Sesión 5. Referencias a memoria

Empezaremos remarcando de nuevo que C es un lenguaje de programación que requiere de su compilación antes de poder ejecutar el programa. Sin embargo, gracias a Jupyter, ese paso se hace automáticamente. Para poder ejecutar un código en este entorno, basta con seleccionar la celda que contiene el bloque de código y pulsar las teclas **Shift+Enter**.

Sección 1

Para repasar como funcionan los punteros, aquí tenemos el código de la explicación. Ejecútalo y observa su salida. Verás que las direcciones de memoria que se muestran no son siempre las mismas.

In [ ]:
```
#include <stdio.h>

int main(void)
{
    int a = 8;
    int * p_a;

    printf("El valor de a es %d y su dirección %p\n", a, &a);
    printf("El valor de p_a es %p\n",p_a);
    p_a = &a; //a p_a le damos el valor de la dirección de memoria de a

    //El valor de a no cambia
    printf("El valor de a es %d y su dirección %p\n", a, &a);
    //El valor de p_a ahora es distinto
    printf("El valor de p_a es %p y su valor %d\n",p_a, *p_a);
    return 0;
}
```

¿Tiene sentido lo que acabas de ver? Sabemos que este nuevo concepto es un poco abstracto, por tanto te recomendamos que veas de nuevo el vídeo y que utilices el ejemplo para entender que relación hay entre el valor y su dirección de memoria.

Sección 2

En esta sección vamos a ver como utilizar los parámetros de entrada de las funciones (las variables que aparecen entre paréntesis a la derecha del nombre de la función) para almacenar resultados calculados dentro de ellas. ¿Para qué? Ahora veras...

Queremos implementar una función que calcule devuelva el valor mínimo y máximo de un vector de N elementos. Esto implica darle un vector a la función y devolver dos valores, a ver cómo lo hacemos...

In [ ]:
```c
#include <stdio.h>
#include <string.h>

void minmax(int * vec, int N){
    int i = 0;
    int min = vec[0];
    int max = vec[0];
    for(i = 1; i < N; i++){
        if (vec[i] < min){
            min = vec[i];
    }
        if (vec[i] > max){
            max = vec[i];
        }
    }
    printf("El máximo es %d y el mínimo es %d\n", max, min);
}

int main(void)
{
    int i, vector[5];

    for(i = 0; i < 5; i++) {
        vector[i] = i*2;
    }

    minmax(vector, 5);
    return 0;
}
```

Lo que hemos hecho es pasarla el vector (realmente la dirección de memoria dónde se encuentra su primer elemento) y el tamaño del vector para poder recorrerlo. Pero el enunciado decía que la función tenía que DEVOLVER los resultados, no mostrarlos, como estamos haciendo. ¿Y cómo lo hacemos si el "return" sólo puede devolver un valor?

Nos tocará utilizar los parámetros de la función para almacenar el resultado.

In [ ]:
```c
#include <stdio.h>
#include <string.h>

int minmax(int * vec, int min, int max, int N){
    if ( N < 1 ) return 0;

    int i = 0;
    min = vec[0];
    max = vec[0];
    for(i = 1; i < N; i++){
        if (vec[i] < min){
            min = vec[i];
    }
        if (vec[i] > max){
            max = vec[i];
        }
    }
    return 1;
}

int main(void)
{
    int minimo=99, maximo=-33;
    int i, vector[5];

    for(i = 0; i < 5; i++) {
        vector[i] = i*2;
    }

    if(minmax(vector,minimo,maximo,5)){
        printf("El máximo es %d y el mínimo es %d\n", maximo, minimo);
    }
    else{
        printf("El vector debe contener algún elemento\n");
    }
    return 0;
}
```

¿Qué ha pasado? No nos ha calculado lo que esperábamos. Espera... ¡le hemos pasado los parámetros a la función por valor y no por referencia!

Si los pasamos por valor, los cambios son locales a la función, mientras que si le pasamos las referencias (direcciones de memoria) los cambios permaneceran durante toda la ejecución.

¡Manos a la obra!

In [ ]:
```c
#include <stdio.h>
#include <string.h>

int minmax(int * vec, int * min, int * max, int N){
    if ( N < 1 ) return 0;

    int i = 0;
    *min = vec[0];
    *max = vec[0];
    for(i = 1; i < N; i++){
        if (vec[i] < *min){
            *min = vec[i];
    }
        if (vec[i] > *max){
            *max = vec[i];
        }
    }
    return 1;
}

int main(void)
{
    int minimo=99, maximo=-33;
    int i, vector[5];

    for(i = 0; i < 5; i++) {
        vector[i] = i*2;
    }

    if(minmax(vector,&minimo,&maximo,5)){
        printf("El máximo es %d y el mínimo es %d\n", maximo, minimo);
    }
    else{
        printf("El vector debe contener algún elemento\n");
    }
    return 0;
}
```

¿Has visto como se ha modificado el código? Es importante que tomes un tiempo para asimilarlo ya que fácil no es.

¡Además hemos utilizado el return para comprobar que la función ha hecho su cometido!

Por cierto, el primer código de la sección muestra una función sin "return", ¿por qué? sigue con el siguiente vídeo y descubrirás los "procedimientos".

Ejercicio: Calculadora

Implementa la última versión de la calculadora. En este caso debe poder operar con vectores, por lo tanto habrá que añadir el modo vector a los ya existentes. En este modo, dados dos vectores, se podrá efectuar la suma, resta, multiplicación y división de sus elementos.

Además, las funciones implementadas en la sesión 4 (modo “números”), deberán convertirse en procedimientos. En lugar de devolver su resultado con el return, se deberá pasar una variable por referencia para almacenarlo.

Para ayudarte, te damos la estructura del programa:

In [ ]:
```c
#include <stdio.h>
#include <string.h>

void suma (float a, float b, float * res) {
    *res = a + b;
}

float modoNum(float n1, float n2, int op){
        float res;
        switch (op){
        case (0):
            printf("Operación:\tSuma\n");
            suma(n1, n2, &res);
            break;
        case (1):
            printf("Operación:\tResta\n");
            //res = num1 - num2;
            break;
        case (2):
            printf("Operación:\tMultiplicación\n");
            //
            break;
        case (3):
            printf("Operación:\tDivisión\n");
            //
            break;
        default:
            printf("Código de operación no valido.\n");
    }

    return res;
}

void modoVect(int op, int *v1, int *v2, float *vRes, int N) {
    int i;
    switch (op){
        case (0):
            printf("Operación:\tSuma\n");
            for (i=0; i<N; i++) {
                vRes[i] = v1[i] + v2[i];
            }
            break;
        //case (1):
        // ...
    }
}

int main(void)
{
    /*  Declaración de variables. */
    int modo = 1; //0: números, 1: vector.
    int op = 0; //operación
    /*
    * operación: 0: suma, 1: resta, 2: multiplicación, 3: división.
    */

    float res;

    /* Cuerpo de programa */
    printf("Bienvenido/a a \"Calcooladora v5.0\".\n\n");

    if (modo == 0) {
        float num1 = 3;
        float num2 = 4;
        res = modoNum(num1, num2, op);
        printf("Resultado:\t %0.2f\n", res);
    } else if (modo == 1) {
        int v1[4] = {4,8,20,3};
```