

Estructuras de control

- Estas son un conjunto mínimo de instrucciones.
 - *read, write*
 - *if, case*
 - *while, repeat until*
 - *for*

Programas en Pascal:

```
Ejemplo.pas
Program nombre;
Const
    {(Constantes del programa)}
módulos
    {(Se verá mas adelante)}
var
    {(Variables del programa)}
Begin {(Inicialización)}
    {(Se verá mas adelante)}
end. {(Finalización)}
```

Ejemplo de programa estructurado:

```
Ejemplo.pas
Program ejemplo;
Const{ (durante toda la ejecución, el valor se mantiene)}
    N = 25;
    pi = 3.14;
var
    edad: integer;
    peso: real;
    letra: char;
    resultado: boolean;
Begin
    edad := 5;
    peso := 63.5;
    edad := edad + N; {(siendo 30 por Const)}
    letra := 'A';
    resultado := letra = 'A'; {(como A y a no son iguales, devuelve dato false (boolean definido en var))}
end.
```

PRE y POST condiciones

- **Pre-condición:** La información que se conoce como verdadera ANTES de iniciar el programa (o módulo)

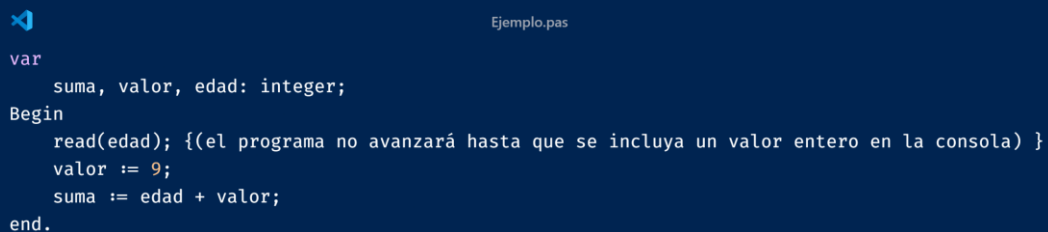
- **Post-condición:** La información que debería ser verdadera al concluir el programa (o módulo) si se cumplen adecuadamente los pasos especificados.

- **READ:** Operación que contienen la mayoría de los lenguajes de programación. Se utiliza para tomar datos desde un dispositivo de entrada (por defecto el teclado) y asignarlos a las variables correspondientes.

Asignación de valores:

- Hasta ahora sabemos que para asignar un valor se utiliza := pero podemos utilizar la estructura de control "read(x)"

Por ejemplo:



```
var
    suma, valor, edad: integer;
Begin
    read(edad); {(el programa no avanzará hasta que se incluya un valor entero en la consola) }
    valor := 9;
    suma := edad + valor;
end.
```

- **WRITE:** Operación que contienen la mayoría de los lenguajes de programación. Se utiliza para mostrar el contenido de una variable. Tiene 4 posibilidades
 - Para enseñar solo texto: write("texto")
 - Para enseñar solo una variable: write(variable)
 - Para enseñar un conjunto de texto y variable: write("texto", variable)
 - Para enseñar PEQUEÑAS operaciones: write(num + 4)
- **SECUENCIA:** Representada por una sucesión de operaciones (por ej. asignaciones) en la que el orden de ejecución coincide con el orden físico de aparición de las instrucciones.

- **DECISIÓN:** Algoritmo representativo de un problema real cuando es necesario tomar decisiones en función de los datos del problema.

IF/ELSE (TIP sintaxis: la acción anterior a "else" NO lleva ";")

Cuando el "else" tiene una acción si la cond es falsa {

```
if (condición) then
```

```
    acción1
```

```
else
```

```
    acción2;
```

Si en el "if" quiero más de una acción

```
if (condición) then
```

```
    begin
```

```
        acción1;
```

```
        acción2;
```

```
    end
```

```
else
```

```
    Acción3
```

Si en ambas es más de una acción

```
if (condición) then
```

```
    begin
```

```
        acción1;
```

```
        acción2;
```

```
    end
```

```
else
```

```
    begin
```

```
        acción1;
```

```
        acción2;
```

```
    end;
```

```
},
```

},

Ejemplo 3.0 = Realizar un programa que lea dos numeros enteros e informe si la suma de los mismos es mayor a 20.

Resolución = Primer año\1er Cuatrimestre\CADP\Teorías\3.01Ej1.pas

Ejemplo 3.02 = Realizar un programa que lea un numero (suponga que > 0) y asigne el valor "10" si el número es menor a 10; asigne "50" a la misma variable si el número es mayor a 10 pero menor que 50; y asigne "100" si es mayor que 50.

Resolución = Primer año\1er Cuatrimestre\CADP\Teorías\3.01Ej2.pas

- **ITERACIÓN:** Puede ocurrir que se desee ejecutar un bloque de instrucciones desconociendo el numero exacto de veces que se ejecuta. Para esto, existe la iteración en la mayoría de los lenguajes de programación. Se clasifican en PRE y POST condicionales.

Iteración PRE condicional = Evalúan la condición y si es verdadera se ejecuta el bloque de acciones. Se puede ejecutar 0 (falsa, no se ejecuta), 1 (verdadera una vez) y/o mas veces.

WHILE DO: Sintaxis

```
while (condición) do
```

```
    acción;
```

(Cuando tiene mas de 1 acción)

```
while (condición) do
```

```
    begin
```

```
        acción1;
```

```
        acción2;
```

```
    end;
```

Ejemplo 3.04 = Realizar un programa que lea códigos de productos hasta leer un código igual a 30. Al finalizar informe sobre la cantidad de productos con código par.

Resolución = Primer año\1er Cuatrimestre\CADP\Teorías\3.01Ej4.pas

*** No olvidar**

- Para que while funcione debe tener algo que evalúe la condición.
- Si nunca leo otro producto, el while toma el primero y nunca finaliza.
- Utilizo while cuando NO se la cantidad de veces de lo que pida.

ITERACIÓN POST condicional = Ejecutan las acciones, luego evalúan la condición y ejecutan las acciones mientras la condición es falsa.

REPEAT UNTIL: Sintaxis

```
repeat
    acción;
until (condición);

(no se utiliza begin/end si hay mas de 1)
```

Ejemplo 3.5 = Realizar un programa que lea códigos de productos hasta leer un código igual a 30. Al finalizar informe sobre la cantidad de productos con código par. El último DEBE procesarse.

Resolución = Primer año\1er Cuatrimestre\CADP\Teorías\3.01Ej5.pas

* **No olvidar**

- Siempre pedirán cuando DEBA procesarse el último, de modo contrario, se asume que NO debe procesarse.

- Si hay más de una acción NO se agrega begin/end.

- **REPETICIÓN** = Extensión natural de la secuencia. Consiste en repetir N veces un bloque de acciones. Este N de veces ya debe ser sabido de antemano.

FOR = Sintaxis

```
for indice := valor_inicial to valor_final do
    acción1;

(Si se tiene más de una acción)

for indice := valor_inicial to valor_final do
    begin
        acción1;
        acción2;
    end;
```

Ejemplo 3.6 = a - Realizar un programa que lea precios de 10 productos que vende un almacén. Al finalizar, informe la suma de todos los precios.

b - Consultar el precio del 5to producto.

Resolución = a - Primer año\1er Cuatrimestre\CADP\Teorías\3.01Ej6.pas

b - Primer año\1er Cuatrimestre\CADP\Teorías\3.01Ej6b.pas

*** No olvidar**

- La variable índice debe ser de tipo simple/ordinal.
- Nunca puede modificarse.
- Se incrementa/decrementa sola.
- Cuando el for termina, no toma ningún valor (ni el último)
- Si se necesita de menor a mayor, el TO cambia a DOWNTO.