

Punteros y Manejo de Memoria Dinámica

1.
 - a. ¿ Por qué es posible asignar NIL a una variable de apuntador independientemente del tipo del apuntador ?
 - b. ¿ En qué sentido esto puede considerarse una excepción a las reglas de tipo de Pascal ?
 - c. ¿ Qué indica esto acerca de la probable representación de un apuntador de valor NIL?

2. ¿ Por qué es razonable que Pascal no autorice el uso de los operadores $<$, $<=$, $>$, $>=$ para comparar dos apuntadores ? Idem para operaciones aritméticas.
3. Determine y justifique cuáles de las siguientes instrucciones son válidas, si se supone que se hicieron las siguientes definiciones y declaraciones:
4. TYPE
5. apunta = ^INTEGER;
6. apuntb = ^CHAR;
7. VAR
8. ap1, ap2 : apunta;
9. ap3, ap4 : apuntb;

a) NEW(ap1);	e) ap1 := NIL;	i) ap2 := NEW(ap1);
b) DISPOSE(ap4^);	f) ap4^ := NIL;	j) DISPOSE(ap3);
c) ap1 := ap3;	g) DISPOSE(apuntb);	
d) ap2 := ap2^ + ap1^;	h) ap3 := ap4^ * ap1^;	

Listas Encadenadas y variantes

10.
 - a. De una representación para la lista simple de enteros (LEnt).
 - b. Implementar las siguientes operaciones funcionales:
 - **FUNCTION Null() : LEnt;**
(* Crea la lista vacía *)
 - **FUNCTION Cons(x : INTEGER; l : LEnt) : LEnt;**
(* Inserta un elemento al principio de la lista *)
 - **FUNCTION Empty(l : LEnt) : BOOLEAN;**
(* Verifica si la lista está vacía *)
 - **FUNCTION Head(l : LEnt) : INTEGER;**
(* Retorna, si la lista no es vacía, el primer elemento de la lista *)

- **FUNCTION Tail(l : LEnt) : LEnt;**
(* Retorna, si la lista no es vacía, la lista sin su primer elemento *)
- c. Implementar las siguientes operaciones procedurales:
 - **PROCEDURE Null(VAR l : LEnt);**
(* Crea la lista vacía *)
 - **PROCEDURE Cons(x : INTEGER; VAR l : LEnt);**
(* Inserta un elemento al principio de la lista *)
 - **PROCEDURE Empty(l : LEnt; VAR e : BOOLEAN);**
(* Verifica si la lista está vacía *)
 - **PROCEDURE Head(l : LEnt; VAR h : INTEGER);**
(* Retorna, si la lista no es vacía, el primer elemento de la lista *)
 - **PROCEDURE Tail(VAR l : LEnt);**
(* Retorna, si la lista no es vacía, la lista sin su primer elemento *)

11. Dadas las siguientes operaciones funcionales sobre lista de enteros (LEnt):

- **FUNCTION Null() : LEnt;**
(* Crea la lista vacía *)
- **FUNCTION Cons(x : INTEGER; l : LEnt) : LEnt;**
(* Inserta un elemento al principio de la lista *)
- **FUNCTION Empty(l : LEnt) : BOOLEAN;**
(* Verifica si la lista está vacía *)
- **FUNCTION Head(l : LEnt) : INTEGER;**
(* Retorna, si la lista no es vacía, el primer elemento de la lista *)
- **FUNCTION Tail(l : LEnt) : LEnt;**
(* Retorna, si la lista no es vacía, la lista sin su primer elemento *)

Implementar las siguientes operaciones recursivamente:

- **FUNCTION IsElement(x : INTEGER; l : LEnt) : BOOLEAN;**
(* Verifica si un natural dado pertenece a la lista *)
- **FUNCTION Remove(x : INTEGER; l : LEnt) : LEnt;**
(* Elimina un natural dado de la lista *)
- **FUNCTION Length(l : LEnt) : INTEGER;**
(* Retorna la cantidad de elementos de la lista *)
- **FUNCTION Snoc(x : INTEGER; l : LEnt) : LEnt;**
(* Inserta el elemento x al final de la lista l *)
- **FUNCTION Append(l, p : LEnt) : LEnt;**
(* Agrega la lista p al final de la lista l *)
- **FUNCTION Reverse(l : LEnt) : LEnt;**
(* Invierte la lista l *)
- **FUNCTION Insert(x : INTEGER; l : LEnt) : LEnt;**
(* Inserta ordenadamente el elemento x en la lista ordenada l *)
- **FUNCTION Sort(l : LEnt) : LEnt;**
(* Ordena la lista l *)

- **FUNCTION Last(l : LEnt) : INTEGER;**
(* Retorna, si la lista l es no vacia, su último elemento *)
- **FUNCTION HowMany(x : INTEGER; l : LEnt) : INTEGER;**
(* Cuenta las ocurrencias del natural n en la lista l *)
- **FUNCTION Max(l : LEnt) : INTEGER ;**
(* Retorna, si la lista l es no vacia, su máximo elemento *)
- **FUNCTION n_esimo(n : INTEGER; l : LEnt) : INTEGER;**
(* Retorna, si la lista contiene por lo menos n elementos, el elemento que se encuentra en la posición n de la lista l *)
- **FUNCTION Ordered(l : LEnt) : BOOLEAN;**
(* Verifica si la lista esta ordenada *)
- **FUNCTION Change(x, y : INTEGER; l : LEnt) : LEnt;**
(* Retorna la lista resultado de cambiar el valor x por el valor y en la lista l *)
- **FUNCTION InsBefore(x, y : INTEGER; l : LEnt) : LEnt;**
(* Inserta el elemento x inmediatamente antes del elemento y en la lista l *)
- **FUNCTION InsAround(x, y : INTEGER; l : LEnt) : LEnt;**
(* Inserta el elemento x delante y detrás del elemento y en la lista l *)
- **FUNCTION Equals(l, p : LEnt) : BOOLEAN;**
(* Verifica si las listas l y p son iguales *)
- **FUNCTION Merge(l, p : LEnt) : LEnt;**
(* Genera una lista fruto de intercalar ordenadamente las listas l y p, que vienen ordenadas*)
- **FUNCTION IsInclude(l, p : LEnt) : BOOLEAN;**
(* Verifica si la lista p está incluida en la lista l *)
- **PROCEDURE Show(l : LEnt);**
(* Muestra los elementos de la lista *)

12. Implementar las siguientes operaciones accediendo directamente a la representación, iterativamente y sin usar procedimientos auxiliares:

- **FUNCTION IsElement(x : INTEGER; l : LEnt) : BOOLEAN;**
(* Verifica si un natural dado pertenece a la lista *)
- **FUNCTION Length(l : LEnt) : INTEGER;**
(* Retorna la cantidad de elementos de la lista *)
- **FUNCTION Last(l : LEnt) : INTEGER;**
(* Retorna, si la lista l es no vacía, el último elemento *)
- **FUNCTION Max(l : LEnt) : INTEGER;**
(* Retorna, si la lista l es no vacía, su máximo elemento *)
- **FUNCTION Average(l : LEnt) : REAL;**
(* Retorna, si la lista l es no vacía, el promedio de sus elementos *)
- **FUNCTION Insert(x : INTEGER; l : LEnt) : LEnt;**
(* Dada la lista l ordenada, inserta a x en l ordenadamente *)
- **FUNCTION Snoc(x : INTEGER; l : LEnt) : LEnt;**
(* Inserta el elemento x al final de la lista l *)
- **PROCEDURE Snoc(x : INTEGER; VAR l : LEnt);**
(* Inserta el elemento x al final de la lista l *)

- **FUNCTION Remove(x : INTEGER; l : LEnt) : LEnt;**
(* Elimina un natural dado de la lista l *)
- **PROCEDURE Remove(x : INTEGER; VAR l : LEnt);**
(* Elimina un natural dado de la lista l *)
- **FUNCTION Equals(l, p : LEnt) : BOOLEAN;**
(* Verifica si las listas l y p son iguales*)
- **FUNCTION IsInclude(l, p : LEnt) : BOOLEAN;**
(* Verifica si la lista p está incluida en la lista l *).

13. Implementar las siguientes operaciones accediendo directamente a la representación, recursivamente y sin usar procedimientos auxiliares:

- **FUNCTION IsElement(x : INTEGER; l : LEnt) : BOOLEAN;**
(* Verifica si un natural dado pertenece a la lista *)
- **FUNCTION Length(l : LEnt) : INTEGER;**
(* Retorna la cantidad de elementos *)
- **PROCEDURE Insert(x : INTEGER; VAR l : LEnt);**
(* Dada la lista l ordenada, inserta a x en l ordenadamente *)
- **PROCEDURE Snoc(x : INTEGER; VAR l : LEnt);**
(* Inserta el elemento x al final de la lista l *)
- **PROCEDURE Remove(x : INTEGER; VAR l : LEnt);**
(* Elimina un natural dado de la lista l *)
- **FUNCTION Equals(l, p : LEnt) : BOOLEAN;**
(* Verifica si las listas l y p son iguales*)
- **FUNCTION IsInclude(l, p : LEnt) : BOOLEAN;**
(* Verifica si la lista p está incluida en la lista l *).

14. Una variante a la implementación clásica de listas encadenadas es la llamada Lista Doblemente Encadenada. En esa implementación cada elemento de la lista referencia no sólo a su siguiente elemento sino también al anterior. Dar una representación e implementar las siguientes operaciones para Lista Doblemente Encadenada de Naturales:

- **FUNCTION Null() : LEnt;**
(* Crea la lista vacía *)
- **FUNCTION Cons(x : INTEGER; l : LEnt) : LEnt;**
(* Inserta un elemento al principio de la lista *)
- **FUNCTION Empty(l : LEnt) : BOOLEAN;**
(* Verifica si la lista está vacía *)
- **FUNCTION IsElement(x : INTEGER; l : LEnt) : BOOLEAN;**
(* Verifica si un natural pertenece a la lista *)
- **FUNCTION Remove(x : INTEGER; l : LEnt) : LEnt;**
(* Elimina todas las ocurrencias del natural en la lista *)
- **FUNCTION Insert(x : INTEGER; l : LEnt) : LEnt;**
(* Inserta ordenadamente *)

15. Otra variante a la implementaciones clásicas de listas encadenadas es la llamada Lista Encadenada Circular. En esa implementación el último elemento de la lista referencia al primero. Dar una representación e implementar las siguientes operaciones para Lista Encadenada Circular de Naturales accediendo directamente a la representación, iterativamente y sin utilizar procedimientos auxiliares:

- **FUNCTION Null() : LEnt;**
(* Crea la lista vacia *)
- **FUNCTION Cons(x : INTEGER; l : LEnt) : LEnt;**
(* Inserta un elemento al principio de la lista *)
- **FUNCTION Empty(l : LEnt) : BOOLEAN;**
(* Verifica si la lista está vacia *)
- **FUNCTION IsElement(x : INTEGER; l : LEnt) : BOOLEAN;**
(* Verifica si un natural pertenece a la lista *)
- **FUNCTION Remove(x : INTEGER; l : LEnt) : LEnt;**
(* Elimina todas las ocurrencias del natural en la lista *)
- **FUNCTION Insert(x : INTEGER; l : LEnt) : LEnt;**
(* Inserta ordenadamente *)

Árboles Binarios y de Búsqueda

16.

- . Dar una representación para árboles binarios de naturales (BinaryTree).
- a. Implementar las siguientes operaciones funcionales:
 - **FUNCTION NullTree() : BinaryTree;**
(* Devuelve el árbol vacío*)
 - **FUNCTION ConsTree(x : INTEGER; l, r : BinaryTree) : BinaryTree;**
(* Crea un árbol no vacío a partir de un natural y otros dos árboles*)
 - **FUNCTION IsEmptyTree(t : BinaryTree) : BOOLEAN;**
(* Determina si un árbol dado es o no vacío *)
 - **FUNCTION RootTree(t : BinaryTree) : INTEGER;**
(* Devuelve el valor en la raíz de un árbol no vacío *)
 - **FUNCTION LeftTree(t : BinaryTree) : BinaryTree;**
(* Devuelve el subárbol izquierdo de un árbol no vacío *)
 - **FUNCTION RightTree(t : BinaryTree) : BinaryTree;**
(* Devuelve el subárbol derecho de un árbol no vacío *)

17. Implementar las operaciones de recorrida sobre arboles binarios devolviendo la lista que resulta:

- **FUNCTION InOrder(t : BinaryTree) : LEnt;**
- **FUNCTION PreOrder(t : BinaryTree) : LEnt;**

- o **FUNCTION PostOrder(t : BinaryTree) : LEnt;**
- c. Utilizando solamente las operaciones de los ejercicios 4.b y 10.b (sin acceder a la representación)
- d. Accediendo directamente a la representación

18. Considere las siguientes operaciones sobre árboles binarios:

- o **FUNCTION Height(t : BinaryTree) : INTEGER;**
(* Retorna la distancia máxima desde la raíz a un nodo + 1 *)
- o **FUNCTION CountElements(t : BinaryTree) : INTEGER;**
(* Retorna la cantidad de nodos del árbol *)
- o **FUNCTION SINV(t : BinaryTree) : INTEGER;**
(* Número de subárboles izquierdos no vacíos *)
- o **FUNCTION SDNV(t : BinaryTree) : INTEGER;**
(* ídem anterior para subárboles derechos *)
- o **FUNCTION NH(t : BinaryTree) : INTEGER;**
(* Cantidad de hojas, es decir nodos sin hijos *)
- o **FUNCTION NT(t : BinaryTree) : INTEGER;**
(* cantidad de nodos con algún hijo *)
- o **FUNCTION NE2H(t : BinaryTree) : INTEGER;**
(* Cantidad de nodos con exactamente 2 hijos *)
- o **FUNCTION NE1H(t : BinaryTree) : INTEGER;**
(* Cantidad de nodos con exactamente 1 hijo *)
- o **FUNCTION NSHI(t : BinaryTree) : INTEGER;**
(* Cantidad de nodos con sólo hijo izquierdo *)
- o **FUNCTION NSHD(t : BinaryTree) : INTEGER;**
(* Cantidad de nodos con sólo hijo derecho *)
- j. Dar una representación e implementar las operaciones anteriores accediendo directamente a la estructura
- k. Implementar las operaciones anteriores trabajando solamente con las operaciones del ejercicio 10.b

19.

- . Dibuje el árbol binario de búsqueda generado por la secuencia de inserciones siguiente, con las letras ordenadas alfabéticamente: H, J, A, C, B, Z, T.
- a. Lo mismo para la siguiente secuencia de números en el orden habitual: 7, 3, 6, 2, 1, 4, 8, 5.
- b. Aplique los tres algoritmos de recorrido a los árboles producidos en las dos partes anteriores.

20.

- . Dar una representación para los Árboles Binarios de Búsqueda de Naturales (BST)
- a. Implementar las siguientes operaciones funcionales:

- **FUNCTION NullBST() : BST;**
(* Crea el árbol vacío *)
- **FUNCTION InsertBST(x : INTEGER; st : BST) : BST;**
(* Agrega un natural dado a un árbol dado. Debe de mantener la cualidad de árbol binario de búsqueda*)
- **FUNCTION IsEmptyBST(st : BST) : BOOLEAN;**
(* Determina si un árbol dado es o no vacío *)
- **FUNCTION RootBST(st : BST) : INTEGER;**
(* Devuelve el natural contenido en la raíz de un árbol no vacío dado *)
- **FUNCTION LeftBST(st : BST) : BST;**
(* Devuelve el subárbol izquierdo de uno no vacío dado *)
- **FUNCTION RightBST(st : BST) : BST;**
(* Devuelve el subárbol derecho de uno no vacío dado *)

b. Implementar las siguientes operaciones procedurales:

- **PROCEDURE NullBST(VAR st : BST);**
(* Crea el árbol vacío *)
- **PROCEDURE InsertBST(x : INTEGER; VAR st : BST);**
(* Agrega un natural dado a un árbol dado. Debe de mantener la cualidad de árbol binario de búsqueda*)
- **PROCEDURE IsEmptyBST(st : BST; VAR e : BOOLEAN);**
(* Determina si un árbol dado es o no vacío *)
- **PROCEDURE RootBST(st : BST; VAR r : INTEGER);**
(* Devuelve el natural contenido en la raíz de un árbol no vacío dado *)
- **PROCEDURE LeftBST(VAR st : BST);**
(* Devuelve el subárbol izquierdo de uno no vacío dado *)
- **PROCEDURE RightBST(VAR st : BST);**
(* Devuelve el subárbol derecho de uno no vacío dado *)

21. Implementar las siguientes operaciones sobre Árbol Binario de Búsqueda accediendo directamente a la representación:

- **FUNCTION IsElementBST(x : INTEGER; bs : BST) : BOOLEAN;**
(* Determina si el elemento x se encuentra el árbol *)
- **FUNCTION MaxBST(bs : BST) : INTEGER;**
(* Computa el máximo valor de un árbol no vacío *)
- **FUNCTION MinBST(bs : BST) : INTEGER;**
(* Computa el mínimo valor de un árbol no vacío *)
- **PROCEDURE RemoveMaxBST(VAR bs : BST);**
(* Elimina el máximo de un árbol *)
- **PROCEDURE RemoveMinBST(VAR bs : BST);**
(* Elimina el mínimo de un árbol *)
- **PROCEDURE RemoveBST(x : INTEGER; VAR bs : BST);**
(* Elimina de un árbol un valor dado *)

22. Considerar las declaraciones, en Pascal, del tipo de los nodos de un árbol binario de búsqueda de naturales y del tipo de los nodos de una lista de naturales que siguen:

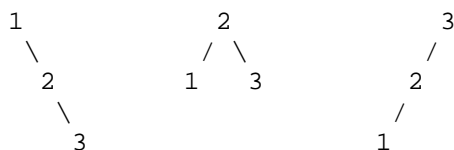
```

23. ABB = ^ABBNode;
24. ABBNode = RECORD
25.     info: INTEGER;
26.     left: ABB;
27.     right: ABB;
    END;
List = ^ListNode;
ListNode = RECORD
    data: INTEGER;
    next: List;
    END;

```

Se pide:

Definir una función que dados dos árboles binarios de búsqueda de naturales retorne true si, y solamente si, ambos árboles representan a un mismo conjunto. Por ejemplo para el conjunto {1, 2, 3} hay tres árboles binarios de búsqueda que lo representan:



28. Considere la definición de la función f que sigue:

```

29. PROCEDURE f(t: ABB; k: INTEGER): BOOLEAN;
30.     BEGIN
31.         IF t = NIL THEN
32.             RETURN (k=0)
33.         ELSE RETURN (f(t^.izq,k-1) AND f(t^.der,k-1))
34.     END f;

```

- . ¿Qué retorna la función f?
- a. ¿Cuántos nodos tiene el árbol si la función retorna true?. Justifique.