Tema 2: Tipos de datos elementales. Constantes y variables.

1 Introducción

En este tema vamos a estudiar uno de los pilares de cualquier lenguaje de programación, esto es, sus tipos de datos. Casi todos los lenguajes poseen un conjunto de datos y operaciones similar. Nosotros nos centraremos en los datos elementales que nos proporciona el lenguaje C.

Un dato es como una caja, en la que podemos guardar un único elemento de información. Los datos se caracterizan por tener las siguientes propiedades: Nombre o identificador, tipo y valor.

1.1 Nombre o identificador

El nombre o identificador de un dato es el nombre por el que nos referiremos a cada dato. Ha de ser único, esto es, cada dato tiene un nombre distinto para poder referirnos a él unívocamente. Es como la etiqueta de la caja. En C, los nombres de los datos se constituyen por una única palabra que puede estar compuesta por letras (excepto la ñ), números y algunos símbolos del código ASCII. Es importante recordar que para el lenguaje C, las letras en mayúscula y en minúscula son distintas. Además, los identificadores no pueden comenzar por un número (aunque sí contener alguno. Por ejemplo, los siguientes identificadores serían válidos (y distintos entre sí): Hola, MaNoTas, Curso_01, curSo_01, etc…

Además de estas normas, es necesario que el identificador no coincida con ninguna palabra reservada, esto es, nombres que se correspondan con palabras propias del lenguaje (instrucciones, estructuras de control, etc…). Las palabras reservadas en C son las siguientes:

Palabras reservadas en C			
auto	Break	case	char
const	Continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while

1.2 Tipo

El tipo es el conjunto de valores que puede tomar un dato. Como veremos más adelante, el tipo de un dato determina el conjunto de operaciones que se pueden realizar sobre el mismo. En nuestro ejemplo metafórico de la caja, el tipo de la caja determinaría qué puede contener. No podríamos guardar un ordenador en una caja de zapatos y al contrario sería un derroche de espacio.

Aunque más adelante hablaremos de los tipos de datos en C, vamos a introducir uno de ellos: el tipo entero. La palabra reservada asociada al tipo entero es *int* (proviene del término *integer*). Un dato de tipo int está preparado para contener valores de tipo entero (sin decimales) que se encuentren en el rango entre -32768 y 32767 (16 bits). Debido a la naturaleza de este tipo de dato, podremos realizar con él operaciones numéricas como: resta (-), división (/), multiplicación (*), etc…

1.3 Valor

El valor es la información que almacena el dato en un momento dado. En el ejemplo, una caja de zapatos no puede contener otra cosa que no sean zapatos, pero no tiene por qué contener siempre los mismos. Puede contener cualquier modelo y podemos usarla a lo largo del tiempo para guardar otro modelo distinto.

En C, para dar valor a un dato utilizamos el operador asignación, que se expresa mediante el símbolo igual (=):

```
Identificador = valor o expresión;
```

Ejemplo:

```
DiaDelMes = 7;
TemperaturaActual = 0.75;
Resultado = 3+5*4+ResultadoAnterior;
```

Como podemos observar, a la hora de escribir un número real, el separador entre la parte entera y la decimal es el punto (y no la coma como estamos acostumbrados).

De igual forma, en los ejemplos también se puede ver que las instrucciones en C terminan en punto y coma (;). Dicho símbolo es el separador de instrucciones, no el salto de línea. Dicho de otro modo, si no pusiésemos el punto y coma, el compilador creería que la operación continúa, aunque cambiemos de línea.

Es importante señalar que el operador de asignación no es una igualdad matemática. Al realizar una asignación se calcula el valor de la expresión que hay a la derecha del símbolo = y se le asigna al dato que hay a la izquierda. Por ejemplo, veamos la siguiente cadena de instrucciones:

```
MiDato = 5;
MiDato = MiDato * 2 + 1;
```

En el ejemplo, en la primera línea el dato toma el valor 5. En la siguiente línea se realiza el cálculo de lo que contenía anteriormente la variable (valor 5) multiplicado por dos y mas uno. El resultado de dicha operación es 5*2+1=11. Dicho resultado se guarda de nuevo en el dato, eliminando el valor anterior. Como vemos, esto nada tiene que ver con una igualdad matemática.

2 Constantes y variables

Dependiendo de la forma en la que se crean los datos, pueden ser de dos tipos: constantes y variables.

2.1 Constantes

Un dato constante es aquel que no modifica su valor a lo largo de la ejecución de un programa. Se declaran al principio del programa y su sintaxis en C es la siguiente:

```
#define Nombre_del_dato  Valor_del_dato
```

Como vemos, para declarar una constante no se usa el operador de asignación ni finalizamos la instrucción con el separador ;. Además, solamente es necesario especificar el valor del dato, no su tipo, puesto que va implícito en su declaración.

Ejemplo:

```
#define PI 3.141516
#define DIAS_DE_LA_SEMANA 7
```

Una vez declarados, podremos referirnos a ellos por su nombre (no por su valor) a lo largo de todo el programa:

```
AnguloEnGrados = AnguloEnRadianes * 180.0 / PI;
DiasDelMes = DIAS_DE_LA_SEMANA * 4;
```

Imaginemos que la constante PI definida en el ejemplo se utiliza 140 veces en distintas operaciones de nuestro programa. Mediante el uso de las constantes nos evitamos tener que escribir su valor 140 veces. Además, si decidimos poner otro valor a la constante solamente tendríamos que hacerlo en la definición, no en todas las operaciones en las que aparezca. Por otro lado, el código es mucho más legible y evitamos posibles errores de escritura al repetir el dato en distintos sitios a la vez.

Puesto que los datos constantes no pueden cambiar su valor a lo largo del programa, no es posible utilizar sobre ellos el operador de asignación (=).

2.2 Variables

Las variables son datos que pueden modificar su valor a lo largo de la ejecución del programa. Su sintaxis en C es la siguiente:

```
Tipo nombre_1, nombre_2, … nombre_n;
```

Adicionalmente también es posible asignar un valor inicial a las variables en la propia declaración:

```
Tipo nombre_1 = valor_1, nombre_2 = valor_2, …, nombre_n = valor_n;
```

Por ejemplo, vamos a declarar tres variables de tipo entero:

```
int PrimerOperando = 3, resultaDo, SegundoOperando = 7;
```

En el ejemplo hemos declarado tres variables de tipo entero, dando un valor inicial en la declaración a la primera y a la tercera, pero no a la segunda. Al no dar valor a la segunda variable, el compilador le asignaría un valor residual aleatorio. Una vez declaradas podemos realizar distintas operaciones aritméticas con ellas, como por ejemplo:

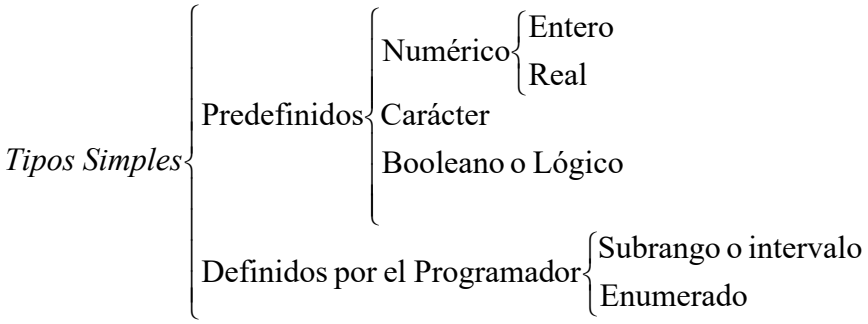
```
PrimerOperando = PrimerOperando +1;
SegundoOperando = SegundoOperando - 9;
resultaDo = PrimerOperando * SegundoOperando;
```

En la primera instrucción sumaríamos una unidad a la variable PrimerOperando. El resultado se almacenaría en la propia variable. Por tanto, después de la primera instrucción el valor de dicha variable sería 4. De igual forma, en la segunda instrucción restamos nueve al valor de la variable SegundoOperando, con lo que al finalizar ésta, el valor de la variable sería –2. Finalmente, en la última instrucción realizaríamos el producto de PrimerOperando y SegundoOperando, almacenando el resultado en la variable resultaDo, quedando ésta con el valor 4*(-2) = -8. Como vemos, las variables pueden cambiar su valor a lo largo del programa tengan valor inicial o no.

3 Tipos de datos simples

El tipo de un dato, como hemos dicho anteriormente, define qué elementos puede albergar un dato y, por tanto, qué operaciones se podrán realizar con él. En este apartado veremos los distintos tipos de datos simples en el lenguaje C.

Dentro de los tipos de datos simples podemos diferenciar entre tipos predefinidos (que vienen por defecto en el lenguaje) o tipos definidos por el programador:



Adicionalmente, un dato puede no ser de ningún tipo. En ese caso diremos que es de tipo vacío y para especificarlo utilizaremos la palabra reservada *void*. Este tipo de datos se usa, por ejemplo en funciones y punteros, que veremos en temas posteriores.

3.1 Tipos Predefinidos

Los tipos de datos predefinidos son aquellos que vienen por defecto en el lenguaje. Son los tipos de datos más simples, a partir de los cuales se formarán tipos más complejos. En C hay tres tipos de datos predefinidos: numérico, carácter y booleano o lógico. Sobre dichos tipos simples se pueden aplicar distintos modificadores para cambiar el rango de valores que pueden albergar.

3.1.1 Tipo numérico

Como su propio nombre indica, los datos de tipo numérico nos sirven para almacenar números. Dentro de este grupo distinguiremos entre dos tipos: tipo entero y tipo real.

Tipo entero: Se caracteriza por almacenar números de tipo entero (sin parte decimal). Para usar este tipo utilizaremos la palabra reservada *int* (proviene del inglés *integer*). Con este tipo podremos almacenar valores dentro del rango entre –32768 y 32767 (16 bits).

En el siguiente ejemplo, declaramos tres variables de tipo entero y realizamos una operación aritmética con ellas:

```
int A = 5, B = 2, C;
C = A / B;
```

Como vemos, declaramos 3 variables: A y B que actuarán como operandos de la división y C que servirá para almacenar el resultado. Si estuviésemos trabajando con números reales, el resultado sería 2.5 pero como estamos trabajando con números enteros el resultado final que se introducirá en C es 2. Sea cual sea el resultado real, en una operación entera siempre se redondea por abajo (entero inmediatamente menor o igual al resultado real).

Tipo real: Se caracteriza por almacenar números en coma flotante (con parte decimal). Para usar este tipo utilizaremos la palabra reservada *float* si queremos un número real de precisión simple (32 bits) o *double* si queremos un número real con doble precisión (64 bits). Se pueden expresar de dos formas: parte real y parte decimal

separadas por un punto (ej: 5.67) o bien en notación científica añadiendo el exponente separado por la letra 'e' (ej: 5.67e22).

En el siguiente ejemplo podemos observar cómo se declaran este tipo de variables y cómo se opera con ellas:

```
float operando1 = 15.04, operando2 = 3.2, Resultado;
Resultado = operando1 / operando2;
```

Como vemos, se declaran tres variables de tipo flotante (reales de precisión simple) y se divide la primera entre la segunda, almacenando el resultado en la tercera. En este caso no hay redondeo puesto que estamos trabajando con valores reales, luego el resultado de la operación sería 4.7.

El mismo ejemplo con variables de tipo double tendría el mismo resultado puesto que esta operación se puede realizar con pocos decimales. Utilizaremos dicho tipo en operaciones en las que se necesite una gran precisión.

3.1.2 Tipo carácter

El tipo carácter se caracteriza, como su propio nombre indica, por almacenar caracteres. Los caracteres pueden ser: letras minúsculas, letras mayúsculas, letras acentuadas, números, símbolos o caracteres de control (no visibles). Todos los caracteres se reúnen en una tabla con 256 elementos llamada ASCII. El tipo carácter puede almacenar cualquiera de estos elementos, tiene una longitud de 8 bits, lo cual hace que posea exactamente las 256 combinaciones necesarias. Para usar este tipo utilizaremos la palabra reservada *char*.

Existen dos maneras de expresar un carácter: su posición numérica en la tabla de códigos ASCII o el propio carácter expresado entre comillas simples. Las siguientes declaraciones serían equivalentes:

```
char MiVariableCaracter = 65;
char MiVariableCaracter = 'A';
```

Como vemos, para utilizar el primer tipo de representación, necesitaríamos conocer toda la tabla ASCII. En adelante utilizaremos el segundo tipo puesto que es mucho más sencillo y legible. Sin embargo, el primer tipo de representación nos sirve para apreciar que, en realidad, un tipo char es al final un número. Por tanto, podremos hacer determinadas operaciones aritméticas con él. Por ejemplo, para calcular el carácter siguiente a la 'A' en la tabla de códigos ASCII podríamos hacer lo siguiente:

```
char MiVariable = 'A';
MiVariable = MiVariable +1;
```

Como hemos dicho, en la tabla ASCII existen algunos caracteres especiales. En la siguiente tabla expresamos algunos que nos serán de utilidad en el futuro:

Caracteres especiales	
'\b'	Back space
'\t'	Tabulación horizontal
'\n'	Nueva línea
'\"'	Comillas dobles
'\''	Comillas simples
'\\'	Barra invertida
'\0'	Nulo

3.1.3 Tipo booleano o lógico

El tipo booleano se caracteriza por almacenar los valores verdadero o falso. En C no existe un tipo booleano como tal, por lo que en su lugar se utiliza un entero donde el valor 0 representa falso y cualquier otro valor representa verdadero. Como veremos, este tipo se utiliza mucho dentro del lenguaje C. En general, cuando una variable cualquiera presente valor distinto de 0 diremos que es cierta y cuando, por el contrario, sea igual a 0 diremos que es falsa.

3.1.4 Modificadores de tipo

Sobre estos tipos simples se pueden aplicar modificadores, los cuales aportan características especiales. Los modificadores de tipo se especifican a la izquierda del tipo en la declaración de la variable:

```
Modificador_de_tipo Tipo Nombre_de_variable;
```

Los modificadores de tipo son los siguientes:

Modificadores de tipo en C	
signed	Especifica que el tipo se utilizará con signo. Este modificador viene por defecto en cada tipo simple, por lo que se omite.
unsigned	Especifica que el tipo se utilizará sin signo. Por tanto, en un dato definido con este modificador no podremos utilizar números negativos pero sí el doble de números positivos que en el tipo normal.
long	Especifica que el tipo utilizará el doble de bits para su representación. Por tanto el rango de datos que podremos asignarle será mucho mayor.
short	Especifica que el tipo utilizará la mitad de bits para su representación. Por tanto el rango de datos que podremos asignarle será mucho menor.

Veamos algunos ejemplos:

```
unsigned char VariableA;
unsigned long int VariableB;
```

En estos ejemplos, la VariableA está definida sobre un tipo simple char. Recordemos que este tipo posee 8 bits, por lo que puede almacenar datos entre -128 y 127. Al incluir el modificador de tipo sin signo no podrá aceptar números negativos. Esto significa que ahora podrá aceptar números de 0 a 255. De igual forma, la VariableB está definida sobre un tipo int, cuyo rango está entre -32768 y 32767 (16 bits). Al utilizar el modificador long cambiamos el tipo para que utilice el doble de bits, por lo que en vez de 16 bits tomará 32. El rango cambia por tanto de -2147483648 a 2147483649. Además, al utilizar el modificador unsigned, se eliminan los valores negativos por lo que el rango de la variable estará entre 0 y 4294967296.

3.1.5 Casting

En ocasiones, puede sernos de utilidad que en una determinada operación un dato cambie de tipo momentáneamente comportándose como otro distinto. Por ejemplo, observemos las siguientes líneas de código:

```
int a = 25, b;
float c = 8.9;
b = a*c;
```

Como vemos, se han declarado tres variables: a y b de tipo entero y c de tipo real de simple precisión. Al realizar la multiplicación, a y c son de distinto tipo por lo que el compilador transforma ambos al tipo más preciso, en este caso al tipo real. Por tanto, el resultado de la operación sería: 25.0 * 8.9 = 222.5 Al hacer la asignación, nos encontramos con que el dato de destino es un entero, por tanto el compilador transforma el resultado redondeándolo por abajo, quedando finalmente b con el valor 222.

Imaginemos que, lo que queríamos realmente era multiplicar a por la parte entera de c. Para realizar esta operación tendríamos que convertir momentáneamente la variable real a entera y operar con ella. Para ello, ponemos delante del dato el tipo al que queremos convertirlo entre paréntesis, en este caso int:

```
int a = 25, b;
float c = 8.9;
b = a*(int)c;
```

Haciendo esto, la multiplicación quedaría 25*8 cuyo resultado sería 200.

3.2 Tipos definidos por el programador

Los tipos definidos por el programador son aquellos que no se encuentran por defecto en el lenguaje, sino que los construimos nosotros mismos para un uso concreto. Pueden ser de dos tipos: tipo subrango o intervalo y tipo enumerado.

3.2.1 Tipo subrango o intervalo

Un subrango o intervalo representa una lista de valores ordenados. Por ejemplo, el rango 2…7 representaría los números entre 2 y 7 (ambos inclusive). De igual forma ‘C’…’F’ representaría todas las letras mayúsculas entre C y F.

Aunque este tipo de datos resulta bastante útil no tiene representación directa en C aunque sí aparece en otros lenguajes de programación como Pascal.

3.2.2 Tipo enumerado

El tipo enumerado representa un conjunto de valores. Una variable declarada de este tipo solo podrá tomar los valores especificados en su definición.

La sintaxis en C para definir un tipo enumerado sería:

```
enum NombreEnumeracion {constante1,constante2, … constanteN };
```

Una vez definido el tipo, podemos definir variables del mismo de la forma:

```
NombreEnumeracion NombreVariable = valorInicial;
```

Por ejemplo, vamos a definir un tipo enumerado que simbolizará los días laborales de la semana y una variable de ese tipo colocada en el valor Martes:

```
enum TipoSemanaLaboral { Lunes, Martes, Miercoles, Jueves, Viernes };
TipoSemanaLaboral MiSemana = Martes;
```

Este tipo de datos se reduce a una lista ordenada en el que el primer valor que hemos puesto está en la posición 0, el siguiente en la 1, etc.. Por tanto también podemos asignar a la variable el valor Martes haciendo:

```
MiSemana = 1;
```

Además, si queremos cambiar, por ejemplo al siguiente día, puesto que como vemos son números, podremos realizar una operación matemática de la forma:

```
MiSemana = MiSemana +1;
```

4 Operadores

Como hemos dicho anteriormente, el tipo de un dato determina el conjunto de operaciones que podemos realizar con él. En este sentido, un operador maneja uno o varios datos (operandos) para realizar un determinado cálculo con ellos. Dependiendo del tipo de los operandos se podrán aplicar unos tipos de operadores u otros. Existen cuatro grandes tipos de operadores:

Operadores { Aritméticos, Alfanuméricos, Relacionales, Lógicos

En este apartado estudiaremos cómo se usan estos conjuntos de operadores en lenguaje C.

Operadores aritméticos: Realizan operaciones entre datos numéricos. Su resultado es un valor numérico:

Operador	Significado
*	Producto
/	División Real (si algún operando es real el resultado de la división es real)
/	División Entera (si los dos operandos son enteros el resultado de la división es entera)
%	Resto o módulo de la división entera.
+	Suma
−	Resta

Todos estos operadores aritméticos son binarios. Esto significa que el operador se aplica sobre dos operandos. Ejemplo:

```
int a,b,c,d=2;
a = 5;
b = a+3;
c = a*b;
d = a+b-c/d;
```

Operadores alfanuméricos. Realizan operaciones entre cadenas de caracteres. El resultado es una cadena de caracteres. En C, las cadenas de caracteres se operan a través de funciones. Veremos su funcionamiento en el tema correspondiente.

Operadores relacionales: Realizan operaciones de comparación entre distintos tipos de datos. El resultado es un valor booleano (verdadero o falso):

Operador	Significado
==	Comparación
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que
!=	Distinto de

Todos estos operadores, al igual que los aritméticos son binarios. En estos operadores se refleja el significado de los valores lógicos. Por ejemplo, el resultado de comparar dos variables enteras a y b (a==b) resultaría un valor booleano (cierto o falso).

Operadores lógicos: Realizan operaciones entre datos booleanos. El resultado es un valor booleano.

Operador	Significado
!	Negación ($\neg$)
&&	Conjunción ($\wedge$)
||	Disyunción ($\vee$)

Estos operadores están basados en el **Álgebra de Boole**, la cual recoge todas aquellas operaciones relacionadas con valores lógicos. Cada operador, pose una especificación en el álgebra de Boole llamada *tabla de verdad*. Veamos el significado de cada uno:

La negación es un operador unario (se aplica sobre un único operando). Al anteponerlo al dato que queremos negar devuelve el valor booleano contrario al que contenía. O sea, si el dato era verdadero devolverá falso y viceversa. Su tabla de verdad es la siguiente:

A	$\neg$A
F	V
V	F

La conjunción es un operador binario (se aplica sobre dos operandos). El resultado de la operación es cierto si los dos operandos son ciertos y falso en otro caso. Su tabla de verdad es la siguiente:

A	B	A∧B
F	F	F
F	V	F
V	F	F
V	V	V

La disyunción también es un operador binario. El resultado en este caso es falso si los dos operandos son falsos y cierto en otro caso. Su tabla de verdad es la siguiente:

A	B	A∨B
F	F	F
F	V	V
V	F	V
V	V	V

En temas posteriores, cuando estudiemos las distintas estructuras de control, tanto los datos como los operadores lógicos cobrarán mayor sentido. Por ahora nos conformaremos simplemente con conocer su existencia.

4.1 Orden de evaluación de los operadores

El orden en el que se efectúan las operaciones en un lenguaje es un factor crítico a tener en cuenta si queremos obtener los resultados deseados. Vamos a observar el siguiente fragmento de código:

```
float a=5.0, b=7.0, c=2, d;
d = a + b * c;
```

En el ejemplo, la operación puede realizarse de dos formas: podemos sumar a y b y el resultado multiplicarlo por c o bien multiplicar b y c y al resultado sumarle a. Se trata de operaciones distintas, de la primera forma el resultado sería 22.0 y de la segunda 19.0.

Cada lenguaje asigna a cada tipo de operador una prioridad, de forma que una operación compuesta por varios operadores siempre se resuelva de igual forma. Por ejemplo, sabemos que en C la operación de multiplicación es más prioritaria que la de la suma, por tanto, en el ejemplo anterior la operación se resolvería multiplicando primero b por c para después sumar a.

Para modificar la preferencia de los operadores, es posible agrupar las operaciones enmarcándolas entre paréntesis ( ). De esta forma, si en el ejemplo anterior queremos realizar primero la suma y después multiplicar por el resultado, tendremos que escribirlo de la forma:

```
float a=5.0, b=7.0, c=2, d;
d = ( a + b ) * c;
```

Los paréntesis se consideran como los operadores de máxima prioridad.

La preferencia de operadores en C es la siguiente (de mayor a menor preferencia):

a) Paréntesis
b) Signo
c) Productos y divisiones
d) Sumas y restas
e) Relacionales
f) Negación
g) Conjunción
h) Disyunción

Veamos un ejemplo:

```
int a = 1, b = 2, c = 3, d;
d = (a+5*b-3*(5+2))*-c;
```

El resultado de esta operación aplicando la prioridad sería 30.

```
int a = 1, b = 0, c = 0, d;
d = a || b  &&  !c;
```

El resultado de esta operación sería verdadero.

5 Expresiones

Una expresión es una combinación de una o varias operaciones para realizar un determinado cálculo. Este concepto hemos venido utilizándolo en los ejemplos a lo largo de todo el tema, puesto que todos los cálculos que hemos hecho utilizando los operadores son expresiones.

Dependiendo del tipo de operadores que aparezcan en una expresión distinguimos entre:

- **Expresiones aritméticas**: Expresiones construidas con datos y operadores numéricos, cuyo resultado es un valor numérico.
- **Expresiones alfanuméricas**: Operaciones con cadenas basadas en funciones (se verá en el tema correspondiente).
- **Expresiones lógicas o booleanas**: Expresiones construidas con datos y operadores booleanos, cuyo resultado es un valor booleano.