

PROGRAMACIÓN

Unidad 7 – Parte 3. Funciones, punteros y estructuras.
Aplicaciones.

Revisamos la tarea pendiente



1. Explica cómo puedes pasar una estructura como argumento a una función en C.
Se puede pasar una estructura completa como argumento por valor o por partes, es decir, pasar cada campo o miembro de la estructura ya sea por valor o por referencia. Por ejemplo: void cargar(struct alumno estudiante);
2. ¿Puedes devolver una estructura desde una función en C? ¿Cómo?
Sí, se puede devolver una estructura desde una función. Se debe declarar la función con el tipo de la estructura como su tipo de retorno y se usa return para devolver la estructura.
3. Proporciona un ejemplo de cómo usar una estructura en una función.

```
Alumno cargarDatos()
{
    Alumno estudiante;
    printf("Ingrese el apellido:");
    gets(estudiante.apellido);

    printf("Ingrese la edad:");
    scanf("%d", & estudiante.edad);

    return(alum);
}
```

Revisamos la tarea pendiente



4. ¿Cómo se declara un arreglo de estructuras en C?

Se declara utilizando la definición de la estructura seguida por el nombre del arreglo y el tamaño entre corchetes.

5. Explica cómo se inicializan los elementos de un arreglo de estructuras.

Se pueden inicializar proporcionando valores entre llaves para cada elemento del arreglo durante su declaración o mediante asignación después de la declaración.

6. ¿Cuál es la diferencia entre acceder a un miembro de una estructura en una estructura individual y en un arreglo de estructuras?

En un arreglo de estructuras, se utiliza el índice del arreglo para acceder a una estructura específica, y luego se accede al miembro de esa estructura a través del operador punto.

7. ¿Cómo se pasa un arreglo de estructuras a una función en C?

Se puede pasar el arreglo completo como un arreglo de cualquier tipo, siempre se debe recordar que al pasar un arreglo pasamos la dirección de memoria del primer elemento del arreglo.

Punteros a Estructuras

- Los punteros a estructuras son como los punteros a variables ordinarias.

Si partimos de la siguiente estructura:

```
struct Punto {  
    int x;  
    int y;  
};
```

Entonces la declaración del puntero a la estructura será:

```
struct Punto *punt;
```

Ejemplo

```
#include <stdio.h>

struct Punto { int x, y;};

int main()
{
    struct Punto coordenadas, *Ppunto;
    Ppunto = &coordenadas;

    (*Ppunto).x = 3;
    (*Ppunto).y = 5;

    printf("Las coordenadas del punto son: %d, %d", (*Ppunto).x, (*Ppunto).y);
    return 0;
}
```

Importante!

En la operación : $(*Ppunt).x = 3;$

Los $()$ son necesarios porque la precedencia del operador punto es mayor que la precedencia del operador de indirección *

Operador Flecha

- El operador flecha es usado cuando trabajamos con estructuras y punteros. Nos permitirá mas claridad en la notación. Si p es un puntero a la estructura, entonces la notación es: $p \rightarrow \text{miembroDeLaEstructura}$

```
#include <stdio.h>

struct Punto { int x, y;};

int main()
{
    struct Punto coordenadas, *Ppunto;

    Ppunto = &coordenadas;

    Ppunto->x = 3;
    Ppunto->y = 5;

    printf("Las coordenadas del punto son: %d, %d", Ppunto->x, Ppunto->y);
    return 0;
}
```

Arreglo de estructuras y punteros

Cuando tenemos un puntero a un arreglo de estructura, para acceder a cada campo de la estructura lo hacemos mediante el operador flecha.

```
int main()
{
    struct Alumno estudiante[20], *palum;

    float prom;

    palum = estudiante;

    for (int i = 0; i < 20; i++)
    {
        printf("Datos del alumno N°: %d\n", i);

        puts("Ingrese el apellido del alumno : ");
        gets(palum->ape);

        puts("Ingrese el nombre del alumno:");
        gets(palum->nom);

        puts("Ingrese la edad del alum:");
        scanf("%d", &palum->edad);

        puts("Ingrese 1ra nota:");
        scanf("%f", &palum->nota1);

        puts("Ingrese 2da nota:");
        scanf("%f", &palum->nota2);

        puts("Ingrese 3ra nota:");
        scanf("%f", &palum->nota3);

        prom = promedio(palum->nota1, palum->nota2, palum->nota3);

        printf("El promedio del alumno es: %f\n", prom);

        palum++;
    }
}
```


Retomemos nuestro Ejercicio:

Ahora jugaremos que somos los informáticos de Mercado Libre.

1. Armaremos una estructura general, que le permitirá a los vendedores cargar sus productos. Para sintetizar la tarea solo nos limitaremos a los siguientes datos:
 - Marca
 - Categoría
 - Stock
 - Productos disponibles (modelo, precio)
2. Realizar la carga de los productos según el stock.
3. Mostrar los productos cargados.



Ejercicio – versión 1:

Comenzaremos con el punto 1 del ejercicio: armando las estructuras necesarias para poder desarrollar lo solicitado por el ejercicio.

En este caso vamos a **suponer** que solo serán cargados 3 productos por marca

```
struct{
    char modelo[MAX];
    float precio;
} typedef producto;

struct{
    char marca[MAX];
    char categoria[MAX];
    int stock;
    producto disponibles[3];
} typedef venta;
```

Nota: MAX está definida como una constante en la cabecera del código.

Ejercicio – versión 1:

Continuamos con el punto 2 del ejercicio: se define una variable estructurada **prod1** y se carga en esta primera instancia sin tener en cuenta el stock, ya que estamos suponiendo que solo se cargarán 3 productos.

```
int main()
{
    venta prod1;

    printf("Ingrese la marca del producto:");
    gets(prod1.marca);
    printf("Ingrese la categoria del producto:");
    gets(prod1.categoria);
    printf("Ingrese el stock:");
    scanf("%d", &prod1.stock);
    fflush(stdin);
    for (int i = 0; i < 3; i++)
    {
        printf("Ingrese el modelo del producto:");
        gets(prod1.disponibles[i].modelo);
        fflush(stdin);
        printf("Ingrese el precio del producto:");
        scanf("%f", &prod1.disponibles[i].precio);
        fflush(stdin);
    }

    return 0;
}
```

Ejercicio – versión 1:

En el punto 3 del ejercicio: codificaremos una función para mostrar los datos anteriormente cargados. Pasando por parámetro la variable estructurada.

```
void mostrarProductos(venta producto)
{
    puts("*****MERCADERIA*****");
    printf("----MARCA => %s \n", producto.marca);
    printf("Categoria: %s\n", producto.categoria);
    printf("----STOCK => %d \n", producto.stock);

    printf("----PRODUCTOS DISPONIBLES----- \n");
    for (int j = 0; j < producto.stock; j++)
    {
        printf("Modelo: %s\n", producto.disponibles[j].modelo);
        printf("Precio: %.2f \n", producto.disponibles[j].precio);
    }
    puts("-----");
}
```

```
void mostrarProductos(venta producto);
int main()
{
    venta articulo;

    printf("Ingrese la marca del producto");
    gets(articulo.marca);
    printf("Ingrese la categoria del producto");
    gets(articulo.categoria);
    printf("Ingrese el stock:");
    scanf("%d", &articulo.stock);

    fflush(stdin);
    for (int i = 0; i < 3; i++)
    {
        printf("Ingrese el modelo del producto: ");
        gets(articulo.disponibles[i].modelo);
        printf("Ingrese el precio del producto: ");
        scanf("%f", &articulo.disponibles[i].precio);
    }
    mostrarProductos(articulo);

    return 0;
}
```

Ejercicio – versión 2:

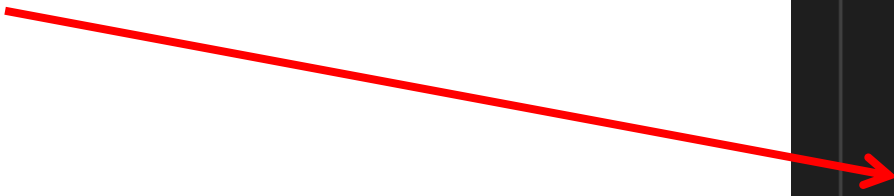
Si bien hemos resuelto el ejercicio completo, no cumplimos con la solicitud de que se cargue la cantidad de productos según el stock.

Para esto realizaremos una modificación en la definición de nuestra estructura: es vez de definir un arreglo de estructura (producto disponibles[3]), definiremos un puntero del tipo producto.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100

struct{
    char modelo[MAX];
    float precio;
} typedef producto;

struct{
    char marca[MAX];
    char categoria[MAX];
    int stock;
    producto *disponibles;
} typedef venta;
```



Ejercicio – versión 2:

El puntero definido dentro de la estructura producto, lo “utilizaremos” para realizar una reserva dinámica de memoria, esto nos permitirá cargar la cantidad de productos según el stock cargado.

Nota: la función `mostrarProductos()` no cambia respecto a la versión 1 del ejercicio.

```
void mostrarProductos(venta producto);
int main()
{
    venta articulo;

    printf("Ingrese la marca del producto");
    gets(articulo.marca);
    printf("Ingrese la categoria del producto");
    gets(articulo.categoria);
    printf("Ingrese el stock:");
    scanf("%d", &articulo.stock);
    articulo.disponibles =(producto *) malloc(articulo.stock * (sizeof(producto)));

    fflush(stdin);
    for(int i = 0; i < articulo.stock; i++)
    {
        printf("Ingrese el modelo del producto: ");
        gets(articulo.disponibles[i].modelo);
        printf("Ingrese el precio del producto: ");
        scanf("%f", &articulo.disponibles[i].precio);
    }

    mostrarProductos(articulo);
    free(articulo.disponibles); //LIBERO LA MEMORIA SOLICITADA

    return 0;
}
```

Ejercicio – versión 3:

Ahora vamos a acercar nuestra propuesta de solución del ejercicio de mercado libre, más a la realidad. Lo haremos pensando que un vendedor puede cargar más de un producto de diferente marca o incluso categoría. Para esto, no será necesario cambiar las estructuras declararemos un arreglo de estructura del tipo venta

```
void mostrarProductos(venta producto[2]);
int main()
{
    venta articulo[2];
    for (int j = 0; j < 2; j++)
    {
        printf("Ingrese la marca del producto");
        gets(articulo[j].marca);
        printf("Ingrese la categoria del producto");
        gets(articulo[j].categoria);
        printf("Ingrese el stock:");
        scanf("%d", &articulo[j].stock);
        articulo[j].disponibles =(producto *) malloc(articulo[j].stock * (sizeof(producto)));

        fflush(stdin);
        for (int i = 0; i < articulo[j].stock; i++)
        {
            printf("Ingrese el modelo del producto: ");
            gets(articulo[j].disponibles[i].modelo);
            printf("Ingrese el precio del producto: ");
            scanf("%f", &articulo[j].disponibles[i].precio);
            fflush(stdin);
        }
    }

    mostrarProductos(articulo);
    return 0;
}
```

Ejercicio – versión 3:

La función mostrarProductos
cambia ya que debemos
mostrar un arreglo de
productos.
Además en esta función se
libera la memoria del
puntero disponibles.

```
void mostrarProductos(venta producto[2])
{
    puts("*****MERCADERIA*****");
    for (int j = 0; j < 2; j++)
    {
        printf("MARCA: %s\n", producto[j].marca);
        printf("Categoría: %s\n", producto[j].categoria);
        printf("STOCK: %d\n", producto[j].stock);

        printf("-----PRODUCTOS DISPONIBLES-----");
        for (int i = 0; i < producto[j].stock; i++)
        {
            printf("Modelo: %s\n", producto[j].disponibles[i].modelo);
            printf("Precio: %.2f\n", producto[j].disponibles[i].precio);
        }
        puts("*****");
        free(producto[j].disponibles);
    }
}
```


Ejercicio – versión 4:

Agregamos la opción para que el usuario cargue la cantidad de artículos que cargará, esto implica crear un puntero y realizar, también, una asignación dinámica de memoria.

```
int main() {
    int cantArt;
    printf("Ingrese la cantidad de articulos ha cargar: ");
    scanf("%d", &cantArt);

    Venta *articulo = (Venta *)malloc(cantArt * sizeof(Venta));

    for (int j = 0; j < cantArt; j++) {
        printf("Ingrese la marca del producto: ");
        gets(articulo[j].marca);
        printf("Ingrese la categoria del producto: ");
        gets(articulo[j].categoria);
        printf("Ingrese el stock: ");
        scanf("%d", &articulo[j].stock);
        fflush(stdin);

        articulo[j].disponibles = (Producto *)malloc(articulo[j].stock * sizeof(Producto));

        for (int i = 0; i < articulo[j].stock; i++) {
            printf("Ingrese el modelo del producto: ");
            gets(articulo[j].disponibles[i].modelo);
            printf("Ingrese el precio del producto: ");
            scanf("%f", &articulo[j].disponibles[i].precio);
            fflush(stdin);
        }
    }

    mostrarProductos(articulo, cantArt); // dentro de la funcion libera el puntero de productos

    free(articulo); // Libera el puntero de ventas

    return 0;
}
```

Ejercicio –Bonus Track:

En muchas ocasiones, es necesario inicializar con datos variables estructuradas o arreglos de estructuras, para poder probar rápidamente módulos que realicen operaciones particulares. Por este motivo realizamos un ejemplo de como se inicializaría estructuras con el ejercicio que venimos trabajando.

```
void mostrarProductos(venta *punteroVenta);
int main()
{
    //inicializo productos por separado
    producto p1 = {"Galaxy A13", 60999};
    producto p2 = {"Z Flip 4", 242999};
    producto p3 = {"Galaxy A53", 136999};
    producto p4 = {"13-bb0031a", 229999};
    producto p5 = {"14-cf25351a", 180999};
    producto p6 = {"15-dy20541a", 219998};

    //armo arreglos de productos y declaro punteros para manejarlos
    producto sansumg[3] = {p1, p2, p3};
    producto *psansumg;
    psansumg = sansumg;

    producto Hp[3] = {p4, p5, p6};
    producto *pHp;
    pHp = Hp;

    //declaro arreglo general y completo la inicializacion final
    venta mercaderia[] = {{ "sansumg", "Celulares", 3, psansumg},
        { "HP", "Notebook", 3, pHp}};

    venta *punteroVenta;

    punteroVenta = mercaderia;

    mostrarProductos(punteroVenta);
    return 0;
}
```

Ejercicio –Bonus Track:

Además realizamos un pequeño cambio en la función mostrarProductos, solo para poder ver, con un ejemplo, como sería la notación de punteros.

```
void mostrarProductos(venta *punteroVenta){
    puts("*****ARREGLO DE MERCADERIA*****");
    for (int i = 0; i < 2; i++)
    {
        printf("\n-----\n");
        printf("----MARCA => %s \n", punteroVenta->marca);
        printf("----CATEGORIA => %s \n", punteroVenta->categoria);
        printf("----STOCK => %d \n", punteroVenta->stock);
        printf("----PRODUCTOS DISPONIBLES----- \n");
        for (int j = 0; j < punteroVenta->stock; j++)
        {
            printf("Categoria: %s\n", punteroVenta->disponibles[j].modelo);
            printf("Precio: %.2f \n", punteroVenta->disponibles[j].precio);
        }
        punteroVenta++;
    }
    puts("-----");
}
```

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

```
int main()
```

```
{
```

```
    printf("Hasta la próxima clase!!\n");
```

```
    return 0;
```

```
}
```