ESTRUCTURA DE DATOS Y ALGORITMOS
UTU, Buceo
PRACTICO 5

Ejercicio 1)

Dadas las siguientes operaciones funcionales sobre listas de naturales:

```
struct nodo{
int valor;
nodo *sig;};

typedef nodo *lista;


lista Null();
// Crea la lista vacia.

lista Cons(lista l,int x);
// Inserta el elemento x al principio de la lista l.

bool IsEmpty(lista l);
// Verifica si la lista esta vacia

int Head(lista l);
// Retorna si la lista no es vacia el primer elemento de la lista.

lista Tail(lista l);
// Retorna si la lista no es vacia, la lista sin su primer elemento.
```

Implemente las siguientes operaciones **recursivamente** utilizando las operaciones anteriores:

```
bool IsElement(int x,lista l);
// Verifica si un natural dado pertenece a la lista

lista Remove(int x, lista l);
// Elimina un natural dado de la lista

int Length(lista l);
// Retorna la cantidad de elementos de la lista

lista Snoc(int x, lista l);
// Inserta el elemento x al final de la lista l

lista Append(lista l, lista p);
// Agrega la lista p al final de la lista l

lista Reverse(lista l);
// Invierte la lista l
```

```
lista Insert(int x,lista l);
// Inserta ordenadamente el elemento x en la lista ordenada l

int Last(lista l);
// Retorna si la lista l no es vacia, su ultimo elemento

int HowMany(int x,lista l);
// Cuenta las ocurrencias del natural x en la lista l

int Max(lista l);
// Retorna si la lista l no es vacia, su maximo elemento

bool IsSorted(lista l);
// Verifica si la lista esta ordenada

lista Change(int x, int y, lista l);
// Retorna la lista resultado de cambiar x por y en l

lista InsBefore(int x,int y, lista l);
// Inserta el elemento x inmediatamente antes del elemento y en l

lista InsAround(int x,int y,lista l);
// Inserta x delante y atras de y en l

bool Equals(lista l,lista p);
// Verifica si las listas l y p son iguales

void Show(lista l);
// Muestra los elementos de la lista l.
```

Ejercicio 2)

Las llamadas listas generales de naturales son listas encadenadas donde cada elemento de la lista en una lista encadenada de naturales.

Implemente las siguientes operaciones manipulando la representación:

```
struct nodogral{
        lista lg;
        nodogral *sig;};

typedef nodogral *listagral;

listagral NullLG();
// Crea la lista general vacia.

listagral ConsLG(listagral l,lista x);
// Inserta el elemento x al principio de la lista general l.

bool IsEmptyLG(listagral l);
// Verifica si la lista general esta vacia.

lista HeadLG(listagral l);
```

// Retorna si la lista general no es vacia su primer elemento.

listagral TailLG(listagral l);
// Retorna si la lista general no es vacia, esta sin su primer elemento.

Implementar **recursivamente** las siguientes operaciones utilizando las operaciones anteriores y las del ejercicio anterior:

listagral ReverseAll(listagral l);
// Invierte cada lista de naturales de la lista general l.

int LengthLG(listagral l);
// Retorna la cantidad de naturales de la lista general l.

void ShowLG(listagral l);
// Muestra la lista general separando los naturales y listas de
//naturales por comas y encerrando cada lista de naturales entre
//parentesis.

Ejercicio 3)

Implemente las siguientes operaciones accediendo directamente a la representación **iterativamente** y sin usar procedimientos auxiliares.

Coloque en las operaciones si es necesario, que el argumento lista debe pasarse por referencia.

bool IsElement(int x,lista l);
// Verifica si un natural dado pertenece a la lista

int Length(lista l);
// Retorna la cantidad de elementos de la lista

int Last(lista l);
// Retorna si la lista l no es vacia, su ultimo elemento

int Max(lista l);
// Retorna si la lista l no es vacia, su maximo elemento

float Average(lista l);
// Retorna si la lista no es vacia el promedio de sus elementos

lista Insert(int x,lista l);
// Inserta ordenadamente el elemento x en la lista ordenada l

lista Snoc(int x, lista l);
// Inserta el elemento x al final de la lista l

lista Remove(int x,lista l);
// Elimina un natural dado de la lista l

bool Equals(lista l,lista p);

// Verifica si las listas l y p son iguales

Ejercicio 4)

Implemente las siguientes operaciones accediendo directamente a la representación **iterativamente** y sin usar procedimientos auxiliares y sin que las soluciones retornadas compartan memoria con los parametros.

Coloque en las operaciones si es necesario, que el argumento lista debe pasarse por referencia.

```
lista Take(int i,lista l);
// Retorna la lista resultado de tomar los primeros i elementos.

lista Drop(int u,lista l);
// Retorna la lista resultado de no tomar los primeros i elementos.

lista Merge(lista l,lista p);
// Genera una lista fruto de intercalar ordenadamente las listas
// l y p que vienen ordenadas.

lista Append(lista l,lista p);
// Agrega la lista p al final de la lista l.
```

Ejercicio 5)

Implemente las siguientes operaciones **recursivamente** sin que las soluciones retornadas compartan memoria con los parametros.

Coloque en las operaciones si es necesario, que el argumento lista debe pasarse por referencia.

```
lista Take(int i,lista l);
// Retorna la lista resultado de tomar los primeros i elementos.

lista Drop(int u,lista l);
// Retorna la lista resultado de no tomar los primeros i elementos.

lista Merge(lista l,lista p);
// Genera una lista fruto de intercalar ordenadamente las listas
// l y p que vienen ordenadas.

lista Append(lista l,lista p);
// Agrega la lista p al final de la lista l.
```