

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

Práctico 3

Objetivos: Continuar aplicando los fundamentos vistos en teórico sobre recursión y en particular utilizarlos sobre listas.

1) Dada la siguiente definición de función X en los naturales:

$$X(n) = f(n-1) + 5 + X(n-1) \text{ -Si } n \geq 1$$

$$X(n) = 3 \text{ -Si } n = 0;$$

Donde f es la función de fibonacci

- a) Escriba el Pseudocódigo y luego Implemente en forma iterativa a la función X
- b) Escriba el Pseudocódigo y luego Implemente en forma Recursiva la función X

2. Dadas las siguientes operaciones funcionales sobre lista de naturales (LNat):

- o **LNat Null();**
/* Crea la lista vacía */
- o **LNat Cons(int x, LNat l);**
/* Inserta un elemento al principio de la lista */
- o **bool IsEmpty(LNat l);**
/* Verifica si la lista está vacía */
- o **int Head(LNat l);**
/* Retorna, si la lista no es vacía, el primer elemento de la lista */
- o **LNat Tail(LNat l);**
/* Retorna, si la lista no es vacía, la lista sin su primer elemento */

Implementar las siguientes operaciones recursivamente:

- o **bool IsElement(int x, LNat l);**
/* Verifica si un natural dado pertenece a la lista */
- o **LNat Remove(int x, LNat l);**
/* Elimina un natural dado de la lista */
- o **int Length(LNat l);**
/* Retorna la cantidad de elementos de la lista */
- o **LNat Snoc(int x, LNat l);**
/* Inserta el elemento x al final de la lista l */
- o **LNat Append(LNat l, LNat p);**
/* Agrega la lista p al final de la lista l */
- o **LNat Reverse(LNat l);**
/* Invierte la lista l */
- o **LNat Insert(int x, LNat l);**
/* Inserta ordenadamente el elemento x en la lista ordenada l */
- o **LNat Sort(LNat l);**
/* Ordena la lista l */
- o **int Last(LNat l);**
/* Retorna, si la lista l es no vacia, su último elemento */

- **int HowMany(int x, LNat l);**
/* Cuenta las ocurrencias del natural n en la lista l */
- **int Max(LNat l);**
/* Retorna, si la lista l es no vacia, su máximo elemento */
- **int Nth(int n, LNat l);**
/* Retorna, si la lista contiene por lo menos n elementos, el elemento que se encuentra en la posición n de la lista l */
- **bool IsSorted(LNat l);**
/* Verifica si la lista esta ordenada */
- **LNat Change(int x, int y, LNat l);**
/* Retorna la lista resultado de cambiar el valor x por el valor y en la lista l
- **LNat InsBefore(int x, int y, LNat l);**
/* Inserta el elemento x inmediatamente antes del elemento y en la lista l */
- **LNat InsAround(int x, int y, LNat l);**
/* Inserta el elemento x delante y detrás del elemento y en la lista l */
- **bool Equals(LNat l, LNat p);**
/* Verifica si las listas l y p son iguales */
- **LNat Merge(LNat l, LNat p);**
/* Genera una lista fruto de intercalar ordenadamente las listas l y p, que vienen ordenadas*/
- **bool IsIncluded(LNat l, LNat p);**
/* Verifica si la lista p está incluida en la lista l */.
- **void Show(LNat l);**
/* Muestra los elementos de la lista */

3. Las llamadas Listas Generales de Naturales (LGNat) son Listas Encadenadas donde cada elemento de

la lista es una Lista Encadenada de Naturales (llamada sublista).

Dadas las operaciones sobre lista de naturales del ejercicio anterior y las siguientes operaciones de

Listas Generales de Naturales:

- **LGNat NullLG();**
/* Crea la lista general vacía */
- **LGNat ConsLG(LNat l, LGNat lg);**
/* Inserta un elemento al principio de la lista */
- **void IsEmptyLG(LGNat lg);**
/* Verifica si la lista está vacía */
- **LNat Head(LGNat l);**
/* Retorna, si la lista no es vacía, el primer elemento de la lista */
- **LGNat Tail(LGNat l);**
/* Retorna, si la lista no es vacía, la lista sin su primer elemento */

Implementar recursivamente las siguientes operaciones:

- **LGNat Reverse(LGNat lg);**
/* Invierte la lista general lg sin invertir sus sublistas */
- **LGNat; ReverseAll(LGNat lg);**
/* Invierte la lista general lg invirtiendo cada sublista */
- **int Length(LGNat lg);**
/* Retorna la cantidad de naturales de la lista general lg */

o **void Show(LGNat lg);**

/* Muestra la lista general separando los ítems -naturales y sublistas- por comas y encerrando cada sublista entre paréntesis */

4. Considere la siguiente representación de números binarios mediante Listas de dígitos: el número binario 1 (en base 10 es el 1) se representa mediante la lista 1
el número binario 10 (en base 10 es el 2) se representa mediante la lista 1,0
el número binario 110 (en base 10 es el 6) se representa mediante la lista 1,1,0

Implemente en Modula-2 un programa recursivo Incremento que le sume 1 a un número binario, sin

duplicar la lista. Algunos ejemplos de resultados son:

Incremento(1) = 1,0

Incremento(1,0) = 1,1

Incremento(1,1,0) = 1,1,1

Sugerencia: Considere como precondition de su procedimiento que la lista no puede ser vacía.

Utilizar las operaciones elementales del ejercicio 1.

5. Escriba un programa en C/C++ que dado un mapa representado por una matriz de tamaño N*M de

ceros y unos, donde 0 representa agua y 1 tierra, imprima las islas encontradas en el mismo. Una isla

es una agrupación de celdas (al menos una) con valor 1, cada una de las cuales es adyacente

horizontal y/o verticalmente a una celda del mismo grupo. Una isla está delimitada, horizontal y

verticalmente, por celdas con valor 0 o por los límites (fronteras) del mapa.

Por ejemplo, en el siguiente mapa de tamaño 5*10, hay 6 islas.

```
1 1 1 0 0 1 1 1 1 1
```

```
1 1 0 0 0 0 1 0 0 1
```

```
0 0 0 1 0 0 1 0 0 0
```

```
0 0 0 0 1 0 1 0 0 0
```

```
1 1 1 0 0 0 1 1 0 1
```

Las 6 islas están compuestas de las siguientes celdas, cuyas coordenadas son:

Isla 1: (1,1) (1,2) (1,3) (2,1) (2,2)

Isla 2: (3,4)

Isla 3: (4,5)

Isla 4: (5,1) (5,2) (5,3)

Isla 5: (1,6) (1,7) (1,8) (1,9) (1,10) (2,10) (2,7) (3,7) (4,7) (5,7) (5,8)

Isla 6: (5,10)

Notar que (3,4) y (4,5) no forman una isla sino que son dos islas distintas.

Se pide que el programa imprima las celdas de cada isla del mapa siguiendo el formato del ejemplo.

El orden de impresión entre las islas no es relevante, ni tampoco el orden en que se imprimen las

celdas de cada isla.