

Práctico 12 - Memoria dinámica. Punteros y Listas

Programación 1
InCo - Facultad de Ingeniería, Udelar

1. Dadas las siguientes declaraciones:

```
type
  PunteroInt = ^integer;
  PunteroChar = ^char;
var
  apun1, apun2: PunteroInt;
  apun3, apun4: PunteroChar;
```

(a) Determine cuáles de las siguientes instrucciones son válidas:

- new(apun1)
- new(apun1^)
- apun1 := apun3
- apun2^ := apun2^ + apun1^
- writeln(apun2, apun3)
- read(apun1^, apun4^)
- apun2 := new(apun1)
- dispose(apun3)
- apun1 := NIL
- apun3^ := NIL
- apun3 := apun4 and (apun3 = NIL)

(b) Determine la salida que despliega el siguiente fragmento de código:

```
new(apun1);
new(apun2);
new(apun3);
apun2 := apun1;
apun1^ := 2;
apun2^ := 3;
apun3^ := 'A';
writeln(apun1^, apun2^, apun3^)
```

(c) ¿Tiene algún error el siguiente fragmento de código?

```
new(apun1);
read(apun1^);
writeln(apun1^);
dispose(apun1);
writeln(apun1^)
```

(d) Determine la salida que despliega el siguiente fragmento de código:

```
new(apun3);
new(apun1);
apun3^ := 'Z';
apun2 := NIL;
apun4 := NIL;
if (apun3 <> NIL) and (apun2 = NIL) then
  writeln ('A');
if apun3^ = 'Z' then
```

```

        writeln ('Z')
    else
        writeln ('X')

```

2. Dado el siguiente programa:

```

Program
type
  TipoVehiculo = (barco, camion);
  Transporte = record
    capacidad : integer;
    case vehiculo : TipoVehiculo of
      barco : (habitaciones : integer);
      camion : ();
    end;
  PunteroTransporte = ^Transporte;

var a, b, c : PunteroTransporte;

begin
  new(a);
  a^.capacidad := 30;
  a^.vehiculo := barco;
  a^.habitaciones := 4;

  new(b);
  b^.capacidad := 5;
  b^.vehiculo := camion;

  new(c);
  c^.capacidad := 5;
  c^.vehiculo := camion;

end.

```

(a) ¿Cuál de las figuras (Figura 1) representa el estado de la memoria tras ejecutarse las instrucciones anteriores?

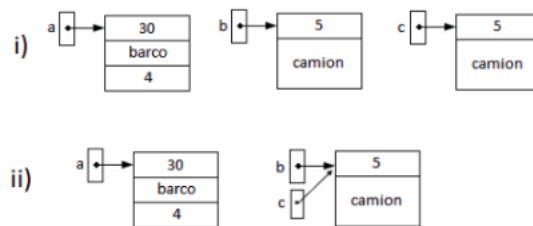


Figura 1: Ejercicio 2a

- i
- ii

(b) Suponiendo que al final del programa se agrega la instrucción: `b := c`

I) ¿Cuál de las figuras (Figura 2) representa el estado de la memoria posterior a la ejecución de dicha instrucción?

- i
- ii
- iii

II) ¿Cuál o cuáles de las afirmaciones son correctas?

- Cada puntero referencia un lugar de memoria distinto
- b y c referencian al mismo lugar de memoria. No se desperdicia memoria

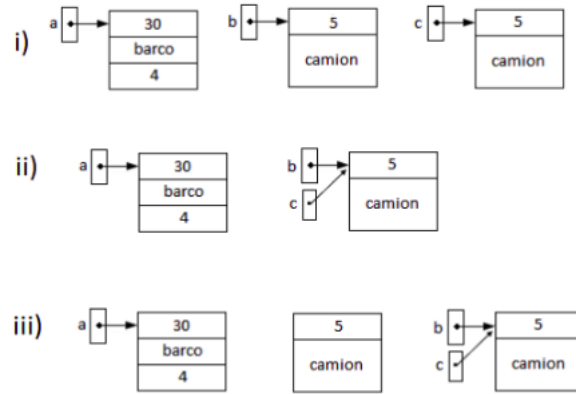


Figura 2: Ejercicio 2b

- b y c referencian al mismo lugar de memoria, queda memoria ocupada y se pierde su referencia
- III) ¿Sería correcto hacer `dispose(b)` antes de la instrucción `b := c`? Justifique.
- (c) Suponiendo ahora que al final del programa dado al comienzo del ejercicio, se le agregan las instrucciones
- ```

c^.capacidad := 4;
b^ := c^

```
- I) Dibuje el estado de la memoria posterior a la ejecución de dicha instrucción
- II) ¿Qué valor tiene `b^.capacidad` tras la ejecución de dichas instrucciones?
- 30
- 4
- 5

Para los ejercicios que siguen considere la siguiente definición de tipos para representar una lista encadenada de enteros:

```

type
 Positivo = 1..MaxInt;
 ListaInt = ^TCelda;
 TCelda = record
 info : integer;
 sig : ListaInt
 end;

```

3. Escribir los siguientes subprogramas:

- (a) Una función que retorne la suma de todos los elementos de la lista. Si la lista es vacía retorna 0.

```
function suma(l : ListaInt) : integer;
```

- (b) Una función que retorne el mínimo de todos los elementos de la lista. Se asume que la lista no es vacía.

```
function minimo(l : ListaInt) : integer;
```

- (c) Una función que retorne el producto de todos los elementos de la lista. Tenga en cuenta que alguno de los elementos puede ser 0.

```
function producto(l : ListaInt) : integer;
```

- (d) Una función que retorne la cantidad de números pares que hay en la lista.

```
function cuantosPares(l : ListaInt) : integer;
```

- (e) Una función que retorne la primera posición en la cual aparece valor dentro de la lista l. Las posiciones se cuentan desde 1 en adelante. Si valor no aparece en la lista, la función retorna -1.

```
function posicion(valor : integer; l : ListaInt) : integer;
```

- (f) Un procedimiento que obtiene el valor del elemento de la lista que está en la posición pos. Más precisamente el procedimiento retorna un registro con variante de tipo PosibleElem ya que puede no existir el elemento buscado.

```
type
PosibleElem = record case ok : boolean of
 true : (elem : integer);
 false : ()
end;
```

```
procedure elemEnPos(pos : Positivo; l : ListaInt; var resultado : PosibleElem);
```

- (g) Una función que retorna el último elemento de la lista. Se asume que la lista no es vacía.

```
function ultimo(l : ListaInt) : integer;
```

- (h) Una función que determina si la lista está ordenada de menor a mayor:

```
function ordenada(l : ListaInt) : boolean;
```

4. Escribir los siguientes subprogramas:

- (a) Insertar un elemento luego del segundo elemento de la lista. Si no hubiera segundo, la lista no cambia.

```
procedure insertarTercero(elem : integer; var l : ListaInt);
```

- (b) Insertar un elemento antes del último. Si no hubiera último, la lista no cambia.

```
procedure insertarPenultimo(elem : integer; var l : ListaInt);
```

- (c) Insertar el elemento nuevo luego del elemento que está en la posición pos. Si no existiera un elemento en tal posición, la lista no cambia.

```
procedure InsertarLuegoPos(nuevo : integer; pos : Positivo; var l : ListaInt);
```

- (d) Insertar un elemento nuevo antes de la primera aparición del elemento valor. Si no existiera tal elemento, la lista no cambia.

```
procedure InsertarAntes(nuevo, valor : integer; var l : ListaInt);
```

- (e) Insertar un elemento en una lista ordenada. El orden debe mantenerse luego de la inserción.

```
procedure InsertarOrdenado(nuevo : integer; var l : ListaInt);
```

- (f) Borrar el segundo elemento de la lista. Si no hubiera segundo, la lista no cambia.

```
procedure BorrarSegundo(var l : ListaInt);
```

- (g) Escribir procedimientos para las siguientes operaciones sobre una lista de enteros:

- I) borrar el último
- II) borrar el primer número impar
- III) borrar todos los números pares

- (h) Borrar la primera aparición de valor en la lista. Si no existiera tal elemento, la lista no cambia.

```
procedure BorrarPrimeraAparicion(valor : integer; var l : ListaInt);
```

5. Indique qué errores presentan los siguientes procedimientos, en los que hay una o más instrucciones incorrectas. Explique.

- (a) Se desea crear una lista de un solo elemento con valor 1.

```

procedure crearListaUnitaria (var nuevo : ListaInt);
begin
 new(nuevo);
 nuevo := NIL;
 nuevo^.dato := 1;
 nuevo^.sig := NIL;
end;

```

- (b) Se desea insertar un elemento al final de una lista.

```

procedure agregarAlFinal (dato : Integer; var lista : ListaInt);
var it, nuevo : ListaInt;
begin
 new(nuevo);
 nuevo^.dato := dato;
 if lista = NIL then
 lista := nuevo
 else
 begin
 new(it);
 it := lista;
 while it^.sig <> NIL do
 it := it^.sig;
 end;
 it^.sig := nuevo
 end
 end;
end;

```

- (c) Lo mismo que en la parte anterior, pero con otro error.

```

procedure agregarAlFinal (dato : Integer; var lista : ListaInt);
var it, nuevo : ListaInt;
begin
 new(nuevo);
 nuevo^.dato := dato;
 nuevo^.sig := NIL;
 if lista = NIL then
 lista := nuevo
 else begin
 it := lista;
 while it <> NIL do
 it := it^.sig;
 end;
 it := nuevo
 end
end;

```

6. Implementar los siguientes subprogramas:

- (a) Retornar una lista con los primeros  $k$  múltiplos positivos de  $num$ . La lista debe estar ordenada de menor a mayor. Se supone  $num > 1$

```
function multiplos(k : Positivo; num : Positivo) : ListaInt;
```

- (b) Retornar una copia limpia de la lista  $l$ . Esto es, una lista con el mismo contenido que  $l$  pero que no comparte ninguna celda con esta.

```
function copia(l : ListaInt) : ListaInt;
```

- (c) Retornar la lista  $l$  invertida. La lista resultado no comparte memoria con la lista  $l$ .

```
function invertir(l : ListaInt) : ListaInt;
```

- (d) Invertir la lista  $l$ , modificando directamente los punteros dentro de la lista sin crear nuevas celdas.

```
procedure invertir(var l : ListaInt);
```

- (e) Realizar la concatenación de dos listas  $l1$  y  $l2$ . En  $l1$  quedan todos los elementos originales seguidos de los elementos de  $l2$ . No se deben utilizar celdas nuevas.

```
procedure concatenar(var l1 : ListaInt; l2 : ListaInt);
```

- (f) Partir una lista `l` en dos listas de tal manera que `l1` contenga los primeros `k` elementos de `l` y `l2` contenga los restantes. Si `l` tuviera menos de `k` elementos, `l2` queda vacía y `l1` queda igual a `l`. No se deben utilizar celdas nuevas.

```
procedure partir(k : Positivo; l : ListaInt;
 var l1,l2 : ListaInt);
```

- (g) Escriba una función que, dadas dos listas de números enteros `l1` y `l2`, ordenadas de manera ascendente, obtenga una nueva lista que tenga los elementos de ambas listas ordenados de manera ascendente. La nueva lista no debe compartir memoria ni con `l1` ni con `l2`.

```
function IntercalarListas (l1, l2: ListaEnteros) : ListaEnteros;
```