RESOLUCIÓN DE PROBLEMAS Y ALGORITMOS

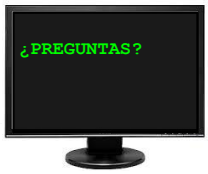
CLASE 3

Programación en Pascal.
Tipos de datos. Expresiones.

Luciano H. Tamargo
http://cs.uns.edu.ar/~lt
Depto. de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur, Bahía Blanca
2016

CONCEPTOS DE LA CLASE PASADA

1. Algoritmo.
   Primitiva.
   Traza.
2. Lenguaje de programación.
   Pascal:
   - Identificadores
   - Constantes y variables
   - Tipos de datos.
   - Primitiva de asignación (:=)
   - Primitivas read y write



TEMARIO

- Introducción.
- Diagrama sintáctico.
- Asignación.
- Código fuente.
- Tipos de datos.

OBJETIVOS DE LA MATERIA RPA

- El objetivo principal de RPA es que los alumnos adquieran la capacidad de desarrollar programas de computadoras para resolver problemas de pequeña escala.
- El desarrollo de un programa se concibe como un proceso que abarca varias etapas:
  1. La interpretación adecuada del enunciado a través del cual se plantea el problema.
  2. El diseño de un algoritmo que especifica la resolución del problema.
  3. La implementación del algoritmo en un lenguaje de programación imperativo.
  4. La verificación de la solución.

ETAPAS

- El objetivo principal de RPA es que los alumnos adquieran la capacidad de desarrollar programas de computadoras para resolver problemas de pequeña escala.
- El desarrollo de un programa se concibe como un proceso que abarca varias etapas:
  1. La interpretación adecuada del enunciado a través del cual se plantea el problema.
  2. El diseño de un algoritmo que especifica la resolución del problema.
  3. La implementación del algoritmo en un lenguaje de programación imperativo.
  4. La verificación de la solución.

Estas etapas están en sintonía con el proceso de ingeniería de software que veremos más adelante

CONCEPTO: LENGUAJE DE PROGRAMACIÓN

- Un lenguaje de programación es un lenguaje artificial creado para expresar procesos que pueden ser llevados a cabo por computadoras.
  - Ejemplos: Pascal, C, C++, Java, Prolog, Lisp.
- Un lenguaje de programación está definido por:
  1. un conjunto de símbolos,
  2. reglas sintácticas que definen su estructura, y
  3. reglas semánticas que definen el significado de sus elementos.

CONCEPTOS: DIAGRAMA SINTÁCTICO

- La sintaxis de un lenguaje de programación es un conjunto de reglas que indica la estructura de los programas en ese lenguaje.
- Un **diagrama sintáctico** (syntax diagram or railroad diagrams) es una forma gráfica de representar la sintaxis de un lenguaje de programación.
- Permite **describir sin ambigüedad** la sintaxis de un lenguaje de una manera simple y formal.
- Los diagramas sintácticos de Pascal que usaremos en RPA se encuentran en la página de la materia siguiendo este enlace: http://cs.uns.edu.ar/~lt/rpa/downloads/Practicos/Diagramas_Sintacticos_Pascal.pdf
- La sintaxis original escrita por N. Wirth está en la pág. 47 de : http://e-collection.library.ethz.ch/eserv/eth:3059/eth-3059-01.pdf

TEMARIO

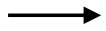
- Introducción.
- Diagrama sintáctico.
- Asignación.
- Código fuente.
- Tipos de datos.

ELEMENTOS DE UN DIAGRAMA SINTÁCTICO

(1) **Un nombre y flecha** indican el comienzo de un diagrama para la definición de nombre.

(2) **Las figuras "redondeadas"** indican que texto se debe incluir tal cual como aparece.

(3) Los **rectángulos** indican que nombre **está definido en algún otro diagrama** sintáctico.



(4) Las **flechas** indican el **orden** de lectura en el diagrama.

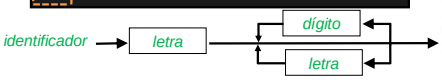
Todos los programas en Pascal tienen esta estructura sintáctica:

DIAGRAMAS SINTÁCTICOS

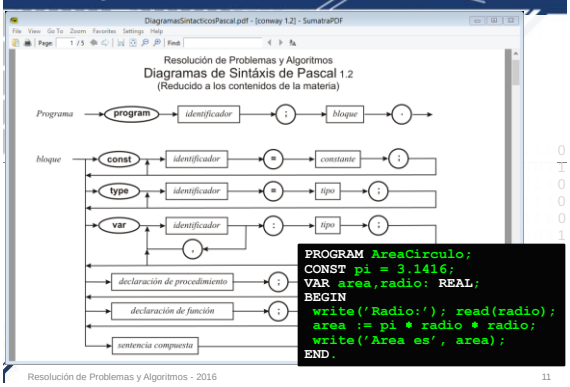
Todos los programas en Pascal tienen esta estructura sintáctica:

Ejemplo:

```
PROGRAM AreaCirculo;
CONST pi = 3.1416;
VAR area,radio: REAL;
BEGIN
     write ('Ingrese Radio:');
     read(radio);
     area := pi * radio * radio;
     write("Area es", area);
END.
```



PARTE DEL ARCHIVO DISPONIBLE EN LA PÁGINA DE RPA



TEMARIO

- Introducción.
- Diagrama sintáctico.
- Asignación.
- Código fuente.
- Tipos de datos.

METODOLOGÍA GENERAL PROPUESTA

PROBLEMA
↓
SOLUCIÓN
↓
ALGORITMO
↓
verificación
↓
PROGRAMA
↓
verificación

- **Observación importante**: los diagramas sintácticos nos ayudan a entender la sintaxis del lenguaje y poder identificar *errores sintácticos*.
- Un programa puede ser sintácticamente válido (cumple con todas las reglas de sintaxis) pero igualmente tener errores que no son sintácticos.
- Para poder identificar errores en la lógica del programa se puede utilizar una **traza**.

PRIMITIVA DE ASIGNACIÓN

- En una asignación: variable := expresión
  1) **primero** se evalúa la expresión de la derecha y se obtiene un valor,
  2) **luego** se modifica el valor de la variable, perdiéndose el valor anterior.

```
PROGRAM Asigna2;
CONST c = 2.3;
VAR a, b: REAL;
BEGIN
  b:= 10;
  a:= 1;
  a:= a + 1;
  a:= a + 1;
  a:= a + 1;
END.
```

Traza de los valores almacenados en memoria para cada variable:

a	b
?	?
?	10
1	10
2	10
3	10
4	10

"?" indica "sin valor"

PRIMITIVA DE ASIGNACIÓN

- En una asignación: variable := expresión
  1) **primero** se evalúa la expresión de la derecha y se obtiene un valor,
  2) **luego** se modifica el valor de la variable, perdiéndose el valor anterior.

```
PROGRAM Asigna3;
VAR a: REAL;
BEGIN
  a:= 1;
  a:= a + a;
  a:= a + a;
  a:= a + a;
  a:= a + a;
END.
```

Traza de los valores almacenados en memoria para cada variable:

a
?
1
2
4
8
16

PRIMITIVA DE ASIGNACIÓN

- En una asignación: variable := expresión
  1) **primero** se evalúa la expresión de la derecha y se obtiene un valor,
  2) **luego** se modifica el valor de la variable, perdiéndose el valor anterior.

```
PROGRAM AsignaMAL;
CONST c = 2.3;
VAR a, b: REAL;
BEGIN
  a:= b;
END.
```

MAL

b no tiene valor

Traza de los valores almacenados en memoria para cada variable:

a	b
?	?
?	?

Observe que el programa **AsignaMAL** es sintácticamente valido y aún así tiene un error.

PRIMITIVA DE ASIGNACIÓN

- En una asignación: variable := expresión
  1) **primero** se evalúa la expresión de la derecha y se obtiene un valor,
  2) **luego** se modifica el valor de la variable, perdiéndose el valor anterior.
- Problema: Intercambiar los valores de dos variables **a** y **b** de forma tal que el valor de **a** quede en **b** y el de **b** quede en **a**.

MAL

```
PROGRAM IntercambiaMAL;
VAR a, b: integer;
BEGIN
  write('Ingrese 2 enteros');
  read(a,b);
  a:= b;
  b:= a;
  writeln(a,' ',b);
END.
```

¿Qué casos de prueba usaría?

Observe que **IntercambiaMAL** es sintácticamente valido y aún así tiene un error.

INTERCAMBIAR LOS VALORES DE LAS VARIABLES

- En una asignación: variable := expresión
  1) **primero** se evalúa la expresión de la derecha y se obtiene un valor,
  2) **luego** se modifica el valor de la variable, perdiéndose el valor anterior.
- Problema: Intercambiar los valores de dos variables **a** y **b** de forma tal que el valor de **a** quede en **b** y el de **b** quede en **a**.

BIEN

```
PROGRAM IntercambiaBIEN;
VAR a, b, aux: integer;
BEGIN
  write('Ingrese 2 enteros');
  read(a,b);
  aux:= a;
  a:= b;
  b:= aux;
  write(a,' ', b);
END.
```

Preservo el valor de **a** en **aux**.

¿Qué casos de prueba usaría?

PRIMITIVA DE ASIGNACIÓN

- El **tipo** del resultado de la **expresión** tiene que ser compatible con el tipo de la variable que se quiere modificar (esto se explicará en mayor detalle muy pronto).

```
PROGRAM AsignaMAL;
VAR n,p: INTEGER;
BEGIN
  n:= 5;
  p:= n/2;
END.
```

MAL

n/2 da un resultado REAL y p es de tipo INTEGER

Traza de valores en memoria

n	p
?	?
5	?
5	

PRIMITIVA DE ASIGNACIÓN

- El **tipo** del resultado de la **expresión** tiene que ser compatible con el tipo de la variable que se quiere modificar (esto se explicará en mayor detalle muy pronto).

```
PROGRAM AsignaBien;
VAR n,p: INTEGER;
BEGIN
  n:= 5;
  p:= trunc(n/2);
END.
```

BIEN

n/2 da un resultado REAL y p es de tipo INTEGER

Traza de valores en memoria

n	p
?	?
5	?
5	2

TEMARIO

- Introducción.
- Diagrama sintáctico.
- Asignación.
- Código fuente.
- Tipos de datos.

CONCEPTOS: PROGRAMA – CÓDIGO FUENTE

- Un **programa** de computadoras *(computer program)*, es un conjunto organizado de instrucciones que están escritas en un lenguaje de programación, y al ser ejecutadas por una computadora resuelven una tarea específica.
- Cada lenguaje de programación nos brinda una manera de organizar las instrucciones de un programa.
- Pascal es un lenguaje de programación **secuencial** e **imperativo**, pensado para crear programas que serán ejecutados secuencialmente en un solo procesador.
- El texto de un programa de computadoras se conoce como **código** fuente *(source code)*.

TEMARIO

- Introducción.
- Diagrama sintáctico.
- Asignación.
- Código fuente.
- Tipos de datos.

CONCEPTOS: TIPOS DE DATOS

- **Tipo de Dato**: define el *conjunto de valores* posibles que puede tomar una variable, y *las operaciones* que pueden aplicarse.
- En Pascal (y otros lenguajes de programación) las variables deben ser declaradas indicando su TIPO de DATO.
- En Pascal los tipos de datos pueden ser:
  1. **predefinidos** (ya tienen un conjunto de valores establecido y proveen operaciones para su uso). Ejemplos que veremos hoy:
     - INTEGER (enteros)
     - REAL (reales)
     - CHAR (caracteres)
     - BOOLEAN (lógico: verdadero o falso)
  2. **definidos por el programador** (el programador define los valores y las operaciones).

TIPO DE DATO PREDEFINIDO DE PASCAL: INTEGER

- **Nombre:** INTEGER (entero)
- **Valores:** cualquier número entero entre un mínimo y un máximo definido por el compilador usado.
- **Operaciones predefinidas**: que se pueden usar con tipo INTEGER.
  + (adición)
  – (substracción)
  * (multiplicación)
  div (división entera) y
  mod (resto de la división entera).

TIPO DE DATO PREDEFINIDO DE PASCAL: INTEGER

- Operadores relacionales para comparar enteros:
  =, >, <, <> (distinto), >= (mayor o igual) y <= (menor o igual)
- *Obs: los operadores tienen la precedencia usual y los paréntesis () permiten cambiar el orden de evaluación.*
- **Constante predefinida**: MAXINT (máximo valor INTEGER).
- **Función predefinida**: SQR (devuelve el cuadrado (square) de un entero).
  Ejemplo:
  SQR(3) = 9.
  SQR(SQR(3)) =81

REALICE UNA TRAZA Y LUEGO PASE A LA MÁQUINA

```
PROGRAM EjemploInteger; {Algunos ejemplos para el tipo entero}
VAR  N1, N2, N3, N4: INTEGER;
       form: INTEGER; {para el "formateo" en pantalla}
BEGIN
    writeln('Máximo entero: ', MAXINT);
    N1 := 2000 mod 2;
    N2 := 2000 div 2 ;
    N3 := SQR(SQR(3));
    N4 := MAXINT;
    form:= 15; //valor para el formateo
    writeln(n1:form, n2:form, n3:form, n4:form);
END.
```


TIPO DE DATO PREDEFINIDO DE PASCAL: BOOLEAN

- **Nombre**: BOOLEAN (Booleano o lógico)
- **Valores**: (solamente dos) true, false.
- **Operadores predefinidos:**
  - and (conjunción "y"),
  - or (disyunción "o")
  - not (negación "no")
- **Operadores relacionales:** =, >, <, <>, >=, <=

CONCEPTOS: EXPRESIONES LÓGICAS

- Hay sólo dos valores lógicos (también llamados valores de verdad):
  – verdadero (`true`)
  – falso (`false`)
- Los operadores relacionales retornan un valor de verdad de tipo BOOLEAN (true o false):
  – `5 > 3` retorna `true`,
  – `5 <> 5` retorna `false`
- Los operadores lógicos: `and`, `or` y `not`, reciben valores de verdad y retornan valores de verdad.

CONCEPTOS: EXPRESIONES LÓGICAS

- Los operadores lógicos permiten construir **expresiones lógicas** que retornarán un valor de verdad.
  Ejemplo: considere `num` de tipo entero:
  `(num > 0) and (num < 10) or not (num = 5)`
- A diferencia de los operadores numéricos, para definir el resultado de un operador lógico, **alcanza con una tabla** (llamada Tabla de Verdad).

TABLAS DE VERDAD

- Una **tabla de verdad** para un operador lógico, muestra explícitamente el resultado para cada valor posible.
- Sea **A** una expresión lógica cualquiera (esto es, su resultado es verdadero o falso), la tabla de verdad de la negación es la siguiente:

`not A` ← **En Pascal**

A	no A
true	false
false	true

- **Ejemplo**: **A** podría representar "tengo señal de wi-fi"
- En este caso, si "tengo señal de wi-fi" es verdadero, entonces "no tengo señal de wi-fi" es falso.

TABLAS DE VERDAD

- Sean **A** y **B** dos expresiones lógicas cualquiera, las tablas de verdad de la conjunción (**y**) y de la disyunción (**o**) son:

		`A and B`	`A or B`
A	B	A y B	A o B
true	true	true	true
true	false	flase	true
false	true	false	true
false	false	false	false

- Ejemplos:
  - podría representar las condiciones para **realizar una llamada** con la expresión "tengo señal y tengo saldo"
  - podría representar las condiciones para **utilizar Internet** con la expresión "hay red wi-fi o hay red de datos móviles"

PRECEDENCIA DE LOS OPERADORES LÓGICOS

- Los operadores pueden combinarse.
- La precedencia es:
  - **no** (**not**),
  - **y** (**and**),
  - **o** (**or**)
- Los **paréntesis** cambian el orden de evaluación.
- **Por ejemplo**, quiero representar una condición con la cual puedo visitar una página de internet.

  Compare estas dos expresiones para diferentes casos de prueba:

  hay wi-fi o hay red-datos y no (batería = 0)

  (hay wi-fi o hay red-datos) y no (batería = 0)

  Caso de prueba:

  hay wifi = verdadero,

  hay red-datos = falso,

  batería = 0
- **Compare**: no A y B con no (A y B)

  Pruebe con: **A** = falso **B**= falso; y luego con **A**= verdadero **B**=falso

PRECEDENCIA DE LOS OPERADORES EN PASCAL

- Tabla 12.1: Precedencia de operadores en Free Pascal

  http://www.freepascal.org/docs-html/ref/refch12.html

Operador	Precedencia	Categoría
not	La más alta (primero)	Unario
*** / div mod and**	segundo	binario
+ - or	tercero	binario
= <> < > <= >=	La más baja (último)	relacional

- Para alterar el orden de precedencia en la evaluación se utilizan los () paréntesis.

EXPRESIONES LÓGICAS EQUIVALENTES

- Dos expresiones lógicas son **equivalentes** si para todos los casos donde una es verdadera la otra también es verdadera.
- **Ejemplos**:

  A > 0 es equivalente a not (A <= 0)

  not (A or B) no es equivalente a (not A or not B)

- **Importante**:
  - con un ejemplo alcanza para mostrar que no es equivalente,
  - sin embargo, para mostrar que sí es equivalente hay que mostrar para todos los casos posibles.

EXPRESIONES LÓGICAS EQUIVALENTES

¿Por qué son importantes las expresiones equivalentes?

- Porque la misma condición puede escribirse de diferentes maneras, y hay que buscar la más adecuada.
- **Ejemplo**

La condición ***"El número entero N tiene un solo dígito"*** puede representarse con cualquiera de estas cuatro expresiones equivalentes:

(N=1) or (N=2) or (N=3) or (N=4) or (N=5) or (N=6) or (N=7) or (N=8) or (N=9) or (N=0) or (N=-1) or (N=-2) or (N=-3) or (N=-4) or (N=-5) or (N=-6) or (N=-7) or (N=-8) or (N=-9)

(N > -10) and (N < 10)

(N >= -9) and (N <= 9)

N div 10 = 0

¿Cuál usaría?

REALICE UNA TRAZA Y LUEGO PASE A LA MÁQUINA

```
PROGRAM EjemploBool; {Algunos ejemplos con el tipo Boolean}
VAR   R1: REAL;
      es_par, es_positivo, mayor_a_maxint: BOOLEAN;
      condicion: BOOLEAN;
BEGIN
write('ingrese un número');
read(R1);
es_par :=  (TRUNC(R1) mod 2) <> 1;
es_positivo := R1 >= 0;
mayor_a_maxint := R1 > MAXINT;
condicion := es_par and es_positivo and not mayor_a_maxint;
writeln('Resultado de la expresión lógica:  ', condicion);
readln;
END.
```


EXPRESIONES NUMÉRICAS Y LÓGICAS

- Al introducir la primitiva de asignación, se mostró que el lado derecho al símbolo "**:=**" es una **expresión** que da un valor.
- Las expresiones indican ("expresan") como calcular adecuadamente un valor.
- Saber construir correctamente expresiones es muy importante porque:
  - se utilizan de muchas maneras en un algoritmo (no solo en asignaciones)
  - hay expresiones de muchos tipos de valores (no solo numéricos).

OPERADORES Y VALORES

- Operadores **numéricos**: (ej, **+ - / * mod div**).
  Toman números y tienen un número por resultado.
- Operadores **relacionales**: (ej. **= > < <> >= <=**).
  Relacionan dos datos del mismo tipo y tienen un resultado que es **verdadero** o **falso**.
- Operadores **lógicos**: (ej. **and or not**).
  Toman valores del conjunto { verdadero, falso } y su resultado es un valor verdadero o falso.

EXPRESIONES NUMÉRICAS Y LÓGICAS

- **Tarea**, escriba expresiones para:
  - Un número N es mayor a 10
  - N es mayor a 10 y menor a 100.
  - N tiene a lo sumo 4 dígitos.
  - N tiene 4 dígitos (exactamente).
  - N tiene dos o cuatro dígitos.
  - N es un número impar.
  - N es divisible por 7 y divisible por 11 y tiene dos dígitos.

EL CÓDIGO ASCII

- American Standard Code for Information Interchange (**ASCII**)
- Está formado por 256 símbolos, **aquí se muestran algunos**:

				32		33	!	34	"	35	#	36	$	37	%	38	&	39	'
40	(	41	)	42	*	43	+	44	,	45	-	46	.	47	/	48	0	49	1
50	2	51	3	52	4	53	5	54	6	55	7	56	8	57	9	58	:	59	;
60	<	61	=	62	>	63	?	64	@	65	A	66	B	67	C	68	D	69	E
70	F	71	G	72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W	88	X	89	Y
90	Z	91	[	92	\	93	]	94	^	95	_	96	`	97	a	98	b	99	c
100	d	101	e	102	f	103	g	104	h	105	i	106	j	107	k	108	l	109	m
110	n	111	o	112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124	|	125	}	126	~	127	|	128	Ç	129	ü
130	é	160	á	161	í	162	ó	163	ú	164	ñ	165	Ñ			168	¿		

TIPO DE DATO PREDEFINIDO DE PASCAL

- **Nombre**: CHAR (caracter)
- **Valores**: es el conjunto de los 256 caracteres del código ASCII (American Standard Code for Information Interchange)
- **Operaciones predefinidas**: (relacionales) =, >, <, <>, >=, <=
- **Funciones predefinidas**: (ver a continuación).

*¿Cómo se diferencia en Pascal entre **una variable cuyo identificador es A y el símbolo ASCII A** ?*

- Para indicar un **valor** de tipo CHAR, se utilizan las comillas simples. Ej : ´a´, ´?´, ´ + ´, ´ ´, etc.
- En Pascal: ´**A**´ es un valor del tipo char, pero en cambio **A** es un identificador creado por un programador.

FUNCIONES PREDEFINIDAS DEL TIPO CHAR

- **CHR**: permite obtener un caracter cualquiera a partir de su código ASCII.
  Ejemplo: chr(65) = ´A´; chr(33) = ´!´.
- **ORD**: dado un caracter cualquiera, devuelve su código ASCII.
  Ejemplos: ord(´A´) = 65, ord(´!´) = 33.
- **SUCC**: retorna el siguiente ASCII si es que existe.
  Ejemplos: succ(´A´) = ´B´ , succ(´0´) = '1'
- **PRED**: retorna el anterior ASCII si es que existe.
  Ejemplos: pred(´B´) = ´A´, pred(´A´) = '@'

```
PROGRAM Espaniol;
BEGIN
write(chr(160),chr(130),chr(161),chr(162),chr(163));
write(chr(164),chr(165),chr(168));
END.
```

TIPO DE DATO PREDEFINIDO DE PASCAL

- **Nombre**: **REAL** (real)
- **Valores**: Se pueden expresar con punto decimal (3.5459), o en notación científica:
  Ejemplo: $3.5*10^{-3} = 0.0035$ en Pascal es 3.5E-3 y
  $1.28*10^{8} = 128000000$ es 1.28E8
- Es un subconjunto de los números reales en dos sentidos: (1) tiene mínimo y máximo, y (2) tiene una "precisión" máxima. No se cumple que "entre dos números reales existe siempre otro número real".
- **Operaciones predefinidas**: +, –, *, / (división)
- **Operadores relacionales**: =, >, <, <>, >=, <=
- **Funciones predefinidas**: ver a continuación

ALGUNAS FUNCIONES PREDEFINIDAS PARA REAL

- **Funciones trigonométricas**: **SIN, COS** y **TAN**. Dado un valor de un ángulo (en radianes), devuelven su seno, coseno o tangente.
  Ejemplos: SIN(0) = 0, COS(0) = 1.
- **Función raíz cuadrada** (square root) **SQRT**
  Ejemplo: SQRT(4) = 2.0
- **Función de redondeo ROUND**: dado un valor real, devuelve el **entero** más cercano.
  Ejemplos: ROUND(2.9) = 3 ROUND(2.3) = 2
  ROUND(2.5) = 2
- **Función truncado TRUNC**: dado un valor real, devuelve el **entero** que resulta de eliminar la parte decimal.
  Ejemplos: TRUNC (2.9) = 2 TRUNC(2.3) = 2

REALICE UNA TRAZA Y LUEGO PASE A LA MÁQUINA

```
PROGRAM EjemploReal; {Algunos ejemplos con el tipo real}
VAR
   N1,N2:INTEGER;
   R1,R2: REAL;
BEGIN
  R1 :=  20 mod 2; // Todo entero es un real
  R2 := MAXINT + 1; // Los reales tienen un rango más amplio
  N1 := TRUNC(2.5);
  N2 := ROUND(2.5);
  write(R1:5:2, R2:25:2,N1:5,N2:5);
  readln; //mantiene consola abierta
END.
```


CONCEPTOS: TIPOS ORDINALES EN PASCAL

- De los 4 tipos simples predefinidos que hemos visto, **INTEGER,CHAR** y **BOOLEAN** son **tipos ordinales** (REAL no es ordinal).
- Los **tipos ordinales** tienen estas características:
  - Tienen un primer y último elemento.
  - Tienen un orden, y para cada elemento está definido el siguiente (a excepción del último) y el anterior (a excepción del primero).
- Esto permite utilizar sobre los tipos ordinales las operaciones predefinidas:
  - succ() retorna el siguiente (excepto del último)
  - pred() retorna el anterior (excepto del primero)
  - ord() retorna el número de orden