Tema 2
Instrucciones estructuradas

Grado de Ingeniería Informática

Introducción a la Programación

Instrucciones estructuradas

2.1. Instrucciones compuestas

2.2. Instrucciones de selección

2.3. Instrucciones de iteración

2.4. Corrección y depuración de las instrucciones estructuradas

Objetivo

- Exponer al alumno la forma de controlar el orden de ejecución de las acciones del programa:
  - Control por **selección**
  - Control por **iteración**

2.1 Instrucciones compuestas

- Bloque de instrucciones o instrucción compuesta:
  - Empieza por **BEGIN**
  - Seguida de una secuencia de instrucciones separadas por el carácter ;
  - Finaliza con **END**
    - NOTA: no es necesario el **;** después de la última instrucción antes del END

Bloque de instrucciones

- Un bloque de instrucciones se usa en:
  - Instrucciones estructuradas:
    - De selección y de iteración
    - No tienen porqué formar una entidad lógica (para el programador)
  - Subprogramas:
    - Se usan para modelar subproblemas y sus soluciones algorítmicas en el marco del diseño descendente y del refinamiento sucesivo
    - Son acciones/cálculos lógicos (para el programador)

2.2 Instrucciones de selección

- La instrucción IF (SI... ENTONCES)
- La instrucción CASE (EN CASO DE…)

Instrucciones de selección

- Estructura de control de selección:
  - Sirve para cambiar el flujo de ejecución secuencial.
  - Permite elegir dinámicamente (en tiempo de ejecución) entre diferentes secuencias de instrucciones.
- Ejemplo:

SI apruebo el examen de junio ENTONCES
haré las maletas y
me iré de vacaciones
SI NO
me quedaré en casa y
estudiaré para el examen de septiembre

La instrucción SI … ENTONCES Pseudócodigo

- SELECCIÓN SIMPLE

  SI <condición> ENTONCES

  instrucción/es I

  FIN_SI

- SELECCIÓN MÚLTIPLE

  SI <condición> ENTONCES

  instruccion/es I

  SI_NO

  instrucción/es II

  FIN_SI

La instrucción IF ... THEN Pascal

- SELECCIÓN SIMPLE:
- Sintaxis:

```
IF <ExpresiónBooleana> THEN
    <Instrucción1>
```

- Semántica asociada:
  - Si la *ExpresionBooleana* devuelve un valor TRUE, se ejecuta la *Instrucción1.*
  - Si devuelve un valor FALSE no se ejecuta la *instrucción1*

Instrucción IF … THEN
Pascal. Ejemplo 1

- Utilidad:
  - Con la instrucción IF … THEN se **decide** si se ejecuta una instrucción o no
  - Con la instrucción IF … THEN … ELSE se **elige** entre dos instrucciones alternativas

```
BEGIN {bloque}
    readln(anyo);
    noDias := 365;
    IF anyo mod 4 = 0 THEN
        noDias := 366;
    writeln(anyo,'tiene ',noDias, 'dias')
END. {bloque}
```

La instrucción IF … THEN … ELSE Pascal

- SELECCIÓN MÚLTIPLE:
- Sintaxis:

```
IF <ExpresiónBooleana> THEN
    <Instrucción1>
ELSE
    <Instrucción2>
```

- Semántica asociada
  - Si la *ExpresionBooleana* devuelve un valor TRUE, se ejecuta la *Instrucción1.*
  - Si devuelve un valor FALSE se ejecuta la *instrucción2*

Instrucción IF. Ejemplo 1

```
...
BEGIN
   readln(x,y);
   IF x>y THEN
      max := x
   ELSE
      max := y;
   writeln('El máximo es ',max)

END.
```

Selección de secuencias de instrucciones

- Para elegir entre secuencias de instrucciones, hay que definirlas como un bloque
- Al igual que las instrucciones, dichos bloques han de sangrarse adecuadamente
- Un **;** antes de ELSE produce error

Instrucción IF. Ejemplo 2

```
BEGIN {bloque exterior}
   writeln('Introduzca dos números');
   readln(x,y);
   IF x > y THEN BEGIN {bloque interior1}
       max := x;
       writeln('x es mayor que y');
   END
   ELSE BEGIN {bloque interior2}
       max := y;
       writeln('y es mayor o igual que x');
   END;
END; {bloque exterior}
```

Errores comunes

- El sangrado facilita la lectura del programa *para el programador*
- *El compilador* ignora el sangrado

```
IF <cond.> THEN

   instrucción1;
   instrucción2;
   …
   instrucciónN
```

Se ejecuta como:

```
IF <cond.> THEN
instrucción1;
instrucción2;
…
instrucciónN
```

Errores comunes

- Poner un **;** antes de la parte ELSE

```
IF <cond.> THEN
   instrucción1;
ELSE
   instrucción2;
```

```
IF <cond.> THEN
BEGIN
      instrucción1;
      instrucción2;
END;
ELSE
      instrucciónN
```

ERRORES DE COMPILACIÓN

Recomendaciones

- Sangrar las instrucciones
- Documentar el fin del bloque con un comentario
- No repetir instrucciones comunes en las distintas ramas. (¿Qué ejemplo es preferible, 1 ó 2?)
- No utilizar instrucciones IF para la asignación de valores booleanos

Instrucción IF. Ejemplo recomendaciones

- En lugar de usar:

```
  IF (x>=0) and (x<10)
THEN
          B:=true
ELSE
          B:=false;
```

- Mejor emplear:

```
B:= (x>=0) and (x<10)
```

Alternativas múltiples

- Para elegir entre tres o más alternativas:
  - Anidar instrucciones IF
  - Usar la instrucción CASE

Instrucciones IF anidadas: Ejemplo 1

```
PROGRAM MaximoDeTres;
{ Pre: x,y,z ∈ N
{ Post: max es el máximo de x,y,z}
VAR
 x, z, y, max : integer;

BEGIN {PP}
   IF x>y THEN
       IF x>z THEN
              max:=x
       ELSE
              max:=z
   ELSE
       IF y>z THEN
              max:=y
       ELSE
              max:=z;
END. {PP}
```

Instrucciones IF anidadas: Ejemplo 2

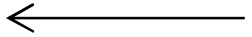
```
PROGRAM ListaNota;
VAR
   nota:integer;
BEGIN {ListaNota}
   readln(nota);
   IF (nota=1) THEN
       writeln('Aprobado')
   ELSE IF (nota=2) THEN
        writeln('Notable')
   ELSE IF (nota=3) THEN
        writeln('Sobresaliente')
   ELSE IF (nota=4) THEN
       writeln('Matricula de honor')
   ELSE
      writeln('La nota no es válida');
END. {ListaNota}
```

Instrucción CASE. Ejemplo 1

```
PROGRAM ListaNota;
VAR
   nota:integer;
BEGIN {ListaNota}
   readln(nota);
   CASE nota OF
        1:writeln('Aprobado');
        2:writeln('Notable');
        3:writeln('Sobresaliente');
        4:writeln('Matricula de honor')
        ELSE
          writeln('La nota no es válida');
   END; {CASE}
END; {ListaNota}
```

La instrucción EN CASO … SEA
Pseudocódigo

```
EN CASO <exp. selectora> SEA
<Valor1>: <Instrucción1>
<Valor2>: <Instrucción2>
. . .
<ValorN>: <InstrucciónN>
[SI_NO
<InstrucciónELSE> ]
FIN_CASO
```



La instrucción CASE … OF Pascal

- SINTAXIS:

CASE <exp. selectora> **OF**
<etiqueta1>: <Instrucción1>;
<etiqueta2>: <Instrucción2>;
. . .
<etiquetaN>: <InstrucciónN>;
[**ELSE**
<InstrucciónELSE>;]
END {CASE}

No existe en Pascal estándar y es opcional en TP

La instrucción CASE … OF Pascal

- SEMÁNTICA:
  1. Se evalúa la *ExpresiónSelectora*.
  2. Se compara con las etiquetas de forma ordenada, empezando con la primera.
     - Si la comparación es TRUE se ejecuta la *instrucción* asociada a esa etiqueta y termina la instrucción CASE
     - Si la comparación es FALSE, se compara con la siguiente etiqueta
  3. Si ninguna etiqueta coincide con la *ExpresiónSelectora*, se ejecuta la instrucción asociada a SI_NO (si existe)

La instrucción CASE

- **Observaciones**:
  - Expresión selectora: Tiene que ser de tipo ordinal
    - integer, char, boolean, enumerado.
  - Las etiquetas tienen que tener valor/es constante/s:
    - constantes, literales, listas de constantes o subintervalos
  - Las instrucciones: pueden ser una sola instrucción o un bloque

Instrucción CASE. Ejemplo 2

```
CASE opcion OF
  1..15: writeln('Primera Quincena');
     16: writeln('Primera Prueba');
 17..29: BEGIN {bloque}
           writeln('Esto es un
  bloque');
           writeln('Segunda Quincena')
          END; {bloque}
  30,31: writeln('Segunda Prueba')
 ELSE  writeln('valor erróneo');
END {CASE}
```

IF anidado o CASE

- Elegiremos:
- **CASE**:
  - Cuando la decisión dependen de una misma expresión selectora
- **IF anidado**
  - Cuando haya alternativas múltiples que dependen de selectores diferentes

Instrucciones IF anidadas. Ejemplo 3

- Ejemplo:
  - Decidir si se alquila un coche en función de dos variables selectoras:
    - La posesión del carnet de conducir
    - La edad
  - Regla:
    - No tiene carnet → no se alquila coche
    - Tiene carnet y es menor de 25 → se alquila a precio alto
    - Tiene carnet y es mayor de 25 → se alquila a precio bajo

Instrucciones IF anidadas. Ejemplo 3

```
...
IF tieneCarnet THEN
  IF edad <= 25 THEN
     {tienecarnet ∧ edad≤25}
     writeln('Precio alto')
  ELSE
     {tienecarnet ∧ ¬(edad≤25)}
     writeln('Precio bajo')
ELSE
  {¬ tienecarnet}
  writeln('No se alquila coche');
```

Normas de estilo

```
IF (Condición) THEN
BEGIN
   Instrucción1;
   ...;
   InstrucciónN
END    {IF}
ELSE
BEGIN
   InstrucciónA;
   ...;
   InstrucciónZ
END; {ELSE}
```

```
IF (Condición1) THEN BEGIN
   Instrucción1;
   ...;
   InstrucciónN
END    {IF}
ELSE IF (Condición2) THEN
BEGIN
   InstrucciónA;
    ...;
   InstrucciónZ
END {ELSE IF condición2}
ELSE BEGIN
   Instruccióna;
    ...
   Instrucciónz;
END; {ELSE}
```

Normas de estilo

```
CASE Expresión OF
    valor1: BEGIN
        Instrucción1;
        ...;
        InstrucciónN
    END; {Valor1}
     ...
    valorN: BEGIN
        InstrucciónA;
        ...;
        InstrucciónZ
    END; {ValorN}
    ELSE
        InstruccionElse;
END; {CASE}
```

3.3 Instrucciones de iteración

- DEFINICIONES:
  - Ciclo o bucle o iteración: Secuencia de instrucciones que se ejecutan repetidamente.
  - La iteraciones pueden ser controladas por las siguientes instrucciones:
    - MIENTRAS (WHILE)
    - REPETIR … HASTA (REPEAT ... UNTIL)
    - DESDE o PARA (FOR)

3.3 Instrucciones de iteración

- DEFINICIONES:
  - Cuerpo del bucle: Son las instrucciones que se encuentran dentro del bucle.
  - Variables controladoras: Son las variables de las que depende la ejecución del cuerpo del bucle.
    - Deben ser modificadas dentro del cuerpo del bucle.

MIENTRAS … HACER
Pseudocódigo

MIENTRAS (condición) HACER

    <instrucción/es>

FIN_MIENTRAS

La instrucción WHILE

- Sintaxis: Pascal

```
WHILE <expresión booleana> DO
    <instrucción>;
```

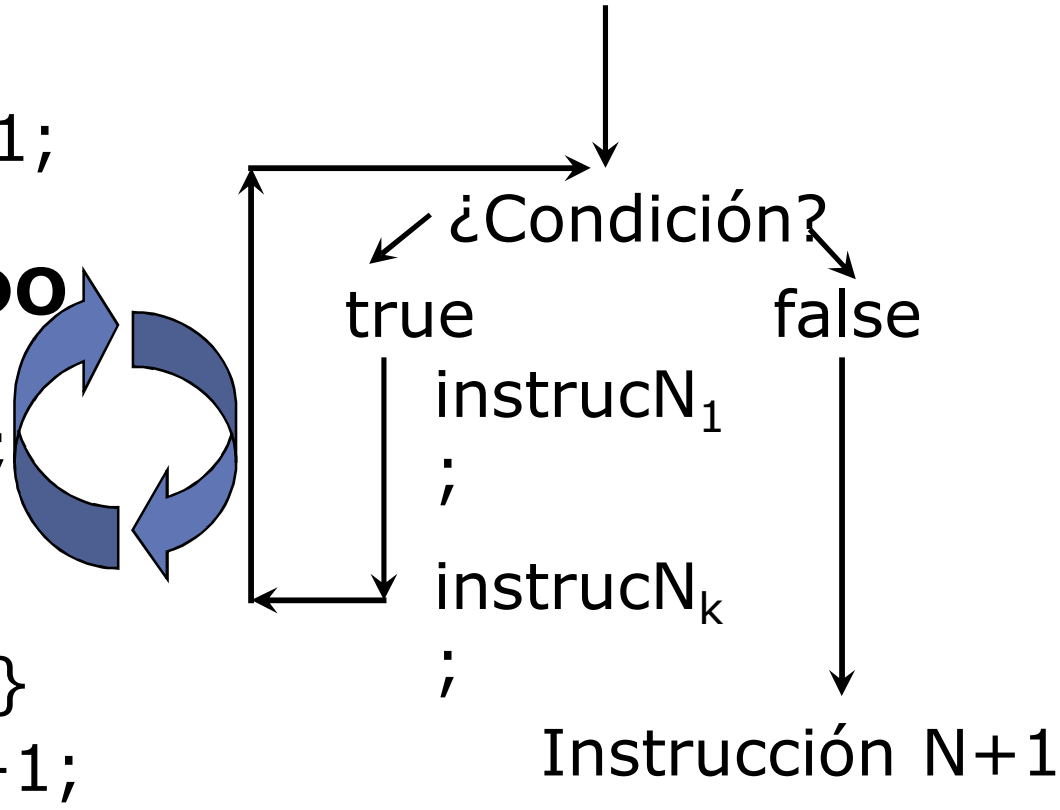
ó

```
WHILE <expresión booleana> DO
BEGIN
 <instrucción1>;
  <instrucción2>;
  …
END
```

La instrucción WHILE

- Semántica:

```
Instrucción N-1;
WHILE
<condición> DO
BEGIN
      instrucN1;
        . . .
      instrucNk
END; {WHILE}
Instrucción N+1;
```



Características y ejemplo

- La estructura WHILE modela ciclos **preprobados**. Si es falsa la condición no se ejecuta
- La estructura WHILE realiza un **número variable de iteraciones**: cero, una o varias
- Ejemplo:
  - Calcular $\sum i = 1+2+...+n$ utilizando WHILE

Instrucción WHILE. Ejemplo 1

```
...
BEGIN {bloque}
   readln(n);
   suma := 0;
   contador := 1;
   WHILE contador <= n DO BEGIN
      suma := suma + contador;
      contador := contador+1
      END; {WHILE}
   writeln(suma)
END {bloque}
```

Errores comunes

- Realizar ciclos infinitos
  - ¿Cómo se evitan? Modificando dentro del cuerpo la variable/s que lo controlan
- No inicializar las variables de control y de proceso del bucle
  - ¿Cómo se evita? Dándoles valor previamente
- Dejar el cuerpo del bucle vacío
  - ¿Cómo se evita? **Jamás** colocar un **;** detrás del DO, pues provocaría, además, un ciclo infinito

Instrucciones WHILE anidadas

- Es posible anidar estructuras WHILE:

WHILE <condición-1> **DO BEGIN**
    <instrucciones>;
    WHILE <condición-2> **DO BEGIN**
      <instrucciones>
    END; {WHILE interior}
    <instrucciones>
END {WHILE exterior}

REPETIR ... HASTA
Pseudocódigo

REPETIR

<instrucción1>

...

<instrucciónN>

HASTA <condición>

REPEAT ... UNTIL
Pascal

- Sintaxis:

REPEAT
    <instrucción1> ;
    ...
    <instrucciónN> ;
UNTIL <expresión booleana>

La instrucción REPEAT ... UNTIL

- Nótese:
  - Excepcionalmente, todas las instrucciones entre REPEAT y UNTIL se consideran un bloque de instrucciones, aunque no estén delimitadas por BEGIN y END

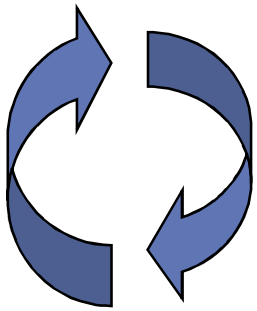
La instrucción REPEAT ... UNTIL

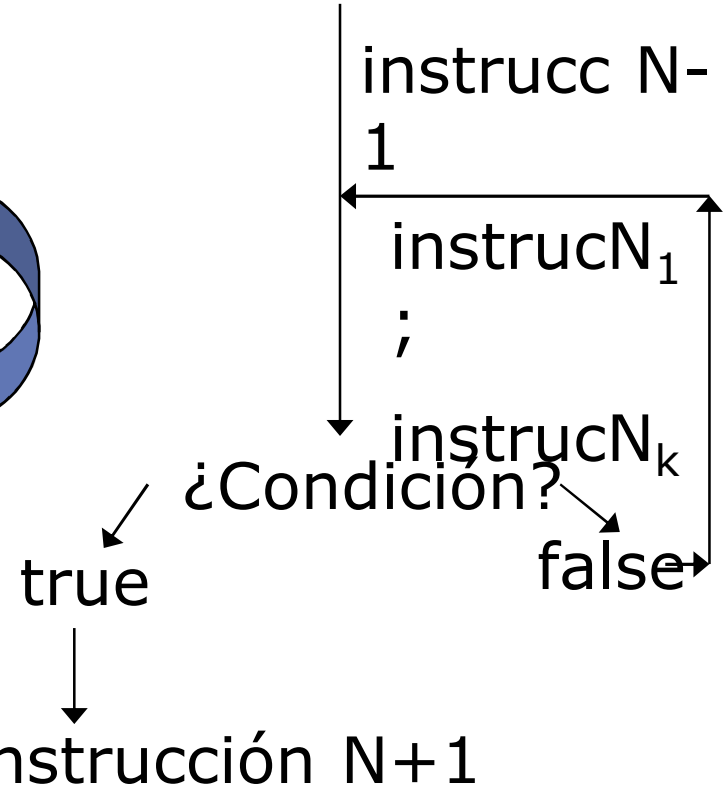
- Semántica:

```
instrucción N-1;

REPEAT
      instrucN1;
   . . .
      instrucNk
UNTIL <condición>;

instrucción N+1;
```





Instrucción REPEAT. Ejemplo 1

- **Lectura de datos** enteros asegurándose de que el número es positivo utilizando REPEAT

```
BEGIN {bloque para leer un entero positivo}
   REPEAT
      write('Introduzca un entero positivo:
       ');
      readln(numero)
   UNTIL numero > 0;
   {número > 0}
END; {bloque}
```

  - Permite validar los datos de entrada

Instrucción REPEAT. Ejemplo 2

- Realizar un programa que calcule $\sum i = 1+2+...+n$ utilizando REPEAT

```
BEGIN {bloque}
   readln(n);
   suma := 0;
   contador := 0;
   REPEAT
      suma := suma + contador;
      contador := contador+1
   UNTIL contador > n;
   writeln(suma)
END; {bloque}
```

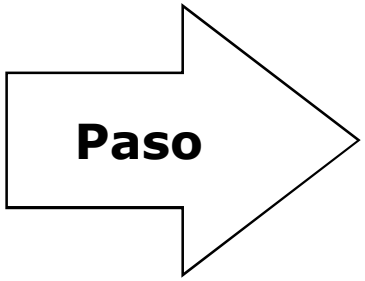
Características y ejemplos

- La estructura REPEAT modela ciclos **postprobados**. El cuerpo siempre se ejecuta
- La estructura REPEAT realiza un **número variable de iteraciones**
- Cualquier ciclo REPEAT puede traducirse en un ciclo WHILE
- Ejemplo:
  - Transformar un bucle REPEAT en un WHILE

Transformación de un bucle REPEAT en WHILE. Ejemplo

- Cualquier ciclo REPEAT puede traducirse en un ciclo WHILE

```
REPEAT
   I1;
   ...
   In
UNTIL C;
```



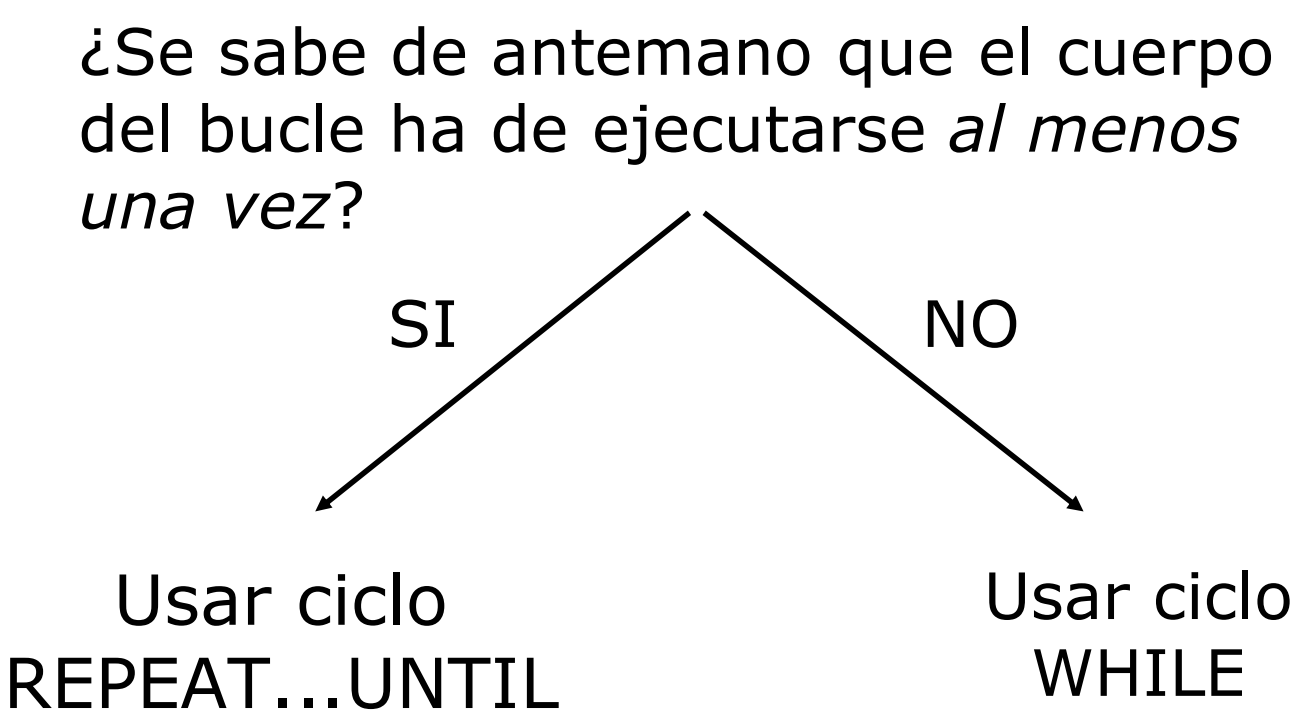

```
C := false;
WHILE not C DO
  BEGIN
   I1;
   ...
   In
   END; {WHILE}
```

Errores comunes

- Realizar ciclos infinitos
  - ¿Como se evitan? Modificando dentro del cuerpo la variable/s que lo controlan
- Pensar que puede no ejecutarse ninguna vez
  - Todas las operaciones durante la primera ejecución del cuerpo son válidas
  - La primera ejecución del cuerpo modifica las variables de la condición

WHILE y REPEAT ... UNTIL

- Recomendaciones técnicas (preliminares):



DESDE ... HASTA ... HACER
Pseudocódigo

```
DESDE índice:=<ValorInicial> HASTA
   <ValorFinal>
                              (con incremento +1 o -1)
   HACER
            <instrucción1>
            <instrucción2>
               …
FIN_DESDE
```

FOR … TO … DO
Pascal

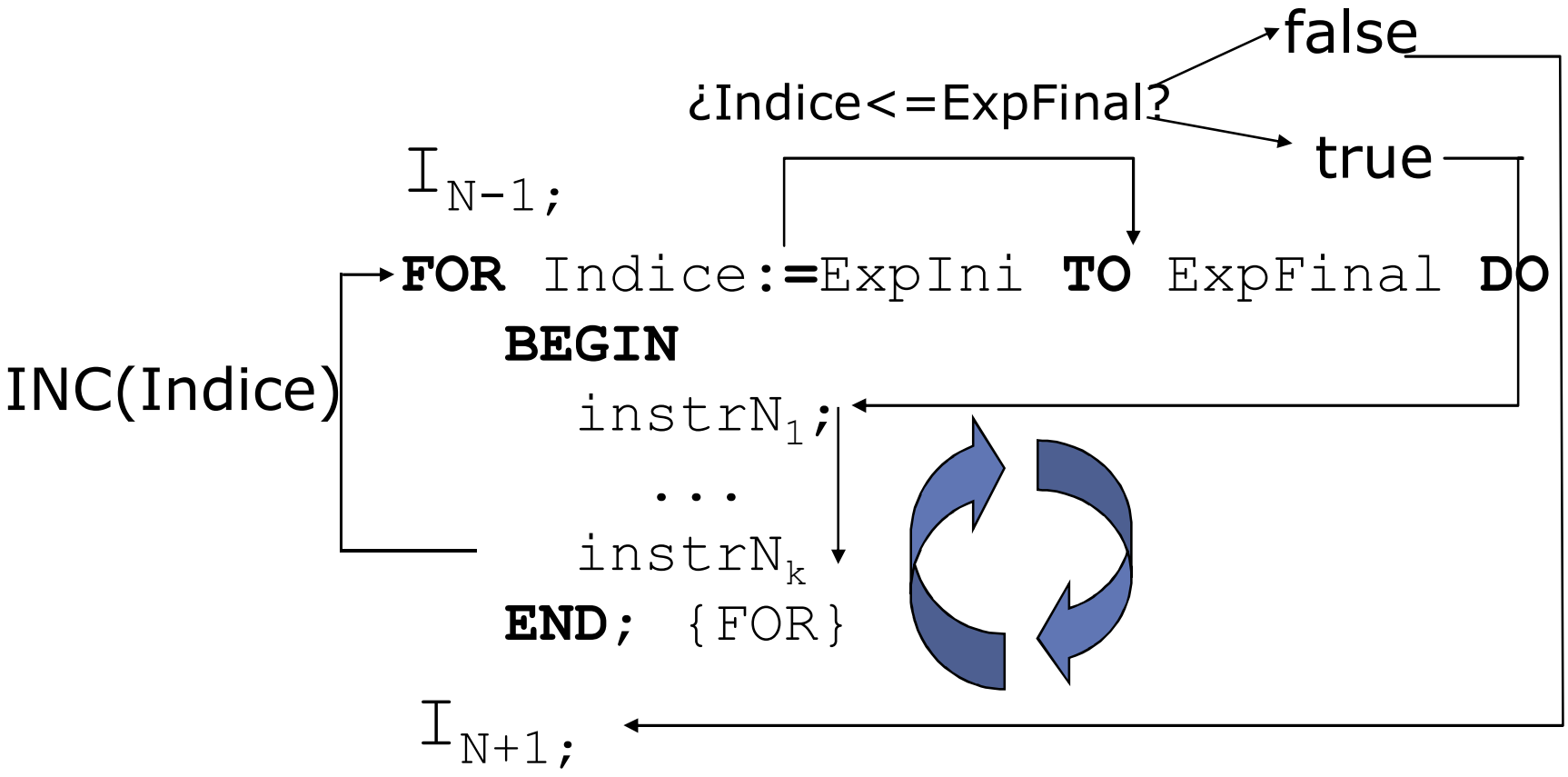
- Sintaxis (ciclo ascendente):

FOR indice**:=** <expr. inicial> **TO** <expr. final> **DO**
<instrucción>

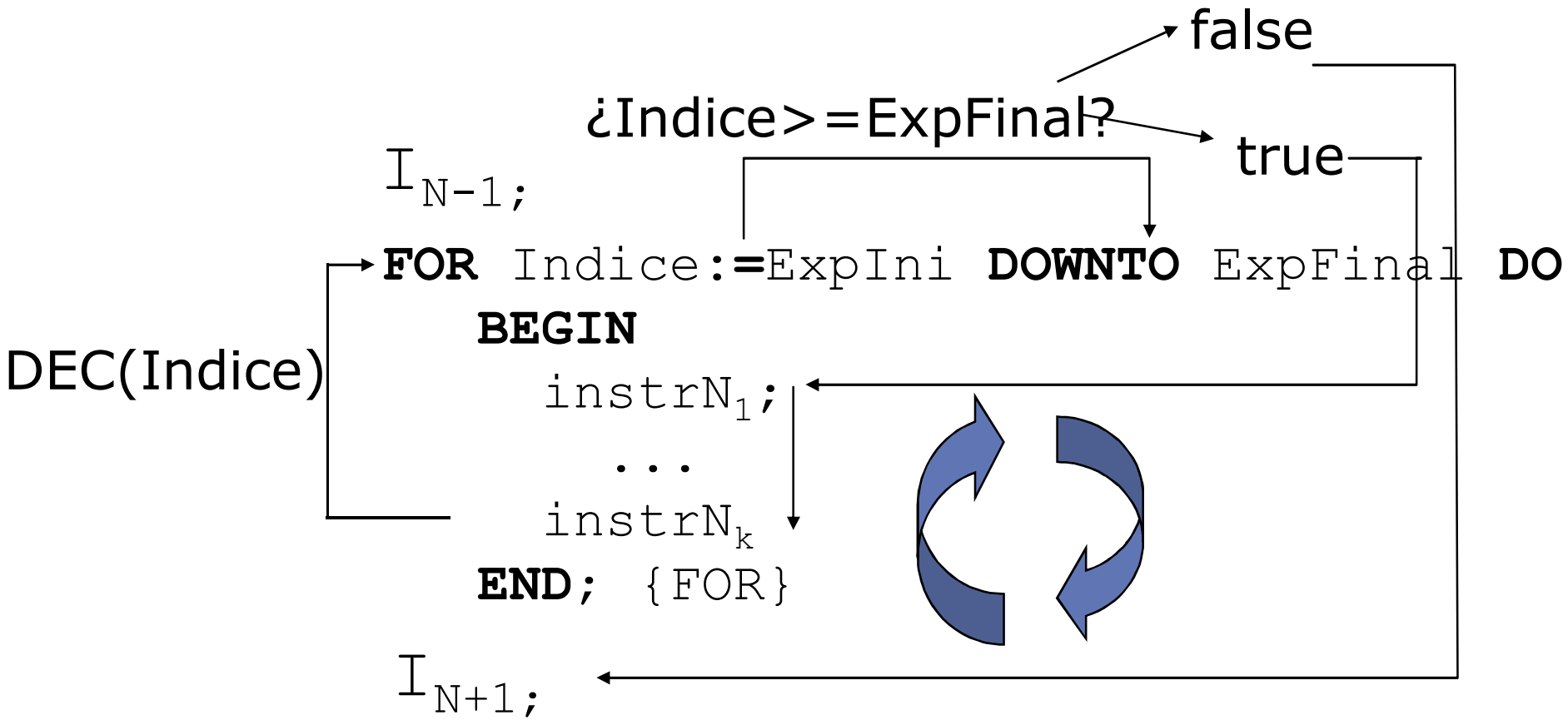
- Sintaxis (ciclo descendente):

FOR indice**:=** <expr. inic> **DOWNTO** <expr. fin> **DO**
<instrucción>

Semántica ciclo FOR ascendente



Semántica ciclo FOR descendente



Estructura FOR … TO … DO

- Ciclo *ascendente*
  - WHILE

```
indice:=expInicial;
WHILE indice<=expFinal DO BEGIN
    <instrucciones>;
     indice:=succ(indice);
END; {WHILE}
```

FOR equivalente

```
FOR indice:=expInicial TO expFinal
DO BEGIN
    <instrucciones>;
END; {FOR}
```

Estructura FOR … TO … DO

- Ciclo *descendente*
  - WHILE

```
indice:=expFinal;
WHILE indice>=expInicialDO BEGIN
    <instrucciones>;
    indice := pred(indice);
END; {WHILE}
```

  - FOR equivalente

```
FOR indice:=expFinal DOWNTO expInicial
DO BEGIN
    <instrucciones>;
END; {FOR}
```

Expresión inicial y final

- Las expresiones inicial y final cumplen que:
  - Son tipos compatibles con el tipo de índice
  - Se evalúan sólo una vez
  - Dependiendo de sus valores, puede no ejecutarse el cuerpo del bucle:
    - Si inicialmente *expFinal<expInicial* en ciclo ascendente
    - Si inicialmente e*xpFinal>expInicial* en ciclo descendente

  Recomendación importante:
  - No modificar su valor dentro del ciclo

Restricciones para la variable contadora

- El índice o contador:
  - Debe declararse como variable
  - Tiene que ser de tipo ordinal. No solo integer, puede ser también **char** o **boolean**. Ejemplo:

```
FOR c:= 'A' TO 'Z' DO
  write(c);
writeln;
```

- Restricciones para el índice:
  - No modificar dentro del cuerpo (read, asignación,…)
  - No usar como índice en un FOR interior
  - Al salir del ciclo FOR, suponemos que su valor está sin definir

Instrucción FOR. Ejemplo 1

- Ejemplo:

  Calcular $\sum i = 1+2+...+n$ utilizando FOR

```
BEGIN {bloque}
     readln(n);
     suma := 0;
       FOR contador := 1 TO n DO BEGIN
            suma := suma + contador;
         END; {FOR}
       writeln(suma)
END; {bloque}
```

Ejemplo de FOR anidados

- Ejemplo:
  - Leer dos números enteros positivos (n y m) y pedir n tandas de m números reales. Sacar por pantalla la media de cada tanda de números.

Errores comunes

- Errores comunes:
  - Modificar el contador
    - ¿Como se evita? No usando el contador dentro del bucle
  - Poner un **;** detrás del **DO**
    - Indicará cuerpo vacío, no hacerlo **jamás**
  - No delimitar cuerpo del bucle entre **BEGIN** y **END**

Limitaciones

- Limitaciones:
  - Los incrementos y decrementos solo pueden hacerse de uno en uno. Cuando es preciso usar más, la alternativa es WHILE.
  - Ejemplo: enumerar los números impares hasta *k*

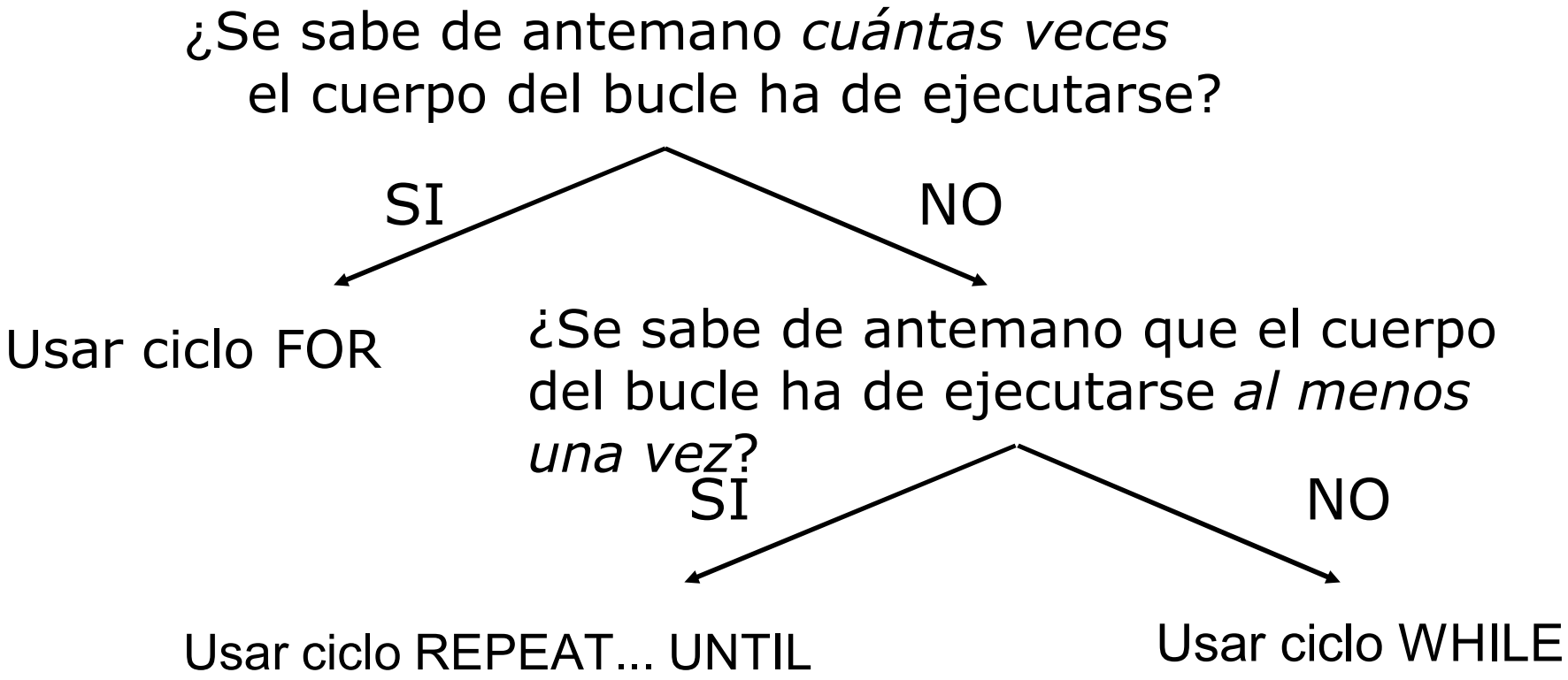
```
i := 1;
WHILE i =< k DO BEGIN
   writeln(i);
   i:=i+2;
END; {WHILE}
```

Características del FOR

- La estructura FOR modela ciclos **preprobados**
- La estructura FOR realiza un **número fijo de iteraciones**

WHILE, REPEAT y FOR

- Recomendaciones técnicas:



Normas de estilo

```
 WHILE Condicion
DO
   BEGIN
       Instrucción1;
       ...;
       InstrucciónN
   END{WHILE}
```

```
WHILE Condicion DO
BEGIN
       Instrucción1;
       ...;
       InstrucciónN
END    {WHILE}
```

Normas de estilo

```
FOR indice:=expInic TO expFinal DO
BEGIN
   Instrucción1;
   ...;
   InstrucciónN
END; {FOR}
```

```
FOR indice:=expInic DOWNTO
expFinal DO
BEGIN
   Instrucción1;
   ...;
   InstrucciónN
END; {FOR}
```

Normas de estilo

```
REPEAT
    Instrucción1;
    ...;
    InstrucciónN
UNTIL Condición;
```

Esquematización de algoritmos iterativos

- Diferentes estrategias de diseño ➔ diferentes algoritmos iterativos (esquemas):
  - Esquema de recorrido:
    - Procesar todos los elementos
    - Saber o no a priori el numero de elementos a procesar ➔ determina la elección de While, Repeat o For
  - Esquema de búsqueda:
    - Localizar un elemento con una característica
    - Basta con encontrar al primero

Esquema de recorrido. Ejemplos

- Ejemplo 1:
  - Contar las apariciones de la letra ‘a’ en una secuencia de caracteres acabada en ‘.’

```
BEGIN {bloque}
      contador:= 0;
      WHILE car <> '.' DO BEGIN
        IF car = 'a' then
             contador := contador + 1;
        read(car);
       END; {WHILE}
      writeln(contador);
END {bloque}
```

Esquema de recorrido. Ejemplos

- Ejemplo 2:
  - El número e se define como la suma de la siguiente serie: 1/1! + 1/2! + 1/3! + ..., se desea obtener el valor de la serie para n términos

```
BEGIN {bloque}
      readln(n);
      serie := 0;
      fact :=1;
      FOR i:=1 TO n DO BEGIN
        fact := fact * i;
        serie := serie + 1/fact
       END; {FOR}
      writeln(serie);
END {bloque}
```

Esquema de recorrido

- Esquema de recorrido sin saber a priori número de elementos que se deben procesar

```
obtener/calcular  primer elemento
iniciar tratamiento
   WHILE no último elemento DO BEGIN
      tratar elemento
      obtener/calcular siguiente elemento
   END;
   tratamiento final
```

Esquema de recorrido

- Esquema de recorrido si sabemos a priori el numero elementos (iteraciones)

```
iniciar tratamiento
FOR primer elemento TO ultimo elemento DO
BEGIN
    tratar elemento
  END;
  tratamiento final
```

Esquema de búsqueda. Ejemplo

- Ejemplo 1:
  - Determinar si aparece la letra ‘a’ en una secuencia de caracteres acabada en ‘.’

```
BEGIN {bloque}
   WHILE (car <> '.') AND (car <> 'a')
   DO
      read(car);
      esta_presente := car = 'a';
END {bloque}
```

Esquema de búsqueda

- Esquema de búsqueda:

```
obtener/calcular posible primer
  elemento
iniciar tratamiento
WHILE no último elemento y no
  satisface A DO
  obtener/calcular siguiente elemento
determinar si encontrado
```

Esquema mixto. Ejemplos

- Utilizando ambos esquemas obtenemos esquema mixto
- Ejemplo 1:Contar el número de caracteres anteriores a la primera aparición de la letra ‘a’ en una secuencia de caracteres acabada en ‘.’

```
BEGIN {bloque}
      num_car:= 0;
      WHILE (car <> '.') AND (car <> 'a') DO
    BEGIN
        num_car:= num_car + 1;
        read(car);
     END; {WHILE}
      writeln(num_car);
END {bloque}
```

Esquema mixto. Ejemplos

- Ejemplo 2:
  - El número e se define como la suma de la siguiente serie: 1/1! + 1/2! + 1/3! + ... Escribe un bloque de programa que pida un umbral de error y aproxime el número e hasta que la diferencia entre dos aproximaciones sucesivas sea menor que el umbral.

Esquema mixto. Ejemplos

```
BEGIN {bloque}
      readln(error);
      aprox_ant:= 0;
      aprox_sig:= 1;
      fact:=1;
      i:=1;
      WHILE (aprox_sig-aprox_ant)>= error DO BEGIN
        i:= i + 1;
        fact:= fact * i;
        aprox_ant:= aprox_sig;
        aprox_sig:= aprox_ant + 1/fact
       END; {WHILE}
      writeln(error, aprox_sig);
END {bloque}
```

3.4. Depuración

- A la hora de depurar instrucciones estructuradas, puede hacerse de varias maneras:
  - Como si se tratase de una única instrucción primitiva
  - Como un conjunto de subinstrucciones. En este caso, hay que distinguir:
    - En las de selección: las instrucciones de cada una de las ramas a ejecutar
    - En las de iteración: las instrucciones del cuerpo del ciclo

ascension.lovillo@urjc.es