

# Lenguaje C, primer bloque

José Otero

Departamento de informática  
Universidad de Oviedo

23 de octubre de 2007

## Índice

### 1 Introducción al lenguaje C

- Características
- Entorno de C
- Elementos de C
- Estructura de un programa en C

### 2 Tipos de datos

### 3 Variables

### 4 Operadores

### 5 Entrada/Salida

# Índice

## 1 Introducción al lenguaje C

### ■ Características

■ Entorno de C

■ Elementos de C

■ Estructura de un programa en C

## 2 Tipos de datos

## 3 Variables

## 4 Operadores

## 5 Entrada/Salida

- Lenguaje de alto nivel: más cercano al lenguaje humano que el código máquina.
- lenguaje estructurado:
  - Un cálculo se especifica mediante la sucesión de pasos o instrucciones que debe efectuar el computador para completarlo.
  - El conjunto de instrucciones que determinan un cálculo puede ser utilizado varias veces dentro de un mismo programa sin tener que repetirlos.
  - Se puede dar un nombre a ese conjunto de instrucciones y utilizar ese nombre para referirse al conjunto de instrucciones.

- Lenguaje modular.
  - Un programa escrito en C puede constar de distintos bloques separados.
- Permite el desarrollo de programas mediante refinamientos sucesivos.
- Produce programas pequeños, rápidos, necesitan pocos recursos.
- Portable.
- En el código fuente, distingue entre mayúsculas y minúsculas.
- Las instrucciones se ejecutan de izquierda a derecha y de arriba a abajo, una en cada instante.

# Índice

## 1 Introducción al lenguaje C

- Características

- **Entorno de C**

- Elementos de C

- Estructura de un programa en C

## 2 Tipos de datos

## 3 Variables

## 4 Operadores

## 5 Entrada/Salida

Además del compilador, para producir un ejecutable a partir de sus fuentes en C, son necesarios:

- Preprocesador.
- Archivos de cabeceras.
- Librerías.

## Preprocesador:

Es un programa auxiliar que, como su nombre indica, preprocesa los fuentes en C.

- Las *directivas* indicando el proceso comienzan con el caracter #.
- Entre otras tareas, en este curso las emplearemos para:
  - Definición de constantes:
    - `#define nombre valor`, busca y reemplaza nombre por valor.
    - Ejemplo, `#define PI 3.14159`
  - Inclusión de ficheros.
    - `#include <fichero>` (si está en su ubicación estandard)
    - `#include "/path/completo/fichero"` (si no lo está)
    - Ejemplo: `#include <stdio.h>`



## Archivos de cabeceras:

- En esos archivos se encuentra información sobre funciones, constantes,...predefinidas.
- Es necesario incluir las correspondientes a las que se usen en el programa.

## Librerías:

- Existe código precompilado (funciones matemáticas,...) que se puede añadir al producido por los fuentes durante el proceso de compilación.

# Índice

## 1 Introducción al lenguaje C

- Características
- Entorno de C
- **Elementos de C**
- Estructura de un programa en C

## 2 Tipos de datos

## 3 Variables

## 4 Operadores

## 5 Entrada/Salida

Para su compilación, el fichero fuente se descompone en unidades básicas denominadas *token*, tipos:

- Palabras clave o reservadas. Su uso está predefinido en el lenguaje.
- Identificadores: nombre con el que se designa una entidad definida por el programador. Construcción.
  - El primer carácter debe de ser una letra o un subrayado '\_'.
  - Los siguientes pueden ser letras, dígitos o '\_'
  - No se admiten 'ñ', acentos, espacios en blanco, puntos...

- Constantes: valores que se escriben directamente en el código. Pueden ser numéricas (enteras o reales), de tipo carácter o cadena de caracteres.
  - Reales: 31.7, -0.2, 12, 13.1E-3
  - Enteras: 1436, -5
  - Carácter: 'x', '\$', ' '
  - Cadenas de caracteres: "laralilo"

- Operadores: uno o varios signos representando una operación. Los hay unarios, binarios o ternarios, en función del número de operandos.
- Comentarios: porciones del fuente que se excluyen de la compilación.
  - Comprendidos entre `/*` y `*/` o
  - desde `//` hasta fin de línea
- Separadores: espacios en blanco, retornos de carro, tabuladores.

- Los token se agrupan en sentencias.
  - Simples: acabadas en ;
  - Compuestas: de varias simples, encerradas entre {}
- Es irrelevante como se escriban las sentencias, varias en una línea, en distintas líneas.
- Los aspectos estéticos sólo afectan a la legibilidad, no a la corrección.

## Índice

- 1** **Introducción al lenguaje C**
  - Características
  - Entorno de C
  - Elementos de C
  - Estructura de un programa en C**
- 2 Tipos de datos
- 3 Variables
- 4 Operadores
- 5 Entrada/Salida

```
//directivas del preprocesador
#include...
#define...
//declaraciones y/o definiciones globales
...
int main()
{
//declaraciones locales
...
//sentencias
...
}
//definiciones, aqui o en otro fichero
//que se incluya con #include
...
```



## Índice

- 1 Introducción al lenguaje C
  - Características
  - Entorno de C
  - Elementos de C
  - Estructura de un programa en C
- 2 Tipos de datos**
- 3 Variables
- 4 Operadores
- 5 Entrada/Salida

Un *tipo de dato* es un sinónimo de *conjunto*

Debe especificarse a que conjunto pertenece cualquier entidad definida por el programador.

- El conjunto de los caracteres se denomina `char`.
- El conjunto de los enteros en C se denomina `int`.
- El conjunto de los reales en C se denomina `float` o `double` (mayor precisión).
- `void` se usa para especificar ausencia de tipo.

Cada tipo de dato ocupa distinta cantidad de memoria y tiene un rango de valores distinto:

```
void<char<int<float<double
```

Dependen de la arquitectura.

- Para `char` e `int` es posible limitar el rango a cantidades positivas, incrementando el rango en esa dirección:
  - `unsigned char`
  - `unsigned int`
- Para `int` es posible incrementar el tamaño en memoria y simultáneamente el rango
  - `long int`
- Lo mismo para `double`
  - `long double`

## Índice

- 1 Introducción al lenguaje C
  - Características
  - Entorno de C
  - Elementos de C
  - Estructura de un programa en C
- 2 Tipos de datos
- 3 Variables**
- 4 Operadores
- 5 Entrada/Salida

Las variables son nombres simbólicos para valores que intervienen en cálculos y sus resultados.

- Cada tipo de dato ocupa distinta cantidad de memoria.
- Se codifica de forma diferente.
- Por eso es necesario *declarar* las variables:
  - Especificar el tipo al que pertenecen.
  - Especificar el nombre.
  - Opcionalmente puede dárseles un valor inicial.
  - Implícitamente se le asigna un lugar y un espacio de memoria.

La forma más simple consiste en especificar el tipo y el nombre:

- `tipo nombre;`

Ejemplos:

- `int a; //una variables de tipo entero`

- `float x,y,suma; //tres de tipo real`

Pueden mezclarse declaraciones simples e inicializaciones:

- `float factor=1.76,x;`

Es posible hacer que no se pueda cambiar el valor de una variable después de inicializada, mediante el modificador `const`:

- `const tipo nombre=valor_inicial;`

Ejemplo:

- `const int num=1436;`

Si el programa contiene sentencias que signifiquen modificar `num`, no compilará.

Ámbito y duración de una variable:

Un *bloque* es una porción de código encerrada entre `{ }`.

- Una variable se puede utilizar dentro del bloque en donde se declara.
  - Si se declara *fuera* de `main` y de cualquier bloque es *global* y se puede utilizar en cualquier parte del fuente.
  - Si se declara *dentro* de algún bloque, es local (a ese bloque) y solo se puede utilizar dentro de ese bloque.
  - Se desaconseja el uso de variables globales no constantes.
- Duración: una variable se crea cuando alcanza su declaración y se destruye al salir de él.



## Índice

- 1 Introducción al lenguaje C
  - Características
  - Entorno de C
  - Elementos de C
  - Estructura de un programa en C
- 2 Tipos de datos
- 3 Variables
- 4 Operadores**
- 5 Entrada/Salida

Según el tipo de operación que representan:

- Aritméticos.
- Lógicos.
- Relacionales.

Según el número de operadores:

- Unarios.
- Binarios.
- Ternarios.

Operadores aritméticos, devuelven el valor de la operación indicada.

Operadores aritméticos binarios:

- +
- -
- \*
- /
- % sólo aplicable a `int`, devuelve el resto de dividir dos números.
  - Ejemplo: `11%3` vale 2

## Operadores aritméticos unarios:

- ++ incremento, suma 1
- -- decremento, resta 1
- - signo "menos".
  - Aclaración: dada la declaración `int a=3;`, `-a` vale `-3`, pero `a` no cambia.

Operadores lógicos: devuelven cierto o falso, se aplican a valores que representen cierto o falso.

En C, falso es 0 y cierto cualquier cosa distinto de cero. No es buena práctica de programación suponer que cierto es 1.

Operadores lógicos binarios:

- `&&`, conjunción, y, AND
- `||`, disyunción, o, OR

Operador lógico unario:

- `!`, negación, no, NOT

Operadores relacionales: devuelven cierto o falso según la condición que representan sea cierta o falsa, se aplican a valores de cualquier tipo.

- < menor
- > mayor
- <= menor o igual
- >= mayor o igual
- == igual
- != distinto

## Expresiones:

- Los operadores pueden encadenarse para formar *expresiones*.
- Se puede definir inductivamente como:
  - Una constante.
  - Una variable.
  - Una función de una expresión.
  - Dos expresiones relacionadas con un operador.

## Prioridad y asociatividad:

- Cuando en una expresión intervienen varios operadores, se realizan primero las operaciones indicadas por los operadores de mayor prioridad.
- Cuando los operadores tienen la misma prioridad, el orden de ejecución lo establece la asociatividad.

## Prioridad y asociatividad de los operadores más frecuentes (algunos se verán más tarde).

- ( ) [ ] izquierda a derecha
- - ! ++ -- derecha a izquierda
- \* / % izquierda a derecha
- + - izquierda a derecha
- < <= > >= izquierda a derecha
- == != izquierda a derecha
- && izquierda a derecha
- || izquierda a derecha
- = += -= \*= /= %= derecha a izquierda



Ejemplos de expresiones en notación algebraica y su equivalente en C.

■  $\frac{a}{b+c} = a / (b+c)$

■  $\frac{a}{b+c} \times d = a / (b+c) * d = a * d / (b+c)$

■  $\frac{a}{(b+c) \times d} = a / (b+c) / d = a / ((b+c) * d)$

■  $\frac{a}{b \times c} = a / b / c = a / (b * c)$

■  $\frac{a+b}{c} = (a+b) / c$

■  $\sqrt{a^2 + b^2} =$

`sqrt(a*a+b*b)=sqrt(pow(a,2)+pow(b,2))`

## Asignación

- Permite *asignar* el valor de una expresión a una variable:
  - `variable=expresion;`
    - 1 Se evalúa `expresion`
    - 2 El valor calculado se guarda en `variable`
- Además, como operador binario, `=` devuelve el valor del operando derecho.
  - Por esa razón se pueden hacer asignaciones múltiples como `a=b=c;`
- Notación abreviada: la asignación `variable=variable operador expresion;` se puede abreviar como `variable operador=expresion;`  
Ejemplo: `a=a+b;` equivale a `a+=b;`

## Conversión de tipos

- Conversión *implícita*: cuando en una expresión intervienen tipos distintos, las operaciones se realizan en el tipo de mayor rango `char<int<float<double`.

- Ejemplo:

```
int a=1,x=2,y;  
float b=2,c;  
y=a/x;//mismo tipo, no hay conversion, y=0  
c=a/b;//a se evalua como 1.0,c vale 0.5
```

- Conversión explícita o *cast*. Se puede forzar la conversión de una variable o cte antecediéndola del tipo de destino entre paréntesis.

- Ejemplo:

```
int a=1,b=2,c;  
float x;  
c=a/b;//c=0, no hay conversion  
x=(float)a/b;//x=0.5, a se evalua como 1.0
```

Además, antes de realizar una asignación, el valor de la expresión se convierte al tipo de la variable de destino:

```
int a;  
float x=1,y=2;  
a=x/y;//a vale 0
```

## Índice

- 1 Introducción al lenguaje C
  - Características
  - Entorno de C
  - Elementos de C
  - Estructura de un programa en C
- 2 Tipos de datos
- 3 Variables
- 4 Operadores
- 5 Entrada/Salida**

- La *Salida* es el proceso mediante el cual el programa pone a disposición del usuario los resultados de sus cálculos.
  - `printf` es muy usada para mostrar resultados por la pantalla.
- La *Entrada* es el proceso de introducción de datos en el programa.
  - `scanf` es muy usada para leer datos desde el teclado.

Para poder usar `printf` o `scanf` es necesaria la directiva `#include<stdio.h>`

La sintaxis de `printf` es:

```
printf("xxx%...yyy%...zzz...\...", exp1, exp2, ...);
```

Entre comillas está la *cadena de control*, incluye:

- Códigos de formato: precedidos de `%`. Indican *como* se quiere mostrar un dato determinado y el *lugar* en el que se muestra.
- Secuencias de escape: precedidas de `\` representan caracteres no imprimibles o especiales.
- El resto de caracteres se muestra tal cual.

`exp1, exp2, ...` son las expresiones cuyo valor se quiere mostrar.

- El primer código de control indica como se va a mostrar `exp1`, el segundo `exp2` y así sucesivamente.

## Códigos de formato habituales:

- %c carácter.
- %s cadena de caracteres.
- %d ó %i entero con signo.
- %u entero sin signo.
- %f real.



## Secuencias de escape más habituales:

- `\n` representa un *retorno de carro*, lo que va después se muestra en la siguiente línea.
- `\t` tabulador.
- `\'` `\"` `\\`, la barra *desactiva* un carácter que tiene un significado especial. Representan el segundo carácter *literalmente*.

## Ejemplo:

```
int a=1,b=2;  
float c=7.1;  
char x='*';  
printf("\n%d %f %d %c",a,c,b,x);
```

## Muestra:

```
1 7.100000 2 *
```

## Ejemplo:

```
int a=1,b=2;
float c=7.1;
char x='*';
printf("\n%d %f",a,c);
printf("\n%d %c",b,x);
```

## Muestra:

```
1 7.100000
2 *
```

## Errores comunes:

- No usar las comillas dobles u olvidar alguna.
- No usar el código de formato adecuado.
- Distinto número de códigos de formato y expresiones.

## Recordar:

- `exp1 . . .` no tienen por que ser variables ni constantes, pueden ser expresiones de cualquier tipo.

La forma de scanf que usaremos es:

```
scanf ("%...%...", &var1, &var2, ...);
```

- Cada %... será uno de los códigos de formato (excepto %s).
  - Especifica como deben de ser interpretados los datos tecleados.
- var1... denotan las *variables* en las que se guardan los datos leídos.
  - El & antes de cada una es obligatorio y su necesidad se explicará más adelante.
- Uso: los valores de las variables se introducen separados por espacios en blanco, retornos de carro o tabuladores. Se finaliza con un enter después del último valor.

## Errores comunes:

- Olvidar las comillas.
- Olvidar algún `&`.
- Distinto número de códigos de formato y variables.
- Código de formato inadecuado.
- Incluir espacios u otros caracteres en la cadena de control.

## Recomendación:

- No usar con `char` ni con cadenas de caracteres.

## Ejemplo:

```
int a,b;  
float x;  
scanf("%d%d%f",&a,&b,&x);
```

Si se teclean 10 20 30.7 a b y x toman esos valores.  
Es indiferente usar varios scanf:

```
int a,b;  
float x;  
scanf("%d",&a);  
scanf("%d",&b);  
scanf("%f",&x);
```

- `getchar` lee un carácter, se muestra en la pantalla, después hay que pulsar enter. Está en `stdio.h`.
- `getch` lee un carácter, no se muestra en la pantalla, no hace falta pulsar enter. Está en `conio.h`.
- `getche` lee un carácter, se muestra en la pantalla, no hace falta pulsar enter. Está en `conio.h`.

### Ejemplos:

```
char c;  
c=getchar();  
c=getch();  
c=getche();
```