

# Sesión 2: Ejecutando C

## Hoja de problemas


Programación para Sistemas


Ángel Herranz


aherranz@fi.upm.es


Universidad Politécnica de Madrid


2019-2020

 **Ejercicio 1.** Repasa las transparencias de clase. Observa los detalles, especialmente los relacionados con GDB.

 **Ejercicio 2.** Busca en internet GDB, explora.

 **Ejercicio 3.** Experimenta con `argc` y `argv`. Por ejemplo, puedes escribir un programa que diga cuál es el valor de `argc`, supongamos que  $n$  y que luego vaya imprimiendo los valores de `argv[0]`, `argv[1]`, `argv[2]`, ..., `argv[n - 1]`. ¿Qué ocurre si intentas imprimir también el dato `argv[argc]`? ¿Y `argv[argc+1]`? ¿Y `argv[argc+2]`? ¿Y ...?

 **Ejercicio 4.** Corrige los bugs del programa `fact.c` de las transparencias y haz una ejecución paso a paso con GDB sin entrar en las funciones de biblioteca y mirando cómo cambian las variables en cada paso.

 **Ejercicio 5.** Cuando un programa termina su ejecución, además de todo lo que imprime en la salida estándar o en la salida de error, nos ofrece otra información: el código de terminación. El código de terminación de un programa C es el valor que devuelve su función `main`.

Para saber el valor que ha devuelto el último programa ejecutado desde Bash podemos usar el mandato **echo** `$?`. Prueba lo siguiente:

```
$ ls hola.c
hola.c
$ echo $?
0
$ ls supercalifragilisticoespialidoso.c
ls: cannot access 'supercalifragilisticoespialidoso.c': No such file or directory
$ echo $?
2
```

En este ejercicio tienes que inspeccionar el código de terminación de tu programa `fact.c`.

- » **Ejercicio 6.** En C se pueden definir macros apoyándonos en el preprocesador. Una de las directivas del preprocesador de C más usada es **#define**. Dicha directiva sustituye en tiempo de compilación<sup>1</sup> un nombre por una expresión.

Prueba el siguiente código (puedes llamar al fichero `prueba_define.c`):

```
#include <stdio.h>
#define N 5
int main() {
    int x = N;
    printf("El valor de x es %i\n", x);
    return 0;
}
```

Puedes ver el efecto de **#define** ejecutando `gcc -E prueba_define.c`

- » **Ejercicio 7.** Parece que la directiva **#define** sirve para definir constantes pero... Consideremos este código:

```
double x = 18.0 / squared( 2 + 1 );
```

¿Cuál será el valor de `x` para cada una de las siguientes macros?

- **#define** squared(x) x\*x
- **#define** squared(x) (x\*x)
- **#define** squared(x) (x)\*(x)
- **#define** squared(x) ((x)\*(x))

- **Ejercicio 8.** Escribe un programa para comprobar tu respuesta a la última pregunta.
- **Ejercicio 9.** Invoca el preprocesador (`gcc -E MI_PROGRAMA`) para ver el efecto que tiene usar la directiva **#define**. Evita usar la directiva **#include**: *cuanto menos bulto más claridad*.
- » **Ejercicio 10.** Durante la clase hemos visto varias declaraciones de variables, todas ellas de tipo entero (**int**). En este ejercicio deberás ir al mítico libro *The C Programming Language* de Brian W. Kernighan y Dennis M. Ritchie y explorar qué otros tipos hay, declara varias variables, inicialízalas e imprime sus valores con `printf`.
- » **Ejercicio 11.** Siguiendo con el libro, durante la clase hemos visto varios usos de la función `printf`. En este ejercicio deberás explorar las conversiones básicas que es capaz de realizar la función: `%d` y `%i` para enteros en notación decimal, `%x` para enteros en notación hexadecimal, `%s` para strings, etc.
- 🔊 **Ejercicio 12.** Realiza el siguiente tutorial de GDB de Andrew Gilpin (profesor de la universidad Carnegie Mellon):

<https://www.cs.cmu.edu/~gilpin/tutorial/>

---

<sup>1</sup>Es decir, antes de ejecutar.