# Sesión 2: Conceptos básicos de programación (4h)

Esta sesión de laboratorio es una introducción al lenguaje de programación C. Mostraremos la estructura básica de un programa en C y describiremos todos sus componentes. Además, trabajaremos de forma práctica con los siguientes conceptos: variables, constantes, tipos de datos elementales, operadores aritméticos y lógicos, precedencia de operadores, funciones `printf` y `scanf`, sentencias condicionales `if-else` y `switch`.
El objetivo de esta práctica de laboratorio es que el alumno se familiarice con el entorno de programación y comience a realizar programas sencillos en C.

## *La estructura de un programa en C*

La estructura general simplificada de un programa en C es la siguiente:

```
/* Incluir librerías */
#include <nom_lib.h>
#include "nom_lib.h"

/* Definir constantes */
#define NOM_CONS valor_cons

/* Programa principal */
main()
{
   /* Declaración de variables */
   tipo_var nom_var;

   /* Sentencias */
}
```

En primer lugar debemos saber que en el lenguaje C, todo el texto escrito entre los símbolos `/*` y `*/` es un comentario y, por tanto, es ignorado por el compilador (no forma parte del programa).
Podemos diferenciar tres componentes en la estructura básica de un programa: incluir librerías, definir constantes y el programa principal.

### Incluir librerías

Los compiladores de C proporcionan librerías de funciones. Cada librería tiene asociada un fichero de definición que se llama *cabecera* (*header* en inglés). Para utilizar cualquier función de la librería estándar de C, debemos incluir en nuestro programa el fichero *cabecera* correspondiente de la siguiente manera:

```
#include <nom_lib.h>
```

donde `nom_lib.h` es el fichero *cabecera* de la librería que se desea utilizar. Por ejemplo: `stdio.h` para la librería estándar de entrada/salida, `string.h` para la librería que manipula cadena de caracteres, `math.h` para la librería matemática, etc.

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

También podemos tener nuestras propias librerías. En este caso debemos tener un fichero *cabecera* (con extensión `.h`) y un fichero de código (con extensión `.c`) que contiene la implementación de las funciones. Para utilizar las funciones de nuestra propia librería, debemos incluir el fichero *cabecera* en el programa de la siguiente manera:

```
#include "nom_lib.h"
```

Nótese que en lugar de los símbolos < y > se usan las dobles comillas (`""`) y que no se incluye el fichero con extensión `.c`. En este caso, el fichero cabecera *`nom_lib`*`.h` y el fichero de código *`nom_lib`*`.c` deben estar almacenados en el mismo directorio que el programa que lo incluye.
Por ejemplo, si tuviéramos una librería propia con funciones que permiten escribir en colores por pantalla, entonces debemos tener dos ficheros: `colours.h` (la cabecera) y `colours.c` (fichero código con la implementación de las funciones). Si queremos escribir un programa que usa esta librería, debemos incluirla, escribiendo:

```
#include "colours.h"
```

y al compilar el programa, debemos hacerlo junto con el fichero `colours.c`.

### Definir constantes

Una constante en C es un identificador que representa un valor que no puede ser modificado durante la ejecución del programa. Para definir una constante, debemos escribir:

```
#define NOM_CONS valor_cons
```

Donde *`NOM_CONS`* es el nombre de la constate y *`valor_cons`* el valor que tiene. El nombre de la constante lo escribiremos siempre en mayúsculas. A continuación mostramos algunos ejemplos típicos:

```
#define PI 3.1415
#define CIERTO 1
#define FALSO 0
#define SI 'S'
```

### Programa principal

Todos los programas tienen una función principal llamada `main` que indica el punto inicial de entrada. Es la función que se ejecuta al inicio. Dentro del programa principal se declaran las variables y se indica el orden de ejecución de las sentencias o acciones.
El programa principal está englobado entre llaves.

```
void main()
 {
   /* Declaración de variables */

   /* Sentencias */
 }
```

*Declaración de variables:* Una variables es un identificador que guarda un único valor que puede ser consultado y modificado durante la ejecución de un programa. Se pueden declarar de las siguientes maneras:

```
tipo_var nom_var1;
tipo_var nom_var2, nom_var3;
tipo_var nom_var4=valor_inicial;
```

donde *tipo_var* indica el tipo de dato de la variable, *nom_varX* (con X=1,2,3,4) el nombre de esta y *valor_inicial* el valor que le damos a la variable al inicio del programa. De momento trabajaremos únicamente con 3 tipos de datos: `int` (para almacenar un valor entero), `char` (para almacenar un carácter) y `float` (para almacenar un valor real). El nombre de la variable siempre lo escribiremos en minúsculas.
A continuación se muestran algunos ejemplos:

```
int valor;          /* declara valor como entero, sin inicializar */
char letra='F';     /* declara letra como carácter, inicializada a
                       'F' */
float area1,area2;  /* declara area1 y area2 como reales, sin
                       inicializar */
```

*Sentencias:* Una sentencia es una acción. De momento vamos a introducir únicamente tres tipos de sentencias: asignación, llamada a la función `printf` y llamada a la función `scanf`.

(a) La sentencia de asignación asigna un valor a una variable:

```
nom_var=expresion;  /* asigna a la variable nom_var el resultado
                       de la expresión */
```

Algunos ejemplos de sentencias de asignación son los siguientes:

```
valor=8*4;   /* asigna a la variable valor el valor 32 */
letra='M';   /* asigna a la variable letra el carácter 'M' */
area1=PI*valor*valor; /* asigna a la variable area1 el resultado
                         de la expresión PI*valor*valor, es
                         decir, 3215.36 (3.14*32*32)
```

(b) La llamada a la función `printf` nos permitirá mostrar mensajes por pantalla. Para usar esta función, debemos escribir:

```
printf("Texto que queremos mostrar por pantalla", param1, …);
```

Entre comillas dobles se indica el texto que se mostrará por pantalla y a continuación, separados por comas, las variables o expresiones (*param1,…*) cuyos valores también se desean mostrar por pantalla. El texto puede incluir los especificadores de formato `%d` (para valores de tipo `int`), `%c` (para valores de tipo `char`) o `%f` (para valores de tipo `float`) dentro de las comillas dobles. El `printf` sustituye cada especificador de formato por el valor de la variable o expresión correspondiente (*param1,…*).

Algunos ejemplos de uso de la función `printf` son los siguientes:

```
printf("Buenos dias\n");
printf("Variable valor: %d, Variable letra: %c\n", valor, letra);
printf("El area de la circunferencia es %.2f\n", area1);
```

Algunas consideraciones en el uso del `printf`:

- **No utilice acentos ni letras especiales** (`ñ`, `ç`, etc) en el texto que se quiere mostrar en pantalla.
- El carácter especial `'\n'` indica salto de línea. En pantalla veremos que el cursor se sitúa en la siguiente línea.
- Otros caracteres especiales son: `'\t'` (tabulador), `'\\'` (para mostrar la `'\'` en pantalla) y `'%%'` (para mostrar el carácter `'%'` en pantalla).
- Para mostrar por pantalla el valor de una variable o expresión debemos usar el especificador de formato `%d` (si el valor es de tipo `int`), `%c` (si es de tipo `char`) o `%f` (si es de tipo `float`) dentro de las comillas dobles y añadir como parámetro la variable o expresión asociada al valor que se desea mostrar por pantalla. El `printf` sustituirá el especificador de formato por el valor de la variable o expresión correspondiente. Podemos tener varios especificadores de formato en un único `printf`:

  ```
  printf("area1: %f \t area2: %f \t Suma: %.2f\n", area1,
          area2, area1+area2);
  ```

- El especificador `%f` muestra un valor real con seis decimales. Si queremos mostrar menos decimales podemos usar `%.2f` (2 decimales), `%.3f` (3 decimales), etc.

(c) La llamada a la función `scanf` nos permitirá leer datos introducidos desde teclado por el usuario del programa. Para usar esta función en su versión más simple, debemos escribir:

```
scanf("especificador_de_formato", &nom_var);
```

donde el *`especificador_de_formato`* es `%d` (si queremos leer un valor entero), `%c` (si queremos leer un carácter) o `%f` (si queremos leer un valor real) y *`nom_var`* es el nombre de la variable donde se guardará el valor leído del teclado. Obviamente, esta variable debe ser declarada del tipo correspondiente al inicio del programa.
Algunos ejemplos de uso de la función `scanf` son los siguientes:

```
scanf("%d", &valor);
scanf("%f %f%", &area1, &area2);
```

Algunas consideraciones en el uso del `scanf`:

- La función `scanf` no muestra nada por pantalla. Si queremos pedir al usuario que introduzca un valor entero debemos hacerlo con dos sentencias:

  ```
  printf("Introduzca un valor entero:");
  scanf("%d", &valor);
  ```

- Los especificadores de formato `%d` y `%f` ignoran los caracteres espacios, tabuladores y/o saltos de líneas que puedan haber delante del número. Sin embargo, el especificador de formato `%c` no los ignora. Por ejemplo, si ejecutamos el siguiente código:

```
printf("Introduzca un valor entero y otro real: ");
scanf("%d%f", &valor, &area1);
printf("Introduzca un caracter: ");
scanf("%c", &letra);
```

y la entrada que introduce el usuario (marcada en negrita) es:

```
Introduzca un valor entero y otro real: 15 3.6
Introduzca un caracter:
```

en la variable `valor` quedará almacenado el valor `15` y en `area1` el `3.6`. Sin embargo, el programa no esperará a que el usuario introduzca un carácter. En su lugar, en la variable `letra` quedará almacenado un `'\n'` (salto de línea) y el programa finalizará. Esto se debe a que después de introducir el valor `3.6`, el usuario ha pulsado la tecla <ENTER> o salto línea (que en pantalla no se visualiza). Cuando se ejecuta la sentencia `scanf("%c", &letra)` se lee el carácter `'\n'` (salto de línea) de la entrada y se almacena en `letra`.

- Para solventar el problema anterior, se usa el **especificador de formato `%*c`** que **lee del teclado un carácter y lo descarta**, es decir, no lo almacena en ninguna variable. Este especificador de formato se usa para el leer el salto de línea (`'\n'`) que pulsamos después de introducir los datos. El código correcto del ejemplo anterior es:

```
printf("Introduzca un valor entero y otro real: ");
scanf("%d%f%*c", &valor, &area1);
printf("Introduzca un caracter: ");
scanf("%c%*c", &letra);
```

Si ahora la entrada que introduce el usuario (marcada en negrita) es:

```
Introduzca un valor entero y otro real: 15 3.6
Introduzca un caracter: H
```

en la variable `valor` quedará almacenado el valor `15`, en `area1` el `3.6` y en `letra` el carácter `'H'`. El especificador de formato `%*c` lee el salto de línea (`'\n'`) y lo descarta. De esta manera, cuando se ejecuta la sentencia `scanf("%c%*c", &letra)`, el programa quedará a la espera de que el usuario introduzca un carácter, que posteriormente leerá y almacenará en la variable `letra`.

- La función `scanf` espera del teclado el formato exacto especificado entre las comillas dobles. Si los datos no se introducen tal y como se indica, la función `scanf` deja de leer del teclado y finaliza su ejecución, pero no la ejecución del programa. Por ejemplo, si tenemos el siguiente código:

```
int val1, val2;

printf("Introduzca dos valores enteros:");
scanf("%d-%d%*c", &val1, &val2);
```

y el usuario introduce desde el teclado `7,8` (en lugar de `7-8`), el programa lee el valor `7` y lo almacena en la variable `val1`; sin embargo el valor `8` no lo lee porque el usuario no ha respetado el formato de entrada especificado en el primer parámetro ( `%d-%d%*c`). Después de leer el `7`, la función `scanf` finaliza su ejecución y el programa continuará ejecutándose con la variable `val1` valiendo `7` y la variable `val2` almacenando un valor indeterminado.

## *Nuestro primer programa en C*

Ahora que ya conocemos la estructura básica de un programa, podemos escribir nuestro primer programa en C que muestra por pantalla un mensaje de texto.

```
#include <stdio.h>

void main()
 {
    printf("Has conseguido hacer tu primer programa en C.\n");
 }
```

Algunas consideraciones sobre la escritura de los programas en C (normas de estilo):

- Las llave (`{}`) del `main` se abren y cierran en una línea exclusiva para ellas y están alineadas en la misma columna que el carácter `m` de `main`.
- Todas las líneas de código dentro del `main` (declaración de variables y sentencias) tienen un sangrado o indentación de 2 a 4 espacios respecto a las llaves del `main`.
- Entre la declaración de variables y el conjunto de sentencias hay siempre una línea en blanco.

## *Desarrollo de la práctica (3h)*

### Ejercicio 1:

Suponiendo que tenemos la siguiente declaración de variables:

```
int k=7, resi;
char c=´a´, resc;
float f=5.5, g=-3.25, resf;
```

Calcule mentalmente el valor de las variables `resi`, `resc` y `resf` tras evaluar cada una de las siguientes sentencias de asignación y anote su valor en el espacio facilitado.

Tenga en cuenta que:

- el resultado de la división es entero si el dividendo y el divisor son enteros.
- el resultado de la división es real si el dividendo y/o el divisor son reales.
- el operador `%` (módulo) devuelve el resto después de dividir el primer operando entre el segundo (el resultado de `5%3` es `2`, y de `4%2` es `0`).

- El orden de las operaciones, al igual que en matemáticas, es de izquierda a derecha, evaluando primero los términos entre paréntesis, luego las multiplicaciones, divisiones y módulos y finalmente sumas y restas.

```
resi = -2  +  k;        /* la variable resi vale _______ */
resi = resi + 2;        /* la variable resi vale _______ */
resc = c;               /* la variable resc vale _______ */
resi = k%5;             /* la variable resi vale _______ */
resf = (f - g)/2;       /* la variable resf vale _______ */
resi = resi*(k - 3);    /* la variable resi vale _______ */
resf = k/(resi - 2);    /* la variable resf vale _______ */
resf = f/(resi - 2);    /* la variable resf vale _______ */
resi = 2*(k - 3)%3/2;   /* la variable resi vale _______ */
```

Ahora compruebe si los resultados son correctos escribiendo un programa (en un fichero llamado *sesion2_ej1.c*) que incluya la declaración de variables y las sentencias anteriores y que muestre por pantalla el valor de la variable `resi`, `resc` o `resf` (según corresponda) después de ejecutarse cada una de las sentencias anteriores.

## Ejercicio 2:

La siguiente tabla muestra la prioridad y asociatividad de los operadores en el lenguaje C:

| Nivel | Operador | Símbolo | Asociatividad |
|---|---|---|---|
| **1** | Paréntesis | () | De izquierda a derecha |
| **2** | Negación lógica<br>Signo negativo | !<br>- | De derecha a izquierda |
| **3** | Multiplicación<br>División<br>Módulo | *<br>/<br>% | De izquierda a derecha |
| **4** | Suma<br>Resta | +<br>- | De izquierda a derecha |
| **5** | Menor que<br>Menor que o igual a<br>Mayor que<br>Mayor que o igual a | <<br><=<br>><br>>= | De izquierda a derecha |
| **6** | Igual a<br>Distinto que | ==<br>!= | De izquierda a derecha |
| **7** | Conjunción | && | De izquierda a derecha |
| **8** | Disyunción | \|\| | De izquierda a derecha |

(Tabla extraída del Cap. 2 del libro "Fundamentos de ordenadores: programación en C" de M. Jiménez y B. Otero )

**Nota**: La prioridad de los operadores va disminuyendo al ir descendiendo en la tabla. Por lo tanto, cuanto menor es el nivel de un operador, mayor es su prioridad. Los operadores de un mismo nivel tienen el mismo orden de prioridad entre ellos y, por tanto, han de resolverse considerando el orden de asociatividad que tenga el nivel del operador.
Evalúe mentalmente las expresiones de la siguiente tabla y anote el resultado en la columna derecha de la tabla en valor numérico. Tenga en cuenta que:

- Todos los operandos son de tipo entero.
- El valor de una expresión lógica es 0 si es falsa y 1 si es cierta.
- En C, cualquier expresión cuyo resultado sea 0, se interpreta como falsa y cualquier expresión cuyo resultado sea distinto de 0 se interpreta como cierta.

| Expresión | Resultado de la expresión |
|---|---|
| `5/2 + 20%6` | |
| `4*6/2 - 15/2` | |
| `5*15/2/(4 - 2)` | |
| `8 == 16 \|\| 7 != 4 && 4 < 1` | |
| `(4*3 < 6 \|\| 3 > 5 - 2) && 3 + 2 < 12` | |
| `2 \|\| 0` | |
| `1 && 0` | |

Ahora compruebe si los resultados son correctos escribiendo un programa (en un fichero llamado *sesion2_ej2.c*) que asigne a una variable de tipo `int` las expresiones anteriores y muestre el valor por pantalla después de cada asignación.

## Ejercicio 3:

a) Escriba un programa en C (en un fichero llamado *sesion2_ej3.c*) que muestre por pantalla el mensaje "Hola".

b) Modifique el programa anterior para que primero pida al usuario que introduzca las tres iniciales de su nombre, las lea del teclado y finalmente muestre por pantalla el mensaje: "Hola *XXX*", donde *XXX* son las tres iniciales introducidas por el usuario.
Ejemplo de ejecución (en negrita los datos introducidos por el usuario):

```
Introduce tus tres iniciales: MHG
Hola MHG
```

**Pista**: Se debe declarar tres variables de tipo `char` para almacenar las tres iniciales (`char ini1, ini2, ini3;`).

c) Añada al programa anterior el código necesario para que pregunte al usuario por su año de nacimiento y muestre por pantalla los años que tendrá el 31/DIC/2015.
Ejemplo de ejecución (en negrita los datos introducidos por el usuario):

```
Introduce tus tres iniciales: MHG
Introduce tu anyo de nacimiento: 1970

Hola MHG
El 31/DIC/2015 tendras 45 años
```

d) Ahora modifique el programa anterior para que pregunte primero por el año de nacimiento y luego por las tres iniciales.
Ejemplo de ejecución (en negrita los datos introducidos por el usuario):

```
Introduce tu anyo de nacimiento: 1970
Introduce tus tres iniciales: MHG

Hola MHG
El 31/DIC/2015 tendras 45 años
```

e) Ahora disponemos de una librería propia compuesta por los ficheros `colours.h` (fichero cabecera) y `colours.c` (fichero de código). Esta librería nos ofrece la función `cambiar_color(color)` que cambia el color con que se escribe en la pantalla. El parámetro *`color`* puede ser: `RED`, `GREEN`, `YELLOW`, `BLUE`, `MAGENTA`, `CYAN` o `DEFAULT` (el color por defecto del terminal).
Modifique el programa anterior para que incluya esta librería y muestre las siglas del nombre en color rojo y la edad en amarillo.
Ejemplo de ejecución (en negrita los datos introducidos por el usuario):

```
Introduce tu anyo de nacimiento: 1970
Introduce tus tres iniciales: MHG

Hola MHG
El 31/DIC/2015 tendras 45 años
```

Los ficheros `colours.h` y `colours.c` se proporcionan junto al enunciado de esta práctica. Recuerde que para compilar con una librería propia debe ejecutar el siguiente comando:

***gcc –o sesion1_ej1 sesion1_ej1.c colours.c***

## Ejercicio 4:

Escriba un programa en C (fichero *sesion2_ej4.c*) que pida al usuario el radio de un círculo y muestre por pantalla su área (π*radio*radio) y su perímetro (2*π*radio). El radio, el área y el perímetro son valores de tipo `float`. Defina en el programa una constante `PI` con valor `3.14`.
Ejemplos de ejecución (en negrita los datos introducidos por el usuario):

```
Introduzca el radio del circulo: 1

Area del circulo: 3.14
Perimetro del circulo: 6.28
```

```
Introduzca el radio del circulo: 2.3

Area del circulo: 16.61
Perimetro del circulo: 14.44
```

## Ejercicio 5:

Complete el siguiente programa que encontrará en el fichero *sesion2_ej5.c*. para que los valores que almacenan las variables `a` y `b` después del `scanf` queden intercambiados.

```
#include <stdio.h>

main()
{
   int a, b, aux;

   printf("Introduzca dos valores para las variables a y b: ");
   scanf("%d %d%*c", &a, &b);

   . . .

   printf("Variable a: %d\n", a);
   printf("Variable b: %d\n", b);
}
```

Ejemplo de ejecución (en negrita los datos introducidos por el usuario):

```
Introduzca dos valores enteros para la variables a y b: 7 23

Variable a: 23
Variable b: 7
```

### Ejercicio 6:

Escriba un programa en C (fichero *sesion2_ej6.c*) que pida al usuario un valor entero que representa una hora expresada en segundos y saque por pantalla esa misma hora expresada en horas, minutos y segundos.
Ejemplos de ejecución (en negrita los datos introducidos por el usuario):

```
Introduzca la cantidad en segundos: 66

0h 1m 6s
```

```
Introduzca la cantidad en segundos: 3668

1h 1m 8s
```

### Ejercicio 7:

Escriba un programa en C que pida al usuario una expresión horaria en formato horas:minutos:segundos y muestre por pantalla la expresión horaria después de haberle sumado un segundo.

a) Escriba una primera versión del programa (en el fichero *sesion2_ej7a.c*) que solo utilice sentencias de asignación y llamadas a las funciones `scanf` y `printf`. No se pueden usar sentencias condicionales.

b) Escriba una segunda versión del programa (en el fichero *sesion2_ej7b.c*) utilizando sentencias condicionales del tipo `if-else`.

Ejemplos de ejecución (en negrita los datos introducidos por el usuario):

```
Introduzca una hora en formato hh:mm:ss : 11:33:15

11:33:16
```

```
Introduzca una hora en formato hh:mm:ss : 12:59:59

13:00:00
```

```
Introduzca una hora en formato hh:mm:ss : 23:59:59

00:00:00
```

### Ejercicio 8:

Escriba un programa en C (en el fichero *sesion2_ej8.c*) que pida al usuario la descripción de un triángulo o de un círculo y muestre por pantalla el área correspondiente. La entrada de datos por el teclado comienza con un carácter que indica el tipo de figura (`'T'` o `'t'` para el triángulo y `'C'` o `'c'` para el círculo). A continuación le sigue la descripción de la figura: Si es un triángulo, le siguen dos reales positivos que indican la base y la altura;

si es un círculo, le sigue un real positivo que indica el radio. Como valor de π, defina la constate PI como 3.1415.

Ejemplos de ejecución (en negrita los datos introducidos por el usuario):

```
Introduzca la descripcion de la figura: T 3.2 4
Introduzca la descripcion de la figura: c 3.4

Area del circulo = 36.32
```

```
Introduzca la descripcion de la figura: f 3.2

El tipo de la figura es incorrecto
```

### Ejercicio 9:

Escriba un programa en C (en el fichero *sesion2_ej9.c*) que pida al usuario que introduzca un carácter e indique si es una letra (carácter alfabético), si es una vocal, si es una consonante, si es una letra mayúscula, si es una letra minúscula y si es un dígito (carácter numérico). Además, en caso de que sea una letra se debe mostrar la misma letra con la capitalización invertida, es decir, si estaba en mayúscula mostrarla en minúsculas y viceversa.

Ejemplos de ejecución (en negrita los datos introducidos por el usuario):

```
Introduzca un caracter: J
Es una letra
No es una vocal
Es una consonante
Es una mayuscula
No es una minuscula
No es un digito
El caracter con la capitalizacion invertida es j
```

```
Introduzca una caracter: k
Es una letra
No es una vocal
Es una consonante
No es una mayuscula
Es una minuscula
No es un digito
El caracter con la capitalizacion invertida es K
```

```
Introduzca una caracter: 3
No es una letra
No es una vocal
No es una consonante
No es una mayuscula
No es una minuscula
Es un digito
```

### Ejercicio 10:

Escriba un programa en C (en el fichero *sesion2_ej10.c*) que pida al usuario que

introduzca un valor entero y muestre por pantalla el número indicado escrito en palabras si el valor introducido está entre 0 y 9. Además la palabra se mostrará en verde si el número es par y en magenta si es impar.
Si el número introducido no está entre 0 y 9, se mostrará un mensaje indicando que el número es difícil de escribir. Utilice la sentencia `switch` para la implementación.
Ejemplos de ejecución (en negrita los datos introducidos por el usuario):

```
Introduzca un valor entero entre 0 y 9: 3

Ha introducido el TRES
```

```
Introduzca un valor entero entre 0 y 9: 0

Ha introducido el CERO
```

```
Introduzca un valor entero entre 0 y 9: 23

El numero introducido es dificil de escribir
```

Recuerde que para compilar con la librería de colores debe incluir el fichero `colours.h` y ejecutar el siguiente comando:

***gcc –o sesion2_ej10 sesion2_ej10.c colours.c***