Segundo Parcial

Instituto de Computación - Facultad de Ingeniería

Noviembre 2012

Leer con atención

- Todos los programas o fragmentos de programas deben ser escritos en el lenguaje **Pascal** tal como fue dado en el curso. A grandes rasgos este es el Pascal estándar con algunos agregados, a saber:
  - Utilización de `else` en la instrucción `case`.
  - 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 sólo lado de la hoja y comience cada ejercicio en una nueva hoja.

Ejercicio 1 (20 puntos)

El método de exponenciación rápida es un algoritmo que permite calcular el resultado de una operación de exponenciación de números naturales en forma eficiente.
Para calcular *B* elevado a la *n* se procede de esta forma: se escriben dos secuencias de números:

$$\begin{array}{ll} B_0 & n_0 \\ B_1 & n_1 \\ . & . \\ . & . \\ B_k & n_k \end{array}$$

de manera tal que:

- $B_0 = B$ y $n_0 = n$
- $n_k = 1$
- Cada número de la columna de la izquierda se obtiene como el cuadrado del número anterior. Esto es $B_{i+1} = B_i^2$
- Cada número de la columna de la derecha se obtiene como el cociente entero de dividir por 2 el número anterior .

  Esto es $n_{i+1} = n_i \text{ div } 2$

Así por ejemplo si B=3 y n=10

```
   Bs   ns
==============
   3    10
   9     5 <-- impar
  81     2
6561     1 <-- impar
```

Luego se obtiene el resultado como la multiplicación de todos los números B_i de la columna de la izquierda cuyo correspondiente n_i en la columna de la derecha es impar:

$$3^{10} = 9 \times 6561$$

Se pide escribir una función:

```
function PotenciaRapida(base,exponente: integer): integer;
```

Se puede suponer que el exponente es mayor que 0.

La función debe implementar el método descripto más arriba sin utilizar estructuras auxiliares como arreglos o listas.

Ejercicio 2 (25 puntos)

Se consideran la siguiente definición de una estructura de arreglo que permite almacenar en sus celdas números enteros o reales indistintamente:

```
const
   MAX = 100;
type
   tnum   = (tentero,treal);
   Numero = record case tipo : tnum of
              tentero : (ivalor : integer);
              treal   : (rvalor : real);
            end;
   RangoArreglo  = 1 .. MAX;
   ArregloNumero = array [RangoArreglo] of Numero;
```

Se define también un arreglo con tope de reales:

```
   ArregloConTope =   record
                        info : array [RangoArreglo] of real;
                        tope : 0..MAX;
                      end;
```

a) Escribir una función:

```
 function MenorNumero(a,b: Numero) : boolean;
```

que retorne `true` en el caso que el valor guardado en `a` sea menor que el valor guardado en `b` y que retorne `false` en caso contrario.

b) Dada una estructura del tipo `ArregloNum` y un número `num` se considera una partición del mismo en tres regiones

```
+-----------+------------------+--------------+
|   A       |          B       |     C        |
+-----------+------------------+--------------+
```

de manera tal que: todos los números de *A* son mayores o iguales que `num` ; todos los números de *B* son menores que `num` ; y si *C* no es vacía, su primera celda contiene un número mayor o igual que `num` .

Ejemplos:

En todos los ejemplos aparecen números enteros por simplicidad, pero se debe tener en cuenta que las celdas del arreglo pueden contener tanto reales como enteros.

En todos los ejemplos se considera `num = 4`:

```
              A           B            C
   +---------------+------------+--------------+
   | 4  10   22  7  |  1  3   -2 | 28  11  2  1 |        A, B y C no vacías
   +---------------+------------+--------------+

                 B               C
    +--------------------------+--------------+
    | 1  0   2  1    1  3   -2 | 28  11  2  1 |          A vacía
    +--------------------------+--------------+

              A                 B
    +---------------+-----------------------+
    | 4  10   22  7  |  1  3   -2  2  1  2  1 |          C vacía
    +---------------+-----------------------+

                    A
     +-------------------------------------------+
     | 4  10   22  7    11  9   8  28  11  12  11 |       B y C vacías
     +-------------------------------------------+

                      B
     +-------------------------------------+
     | 3  1   2  -7   1  -9   -8  2  1  2  1 |           A y C vacías
     +-------------------------------------+
```

Escribir un procedimiento:

```
procedure obtener_region_b(    num       : Numero;
                               arreglo   : ArregloNumero;
                           var region_b  : ArregloConTope);
```

que guarda en la estructura de arreglo con tope `region_b`**,** todos los números de la región *B* del parámetro `arreglo`. Los posibles valores enteros se guardarán en `region_b` como reales.

Ejercicio 3 (15 puntos)

Se considera la siguiente definición de lista:

```
type
   Lista = ^Celda;
   Celda = record
                elem : Integer;
                sig : Lista;
           end
```

Se pide implementar el procedimiento:

```
  procedure rotacion(var l: Lista);
```

que mueve el primer elemento de la lista, al final.

Si la lista es $[x_1,x_2,...,x_k]$, luego de aplicar el procedimiento quedará $[x_2,...,x_k,x_1]$

Si la lista es vacía, la invocacion del procedimiento deja a la lista inalterada.

A continuación se muestran algunos ejemplos:

```
      Entrada          Rotación
      ========================
       []                []
       [7]               [7]
       [3,5,8,4]         [5,8,4,3]
```