Programación I
Funciones

Iván Cantador
Escuela Politécnica Superior
Universidad Autónoma de Madrid

Contenidos

- Funciones: definición, sintaxis, ejemplos
- Alcance de variables
- Paso de argumentos de entrada (I)
  - Paso de argumentos de entrada por valor
  - Paso de argumentos de entrada por referencia
- Paso de argumentos de entrada (II)
  - Paso de arrays de entrada
  - Paso de estructuras de entrada

Contenidos

- **Funciones: definición, sintaxis, ejemplos**
- Alcance de variables
- Paso de argumentos de entrada (I)
  - Paso de argumentos de entrada por valor
  - Paso de argumentos de entrada por referencia
- Paso de argumentos de entrada (II)
  - Paso de arrays de entrada
  - Paso de estructuras de entrada

Funciones: definición, sintaxis, ejemplos (I)

- Una **función** es un bloque de sentencias identificado con un nombre que se ejecutan de manera secuencial ofreciendo una "funcionalidad" dada
  - Ejemplo de definición de función

```
// Funcion que devuelve la suma de dos numeros enteros: a, b
int sumarEnteros(int a, int b) {
   int suma;

   suma = a + b;

   return suma;
}
```

- Una función es como un programa, al poseer argumentos de entrada, sentencias (algoritmo), y retorno de salida
  - De hecho, en C todo programa está constituido por una función principal: **main**

Funciones: definición, sintaxis, ejemplos (II)

- En C la **declaración de una función** es:

```
tipoRetorno nombreFuncion(tipoArg1 arg1, tipoArg2 arg2, ...)
```

- En C una función puede:
  - devolver 0 ó 1 **variable de retorno**
    - En caso de 0, la declaración de la función es con retorno void:

```
void funcion(...)
```

  - recibir 0 ó más **argumentos de entrada**

```
void main(int argc, char *argv[])
int rand()
double sqrt(double x)
int strlen(char *s)
int strcmp(char *s1, char *s2)
char *strcat(char *s1, char *s2)

int sumarEnteros(int a, int b)
char *tituloLibro(Libro l)
```


Funciones: definición, sintaxis, ejemplos (III)

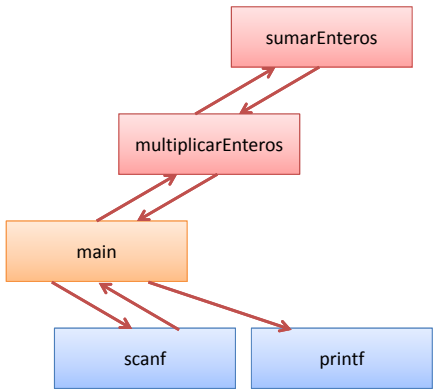
- A la ejecución de una función también se le denomina **invocación de la función**
  - Ejemplo de definición e invocación de función

```
#include <stdio.h>

// Definición de función
int sumarEnteros(int a, int b) {
    int suma;
    suma = a + b;
    return suma;
}

void main() {
    int n1, n2, r;

    printf("Introduce el primer numero: ");
    scanf("%d", &n1);
    printf("Introduce el primer numero: ");
    scanf("%d", &n2);

    r = sumarEnteros(n1, n2);   // Invocación de función

    printf("La suma de %d y %d es %d.\n", n1, n2, r);
}
```


Funciones: definición, sintaxis, ejemplos (IV)

- En definitiva, un programa es una serie de funciones que se invocan unas a otras

```
#include <stdio.h>

int sumarEnteros(int a, int b) {
    int suma;
    suma = a + b;
    return suma;
}
int multiplicarEnteros(int a, int b) {
    int i, producto = 0;
    for ( i = 0; i < b; i++ ) {
        producto = sumarEnteros(producto, a);
    }
    return producto;
}
void main() {
    int n1, n2, r;

    printf("Introduce el primer numero: ");
    scanf("%d", &n1);
    printf("Introduce el primer numero: ");
    scanf("%d", &n2);

    r = multiplicarEnteros(n1, n2);

    printf("El producto entre %d y %d es %d.\n", n1, n2, r);
}
```



Funciones: definición, sintaxis, ejemplos (V)

- Un programa C está constituido por una función principal, llamada **main**, que es la que se invoca al ejecutar el programa
  - El **valor de retorno** de main puede ser:
    - **ninguno**, indicado como void
    - **un entero**, normalmente usado para devolver un código de ejecución correcta o incorrecta del programa
  - Los **argumentos de entrada** de main pueden ser:
    - **ninguno**
    - un entero **argc** que es el número de parámetros del programa + 1, y un array de cadenas de caracteres **argv** que contiene en argv[0] el nombre del programa, en argv[1] el primer parámetro del programa, en argv[2] el segundo parámetro de entrada, …, en argv[argc-1] el último parámetro de entrada

```
void main(int argc, char *argv[])
```

Funciones: definición, sintaxis, ejemplos (VI)

- Un programa C está constituido por una función principal, llamada **main**, que es la que se invoca al ejecutar el programa

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

void main(int argc, char *argv[]) {
    int i, n;

    printf("Los argumentos del programa %s se listan abajo.\n", argv[0]);

    for ( i = 1; i < argc; i++ ) {
        printf("Argumento %d: %s.\n", i, argv[i]);
    }

    // Asumiendo que el 1er parametro es un entero hacemos su conversion a int
    if ( argv[1] != NULL ) {
        n = atoi(argv[1]);
        printf("Primer parametro: %d.\n", n);
    }
}
```


Contenidos

- Funciones: definición, sintaxis, ejemplos
- **Alcance de variables**
- Paso de argumentos de entrada (I)
  - Paso de argumentos de entrada por valor
  - Paso de argumentos de entrada por referencia
- Paso de argumentos de entrada (II)
  - Paso de arrays de entrada
  - Paso de estructuras de entrada

Alcance de variables (I)

- En C los argumentos de entrada y las variables (internas) de una función son **locales**, esto es, son accesibles sólo dentro de la función

```
#include <stdio.h>

int sumarEnteros(int a, int b) {
    int suma;              // Variable suma solo accesible en sumarEnteros
    suma = a + b;
    return suma;
}

void main() {
    int n1, n2;

    printf("Introduce el primer numero: ");
    scanf("%d", &n1);
    printf("Introduce el primer numero: ");
    scanf("%d", &n2);

    suma = sumarEnteros(n1, n2); // ERROR! Variable suma NO declarada en main

    printf("La suma de %d y %d es %d.\n", n1, n2, suma);
}
```


Alcance de variables (II)

- En C los argumentos de entrada y las variables (internas) de una función son **locales**, esto es, son accesibles sólo dentro de la función

```
#include <stdio.h>

int sumarEnteros(int a, int b) {
    int suma;              // Variable suma solo accesible en sumarEnteros
    suma = a + b;
    return suma;
}

void main() {
    int n1, n2, suma;              // Variable suma solo accesible en main

    printf("Introduce el primer numero: ");
    scanf("%d", &n1);
    printf("Introduce el primer numero: ");
    scanf("%d", &n2);

    suma = sumarEnteros(n1, n2); // Variable suma de main, no de sumarEnteros

    printf("La suma de %d y %d es %d.\n", n1, n2, suma);
}
```

Alcance de variables (III)

- En C se pueden usar variables **globales**, que se declaran fuera de las funciones de un programa y que pueden ser accesibles desde cualquiera de estas últimas
  - **¡¡¡NO SE RECOMIENDA EL USO DE VARIABLES GLOBALES!!!**

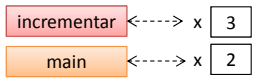
```
#include <stdio.h>

int suma;        // Variable suma global, accesible desde todas las funciones

void sumarEnteros(int a, int b) {
   suma = a + b;
}

void main() {
   int n1, n2;

   printf("Introduce el primer numero: ");
   scanf("%d", &n1);
   printf("Introduce el primer numero: ");
   scanf("%d", &n2);

   sumarEnteros(n1, n2);

   printf("La suma de %d y %d es %d.\n", n1, n2, suma);
}
```


Contenidos

- Funciones: definición, sintaxis, ejemplos
- Alcance de variables
- **Paso de argumentos de entrada (I)**
  - **Paso de argumentos de entrada por valor**
  - Paso de argumentos de entrada por referencia
- Paso de argumentos de entrada (II)
  - Paso de arrays de entrada
  - Paso de estructuras de entrada

Paso de argumentos de entrada por valor

- En C los argumentos de entrada de las funciones se pasan **por valor**; esto es, al invocar una función ésta:
  - **recibe los valores** de las variables con los que tiene que ejecutarse
  - **no modifica las variables** cuyos valores se le pasaron como entrada



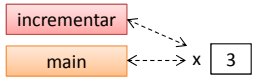

```
#include <stdio.h>

void incrementar(int x) {  // Se recibe valor 2
   x = x + 1;
   printf("El valor de x en incrementar es %d.\n", x);  // x es 3
}

void main() {
   int x = 2;                                            // x es 2

   incrementar(x);          // Se pasa el valor 2

   printf("El valor de a en main es %d.\n", x);          // x es 2
}
```


Contenidos

- Funciones: definición, sintaxis, ejemplos
- Alcance de variables
- **Paso de argumentos de entrada (I)**
  - Paso de argumentos de entrada por valor
  - **Paso de argumentos de entrada por referencia**
- Paso de argumentos de entrada (II)
  - Paso de arrays de entrada
  - Paso de estructuras de entrada

Paso de argumentos de entrada por referencia (I)

- En C se pueden pasar los argumentos de entrada de una función **por referencia**: esto es, al invocar una función ésta:
  - recibe las direcciones (referencias, **punteros**) a las variables con los que tiene que ejecutarse
  - modifica las variables cuyos punteros se le pasaron como entrada




```c
#include <stdio.h>

void incrementar(int *a) {    // Se recibe la dirección de x (la del main)
   *a = *a + 1;               // *a = "el contenido de lo apuntado por a"
   printf("El valor de x en incrementar es %d.\n", *a);  // imprime 3
   printf("El valor de x en incrementar es %d.\n", a);
                                          // imprime dirección de x
}
void main() {
   int x = 2;                                          // x es 2

   incrementar(&x);           // Se pasa &x, la "direccion" (puntero) de x

   printf("El valor de x en main es %d.\n", x);          // x es 3
}
```


Paso de argumentos de entrada por referencia (II)

- Ejemplo: paso de argumentos de entrada por palor y por referencia

```c
int sumarEnterosPorValor(int a, int b) {
   int r;
   r = a + b;
   return r;
}

void sumarEnterosPorReferencia(int a, int b, int *r) {
   *r = a + b;
}

void main() {
   int n1 = 2, n2 = 6; s;

   // Llamadas a funcion equivalentes
   s = sumarEnterosPorValor(n1, n2);

   sumarEnterosPorReferencia(n1, n2, &s);
}
```


Paso de argumentos de entrada por referencia (III)

- Al comienzo de una función se deben comprobar que los argumentos de entrada son correctos, especialmente si son punteros

```c
int sumarEnterosPorReferencia(int a, int b, int *r) {
   // Control de argumentos de entrada
   if ( r == NULL ) {
      return -1;
   }

   *r = a + b;

   return 0;
}

void main() {
   int n1 = 2, n2 = 6, s;

   sumarEnterosPorReferencia(n1, n2, &s);
}
```

Contenidos

- Funciones: definición, sintaxis, ejemplos
- Alcance de variables
- Paso de argumentos de entrada (I)
  - Paso de argumentos de entrada por valor
  - Paso de argumentos de entrada por referencia
- **Paso de argumentos de entrada (II)**
  - **Paso de arrays de entrada**
  - Paso de estructuras de entrada

Paso de arrays de entrada (I)

- En C una variable array [] es equivalente a un puntero
  - Por ello, las variables array se pasan por referencia sin precederlas con &

```
int mediaListaEnteros(int *lista, int longitud) {
    int m, i;

    // Control de argumentos de entrada
    if ( lista == NULL || longitud <= 0 ) {
        return INT_MIN;
    }

    m = 0;
    for ( i=0; i<longitud; i++ ) {
        m += lista[i];
    }
    m /= longitud;

    return m;
}

void main() {
    int l1[] = {1, 2, 3};
    int *l2 = {4, 5, 6, 7, 8};  // Declaraciones de l1 y l2 analogas
    int m1, m2;

    m1 = mediaListaEnteros(l1, 3);    // No se pasa &l1
    m2 = mediaListaEnteros(l2, 5);    // No se pasa &l2
}
```


Paso de arrays de entrada (II)

- En C una variable array [] es equivalente a un puntero
  - Por ello, las cadenas de caracteres (es decir, char[] o char*) se pasan por referencia sin precederlas con &

```
int longitudCadena(char *cadena) {
    int longitud, i;

    // Control de argumentos de entrada
    if ( cadena == NULL ) {
        return -1;
    }

    longitud = 0;
    while ( cadena[i] != '\0' ) {
        longitud++;
        i++;
    }

    return longitud;
}

void main() {
    char s1[] = "Hola"; // Declaraciones de s1 y s2 analogas
    char *s2 = "mundo";
    int l1, l2;

    l1 = longitudCadena(s1);    // No se pasa &s1
    l2 = longitudCadena(s2);    // No se pasa &s2
}
```


Contenidos

- Funciones: definición, sintaxis, ejemplos
- Alcance de variables
- Paso de argumentos de entrada (I)
  - Paso de argumentos de entrada por valor
  - Paso de argumentos de entrada por referencia
- **Paso de argumentos de entrada (II)**
  - Paso de arrays de entrada
  - **Paso de estructuras de entrada**

Paso de estructuras de entrada

- Paso de estructuras por valor y por referencia
  - Los atributos de una estructura se acceden mediante '.' cuando se pasa por valor y mediante '->' cuando se pasa por referencia

```
typedef struct {
    char titulo[256];
    char autor[256];
    int anio;
} Libro;

void imprimirLibroPorValor(Libro libro) {
    printf("Titulo: %s\n", libro.titulo);
    printf("Autor: %s\n", libro.autor);
    printf("Anio: %d\n", libro.anio);
}

void imprimirLibroPorReferencia(Libro *libro) {   // Esta es la forma que se suele usar
    printf("Titulo: %s\n", libro->titulo);
    printf("Autor: %s\n", libro->autor);
    printf("Anio: %d\n", libro->anio);
}

void leerLibro(Libro *libro) {    // Se pasa por referencia para modificar la variable
    scanf("%s", libro->titulo);   // libro->titulo es equivalente a (*libro).titulo
    scanf("%s", libro->autor);
    scanf("%d", &libro->anio);    // Atencion al uso de & por leer un int
}
```

```
void main() {
    Libro l;

    leerLibro(&l);

    imprimirLibroPorValor(l);
    imprimirLibroPorReferencia(&l);
}
```