Seguimiento 4

Integrantes:

- Alejandro Varela Franco
- Gabriel Suárez Baron

TAD-LinkedList

TAD → Node		
Node $\{element = < element >, next = < next >\}$		
$inv = \{\}$		
Operaciones elementales		
Método	**Entrada**	**Retorno**
Node		Node
Next		Node
SetNext	New Element	Node

Node ()

"Constructor del nodo"

{*pre:* True}
{*pos:* $Node = \{Next < next >\}$}

Next ()

"Obtiene el siguiente nodo del elemento"

{*pre:* element != null}
{*pos:* element.Next}

SetNext(New Element)

"Cambia el valor del siguiente nodo"

{*pre:* element != null ^ element.Next != null}
{*pos:* element.Next = new element}

TAD → LinkedList		
LinkedList = $\{FirstNode = < firstNode >\}$		
Tenemos que $S = LinkedList, n = tamaño\ de\ la\ lista\ y\ Z = nodo\ de\ la\ lista$ $\{inv: S = Z_{n1}, Z_{n2}, Z_{n3} \dots Z_{n-1}, Z_n\ tales\ que\ Z_{n1} \leq Z_{n2} \leq Z_{n3} \dots \leq Z_{n-1} \leq Z_n\}$		
Operaciones elementales		
Método	**Entrada**	**Retorno**
LinkedList		LinkedList
AddNode	Element	LinkedList
Size	LinkedList	Integer
EditNode	Position x Integer, Element	LinkedList
isEmpy	LinkedList	Boolean
RemoveNode	Position x Integer	LinkedList

LinkedList()

"Constructor de la lista enlazada, crear una lista enlazada"

{*pre:* True}
{*pos:* $LinkedList = \{firstNode: null\}$}

AddNode()

"Añade un nodo a la lista enlazada".

Caso 1
Si la lista enlazada no tiene un elemento primero, el nuevo elemento se añadirá en este.

{*pre:* LinkedList.FirstNode != null ^ element != null}
{*pos:* LinkedList.FirstNode = element}

Caso 2

Si la lista enlazada tiene un elemento primero y el siguiente es nulo, se comparará si el nuevo elemento es menor al primero, entonces el primero será el nuevo elemento y el siguiente será el primer elemento.

{*pre:* LinkedList.FirstNode != null ^ element != null ^ element < LinkedList.FirstNode}
{*pos:* element.Next = LinkedList.FirstNode ^ LinkedList.FirstNode = element}

3 Caso

Si la lista tiene más de 2 elementos, entonces se recorrerá la lista buscando posicionar el elemento entre un elemento mayor y uno menor

{*pre:* La lista debe tener más de 2 elementos ^ element != null}
{*pos:* Lista ordenada con el nuevo elemento}

Size()

“Retorna el tamaño de la lista enlazada”

{*pre:* True}
{*pos:* Tamaño de la lista enlazada}

isEmpty()

“Retorna un booleano donde determina si la lista está vacía o no”

1 Caso
{*pre:* LinkedList.Size() == 0}
{*pos:* True}

2 Caso
{*pre:* LinkedList.Size() != 0}
{*pos:* False}

SearchNode(pos)

“Busca un nodo en una determinada posición, la LinkedList no tiene que estar vacía y la posición a buscar no debe ser mayor a el tamaño de la lista ”

{pre: ¬LinkedList.isEmpty() ^ pos <= LinkedList.Size()}
{pos: element}

EditNode(pos, Element)

“Busca un nodo a eliminar, luego de esto, añade otro nodo, utilizará los siguientes métodos, RemoveNode y AddNode para que este funcione

{pre: pos < LinkedList.Size() ^ ¬LinkedList.isEmpty() ^ Element != null, la cuenta empieza desde 0}
{pos: ...LinkedList.FirstNode.Next… < Element < Element.Next...}

RemoveNode(pos)

“Eliminar un elemento de la lista enlazada, la posición debe ser menor que el tamaño de la lista”

{pre: ¬LinkedList.isEmpty() ^ pos < LinkedList.Size(), la cuenta empieza desde 0}
{pos: LinkedList.Size() = LinkedList.Size() - 1}