Tipos de Datos Simples

Cualquier programa, sin tener en cuenta el lenguaje usado, puede ser visto como la especificación de un conjunto de operaciones que son aplicadas a ciertos datos en una determinada secuencia. Estos tres elementos (datos, operaciones y control), sientan la bases para los próximos temas y también proporcionan elementos de comparación entre diversos lenguajes. Este documento considera los datos, tipos y operaciones que podemos encontrar en los diferentes lenguajes.

Elementos básicos

Inicialmente veremos las propiedades que definen los diferentes tipos de datos que se encuentran en un lenguaje de programación, partiendo de los tipos de datos elementales hasta llegar a tipos de datos más complejos, formados a partir de los anteriores.

Datos: Constantes y Variables

Toda computadora ha sido creada para manejar información o **datos**, los cuales pueden ser definidos como aquellos objetos que procesa una computadora.

Las áreas donde se almacenan los datos de una computadora real, tales como la memoria, registros, y medios externos, normalmente tienen una estructura relativamente simple como secuencias de bits agrupadas en bytes o palabras. Sin embargo, el almacenamiento de datos con los que pueden trabajar los lenguajes de programación permite una organización más compleja, como vectores, pilas, cadenas de caracteres, y otras formas de datos existentes en distintos puntos durante la ejecución de un programa.

Algunos de los datos objeto que existen durante la ejecución del programa son **definidos por el programador**, por ejemplo, variables, constantes, vectores, ficheros, etc., que el programador explícitamente crea y manipula desde las declaraciones y sentencias en el programa. Otros datos objeto serán los **definidos por el sistema**, por ejemplo, datos objeto que la computadora necesita durante la ejecución del programa y que no son directamente accesibles por el programador, tales como las pilas de almacenamiento de tiempo de ejecución, registros de activación de subprogramas, buffers de ficheros, y listas dinámicas. Los **datos objeto** definidos por el sistema son normalmente generados de forma automática a medida que se necesitan durante la ejecución del programa sin especificación explícita por el programador.

Un dato representa un **recipiente para los valores de los datos**, un lugar donde los valores de los datos pueden ser almacenados y más tarde recuperados.

Todo dato se caracteriza por un conjunto de **atributos**: nombre, tipo y valor.

Nombre o identificador: servirá para identificarlo. Mediante este atributo el objeto puede ser referenciado durante la ejecución del programa, siendo normalmente creado en la parte declaratoria del programa y modificado por sentencias de asignación y retornos de subprogramas.

El nombre o identificador se formará mezclando letras (mayúsculas o minúsculas, exceptuando la ñ, ç, ... y todos aquellos caracteres que no pertenecen al alfabeto inglés) y números, siempre con la condición de

que el primer carácter sea una letra y no se utilicen signos de puntuación a excepción del símbolo de subrayado.

Ejemplos válidos: `A, Hola, A3, b23, num, saldo_medio`, etc.

Es conveniente, a ser posible, elegir un nombre significativo que describa la información que va a contener el dato.

No serán nombres válidos los siguientes: `23a, 454, 8=+, +, a+, /3`, etc.

Tipo: Conjunto de valores que puede tomar el elemento y que determinará, además el conjunto de operaciones que se pueden realizar con el mismo.

Los tipos básicos se dividen en: numérico entero, numérico real, carácter (un símbolo) y lógico o booleano (el elemento que se considere de este tipo sólo podrá tomar los valores verdadero o falso). Además, cada dato podrá intervenir en una serie de operaciones según el tipo de dato con el que haya sido declarado; es por este motivo que podremos hablar de operadores aritméticos, alfanuméricos, etc., en función del tipo básico que pueden operar.

Estos *operadores,* así como las operaciones que con ellos se pueden realizar, se clasifican como sigue:

❖ **Aritméticos:** servirán para facilitar la operación entre datos de tipo numérico (enteros y/o reales) dando como resultado otro valor también numérico. Entre los operadores aritméticos podemos destacar los siguientes, cuya representación es dependiente del lenguaje de programación:

+		Suma (o signo positivo)
-		Resta (o signo negativo)
*		Producto
/		División real
div	/	División entera
mod	%	Resto de la división entera

Cabe destacar que los operadores DIV y MOD (%, según lenguaje) sólo pueden operar con datos de tipo entero, produciendo a su vez un resultado también entero. Además, estos operadores no existen en todos los lenguajes de programación (por ejemplo, COBOL o FORTRAN no los contemplan) ya que en general cuando en una operación intervienen datos del mismo tipo el resultado que se produce conserva el tipo de los operandos implicados en ella.

❖ **Alfanuméricos:** son aquellos que permiten tratar con datos de tipo carácter o con combinaciones de éstos (las llamadas cadenas de caracteres de las cuales se hablará en temas posteriores).

Existe únicamente uno de estos operadores, el cual hace posible la unión de caracteres:

+	Concatenación

❖ **Relacionales:** posibilitan la comparación entre objetos del mismo tipo dando como resultado valores tales como **verdadero** o **falso**, es decir, de tipo **booleano**. Estos operadores se resumen en:

==	= (según lenguaje)	Igual a
<		Menor que
<=		Menor o igual que
>		Mayor que
>=		Mayor o igual que
¡=	<> (según lenguaje)	Distinto a

❖ **Lógicos:** también el tipo lógico dispone de una serie de operadores que aplicados a éstos dan como resultado un valor también lógico o booleano. Entre éstos se encuentran:

no	not	!	Negación
y	and	&&	Conjunción
o	or	||	Disyunción

Su funcionamiento está determinado por las llamadas **tablas de verdad** que se recogen a continuación.

- Operador `NO:`

A	**no** A
F	V
V	F

- Operador `Y:`

A	B	A **y** B
F	F	F
F	V	F
V	F	F
V	V	V

- Operador `O:`

A	B	A **o** B
F	F	F
F	V	V
V	F	V
V	V	V

❖ **Paréntesis:** ( ) Se utilizan para anidar expresiones.

Orden de evaluación de los operadores.

Las expresiones en las que intervienen los diferentes tipos de operadores se evalúan, en general, según el siguiente orden:

1. Paréntesis (comenzando por los más internos).
2. Signo.
3. Potencias.
4. Productos y divisiones.
5. Sumas y restas.
6. Concatenación.
7. Relacionales.
8. Negación.
9. Conjunción.
10. Disyunción.

operadores	significado	asociatividad
- !	signo negativo de un número, NOT lógico	de derecha a izquierda
* / %	multiplicación, división, resto	de izquierda a derecha
+ -	suma, resta	de izquierda a derecha
< > <= >=	de relación	de izquierda a derecha
== !=	de igualdad	de izquierda a derecha
&&	AND lógico	de izquierda a derecha
||	OR lógico	de izquierda a derecha

La evaluación de operadores de igual prioridad (asociatividad) se realiza siempre de izquierda a derecha. Este orden de evaluación tiene algunas modificaciones en determinados lenguajes de programación.

Valor: información que almacena del dato. Suele ser el resultado de una operación de asignación.

Para asignar valores a los objetos que maneja un programa se utiliza el **operador de asignación** cuya sintaxis o formato de uso en un pseudocódigo podría ser:

```
Identificador := valor o expresión
```

Así, por ejemplo, podemos declarar el elemento *dia* de tipo entero y asignarle el valor *7* con la siguiente instrucción:

```
dia := 7
```

Con esto, el elemento *dia* almacena el valor *7*. Otra representación del operador de asignación sería:

```
dia ← 7
```

¡Atención! El operador de asignación funciona de tal manera que si al dato implicado en la operación le hubiera sido asignado un valor con anterioridad, éste se perdería definitivamente el ser reemplazado por el nuevo.

En C sería

```
dia = 7;
```

Esta información (nombre, tipo y valor) se guarda en *constantes* y *variables*.

Constantes

Una *constante* es un dato que posee un nombre y que está ligado a un valor (o valores) que no cambia mientras dure la ejecución del programa que lo creó.

Si el valor de una constante está ligado a su nombre permanentemente durante su tiempo de vida, este vínculo se debe conocer de forma previa a la compilación. Por ello, si un programador define una constante `MAX` y le asigna el valor `30`, posteriormente no podrá reasignar a la constante el valor `4`, ya que `MAX` es siempre la constante `30`. Algunas veces el compilador puede utilizar información mediante el valor de la constante para generar el código para una sentencia o expresión. Por ejemplo, en la sentencia `si` :

```
si (MAX<2) {... }
```

el compilador ya conoce el valor del dato para la constante MAX y la constante 2, pudiendo deducir que es falso que 30 sea menor que 2.

Ejemplos:
```
El número pi = 3,141592.
La milla es 1.852 metros.
Los días de la semana son 7.
```

De ese modo en un programa desarrollado en C escribiríamos:

```
#define MAX 3.141592
#define MILLA 1852
#define DIAS_SEMANA 7
```

Y en un programa desarrollado en C++ escribiríamos:

```
const int MAX = 3.141592;
const int MILLA = 1852;
const int DIAS_SEMANA = 7;
```

La diferencia entre "`define`" y "`const`" radica en que la primera es una directiva de preprocesador que se activa antes de que el compilador entre en acción sustituyendo cada una de las etiquetas por el valor definido previamente, frente a "`const`" que deja el tipo explícitamente definido y el ámbito claramente acotado.

Variables

Variables son todos aquellos datos objeto cuyos valores no tienen por qué ser los mismos a lo largo de la vida de un programa sino que pueden cambiar (claro está, conservando el tipo con el que fue declarado) durante la ejecución del mismo.

Evidentemente, necesitamos que cada variable tenga un nombre o identificador para poder referirnos a ella sin ambigüedad. Además, si queremos delimitar de antemano qué operaciones pueden realizarse con tal o cual variable o protegernos contra errores inadvertidos como, por ejemplo, introducir valores numéricos donde en principio sólo tendría que haber letras, es imprescindible clasificar las variables en *tipos*.

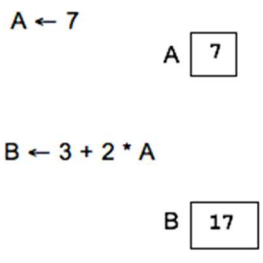
Así pues declaramos la presencia de una variable con:

Un **nombre** o **identificador** que es el conjunto de caracteres, letras y números, con los cuales se identifica un valor en un momento determinado.

Un **tipo** que es la clase de información que puede almacenar esa variable.

Ejemplo: En una función matemática Y = 5X + 2, el valor que toma la variable Y depende del valor que tome la variable X.

Las variables también se pueden entender como cajas con nombres en las que podemos introducir valores los cuales pueden ser alterados mediante el uso del operador de asignación:

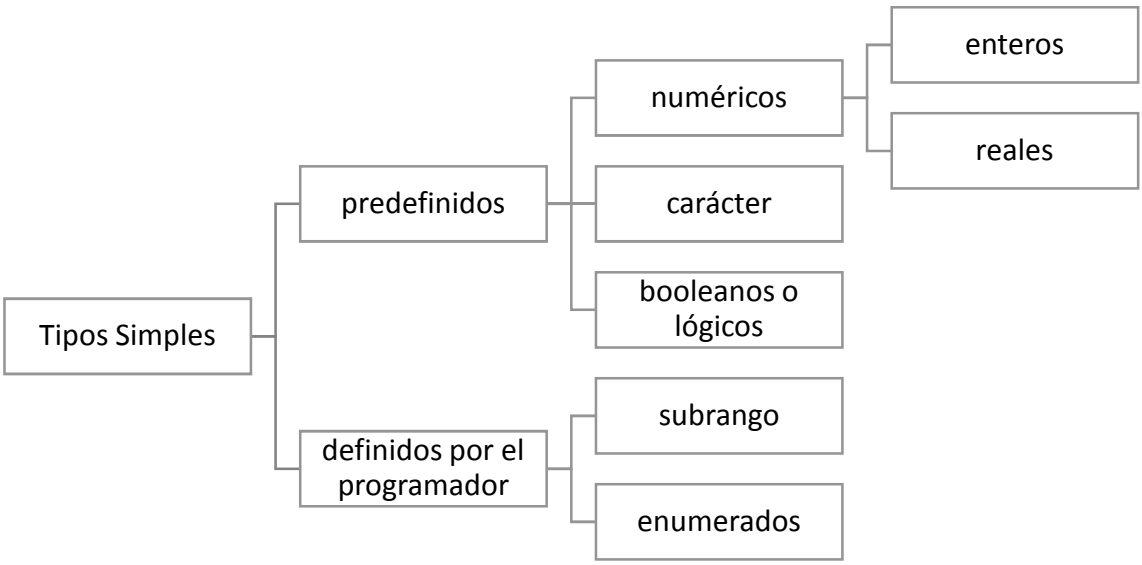


En C escribiríamos:

```
A = 7;
B = 3 + 2 * A;
```

Tipos de datos simples (o sin estructura)

Vamos a considerar los datos elementales que son definidos por el lenguaje de programación. En siguientes temas, extenderemos este concepto a tipos de datos más complejos (o estructurados) tales como vectores, listas, registros, etc. Los tipos simples o sin estructura pueden ser clasificados de acuerdo con el siguiente esquema:



Tipos de datos predefinidos

Tipo numérico entero

Es un subconjunto de los números enteros cuyo rango o tamaño dependen del lenguaje, computadora utilizada y sistema operativo (*MAXINT*).

Los datos de este tipo se expresan mediante una cadena de dígitos que puede ir precedida de signo (+ o -).

Los datos numéricos enteros también son conocidos como números en **punto fijo** o **coma fija** (ya que ésta, aunque no se represente, está implícita a la derecha del mismo).

Ejemplos:

```
1987
-12
+3300
```

Tipo numérico real.

Es un subconjunto de los números reales limitado no sólo en cuanto al tamaño, sino también en cuanto a la precisión o cantidad de decimales que posee. Se expresan de dos maneras diferentes denominadas notación en **punto flotante** y **notación exponencial** (o **científica**). En la primera, un valor consiste en una cadena de dígitos que puede llevar signo y un punto decimal que puede ocupar cualquier posición.

Ejemplos:

```
97.84
-12.00
+0.5
```

Un valor en notación científica o exponencial tiene la forma

mantisa E exponente

donde *mantisa* es un número real y *exponente* un número entero, según la cual, la cantidad representada equivale a la *mantisa multiplicada por 10 elevado al exponente*.

Ejemplos:

```
0.9784E2 (= 0.9784 x 10² = 97.84)
-120000E-4
+0.0005E+3
```

Tipo carácter.

Es el conjunto formado por todos los caracteres o símbolos representados en el teclado de la computadora más algunos otros recogidos en la tabla de códigos ASCII que no aparecen en el mismo.

Se expresan según el lenguaje de programación mediante el carácter colocado entre apóstrofos (‘carácter’).

El conjunto de los caracteres está formado por:

- Las mayúsculas.
  ‘A’, ‘B’, ‘C’, ‘D’, ..., ‘Z’
- Las minúsculas.
  ‘a’, ‘b’, ‘c’, ‘d’, ..., ‘z’
- Los dígitos numéricos.
  ‘0’, ‘1’, ‘2’, ‘3’, ..., ‘9’
- Caracteres especiales.
  ‘ ’ (espacio o carácter blanco)
  ‘+’, ‘-’, ‘*’, ‘/’, ...
  ‘=’, ‘<’, ‘>’, ...
  ‘(’ , ‘)’, ‘!’, ‘?’, ...

Tipo booleano o lógico.

Es el conjunto formado por dos únicos valores FALSO (`true`) y `CIERTO(false)`.

En C tenemos los siguiente tipos simples:

Tipo	Significado	Bytes
char	carácter	1
unsigned char	carácter sin signo	1
int	entero	2-4
short	entero corto	2
long	entero largo	4
long long	entero largo	8
unsigned	entero sin signo	2-4
unsigned short	entero corto sin signo	2
unsigned long	entero largo sin signo	4
unsigned long long	entero largo sin signo	8
float	coma flotante (número real)	4
double	coma flotante de doble precisión	8
long double	coma flotante de doble precisión extendida	16
bool	Booleano	1

Tipos de datos definidos por el programador.

Los tipos de datos **enumerado** y **subrango** son también conocidos como datos definidos por el usuario, por ser este último quien puede definir tanto el tipo de los datos como el número de elementos que lo componen.

En realidad estamos hablando de tipos de datos algo especiales ya que se trata de subconjuntos de los tipos simples que hemos estudiado agrupados siguiendo el criterio del programador

Tipo subrango o intervalo.

Es el tipo de datos más simple que el programador podrá definir en un algoritmo. Un tipo subrango se define a partir de un tipo ordinal y finito, especificando dos constantes de ese tipo, que actúan como límite inferior y superior de dicho subconjunto.

Ejemplos:

1. 1 . . 10 este tipo subrango consta de los elementos 1, 2, 3, 4, 5, 6, 7, 8, 9 y 10.
2. ‘J’ .. ‘N’ este subrango consta de los caracteres ‘J, ‘K’, ‘L’, ’M’, ‘N’.
3. ‘a’ .. ‘z’ este subrango consta de los caracteres comprendidos entre la ‘a’ y la ‘z’.
4. ‘0’ .. ‘9’ este subrango consta de los caracteres que se encuentran entre el ‘0’ y el ‘9’.

De este modo se pueden crear variables cuyos valores se restrinjan a parte del conjunto total formado por los elementos que pertenecen a un tipo dado.

Tipo enumerado.

Un tipo **enumerado** no es más que una lista de valores (a cada uno de los cuales corresponderá un valor ordinal comenzando por 0 (o 1 en algunos lenguajes) que el programador crea en la parte declaratoria del algoritmo. Por tanto, no es más que un conjunto de constantes simbólicas con valor entero.

Eligiendo adecuadamente nombres significativos para los identificadores se pueden hacer programas más fáciles de leer.

Ejemplos en C:

```
enum DiasSemana {lunes, martes, miercoles, jueves, viernes, sabado,
domingo};
```

```
// posteriormente se crea una variable del tipo enumerado anterior
    enum DiasSemana dia;

// y ya es posible asignarle un valor
dia = lunes;
```

Características:

- Un tipo de dato enumerado es un tipo ordinal cuyo número de orden coincide con la disposición dada a los valores en la definición del mismo. En este sentido, se hará corresponder el 0 (o 1 para determinados lenguajes de programación) al primer elemento, el 1 al segundo, etc., lo cual, aplicado el tipo `DiasSemana` del ejemplo anterior se traduce en que `lunes` es igual a 0, `martes` igual a 1, y así sucesivamente.

- Los valores que componen dicha enumeración no son en ningún caso cadenas de caracteres que puedan ser visualizadas por pantalla sino elementos que, como ya hemos comentado, tendrán asignado un número de orden, de modo que si intentamos escribir alguno de ellos tan solo lograremos que en la pantalla aparezca su valor ordinal correspondiente.
- Las variables de tipo enumerado sólo pueden tomar valores de estos tipos.

Expresiones.

A la combinación de constantes y/o variables mediante símbolos de operación, de función y paréntesis se le denomina *expresión*, siendo los operadores (de los cuales ya hemos visto anteriormente los más comunes según el tipo de datos) aquellos nexos que permiten establecer dicha combinación entre los diferentes elementos.

Definición.

Una expresión es la representación de un cálculo necesario para la obtención de un resultado, el cual se determina operando las constantes y los valores de las variables o de las funciones implicadas según lo que indican los operadores que intervienen en la misma.

Se define una expresión de la siguiente forma:

1. Un valor es una expresión. Ejemplo: `1.25, " JUAN"`.
2. Una constante o variable es una expresión. Ejemplo: `PI, E, X`.
3. Una función es una expresión. Ejemplo: `coseno(x), raiz(25)`.
4. Una combinación de valores, constantes, variables, funciones y operadores que cumplen determinadas reglas de construcción es una expresión. Ejemplo:

```
coseno (PI * X) + 1.25
2 * PI * X
N = "JUAN"
```

Tipos de expresiones.

Las expresiones, según el tipo de los datos que manejan y el resultado que producen, se clasifican en:

Aritméticas: Son las que producen resultados de tipo numérico. Se construyen con los operadores de tipo aritméticos y datos de tipo numérico. Ejemplo:

```
PI * raiz(X)
```

Alfanuméricas: Son las que producen resultados de tipo alfanumérico (carácter o combinación de caracteres). Se construyen mediante los operadores alfanuméricos y datos de tipo alfanumérico. Ejemplo de concatenación:

```
"Don " + 'N'
```

En C no existe este operador, pero se puede utilizar la función strcat() disponible en una de las librerías del lenguaje.

Lógicas o booleanas: Son las que producen resultados de tipo lógico (CIERTO o FALSO). Se construyen enlazando constantes y variables lógicas mediante operadores relacionales y lógicos.

```
A>0 && B<=5
```

Ejercicios resueltos

Ejercicio 1. Evaluar las siguientes expresiones:

A: `( 6 * 6 + (8 – 2) ) / 7 + 35 / 2 – 8 * 5 / 4 * 2`

B: `5 - 2 > 4 && ! 0.5 == 1 / 2`

La solución sería:

A: (**6 * 6** + (**8 – 2**)) / 7 + **35 / 2** – **8 * 5** / 4 * 2
(**36 + (6)**) / 7 + 17 – **40 / 4** * 2
(42) / 7 + 17 – **10 * 2**
6 + 17 - 20
23 - 20
3

B: **5 - 2** > 4 && !(**0.5 == 1/2**)
3 > 4 && **!(FALSO)**
FALSO && VERDADERO
FALSO

Ejercicio 2. ¿Son correctas las siguientes expresiones?. ¿Qué tipo de datos devuelven?

1. `A = 2 * -3`
2. `B = 2 + 8 == 7 * 12`
3. `A + 2 = B`
4. `6.4 % 3`
5. `C = 2 + 3.0`

La respuesta a ambas preguntas para cada expresión es la siguiente:

1. Es correcta. Devuelve -6
2. Es correcta. Devuelve un dato de tipo booleano, false, o sea 0.
3. No es correcta. Debería ser B = A + 2.
4. No es correcta. % sólo se aplica a datos de tipo entero.
5. Es correcta. Devuelve un dato de tipo real.

Ejercicio 3. Indica cuales de los siguientes identificadores no son correctos y explica porqué.

1. ALFA
2. α1

3. x009
4. CH_2O
5. 1ABC

La solución es la siguiente:

1. Es correcto.
2. No. El carácter griego α no está permitido.
3. Es correcto.
4. No se permiten subíndices.
5. No. El primer carácter no puede ser un número.

Ejercicio 4. Escribe las expresiones algorítmicas correctas que reflejen las siguientes frases:

1. **"la variable cantidad es mayor o igual a 15 pero menor que 9"**
2. **"evaluar si el contenido de la variable cantidad termina en 0 o en 7"**

Las soluciones son las siguientes:

```
1. cantidad >= 15 && cantidad < 9
2. cantidad%10 == 0    ||   cantidad%10 == 7
```

Ejercicio 5. Escribe una expresión que nos indique si una persona está jubilada (con edad igual o superior a 65 años), conociendo su fecha de nacimiento y la fecha actual, ambas expresadas en día, mes y año.

La solución sería:

```
(aAct – aNac > 65) ||
( (aAct – aNac == 65)  && (mAct > mNac) )  ||
( (aAct – aNac == 65) && (mAct == mNac)  && (dAct >= dNac) )
```

suponiendo:

aAct .- año actual	aNac .- año nacimiento
mAct .- mes actual	mNac .- mes nacimiento
dAct .- día actual	dNac .- día nacimiento