

Informática

Ingeniería en Electrónica y Automática Industrial

Datos en lenguaje C

V1.1 © Autores

Datos en lenguaje C

- Introducción
- Tipos de datos básicos y modificadores
- Tipos de datos enteros
- Tipos de datos reales
- Tamaño y rango de los datos en C
- Otros tipos de datos
 - Tipos de datos derivados
 - Tipos de datos definidos
- Constantes
 - Constantes enteras
 - Constantes reales
 - Constantes de caracteres
 - Constantes simbólicas
- Declaración de variables
 - Variables locales
 - Variables globales
- Inicialización de variables
- Otros modificadores de tipos de datos
 - Modificadores de tipo de acceso
 - Modificadores de clase de almacenamiento

V1.2 © Autores

Introducción a los tipos de datos en lenguaje C

- Los **datos** son el objeto de procesamiento en los programas de ordenador
 - En lenguajes avanzados se habla de **objetos**, como denominación más genérica
- En lenguaje C las **variables** y las **constantes** deben **declararse** antes de ser utilizadas
- La **declaración** de un dato requiere expresar
 - El tipo de dato
 - El modificador (opcional)
 - El identificador

```
modificador tipodato identificador;
```

V1.2 © Autores

Tipos de datos básicos y modificadores (I)

- Los **tipos de datos** establecen la diferencia entre los objetos que se van a procesar, en cuanto a
 - Memoria que ocupan
 - Rango o valores que se pueden almacenar
 - Modo en el que van a ser tratados
- La **cantidad de memoria** necesaria para el almacenamiento de datos, así como el margen de **variación (rango)** de dichos datos depende:
 - Del compilador
 - Del sistema operativo
 - De la máquina

V1.2 © Autores

Tipos de datos básicos y modificadores (II)

- Las palabras reservadas en lenguaje C para los **tipos de datos básicos** son:

○ char	Carácter
○ int	Número entero
○ float	Número real
○ double	Número real de doble precisión
○ void	Tipo que no existe
○ enum	Tipo enumeración, lista de valores enteros
- Los **modificadores** que se pueden aplicar a los tipos de datos básicos son:

○ signed	Con signo
○ unsigned	Sin signo
○ long	Largo, de mayor tamaño de almacenamiento
○ short	Corto, de menor tamaño de almacenamiento
- Los datos fundamentales utilizados en lenguaje C se obtienen de las combinaciones permitidas de tipos básicos y modificadores.

V1.2 © Autores

Tipos de datos enteros (I)

- Los tipos de datos **enteros** permiten representar cantidades numéricas enteras
 - char (**signed char**). Tipo carácter
 - Normalmente ocupa un byte (permite almacenar un símbolo ASCII)
 - int (**signed int**). Tipo entero con signo
 - Normalmente ocupa cuatro bytes
 - short (**signed short int**). Tipo entero corto
 - Normalmente ocupa dos bytes
 - long (**signed long int**). Tipo entero en formato largo
 - En máquinas de 32 bits: 4 bytes; en 64 bits: 8 bytes
 - enum. Tipo **enumerado**. Declara una variable que puede tomar como valores una lista de símbolos arbitrarios

V1.2 © Autores

Tipos de datos enteros (II)

- La relación entre tamaños que se cumple siempre es:
`short ≤ int ≤ long`
- Representación interna de números enteros
 - Números **sin signo**: aritmética binaria de módulo 2^n siendo n el número de bits empleados
 - Números **con signo**: Complemento a dos con el bit de mayor peso como bit de signo.
- Ejemplos (I)
 - Variable *letra* de tipo carácter:
`char letra;`
 - Variable *cantidad* de tipo entero:
`int cantidad;`
 - Variable *edad* de tipo entero corto:
`short edad;`

V1.2

© Autores

7

Tipos de datos enteros (III)

- Ejemplos (II):
 - Variable *memoria* de tipo largo:
`long memoria;`
 - Definición y utilización de un tipo de enumeración:
`enum semana = {lunes, martes, miercoles, jueves, viernes, sabado, domingo};`
`enum semana hoy;`
`hoy = martes;`
 - `semana` es un tipo de enumeración
 - `hoy` es una variable de tipo enumerado que se ha cargado con el valor `martes`, que si se imprime, mostraría un «1».
 - Si `hoy` se inicializase con el valor `domingo`, al imprimirse, mostraría un «6» (lunes equivale a «0»)

V1.2

© Autores

8

Tipos de datos reales (I)

- Los tipos de datos **reales** permiten representar cantidades numéricas en notación científica y de mayor rango
- Los números reales, se almacenan en memoria en un formato normalizado ya explicado en el que se distinguen tres campos:
 - El signo del número
 - La mantisa
 - El exponente (en representación sesgada)

V1.2

© Autores

9

Tipos de datos reales (II)

- Tipos:
 - `float`. Tipo real de simple precisión
 - Hasta 7 dígitos significativos
 - `double`. Tipo real de doble precisión
 - Hasta 16 dígitos significativos
 - `long double`. Tipo real de doble precisión con formato largo.
 - Puede llegar a tener hasta 19 dígitos significativos

V1.2

© Autores

10

Otros tipos de datos

- Tipo indefinido
 - El tipo `void` indica un dato de tipo indefinido
 - Se usa como genérico para instanciarlo más adelante
- Tipos de datos **derivados**
 - Son datos complejos que se obtienen a partir de los datos fundamentales
 - Arrays, funciones, punteros, estructuras y uniones
- Tipos de datos **definidos**
 - Son tipos creados por el usuario, con un nombre y definición propios
`typedef tipodato nuevonombre;`
 - Facilitan la lectura y escritura de programas
 - Ejemplo:
`typedef unsigned long int mitipo;`
`/* Se ha creado un nuevo tipo de dato: mitipo */`

V1.2

© Autores

11

Constantes (I)

- Las **constantes** son valores fijos que no pueden ser alterados por el programa
- Pueden ser de cualquiera de los tipos de datos posibles en lenguaje C
- Pueden ser
 - Constantes enteras
 - Constantes reales
 - Constantes de caracteres
 - Constantes simbólicas

V1.2

© Autores

12

Constantes (II)

- **Constantes enteras (I)**
 - Para su almacenamiento el compilador escoge el tipo de dato más pequeño compatible con esa constante
 - Pueden expresarse
 - En *decimal*: La opción por omisión
 - El dígito de mayor peso no puede ser un «0»
 - Sólo son válidos los caracteres numéricos entre el 0 y el 9
 - En *octal*
 - El dígito de mayor peso es siempre un «0»
 - Sólo son válidos los caracteres numéricos entre el 0 y el 7
 - En *hexadecimal*:
 - Van precedidas por los símbolos «0x»
 - Son válidos los caracteres numéricos del 0 al 9 y las letras A, B, C, D, E y F tanto mayúsculas como minúsculas

V1.2

© Autores

13

Constantes (III)

- **Constantes enteras (II)**
 - Al escribirlas, se distinguirán los siguientes campos:
 - El prefijo para las hexadecimales o el carácter «0» para las octales.
 - El signo (opcional en el caso de números positivos)
 - El valor numérico
 - Un sufijo opcional que permite modificar el tamaño que el compilador debe asignarle:
 - U para indicar unsigned
 - L para indicar long
 - UL para indicar unsigned long
 - Ejemplos:


```
-23L /* el número -23 almacenado como long */
010 /* el octal 10 que equivale al 8 en decimal*/
0xF /* el 0F hexadecimal que es el 15 decimal */
```

V1.2

© Autores

14

Constantes (IV)

- **Constantes reales**
 - En la asignación o definición, el compilador las crea siempre de tipo `double`
 - Al escribirlas, se distinguirán los siguientes campos:
 - El signo (opcional en el caso de números positivos)
 - Una parte entera precediendo al punto decimal «.»
 - La parte fraccionaria a la derecha del punto decimal
 - Se permite también la notación científica con «e» o «E»
 - Un sufijo opcional que permite modificar el tamaño que el compilador debe asignarle:
 - F para indicar float
 - L para indicar long double
 - Ejemplos


```
33.78 /* constante real de tipo double */
1.25E-12 /* constante real de tipo double */
45F /* constante real de tipo float */
33L /* constante real de tipo long double */
```

V1.2

© Autores

15

Constantes (V)

- **Constantes de caracteres (I)**
 - Las *constantes de un solo carácter* son de tipo `char` y se expresan poniendo el carácter entre comillas simples: `'A'`
 - Las *constantes de barra invertida o caracteres de escape*
 - Permiten representar códigos ASCII sin símbolo
 - Se expresan mediante el valor numérico de su código ASCII precedidos de la barra invertida y entre comillas: `'\código'`
 - El *código* puede representarse
 - En decimal, con hasta tres dígitos: `'\ddd'`
 - En octal, con dos dígitos: `'\0oo'`
 - En hexadecimal, con dos dígitos: `'\0xhh'`

V1.2

© Autores

16

Constantes (VI)

- **Constantes de caracteres (II)**
 - Ejemplos:


```
'6' /* Carácter 6, código ASCII 0x36 */
'\12' /* Código ASCII 12 (Salto de línea) */
'\0x20' /* Código ASCII 32 (Espacio) */
```
 - Las *constantes de cadena*
 - No son un tipo de dato
 - Definen un vector de caracteres almacenados de forma consecutiva cada uno en un byte
 - Se representan entre comillas dobles
 - "Esto es una cadena de caracteres"
 - El compilador almacena la lista de caracteres y un carácter terminador para representar el final de la cadena: **el carácter nulo** `'\0'` (simplemente es un 0)

V1.2

© Autores

17

Símbolos del pre-procesador (VII)

- **Constantes simbólicas**
 - Se definen mediante la directiva


```
#define NOMBRECONSTANTE Equivalencia
```

 - La directiva **NO** es una sentencia de lenguaje C
 - **NOMBRECONSTANTE** es el identificador de la constante simbólica (recomendado en mayúsculas)
 - *Equivalencia* representa los símbolos que va a representar **NOMBRECONSTANTE**
 - Siempre que en el programa aparezca **NOMBRECONSTANTE** será sustituido antes de compilar por *Equivalencia*
 - Ejemplo:


```
#define MAXIMO 100 /* MAXIMO toma el valor 100 */
#define FRASE "Pulsa una tecla"
```

V1.2

© Autores

18

Declaración de variables (I)

- Todas las variables deben *declararse* antes de ser utilizadas para que el compilador les asigne la memoria necesaria
- La *declaración* de una variable es una sentencia
 - Consiste en escribir el nombre de la variable precedida por el tipo de dato


```
tipodedato nombrevariable;
```

 - `tipodedato` representa la palabra o palabras que definen el tipo de dato
 - `nombrevariable` es el identificador de la variable

Ejemplos:

```
char letra;          /* variable tipo carácter */
int actual, mayor, menor; /* variables enteras */
float resultado;    /* variable real */
```

V1.2

© Autores

19

Declaración de variables (II)

- Según el punto del programa donde se declaran, las variables pueden ser *locales*, *globales* o *parámetros formales*.
- **Variables locales**, *variables dinámicas* o *variables automáticas* (auto)
 - Se declaran dentro de una bloque de código (función)
 - La declaración debe situarse al comienzo de la función o bloque de código, antes de realizar cualquier otra operación
 - Sólo son válidas dentro de ese bloque de código
 - Desaparecen cuando se finaliza la ejecución de ese bloque de código
 - Si el bloque de código se ejecuta varias veces, en cada ocasión la variable es creada al inicio y destruida al finalizar
 - Hasta que se inicializan, contienen valores "basura"
 - Se almacenan en una zona de memoria que funciona como memoria *pila* (LIFO-Last Input First Output; último en entrar, primero en salir)

V1.2

© Autores

20

Declaración de variables (III)

Variables globales

- Se declaran fuera de cualquier función
- Permanecen activas durante todo el programa
- Se almacenan en una zona fija de memoria establecida por el compilador
- Pueden ser utilizadas en cualquier función a partir del punto de definición. Cualquier sentencia de tales funciones puede operar con ellas sin restricciones
- Pueden estar definidas en otro fichero, en cuyo caso deben definirse con el modificador `extern` en el fichero en que se utilicen
- Al definirse, el compilador las inicia a cero
- No se aconseja su uso, salvo cuando sea imprescindible ya que
 - Hacen las funciones menos portátiles
 - Ocupan la memoria permanentemente
 - Provocan fácilmente errores

V1.2

© Autores

21

Declaración de variables (IV)

Parámetros formales

- Son variables que reciben los valores que se pasan a la función
- Son siempre *locales* a la propia función
- Se declaran en la línea de nombre de la función
- Ejemplo

```
long int Mifuncion(int base, int exponente)
{
    /* Cuerpo de la función */
}
```

V1.2

© Autores

22

Inicialización de variables

- La inicialización de variables sirve para asignar el primer valor
 - Por omisión:
 - Las variables globales se inicializan a cero
 - Las variables locales adquieren el valor de lo que haya en la memoria donde se almacenan (puede ser basura)
 - Puede realizarse en la misma declaración y se realiza mediante un operador de asignación:

```
tipodato nombrevariable = valorinicial;
```

Ejemplo:

```
unsigned int edad = 25;
```

V1.2

© Autores

23

Otros modificadores de datos (I)

Modificadores de tipo de acceso

- Complementan la declaración de una variable para *cambiar la forma en la que se acceden o modifican las variables*
 - `const`. Define una variable como constante, que no podrá ser modificada durante la ejecución del programa.
 - `volatile`. Crea una variable cuyo contenido puede cambiar, incluso por medios ajenos al programa
- Ejemplo

```
unsigned int const anio = 2006;
```

V1.2

© Autores

24

Otros modificadores de datos (II)

- Modificadores de **tipo de almacenamiento**
 - Permiten indicar al compilador el modo de almacenamiento de la variable
 - `extern`. Declara una variable que ha sido definida en un archivo diferente al de la función (ya tienen memoria asignada)
 - `static`. (dentro de una función) Declara una variable local que mantiene su valor entre llamadas.
 - `static`. (fuera de una función) Declara una variable global privada del fichero en que se define
 - `register`. Indica al compilador que la variable debe ser almacenada en un lugar en el que se optimice el tiempo de acceso a ella (preferiblemente en un registro de la CPU)
 - `auto`. Declara una variable local a una función o a un bloque de código (es la opción por omisión)

V1.2

© Autores

25