

TEMA 3: Programación en lenguajes de alto nivel

3.1.-Características de un lenguaje de programación de alto nivel

A partir de ahora estudiaremos los conceptos que se utilizan en los lenguajes de programación en general, aunque nos centraremos especialmente en los lenguajes procedurales. Estudiaremos los conceptos en el caso concreto del lenguaje C++.

El lenguaje C++ es una evolución del lenguaje C (1975). El lenguaje C fue desarrollado inicialmente como lenguaje de programación de sistemas operativos, en concreto, para desarrollar y mantener el sistema operativo UNIX. El lenguaje C es muy potente, debido a que permite operaciones a bajo nivel. Pero también a causa de esto, es muy propenso a los errores por parte del programador. Para subsanar esto se creó a principios de los 80 el C++, que añade al C la programación orientada a objetos. En 1998 se creó el estándar de C++ que se denomina ANSI/ISO C++, y es el que se sigue actualmente.

En el vocabulario de los lenguajes de alto nivel se pueden distinguir claramente tres tipos de palabras (*tokens*): palabras reservadas, identificadores y símbolos (*,+/,^, ...).

3.1.1.-Palabras reservadas

Palabras que tienen un significado especial en el lenguaje y no pueden ser utilizadas para ninguna otra cosa. Definen la estructura del programa y las instrucciones más básicas.

Ejemplo:

```
if / for / while ...
```

3.1.2.-Identificadores

Son palabras del lenguaje que hacen referencia a elementos del programa (variables, subprogramas, ...)

En C++ un identificador es una secuencia de caracteres que cumple las siguientes reglas:

- Sólo pueden utilizarse caracteres alfanuméricos estándar (no Ñ, Ç o acentuados) y el carácter subrayado ‘_’. No puede utilizarse el carácter blanco.
- El primer carácter ha de ser una letra o el carácter subrayado.
- No pueden utilizarse palabras reservadas como identificadores.

En C++ se distingue entre mayúsculas y minúsculas, por lo que hay que tener cuidado de cómo se escribe un identificador.

Ejemplo:

<u>Válidos</u>	<u>No Validos</u>
num_2	año
PracticaDos	Practica Dos
_uno	12x
	if

3.1.3.-Símbolos

Existen símbolos de diferentes tipos y con diferentes significados en cada lenguaje de programación, pero básicamente se pueden clasificar en los siguientes grupos:

Operadores:

Son símbolos que indican la realización de una cierta operación entre valores y/o variables.

Ejemplo:

En C++:

+	Suma	-	Resta
%	Resto de la división entera	&&	Y (lógico)
=	Asignación	==	Comparación

En Pascal:

+	Suma	-	Resta
:=	Asignación	=	Comparación

Símbolos de comentarios:

Los comentarios son explicaciones del programa. Son ignoradas por el compilador.

Ejemplo:

En C++:

```
// Inicio de comentario en una sola línea
/* Inicio de comentario en una o varias líneas
*/ Fin de comentario
```

En Pascal:

```
{ Inicio de comentario
} Fin de comentario
```

Directivas del compilador:

Son órdenes especiales para el compilador que no forman parte del lenguaje.

Ejemplo:

En C++ se escriben con una almohadilla delante:

Directiva para incluir un fichero.

```
#include <iostream.h>
```

3.2.-Concepto de variable

En lenguajes de bajo nivel, los datos están en celdas de memoria. La variable es una abstracción del concepto de celda de memoria.

Existe una relación biunívoca entre el nombre de la variable y la dirección de memoria, y el valor de la variable y el contenido de la celda.

Una variable tiene un nombre que la identifica y cuatro atributos básicos:

- *Valor:* Está codificado en la posición de memoria asociada a la variable.
- *Tipo:* Describe el conjunto de valores que puede tomar una variable así como las operaciones que soporta (dominio.)
- *Ámbito:* Rango de sentencias del programa en el que la variable es conocida.

- *Tiempo de vida*: Rango de sentencias en que la variable está asociada a la posición de memoria.

Existen también unas operaciones básicas sobre las variables:

Indirección: Consiste en recuperar el valor de la variable haciendo uso de su nombre.

Ejemplo:

Si la variable X tiene valor 4

$x + 3 ==> 4 + 3$

Asignación: Consiste en modificar el valor de la variable.

Ejemplo:

$X \leftarrow 3$ A partir de ahora X contendrá el valor 3.

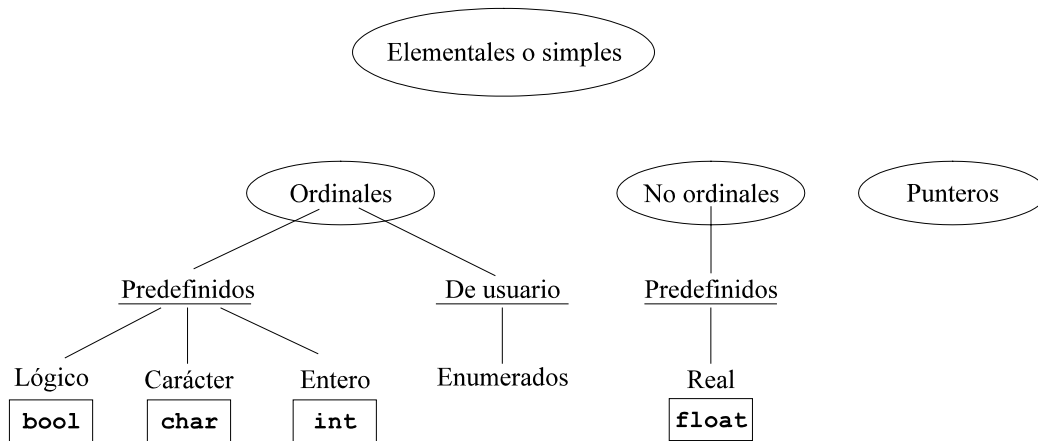
(En C++ $x = 3$ / En Pascal $x := 3$)

3.3.-Tipos simples de datos

Un tipo de datos es *simple* cuando no puede ser descompuesto en tipos de datos más simples (es decir, es atómico). Cuando sí que es posible descomponerlo diremos que es compuesto.

Un tipo es *ordinal* o *numerable* cuando se puede establecer una biyección entre los elementos de ese tipo y los números naturales.

Un tipo es *ordenado* cuando existe una relación de orden. Todos los tipos simples son ordenados.



Los ordinales, debido a la biyección con los naturales, se codifican como enteros.

En la asignatura de Estructura de Computadores I se explica la representación interna de los tipos enteros, reales, lógicos y caracteres.

3.3.1.-Lógicos

Solamente posee los valores **VERDADERO** y **FALSO**.

Como todo tipo ordinal posee un orden: $F < V$

En C++: **bool**

Los valores son: **true** o **false**

Operaciones en C++ sobre variables y valores lógicos:

Los operadores sobre tipos lógicos son: **Y** (&&), **O** (| |), **no** (!).

X1	NO
F	V
V	F

X1	X2	Y	O
F	F	F	F
F	V	F	V
V	F	F	V
V	V	V	V

Ejemplo:

```
bool b;
```

```
int a;
```

```
b = a > 5;
```

3.3.2.-Carácter

Tipo para representar caracteres alfanuméricos, que corresponde con una serie estándar de caracteres. Normalmente se utiliza la serie ASCII, aunque casi todas las series contienen lo siguiente:

- Códigos de control.
- Conjunto de letras por orden alfabético que no contiene caracteres especiales (Ñ, acentos, etc.)
- Conjunto de números del 0 al 9.
- Caracteres especiales: Ñ, Ç, acentos, ?, !, etc.

Los caracteres se codifican en el ordenador utilizando su código ASCII.

En C++:

TIPO	FORMATO (Codificación)
char	16 bits

Los caracteres se escriben con comillas simples 'A'.

```
'A' < 'B' → TRUE
```

TABLA ASCII

		Código ASCII estándar								Código ASCII extendido							
Hex		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Hex	Bin	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000	<i>NUL</i>	<i>DLE</i>		0	@	P	`	p	Ç	É	á	☒	Ł	ł	α	≡
		0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
1	0001	<i>SOH</i>	<i>DC1</i>	!	1	A	Q	a	q	ü	æ	í	☒	Ł	ł	β	±
		1	17	33	49	65	81	97	111	129	145	161	177	193	209	225	241
2	0010	<i>STX</i>	<i>DC2</i>	“	2	B	R	b	r	é	Æ	ó	☒	Ł	ł	Γ	≥
		2	18	34	50	66	82	98	114	130	146	162	178	194	210	226	242
3	0011	<i>ETX</i>	<i>DC3</i>	#	3	C	S	c	s	â	ô	ú		ł	ł	π	≤
		3	19	35	51	67	83	99	115	131	147	163	179	195	211	227	243
4	0100	<i>EOT</i>	<i>DC4</i>	\$	4	D	T	d	t	ä	ö	ñ	ł	-	ł	Σ	ƒ
		4	20	36	52	68	84	100	116	132	148	164	180	196	212	228	244
5	0101	<i>ENQ</i>	<i>NAK</i>	%	5	E	U	e	u	à	ò	Ñ	ł	ł	ł	σ	Ј
		5	21	37	53	69	85	101	117	133	149	165	181	197	213	229	245
6	0110	<i>ACK</i>	<i>SYN</i>	&	6	F	V	f	v	â	û	ª	ł	ł	ł	μ	÷
		6	22	38	54	70	86	102	118	134	150	166	182	198	214	230	246
7	0111	<i>BEL</i>	<i>ETB</i>	‘	7	G	W	g	w	ç	ù	º	ł	ł	ł	τ	≈
		7	23	39	55	71	87	103	119	135	151	167	183	199	215	231	247
8	1000	<i>BS</i>	<i>CAN</i>	(8	H	X	h	x	ê	ÿ	¿	ł	ł	ł	Φ	°
		8	24	40	56	72	88	104	120	136	152	168	184	200	216	232	248
9	1001	<i>HT</i>	<i>EM</i>)	9	I	Y	i	y	ë	ÿ	ƒ	ł	ł	ł	θ	•
		9	25	41	57	73	89	105	121	137	153	169	185	201	217	233	249
A	1010	<i>LF</i>	<i>SUB</i>	*	:	J	Z	j		è	Û	ƒ	ł	ł	ł	Ω	·
		10	26	42	58	74	90	106	122	138	154	170	186	202	218	234	250
B	1011	<i>VT</i>	<i>ESC</i>	+	;	K	[k	{	ï	ç	¼	ł	ł	ł	δ	√
		11	27	43	59	75	91	107	123	139	155	171	187	203	219	235	251
C	1100	<i>FF</i>	<i>FS</i>	,	<	L	\	l		î	£	½	ł	ł	ł	∞	ⁿ
		12	28	44	60	76	92	108	124	140	156	172	188	204	220	236	252
D	1101	<i>CR</i>	<i>GS</i>	-	=	M]	m	}	ì	¥	ı	ł	=	ł	∅	²
		13	29	45	61	77	93	109	125	141	157	173	189	205	221	237	253
E	1110	<i>SO</i>	<i>RS</i>	.	>	N	^	n	~	Ä	£	«	ł	ł	ł	€	■
		14	30	46	62	78	94	110	126	142	158	174	190	206	222	238	254
F	1111	<i>SI</i>	<i>US</i>	/	?	O	_	o	DEL	À	f	»	ł	ł	ł	∩	
		15	31	47	63	79	95	111	127	143	159	175	191	207	223	239	255

Caracteres de Control

Caracteres Gráficos

Para escribir caracteres especiales se utiliza el símbolo \ (barra invertida):

Ejemplo:

'\n' Salto de línea
 '\'' Comilla simple
 '\\ ' Símbolo \

3.3.3.-Enteros

Es un subconjunto del conjunto matemático de los enteros. Es por tanto ordenado y numerable.

Distintos tipos de enteros en C++ (el rango y formato dependen del compilador y la máquina):

TIPO	RANGO	FORMATO (Codificación)
int	[-2147483648, 2147483647]	32 bits con signo
long	"	32 bits con signo
short	[-32768, 32767]	16 bits con signo
unsigned int	[0, 4294967295]	32 bits sin signo
unsigned short	[0, 65535]	16 bits sin signo

Operaciones sobre enteros:

- Aritméticas: +, -, *, /, %
 (¡¡Entre enteros '/' es la división entera!!)
- Relacionales: ==, !=, <, >, <=, >=
- Funciones de biblioteca: **abs** (stdlib.h)

3.3.4.-Reales

Representación de los números matemáticos reales. Es ordenado pero no numerable.

Se codifica en el ordenador en notación de coma flotante, es decir, el número está dividido en dos partes: mantisa y exponente.

$$\text{Real} = \text{Mantisa} * \text{Base}^{\text{exponente}}$$

donde la Base normalmente es binaria (base 2.)¹

La representación de constantes de tipo real en el programa se puede realizar tanto con números con punto decimal como con notación científica.

Ejemplos:

50.0	0.5 E+2
0.05	5 E -2

Los diferentes tipos de reales en C++ son:

TIPO	PRECISIÓN	RANGO	FORMATO (Codificación)
float	7 cifras	$10^{38}..10^{-38}$ ($E^{38}..E^{-38}$)	32 bits con signo
double	15 cifras	$10^{300}..10^{-300}$ ($E^{300}..E^{-300}$)	32 bits con signo
long double	18 cifras	$10^{4932}..10^{-4932}$ ($E^{4932}..E^{-4932}$)	16 bits con signo

Los números reales no se deben comparar directamente para ver si son iguales pues pueden haber diferencias debido al redondeo.

Tampoco se deben hacer más operaciones de las necesarias, ya que en cada operación se puede perder información.

Operaciones en C++ sobre reales:

- Aritméticas: +, -, *, /
(¡¡Entre reales ‘/’ es la división real!!)
- Relacionales: !=, <, >, <=, >= (Cuidado con la comparación ==)
- Funciones de biblioteca:
fabs(X), pow(X,Y), sqrt(X), sin(X), cos(X), tan(X),
asin(X), acos(X), atan(X), log(X), exp(X) (math.h)

3.3.5.-Enumerado

El tipo enumerado permite definir qué valores tendrá el tipo. Los valores son una secuencia de identificadores, donde el orden de los valores viene dado por el orden de la secuencia.

En C++ la declaración de un tipo enumerado se realiza:

```
enum Tipo { Valor1, Valor2, ..., ValorN };
```

Ejemplo:

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

3.3.6.-Puntero

El tipo puntero sirve para almacenar direcciones de memoria. Este tipo se estudiará en el *Tema 8*.

3.4. Expresiones

Una expresión es una combinación de operadores y variables o constantes. Puede ser de dos tipos: aritmética o lógica.

3.4.1.-Aritméticas

Cuando el resultado de evaluar la expresión sea un número. Los operadores que utiliza son siempre aritméticos (+, -, *, /, %)

Ejemplo:

```
8 / 2          → 4
2 * 3.0 + 2    → 8.0
2 * 3 + 2      → 8
```

3.4.2.-Lógicas

Cuando el resultado de evaluar la expresión sea un valor lógico. Los operadores que utiliza son relacionales (==, !=, <, >, <=, >=) o lógicos (!, &&, ||).

Ejemplo:

`((1 + 4) < 5) || !('a' > 'b') → true`

Cortocircuito de expresiones: En algunas expresiones lógicas es posible saber el resultado de la expresión completa sin necesidad de evaluarla totalmente (sólo se evalúa la parte necesaria para obtener el resultado).

Ejemplo:

`(1 > 2) && ('a' > 'b') → false`


Sólo es necesario evaluar `(1 > 2)` para saber que el resultado es **false**.

3.4.3.-Orden de evaluación

Cuando existe más de un operador en una expresión, éstos se han de evaluar en un cierto orden.

- 1º Todas las subexpresiones entre paréntesis. Primero los paréntesis más interiores.
- 2º Se evalúan los operadores según su orden de prioridad (tabla simplificada):

Tipo de Operador	Orden de prioridad
Unarios	!, (Signos: -, +), ++, --
Multiplicativos	*, /, %
Aditivos	+, -
Relacionales	>, <, >=, <=, ==, !=
And	&&
Or	


 +
 Nivel de
 prioridad
 -

- 3º Operadores con el mismo nivel de prioridad se evalúan de izquierda a derecha.

Ejemplo:

$$\begin{aligned}
9 + 3 * 5 / 4 \% (7 + 1) &= 9 + 3 * 5 / 4 \% 8 = \\
&= 9 + 15 / 4 \% 8 = \\
&= 9 + 3 \% 8 = \\
&= 9 + 3 = \\
&= 12
\end{aligned}$$

3.4.4.-Conversión implícita de tipos

Cuando se evalúa una expresión algebraica que implica diferentes tipos de datos, se transforman los operandos implicados al mismo tipo (el de mayor capacidad) dando el resultado en ese tipo.

Ejemplo:

$$\begin{aligned}
5.0 + 2 &\rightarrow 7.0 \\
\text{Real} + \text{Entero} &\rightarrow \text{Real} + \text{Real} \rightarrow \text{Real}
\end{aligned}$$

(los operandos se convierten al tipo más "grande")

3.4.5.-Conversión explícita de tipos (*casting*)

Hay ocasiones en que se desea transformar a un tipo concreto los datos con los que estamos trabajando, de manera que explicitamos el tipo al que deseamos la conversión. A esta conversión se la llama conversión explícita de datos.

Ejemplo:

$$\begin{aligned}
\text{int}(5.7) &\rightarrow 5 & \text{float}(5) &\rightarrow 5.0 \\
\text{int}('A') &\rightarrow 65 & \text{char}(65) &\rightarrow 'A' \\
\text{Dias}(1) &\rightarrow \text{martes} & \text{int}(\text{martes}) &\rightarrow 1
\end{aligned}$$

Antes de realizar un casting se debe estar seguro de que el resultado no se saldrá del tipo.

Ejemplo:

```
int(2E100)
```

3.5.-Estructura general de un programa

Un programa se divide en dos partes bien diferenciadas. Un bloque de declaraciones, donde se especifican todos los elementos (variables, tipos, funciones, ...) que se van a utilizar en el programa y un bloque de definiciones, donde se escribe el código de las distintas funciones que componen el programa. Esta estructura se vuelve a repetir para cada función que se define (las funciones se verán en el Tema 5).

- Cabecera (Nombre del programa, autor, propósito, ...)

- Parte declarativa	{	<ul style="list-style-type: none"> - Declaración de constantes - Declaración de tipos - Declaración de variables - Declaración de funciones (prototipos)
---------------------	---	--

- Definición de la función principal (main())

- Definición de funciones

Cualquier identificador que se vaya a utilizar debe haber sido declarado previamente.

3.5.1.-Declaración de constantes:

Las constantes declaradas tienen las siguientes ventajas:

- Claridad: Es más claro para que sirve NUMMAX que 25.
- Facilidad de modificación: Sólo se modifica la declaración.
- Prevención de errores de escritura: Es más difícil equivocarse poniendo PI que 3.1415926.

En C++:

```
const Tipo ident = Valor;
```

Ejemplo:

```
const int NUMMAX = 25;

const float IVA = 13.0;
```

3.5.2.-Declaración de tipos:

C++:

```
typedef Tipo ident;
```

Ejemplo:

```
typedef int entero;
```

3.5.3.-Declaración de variables:

C++:

```
tipo ident; // Declaración de una variable
tipo ident1, ident2, ...; // Declaración de varias variables
tipo ident = Valor; // Declaración de var. con valor inicial
```

Ejemplo:

```
int contador;

float num1, num2;

int cont = 1;
```

3.5.4.-Función principal:

La función principal siempre tendrá una de las siguientes formas:

```
int main()                                0 int main(int argc, char* argv[])
{                                          {
...(Cuerpo de la función)                ... (Cuerpo de la función)
return 0;                                  return 0;
}
```

Ejemplo:

```

/*****/
/* Programa 1 */
/* Autor: F. Barber */
/*****/

const float PI = 3.14159265;

typedef int entero;

int contador;
entero contador2;
float real;

int main()
{
    cout << "Hola";
    return 0;
};

/*****/
/* Programa 2 */
/* Autor: F. Barber */
/* Programa para el cálculo del área de un triangulo */
/*****/

int main()
{
    float base, altura, area;

    cout << " Cálculo del área de un triangulo" << endl;
    cout << "Introduce la base del triangulo: ";
    cin >> base;
    cout << "Introduce la altura del triangulo: ";
    cin >> altura;
    area = base * altura / 2;
    cout << "El área del triángulo es: " << area << endl;
    return 0;
};

```

3.6. Sentencias básicas

3.6.1.-Asignación:

Se modifica el valor de una variable.

Pseudocódigo: Var ← Expr

C++: Var = Expr

Ejemplo:

```
X = 5;  
A = 3 * 5 + 2;
```

3.6.2.-Sentencia vacía:

Sentencia que no hace nada.

Ejemplo:

```
A = B; ;
```

3.6.3.-Sentencias de entrada y salida:

Pseudocódigo: **leer A**

escribir A

C++: **cin >>** } (Se verán en detalle en prácticas).
 cout <<

Para poder utilizarlas hay que incluir antes la librería de entrada/salida:

```
#include <iostream.h>
```