Programación en C: Funciones - III

1ºDAM IoT

Paso de parámetros por valor y referencia

Paso por valor y paso por referencia

En los lenguajes de programación hay dos formas de pasarle el valor de un parámetro a una función:

- **Paso por valor**:
  - Los valores son copiados a las variables locales de la función. Los cambios no afectan a las variables originales. Es lo que hemos usado en los ejemplos anteriores.
  - C usa el paso por valor por defecto.
- **Paso por referencia**:
  - Obliga a utilizar variables en la invocación a la función. No podemos hacer *suma(5,2);*
  - Cualquier cambio producido sobre los parámetros formales repercute en las variables de la invocación. Es decir, si modifico la variable en la función fuera se ve el cambio.
  - Es una buena solución cuando deseamos modificar el valor de las variables usadas en la invocación a la función.
  - Cuando devolver un único valor no es suficiente, el paso por referencia puede servir para que la función pueda devolver más valores.

Paso por referencia

Los parámetros que son pasados por referencia llevan un **asterisco** * antes del nombre.
Una misma función puede contener variables pasadas por valor y por referencia.

*tipo_valor_retorno identificador_funcion (tipo_dato par1, **tipo_dato** ***par2**, … tipo_dato parn)*
{
    instrucción1;
    instrucción2;
    par2 = 10;
    instrucciónn;
    return tipo_valor_retorno;
}
*Llamada a la función → miFuncion(var1, **&var2**, var3);*

Parámetro pasado por referencia. Cualquier modificación que se haga dentro de la función se está haciendo sobre la variable original.

Cuando se usa el parámetro pasado por referencia dentro de la función se pone el asterisco delante del nombre.

Las variables por referencia cuando se llaman a la función se indican con el símbolo **&**.

Paso por referencia

Ejemplo de paso por referencia:

```c
#include<stdio.h>

void inicializar(int *pe1,int *pe2)
{
    *pe1=100;
    *pe2=200;
}


int main()
{
    int x1,x2;
    inicializar(&x1,&x2);
    printf("%i   %i",x1,x2);
    return 0;
}
```

Ambos parámetros son pasados por referencia, se les añade el *.

Como son pasados por referencia para usarlos debemos añadir el * delante dentro de la función.

Cuando llamamos a la función añadimos el símbolo **&** a los parámetros que son pasados por referencia. Como hacemos con scanf.

Al imprimir los valores, veremos los valores de x1 y x2 con la asignación que ha hecho la función inicializar.

Paso por referencia

Ejercicio: Dado el siguiente ejercicio. Modifica la función calcularSuperficie para que no devuelva nada e incluya un parámetro superficie pasándolo por referencia.

```c
#include<stdio.h>

int calcularSuperficie(int lado)
{
    int superficie=lado*lado;
    return superficie;
}

int main()
{
    int valor;
    int sup;
    printf("Ingrese el valor del lado del cuadrado:");
    scanf("%i",&valor);
    sup=calcularSuperficie(valor);
    printf("La superficie del cuadrado es %i",sup);
    return 0;
}
```

Tendrás que modificar también la llamada a la función para pasarle la variable sup por referencia.

Paso por referencia

Solución:

```c
#include <stdio.h>

void calcularSuperficie(int lado, int *superficie)
{
    *superficie = lado * lado;
}

int main()
{
    int valor;
    int superficie;
    printf("Ingrese el valor del lado del cuadrado:");
    scanf("%i", &valor);
    calcularSuperficie(valor, &superficie);
    printf("La superficie del cuadrado es %i", superficie);
    return 0;
}
```

El parámetro lado se pasa por valor.

El parámetro superficie se pasa por referencia.

Como superficie es pasada por referencia debemos anteponer el *.

Cuando llamamos a la función añadimos el símbolo **&** al parámetro que pasamos por referencia. Como hacemos con scanf.

Funciones con arrays como parámetros

Funciones con arrays como parámetro

Una función puede recibir como parámetros arrays. Si modificamos el parámetro de tipo array dentro de la función se modifica la variable definida en la función main o donde se haya definido.

```c
#include <stdio.h>

void inicializar(int vec[5])
{
    int i;
    for (i = 0; i < 5; i++)
    {
        printf("Ingrese elemento:");
        scanf("%i", &vec[i]);
    }
}

int main()
{
    int vector[5];
    inicializar(vector);
    int i;
    for (i = 0; i < 5; i++)
    {
        printf("%i ", vector[i]);
    }
    return 0;
}
```

A la función se le pasa un array y se modifica su contenido.

Internamente los arrays se pasan por referencia sin tener que indicarlo nosotros.

Llamada a la función inicializar

Si imprimimos el array comprobaremos que tiene los valores asignados dentro de la función.

Funciones con arrays como parámetro

Ejercicio: Crea un programa en el que pides el nombre y apellidos de un usuario. Para leer el nombre y guardarlo en un array vas a crear una función a la que le pasas un texto a mostrar al usuario y un array, y le asigna el valor que escribe el usuario por teclado. Date cuenta que a esta función la llamarás dos veces, una para rellenar el nombre y otra para rellenar los apellidos. Termina el programa con la impresión por pantalla del nombre y los apellidos.

A continuación, una posible salida del programa.

```
Introduce tu nombre
Mariajo
Introduce tus apellidos
Casalins Pina
Tu nombre es Mariajo y tus son apellidos Casalins Pina
```

Funciones con arrays como parámetro

Solución:

```c
#include <stdio.h>

void solicitaDatos(char cadena[50], char aux[100])
{
    printf("%s\n", cadena);
    fflush(stdin);
    gets(aux);
}

int main()
{
    char nombre[100];
    char apellidos[100];
    solicitaDatos("Introduce tu nombre", nombre);
    solicitaDatos("Introduce tus apellidos", apellidos);
    printf("Tu nombre es %s y tus apellidos son %s", nombre, apellidos);
    return 0;
}
```

Funciones que devuelven struct

Funciones que devuelven estructuras

Para indicar que una función devuelve una estructura en el tipo de dato que devuelve se indica que es struct nombre_struct.

struct nombre_struct *identificador_funcion (<parametros>)*
{
    instrucción1;
    instrucción2;
    …
    instrucciónn;
    return tipo_valor_retorno; ← Debe ser de tipo struct.
}

Funciones que devuelven estructuras

```c
#include <stdio.h>

struct producto
{
    int codigo;
    char descripcion[41];
    float precio;
};

struct producto inicializar()
{
    struct producto pro;
    printf("Ingrese el codigo de producto:");
    scanf("%i", &pro.codigo);
    fflush(stdin);
    printf("Ingrese la descripcion:");
    gets(pro.descripcion);
    printf("Ingrese el precio:");
    scanf("%f", &pro.precio);
    return pro;
};

int main()
{
    struct producto miproducto;
    miproducto = inicializar();
    return 0;
}
```

Declara la estructura fuera del main para que la función reconozca este tipo de dato.

El tipo de dato que devuelve es struct producto.

La variable pro es de tipo struct.

La función inicializar devuelve una variable de tipo struct.

Funciones que devuelven estructuras

Ejercicio: Modifica el ejercicio anterior. Ahora vamos a tener un array de tipo struct producto de tamaño 3. Asigna los valores a cada posición del array ayudándote de la función inicializar(). Para finalizar, recorre el array e imprime todos sus valores.

Funciones que devuelven estructuras

Solución:

```c
#include <stdio.h>
#define TAM 3
struct producto{
    int codigo;
    char descripcion[41];
    float precio;
};
struct producto inicializar(){
    struct producto pro;
    printf("Ingrese el código de producto: ");
    scanf("%i", &pro.codigo);
    fflush(stdin);
    printf("Ingrese la descripción: ");
    gets(pro.descripcion);
    printf("Ingrese el precio: ");
    scanf("%f", &pro.precio);
    return pro;
};
int main(){
    struct producto productos[TAM];
    int i=0;
    for(i=0;i<TAM;i++){
        productos[i] = inicializar();
    }
    //Imprimo el array de productos
    for(i=0;i<TAM;i++){
        printf("Producto nº%i\n", i+1);
        printf("Código: %i, Descripción: %s, Precio: %f\n", productos[i].codigo, productos[i].descripcion, productos[i].precio);
    }
    return 0;
}
```

Funciones con estructuras como parámetros

Funciones con estructuras como parámetros

Las estructuras pueden ser pasadas una función como parámetros de dos formas:

- **Por valor**: No queremos modificar los valores de la estructura.
- **Por referencia**: Queremos modificar la estructura y que el cambio se vea fuera.

Funciones con estructuras como parámetros

Si la función NO va a modificar la estructura se pasa por valor.

Declara la estructura fuera del main para que la función reconozca este tipo de dato.

```c
#include <stdio.h>
#include <string.h>
struct producto
{
    int codigo;
    char descripcion[41];
    float precio;
};
void imprimir(struct producto producto)
{
    printf("Codigo:%i\n", producto.codigo);
    printf("Descripcion:%s\n", producto.descripcion);
    printf("Precio:%0.2f", producto.precio);
}
int main()
{

    struct producto prod;
    prod.codigo = 1;
    strcpy(prod.descripcion, "Mi producto");
    prod.precio = 10;
    imprimir(prod);
    return 0;
}
```

La función imprimir() no modifica nada de la estructura solamente imprime su contenido.

Para asignar una cadena a un array de caracteres ya declarado usa strcpy

Llamada a la función imprimir

Funciones con estructuras como parámetros

Ejercicio: Modifica el ejercicio que hiciste del array de struct producto. Crea una función para imprimir los valores de una estructura producto. Esta función no devuelve nada sólo imprime los valores de la estructura producto.

Funciones con estructuras como parámetros

Solución:

```c
#include <stdio.h>
#define TAM 3
struct producto{
    int codigo;
    char descripcion[41];
    float precio;
};
struct producto inicializar(){
    struct producto pro;
    printf("Ingrese el código de producto: ");
    scanf("%i", &pro.codigo);
    fflush(stdin);
    printf("Ingrese la descripción: ");
    gets(pro.descripcion);
    printf("Ingrese el precio: ");
    scanf("%f", &pro.precio);
    return pro;
};
```

```c
void imprimirProducto(struct producto p, int i){
        printf("Producto nº%i\n", i);
        printf("Código: %i, Descripción: %s, Precio: %f\n", p.codigo, p.descripcion, p.precio);
}
int main(){
    struct producto productos[TAM];
    int i=0;
    for(i=0;i<TAM;i++){
        productos[i] = inicializar();
    }
    //Imprimo el array de productos
    for(i=0;i<TAM;i++){
        imprimirProducto(productos[i], i);
    }
    return 0;
}
```

Funciones con estructuras como parámetros

Si vamos a pasar cómo parámetro una estructura que la función va a modificar debemos pasarla por referencia para que los cambios puedan verse fuera.

Funciona igual que cuando pasábamos por referencia algún parámetro de tipo int, float o char.

*tipo_valor_retorno identificador_funcion (tipo_dato par1, **struct estructura *par2**, … tipo_dato parn)*
{
instrucción1;
instrucción2;
par2.campo1 = 10;
scanf("%d", &(*par2).campo2);
return tipo_valor_retorno;
}
*Llamada a la función → miFuncion(var1, **&miEstructura**, var3);*

Funciones con estructuras como parámetros

Al acceder a la variable tenemos que poner su nombre entre paréntesis y anteponer el *.

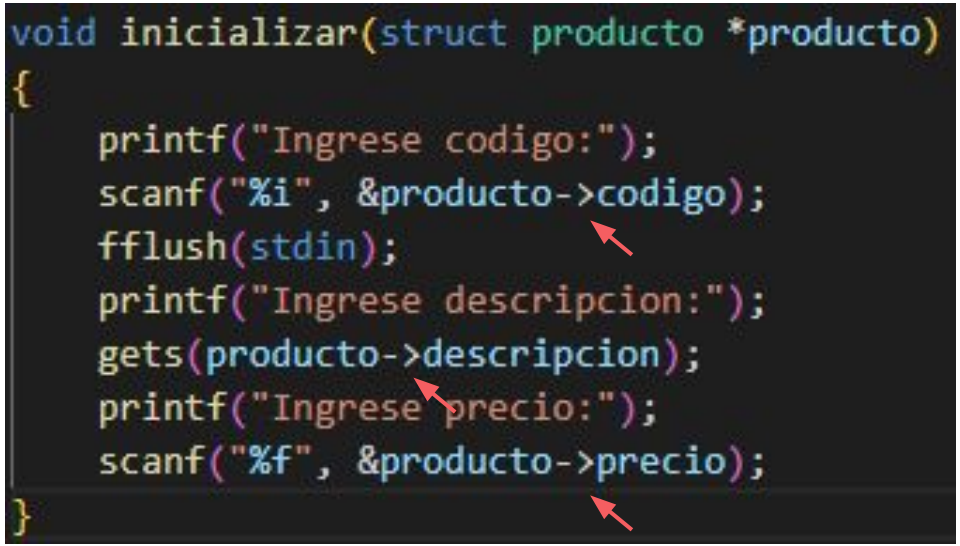
```c
void inicializar(struct producto *producto){
    printf("Ingrese codigo");
    scanf("%i", &(*producto).codigo);
    fflush(stdin);
    printf("Ingrese descripción");
    gets((*producto).descripcion);
    printf("Ingrese precio");
    scanf("%f", &(*producto).precio);
}

int main(){
    struct producto prod;
    inicializar(&prod);
    imprimir(prod);
    return 0;
}
```

Con el * indicamos que la estructura se pasa por referencia.

En la llamada anteponemos el símbolo **&** para indicar que se pasa por referencia.

Funciones con estructuras como parámetros

En estos casos para hacer más legible el código podemos utilizar el símbolo “->” para acceder al campo de una estructura desde la función. En ese caso no es necesario escribir el *.



```
void inicializar(struct producto *producto)
{
    printf("Ingrese codigo:");
    scanf("%i", &producto->codigo);
    fflush(stdin);
    printf("Ingrese descripcion:");
    gets(producto->descripcion);
    printf("Ingrese precio:");
    scanf("%f", &producto->precio);
}
```

Funciones con estructuras como parámetros

Ejercicio: Modifica el ejercicio del array de struct para utilizar la función inicializar que recibe un struct producto por referencia. Puedes utilizar cualquiera de las dos notaciones para utilizar un struct pasado por referencia.

Funciones con estructuras como parámetros

Solución: Utilizando notación **->**

```c
#include <stdio.h>
#define TAM 3
struct producto{
    int codigo;
    char descripcion[41];
    float precio;
};
void inicializar(struct producto *pro){
    printf("Ingrese el código de producto: ");
    scanf("%i", &pro->codigo);
    fflush(stdin);
    printf("Ingrese la descripción: ");
    gets(pro->descripcion);
    printf("Ingrese el precio: ");
    scanf("%f", &pro->precio);
};
void imprimirProducto(struct producto p, int i){
        printf("Producto nº%i\n", i);
        printf("Código: %i, Descripción: %s, Precio: %f\n", p.codigo, p.descripcion, p.precio);
}
int main(){
    struct producto productos[TAM];
    int i=0;
    for(i=0;i<TAM;i++){
        inicializar(&productos[i]);
    }
    //Imprimo el array de productos
    for(i=0;i<TAM;i++){
        imprimirProducto(productos[i], i);
    }
    return 0;
}
```

Funciones con estructuras como parámetros

Solución: Utilizando notación * y .

```c
#include <stdio.h>
#define TAM 3
struct producto{
    int codigo;
    char descripcion[41];
    float precio;
};
void inicializar(struct producto *pro){
    printf("Ingrese el código de producto: ");
    scanf("%i", &(*pro).codigo);
    fflush(stdin);
    printf("Ingrese la descripción: ");
    gets((*pro).descripcion);
    printf("Ingrese el precio: ");
    scanf("%f", &(*pro).precio);
};
void imprimirProducto(struct producto p, int i){
        printf("Producto nº%i\n", i);
        printf("Código: %i, Descripción: %s, Precio: %f\n", p.codigo, p.descripcion, p.precio);
}
int main(){
    struct producto productos[TAM];
    int i=0;
    for(i=0;i<TAM;i++){
        inicializar(&productos[i]);
    }
    //Imprimo el array de productos
    for(i=0;i<TAM;i++){
        imprimirProducto(productos[i], i);
    }
    return 0;
}
```

Funciones recursivas

Funciones recursivas

Es una técnica de programación que nos permite que un bloque de instrucciones se ejecute n veces.

Reemplaza en ocasiones a estructuras repetitivas.

En C las funciones pueden llamarse a sí mismos. Si dentro de una función existe la llamada a sí mismo decimos que la función es recursiva.

Son útiles para recorrer una estructura en árbol, por ejemplo para recorrer un árbol de directorios de nuestra máquina. Haremos ejercicios de este tipo con C#.

Funciones recursivas

Ejemplo: Escribir una función que dado un número imprima de forma descendente los números desde ese número hasta el 1. Llamaremos a la función con 5.

```c
#include <stdio.h>

void imprimir(int x){
    int i = 0;
    for(i=x;i>0;i--){
        printf("%i ", i);
    }
}

int main()
{
    imprimir(5);
    return 0;
}
```

Lo implementamos con una función que contiene un bucle for, recorremos los números descendientes y los imprimimos.

Funciones recursivas

Ejemplo: Si lo implementamos con una función recursiva se quedaría de la siguiente forma.

```c
void imprimirRecursivo(int x)
{
    //x>0 es la condición de parada para no seguir llamando a la función recursiva infinitamente.
    if (x > 0)
    {
        printf("%i ", x);
        //Llamamos a la función que imprime con un valor menos
        imprimirRecursivo(x - 1);
    }
}

int main()
{
    imprimirRecursivo(5);
    return 0;
}
```

```c
void imprimir(int x){
    int i = 0;
    for(i=x;i>0;i--){
        printf("%i ", i);
    }
}
```

Funciones recursivas

Ejercicio: Imprimir los números de 1 a 5 en pantalla utilizando recursividad.

Funciones recursivas

Solución: Imprimir los números de 1 a 5 en pantalla utilizando recursividad.

```c
#include <stdio.h>

void imprimirRecursivo(int x)
{
    //x>0 es la condición de parada para no seguir llamando a la función recursiva infinitamente.
    if (x > 0)
    {
        //Llamamos a la función que imprime con un valor menos
        imprimirRecursivo(x - 1);
        printf("%i ", x);
    }
}

int main()
{
    imprimirRecursivo(5);
    return 0;
}
```