El lenguaje de programación C
- Funciones -

Introducción

- Las funciones nos permiten modularizar nuestro código
  - Haciéndolo más legible
  - Evitando repeticiones innecesarias
  - Facilitando la reutilización del código
- Se utilizan para realizar tareas concretas
  - Que dependan de un determinado número de parámetros (argumentos)
- Ya hemos usado varias: `printf()`, `scanf()`, `sqrt()`…
- Ya hemos implementado una: `main()`
- Las funciones, como las variables, hay que declararlas antes de usarlas
  - También hay que implementarlas
  - La implementación sirve como declaración

Elementos de una función

- Nombre:
  - Un identificador
- Argumentos:
  - Son los distintos parámetros o valores que necesita la función para hacer sus operaciones
  - Se especifica el tipo de cada uno de ellos
  - Por defecto la función recibe una copia de cada parámetro → Paso de argumentos por valor
    - Si modificamos el valor de la variable dentro de la función, la variable externa permanece igual
- Resultado:
  - Es el valor que devuelve la función una vez ejecutada
  - Se especifica el tipo que va a tener
  - Solo hay uno
  - Si no se espera ningún valor se usa `void`

Declarando una función

- Estructura básica

```
tipo nombrefunción(tipo1 arg1, tipo2 arg2,…)
```

- Tipo: `void` (si no devuelve nada), `int`, `float`, etc…
- nombrefunción: Un identificador único
- $Tipo_i$: Tipo de cada uno de los argumentos
- Arg_i: Nombre de cada uno de los argumentos
  - Es el nombre que tendrá la variable que internamente usemos en la función
  - No son necesarios en la declaración

Declarando una función: Ejemplos

```
/*Una funcion que no toma argumentos y no
devuelve nada*/
void f1();

/*Una funcion que no toma argumentos y no
devuelve nada*/
float random();

/*Una funcion que no toma argumentos y devuelve un
numero real*/
void muestra(float x, float y);

/*Una funcion que toma argumentos dos argumentos y
devuelve un numero real de precision doble*/
double serie(double k, int i);
```

Implementando una función

- Se hace de manera muy similar a la función `main()`
- Para devolver un resultado se utiliza `return` seguido del valor que se quiere devolver
  - Si la función no devuelve nada, se usar `return` sólo
- Ejemplo:

```
/*Función que devuelve el termino i de la serie*/
double serie(double k, int i) {
  double termino = k/i;
  return termino;
}
```

Sí, pero ¿dónde?

- Las declaraciones de las funciones han de hacerse **antes de utilizarlas por primera vez**
  - Normalmente se hacen justo después de los includes
  - … o en un archivo externo que se incluye → Lo veremos más adelante
- Las implementaciones de las funciones se pueden hacer **en cualquier parte**
  - Recomiendo hacerlo al final, después de la función `main()`
  - … o en un archivo externo que incluye el de las declaraciones y que es compilado aparte → Lo veremos más adelante

Ejemplo:

```
int bisiesto(int a);
int main() {
    //Declaración de variables
int anio;
printf ("Introduzca el anio ");     scanf ("%d",
&anio);
if (bisiesto(anio))
    printf ("Bisiesto ");
else
        printf ("Bisiesto ");
    return 0;
}

int bisiesto(int a)
{
if(a%4==0 and a%100!=0 or a%400==0)
                return 1;
        else
                return 0;
}
```

Ejemplo:

```
Void Nombre_Proc (Parámetros)
{
        instrucciones;
}

Tipo_de_Dato Nombre_Func(parametros)
{
        Instrucciones;
        return;
}
```

Pasaje de parámetros:

Existen dos tipos de paso de parámetros:

- **Referencia**: apuntan a una localidad de memoria lo que les da la capacidad de crear o modificar los datos.

- **Valor:** solo copia el contenido para ser utilizado dentro de la abstracción. Se mantiene su valor original, una vez terminadas las acciones del procedimiento

Pasaje de parámetros:

En el caso de paso de parámetros por referencia se antepone un

- Asterisco (*) al nombre de la variable.
- ampersand (&) al nombre de la variable al llamar.
- En el caso de paso de parámetros por valor, sólo se pone el nombre de la variable.
- Por valor **int Mod(int variable)**
- Por referencia **int Mod(int *variable)**

Pasaje de parámetros:

```
void ordenarTres(int& valUno, int& valDos, int&
valTres)
{
   int auxiliar;

   if (valUno > valDos) {
      auxiliar = valUno;
      valUno = valDos;
      valDos = auxiliar;
   }

   if (valTres < valUno) {
      auxiliar = valTres;
      valTres = valDos;
      valDos = valUno;
      valUno = auxiliar;
   }   else if (valTres < valDos) {
      auxiliar = valDos;
      valDos = valTres;
      valTres = auxiliar;
   }
}
```

Ejemplo:

```
void cubo(float x, float *res)
 {
                    res=(x*x*x);
}

float cubo(float x)
 {
        return (x*x*x);
}
-----------------------------------------
cubo(x1, &res); //llamando el procedimiento

res= cubo(x1); //llamando la funcion
```

Ejemplo:

```
int bisiesto(int a)   //definición de la función
{
   if(a%4==0 and a%100!=0 or a%400==0)
      return 1;
   else
      return 0;
```

Ejemplo:

```
int fecha_valida(int d, int m, int a) //definición de fecha_valida
{
   if(d < 1 or d > 31 or m < 1 or m > 12 or a < 1)
   {
      return 0;
   }
   switch(m)
   {
      case 4:
      case 6:
      case 9:
      case 11: if(d > 30)
               {
                  return 0;
               }
               break;
      case 2: if(bisiesto(a))
              {
                 if(d > 29)
                 {
                    return 0;
                 }
              }
              else if(d > 28)
                  {
                     return 0;
                  }
              break;
   }
   return 1;
}
```

Ejemplo:

```
int calcular_edad(int da, int ma, int aa, int dn,
int mn, int an)
{
    int edad = aa - an;
    if(ma < mn)
       edad--;
    else if(ma == mn and da < dn)
           edad--;
    return edad;

}
```