

## Capítulo 1

1. Algoritmo – Procesador
2. Sintaxis Pascal
3. Tipos de datos, operadores
4. Estructura de un programa (Declaraciones y Sentencias ejecutables)
5. Errores sintácticos y semánticos
6. Ejemplos con estructura secuencial
7. Funciones del lenguaje Pascal

1. Un **algoritmo** es una secuencia ordenada y finita de pasos, exenta de ambigüedades que lleva a la solución de un problema dado. Cada paso describe una acción.

Se puede desarrollar un algoritmo para **escribir los N primeros números naturales (para un valor de N dado)**, en cambio no es posible hacerlo para escribir todos los números naturales, pues los pasos de la solución serían infinitos.

Las etapas para construir el algoritmo que resuelve un problema son:

- a. Leer el enunciado del problema con atención, comprender su complejidad, determinar qué resultados se desea obtener y cuáles son los datos conocidos, o sea nuestro punto de partida.
- b. Inventar un ejemplo concreto para el problema planteado. (Ejemplo :  $N=5 \rightarrow 1, 2, 3, 4, 5$ )
- c. Resolverlo "manualmente", analizando qué operaciones debemos efectuar sobre los datos, cómo están relacionados los mismos, cómo se condicionan entre ellos (comenzaría con el 1, lo escribo, luego le sumo uno, lo escribo ..., repito estas operaciones hasta que el número que obtengo es igual a N)
- d. Describir el algoritmo en un lenguaje apropiado (por ej: Pascal).

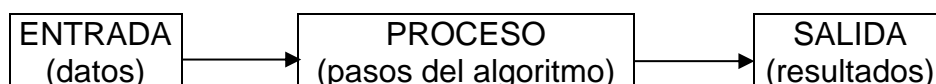
El lenguaje en el que se describe un algoritmo depende del problema a resolver y luego el programa ejecutable será dependiente del procesador en el que se ejecutará.

**Procesador**, entidad capaz de ejecutar código.

Una computadora es, entre otras cosas, un procesador capaz de captar datos, operarlos y mostrar/generar resultados

Para desarrollar algoritmos (programas) que ejecutará una computadora no basta con conocer el tipo de tareas que puede comprender y realizar, sino también la sintaxis para expresarlas.

Como punto de partida se considera un modelo computacional muy sencillo



La computadora que corresponde a este modelo es:



Los pasos que describe el algoritmo se realizarán en el orden en el que están escritos, deben estar dentro del repertorio que comprenda la computadora y ser eficaces para que produzcan la salida correcta para todas las posibles variaciones de la entrada. Incluso deben considerarse valores críticos para la entrada (ejemplo  $N=0$ ) a tener en cuenta en el planteo de la solución.

Los procesos leen los datos, los operan y escriben los resultados.

Para evaluar el funcionamiento de un algoritmo hay que determinar su estado, se concibe como una instantánea que describe el antes y el después de la ejecución de uno o varios pasos. Es de especial interés el estado inicial y final de un algoritmo.

El primero es la descripción de la entrada antes de que se ejecute el primer paso del algoritmo y recibe el nombre de precondición. En el ejemplo visto sería N entero positivo,  $N > 0$ . El estado final describe la entrada y la salida después que se ha ejecutado el algoritmo y recibe el nombre de poscondición. En general la entrada queda vacía, esto indica que, una vez leídos los datos, no pueden leerse de nuevo. Los datos ingresan por lectura en el mismo orden en el que son tipeados en el teclado, o que han sido grabados en un archivo.

Los estados intermedios son las transformaciones que van sufriendo los datos para llegar a los resultados requeridos.

Estas especificaciones requieren del concepto de **variable**, nombre simbólico que se asocia a un valor durante la ejecución del algoritmo, pero que no puede determinarse (indefinida) en el momento que se construye o formula el algoritmo.

Un lenguaje algorítmico

- ✓ tiene una sintaxis y un vocabulario limitado
- ✓ comprende solo el tipo de acciones que una computadora puede entender y realizar
- ✓ cada acción tiene una forma rigurosa de expresarse y una significación propia (sintaxis y semántica)

La descripción de un algoritmo en un lenguaje de programación constituye un **programa**

## 2. Sintaxis del lenguaje Pascal

Un programa se compone de

☞ **Declaraciones:** describen características de los objetos/elementos que utiliza el programa.

☞ **Sentencias** o instrucciones ejecutables: describen las acciones que el programa realizará. Pueden ser simples o compuestas.

### SIMPLES

No contienen otra sentencia

Ejemplos : Write, Read

### COMPUESTAS

Grupo de sentencias simples encerradas entre un BEGIN y un END.

Ejemplo: BEGIN

sentencia 1;

sentencia 2;

sentencia n;

END;

A continuación se describen algunos elementos del lenguaje y características generales

**Delimitadores:** se utilizan para agrupar o separar declaraciones o sentencias.

Ejemplos: ';' - 'Begin' - End'.

**Comentarios:** se utilizan para hacer aclaraciones en cualquier lugar del programa (son ignorados por el compilador). Se recomienda su uso para mayor legibilidad. Se encierran entre llaves o paréntesis y asterisco. Ejemplo: {esto es un comentario} , (\*esto es un comentario\*).

**Palabras reservadas:** son aquellas que sólo pueden usarse para un fin específico.

Ejemplos: *begin*, *function*, *program*, etc.

**Identificador:** es el nombre que se le asigna a un objeto (variable, constante, programa, subprograma), debe comenzar con una letra y a continuación letras o dígitos o el carácter '\_'.

Ejemplo: *salario\_bruto*, *cuenta*, *x*

No hay distinción entre mayúsculas y minúsculas.

Ejemplo: *Suma* y *suma* son la misma variable

No hay restricción respecto a la longitud, pero debe considerarse la legibilidad así como también la facilidad para comprender y escribir el código. Es conveniente que el identificador esté asociado al rol de dicho objeto (Precio, Nombre, Promedio, SueldoProm, etc.)

### 3. Tipos de Datos

Los datos con los que opera un programa tienen características propias. Por ejemplo, la edad es un entero, el precio es un real, el nombre es una cadena de caracteres, etc.

El lenguaje provee tipos estándar, o simples, para enteros, reales, caracteres y lógicos. Los dos primeros tienen a su vez varias posibilidades de rango y precisión.

Los tipos simples son aquellos que permiten almacenar un único valor.

El programador puede declarar otros tipos para describir sus datos.

El tipo cadena permite almacenar una secuencia de caracteres y tratarla como una unidad de información.

SIMPLES (escalares estandar)	REALES	SINGLE	1.5E-45 a 3.4E38
		REAL	2.9E-39 a 1.7E38
		DOUBLE	5.0E-324 a 1.7E308
		EXTENDED	1.9E-4851 a 1.1E4932
	ENTEROS	SHORTINT	-128 a 127
		BYTE	0 a 255
		WORD	0 a 65535
		INTEGER	-32,768 a 32,767
		LONGINT	-2,147,483,648 a 2,147,483,648
	CHAR		
	BOOLEAN		
DEFINIDOS por el USUARIO	SUBRANGO		
	ENUMERADOS		
STRING			
ESTRUCTURADOS	ARRAY		
	RECORD		
	SET		
	FILE		

A continuación se presenta una parte (la más utilizada) de la tabla ASCII, para la representación de caracteres, cada uno tiene una representación numérica, por ejemplo el carácter 'A' se codifica internamente con el número 65, el carácter blanco como el 32. El número 31 no es representable como carácter y la representación del 30 coincide con la del 45.

Para visualizar un carácter debe mantener presionada la tecla **alt** mientras digita su respectivo código numérico.

ACII	0	1	2	3	4	5	6	7	8	9
30	-		blanco	j	“	#	\$	%	&	‘
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	¿	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{		}	~	•	Ç	ü

Como se ve en la tabla, existe un orden entre los caracteres y es posible compararlos en forma aislada 'A' < 'h'.

Notar que es diferente la representación del valor entero 5 (internamente un 5 en binario) del carácter '5' (internamente un 53 en binario)

Un variable de tipo string es una secuencia de hasta 255 caracteres, delimitada entre apóstrofes. Es posible limitar el número de caracteres de un *string* especificando entre corchetes dicha cantidad, por ejemplo `string[n]`, siendo **n una constante entera** menor a 256.

La longitud de la cadena almacenada se obtiene con la función *length(...)*, dicho valor coincide con la posición del último carácter.

Es posible el acceso individual a un carácter de la cadena, indicando a continuación del nombre de la variable, su posición entre corchetes.

Se puede establecer una relación entre cadenas, ya que al estar conformadas por caracteres el orden de estos determina el resultado de la relación. Por ejemplo 'Papa' > 'PAPA'.

El tipo boolean toma valores TRUE o FALSE, estos valores no pueden ser ingresados (lectura), sí pueden ser visualizados (escritura).

A los tipos integer, boolean, char se los conoce como ordinales. Un tipo de datos es ordinal cuando el conjunto de valores que representa se puede contar, es decir, se puede establecer una relación uno a uno entre sus elementos y el conjunto de los números naturales. Esta relación es de orden porque se puede establecer un predecesor y un sucesor para cada valor del tipo ordinal. Esta característica será importante en la implementación de ciertas estructuras de control (Case, For).

Los tipos estructurados permiten almacenar un conjunto de datos, a diferencia de los simples que almacenan un único dato.

**Variable:** objeto de un programa cuyo valor puede “variar”, es una o más celdas de memoria en las que se almacena información. Tiene dos atributos o características propias: un *identificador* y un *tipo*.

Es importante que el identificador, o nombre, de las variables informe acerca de su contenido, esto hace más fácil la comprensión de un algoritmo. Se deben establecer criterios para la elección de los identificadores, adoptaremos que las palabras que lo integran comiencen con mayúscula. Ejemplo: SalarioBruto

No confundir identificadores o nombres de variables con valores de tipo cadena, estos últimos están delimitados por apóstrofes y no son equivalentes las mayúsculas y minúsculas.

Ejemplo : 'PERA' <> 'peRa'

Todas las variables que se utilizan en un programa deben ser declaradas especificando su nombre y tipo.

**Constante** objeto de un programa cuyo valor no cambia.

## Operadores

Ejemplo

VAR

C, D, Z, Y : integer;

A, B, E :real;

Car :char;

Cad :string;

OPERADORES	TIPOS	EJEMPLOS
ARITMETICO	*, / , MOD , DIV, + , -	entero, real
ALFANUMERICO	+	char, string
RELACIONAL	< , > , <= , >= , = , <>	todos
LOGICO	NOT , AND , OR	boolean

- ✓ Orden de prioridad : aritmético y alfanumérico – lógico – relacional
- ✓ Dicha prioridad se altera utilizando paréntesis
- ✓ Si en una expresión el orden de precedencia de los operadores es el mismo, se resuelve de izquierda a derecha.

Existen funciones definidas por el lenguaje que facilitan los cálculos, a continuación se presentan alguna de las más utilizadas en Pascal.

Nombre	Tipo del argumento	Resultado	Tipo del resultado
Abs(x)	Real o Integer	el valor absoluto del argumento	Real o Integer
Frac(x)	Real	la parte decimal del argumento	Real
Int(x)	Real	la parte entera del argumento	Real
Round(x)	Real o Integer	el entero más próximo al argumento	Integer
Sqr(x)	Real o Integer	el cuadrado del argumento	Real
Sqrt(x)	Real o Integer	la raíz cuadrada del argumento	Real
Trunc(x)	Real	la parte entera del argumento	Integer
Pi	no posee	3.1415926535897932385	Real
Ucase(x)	char	la mayúscula del argumento si éste es una letra minúscula, sino devuelve el mismo caracter	char
Odd(x)	entero	el valor lógico True si el argumento de la función es impar y False si es par	boolean
Random [(n)]	entero (opcional)	devuelve un número aleatorio. Sin argumento el número aleatorio real entre 0 y 1. Con argumento el número aleatorio entero entre 0 y n-1	Real o Integer
Length(S)	cadena de caracteres	la longitud lógica de S	byte

## 4. Estructura de un Programa

Un programa en Pascal consta de un encabezamiento y un bloque dividido en secciones; las primeras son declarativas, y la última ejecutable. Todas las secciones excepto la última pueden estar vacías.

<pre>PROGRAM &lt;identificador&gt;; {encabezamiento del programa}  USES   &lt;identificadores&gt;;  CONST   &lt;definiciones de constantes&gt;;  TYPE   &lt;declaración de tipos de datos def. por el usuario&gt;;  VAR   &lt;declaración de variables&gt;;  &lt;definiciones de procedimientos y funciones&gt;;  BEGIN   &lt;sentencias&gt; END.</pre>	<div>DECLARATIVA</div> <div>EJECUTABLE</div>
---	--

### 4.1. Descripción de la sección Declarativa

**4.1.1. USES** declara las unidades (módulos compuestos por subprogramas compilados independientemente). Permite al programa utilizar cualquier subprograma incluido en las unidades que declara 'usar'.

**4.1.2. CONST** declara las constantes, identificándolas con un nombre.

```
CONST
  n1 = v1; {n1, n2 identificadores}
  n2 = v2; {v1, v2 números, caracteres, cadenas, expresiones o identificadores de
constantes}
```

**4.1.3. TYPE** en esta sección se declaran los tipos de datos definidos por el programador.

```
TYPE
  i1 = t1; {i1, i2 identificadores de los nuevos tipos}
  i2 = t2; {t1, t2 indican la estructura del tipo que se está definiendo }
```

**4.1.4. VAR** Debe especificarse el nombre de la variable y su tipo, donde el nombre es un identificador y el tipo puede haber sido declarado en la sección de tipos previamente o ser un tipo predefinido

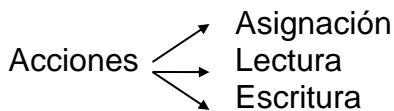
```
VAR
  nombre1 : tipo1;
  nombre2 : tipo2;
  ...
  nombre3, nombre4 : tipo3; {distintas variables tienen el mismo tipo}
```

**4.1.5. FUNCTION Y PROCEDURE** son las palabras reservadas que identifican a las funciones y procedimientos que se declaran y desarrollan (condición necesaria para poder utilizarlos). Estos subprogramas son módulos de programación autónomas que pueden ser invocados (o activados) desde la sección de sentencias ejecutables o por otros procedimientos o funciones.

Las funciones estándar o predefinidas no se declaran.

## 4.2. Sentencias de la sección Ejecutable

Una variable puede recibir valor de dos formas **asignación y lectura**, en ambos casos si tiene un valor almacenado previamente lo pierde. Su contenido se puede visualizar la mediante **escritura**.



### 4.2.1. ASIGNACION

VARIABLE := <expresión>

Ejemplos :  
 A := B \* C  
 Cad := 'casa'

El **valor** a la derecha se almacena en la variable de la izquierda, o sea que almacenamos un dato en la posición de memoria asociada al nombre de esa variable.

El tipo de la expresión debe ser igual al de la variable, salvo dos excepciones :

- La variable es real y la expresión es entera
- La variable es cadena y la expresión es carácter

*La variable de la izquierda pierde el valor que almacena, al recibir otro por asignación.*

*Las variables que figuran a la derecha de la asignación no pierden su valor.*

### 4.2.2. LECTURA

**Readln** (lista de variables separadas por comas); {lee los datos desde teclado almacenando en la lista de variables según el orden en que se ingresan}

Los datos que darán valor a las variables de la lista son ingresados desde el teclado.

*Importante: No tiene sentido asignar un valor a una variable inmediatamente antes o después de haber leído sobre la misma, ya que lo pierde quedando el correspondiente a la última operación*

### 4.2.3. ESCRITURA

**Write**(lista de salida); {escribe los elementos de la lista y queda posicionado en la misma línea}

**Writeln**(lista de salida); {escribe los elementos de la lista y queda posicionado al comienzo de la línea siguiente}

### Escritura con formato

Es posible establecer, para cada elemento de la lista de salida, el *ancho* del campo donde se escribe. Basta colocar después del elemento entero o cadena o carácter, el carácter ':' y la cantidad de columnas del campo de escritura.

Writeln(*elemento*: *Ancho*);

*Ancho* es un entero que indica la cantidad de columnas del campo en el que se escribe el elemento.

WriteIn(elemento real : Ancho: Decimales);

*Decimales* es un entero que especifica la cantidad de dígitos decimales con los que se escribirá el número real.

A: real; C: integer;

C:=10 ;

```
WriteLn(C:10) ;
```

```
WriteIn(A:2:4) ;
```

1	2	3	4	5	6	7	8	9	10
				3	.	4	6		
3	.	4	5	6	0			1	0

## 5. Errores de programa

WriteIn (A ; es el resultado),

Prom := A + B+ C / 3;

**1.- Conociendo el largo y ancho de un rectángulo, informar el perímetro y la superficie**

Var

Begin

Readln (Largo, Ancho);

Sup:= Largo \* Ancho;

End.

Program Dos;

Var

Importe, PrecioUnitario: real;

Cantidad: byte;

Begin

```
Writeln('Ingrese la cantidad de unidades que compró');
```

Readln(Cantidad);

```
Writeln('Ingrese el importe que pagó');
```

```
Readln(Importe);
```

```
PrecioUnitario := Importe/Cantidad; {es necesaria esta variable?}
```

Writeln ('El precio unitario es ', PrecioUnitario ) {podríamos mejorar el modo de mostrar este valor?}

End.

**3.-** A partir de dos puntos en el plano, calcular e informar la distancia entre ambos.

Program Tres;

Var

    X1 , Y1 , X2 , Y2 : real;

Begin

Writeln('Ingrese coordenadas del primer punto');

Readln (X1, Y1);

Writeln('Ingrese coordenadas del segundo punto');

Readln (X2, Y2);

Writeln('La distancia entre ambos puntos es', sqrt( sqr(X2 - X1) + sqr( Y2 - Y1) ));

End.