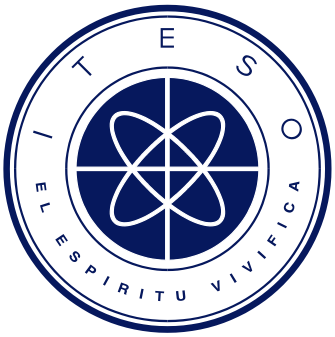
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Departamento de Electrónica, Sistemas e Informática

INGENIERÍA EN SISTEMAS COMPUTACIONALES



PROGRAMACIÓN CON MEMORIA DINÁMICA

TAREA 2. MEMORIA DINÁMICA Y ARCHIVOS

Autor: Vazquez Martinez Ignacio Itamar

Presentación: 10 pts.
Funcionalidad: 20 pts.
Pruebas: 10 pts.

12 de junio de 2018. Tlaquepaque, Jalisco,

- Falta describir las pruebas (escenario, y resultados de la experimentación).
- Falto cubrir requerimientos funcionales sobre manejo de cuentas y transacciones.

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

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

```
//  main.c
//  Tarea3_Ignacio_Vazquez_final
//
//  Created by Itamar Vazquez on 21.06.18.
//  Copyright © 2018 Ignacio Itamar Vazquez Martinez. All rights reserved.
//

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define enter printf("\n");
int a =2;
typedef struct Struct_Files{
    FILE *clientes;
    FILE *cuentas;
    FILE *transaciones;
    FILE *text_file;
}struct_files;

typedef struct Users{
    int id;
    char name[50];
    char nachname[50];
    char nachname2[50];
```

```
    char birthdate[10];
}users;

typedef struct Transactions{
    int id_cuenta;
    int monto;
}transactions;


void doBinariesExist(struct_files *Dok, char location[50]);
int alreadyInstalled(struct_files *Dok);
users *addUser(users *Benutzer);
void upload(users *Benutzer);
void download(users *Benutzer);
void printCurrentUsers(users *Benutzer);
void eraseLocalUsers(users * Benutzer);
void deleteUser(users * Benutzer, int number);
int lookUp(users *Benutzer, int number);
void printSingleUser(users *Benutzer, int number);

struct_files Dok;
users Benutzer[500];
transactions[500];
int main(int argc, const char * argv[]) {

//    Variables
    int choice;
    int choice_user;
    char location[50];
```

```
    char location2[50];
    int number;









//    This only gets executed if the program was "installed for the first time"
if(alreadyInstalled(&Dok)==1){
    printf("Please enter the location where you want to save your files");enter
    scanf("%s",location);
    strcpy(location, "/Users/Itamarvazquez/Documents/Dinamica");
    strcpy(location2, "/Users/Itamarvazquez/Documents/Dinamica");
    doBinariesExist(&Dok, location2);
}



   // User Interface
GOTO:
GOTO1:
GOTO2:
GOTO3:
    printf("Main Menu");enter
    printf("(1) - Customers");enter
    printf("(2) - Accounts");enter
    printf("(3) - Transactions");enter
    printf("(4) - Exit");enter
```

```
scanf("%d", &choice);
switch (choice) {
    case 1://Customers
        printf("USERS");enter
        printf("(1) - Add New User");enter
        printf("(2) - Look up User");enter
        printf("(3) - Delete User");enter
        printf("(4) - Print All Users");enter
        scanf("%d", &choice_user);
        switch (choice_user) {
            case 1:
                download(Benutzer);
                addUser(Benutzer);

                printCurrentUsers(Benutzer);
                goto GOTO;
            case 2:
                printf("What is the Users's ID?");enter
                scanf("%d", &number);
                printSingleUser(Benutzer, number);
                goto GOTO1;
            case 3:printf("What is the Users's ID you want to delete?");enter
                scanf("%d", &number);
                deleteUser(Benutzer, number);
                printCurrentUsers(Benutzer);
                printCurrentUsers(Benutzer);
                goto GOTO2;
            case 4:
```

```
                download(Benutzer);
                printCurrentUsers(Benutzer);
                goto GOTO3;
                break;
            default:
                break;
        }
        break;
    case 2://Accounts
        break;
    case 3://Transactions
        break;
    case 4://Exit
        break;
    default:
        break;
  }
  return 0;


//    printf("Here is the info that array contains at first");enter
//
//    printCurrentUsers(Benutzer);
//
//    printf("This is the array with the contents downloaded");enter
//    download(Benutzer);
//    printCurrentUsers(Benutzer);
//    printf("This is the array with the contents added");enter
```

```
//    addUser(Benutzer);
//    printCurrentUsers(Benutzer);
//    printf("This is the array with the contents uploaded and reloaded back
on");enter
//    upload(Benutzer);
//    download(Benutzer);
//    printCurrentUsers(Benutzer);

}



//Functions

void doBinariesExist(struct_files *Dok, char location[50]){
    //    printf("This is a test");

    char temp_location[100];
    char temp_location2[100];
    char temp_location3[100];
    //    char temp[100];
    strcpy(temp_location, location);
    strcpy(temp_location2, location);
    strcpy(temp_location3, location);
    enter
    //Clientes
    strcat(temp_location, "/clientes.dat");
    (*(Dok)).clientes = NULL;
```

```
(*(Dok)).clientes = fopen(temp_location, "rb");enter
if((*(Dok)).clientes == NULL){
    (*(Dok)).clientes = fopen(temp_location, "w+b");
    printf("This is a test");
}
printf("Location: %s", temp_location);enter

//Cuentas
strcat(temp_location2, "/cuentas.dat");
(*(Dok)).clientes = NULL;
(*(Dok)).cuentas = fopen(temp_location2, "rb");
if((*(Dok)).cuentas == NULL){
    (*(Dok)).cuentas = fopen(temp_location2, "w+b");
    printf("This is a test");
}
//    printf("Cuentas: %s", temp_location2);enter
printf("Location: %s", temp_location2);enter
//Transacciones
strcat(temp_location3, "/transacciones.dat");
(*(Dok)).clientes = NULL;
(*(Dok)).transaciones = fopen(temp_location3, "rb");
if((*(Dok)).transaciones == NULL){
    (*(Dok)).transaciones = fopen(temp_location3, "w+b");
    printf("This is a test");
}
//    printf("Transacciones: %s", temp_location3) ;enter
//    printf("This is another test");
```

```
    printf("Location: %s", temp_location3);enter
    fclose((*(Dok)).clientes);
    fclose((*(Dok)).cuentas);
    fclose((*(Dok)).transaciones);
}


int alreadyInstalled(struct_files *Dok){
    // If the document was not previously created it will return 1, if it was already
there then it will return 0;
    int x =0;
    (*Dok).text_file = NULL;
    (*Dok).text_file = fopen("mydb.sys", "r");
    if((*Dok).text_file==NULL){
        (*Dok).text_file = fopen("mydb.sys", "w+");
        x++;
    }
    fclose((*Dok).text_file);
    return x;

}
users *addUser(users *Benutzer){
    int x=0;
    enter
    printf("Please add the new customer's info below:");enter
    for(int i=0; i<500; i++){
        if((*(Benutzer+i)).id == 0){
            (*(Benutzer+i)).id = a;
```

```
            printf("Enter Name:");enter
            scanf("%s", (*(Benutzer+i)).name);
            printf("Enter Last Name:");enter
            scanf("%s", (*(Benutzer+i)).nachname);
            printf("Enter Other Last Name:");enter
            scanf("%s", (*(Benutzer+i)).nachname2);
            a++;
            x++;
            break;
        }
    }
    return Benutzer;
}

void upload(users *Benutzer){
    int i=0;
    FILE *fp;

    fp=fopen("users.dat", "w");

    for(i=0; i<500; i++){
        fwrite(Benutzer, sizeof(Benutzer), 1, fp);
    }
    fclose(fp);
}

void download(users *Benutzer){
    FILE *fp;
```

```
    fp=fopen("users.dat", "r");


    fread(Benutzer, sizeof(Benutzer), 1, fp);
    fclose(fp);
}

void printCurrentUsers(users *Benutzer){
    for(int i=0; i<500; i++){
        if((*(Benutzer+i)).id != 0){
            printf("%d  |   ", (*(Benutzer+i)).id);
            printf("%s ", (*(Benutzer+i)).name);
            printf("%s ", (*(Benutzer+i)).nachname);
            printf("%s ", (*(Benutzer+i)).nachname2);
            printf("%s", (*(Benutzer+i)).birthdate);enter
        }
    }
}
void eraseLocalUsers(users * Benutzer){
    for(int i=0; i<500; i++){
        if((*(Benutzer+i)).id != 0){
            (*(Benutzer+i)).id = 0;
            strcpy((*(Benutzer+i)).name,"");
            strcpy((*(Benutzer+i)).nachname, "");
            strcpy((*(Benutzer+i)).nachname2, "");
        }
    }
}
```

```
void deleteUser(users * Benutzerl, int number){
    for(int i=0; i<500; i++){
        if((*(Benutzer+i)).id == number){
            (*(Benutzer+i)).id = 0;
            strcpy((*(Benutzer+i)).name,"");
            strcpy((*(Benutzer+i)).nachname, "");
            strcpy((*(Benutzer+i)).nachname2, "");
        }
    }
}
int lookUp(users *Benutzer, int number){
    for(int i=0; i<500; i++){
        if((*(Benutzer+i)).id == number){
            return (*(Benutzer+i)).id;
        }
    }
    return 0;
}
void printSingleUser(users *Benutzer, int number){
    for(int i=0; i<500; i++){
        if((*(Benutzer+i)).id == number){
            printf("%d  |   ", (*(Benutzer+i)).id);
            printf("%s ", (*(Benutzer+i)).name);
            printf("%s ", (*(Benutzer+i)).nachname);
            printf("%s ", (*(Benutzer+i)).nachname2);
            printf("%s", (*(Benutzer+i)).birthdate);enter
        }
```

```
  }
}
```

Ejecución

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

```
Please add the new customer's info below:
Enter Name:
abril
Enter Last Name:
vz
Enter Other Last Name:
mt
1  |   Itam
2  |   abril vz mt
Main Menu
(1) - Customers
(2) - Accounts
(3) - Transactions
(4) - Exit
1
USERS
(1) - Add New User
(2) - Look up User
(3) - Delete User
(4) - Print All Users
2
What is the Users's ID?
2
2  |   abril vz mt
Main Menu
(1) - Customers
(2) - Accounts
(3) - Transactions
(4) - Exit
1
USERS
(1) - Add New User
(2) - Look up User
(3) - Delete User
(4) - Print All Users
4
1  |   Itam
2  |   abril vz mt
Main Menu
(1) - Customers
(2) - Accounts
(3) - Transactions
(4) - Exit
```

Conclusiones (obligatorio):

- ✓ Lo que aprendí con esta práctica. Lo que ya sabía.
- ✓ Lo que me costó trabajo y cómo lo solucioné.
- ✓ Lo que no pude solucionar.

Aprendi que los archivos binarios son mucho mas dificiles de utilizar que los de texto. Tambien aprendi que es mejor usar un tda porque hace que tu codigo sea mas facil de comprender. No pude completar la parte de los clientes, me falto tiempo. Los usuarios me funcionarion hasta el final y no alcance a terminar las otras partes.