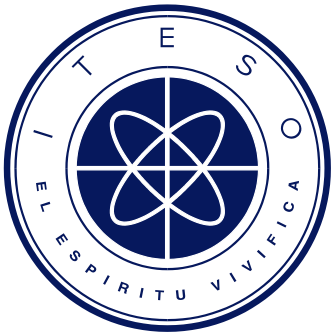
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Departamento de Electrónica, Sistemas e Informática

INGENIERÍA EN SISTEMAS COMPUTACIONALES



PROGRAMACIÓN CON MEMORIA DINÁMICA

TAREA 2. MEMORIA DINÁMICA Y ARCHIVOS

Autor: Salazar Valdovinos, Sofía

Presentación: 5 pts.
Funcionalidad: 30 pts.
Pruebas: 10 pts.

12 de junio de 2018. Tlaquepaque, Jalisco,

Ir a resultados

- Falta describir las pruebas (escenario, y resultados de la experimentación).
- Falta cubrir requerimientos funcionales que requieren del manejo de múltiples archivos..

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

```
/*
 ============================================================================
 Name        : Tarea3.c
 Author      : Momo (Sofía Michel Salazar Valdovinos)
 Version     : 1.0
 Copyright   : Your copyright notice
 Description : myDataBaseSystem
 ============================================================================
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
typedef struct{
	unsigned short dd, mm, aa;
}Fecha;
typedef struct{
	unsigned short id_usuario;
	char nombre[15], apellido_p[15], apellido_m[15];
	Fecha fecha_u;
}Usuario;
typedef struct{
	unsigned short id_cuenta, id_usuario;
	float saldo;
	Fecha fecha_c;
}Cuenta;
typedef struct{
	unsigned short id_transaccion, tipo_operacion, id_origen, id_destino;
	Fecha fecha_t;
	float monto;
}Transferencia;
void delay(int m);
int existe_id_cliente(FILE *file, char ruta[55],  unsigned short id);
int existe_id_cuenta(FILE *file, char ruta[55],  unsigned short id);
Usuario capturarUsuario();
void grabarUsuario(FILE *file, char ruta[55]);
void buscarCliente(FILE *file, char ruta[55]);
void eliminarCliente(FILE *file, char ruta[55]);
void imprimirClientes(FILE *file, char ruta[55]);
Cuenta capturarCuenta();
void grabarCuenta(FILE *file, char ruta[55]);
void buscarCuenta(FILE *file,char ruta[55], FILE *file_clientes, char
ruta_clientes[55]);
void eliminarCuenta(FILE *file, char ruta[55]);
void imprimirCuentas(FILE *file, char ruta[55]);

Transferencia deposito();
Transferencia retiro();
```

```
Transferencia transferencia();
void grabaroperacion(FILE *file, char ruta[55], char ruta_cuentas[55], int opc);
void afectarCuenta(char ruta_cuentas[55], Transferencia trans);
int main(void) {
	setvbuf(stderr, NULL, _IONBF, 0);
	setvbuf(stdout, NULL, _IONBF, 0);
	unsigned short opcion = 0, menu = 0;
	char ruta[40],ruta_clientes[55],ruta_tran[55], ruta_cuentas[55];
	FILE *f_clientes, *f_cuentas, *f_tran, *f_mydb;
	printf("<<Sistema MyDB>>\n");
	f_mydb = fopen("mydb.sys", "rb+");
	fclose(f_mydb);
	if(f_mydb == NULL) {
		printf("Bienvenido al sistema. Por favor especifica la ruta de
acceso a tus archivos.\n");
		printf("Ejemplo: c:\\\\carpeta\\\\:\t");
		scanf("%s",ruta);
		f_mydb = fopen("mydb.sys", "wb");
		fwrite(ruta,sizeof(ruta),1,f_mydb);
		fclose(f_mydb);
	}
	else{
		printf("Hola de nuevo!\n");
		f_mydb = fopen("mydb.sys", "rb+");
		fread(&ruta,sizeof(ruta),1,f_mydb);
		fclose(f_mydb);
	}

	printf("Ruta: %s\n", ruta);
	strcpy(ruta_clientes, ruta);
	strcpy(ruta_tran, ruta);
	strcpy(ruta_cuentas, ruta);
	strcat(ruta_clientes,"clientes.dat");
	strcat(ruta_tran,"transacciones.dat");
	strcat(ruta_cuentas,"cuentas.dat");
	f_clientes = fopen(ruta_clientes, "wb");
	f_cuentas = fopen(ruta_cuentas, "wb");
	f_tran = fopen(ruta_tran, "wb");
	fclose(f_clientes);
	fclose(f_cuentas);
	fclose(f_tran);


	do{
		//delay(1);
		system("cls");
		printf("<<Sistema MyDB>>\n");
		printf("1.Clientes\n2.Cuentas\n3.Transacciones\n4.Salir\n");
		scanf("%hu",&opcion);
		system("cls");
		//delay(1);
		switch(opcion){
		case 1:
			do{
				printf("-->Clientes<--\n");
```

```
                        printf("[1]Nuevo Cliente\n[2]Buscar
Cliente\n[3]Eliminar Cliente\n[4]Imprimir Clientes\n");
                        scanf("%hu",&menu);
                        system("cls");
                        //delay(1);
                        switch(menu){
                        case 1:
                              printf("=REGISTRO DE USUARIOS=\n");
                              grabarUsuario(f_clientes, ruta_clientes);
                              break;
                        case 2:
                              printf("=BÚSQUEDA DE CLIENTE=\n");
                              buscarCliente(f_clientes,ruta_clientes);
                              break;
                        case 3:
                              printf("=ELIMINAR CLIENTE=\n");
                              eliminarCliente(f_clientes, ruta_clientes);
                              break;
                        case 4:
                              printf("=IMPRIMIR CLIENTES=\n");
                              imprimirClientes(f_clientes, ruta_clientes);
                              break;
                        }

                  } while (menu>0 && menu <5);
                  break;
            case 2:
                  do{
                        printf("-->Cuentas<--\n");
                        printf("[1]Nueva Cuenta\n[2]Buscar Cuenta\n[3]Eliminar
Cuenta\n[4]Imprimir Cuentas\n");
                        scanf("%hu",&menu);
                        system("cls");
                        switch(menu){
                        case 1:
                              printf("=CREAR CUENTA=\n");
                              grabarCuenta(f_cuentas, ruta_cuentas);
                              break;
                        case 2:
                              printf("=BUSCAR CUENTA=\n");
                              buscarCuenta(f_cuentas, ruta_cuentas,
f_clientes, ruta_clientes);
                              break;
                        case 3:
                              printf("=ELIMINAR CUENTA=\n");
                              eliminarCuenta(f_cuentas, ruta_cuentas);
                              break;
                        case 4:
                              printf("=IMPRIMIR CUENTAS=\n");
                              imprimirCuentas(f_cuentas, ruta_cuentas);
                              break;
                        }
                  }while(menu>0 && menu <5);

                  break;
```

```
		case 3:
			do{
				printf("-->Transacciones<--\n");
				printf("[1]Depositar\n[2]Retirar\n[3]Transferir\n");
				scanf("%hu",&menu);
				system("cls");
				switch(menu){
				case 1:
					printf("=DEPOSITAR=\n");
					grabaroperacion(f_tran, ruta_tran,
ruta_cuentas,1);
					break;
				case 2:
					printf("=RETIRAR=\n");
					grabaroperacion(f_tran, ruta_tran,
ruta_cuentas,2);
					break;
				case 3:
					printf("=TRANSFERIR=\n");
					grabaroperacion(f_tran, ruta_tran,
ruta_cuentas,3);
					break;
				}
			}while(menu > 0 && menu <4);

			break;
		case 4:
			printf("Saliendo del sistema...\n");
			return -1;
		}
		system("cls");

	}while(opcion >= 1 && opcion <=4);
	printf("Has salido del sistema con éxito.");
	return EXIT_SUCCESS;
}
Usuario capturarUsuario(){
	Usuario nuevo;
	fflush(stdin);
	printf("Nombre:\t");
	scanf("%s",nuevo.nombre);
	printf("Apellido Paterno:");
	scanf("%s",nuevo.apellido_p);
	fflush(stdin);
	printf("Apellido Materno:");
	scanf("%s",nuevo.apellido_m);
	fflush(stdin);
	printf("Fecha de Nacimiento: dd/mm/aaaa:");
	fflush(stdin);
	scanf("%hu/%hu/%hu",&nuevo.fecha_u.dd, &nuevo.fecha_u.mm,
&nuevo.fecha_u.aa);
	return nuevo;
}
Cuenta capturarCuenta(){
	Cuenta nueva;
```

```
	fflush(stdin);
	printf("ID usuario: ");
	scanf("%hu",&nueva.id_usuario);
	printf("Saldo: ");
	scanf("%f",&nueva.saldo);
	printf("Fecha de aperutra: dd/mm/aaaa:");
	fflush(stdin);
	scanf("%hu/%hu/%hu",&nueva.fecha_c.dd, &nueva.fecha_c.mm,
&nueva.fecha_c.aa);
	return nueva;
}
void grabarUsuario(FILE *file, char ruta[55]){
	Usuario nuevo, aux;
	nuevo = capturarUsuario();
	unsigned short len;
	file = fopen(ruta,"rb");
	fseek(file, 0, SEEK_END);
	len = ftell(file) / sizeof(Usuario);
	if (len > 0){
		fseek(file, ((len-1)*sizeof(Usuario)),SEEK_SET);
		fread(&aux,sizeof(Usuario),1,file);
		nuevo.id_usuario = aux.id_usuario +1;
	} else nuevo.id_usuario = 1;
	fclose(file);

	file = fopen(ruta,"ab");
	if(file != NULL) {
		fwrite(&nuevo,sizeof(Usuario),1,file);
		printf("Grabado éxitosamente.\n");
	}
	fclose(file);
}
void grabaroperacion(FILE *file, char ruta[55], char ruta_cuentas[55], int opc){
	Transferencia nueva, aux ;
	switch (opc){
	case 1:
		nueva = deposito();
		break;
	case 2:
		nueva = retiro();
		break;
	case 3:
		nueva = transferencia();
		break;
	}
	unsigned short len;
	file = fopen(ruta,"rb");
	fseek(file, 0, SEEK_END);
	len = ftell(file) / sizeof(Transferencia);
	if (len > 0){
		fseek(file, ((len-1)*sizeof(Transferencia)),SEEK_SET);
		fread(&aux,sizeof(Transferencia),1,file);
		nueva.id_transaccion = aux.id_transaccion +1;
	} else nueva.id_transaccion = 1;
	fclose(file);
```

```
	file = fopen(ruta,"ab");
	if(file != NULL) {
		fwrite(&nueva,sizeof(Transferencia),1,file);
		printf("Grabado éxitosamente.\n");
	}
	afectarCuenta(ruta_cuentas, nueva);
}
void afectarCuenta(char ruta_cuentas[55], Transferencia trans){
	unsigned short len, i_origen, i_destino, i ;
	Cuenta *p_array ;
	FILE *file = fopen (ruta_cuentas,"ab");
	file = fopen(ruta_cuentas,"rb");
	fseek(file, 0, SEEK_END);
	len = ftell(file) / sizeof(Cuenta);
	fclose(file);
	p_array = (Cuenta * ) malloc(sizeof(Cuenta)* len);
	file = fopen(ruta_cuentas,"rb");
	for(i = 0; i < len; i++){
		fread((p_array+i),sizeof(Cuenta),1,file);
		if((p_array+i)->id_usuario == trans.id_origen) {
			i_origen = i;
		}
		if((p_array+i)->id_usuario == trans.id_destino) {
			i_destino = i;
		}
	}
	fclose(file);
	file = fopen(ruta_cuentas,"wb");
		for(i = 0; i < len ; i++){
			if(i == i_origen){
				(p_array+i)->saldo -= trans.monto;
			}
			if(i == i_destino){
				(p_array+i)->saldo += trans.monto;
			}
			fwrite((p_array+i),sizeof(Cuenta),1,file);
		}
	fclose(file);

}
void grabarCuenta(FILE *file, char ruta[55]){
	Cuenta nueva, aux;
	nueva = capturarCuenta();
	unsigned short len;
	file = fopen(ruta,"rb");
	fseek(file, 0, SEEK_END);
	len = ftell(file) / sizeof(Cuenta);
	if(len > 0){
		fseek(file, ((len-1)*sizeof(Usuario)),SEEK_SET);
		fread(&aux,sizeof(Usuario),1,file);
		nueva.id_cuenta = aux.id_cuenta + 1;
	} else nueva.id_cuenta = 1;
	fclose(file);
	file = fopen(ruta,"ab");
	if(file != NULL) {
```

```
			fwrite(&nueva,sizeof(Cuenta),1,file);
			printf("Cuenta grabada éxitosamente.\n");
		}
		fclose(file);
}


void buscarCliente(FILE *file, char ruta[55]){
		Usuario aux;
		unsigned short len;
		unsigned short id;
		file = fopen(ruta,"rb");
		fseek(file, 0, SEEK_END);
		len = ftell(file) / sizeof(Usuario);
		if(len > 0){
			printf("ID:\t");
			scanf("%hu",&id);
			if(existe_id_cliente(file,ruta,id) != 0)printf("No se encontró el
ID. \n");
			else {
				for(int i = 0; i < len; i++){
					fseek(file, (i*sizeof(Usuario)),SEEK_SET);
					fread(&aux,sizeof(Usuario),1,file);
					if(aux.id_usuario == id) {
						printf("%d\t%s\t%s\t%s\t%02d/%02d/%04d\n
",aux.id_usuario, aux.nombre, aux.apellido_p, aux.apellido_m, aux.fecha_u.dd,
aux.fecha_u.mm, aux.fecha_u.aa);
					}
				}
			}

		} else printf("No se encontraron registros. \n");
		fclose(file);
}
void buscarCuenta(FILE *file,char ruta[55], FILE *file_clientes, char
ruta_clientes[55]){
		Cuenta cuenta;
		unsigned short len, id_cuenta;
		file = fopen(ruta,"rb");
		fseek(file, 0, SEEK_END);
		len = ftell(file) / sizeof(Cuenta);
		if(len > 0){
			printf("ID:\t");
			scanf("%hu",&id_cuenta);
			if(existe_id_cuenta(file,ruta,id_cuenta) != 0) printf("No se
encontró el ID cuenta.\n");
			else{
				for(int i = 0; i < len; i++){
					fseek(file, (i*sizeof(Cuenta)),SEEK_SET);
					fread(&cuenta,sizeof(Cuenta),1,file);
					if(cuenta.id_cuenta == id_cuenta) {
						unsigned short len_c;
						Usuario aux;
						file_clientes = fopen(ruta_clientes,"rb");
						fseek(file, 0, SEEK_END);
```

```
                                    len_c = ftell(file) / sizeof(Usuario);
                                    for(int i = 0; i < len_c; i++){
                                          fseek(file_clientes,
(i*sizeof(Usuario)),SEEK_SET);
                                          fread(&aux,sizeof(Usuario),1,file);
                                          if(aux.id_usuario == cuenta.id_usuario) {
                                                printf("%d \t %s \t %.4f",
cuenta.id_cuenta, aux.nombre, cuenta.saldo);
                                          }
                                    }
                              }
                        }
                  }
      } else printf("No se encontraron registrosn\n");
}
void eliminarCliente(FILE *file, char ruta[55]){
      unsigned short len, index;
      file = fopen(ruta,"rb");
      fseek(file, 0, SEEK_END);
      len = ftell(file) / sizeof(Usuario);
      fclose(file);
      unsigned short id, i;
      if(len > 0){
            printf("ID:\t");
            scanf("%hu",&id);
            if(existe_id_cliente(file,ruta,id) != 0) printf("No se encontró el
ID. \n");
            else{
                  Usuario *p_array = (Usuario*)malloc(sizeof(Usuario)*len);
                  file = fopen(ruta,"rb");
                  for(i = 0; i < len; i++){
                        fread((p_array+i),sizeof(Usuario),1,file);
                        if((p_array+i)->id_usuario == id) {
                              index = i;
                        }
                  }
                  fclose(file);
                  file = fopen(ruta,"wb");
                  if(index == len-1 ){
                        for(i = 0; i < len ; i++){
                              if(i!= index)
fwrite((p_array+i),sizeof(Usuario),1,file);
                        }
                  }
                  else if(index == 0){
                        for(i = 0; i < len ; i++){

      if(i!=0)fwrite((p_array+i),sizeof(Usuario),1,file);
                        }
                  }
                  else{
                        for(i = 0; i < len ; i++){
                              if(i != index)
fwrite((p_array+i),sizeof(Usuario),1,file);
                        }
```

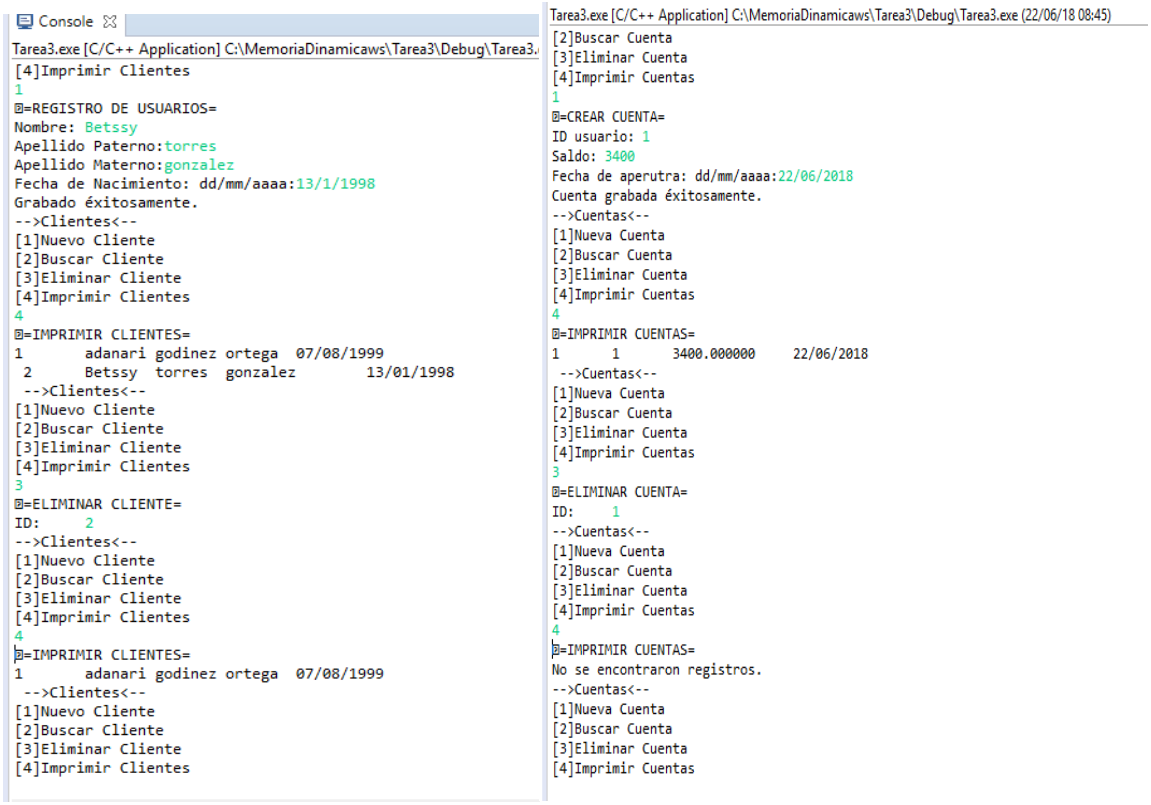
```
                        }
                        fclose(file);
                }
        }else printf("No se encontraron registros. \n");
}
void eliminarCuenta(FILE *file, char ruta[55]){
        unsigned short len, index;
        file = fopen(ruta,"rb");
        fseek(file, 0, SEEK_END);
        len = ftell(file) / sizeof(Cuenta);
        fclose(file);
        unsigned short id, i;
        if(len > 0){
                printf("ID:\t");
                scanf("%hu",&id);
                if(existe_id_cuenta(file,ruta,id) != 0) printf("No se encontró el
ID. \n");
                else{
                        Cuenta *p_array = (Cuenta*)malloc(sizeof(Cuenta)*len);
                        file = fopen(ruta,"rb");
                        for(i = 0; i < len; i++){
                                fread((p_array+i),sizeof(Cuenta),1,file);
                                if((p_array+i)->id_cuenta == id) {
                                        index = i;
                                }
                        }
                        fclose(file);
                        file = fopen(ruta,"wb");
                        if(index == len-1 ){
                                for(i = 0; i < len ; i++){
                                        if(i!= index)
fwrite((p_array+i),sizeof(Cuenta),1,file);
                                }
                        }
                        else if(index == 0){
                                for(i = 0; i < len ; i++){

        if(i!=0)fwrite((p_array+i),sizeof(Cuenta),1,file);
                                }
                        }
                        else{
                                for(i = 0; i < len ; i++){
                                        if(i != index)
fwrite((p_array+i),sizeof(Cuenta),1,file);
                                }
                        }
                        fclose(file);
                }
        }else printf("No se encontraron registros. \n");
}
void imprimirClientes(FILE *file, char ruta[55]){
        int len;
        Usuario aux;
        file = fopen(ruta,"rb");
        fseek(file, 0, SEEK_END);
```

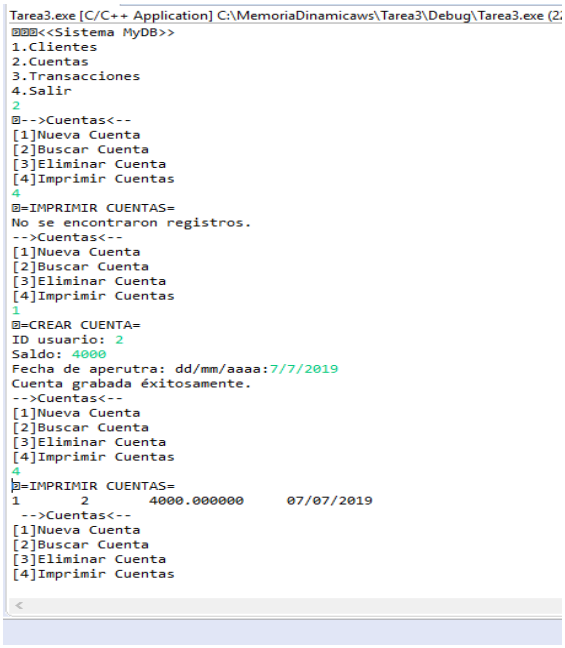
```
	len = ftell(file) / sizeof(Usuario);
	//printf("Usuarios grabados: %d\n", len);
	if(len>0){
		for(int i = 0; i < len; i++){
			fseek(file, (i*sizeof(Usuario)),SEEK_SET);
			fread(&aux,sizeof(Usuario),1,file);
			printf("%d\t%s\t%s\t%s\t%02d/%02d/%04d\n ",aux.id_usuario,
aux.nombre, aux.apellido_p, aux.apellido_m, aux.fecha_u.dd, aux.fecha_u.mm,
aux.fecha_u.aa);
		}
	}else  printf("No se encontraron registros. \n");
}
void imprimirCuentas(FILE *file, char ruta[55]){
	int len;
	Cuenta aux;
	file = fopen(ruta,"rb");
	fseek(file, 0, SEEK_END);
	len = ftell(file) / sizeof(Cuenta);
	if(len>0){
		for(int i = 0; i < len; i++){
			fseek(file, (i*sizeof(Cuenta)),SEEK_SET);
			fread(&aux,sizeof(Cuenta),1,file);
			printf("%d\t%d\t%f\t%02d/%02d/%04d\n ",aux.id_cuenta,
aux.id_usuario, aux.saldo,aux.fecha_c.dd, aux.fecha_c.mm, aux.fecha_c.aa);
		}
	}else  printf("No se encontraron registros. \n");
	fclose(file);
}


void delay(int m)
{
	long pause;
	clock_t now,then;
	pause = m*(CLOCKS_PER_SEC);
	now = then = clock();
	while( (now-then) < pause )
		now = clock();
}
int existe_id_cliente(FILE *file, char ruta[55], unsigned short id){
	Usuario aux;
	unsigned short len, l = -1;
	file = fopen(ruta,"rb");
	fseek(file, 0, SEEK_END);
	len = ftell(file) / sizeof(Usuario);

	for(int i = 0; i < len; i++){
		fseek(file, (i*sizeof(Usuario)),SEEK_SET);
		fread(&aux,sizeof(Usuario),1,file);
		if(aux.id_usuario == id) {
			l += 1;
		}
		if(l == -1) return -1;
```

```
	}
	return 0;
}
int existe_id_cuenta(FILE *file, char ruta[55],  unsigned short id){
	Cuenta aux;
	unsigned short len, l = -1;
	file = fopen(ruta,"rb");
	fseek(file, 0, SEEK_END);
	len = ftell(file) / sizeof(Cuenta);

	for(int i = 0; i < len; i++){
		fseek(file, (i*sizeof(Cuenta)),SEEK_SET);
		fread(&aux,sizeof(Cuenta),1,file);
		if(aux.id_cuenta == id) {
			l += 1;
		}
		if(l == -1) return -1;


	}
	return 0;
}
Transferencia deposito(){
	Transferencia nueva;
	nueva.tipo_operacion = 1;
	nueva.id_destino = 0;
	printf("ID cuenta a depositar: ");
	scanf("%hu", &nueva.id_origen);
	printf("Monto: ");
	scanf("%f", &nueva.monto);
	printf("Fecha de hoy: dd/mm/aaaa");
	scanf("%hu/%hu/%hu",
&nueva.fecha_t.dd,&nueva.fecha_t.mm,&nueva.fecha_t.aa);
	return nueva;
}
Transferencia retiro(){
	Transferencia nueva;
	nueva.tipo_operacion = 2;
	printf("ID cuenta:");
	scanf("%hu", &nueva.id_origen);
	nueva.id_destino = 0;
	printf("Monto: ");
	scanf("%f", &nueva.monto);
	printf("Fecha de hoy: dd/mm/aaaa");
	scanf("%hu/%hu/%hu",
&nueva.fecha_t.dd,&nueva.fecha_t.mm,&nueva.fecha_t.aa);
	return nueva;
}
Transferencia transferencia(){
	Transferencia nueva;
	nueva.tipo_operacion = 3;
	printf("ID cuenta origen:");
	scanf("%hu", &nueva.id_origen);
	printf("ID cuenta destino:");
	scanf("%hu", &nueva.id_destino);
```

```
        printf("Monto: ");
        scanf("%f", &nueva.monto);
        printf("Fecha de hoy: dd/mm/aaaa");
        scanf("%hu/%hu/%hu",
&nueva.fecha_t.dd,&nueva.fecha_t.mm,&nueva.fecha_t.aa);
        return nueva;
}
```

Ejecución





Conclusiones (obligatorio):

Para esa practica hice uso de conocimientos previos y los reforcé. Considero que lo más difícil de la práctica es plantear el algoritmo de la forma más eficiente, de manera que las funciones creadas te sirvan para ser utilizadas en más de uno o dos segmentos de código. No pude solucionar algunos detalles de enlace en donde se ve reflejado el afectar a más de un archivo al realizar una operación.