

(Versión de fecha 29 de noviembre de 2024)

Es frecuente que, al arrancar la ejecución de un programa, se le pasen algunos parámetros. Un ejemplo podría ser la llamada al compilador de C:

```
gcc programa.c -o programa.exe
```

En la instrucción anterior, el programa que se va a ejecutar es el compilador gcc. El resto son parámetros que se pasan a gcc para que pueda completar su ejecución: el nombre del fichero que queremos compilar y el del programa ejecutable que se debe generar.

En este documento se va a explicar cómo preparar la función *main()* de nuestros programas para que acepte parámetros y cómo se pueden procesar dichos parámetros.

Se mostrará primero un ejemplo muy sencillo que permita entender el mecanismo de parámetros de la función *main()* y cómo pasar esos parámetros al programa al ejecutarlo desde el terminal o desde VSCode. A continuación, se presentará un ejemplo un poco más complejo que permitirá entender el tratamiento de los parámetros para extraer información numérica de los mismos.

1 HOLA MUNDO CON PARÁMETROS

En todos los programas que hacemos en C, la ejecución comienza por la función *main()*. Hasta ahora, en los programas de ejemplo que se han visto a lo largo del curso, la función *main()* tenía la siguiente estructura:

```
int main() { ... }
```

Es posible declarar la función *main()* para que admita parámetros. En ese caso, la estructura debe ser la siguiente:

```
int main(int argc, char* argv[]) { ... }
```

En la expresión anterior, los parámetros de la función *main()* son:

- **int argc**: es el *argument counter*. Es un entero cuyo valor indica el número de cadenas de caracteres en la línea de llamada al programa. Por ejemplo, al llamar al compilador con la siguiente instrucción, el número de cadenas de caracteres sería 4:

```
gcc programa.c -o programa.exe
```

- **char* argv[]**: es el *argument vector*, un puntero a cadenas de caracteres, o lo que es lo mismo, una colección de cadenas de caracteres. En la llamada al compilador que estamos utilizando como ejemplo, las cadenas de caracteres son: "gcc", "programa.c", "-o" y "programa.exe". La función *main()* recibe un array cuyos elementos son dichas cadenas de caracteres. Así, una vez dentro de *main()*, *argv[0]* sería la cadena "gcc", *argv[1]* sería la cadena "programa.c", *argv[2]* sería la cadena "-o" y *argv[3]* sería la cadena "programa.exe".

Vamos a desarrollar un pequeño programa que nos permita mostrar cómo recoger los parámetros que

recibe la función *main()* y cómo utilizarlos dentro del programa. También vamos a explicar como realizar las llamadas al programa pasándole parámetros, tanto al hacer las llamadas desde el terminal como al hacerlas desde VSCode.

El primer paso es crear un nuevo proyecto para VSCode. Para ello, utilizando la opción de menú *Fichero -> Abrir carpeta*, crea una nueva carpeta en la que puedas guardar el código fuente del programa que vamos a desarrollar. En nuestro caso, dentro de la carpeta *Prog_1* que venimos usando a lo largo del curso, hemos creado la carpeta *HelloParametros* y la hemos abierto con VSCode.

A continuación, debes crear un fichero dentro de la carpeta y teclear el código del programa. Nosotros hemos llamado al fichero *hello.c* y su código sería el siguiente:

Ejemplo 1 Código de *hello.c*: un «Hola, mundo» con parámetros

```
#include <stdio.h>

int main(int argc, char* argv[]) {

    // Numero de argumentos
    printf("argc= %d\n", argc);

    // Cadenas de los argumentos
    for(int i=0; i<argc; i++) {
        printf("argv[%d]= %s\n", i, argv[i]);
    }

    // Saludo
    if(argc>1) {
        printf("Hola, %s!\n", argv[1]);
    } else {
        printf("Hola, mundo!\n");
    }
}
```

El programa imprime en primer lugar el número de cadenas de caracteres de las que consta la llamada, el valor de *argc*. A continuación, imprime esas cadenas, utilizando una línea para cada cadena. Por último hace el saludo: si hay algún parámetro, lo utiliza para saludar, si no, realiza el clásico saludo *Hola, mundo!*.

Ahora debes compilar el programa. Lo puedes hacer desde VSCode, eligiendo la opción de menú *Terminal -> Ejecutar tarea de compilación* o utilizando el atajo de teclado *CTRL+SHIFT+B*.

Para ejecutar el programa, vamos a utilizar un terminal externo. Pulsa *CTRL+SHIFT+C* para que se abra un terminal situado en el directorio raíz del proyecto.

Una vez en el terminal, teclea el nombre del programa para que se ejecute. En este caso, además del nombre del programa, añade alguna palabra extra para que el programa la utilice como parámetro. En la siguiente figura puedes ver las tres ejecuciones que hemos hecho nosotros: una primera llamada sin parámetros adicionales, una segunda con un parámetro y una tercera con dos parámetros.

```
C:\Windows\System32\cmd.exe

C:\Prog_I\Hello>hello
argc= 1
argv[0]= hello
Hola, mundo!

C:\Prog_I\Hello>hello Mariola
argc= 2
argv[0]= hello
argv[1]= Mariola
Hola, Mariola!

C:\Prog_I\Hello>hello planeta Tierra
argc= 3
argv[0]= hello
argv[1]= planeta
argv[2]= Tierra
Hola, planeta!

C:\Prog_I\Hello>
```

Es posible lanzar los programas desde dentro de VSCode utilizando una de las opciones de menú: *Ejecutar* -> *Ejecutar sin depuración (CTRL+F5)* o bien *Ejecutar* -> *Iniciar depuración (F5)*. Para pasar parámetros al programa al ejecutarlo, hay que personalizar el fichero *launch.json* del proyecto, situado dentro la carpeta *.vscode*.

Lo primero que tenemos que hacer es generar dicho fichero, si no existe. Para ello, teniendo abierto en el editor el fichero *hello.c*, pincharemos con el botón derecho del ratón dentro del código y seleccionaremos la opción *Agregar configuración de depuración*, en el menú flotante que aparece. Al hacerlo, se creará una carpeta llamada *.vscode*, dentro de la carpeta del proyecto y, dentro de ella, los ficheros *launch.json* y *tasks.json*.

Como se ha comentado a lo largo del curso, el fichero *tasks.json* es el que le dice a VSCode cómo compilar el proyecto. Por su parte, el fichero *launch.json* es el que le dice a VSCode cómo ejecutar el programa. Este último es el que tendremos que personalizar para que, al ejecutarse el programa, reciba determinados parámetros.

Haz doble click con el ratón en el fichero *launch.json*, para abrirlo en el editor. Hay que añadir los parámetros en la línea *args*. Nosotros hemos añadido dos parámetros. Cada parámetro se debe encerrar entre comillas, hay que separarlos por comas y ponerlos todos dentro de los corchetes de la línea *args*, que quedará así:

```
"args": ["planeta", "Tierra"],
```

En la figura siguiente se puede ver el fichero *launch.json*, tras la modificación:

```
{ launch.json x
.vscode > {} launch.json > Launch Targets > {} C/C++: gcc build and debug active file
1  {
2      "configurations": [
3          {
4              "name": "C/C++: gcc build and debug active file",
5              "type": "cppdbg",
6              "request": "launch",
7              "program": "${fileDirname}/${fileBasenameNoExtension}",
8              "args": ["planeta", "Tierra"],
9              "stopAtEntry": false,
10             "cwd": "${fileDirname}",
11             "environment": [],
12             "externalConsole": false,
13             "MIMode": "gdb",
14             "setupCommands": [
15                 {
16                     "description": "Enable pretty-printing for gdb",
17                     "text": "-enable-pretty-printing",
18                     "ignoreFailures": true
19                 },
20             ]
21         }
22     ]
23 }
```

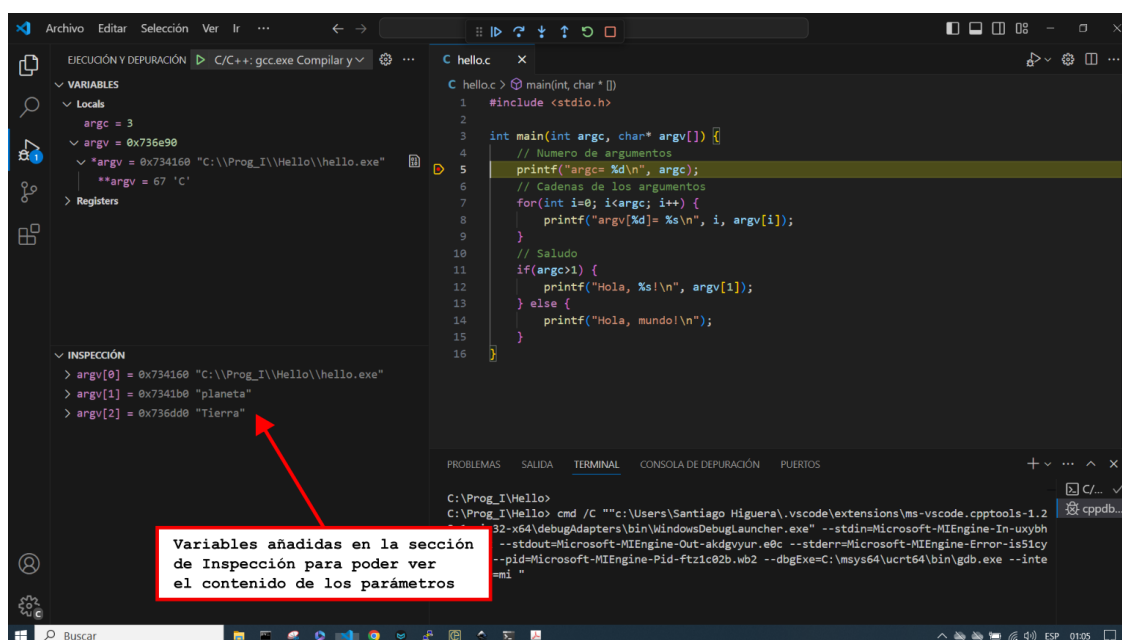
Esta es la línea en la que hay que añadir los parámetros

Ahora, cuando ejecutemos el programa desde dentro de VSCode, los parámetros que hayamos puesto en *launch.json* se le pasarán al programa.

Cuando se quiere utilizar un programa al que hay que pasar parámetros, suele ser más cómodo ejecutarlo directamente desde el terminal, tecleando el nombre del programa y los parámetros que se le quieran pasar. Modificar el fichero *launch.json* entre una ejecución y otra para cambiar los parámetros, puede ser un poco engorroso.

Sin embargo, mientras se está desarrollando el programa, puede ser cómodo fijar unos parámetros en *launch.json* que se le pasen al programa cada vez que se ejecute. Esto permitirá probar el programa, sin salir de VSCode, para ir corrigiendo los errores que hubiera o para irlo perfeccionando.

También es obligado hacerlo así si lo que queremos es lanzar el programa en el depurador. En la figura siguiente se muestra una ejecución del programa anterior, dentro del depurador. Para ello, se ha puesto un punto de ruptura en la primera línea de *main()* y se ha pulsado la tecla *F5*, que es el atajo de teclado correspondiente a la opción de menú *Ejecutar -> Iniciar depuración*:



Observa que, en la sección de *variables locales*, no se accede bien al valor de los parámetros recibidos por el programa. Por ello, hemos añadido 3 variables en la sección de *Inspección* para poder acceder de manera individualizada al contenido de los parámetros enviados al programa.

2 CONVERSIÓN DE CADENAS DE CARACTERES A VALORES NUMÉRICOS

Algunas aplicaciones necesitan interpretar los parámetros como números, no como cadenas de caracteres. Para ello, se pueden utilizar las funciones que proporciona la librería *stdlib.h*. Dos de estas funciones son las siguientes:

- **atoi():** (*ascii to integer*) recibe una cadena de texto como argumento y devuelve su interpretación como número entero.
- **atof():** (*ascii to float*) recibe una cadena de caracteres como argumento y devuelve su interpretación como número en coma flotante.

Es muy importante entender que una cadena de caracteres es diferente de un número, incluso cuando la cadena de caracteres solo tiene números y no letras. Por ejemplo, la cadena "30" no es un número, es un cadena con los caracteres '3' y '0'.

El siguiente código puede servir para comprender la función `atoi()`.

Ejemplo 2 Ejemplo de `atoi()`

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

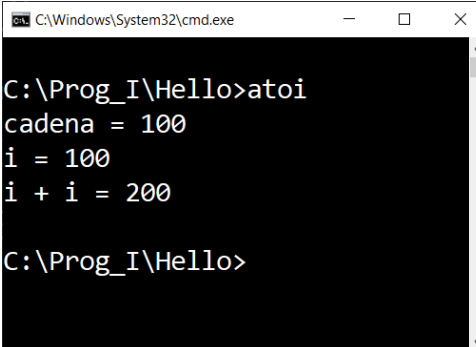
int main(void){

    char cadena[]="100";
    int i = atoi(cadena);

    printf("cadena = %s \n", cadena);
    printf("i = %d \n", i);

    i = i + i;
    printf("i + i = %d\n", i);

    return 0;
}
```



```
C:\Windows\System32\cmd.exe
C:\Prog_I\Hello>atoi
cadena = 100
i = 100
i + i = 200
C:\Prog_I\Hello>
```

3 EL PROGRAMA CALCULISTA

Vamos a desarrollar un programa un poco más complejo que sirva para afianzar los conceptos estudiados acerca del paso de parámetros a los programas y la conversión de cadenas de caracteres a valores numéricos. El programa es de complejidad media y servirá también para profundizar en el nivel de programación que se pide a los alumnos en el curso.

Para mostrar su funcionamiento, vamos a desarrollar una aplicación que opere como una calculadora que permite sumar o multiplicar la lista de números enteros que recibe entre los parámetros. El programa va a utilizar la sintaxis habitual de Lisp¹: el primer parámetro será el operador, que puede ser el signo + o el signo × y, a continuación, recibirá un número indeterminado de valores enteros. El programa calculará e imprimirá el resultado de la suma o el producto de los números que reciba.

Las siguientes podrían ser algunas ejecuciones del programa calculista:

calculista + 1 2 ⇒ El programa imprime 3

calculista + 1 2 3 4 ⇒ El programa imprime 10

calculista × 1 2 ⇒ El programa imprime 2

calculista × 1 2 3 ⇒ El programa imprime 6

El código del programa podría ser el que aparece en el Ejemplo 3: consta de una función `main()` y cuatro funciones auxiliares:

- `print_instrucciones()`: esta función se llama cuando el programa detecta que los parámetros no son correctos.
- `construye_array_numeros()`: construye un array de números a partir de las cadenas de caracteres de los operandos de la llamada al programa. Ahí es donde se utiliza la función `atoi()`.

¹Lisp es el acrónimo de *List Processing* y es el nombre de un lenguaje de programación funcional. En Lisp, las operaciones se indican poniendo el signo del operador seguido de la lista con los operandos. Históricamente, se considera que *Lisp*, junto con *Fortran* y *Cobol*, fueron los primeros lenguajes de alto nivel que se inventaron

- *calcula_suma()* y *calcula_producto()* utilizan el array de números de los operandos para calcular su suma o su producto.

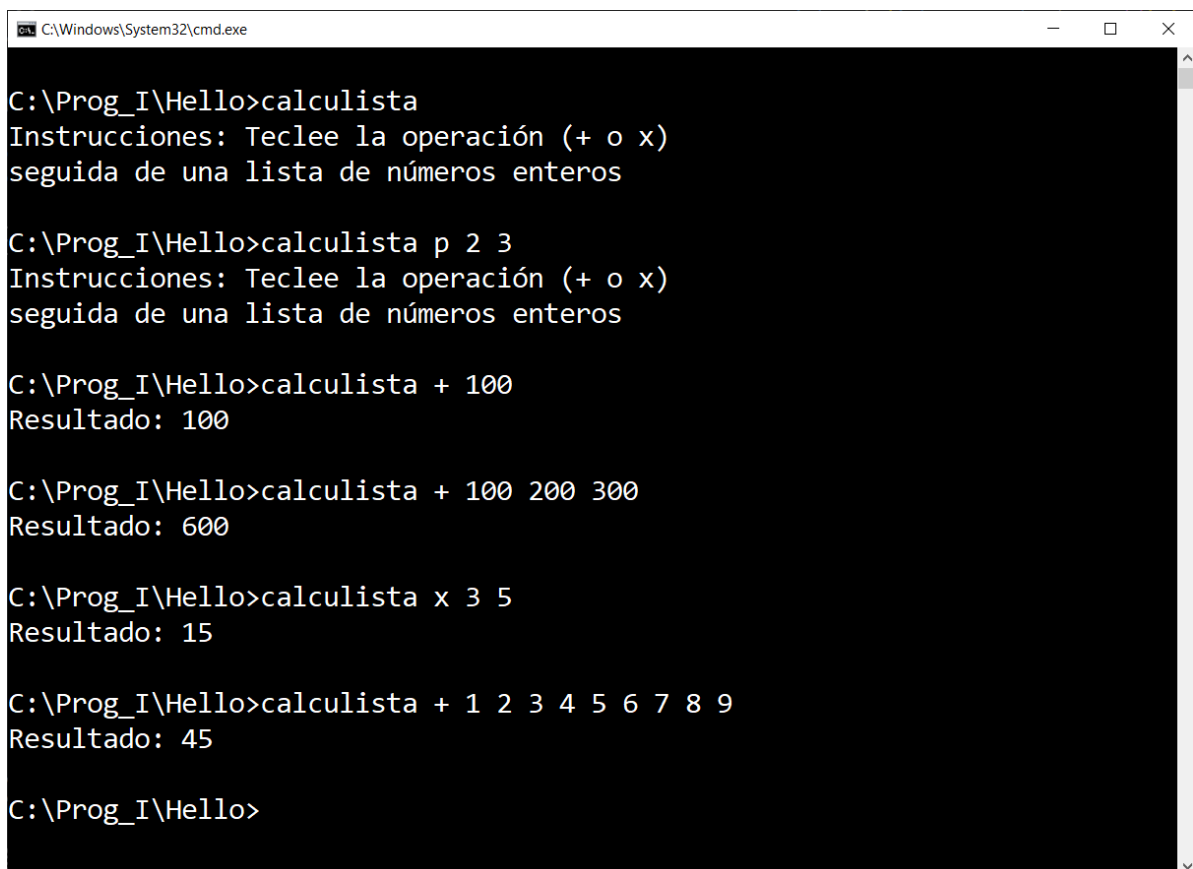
En la función *main()*, lo primero que se hace es comprobar que los parámetros son correctos. Se hacen tres comprobaciones en la misma condición del *if*:

- La primera es que *argc* sea al menos 3: nombre del programa, operador y 1 operando.
- La segunda comprobación es que *argc* no sea mayor que *MAX_OPERANDOS+2*, para evitar que el número de operandos sea mayor que el tamaño del array destinado para guardarlos.
- La tercera comprobación es que no se dé la circunstancia de que el operador no sea ni “+” ni “x”.

Si los parámetros no son correctos, se llama a la función *imprime_instrucciones()* y el programa termina. Si son correctos, se prosigue la ejecución construyendo el array numérico con los operandos a sumar o multiplicar y, una vez que se dispone del array con los operandos, se bifurca a calcular su suma o multiplicación y se imprime el resultado.

Es importante entender todo el programa. En particular, piensa un momento en lo difícil que habría sido resolver el tratamiento de los parámetros si los símbolos de los operadores se hubieran intercalado entre cada dos operandos, como se hace en la notación algebraica, en lugar de utilizar la notación tipo *Lisp* de poner primero el operador y luego los operandos.

El código del programa es el del Ejemplo 3. En la siguiente figura se puede ver la salida por pantalla de algunas ejecuciones del programa *calculista*.



```
C:\Windows\System32\cmd.exe
C:\Prog_I\Hello>calculista
Instrucciones: Teclee la operación (+ o x)
seguida de una lista de números enteros

C:\Prog_I\Hello>calculista p 2 3
Instrucciones: Teclee la operación (+ o x)
seguida de una lista de números enteros

C:\Prog_I\Hello>calculista + 100
Resultado: 100

C:\Prog_I\Hello>calculista + 100 200 300
Resultado: 600

C:\Prog_I\Hello>calculista x 3 5
Resultado: 15

C:\Prog_I\Hello>calculista + 1 2 3 4 5 6 7 8 9
Resultado: 45

C:\Prog_I\Hello>
```

Ejemplo 3 Código de *calculista.c*

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

#define MAX_OPERANDOS 20

void print_instrucciones();
void construye_array_numeros(int* numeros, int num_operandos, char* argv[]);
int calcula_suma(const int* operandos, int num_operandos);
int calcula_producto(const int* operandos, int num_operandos);

int main(int argc, char* argv[]) {
    // Inicialización de variables
    int operandos[MAX_OPERANDOS] = {0};
    int num_operandos;
    int resultado;

    // Comprobación parámetros correctos
    if(argc<3 || argc > MAX_OPERANDOS+2 ||
       ( (strcmp(argv[1], "+")!=0) && (strcmp(argv[1], "x")!=0) ) ) {
        // Los parámetros no son correctos,
        // se imprimen las instrucciones y acaba el programa
        print_instrucciones();
    } else {
        // Preparación array de operandos
        num_operandos = argc-2;
        construye_array_numeros(operandos, num_operandos, argv);
        // Cálculo de la operación
        if(strcmp(argv[1], "+")==0) {
            resultado = calcula_suma(operandos, num_operandos);
        } else {
            resultado = calcula_producto(operandos, num_operandos);
        }
        // Salida resultados
        printf("Resultado: %d\n\n", resultado);
    }
}

void print_instrucciones() {
    printf("Instrucciones:\n");
    printf("    Teclee la operación (+ o x)\n");
    printf("    seguida de una lista de números enteros\n");
    printf("    (Número máximo de operandos %d)\n\n", MAX_OPERANDOS);
}

void construye_array_numeros(int* numeros, int num_operandos, char* argv[]) {
    for(int i=0; i<num_operandos; i++) {
        numeros[i] = atoi(argv[2+i]);
    }
}

int calcula_suma(const int* operandos, int num_operandos) {
    int suma = 0;
    for(int i=0; i< num_operandos; i++) {
        suma = suma + operandos[i];
    }
    return suma;
}

int calcula_producto(const int* operandos, int num_operandos) {
    int prod = 1;
    for(int i=0; i<num_operandos; i++) {
        prod = prod * operandos[i];
    }
    return prod;
}
```