

UT 1 - Programación multiproceso

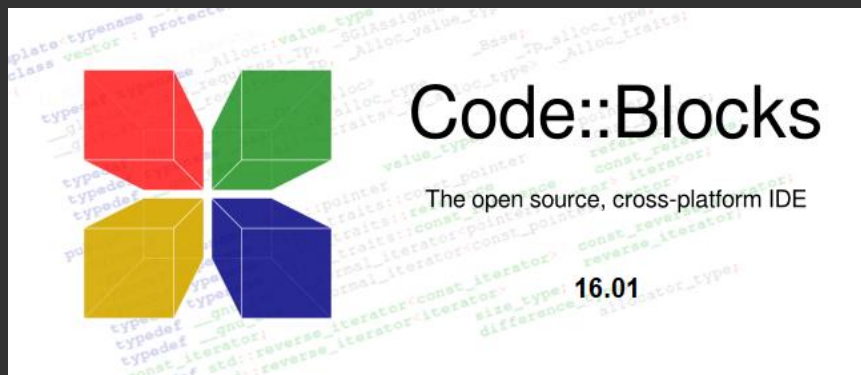
2º DAM – Curso 2024/2025

Parte 1 – Básicos de C

Andrés Marina Díaz
andresmd@educastur.org

OBJETIVOS

- Familiarización con el lenguaje de programación C
- Entorno de programación: **CodeBlocks**



Code::Blocks / Downloads / Binary releases

Binary releases

Please select a setup package depending on your platform:


- [Windows XP / Vista / 7 / 8.x / 10](#)
- [Linux 32 and 64-bit](#)
- [Mac OS X](#)

NOTE: For older OSes use older releases. There are releases for many OS version and platforms [Sourceforge.net](#) page.

NOTE: There are also more recent nightly builds available in the [forums](#) or (for Ubuntu users) in the [PPA repository](#). Please note that we consider nightly builds to be stable, usually.

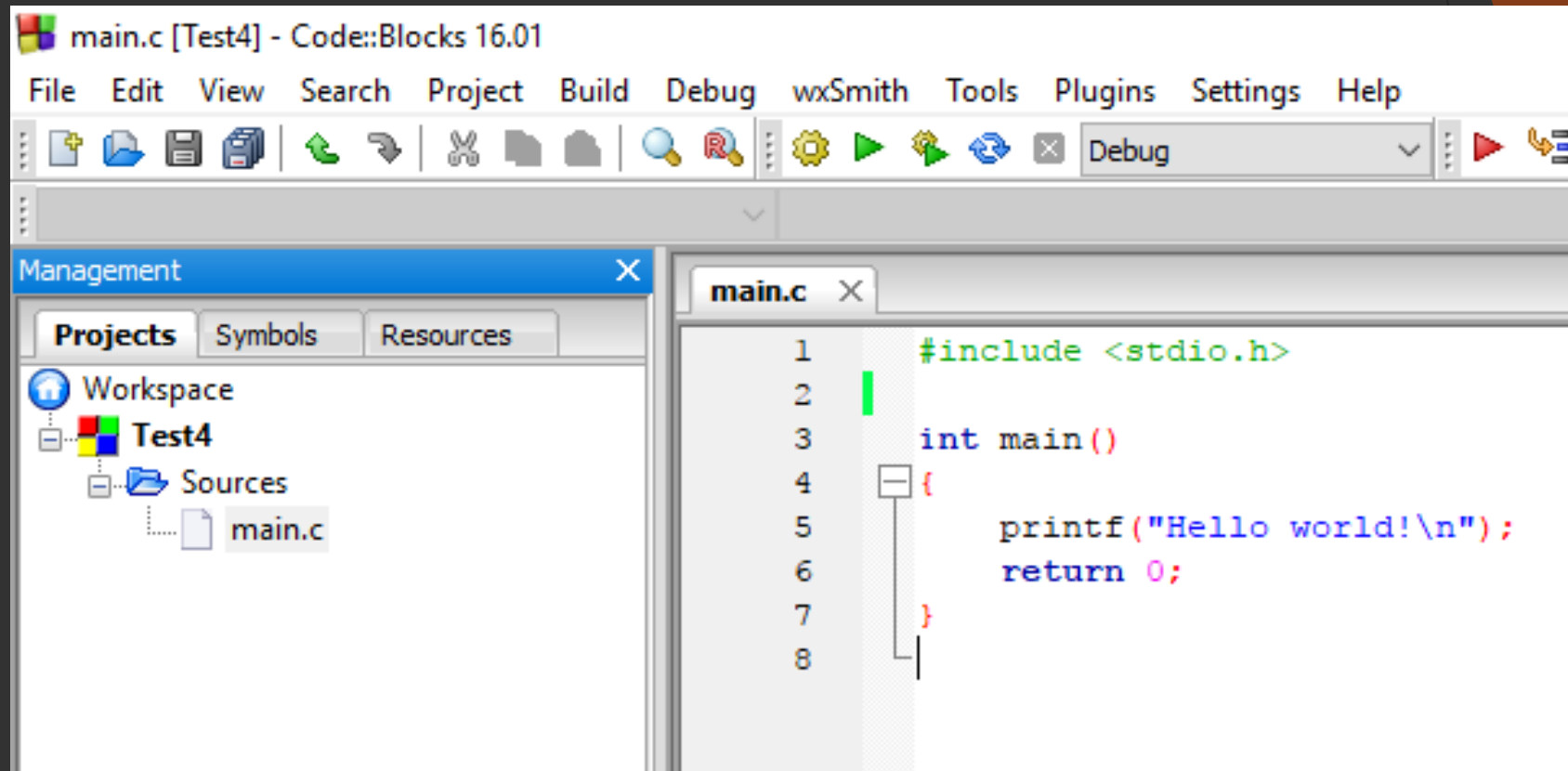
NOTE: We have a [Changelog for 20.03](#), that gives you an overview over the enhancements and fixes we have put in the new release.

NOTE: The default builds are 64 bit (starting with release 20.03). We also provide 32bit builds for convenience.

 **Microsoft Windows**

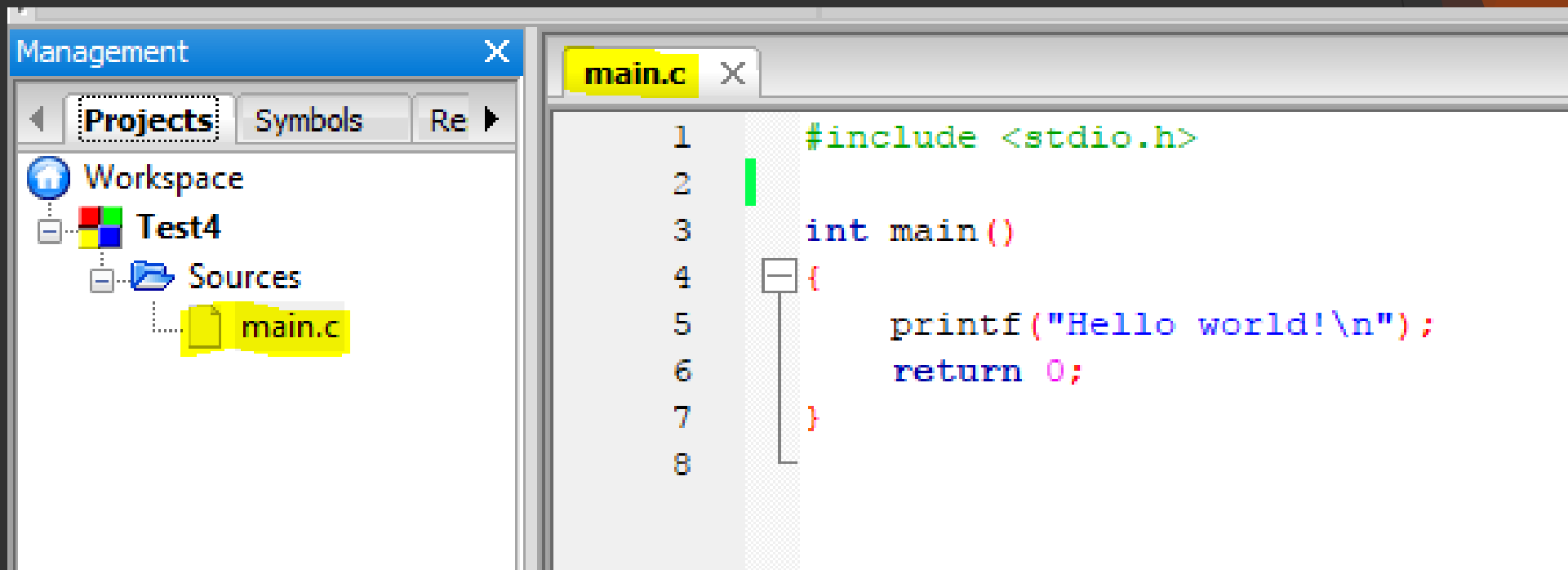
File	Download from
codeblocks-20.03-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03-setup-nonadmin.exe	FossHUB or Sourceforge.net
codeblocks-20.03-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03mingw-setup.exe	FossHUB or Sourceforge.net

HOLA MUNDO EN C



HOLA MUNDO - DESGLOSE

- Fichero .c donde tenemos nuestra función *main* (no tiene por qué llamarse main.c)

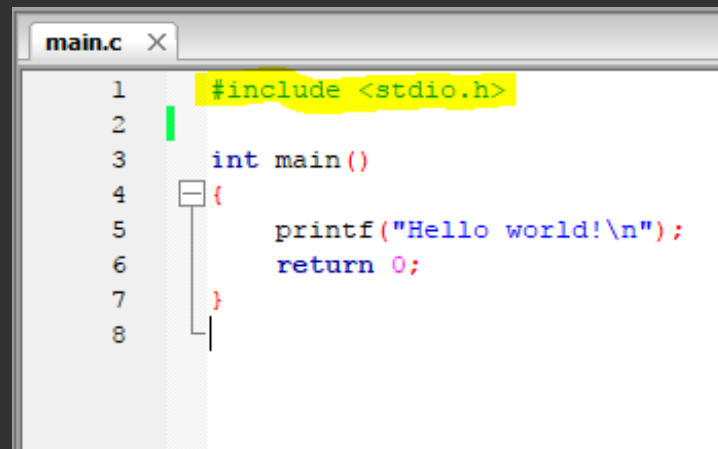


HOLA MUNDO - DESGLOSE

#include <stdio.h>

"standard input-output header" (cabecera estándar E/S)

Contiene las definiciones de las macros, las constantes, las declaraciones de funciones de la biblioteca estándar del lenguaje de programación C para hacer operaciones, estándar, de entrada y salida, así como la definición de tipos necesarias para dichas operaciones. ***PRINTF y SCANF***

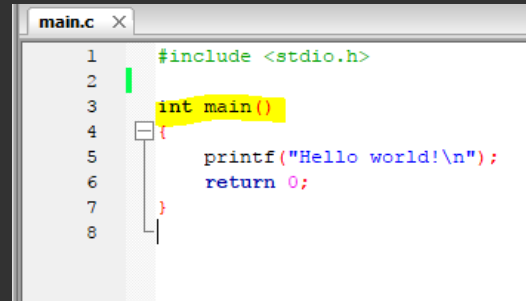


```
main.c x
1  #include <stdio.h>
2
3  int main()
4  {
5      printf("Hello world!\n");
6      return 0;
7  }
8
```

HOLA MUNDO - DESGLOSE

int main()

Función principal del programa. Todos los programas de C deben tener una función llamada *main*. Es la que primero se ejecuta. El **int** (viene de Integer=Entero) que tiene al principio significa que cuando la función *main* acabe devolverá un número entero. Este valor se suele usar para saber cómo ha terminado el programa. Normalmente este valor será 0 si todo ha ido bien, o un valor distinto si se ha producido algún error.



```
main.c x
1  #include <stdio.h>
2
3  int main()
4  {
5      printf("Hello world!\n");
6      return 0;
7  }
8
```

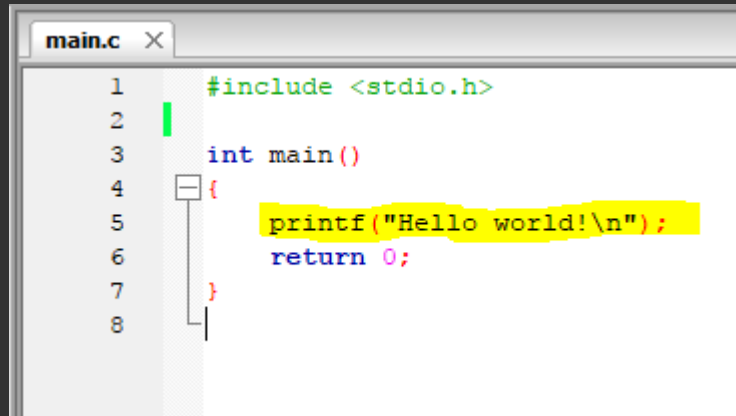
HOLA MUNDO - DESGLOSE

```
printf("Hello world!\n");
```

La función printf muestra un mensaje por la pantalla. Al final del mensaje "Hola mundo" aparece el símbolo '\n'; este hace que después de imprimir el mensaje se pase a la línea siguiente.

Fíjate en el ";" del final, es la forma que se usa en C para separar una instrucción de otra.

Case sensitive, C distingue entre mayúsculas y minúsculas!

A screenshot of a code editor window titled 'main.c'. The code is as follows:

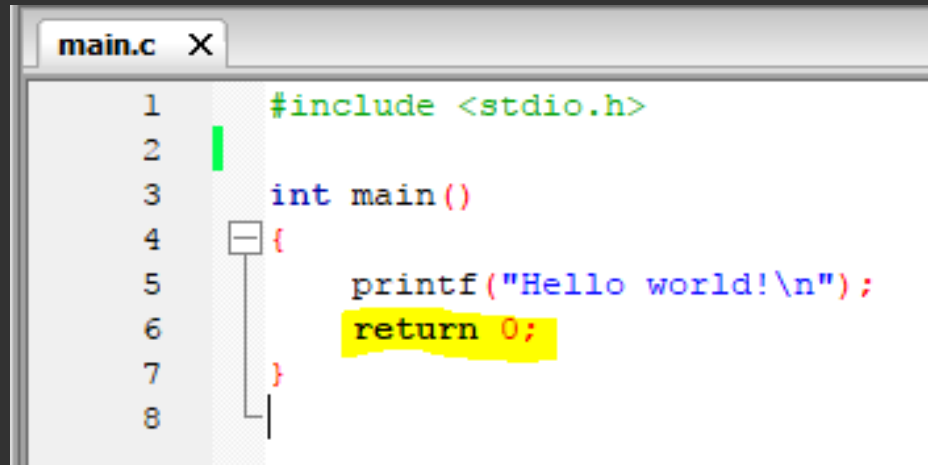
```
1  #include <stdio.h>
2
3  int main()
4  {
5      printf("Hello world!\n");
6      return 0;
7  }
8
```

The line containing the printf statement is highlighted in yellow. The code is color-coded: #include is green, int is blue, main() is blue, { is blue, printf is blue, "Hello world!\n" is black, ; is black, return is blue, 0 is blue, and } is red.

HOLA MUNDO - DESGLOSE

return 0;

La función `int main` devuelve un valor entero. Como en este programa no se pueden producir errores la salida siempre será 0. La forma de hacer que el programa devuelva un 0 es usando `return`.



```
main.c X
1  #include <stdio.h>
2
3  int main()
4  {
5      printf("Hello world!\n");
6      return 0;
7  }
8
```


PRINTF

- Imprime información por pantalla
- Incluida en `<stdio.h>`

Borrar pantalla

➡ `system("cls");`

Pausa

➡ `system("pause");`

Variables en C

- C almacena datos en distintos **tipos de variables** (int, double, char, etc)
- A las variables no se les puede dar cualquier nombre. Admiten letras de la “a” a la “z” (no vale la ñ), números y el símbolo “_”. Tampoco se pueden poner signos de admiración ni de interrogación. No pueden empezar por un numero ni puedes usar una palabra reservada por el compilador (main, do, while, etc)
- C distingue entre mayúsculas y minúsculas

Tipos de Variables en C

TIPO	ANCHO EN BIT	RANGO EN PC
<i>char</i>	8	-128 a 127
<i>unsigned char</i>	8	0 a 255
<i>signed char</i>	8	-128 a 127
<i>int</i>	16	-32768 a 32767
<i>unsigned int</i>	16	0 a 65535
<i>signed int</i>	16	-32768 a 32767
<i>short int</i>	16	-32768 a 32767
<i>unsigned short int</i>	16	0 a 65535
<i>signed short int</i>	16	-32768 a 32767
<i>long int</i>	32	-2147483648 a 2147483647
<i>signed long int</i>	32	-2147483648 a 2147483647
<i>unsigned long int</i>	32	0 a 4294967295
<i>float</i>	32	3.4E-38 a 3.4E+38
<i>double</i>	64	1.7E-308 a 1.7E+308
<i>long double</i>	64	1.7E-308 a 1.7E+308

Declaración de variables

¿Dónde declaramos las variables? Variables locales y variables globales.

Variable Global

```
#include <stdio.h>

int x;

int main()
{
}
```

Variable Local

```
#include <stdio.h>

int main()
{
    int x;
}
```

Declaración de variables

¿Por qué se desaconseja el uso de las variables globales?

- Legibilidad menor.
- Atenta contra uno de los principios de la programación, la **modularidad**. El bajo acoplamiento supone no compartir espacios de memoria con otras funciones, y potenciar el paso de información (llamadas) para que la función trate la información localmente.

Declaración de variables

¿Por qué se desaconseja el uso de las variables globales?

- El uso indiscriminado de variables globales produce efectos colaterales. Esto sucede cuando existe una alteración no deseada del contenido de una variable global dentro de una función, bien por invocación, bien por olvidar definir en la función una variable local o un parámetro formal con ese nombre.

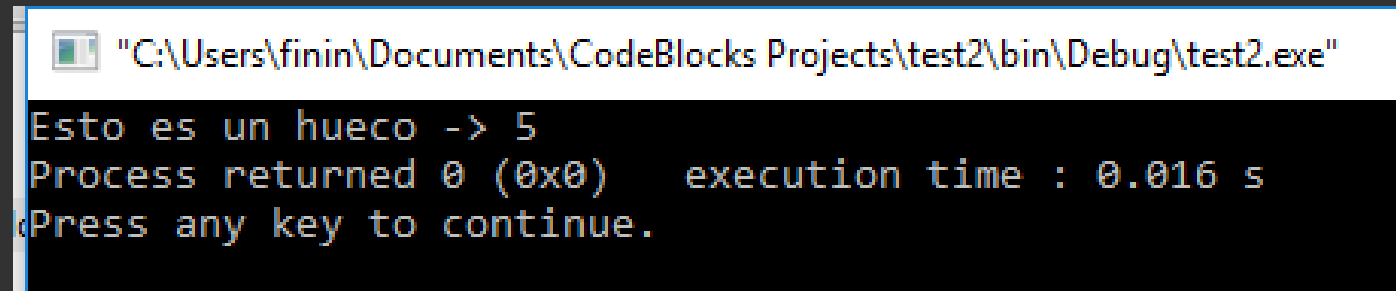
Mostrar variables

¿Cómo se muestran las variables por pantalla?

Al usar la función *printf* de C tenemos primero que especificar una serie de “huecos” para las variables y después incluir la variables que queremos que ocupe ese hueco al final de la instrucción.

```
int i = 5;  
printf("Esto es un hueco -> %d", i);
```

Resultado al compilar:



```
"C:\Users\finin\Documents\CodeBlocks Projects\test2\bin\Debug\test2.exe"  
Esto es un hueco -> 5  
Process returned 0 (0x0)   execution time : 0.016 s  
Press any key to continue.
```

Mostrar variables

¿Cómo se muestran las variables por pantalla?

Si hay más de una variable hay que tener cuidado con el orden en el que son colocadas:

```
int i = 5;  
int j = 2;  
int z = 7;  
printf("Esta primero %d luego esta %d y finalmente %d", z,j,i);
```

¿Resultado al compilar?

Mostrar variables

¿Cómo se muestran las variables por pantalla?

Cada tipo de variable tiene su “hueco” específico:

Tipo de dato	Especificadores de formato	Comentario
Int	%d	Entero con signo
Float	%f	Real con signo
Char	%c	carácter
Char [n]	%s	Cadena de caracteres

Operadores

¿Qué es un operador?

Un operador sirve para manipular datos. Los hay de varios tipos: de asignación, de relación, lógicos, aritméticos y de manipulación de bits.

Operador de asignación (=)

Sirve para dar un valor a una variable. Este valor puede ser un número que tecleamos directamente u otra variable:

```
a = 3; /* Asignamos un valor directamente */
```

```
a = b; /* Le asignamos el valor de una variable */
```

Operador suma(+)

Permite sumar variables.

```
total = a + b;
```

Peculiaridades:

x = x + 5 se puede escribir como **x +=5**

Operador Incremento(++)

Suma uno a una variable

```
x++;
```

¿Es lo mismo que?

```
++x;
```

Operadores resta (-) y decremento(--)

Misma dinámica que la suma, pero con la resta

`x--`

`x-= 5;`

Operadores multiplicación (*) y división (/)

- Multiplican y dividen.
- El operador * también se usa para los **punteros**.
- El operador división devuelve la división exacta en caso de números *float*, pero en caso de números entero devuelve el cociente.

Operador resto (%)

- Si con el operador (/) operador obteníamos el módulo o cociente de una división entera con éste (%) podemos tener el resto.
- No funciona más que con enteros, no vale para números float o double.

Operadores de comparación

==	igual que	se cumple si son iguales
!=	distinto que	se cumple 1 si son diferentes
>	mayor que	se cumple si el primero es mayor que el segundo
<	menor que	se cumple si el primero es menor que el segundo
>=	mayor o igual que	se cumple si el primero es mayor o igual que el segundo
<=	menor o igual que	se cumple si el primero es menor o igual que el segundo

Operadores de comparación

¿Qué resultado da este código?

```
int main() {  
  
    printf( "10 > 5 da como resultado %i\n", 10>5 );  
    printf( "10 > 5 da como resultado %i\n", 10>5 );  
    printf( "5== 5 da como resultado %i\n", 5==5 );  
    printf( "10==5 da como resultado %i\n", 10==5 );  
  
    return 0;  
}
```

"C:\Users\finin\Documents\CodeBlocks Projects\test2\bin\Debug\test2.exe"

```
10 > 5 da como resultado 1  
10 > 5 da como resultado 1  
5== 5 da como resultado 1  
10==5 da como resultado 0
```

```
Process returned 0 (0x0)   execution time : 0.016 s  
Press any key to continue.
```

Operadores lógicos

Estos son los que nos permiten unir varias comparaciones: `10>5` y `6==6`.

Los operadores lógicos son: AND (`&&`), OR (`||`), NOT(`!`).

Operador lógico AND (&&)

Operador && (AND, en castellano Y):
Devuelve un 1 si se cumplen todas las condiciones.

```
printf( "Resultado: %d", (10==10 && 5>2 );
```

Operador lógico OR (||)

Operador || (OR, en castellano O):
Devuelve un 1 si se cumple alguna de las condiciones.

```
printf( "Resultado: %d", (10==9 || 5>2 );
```

Operador lógico NOT (!)

Operador ! (NOT): Devuelve un 1 si es un 0 y un 0 si es un 1.

```
int a = 14;  
printf( "Resultado: %d",!a);
```


Operador sizeof

- Este operador nos permite conocer el tamaño **en bytes** de una variable.
- El tamaño de una variable cambia de un compilador a otro, es la mejor forma de asegurarse.
- Se usa poniendo el nombre de la variable después de sizeof y separado de un espacio

Operador sizeof

```
int main() {  
  
    int variable;  
  
    printf( "Tamanho de la variable: %d\n", sizeof variable );  
  
    return 0;  
}
```

"C:\Users\finin\Documents\CodeBlocks Projects\test2\bin\Debug\test2.exe"

Tamanho de la variable: 4

Process returned 0 (0x0) execution time : 0.000 s

Press any key to continue.

Operador sizeof

- También se puede usar con los especificadores de tipos de datos (char, int, float, double...). Pero en éstos se usa de manera diferente, hay que poner el especificador entre paréntesis:

```
printf("Las variables tipo int ocupan: %d\n", sizeof(int) );
```

scanf

- El uso de *scanf* es muy similar al de *printf* con una diferencia, nos da la posibilidad de que el usuario introduzca datos en vez de mostrarlos. No nos permite mostrar texto en la pantalla, por eso si queremos mostrar un mensaje usamos un *printf* delante. Un ejemplo:

```
int main() {  
  
    int num;  
    printf( "Introduce un numero " );  
    scanf( "%d", &num );  
    printf( "Has tecleado el numero %d\n", num );  
    return 0;  
}
```

scanf

- Lo primero nos fijamos que hay una cadena entre comillas. Esta es similar a la de *printf*, nos sirve para indicarle al compilador qué tipo de datos estamos pidiendo. Como en este caso es un *integer* usamos %d. Después de la coma tenemos la variable donde almacenamos el dato, en este caso 'num'.
- En el *scanf* la variable 'num' lleva delante el símbolo &. Nos sirve para indicar al compilador cual es la dirección (o posición en la memoria) de la variable.

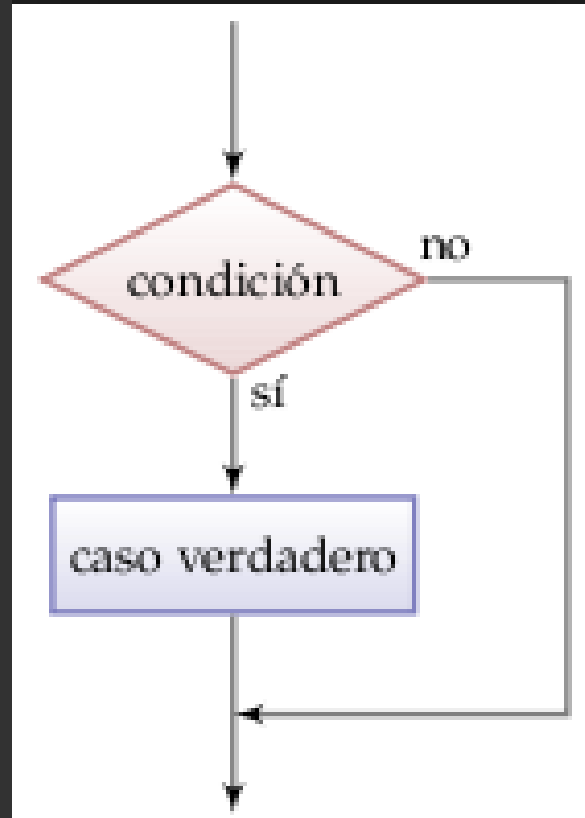
scanf

Podemos preguntar por más de una variable a la vez en un sólo scanf, hay que poner un %d por cada variable (después de cada numero pulsar enter):

```
int main() {  
  
    int a, b, c;  
  
    printf( "Introduce tres numeros: " );  
    scanf( "%d %d %d", &a, &b, &c );  
    printf( "Has tecleado los numeros %d %d %d\n", a, b, c );  
    return 0;  
}
```

La estructura de selección if

- Utilizamos la estructura de selección if (si...) para elegir entre cursos alternativos de acción.



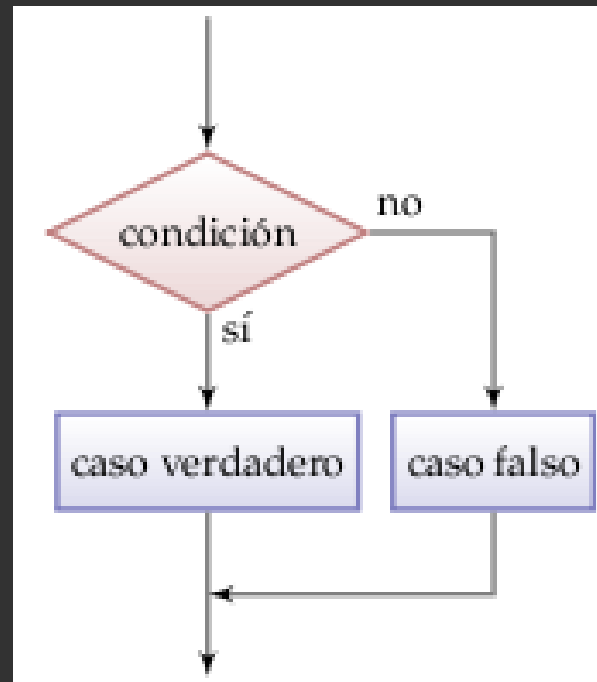
La estructura de selección if

- Ejemplo: código en C que muestre “aprobado” en caso de que la variable int nota sea mayor o igual que 60:

```
int main() {  
    int nota;  
  
    printf("Que nota ha sacado el alumno: ");  
    scanf("%d", &nota);  
  
    if(nota >= 60) printf("Aprobado");  
  
    return 0; }
```


La estructura de selección if/else

- La estructura de selección if/else permite que el programador especifique que se ejecuten acciones distintas cuando la condición sea verdadera que cuando la condición sea falsa.



La estructura de selección if/else

- Con el ejemplo anterior añadir la condición “falsa”, es decir, que la variable nota sea menor que 60. (¡ojo a las negaciones de las condiciones!)

```
int main() {  
    int nota;  
  
    printf("Que nota ha sacado el alumno: ");  
    scanf("%d", &nota);  
  
    if(nota >= 60) printf("Aprobado");  
    else printf("Suspenso");  
  
    return 0; }
```

La estructura de selección if/else

- Cuando la condición es más compleja la negación deja de ser tan obvia.

```
if("premisa A" && "premisa B") printf("Correcto")  
else printf("No correcto")
```

- **LEYES DE MORGAN**

La estructura de selección if/else

LEYES DE MORGAN:

La negación de la conjunción es la disyunción de las negaciones.
La negación de la disyunción es la conjunción de las negaciones.

Las reglas pueden ser expresadas en un lenguaje formal con dos proposiciones P y Q , de esta forma:

$$\sim(P \& \& Q) == (\sim P) \mid \mid (\sim Q)$$

$$\sim(P \mid \mid Q) == (\sim P) \& \& (\sim Q)$$

La estructura de selección if/else

Podemos anidar estructuras if/else (colocar una estructura dentro de otra) para conseguir más casos. Ejemplo de las notas:

```
if(nota >= 90) printf("Sobresaliente");  
else  
    if(nota >= 65) printf("Notable");  
    else  
        if(nota >= 50) printf("Aprobado");  
        else  
            printf("Suspenso");
```

La estructura de selección if/else

Para una mayor claridad y evitar el excesivo sangrado a la derecha podemos usar else if:

```
if(nota >= 90) printf("Sobresaliente");  
else if(nota >= 65) printf("Notable");  
else if(nota >= 50) printf("Aprobado");  
else printf("Suspenso");
```

La estructura de selección if/else

La estructura if/else espera un enunciado dentro de su cuerpo. Para incluir varios enunciados en el cuerpo de un if hay que usar llaves:

```
if(nota >= 60) printf("Aprobado");  
    else {  
        printf("Suspenso");  
        printf("\nY mira que era dificil suspender");  
    }
```

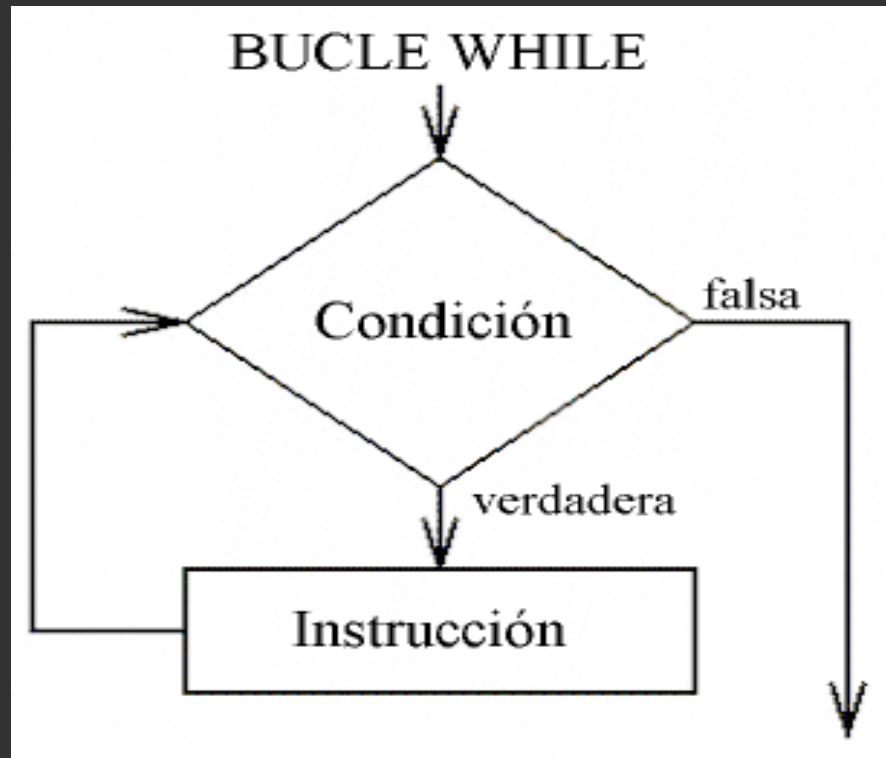
Ejemplo

Salidas para este código:

```
int main() {  
    int nota1 = 60;  
    int nota2 = 60;  
  
    printf("Primera nota: ");  
    scanf("%d", &nota1);  
    printf("\nSegunda nota ");  
    scanf("%d", &nota2);  
  
    if((nota1 >= 60) && !(nota2 < 40)) printf("Aprobado");  
        else printf("Suspenso");  
  
    return 0; }
```


El bucle while

El bucle while permite especificar que se repita una acción en tanto cierta condición se mantenga verdadera.



El bucle while

¿Cuántos “hola” se van a imprimir por pantalla?

```
int main() {  
    int contador = 1;  
  
    while (contador < 10){  
        printf("Hola");  
    }  
    return 0;  
}
```

El bucle while

¿Y ahora?

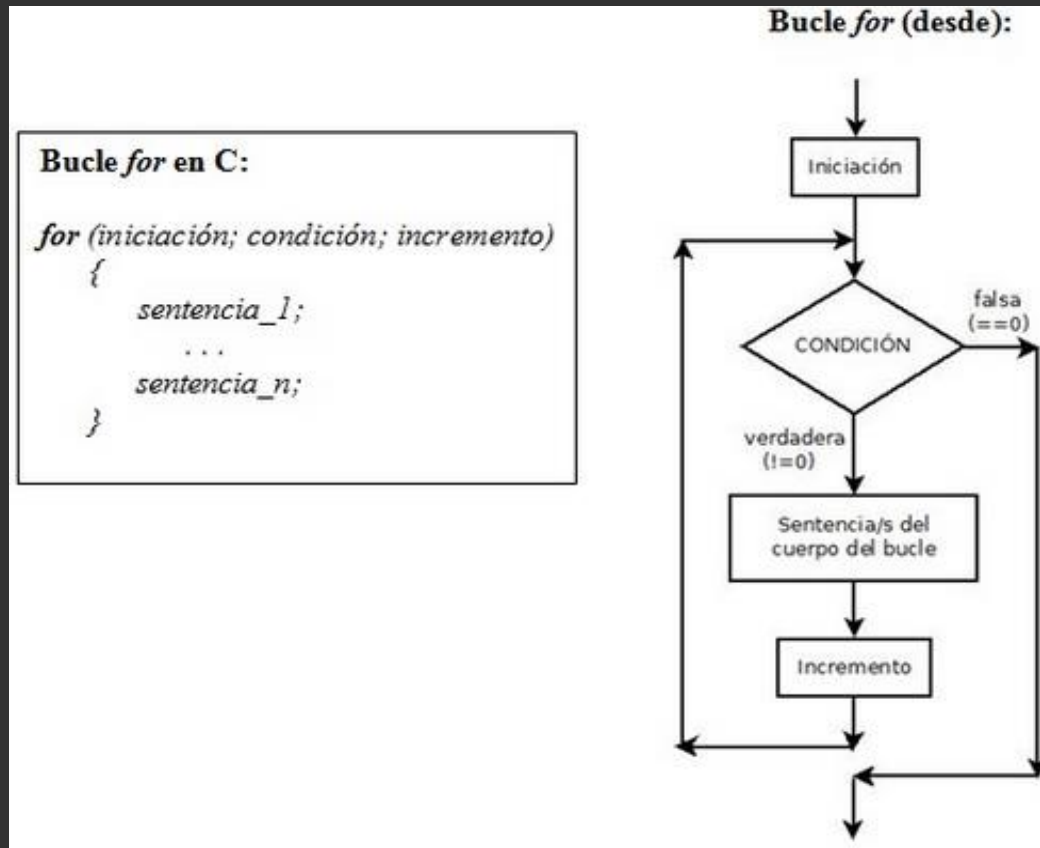
```
int main() {  
    int contador = 1;  
  
    while (contador < 5){  
        printf("\nHola");  
        contador++;  
    }  
    return 0;  
}
```

Ejemplo While

Usando un bucle while realizar un programa en C en el que introduzcas por teclado 10 notas y calcule su media.

El bucle for

El bucle for maneja de manera automática todos los detalles de la repetición controlada por contador:



El bucle for

El bucle for maneja de manera automática todos los detalles de la repetición controlada por contador:

```
int i;  
    for(i = 0;i<=10;i++){  
        printf("Hola\n");  
    }
```

El bucle for

Ejercicio: Programa utilizando bucles un programa en el que introduzcas el número de filas de una semi-pirámide y construya una semi pirámide de asteriscos:

```
Escriba el numero de filas para la semi piramide 5
*
**
***
****
*****

Process returned 0 (0x0)   execution time : 1.985 s
Press any key to continue.
```

Funciones

¿Por qué usar funciones?

- La experiencia ha demostrado que la mejor forma de desarrollar y mantener un programa grande es construirlo a partir de piezas menores o módulos (divide y vencerás).
- En C (y en la mayoría de los lenguajes de programación) estos módulos se llaman funciones.
- Existen infinidad de funciones empaquetadas en las bibliotecas standard de C (cálculos matemáticos, manipulación de cadenas, etc.)

Funciones

Ejemplo:

```
int suma_2_numeros(int numero1, int numero2){  
    int resultado = numero 1 + numero2;  
    return resultado;  
}
```

Funciones

- Todas las variables declaradas en las definiciones de las funciones son **variables locales** (son conocidas solo en la función en la cual están definidas).
- La mayor parte de las funciones tienen una lista de **parámetros**. Que ofrecen la forma de comunicar información entre funciones.

- Ejemplo:

```
int suma_2_numeros(int numero1, int numero2){  
    int resultado = numero 1 + numero2;  
    return resultado;  
}
```

Funciones

Definición de funciones:

- Cada programa que hemos presentado ha consistido en una función llamada *main*, que para llevar a cabo sus tareas ha llamado a funciones estándar de la biblioteca.
- Vamos a escribir como ejemplo una función que calcule el cuadrado de un número entero.

Funciones

Definición de funciones:

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

```
int cuadrado (int); //Prototipo de la función
```

```
int main(){  
    int x;  
    for (x = 2;x<=10; x = x+2) printf("%d ", cuadrado(x));  
    return 0;  
}
```

```
//Definición de función  
int cuadrado(int y){  
    return y*y;  
}
```

Funciones

Definición de funciones:

- La función “cuadrado” recibe una copia del valor de x en el parámetro y. Después devuelve el valor $y*y$.
- El resultado regresa a la función main, que fue donde llamamos a “cuadrado”.
- La definición de “cuadrado” muestra que esta espera un parámetro entero (llamado y).
- La palabra reservada “int” que precede al nombre de la función indica que “cuadrado” va a devolver un valor entero

Funciones

¿Prototipos de funciones?

```
int cuadrado (int); //Prototipo de la función
```

- Los prototipos de funciones indican al compilador el tipo de dato regresado por la función, el número de parámetros que la función espera recibir, los tipos de dichos parámetros y el orden en el cual se esperan.
- Anteriormente (a día de hoy, muy anteriormente) esta comprobación no se llevaba a cabo, pudiendo dar lugar a errores graves o no tan graves pero muy difíciles de detectar.

Funciones

Ejemplo: Función que simule la tirada de dos dados

Consejos y ayudas:

- Usando la librería `<stdlib.h>` podemos usar la función `rand()`, que genera un número entre 0 y la constante `RAND_MAX` (32767).

- La fórmula para tener un número aleatorio entre un valor determinado es:

$$\text{numero} = a + \text{rand()} \% b$$

- Donde **a** marca el punto de inicio y **b** representa el número de opciones posibles, `numero = 1 + rand()%6` para un dado normal de 6 caras.

Funciones

Práctica de aula 1 : Diseñar una función en C que simule la tirada de un dado de X caras (siendo X un número entero entre 2 y 10 que se le pasa como parámetro de la función)

Funciones

Práctica de aula 2, implementar en código C el siguiente juego: “*Craps*”

Un jugador tira dos dados. Cada dado tiene seis caras con puntuaciones de 1 a 6. Una vez hecha las tiradas se suman las puntuaciones obtenidas.

Si a la primera tirada la puntuación es 7, o bien 11, el jugador gana. Por el contrario, si la suma es 2,3, o 12 el jugador pierde.

En otro caso (4,5,6,7,8,9 o 10) la tirada se convierte en la tirada de referencia y el jugador tira otra vez. El objetivo ahora es conseguir repetir la tirada de referencia (la puntuación de la primera tirada) antes de sacar un 7. Si la repite gana, si sale un 7 pierde.

Funciones

Parámetros por valor o por referencia:

- Cuando los argumentos se pasan por valor se efectúa una copia del valor del argumento y esta se pasa a la función llamada. Las modificaciones de la copia no afectan a la variable original.
- Cuando un argumento es pasado por referencia se pasa a la función directamente la dirección de memoria de la variable. Este hecho permite que la función modifique el valor original de esta.
- Si no hace falta modificar el valor de la variable se emplea la opción por valor por seguridad.

Funciones

Operadores * y &:

- Utilizamos * para declarar punteros, pero también es utilizamos como operador de **indirección**. De momento quedaros con que nos va a servir para pasarle direcciones de memoria a las funciones.
- Utilizamos & para acceder a la dirección de memoria de una variable.

Funciones

Suma_referencia:

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

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

```
void suma_referencia(int,int,int*);
```

```
int main(){
```

```
    int a, b, resultado;
```

```
    printf("Primer numero a sumar: ");
```

```
    scanf("%d", &a);
```

```
    printf("Segundo numero a sumar: ");
```

```
    scanf("%d", &b);
```

```
    suma_referencia(a,b,&resultado);
```

```
    printf("\nEl resultado es %d", resultado);
```

```
    return 0;}
```

Suma_referencia:

```
void suma_referencia(int x,int y,int *resultado){  
    *resultado = x + y;  
}
```

Funciones - Recursividad

- Hasta ahora hemos visto programas con funciones que se llaman unas a otras de forma disciplinada y jerárquica.
- Para algunos tipos de programas es útil tener funciones que se llamen a sí mismas (funciones recursivas).
- Una función recursiva es aquella que se llama a sí misma, ya sea directa, o indirectamente a través de otra función.

Funciones - Recursividad

Ejemplo factorial, calcular una función no recursiva que calcula el factorial de un número

$$\text{factorial}(n) = n * (n-1) * (n-2) * \dots * 1$$

Funciones - Recursividad

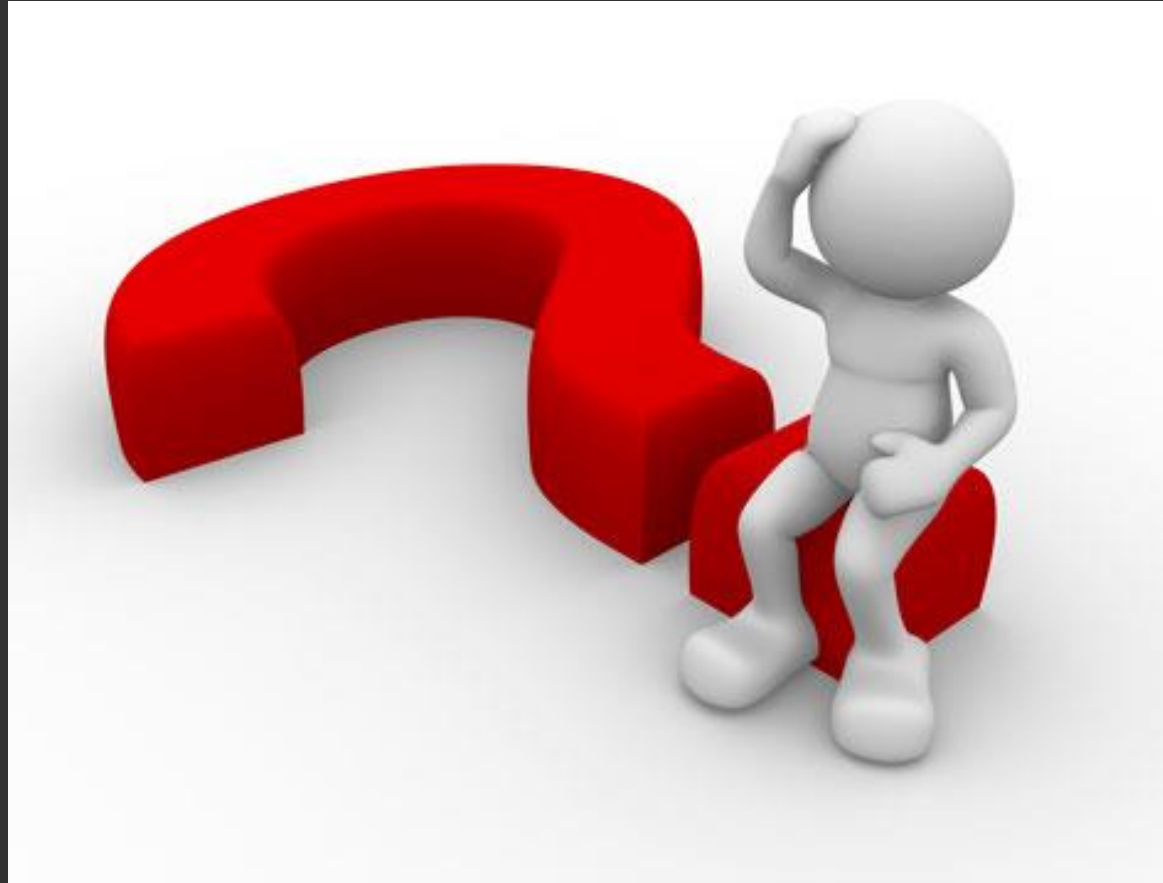
- Ejemplo factorial: Lo más importante a la hora de trabajar con recursividad es establecer la fórmula recursiva y el caso base

Fórmula recursiva: $n! = n * (n-1)!$

Caso base: para $n = 0 \rightarrow n! = 1$

Con esto la función sale sola

DUDAS, PREGUNTAS, INQUIETUDES



WEBGRAFÍA

<https://www.udacity.com>

<http://www.programmr.com>

<https://www.codecademy.com/>

<http://coursera.org>

<http://www.w3schools.com>

<https://www.tutorialspoint.com>

<http://jorgesanchez.net>