: ");
	scanf("%d",&new.id_usuario);
	if(BuscarCliente(r_clientes,new.id_usuario,false)==false){
		printf("Usuario no registrado.\n");
		return false;
	}
	else{
		cu=fopen(r,"r+b");
		if(cu==NULL){
			fclose(cu);
			cu=fopen(r,"ab");
			Cuenta cuenta={NULL};
			fwrite(&cuenta,sizeof(Cuenta),1,cu);
			fclose(cu);
			cu=fopen(r,"r+b");
			c=1;
			}

			while(!feof(cu)){
			fread(&temp,sizeof(Cuenta),1,cu);
			if(temp.id_cuenta==0){
				pos=ftell(cu)-sizeof(Cuenta);
```

```
                        fseek(cu,pos,SEEK_SET);
                        break;
                }
                c=temp.id_cuenta+1;
                }

        new.id_cuenta=c;
        *t=c;
        printf("ID de cuenta: %d\n",new.id_cuenta);
        fflush(stdin);
        printf("Saldo inicial: ");
        scanf("%d",&new.saldo);
        printf("Fecha de apertura ");
        new.f_apertura=GetDate();
        fwrite(&new,sizeof(Cuenta),1,cu);
        fclose(cu);
        return true;
    }
}
bool BuscarCuenta(char r[], char r_cl[], int id, bool print){
    FILE *cu;
    bool found=false;
    Cuenta cuenta;
    cu=fopen(r,"rb+");
    if(cu==NULL){
        printf("Aún no existe un registro de cuentas o se encuentra en una
ruta distinta.\n");
        fclose(cu);
        return false;
    }
    while(!feof(cu)){
        fread(&cuenta,sizeof(Cuenta),1,cu);
        if(cuenta.id_cuenta==id){
            found=true;
            break;
        }
    }
    if(print==false)
        return found;

    if(found==false){
        printf("Cuenta no registrada.\n");
        return found;
    }
    else{
        printf("ID de cuenta: %d\n",cuenta.id_cuenta);
        printf("Nombre: ");
        BuscarCliente(r_cl,cuenta.id_usuario,other);
        printf("\nSaldo: %d\n",cuenta.saldo);
        printf("Fecha de apertura: ");
        PrintDate(cuenta.f_apertura);
    }
    fclose(cu);
    return found;
}
```

```
bool EliminarCuenta(char r[]){
	FILE *cu;
	Cuenta cuenta,temp;
	int opc;
	cu=fopen(r,"rb+");
	if(cu==NULL){
		printf("No existe un registro de cuentas o se encuentra en una ruta
distinta.\n");
		fclose(cu);
		return false;
	}
	printf("ID de la cuenta que desea eliminar: ");
	scanf("%d",&cuenta.id_cuenta);
	if(BuscarCuenta(r,r,cuenta.id_cuenta,true)==false){
		fclose(cu);
		return false;
	}

	printf("¿Seguro que desea eliminar esta cuenta?\n[1]Sí\n[2]No\n");
	scanf("%d",&opc);
	if(opc==2){
		fclose(cu);
		return false;
	}
	int pos;
	while(!feof(cu)){
		fread(&temp,sizeof(Cuenta),1,cu);
		if(temp.id_cuenta==cuenta.id_cuenta){
			Cuenta cuenta={NULL};
			pos=ftell(cu)-sizeof(Cuenta);
			fseek(cu,pos,SEEK_SET);
			fwrite(&cuenta,sizeof(Cuenta),1,cu);
			break;
		}
	}
	printf("Se ha eliminado la cuenta.\n");
	fclose(cu);
	return true;
}
bool ImprimirCuentas(char r[], char r_cl[]){
	FILE *cu;
	Cuenta cu_temp;
	int c=0;
	cu=fopen(r,"rb");
	if(cu==NULL){
		printf("No existe un registro de cuentas o se encuentra en una ruta
distinta.\n");
		fclose(cu);
		return false;
	}
	printf("ID cuenta\tID usuario\t  Nombre\t   Saldo\t  Fecha de
apertura\n");
	while(!feof(cu)){
		fread(&cu_temp,sizeof(Cuenta),1,cu);
		c++;
```

```
	}
	int i;
	rewind(cu);
	for(i=0;i<c-1;i++){
		fread(&cu_temp,sizeof(Cuenta),1,cu);
		if(cu_temp.id_cuenta!=0){
		printf("   %d\t\t   %d\t   ",cu_temp.id_cuenta,cu_temp.id_usuario);
		BuscarCliente(r_cl,cu_temp.id_usuario,other);
		printf("\t          $%d\t     \t  ",cu_temp.saldo);
		PrintDate(cu_temp.f_apertura);
		}
	}
	fclose(cu);
	return true;
}
//Manejo de Transacciones
void Transacciones(char ruta[]){
	int s=0;
	char ruta_new[100];
	strcpy(ruta_new,ruta);
	strcat(ruta_new,"transacciones.dat");
	char ruta_cuenta[100];
	strcpy(ruta_cuenta,ruta);
	strcat(ruta_cuenta,"cuentas.dat");
	while(s!=4){
		puts(ruta_new);

	printf("TRANSACCIONES\n[1]Depósito\n[2]Retiro\n[3]Transferencia\n[4]Salir
\n");
		scanf("%d",&s);
		if(s<4&&s>0){
			NewTr(ruta_new,ruta_cuenta,s);
		}
	}
}
bool NewTr(char r[], char ruta_cuenta[], int tipo){
	FILE *tr;
	Transaccion new,temp;
	int c;
	if(tipo>1)
	    printf("ID cuenta origen: ");
	else
		printf("ID cuenta destino: ");
	scanf("%d",&new.id_c_origen);
	if(BuscarCuenta(ruta_cuenta,r,new.id_c_origen,false)==false){
		printf("Cuenta no registrada.\n");
		return false;
	}
	switch(tipo){
			case 1:
				new.id_c_destino=0;
				strcpy(new.tipo,"Depósito");
				break;
			case 2:
				new.id_c_destino=0;
```

```
                        strcpy(new.tipo,"Retiro");
                        break;
                case 3:
                        printf("ID cuenta destino: ");
                        scanf("%d",&new.id_c_destino);
                        strcpy(new.tipo,"Transferencia");

        if(BuscarCuenta(ruta_cuenta,ruta_cuenta,new.id_c_destino,false)==false){
                                printf("Cuenta no registrada.\n");
                                return false;
                        }
                        break;
                }
        printf("Monto: $");
        scanf("%d",&new.monto);
        if(VerifSaldo(new,ruta_cuenta)==false&&tipo>1){
                printf("Saldo insuficiente.\n");
                return false;
        }
        tr=fopen(r,"rb");
        if(tr==NULL){
                c=1;
        }
        else{
                while(!feof(tr)){
                fread(&temp,sizeof(Transaccion),1,tr);
                c=temp.id_tr+1;
                }
        }

        fclose(tr);
        tr=fopen(r,"ab");
        new.id_tr=c;
        fflush(stdin);
        printf("ID de transacción: %d\n",new.id_tr);
        printf("Tipo de transacción: %s\n",new.tipo);
        printf("Fecha de transacción ");
        new.f_tr=GetDate();
        fwrite(&new,sizeof(Transaccion),1,tr);

        fclose(tr);
        ModifSaldo(ruta_cuenta,new,tipo);
        return true;
}
Fecha GetDate(){
        Fecha fch;
        printf("(dd/mm/aaaa):");
        scanf("%d/%d/%d",&fch.d,&fch.m,&fch.a);
        return fch;
}
void PrintDate(Fecha f){
        printf("%d/%d/%d\n",f.d,f.m,f.a);
}
bool VerifSaldo(Transaccion tr, char r[]){
        FILE *cu;
```

```
	bool found=false;
	Cuenta cuenta;
	cu=fopen(r,"rb");
	if(cu==NULL){
		printf("Aún no existe un registro de cuentas o se encuentra en una
ruta distinta.\n");
		fclose(cu);
		return false;
	}
	while(!feof(cu)){
		fread(&cuenta,sizeof(Cuenta),1,cu);
		if(cuenta.id_cuenta==tr.id_c_origen){
			found=true;
			break;
		}
	}
	fclose(cu);
	if(cuenta.saldo<tr.monto)
		return false;
	else
		return true;
}
void ModifSaldo(char r[], Transaccion tr, int tipo){
	FILE *cu;
	Cuenta cuenta,temp;
	cu=fopen(r,"r+b");
	int pos;
	cuenta.id_cuenta=tr.id_c_origen;

	while(!feof(cu)){
		fread(&temp,sizeof(Cuenta),1,cu);
		if(temp.id_cuenta==cuenta.id_cuenta){
			cuenta.f_apertura=temp.f_apertura;
			cuenta.id_usuario=temp.id_usuario;
			switch(tipo){
			case 1:
				cuenta.saldo=temp.saldo+tr.monto;
				break;
			case 2:
				cuenta.saldo=temp.saldo-tr.monto;
				break;
			case 3:
				cuenta.saldo=temp.saldo-tr.monto;
				break;
			}
			pos=ftell(cu)-sizeof(Cuenta);
			fseek(cu,pos,SEEK_SET);
			fwrite(&cuenta,sizeof(Cuenta),1,cu);
			break;
		}
	}
	if(tipo==3){
		cuenta.id_cuenta=tr.id_c_destino;
		int pos;
		while(!feof(cu)){
```

```
                        fread(&temp,sizeof(Cuenta),1,cu);
                        if(temp.id_cuenta==cuenta.id_cuenta){
                                cuenta.f_apertura=temp.f_apertura;
                                cuenta.id_usuario=temp.id_usuario;
                                cuenta.saldo=temp.saldo+tr.monto;
                                pos=ftell(cu)-sizeof(Cuenta);
                                fseek(cu,pos,SEEK_SET);
                                fwrite(&cuenta,sizeof(Cuenta),1,cu);
                                break;
                        }
        }
        }

                fclose(cu);
                printf("TRANSACCIÓN REALIZADA.\n\n");
}
```

Ejecución

Conclusiones (obligatorio):

Uno de los principales problemas al realizar este programa fue el acomodarse a manejar los punteros para editar información en los archivos, además de encontrar el modo de apertura que permitía sobreescribir sin que toda la información se borrar. Este problema se pudo solucionar, sin embargo, no hallé manera de borrar las cuentas asociadas a un cliente cuando este era eliminado.

Igualmente, tuve que realizar varias modificaciones a las funciones pues, al volver a leer las especificaciones, me encontré con detalles que había pasado por alto de primera, y con el fin de no modificar todo realicé ciertas funciones más complicadas de lo que podrían haber sido si todo se hubiera planeado bien y considerado todos los puntos desde un inicio.