Práctico Listas

- ## Ejercicio 1

Dada la siguiente definición de Lista:

```
typedef struct nodo_lista * lista;

struct nodo_lista{
   int dato;
   lista sig;
}
```

Implemente ***iterativamente*** las siguientes operaciones accediendo directamente a la representación y sin usar procedimientos auxiliares *(Si considera necesario, puede pasar la lista por referencia agregando &).*

```
bool IsElement(int x, lista l);
// Retorna true si x pertenece a l, false en caso contrario.

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

int Last(lista l);
// Retorna el último elemento de l.
// Pre: l no es vacía.

int Max(lista l);
// Retorna el máximo elemento de l.
// Pre: l no es vacía.

float Average(lista l);
// Retorna si la lista no es vacía el promedio de sus elementos.
// Pre: l no es vacía.

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 todas las ocurrencias de x en la lista l

bool Equals(lista l, lista p);
// Verifica si las listas l y p son iguales (mismos elementos en el mismo orden).
```

- ## Ejercicio 2

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

```
lista Take(int i, lista l);
// Retorna la lista resultado de tomar los primeros i elementos.
// l no comparte memoria con la lista resultado.

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

lista Merge(lista l, lista p);
// Genera una lista fruto de intercalar ordenadamente las listas
// l y p que vienen ordenadas.
// l y p no comparten memoria con la lista resultado.

lista Append(lista l, lista p);
// Agrega la lista p al final de la lista l.
// l y p no comparten memoria con la lista resultado.
```

- ***Ejercicio 3***

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

```
lista Take(int i, lista l);
// Retorna la lista resultado de tomar los primeros i elementos.
// l no comparte memoria con la lista resultado.

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

lista Merge(lista l, lista p);
// Genera una lista fruto de intercalar ordenadamente las listas
// l y p que vienen ordenadas.
// l y p no comparten memoria con la lista resultado.

lista Append(lista l, lista p);
// Agrega la lista p al final de la lista l.
// l y p no comparten memoria con la lista resultado.
```

- ***Ejercicio 4***

Dada la siguiente definición del TAD Lista:

```
typedef struct nodo_lista * lista;

struct nodo_lista{
   int dato;
   lista sig;
}

lista Null();
// Crea la lista vacía.

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

bool IsEmpty(lista l);
// Retorna true si l es vacía, false en caso contrario.

int Head(lista l);
// Retorna el primer elemento de la lista.
// Pre: l no vacía.

lista Tail(lista l);
// Retorna la lista sin su primer elemento.
// Pre: l no vacía.
```

Implemente las siguientes operaciones ***recursivamente*** utilizando exclusivamente las operaciones anteriores *(sin acceder a la representación interna)*:

```
bool IsElement(int x, lista l);
// Retorna true si x pertenece a l, false en caso contrario.

lista Remove(int x, lista l);
// Retorna la lista fruto de eliminar x en l.
// l no comparte memoria con la lista resultado.

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

lista Snoc(int x, lista l);
// Retorna la lista fruto de insertar el elemento x al final de la lista l.
// l no comparte memoria con la lista resultado.

lista Append(lista l, lista p);
// Retorna la lista fruto de agregar la lista p al final de la lista l.
// l y p no comparten memoria con la lista resultado.
```

```
lista Insert(int x, lista l);
// Retorna la lista fruto de insertar ordenadamente el elemento x en la lista ordenada l.
// l no comparte memoria con la lista resultado.

int Last(lista l);
// Retorna el último elemento.
// Pre: l no vacía.

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

int Max(lista l);
// Retorna el máximo elemento de l.
// Pre: l no vacía.

bool IsSorted(lista l);
// Retorna true si l está ordenada, false en caso contrario.

lista Change(int x, int y, lista l);
// Retorna una nueva lista fruto de cambiar x por y en l.
// l no comparte memoria con la lista resultado.

lista InsBefore(int x, int y, lista l);
// Retorna una nueva lista fruto de insertar x antes de y en l.
// l no comparte memoria con la lista resultado

lista InsAround(int x, int y, lista l);
// Retorna una nueva lista fruto de insertar x antes y después de y en l.
// l no comparte memoria con la lista resultado.

bool Equals(lista l, lista p);
// Retorna true si las listas l y p son iguales (mismos elementos en el mismo orden)
// false en caso contrario.

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

- ***Ejercicio 5***

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

```
typedef nodo_listagral * listagral;

struct nodo_listagral{
   lista dato;
   listagral sig;
}
```

```
typedef struct nodo_lista * lista;

struct nodo_lista{
      int dato;
      lista sig;
}
```

Implemente las siguientes operaciones manipulando la representación interna:

```
listagral NullLG();
// Crea la lista general vacía.

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

bool IsEmptyLG(listagral lg);
// Verifica si la lista general está vacía.

lista HeadLG(listagral lg);
// Retorna la primer lista elemento.
// Pre: lg no vacía.

listagral TailLG(listagral lg);
// Retorna lg sin su primer elemento.
// Pre: lg no vacía.
```

Implementar ***recursivamente*** las siguientes operaciones utilizando las operaciones anteriores y las del Ejercicio 4:

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

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