

Ejercicio 1 (35 puntos)

Las siguientes definiciones de constantes y tipos representan las palabras que componen un texto y sus valores de polaridad. Estos valores son neutro (por ejemplo, "mesa"), positivo (por ejemplo, "feliz") o negativo (por ejemplo, rechazo).

```
const
  Max = ...; (* valor mayor que 1 *)

type
  Polaridad = record
    case esNeu:Boolean of
      false: (valor: (pos, neg));
      true: ();
    end;
  Palabra = record
    pal: Cadena; (* se asume definido, no se usa *)
    pol: Polaridad
  end;
  Texto = record
    pals: array [1..Max] of Palabra;
    tope: 0..Max
  end;
```

a) (15 puntos)

Implemente el procedimiento:

```
procedure polaridadTexto(tex : Texto; var polTex : Polaridad);
```

Este procedimiento calcula la polaridad del texto y la devuelve en el parámetro polTex. La polaridad del texto se establece según la polaridad predominante entre las palabras que lo componen:

- si la cantidad de palabras positivas es mayor que la de negativas y también mayor que la de neutras, el texto es positivo
- si la cantidad de palabras negativas es mayor que la de positivas y también mayor que la de neutras, el texto es negativo.
- en cualquier otro caso el texto es neutro. Asuma que el texto tiene al menos una palabra.

Solución:

```
procedure polaridadTexto(tex : Texto; var polTex : Polaridad);

var i, cantPos, cantNeg, cantNeu : Integer;

begin
  cantPos := 0;
  cantNeg := 0;
  cantNeu := 0;

  for i := 1 to tex.tope do
    if tex.pals[i].pol.esNeu then
      cantNeu := cantNeu+1
    else
      case tex.pals[i].pol.valor of
        pos : cantPos := cantPos+1;
        neg : cantNeg := cantNeg+1
      end;
    end;

    if (cantPos > cantNeg) and (cantPos > cantNeu) then
      begin
        polTex.esNeu := false;
        polTex.valor := pos
      end
    else if (cantNeg > cantPos) and (cantNeg > cantNeu) then
      begin
        polTex.esNeu := false;
        polTex.valor := neg
      end
    else
      polTex.esNeu := true
    end;
  end;
```

Nota: Existen soluciones más eficientes que utilizan una búsqueda en lugar de una recorrida, dado que la recorrida podría detenerse cuando la cantidad de palabras restantes no es suficiente para cambiar el resultado. En este caso se considera correcta la solución que recorre todo el arreglo debido a que la solución con iteración condicional tiene mayor complejidad de lo usual en las evaluaciones del curso.

b) (20 puntos)

Implemente la función:

```
function alMenosUno(tex : Texto) : Boolean;
```

Esta función devuelve TRUE si en el texto hay al menos una palabra con cada polaridad y FALSE en caso contrario.

Solución:

```
function alMenosUno(tex : Texto) : Boolean;
var i, cantPos, cantNeg, cantNeu : Integer;
begin
  cantPos := 0;
  cantNeg := 0;
  cantNeu := 0;
  i:=1;
  (* itero mientras no llegue al tope y alguna de las polaridades siga siendo 0 *)
  while (i <= tex.tope) and ((cantPos=0) or (cantNeg = 0) or (cantNeu = 0)) do
    begin
      if tex.pals[i].pol.esNeu then
        cantNeu := cantNeu+1
      else
        case tex.pals[i].pol.valor of
          pos : cantPos := cantPos+1;
          neg : cantNeg := cantNeg+1
        end;
      i := i+1
    end;
  alMenosUno := (cantPos>0) and (cantNeg > 0) and (cantNeu > 0)
end;
```

Ejercicio 2 (20 puntos)

Dada la siguiente declaración:

```
const M = ...; (* valor mayor que 1 *)  
type arr = array [1..M] of integer;
```

Implemente la función:

```
function esSumaSiguietes (var a: arr): integer;
```

Esta función recibe como parámetro un arreglo y retorna el índice del arreglo donde se encuentra un elemento que sea igual a la suma de todos los siguientes. En caso de que no exista, devuelve 0. Por ejemplo, en el arreglo [1, 3, 4, 13, 4, 6, 3] La función deberá retornar 4, ya que en esa posición se encuentra el número 13, que es la suma de los siguientes (4+6+3=13). Puede asumir que a lo sumo existe un elemento que cumpla esta condición.

Solución:

```
function esSumaSiguietes (var a: arr): Integer;  
var i, acum: integer;  
begin  
    i := M;  
    acum := 0;  
    while (i > 0) and (a[i] <> acum) do  
        begin  
            acum := acum + a[i];  
            i := i-1  
        end;  
  
    esSumaSiguietes := i  
end;
```

Ejercicio 3 (25 puntos)

Dada la siguiente declaración:

```
type
  Lista = ^TipoCelda;
  TipoCelda = record
    dato: integer;
    sig: Lista
  end;
```

Implemente el procedimiento:

```
procedure intercalarListas(var l1 : Lista; var l2 : Lista; var l3 : Lista);
```

Este procedimiento recibe dos listas del mismo largo (l1 y l2), y retorna en l3 los elementos de l1 y l2 intercalados. No debe crear memoria nueva sino utilizar las celdas ya existentes en l1 y l2. Al terminar el procedimiento, l1 y l2 deben ser nil.

Ejemplos:

Entrada:

l1 = 5 → 3 → 4 → nil l2 = 6 → 1 → 8 → nil

Salida:

l1 = nil l2 = nil l3 = 5 → 6 → 3 → 1 → 4 → 8 → nil

Entrada:

l1 = 5 → nil l2 = 6 → nil

Salida:

l1 = nil l2 = nil l3 = 5 → 6 → nil

Entrada:

l1 = nil l2 = nil

Salida:

l1 = nil l2 = nil l3 = nil

Solución:

```
procedure intercalarListas(var l1 : Lista; var l2 : Lista; var l3 : Lista);
var aux1,aux2 : Lista;
begin
  l3 := l1;
  while l1 <> nil do
    begin
      aux1 := l1^.sig;
      aux2 := l2^.sig;

      l1^.sig := l2;
      l2^.sig := aux1;

      l1 := aux1;
      l2 := aux2;
    end
  end
  l1 := nil;
  l2 := nil
end;
```

Ejercicio 4 (10 puntos)

Diga cuál es la salida del siguiente programa, si se lee de la entrada el dígito posterior al guión de su cédula.

```
program alcance(input,output);  
  
var num : integer;  
  
function f (a: integer) : boolean;  
var b : integer;  
begin  
  b := (3 * a) div 2;  
  f := (a = num) or (b = num)  
end;  
  
procedure p (var c: integer);  
begin  
  c := (c + 3) div 5;  
  writeln(c, num)  
end;  
  
begin  
  readln(num);  
  num := num + 2;  
  p(num);  
  writeln(num);  
  if f(num) then  
    num := num * 2  
  else  
    num := num * 3;  
  writeln(num);  
end.
```

Solución:

Entrada:	0
Salida:	
1	1
1	
2	

Entrada:	1
Salida:	
1	1
1	
2	

Entrada:	2
Salida:	
1	1
1	
2	

Entrada:	3
Salida:	
1	1
1	
2	

Entrada:	4
Salida:	
1	1
1	
2	

Entrada:	5
Salida:	
2	2
2	
4	

Entrada:	6
Salida:	
2	2
2	
4	

Entrada:	7
Salida:	
2	2
2	
4	

Entrada:	8
Salida:	
2	2
2	
4	

Entrada:	9
Salida:	
2	2
2	
4	
