

(Versión de fecha 8 de octubre de 2024)

1 EJ_1.C: EJEMPLO BÁSICO DE ENTRADA-SALIDA

- El programa calcula el área de un rectángulo.
- Tiene los tres bloques: entrada, procesamiento y salida.
- La entrada se hace directamente en el código del programa, dando valor a las variables *base* y *altura*. El inconveniente es que, cada vez que se quiera calcular el área de un rectángulo, hay que cambiar los valores y volver a compilar.
- En la salida, observa el formato que se utiliza para mostrar una variable del tipo *double*: %1f (*long float*).

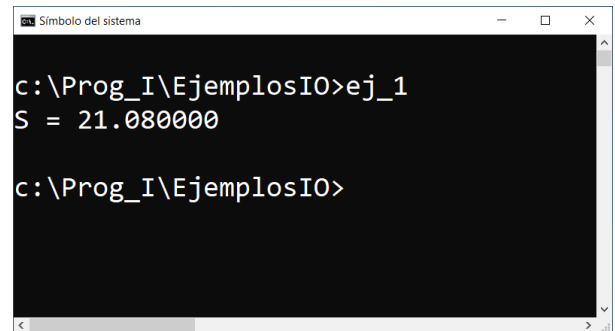
Ejemplo 1 ej_1.c

```
#include <stdio.h>

int main() {
    // Entrada
    double base = 3.4;
    double altura = 6.2;

    // Procesamiento
    double area = base*altura;

    // Salida
    printf("S = %1f \n", area);
}
```



2 EJ_2.C: ENTRADA DEL USUARIO UTILIZANDO SCANF()

- El programa es el mismo que el del Ejemplo *ej_1.c* para calcular el área de un rectángulo, pero cambiando la forma de hacer la *Entrada* de datos.
- Se añade un bloque *Inicialización*, con la declaración de las variables *base* y *altura*.
- En este caso, la *Entrada* se hace solicitando al usuario los datos de base y altura con instrucciones *scanf()*. Observa el formato utilizado para solicitar un *double*: %1f (*long float*). Observa también cómo se indica la variable donde se debe almacenar el valor que teclee el usuario: &nombre_variable. El símbolo & se puede leer como *la dirección de memoria de*.
- En el bloque de *Entrada*, se imprime un mensaje antes de solicitar cada variable para orientar al usuario.
- El procesamiento es el mismo que el del ejemplo *ej_1.c*.
- En la *Salida* se ha decidido mostrar el área con dos decimales, observa en el código cómo se hace: %.21f.
- La compilación se ha hecho de manera "manual" en el terminal, observa la figura. La instrucción de compilación es la siguiente:

```
gcc ej_2.c -o ej_2.exe
```

Se llama al compilador, *gcc* y se le pasa el nombre del fichero punto c que queremos compilar y el nombre del fichero ejecutable que queremos generar, este último precedido de *-o* (*output*).

Ejemplo 2 ej_2.c

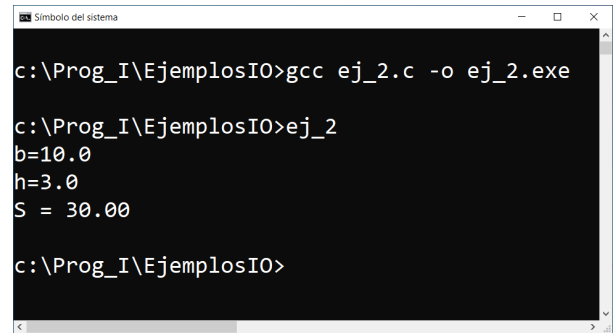
```
#include <stdio.h>

int main() {
    // Inicialización
    double base, altura;

    // Entrada
    printf("b=");
    scanf("%lf", &base);
    printf("h=");
    scanf("%lf", &altura);

    // Procesamiento
    double area = base*altura;

    // Salida
    printf("S = %.2lf \n", area);
}
```



```
Símbolo del sistema
c:\Prog_I\EjemplosIO>gcc ej_2.c -o ej_2.exe
c:\Prog_I\EjemplosIO>ej_2
b=10.0
h=3.0
S = 30.00
c:\Prog_I\EjemplosIO>
```

3 EJ_3.C: CÁLCULO DEL ÁREA DE UN TRIÁNGULO

- El ejemplo es similar a los anteriores pero, en esta ocasión, se calcula el área de un triángulo y se añaden algunas mejoras a modo de buenas prácticas de codificación.
- Se añade una línea de comentario al inicio del código explicando el objetivo del programa para que sirva de ayuda al que lea el código. Es habitual también poner en los comentarios el autor del código y la fecha en la que se ha escrito el programa.
- En el bloque de inicialización también se añade un *printf()* para mostrar en pantalla al usuario el objetivo del programa.
- El resto del programa es similar a los anteriores, con la salvedad de que el cálculo del área se ajusta a la fórmula del área de un triángulo. Observa el 2.0 del denominador. Es un *double* y no conviene olvidar el *punto cero*.
- La compilación y ejecución también se ha hecho directamente en el terminal (observa la figura).

Ejemplo 3 ej_3.c

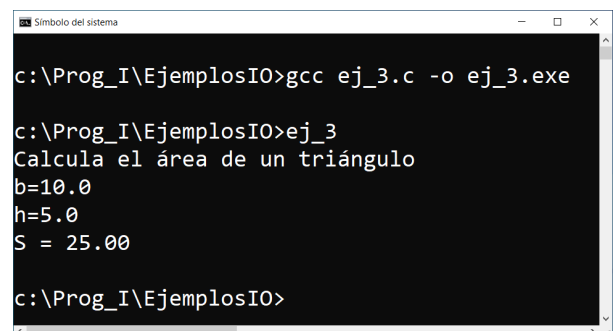
```
// Calcula el área de un triángulo
#include <stdio.h>

int main() {
    // Inicialización
    printf("Calcula el área de un triángulo\n");
    double base, altura;

    // Entrada
    printf("b=");
    scanf("%lf", &base);
    printf("h=");
    scanf("%lf", &altura);

    // Procesamiento
    double area = base*altura/2.0;

    // Salida
    printf("S = %.2lf \n", area);
}
```



```
Símbolo del sistema
c:\Prog_I\EjemplosIO>gcc ej_3.c -o ej_3.exe
c:\Prog_I\EjemplosIO>ej_3
Calcula el área de un triángulo
b=10.0
h=5.0
S = 25.00
c:\Prog_I\EjemplosIO>
```

4 EJ_4.C: CÁLCULO DE LA HIPOTENUSA DE UN TRIÁNGULO RECTÁNGULO

- El programa es similar a los anteriores, pero se quiere calcular la hipotenusa de un triángulo rectángulo a partir del valor de los dos catetos.
- La fórmula es la del teorema de Pitágoras. Llamando a a la hipotenusa y b y c a los catetos, se tiene:

$$a = \sqrt{b^2 + c^2}$$

- Para calcular la raíz cuadrada de un número se puede utilizar la función `sqrt()` de la librería `math.h`. Observa la instrucción `#include <math.h>` al principio del programa para poder utilizar las funciones de la librería `math.h`.
- La función `sqrt()` recibe como parámetro de entrada un número `double` y devuelve como resultado otro número `double`. Observa el código: entre los paréntesis de la función ponemos una expresión cuyo resultado será el parámetro de entrada a la función. Observa también cómo calculamos los cuadrados: multiplicando el número por sí mismo.
- Observa cómo se utiliza una función: se le pasan los parámetros entre los paréntesis y la función devolverá el resultado. En el código, el resultado que devuelve la función `sqrt()` es el valor que se asigna a la variable `hipot`.
- En el siguiente enlace puedes consultar las funciones que ofrece la librería `math.h`:

https://www.tutorialspoint.com/c_standard_library/math_h.htm

Ejemplo 4 ej_4.c

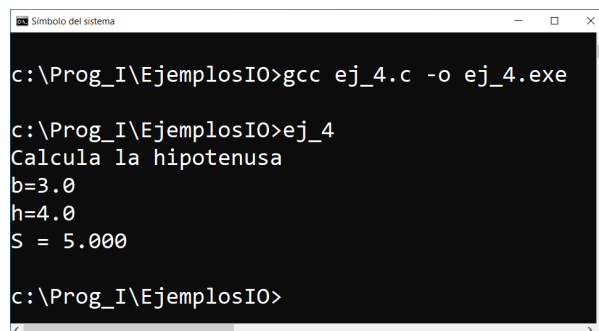
```
// Calcula la hipotenusa
#include <stdio.h>
#include <math.h>

int main() {
    // Inicialización
    printf("Calcula la hipotenusa\n");
    double cat1, cat2;

    // Entrada
    printf("b=");
    scanf("%lf", &cat1);
    printf("h=");
    scanf("%lf", &cat2);

    // Procesamiento
    double hipot = sqrt(cat1*cat1 + cat2*cat2);

    // Salida
    printf("S = %.3lf \n", hipot);
}
```



```
Símbolo del sistema
c:\Prog_I\EjemplosIO>gcc ej_4.c -o ej_4.exe
c:\Prog_I\EjemplosIO>ej_4
Calcula la hipotenusa
b=3.0
h=4.0
S = 5.000
c:\Prog_I\EjemplosIO>
```

5 EJ_5: UTILIZACIÓN DEL NÚMERO π

Se reproducen a continuación tres versiones de un programa para calcular el área de un círculo. La entrada se hace directamente en el código del programa, para facilitar la ejecución, pues el objetivo es mostrar tres maneras diferentes de incorporar el valor de π en los programas.

En la primera versión del programa, `ej_5_a.c`, el valor de π se teclea directamente en la línea de código del cálculo del área. Es una solución que puede servir para pequeños programas como éste. Por supuesto, se podría haber utilizado una aproximación con más decimales, por ejemplo, 3.141592.

Ejemplo 5 ej_5_a.c

```
#include <stdio.h>

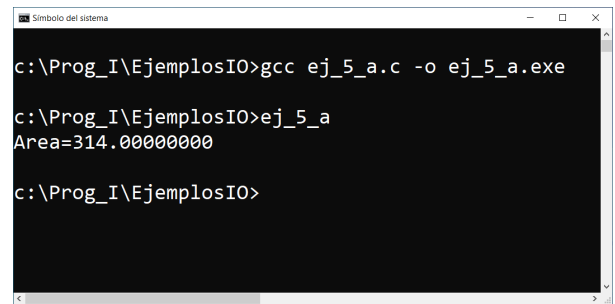
int main() {

    double r = 10.0;

    double area = 3.14 * r * r;

    printf("Area=%.8lf\n", area);

}
```



```
c:\Prog_I\EjemplosIO>gcc ej_5_a.c -o ej_5_a.exe
c:\Prog_I\EjemplosIO>ej_5_a
Area=314.00000000
c:\Prog_I\EjemplosIO>
```

La segunda versión del programa, *ej_5_b.c*, utiliza lo que se denomina una *directiva de preprocesamiento*, en este caso, una cláusula *#define*:

```
#define PI 3.142
```

Esta instrucción asocia un valor determinado con un nombre. En este caso, se asocia el valor 3.142 con el nombre *PI*. Luego, en el código del programa, cada vez que aparezca el nombre *PI*, el compilador lo sustituirá por el valor 3.142.

Esta solución es mucho más eficiente, sobre todo si el valor se va a utilizar en diferentes partes del programa. Si más adelante se quisiera repetir el programa, pero utilizando un valor de π con más decimales, bastaría cambiar el valor en la cláusula *#define* y quedaría actualizado en todas las partes del programa en las que se utilice. De haber usado la solución del ejemplo anterior, habría que haber ido buscando dónde aparece el valor e irlo cambiando y ése es un proceso más tedioso y mas propenso a cometer errores.

Ejemplo 6 ej_5_b.c

```
#include <stdio.h>

#define PI 3.142

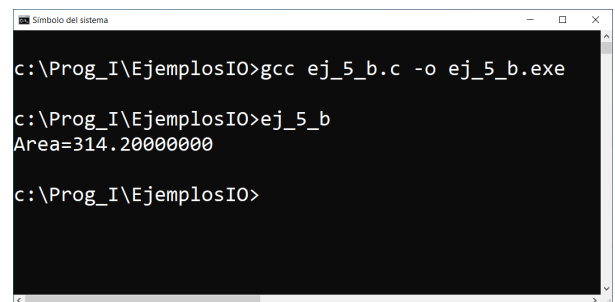
int main() {

    double r = 10.0;

    double area = PI * r * r;

    printf("Area=%.8lf\n", area);

}
```



```
c:\Prog_I\EjemplosIO>gcc ej_5_b.c -o ej_5_b.exe
c:\Prog_I\EjemplosIO>ej_5_b
Area=314.20000000
c:\Prog_I\EjemplosIO>
```

La última alternativa que presentamos aquí es utilizar el valor de π que proporciona la librería *math.h*. La librería *math.h* proporciona la constante *M_PI*, que es una aproximación del valor de π con quince decimales. A diferencia de las dos soluciones anteriores, será necesario incorporar la librería *math.h* al programa, poniendo al principio del mismo la instrucción *#include <math.h>*.

Ejemplo 7 ej_5_c.c

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

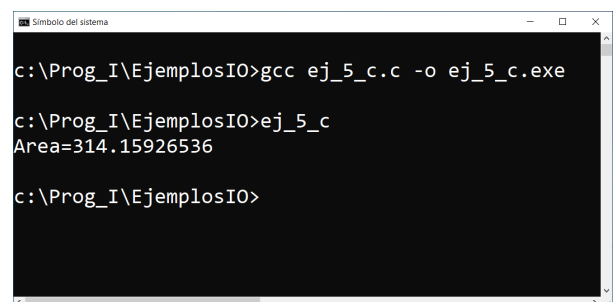
int main() {

    double r = 10.0;

    double area = M_PI * r * r;

    printf("Area=%.8lf\n", area);

}
```



```
c:\Prog_I\EjemplosIO>gcc ej_5_c.c -o ej_5_c.exe
c:\Prog_I\EjemplosIO>ej_5_c
Area=314.15926536
c:\Prog_I\EjemplosIO>
```

Nota

Al momento de escribir estas líneas, el editor VSCode subrayaba en rojo la constante M_{PI} e indicaba que el identificador no estaba definido. Evidentemente es un fallo de la extensión para C de VSCode, pues el programa compila y ejecuta perfectamente, como se puede ver en la figura que acompaña al código.

int \rightarrow 4 bytes (?) \rightarrow 32 bits

double \rightarrow 8 bytes ; 3.75

$1.2 \text{e-}3$

\downarrow
 1.2×10^{-3}

3. (\approx 15 decim)

$\frac{3.}{2.}$ \rightarrow int \rightarrow 1
 \rightarrow double \rightarrow 1.5

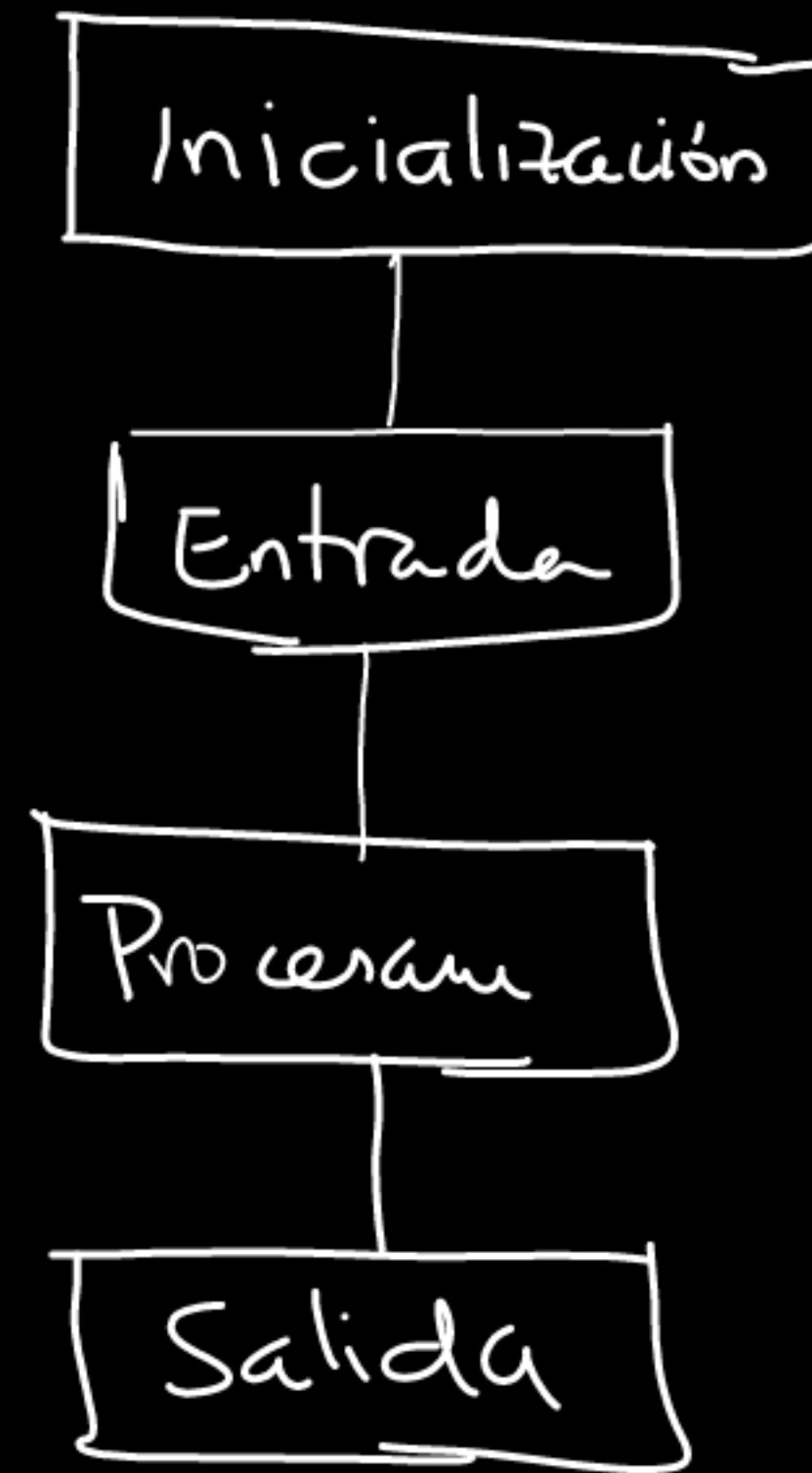
`printf("cad_formato", v1, v2, ...)`

↳ caracteres (texto)

↳ caract. especiais → `\n`

↳ especif. formato → % → $\begin{cases} \%d & \text{(decimal)} \\ \%lf & \text{(long float)} \end{cases}$

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



1111 \rightarrow 4 bits $\rightarrow 2^4$ values different $\rightarrow 16$

32 bits $\rightarrow 2^{32}$

int (32 bits) $\rightarrow -2^{31} \rightarrow 0 \rightarrow 2^{31}-1$

sizeof(int) $\rightarrow 4$

% .4 lf

└─ con 4 decimals.

#include <math.h>