

PROGRAMACIÓN

Unidad 7 – Parte 1. Tipos de datos derivados estructuras.
Estructuras anidadas.

Revisamos la tarea pendiente



1. ¿Qué significa pasar un parámetro por valor en C?

Pasar un parámetro por valor significa que se envía una copia de la variable original al llamar a una función.

2. ¿Los cambios en un parámetro pasado por valor en una función afectan a la variable original fuera de la función?

No, los cambios en un parámetro pasado por valor en una función no afectan a la variable original fuera de la función, ya que se opera en una copia de la variable original.

3. ¿Cómo se pasa un parámetro por referencia en C?

Para pasar un parámetro por referencia en C, se utiliza un puntero o una dirección de memoria como parámetro. Esto permite realizar cambios en la variable que se verán reflejados fuera de la función.

4. ¿Cuál es la ventaja de pasar parámetros por referencia en comparación con por valor?

La ventaja de pasar parámetros por referencia es que permite modificar el valor original y evitar copiar una variable, lo que puede ser más eficiente en términos de memoria y rendimiento.

Revisamos la tarea pendiente



5. ¿Puedes dar un ejemplo de una función que acepte un parámetro por referencia en C?

```
void duplicar(int *numero)
{
    *numero *= 2;
}
```

6. ¿Qué es la asignación dinámica de memoria en C y por qué es importante?

La asignación dinámica de memoria en C se refiere a la técnica de asignar y liberar memoria durante la ejecución del programa. Es importante para gestionar eficientemente la memoria.

7. ¿Cuáles son las funciones en C que se utilizan para asignar memoria dinámicamente?

La función vista en clases para asignar memoria dinámica en C es malloc. La función malloc asigna un bloque de la pila, la cual se compone de la memoria sin utilizar que no ha sido asignada ni al sistema operativo ni, ni a ningún programa ni a ninguna variable de programa.

8. ¿Cual es la diferencia entre malloc() y realloc()?

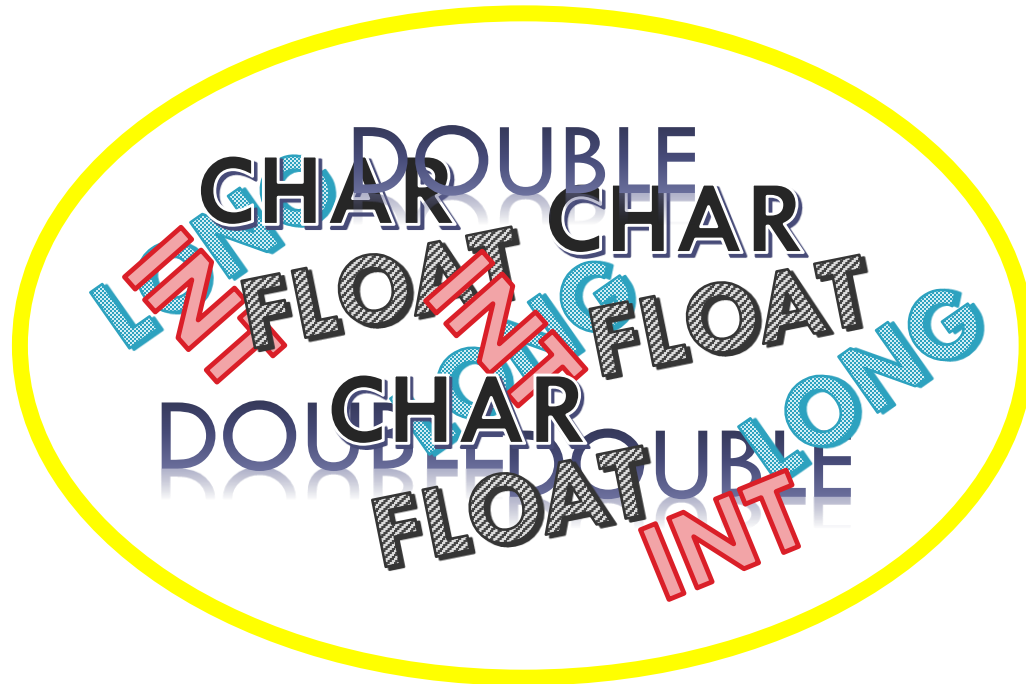
La función malloc asigna un bloque de memoria inicial, mientras que realloc redimensiona un bloque de memoria ya asignado dinámicamente.

9. ¿Por qué es necesario liberar la memoria asignada dinámicamente con free()?

Es necesario liberar la memoria asignada dinámicamente con free() para garantizar que la memoria se devuelva al sistema para su posterior uso.

¿Qué es una Estructura?

Una estructura es una colección de uno o mas tipos de variables, agrupadas bajo un mismo nombre para un manejo conveniente.



Ejemplo de Estructuras

Las estructuras permiten asociar información, o lo que es lo mismo, permiten que un grupo de variables relacionadas sean tratadas como una unidad en lugar de entidades separadas.

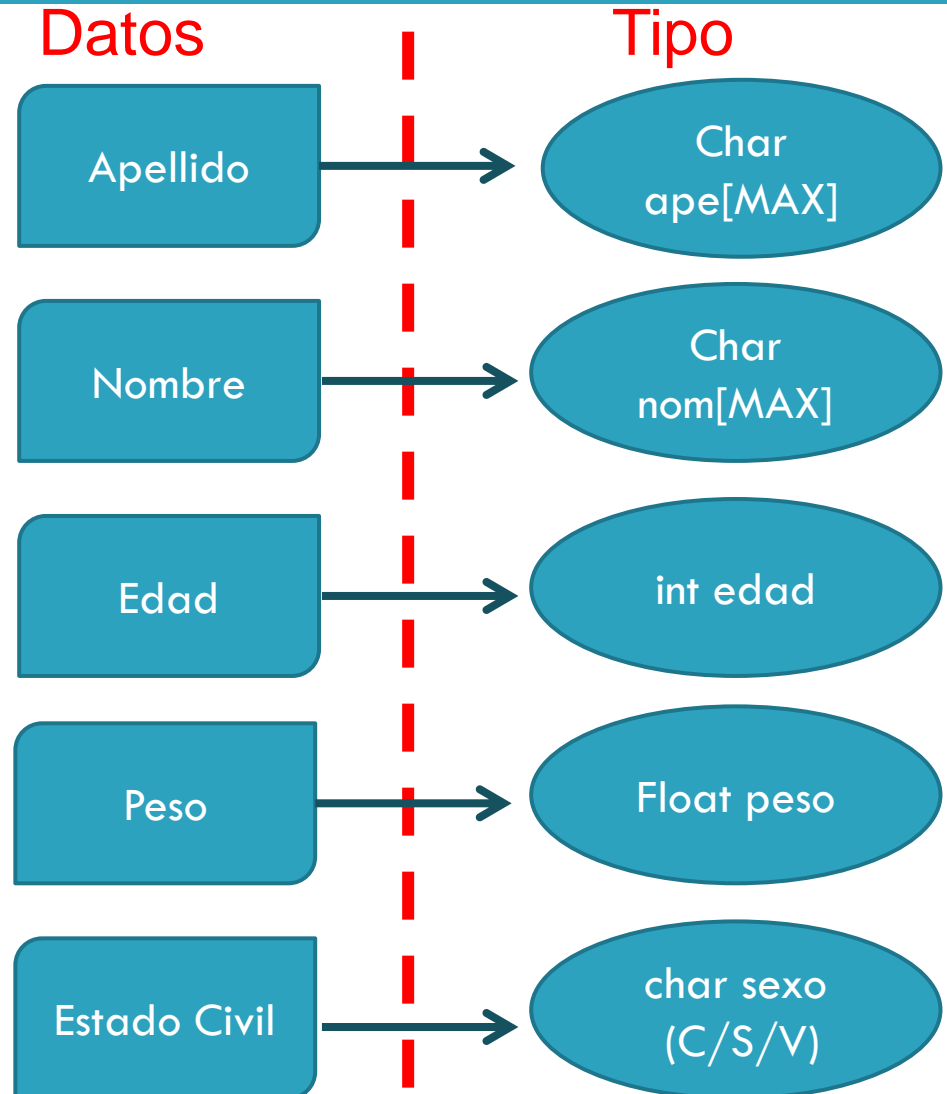


Ejemplo:


Una idea sencilla de una estructura son los datos de una persona:

- Nombre
- Apellido
- Edad
- Fecha de nacimiento
- Peso
- Estado Civil

Ejemplo de Estructuras



Características generales de las estructuras



- ❑ Tienen un nombre
- ❑ Elementos de distintos tipos llamados campos o miembros
- ❑ Acceso a las componentes vía índice o puntero

¿Como crear una Estructura?

Se habrá de usar la Palabra Reservada : **struct**

```
struct [nombre] {  
    tipo1 var1;  
    tipo2 var2;  
    :      :  
    tipoi  vari;  
    :      :  
    tipon  varn;  
};
```

- **Nombre:** optativo, se llama rótulo o etiqueta de la estructura.
- **tipo_i var_i:** son los campos o miembros de la estructura. Son variables que se declaran con su tipo asociado, que podrá ser cualquier tipo válido.

Problema:



Suponga que usted quisiera llevar registro de sus compras en mercado libre. Querría agrupar para su referencia la siguiente información de sus comprar:

- Nombre del producto
- Categoría
- Precio
- Cantidad
- Forma de pago
- Fecha de entrega

Crear una estructura que cumpla con la información detallada



Declaración de variables tipo estructurado

Una vez definido el nuevo tipo de datos, se estará en condiciones de declarar variables asociadas a el, por ejemplo:

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

struct persona {
    char nom[MAX];
    char ape[MAX];
    int edad;
    float peso;
    char sexo;
};
```

```
int main(){
    struct persona PACIENTE;
    struct persona CLIENTE;
```

```
int main(){
    struct persona PACIENTE, CLIENTE;
```

La declaración de las variables “PACIENTE” y “CLIENTE” asociadas al tipo persona SI IMPLICA UNA RESERVA DE MEMORIA.

INICIALIZACION DE UNA VARIABLE ASOCIADA AL TIPO ESTRUCTURA

Se puede inicializar una variable asociada al tipo estructura en su declaración, ellos se hará siguiendo su declaración con una lista de inicializadores, cada uno con una expresión constante para sus miembros.

```
#include <stdio.h>

struct PUNTO {    int puntoX;
                  int puntoY;
};

int main()
{
    struct PUNTO punto1= { 7, 5};
}
```

COMO ACCEDER A LOS CAMPOS O MIEMBROS DE UNA ESTRUCTURA

- El acceso a los miembros o campos: de una variables asociada al tipo estructura se hace a través del **OPERADOR PUNTO**

- La forma de uso del mismo es:

Nombre_de_la_variable.nombre_del_campo

COMO CARGAR DATOS EN UNA VARIABLE ESTRUCTURADA

Podemos usar esta variable para conocer de que manera puedo acceder a los campos de la variable. Si lo que me propongo es por ejemplo, “cargar los datos del individuo”, deberé para ello usar el operador que me permite acceder a los campos de la variable estructurada, o sea el operador punto, de la siguiente manera:

```
gets(individuo.ape);
```

```
scanf("%d", &individuo.edad);
```

Ejemplo

```
#include <stdio.h>
#define MAX 50

struct persona{
    char nom[MAX];
    char ape[MAX];
    int edad;
    float peso;
    char sexo;
};

int main()
{
    struct persona PACIENTE;
    puts("Ingrese la edad del paciente:");
    scanf("%d", &PACIENTE.edad);

    puts("Ingrese el peso del paciente:");
    scanf("%f", &PACIENTE.peso);

    puts("Los datos ingresados son: ");
    printf("Edad: %d \n", PACIENTE.edad);
    printf("Peso: %.3f", PACIENTE.peso);

    return 0;
}
```

OPERACIONES



Con las variables estructuradas se pueden llevar a cabo las siguientes acciones:

- copiar como una unidad
- asignar entre ellas
- acceder y trabajar con sus campos
- anidar estructuras
- usar como argumentos de funciones: completas y por partes
- tomar su dirección (&)
- ser devueltas por funciones

Ejemplo



```
#include <stdio.h>

✓ struct persona {
    |             |
    |             | char ape[100];
    |             | char nom[100];
    |             | };

int main()
✓ {
    struct persona PACIENTE1;
    struct persona PACIENTE3;

    puts("Paciente N° 1");
    puts("Ingrese el apellido del paciente");
    gets(PACIENTE1.ape);
    puts("Ingrese el nombre del paciente");
    gets(PACIENTE1.nom);

    PACIENTE3 = PACIENTE1;

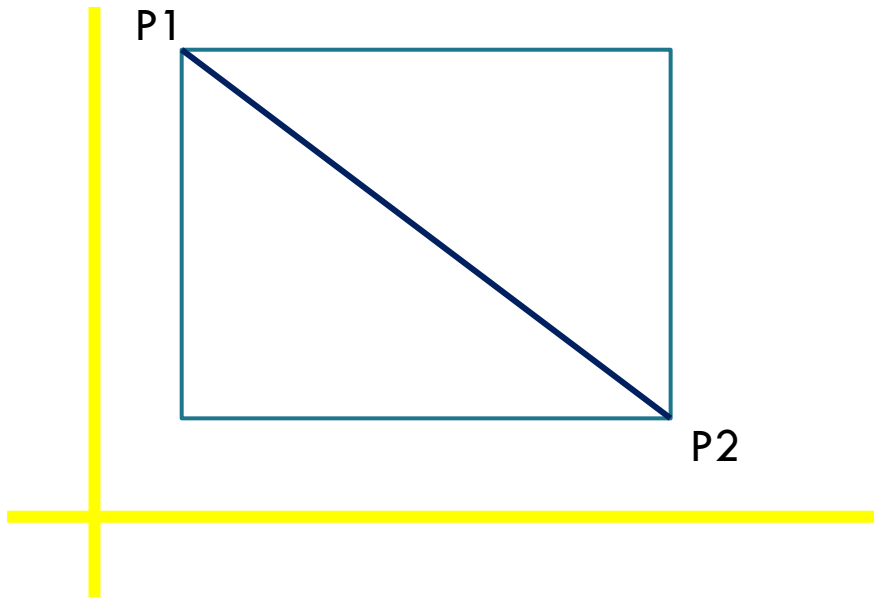
    puts("El nombre del Paciente 1 es:");
    puts(PACIENTE1.nom);

    puts("El nombre del Paciente 3 es:");
    puts(PACIENTE3.nom);

    return 0;
}
```


ESTRUCTURAS ANIDADAS

Las estructuras pueden anidarse. Una representación de un rectángulo se puede hacer como un par de puntos que denotan las esquinas diagonalmente opuestas



```
#include <stdio.h>

struct PUNTO {
    int puntoX;
    int puntoY;
};

struct RECTA {
    struct PUNTO P1;
    struct PUNTO P2;
};
```

Typedef

- Typedef permite crear identificadores para algún tipo de dato que ya existe.
- Es una facilidad proporcionada por el lenguaje C para trabajar con identificadores
- Typedef no introduce nuevos tipos, sólo formas sinónimos para tipos que se podrían mencionar de otra forma.

```
#include <stdio.h>

int main()
{
    typedef int entero;

    entero numero=5;

    printf("El valor del número es: %d", numero);

    return 0;
}
```

Hace del nombre **entero** sea sinónimo de **int**.

```
#include <stdio.h>

struct cd{
    char nombre[30];
    char interprete[25];
    int cant_temas;
    int duracion;
};

typedef struct cd Discos;

int main()
{
    Discos mios;
}
```

Hace que el nombre **Discos** sea sinónimo de **struct cd**.

```
#include <stdio.h>

struct
{
    char nombre[30];
    char interprete[25];
    int cant_temas;
    int duracion;
}typedef Discos;

int main()
{
    Discos mios;
}
```

Define el nombre de la estructura en la misma declaración, evitando el uso de la palabra **struct**.

Retomemos nuestro Ejercicio:



1. ¿En que caso podríamos tener estructuras anidadas en nuestro ejercicio?
2. Cargue los datos de dos productos comprados.
3. Muestre por pantalla cada uno de los campos de la estructura.
4. Compare los precios de ambos productos, e indique cual es el mas costoso.



Bonus track: Uniones

En el lenguaje C, las uniones son una estructura de datos que permite almacenar diferentes tipos de datos en la misma ubicación de memoria. Al igual que las estructuras, las uniones son una forma de definir un nuevo tipo de datos personalizado. Sin embargo, hay algunas diferencias clave entre las estructuras y uniones en C:

- ❑ Almacenamiento compartido: Mientras que en una estructura, cada miembro tiene su propia ubicación de memoria y ocupa espacio separado, en una unión, todos los miembros comparten la misma ubicación de memoria. Esto significa que solo uno de los miembros de la unión puede tener un valor válido en un momento dado.
- ❑ Tamaño: El tamaño de una unión es igual al tamaño de su miembro más grande. Esto se debe a que todos los miembros comparten la misma ubicación de memoria, y el tamaño de la unión debe ser lo suficientemente grande para acomodar el miembro más grande.
- ❑ Acceso a los miembros: Aunque una unión puede contener múltiples miembros, solo se puede acceder a uno de ellos a la vez. Esto se hace para garantizar que los datos almacenados en la unión se interpreten de manera correcta. Se puede acceder a un miembro de una unión utilizando el operador "." (punto) de la misma manera que lo haríamos con una estructura.

Bonus track: Uniones

```
#include <stdio.h>

union Datos {
    int entero;
    float flotante;
    char character;
};

int main() {
    union Datos miUnion;

    miUnion.entero = 42; // Asignar un valor entero a la unión
    printf("Valor entero: %d\n", miUnion.entero);

    miUnion.flotante = 3.14; // Ahora almacenamos un valor real
    printf("Valor flotante: %f\n", miUnion.flotante);

    miUnion.character = 'A'; // Almacena un carácter
    printf("Carácter: %c\n", miUnion.character);

    return 0;
}
```

Tarea para la casa



1. ¿Qué es una estructura en C y para qué se utiliza?
2. ¿Cuál es la diferencia entre una estructura y un arreglo en C?
3. ¿Cuál es la diferencia entre una estructura y una unión en C?
4. ¿Cómo se declara una estructura en C?
5. ¿Cómo se accede a los miembros de una estructura en C?
6. ¿Qué es una estructura anidada y cómo se utiliza?
7. ¿Cómo se inicializa una estructura en C?
8. ¿Cómo crearías una estructura para representar un punto en un espacio tridimensional?

Lo revisamos la próxima clase.

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

```
int main()
```

```
{
```

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

```
    return 0;
```

```
}
```