Implementation of a vector with an array

Variables

sequence: array of elements
size: integer
invariant: $sequence[0], \ldots, sequence[size - 1]$ are the elements of the vector

Initialization

size $\leftarrow 0$

Algorithms

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

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

```
elemAtRank($rank$):
   precondition: $rank$ is valid[1]
   input: rank of element to be returned
   output: element at $rank$
return $sequence[rank]$
```

```
replaceAtRank($rank$, $element$):
   precondition: $rank$ is valid
   postcondition: element at $rank$ has been replaced by $element$
   input: rank of element to be replaced and replacement element
   output: replaced element
$temp \leftarrow sequence[rank]$
$sequence[rank] \leftarrow element$
return $temp$
```

```
insertAtRank($rank$, $element$):
   precondition: $rank$ is valid or $rank = size$, and $sequence$ is not full
   postcondition: $element$ has been inserted at $rank$
   input: element to be inserted and rank at which element has to be inserted
move $sequence[rank], \ldots, sequence[size - 1]$ one position to the right        (1)
$sequence[rank] \leftarrow element$
$size \leftarrow size + 1$
Ad (1):
for $i = size - 1, \ldots, rank$ do
      loop-invariant: $\forall j : i < j < size : sequence[j]$ has been moved one position to the right
      $sequence[i + 1] \leftarrow sequence[i]$
```

```
removeAtRank($rank$):
   precondition: $rank$ is valid
   postcondition: element at $rank$ has been removed
   input: rank of the element to be removed
   output: element at $rank$
$temp \leftarrow sequence[rank]$
```

[1] $rank$ is invalid if $rank < 0 \vee rank \geq size$.

```
move $sequence[rank+1], \ldots, sequence[size-1]$ one position to the left    (2)
$size \leftarrow size - 1$
**return** $temp$
```

Ad (2):

```
**for** $i = rank+1, \ldots, size-1$ **do**
    loop-invariant: $\forall j : rank < j < i : sequence[j]$ has been moved one position to the left
    $sequence[i-1] \leftarrow sequence[i]$
```

Implementation of a list with a circular array

Variables

sequence: array of positions; each position contains an element and an index
first: integer
last: integer
capacity: integer
invariant: if $first \leq last$, then the elements stored in the positions $sequence[first], \ldots, sequence[last-1]$ are the elements of the list; otherwise, the elements stored in the positions $sequence[first], \ldots, sequence[capacity - 1], sequence[0], \ldots, sequence[last-1]$ are the elements of the list; the indices stored in the positions correspond to the indices of the array, that is, the index stored in position $sequence[i]$ is i; *capacity* is the capacity of the array *sequence*

Initialization

```
$first \leftarrow 0$
$last \leftarrow 0$
$capacity \leftarrow$ capacity of the array
```

Algorithms

```
length($begin$, $end$):
  input: indices of array sequence
  output: length of the segment of sequence from (and including) begin upto (and excluding) end
**return** $(capacity + end - begin) \bmod capacity$
```

```
leftOf($index$):
  input: index of array sequence
  output: index of cell to the left of index
**return** $(capacity + index - 1) \bmod capacity$
```

```
rightOf($index$):
  input: index of array sequence
  output: index of cell to the right of index
**return** $(index + 1) \bmod capacity$
```

```
moveLeft($begin$, $end$):
  input: indices of array sequence
  output: move the segment of sequence from (and including) begin upto (and excluding) end one position
      to the left
$index \leftarrow begin$
**while** $index \neq end$ **do**
loop invariant: $sequence[begin], \ldots, sequence[\text{leftOf}(index)]$ have been moved one position to the left
    $sequence[\text{leftOf}(index)] \leftarrow sequence[index]$
    index of $sequence[\text{leftOf}(index)] \leftarrow \text{leftOf}(index)$
    $index \leftarrow \text{rightOf}(index)$
```

```
moveRight(begin, end):
  input: indices of array sequence
  output: move the segment of sequence from (and including) begin upto (and excluding) end one position
      to the right
index $\leftarrow$ end
while index $\neq$ begin do
loop invariant: sequence[index], $\ldots$, sequence[leftOf(end)] have been moved one position to the right
    sequence[index] $\leftarrow$ sequence[leftOf(index)]
    index of sequence[index] $\leftarrow$ index
    index $\leftarrow$ leftOf(index)

size():
  output: size of list
return length(first, last)

isEmpty():
  output: list is empty?
return (first $=$ last)

first():
  precondition: list is nonempty
  output: first position of list
return sequence[first]

last():
  precondition: list is nonempty
  output: last position of list
return sequence[leftOf(last)]

before(position):
  precondition: position is not first position and position is valid[2]
  output: position of list before position
index $\leftarrow$ index of position
return sequence[leftOf(index)]

after(position):
  precondition: position is not last position and position is valid
  output: position of list after position
index $\leftarrow$ index of position
return sequence[rightOf(index)]

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

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

replace(position, element):
  precondition: position is valid
  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
element $\leftarrow$ element of position
```

[2] *position* is valid if it is part of the list

```
element of position ← element
return element

swap(first, second):
  precondition: first and second are valid
  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):
  precondition: array is not full[3]
  postcondition: position with element has been inserted at the beginning of list
  input: element to be inserted
  output: position of inserted element
first ← leftOf(first)
position ← position with element and first
sequence[first] ← position
return position

insertLast(element):
  precondition: array is not full
  postcondition: position with element has been inserted at the end of list
  input: element to be inserted
  output: position of inserted element
position ← position with element and last
sequence[last] ← position
last ← rightOf(last)
return position

insertBefore(position, element):
  precondition: array is not full and position is valid
  postcondition: position with element has been inserted before position in list
  input: element to be inserted before position
  output: position of inserted element
index ← index of position
if length(first, index) ≤ length(index, last) then
    moveLeft(first, index)
    temp ← position with element and leftOf(index)
    sequence[leftOf(index)] ← temp
    first ← leftOf(first)
else
    moveRight(index, last)
    temp ← position with element and index
    sequence[index] ← temp
    last ← rightOf(last)
return temp

insertAfter(position, element):
  precondition: array is not full and position is valid
  postcondition: position with element has been inserted after position in list
  input: element to be inserted after position
  output: position of inserted element
if position = last() then
    return insertLast(element)
```

[3] $capacity - \mathsf{size}() \geq 2$

```
else
    return insertBefore(after(position), element)

remove(position):
  precondition: position is valid
  postcondition: position has been removed from list
  input: position to be removed
  output: element of removed position
element $\leftarrow$ element of position
index $\leftarrow$ index of position
if length(first, index) $\leq$ length(rightOf(index), last) then
    moveRight(first, index)
    first $\leftarrow$ rightOf(first)
else
    moveLeft(rightOf(index), last)
    last $\leftarrow$ leftOf(last)
return element
```