



# PROTECSA

## Proyecto final C

Montiel Juarez, Guillermo Jair<sup>1</sup> and Rincón Villa, Gretchen Itzel<sup>2</sup>

<sup>1</sup>Facultad de Contaduría y Administración, UNAM

<sup>2</sup>Facultad de Ciencias, UNAM

19 Mayo 2025

---

### Resumen :

El proyecto final del curso de C de PROTECSA integra todos los conocimientos adquiridos en un caso de uso como el sistema de compras y gestión de usuarios para una tienda, manejando estructuras de datos, técnicas de lectura de archivos txt para integrar datos al programa, manejo de memoria dinámica, y otros fundamentos útiles para cumplir con los requerimientos del proyecto.

## Índice

<b>1. Especificaciones</b>	<b>1</b>
<b>2. Código</b>	<b>1</b>
2.1. Estructuras Producto, Carrito y Usuario . . . . .	1
2.2. Limpiar pantalla y pausar programa . . . . .	2
2.3. Lista de productos . . . . .	2
2.4. Ver carrito . . . . .	2
2.5. Ver información usuario . . . . .	3
2.6. Crear producto . . . . .	3
2.7. Cargar producto . . . . .	4
2.8. Agregar producto al carrito . . . . .	4
2.9. Mostrar producto . . . . .	5
2.10. Liberar producto y usuario . . . . .	6
2.11. Crear usuario . . . . .	6
2.12. Verificar acceso . . . . .	7
2.13. Gestionar usuarios . . . . .	7
2.14. Menu usuarios . . . . .	8
<b>3. Ejecución del programa</b>	<b>10</b>
<b>4. Conclusiones por Buddy</b>	<b>14</b>
4.1. Jair Montiel . . . . .	14
4.2. Gretchen Rincón . . . . .	14

## Índice de figuras

1. Menú inicial al ejecutar el programa . . . . .	10
2. 1. Agregar usuario . . . . .	11
3. Menú interactivo dentro de la cuenta de usuario . . . . .	11
4. 3. Submenu. Ver lista de productos y agregar al carrito . . . . .	11
5. 1. Submenu. Ver el carrito de compras . . . . .	12
6. 2. Submenu. Ver la información del usuario actual . . . . .	12
7. 1. Agregando más usuarios . . . . .	12
8. 3. Seleccionando usuario dentro de la lista para continuar compras . . . . .	13
9. 2. Entrando como administrador, las credenciales por default. Se configuran directo del código . . . . .	13
10. Lista de usuarios desde el admin . . . . .	13

## 1. Especificaciones

El proyecto consiste en el desarrollo de una aplicación en lenguaje C que simule el funcionamiento básico de una tiendita.

La aplicación estará dividida en dos partes principales: la gestión de usuarios y la gestión de productos. Los usuarios podrán visualizar la lista de productos disponibles y seleccionar aquellos que deseen comprar. Cada selección se irá vinculando a un usuario específico, generando así una pequeña lista de compra que incluirá los productos elegidos y el precio total a pagar.

Una parte importante de la implementación será la recuperación de información desde archivos de texto (.txt), para cargar la lista de productos. Dado que el manejo de archivos no se cubrió durante el curso, se proporcionarán tres ejemplos en C que ilustran cómo leer datos desde un archivo de texto y almacenarlos en estructuras.

Este proyecto tiene como objetivo aplicar conceptos fundamentales de programación estructurada, manejo de archivos, uso de estructuras y memoria dinámica en C.

## 2. Código

El código principal se divide en dos archivos: **main.c** (el principal) y **productos.c**, que almacena las funciones y estructuras necesarias para la gestión de la lista de productos y el carrito del usuario actual. Definimos tres estructuras clave, Producto con dos apuntadores para tener un orden en la lista, carrito con un apuntador para ligarlo a productos, y usuario con un apuntador que lo liga a su carrito.

Para la navegación tenemos un menú de usuario para añadir usuarios, entrar como admin o entrar a tu cuenta de usuario, y un menú principal que se ejecuta dentro del usuario elegido, en el puedes ver tu carrito, tu información y la lista de productos para poder comprar.

### 2.1. Estructuras Producto, Carrito y Usuario

- **Producto:** Representa un producto en la lista de productos o en el carrito de compras. Contiene el nombre del producto, su costo, y apuntadores a los nodos anterior y siguiente en una lista doblemente ligada.

```
1 typedef struct Producto {
2     char nombre[50]; // Nombre del producto
3     float costo; // Costo del producto
4     struct Producto* siguiente; // Apuntador al siguiente nodo en la lista doblemente ligada
5     struct Producto* anterior; // Apuntador al nodo anterior en la lista doblemente ligada
6 } Producto;
```

- **Carrito:** Representa el carrito de compras de un usuario. Contiene un nombre descriptivo y un apuntador al inicio de la lista de productos en el carrito.

```
1 typedef struct Carrito {
2     char nombre[65]; // Nombre del carrito, por ejemplo: "Carrito Amiya"
3     Producto* productos; // Apuntador al inicio de la lista doblemente ligada de productos
4 } Carrito;
```

- **Usuario:** Representa a un usuario del sistema. Contiene el nombre del usuario, su número de celular, y un apuntador al carrito de compras asociado.

```

1  typedef struct Usuario {
2      char nombre[50]; // Nombre del usuario
3      char numeroCelular[15]; // Número de celular del usuario
4      Carrito* carrito; // Apuntador al carrito de compras asociado al usuario
5  } Usuario;

```

## 2.2. Limpiar pantalla y pausar programa

- **limpiarPantalla():** Limpia la pantalla de la consola. Es multiplataforma, utilizando `system("cls")` en Windows y `system("clear")` en Linux.

```

1  void limpiarPantalla() {
2      #ifdef _WIN32
3          system("cls"); // Caso para Windows
4      #else
5          system("clear"); // Caso para Linux
6      #endif
7  }

```

- **pausarPrograma():** Pausa la ejecución del programa hasta que el usuario presione la tecla Enter.

```

1  void pausarPrograma() {
2      printf("Presiona Enter para continuar...\n");
3      getchar(); // Captura el Enter
4      getchar(); // Si hubo un scanf previo, captura el Enter sobrante
5  }

```

## 2.3. Lista de productos

- **cargarProductos(const char\* archivo):** Carga una lista de productos desde un archivo de texto. Cada línea del archivo debe contener el nombre del producto y su costo. Los productos se almacenan en una lista doblemente ligada.

```

1  void verListaProductos(Usuario* usuario) {
2      limpiarPantalla();
3      mostrarProductos(cargarProductos("productos.txt"), usuario);
4  }

```

## 2.4. Ver carrito

- **verCarrito(Usuario\* usuario):** Muestra los productos en el carrito de compras del usuario actual, junto con el costo total. Recorre la lista de productos en el carrito y calcula el total.

```

1  void verCarrito(Usuario* usuario) {
2      limpiarPantalla();
3      Producto* temp = usuario->carrito ? usuario->carrito->productos : NULL; // Puntero temporal
4      float total = 0.0; // Variable para almacenar el total a pagar

```

```

5
6     if (temp == NULL) {
7         printf("\nEl carrito está vacío.\n");
8         pausarPrograma();
9         return;
10    }
11
12    printf("\n=== Carrito de Compras ===\n");
13    while (temp != NULL) {
14        printf("Producto: %s - Costo: $%.2f\n", temp->nombre, temp->costo);
15        total += temp->costo; // Sumar el costo del producto al total
16        temp = temp->siguiente; // Avanzar al siguiente producto
17    }
18    printf("\nTotal a pagar: $%.2f\n", total);
19    pausarPrograma();
20 }

```

## 2.5. Ver información usuario

- **verInformacionUsuario(Usuario\* usuario):** Muestra la información del usuario actual, incluyendo su nombre, número de celular, y el último producto agregado al carrito.

```

1     void verInformacionUsuario(Usuario* usuario) {
2         limpiarPantalla(); // Limpiar la pantalla antes de mostrar la información
3         printf("\n=== Información del Usuario ===\n");
4         printf("Nombre: %s\n", usuario->nombre);
5         printf("Número de Celular: %s\n", usuario->numeroCelular);
6         float total = 0.0;
7         // Mostrar el último producto agregado al carrito
8         Producto* temp = usuario->carrito ? usuario->carrito->productos : NULL;
9         Producto* ultimo = NULL;
10        if (temp == NULL) {
11            printf("Último Producto Agregado: Ninguno (el carrito está vacío).\n");
12        } else {
13            // Recorrer hasta el último producto
14            while (temp != NULL) {
15                total += temp->costo;
16                ultimo = temp;
17                temp = temp->siguiente;
18            }
19            printf("Último Producto Agregado: %s\n", ultimo->nombre);
20            printf("Total del carrito: $%.2f\n", total);
21        }
22
23        pausarPrograma(); // Esperar a que el usuario presione Enter
24    }

```

## 2.6. Crear producto

- **crearProducto(const char\* nombre, float costo):** Crea un nuevo nodo de producto con el nombre y costo especificados. Inicializa los apuntes **siguiente** y **anterior** como NULL.

```

1  Producto* crearProducto(const char* nombre, float costo) {
2  Producto* nuevo = (Producto*)malloc(sizeof(Producto)); // Asigna memoria dinámica para un nu
3  if (nuevo == NULL) { // Verifica si la asignación de memoria falló
4      printf("Error al asignar memoria.\n");
5      exit(1); // Termina el programa si no se pudo asignar memoria
6  }
7  strcpy(nuevo->nombre, nombre); // Copia el nombre del producto al nodo
8  nuevo->costo = costo; // Asigna el costo del producto
9  nuevo->siguiente = NULL; // Inicializa el apuntador al siguiente nodo como NULL
10 nuevo->anterior = NULL; // Inicializa el apuntador al nodo anterior como NULL
11 return nuevo; // Retorna el nodo creado
12 }

```

## 2.7. Cargar producto

- **cargarProductos(const char\* archivo):** Lee productos desde un archivo de texto y los almacena en una lista doblemente ligada. Cada producto se representa como un nodo en la lista.

```

1  Producto* cargarProductos(const char* archivo) {
2  FILE* f = fopen(archivo, "r"); // Abre el archivo en modo lectura
3  if (f == NULL) { // Verifica si el archivo no pudo abrirse
4      printf("Error al abrir el archivo.\n");
5      return NULL;
6  }
7
8  Producto* inicio = NULL; // Apuntador al inicio de la lista de productos
9  Producto* actual = NULL; // Apuntador al nodo actual mientras se construye la lista
10 char nombre[50];
11 float costo;
12
13 // Lee cada línea del archivo hasta llegar al final (EOF)
14 while (fscanf(f, "%s %f", nombre, &costo) != EOF) {
15     Producto* nuevo = crearProducto(nombre, costo); // Crea un nuevo nodo de producto
16
17     if (inicio == NULL) {
18         // Si es el primer nodo, inicializa la lista
19         inicio = nuevo;
20     } else {
21         // Enlaza el nuevo nodo al final de la lista
22         actual->siguiente = nuevo; // El nodo actual apunta al nuevo nodo como siguiente
23         nuevo->anterior = actual; // El nuevo nodo apunta al nodo actual como anterior
24     }
25     actual = nuevo; // Actualiza el nodo actual al nuevo nodo
26 }
27
28 fclose(f); // Cierra el archivo después de leerlo
29 return inicio; // Retorna el inicio de la lista de productos
30 }

```

## 2.8. Agregar producto al carrito

- **agregarProductoAlCarrito(Carrito\* carrito, const char\* nombre, float costo):** Agrega un nuevo producto al final de la lista de productos en el carrito. Si el carrito está vacío, el

producto se convierte en el primer nodo.

```

1 void agregarProductoAlCarrito(Carrito* carrito, const char* nombre, float costo) {
2     Producto* nuevo = crearProducto(nombre, costo); // Crea un nuevo nodo de producto
3     if (carrito->productos == NULL) {
4         // Si el carrito está vacío, el nuevo producto es el primero
5         carrito->productos = nuevo;
6     } else {
7         // Si ya hay productos, recorre la lista hasta el final
8         Producto* temp = carrito->productos;
9         while (temp->siguiente != NULL) temp = temp->siguiente; // Avanza al último nodo
10        temp->siguiente = nuevo; // Enlaza el nuevo nodo al final de la lista
11        nuevo->anterior = temp; // El nuevo nodo apunta al último nodo como anterior
12    }
13 }

```

## 2.9. Mostrar producto

- **mostrarProductos(Producto\* inicio, Usuario\* usuario):** Permite al usuario navegar por la lista de productos disponibles. Ofrece opciones para avanzar al siguiente producto, retroceder al anterior, agregar el producto actual al carrito, o salir.

```

1 void mostrarProductos(Producto* inicio, Usuario* usuario) {
2     Producto* actual = inicio; // Apuntador al nodo actual
3     char tecla;
4
5     if (actual == NULL) { // Verifica si la lista está vacía
6         printf("No hay productos disponibles.\n");
7         return;
8     }
9
10    while (1) {
11        limpiarPantalla();
12        printf("=== Producto ===\n");
13        printf("Nombre: %s\n", actual->nombre);
14        printf("Costo: $%.2f\n", actual->costo);
15        printf("\nPresiona 'S' para siguiente, 'A' para anterior, 'C' para agregar al carrito, 'Q' para salir.\n");
16        scanf(" %c", &tecla); // Lee la tecla ingresada por el usuario
17
18        if (tecla == 'S' || tecla == 's') {
19            // Avanza al siguiente producto si existe
20            if (actual->siguiente != NULL) actual = actual->siguiente;
21            else { printf("Ya estás en el último producto.\n"); pausarPrograma(); }
22        } else if (tecla == 'A' || tecla == 'a') {
23            // Retrocede al producto anterior si existe
24            if (actual->anterior != NULL) actual = actual->anterior;
25            else { printf("Ya estás en el primer producto.\n"); pausarPrograma(); }
26        } else if (tecla == 'C' || tecla == 'c') {
27            // Agrega el producto actual al carrito
28            agregarProductoAlCarrito(usuario->carrito, actual->nombre, actual->costo);
29            printf("Producto '%s' agregado al carrito.\n", actual->nombre);
30            pausarPrograma();
31        } else if (tecla == 'Q' || tecla == 'q') {
32            // Sale del bucle
33            break;
34        }
35    }
36 }

```

```

34         } else {
35             printf("Tecla no válida.\n");
36             pausarPrograma();
37         }
38     }
39 }

```

## 2.10. Liberar producto y usuario

- **liberarProductos(Producto\* inicio):** Libera la memoria ocupada por la lista de productos. Recorre la lista y libera cada nodo individualmente.

```

1 void liberarProductos(Producto* inicio) {
2     Producto* temp; // Puntero temporal para liberar la memoria
3     while (inicio != NULL) { // Mientras haya productos en la lista
4         temp = inicio; // Guarda el nodo actual en temp
5         inicio = inicio->siguiente; // Avanza al siguiente nodo
6         free(temp); // Libera el nodo actual
7     }
8 }

```

- **liberarUsuarios(Usuario\*\* usuarios, int cantidadUsuarios):** Libera la memoria ocupada por el arreglo de usuarios

```

1 void liberarUsuarios(Usuario** usuarios, int cantidadUsuarios) {
2     for (int i = 0; i < cantidadUsuarios; i++) {
3         liberarProductos(usuarios[i]->carrito->productos);
4         free(usuarios[i]->carrito);
5         free(usuarios[i]);
6     }
7     free(usuarios);
8 }

```

## 2.11. Crear usuario

- **crearUsuario(const char\* nombreUsuario, const char\* numeroCelular):** Crea un nuevo usuario con el nombre y número de celular especificados. También crea un carrito de compras vacío asociado al usuario.

```

1 Usuario* crearUsuario(const char* nombreUsuario, const char* numeroCelular) {
2     Usuario* nuevoUsuario = (Usuario*)malloc(sizeof(Usuario)); // Asigna memoria para un nuevo u
3     if (!nuevoUsuario) return NULL; // Retorna NULL si no se pudo asignar memoria
4     strncpy(nuevoUsuario->nombre, nombreUsuario, sizeof(nuevoUsuario->nombre)); // Copia el nomb
5     strncpy(nuevoUsuario->numeroCelular, numeroCelular, sizeof(nuevoUsuario->numeroCelular)); //
6
7     Carrito* nuevoCarrito = (Carrito*)malloc(sizeof(Carrito)); // Asigna memoria para un nuevo c
8     if (!nuevoCarrito) { // Verifica si la asignación de memoria falló
9         free(nuevoUsuario); // Libera la memoria del usuario si falla
10        return NULL;
11    }
12    snprintf(nuevoCarrito->nombre, sizeof(nuevoCarrito->nombre), "Carrito %s", nombreUsuario); //

```



```

13     nuevoCarrito->productos = NULL; // Inicializa la lista de productos como vacía
14     nuevoUsuario->carrito = nuevoCarrito; // Asocia el carrito al usuario
15
16     return nuevoUsuario; // Retorna el usuario creado
17 }

```

## 2.12. Verificar acceso

```

1     int verificar_acceso() {
2         char usuario[50]; // Almacena el nombre de usuario ingresado
3         char contrasena[50]; // Almacena la contraseña ingresada
4         limpiarPantalla(); // Limpia la pantalla antes de solicitar las credenciales
5         printf("=== Acceso de Administrador ===\n");
6         printf("Usuario: ");
7         scanf("%49s", usuario); // Lee el nombre de usuario
8         printf("Contraseña: ");
9         scanf("%49s", contrasena); // Lee la contraseña
10
11         // Compara las credenciales ingresadas con las definidas
12         if (strcmp(usuario, ADMIN_USER) == 0 && strcmp(contrasena, ADMIN_PASS) == 0) {
13             printf("Acceso concedido.\n\n");
14             return 1; // Retorna 1 si las credenciales son correctas
15         } else {
16             printf("Acceso denegado.\n\n");
17             return 0; // Retorna 0 si las credenciales son incorrectas
18         }
19     }

```

## 2.13. Gestionar usuarios

```

1     void gestionarUsuarios(Usuario*** usuariosPtr, int* cantidadUsuarios) {
2         Usuario** usuarios = *usuariosPtr; // Obtiene la lista de usuarios
3         int opcionGestion;
4         limpiarPantalla(); // Limpia la pantalla
5         printf("\n=== GESTIÓN DE USUARIOS ===\n");
6         printf("\nUsuarios registrados:\n");
7         if (*cantidadUsuarios == 0) {
8             printf("No hay usuarios registrados.\n");
9         } else {
10             // Muestra la lista de usuarios registrados
11             for (int i = 0; i < *cantidadUsuarios; i++) {
12                 printf("%d. %s, Celular: %s\n", i + 1, usuarios[i]->nombre, usuarios[i]->numeroCelular);
13             }
14
15             // Pregunta si se desea eliminar un usuario
16             printf("\n¿Desea eliminar un usuario? (1: Sí, 0: No): ");
17             scanf("%d", &opcionGestion);
18             if (opcionGestion == 1) {
19                 int seleccion;
20                 printf("Selecciona el número del usuario a eliminar (1-%d): ", *cantidadUsuarios);
21                 scanf("%d", &seleccion);
22             }

```

```

23 // Verifica que la selección sea válida
24 if (seleccion >= 1 && seleccion <= *cantidadUsuarios) {
25     int idx = seleccion - 1; // Índice del usuario a eliminar
26
27     // Libera la memoria asociada al usuario
28     liberarProductos(usuarios[idx]->carrito->productos);
29     free(usuarios[idx]->carrito);
30     free(usuarios[idx]);
31
32     // Desplaza los usuarios restantes para llenar el hueco
33     for (int j = idx; j < *cantidadUsuarios - 1; j++) {
34         usuarios[j] = usuarios[j + 1];
35     }
36     (*cantidadUsuarios)--; // Reduce el contador de usuarios
37     printf("Usuario eliminado correctamente.\n");
38 } else {
39     printf("Selección no válida.\n");
40 }
41 }
42 }
43 pausarPrograma(); // Pausa el programa para que el usuario pueda leer el mensaje
44 }

```

## 2.14. Menu usuarios

```

1
2 void menuUsuarios(Usuario*** usuariosPtr, int* cantidadUsuarios, Usuario** usuarioActual) {
3     int capacidad = (*cantidadUsuarios > 0) ? *cantidadUsuarios : 2; // Define la capacidad inicial d
4     Usuario** usuarios = *usuariosPtr; // Obtiene la lista de usuarios
5     char opcionStr[10]; // Almacena la opción ingresada como cadena
6     int opcion; // Almacena la opción convertida a entero
7
8     // Si la lista de usuarios no está inicializada, la inicializa
9     if (usuarios == NULL) {
10         usuarios = (Usuario**)malloc(capacidad * sizeof(Usuario*));
11         *cantidadUsuarios = 0;
12     }
13
14     while (1) {
15         limpiarPantalla(); // Limpia la pantalla
16         printf("\n=== MENÚ DE USUARIOS ===\n");
17         printf("1. Agregar usuario\n");
18         printf("2. Gestionar usuarios\n");
19         printf("3. Seleccionar usuario\n");
20         printf("4. Salir del programa\n");
21         printf("Selecciona una opción: ");
22         scanf("%s", opcionStr); // Lee la opción como cadena
23         opcion = atoi(opcionStr); // Convierte la opción a entero
24
25         switch (opcion) {
26             case 1: {
27                 limpiarPantalla();
28                 printf("\n=== AGREGAR USUARIO ===\n");
29                 char nombre[50], celular[15];
30                 printf("Introduce el nombre del usuario: ");

```

```

31     scanf("%s", nombre); // Lee el nombre del usuario
32     printf("Introduce el número de celular: ");
33     scanf("%s", celular); // Lee el número de celular
34
35     // Si se alcanza la capacidad máxima, realoca la memoria
36     if (*cantidadUsuarios >= capacidad) {
37         capacidad *= 2; // Duplica la capacidad
38         usuarios = (Usuario**)realloc(usuarios, capacidad * sizeof(Usuario*));
39         if (usuarios == NULL) {
40             printf("Error al asignar memoria.\n");
41             exit(1); // Termina el programa si falla la asignación
42         }
43     }
44
45     // Crea un nuevo usuario y lo agrega a la lista
46     usuarios[*cantidadUsuarios] = crearUsuario(nombre, celular);
47     if (usuarios[*cantidadUsuarios] == NULL) {
48         printf("Error al crear usuario.\n");
49         pausarPrograma();
50         break;
51     }
52     *usuarioActual = usuarios[*cantidadUsuarios]; // Actualiza el usuario actual
53     printf("Usuario '%s' agregado y seleccionado.\n", nombre);
54     (*cantidadUsuarios)++; // Incrementa el contador de usuarios
55     pausarPrograma();
56     goto salir_usuarios; // Salta al menú de la tienda
57 }
58 case 2: {
59     // Verifica el acceso del administrador antes de gestionar usuarios
60     if (!verificar_acceso()) {
61         pausarPrograma();
62         break;
63     }
64     gestionarUsuarios(&usuarios, cantidadUsuarios); // Llama a la función de gestión de u
65     break;
66 }
67 case 3: {
68     if (*cantidadUsuarios == 0) {
69         printf("No hay usuarios registrados. Agrega uno primero.\n");
70         pausarPrograma();
71     } else {
72         // Muestra la lista de usuarios registrados
73         printf("\nUsuarios registrados:\n");
74         for (int i = 0; i < *cantidadUsuarios; i++) {
75             printf("%d. %s\n", i + 1, usuarios[i]->nombre);
76         }
77         printf("Selecciona un usuario (1-%d): ", *cantidadUsuarios);
78         int seleccion;
79         scanf("%d", &seleccion);
80
81         // Verifica que la selección sea válida
82         if (seleccion >= 1 && seleccion <= *cantidadUsuarios) {
83             *usuarioActual = usuarios[seleccion - 1]; // Actualiza el usuario actual
84             printf("Usuario '%s' seleccionado.\n", (*usuarioActual)->nombre);
85             pausarPrograma();
86             goto salir_usuarios; // Salta al menú de la tienda
87         } else {
88             printf("Selección no válida.\n");

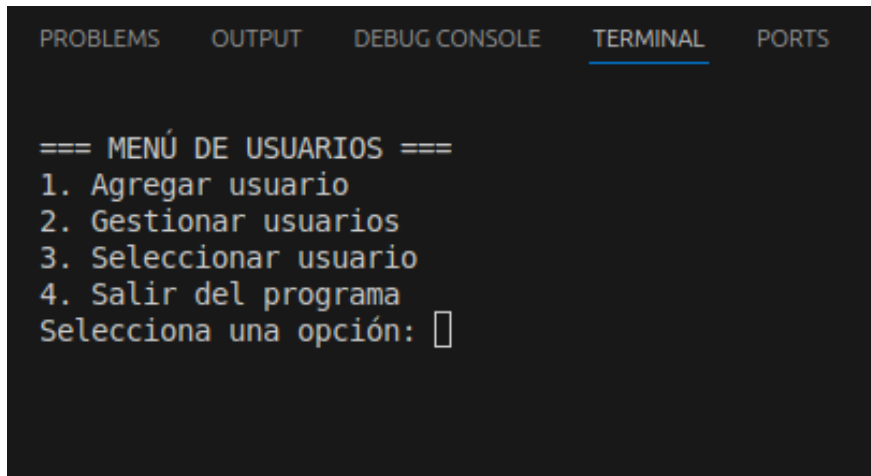
```

```

89         pausarPrograma();
90     }
91 }
92 break;
93 }
94 case 4: {
95     printf("Saliendo del programa...\n");
96     exit(0); // Termina el programa
97 }
98 default: {
99     printf("Opción no válida.\n");
100    pausarPrograma();
101 }
102 }
103 }
104 salir_usuarios:
105     *usuariosPtr = usuarios; // Actualiza el puntero en caso de que se haya realocado la memoria
106 }
107

```

### 3. Ejecución del programa



```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

=== MENÚ DE USUARIOS ===
1. Agregar usuario
2. Gestionar usuarios
3. Seleccionar usuario
4. Salir del programa
Selecciona una opción: █

```

Figura 1: Menú inicial al ejecutar el programa

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

=== AGREGAR USUARIO ===
Introduce el nombre del usuario: Jair
Introduce el número de celular: 551234568
Usuario 'Jair' agregado y seleccionado.
Presiona Enter para continuar...
█

```

Figura 2: 1. Agregar usuario

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

=== Tiendita Interactiva ===
Usuario actual: Jair
1. Ver mi carrito de compras
2. Ver mi información de usuario
3. Ver la lista de productos
4. Salir
Elige una opción: █

```

Figura 3: Menú interactivo dentro de la cuenta de usuario

```

=== Producto ===
Nombre: Huevo
Costo: $0.50

Presiona 'S' para siguiente, 'A' para anterior, 'C' para agregar al carrito, 'Q' para salir.
c
Producto 'Huevo' agregado al carrito.
Presiona Enter para continuar...
█

```

Figura 4: 3. Submenu. Ver lista de productos y agregar al carrito

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

=== Carrito de Compras ===
Producto: Huevo - Costo: $0.50
Producto: Huevo - Costo: $0.50
Producto: Huevo - Costo: $0.50
Producto: Cerveza - Costo: $20.00
Producto: Barritas - Costo: $12.00
Producto: Agua - Costo: $11.00

Total a pagar: $44.50
Presiona Enter para continuar...

```

Figura 5: 1. Submenu. Ver el carrito de compras

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

=== Información del Usuario ===
Nombre: Jair
Número de Celular: 551234568
Último Producto Agregado: Agua
Total del carrito: $44.50
Presiona Enter para continuar...

```

Figura 6: 2. Submenu. Ver la información del usuario actual

Salimos del menú interactivo y agregamos más usuarios.

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

=== AGREGAR USUARIO ===
Introduce el nombre del usuario: Gret
Introduce el número de celular: 5538535633
Usuario 'Gret' agregado y seleccionado.
Presiona Enter para continuar...

```

Figura 7: 1. Agregando más usuarios

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

=== MENÚ DE USUARIOS ===
1. Agregar usuario
2. Gestionar usuarios
3. Seleccionar usuario
4. Salir del programa
Selecciona una opción: 3

Usuarios registrados:
1. Jair
2. Gret
3. Estrella
Selecciona un usuario (1-3): 2
Usuario 'Gret' seleccionado.
Presiona Enter para continuar...

```

Figura 8: 3. Seleccionando usuario dentro de la lista para continuar compras

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

=== Acceso de Administrador ===
Usuario: admin
Contraseña: 1234

```

Figura 9: 2. Entrando como administrador, las credenciales por default. Se configuran directo del código

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

=== Acceso de Administrador ===
Usuario: admin
Contraseña: 1234

```

Figura 10: Lista de usuarios desde el admin

## 4. Conclusiones por Buddy

### 4.1. Jair Montiel

En la programación de un sistema, se necesitan tomar en cuenta muchos factores como los requerimientos a cumplir, la interacción con el usuario, reglas del negocio, y otras necesidades a resolver. El lenguaje C ofrece todo lo necesario para poder ejecutar alguna de estas soluciones, y si bien puede resultar más complejo que otros lenguajes, permite entender en profundidad el funcionamiento de funciones, estructuras, manejo de memoria y otros recursos que se cuentan para tener un control granular en nuestro algoritmo. Cada declaración, estructura, función, debe definirse con gran detalle, ya que se expresará en cada línea de código. El trabajo en equipo facilita mucho de este proceso, donde se pueden aportar ideas, dividir las tareas o funciones, e incluso hacer diferentes pruebas de calidad del programa final.

### 4.2. Gretchen Rincón

En este proyecto se trabajó mucho la modularidad, se crearon funciones para todo, desde limpiar la pantalla hasta un menú, de este modo se logró un main de 50 líneas mucho más ordenado. Claro que hay más cosas que aún podemos simplificar y hacer más eficientes, pues con las pruebas observamos que nuestro programa consumía mucha memoria. Se procuró eficientarlo usando alloc para modificar la memoria de nuestro arreglo de usuarios conforme a las necesidades de cada ejecución, también nos aseguramos de limpiar la memoria y terminar los procesos. Este proyecto también nos permitió reforzar todos los conocimientos básicos, de punteros, memoria dinámica, estructuras y funciones, para poder hacer proyectos más elaborados en un futuro.