

Curso de Estructuras de Datos y Algoritmos – 2008 (Nocturno)

Práctico 4

Objetivos: Aplicar los conceptos de punteros a tipos inductivos con operaciones implementadas iterativamente y recursivamente. Ver los posibles efectos laterales que trae el manejo de memoria al trabajar con el lenguaje (imperativo) como C/C++.

1. Determine y justifique cuáles de las siguientes instrucciones son válidas, si se supone que se hicieron las siguientes definiciones y declaraciones:

```
#include <stdlib.h>
```

```
.....
```

```
typedef int* apunta;
```

```
typedef char * apuntb;
```

```
apunta ap1, ap2;
```

```
apuntb ap3, ap4;
```

```
.....
```

a) `ap1=new apunta;`

b) `delete* ap1;`

c) `ap1=ap2;`

d) `ap2=*ap2 + *ap1;`

e) `ap1=NULL;`

f) `ap4=NULL;`

g) `delete apuntb;`

h) `ap3=*ap4 * *ap1;`

i) `ap2=new(ap1);`

j) `ap2=new int[7];`

k) `ap2=new int;`

l) `ap2=new int(5);`

m) `delete ap3;`

n) `free (ap2);`

o) `ap1=(apunta)malloc(sizeof(int));`

p) `ap2=(apunta)malloc(sizeof(apunta));`

2.a) De una representación para la lista simple de naturales (LNat).

b) Implementar las siguientes operaciones funcionales:

■ **LNat Null()** ;

/* Crea la lista vacía */

■ **LNat Cons(int x, LNat l)**;

/* Inserta un elemento al principio de la lista */

■ **bool IsEmpty(LNat l)**;

/* Verifica si la lista está vacía */

■ **int Head(LNat l);**

/* Retorna, si la lista no es vacía, el primer elemento de la lista */

■ **LNat Tail(LNat l);**

/* Retorna, si la lista no es vacía, la lista sin su primer elemento */

c. Implementar las siguientes operaciones procedurales:

■ **void Null(LNat &l);**

/* Crea la lista vacía */

■ **void Cons(int x, LNat &l);**

■ **void IsEmpty(LNat l, bool &e);**

/* Verifica si la lista está vacía */

■ **void Head(LNat l, int &h);**

/* Retorna, si la lista no es vacía, el primer elemento de la lista */

■ **void Tail(LNat &l);**

/* Retorna, si la lista no es vacía, la lista sin su primer elemento */

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

○ **bool IsElement(int x, LNat l);**

/* Verifica si un natural dado pertenece a la lista */

○ **int Length(LNat l);**

/* Retorna la cantidad de elementos de la lista */

○ **int Last(LNat l);**

/* Retorna, si la lista l es no vacía, el último elemento */

○ **int Max(LNat l);**

/* Retorna, si la lista l es no vacía, su máximo elemento */

○ **float Average(LNat l);**

/* Retorna, si la lista l es no vacía, el promedio de sus elementos */

○ **LNat Insert(int x, LNat l);**

/* Dada la lista l ordenada, inserta a x en l ordenadamente */

○ **LNat Snoc(int x, LNat l);**

/* Inserta el elemento x al final de la lista l */

○ **LNat Remove(int x, LNat l);**

/* Elimina un natural dado de la lista l */

○ **bool Equals(LNat l, LNat p);**

/* Verifica si las listas l y p son iguales*/

○ **bool IsIncluded(LNat l, LNat p);**

/* Verifica si la lista p está incluida en la lista l */.

4. Implementar las siguientes operaciones accediendo directamente a la representación, iterativamente, sin usar procedimientos auxiliares y sin que las soluciones retornadas compartan memoria con los parámetros:

○ **LNat Take(int i, LNat l);**

/* Retorna la lista resultado de tomar los primeros i elementos */

- **LNat Drop(int i, LNat l);**
/* Retorna la lista resultado de no tomar los primeros i-1 elementos */
- **LNat Merge(LNat l, LNat p);**
/* Genera una lista fruto de intercalar ordenadamente las listas l y p, que vienen ordenadas */
- **LNat Append(LNat l, LNat p);**
/* Agrega la lista p al final de la lista l */

5. Implementar las siguientes operaciones recursivamente sin que las soluciones retornadas compartan memoria con los parámetros:

- **LNat Take(int I, LNat l);**
/* Retorna la lista resultado de tomar los primeros i elementos */
- **LNat Drop(int I, LNat l) ;**
/* Retorna la lista resultado de no tomar los primeros i-1 elementos */
- **LNat Merge(LNat l, LNat p);**
/* Genera una lista fruto de intercalar ordenadamente las listas l y p, que vienen ordenadas */
- **LNat Append(LNat l, LNat p);**
/* Agrega la lista p al final de la lista l */

6. 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:

- **LNat Null() ;**
/* Crea la lista vacia */
- **LNat Cons(int x, LNat l);**
/* Inserta un elemento al principio de la lista */
- **bool IsEmpty(LNat l);**
/* Verifica si la lista está vacia */
- **bool IsElement(int x, LNat l);**
/* Verifica si un natural pertenece a la lista */
- **LNat Remove(int x, LNat l);**
/* Elimina todas las ocurrencias del natural en la lista */
- **LNat Insert(int x, LNat l);**
/* Inserta ordenadamente */

7. 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:

- **LNat Null() ;**
/* Crea la lista vacia */
- **LNat Cons(int x, LNat l);**

- /* Inserta un elemento al principio de la lista */
- **bool IsEmpty(LNat l);**
- /* Verifica si la lista está vacía */
- **bool IsElement(int x, LNat l);**
- /* Verifica si un natural pertenece a la lista */
- **LNat Remove(int x, LNat l);**
- /* Elimina todas las ocurrencias del natural en la lista */
- **LNat Insert(int x, LNat l);**
- /* Inserta ordenadamente */

8. La tercer variante a la implementaciones clásicas de listas encadenadas es la llamada Lista Doblemente Encadenada y Circular. Dicha variación es la conjunción de las variaciones de los dos ejercicios anteriores. 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:

- **LNat Null() ;**
- /* Crea la lista vacía */
- **LNat Cons(int x, LNat l);**
- /* Inserta un elemento al principio de la lista */
- **bool IsEmpty(LNat l);**
- /* Verifica si la lista está vacía */
- **bool IsElement(int x, LNat l);**
- /* Verifica si un natural pertenece a la lista */
- **LNat Remove(int x, LNat l);**
- /* Elimina todas las ocurrencias del natural en la lista */
- **LNat Insert(int x, LNat l);**
- /* Inserta ordenadamente */

9. Las llamadas Listas Generales de Naturales son Listas Encadenadas donde cada elemento de la lista es una Lista Encadenada de Naturales (llamada sublista).

a. Dar una representación para la lista general y lista de naturales.

b. Implementar las siguientes operaciones para Lista Encadenada Circular de Naturales accediendo directamente a la representación, iterativamente y sin utilizar procedimientos auxiliares:

- **void Reverse(LGNat &l);**
- /* Invierte la lista general l sin invertir sus sublistas */
- **int Length(LGNat l) ;**
- /* Retorna la cantidad de naturales de la lista general l */
- **LGNat Show(LGNat l) ;**
- /* Muestra la lista general separando los ítems -naturales y sublistas- por comas y encerrando cada sublista entre paréntesis */

c. Implementar la siguiente operación para Lista Encadenada Circular de Naturales accediendo directamente a la representación y recursivamente:

- **LGNat ReverseAll(LGNat l) ;** /* Invierte la lista general l invirtiendo cada sublista */

Nota: Se puede utilizar un procedimiento auxiliar para invertir la lista de naturales

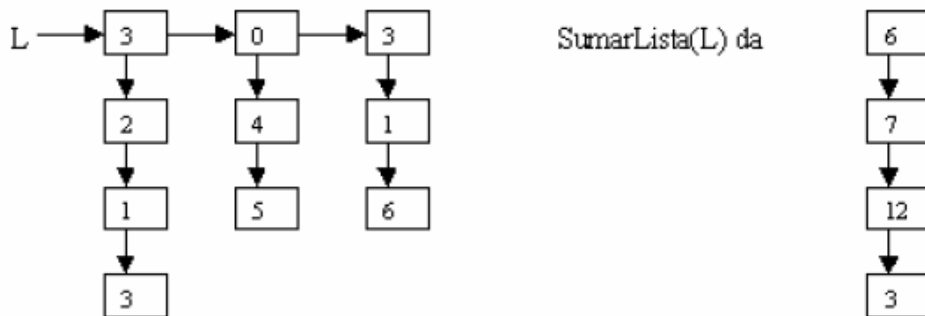
10. Considere la siguiente representación para las Listas Generales de Naturales:

```
Struct Cabezal{  
  LNat primero;  
  LGNat prox;  
}
```

```
Struct Nodo{  
  Int elem;  
  LNat sig;  
}  
typedef struct Nodo LNat;
```

```
typedef struct Cabezal* LGNat ;
```

Implementar una función “SumarLista” que dada una Lista General de Naturales devuelva la suma de todos los números de la lista del siguiente modo:

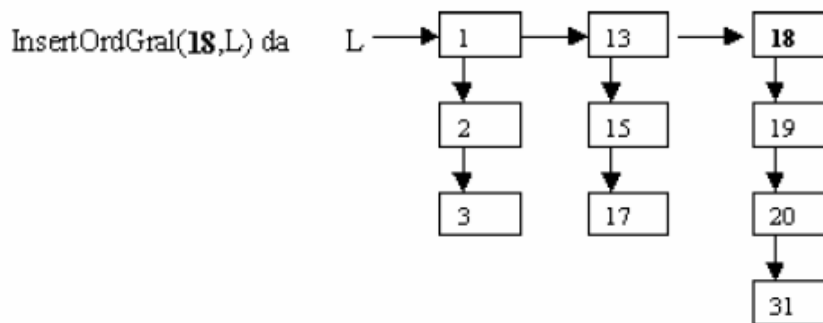
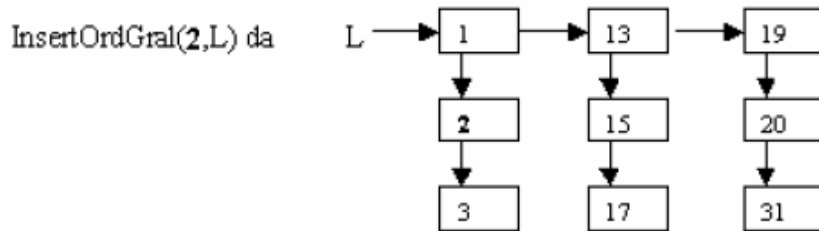
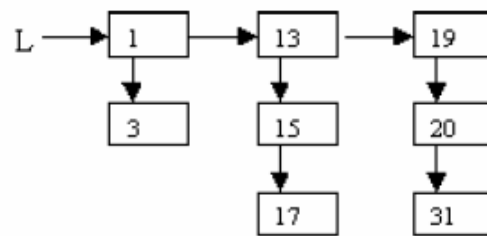


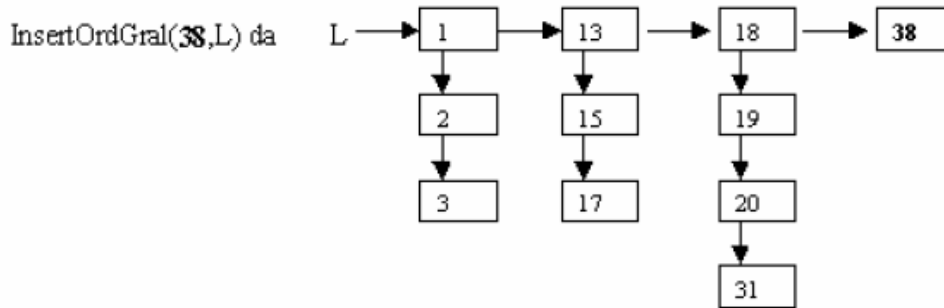
Sugerencia: Considere que la lista esta compuesta por lo menos por una sublista y que no se almacenan sublistas vacías.

11. Considere las Listas Generales de Naturales Ordenadas, las cuales son similares a las Listas Generales

de Naturales, pero además existe un orden entre las sublistas, de modo tal que los elementos de cada sublista son menores que los de la sublista siguiente y mayores que los de la sublista anterior.

Implementar un procedimiento recursivo “InsertOrdGral”, que inserte un natural en un Lista General de Naturales Ordenada. Si el natural es mayor que todos los elementos de una sublista y menor que todos los de la siguiente sublista, debe ser insertado al comienzo de la siguiente sublista (ver en la figura la inserción del 18). Si el elemento a insertar es mayor que todos los elementos se creará una nueva sublista que sólo contenga a ese elemento. Ejemplos:



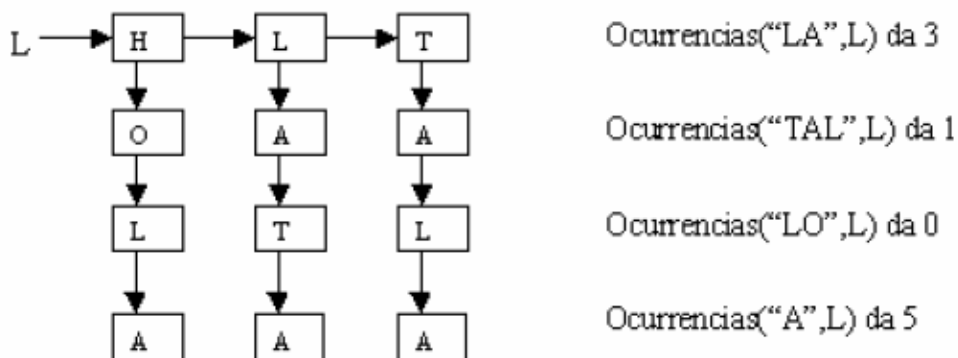


Sugerencia: Considere que la lista esta compuesta por lo menos por una sublista y que no se almacenan sublistas vacías.

12. Considere las Listas Generales de Caracteres, similar a la Listas Generales de Naturales pero compuestas por caracteres, que pueden ser utilizadas para representar una lista de palabras.

Implementar una función “Ocurrencias”, que devuelva la cantidad de ocurrencias de una secuencia de caracteres en las sublistas de una Lista General de Caracteres.

Ejemplo:

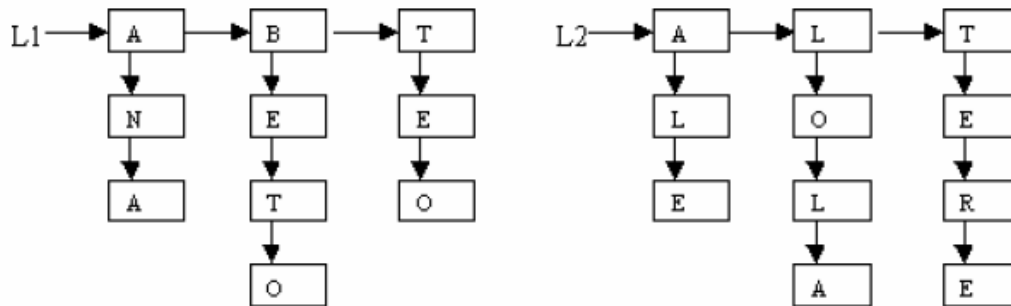


Sugerencia: Considere una lista de caracteres para representar la secuencia de caracteres, que la Lista General de Caracteres esta compuesta por lo menos por una sublista y que no se almacenan sublistas vacías.

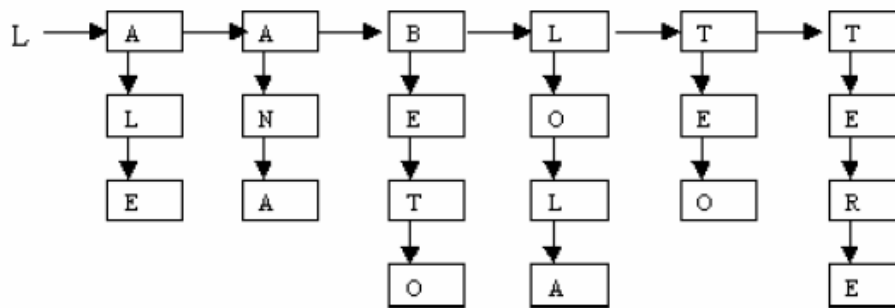
13. Dadas dos Listas Generales de Caracteres, L1 y L2, cuyas sublistas representan palabras, implementar un procedimiento “FusionOrdenada”, que genera una nueva Lista General Caracteres, con las sublistas de L1 y L2 de tal manera que las palabras quedan ordenadas alfabéticamente. Asumimos que L1 y L2 tienen sus palabras ordenadas alfabéticamente.

Observar que resulta conveniente implementar funciones auxiliares que determinen el orden alfabético de las sublistas.

Ejemplo:



IntercalarPalGral (L1, L2) da



Sugerencia: Considere que la lista esta compuesta por lo menos por una sublista y que no se almacenan sublistas vacías.