Resolución de Problemas y Algoritmos

Clase 7:
Repetición Condicional

Dr. Alejandro J. García
http:// cs.uns.edu.ar /~ajg

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Bahía Blanca - Argentina

¿Preguntas?

```
ALGORITMO dar-la-clase-con-transparencias
    Saludar y comenzar la presentación
    Decir: "¿alguna pregunta sobre lo visto hasta ahora? "
    Escuchar la respuesta
    REPETIR MIENTRAS hay preguntas
        -escuchar, pensar y contestar la pregunta
        -decir: " ¿otra pregunta? "
        -escuchar la respuesta
    FIN REPETIR
    REPETIR
        -pasar a la transparencia siguiente
        -explicar los conceptos usando pizarrón
        -Si hay una pregunta entonces contestarla
    HASTA las 16:00 hs, o no quedan más transparencias
    Decir: "muchas gracias, hasta la próxima clase"
```


Repetición condicional en Pascal: WHILE

La sentencia WHILE (mientras) es una forma de especificar repetición condicional en Pascal.

```
WHILE expresión booleana
  DO
     sentencia simple
     o compuesta ;
Otra sentencia siguiente;
```

Mientras el resultado de la expresión sea verdadero ejecuta la sentencia del DO,

si el resultado es falso, saltea la sentencia del DO y sigue en la siguiente al while.

Las sentencias dentro de un ciclo WHILE se ejecutan 0 (cero) o más veces dependiendo de la condición.

While: repetición condicional

Ejemplo:

```
Write('Tope='); read(tope);
cont := 0;
WHILE cont < tope
DO
   Begin
     writeln(cont);
     cont:=cont+1;
   End;
Writeln('----------');
```

Mientras cont < tope sea verdadero ejecuta:

Si cont < tope es falso sigue en:

Las sentencias dentro de un ciclo WHILE se ejecutan 0 (cero) o más veces dependiendo de la condición.

Sentencia simple vs. compuesta

```
N := 0;
WHILE  N < 3  DO
        N := N + 1;
writeln(N);
```

Importante: Si en un WHILE queremos que se repita más de una sentencia (un **bloque** de sentencias), debemos usar la sentencia compuesta con BEGIN-END

```
N := 0;
WHILE  N < 3  DO
        N := N + 1;
        writeln(N);
```

```
N := 0;
WHILE  N < 3  DO
    BEGIN
        N := N + 1;
        writeln(N);
    END;
```



```
PROGRAM EjemploWhile; {muestra una aplicación útil del WHILE}
VAR letra: char; opcion:integer;
BEGIN
  writeln('Ingrese una letra mayúscula');  read(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;
  writeln('Ingrese una opción: ');
  write(' (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...}
```

Problema propuesto

Problema: Escribir un programa para ver si un número N es primo (***Definición**: un número es primo si es un entero positivo mayor que 1, que es divisible solamente por si mismo y la unidad*)

Ejemplos: 2 , 3, 7, 11 y 2003: son primos
4 no es primo es divisible por 2
2001 no es primo, es divisible por 3
121 no es primo, es divisible por 11

Una solución: cuento la cantidad de divisores que tiene N, si tiene 0, entonces es primo.

Otra forma de expresarlo: si tiene un divisor NO es primo

Algoritmo para "esPrimo"

```
ALGORITMO EsPrimo (algoritmo 1)
COMIENZO
 leer(N)
 CantDivisores ← 0 { asumo que cantidad de divisores de N es 0 }
 Divisor ← 2 { el primer divisor a considerar es 2 ... }
 REPETIR MIENTRAS (Divisor < N) y (CantDivisores = 0)
     SI N // Divisor = 0 {Si Divisor divide a N, incremento CantDivisores}
         ENTONCES CantDivisores ← CantDivisores + 1
     Divisor ← Divisor + 1 {paso al próximo Divisor .. }
 fin repetir mientras
 SI (CantDivisores=0) y (N > 1)
  ENTONCES   'N es primo'
  SINO       'N no es primo'
 FIN ALGORITMO
```



```
PROGRAM EsPrimo; {del algoritmo 1}
VAR N,cantDivisores,Divisor:integer;
BEGIN
 writeln('Ingrese un número'); read(N);
 CantDivisores := 0; { asumo que cantidad de divisores de N es 0 }
 Divisor := 2; { el primer divisor a considerar es 2 ... }
 WHILE (Divisor < N) and (CantDivisores = 0) DO
 BEGIN
   IF N mod Divisor = 0 {Si Divisor divide a N, incremento CantDivisores}
      THEN CantDivisores := CantDivisores + 1 ;
   Divisor := Divisor + 1; {paso al próximo Divisor .. }
 END {while}
  IF (CantDivisores=0) and (N > 1)
   THEN  writeln('Es nro. Primo')
   ELSE  writeln('NO es nro. Primo')
END.
```

Observación: en realidad no necesito "contar" los divisores


```
ALGORITMO EsPrimo (algoritmo 2, otra solución)
COMIENZO
 leer(N)
 SI (N > 1)
 ENTONCES  EsPrimo ← Verdadero { asumo que es primo}
 SINO EsNroPrimo ← Falso
 Divisor ← 2 { el primer divisor a considerar es 2 ... }
 REPETIR MIENTRAS (Divisor < N) y (esPrimo = verdadero)
     SI N // Divisor = 0 {Si Divisor divide a N, NO ES PRIMO}
         ENTONCES esPrimo ← falso
     Divisor ← Divisor + 1 {paso al próximo Divisor .. }
 fin repetir mientras
FIN ALGORITMO
```

TAREA: Implemente en Pascal este algoritmo.

Repetición condicional en Pascal

```
REPEAT
   <sentencia 1>
   <sentencia 2>
     . . .
   <sentencia  n>
UNTIL <condición>
<sentencia siguiente al repetir>
```

Si la condición es **falsa** sigue en

Si la condición es **verdadera** sigue en

- **Las sentencias dentro de un REPEAT-UNTIL se ejecutan 1 o más veces dependiendo de la condición final (expresión boolean)**

Repetición condicional en Pascal

```
Write('Tope='); read(tope);
cont := 0;
REPEAT
 writeln(cont);
 cont:=cont+1;
UNTIL cont = tope;
Writeln('----------');
```

Si la condición es **falsa** sigue en

Si la condición es **verdadera** sigue en

- **Ejercicio propuesto para practicar: escriba una solución para ver si un número es primo con REPEAT-UNTIL en lugar de while.**

Conceptos: Diferencias REPEAT y WHILE

REPEAT-UNTIL	WHILE
• si condición es **falsa** sigue repitiendo. • si condición es **verdadera** deja de repetir. • **repite 1 o más veces**: siempre ejecuta al menos una vez la secuencia interna al repetir	• si condición es **verdadera** sigue repitiendo • si condición es **falsa** deja de 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.**

Dígito presente en un número

Problema: Escriba un programa en Pascal para determinar si un dígito D aparece en un número entero N.

Ej: si D = 3 y N=1234, entonces D aparece.

Ej: si D = 6 y N=661, entonces D aparece.

Ej: si D = 3 y N=661, entonces D NO aparece.

Solución: *voy mirando* dígito a dígito y si alguno es igual a D es que aparece; si recorro todo el número y ninguno es igual a D entonces no aparece.

(pero ¿cómo puedo "mirar" un dígito particular?)

→ Para obtener el último dígito de N : N mod 10

→ Para eliminar el último dígito de N : N div 10

Dígito presente en un número

Solución: *voy mirando* dígito a dígito y si alguno es igual a D es que aparece; si recorro todo el número y ninguno es igual a D entonces no aparece.

Observación: un número siempre tiene al menos un dígito (se repetirá 1 o más veces)

Algoritmo:

Si el último dígito de N es igual a D (N mod 10 = D)
entonces aparece
Eliminar el último dígito de N (N ← N div 10)
Repetir los dos pasos anteriores hasta que:
"N no tenga mas dígitos" o "descubrí que aparece"


```
PROGRAM   aparece; {indica si está un dígito en un número}
VAR numero, digito, cant_veces : INTEGER; esta: BOOLEAN;
BEGIN
writeln('ingrese Num y Dig'); readln(numero, digito);
esta:=false; {asumo que no está}
REPEAT
    IF (numero mod 10)  = digito {comparo con el último dígito}
        THEN  esta:=true;
    numero := numero div 10;  {elimino el último dígito}
UNTIL (numero = 0) OR esta;   {hasta recorrer todo o esta}
IF esta {también puede ser "esta=true" pero es redundante}
then writeln(digito, ' aparece en ', numero)
else writeln(digito, ' no aparece en ', numero);
END.
```


Repeticiones anidadas (algunos ejemplos)

```
REPEAT
WHILE <condición>
DO  WHILE <condic.>
    DO  <sent. >
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: sentencias repetitivas en Pascal

Repetición condicional:

```
WHILE <expresión> DO
  <sentencia>
```

Repetición condicional:

```
REPEAT
  <sentencias>
UNTIL  <expresión>
```

Repetición incondicional:

```
FOR <ini> TO <fin> DO
  <sentencia>
FOR <ini> DOWNTO <fin> DO
  <sentencia>
```

Conceptos: repetición condicional vs. incondicional

- La repetición condicional (REPEAT o WHILE) depende de una condición (expresión boolean).
- La repetición incondicional (FOR) no depende de ninguna condición y 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 usarse una repetición condicional.
- Pero hay algunas repeticiones condicionales que NO pueden reescribirse para usar una incondicional (FOR).

Repetición condicional es MÁS 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
```


Conceptos: CICLOS INFINITOS ⊗

- Una repetición condicional mal programada puede caer en una REPETICIÓN INFINITA.

```
{...OK...}
v:=1; w:=1;
REPEAT
    v:=v+1;
UNTIL V = 9;
Write(V);
```

```
{ 1. MAL  }
v:=1; w:=1;
REPEAT
    v:=v+1;
UNTIL w = 0;
Write(V);
```
MAL

```
{ 2. MAL }
v:=1; w:=1;
WHILE V<>9
DO v:=1;
Write(V);
```
MAL

```
{ 3. MAL }
v:=1; w:=1;
REPEAT
    v:= w-1;
UNTIL V = 9;
Write(V);
```
MAL

Algunos problemas clásicos:

1. La condición de corte es errónea.
2. La variable de la condición no se modifica.
3. La variable de la condición se modifica mal.

CICLOS INFINITOS ⊗

Una repetición que se realiza infinitas veces se denomina ciclo infinito

- Un ciclo infinito en un programa será considerado un ERROR GRAVE de programación.
- Cuando realice la traza de sus programas debe asegurarse que sus repeticiones no puedan caer en ciclos infinitos.

¿Qué es ENTER?

- En las máquinas de escribir mecánicas al finalizar un renglón hay que hacer dos movimientos:
  (1) retorno de carro (2) nueva línea
- La tecla ENTER tiene asociados 2 caracteres:
  (1) ASCII 13: retorno de carro (CR: carriage return)
  (2) ASCII 10: nueva línea (LF: line feed)
- Los caracteres 13 y 10 son caracteres de control y al imprimirlos en pantalla producen un efecto en lugar de mostrar algo visible. Vea por ejemplo:

```
Program uno;
Begin
WRITE(CHR(65));
WRITE(CHR(66));
End .
```

```
Program dos;
Begin
WRITE(CHR(65));
WRITE(CHR(13));
WRITE(CHR(10));
WRITE(CHR(66));
End .
```

```
Program tres;
Begin
WRITE(CHR(65));
WRITE(CHR(10));
WRITE(CHR(66));
End .
```