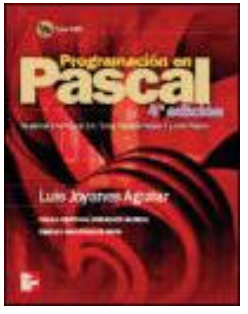
Programación en Pascal

- Segunda parte de la asignatura.
- El profesor:
  - Nombre: Fernando Pérez Nava.
  - Despacho: Edificio de la ETSII 2ª Planta.
  - Correo electrónico: fdoperez@ull.es.
  - Miércoles, Viernes: 3.30 a 5.30.
- Objetivos:
  - Aprender Pascal como primer lenguaje de programación.
  - Dominar los conceptos y las técnicas básicas de programación modular.

Programación en Pascal

- Material
  - Bibliografía:
    - **Joyanes Aguilar, Luis**, Programación en Pascal Ed. McGraw-Hill (2006).



    - **Carmona del Barco, Pablo**, Informática : fundamentos, algorítmica y programación en Pascal. Ed. Universidad de Extremadura (2003).
    - **Leestma, Sanford**, Programación en Pascal Ed.Prentice Hall (2000).
  - Transparencias de clase
  - Hojas de Problemas

El lenguaje de programación Pascal

- ¿Qué es el lenguaje Pascal?:
  - Lenguaje de programación de **alto nivel** y **propósito general** que favorece el uso de la **programación estructurada.**
- Historia:
  - Fue **desarrollado** en 1971 por Nicklaus Wirth en la Universidad de Zurich.
  - Sus **características** principales son:
    - Lenguaje fácil de implementar y eficiente.
    - Permite el desarrollo de programas bien estructurados y bien organizados.
    - Sirve para la enseñanza de los conceptos de programación.
  - Su **nombre** proviene del matemático Blaise Pascal que inventó la primera máquina de calcular.

Estructura de un programa Pascal

- La **estructura básica** de un programa en Pascal es:

```
program nombre_de_programa  (lista_de_ficheros);
uses lista_de_unidades;
  (* declaraciones de unidades *)
const
  (* declaraciones de constantes *)
type
  (* declaraciones de tipos *)
var
  (* declaraciones de variables *)

procedure nombre_procedimiento1 ...
function nombre_funcion1...
  (* declaraciones de subprogramas*)

begin
  (* sentencias ejecutables *)
end.
```

  - Todos los elementos deben estar en este orden aunque algunos pueden omitirse.

Mi primer programa en Pascal

- Código

```
program hola;
begin     (* Comienza bloque*)
     writeln('Hola a todos')
end.      (* Fin de bloque *)
```

  - ¿Qué significa todo esto?
    - **`program`** es la primera palabra de todo programa en Pascal. Es una palabra reservada (sólo se puede utilizar para indicar el comienzo del programa).
    - **`hola`** es el nombre del programa. Es un identificador (identifica con un nombre al programa).
    - **`begin`** indica el comienzo del programa (es otra palabra reservada) y sirve para agrupar sentencias (marca el comienzo de un bloque).
    - **`end`** indica el final del programa (es otra palabra reservada) y sirve también para marcar el final de un bloque.
    - El procedimiento **`writeln`** escribe un texto a la pantalla. El texto debe estar entre comillas.
  - El resultado de ejecutar el programa es:
    - Se muestra por pantalla: Hola a todos.

Comentarios en un Programa Pascal

- ¿Qué es un comentario?
  - Un **texto explicativo** que es ignorado a la hora de ejecutar el programa.
- ¿Cómo comentar en Pascal?
  - Todo comentario comienza con un (* y termina con un *)
  - No se pueden anidar comentarios (* (* *) *)
  - En algunas versiones de Pascal se puede comentar empezando con { y terminando con }
  - Ejemplo:

```
program hola;
begin     (* Comienza bloque*)
     writeln('Hola a todos')
end.      (* Fin de bloque *)
```

Comentarios

- ¿Para que sirven los comentarios?
  - Permiten que el código del programa sea **más fácil de entender** por otros programadores.
  - Facilitan la **eliminación de errores** en el programa al permitir ignorar partes del código.

Elementos básicos de un programa Pascal

- Los **elementos básicos** de un programa Pascal son:
  - Conjunto de caracteres.
  - Identificadores.
  - Tipos de datos.
  - Constantes.
  - Variables.
  - Expresiones.
  - Sentencias.
  - Procedimientos y funciones.

- Conjunto de caracteres
  - Son los que **permiten escribir el programa**.
  - Pascal emplea:
    - De la A a la Z (mayúsculas y minúsculas)
    - Los dígitos del 0 al 9
    - Ciertos símbolos especiales:

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

Identificadores en Pascal

- ¿Para que se utilizan los identificadores?
  - En la mayoría de los programas es necesario **manejar** datos almacenados en la memoria del ordenador (constantes y variables). Para poder **manipular** dichos datos, necesitamos tener acceso al lugar de memoria donde se encuentran; esto se logra **dando nombres** (identificadores) a los datos.
  - Los identificadores también se utilizan para dar nombres a los programas, procedimientos y funciones.
- ¿Cómo se forman?
  - Pueden estar compuestos de caracteres alfabéticos, numéricos y el carácter de subrayado _
  - Deben comenzar con un carácter alfabético o el carácter de subrayado.
  - No se distingue entre mayúsculas y minúsculas.
  - No se permite el uso de identificadores reservados.

Identificadores reservados y predefinidos

- Identificadores reservados.
  - Son elementos del lenguaje Pascal que tienen un **significado predefinido** que **no puede cambiarse** por el programador:

    `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`

- Identificadores predefinidos
  - Pascal tiene una serie de identificadores predefinidos que el programador puede **cambiar**:

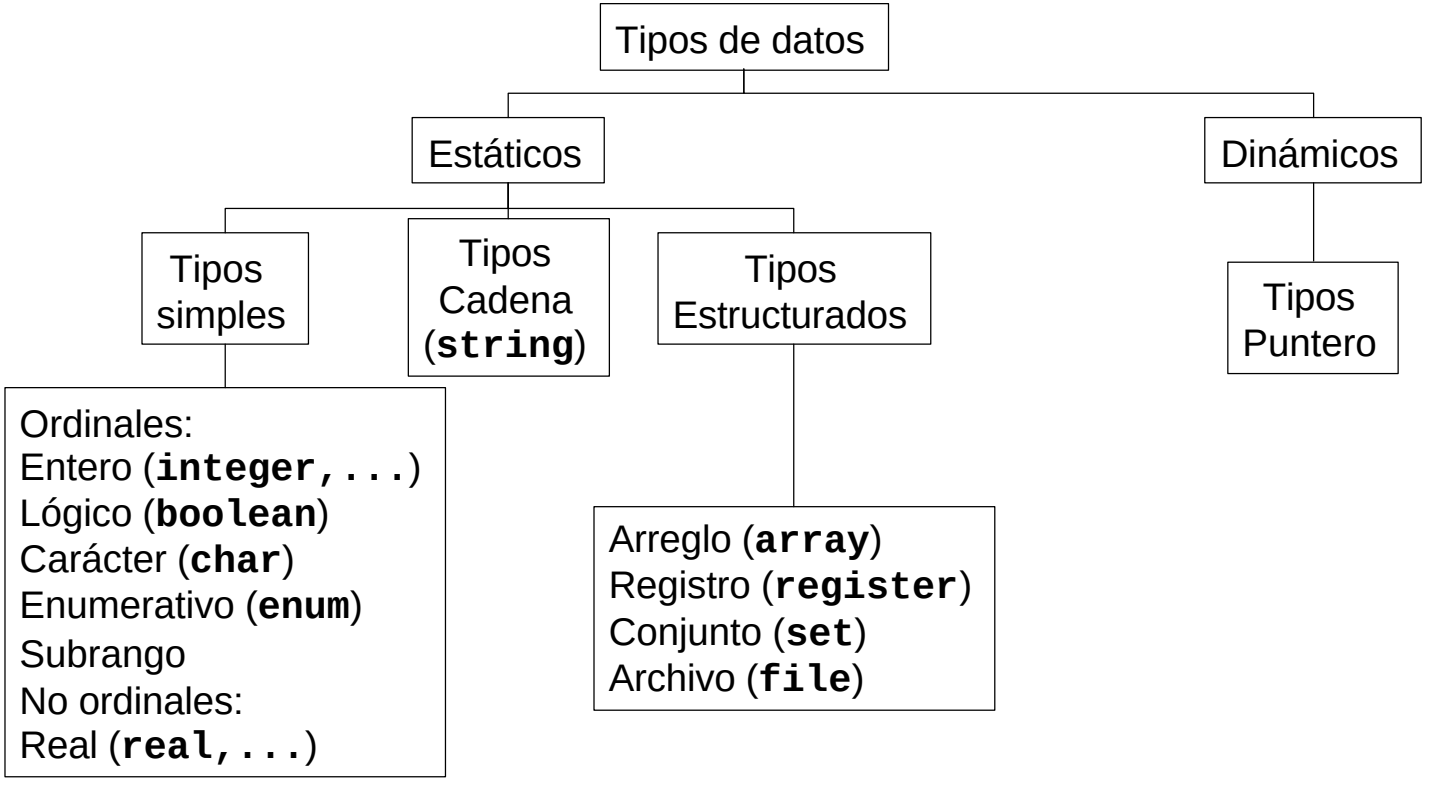
    ```
    abs arctan boolean char cos dispose eof eoln exp false
    input integer  ln maxint new odd ord output  pack page pred
    read readln real  reset rewrite  round sin sqr  sqrt succ
    text  true trunc  write writeln
    ```

Tipos de datos

- Tipos de datos:
  - Además de identificadores, **todos** los datos deben tener asignado algún tipo que indique:
    - La cantidad de memoria necesaria para almacenarlos.
    - El rango de valores que puede tomar.
- ¿Por qué se utilizan los tipos de datos?
  - Facilita la detección de **errores** en los programas
    - Por ejemplo no se puede sumar un número y un carácter.
  - Permite determinar como **ejecutar** ciertas operaciones:
    - El operador + permite en Pascal sumar:
      - Números 2+3=5.
      - Cadenas de caracteres: “Santa ”+”Cruz”=“Santa Cruz”.

Clasificación de los tipos de datos

- Los principales tipos de datos en Pascal son:



Tipo entero

- El tipo entero **más importante** es el **`integer`**
  - Se representa por una secuencia de dígitos precedida (opcionalmente) por un signo.
  - Ejemplo:
    - Enteros válidos: 0, +345 , -5280, 567
    - Enteros no válidos: 12,5 , 34., 10 20
  - El rango de los valores definidos por el tipo **`integer`**, en FreePascal, se encuentra en [-32768, 32767].
  - Cada valor de este tipo se guarda en dos bytes de memoria.
- Otros tipos enteros
  - **`byte`**: Rango [0,255], Almacenamiento: 1 byte.
  - **`longint:`** Rango [-2147483648, 2147483647]. Almacenamiento: 4 bytes.

Tipos reales

- El tipo real **más importante** es el **`real`**
  - Se representa mediante una secuencia de números precedida (opcionalmente) por un signo que debe contener un punto decimal o un exponente (o ambos). Si se incluye un punto decimal, éste debe aparecer entre dos dígitos.
    - Ejemplo:
      - Reales válidos: 0.0, +23.5 , -1280.567, 3.0e+10
      - Reales no válidos: 12. , 1,034.0 , 20, 3.e+10, 3e 10
  - El rango de los valores definidos por el tipo **`real`**, en FreePascal, se encuentra aproximadamente en $[-10^{38}, 10^{38}]$.
  - Cada valor de este tipo se guarda en 6 bytes de memoria.
- Otros tipos reales
  - **`double`**: Rango aproximado $[-10^{308}, 10^{308}]$ Almacenamiento: 8 bytes.

Tipos carácter, lógico y cadena

- El tipo carácter: **`char`**
  - Un valor de tipo **`char`** es cualquier **carácter** que se encuentre dentro del conjunto ASCII ampliado, el cual está formado por los 128 caracteres del ASCII más los 128 caracteres especiales.
  - Se guarda en un byte de memoria.
  - Ejemplos: ‘a’ ,’z’, ‘+’.
- El tipo lógico: **`boolean`**
  - Sirven para representar un **valor lógico**.
  - Se guarda en un byte de memoria.
  - Solo toma dos valores: **`true`** y **`false.`**
- El tipo cadena: **`string`**
  - **Secuencia de caracteres** entre comillas simples.
    - Para usar una comilla simple en una cadena debe escribirse dos veces.
    - Ejemplos: `‘Hola’, ‘30-11-00’, ‘1 euro’, ‘O’’neill’`

Constantes

- Constantes:
  - Una constante es un valor que **no puede cambiar** a lo largo de la ejecución del programa.
- Tipos de Constantes:
  - Literales
    - Es un valor de cualquier tipo que se utiliza en el programa como tal.
      - Ejemplo:

        `VolumenEsfera:=4/3*Pi*Radio*Radio*Radio;`

        `4 y 3` son constantes literales
  - Con nombres
    - Sirven para asociar un dato simple a un identificador de forma permanente a lo largo de todo el programa.
    - Se definen en la sección **`const`** del programa.
    - Ejemplo:

```
const
  pi= 3.1415926535897932;
```

Variables

- Variable:
  - Identificador cuyo valor **varía** a lo largo del programa
  - Cada variable tiene **asociado un tipo** determinado
  - Se **declaran** en la sección **`var`** del programa

```
var
  lista_de_identificadores1=tipo1;
  lista_de_identificadores2=tipo2;
  lista_de_identificadores3=tipo3;
```

    - *`lista_de_identificadores`* es una serie de identificadores separados por comas. Todos son del mismo tipo.
  - Ejemplo:

```
var fila,columna: integer;
    valor:real;
    condicion:boolean;
```

Uso de las Variables

- Asignación de valores a variables
  - Una vez declarada una variable, generalmente se le **asigna** algún valor.
  - En Pascal esta asignación se realiza con el operador `:=`
  - Ejemplo:

```
var
  numero1, numero2, numero3 : integer;
begin
  numero1 := 43; (* hace numero1 igual a 43 *)
  numero2 := 34; (* hace numero2 igual a 34 *)
  numero3 := numero1 + numero2;
          (* hace numero3 igual a numero1 + numero2 *)
end.
```

Reglas en el uso de Variables

- Algunas reglas para el uso de variables
  - Una variable en el lado derecho de una sentencia de asignación debe tener un valor antes de que se ejecute la sentencia de asignación.
    - Ejemplo:

```
y:=x+1 (*x debe tener un valor*)
```

  - En la izquierda de una sentencia de asignación sólo puede haber una variable.
    - Ejemplo:

```
Precio-Descuento:=10; (*Error*)
```

  - El símbolo de igualdad “=“ sólo se utiliza para operaciones lógicas, no de asignación.
    - Ejemplo:

```
Precio=20 (*Se comprueba se precio vale 20*)
```

Operaciones de asignación especiales

- Contador
  - Un contador es una variable que se incrementa, cuando se ejecuta la instrucción, en una cantidad constante.
    - Ejemplo:

```
contador:=0;                (*contador vale 0*)
contador:=contador+1;       (*Al ejecutarse
                            contador vale 1*)
```

- Acumulador
  - Un contador es una variable que se incrementa, cuando se ejecuta la instrucción, en una cantidad variable.
    - Ejemplo:

```
x:=5;                 (*x vale 5*)
suma:=3;              (*suma vale 3*)
suma:=suma+x;         (*Al ejecutarse
                      suma vale 8*)
```

Expresiones y Operadores

- Expresiones:
  - Es una **colección de operandos** (números, constantes, variables) **enlazadas por ciertos operadores** (suma, resta,...) que representa un valor.
  - Hay dos tipos de expresiones:
    - Numéricas: representa un valor numérico.
      Ejemplo: `b*b-4*a*c`
    - Lógicas: representa una condición lógica.
      Ejemplo: `hora>8`
- Operadores
  - Permiten obtener **nuevos** valores mediante la combinación de operandos.
  - Tipos de operadores en Pascal:
    - Aritméticos.
    - Relacionales.
    - Lógicos.

Operadores Aritméticos

- Sirven para realizar operaciones numéricas:

Operador	Operación	Operandos	Ejemplo	Resultado
-	Menos	real , integer	-a	Cambia el signo de a
+	Suma	real , integer	a + b	Suma de a y b
-	Resta	real , integer	a - b	Diferencia de a y b
*	Multiplicación	real , integer	a * b	Producto de a por b
/	División real	real , integer	a / b	Cociente de a y b
div	División entera	integer	a div b	Cociente entero de a y b
mod	Módulo	integer	a mod b	Resto de a por b

- Observaciones
  1. Cuando los dos operandos sean del tipo **`integer`**, el resultado será de tipo **`integer`**. (Excepto en la división real).
  2. Cuando cualquiera de los dos operandos sean del tipo **`real`**, el resultado será de tipo **`real.`**
- Ejemplos:
  - `1+1 = 2,  2.5*4 = 10, 3/2 = 1.5, 5 div 2 =2, 5 mod 2=1`

Operadores Relacionales

- Relaciones:
  - Consisten de dos operandos separados por un operador relacional.
  - Si la relación es satisfecha, el resultado tendrá un valor lógico **`true`** ; si la relación no se satisface, el resultado tendrá el valor **`false`**.
  - Operadores relacionales utilizados en Pascal:

Operador	Operación	Operandos	Ejemplo	Resultado
=	Igual	real , integer, char,boolean	a = b	`true` si a = b `false` si a ≠ b
<>	Distinto	real , integer, char,boolean	a <> b	`true` si a ≠ b `false` si a = b
<	Menor	real , integer, char,boolean	a < b	`true` si a < b `false` si a ≥ b
>	Mayor	real , integer, char,boolean	a > b	`true` si a > b `false` si a ≤ b
<=	Menor o igual	real , integer, char,boolean	a <= b	`true` si a ≤ b `false` si a > b
>=	Mayor o igual	real , integer, char,boolean	a >= b	`true` si a ≥ b `false` si a < b

  - Ejemplo:
    - 2=3 (**`true`**), `'a'<>'a'` (**`false`**)

Operadores Lógicos

- Los operadores lógicos actúan sobre operandos lógicos y devuelven `true` o `false`

Operador	Operación	Ejemplo	Resultado
`not`	no lógico	`not a`	`true` si a es `false` `false` si a es `true`
`and`	y lógico	`a and b`	`true` si a y b son `true` `false` si a o b son `false`
`or`	o lógico	`a or b`	`true` si a o b son `true` `false` si a y b son `false`
`xor`	o exclusivo	`a xor b`	`true` si uno de a o b es `true` `false` si a y b son `true o false`

- Ejemplo:
  - `not true = false, true and true = true, true or false = true`

Evaluación de Expresiones

- Reglas de evaluación de expresiones
  1. Todas las subexpresiones entre paréntesis se evalúan primero. Si hay varias subexpresiones anidadas se evalúan de dentro a fuera.
  2. Dentro de una misma expresión o subexpresión, los operadores se evalúan en el siguiente nivel de prioridad
     - Operadores Aritméticos y Relacionales
       - Alta prioridad : `*`, `/` , `div`, `mod`
       - Media prioridad: `+`, `-`
       - Baja prioridad: =, <, >, <=, >=, <>
     - Operadores Lógicos
       - Alta prioridad : `not`
       - Media prioridad: `and`
       - Baja prioridad: `or`, `xor`
  3. Los operadores en una misma expresión o subexpresión con igual nivel de prioridad se evalúan de izquierda a derecha.

Sentencias (1)

- Sentencias:
  - **Instrucción o grupo de instrucciones** que hacen que el ordenador realice ciertas **acciones**
  - Pueden ser simples o estructuradas.
    - Sentencias simples:
      - Asignar un dato a una variable, por ejemplo: `dia:=20;`
      - Acceder a un subprograma, por ejemplo: `writeln('hola');`
      - Transferir el control del programa, por ejemplo: **`goto`** `100;`
    - Sentencias estructuradas:
      - Sentencias compuestas
        - » Formadas por un grupo de sentencias simples dentro de un bloque.
        - » Ejemplo:

```
begin (* comienzo de bloque *)
      read(nombre);
      writeln(nombre)
end;   (* fin de bloque *)
```

Sentencias (2)

- Otras sentencias estructuradas
  - Sentencias condicionales
    - Sirven para realizar o no una sentencia simple o una sentencia estructurada en base a una condición
    - Ejemplo

```
if(hora >13) then
        writeln('fin');
```

  - Sentencias repetitivas
    - Sirven para realizar de forma repetitiva una sentencia simple o una sentencia estructurada
    - Ejemplo:

```
for hora:=0 to 23 do
        writeln(hora);
```

- Todas las sentencias y líneas del programa se terminan con un punto y coma excepto las palabras claves **begin** y **end**. Las sentencias anteriores a un **end** no requieren punto y coma.
- El último **end** del programa se termina con un punto.

Procedimientos y funciones (1)

- Procedimientos y Funciones:
  - Son **elementos autónomos** de programa que realizan acciones determinadas.
  - Características:
    - Pueden ser llamados desde cualquier punto del programa pudiendo recibir información diferente en cada llamada.
    - La información suministrada se procesa según las sentencias del módulo. Generalmente esto hace que se genere nueva información que se devuelve al programa en el punto donde se invocó al módulo, posteriormente el programa continúa su ejecución
    - La información transferida a un módulo se transmite mediante una lista de elementos (constantes, variables, expresiones) llamados parámetros.
    - Algunos de los parámetros pueden ser utilizados para devolver la información que se genera dentro del módulo

Procedimientos y funciones (2)

- Procedimientos
  - Se utilizan mediante una sentencia simple consistente en su nombre y una lista(opcional) de parámetros.
  - Ejemplo:
    - `writeln('hola');`
- Funciones
  - Se utilizan especificando su nombre dentro de una expresión seguida por una lista de parámetros.
  - La función devuelve un solo dato (el resultado) que está representado por el nombre de la función.
  - La función debe ser del tipo adecuado a la expresión en la que está.
  - Ejemplo:
    - `resultado:=area_circunferencia(lado);`

Procedimientos y funciones (3)

- Algunas funciones predefinidas

Nombre	Descripción	Tipo del argumento	Tipo del retorno
`abs`	valor absoluto	`real` o `integer`	Tipo del argumento
`arctan`	arco tangente (radianes)	`real` o `integer`	`real`
`cos`	coseno (radianes)	`real` o `integer`	`real`
`exp`	exponencial	`real` o `integer`	`real`
`ln`	logaritmo neperiano	`real` o `integer`	`real`
`round`	redondeo	`real`	`integer`
`sin`	seno (radianes)	`real` o `integer`	`real`
`sqr`	cuadrado (elevado a 2)	`real` o `integer`	Tipo del argumento
`sqrt`	raíz cuadrada (elevado a 1/2)	`real` o `integer`	`real`
`trunc`	truncar	`real` o `integer`	`integer`

Función	Descripción	Tipo del argumento	Tipo del retorno
`chr`	caracter con un número ASCII dado	`integer`	`char`
`ord`	número ASCII de una caracter dado	`char`	`integer`
`pred`	predecesor	`integer` o `char`	Tipo del argumento
`succ`	sucesor	`integer` o `char`	Tipo del argumento

Procedimientos de entrada y salida estándar

- Instrucciones de entrada y salida:
  - Sirven para que el programa se **comunique** con un periférico del ordenador tal como una terminal, una impresora o un disco.
  - Las instrucciones de **entrada** estándar, sirven para **leer** caracteres desde el teclado, y las instrucciones de **salida** estándar **muestran** caracteres en la pantalla.
  - En Pascal todas las operaciones de entrada/salida se realizan ejecutando los procedimientos:
    - Procedimientos de entrada: `read`, `readln`
    - Procedimientos de salida: `write`, `writeln`

Procedimientos `read` y `readln` (1)

- Los procedimientos predefinidos read y readln permiten **introducir** datos durante la ejecución de un programa.
  - Tienen la siguiente **forma** :
    - `read(`*`lista_de_variables`*`);`
    - `readln(`*`lista_de_variables`*`);` donde :

      `lista_de_variables` : es una lista de variables separadas por comas.

      Los datos que se pueden leer son : enteros, reales, caracteres, o cadenas.
  - Al ejecutarse la instrucción se obtienen del teclado, tantos valores de datos como elementos hay en *`lista_de_variables`*. Los datos deberán ser compatibles con los tipos de las variables correspondientes en la lista.

Procedimientos `read` y `readln` (2)

- La diferencia entre las instrucciones `read` y `readln` consiste en que `read` permite que la siguiente instrucción continúe leyendo valores en la misma línea; mientras que con `readln` la siguiente lectura se hará después de que se haya tecleado el carácter de fin de línea.
- Ejemplo:

```
var
nombre:string[15];
salario:real;
edad:integer
...
read(nombre);      (*introducimos por teclado
  Luis*)
read(salario,edad);(*introducimos por teclado
                    1000 40*)
```

- El efecto es que las variables: `nombre, salario` y `edad` toman los valores Luis, 1000 y 40.

Procedimientos `write` y `writeln` (1)

- Los procedimientos predefinidos `write` y `writeln` permiten **mostrar** datos durante la ejecución de un programa.
  - Las instrucciones para llamar a los procedimientos `write` y `writeln` son de la siguiente forma :
    - `write(`*`lista_de_variables`*`);`
    - `writeln(`*`lista_de_variables`*`);` donde :

      `lista_de_variables` : es una lista de identificadores de variables separados por comas.

      Los datos que se pueden mostrar son : enteros, reales, caracteres, o cadenas.
  - La diferencia entre las instrucciones `write` y `writeln` consiste en que `write` permite que la siguiente instrucción continúe en la misma línea; mientras que, con `writeln` se hará en la siguiente

Procedimientos `write` y `writeln` (2)

- Formatos de salida
  - Cada dato que se escribe con `write` y `writeln` aparece con un determinado número de caracteres denominado **longitud de campo**.
  - Cuando un dato se escribe sin una especificación de longitud de campo, se utilizará la especificación de campo por defecto
  - Para especificar **una longitud de campo determinada** se emplea:
    - `writeln (`*`valor:anchura`*`...);` con
      `anchura` : expresión entera que especifíca la anchura total del campo en que se escribe el valor.
    - `writeln (`*`valor:anchura:digitos`*` ...);`
      `dígitos`: dígitos decimales de un número real, *`anchura`*: Total de dígitos del número real contando parte entera, punto decimal y dígitos decimales.
  - Ejemplos:
    - `valor:= 25.0776;`
      `writeln(valor);2.5077600000E+01 writeln(valor:2);2.5E+01`
      `writeln(valor:2:1);25.1 writeln(valor:2:8);25.07760000`
    - `writeln('Tec':3);Tec writeln('Tec':5);••Tec`