

# Iteración. Ejemplos Avanzados

Programación 1

InCo - FING

## Section 1

# Resumen Instrucciones de Control

**Simples** Asignación, Llamada a procedimiento.

**Compuestas** Secuencia, selección, iteración.

- **Secuencia:** `begin ...;... end`
- **Selección:** `if-then-else, case`
- **Repetición:** `for, while, repeat`

# Las instrucciones de repetición

- for** Repetición controlada por una variable que toma valores en un rango determinado. Puede no ejecutarse nunca el cuerpo. Siempre termina.
- while** Repetición controlada por una expresión booleana. Puede no ejecutarse nunca el cuerpo. Puede no terminar nunca.
- repeat** Repetición controlada por una expresión booleana. El cuerpo siempre se ejecuta una vez al menos. Puede no terminar nunca.

## Section 2

### Pragmática del for

## Cuándo utilizar for (1)

- Repetir algo  $N$  veces. Donde  $N$  es una expresión entera.

```
(* escribir N asteriscos *)  
for i:= 1 to N do  
    write('*');
```

```
(* dejar N líneas *)  
for i:= 1 to N do  
    WriteLn;
```

## Cuándo Utilizar for (2)

- Una función numérica expresada por sumatoria o productoria:

```
(* factorial *)
factorial:= 1;
for i:= 2 to N do
    factorial:= factorial * i;

(* potencia *)
potencia:= 1;
for i:= 1 to exponente do
    potencia:= potencia * base;
```

## Cuándo Utilizar for (3)

- Recorrer un rango de enteros (o caracteres) completamente.

```
(* imprimir el código de todas las letras mayúsculas *)  
for car:= 'A' to 'Z' do  
    WriteLn(car, '--->', ord(car));
```

```
(* mostrar los múltiplos de m en el intervalo [a,b] *)  
for num:= a to b do  
    if num mod m = 0 then  
        WriteLn(num);
```



## Errores con for (1)

- Incluir un control de que el primer valor es menor o igual que el segundo.

```
(* INCORRECTO!! *)  
if a <= b then  
  for num:= a to b do  
    if num mod m = 0 then  
      WriteLn(num);
```

El if es innecesario, el for ya controla esa situación.

## Errores con for (2)

- Modificar el valor de la variable de control para avanzar “más rápido”.

```
(* INCORRECTO!! *)  
for i:= 1 to 10 do  
begin  
  i:= i + 1; (* NO MODIFICAR LA VARIABLE DE CONTROL *)  
  WriteLn(i);  
end;
```

```
(* correcto *)  
for i:= 1 to 5 do  
  WriteLn(2*i);
```

## Errores con for (3)

- Modificar el valor de la variable de control para forzar la terminación:

```
(* INCORRECTO!! *)
for intentos:= 1 to 10 do
begin
  ReadLn(numLeido);
  if numLeido <> NUMERO then
    WriteLn('Incorrecto. ', intentos:2, ' intentos')
  else
    intentos:= 10; (* NO MODIFICAR VARIABLE DE CONTROL *)
end;
```

La versión correcta utiliza while (ver **clase anterior**).

## Errores con for (4)

- Utilizar el valor de la variable de control una vez terminada la repetición:

```
(* INCORRECTO! *)  
for i:= algo1 to algo2 do  
begin  
    hacer_algo;  
    ...  
end;  
  
write(i); (* valor indefinido *)
```

## Errores con for (5)

Anidar un if innecesariamente dentro del cuerpo de la iteración.

**Ejemplo:** Desplegar los números en el intervalo  $[a,b]$  separados por coma.  
Suponemos  $a \leq b$ .

```
(* INCORRECTO! *)
for i:= a to b do
begin
  write(i);
  if i <> b then
    write(',')
  else
    WriteLn;
end;

(* correcto *)
write(a);
for i:= succ(a) to b do
  Write(',',i);
WriteLn;
```

## Section 3

### Pragmática de while

# Cuándo utilizar while (1)

No se conoce a priori el número de repeticiones:

- *Lectura con centinela*. Se termina cuando se lee el centinela. (clase 6)
- *División entera por restas*. Se termina cuando el resto es menor que el dividendo. (clase 6)

En todos los casos la cantidad de repeticiones puede ser 0.

## Cuándo utilizar while (2)

Recorrida de una secuencia de valores en *búsqueda de una condición*. Si se encuentra un valor que cumple la condición la recorrida termina (tempranamente).

```
valor:= PRIMERO;
while (valor <= ULTIMO) and not cumple_condicion(valor) do
    valor:= siguiente(valor);

if valor<= ULTIMO then
    WriteLn('Exito: ', valor)
else
    WriteLn('Fracaso');
```

**Ejemplos** (ver clase 6):

- Determinar si un número es primo
- Adivinar un número en menos de 10 aciertos.



## Errores con while (1)

El problema admite una solución utilizando for:

```
(* INCORRECTO! *)
```

```
  i:= PRIMERO;  
  while i<= ULTIMO do  
  begin  
    hacer_algo;  
    i:= i+1;  
  end;
```

```
(* CORRECTO *)
```

```
  for i:= PRIMERO to ULTIMO do  
    hacer_algo;
```

## Errores con while (2)

- Utilizar un if considerando un caso de borde ya previsto en la condición del while.

Ejemplo: *división entera por restas*:

```
begin
  readln(dividendo,divisor);
  if dividendo < divisor then
    writeln('El resto es: ', dividendo, ' y El cociente es: ', 0)
  else
    begin
      resto:= dividendo;
      cociente:= 0;
      while resto >= divisor do
        begin
          resto:= resto - divisor;
          cociente:= cociente + 1
        end;
      writeln('El resto es: ', resto, 'y El cociente es: ', cociente)
    end
  end.
```

El if es innecesario. La solución correcta está en la **clase 6**.

## Errores con while (3)

- Lectura con centinela “mal diseñada”:

```
(* INCORRECTO! *)
seguir:= TRUE;
while seguir do
begin
  readLn(numero);
  if numero > 0 then
    [procesar_numero]
  else
    seguir:= false
end;
```

Ver solución correcta en clase anterior.

## Errores con while (4)

“Forzar” entrada al while con un valor ficticio:

```
car:= '#'; (* valor ficticio *)
while car <> FIN do
begin
  ReadLn(car);
  case car of
    ... (* opciones *)
  end;
end;
```

Solución correcta: usar repeat o leer antes del while.

## Section 4

### Pragmática del repeat

## Cuándo usar repeat (1)

Lectura con centinela cuando este es un dato válido:

```
repeat
  ReadLn(dato);
  procesar_dato;
until es_centinela(dato);
```

La solución con while no es adecuada en este caso:

```
(* INCORRECTO! *)
ReadLn(dato);
while NOT es_centinela(dato) do
begin
  procesar_dato;
  ReadLn(dato)
end;
procesar_dato; (*centinela*)
```

## Cuándo usar repeat (2)

Tratamiento de un menú de opciones:

```
repeat
  ReadLn(opcion);
  case opcion of
    ....
  else
    WriteLn('Opcion Erronea');
until opcion = FIN;
```

## Cuándo usar repeat (3)

Ignorar datos de entrada hasta el primero que cumpla una condición:  
(ejemplo: saltar espacios)

```
repeat
  read(dato);
until cumple_condicion(dato);
```



## Errores con repeat

Son similares a los errores que pueden cometerse con `while`.

Observar que siempre se puede forzar un `repeat` para que se comporte como `while`:

```
(* INCORRECTO! *)
terminar := FALSE;
repeat
  if condicion then
  begin
    hacer_algo;
  end
else
  terminar:= TRUE;
until terminar;
```

```
(* INCORRECTO! *)
repeat
  if condicion then
  begin
    hacer_algo;
  end
until not condicion;
```

## General para todo tipo de repetición:

Un acumulador es una variable cuyo valor se calcula en forma incremental en los pasos de ejecución de una repetición. El ciclo de vida de un acumulador es:

- *Inicialización*: Antes de la repetición. Se le da un valor inicial a la variable  $v := \text{valor}$ .
- *Acumulación*: Dentro de la repetición. Actualización del valor del acumulador en función del valor anterior. En general tiene la forma  $v := f(v)$ .
- *Utilización del resultado*: Luego de la iteración.

Son errores habituales omitir o ubicar en lugar equivocado estos pasos.