Implementation of a list with a doubly linked list

Variables

size: integer
sequence: doubly linked list with dummy nodes at the beginning and the end; each node, apart from the nodes n_0 and n_{m+1}, contains an element of the list

$$\begin{tikzcd}
n_0 \arrow[r, bend left] & n_1 \arrow[l, bend left] \arrow[r, bend left] & n_2 \arrow[l, bend left] \arrow[r, bend left] & \cdots \arrow[l, bend left] \arrow[r, bend left] & n_{m-1} \arrow[l, bend left] \arrow[r, bend left] & n_m \arrow[l, bend left] \arrow[r, bend left] & n_{m+1} \arrow[l, bend left]
\end{tikzcd}$$

head: pointer to node
tail: pointer to node
invariant: the nodes $n_1, \ldots, n_m$ of *sequence* contain the elements of the list listed from first to last element. *size* is the size of the list. *head* points to n_0 and *tail* points to n_{m+1}.

Initialization

size $\leftarrow 0$

sequence $\leftarrow$ $\begin{tikzcd} n_0 \arrow[r, bend left] & n_1 \arrow[l, bend left] \end{tikzcd}$

head points to n_0
tail points to n_1

Algorithms

```
size()
  output: size of list
return size
```

```
isEmpty()
  output: list is empty?
return ($size = 0$)
```

```
elements():
  output: collection of elements of the list
col ← empty collection
node ← second node of sequence
while $node \neq tail$ do
loop invariant: col contains the elements of nodes after head and before node
    add element of node to col
return col
```

```
positions():
  output: collection of positions of the list
col ← empty collection
node ← second node of sequence
while $node \neq tail$ do
loop invariant: col contains the nodes after head and before node
    add node to col
return col
```

```
first():
  precondition: list is nonempty
  output: first position of list
return second node of sequence
```

```
last():
  precondition: list is nonempty
  output: last position of list
return one but last node of sequence

before(position):
  precondition: position is not the first position of list
  output: position of list before position
return node before position in sequence

after(position):
  precondition: position is not the last position of list
  output: position of list after position
return node after position in sequence

isFirst(position):
  output: is position first position of list?
return (position = first())

isLast(position):
  output: is position last position of list?
return (position = last())

replaceElement(position, element):
  postcondition: element at position in list has been replaced with element
  input: position element of which is to be replaced with element
  output: replaced element
temp ← element of position
element of position ← element
return temp

swapElements(first, second):
  postcondition: elements of first and second have been swapped
  input: positions elements of which are to be swapped
swap elements of first and second

insertFirst(element):
  postcondition: position with element has been inserted at the beginning of list
  input: element to be inserted
  output: position of inserted element
node ← node with element
insert node in between the first and second node of sequence
size ← size + 1
return node

insertLast(element):
  postcondition: position with element has been inserted at the end of list
  input: element to be inserted
  output: position of inserted element
node ← node with element
insert node in between the last and one but last node of sequence
size ← size + 1
return node

insertBefore(position, element):
  postcondition: position with element has been inserted before position in list
  input: element to be inserted before position
  output: position of inserted element
```

```
$node \leftarrow$ node with $element$
insert $node$ before $position$ in $sequence$
$size \leftarrow size + 1$
**return** $node$

insertAfter($position$, $element$):
  $postcondition$: position with $element$ has been inserted after $position$ in list
  $input$: $element$ to be inserted after $position$
  $output$: position of inserted element
$node \leftarrow$ node with $element$
insert $node$ after $position$ in $sequence$
$size \leftarrow size + 1$
**return** $node$

remove($position$):
  $postcondition$: $position$ has been removed from list
  $input$: position to be removed
  $output$: element of removed position
$temp \leftarrow$ element of $position$
remove $position$ from $sequence$
$size \leftarrow size - 1$
**return** $temp$
```