

RESOLUCIÓN DE PROBLEMAS Y ALGORITMOS

CLASE 7

Repetición condicional

0 1 1 0 0
 1 0 0 1 1
 1 0 1 1 0
 0 1 1 1 0
 0 1 1 0 0
 1 0 0 1 1
 1 0 1 1 0
 0 1 1 1 0
 1 0 0 1 1
 1 1 1
 0 0
 1

Luciano H. Tamargo
 http://cs.uns.edu.ar/~lt
 Depto. de Ciencias e Ingeniería de la Computación
 Universidad Nacional del Sur, Bahía Blanca
 2016

MOTIVACIÓN

- Existen algoritmos donde:
 - se debe repetir una secuencia de acciones, pero
 - no se sabe de antemano cuantas veces** se van a repetir.
- Por ejemplo:

Algoritmo: cursada
 Repetir
 intentar cursar materia
 hasta materia cursada

⇨
 ↓

Algoritmo: envasar productos
 Repetir mientras hay productos

 - tomar producto
 - esperar por envase vacío
 - poner producto en envase
 - cerrar envase

Resolución de Problemas y Algoritmos - 2016 2

CONCEPTOS: SENTENCIAS REPETITIVAS EN PASCAL

- Repetición **incondicional**:


```
FOR var := <ini> TO <fin> DO
  <sentencia>
FOR var := <ini> DOWNTO <fin> DO
  <sentencia>
```
- Repetición **condicional** (depende de una condición):

WHILE <condición> DO
 <sentencia>

REPEAT
 <sentencias>
 UNTIL <condición>
- Importante: consultar los diagramas sintácticos de Pascal ([aquí](#)) para conocer los detalles de la sintaxis de todas estas sentencias.

Resolución de Problemas y Algoritmos - 2016 3

REPETICIÓN BASADA EN CONDICIONES

- Por ejemplo, considere que se quiere validar el ingreso de datos y se quiere **repetir el ingreso de datos mientras** estos no sean correctos.


```
mostrar('Ingrese una letra mayúscula')
leer(letra) {validación de los datos ingresados por el usuario}
MIENTRAS (letra < 'A') o (letra > 'Z') {i.e. ,no sea mayúscula}
  mostrar('Error, ingrese una letra mayúscula')
  leer(letra)
FIN REPETIR
{... Datos validados: si llega a este punto es porque
letra tiene una mayúscula ...}
```

Resolución de Problemas y Algoritmos - 2016 4

REPETICIÓN CONDICIONAL EN PASCAL: WHILE

- La **sentencia** de un ciclo **WHILE** se ejecutará **0 (cero) o más veces** dependiendo del resultado (true o false) que se obtiene al evaluar la expresión booleana que representa la condición.

```

    sentencia WHILE → (while) → [expresión] → (do) → [proposición]
    
```

Resolución de Problemas y Algoritmos - 2016 5

REPETICIÓN CONDICIONAL EN PASCAL: WHILE

Primero se evalúa la expresión booleana

```
WHILE [expresión booleana] DO
  <sentencia>
Otra sentencia siguiente;
```

TRUE Si el resultado es TRUE se ejecuta la **sentencia** que sigue al **DO**

Una vez ejecutada la **sentencia** que sigue al **DO**, se vuelve a evaluar la expresión booleana

FALSE Si el resultado es FALSE, saltea (no ejecuta) la **sentencia** del **DO** y sigue en la siguiente al **WHILE**.

Resolución de Problemas y Algoritmos - 2016 6

Resolución de Problemas y Algoritmos

"WHILE LOOP": REPETICION CONDICIONAL

- Las sentencias dentro de un **WHILE** se ejecutan **0 (cero) o más veces**.

Casos de prueba:
tope con -1, 0, 2

Obs.: si la sentencia del while se repitió N veces, la expresión `cont < tope` se evaluó N+1 veces.

Mientras `cont < tope` sea **true** ejecuta:

Mientras `cont < tope` sea **false** sigue en:

```
PROGRAMA ejemplo;
VAR tope, cont: integer;
BEGIN
  Write('Ingre un tope: ');
  readln(tope);
  cont := 0;
  WHILE cont < tope DO
  BEGIN
    writeln(cont);
    cont:=cont+1;
  END;
  Writeln('press enter');
  readln;
END.
```

tope	cont	cont<tope
-1	0	?
0	0	?
2	0	?

Resolución de Problemas y Algoritmos - 2016

EJEMPLO DE APLICACIÓN DE WHILE

- Validar el ingreso de datos y **repetir el ingreso de datos mientras estos no sean correctos**.

```
PROGRAM EjemploWhile;
{muestra una aplicación útil del WHILE}
VAR letra: char;
BEGIN
  writeln('Ingrese una letra mayúscula');
  readln(letra);
  {validación de los datos ingresados por el usuario}
  WHILE (letra<'A')OR(letra >'Z') DO
  BEGIN
    writeln('Incorrecto. Ingrese letra mayúscula');
    read(letra);
  END;
  {...aquí estoy seguro que letra tiene una mayúscula y continúo con el resto del programa...}
```

Casos de prueba
(letra < 'A') OR (letra > 'Z')

letra	¿repite?
A	FALSE no
a	TRUE sí
N	FALSE no
n	TRUE sí
\$	TRUE sí
1	TRUE sí

Resolución de Problemas y Algoritmos - 2016

EJEMPLO DE APLICACIÓN DE WHILE

- Validar el ingreso de datos y **repetir el ingreso de datos mientras estos no sean correctos**.

```
PROGRAM EjemploWhile;
{muestra una aplicación útil del WHILE}
VAR opcion:integer;
BEGIN
  writeln('Ingrese opción:
  (1) pasar a minúscula
  (2) mostrar código ASCII ');
  read(opcion);
  {validación de los datos ingresados por el usuario}
  WHILE (opcion<>1) and (opcion<>2) DO
  BEGIN
    write(' Incorrecto. Ingrese 1 o 2 ');
    read(opcion);
  END
  {... resto del programa...}
```

Casos de prueba
(opcion<>1) and (opcion<>2)

opcion	¿repite?
1	FALSE no
0	TRUE sí
2	FALSE no
5	TRUE sí

Resolución de Problemas y Algoritmos - 2016

REPETICION BASADA EN CONDICIONES

- Por ejemplo, se quiere permitir al usuario **repetir la ejecución del programa hasta que el decida**.

REPETIR
{ esta parte del programa se repetirá hasta que el usuario indique fin }
mostrar ("Pulse la letra (S) para ejecutar nuevamente o cualquier otra letra para salir de la aplicación")
leer (letra)
HASTA que se pulse letra distinta de 'S';
mostrar ("Muchas gracias por utilizar el programa")

Resolución de Problemas y Algoritmos - 2016

REPETICION CONDICIONAL EN PASCAL

- Las sentencias dentro de un **REPEAT-UNTIL** se ejecutan **1 o más veces** dependiendo del resultado (true o false) que se obtiene al evaluar la expresión booleana que representa la condición.

sentencia REPEAT → repeat → proposición → until → expresión

```
REPEAT
<sentencia 1>
<sentencia 2>
...
<sentencia n>
UNTIL <expresión booleana>
<sentencia siguiente>
```

Si el resultado es **FALSE** vuelve a ejecutar la secuencia a partir de la palabra reservada **REPEAT**

Si el resultado es **TRUE** no vuelve a repetir y sigue en la sentencia siguiente a **UNTIL**

Resolución de Problemas y Algoritmos - 2016

"REPEAT LOOP": REPETICION CONDICIONAL

```
PROGRAMA ejemplo2;
VAR tope, cont: integer;
BEGIN
  Write('Ingre un tope: ');
  readln(tope);
  cont := 0;
  REPEAT
  writeln(cont);
  cont := cont + 1;
  UNTIL cont >= tope;
  Writeln('press enter');
  readln;
END.
```

Las sentencias dentro de un **REPEAT** se ejecutan **1 (una) o más veces**.

Si el resultado es **false** vuelve a repetir desde

Si el resultado es **true** sigue en

Casos de prueba:
tope con -1, 0, 2

tope	cont	Cont>=tope
-1	0	?
0	0	?
2	0	?

Resolución de Problemas y Algoritmos - 2016

EJEMPLO CON REPEAT-UNTIL

```

PROGRAM EjemploRepeat;
{muestra una aplicación útil del repeat}
VAR letra: char;
BEGIN
REPEAT
{esta parte del programa se repetirá hasta
que el usuario lo indique}
writeln('Pulse la letra (S) para
ejecutar nuevamente o cualquier otra
letra para salir de la aplicación');
readln(letra);
{se volverá a repetir si presiona la
tecla S }
UNTIL (letra<>'S') and (letra<>'s');
writeln('Muchas gracias por utilizar el
programa. ');
END.
    
```

Casos de prueba
(letra <> 'S') OR (letra <> 's')

letra	¿repite?
S	FALSE sí
s	FALSE sí
N	TRUE no
n	TRUE no
\$	TRUE no
1	TRUE no

CONCEPTOS: DIFERENCIAS REPEAT Y WHILE

REPEAT-UNTIL	WHILE
si condición es falsa sigue repitiendo.	si condición es verdadera sigue repitiendo.
si condición es verdadera deja de repetir.	si condición es falsa deja de repetir.
repite 1 o más veces: siempre ejecuta al menos una vez la secuencia interna al repetir.	repite 0 o más veces: puede no ejecutar nunca la secuencia interna al repetir.

- Ejercicio propuesto para practicar:** escriba las diferencias y similitudes entre las tres sentencias repetitivas FOR, WHILE y REPEAT.

REPETICIONES ANIDADAS (ALGUNOS EJEMPLOS)

```

REPEAT
  WHILE <condición> DO
    WHILE <condición> DO
      <sentencia>
    UNTIL <condición>
  UNTIL <condición>
    
```

```

WHILE <condición> DO
  FOR v:=... TO ... DO
    <sentencia>
  
```

```

REPEAT
  REPEAT
    <sentencia>
  UNTIL <condición>
UNTIL <condición>
    
```

El límite está en la imaginación del programador

CONCEPTOS: REPETICIÓN CONDICIONAL VS. INCONDICIONAL

- La repetición **condicional** (REPEAT o WHILE) depende de una condición (expresión boolean).
- La repetición **incondicional** (FOR) se ejecuta un **número fijo** de veces que se conoce antes de comenzar a repetirse la sentencia.
- Toda vez** que se puede usar una repetición **incondicional** (FOR), el código podría **reescribirse** para usar una repetición condicional (WHILE o REPEAT).
- Pero **no todas** las repeticiones **condicionales** (WHILE o REPEAT) **pueden reescribirse** para usar una incondicional (FOR).

REPETICIÓN CONDICIONAL ES MAS GENERAL

```

a:=1;
FOR v:=1 TO 5 DO
  a := a * v;
write(a);
    
```

```

a:=1;
v:=1;
REPEAT
  a := a*v;
  v := v+1;
UNTIL v > 5
write(a);
    
```

```

a:=1;
v:=1;
WHILE v<=5 DO
  BEGIN
    a := a * v;
    v := v + 1;
  END
write(a);
    
```

```

write('ingrese nro. positivo:');
read(num);
WHILE num < 0 DO
  Read(num);
    
```

```

Write('ingrese nro. positivo:');
REPEAT
  Read(num);
UNTIL num >= 0;
    
```

FOR ? TO ? DO

CONCEPTO: CICLO INFINITO

Una repetición que se realiza infinitas veces se denomina **ciclo infinito**.

- En RPA un ciclo infinito en un programa será considerado un **ERROR GRAVE** de programación.
- Cuando realice la traza de sus programas **debe asegurarse** que en las repeticiones **no exista ningún caso** en el cual el programa pueda **caer en un ciclo infinito**.

CICLO INFINITO

CONCEPTOS: CICLOS INFINITOS

- Una repetición condicional mal programada puede caer en una **REPETICIÓN INFINITA**, lo cual es un error grave de programación.

BIEN	MAL	MALDO	MAL
<pre>v:=1; w:=1; REPEAT v:=v+1; UNTIL v=3; Write (v);</pre>	<pre>v:=1; w:=1; REPEAT v:=v+1; UNTIL w=0; Write (v);</pre>	<pre>v:=1; w:=1; WHILE v<>3 v:=1; Write (v);</pre>	<pre>v:=1; w:=1; REPEAT v:= w-1; UNTIL v=3; Write (v);</pre>

- Algunos problemas clásicos:**
 - La condición de corte es errónea.
 - La variable de la condición no se modifica.
 - La variable de la condición se modifica mal.

Resolución de Problemas y Algoritmos - 2016 19

PROBLEMA PROPUESTO

- Escriba un programa que calcule el promedio de números reales ingresados por el usuario.

Resolución de Problemas y Algoritmos - 2016 20

SUCESIÓN DE FIBONACCI

- La sucesión de Fibonacci es una sucesión infinita de números naturales que inicia con 0 y 1, y a partir de ahí cada elemento es la suma de los dos anteriores:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
- Cada elemento de esta sucesión se llama número de Fibonacci.

```
anterior ← 0
mostrar(0)
ultimo ← 1
mostrar(1)
repetir
  nuevo ← ultimo + anterior
  mostrar(nuevo)
  anterior ← ultimo
  ultimo ← nuevo
hasta nuevo > tope
```

- La sucesión fue descrita por Fibonacci, en su libro *Liber Abaci*, como la solución a un problema de la cría de conejos.
- Antes de que Fibonacci escribiera su trabajo, la sucesión de los números de Fibonacci había sido descubierta por matemáticos indios tales como Gopala (antes de 1135) y Hemachandra (1150),

Resolución de Problemas y Algoritmos - 2016 21