Introducción al lenguaje C

Informática
Belarmino Pulido Junquera

Índice

1. Presentación del lenguaje C

- Creado en 1972 por D. Ritchie
- Lenguaje de propósito general
- Portátil o transportable (generalmente)
- Inicialmente de nivel medio (entre alto nivel y ensamblador)
- Pensado para (gestionar / programar) sistemas/comunicaciones
- Lenguaje compilado (compilar + enlazar)
- Modular (permite usar bibliotecas propias o estándar que se enlazan con nuestros programas principales)
- (Demasiado) conciso
- (Relativamente) sencillo de aprender

- 28 palabras reservadas:

auto	double	if	static
break	else	int	struct
case	entry	long	switch
char	extern	register	typedef
continue	float	return	union
default	for	sizeof	unsigned
do	goto	short	while

2. Estructura de un programa

Algoritmo principal es
<declaración tipos>
<declaración variables>
Inicio
<composición secuencial acciones>
Fin

#include <stdio.h>
[int] main([<parámetros línea comando>])
[<declaración de parámetros>]
{
<declaración tipos>
<declaración variables>
<composición secuencial acciones>
}

- Un programa en C es una colección de una o más funciones (algoritmos con nombre) que se llaman unas a otras, siendo siempre **main()** el nombre de la principal**.**

[<tipo>] nombre_función ([<lista parámetros>])
[<declaración de lista parámetros>]
{
<declaración tipos>
<declaración variables>
<composición secuencial acciones>
[return <expresión tipo de la función>]
}

- Al finalizar una función ésta devuelve el control a la función que la ha llamado o al S.O. en el caso de la función **main()**.

- Formato libre de líneas (no hay límite de tamaño)
- Las sentencias deben separarse mediante punto y coma **;**
- Las llaves {} agrupan conjuntos de sentencias lógicamente relacionadas (como inicio-fin, hacer-fin para,...)
- Los comentarios se ponen ***/* así */***
- Todas las funciones llevan un tipo (**int,** por defecto)
- La instrucción **return** nos permite devolver **explícitamente** un valor (equivalente a un parámetro dato en lenguaje algorítmico).
  Si no se especifica, se devolverá un valor arbitrario, del tipo de la función.
- Es **obligatorio** definir **todas** las **variables** que se van a utilizar**.**

3. Instrucciones del Preprocesador

- Se pueden dar instrucciones al compilador para que las ejecute antes de compilar **(instrucciones del preprocesador)**
- Comienzan por # en la primera columna de la línea

- #include → Indica qué biblioteca o fichero queremos utilizar (funciones ya definidas en C)

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

- #define → Indica al preprocesador que substituya todas las apariciones de un símbolo por su valor; por ejemplo: N es un tamaño. Podemos definirlo al principio como 10, 100, 1000,… según lo necesitemes
- También podemos pensar en una forma de definir constantes

```
#define N 1000
#define PI 3.14159
```

4. Tipos de datos escalares

- **Tipos básicos**
  - **char**, carácter
  - **int**, entero
  - **float**, real de simple precisión
  - **double**, real de doble precisión
  - NO HAY LÓGICOS
- **Tipos derivados de los básicos**
  - signed char, unsigned char --> como int
  - short, signed short, unsigned, unsigned short: short int, signed short int, unsigned short int, signed int, unsigned int
  - long, signed long, unsigned: long int, signed long int, unsigned long int

4. Tipos de datos escalares

- **char**, carácter (1 byte → 1 carácter en la representación interna usada, generalmente ASCII)
- Se expresa como un carácter entre comillas simples
- Internamente se almacena como un entero (el valor de su código ASCII)

Ejemplos válidos:

'a'

'Z'

'9'

'\n' → carácter no imprimible (nueva línea)

'\0' → carácter terminación de cadena

'\007' → carácter con código ASCII 007 en octal (sonido)

'\x7' → ídem, pero en hexadecimal

4. Tipos de datos escalares

- **int**, entero; **short** int ó long **int**, enteros (pequeño / grande)
- Representan a los enteros en la máquina (su tamaño dependerá de la máquina y el compilador)
- Habitualmente short int usa 16 bits, long int usa 32
- Int suele ser 16 ó 32 bits
- Rangos con 32 bits:
  - Short (16 bits): -32768 ≤ N ≤ 32767
  - Unsigned short (16 bits): 0 ≤ N ≤ 65535
  - Int : -2147483648 ≤ N ≤ 2147483647
  - Long int : -2147483648 ≤ N ≤ 2147483647
  - Unsigned int: 0 ≤ N ≤ 4294967295
  - Unsigned long: 0 ≤ N ≤ 4294967297

- Ejemplos válidos: 0, 10, -2, 1960, -2122, +120

4. Tipos de datos escalares

- **float**, real de simple precisión (32 bits)
- **double**, real de doble precisión (64 bits)

- Rangos:
  - Float (32 bits) : 1,17549 e-38 ≤ N ≤ 3,40282 e+38
  - Double (64 bits) : 2,22507 e-308 ≤ N ≤ 1,79769 e+308

Ejemplos válidos:

- Reales simple:
  - 1.25 -2.5 .01 -2.15e+2 2.1e-2
- Reales doble
  - 0.0000001 -2.33333333333333 -2.0e100
  - 2.0e-125

- **Declaración de variables o parámetros**
  <tipo> <variable> {, <variable>};

```
main() {
int entero1, entero2;
char caracter1, c, car2;
float real1, r2;
double d1, d2;

short int s;
long int entero_largo;
unsigned char cc;
return 0;
}
```

5. Operadores básicos

- **Asignación interna en C (=)** (:= en algorítmico)

```
int entero = 1, otro_entero;
float r1, r2 = 1.2e-5;
double d=1.23456789;
char c = 'a', nueva_linea = '\n';

otro_entero = 12;
r1 = r2 / 0.5;
```

- Declaración de constantes

```
#define PI 3.1416
#define TAMANO_MAX 200
```

- **Aritméticos**

suma:	a + b	**a = a + 1**	**a += 1**
resta:	c **-** d	**c = c - d**	**c -= d**
incremento	**a=a+1**	**a++**	**++a**
decremento	**b=b-1**	**b--**	**--a**

  Los operadores auto(incremento/decremento) ++a/a++ no son equivalentes

multiplicación:	e ***** f	e = e * 2	e *= 2
división:	g **/** h	g = g / 10	g /= 10
módulo/resto:	i **%** 14	i = i % 2	i %= 2

- **Relacionales / Comparación**

mayor:	j **>** k
mayor o igual:	ll **>=** m
menor:	n **<** op
menor o igual:	q **<=** r
igual:	**s == t (es doble signo =, no confundir con =)**
distinto:	**u != v**

- **Lógicos** (falso == 0, cierto es != 0)

and:	w **&&** y
or:	w **||** y
negación:	**!** z

- **Conversiones de tipos**
  - conversión al tipo de mayor precisión
  - los char se tratan como short int
  - ahormado o *cast*:
    (tipo_t) <expresión>
    fuerza a que el tipo resultado de evaluar **<expresión>** sea de **tipo_t**
    res = (int) real / entero;
    res2 = (float) entero / 2;
- En las bibliotecas (#include <math.h>,...) existen otras funciones para realizar operaciones: sqrt(), sin(), cos(),...
  - además existen funciones estándar para forzar conversiones

6. Introducción a las funciones E/S

Escritura:

int printf ("<cadena control>", <argumentos>);

- La cadena de control especifica el formato y el número de argumentos
- Los argumentos son las variables o expresiones a escribir
- Devuelve el número de argumentos correctamente escritos
- En la cadena de control pueden aparecer:
  - constantes carácter o cadena, que aparecen como tales,
  - constantes tipo carácter: \b, \n, \t, \', \",...
  - descriptores de formato, **%?**, que indican el formato con el que mostrarán los argumentos, donde **?** es uno de los siguientes:

Código formato	Descripción
%c	carácter sencillo
%d	entero
%e	real en notación científica
%f	real simple precisión en notación científica
%g	el más corto de %e, %f
%o	octal
%x	hexadecimal
%s	cadena de caracteres
%u	decimal sin signo

Los descriptores se pueden especificar mediante

%m.n?

Ejemplos:

%10d %10.5f %20s

Ejemplos:

```
printf ("Un entero en una linea: %d \n", entero);
printf ("Dos enteros en una linea: %d, %d\n", entero1, entero2);
printf ("Una cadena %s de caracteres.\n", cadena);
printf ("Varios %d tipos %c mezclados %s\n", entero, caracter,
        cadena);
```

Lectura:

int scanf ("<cadena de control>", <argumentos>);

- cadena de control: ídem que en el printf
- Devuelve número de argumentos leídos correctamente.
- Los argumentos son las variables o expresiones a leer.
- **Los argumentos que sean de tipo dato-resultado o resultado y sean de tipos escalares, deben llevar delante el operador &: &: indirección**, pasa la dirección de la variable y no su valor. Esta es la forma en la que C modifica los valores de los argumentos de una función *(parámetros dato-resultado en código algorítmico):*

```
scanf ("%d", &entero);        /* lee un entero */
scanf ("%d, %d", &entero1, &entero2);       /* ha leido dos enteros*/
scanf ("%s", cadena);         /* leida una cadena, que no es escalar*/
scanf ("%d %c %s\n", &ent3, &c, string);
/* lee un entero, ent3, y un carácter, c, ambos escalares, y una
      cadena, string, que no es escalar*/
```

- Descriptores de formato para la lectura con scanf

Código formato	Descripción y argumento
%s	cadena de caracteres: string **(sin &)** porque es un array de caracteres
%c	carácter sencillo: &caracter
%d	entero: &entero
%e, %f, %g	real en notación científica, con signo y exponente opcionales: &real
%o	octal: &octal
%x	hexadecimal: &hexa
%u	decimal sin signo: &sin_signo

9. Funciones y argumentos

- En C sólo hay funciones para representar algoritmos con nombre:

```
[<tipo>] nombre_función ([<lista parámetros>])
[<declaración de lista parámetros>]
{
        <declaración tipos>
        <declaración variables>
        <composición secuencial acciones>
        [return <expresión tipo de la función>]
}
```

- Todas las funciones devuelven un valor (que define su tipo). Por defecto, devuelve un entero (int) que no hace falta declarar
- Mediante **return** se devuelve (un valor concreto y) el control a la función que la haya llamado.

- Existen dos formas de pasar argumentos:
  - por valor (equivalente a par. dato), y es la forma por defecto → no modifica el valor del argumento
  - por referencia (equivalente a par. dato-resultado) → se puede modificar el valor del argumento.

- El paso por referencia implica pasar la dirección de la variable (direccionamiento indirecto) mediante el operador **& (indirección)**.

- Ejemplo, en el caso de *printf* y *scanf*, que son funciones de la biblioteca estándar de C, los argumentos se pasan:
  - por valor a printf, ya que no los modifica
  - por referencia (mediante &) a scanf porque sí los modifica

```
/* x e y son dos enteros que se pasan por valor a la función multiplica
        x, y, resp son enteros que se pasan por valor a la función printf
   El resultado de la multiplicación, res, estará asociado al nombre de la función */

#include <stdio.h>
main() {
int x, y, res;
x = 10; y = 20;
Res = multiplica(a, b);
printf ("\nEl resultado de multiplicar %d por %d es %d", x, y, res);
}

/*      a y b son enteros y se pasan por valor → no se modifican
        El resultado, asociado a la variable res del código algorítmico, queda asociado al
        nombre de la función (como parámetro resultado) */
int multiplica (a, b)
int a, b;
{
 int resultado;
 resultado = a * b;
 return resultado;
}
```