Segundo Parcial - Programación 1
Instituto de Computación - Facultad de Ingeniería
Noviembre 2019

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 solo lado de la hoja y comience cada ejercicio en una nueva hoja.

Ejercicio 1 (25 pts)

La secuencia de ADN se representa como una cadena de letras. Se definen las siguiente estructuras::

```
const MAX = ...; (* algún valor apropiado *)

type TLetra = 'A'..'Z';
     TLargo  = 0 .. MAX;
     TIndice = 1 .. MAX;
     TCadADN = record
                  cad  : array [tIndice] of TLetra;
                  tope : TLargo;
               end;
```

Parte a) – 10 pts

Escriba el siguiente procedimiento que agrega la *letra* al final de la *cadena* tantas veces como se indica en *cant*. El procedimiento agrega tantas veces como pueda mientras el arreglo no se llene.

```
procedure amplificarADN ( letra : TLetra; cant : TLargo; VAR cadena : TCadADN );
```

Ejemplos (asumiendo MAX = 5) con letra = ‘A’ y cant = 3:

- entrada (cadena.cad = [‘T’,?,?,?,?], cadena.tope = 1), salida (cadena .cad = [‘T’,‘A’,‘A’,‘A’,?], cadena.tope = 4)
- entrada (cadena .cad = [‘T’,‘T’,‘T’,?,?], cadena.tope = 3), salida (cadena .cad = [‘T’,‘T’,‘T’,‘A’,‘A’], cadena.tope = 5)
- entrada (cadena .cad = [‘T’,‘A’,?,?,?], cadena.tope = 2), salida (cadena .cad = [‘T’,‘A’,‘A’,‘A’,‘A’], cadena.tope = 5)
- entrada (cadena .cad = [‘T’,‘A’,‘T’,‘T’,‘T’], cadena.tope = 5), salida (cadena .cad = [‘T’,‘A’,‘T’,‘T’,‘T’], cadena.tope= 5)

Parte b) – 15 pts

Escriba la siguiente función que dada una *cadena*, una *letra* y un *largo*, determine si existe una subcadena de letras consecutivas iguales a *letra* de largo al menos *largo*. Si existe, devuelve el indice de ocurrencia de la primera letra, devuelve -1 en caso contrario. De existir más de una subcadena, devuelve la de menor índice. Si *largo* es menor que 1 devuelve -1.

```
function indSubCadADN( cadena : TCadADN; letra : TLetra; largo : TLargo ) : integer;
```

Ejemplos (asumiendo MAX = 5):

- con (cadena .cad = [‘T’,‘A’,‘A’,‘A’,?], cadena.tope = 4) , letra = ‘A’, largo = 4, retorna -1.
- con (cadena .cad = [‘A’,‘T’,‘A’,‘A’,?], cadena.tope = 4) , letra = ‘A’, largo = 2, retorna 3.
- con (cadena .cad = [‘A’,‘T’,‘A’,‘A’,?], cadena.tope = 4) , letra = ‘A’, largo = 1, retorna 1.
- con (cadena .cad = [‘A’,‘T’,‘A’,‘A’,?], cadena.tope = 4) , letra = ‘A’, largo = 0, retorna -1.

Solución

Parte a)

Con for:

```
procedure amplificarADN ( letra: TLetra; cant : TLargo; VAR cadena : TCadADN );
var i, total : TLargo;
begin
   total := cant;
   if total + cadena.tope > MAX then
      total := MAX - cadena.tope;
   for i := 1 to total do
      cadena.cad[cadena.tope+i] := letra;
   cadena.tope := cadena.tope + total
end;
```

Con while:

```
procedure amplificarADN ( letra: TLetra; cant : TLargo; VAR cadena : TCadADN );
var cont : TLargo;
begin
   cont := 0;
   while (cadena.tope < MAX) and (cont < cant) do
   begin
      cadena.tope := cadena.tope + 1;
      cadena.cad[cadena.tope] := letra;
      cont := cont + 1
   end
end;
```

Parte b)

```
function indSubCadADN( cadena : TCadADN; letra : TLetra; largo : TLargo ) : integer;
var i, j, cant, res : integer;
begin
   res  := -1;
   if largo > 0 then
   begin
      i := 1;
      cant := 0;
      while (i <= cadena.tope - largo + 1) and (cant < largo) do
      begin
       j    := i;
       cant := 0;
       while (cant < largo) and (cadena.cad[j] = letra) do
       begin
          j    := j + 1;
          cant := cant + 1
       end;
       if cant = largo then res := i;
       i := j + 1
      end
   end;
   indSubCadADN := res
end;
```

Ejercicio 2 (30 pts)

Considere la siguiente definición en Pascal de las estructuras de datos que permiten representar polinomios de coeficientes enteros y variable real:

```
type  nat = 0..maxint; (* tipo del exponente *)
      monomio = record
                   coef : integer;
                   exp  : nat
                end;
      polinomio = ^termino
      termino = record
                  mon:monomio;
                  sig: polinomio
                end;
```

Parte a) – 6 pts

Escribir una función evalMonomio que dado un valor real *arg* retorna el resultado de evaluar el monomio. No se puede utilizar la función *exp*, la solución debe ser un algoritmo iterativo.

```
      function evalMonomio ( m:monomio; arg:real ): real;
```

- con (m.coef = 10, m.exp = 2), arg = 4.5, retorna 202.5
- con (m.coef = 8, m.exp = 0), arg = 4.5, retorna 8.0

Parte b) – 10 pts

Escribir una función *evalPolinomio*, que dados un polinomio *p* y un valor real *arg*, retorna el resultado de evaluar *p* en *arg*. Si el polinomio es vacío el resultado es 0.

```
      function evalPolinomio ( p:polinomio; arg:real ) : real;
```

Ejemplo:

- con (m.coef = 4, m.exp = 1) → (m.coef = 10, m.exp = 2) → (m.coef = 8, m.exp = 0), arg = 4.5 , retorna 228.5

Parte c) – 14 pts

Dada la siguiente declaración:

```
type MaybeCoef =  record case existeCoef : boolean of
                      true  : (coef:integer);
                      false : ()
                  end;
```

Escribir un procedimiento *coefMonomioGrado*, que dados un polinomio *p* y un exponente *exp* retorna, si existe, el coeficiente del monomio de grado *exp*. Se puede asumir que si existe, hay un solo monomio de ese grado.

```
      procedure coefMonomioGrado ( p:polinomio; exp:nat; var mcoef: MaybeCoef );
```

Ejemplo:

- con (m.coef=4, m.exp=1) → (m.coef=10, m.exp= 2) → (m.coef=8, m.exp=0), exp=1, retorna (m.existeCoef=true, m.coef = 4)
- con (m.coef = 4, m.exp = 1) → (m.coef = 10, m.exp = 2) → (m.coef = 8, m.exp = 0), exp = 3, retorna (m.existeCoef = false)

Solución

Parte a)

```
function evalMonomio ( m: monomio; arg:real ): real;
var i   : nat;
    pot : real;
begin
   pot := 1;
   for i := 1 to m.exp do
      pot := pot * arg;
   evalMonomio := m.coef * pot
end;
```

Parte b)

```
function evalPolinomio ( p:polinomio; arg:real ) : real;
var val  : real;
    paux : polinomio;
begin
   val  := 0;
   paux := p;
   while paux <> nil do
   begin
      val  := val + evalMonomio(paux^.mon,arg);
      paux := paux^.sig
   end;
   evalPolinomio := val
end;
```

Parte c)

```
procedure coefMonomioGrado ( p:polinomio; exp:nat; var mcoef: MaybeCoef );
var paux : polinomio;
begin
   mcoef.existeCoef := false;
   paux := p;
   while (paux <> nil) and (paux^.mon.exp <> exp) do
      paux := paux^.sig;
   if paux <> nil then
   begin
      mcoef.existeCoef := true;
      mcoef.coef := paux^.mon.coef
   end
end;
```

Ejercicio 3 (5 pts)

Dado el siguiente programa, escribir cuál será su salida cuando la variable *z* se carga de la entrada estándar con **el último dígito** de **su** CI (antes del dígito verificador). Por ejemplo, si su CI es 1.234.567-8, el último dígito es 7:

```
program segundoParcial;
var z : integer;
procedure primera(var a : integer; var b : integer);
var z : integer;
begin
      z:= a+1;
      a:= z;
      b:= b*2;
      writeln (z)
end; (*primera*)
procedure tercera(a : integer; var b : integer);
begin
      a:= a+1;
      b:= b*2;
      writeLn(a,' ',b,' ',z)
end; (*tercera*)
begin
      readln(z);
      tercera(z,z);
      primera(z,z);
      writeLn(z)
end.
```

Solución

0	1	2	3	4	5	6	7	8	9
1 0 0 1 2	2 2 2 3 6	3 4 4 5 10	4 6 6 7 14	5 8 8 9 18	6 10 10 11 22	7 12 12 13 26	8 14 14 15 30	9 16 16 17 34	10 18 18 19 38