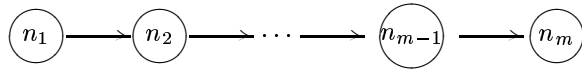


Implementation of a stack with a singly linked list without dummy nodes

Variables

size: integer

stack: singly linked list, each node of which contains an element of the stack



top: pointer to node

invariant: the nodes n_1, \dots, n_m of *stack* contain the elements of the stack listed from top to bottom. *size* is the size of the stack. *top* points to n_1 if this nodes exists, and points to nothing otherwise.

Initialization

size $\leftarrow 0$

stack \leftarrow empty linked list

top points to nothing

Algorithms

size():

output: size of stack

return *size*

isEmpty():

output: stack is empty?

return (*size* = 0)

top():

precondition: stack is nonempty

output: top element of stack

return element of the first node of *stack*

push(*element*):

postcondition: *element* has been added onto top of stack

input: *element* to be added to stack

node \leftarrow new node containing *element*

if *stack* is empty **then**

stack \leftarrow list consisting of *node*

else

add *node* at the beginning of *stack*

top points to *node*

size \leftarrow *size* + 1

pop():

precondition: stack is nonempty

postcondition: top element has been removed from stack

output: top element of stack

temp \leftarrow element of first node of *stack*

remove first node from *stack*

if *stack* is empty **then**

top points to nothing

else

top points to first node of *stack*

size \leftarrow *size* - 1

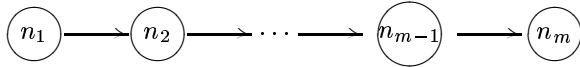
return *temp*

Implementation of a queue with a singly linked list without dummy nodes

Variables

size: integer

queue: singly linked list, each node of which contains an element of the queue



head: pointer to node

tail: pointer to node

invariant: the nodes n_1, \dots, n_m of *queue* contain the elements of the queue listed from front to rear. *size* is the size of the queue. *head* points to n_1 and *tail* points to n_m if these nodes exist, and point to nothing otherwise.

Initialization

size $\leftarrow 0$

queue \leftarrow empty linked list

head points to nothing

tail points to nothing

Algorithms

size():

output: size of queue

return *size*

isEmpty():

output: queue is empty?

return (*size* = 0)

front():

precondition: queue is nonempty

output: front element of stack

return element of the first node of *queue*

enqueue(*element*):

postcondition: *element* has been added to rear of queue

input: *element* to be added to queue

node \leftarrow new node containing *element*

if *queue* is empty **then**

queue \leftarrow list consisting of *node*

head and *tail* point to *node*

else

add *node* at the end of *queue*

tail points to *node*

size \leftarrow *size* + 1

dequeue():

precondition: queue is nonempty

postcondition: front element has been removed from queue

output: front element of queue

temp \leftarrow element of first node of *queue*

remove first node from *queue*

if *queue* is empty **then**

head and *tail* point to nothing

```

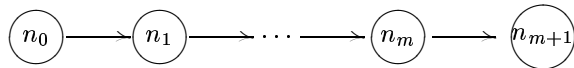
else
    head points to first node of queue
    size ← size - 1
    return temp

```

Implementation of a queue with a singly linked list with dummy nodes

Variables

size: integer
queue: singly linked list with all nodes, except for the first and the last one, containing an element of the queue



head: pointer to node
tail: pointer to node
invariant: the nodes n_1, \dots, n_m of *queue* contain the elements of the queue listed from front to rear. *size* is the size of the queue. *head* points to n_0 and *tail* points to n_{m+1} .

Initialization

```

size ← 0
queue ← (n0) → (n1)
head points to n0
tail points to n1

```

Algorithms

```

size():
    output: size of queue
    return size

isEmpty():
    output: queue is empty?
    return (size = 0)

front():
    precondition: queue is nonempty
    output: front element of stack
    return element of the second node of queue

enqueue(element):
    postcondition: element has been added to rear of queue
    input: element to be added to queue
    store element in the last node of queue
    add a new dummy node to the end of queue
    tail points to new node
    size ← size + 1

dequeue():
    precondition: queue is nonempty
    postcondition: front element has been removed from queue
    output: front element of queue
    temp ← element of second node of queue
    remove second node from queue
    size ← size - 1
    return temp

```

Implementation of a deque with a doubly linked list with dummy nodes

Variables

size: integer

deque: doubly linked list with dummy nodes at the front and the rear; each node, apart from the dummy nodes n_0 and n_{m+1} , contains an element of the deque



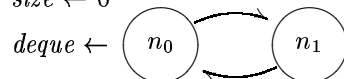
header: pointer to node

trailer: pointer to node

invariant: the nodes n_1, \dots, n_m of *deque* contain the elements of the deque listed from front to rear. *size* is the size of the deque. *header* points to n_0 and *trailer* points to n_{m+1} .

Initialization

$size \leftarrow 0$



header points to n_0

trailer points to n_1

Algorithms

size()

output: size of deque

return *size*

isEmpty()

output: deque is empty?

return ($size = 0$)

first()

precondition: deque is nonempty

output: element at the front of deque

return element of second node of *deque*

last()

precondition: deque is nonempty

output: element at the rear of deque

return element of one but last node of *deque*

insertFirst(*element*)

postcondition: *element* has been added to the front of deque

input: element to be added to deque

add new node with element *element* in between *header* and the second node of *deque*

$size \leftarrow size + 1$

insertLast(*element*)

postcondition: *element* has been added at the rear of *deque*

input: element to be added to deque

add new node with element *element* in between *trailer* and the one but last node of *deque*

$size \leftarrow size + 1$

removeFirst()

precondition: deque is nonempty

postcondition: first element has been removed from deque
output: first element of deque
 $temp \leftarrow$ element of second node of *deque*
remove second node from *deque*
 $size \leftarrow size - 1$
return *temp*

removeLast()

precondition: deque is nonempty
postcondition: last element has been removed from deque
output: last element of deque
 $temp \leftarrow$ element of one but last node of *deque*
remove one but last node from *deque*
 $size \leftarrow size - 1$
return *temp*