



Tema 1

Elementos básicos de Pascal

Introducción a la Programación

Grado de Ingeniería Informática



Tema 1

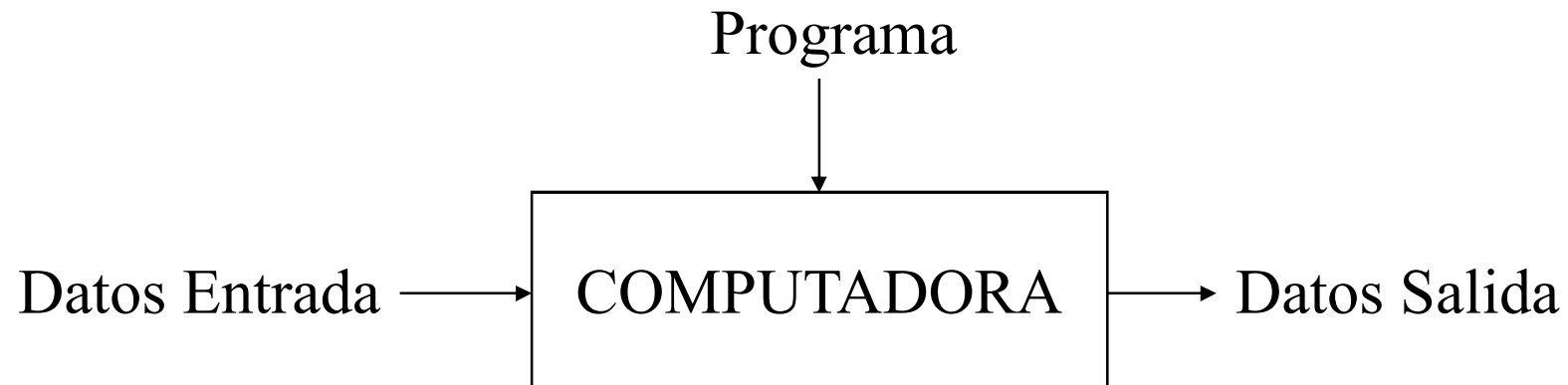
- Introducción
 - Problemas, algoritmos y programas
 - Paradigmas y lenguajes de programación
 - Desarrollo sistemático de aplicaciones
- Elementos básicos de Pascal
 - Historia de Pascal
 - Tipos de datos básicos
 - Elementos básicos del lenguaje
 - La documentación del programa
- Tipos de datos definidos por el programador: subrangos



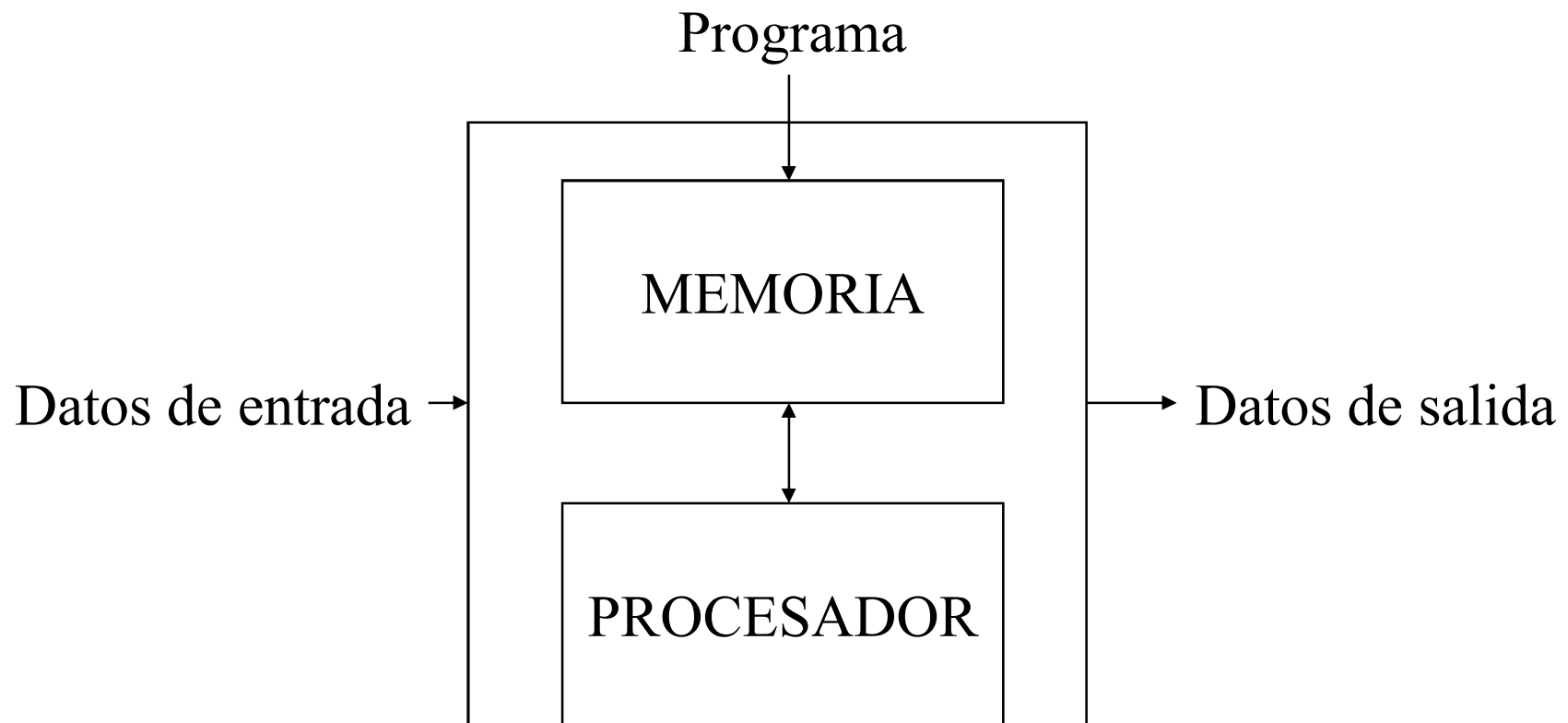
Objetivo

- Exponer los conceptos clave para la resolución de problemas por medio de la computadora u ordenador
- Presentar los elementos básicos del lenguaje de programación Pascal:
 - Conocer el concepto de tipo de dato, en concreto, los tipos de datos básicos predefinidos en el lenguaje
- Presentar la correcta construcción de programas sencillos en Pascal:
 - Conocer la estructura general de un programa
 - Utilizar las instrucciones de entrada y salida de datos

■ Organización de una computadora



■ Componentes de una computadora





- ¿En qué consiste la programación?

Describir lo que debe hacer la computadora para resolver un problema concreto utilizando un lenguaje de programación

- Fases para resolver un problema con una computadora:
 1. Análisis del problema
 2. Descripción de un método (algoritmo) que lo resuelva
 3. Escritura del algoritmo en un lenguaje de programación
 4. Comprobación del correcto funcionamiento
-



1.1 Problemas, algoritmos y programas



La definición del problema

- Definición: Problema
 - Proposición encaminada a averiguar el modo de obtener un resultado, cuando se conocen ciertos datos de partida
 - Tipos de Problemas
 - **Sin solución**
 - **Determinados:** con una única solución
 - **Indeterminados:** con un número indefinido de soluciones
-



La definición del problema

- Análisis del problema

Consiste en establecer con precisión **qué** se plantea

- Especificación

Descripción precisa del problema:

- datos de partida
- resultado

lenguaje natural



puede resultar impreciso

lenguajes formales



lógica, matemáticas



Un ejemplo

Ejemplo de Especificación: Problema de división euclídea

Especificación:

Datos

- 2 enteros, dividendo y divisor (D, d)
- d no nulo

Resultado

- 2 enteros, cociente y resto (C, R)
- $0 \leq R < d$, tal que $D = d * C + R$



Algoritmo

- Definición 1:

- Descripción precisa de los pasos que nos llevan a la solución de un problema planteado

- Definición 2:

- Método tal que partiendo de datos apropiados, conduce sistemáticamente a los resultados requeridos en la especificación del problema
-



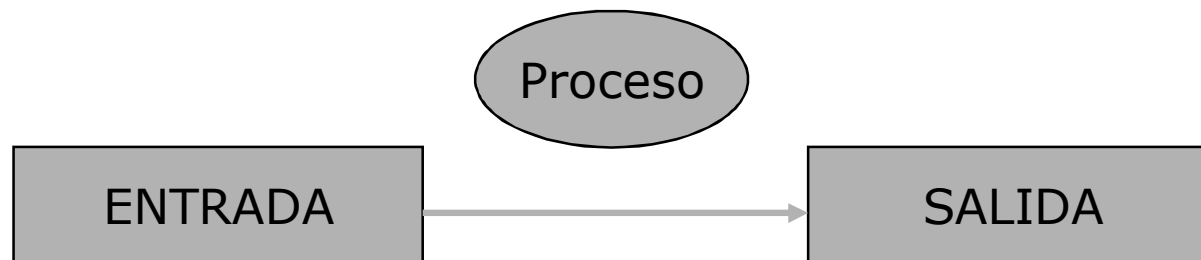
Caracterización de un algoritmo

La descripción de un algoritmo afecta a:

Entrada, que son los datos necesarios

Proceso o instrucciones a ejecutar

Salida de resultados





Caracterización de un algoritmo

- **Algoritmo \cong función matemática**

Ejemplo: Suma Lenta:

$$\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$$

$$a + b \rightarrow c, \quad c = a + b$$

Es constructivo: hay que precisar también el ***proceso*** de cálculo



Caracterización de un algoritmo

- Precisión (sin ambigüedad) en cuanto a:
 - **Orden**: secuencia de pasos que han de llevarse a cabo
 - **Contenido**: qué se realiza en cada paso
- Determinismo:
 - Debe responder del mismo modo ante las mismas condiciones
- Finitud:
 - Debe tener fin



Lenguajes algorítmicos

- Sirven para describir un algoritmo
- Son más precisos que el lenguaje natural, pero menos rígidos (o formales) que un lenguaje de programación
 - Se los considera un lenguaje intermedio
 - Tienen cierta independencia de los lenguajes de programación
- Ejemplos:
 - Pseudocódigo, diagramas de flujo



Ejemplo: Algoritmo Suma Lenta

Partimos de dos cantidades: a y b . El método de suma lenta consiste en ir pasando de a a b una unidad cada vez, de forma que cuando $a=0$, el resultado será el valor de b

Algoritmo Suma Lenta (Pseudocódigo)

1. Sean $a, b \in \mathbb{N}$
 2. Leer (a, b)
 3. Mientras ($a \neq 0$) hacer
 - $a := a-1$
 - $b := b+1$
 - fin_mientras
 4. Escribir (b)
-



Aspectos de un algoritmo

■ Obligados

- ❑ **Corrección**: respecto a las especificaciones
- ❑ **Complejidad**: recursos que un algoritmo necesita. En máquinas secuenciales (tiempo y memoria)

■ Deseables

- ❑ **Generalidad**: sirva para una clase de problemas lo más amplia posible
- ❑ **Eficiencia**: será más eficiente en la medida que necesita de menos pasos



Problemas y algoritmos

- Algunos problemas tienen distintas soluciones algorítmicas.
 - Ejemplo: máximo común divisor (mcd)
- Algunos problemas no tienen solución algorítmica
 - Ejemplo: problema de parada (encontrar un algoritmo que determine si otro algoritmo finaliza o no con unos determinados datos de entrada)



Programas

- Programa: Conjunto de instrucciones precisas, en un lenguaje entendible por la computadora
- Programación: Proceso de construcción de programas
- Fases:
 - Análisis del problema
 - Solución conceptual del problema
 - Escritura del algoritmo en un lenguaje de programación
 - Comprobación de resultados



Ejemplo: Programa Suma Lenta

Programa Pascal (codificando el algoritmo Suma Lenta)

```
PROGRAM SumaLenta (input,output);  
  
    {Se suman dos enteros positivos,  
    pasando unidades de uno a otro}  
  
VAR  
    a,b:integer;  
BEGIN  
    readln(a,b);  
    WHILE a <> 0 DO BEGIN  
        a:=a-1;  
        b:=b+1  
    END; {while}  
    writeln(b)  
END. {SumaLenta}
```



1.2 Paradigmas y Lenguajes de programación



Lenguajes de programación

- Definición:
 - ❑ Se trata de un lenguaje artificial diseñado para representar algoritmos de forma inteligible para las computadoras

- Lenguaje de programación y lenguaje natural
 - ❑ Los lenguajes de programación son más formales y rigurosos
 - ❑ Además, son más simples en su sintaxis y semántica



Lenguajes de programación

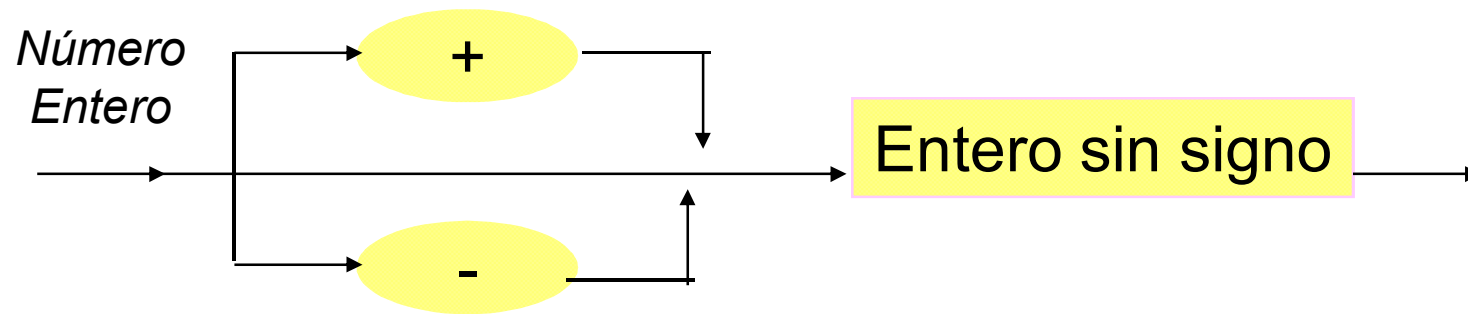
- Algunas características relevantes:
 - Sintaxis
 - Semántica
 - Traducción y ejecución
 - Errores y cómo subsanarlos



Sintaxis

- Especifica inequívocamente cómo están contruidos los programas de un LP
- Especificación de la sintaxis
 - Gramáticas (BNF)
 - Diagramas Sintácticos

Ejemplo: Sintaxis de un número entero



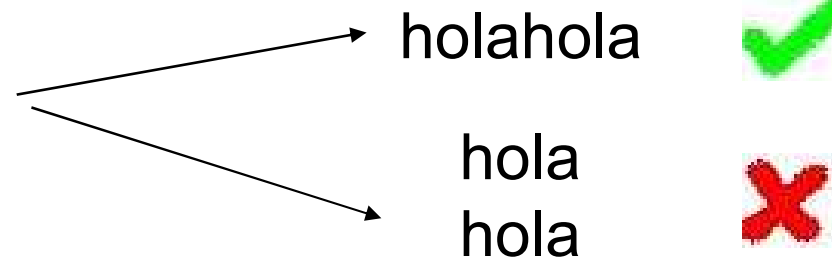


Semántica

- Asigna un significado a cada tipo de construcción de un LP
- Formas de especificación:
 - ejemplos (y contraejemplos) en los manuales
 - definición formal

Ejemplo:

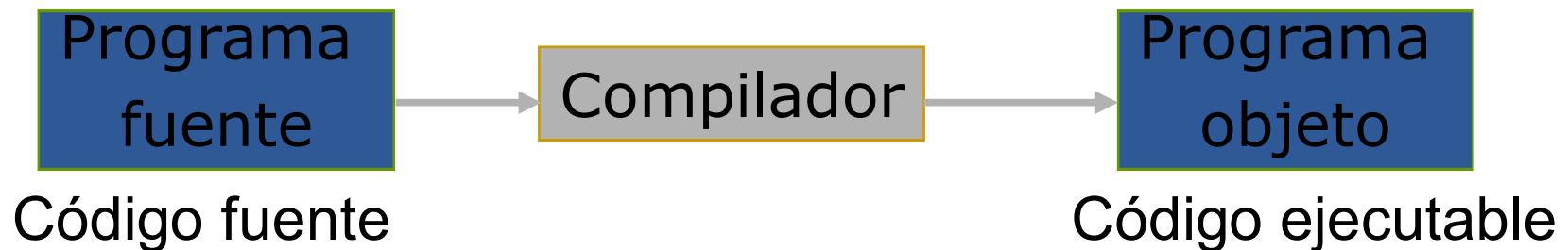
```
write('hola');  
write('hola');
```





Traducción y ejecución

- El lenguaje de alto nivel ha de traducirse al lenguaje de la máquina
- Formas de traducción:
 - **Compilación:**
 - todo el código fuente (en un archivo) se traduce a código ejecutable (en otro archivo)
 - se ejecuta dicho código ejecutable





Traducción y ejecución

■ **Interpretación:**

- ❑ Se traduce una instrucción del código fuente
- ❑ Se ejecuta dicha instrucción
- ❑ Se repiten los pasos anteriores con todas las instrucciones del código fuente



Errores

- Errores de compilación
 - ❑ Surgen a la hora de traducir (“compilar”) el código fuente
 - ❑ Errores sintácticos, de tipo, etc.
 - Errores de ejecución
 - ❑ Surgen al ejecutar el código ejecutable
 - ❑ Operaciones ilegales (división por cero), errores lógicos etc.
-



Evolución lenguajes de programación

- ❑ Prog. en código máquina
- ❑ Prog. en ensamblador
- ❑ Prog. con lenguajes de alto nivel

- ❑ Prog. estructurada
- ❑ Prog. modular
- ❑ Prog. con TAD's
- ❑ Prog. orientada a objetos
- ❑ ...



tiempo



Evolución lenguajes de programación

- Motores que impulsan la evolución de los lenguajes de programación:
 - ❑ Abstracción
 - ❑ Encapsulación
 - ❑ Modularidad
 - ❑ Jerarquía
-



Evolución lenguajes de programación

- Abstracción:

- Proceso mental por el que el ser humano extrae las características esenciales de algo, e ignora los detalles superfluos

- Encapsulación:

- Proceso por el que se ocultan los detalles de las características de una abstracción



Evolución lenguajes de programación

- Modularización:

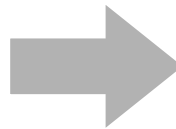
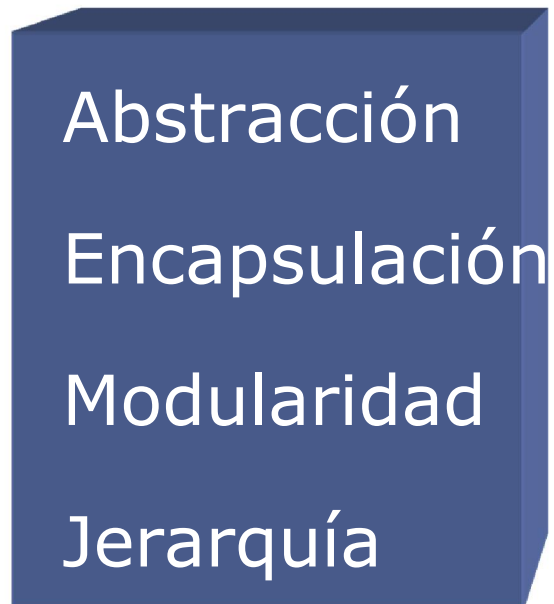
- Proceso de descomposición de un sistema en un conjunto de elementos poco acoplados (independientes) y cohesivos (con significado propio)

- Jerarquía:

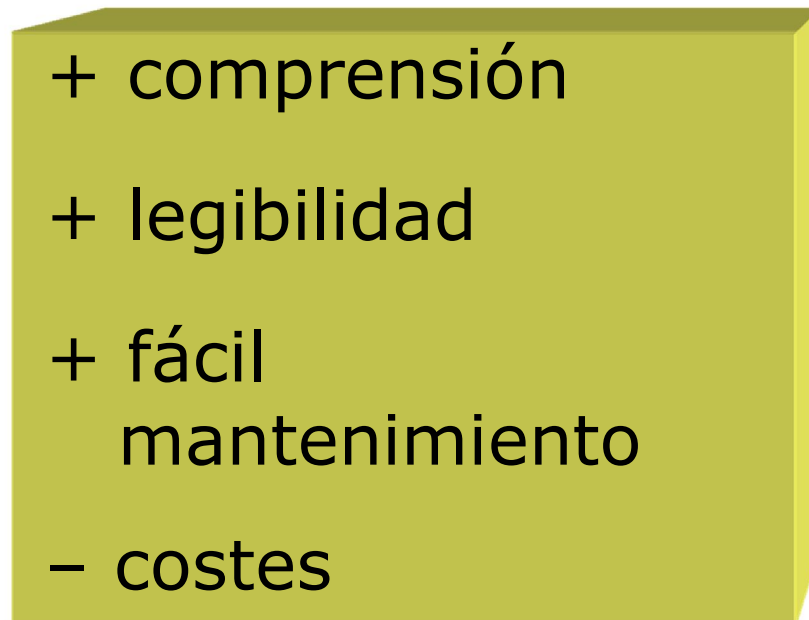
- Proceso de estructuración por el que se organizan un conjunto de elementos en distintos niveles, atendiendo a determinados criterios (responsabilidad, composición, etc.)
-

Preservar los motores

"Motores"



Ventajas





Paradigmas de programación

- Definición:

- Una colección de patrones conceptuales que moldean la forma de razonar sobre problemas, de formular algoritmos y, a la larga, de estructurar programas

- Paradigmas:

- Programación imperativa
 - Programación funcional
 - Programación lógica
-



Programación imperativa

- Basada en el modelo von Neumann
 - Un conjunto de operaciones primitivas
 - Ejecución secuencial
- Abstracción
 - Variables, expresiones, instrucciones
- Programar:
 - Declarar variables necesarias
 - Diseñar una secuencia adecuada de instrucciones (asignaciones)



Programación imperativa

Ejemplo: Indicar cuál es el mayor de dos números en Pascal

- Programa:

```
PROGRAM mayorDeDosNumeros;  
VAR x,y, mayor: integer;  
BEGIN  
    Read (x,y);  
    if x > y then  
        mayor := x  
    else  
        mayor := y;  
    Write (mayor);  
END.
```



Programación funcional

- Basada en la noción de función matemática
 - $f: \text{Dominio} \rightarrow \text{Rango}$
- Programar:
 - Definir funciones básicas (con parámetros) (p.e. por enumeración)
 - Diseñar funciones complejas (p.e. por comprensión)
 - Evaluar las funciones sobre los datos de entrada



Programación funcional

Ejemplo: Calcular el máximo de 3 números en LISP

- Programa:

```
(defun max (X Y)
  (if (> X Y)
      X
      Y))
(defun max3 (X Y Z)
  (max X (max Y Z)))
```

- Ejecución:

```
> (max3 3 6 5)
> 6
```



Programación lógica

- Basada en la inferencia automática en (un subconjunto de) lógica de primer orden
- Programar:
 - ❑ Definir hechos (predicados básicos)
 - ❑ Diseñar implicaciones para definir predicados complejos
 - ❑ Determinar la verdad de los predicados para individuos concretos



Programación lógica

Ejemplo: determinar “antecedentes” en Prolog

- Programa:

```
padre(juan, antonio).  
padre(antonio, pepe).  
antecesor(X,Y) ← padre(X,Y).  
antecesor(X,Z) ←  
    padre(X,Y) ∧ antecesor(Y,Z) .
```

- Ejecución:

```
? antecesor(pepe, juan) ⇒ no  
? antecesor(A, pepe) ⇒      A = antonio;  
                             A = juan
```



Paradigmas y lenguajes

		<i>Prog. Concurrente</i>		<i>Prog. orientada a objetos</i>	
Prog. Funcional (P.Declarativa)	LISP			CLOS	
	Hope		Haskel		
Prog. Lógica (P.Declarativa)	Prolog	Ciao-Prolog		Prolog++	
Prog. Imperativa	C	Ada		Dephi	Smalltalk
	PASCAL		Ada-95		
	Fortran	Pascal FC		C++	
	COBOL			Java	Eiffel



1.3 Desarrollo sistemático de aplicaciones



Fases de un desarrollo sistemático

- Planificación
 - Análisis
 - Diseño
 - Codificación
 - Validación
 - Mantenimiento
-



Planificación

- Determinar las necesidades de programación
 - Estimación de recursos de desarrollo
 - Predicción aproximada de coste y tiempo
 - Determinar si el desarrollo del software es viable económicamente
-



Análisis de requisitos

- Definir detalladamente las funciones de cada módulo, de acuerdo con los deseos del cliente
 - Definir detalladamente el trabajo conjunto de los distintos módulos
 - Definir criterios y sistema de validación
 - Redactar especificaciones detalladas del funcionamiento general del software
-



Diseño

- Diseñar el conjunto de bloques o módulos
 - Se dividen en partes o tareas
 - Se asignan tareas a equipos de trabajo, que las desarrollan y prueban
-



Codificación

- Escribir los algoritmos en el lenguaje de programación elegido
 - Integrar las partes para que formen un programa completo
-



Validación

- Aplicar el sistema de pruebas descrito en la fase de análisis de requerimientos
 - Métodos de validación
 - pruebas (tests), inspecciones ...
 - verificación formal
 - Objetos de validación:
 - los módulos de programa
 - las conexiones entre ellos (integración)
 - la aplicación entera
-



Mantenimiento

- Redactar la documentación actualizada
 - Iniciar la explotación
 - Detectar y subsanar errores cometidos en etapas anteriores
 - Adaptar la aplicación a requisitos cambiados
-



2.1 Historia de Pascal



2.1. Características

- Se trata de un lenguaje de alto nivel
 - Es de propósito general, aunque fue diseñado para la enseñanza de la programación
 - Estructurado (datos e instrucciones)
 - Modular
 - Compacto y fácil de entender
 - La mayoría de los traductores son compiladores
 - Facilita la adquisición de buenos hábitos para la programación
-



2.1. Evolución

- Niklaus Wirth fue su creador
 - 1968 (Algol 68)
 - 1970 (PASCAL) Instituto Politécnico de Zurich
- Debe su nombre a Blaise Pascal (1623-1662), matemático francés que inventó la primera máquina de calcular
- Pascal estándar
- Pascal extendido (Turbo Pascal)



2.2 Tipos de datos simples



2.2. Tipos de datos básicos

- **Dato:** Es la representación de un objeto mediante símbolos manejables por el ordenador
- Todos los programas manipulan datos
- Casi todos los lenguajes disponen de tipos de datos básicos (enteros, reales, ...) llamados predefinidos o estándar



2.2. Tipos de datos básicos

- Un tipo de dato se caracteriza por:
 - Su **dominio** o conjunto de valores que puede tomar
 - Las **operaciones** que se pueden realizar sobre ellos
-



2.2. Tipos de datos básicos

Tipos predefinidos en Pascal:

- Entero: `integer`
 - Real: `real`
 - Carácter: `char`
 - Booleano: `boolean`
- Dominio o rango
 - Operaciones
 - Construcción de expresiones



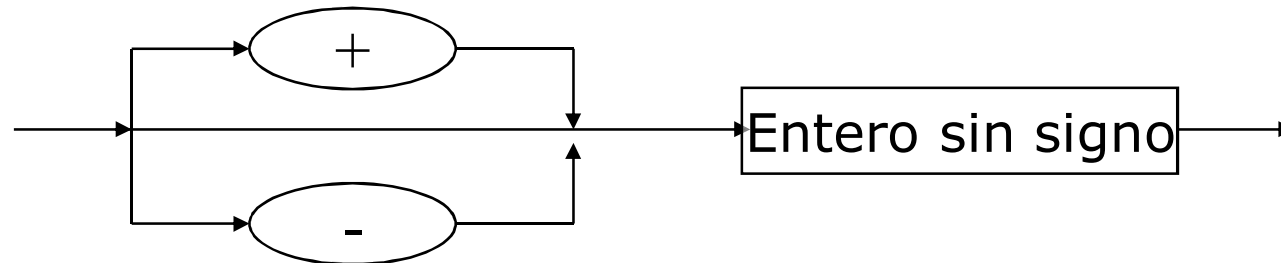
2.2.1 El tipo entero

- Nombre del tipo: integer
- Dominio: \mathbb{Z} y, debido a las limitaciones de representación, el dominio está acotado
 - En Turbo Pascal: acotado por la constante predefinida
`MAXINT` $\{-32768, \dots, 32767\}$
- Representación: Se escriben sin espacios ni puntos entre sus cifras y el signo, en caso de aparecer, precede al número



Tipo entero: Sintaxis (Diagrama sintáctico)

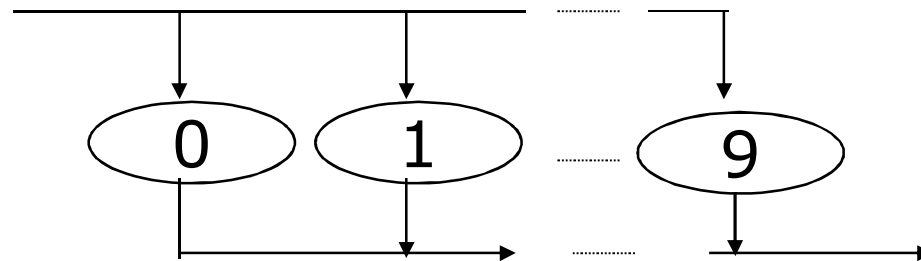
Número Entero



Entero sin signo



Dígito





Tipo entero: Operaciones

- Operadores aritméticos:
 - Binarios. Notación infija (Ej: $a+b$)
 - $+$: $Z \times Z \rightarrow Z$ (adición)
 - $-$: $Z \times Z \rightarrow Z$ (resta)
 - $*$: $Z \times Z \rightarrow Z$ (multiplicación)
 - **div** : $Z \times Z \rightarrow Z$ (división entera)
(Ej.: $5 \text{ div } 2 \rightarrow 2$)
 - **mod** : $Z \times Z \rightarrow Z$ (resto de la división entera)
(Ej.: $5 \text{ mod } 2 \rightarrow 1$)
 - Monarios. Notación prefija (Ej: -3)
 - $-$: $Z \rightarrow Z$ (cambio de signo)





Tipo entero: Funciones

- Funciones aritméticas (Monarias)
 - Sintaxis: Notación prefija
 - **abs**: valor absoluto (Ej.: $\text{abs}(-3) \rightarrow 3$)
 - **sqr**: cuadrado (Ej.: $\text{sqr}(-3) \rightarrow 9$)
 - **pred**: predecesor (Ej.: $\text{pred}(-3) \rightarrow -4$)
 - **succ**: sucesor (Ej.: $\text{succ}(-3) \rightarrow -2$)
- Semántica: Como operadores y funciones sobre enteros
 - Limitación: Posibilidad de desbordamiento



Tipo entero: Expresiones

- Una expresión entera permite la combinación de números, operaciones y funciones. El resultado de su evaluación será un entero.
- Precedencias:
 1. **()**
 2. **Funciones**
 3. ***, div, mod**
 4. **+, -**

↑ más alta
↓ más baja
- Asociatividad:
 - De izquierda a derecha
 - Ej.: $5 + 3 - \text{sqr}(-7) * 2 = -90$  -82 



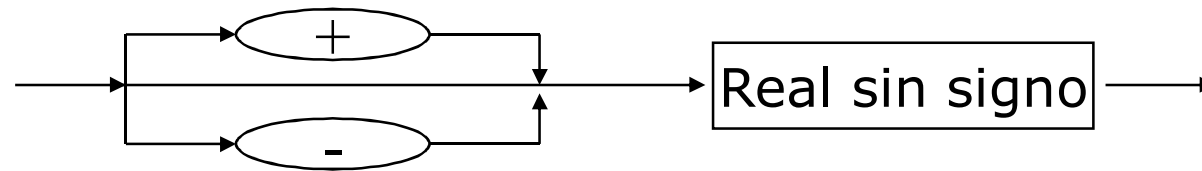
2.2.2 El tipo real

- Nombre del tipo: `real`
- Dominio: `R`
- Intervalo acotado
 - Magnitud: Entre $\pm 2.9 \times 10^{-39}$ y 1.7×10^{38}
 - Precisión: Entre 11 y 12 cifras significativas
- Dos formas de representación:
 - Coma fija (Ej.: 3.25, -3.0, 666.444)
 - Notación científica o coma flotante (Ej.: 6.023E+23)

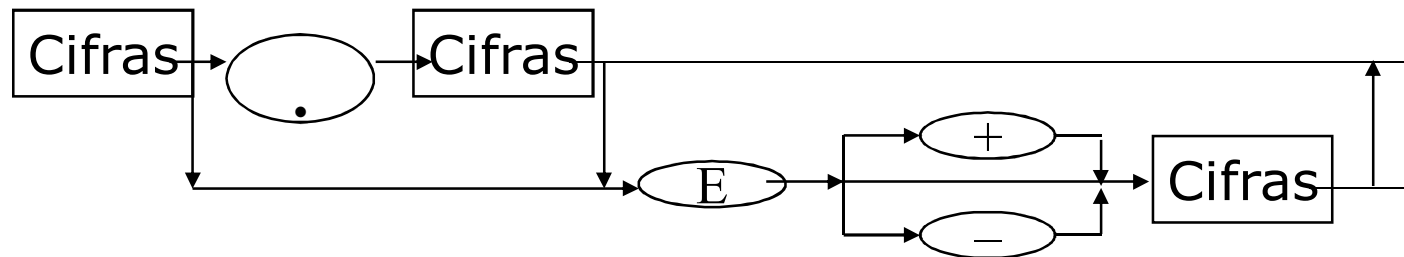


Tipo real: Sintaxis (Diagrama sintáctico)

Número Real



Real sin signo



Cifras





Tipo real: Operaciones

- Operadores aritméticos.
 - Binarios. Notación infija (Ej: $a+b$)
 - $+$: $R \times R \rightarrow R$ (adición)
 - $-$: $R \times R \rightarrow R$ (resta)
 - $*$: $R \times R \rightarrow R$ (multiplicación)
 - $/$: $R \times R \rightarrow R$ (división)
 - Monarios. Notación prefija (Ej: -3.9)
 - $-$: $R \rightarrow R$ (cambio de signo)



Tipo real: Funciones

■ Funciones aritméticas (Monarias)

□ Sintaxis: Notación prefija

□ **abs:** Valor absoluto

□ **sqr:** Cuadrado

□ **sqrt:** Raíz cuadrada

□ **sin:** Seno

□ **cos:** Coseno

□ **arctan:** Arcotangente

↓
Ángulo en radianes

□ **ln:** Logaritmo neperiano

□ **exp:** Función exponencial

↓
Base e

■ Semántica: Como operadores y funciones sobre enteros

□ Limitaciones:

■ Posibilidad de desbordamiento

■ Posibilidad de error de redondeo



Sobrecarga de funciones y operadores

- Se considera que una función o un operador están sobrecargados cuando se pueden utilizar con operandos de distinto tipo (tanto con enteros como con reales)

Operadores	+	}	$Z \times Z \rightarrow Z$
	-		$R \times R \rightarrow R$
	*		
Funciones	abs	}	$Z \rightarrow Z$
	sqr		$R \rightarrow R$



Conversión de tipos

- $\mathbb{Z} \subset \mathbb{R}$, por lo que $\mathbb{Z} \times \mathbb{R} \rightarrow \mathbb{R}$
- Conversión automática de \mathbb{Z} a \mathbb{R}
 - Ejemplo: $8 + 9.4 \rightarrow 17.4$
- Pero, la conversión de real a entero no es automática.
- Funciones de conversión de \mathbb{R} a \mathbb{Z}
 - **trunc(x)**: Devuelve la parte entera de x
 - **round(x)**: Redondea x al entero más próximo
 - Ejemplos:
 - $\text{trunc}(99.9) = 99$
 - $\text{round}(-5.8) = -6$
 - $-\text{round}(-99.9) = 100$



Tipo real: Expresiones

- Una expresión real permite la combinación de números, operaciones y funciones. El resultado de su evaluación será un real.
- Precedencias:

1.	()	↑ más alta más baja
2.	Funciones	
3.	* /	
4.	+ -	
- Asociatividad:
 - De izquierda a derecha



Expresiones reales

<i>Función</i>	<i>Expresión equivalente</i>	<i>Restricción</i>
x^y :	$\exp(y * \ln(x))$	Para $x > 0$
$\text{tg}(x)$	$\sin(x)/\cos(x)$	Para $x \neq \pi/2 + k\pi$
$\arcsen(x)$	$\arctan(x/\sqrt{1-\text{sqr}(x)})$	Para $0 < x < 1$
$\log_b(x)$	$\ln(x)/\ln(b)$	Para $b > 1, x > 0$
Número de cifras de un entero n	$\text{trunc}(\ln(n) / \ln(10)) + 1$	Para $n > 0$



2.2.3 El tipo carácter

- Nombre del tipo: `char`
- Dominio: `C` y se corresponde con el juego de caracteres disponibles en el ordenador
- Intervalo acotado
 - Juego de caracteres ASCII de 8 bits (hasta 256 caracteres)
- Representación: Un carácter escrito entre apóstrofos o comillas simples (`' '`), exceptuando el apóstrofo propiamente dicho, que se “libera” con otro apóstrofo: `' '`.
- Ejemplos: `'h'`, `'A'`, `'@'`, `' ' ' '`



Tipo carácter:

Operaciones y Funciones

- Operadores Binarios (+). Notación infija
 - Concatena caracteres.
 - Ej.: 'a' + 'b' → 'ab' [**ATENCIÓN. Ver strings**]
- Funciones monarias: Notación prefija
 - **pred**: $C \rightarrow C$ (carácter anterior en la tabla ASCII)
 - **succ**: $C \rightarrow C$ (carácter siguiente en la tabla ASCII)
- Funciones de conversión char / integer
 - **ord**: $C \rightarrow Z$ n^0 de orden (ASCII) del carácter
 - **chr**: $Z \rightarrow C$ carácter asociado a un n^0 de orden (ASCII)



Juego de caracteres

char	dec	char	dec	char	dec	char	dec
<NU>	0	<SPACE>	32	@	64	`	96
<SH>	1	!	33	A	65	a	97
<SX>	2	"	34	B	66	b	98
<EX>	3	#	35	C	67	c	99
<ET>	4	\$	36	D	68	d	100
<EQ>	5	%	37	E	69	e	101
<AK>	6	&	38	F	70	f	102
<BELL>	7	'	39	G	71	g	103
<BS>	8	(40	H	72	h	104
<HT>	9)	41	I	73	i	105
<LF>	10	*	42	J	74	j	106
<VT>	11	+	43	K	75	k	107
<FF>	12	,	44	L	76	l	108
<CR>	13	-	45	M	77	m	109
<SO>	14	.	46	N	78	n	110
<Sl>	15	/	47	O	79	o	111
<DL>	16	0	48	P	80	p	112
<D1>	17	1	49	Q	81	q	113
<D2>	18	2	50	R	82	r	114
<D3>	19	3	51	S	83	s	115
<D4>	20	4	52	T	84	t	116
<NK>	21	5	53	U	85	u	117
<SY>	22	6	54	V	86	v	118
<EB>	23	7	55	W	87	w	119
<CN>	24	8	56	X	88	x	120
	25	9	57	Y	89	y	121
<SB>	26	:	58	Z	90	z	122
<EC>	27	;	59	[91	{	123
<FS>	28	<	60	\	92		124
<GS>	29	=	61]	93	}	125
<RS>	30	>	62	^	94	~	126
<US>	31	?	63	_	95	À	127



Cadenas de caracteres

- Nombre del tipo: **string** (propio de Turbo Pascal)
- Dominio: Secuencia de caracteres escrita entre apóstrofes y que es capaz de contener espacios en blanco, eñes, acentos, etc.
 - ❑ 'Esto es una cadena de caracteres'
 - ❑ 'Esta secuencia ' 'lleva una comilla'



2.2.4 El tipo boolean

- Nombre del tipo: **boolean**
- Dominio: B (dos valores lógicos)
- Intervalo acotado
 - $\{\text{false}, \text{true}\}$
- Se emplea en expresiones lógicas. Fundamentalmente, en las tareas de control de bucles y selecciones



Tipo boolean: Operaciones

- Operadores lógicos

- Monarios. Notación prefija

- **not**: $B \rightarrow B$ negación lógica

- Binarios. Notación infija

- **and**: $B \times B \rightarrow B$ conjunción lógica

- **or**: $B \times B \rightarrow B$ disyunción lógica

Precedencia

↑ más alta

↓ más baja

- Tabla de verdad:

A	B	$A \text{ and } B$	$A \text{ or } B$	$\text{not } A$
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false



Tipo boolean: Operaciones

- Operadores relacionales
 - Binarios. *Notación infija*
 - = igual
 - <> distinto
 - < menor
 - <= menor o igual
 - > mayor
 - >= mayor o igual
 - Todos están *sobrecargados*, permitiendo que de la comparación de cualesquiera dos tipos básicos de valores el resultado sea un valor lógico
 - $C \times C \rightarrow B$ $R \times R \rightarrow B$
 - $B \times B \rightarrow B$ $Z \times Z \rightarrow B$
-



Operadores relacionales lógicos

- Los siguientes operadores al actuar sobre operandos de tipo booleano reciben nombres especiales:
 - $=$ Equivalencia lógica
 - $<>$ Or exclusivo lógico
 - $<=$ Implicación lógica

A	B	$A = B$	$A <> B$	$A <= B$
false	false	true	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	false	true



Tipo boolean: Funciones predefinidas

■ Funciones monarias. Notación prefija

- **Succ:** $B \rightarrow B$ $\text{Succ}(\text{false}) \rightarrow \text{true} \mid \mid \text{Succ}(\text{true}) \rightarrow \text{false}$
Valor siguiente
- **Pred:** $B \rightarrow B$ $\text{Pred}(\text{false}) \rightarrow \text{true} \mid \mid \text{Pred}(\text{true}) \rightarrow \text{false}$
Valor anterior
- **Ord:** $B \rightarrow \mathbb{Z}$ $\text{Ord}(\text{false}) \rightarrow 0 \mid \mid \text{Ord}(\text{true}) \rightarrow 1$
Número de orden
- Son funciones sobrecargadas
- **Odd:** Indica si un entero es impar

$$\text{Odd}(n) \begin{cases} \text{True si } n \text{ es impar} \\ \text{False en otro caso} \end{cases} \quad \text{Odd}(n) \Leftrightarrow n \bmod 2 = 1$$



Evaluación perezosa (ciclo corto)

- El compilador NO completa el proceso de evaluación de la expresión.
- Ejemplo:
 - Sean $\text{num}, \text{den} \in \mathbb{R}$, $\text{den} \neq 0$
 - Expresión: $(\text{den} \neq 0) \text{ and } (\text{num}/\text{den} = 0)$
 - Evaluación perezosa: Devolvería valor false (Turbo Pascal por defecto)
 - Evaluación completa (ciclo largo): Se produciría un error (Pascal estándar)



2.2.5 Expresiones

- **Sintaxis:** Una expresión está formada por constantes, variables, funciones aplicadas a una expresión u operaciones entre expresiones
- Las operaciones se aplican según la precedencia y asociatividad (izquierda → derecha)
- Las funciones se aplican a sus parámetros entre paréntesis



Precedencia de operadores

- Semántica: Evaluación de las expresiones

Operadores	Categoría	Nivel de Precedencia
()	Paréntesis	Máximo: 1
¬, NOT (monarios)	Negación	2
*, /, DIV, MOD, AND (binarios)	Multiplicativos	3
+, -, OR (binarios)	Aditivos	4
=, <, <=, >, >= (binarios)	Relacionales	Mínimo: 5

- A igual precedencia, se aplica la asociatividad de izquierda a derecha



Tipo de una expresión

- Cada expresión tiene un tipo
 - Depende del tipo de los operadores, funciones, etc., que componen la expresión
 - Si el argumento de una función o de un operador es una expresión, entonces el tipo de la expresión ha de coincidir con el tipo requerido para el argumento
(Excepción: conversión automática de \mathbb{Z} a \mathbb{R})
 - El compilador comprueba la corrección sintáctica de las expresiones y la consistencia de tipos
-



Tipos ordinales

- Son los tipos **integer**, **char** y **boolean**
- Se pueden enumerar sus dominios asignando a sus elementos un número de posición
- **ord()** es la función que devuelve el número de posición de su argumento en su dominio
- Las funciones **pred()** y **succ()** son exclusivas de los tipos ordinales



2.3 Elementos básicos del lenguaje



2.3. Elementos básicos del lenguaje

- Vocabulario
- Constantes y variables
- Instrucciones básicas
- Estructura de programas



2.3.1 Vocabulario

■ **Palabras reservadas:**

- Aquellas que tienen un significado predefinido en el lenguaje de programación

- Pascal estándar:

and, array, begin, case, const, div, do, downto, else, end, file, for, forward, function, goto, if, in, label, mod, nil, not, of, or, packed, procedure, program, record, repeat, set, then, to, type, until, var, while, with

- Turbo Pascal:

implementation, interface, string, unit, uses



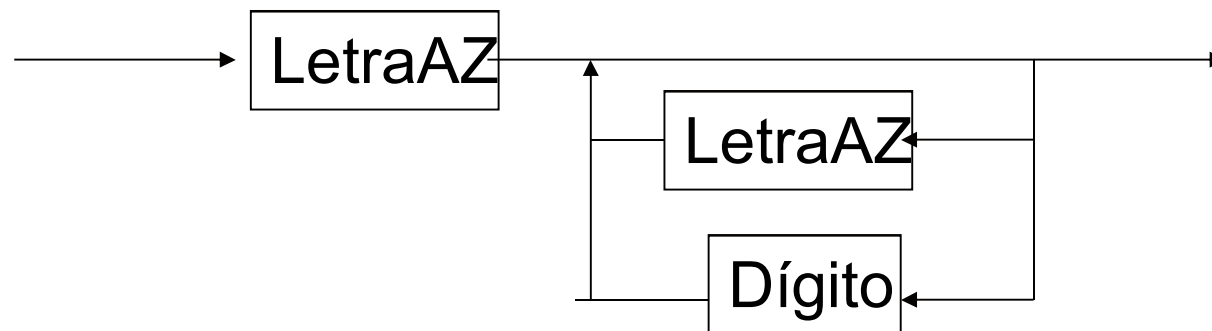
2.3.1 Vocabulario

- **Identificadores:** Nombres asociados a diferentes elementos de un lenguaje (dispositivos, tipos, constantes, variables)
- Identificadores predefinidos:
 - Archivos estándar: **input, output**
 - Constantes: **FALSE, TRUE, MAXINT, PI**
 - Tipos: **boolean, char, integer, real, text**
 - Funciones: **abs, arctan, chr, cos, eof, eoln, ln, odd, pred, round, sin, sqr, sqrt, succ, trunc**
 - Procedimientos: **dispose, get, new, pack, page, put, read, readln, reset, rewrite, unpack, write, writeln**

2.3.1 Vocabulario

■ **Identificadores definidos por el programador:**

- Representan diversos elementos que el programador va creando según sus necesidades
- Empiezan con una letra, seguida por letras y cifras





2.3.1 Vocabulario

- Reglas adicionales de identificadores definidos por el programador:
 - ❑ NO se permiten caracteres especiales (excepto “_” en Turbo Pascal)
 - ❑ No se permite letras “especiales” (como, por ejemplo: ñ, é, ú, ó, á, í)
 - ❑ No se distinguen letras mayúsculas y minúsculas
 - ❑ La longitud máxima para el identificador es de 127 caracteres, de ellos solo son significativos los primeros 63



2.3.1 Vocabulario

- Símbolos especiales:

+ - * / := . , ; = < > <= >= <> () []
(* *) {} (. .) _ ...

- Literales:

'Los 256 caracteres ASCII', false, 3.1654,
'd', -45



2.3.1 Vocabulario

■ **Comentarios:**

- ❑ Texto intercalado en el programa fuente con objeto de aclarar su contenido
 - ❑ Son ignorados por el compilador, por lo que no generan código ejecutable
 - ❑ Se escriben entre los caracteres:
(* *) o { }
 - ❑ No se pueden anidar, no puede haber un comentario dentro de otro
-



2.3.2 Constantes y variables

- **Constante:** Elemento que no cambia su valor a lo largo de la ejecución de un programa:
 - **Anónimas:** 5, 3.14, 'a' (son literales)
 - **Predefinidas:** PI, MAXINT, TRUE , FALSE
 - **Definidas por el programador** con nombre
 - **Variable:** Elemento que puede cambiar su valor a lo largo de la ejecución del programa. Las define el programador
-

2.3.2 Constantes y variables

- Las constantes y variables tienen:
 - Un **identificador** por el que se nombran
 - Un **tipo** que determina su dominio y las operaciones permitidas
 - Un **valor** que inicialmente es desconocido a no ser que se trate de una constante



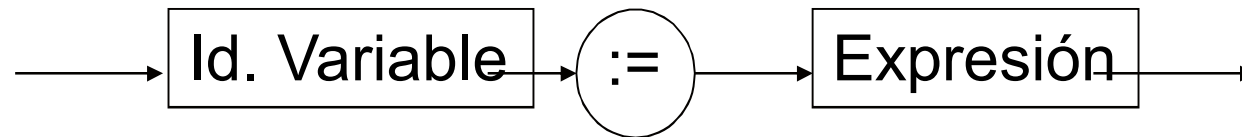
2.3.3 Instrucciones básicas

- **Instrucciones:** Representan *acciones* del programa
- Instrucciones básicas:
 - **Asignación:** Cambia el valor de una variable
 - **Escritura:** Instrucción de salida (de datos)
 - **Lectura:** Instrucciones de entrada (de datos)



Asignación

- Operador **`:=`**
- Sintaxis:



- Ejemplos
 - `base := 10.0`
 - `altura := 20.0`
 - `area := base * altura / 2`
 - `contador := contador + 1`
 - `x := (-b + sqrt(sqr(b) - 4 * a * c)) / (2 * a)`



Asignación

- Semántica:
 - ❑ Se *evalúa* la expresión de *la parte derecha* y
 - ❑ Se *almacena* el resultado en la *variable* de la parte izquierda
- Asignación ($:=$) y igualdad ($=$)
 - ❑ La asignación es una instrucción que *asigna un valor* a una variable
 - ❑ La igualdad es un *operador relacional* que se usa dentro de expresiones



Asignación

- En Pascal, una variable presenta un valor indefinido al iniciarse el programa (valor “basura”)
- En los programas habrá que tenerlo en cuenta y valorar si es necesario asignar a las variables un valor inicial



Escritura: *archivo output*

- La salida de datos podrá ser por pantalla, por impresora o enviarse a un dispositivo de almacenamiento
 - Se asume un dispositivo de salida estándar o por defecto (*archivo output*) que suele ser el monitor o pantalla
-



Escritura: archivo *output*

- Procedimientos de salida:

write (Nombre de fichero, expr1, expr2, ...
exprn)

writeln (Nombre de fichero, expr1, expr2,
...exprn)

- Si se omite **Nombre de fichero**, la salida es por el dispositivo estándar



Escritura: archivo *output*

- Semántica:
 - **write**: Evalúa sus argumentos y escribe el resultado en el **Nombre de fichero** o en la salida estándar, en caso de que no se especifique nada
 - **writeln**: Funciona como **write** + salto de línea



Escritura: salida formateada

- **Enteros: write** (expresionEntera :m, ...)
 - Escribe un número entero indicando el espacio en el que se justificará dado por el valor de *m*
 - **writeln** (12:5, 23:5, 2541:5, 123:5)
 - La justificación se hará por la derecha
 - **writeln** (12:5,123:4) \succ bbb12b123
 - La salida puede rebasar el espacio reservado por *m*
 - **writeln** (12332:2) \succ 12332



Escritura: salida formateada

- **Reales:** La salida por defecto es en notación exponencial
- **writeln** (expresionReal :m, ...)
 - Justificación a la derecha en el espacio dado por el valor *m*
 - `x:= 6.12849000000E+2`
 - **writeln** (X:25) bbbbbb6.1284900000E+02
- **writeln** (expresionReal :m :n, ...)
 - *m* igual que en el ejemplo anterior y *n* es el número de decimales a representar
 - Sirve para convertir notación científica en coma fija redondeando
 - **writeln** (X:10:2) bbbb612.85



Escritura: salida formateada

- **Caracteres** y cadenas de caracteres:

write (caracteres :*m*, ...)

- Justificación a la derecha en el espacio dado por el valor *m*

- **writeln** ('3':4) ⤴ bbb3

- **writeln** ('Hola':7) ⤴ bbbHola

- **Booleanos: write** (booleanos :*m*, ...)

- Justificación a la derecha en el espacio dado por el valor *m*

- **writeln** (true:6) ⤴ bbtrue



Lectura: archivo *input*

- La entrada de datos puede realizarse ser por medio del teclado o de un dispositivo de almacenamiento
 - Se asume un dispositivo de entrada estándar o por defecto (archivo *input*) que suele ser el teclado
-



Lectura: archivo *input*

- Procedimientos de entrada:

read (Nombre de fichero, var1, var2,..., varn)

readln(Nombre de fichero, var1, var2,..., varn)

- Si se omite **Nombre de fichero**, la entrada es por el dispositivo estándar



Lectura: archivo *input*

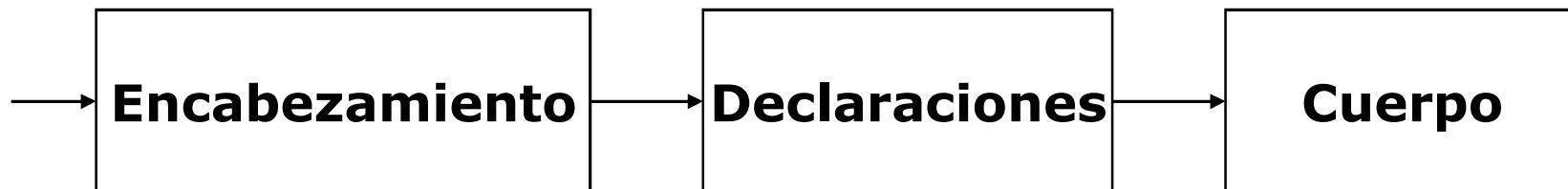
■ Semántica:

- ❑ **read**: El ordenador lee los valores de **Nombre de fichero** o de la entrada estándar, si no se especifica, y los asigna a las variables expresadas como argumentos
- ❑ **readln**: Una vez leídos sus argumentos ignora el resto de valores, si los hubiera, y posiciona el puntero en la siguiente línea. **readln()**, al carecer de argumentos, permanece a la espera de que se le introduzca un salto de línea (con la tecla **Intro**)



2.3.4 Estructura de programas

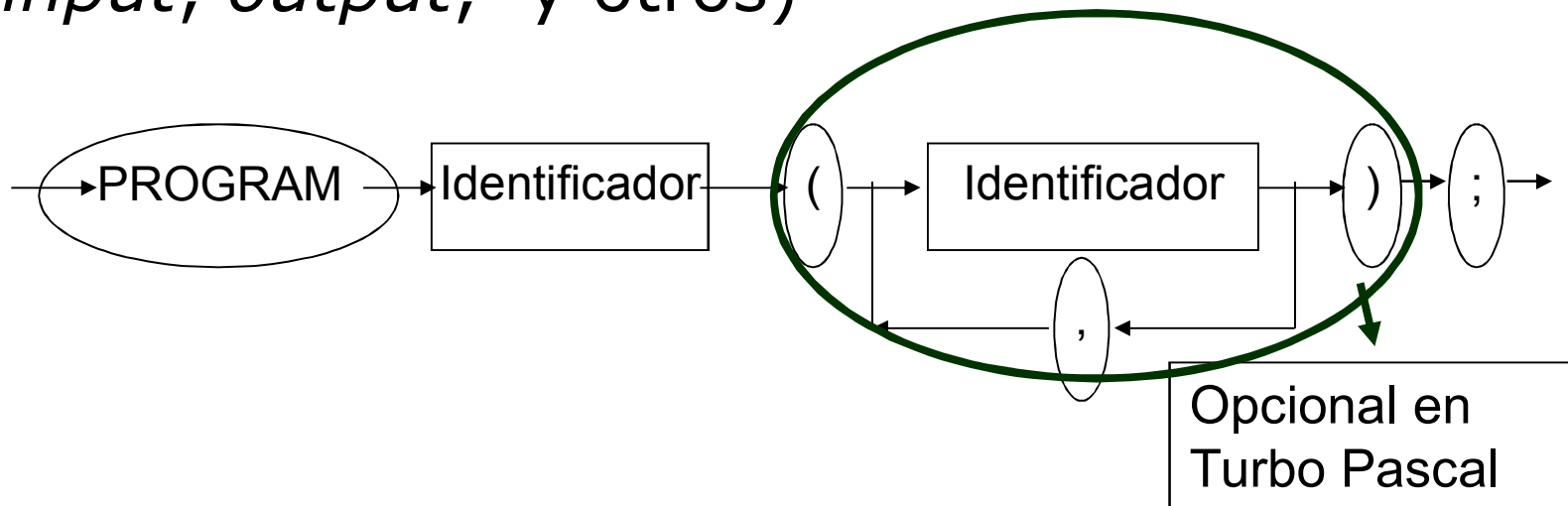
- Partes de un programa:
 - ❑ **Encabezamiento**
 - ❑ **Definiciones y declaraciones**
 - ❑ **Cuerpo del programa**





Encabezamiento

- Establece:
 - ❑ La **identificación** del programa
 - ❑ Los **elementos externos** con los que intercambia información (archivos estándar *input*, *output*, y otros)





Declaraciones y definiciones

- Se deben declarar los objetos no predefinidos en el lenguaje que se van a utilizar
- Tipos de objetos:
 - constantes - procedimientos
 - variables - funciones
 - unidades - ...
- Esto permite al compilador
 - **Reservar** espacio de memoria para cada identificador
 - **Verificar** el correcto uso de los constructores
- El equivalente en notación matemática a una declaración sería: sean $n \in \mathbb{Z}$, $x \in \mathbb{R}$, $\pi=3.14$



Definición de unidades

- Permite utilizar desde un programa otros módulos ya compilados (no disponible en Pascal estándar)
- Palabra reservada: **USES**
- Formato:

Ventaja: **Reutilización**
de código

USES

identificadorUnidad;



Definición de constantes

Ventaja: **Facilita** el mantenimiento

- Palabra reservada: **CONST**
- Formato:

CONST

identificadorConstante = valor;



Declaración de variables

- Palabra reservada: **VAR**
- Formato:

```
VAR  
    identificadorVariable: tipo;
```

- El **tipo** de una variable permanece **inalterable**.
- El **valor no se conoce a priori** y cambia a lo largo de la ejecución del programa.
- El compilador **reserva** espacio de **memoria** y **verifica** según sea el tipo



Cuerpo del programa

- Contiene las instrucciones ejecutables del programa
- Comienza con la palabra reservada **BEGIN** y finaliza con la palabra reservada **END** seguida de punto . (**END.**)
- Las instrucciones se separan con el símbolo “;”

BEGIN

{las instrucciones ejecutables}

END.



Resumen

- El **encabezamiento** del programa es **obligatorio**
- La **declaración** de un objeto es **obligatoria** si este **se utiliza** en el programa
- El **cuerpo** del programa es **obligatorio**
- Principales instrucciones (hasta el momento):
 - **asignación** (`:=`)
 - **entrada** (`write/writeln`)
 - **salida** (`read/readln`)



2.4 La documentación del programa



2.4. La documentación del programa

- Aspectos relevantes:
 - ❑ Comentarios
 - ❑ Estructuración del código fuente
 - ❑ Elección de identificadores
 - ❑ . . .



Comentarios

- El programa ha de estar documentado con comentarios `{ }` o `(* *)`
 - ❑ Descripciones de funcionamiento: De subprogramas, bloques, decisiones, etc.
 - ❑ Precondiciones: Condiciones de entrada
 - ❑ Postcondiciones: Condiciones de salida
 - ❑ Invariantes: Aserciones de ejecución



Estructuración del código fuente

- Claridad del formato de escritura del código fuente:
 - ❑ Uso de retornos de línea: Solo una instrucción por línea
 - ❑ Uso de tabuladores: Reflejar la estructuración del programa
 - ❑ Longitud de una línea: No debe exceder la longitud de la pantalla
-



Estructuración del código fuente

PROGRAM NombrePrograma(input, output);

{ Propósito: }

{ Entrada: }

{ Salida: }

CONST {Definición de constantes}

CONSTANTES = #VALOR; {}

VAR {Declaración de variables}

variables : tipo; {}

BEGIN {Programa principal}

Instrucción1;

Instrucción2;

...;

InstrucciónN

END. {Programa principal}

Elección de identificadores

- Criterio para la **semántica** de identificadores:
 - Debe describir el **objetivo** del identificador
 - De hecho, siempre que haya dudas de la utilidad o necesidad de un elemento, se añadirá un comentario descriptivo al final de la línea donde se introduzca o declare
- Criterio para la **sintaxis** de identificadores
 - Equilibrio en la longitud del identificador



Elección de identificadores

- Elección de un **criterio de tipografía**:
 - ❑ **Constantes**: En **mayúsculas**
(MAXINT, PI)
 - ❑ **Variables**: Comienzan con **minúscula**
(i, opcion)
 - ❑ **Palabras clave**: En **mayúsculas**
(BEGIN, END)
 - ❑ **Tipos**: Comienzan con **T**



2.5 Aspectos formales



2.5 Aspectos formales

- Aspectos formales de programas:
 - Complejidad:
 - ¿Cuánta cantidad de recursos necesita el programa?
 - Corrección:
 - ¿El programa resuelve el problema especificado de forma satisfactoria?



Complejidad

- Tipos de recursos usados:
 - Complejidad en tiempo:
 - ¿Cuánto tiempo tarde el programa en ejecutarse?
 - Complejidad en espacio:
 - ¿Cuánto espacio de memoria requieren sus datos al ejecutarse?
 - Eficiencia:
 - Se dice que un programa es más eficiente (o de menor complejidad) que otro, cuando utiliza menos recursos que ese otro.
-



Complejidad

- Cómo medir la complejidad:
 - En tiempo:
 - Tiempo total necesario para la ejecución del programa
 - **Número de “operaciones elementales”**
 - Contar operaciones por “clases de instrucciones” (comparaciones, asignaciones, etc.)
 - En espacio:
 - Número de bytes necesario para los datos del programa
 - **Número de variables de tipos elementales**
 - Contar valores por “clases de variables”
-



Complejidad en tiempo

- Operaciones elementales:
 - ❑ Operadores predefinidos (aritméticos, relacionales, ...)
 - ❑ Funciones predefinidas
 - ❑ Asignación
 - ❑ Instrucciones primitivas (write, read, ..., dependiendo del número de argumentos)
- Recuento:
 - ❑ Contar todas las operaciones elementales de una instrucción
 - ❑ Ejemplo:
 - `writeln(ord(sqr(5 mod 2)>7)+1)`
 - Complejidad: 6 operaciones elementales



Ejemplo de complejidad

```
PROGRAM AreaCirculo (input, output):  
  { Propósito: hallar el área de un círculo dado su radio}  
  { Entrada:  el radio del círculo}  
  { Salida:   el área del círculo}  
  
  CONST                                {Definición de constantes}  
    pi = 3.14;                          {pi es una constante}  
  
  VAR                                {Declaración de variables}  
    radio, area: integer; {radio y area son variables}  
  
  BEGIN                                {Inicio del Cuerpo del  
programa}  
    write('¿Cuál es el radio?:  ');  
    readln(radius);  
    area := pi * sqr(radius);  
    writeln('Área = ', area)  
  
  END.                                {Fin del cuerpo del programa}
```

- Complejidad en tiempo: 7
- Complejidad en espacio: 3



Corrección

- Aspectos de la corrección de programas que hay que considerar:
 - Sintaxis
 - Semántica
 - Pragmática
 - Corrección sintáctica:
 - ¿El programa se ajusta a los reglas de sintaxis?
 - Aspectos: Puntuación, corrección en los tipos
-



Corrección

- Corrección semántica:
 - ¿El programa produce los resultados esperados para cualquier entrada permitida?
 - Aspectos: Errores en la lógica del programa, desbordamiento, errores de redondeo, variables sin iniciar
 - Corrección pragmática:
 - ¿El programa se ajusta a los reglas de "buen estilo"?
-



Corrección semántica

- Depuración:
 - Localización sistemática de errores
- Especificación Pre/Post
 - Método para especificar la semántica intencionada de un programa
- El sistema formal de Hoare
 - Método para probar la corrección semántica de un programa

Depuración

Qué hacer cuando hay un error

- ❑ Prueba y error:



Modificar el programa aleatoriamente hasta que funcione (o uno se desespere)

- ❑ Trazado:



Examinar la sucesión de estados de cómputo (para entradas concretas) para delimitar el error



Ejemplo: Intercambio de variables

	posicion	input	output	x	y
		[3 4]	[]	?	?
readln(x,y)	1	[]	[]	3	4
x := x+y;	2	[]	[]	7	4
y:= x-y;	3	[]	[]	7	3
x := x-y;	4	[]	[]	4	3
writeln(x, ' ', y)	5	[]	[4 3]	4	3



Depuradores

- Problemas de la depuración manual

- Muchos estados de cómputo
- Estados de cómputo muy grandes

- Depurador

Herramienta que apoya al programador en la tarea de depurar el programa



Depuradores

- Permiten seleccionar estados de cómputo
 - ❑ Instrucción por instrucción
 - ❑ Bloques de instrucciones (“puntos de ruptura”)
 - ❑ Hasta que se cumpla una condición
 - Permiten centrarse en parte del estado de cómputo
 - ❑ Valor de ciertas variables
 - ❑ Valor de expresiones
-



El depurador de Turbo Pascal

- Examinar parte del estado de cómputo actual
 - Menú *Debug*
 - Opción *Watch*
 - Insertar variable o expresión a inspeccionar
- Pasar al siguiente estado de cómputo
 - Menú *Run*
 - Opción *Trace into* o *Step over*



3.1. Tipos definidos por el programador: subrangos



Objetivos

- Saber definir tipos distintos de los tipos básicos
- Conocer los tipos de datos simples definidos por el programador y las condiciones de su aplicación



Tipos definidos por el programador

- **Programa** = acciones + datos

Estructuración

Instrucciones estructuradas

- Subprogramación

¿ ?



Tipos definidos por el programador

■ **Objetivo:**

- Modelar la estructura de los objetos del mundo real mediante un tipo de dato estructurado:
 - Color de un semáforo: (rojo, amarillo, verde)
 - Día del mes: 1..31
 - Conjunto de vocales: ['a', 'e', 'i', 'o', 'u']
 - Vector en el espacio: $\mathbb{R} \times \mathbb{R} \times \mathbb{R}$
 - Ficha de una persona: <Nombre, Apellidos, DNI>
 - Texto: secuencia de líneas



Tipos definidos por el programador

- Aspectos a tener en cuenta:
 - Dominio del tipo de datos
 - Operaciones permitidas con los valores del tipo
-



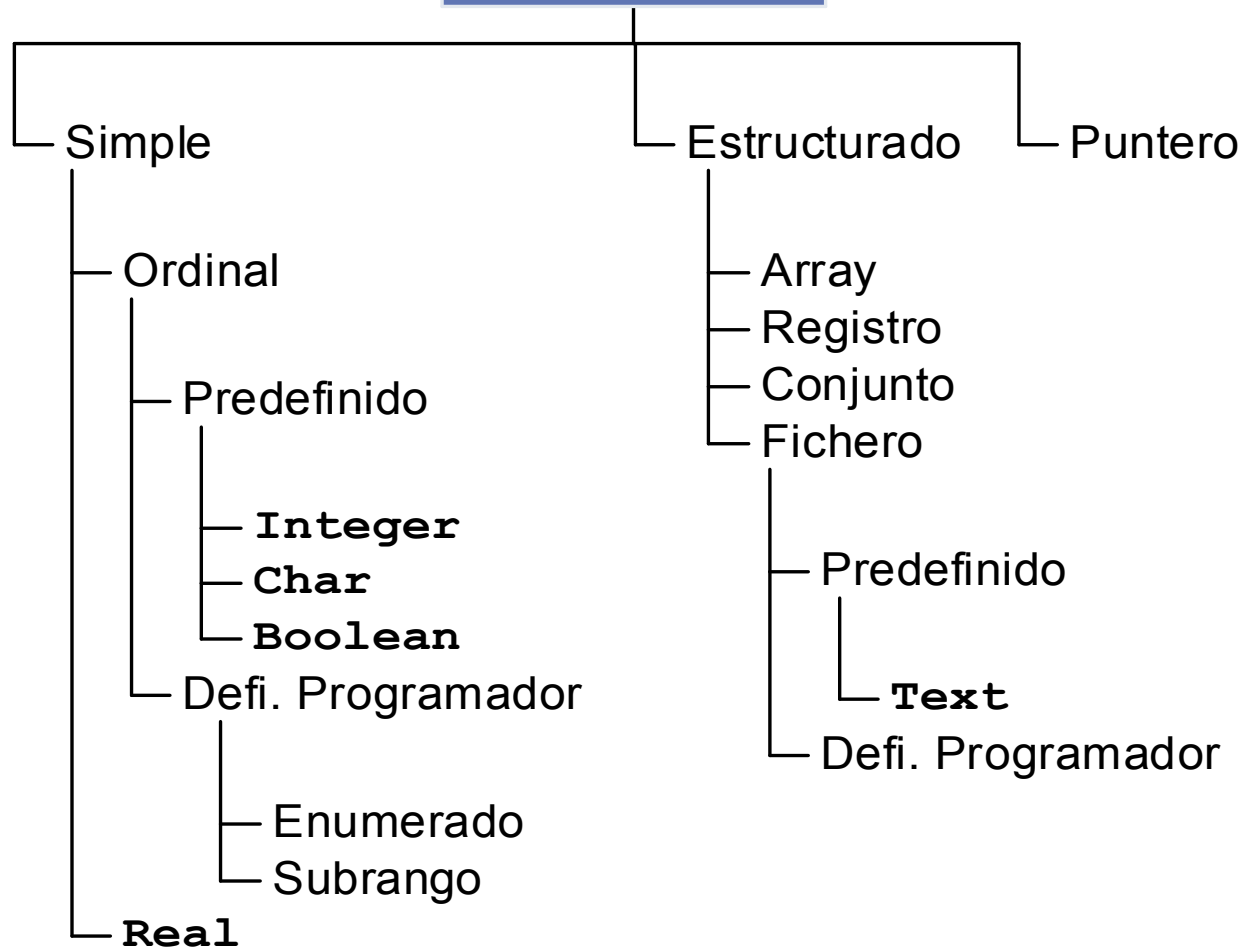
Tipos definidos por el programador

- Un tipo de datos se define por medio de:
 - Un **conjunto** de posibles **valores** para el tipo
 - Un **conjunto** de **operaciones** sobre ese tipo
 - Un **conjunto** de **relaciones** entre elementos de ese tipo
 - Ejemplo: El tipo integer:
 - Valores: -32768, ..., -3, -2, -1, 0, 1, 2, 3, ..., 32767
 - Operaciones: +, -, *, div, mod
 - Relaciones: <, >, ...



Tipos de datos en Pascal: resumen

Tipos de Datos





Tipos definidos por el programador

- **Tipos simples:** Sus valores son atómicos, indivisibles
 - **Tipo ordinal:** Si todo valor tiene predecesor y sucesor excepto el primero y el último
 - **Tipos estructurados:** Sus valores no son atómicos, se pueden descomponer
-



Tipos definidos por el programador

- **Tipo predefinido:** Si su nombre, valores, operaciones y relaciones vienen fijados en el lenguaje.
 - **Tipo definido por el usuario:** Este puede especificar valores, nombres y operaciones según el constructor empleado.
-



Tipos definidos por el programador

- **Tipos simples:**

- Ampliación de los tipos simples predefinidos

- **Subrango**

- **Tipos estructurados (compuestos) :**

- Engloban a varios datos simultáneamente:

- **Array (String):** Tema 5

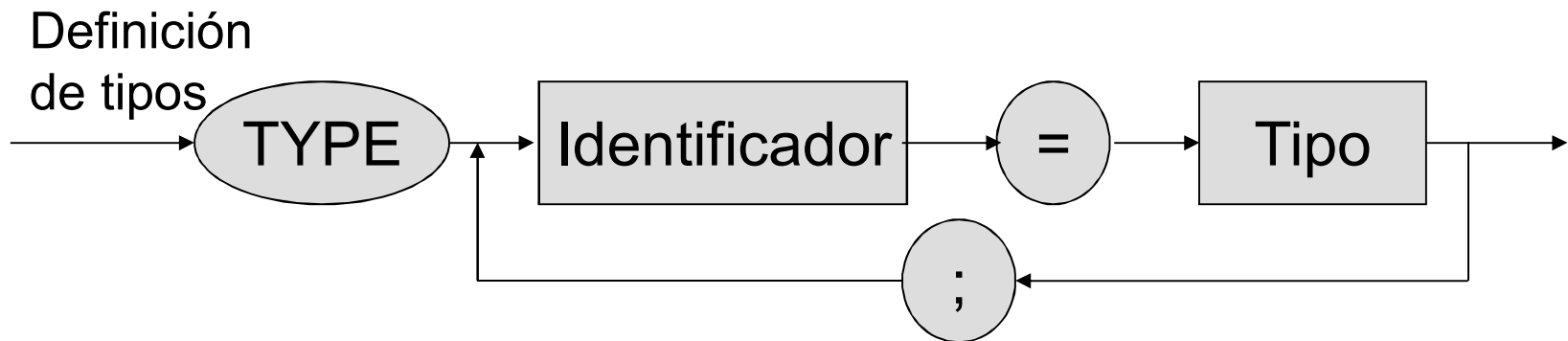
- **Registro:** Tema 6

- **Fichero:** Tema 6

3.1.1 Definición de tipos

■ Sintaxis: TYPE

IdTipo = Tipo;



■ Semántica:

- El identificador puede usarse como nombre de tipo en la declaración de variables del programa



Ejemplo 1.1

PROGRAM

```
EjemploDeTipos (input, output) ;
```

TYPE

```
TEntero = integer;
```

```
Tletra = char;
```

```
TEnteroLargo = longint;
```

```
{TurboPascal}
```

VAR

```
i, j: TEntero;
```

```
c: Tletra;
```

```
l: TEnteroLargo;
```

```
. . .
```



Definición de tipos: esquema

PROGRAM

CONST

...

TYPE

...

VAR

...

PROCEDURE

FUNCTION

BEGIN {programa principal}


...

END. {programa principal}



Definición de tipos: normas de estilo

- Elección de identificadores para tipos:
 - Nombres con significado (TEntero en vez de Txyz)
 - Criterio de tipografía: empezar con T (TEntero en vez de Entero)
 - ¡**No intentar redefinir palabras reservadas!**
- Ejemplo:

TYPE while = integer 



3.2. El tipo subrango

- **Objetivo:** Modelar objetos
 - Cuyos posibles estados son solo un **subrango** del dominio de otro tipo
 - Ejemplo: número de los días del mes (subrango de *integer*)
- **Definición** de tipo subrango
 - Tipo cuyos valores están restringidos a un intervalo cerrado predefinido de un tipo ordinal.
 - El tipo ordinal en el que construimos un tipo subrango se llama **tipo base** o **tipo anfitrión**.

Tipo subrango

■ Sintaxis:



```
TYPE TMesesAnyo = 1..12;  
    TMayusculas = 'A'..'Z';  
    ...  
    TDiasLaborables = Lun..Vie;
```

■ Semántica:

- Los posibles valores de una variable del tipo quedan restringidos al subrango especificado.



Tipo subrango

■ Limitaciones:

- ❑ Las constantes han de ser de un **tipo ordinal**

TYPE TTerapia1 = 1.0..12.0



- ❑ Las constantes han de ser del **mismo tipo**

TYPE TTerapia2 = 1..'a'



- ❑ Las constantes han de estar en **orden creciente**

TYPE TTerapia3 = 10..-10





Ejemplo 1.2

```
PROGRAM EjemploTipoSubrango(input, output);  
TYPE {Definición del nuevo tipo}  
    TMesesAnyo = 1..12;  
VAR   {Declaración de variables del nuevo tipo}  
    mes1,mes2: TMesesAnyo;  
    ...  
BEGIN {Programa principal}  
    mes1:= 2; {Uso de las variables}  
    mes2:= 5;  
    writeln (mes1 + mes2);  
    ...  
END. {Programa principal}
```



Tipo subrango: operaciones

■ Operaciones:

- ❑ Se pueden aplicar **todas** las operaciones y funciones del **tipo base** o **anfitrión**.

■ Posibilidad de desbordamiento:

- ❑ Existe la posibilidad de que el resultado de la operación se salga del rango permitido

`mes1 := mes2 + 12`

`readln (mes1)` {si la entrada es por ejemplo 0}

- ❑ En TurboPascal solo se produce un error de ejecución, si la opción “range checking” del compilador está activada



Tipo subrango

■ **Ventajas:**

□ Autodocumentación:

- Se indican explícitamente el intervalo de valores que tiene sentido en una variable
- Se expresan (parte de) las precondiciones y postcondiciones de un subprograma en el propio código fuente

□ Depuración:

- Se facilita la detección de errores de desbordamiento (con tal de que la opción de control de rango esté activada)



ascension.lovillo@urjc.es