Examen de Programación 1
Instituto de Computación - Facultad de Ingeniería
Febrero 2020

Leer con atención

- Todos los códigos deben ser escritos en el lenguaje **Pascal** tal como fue dado en el curso. A grandes rasgos, este es el Pascal estándar con los siguientes agregados: utilización de `else` en la instrucción `case` y evaluación por circuito corto de las operaciones booleanas (`and` y `or`).
- En todos los problemas se evaluará, además de la lógica correcta, la utilización de un buen estilo de programación de acuerdo a los criterios impartidos en el curso. De esta manera se restarán puntos entre otros conceptos por: mala o nula indentación, mala utilización de las estructuras de control, código confuso e innecesariamente largo, programas ineficientes, utilización de variables globales, pasaje incorrecto de parámetros, etc. No obstante, por razones prácticas, no exigimos que incluya comentarios en los códigos que escriba en la prueba.
- **Escriba su nombre completo y cédula en todas las hojas. Numere todas las hojas y escriba la cantidad total de hojas.**
- **Escriba de un solo lado de la hoja y comience cada ejercicio en una nueva hoja**.

Ejercicio 1 50 pts

Dada las siguientes declaraciones:

```
const MAX = ...; (* valor entero mayor que 0 *)
type
   RangoIndice = 1 .. MAX;
   TipoLista  = ^TipoNodo;

   TipoNodo = record
      valor : char;
      sig : TipoLista;
   end;

   TipoCasillas = array [RangoIndice] of TipoLista;
```

Parte a) 5 pts

Escribir la función *largoLista* que devuelve el largo de la lista dada.

```
function largoLista (lista : TipoLista) : integer;
```

Ejemplos:

lista	largoLista
vacía	0
a -> b -> a	3

Parte b) 15 pts

Escribir la función *menorLargo* que devuelve el índice de la lista de menor largo del arreglo casillas. Si hay listas de igual largo toma la de menor índice.

```
function menorLargo (casillas : TipoCasillas) : RangoIndice;
```

Ejemplos con MAX = 4:

casillas	menorLargo
[Vacía, Vacía, Vacía, Vacía]	1
[a -> b -> a , b , c -> d , a -> c]	2
[a -> b -> a , a -> b -> a -> c , b -> c , c -> d]	3
[a -> b -> a , Vacía , b -> c , c -> d]	2

Parte c) 25 pts

Escribir el procedimiento *agregaElemento* que inserta un elemento en una lista, de la siguiente manera:

- Si el elemento ya está en la lista, lo inserta a continuación de la primera aparición del mismo.
- Si el elemento no aparece en la lista, lo agrega al final.

```
procedure agregaElemento (ele : Char; var lista : TipoLista);
```

Ejemplos:

ele	lista (entrada)	lista (salida)
a	a -> b -> a -> c	a -> a -> b -> a -> c
g	a -> b -> c	a -> b -> c -> g
c	vacía	c

Parte d) 5 pts

Escribir el procedimiento *enLaDeMenorLargo* que dado un arreglo de tipo TipoCasillas y un elemento de tipo Char, agrega el elemento a la lista de menor largo del arreglo, de la misma manera que se especifica en la parte c. Si hay listas de igual largo lo agrega en la de menor índice.

```
procedure  enLaDeMenorLargo (ele : Char; var casillas : TipoCasillas);
```

Ejemplos con MAX = 4:

ele	casillas (entrada)	casillas (salida)
a	[Vacía, Vacía, Vacía, Vacía]	[a, Vacía, Vacía, Vacía]
a	[a -> b -> a , b , c -> d , a -> c]	[a -> b -> a , b -> a , c -> d , a -> c]
b	[a -> b -> a , a -> b -> a -> c , b -> c , c -> d]	[a -> b -> a , a -> b -> a -> c , b -> b -> c , c -> d]
c	[a -> b -> a , Vacía , b -> c , c -> d]	[a -> b -> a , c , b -> c , c -> d]

Ejercicio 2 25 pts

Dadas las siguientes declaraciones:

```
const N = ... ;   {un entero mayor que 1}
type
   TipoFila = array [1..N] of Integer;
   TipoMatriz = array [1..N] of TipoFila;
```

Parte a) 10 pts

Escribir una función *ordenado* que dado un arreglo de tipo TipoFila devuelve True si está ordenado de menor a mayor, False si no. Los elementos repetidos se consideran ordenados (ver último ejemplo).

```
function ordenado (fila:TipoFila):Boolean;
```

Ejemplos con N = 3:

fila	ordenado
[2,4,1]	False
[5,6,7]	True
[7,6,5]	False
[1,1,1]	True

Parte b) 15 pts

Suponga que dispone ya implementado un procedimiento *ordenarFila* que ordena de menor a mayor un arreglo de tipo TipoFila:

```
procedure ordenarFila (var fila : TipoFila);
```

Escribir un procedimiento *ordenarMatriz* que dada una matriz de tipo TipoMatriz, la devuelve con todas las filas ordenadas de menor a mayor y en el parámetro cantidad devuelve la cantidad de filas que fue necesario ordenar .

```
procedure ordenarMatriz (var matriz : TipoMatriz; var cantidad : Integer);
```

Ejemplos con N = 3 :

matriz (entrada)	matriz (salida)	cantidad
[[5,6,7] , [2,4,1] , [8,9,10]]	[[5,6,7] , [1,2,4] , [8,9,10]]	1
[[5,6,7] , [2,4,1] , [8,11,10]]	[[5,6,7] , [1,2,4] , [8,10,11]]	2
[[5,6,7] , [1,2,3] , [9,9,10]]	[[5,6,7] , [1,2,3] , [9,9,10]]	0

Ejercicio 3 25 pts

Dadas las siguientes declaraciones:

```
const N = ...;   {un entero mayor que 1}
type
   RangoIndice = 1..N;
   RangoTope = 0..N;
   TipoArreglo = array [RangoIndice] of integer;

   TipoArregloTope = record
      arr : TipoArreglo;
      tope : RangoTope;
   end;
```

Escribir un procedimiento *unir* que dados dos arreglos de enteros ordenados de menor a mayor, inserta al final del primero todos los elementos del segundo que sea posible agregar, manteniendo el orden de menor a mayor.

```
procedure unir (var a : TipoArregloTope, b : TipoArregloTope);
```

Ejemplos con N = 6

a.arr (entrada)	a.tope (entrada)	b.arr	b.tope	a.arr (salidda)	a.tope (salida)
`[1, 3, 5, _, _, _]`	`3`	`[2, 3, 5, 7, _, _]`	`4`	`[1, 3, 5, 5, 7, _]`	`5`
`[1, 3, 5, 8, _, _]`	`4`	`[2, 3, 9, 11, 12, _]`	`5`	`[1, 3, 5, 8, 9, 11]`	`6`
`[_, _, _, _, _, _]`	`0`	`[2, 3, 9, 11, 12, 15]`	`6`	`[2, 3, 9, 11, 12, 15]`	`6`
`[1, 3, 5, 8, _, _]`	`4`	`[2, 3, 5, 7, _, _]`	`4`	`[1, 3, 5, 8, _, _]`	`4`