

Laboratorio 2

Estructuras y Archivos

Objetivos

- Definir el concepto de estructura.
- Mostrar las diferentes formas de definir y declarar una estructura.
- Abrir, leer, escribir y cerrar ficheros.

1. Estructuras

Una estructura en C es una **colección de una o más variables**, estas variables pueden ser de tipos diferentes, agrupadas bajo un mismo nombre para facilitar su manejo. A esta colección de variables se les conoce como **miembros** de la estructura.

1.1. Definición, declaración e inicialización de estructuras

Para definir una estructura se hace uso de la palabra reservada `struct`, como se muestra a continuación:

```
struct structure_name {  
    // miembros de la nueva estructura  
};
```

Para utilizar esta estructura es necesario declararla siempre utilizando la palabra `struct`:

```
struct structure_name myFirstStructure;
```

Una segunda forma de definir estructuras es mediante el uso de `typedef`, palabra reservada que permite asignarle un alias a tipos de datos existentes:

```
typedef struct {  
    // miembros de la nueva estructura  
} structure_name;
```

Esta forma evita el uso de la palabra reservada `struct` en la declaraciones de estructuras:

```
structure_name myFirstStruct;
```

La definición de estructuras se realiza generalmente antes de declarar las funciones o prototipos y después de declarar las constantes y macros:

```
/* Definition of macros and constants. */  
#define MI_CONTANTE 10
```

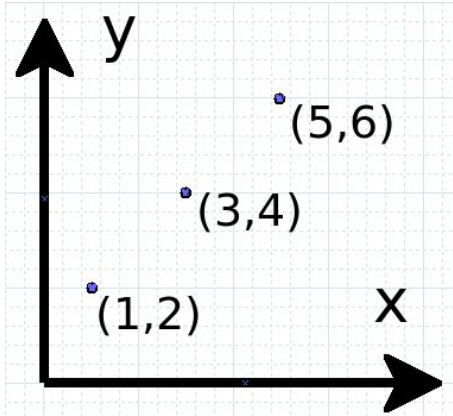
```
/* Definition of structures. */  
typedef struct {  
    // miembros de la nueva estructura  
} structure_name;
```

```
/* Declaring functions. */  
void miFuncionX();
```

Ya que estas nuevas estructuras son esencialmente nuevos tipos de datos, puede declararse objetos de esta estructura fuera del `main` (como una variable global), dentro del `main` o dentro de una función. Se recomienda realizar estas declaraciones en el `main`:

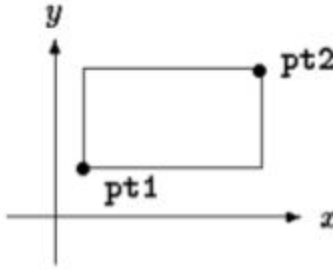
```
/* Main function. */  
int main(){  
    /* Local variables. */  
    structure_name miEstructura;  
    // ...  
    // ...  
    return 0;  
}
```

A continuación se muestran algunos ejemplos de estructuras:

Estructura	point ¹
Función / Utilidad	Almacenar la posición o coordenadas de un punto en el espacio
<pre>struct point { int x; int y; };</pre>	
Inicialización	<pre>struct point myPoint1 = {1, 2}; struct point myPoint2, myPoint3; myPoint2 = {3, 4}; //Inicializando cada miembro por separado myPoint3.x = 5; myPoint3.y = 6;</pre>
	

¹ Este ejemplo ha sido tomado del libro de Kernighan y Ritchie "The Programming Language C".

Estructura	cdMusic
Función / Utilidad	Almacenar la información de un CD de música.
<pre>struct cdMusic { char title[40]; char artist[40]; char genre[15]; int numSongs; int releaseYear; float price; } cd1, cd2; // Se declaran 2 variables después de // definir la estructura.</pre>	
Inicialización	<pre>cd1 = {"Brindo con el alma", "Diomedes Diaz", "Vallenato", 11, 1986, 19900.0}; //Iniciando cada miembro por separado strcpy(cd2.title, "Incarnate"); strcpy(cd2.artist, "Killswitch Engage"); strcpy(cd2.genre, "Metalcore"); cd2.numSongs = 12; cd2.releaseYear = 2016; cd2.price = 200.0</pre>

Estructura	rect
Función / Utilidad	Define un rectángulo utilizando dos puntos.
<pre>typedef struct { struct point pt1; // Es posible anidar estructuras struct point pt2; } rect;</pre>	
Inicialización	rect r = {{1, 1}, {5, 4}};
	

Note en los ejemplos que para acceder a los miembros de una estructura se utiliza el operador punto:

```
cd2.numSongs = 12;
```

1.2. Apuntadores a estructuras

Un apuntador a una estructura no es más que una variable que contiene la dirección en memoria de la estructura. Con esta definición se observa que un apuntador a una estructura no es muy diferente de un apuntador a un objeto de un tipo existente como se ha visto en anteriores laboratorios:

Si tenemos:	Un apuntador sería:
<pre>// Definición de la // estructura struct student { char name[40]; int numStudent; float grade; }; // Inicialización de una // nueva estructura struct student student1 = {"Fulano", "5", "2.1"};</pre>	<pre>// Apuntador struct student *ptrStudent; ptrStudent = &student1;</pre>

Para acceder a los miembros de una estructura mediante un apuntador, es necesario hacer uso del operador arrow (flecha):

```
pointer_to_struct->member;
```

Para la estructura 'student1' utilizando 'ptrStudent' sería:

```
ptrStudent->numStudent = 5;
printf("%d", ptrStudent->numStudent);
```

Ejercicio:

Escriba las líneas de código necesarias para imprimir los valores de los miembros de 'student1' utilizando el apuntador 'ptrStudent'.

Ejemplos:

1. Realice un programa que reciba la información necesaria para abrir una cuenta de ahorros en un banco. El programa debe solicitar el nombre del nuevo ahorrador, el número de la cuenta y el monto de apertura. Al finalizar, el programa debe mostrar la información del nuevo usuario en un formato tabulado. Además, debe determinar y mostrar el tipo de ahorrador según el monto ingresado como se muestra a continuación:

Monto de apertura

Menor a 100000	Mayor a 200000	Entre 100 y 200 mil
MiniAhorrador	Ahorregular	PreferencialProX

Solución:

Con la información dada en el enunciado es fácil definir una estructura:

```
typedef struct {
    char *nombre;
    int numeroCuenta;
    float montoApertura;
} cuentaAhorro;
```

Para obtener la información del nuevo cliente desde el main (y suponiendo que declaramos un objeto de esta estructura también en el main), utilizamos la función 'scanf' junto con el operador 'punto':

```
cuentaAhorro ahorrador;
// ...
scanf("%s", ahorrador.nombre);
// ...
scanf("%d", &ahorrador.numeroCuenta);
// ...
scanf("%f", &ahorrador.montoApertura);
```

Recuerde que la función 'scanf' requiere de la dirección del objeto (que se indica utilizando el operador '&'). Ya que el miembro 'nombre' de la estructura es un apuntador, no es necesario utilizar este operador.

Para ver la implementación completa donde se realicen todas las operaciones solicitadas en el enunciado, abra el archivo 'banco.c' adjunto a este laboratorio.

Como ejercicio modifique el archivo 'banco.c' para obtener la información de un nuevo cliente mediante una nueva función. Se sugiere la función:

```
void obtenerInformacion(cuentaAhorro *c) { ... }
```

Además, modifique la función 'mostrarInformacion' de tal modo que utilice un apuntador al objeto declarado:

```
void mostrarInformacion(cuentaAhorro *c) { ... }
```

2. Diseñe un programa que le permita sumar, restar y multiplicar dos números complejos. Es necesario que defina una estructura que le permita representar números complejos.

Solución:

Utilizando la definición de un número complejo como un par ordenado de números reales, una estructura que permita representar números reales se muestra a continuación:

```
typedef struct {  
    int parteReal;  
    int parteImaginaria;  
} numComplejo;
```

Con esta definición es posible operar números complejos como si se trataran de tipos de objetos "nativos" de C. Por ejemplo, una implementación de la suma podría ser la que se muestra a continuación:

```
numComplejo sumarComplejos(numComplejo c1, numComplejo c2) {  
    numComplejo solucion;  
    solucion.parteReal = c1.parteReal + c2.parteReal;  
    solucion.parteImaginaria = c1.parteImaginaria +  
c2.parteImaginaria;  
    // IMPORTANTE: Se retorna un objeto de tipo numComplejo  
    return solucion;  
}
```



Note que con la definición formal de número complejo como un par ordenado, es posible operar números complejos utilizando la estructura 'point' dada al principio de este laboratorio.

Para ver la implementación completa donde se realicen todas las operaciones solicitadas en el enunciado, abra el archivo 'complejos.c' adjunto a este laboratorio.

Como ejercicio modifique el archivo 'complejos.c' para que utilice la estructura 'point' en lugar de 'numComplejo'.

3. La comunidad de art-lovers DeviArt requiere de un programa en C que le permita registrar a nuevos artistas en su base de datos. Haga un programa que le permita registrar el nombre real, el apodo o nickname, el número total de obras realizadas y el nombre de la obra más importante o representativa del artista. El número máximo de artistas que debe tener la base de datos debe ser pasado por consola. El programa también debe permitir ver la información de todos los artistas registrados.

Solución

Paso 0: Utilizar la Plantilla #2 que está en la sección de Recursos.

```
/*
 * Name      : deviant.c
 * Author    : Juangui Restrepo
 * Compilation : gcc -Wall deviant.c -o deviant
 * Execution  : ./deviant 10
 */

/* Including headers or libraries. */
#include <stdio.h>      // printf, scanf, setbuf, fflush
#include <stdlib.h>     // malloc, free
#include <string.h>     // String handling

/* Definition of macros and constants. */
/* Definition of structures. */
/* Declaring functions. */
/* Global variables. */
/* Main function. */
int main(){
    /* Local variables. */
    return 0;
}
/* Implementation of functions. */
```


Paso 1: Definir la estructura con la información dada en el enunciado

```
#define MAX_CHARS 15
// ...
typedef struct {
    char realName[MAX_CHARS];
    char nickname[MAX_CHARS];
    char masterpiece[MAX_CHARS];
    int artworks;
} artist;
```

Paso 2: Validar los argumentos de consola

```
// En Main:
int maxArtists = 0;
// ...
if (argc != 2) {
    printf("Error: Numero de parametros invalido.\nDebe
iniciar asi: ./deviart.out <numero max artistas>\n");
    exit(0);
}

maxArtists = atoi(argv[1]);

if (maxArtists <= 0) {
    printf("Error: El parametro ingresado debe ser mayor a
cero.\n");
    exit(0);
}
```

Paso 3: Reservar (y liberar) espacio en memoria

```
// En Main:
artists = (artist *)malloc(sizeof(artist) * maxArtists);
// ...
free(artists);
```

Paso 4: Obtener la opción seleccionada por el usuario

```
// Nueva función:
void getOption(char *_option) {
    printf("\nDeviArt\n");
```

```

    printf("\ta) Registrar artistas\n");
    printf("\tb) Ver lista de artistas\n");
    printf("\tq) Salir del programa\n");
    printf("Seleccione una opcion: ");
    setbuf(stdin, NULL);
    scanf("\n%c", _option);
}

// En Main:
char option = 'x';
// ...
do {
    getOption(&option);
    switch (option) {
        case 'a':
            registerArtist(artists);
            break;
        case 'b':
            showArtist(artists);
            break;
        case 'q':
            printf("Saliendo del programa...\n");
            break;
        default:
            printf("La opcion '%c' no es valida. Volviendo
al menu.\n", option);
            break;
    }
} while(option != 'q');

```

Paso 5: Registrar nuevos artistas en cualquier momento

```

// Nueva funcion
void registerArtists(artist *art, int *n, int max) {
    int i = 0;
    int newArtists = 0;

    printf("Ingrese el numero de artistas a registrar: ");
    setbuf(stdin, NULL);
    scanf("%d", &newArtists);

    if (*n + newArtists > max) {

```

```

        printf("No es posible registrar %d artistas.\n",
newArtists);
        printf("La capacidad maxima es %d y actualmente estan
registrados %d.\n", max, *n);
        return;
    }

    for (i = *n; i < *n + newArtists; i++) {
        printf("\nNombre del artista #%d: ", i + 1);
        setbuf(stdin, NULL);
        scanf(" %s", art->realName);

        printf("\nNickname: ");
        setbuf(stdin, NULL);
        scanf(" %s", art->nickname);

        printf("\nObra maestra: ");
        setbuf(stdin, NULL);
        scanf(" %s", art->masterpiece);

        printf("\nNumero de obras realizadas: ");
        setbuf(stdin, NULL);
        scanf(" %d", &art->artworks);

        art++;
    }
    *n += newArtists;
    printf("\nSe han registrado %d artistas en la
lista.\n", newArtists);
}

```

Paso 6: Mostrar información de los artistas

```

// Nueva funcion
void showArtists(artist *art, int n) {
    int i = 0;
    printf("\nLista de artistas:\n");
    printf("|%-15s|%-15s|%-15s|%-15s|\n", "Nombre", "Nick",
"Obra Maestra", "Obras realizadas");
    for (i = 0; i < n; i++) {
        printf("|%-15s|%-15s|%-15s|%-15d|\n", art->realName,
art->nickname, art->masterpiece, art->artworks);
    }
}

```

```

        art++;
    }
}

```

Paso 7: Declarar todas las funciones o prototipos

```

void getOption(char *_option);
void registerArtists(artist *art, int n);
void showArtists(artist *art, int n);

```

Pasos intermedios:

Por cada paso realizado anteriormente, haga una prueba, ejecute el programa y cuando finalmente no presente errores, su programa debe mostrar resultados como los siguientes:

```

Se han registrado 2 artistas en la lista.

DeviArt
    a) Registrar artistas
    b) Ver lista de artistas
    q) Salir del programa
Seleccione una opcion: b

Lista de artistas:
|Nombre      |Nick      |Obra Maestra |Obras realizadas|
|Jason       |xione     |LoboAzul    |105              |
|Carlos      |vives     |encontrados  |8                |

```

Para ver la implementación abra el archivo 'deviart.c' adjunto a este laboratorio.

2. Archivos

El manejo básico de un archivo puede dividirse en los siguientes pasos:

- | | | |
|------|-----------------------------------|-------------------|
| I. | Abrir el archivo | fopen |
| II. | Leer los datos en el archivo | fscanf, fgets... |
| III. | (Opcional) Escribir en el archivo | fprintf, fputs... |
| IV. | Cerrar el archivo | fclose |

Para el paso I. siempre será necesario validar que el archivo se abrió correctamente.

Y para el paso II. será necesario conocer previamente el formato de los datos contenidos en el archivo (*¿Es un archivo con datos separados por comas? ¿Por punto y coma? ¿Por saltos de línea? ¿Es un archivo con una sola línea? ¿Utiliza comillas? ¿Cuenta con una línea de encabezado?*).

Para saber el formato del archivo utilice algún editor o cualquier otro programa que acepte el formato que desea procesar.

Ejemplos de algunos formatos	
CSV	Curso,#Estudiantes,#Ganaron,#Año
	"SO", 25, 25, 2016 "LYR", 60, 3, 2016
Tabulado	Nombre Edad Dirección
	Daniel 20 Calle X Carrera Y Eliana 21 Calle W Carrera M Mario 28 Calle Z Carrera N
DVS	Año;Fabricante;Modelo;Longitud
	1997;Ford;E350;2,34 2000;Mercury;Cougar;2,38
JSON	{ "employees": [{"firstName": "John", "lastName": "Doe"}, {"firstName": "Anna", "lastName": "Smith"}, {"firstName": "Peter", "lastName": "Jones"}] }

Para aprender a manipular ficheros observe el siguiente ejemplo de lectura:

a. Abra la consola y ejecute los siguientes comandos:

```
echo "Hola mundo" > input.txt
```

```
cat input.txt
```

```
hexdump -C input.txt
```

b. Escriba y ejecute el siguiente programa:

```
#include <stdio.h>
#include <stdlib.h>
const char FILE_NAME[] = "input.txt";
int main()
{
    int count = 0; // numero de caracteres leidos
    FILE *in_file;
    int ch;
    /* Abrir el archivo. */
    in_file = fopen(FILE_NAME, "r"); // abrir archivo en modo lectura
    // SIEMPRE realice esta verificacion:
    if(in_file == NULL) // no existe el archivo
    {
        printf("No puede abrirse %s\n", FILE_NAME);
        exit(8);
    }
    /* Leer el archivo. */
    while (1) {
        ch = fgetc(in_file);
        printf("%c\n", ch);
        if(ch == EOF) // lea hasta el final del archivo
        {
            break;
        }
        ++count;
    }
    fputc("x", in_file);
    printf("Numero de caracteres en el archivo %s: %d\n", FILE_NAME, count);

    /* Cerrar el archivo. */
    fclose(in_file);
    return 0;
}
```

c. Responda:

- i. ¿Qué hace 'fopen'? ¿Que significa el argumento 'r' de 'fopen'?
- ii. ¿Qué retorna 'fopen' en caso de éxito y en caso de falla?
- iii. ¿Qué es 'EOF'?
- iv. ¿Cuál es la diferencia entre 'fgetc' y 'getc'?
- v. ¿Porque el valor retornado por 'fgetc' debe almacenarse en un int?

Es fácil notar que las operaciones realizadas con un fichero son similares a las operaciones utilizadas en la entrada y salida de datos en consola. Muchas funciones tienen nombres similares, anteceditas por la letra 'f' (de file):

En consola

```
scanf  
printf  
getc  
...
```

En archivos

```
fscanf  
fprintf  
fgetc  
...
```

La diferencia fundamental será que los métodos relativos a los archivos solicitarán la dirección del archivo. Veamos un ejemplo de escritura muy básico:

En consola:

```
int n1 = 1;  
int n2 = 2;  
int n3 = 3;  
printf("%d;%d;%d", n1, n2, n3); // Para un formato separado por ;
```

En archivos

```
int n1 = 1;  
int n2 = 2;  
int n3 = 3;  
FILE *in_file;  
in_file = fopen("numeros.csv", "w+"); // ¿Qué significa  
                                         //'w+'?  
  
// Note la similitud con printf:  
fprintf(in_file, "%d;%d;%d", n1, n2, n3);  
fclose(in_file);
```

En resumen:

	Abrir un archivo	Leer un archivo	Escribir en el archivo	Cerrar el archivo
Función(es)	fopen(descriptor del archivo, modo)	fscanf fgetc fgets ...	fprintf fputc fputs ...	fclose(descriptor)
Opciones y consideraciones	<u>Modos:</u> <u>"r": read</u> Abrir para leer <u>"w": write</u> Abrir para escribir <u>"a": append</u> Abrir al final del archivo (lee y escribe al final). <u>"r+": read/update</u> Abrir para actualizar (leer y escribir) <u>"w+": write/update</u> Crear nuevo archivo. Si ya existe el archivo con el nombre ingresado, descartelo. <u>"a+": append/update</u> Abrir o crear para leer o actualizar al final. Siempre valide que el archivo fue abierto correctamente.	Usar 'rewind' luego de leer (excepto con append). Rewind devuelve el apuntador del archivo al inicio de este.	Usar 'rewind' si es necesario (excepto con append)	La validación es opcional.

Ejercicio:

Consulte las funciones 'fprintf', 'fscanf', 'fgets', 'fputs', 'fputc' y 'fgetc'. Invente un pequeño programa que haga uso de todas estas funciones.

Taller

1. Dadas las siguientes estructuras, cree un programa que reciba desde consola dos puntos e imprima el área del rectángulo que representan.

<pre>struct point { int x; int y; };</pre>	<pre>typedef struct { struct point point1; struct point point2; } rect;</pre>
--	---

2. Utilice la estructura 'cdMusic' mostrada anteriormente para construir un programa que solicite la información de 'n' CDs. El parámetro 'n' debe ingresarse por consola.
3. Complete el siguiente programa de tal modo que:
 - a. Imprima la información de todos los miembros de la estructura, utilizando tanto la estructura como el apuntador.
 - b. Añada una función: al presionar la tecla 'r' se pueda actualizar la información de la estructura e imprima los nuevos valores.

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

```
struct estudiante {  
    char nombre[40];  
    int numEstudiante;  
    int agnoMatricula;  
    float nota;  
};
```

```
int main() {  
    struct estudiante student1 = {"Pablo", 4, 2002, 4.8};  
    struct estudiante ptrStudent1 = &student1;  
    printf("Nota del estudiante: %f (con apuntador)\n",  
ptrStudent1->nota);  
    printf("Nota del estudiante: %f (sin apuntador)\n",  
student1.nota);  
    getchar();  
    return 0;  
}
```

4. Haga un programa que copie el contenido de un archivo de origen a un archivo destino. Los nombres de los archivos deben pasar como argumentos por consola. Además, el programa debe verificar que el archivo de origen no tiene la condición EOF.
5. Realice un programa que permita calcular el promedio ponderado obtenido por un estudiante en el semestre, considerando:

EL programa recibe el archivo de entrada que se muestra a continuación:	El programa debe generar el archivo que se muestra a continuación:
<pre>// materias.txt: Adjunto a este taller materia nota créditos materia nota créditos ...</pre>	<pre>// salida.txt Materia Nota Créditos Total de materias: ... Materias ganadas: ... Materias perdidas: ... Promedio ponderado: ...</pre>

Debe hacer uso de estructuras y de memoria dinámica para almacenar la información del estudiante.

6. Realice un programa que lea el archivo 'pacientes.txt' adjunto al taller. El programa debe generar dos archivo:
 - a. 'pacientesPorEdad.txt' que muestre a los pacientes ordenados de mayor a menor por edad.
 - b. 'pacientesPorEstatura.txt' que muestre a los pacientes ordenados de menor a mayor por estatura.

Debe hacer uso de una estructura y de memoria dinámica para almacenar la información de los pacientes.