UNIVERSIDADE DA CORUÑA

TEMA 7

Funciones

Grado en Ingeniería Eléctrica
Grado en Ingeniería Electrónica Industrial y Automática

Escuela Politécnica de Ingeniería de Ferrol
https://www.udc.es/epef

Índice

7.1.- Introducción

7.2.- Estructura y definición de una función

7.3.- El ámbito de las variables

7.4.- Paso de parámetros: por valor y referencia

7.5.- Anexo: ejercicios resueltos

7.1.- Introducción

- En el lenguaje C un programa puede dividirse en distintos bloques (módulos) denominados **funciones**

- Cada una de las funciones puede ser llamada, enviándole unos datos (o sin enviarle nada), para que realice una determinada tarea y/o proporcione unos resultados

- Beneficios del uso de funciones:
  - Aislar mejor los problemas
  - Desarrollar programas mucho más fáciles de mantener
  - Reutilizar código entre distintos programas

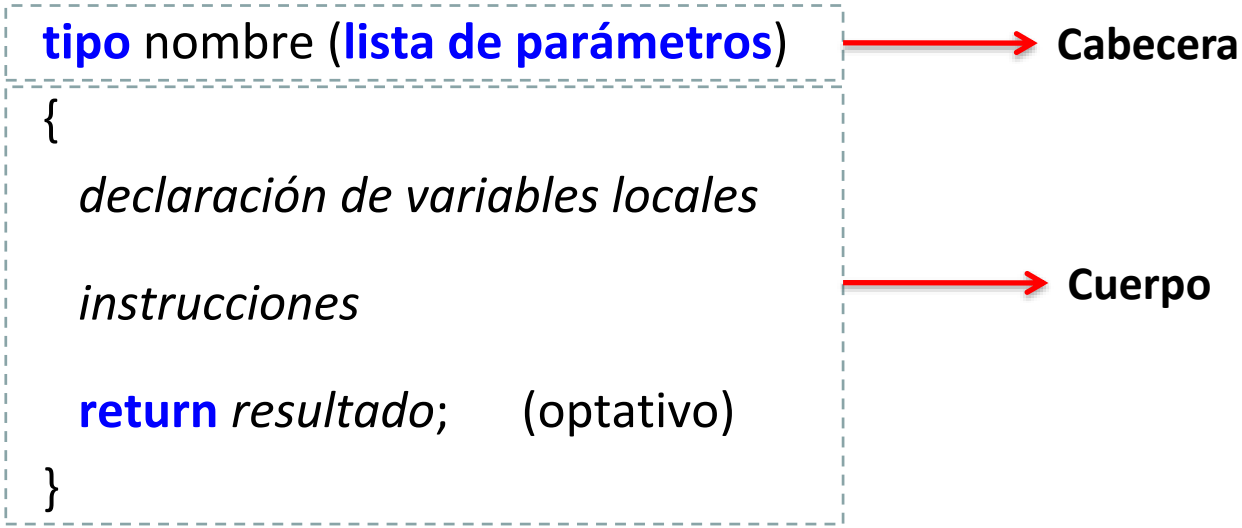
7.2.- Estructura y definición de una función

- Una función consta primero de una ***cabecera*** que contiene:
  - El tipo de dato del valor devuelto por la función
  - El nombre asignado a la función
  - Los argumentos que recibe (datos de entrada a la función)

- A continuación se incluye el ***cuerpo*** de la función:
  - Conjunto de instrucciones que realizan la tarea asignada a la función

7.2.- Estructura y definición de una función

- Definición:



- El **tipo** especifica el tipo de dato del valor devuelto como resultado

  Puede ser cualquier tipo (char, int, float, etc.) y también **void** (indica que no devuelve nada)

- La **lista de parámetros** especifica los datos que se reciben como entrada (y su tipo) separados por comas

7.2.- Estructura y definición de una función

- Sentencia return:
  - Permite finalizar la función y devolver un valor como resultado
  - Sintaxis: **return** *expresión*;
  - Ejemplos:
    - return 3;
    - return ‘e’;
    - return dato;
    - return a+b;
  - Si en la función se especifica que el tipo de dato del resultado es **void** (no hay dato a devolver) no hay que emplear la sentencia return

7.2.- Estructura y definición de una función

- Ejemplos de definición de funciones:

Recibe dos datos como entrada

La función devuelve un valor de tipo entero

```
int mi_funcion(int a, int b)
{
   int suma;

   suma = a*a + b*b;
   return suma;
}
```

No recibe datos de entrada

La función devuelve un valor de tipo flotante

```
float leer_datos()
{
   float dato;

   printf("\nIntroduce un dato: ");
   scanf("%f", &dato);

   return dato;
}
```

7.2.- Estructura y definición de una función

- Ejemplo de uso y llamada de funciones:

```c
#include <stdio.h>

int potencia(int x, int y)
{
  int resultado=1, cont;

  for (cont=1; cont<=y; cont++)
    resultado = resultado * x;
  return resultado;
}

void main()
{
  int dato1, dato2, potencia1, potencia2;

  printf("\nIntroduce un dato positivo: ");
  scanf("%i", &dato1);
  printf("\nIntroduce otro dato positivo: ");
  scanf("%d", &dato2);
  potencia1=potencia(dato1,dato2);
  potencia2=potencia(dato2,dato1);
  printf("\nResultado: %.2f", (float) potencia1/potencia2);
}
```

Definición de la función.
Antes de la función principal (main)

El programa empieza a ejecutarse en la función principal (main)

Primera llamada a la función
Es en ese momento cuando se ejecuta la función

Segunda llamada a la función
Se ejecuta de nuevo la función

7.2.- Estructura y definición de una función

- Alternativa en el uso y llamada de funciones (con prototipos):

```c
#include <stdio.h>

int potencia(int x, int y);

void main()
{
  int dato1, dato2, potencia1, potencia2;

  printf("\nIntroduce un dato positivo: ");
  scanf("%i", &dato1);
  printf("\nIntroduce otro dato positivo: ");
  scanf("%d", &dato2);
  potencia1=potencia(dato1,dato2);
  potencia2=potencia(dato2,dato1);
  printf("\nResultado: %.2f", (float) potencia1/potencia2);
}

int potencia(int x, int y)
{
  int resultado=1, cont;

  for (cont=1; cont<=y; cont++)
    resultado = resultado * x;
  return resultado;
}
```

Como la función está definida después del *main* es necesario incluir antes el prototipo de la función (poniendo la cabecera con un ;)

Definición de la función.
Después de la función principal (main)

7.3.- El ámbito de las variables

- Los parámetros de la función y las variables definidas en ella son **variables locales**
- La función **main** también puede tener variables locales → inaccesibles a otras funciones
- Variables **globales** (no son aconsejables por lo que se sebe evitar su uso):
  - Declaradas fuera de cualquier función
  - Tienen un valor inicial pero es indeterminado (en algunos casos 0)
- Variables **locales**:
  - Sólo las conoce la función que las declara
  - Tiene un valor inicial indeterminado
  - Cuando se termina la función su valor se pierde (aunque se vuelva a llamar a la función)

7.3.- El ámbito de las variables

- Ejemplo con variables locales y globales:

```
#include "stdio.h"

int i, n;

long int sumatorio(int x)
{
  int i;
  long int s;

  for (i=1,s=0;i<=x;i++)
    s = s+i;
  return s;
}

void main()
{
  long int suma;

  printf("\nEscribe un numero: ");
  scanf("%i", &n);
  for (i=1; i<=n; i++) {
    suma = sumatorio(i);
    printf("Los numeros de 1 hasta %i suman %li\n", i, suma);
  }
}
```

Variables globales (definidas fuera de cualquier función).
Pueden ser empleadas por cualquiera de ellas

Hace referencia a la variable local

Hace referencia a la variable global

7.4.- Paso de parámetros: por valor y referencia

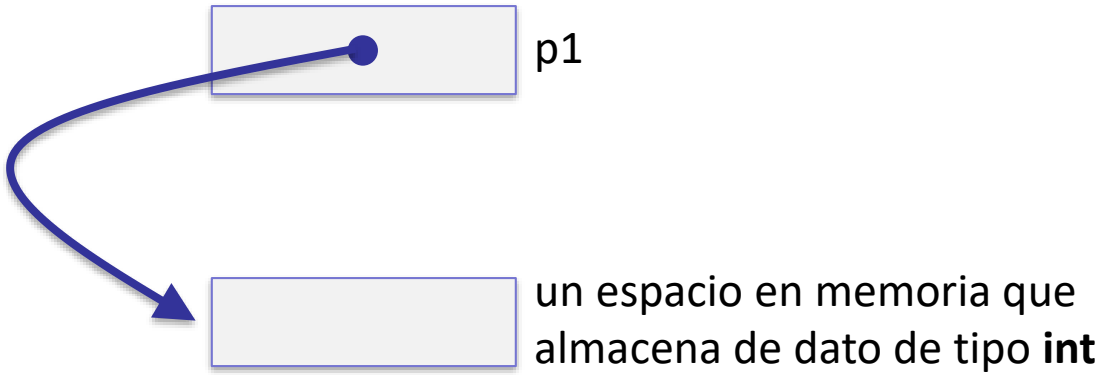
- Paso de parámetros por valor:
  - La función recibe una copia de los valores de los parámetros
  - Si se cambia el valor de un parámetro el cambio sólo afecta a la función y no tiene efecto fuera de ella

- Paso de parámetros por referencia:
  - La función recibe un **puntero** (dirección de memoria) de cada parámetro
  - La función puede acceder a la variable y cambiar su valor
  - Se usa cuando la función debe modificar el valor del parámetro
  - Para pasar parámetros por referencia:
    - Se les antepone el símbolo & en la llamada a la función
    - Se declaran los parámetros con el símbolo * delante del nombre

7.4.- Paso de parámetros: por valor y referencia

- ¿Qué es un puntero?:
  - Es un tipo especial de variable que almacena el valor de una ***dirección de memoria***
  - Ejemplos:

```
int *p1;      /* Declara un puntero a entero */
char *p2;     /* Puntero a un tipo carácter */
```



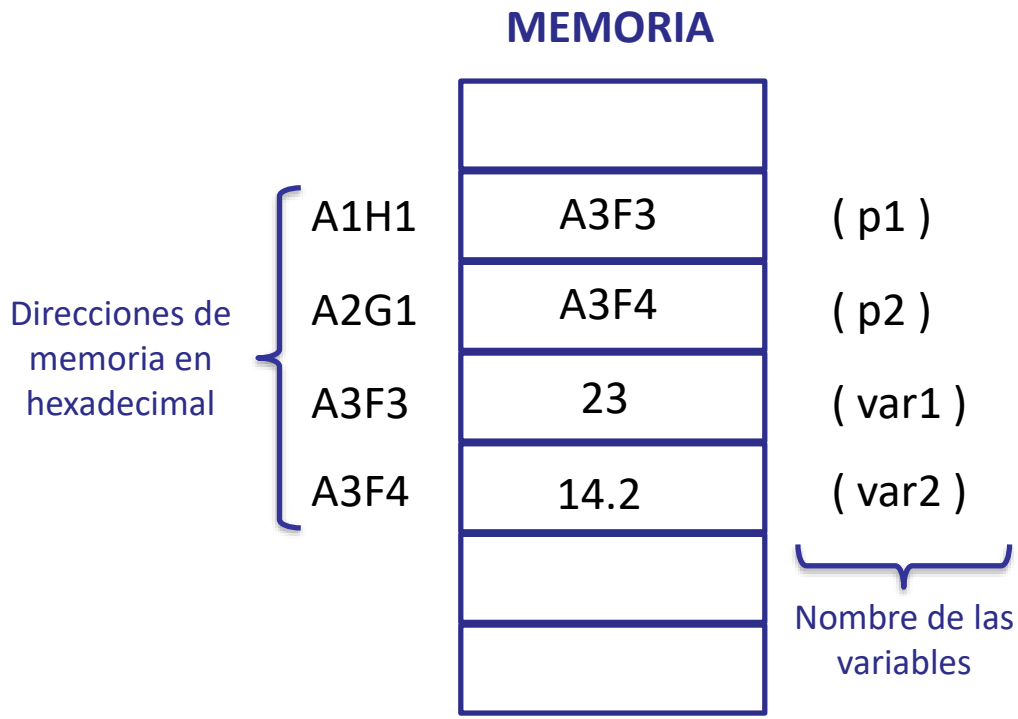
7.4.- Paso de parámetros: por valor y referencia

- Ejemplo:

```
int *p1, var1=23;
float *p2, var2=14.2;
p1 = &var1;
p2 = &var2;
```

- ¿Resultados?:

```
printf("%i",p1);
printf("%i",*p1);
*p2 = 15;
```



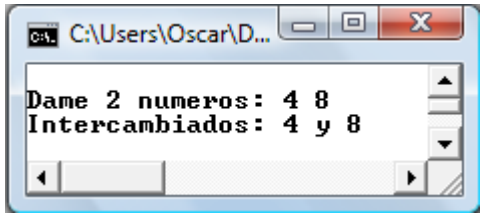
- ¿Qué sucede si se realiza p1 = var1?

7.4.- Paso de parámetros: por valor y referencia

Paso por valor

```c
#include <stdio.h>

void intercambia(int a, int b)
{
  int aux;

  aux = a;
  a = b;
  b = aux;
}

void main()
{
  int d1, d2;

  printf("\nDame 2 numeros: ");
  scanf("%i%i", &d1, &d2);
  intercambia(d1,d2);
  printf("Intercambiados: %i y %i", d1, d2);
}
```

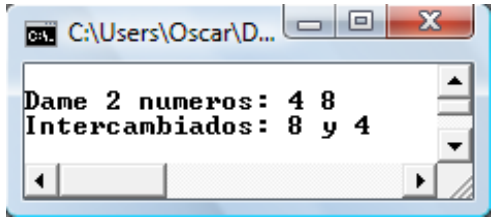
Resultado:



Paso por referencia

```c
#include <stdio.h>

void intercambia(int *a, int *b)
{
  int aux;

  aux = *a;
  *a = *b;
  *b = aux;
}

void main()
{
  int d1, d2;

  printf("\nDame 2 numeros: ");
  scanf("%i%i", &d1, &d2);
  intercambia(&d1,&d2);
  printf("Intercambiados: %i y %i", d1, d2);
}
```

Resultado:



7.4.- Paso de parámetros: por valor y referencia

- Ejemplo: transformación de coordenadas cartesianas a polares

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

void leer_coordenadas (float *coor_x, float *coor_y)
{
  printf("\nIntroduce la coordenada x: ");
  scanf("%f", coor_x);
  printf("\nIntroduce la coordenada y: ");
  scanf("%f", coor_y);
}

void convertir_coordenadas(float x, float y, float *ang, float *rad)
{
  *rad = sqrt(x*x+y*y);
  *ang = atan(y/x);
}

void main()
{
  float x, y, angulo, radio;

  leer_coordenadas(&x, &y);
  convertir_coordenadas(x,y,&angulo,&radio);
  printf("\nLas coordenadas polares son (%.2f, %.2f)", radio, 180*angulo/M_PI);
}
```

Definimos ambos argumentos de la función como punteros porque se van a modificar (leer) en su interior

La variable es un puntero, por tanto no se usa el operador de dirección de memoria (&)

Sólo estos argumentos se definen como punteros porque se van a usar para cambiar las variables del main que contendrán los resultados

Transformación de radianes a grados

Constante definida en la librería math.h

7.4.- Paso de parámetros: por valor y referencia

- Ejercicio: ¿qué valores muestra este programa en pantalla?

```
#include "stdio.h"

int a, b;

void cambiar(int *c, int d)
{
   *c = *c*2;
   b = b+4;
   d = d+1;
   a = a*2;
}

void main()
{
   a=2;
   b=3;
   cambiar(&b, a);
   printf("\n%i, %i\n", a, b);
}
```

Respuesta:

a) 4, 7

b) 2, 6

c) 4, 10

d) Ninguna de las anteriores

7.4.- Paso de parámetros: por valor y referencia

- Independencia de funciones:
  - Como regla general: sólo debe usarse en una función los parámetros que recibe y las variables locales que necesite
  - No es recomendable el uso de variables globales

UNIVERSIDADE DA CORUÑA

ANEXO:
Ejercicios resueltos

Ejercicio 1: enunciado

- Realizar un programa que permita calcular la *n*-ésima potencia de un binomio (*x* + *y*) mediante la siguiente fórmula:

$$(x+y)^n = \sum_{k=0}^{n} \frac{n!}{k!(n-k)!} x^{n-k} y^k$$

El programa pedirá al usuario los tres valores necesarios (*x*, *y*, *n*). Todos ellos deben ser números enteros positivos, en caso contrario se mostrará un mensaje de error y se pedirá de nuevo el dato correspondiente

- Ejemplo de funcionamiento:

```
Introduce el valor de x: -3
Error, introduce de nuevo el dato: -4
Error, introduce de nuevo el dato: 2
Introduce el valor de y: 3
Introduce el valor de n: 2
Resultado = 25
```

Ejercicio 1: una posible solución (parte I)

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

int leer_positivo()
{
  int dato;

  do {
    scanf("%i", &dato);
    if (dato<=0)
      printf("Error, introduce de nuevo el dato: ");
  } while (dato<=0);

  return dato;
}

long factorial (int numero)
{
  long factorial=1;
  int i;

  for(i=2; i<=numero; i++)
    factorial=factorial*i;

  return factorial;
}
```

Primera función: lee de teclado un dato entero, que debe ser positivo, y devuelve como resultado ese valor leído. Esta función no tiene ningún argumento de entrada porque no necesita recibir ningún dato para realizar esa operación

Segunda función: se le proporciona un dato como argumento de entrada (en la variable numero) y devuelve como resultado el factorial de ese número. Como el factorial puede ser un número grande el resultado que devuelve la función se almacena en una variable long

Inicialización: se inicializa el factorial a 1 para luego ir multiplicando por el resto de términos e ir actualizando este valor

El bucle se repite numero-1 veces (desde 2 hasta numero) y en cada paso del mismo se multiplica el valor actual del factorial (resultado parcial) por el siguiente número de la secuencia

Ejercicio 1: una posible solución (parte II)

```
int main()
{
  int x, y, n, k;
  long resultado=0, factorial_n;

  printf("Introduce el valor de x: ");
  x = leer_positivo();
  printf("Introduce el valor de y: ");
  y = leer_positivo();
  printf("Introduce el valor de n: ");
  n = leer_positivo();

  factorial_n = factorial(n);

  for (k=0; k<=n; k++)
    resultado = resultado + pow(x,n-k)*pow(y,k)*factorial_n/(factorial(k)*factorial(n-k));

  printf("\nResultado = %li\n", resultado);

  return 0;
}
```

Se llama a la función leer_positivo para que lea un primer dato por teclado y devuelva ese valor. Ese primer dato se almacena en la variable x de la función principal (main)

Se llama de nuevo a la función leer_positivo para que lea un segundo dato por teclado y devuelva ese valor. Ese segundo dato se almacena en la variable (y) de la función principal (main)

Como este cálculo es siempre el mismo (no depende del valor de k del bucle) se hace una única vez antes del bucle y se almacena ese valor en la variable factorial_n (que se usa luego dentro del bucle)

Llamada a la función pow (disponible en la librería matemática math.h) que calcula, en ese caso el resultado del valor de la variable (y) elevado a k

Llamada a la función factorial (definida en la transparencia anterior) para que calcule el factorial de k

Otra llamada a la función factorial pero ahora se calcula el factorial del número n-k

Ejercicio 2: enunciado

- Realizar un programa que muestre por pantalla los dos primeros números que son múltiplos de 5 entre dos dados por el usuario. El primer número solicitado debe ser positivo y el segundo estrictamente mayor que el anterior, si alguno de ellos no cumple la condición indicada se le pedirá de nuevo al usuario hasta que introduzca uno correcto. Para mostrar el resultado en pantalla habrá que indicar, con un mensaje diferente, el caso en el que existan los dos múltiplos, que sólo exista uno o que no exista ninguno

- Ejemplo de funcionamiento:

```
Introduce el primer dato (positivo): -3
Introduce el primer dato (positivo): 11
Introduce el segundo dato (mayor que el anterior): 8
Error, el dato debe ser mayor que 11.
Introduce el segundo dato (mayor que el anterior): 30
Entre 11 y 30 los dos primeros múltiplos de 5 son el 15 y el 20.
```

Ejercicio 2: una posible solución (parte I)

```c
#include <stdio.h>

void pedir_datos(int *dato1, int *dato2)
{
  do{
    printf("\nIntroduce el primer dato (positivo): ");
    scanf("%i", dato1);
  } while (*dato1<=0);
  do{
    printf("\nIntroduce el segundo dato (mayor que el anterior): ");
    scanf("%i", dato2);
    if (*dato2<=*dato1)
      printf("Error, el dato debe ser mayor que %i.", *dato1);
  } while (*dato2<=*dato1);
}
void buscar_multiplos(int *dato1, int *dato2)
{
  int primero=0, segundo=0, i;

  for (i=*dato1; i<=*dato2 && primero==0; i++)
    if (i%5==0)
      primero=i;

  if (primero!=0 && primero+5<=*dato2)
    segundo = primero+5;

  *dato1 = primero;
  *dato2 = segundo;
}
```

Función que pide dos datos por teclado (el segundo debe ser mayor que el primero) y proporciona, a la función que la llama, esos datos mediante punteros (los argumentos de entrada son por referencia)

Función que dados dos datos (los proporcionados por los argumentos dato1 y dato2) calcula cuál es el primer y segundo múltiplo de 5 entre ellos. Al final de la función se sobreescribe el valor de dato1 y de dato2 para meter esos dos resultados y que la función que la llama pueda acceder a esos datos

En este bucle se busca el primer múltiplo de 5 entre dato1 y dato2. El segundo término de la condición (primero==0) se usa para detener el bucle en el momento que se encuentre un múltiplo de 5 (no hace falta seguir buscando más)

Se usan los argumentos por referencia para meter los resultados

El segundo múltiplo, si lo tiene (para ello se usa el if), se obtiene sumando 5 al anterior múltiplo

Ejercicio 2: una posible solución (parte II)

```c
int main()
{
  int dato1, dato2;

  pedir_datos(&dato1, &dato2);

  printf("\nEntre %i y %i ", dato1, dato2);
  buscar_multiplos(&dato1, &dato2);
  if (dato1 == 0)
    printf("no existe ningun multiplo de 5.\n");
  else
    if (dato1 != 0 && dato2 == 0)
      printf("solo hay un multiplo de 5 y es el numero %i.\n", dato1);
    else
      printf("los dos primeros multiplos de 5 son el %i y el %i.\n", dato1, dato2);

  return 0;
}
```

Llamada la función que pide los dos datos por teclado. En los argumentos se pasa la dirección de cada variable ya que en la función están definidos por referencia (punteros)

Llamada a la función que calcula los dos primeros múltiplos de 5 entre dato1 y dato2 (que han sido leídos previamente). Se pasan por referencia porque dentro de la función buscar múltiplos se pretende cambiar el valor de dato1 y dato2 para que contengan (al acabar la función) los dos múltiplos buscados (si alguno no existe devolverá un 0 en alguno de ellos)