# Entrada y Salida

InCo

2022

# La entrada estándar

- Es una *secuencia* de caracteres.
- Cada instrucción de entrada *consume* caracteres de la entrada y carga esa información en memoria (variables).
- Los caracteres consumidos desaparecen de la entrada.
- El final de la entrada se puede detectar por programa.

# La entrada como secuencia de líneas

- La entrada estándar está organizada en **líneas**.
- Las líneas son subsecuencias consecutivas de caracteres.
- El fin de cada línea está marcado por una secuencia de caracteres especiales.
- Los caracteres de fin de línea son *dependientes de la plataforma*.

# Entrada estándar por teclado

- La entrada estándar está asociada por defecto al teclado.
- La secuencia de caracteres es digitada por el usuario.
- **enter** indica el fin de línea.
- Los caracteres digitados producen *eco* en la terminal.

# La instrucción read

`read` es un procedimiento estándar que lee un dato de la entrada y lo almacena en una variable.

```
read(v);
```

La variable puede ser de tipo: `integer`, `real`, `char`.

# Lectura de enteros

La instrucción `read(num)` lee una secuencia de caracteres de la entrada hasta llegar a un espacio o fin de línea o cualquier carácter que implique el fin del entero (letras, símboles especiales, etc.).

```
var
   num : integer;

...
begin

   ...
   read(num);
   ...
```

Si se logra obtener una secuencia de caracteres que represente un número entero, este se guarda en la variable `num`. De lo contrario se produce un error y el programa aborta.

# Lectura de enteros. Ejemplos

En el siguiente cuadro:

- _ representa un espacio.

| antes | num | después |
|---|---|---|
| 345_ | 345 | _ |
| __-345_12 | -345 | _12 |
| __345# | 345 | # |
| 345a@c | 345 | abc |
| %9q@ | **error** | |

# Lectura de enteros. Ejemplos

En el siguiente cuadro:

- _ representa un espacio.
- # representa la secuencia de fin de línea.

| antes | num | después |
|---|---|---|
| 345_ | 345 | _ |
| __-345_12 | -345 | _12 |
| __345# | 345 | # |
| 345a@c | 345 | abc |
| %9q@ | **error** | |

# Lectura de reales

Es análoga a la lectura de enteros.

La instrucción `read` consume caracteres de la entrada hasta encontrar espacio o fin de línea o cualquier carácter que implique el fin del real.

El real puede estar escrito en notación *decimal* o *exponencial*.

Puede o no tener punto decimal.

# Lectura de caracteres

Cuando la variable del `read` es de tipo `char` se lee *el primer carácter* de la entrada y se almacena en la variable.

Este carácter puede ser espacio o algunos de los caracteres de fin de línea.

# Lectura múltiple

`read` puede ser invocado con muchas variables:

```
read(var1,var2,..varn);
```

Las variables pueden ser de diferentes tipos.

Es equivalente a:

```
read(var1);
read(var2);
...
read(varn);
```

# El procedimiento ReadLn

- `ReadLn` (sin variables) Consume todos los caracteres de la entrada hasta alcanzar la primera secuencia de fin de línea (que también la consume). No guarda ningún valor en ninguna variable.
- `ReadLn(v1,v2,...,vn)` Es equivalente a la siguiente secuencia:

  ```
  read(v1,v2,...,vn); ReadLn;
  ```

- El procedimiento `ReadLn` provee una manera *portable* de procesar la secuencia de fin de línea.

# Las funciones eoln y eof

Pascal provee 2 funciones para controlar el fin de una línea y el fin de la entrada:

- `eoln` retorna `true` si en la entrada viene inmediatamente la secuencia de fin de línea.
- `eof` retorna `true` si no hay más caracteres en la entrada.

Una manera portable de saltear la secuencia de fin de línea:

```
if eoln then ReadLn;
read(c);
```

# Ejemplos con ReadLn

```
var a,b,i,j: Integer;
...
readLn(a,i);
readLn(j);
readLn(b);
```

Entrada:

```
45 34
9 1.1
12
```

¿cómo quedan las variables?

# Salida Estándar

- Es una secuencia de caracteres.
- Cada instrucción de salida *produce* caracteres que los agrega al final de la salida.
- La salida está también dividida en líneas.
- La salida estándar está asociada con la terminal donde se ejecuta el programa.
- Las líneas de la salida se *mezclan* con las líneas del *eco* de la entrada.

# El procedimiento write

La instrucción:

- `write(`*expresión*`)`

se ejecuta de la siguiente forma:

- La expresión se evalúa.
- El resultado de la expresión se convierte a una cadena de caracteres
- Esa cadena de caracteres se "*manda*" a la salida.

El tipo de la expresión puede ser: `char`, `integer`, `real`, `boolean`, cadena.

# Tamaño de la salida

Se puede especificar el largo de la salida númerica de la siguiente manera:

```
var v: integer;
    r: real;
...
write(v:7);          (* 7 digitos *)
write(r:10:3);       (* 10 digitos en total, 3 decimales
```

Si no se especifica el tamaño Pascal despliega en un tamaño pre definido (dependiente de la implementación).

Los números reales se despliegan en notación *exponencial* salvo que se especifiquen los dos tamaños.

# Salida múltiple

Se puede invocar `write` con varios argumentos:

```
write(e1,e2,...,en);
```

lo cual equivale a:

```
write(e1);
write(e2);
...
write(en)
```

# El procedimiento WriteLn.

- `WriteLn` (sin argumentos) Manda a la salida la secuencia de caracteres de fin de línea.
- `WriteLn(e1,e2,...,en)` es equivalente a :

  ```
  write(e1,e2,...,en); WriteLn;
  ```

  Ejemplo:

  ```
  WriteLn('El resultado es: ', result)
  ```

  despliega en la salida:

  ```
  El resultado es: 23
  ```

# Programas Interactivos

En muchas aplicaciones se mezclan la salida del programa con la entrada, generando un diálogo:

```
program Triangulo;
var
 altura,base,area: real;
begin
    (* ingresar datos *)
     write('Ingrese altura y base: ');
     ReadLn(altura,base);
    (* calcular resultado *)
     area := base * altura / 2;
    (* mostrar resultado *)
     WriteLn('El área del triangulo es: ',area:7:2);
  end.
```