



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: M.I. Edgar Tista García

Asignatura: Estructura de Datos y Algoritmos I

Grupo: 01

No de Práctica(s): 03-Tipo de dato abstracto

Integrante(s): Figueroa Ruiz Carolina

*No. de Equipo de
cómputo empleado:* Trabajo en casa

No. de Lista o Brigada:

Semestre: 2022-2

Fecha de entrega: 1 de marzo de 2022

Observaciones:

CALIFICACIÓN: _____

OBJETIVOS

Objetivo general

Utilizarás estructuras en lenguaje C para modelar tipos de dato abstracto e implementarlos en las estructuras de datos lineales.

Objetivo de clase

Diseñar e implementar estructuras para el desarrollo de un programa real, así como combinar el uso de estructuras con funciones para un mejor desarrollo.

ACTIVIDADES PREVIAS

Ejercicio1

Para este primer ejercicio se nos presentó un código que nos empieza a adentrar en el tema de esta práctica número 3, el cual es el estudio de los **Tipo de dato abstractos**, que, en Lenguaje C se crean a través de **estructuras**.

Primeramente, durante las clases teóricas estudiamos en concepto de **Tipo de dato abstracto** y su principal función, se dijo que es cualquier tipo de dato que el programador construya a su imaginación para poder utilizar en los programas desarrollados, esto crea lo que conocemos como una **capa de abstracción** en la programación.

Como mencionamos, y como vamos a ver a continuación, una **estructura** es una colección de una o más características llamadas miembros, los cuales pueden ser de diferentes tipos de datos, los cuales pueden ser **simples** y/o **compuestos**. Los tipos de datos simples (o primitivos) son, por ejemplo: carácter, números enteros o números de punto flotante (reales). Los tipos de datos compuestos son: los arreglos y las estructuras.

Ahora podemos entender mejor lo que nos presenta este primer ejercicio, como vemos al inicio lo primero que nos muestra es la declaración de la estructura, la cual tiene 5 diferentes miembros (de diferentes tipos de datos) y esta creando lo que va a ser el **tipo de dato** "película".

```
struct pelicula{  
    char *nombre;  
    char *genero;  
    short anio;  
    short numDirectores;  
    char *directores[10];  
};
```

Fig. 1. Fragmento de código para ejemplificar.

Toda declaración de estructura debe cumplir con la sintaxis anterior, la cual, más explícitamente nos muestra: Al inicio la palabra reservada **struct**, seguido del nombre de nuestra estructura (todo esto representa al tipo de dato), y dentro de las llaves están declarados cada uno de los miembros (con su tipo de dato y nombre), terminamos con el fin de llave y su respectivo ";".

Posteriormente nos declaran 2 funciones que nos van a ser útiles a lo largo del programa y de las cuales vamos a hablar después, solo vamos a hacer un pequeño paréntesis para recordar la importancia de declarar y usar bien los nombres completos de las estructuras que estamos creando, ya que para las funciones venían mal los nombres de éstas en el código del manual.

Como podemos observar en este código, la función **main** en realidad es muy corta, y esto gracias a nuestra estructura y nuestras funciones. Inicia declarando un arreglo de apuntadores, el cual nos sirve para poder guardar cadenas de caracteres, y en este caso para poder guardar nombres de directores cinematográficos (ya tenemos inicializados 2 de ellos en las posiciones 0 y 1 del arreglo).

Y por último (dentro de la función principal) tenemos el llamado a dos funciones secundarias. La primera es una función que recibe parámetros y que devuelve un valor, pero como podemos observar, el tipo de valor que va a devolver es del tipo de dato abstracto que fue creado en la estructura (es decir, **struct película**), lo que va a hacer es recibir los datos correspondientes para poder llenar una variable de tipo estructura que guarde el conjunto de características. Estos datos ya fueron inicializados como parámetros de la función solamente para hacer ya la asociación, la cual se hace con el nombre de la variable tipo estructura, seguida de un punto (.) y el nombre del miembro al que le vamos a asignar el dato.

```
struct pelicula movie;
movie.nombre = nombre;
movie.genero = genero;
movie.anio = anio;
movie.numDirectores = numDirectores;
int cont = 0;
for ( ; cont < movie.numDirectores ; cont++){
    movie.directores[cont] = directores[cont];
}
```

Fig. 2. Fragmento de código para ejemplificar.

Como observamos también hacemos uso de un ciclo **for**, y esto es porque tenemos declarado un arreglo (de apuntadores) como uno de los miembros de nuestra estructura y para poder llenar las posiciones se hace uso de este ciclo y su respectivo contador que va a controlar los datos ingresados.

La primera función termina y devuelve como valor a la variable tipo estructura "movie" la cual va a ser el parámetro por recibir para la siguiente función secundaria que nos va a ayudar a imprimir los datos que fueron usados para llenar la variable tipo estructura. Para hacer esto, aprendimos que debe hacerse miembro por miembro pues, como son de diferentes tipos de datos, no hay forma de imprimir una estructura completa.

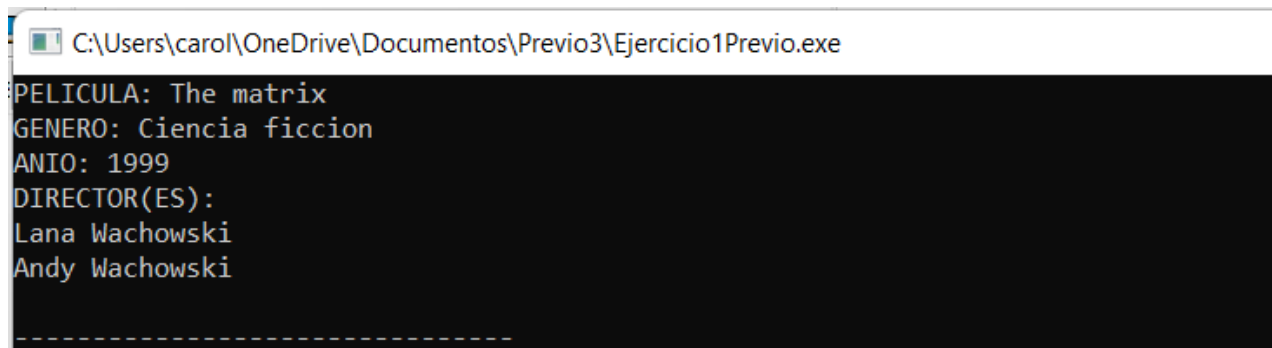
```
printf("PELICULA: %s\n", movie.nombre);
printf("GENERO: %s\n", movie.genero);
printf("ANIO: %d\n", movie.anio);
printf("DIRECTOR(ES):\n");
int cont = 0;
for ( ; cont < movie.numDirectores ; cont++){
    printf("%s\n", movie.directores[cont]);
}
```

Fig. 3. Fragmento de código para ejemplificar.

Como observamos ponemos el tipo de dato con el uso del % y después de la coma ponemos el nombre de la variable tipo estructura, seguido de un punto y el nombre del miembro que queremos mostrar. El ciclo **for** en este caso esta implementado para poder imprimir el arreglo que tiene las cadenas de caracteres con los nombres de los directores, para este caso el ciclo, con su contador, van recorriendo el arreglo e imprimiendo los datos de cada posición de este.

Ejecución:

Nos muestra la impresión de los datos ingresados en la estructura "movie".



```
C:\Users\carol\OneDrive\Documentos\Previo3\Ejercicio1Previo.exe
PELICULA: The matrix
GENERO: Ciencia ficcion
AÑO: 1999
DIRECTOR(ES):
Lana Wachowski
Andy Wachowski
-----
```

Fig. 4. Ejecución del ejercicio 1 del previo

Ejercicio 2

En este código lo primero que podemos notar al irlo programando es que tiene algunos errores básicos de sintaxis, por ejemplo, con el **scanf** al guardar un valor, este debe dirigirse a la dirección de memoria de la variable, por lo tanto, ocupamos un **&**. Y bueno, algunos acentos que con el código ASCII se pudieron mostrar en pantalla.

Analizando el código, podemos ver que lo primero que nos presentan es la declaración de la **estructura** que vamos a estar ocupando para el programa, y en donde definimos un nuevo **tipo de dato**, el cual es "**struct** película" y sus respectivos miembros (5 en donde uno de ellos es un arreglo bidimensional de tipo carácter para guardar a los directores).

Igualmente vemos la función principal **main** de un tamaño muy pequeño, lo que es bueno pues hace que el programa sea mucho más práctico, simple y sencillo de entender, esto gracias a las funciones que encontramos dentro y la estructura. Primero vemos una variable declarada de tipo estructura, pero ahora esta variable es compleja pues se trata de un arreglo, lo cual podemos interpretar (para su mayor comprensión) como que dentro de esa variable va a haber 2 estructuras distintas, ya que su tamaño es **TAM** y éste está definida anteriormente como constante que vale 2).

Posteriormente se manda a llamar a nuestra primera función, una del tipo que no regresa valor, pero si recibe parámetros, el cual es el arreglo de tipo "**struct** película". Y lo que se busca aquí, es llenar ambas estructuras que están dentro de este arreglo. Para eso necesitamos un primer ciclo **for** con su respectivo contador que recorra el arreglo y en cada posición de este arreglo, vaya llenando los miembros de cada estructura correspondiente.

Sin embargo, otra cosa a resaltar es que un miembro de cada estructura es un arreglo también, el cual en su interior guarda en cada posición el nombre de un director de película (ya que igualmente su tamaño es una constante que vale 2), y por esta razón es que necesitamos otro ciclo **for** anidado al anterior, que nos vaya pidiendo el nombre de ambos directores (y guardándolos) para cada estructura.

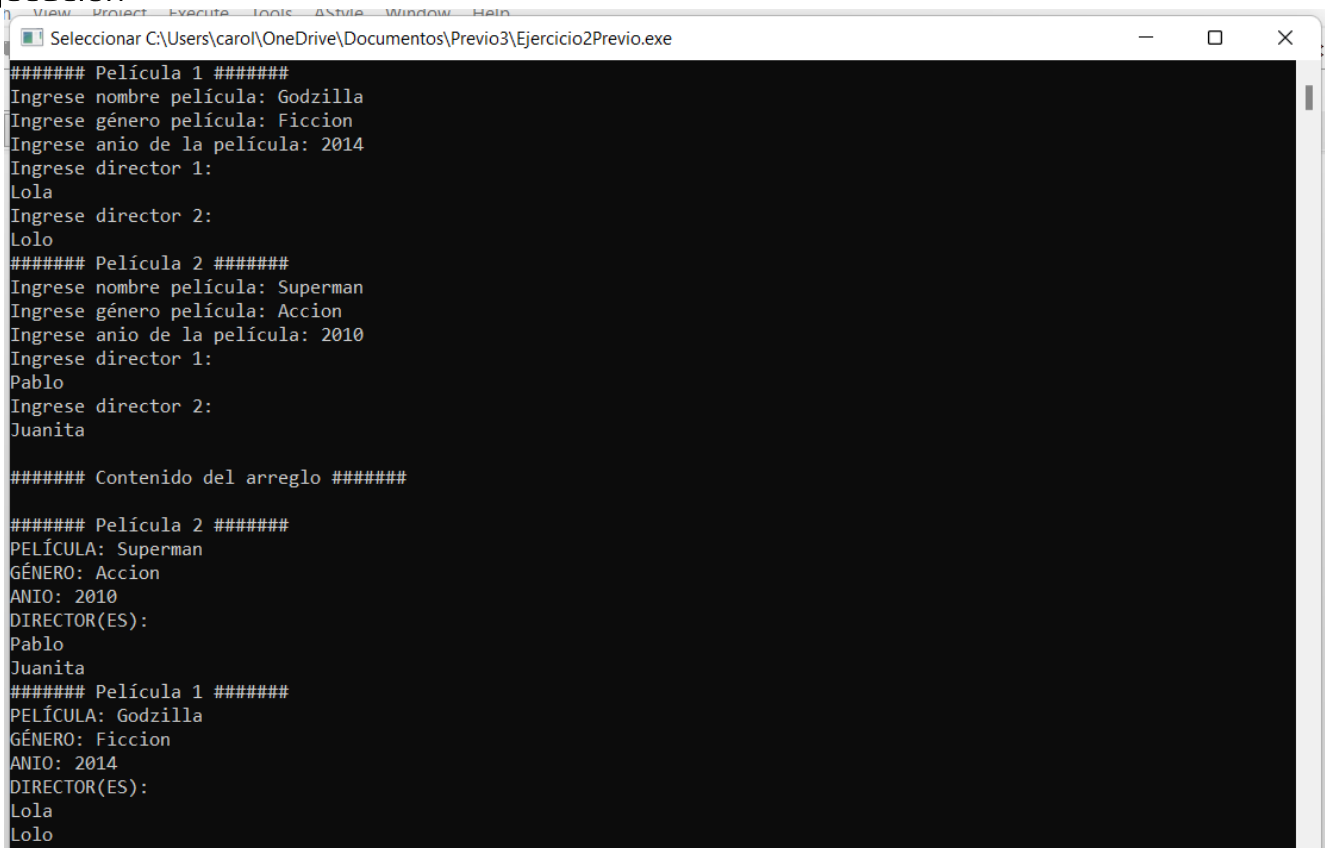
Una vez que esta función se acaba, regresamos al programa principal, en donde como siguiente paso se manda a llamar a la segunda función que vamos a ocupar, que tampoco regresa valor, pero si recibe parámetros, y la cual nos va a ayudar a imprimir el arreglo llenado anteriormente. Por lo que el parámetro que va a recibir es igualmente el arreglo de tipo “**struct** película” y vamos a ocupar 2 ciclos **for** anidados para la impresión de cada elemento del arreglo y de sus respectivos arreglos en cada estructura que guardan los nombres de ambos directores. Sin embargo, analizando el primer ciclo **for**, la manera en que está planteado nos indica que va a empezar a presentar los datos desde la última posición del arreglo hasta la primera, es decir, de la última posición que fue llenada (película 2) en el arreglo, hasta la primera (película 1), presentando sus respectivos miembros de cada una.

También aquí podemos observar la estructura para llenar e imprimir los miembros de cada estructura que está en el arreglo, especificando el nombre del arreglo, su posición (que hace referencia al número de estructura correspondiente), seguida de un punto y el nombre del miembro que se manipulará.

Otra cosa que podemos notar es el uso de instrucciones nuevas como lo son:

Las cuales sirven para lo que se conoce en lenguaje C como “limpiar el buffer” y se utiliza para que antes de guardar un dato solicitado al usuario se vacía lo que podría estar dentro para dejarlo listo al nuevo valor.

Ejecución



```
##### Película 1 #####
Ingrese nombre película: Godzilla
Ingrese género película: Ficción
Ingrese año de la película: 2014
Ingrese director 1:
Lola
Ingrese director 2:
Lolo
##### Película 2 #####
Ingrese nombre película: Superman
Ingrese género película: Accion
Ingrese año de la película: 2010
Ingrese director 1:
Pablo
Ingrese director 2:
Juanita

##### Contenido del arreglo #####

##### Película 2 #####
PELÍCULA: Superman
GÉNERO: Accion
AÑO: 2010
DIRECTOR(ES):
Pablo
Juanita
##### Película 1 #####
PELÍCULA: Godzilla
GÉNERO: Ficción
AÑO: 2014
DIRECTOR(ES):
Lola
Lolo
```

Fig. 5. Ejecución del ejercicio 2 del previo

Ejercicio 3 (Previo 2.0)

Este ejercicio se nos fue presentado para poder practicar más en este tema de **tipos de datos abstractos**, que aquí manejamos a través de las estructuras. Y para este programa lo que nos solicitan es:

La creación de un arreglo de estructuras:

- Programar la estructura alumno
- Crear la estructura Asignatura
- Agregar a la estructura alumno un arreglo de asignaturas (5)
- Inicializar 2 alumnos con al menos 3 materias
- Permitir al usuario ver los datos del alumno, incluyendo las materias
- Permitir al usuario crear alumnos nuevos y llenar todos sus datos desde 0 (incluyendo materias).

Extra Opcional: Permitir al usuario modificar cualquier dato del alumno (nombre, número de cuentas de sus materias)

Como nos indican las instrucciones, lo primero que se hizo fue la declaración de la estructura **asignatura**, en donde usamos un recurso que estudiamos en clase para omitir el tener que poner la palabra reservada **struct** cada vez que escribamos el **tipo de dato** del que se trata, esta herramienta es el **typedef struct**, en donde dentro, definimos los miembros de nuestra estructura y AL FINAL, después de la llave que cierra y antes del “;” ponemos el nombre de nuestra estructura (en este caso “asignatura”), y así, con el uso del **typedef**, este espacio ya no es usado para declarar variables globales del tipo de la estructura sino para asignarle el nombre con el que la estaremos manejando.

```
typedef struct{
    char nombreMateria[15];
    int clave;
    char nombreProfesor[20];
}asignatura;
```

Fig. 6. Fragmento de código para ejemplificar.

Posteriormente tenemos la declaración de la estructura **alumno** (igualmente utilizando **typedef**) y debe ir después porque dentro de ésta, como uno de sus miembros, vamos a utilizar un arreglo unidimensional de tamaño 5 y además de tipo **asignatura**, el cual va a ser usado para almacenar 5 estructuras que nos van a representar 5 asignaturas que lleva el alumno (con sus respectivas características definidas en la estructura anterior).

```
typedef struct{
    char nombre[20];
    char apellido[20];
    int edad;
    int numCuenta;
    int numA;
    asignatura bloqAsig[5]; //Estructura anidada dentro de otra
}alumno;
```

Fig. 7. Fragmento de código para ejemplificar.

Después entramos ya a la función principal **main** y aquí nos piden que sean inicializados 2 alumnos (es decir, dos variables de tipo de dato **alumno**) con sus características y al menos 3 asignaturas para cada uno. Lo que se utilizó fue un arreglo de tipo **alumno** (con capacidad para 10 de ellos) en donde vamos a poder almacenar en cada posición a un alumno distinto con sus respectivos miembros (incluyendo asignaturas).

Lo primero que hacemos es inicializar el arreglo en 2 posiciones, es decir, para dos alumnos, y como vimos en clase, al inicializar podemos hacer uso de llaves e ir poniendo los datos EN EL ORDEN DECLARADO DENTRO DE LA ESTRUCTURA, y como también podemos observar, hay llaves dentro de llaves, pues, dentro de cada alumno tenemos como un miembro, a un arreglo de tipo **asignatura** que nos va a inicializar 3 materias que éstos llevan (con sus miembros respectivos pues es una estructura). Representándonos lo que sería una estructura dentro de otra. Ojo que debemos tener cuidado con el manejo de las llaves para no confundirnos.

```
alumno alu[10]={{ "Carlos", "Ruiz", 18, 319841723, 3, {{ "Calculo", 1126, "Colome" }, { "Mecanica", 1045, "Erica" }, { "Programacion", 6453, "Cinthia" } }}, {
```

Fig. 8. Fragmento de código para ejemplificar.

Posteriormente se hace la presentación del programa y del menú que se va a estar repitiendo y que nos va a mostrar las opciones que podemos elegir para trabajar, las cuales son:

1. Mostrar los alumnos ya inicializados, con sus miembros.
2. Registrar un alumno nuevo, es decir, una posición nueva dentro del arreglo de tipo alumno.
3. Modificar algún dato de un alumno.
4. Y salir (dejar de mostrar el menú).

Para hacer posible el manejo del menú (que está dentro de un ciclo **while** para que se esté repitiendo) se va a utilizar una instrucción de decisión múltiple con la que ya habíamos trabajado, la cual es **switch**, con cada uno de sus casos posibles (4 de las opciones del menú y su default). Para las primeras 3 opciones vamos a hacer uso de funciones secundarias.

```
void mostrarAlum();  
void registrarNuevo();  
void modifAlum();
```

Fig. 9. Fragmento de código para ejemplificar.

Si se escoge la opción de mostrar a los alumnos ya inicializados, lo que vamos a hacer es ir a la función secundaria que no regresa valor, pero sí recibe parámetros (recibe al arreglo de tipo estructura alumno que guarda nuestra información). Y la cual nos va a servir para imprimir cada posición del arreglo, es decir, a cada alumno con sus respectivos miembros inicializados.

Para esto lo que ocupamos es un ciclo **for** que nos permita recorrer el arreglo con las estructuras de alumnos que tenemos inicializadas y posteriormente ir imprimiendo cada posición del arreglo (indicado con el contador), seguido de un punto y el nombre del miembro que queremos ver. Pues recordemos que no se pueden imprimir estructuras

completas, sino que debe ser miembro por miembro.

Sin embargo, recordamos que un miembro de cada alumno es además un arreglo de tipo **asignatura** que tiene guardado la información de 3 estructuras más (sus asignaturas con sus respectivos miembros), por lo que para imprimir éstas también hacemos uso de otro ciclo **for**, el cual nos va a permitir seguir recorriendo el arreglo principal, pero además recorrer el arreglo de cada una de las estructuras tipo asignatura que tiene el último miembro de cada alumno (con otro contador claro), e igualmente especificando la posición del arreglo principal, la posición del arreglo dentro y después del punto el miembro que queremos ver de éste último.

```
printf("Nombre de la materia %d: %s\n", j+1, alu[i].bloqAsig[j].nombreMateria);  
printf("Clave: %d\n", alu[i].bloqAsig[j].clave);
```

Fig. 10. Fragmento de código para ejemplificar.

Si el usuario escoge la opción de registrar un alumno nuevo, es decir, una posición nueva del arreglo tipo **alumno**, nos va a mandar a la función **registrarNuevo**, y lo que se va a preguntar primero es cuántos alumnos nuevos van a ser y cuantas materias para cada uno, esto para poder llevar los contadores de nuestro arreglo principal y del que está dentro de este (asignaturas).

E igualmente ingresamos al primer ciclo **for** en donde se va solicitando y guardando cada miembro para la nuestra posición (estructura) que se va a registrar, con la posición del arreglo que le corresponde, para la cual se debe usar un contador **cont** que sea usado como la primera posición para registrar a un nuevo alumno y también sabremos hasta donde parar de solicitar datos, ya que se le sumará a la variable que guarda los alumnos solicitados; Esto se hace con la finalidad de que cada vez que se quiera registrar un alumno nuevo, no se sobrescriba en la misma posición.

```
for(i=cont; i<n+cont; i++){
```

Fig. 11. Fragmento de código para ejemplificar.

Después de especificar el arreglo con su posición correspondiente, le sigue un punto y posteriormente el nombre del miembro a llenar. Esto se hace hasta llegar al último miembro que ya es un arreglo de tipo **asignatura** y en donde debemos iniciar otro ciclo **for** para poder llenar el arreglo anidado y que el usuario pueda ingresar las características de la cantidad de asignaturas que quiera registrarle al nuevo alumno, especificando la posición del arreglo principal, seguido de punto, la posición del arreglo secundario, otro punto y el miembro específico que se va a ir registrando para cada asignatura. Por último incrementa el contador **cont** con las posiciones que se ingresaron y así llevará la cantidad de alumnos registrados.

Volvemos así a la función principal **main** y acabamos con esa opción. Por lo cual, el menú se repite.

Para el **case 3**, que fue la parte "opcional", lo que se busca es permitirle al usuario modificar alguna característica de algún alumno registrado, incluyendo del arreglo de tipo estructura que está anidado, es decir, alguno de los miembros de las asignaturas registradas.

Para esto se preguntó a cuál alumno se quería modificar y el valor se almacenó en **alum**, posteriormente se presentó un menú de opciones en donde se le daba la opción al usuario de elegir lo que le quería modificar a dicho alumno:

```
printf("1)Nombre\n2)Apellido\n3)Edad\n4)Numero de cuenta\n5)Asignatura\n");
scanf("%d",&op);
```

Fig. 12. Fragmento de código para ejemplificar.

Y el resultado fue almacenado en **op** para poder usarlo como condición en una función de decisión múltiple **switch**, en donde para cada **case** se le solicitaba ingresar el nuevo dato que iba a reemplazar al anterior y se almacenaba en el arreglo de alumnos con la posición **alum** (dato solicitado anteriormente), seguido de un punto y el nombre del miembro al que pertenece.

```
printf("\nIngrese el nuevo nombre para el alumno que eligió: ");
scanf("%s",&alu[alum].nombre);
```

Fig. 13. Fragmento de código para ejemplificar.

Excepto para el **case 5**, ya que este representa a las **Asignaturas**, es decir, al arreglo de tipo estructura que esta anidado, por lo cual ocupamos un procedimiento más detallado. Dentro de este **case** primero que nada se preguntó cuál asignatura deseaba modificar, para el resultado guardarlo en **asig**, la cual nos va a servir más adelante para indicar la posición en el arreglo de asignaturas. Y después se le presentó al usuario otro menú en donde las opciones fueran los miembros de la asignatura elegida que podía modificar:

```
printf("\n¿Que aspecto de la asignatura desea cambiar?");
printf("1)Nombre de la materia\n2)Clave de la materia\n3)Nombre del profesor\n");
scanf("%d",&op2);
```

Fig. 14. Fragmento de código para ejemplificar.

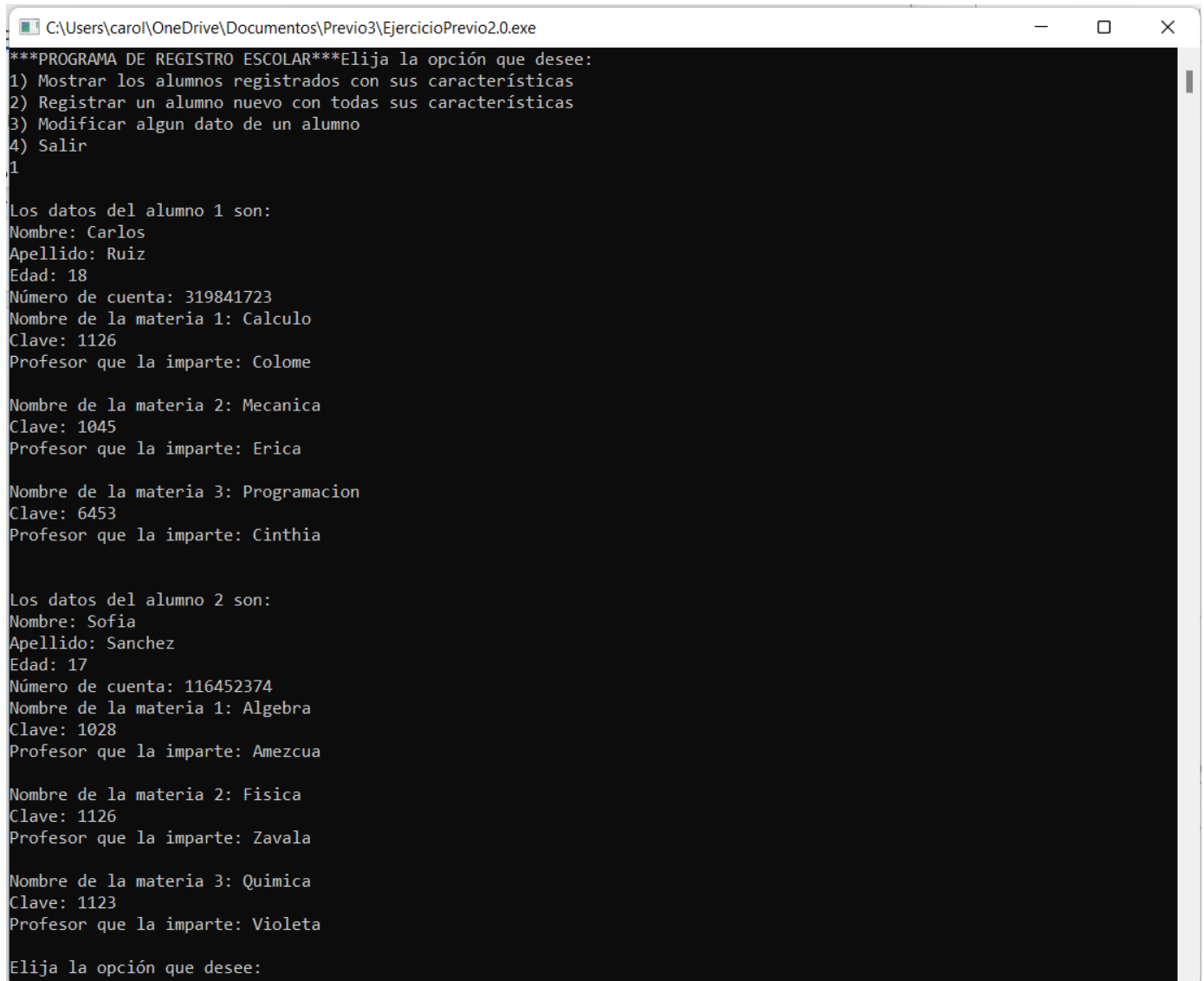
Igualmente, la opción elegida se almacenó en una variable que nos va a servir para utilizar otra decisión múltiple **switch** y poder solicitar (y almacenar) el nuevo dato que va a reemplazar al anterior. Esto se almacena especificando a **alum** como índice del arreglo de alumnos, seguido de un punto y el arreglo de asignaturas con índice **asig**, y por último un punto y el nombre del miembro de la asignatura que se va a modificar.

```
printf("\nIngrese la nueva clave para la asignatura que eligió: ");
scanf("%d", alu[alum].bloqAsig[asig].clave);
break;
```

Fig. 15. Fragmento de código para ejemplificar.

Ejecución

Observamos la presentación del menú y la elección de la opción 1, la cual es mostrar los alumnos que ya están registrado (2), además de que se alcanza a observar la repetición del menú:



```
C:\Users\carol\OneDrive\Documentos\Previo3\EjercicioPrevio2.0.exe
***PROGRAMA DE REGISTRO ESCOLAR***Elija la opción que desee:
1) Mostrar los alumnos registrados con sus características
2) Registrar un alumno nuevo con todas sus características
3) Modificar algun dato de un alumno
4) Salir
1

Los datos del alumno 1 son:
Nombre: Carlos
Apellido: Ruiz
Edad: 18
Número de cuenta: 319841723
Nombre de la materia 1: Calculo
Clave: 1126
Profesor que la imparte: Colome

Nombre de la materia 2: Mecanica
Clave: 1045
Profesor que la imparte: Erica

Nombre de la materia 3: Programacion
Clave: 6453
Profesor que la imparte: Cinthia

Los datos del alumno 2 son:
Nombre: Sofia
Apellido: Sanchez
Edad: 17
Número de cuenta: 116452374
Nombre de la materia 1: Algebra
Clave: 1028
Profesor que la imparte: Amezcu

Nombre de la materia 2: Fisica
Clave: 1126
Profesor que la imparte: Zavala

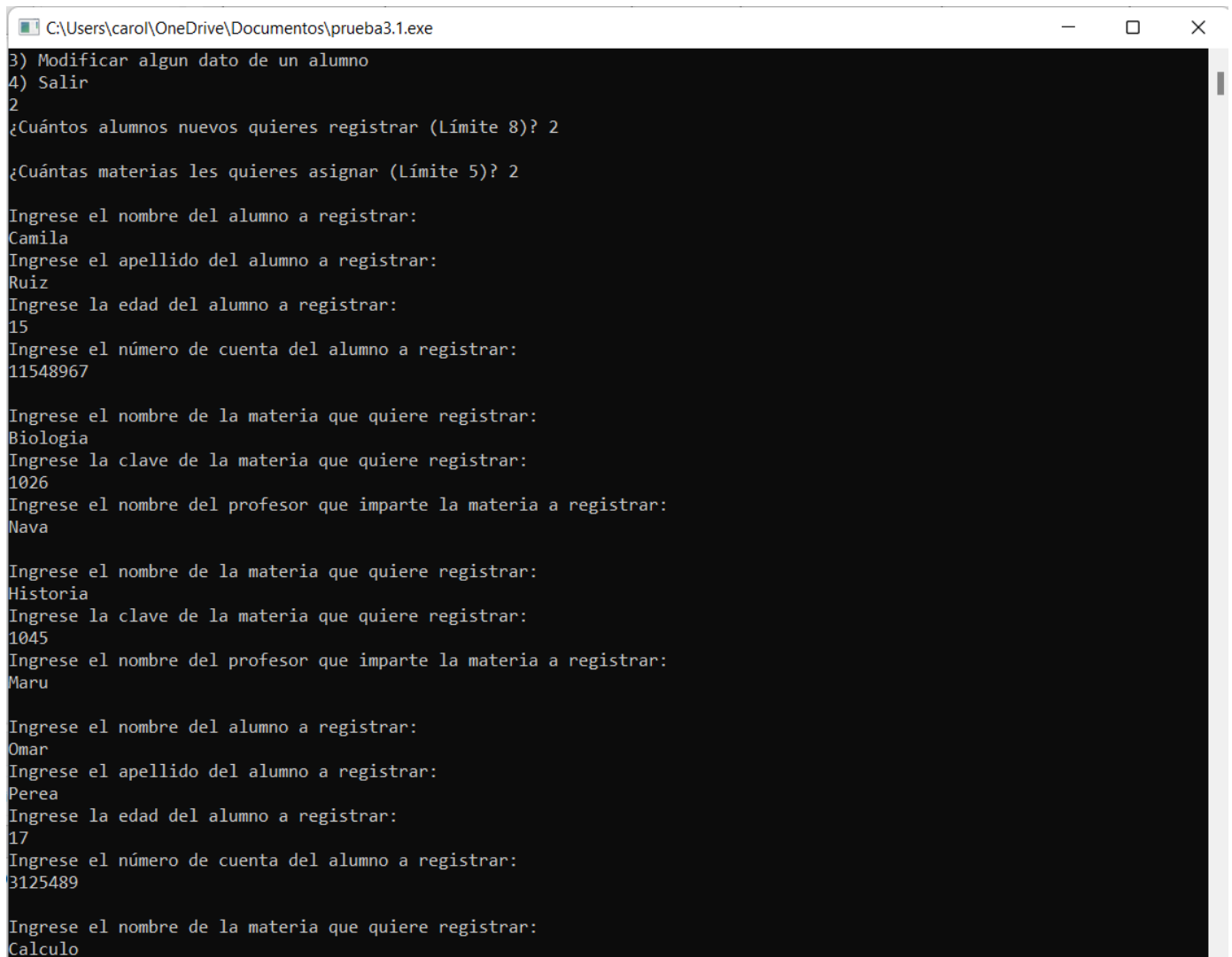
Nombre de la materia 3: Quimica
Clave: 1123
Profesor que la imparte: Violeta

Elija la opción que desee:
```

Fig. 16. Ejecución del ejercicio 3 del previo

Ejecución 2

Aquí observamos un ejemplo sencillo de la elección para la opción 2, la cual es registrar a los alumnos que deseemos con todas sus características, incluyendo asignaturas.



```
C:\Users\carol\OneDrive\Documentos\prueba3.1.exe
3) Modificar algun dato de un alumno
4) Salir
2
¿Cuántos alumnos nuevos quieres registrar (Límite 8)? 2
¿Cuántas materias les quieres asignar (Límite 5)? 2
Ingrese el nombre del alumno a registrar:
Camila
Ingrese el apellido del alumno a registrar:
Ruiz
Ingrese la edad del alumno a registrar:
15
Ingrese el número de cuenta del alumno a registrar:
11548967

Ingrese el nombre de la materia que quiere registrar:
Biología
Ingrese la clave de la materia que quiere registrar:
1026
Ingrese el nombre del profesor que imparte la materia a registrar:
Nava

Ingrese el nombre de la materia que quiere registrar:
Historia
Ingrese la clave de la materia que quiere registrar:
1045
Ingrese el nombre del profesor que imparte la materia a registrar:
Maru

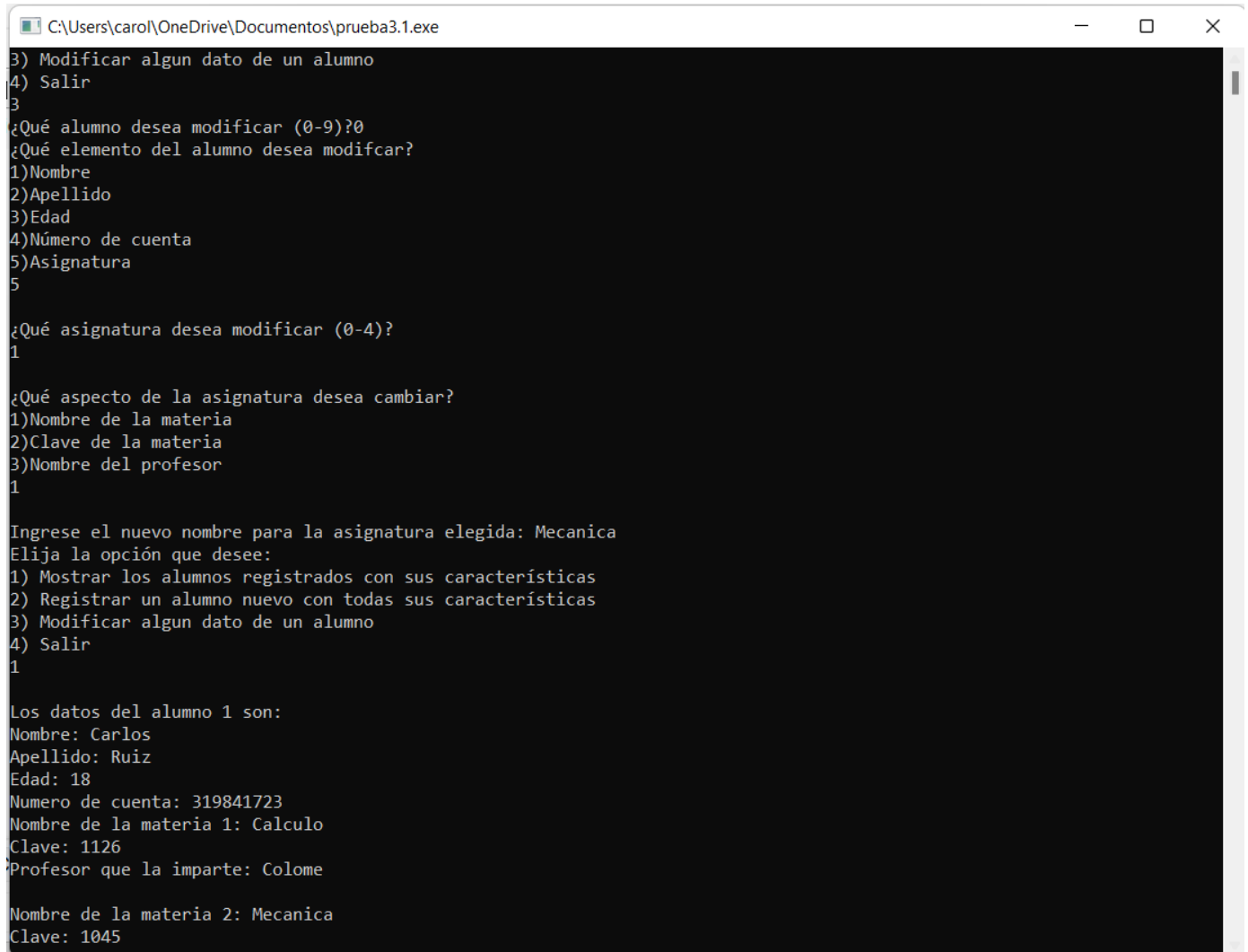
Ingrese el nombre del alumno a registrar:
Omar
Ingrese el apellido del alumno a registrar:
Perea
Ingrese la edad del alumno a registrar:
17
Ingrese el número de cuenta del alumno a registrar:
3125489

Ingrese el nombre de la materia que quiere registrar:
Calculo
```

Fig. 16.1. Ejecución 2 del ejercicio 3 del previo

Ejecución 3

Aquí observamos la ejecución de la opción 3 del menú principal, la cual es poder modificar algún elemento del alumno, incluyendo a sus asignaturas y los miembros de esta estructura anidada:



```
C:\Users\carol\OneDrive\Documentos\prueba3.1.exe
3) Modificar algun dato de un alumno
4) Salir
3
¿Qué alumno desea modificar (0-9)?0
¿Qué elemento del alumno desea modifcar?
1)Nombre
2)Apellido
3)Edad
4)Número de cuenta
5)Asignatura
5
¿Qué asignatura desea modificar (0-4)?
1
¿Qué aspecto de la asignatura desea cambiar?
1)Nombre de la materia
2)Clave de la materia
3)Nombre del profesor
1
Ingrese el nuevo nombre para la asignatura elegida: Mecanica
Elija la opción que desee:
1) Mostrar los alumnos registrados con sus características
2) Registrar un alumno nuevo con todas sus características
3) Modificar algun dato de un alumno
4) Salir
1
Los datos del alumno 1 son:
Nombre: Carlos
Apellido: Ruiz
Edad: 18
Numero de cuenta: 319841723
Nombre de la materia 1: Calculo
Clave: 1126
Profesor que la imparte: Colome
Nombre de la materia 2: Mecanica
Clave: 1045
```

Fig. 16.2. Ejecución 3 del ejercicio 3 del previo

DESARROLLO DE LA PRÁCTICA

Ejercicio 1

Para este primer y único ejercicio de la práctica 3 lo que se busca es diseñar un programa para la administración de una cadena de abarrotes, es decir, un programa que esta más cercano a poder utilizar en la vida real. Para esto debemos crear una serie de estructuras, funciones, arreglos anidados, además de que va a haber estructuras dentro de otras.

Lo que se realizó como primer paso fue la declaración de las estructuras que vamos a estar ocupando a lo largo del programa, y estas son:

- ➔ Producto
- ➔ Empleado
- ➔ Gerente
- ➔ Sección
- ➔ Tienda

En este orden pues en las estructuras declaradas al final (como Tienda y Sección) vamos a necesitar como alguno de sus miembros a otra estructura, la cual tuvo que se declarada antes.

```
typedef struct
{
    int codigoSec;
    char categoria[20];
    Empleado encargadoSec[10];
    Producto productos[10];
    int numE;
    int numP;
} Seccion;

typedef struct
{
    char nombreTie[20];
    int codigoTie;
    Gerente encargadoTie;
    Seccion secciones[5];
    int numS;
} Tienda;
```

Fig. 17. Fragmento de código para ejemplificar.

Después de esto se empezó a implementar la función principal **main**, en la cual lo primero que se realizó fue la presentación del programa y el menú con el que el usuario va a ir escogiendo la opción con la que desea trabajar (esto se hizo dentro de un ciclo **while**), al presentar las opciones (5) me di cuenta que 3 de ellas desde el inicio dan la opción de mostrar tienda/secciones/productos registrados, por lo que decidí inicializar una tienda para que eso fuera lo que se mostrara si el usuario decidiera elegir alguna de estas opciones desde el principio; por esto mismo también regresé al inicio del **main** a declarar un arreglo de tipo de nuestra estructura **Tienda**, que va a guardar ahí dentro tanto a la Tienda inicializada, como a la(s) nueva(s) que el usuario registre.

Para inicializar una Tienda debemos ir miembro por miembro, y como se trata de un arreglo, se debe poner el nombre del arreglo seguido de sus respectivos () para indicar la posición (número de tienda) y seguimos con un punto y el nombre del miembro en esta estructura tipo tienda. Sin embargo, dentro de esta estructura tenemos a su vez un miembro del tipo de otra estructura (Gerente) y un arreglo de tamaño 5 y de otra estructura más (Sección), por lo que ahí empieza lo que llamamos **Estructuras anidadas**, las cuales se definen como estructuras dentro de otras.

Y para manejarlas (inicializarlas en este caso) debemos colocarlas de la misma manera que un miembro normal del arreglo de tipo `Tienda`, pero con la diferencia de que le sigue un punto más, especificando ahora el miembro de esta estructura anidada. Si esto se trata de un arreglo (como es el caso del arreglo anidado tipo `Sección`) se debe indicar también su posición entre sus respectivos `[]`. En otras palabras, se debe especificar cada miembro de cada estructura que este anidada dentro de otras.

```
arrTienda[0].secciones[0].productos[0].precio=45;
```

Fig. 18. Fragmento de código para ejemplificar.

Otra cosa que podemos observar aquí, es el uso de `strcpy`, la cual es una función que debe usarse iniciar arreglos, recibe 2 parámetros, el lugar de destino y la cadena que queremos copiar; Como se puede observar al hacer uso de arreglos de caracteres como miembros de una estructura.

```
strcpy(arrTienda[0].secciones[0].productos[0].nombreProducto, "Zote");
```

Fig. 19. Fragmento de código para ejemplificar.

Después de inicializar una tienda en la posición 0 del arreglo, seguimos trabajando en la función `main`, pero ahora entre del `while`, que es en donde se encuentra el menú y en donde además vamos a utilizar una función de decisión múltiple, `switch`, para poder evaluar la opción que el usuario elija.

El primer caso es para **crear una tienda**, lo cual quería hacer directamente mandando a llamar a una función que lo hiciera, sin embargo, después de algunos intentos fallidos, decidí que lo que iba a hacer era trabajar en las funciones de forma individual (es decir, tienda por tienda) y en el `case` de la función `main` iba a preguntar primero cuántas tiendas se querían registrar y posteriormente hacer uso de un ciclo de repetición `for`, el cual va a repetir las instrucciones que estuvieran dentro para cada tienda deseada (indicando con el contador la posición de la tienda en el arreglo). Para poder crear una tienda con todos los miembros y estructuras anidadas que se solicitan, se hizo uso de varias funciones:

```
printf("%cCu%ntas tiendas m%cs desea registrar (l%cmite 4)? : ",168,160,160,161);
scanf("%d", &t);
for(i=0;i<t;i++){
    arrTienda[count] = crearTienda();
    agregarGerente(&arrTienda[count]);
    agregarSeccion(&arrTienda[count]);
    count++;
}
```

Fig. 20. Fragmento de código para ejemplificar.

La primera fue la función `crearTienda`, la cual no recibía parámetros y sí revolvía un valor de tipo `tienda`, lo que esto nos permitió fue declara una variable de este mismo tipo y permitir que el usuario ingresara el nombre y código de la tienda, los cuales son los miembros más simples ya que son de tipo de datos primitivos y su manejo es sencillo. Posteriormente **retorna** esta variable tipo estructura a la función principal y así almacena estos primeros datos de la tienda.

```
return tiendaEjem;
```

Fig. 21. Fragmento de código para ejemplificar.

Después de eso usamos otra función para poder **agregarle** a la tienda su miembro de tipo **Gerente**, ya que éste tiene varios miembros a su vez dentro. Esta es una función que no regresa valor (**void**), pero que sí recibe parámetro, el cual era el arreglo de tipo tienda (que se pasa como apuntador para trabajar cada tienda de forma individual). Aquí dentro declaramos una variable de tipo **Gerente** y le asignamos el registro de un gerente a través de mandar a llamar otra función. Para posteriormente a nuestra **Tienda** poderle agregar en el miembro que le corresponde (con -> porque la pasamos como apuntador) el registro de este gerente (estructura anidada).

```
Gerente nuevoGerente = registrarGerente();  
tienda->encargadoTie = nuevoGerente;
```

Fig. 22. Fragmento de código para ejemplificar.

La función que se mandó a llamar dentro de la anterior fue **registrarGerente**, la cual ya trabaja con la estructura de **Gerente** de manera individual, es decir que esta función también es **void** porque no recibe parámetros, lo que hace es solicitar los datos para registrar a un gerente (basándonos en los miembros de su estructura) y los asigna uno por uno en cada miembro de la estructura, regresando al final esta variable tipo **Gerente** ya “llena” con la estructura completa, para poder asignarla en la función anterior y a su vez en la **Tienda** principal.

Básicamente la función de **agregarGerente** solo sirve como un puente para conectar o asignarle a **Tienda** su miembro de tipo estructura ya “llenado” por el usuario. Esta función no la implementé desde el principio, pues quería trabajarlo directamente, sin embargo tuve algunos problemas para colocar cada miembro de la estructura anidada dentro de la otra y por eso decidí que sería mejor trabajar de manera individual cada estructura y usar otras funciones intermedias para conectarlas con la estructura principal.

Después de ya tener al miembro de la tienda que guarda al gerente con sus características, regresamos a la función principal **main**, y el siguiente paso a realizar (aún dentro del ciclo **for**) es agregarle las secciones que quiera el usuario para la **Tienda**, es por eso por lo que, aquí se usa un arreglo de secciones, en donde cada posición es una **sección** distinta, y a su vez, cada sección tiene los miembros de su respectiva estructura.

Para esto mandamos a llamar desde la función principal, a otra función llamada **agregarSeccion**, la cual recibe como parámetro a la tienda (como apuntador igualmente, para trabajar cada una por separado). Dentro de esta función vamos a preguntar al usuario cuántas secciones quiere agregarle a la Tienda, ya que recordamos que trabajamos con un arreglo, el valor que ingrese lo guardamos en una variable llamada **s**, e iniciamos un ciclo **while**, el cual nos va a ayudar a llenar cada una de estas secciones solicitadas en el arreglo, por ello el contador indica que se realiza hasta cumplir con **s**.

```
printf("%cCu%ntas secciones quiere agregar (1-5)? : ",168,160);  
scanf("%d",&s);  
while (j<s)  
{
```

Fig. 23. Fragmento de código para ejemplificar.

Dentro de este ciclo vamos a declarar una variable de tipo **Sección**, que lo que va a hacer es invocar a la función de **crearSeccion**, para que vaya creando sección por sección (con sus respectivos miembros en cada una), y así poder ir almacenando cada sección “llena” a cada posición del arreglo **secciones** que está a su vez dentro de la **Tienda** principal, para indicar cada posición del arreglo **secciones**, usamos un contador inicializado en 0 y que va a ir incrementando de 1 en 1.

```
Seccion nuevaSeccion = crearSeccion();
tienda->secciones[index] = nuevaSeccion;
agregarProducto(tienda, index);
agregarEmpleado(tienda, index);
index++;
j++;
```

Fig. 24. Fragmento de código para ejemplificar.

Entonces, la función **crearSeccion** no recibe parámetros, lo que hace es declarar una variable de este mismo tipo e ir preguntando y asignando los dos miembros “simples” de esta estructura, los cuales son “código” y “categoría” y son simples porque su tipo de dato es primitivo. Posteriormente el valor que **regresa** es esta sección “llena” hacia la función anterior, sirve para ir asignándolos en cada sección.

Sin embargo, recordemos que cada **sección** lleva en su estructura también 2 miembros más que ya son de **tipo de dato abstracto**, los cuales son 2 arreglos más, uno de tipo **Empleado**, en donde puede ingresar la cantidad de empleados que desee, y otro de tipo **Productos**, en donde puede ingresar una lista de productos correspondientes a esta sección.

Para agregar estos arreglos de **tipo de dato abstracto** a las secciones que vamos creando, se mandan a llamar aquí dentro 2 funciones más:

```
agregarProducto(tienda, index);
agregarEmpleado(tienda, index);
index++;
```

Fig. 25. Fragmento de código para ejemplificar.

La primera es **agregarProducto**, la cual de parámetros recibe a la tienda que estamos llenando y al índice de la **sección** en la que estos productos se van a encontrar. Dentro de esta función vamos a preguntar también al usuario la cantidad de productos que desea registrar y el valor lo vamos a almacenar en **p**, con la cual vamos a establecer un ciclo **while** más, en donde ahora éste se repita mientras en contador de los productos sea menor a **p**, para así ir llenando cada producto (con sus respectivos miembros) uno por uno.

```
while (i < p)
{
    printf("Producto numero %d:\n", i+1);
    Producto nuevoProducto = registrarProducto();
    t->secciones[j].productos[index] = nuevoProducto;
    index++;
    i++;
}
```

Fig. 26. Fragmento de código para ejemplificar.

Para hacer esto invocamos a otra función más, llamada **registrarProducto**, la cual nos va a ir solicitando y llenando TODOS los miembros que se solicitan para cada producto, ya que todos son de tipo de dato primitivo; una vez realizado esto, nos va a **retornar** al miembro producto ya “lleno” hacia la función **agregarProducto** con la que estamos trabajando y la cual al final nos va a permitir ir agregando uno por uno a estos productos que se están retornando y colocarlos en la sección con índice correcto y en la posición del arreglo de productos que esta siendo controlada por un contador que incrementa en 1. Y a su vez estos productos los va retornando uno por uno para seguir llenando la **sección** en la que estamos trabajando.

Una vez retornados estos productos, volvemos a la función secundaria **agregarSeccion**, para seguir llenando cada sección solicitada. Aquí entra la segunda función, la cual se llama **agregarEmpleado**, y la cual nos va a servir para llenar este arreglo de empleados que esta dentro de cada sección.

```
while (i<n)
{
    printf("Empleado numero %d:\n", i+1);
    Empleado nuevoEmpleado = crearEmpleado();
    t->secciones[seccioni].encargadoSec[index] = nuevoEmpleado;
    index++;
    i++;
}
```

Fig. 27. Fragmento de código para ejemplificar.

Al entrar a esta función secundaria (**agregarEmpleado**), la cual recibe de parámetros también a la tienda como apuntador y el índice de la posición en la que estamos trabajando, nos podemos dar cuenta que sucede algo parecido. Se va a solicitar la cantidad de empleados que se quieran registrar y se guardará en **n**, para poderla usar como condición en el ciclo **while** que usaremos para llenar uno por uno a cada empleado con sus miembros correspondientes. Creando un empleado de éste tipo de estructura y asignándole el valor que regrese la función **crearEmpleado**, la cual nos va a servir para solicitar y llenar cada característica o miembro de la estructura **Empleado**, de forma individual y sencilla (ya que son datos primitivos); Y al final cada empleado que se vaya registrando se va a ir colocando a su vez en la posición de empleado correcta, según el contador que llevamos (**index**) e igualmente en la sección correcta que se nos indica con el parámetro que recibe. Así volvemos a la función de **agregarSeccion**, para terminar de llenar la sección que nos indica nuestro ciclo **while**.

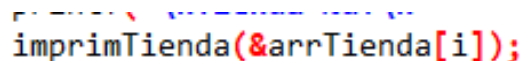
Estas dos funciones que mandamos a llamar, sirvieron para agregar los miembros de tipo estructura de cada una de nuestras secciones, y como vemos, esto lo va almacenando en nuestra **tienda** principal poco en poco, por eso es que trajimos como parámetro al apuntador **tienda**, ya que, al ser un **apuntador**, trabaja con **paso por referencia**, y no es necesario retornar un valor sino que automáticamente lo va guardando en la tienda que estamos llenando.

Esta función de **agregarSeccion**, que mandamos a llamar desde el **main**, para completar la parte de secciones de nuestra Tienda, me fue la más complicada y tardada, ya que no solo lleva muchas funciones y arreglos de estructuras dentro de ella, sino que también debemos hacer que todas conecten y queden en el mismo lugar.

Después de esta última función regresamos a la función principal **main** y terminamos con el llenado de una tienda por completo, ya dependiendo de las solicitadas por el usuario es que se va a repetir el proceso o no. Otra cosa que podemos observar es que cada índice de las tiendas que vamos llenando en el ciclo están representados por un **count**, el cual esta inicializado en 1, ya que en el arreglo de tiendas ya fue llenada la primera posición con la tienda que se inicializó, por lo tanto, las tiendas que el usuario vaya creando se van a empezar a almacenar en la posición 1 (para no sobre-escribir en la anterior) y van a ir incrementando de 1 en 1 las posiciones, por eso el "**cont++**" antes de terminar el ciclo **for**.

Esto nos va a ayudar, además, a tener un registro de la cantidad de tiendas que vamos guardando en nuestro **arreglo de tiendas**, cosa que al inicio no tomé en cuenta e igualmente se me presentó como un problema al almacenar las nuevas tiendas y también más adelante a la hora de imprimir.

El caso número 2 permite mostrar en pantalla las tiendas departamentales que se van registrando, para esto se creó en un inicio un ciclo **for** que se encargada de recorrer el arreglo de **Tiendas**, e ir imprimiendo cada una de ellas a través de una función secundaria llamada **imprimTienda**, que recibe como parámetro a la tienda que indique el contador del ciclo (como apuntador), e ir imprimiendo miembro por miembro cada elemento de ésta. Para lo cual dentro se tuvo que declara primero un ciclo **for** que fuera recorriendo e imprimiendo CADA sección que la tienda tuviera, para esto último se ocuparon otros 2 ciclos **for** que se encargan de recorrer e imprimir los dos arreglos de tipo estructura que se encontraban anidados, los cuales corresponden a los empleados y productos (cada uno de ellos con sus respectivas características).



```
imprimTienda(&arrTienda[i]);
```

Fig. 28. Fragmento de código para ejemplificar.

A partir de aquí se me presentaron 3 problemas:

El primero fue que al poner como condiciones para imprimir en los ciclos **for** el tamaño COMPLETO de los arreglos anidados (**secciones**, **productos**, **empleados**), éstos se imprimían **TODOS**, aunque no se hubiera registrado nada en ellos. Es decir, el tamaño permitido de secciones era 5, y suponiendo que el usuario solo eligiera llenar 3, los 2 restantes aun así se imprimían, pero con lo que llamamos "**basura**". Esto mismo pasaba con los **productos**, **empleados**, y con el mismo arreglo de **Tiendas**; lo cual se veía muy mal pues si NO se llenaban TODAS las tiendas o TODOS los arreglos dentro de éstas, lo que salía en pantalla eran muchísimos datos con "**basura**" que no permitían visualizar fácilmente lo solicitado.

Esto lo arreglé de la siguiente manera: Primero agregué un miembro más a la estructura de **Tienda**, que se llamara **numS**, y dentro de la estructura **Sección**, agregar 2 más llamados **numE** y **numP**. Así, al solicitar la cantidad de cada uno de ellos (secciones, productos, empleados), lo que ingresara el usuario lo guardaba en una variable específica (**s,n,p**) y cada una de estas variables se las asignaba como valor al miembro de su estructura equivalente. Por ejemplo, si al preguntar, el usuario elegía guardar 3 productos de los 10 posibles, se almacenaba como **p=10**, y después asignada el miembro **numP=p**.

```

typedef struct
{
    char nombreTie[20];
    int codigoTie;
    Gerente encargadoTie;
    Seccion secciones[5];
    int numS;
} Tienda;

typedef struct
{
    int codigoSec;
    char categoria[20];
    Empleado encargadoSec[10];
    Producto productos[10];
    int numE;
    int numP;
} Seccion;

```

Fig. 29. Fragmento de código para ejemplificar.

Esto fue con el fin de que estas variables se volvieran de cierta forma “globales” y poder utilizarlos ahora como condición al poner un límite en los ciclos **for** para imprimir cada aspecto de la tienda y así solo mostrara en pantalla los elementos de cada tienda que fueron registrados:

```

for (j = 0; j < tienda->numS; j++)
{
    for (n = 0; n < tienda->secciones->numE; n++)
    {
        for (k = 0; k < tienda->secciones->numP; k++)
        {

```

Fig. 30. Fragmento de código para ejemplificar.

Con esto solucioné el problema de impresiones para cada arreglo anidado dentro de cada **Tienda**, ya que aquí dentro, los arreglos anidados (secciones, productos, empleados) iniciaban siempre en la posición 0. Sin embargo, para el arreglo de **tiendas** no era así, pues ya teníamos inicializada la primera posición. Aquí se me presentó el segundo problema:

Al controlar la cantidad de tiendas que se van a imprimir desde los ciclos **for** en la función principal **main**, en donde se solicitaba al usuario la cantidad de tiendas que se querían llenar y posteriormente el valor guardado en una variable, se utilizaba como condición en el ciclo, lo que yo hice fue que se empezaran a almacenar las tiendas desde la posición 1, lo cual funcionaba a primera vista; Sin embargo, al registrar una tienda y si al aparecer otra vez el menú se quería registrar una nueva, ésta no se acumulaba en la posición siguiente sino que volvía a almacenar todo desde la posición 1 como se le indicó en el ciclo, por lo tanto sobre-escribía tiendas.

Esto casi no logró resolverlo, hasta que coloqué el **contador** mencionado unas páginas atrás, **count**, el cual esta inicializado en 1, y así que se logra llevar la cuenta de cada tienda que se iba registrando al entrar a esa opción e incrementarse.

Entonces, para imprimir estas tiendas se tuvo que crear el ciclo **for** que recorriera éste arreglo y parara **hasta** que se cumpliera el valor de tiendas que se solicitaba registrar + el valor del contador (que llevaba la cuenta de cuantas de cuantas tiendas llevábamos ya registradas) y - 1, lo que hacía que no se sobrepasara el tamaño del arreglo. Esto permitía imprimir consecutivamente todas las tiendas registradas.

```

for(i=0; i<t+count-1; i++){
    printf("\nTienda %d:\n-----\n", i+1);
    imprimTienda(&arrTienda[i]);
}

```

Fig. 31. Fragmento de código para ejemplificar.

Esto igualmente funcionó hasta que intenté imprimir las “tiendas registradas” desde el inicio, es decir, sin registrar ninguna antes, lo cual hacía que la condición del ciclo **for** fuera “mientras $i < 0$ ” por lo cual NO imprimía la tienda que se inicializó desde el inicio en el código, éste fue el tercer problema.

Y para resolverlo, lo que hice fue colocar una decisión simple antes del ciclo **for**, la cual como condición tenía a **count=1**, lo que significa que no se había registrado ninguna tienda aún, y lo que se colocó dentro fue la función de **imprimTienda** pero únicamente mandando como parámetro al arreglo de tiendas en la posición 0 (para la cual también fueron inicializados los miembros de **numS**, **numE** y **numP**, que controlan la función de impresión). Y ya todo lo que no entrara en la condición entraba en el ciclo **for**.

```
if(count==1){
    printf("\nTienda 1:\n-----\n");
    imprimTienda(&arrTienda[0]);
}
```

Fig. 32. Fragmento de código para ejemplificar.

Teniendo estas modificaciones ya hechas, para los **case** siguientes se realizó algo similar: Para el **case 3**, que solicitaba la impresión de las **secciones** registradas, igualmente se colocaron la decisión simple **if** y el ciclo **for** para las **Tiendas**, ya que al final, las secciones están dentro de las tiendas. La diferencia es que aquí se manda a llamar por cada tienda a otra **función**, la cual es **imprimSecc**; lo que hace esta función es que para cada tienda se imprime únicamente las secciones con cada uno de sus miembros. Por lo que dentro de la función encontrados el ciclo **for** que recorre e imprime las secciones, y dentro de este hay 2 más, los cuales recorren e imprimen a los empleados y a los productos, respectivamente (ya que ambos son arreglos de tipo estructura).

```
if(count==1){
    printf("\nTienda 1:\n-----\n");
    imprimSecc(&arrTienda[0]);
}
for(i=0; i<t+count-1; i++){
    printf("\nTienda %d:\n-----\n", i+1);
    imprimSecc(&arrTienda[i]);
}
```

Fig. 33. Fragmento de código para ejemplificar.

Para el **case 4** se solicitaba imprimir únicamente los **productos** de cada sección de cada tienda. Para ello se realizó el mismo **if** y ciclo **for** que permitían recorrer cada una de las tiendas (ya que los productos están guardados dentro de ellas), solo que aquí la función que se mandó a llamar fue **imprimProduc**, a la cual se le envió el arreglo de tiendas y en su interior tenía también el ciclo **for** de **secciones** (porque aquí se encuentra el arreglo de productos) pero no imprimía nada más de las secciones a excepción que los productos registrados (con otro ciclo **for** anidado).

```
if(count==1){
    printf("\nTienda 1:\n-----\n");
    imprimProduc(&arrTienda[0]);
}
for(i=0; i<t+count-1; i++){
    printf("\nTienda %d:\n-----\n", i+1);
    imprimProduc(&arrTienda[i]);
}
```

Fig. 34. Fragmento de código para ejemplificar.

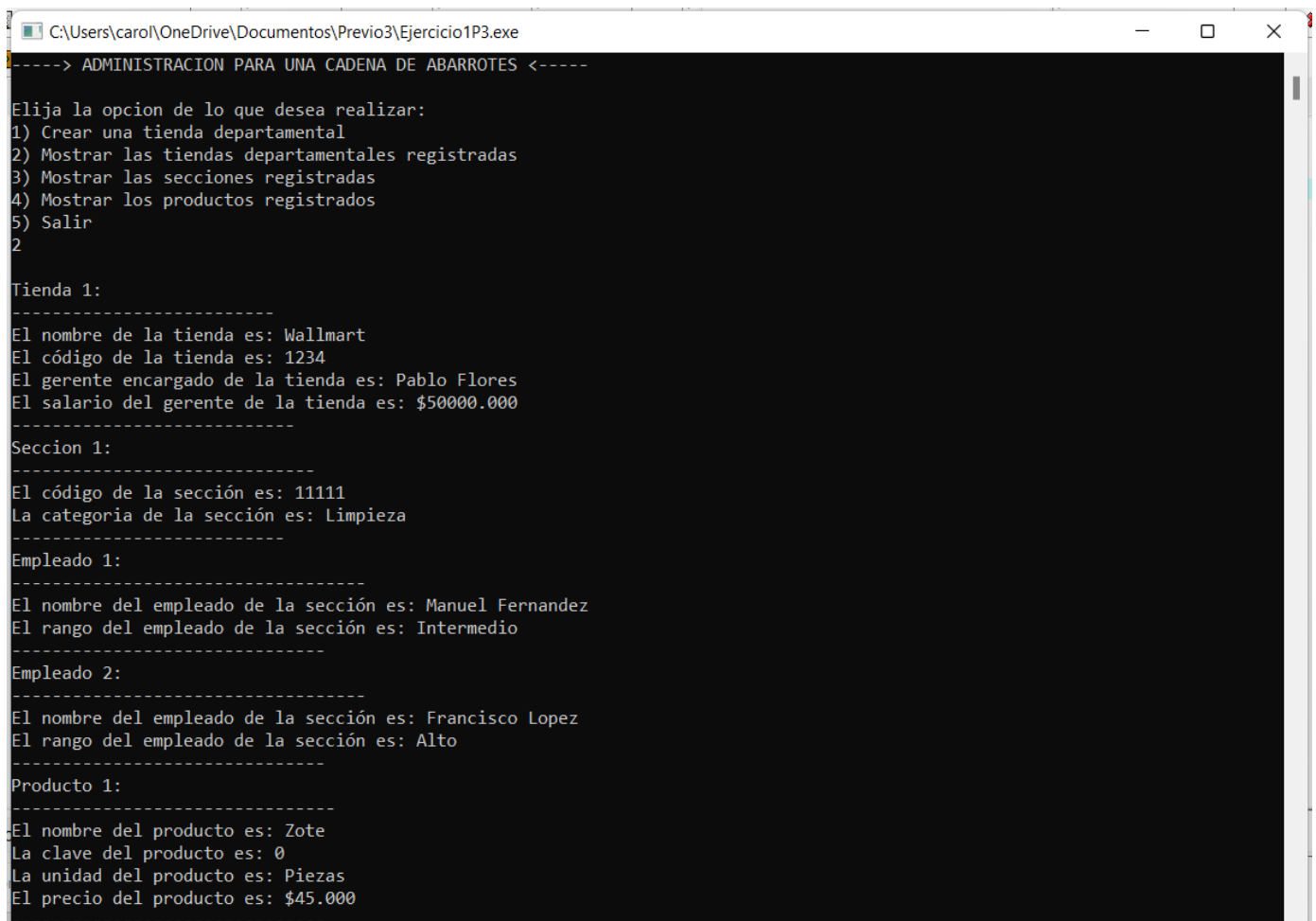
El último **case** (5) es para salir del ciclo de repetición **while**, es decir, para terminar de usar el programaba y que ya no aparezca otra vez el menú. Para esto únicamente colocamos el **return 0;** (o con cualquier otro entero) que nos lleva al fin de nuestra función **int main**. Por ello tampoco es necesario colocar un **break** al final de esta **case**.

Y por último en el menú colocamos el **default**, que nos sirve por si el usuario ingresa una opción del menú que no existe, para eso le ponemos que muestre en pantalla un "Opción no válida".

Y para terminar lo referente al programa, verifiqué cada declaración de las funciones que se usaron, con sus respectivos parámetros (si llevan) y además los acentos fueron cambiados como caracteres indicados gracias al código **ASCII**.

Ejecución 1:

Eligiendo la opción 2 desde el inicio, sin registrar tiendas aún (no se colocó la información que aparece en pantalla completa por cuestiones de espacio):



```
C:\Users\carol\OneDrive\Documentos\Previo3\Ejercicio1P3.exe
-----> ADMINISTRACION PARA UNA CADENA DE ABARROTES <-----

Elija la opcion de lo que desea realizar:
1) Crear una tienda departamental
2) Mostrar las tiendas departamentales registradas
3) Mostrar las secciones registradas
4) Mostrar los productos registrados
5) Salir
2

Tienda 1:
-----
El nombre de la tienda es: Walmart
El código de la tienda es: 1234
El gerente encargado de la tienda es: Pablo Flores
El salario del gerente de la tienda es: $50000.000
-----

Seccion 1:
-----
El código de la sección es: 11111
La categoría de la sección es: Limpieza
-----

Empleado 1:
-----
El nombre del empleado de la sección es: Manuel Fernandez
El rango del empleado de la sección es: Intermedio
-----

Empleado 2:
-----
El nombre del empleado de la sección es: Francisco Lopez
El rango del empleado de la sección es: Alto
-----

Producto 1:
-----
El nombre del producto es: Zote
La clave del producto es: 0
La unidad del producto es: Piezas
El precio del producto es: $45.000
-----
```

Fig. 35. Ejecución del ejercicio 1 de la práctica

Ejecución 2:

Al escoger después la opción de registrar tienda nos solicita uno por uno cada miembro de las estructuras anidadas:

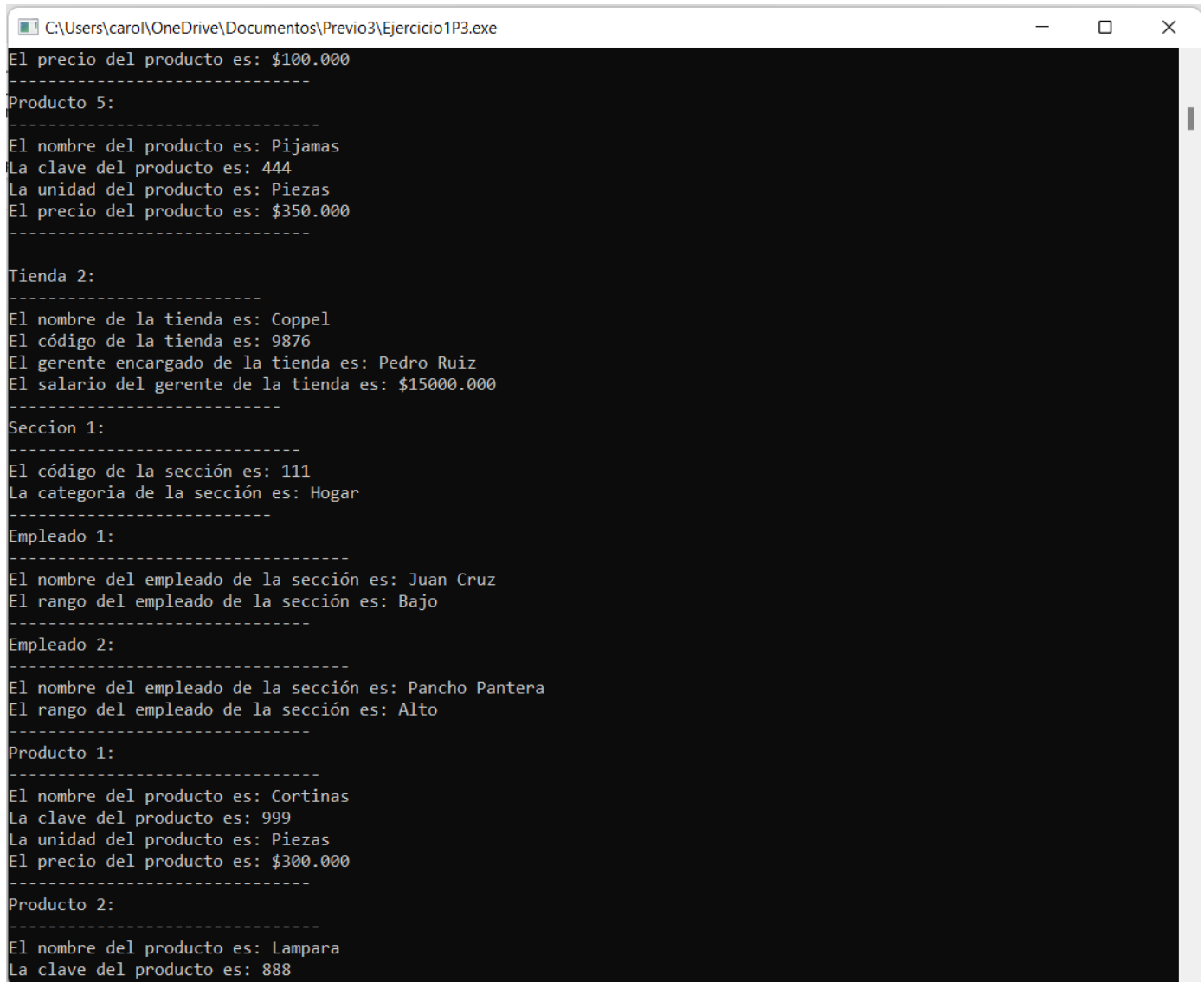
```
C:\Users\carol\OneDrive\Documentos\Previo3\Ejercicio1P3.exe
4) Mostrar los productos registrados
5) Salir
1
¿Cuántas tiendas más desea registrar (límite 4)? 1
Ingrese el nombre de la tienda: Coppel
Ingrese el código de la tienda: 9876
Ingrese el nombre del gerente: Pedro
Ingrese el apellido del gerente: Ruiz
Ingrese el salario del gerente: 15000
¿Cuántas secciones quiere agregar (1-5)? 2
Ingrese el código de la sección: 111
Ingrese la categoría de la sección: Hogar
¿Cuántos productos quieres agregar a la sección (1-10)? 2
Producto numero 1:
Ingrese la clave del producto: 999
Ingrese el nombre del producto: Cortinas
Especifique la unidad en la que esta el producto (kilos, piezas, libras,etc.): Piezas
Ingrese el precio del producto: 300
Producto numero 2:
Ingrese la clave del producto: 888
Ingrese el nombre del producto: Lampara
Especifique la unidad en la que esta el producto (kilos, piezas, libras,etc.): Pieza
Ingrese el precio del producto: 150
¿Cuántos empleados quieres agregar a la sección (1-10)? 2
Empleado numero 1:
Ingrese el nombre del empleado: Juan
Ingrese el apellido del empleado: Cruz
Ingrese el rango del empleado: Bajo
Empleado numero 2:
Ingrese el nombre del empleado: Pancho
Ingrese el apellido del empleado: Pantera
Ingrese el rango del empleado: Alto
Ingrese el código de la sección: 222
Ingrese la categoría de la sección: Escolar
¿Cuántos productos quieres agregar a la sección (1-10)? 2
Producto numero 1:
Ingrese la clave del producto: 777
Ingrese el nombre del producto: Lapiz
Especifique la unidad en la que esta el producto (kilos, piezas, libras,etc.): Pieza
Ingrese el precio del producto: 10
Producto numero 2:
Ingrese la clave del producto: 666
Ingrese el nombre del producto: Goma
```

Fig. 35.1. Ejecución 2 del ejercicio 1 de la práctica

Ejecución 3:

Al mostrar las tiendas registradas (después de registrar ya 1 más):

Vemos como nos presenta la primera que estaba ya inicializada, y posteriormente nos empieza a mostrar los elementos de la Tienda 2 (que registramos nosotros) únicamente con los miembros que solicitamos llenar (no todos los espacios posibles):



```
C:\Users\carol\OneDrive\Documentos\Previo3\Ejercicio1P3.exe
El precio del producto es: $100.000
-----
Producto 5:
-----
El nombre del producto es: Pijamas
La clave del producto es: 444
La unidad del producto es: Piezas
El precio del producto es: $350.000
-----

Tienda 2:
-----
El nombre de la tienda es: Coppel
El código de la tienda es: 9876
El gerente encargado de la tienda es: Pedro Ruiz
El salario del gerente de la tienda es: $15000.000
-----

Seccion 1:
-----
El código de la sección es: 111
La categoria de la sección es: Hogar
-----

Empleado 1:
-----
El nombre del empleado de la sección es: Juan Cruz
El rango del empleado de la sección es: Bajo
-----

Empleado 2:
-----
El nombre del empleado de la sección es: Pancho Pantera
El rango del empleado de la sección es: Alto
-----

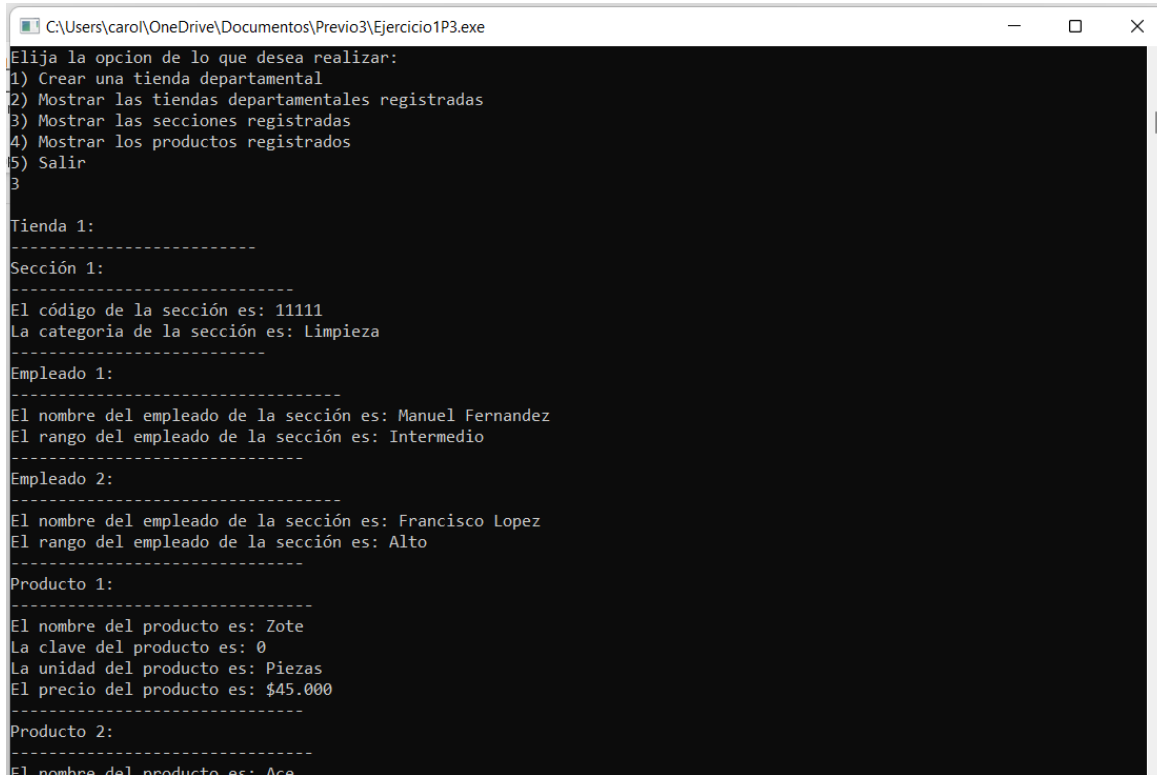
Producto 1:
-----
El nombre del producto es: Cortinas
La clave del producto es: 999
La unidad del producto es: Piezas
El precio del producto es: $300.000
-----

Producto 2:
-----
El nombre del producto es: Lampara
La clave del producto es: 888
```

Fig. 35.2. Ejecución 3 del ejercicio 1 de la práctica

Ejecución 4:

Aquí esta la ejecución al solicitar la opción 3 del menú (imprimir solo secciones registradas):



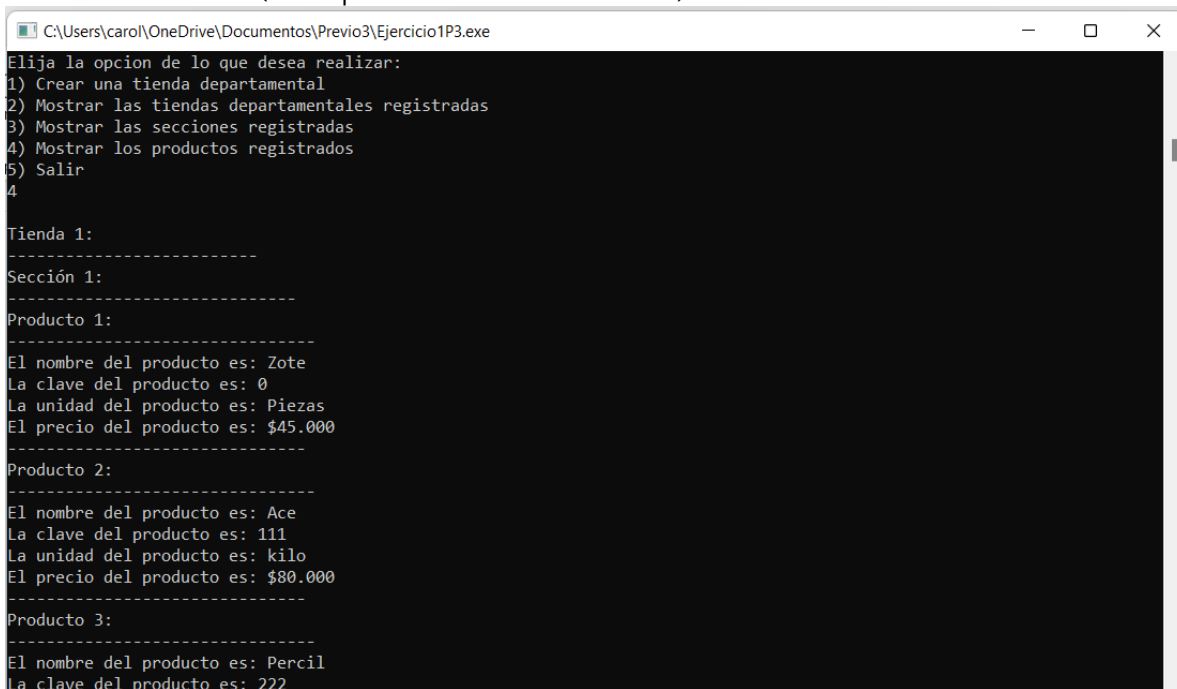
```
C:\Users\carol\OneDrive\Documentos\Previo3\Ejercicio1P3.exe
Elija la opcion de lo que desea realizar:
1) Crear una tienda departamental
2) Mostrar las tiendas departamentales registradas
3) Mostrar las secciones registradas
4) Mostrar los productos registrados
5) Salir
3

Tienda 1:
-----
Sección 1:
-----
El código de la sección es: 1111
La categoría de la sección es: Limpieza
-----
Empleado 1:
-----
El nombre del empleado de la sección es: Manuel Fernandez
El rango del empleado de la sección es: Intermedio
-----
Empleado 2:
-----
El nombre del empleado de la sección es: Francisco Lopez
El rango del empleado de la sección es: Alto
-----
Producto 1:
-----
El nombre del producto es: Zote
La clave del producto es: 0
La unidad del producto es: Piezas
El precio del producto es: $45.000
-----
Producto 2:
-----
El nombre del producto es: Ace
```

Fig. 35.3. Ejecución 4 del ejercicio 1 de la práctica

Ejecución 5:

Y la opción 4 del menú (solo productos solicitados):



```
C:\Users\carol\OneDrive\Documentos\Previo3\Ejercicio1P3.exe
Elija la opcion de lo que desea realizar:
1) Crear una tienda departamental
2) Mostrar las tiendas departamentales registradas
3) Mostrar las secciones registradas
4) Mostrar los productos registrados
5) Salir
4

Tienda 1:
-----
Sección 1:
-----
Producto 1:
-----
El nombre del producto es: Zote
La clave del producto es: 0
La unidad del producto es: Piezas
El precio del producto es: $45.000
-----
Producto 2:
-----
El nombre del producto es: Ace
La clave del producto es: 111
La unidad del producto es: kilo
El precio del producto es: $80.000
-----
Producto 3:
-----
El nombre del producto es: Percil
La clave del producto es: 222
```

Fig. 35.4. Ejecución 5 del ejercicio 1 de la práctica

CONCLUSIONES

Durante el desarrollo trabajamos principalmente con la implementación de **Tipos de datos abstractos**, los cuales en Lenguaje C se realizan mediante estructuras, las cuales ya sabemos que son conjuntos de miembros que pueden ser de tipos de datos distintos; Sin embargo, durante estos ejercicios no solo trabajamos con estructuras, sino que también implementamos todos los conocimientos que hemos estado estudiando hasta ahora, por ejemplo, los apuntadores, las funciones, etc.

Personalmente creo que ambos objetivos se lograron exitosamente, tanto el genera, ya que todo el tiempo estuvimos trabajando con estructuras de datos lineales y por lo tanto modelando tipos de datos abstractos; Así como el objetivo de clase, ya que logramos realizar un programa que ciertamente es muy parecido a algo que se puede ocupar en la vida laboral real, siento que esto nos dio una nueva perspectiva de la programación, además de nuevos enfoques, por ejemplo el buscar siempre la eficiencia en los programas, el fácil entendimiento para el usuario y su correcta funcionalidad.

La verdad fue una práctica pesada, que nos llevó (o por lo menos a mi) muchas horas de análisis, fallas, intentos de resolverlos, etc. Pudimos ser capaces de implementar todo lo que hemos visto hasta el momento, ya que era necesario para que el programa se lograra correctamente. No había practicado tanto las funciones (de todo tipo), apuntadores y arreglos como lo hice en esta práctica.

Me llevo mucho conocimiento y una mejor destreza para analizar la lógica que piden los programas como estos. Cuando hice el previo no pude acabarlo a la primera y se me hizo muy difícil, sin embargo, con el manejo de estructuras anidadas, otras dentro de éstas, arreglos de tipo estructuras, etc. cuando regresé a ver mi previo se me hizo una cosa muy sencilla, y viendo el programa de más de 400 líneas que obtuve, me siento muy satisfecha y contenta de haberlo podido lograr.

Me llevo también una "lección" para no rendirme en problemas futuros, ya que estuve a punto de enviar mi programa incompleto de una parte porque no lograba que compilara, hasta que, a prueba y error, pensar y analizar mucho, se logró exitosamente.

Aprendí las grandes ventajas que traen consigo las estructuras, puedes manipular grandes cantidades de información, simplificar muchas líneas de códigos, te permiten manipular distintos tipos de datos que están creados en una sola categoría, categoría que, además, tú puedes crear según lo que necesites. Es cierto que fue algo complejo, en especial a la hora de almacenar y conectar tantas estructuras anidadas, sin embargo, el uso de funciones, apuntadores, ciclos de repetición, etc. su manipulación es más sencilla. Además, aprendí la gran importancia de ir haciendo copias de tu código cuando queremos cambiar algo, ya que lo podemos "romper" y luego no recordamos ni lo que hicimos.

Ahora me siento más capaz y preparada para resolver problemas programables, además de que tengo más herramientas para mejorar el proceso, además de cada vez más y más práctica con ellas.