

Programación en C: Funciones

1ºDAM IoT

Funciones

La programación estructurada busca dividir o descomponer un problema complejo en problemas más pequeños. La solución de cada uno de esos problemas nos trae la solución del problema complejo.

En C podemos dividir nuestro programa en funciones.

El main que escribimos en nuestro programa sería la función principal de nuestro programa y es la primera en ejecutarse.

En nuestros programas podemos hacer uso de funciones ya definidas o crear nuestras propias funciones.

El lenguaje C tiene librerías que proveen algunas funcionalidades básicas. Algunas de ellas ya las utilizamos en conceptos anteriores como son las funciones: scanf, gets, printf.

Funciones

Para declarar una función en C:

```
tipo_valor_retorno identificador_funcion (<parametros>)  
{  
    instrucción 1;  
    instrucción 2;  
    ...  
    instrucción n;  
    return tipo_valor_retorno;  
}
```

```
int suma (int num1, int num2){  
    return num1+num2;  
}
```

Al declarar una función estamos dando a conocer cómo interactúa esa función con el resto del código que desee hacer uso de ella, particularmente:

- Con qué identificador (nombre) ha de invocarse.
- Qué parámetros de entrada necesita para funcionar.
- Cómo es el valor de retorno que devuelve tras su ejecución.

A veces, no conoceremos la implementación de la función, únicamente su declaración. Por ejemplo, scanf, printf, gets.

Funciones

```
#include <stdio.h>

//Mi función suma que devuelve la suma de dos números pasados como parámetros
int suma (int num1, int num2){
    return num1+num2;
}

int main(){
    int numero1, numero2;
    printf("Introduce número 1\n");
    fflush(stdin);
    scanf("%d", &numero1);
    printf("Introduce número 2\n");
    fflush(stdin);
    scanf("%d", &numero2);
    //Suma los valores introducidos por el usuario
    printf("Resultado suma %s\n", suma(numero1, numero2));
    //Suma 1+2
    printf("Resultado suma %s\n", suma(1, 2));
    return 0 ;
}
```

Declaración de la función creada por el programador. Observa que se declara e implementa antes de usarla en el main.

Posibles llamadas a la función.

Funciones de la librería Math.h

Funciones con math.h

La librería **math.h** ofrece funciones matemáticas para utilizar en nuestros programas.

Por ejemplo, dispone de la función **pow** para elevar un número por otro dado.

Antes de usar las funciones de esta librería debemos incluirla en el programa para que el compilador la reconozca. Observa que no sabemos cómo se ha implementado pero tampoco nos interesa.

Para usar **pow** tienes que añadir la librería **math.h**

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

int main()
{
    int x = 5;
    int y = 2;
    printf("5 elevado a 2 es %lf", pow(x, y));

    return 0;
}
```

%lf para imprimir un double.

Función **pow**: Espera dos números como parámetro y devuelve el resultado de **x** elevado a **y**.

Funciones con math.h

Cuando usamos una función podemos pasarle como argumento el nombre de una variable o un literal.

En el caso de la función `pow` vemos que devuelve un número que imprimimos con `f` pero también podríamos guardarlo en una variable y luego imprimir su valor.

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

int main(){
    //Le paso a la función pow los literales directamente
    printf("5 eleva a 2 es %lf\n", pow(5,2));
    //Guardo el número que me devuelve pow en una variable
    double resultado = pow(5,2);
    printf("5 eleva a 2 es %lf\n", resultado);
    return 0 ;
}
```

Funciones con math.h

Ejercicio: Solicita al usuario el valor del radio de una circunferencia. Calcula el área de la circunferencia. π lo vamos a fijar en 3.141592, utiliza una constante. Utiliza la función `pow` de `math.h` para calcular el área.

Funciones con math.h

Solución:

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

#define PI 3.141592

int main(){
    float area, radio;
    printf("Introduzca el radio de la circunferencia: ");
    scanf("%f", &radio);
    //Área de la circunferencia utilizando pow
    area = PI * pow(radio, 2); ←
    printf("El área de la circunferencia es: %f\n", area);
    return 0 ;
}
```

Funciones con math.h

Algunas funciones útiles:

double **ceil**(double x); → Menor entero no menor que x.

double **floor**(double x); → Mayor entero no mayor que x.

double **fabs**(double x); → Valor absoluto de x

double **sqrt**(double x); → Calcula la raíz cuadrada del valor no negativo de x. Puede producirse un error de dominio si x es negativo.

https://informatica.uv.es/mguia/asignatu/INF/2003_04/PR7/pract07_2003.pdf

Funciones con math.h

Ejercicio: Solicita un número decimal al usuario e imprime el redondeo al alza y a la baja del número. Haz uso de las funciones `ceil` y `floor` de `math.h` para calcularlo.

Funciones con math.h

Solución:

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

int main(){
    int redondeo;
    float numero;
    printf("Introduzca un número con decimales: ");
    scanf("%f", &numero);
    redondeo = ceil(numero);
    printf("Redondeo al alza: %d\n", redondeo);
    redondeo = floor(numero);
    printf("Redondeo a la baja: %d\n", redondeo);
    return 0 ;
}
```

```
Introduzca un número con decimales: 3.65
Redondeo al alza: 4
Redondeo a la baja: 3
```

Funciones con cadenas de la librería string.h

Funciones con cadenas

La librería `string.h` nos proporciona funciones para tratar más fácilmente las cadenas.

Por ejemplo, la función `strlen` recibe como argumento una cadena y devuelve el número de caracteres de la cadena. No devuelve el tamaño, sino el número de caracteres hasta encontrar el `\0`.

Incluimos la librería `string.h`

```
#include <stdio.h>
#include <string.h>
int main()
{
    char palabra[31];
    printf("Ingrese una palabra:");
    gets(palabra);
    int cant = strlen(palabra);
    printf("La palabra %s tiene %i letras", palabra, cant);
    return 0;
}
```

Dada la cadena devuelve un entero con el número de caracteres.

Funciones con cadenas

Otras funciones interesantes son:

- **strcpy(cadena1, cadena2)** : Copia el contenido de cadena2 en cadena1.
- **strcat(cadena1, cadena2)** : Concatena el contenido de 2 al final de 1.
- **strcmp(cadena_1, cadena_2)** : Compara las dos cadenas y devuelve:
 - 0 si las dos cadenas son exactamente iguales.
 - valor > 0 si la cadena1 es mayor alfabéticamente que la segunda.
 - valor < 0 si la cadena2 es mayor alfabéticamente que la primera.

Funciones con cadenas

Ejercicio: Declara dos arrays de caracteres de tamaño 40. Pide al usuario que te diga una palabra y guardala en el primer array. Copia el contenido del primer array en el segundo. Imprime por pantalla el contenido del segundo array.

Funciones con cadenas

Solución:

```
#include <stdio.h>
#include <string.h>
#define TAM 40
int main(){
    char array1[TAM];
    char array2[TAM];
    printf("Dime una palabra\n");
    fflush(stdin);
    gets(array1);
    //Copiamos la palabra en el segundo array
    strcpy(array2, array1);
    printf("El segundo array --> %s\n", array2);
    return 0 ;
}
```

Incluyo la librería string.h

Copio el contenido de array1 en array2.

Funciones con cadenas

Ejercicio: Solicita al usuario dos cadenas de caracteres que como máximo tendrán 100 caracteres. Imprime por pantalla si las cadenas son iguales y si no lo son concatena una a la otra e imprímela por pantalla.

Funciones con cadenas

Solución:

```
#include <stdio.h>
#include <string.h>
#define TAM 10
int main()
{
    char cadena1[TAM];
    char cadena2[TAM];
    printf("Introduzca la primera cadena\n");
    fflush(stdin);
    gets(cadena1);
    printf("Introduzca la segunda cadena\n");
    fflush(stdin);
    gets(cadena2);
    if (strcmp(cadena1, cadena2) == 0)
    {
        printf("Las cadenas son iguales\n");
    }
    else
    {
        strcat(cadena1, cadena2);
        printf("Cadena completa %s\n", cadena1);
    }

    return 0;
}
```

strcmp devuelve 0 si son iguales.

cadena1 guarda la
cadena1+cadena2

Funciones valores aleatorios de la librería stdlib.h

Funciones random/valores aleatorios

Para generar números aleatorios utilizamos funciones de la **librería stdlib.h**.

int rand()

Devuelve un entero pseudoaleatorio mayor o igual que 0 y menor o igual que RAND_MAX. RAND_MAX es una constante entera predefinida en stdlib.h.

El valor de RAND_MAX depende de la implementación pero por lo menos es 32767.

Funciones random/valores aleatorios

Ejemplo:

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int i;
    printf("El mayor número random a generar %d \n", RAND_MAX);
    //Vamos a generar 5 números aleatorios de 0 a RAND_MAX
    for(i=0;i<5;i++){
        printf("%i\n", rand());
    }
    return 0 ;
}
```

Problema: ¿Qué ocurre si vuelvo a ejecutar mi programa? ¿Qué números aleatorios se generan?

```
El mayor número random a generar 32767
41
18467
6334
26500
19169
```

Funciones random/valores aleatorios

Si ejecuto varias veces mi programa siempre me genera los mismo números aleatorios.

Primera ejecución

```
El mayor número random a generar 32767  
41  
18467  
6334  
26500  
19169
```

Segunda ejecución

```
El mayor número random a generar 32767  
41  
18467  
6334  
26500  
19169
```

El problema es que para generar los números aleatorios siempre se ejecutan las mismas operaciones, debemos introducir una “semilla” que varíe las operaciones que realiza la función rand.

Funciones random/valores aleatorios

El problema es que para generar los números aleatorios siempre se ejecutan las mismas operaciones, debemos introducir una “semilla” que varíe las operaciones que realiza la función rand.

Solución → void srand(unsigned int n)


- Reinicializa el generador de números aleatorios.
- El argumento es la semilla.
- Debemos llamar a esta función antes que a rand().

Funciones random/valores aleatorios


Modifico el programa anterior para que introduzca la semilla 2.

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int i;
    printf("El mayor número random a generar %d \n", RAND_MAX);
    srand(2); //Semilla 2
    //Vamos a generar 5 números aleatorios de 0 a RAND_MAX
    for(i=0;i<5;i++){
        printf("%i\n", rand());
    }
    return 0 ;
}
```

Pero si lo ejecuto varias veces verás que siempre obtenemos los mismos números. ¿iQué semilla utilizamos!?



```
El mayor número random a generar 32767
45
29216
24198
17795
29484
```



```
El mayor número random a generar 32767
45
29216
24198
17795
29484
```

Funciones random/valores aleatorios

¡El tiempo! El tiempo cambia constantemente.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(){
    int i;
    printf("El mayor número random a generar %d \n", RAND_MAX);
    printf("El valor de time(NULL): %d", time(NULL));
    srand (time(NULL)); //Semilla tiempo
    //Vamos a generar 5 números aleatorios de 0 a RAND_MAX
    for(i=0;i<5;i++){
        printf("%i\n", rand());
    }
    return 0 ;
}
```

Incluimos la librería time.h

time(NULL) devuelve el tiempo en segundos.

Distintos números aleatorios en cada ejecución

```
El mayor número random a generar 32767
El valor de time(NULL): 1699359204276
3895
4471
20783
6481
```

```
El mayor número random a generar 32767
El valor de time(NULL): 16993597131938
2578
20598
13778
13719
```

Funciones random/valores aleatorios

Ejercicio: También puedes utilizar getpid() de la librería process.h. Investiga que te devuelve esta función y por qué cada vez te da un número distinto que podemos utilizar como semilla. Limita ahora el número de aleatorios a un rango de 0 al 5. ¿Cómo lo harías?

Pista:

Para generar números aleatorios de 0 a N utilizamos esta fórmula `rand() % (N+1);`

Y para generar números aleatorios de N a M ambos inclusivos utilizamos esta fórmula `rand () % (M-N+1) + N;`

Funciones random/valores aleatorios

Solución:

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

#include <process.h>
int main(){
    int i;

    printf("El valor de getpid(): %d",getpid());
    srand (getpid()); //Semilla tiempo
    //Vamos a generar 5 números aleatorios de 0 a 5
    for(i=0;i<5;i++){
        printf("%i\n", rand()%6);
    }

    return 0 ;
}
```

```
El valor de getpid(): 3263925
5
0
2
4
```

```
El valor de getpid(): 95445
0
2
1
1
```

```
El valor de getpid(): 153840
3
4
5
4
```