INTRODUCCIÓN A LA PROGRAMACIÓN
PRÁCTICA 5

Objetivos

- ✓ Introducir las sentencias de composición iterativa de Pascal y su equivalencia con las composiciones MIENTRAS, REPETIR, ITERAR y RECORRIENDO vistas en clase de teoría junto con algunos ejemplos de su uso.
- ✓ Visualizar una "iteración" como el recorrido de una secuencia de valores.
- ✓ Estudiar la utilización de todo lo anterior en programas Pascal resolviendo problemas sencillos de recorrido y búsqueda en secuencias y cálculo de sucesiones y series.

1. Las composiciones iterativas en Pascal

Excepto la composición ITERAR, las demás composiciones iterativas de la notación empleada en clase tienen una instrucción equivalente en Pascal. La tabla de la figura muestra la correspondencia, suponiendo que cond denota una expresión booleana, S, S_1 y S_2 denotan una secuencia de instrucciones y, e_1 y e_2 son expresiones cuyo tipo es un subrango de un tipo ordinal. La semántica de la composición ITERAR debe obtenerse a través del uso de las composiciones WHILE o REPEAT. La estructura iterativa de Pascal equivalente a la composición RECORRIENDO estudiada en clase de teoría es la composición FOR.

Notación	Pascal

MIENTRAS cond **HACER**
S
FIN_MIENTRAS
	while cond **do**
	begin
	S
	end;

REPETIR S **HASTA_QUE** cond	**repeat** S **until** cond

ITERAR
S_1
DETENER cond
S_2
FIN_ITERAR
	No incluida en el lenguaje. Hay que expresarla mediante su equivalencia con iteraciones **while** o **repeat** (preferiblemente **while**):
	
	S1;
	while not cond **do**
	begin
	S2;
	S1
	end;

i RECORRIENDO [a, b] HACER
S;
FIN_RECORRIENDO
	for i := a **to** b **do**
	begin
	S;
	end;

i RECORRIENDO [a, b] EN SENTIDO INVERSO HACER
S;
FIN_RECORRIENDO
	for i := b **downto** a **do**
	begin
	S;
	end

NOTA: Cuando el cuerpo de las iteraciones **while** y **for** consta únicamente de una sentencia, ésta puede ir sola, sin necesidad de estar encerrada entre un par **begin-end**.

Las reglas de la composición **for** son similares a las empleadas en seudocodigo para la instrucción **recorriendo**.

- La variable de control, **i**, el valor inicial, **a**, y el valor final, **b**, deben ser del mismo tipo, que será de un tipo ordinal, ya que la variable **i** se incrementa o decrementa en una unidad de forma automática.
- La variable de control, variable **i,** debe estar declarada en el ámbito donde es utilizada (donde está el **for**).
- Las instrucciones del cuerpo del bucle pueden utilizar el valor de la variable de control **i**, pero no pueden modificar este valor
- Cuando se completa la ejecución de la instrucción **for**, la variable de control queda indefinida.
- No debe suponerse la reevaluación de los límites inferior y superior de la iteración, caso de ser expresiones en las que intervengan variables. De hecho, conviene que no varíen.

Ejemplo 1: Diseñe un programa que dada una línea de texto introducida por teclado cuente el número de blancos.

```
program ContarBlancos;
Var
        cont : integer;         { contador de blancos }
        c: char;                  {  carácter actual}
Begin
        cont := 0;
        while not eoln do       { fin de línea de la entrada de datos }
        begin
                read(c);
                if c = '   ' then cont := cont + 1;
        end;
        write ('El número de blancos es: ', cont );
  End.
```

Ejemplo 2: Diseñe un programa que calcule el número de términos necesarios de la serie armónica de forma que su suma sea mayor que un valor determinado:

1+1/2+1/3+1/4+…+ 1/n > limite

```
program serie_A;
        { En este programa el límite es una constante incorporada en el programa. }
        { También podría ser un valor cualquiera leído de la entrada. }
const
        limite= 5.0;
Var
        suma :  real;           { valor de la suma de términos en un momento determinado }
        cuenta: integer;        { cuenta el número de términos y lo utilizamos de divisor }
Begin
        cuenta:=0;
        suma:=0;

        repeat
                cuenta:=cuenta+1;
                suma:=suma + 1/cuenta;
        until suma>limite;
        writeln ('El número de términos es ', cuenta);
End.
```

Ejemplo 3: Observe los siguientes programas. Intente predecir sin ejecutarlos la salida que producirían. Ejecútelos después y asegúrese que comprende el funcionamiento de la instrucción **for**.

```
program entero;
var
                i : integer;
                linf, lsup : integer;

begin
  for i := 1 to 10 do
                writeln (i);
  for i:= 1 to 10 do
                writeln ('1');

  writeln ('Inroduzca el límite inferior y límite superior');
  readln (linf, lsup);

  for i:= linf-3 to lsup*2 do
                writeln (i)
end.
```

```
program anidado;
var
    i : char;
    j : integer;
begin
    for j := 1 to 10 do
        for i := 'a' to 'z' do
            writeln (j, i)
end.
```

Ejemplo 4: En clase de teoría se ha estudiado una primera versión del algoritmo que dados dos enteros positivos **x** e **y**, 0<x<y, calcula y muestra en pantalla las potencias de 2 en

el intervalo [x, y]: 2^x ,2^{x+1}, ..., 2^y. Implemente en Pascal la versión mejorada que evita la función potencia.

```
program potencias;
Var
        x, y, i, p: integer;
Begin
        readln (x, y);
        p := 1;
        for i:= 1 to x do
                p :=  p*2;
        writeln (p);

        for i:= x+1 to y do
        begin
                p :=  p*2;
                writeln (p)
        end
end.
```

Problema 1.

- Resuelve el ejemplo uno utilizando la composición Repeat.
- Resuelve el ejemplo dos utilizando la composición While.

¿Es necesario utilizar otras composiciones de control para ajustar la semántica?

Problema 2. Supongamos el problema de la práctica tres, en la que realizábamos la codificación de un carácter introducido por teclado. ¿Cómo resolvería el problema si en lugar de un carácter tuviese que codificar una frase? Puede tomar como referencia el Ejemplo 1 en el que se muestra cómo hacer una lectura carácter a carácter de una línea de la entrada.

Problema 3:

Dados dos enteros positivos **x, y** ((x=X ≥ 0) Y (y=Y > 0) calcule el valor del cociente, **c**, y del resto, **r**, de la división entera, **x DIV y**, mediante restas sucesivas. Cuando el algoritmo finalice se cumplirá:

- El dividendo es igual al divisor por el cociente más el resto (x=y*c+r)
- La variable **r** contendrá el valor del resto que será mayor o igual a cero y menor que el valor Y
- La variable c, contendrá el valor del cociente de la división entera

- ¿Cuáles son los valores iniciales de las variable **c** y **r**?
- En cada paso de la iteración ¿Cómo se actualizan los valores de las variables?
- ¿Cuál es la condición de finalización?

Problema 4:

Se define la siguiente sucesión por recurrencia:

$a_0=1$
$a_1=2$
$a_2=3$
$a_n=3a_{n-1} - 2a_{n-3}$ n > 2

a) Escriba un algoritmo que calcule el valor de n más pequeño tal que a_n es superior a un entero $m \geq 0$ dado como dato de entrada al problema.
b) Escriba un algoritmo que calcule el término n-ésimo de la sucesión a_n, donde n es un valor entero no negativo introducido por teclado.

Trabajo personal del alumno

Problema 1

Diseñe un algoritmo que determine si un entero **a > 0** es primo. Mejore de la solución estudiada en clase teniendo en cuenta dos aspectos:

- Los números pares no son primos.
- Si un número a no tiene divisores por debajo de su raíz cuadrada entonces no los tiene tampoco por encima de este valor.

Problema 2

Desarrolle un algoritmo para sumar la serie:

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

Solucione este problema con dos enfoques diferentes:

1. Construya una función que calcule la **factorial** de un número entero **n** y otra que calcule $\mathbf{x^y}$, siendo **n**, **x** e **y** parámetros de entrada. Utilice estas dos funciones en una iteración para calcular la suma de la serie que se propone.
2. Cada término de la serie se puede calcular en función del termino anterior. No es necesario utilizar las dos funciones mencionadas en el apartado anterior. Si por ejemplo el término actual es $\frac{1 \cdot x \cdot x}{1 \cdot 2}$ y el termino siguiente $\frac{1 \cdot x \cdot x \cdot x}{1 \cdot 2 \cdot 3}$, ¿qué se podría hacer en la iteración para obtener este término en función del anterior sin calcular todo el término completo?

- En cualquiera de los dos casos, ¿Cómo finaliza la iteración? Diseñe el algoritmo de forma que calcule la suma hasta un valor **n** entero introducido por el usuario.
- Compare los resultados obtenidos con una y otra versión, así como con la función **exp** (exponencial en base e), disponible librería **math** de Pascal (se puede importar utilizando “uses math;” tras la cabecera del programa).
- Identifique en los algoritmos los elementos del esquema de iteración.

Problema 3

El 1 de enero de 2019 un cliente abrió una cuenta de ahorro con 125 euros. El banco abona mensualmente un interés anual del 4 %. Al inicio de cada mes (a partir del mes de febrero de 2019) el cliente ingresa 180 euros. El cliente quiere alcanzar al menos M euros disponible para poder comprarse un coche. Calcule el número de meses que tiene que esperar el cliente para disponer en la cuenta de ahorro de M euros, siendo M un dato de entrada. Para facilitar la resolución del problema, se dispone de la definición recurrente de la sucesión de saldos mensuales en la cuenta de ahorro:

a_0=125
a_n=a_{n-1} + (0.04/12) * a_{n-1} + 180 para $n > 0$