TAD

Names:

- **Leidy Daniela Londoño – A00392917**
- **Isabella Huila Cerón – A00394751**

STACK ADT
Stack = (e1, e2, e3, …., on), top)
inv: $0 \leq n \wedge Size(Stack) = n \wedge top = e_n$
Operations: ● **Stack - → Stack** ● **Push Stack element → Stack×** ● **Pop Stack → Stack** ● **Top Stack → Element** ● **IsEmptyStack → Boolean**

Stack

Builds an empty stack

before :-

postCondition : Stack s=∅

Push(element, n)

add a new e to Stacks

pre: Stack s= (e1,e2,e3….en) and element e or s =∅
and element e

post: Stack s= (e1,e2,e3,..., en, e) or s=(e)

Pop

Extracts from the stack s, the most recently inserted element.

pre: Stack if ies = (e1,e2,e3,...en)$\neq \emptyset$
post: Stack s = (e1,e2,e3,...)e_{n-1}

Top

Recovers the value of the element of the stack.

pre:Stacks$\neq \emptyset \;\; i.e.s \;=\; (e1, e2, e3, \ldots., en)$
post: Elemente_n

isEmpty

Determine if the Stack is empty or not.

pre:Stacks
post : True if s = .false if s$\emptyset \neq \emptyset$

QUEUE ADT

Queue = (e1,e2,e3,....,en),front,back)

inv:$0 \leq n \wedge size(queue) = n \wedge front = e1 \wedge back = n$

- **Queue -** $\rightarrow$ **Queue**
- **EnQueue Queue** $\rightarrow$ **Queue**$\times$ $element$
- **deQueue Queue** $\rightarrow$ **element**
- **front Queue** $\rightarrow$ **element**
- **IsEmpty Queue** $\rightarrow$ **Boolean**

queue

Builds and empty queue

pre:-
post: queue q =∅

EnQueue(Element)

insert a new element to the back of the queue q.

pre: Queue q = (e1,e2,e3,...., en) and element e or q= and element e.∅

post: queue q = (e1,e2,e3,....., en, e) or q =(e)

deQueue

Extracts the element in Queue q´s front.

pre: Queue q ≠∅ $i.e. q = (e1, e2, e3, \ldots, en)$

post: Queue q = (e2,e3,e4,...,) and element e1e_{n-1}

Front

Recovers the value of the item on the front of the queue

pre: Queue qq = (e1,e2,e3……, in)≠ ∅ $i.e.$

post : element e1.

isEmpty

Determines if the Queue q is empty or not.

pre: Queue q.

post : True if q =∅, false if q≠ ∅

TAD HashTable

HashTable=(table=<, , … >) List=(list=<Node1,

Node2,Node3 … Noden>)$List_1 List_1 List_3 List_n$

Inv:
h(x) ∈ [0, m), where m is the size of the hash table array.
If x ∈ S, then x is stored at position H[h(x)] of the hash table.
If x ∉ S, then position H[h(x)] of the hash table is empty.

primitive operations

- HashTable ➔ HashTable
- Get: HashTable X key ➔ Value
- AddElement: HashTable X Key X Value ➔ HashTable
- Delete: HashTable X Key ➔ Value
- Hash: HashTable X Key

HashTable

A hash table is created

pre:True

post:HashTable:{table:[]}

Get(key)

Searches for an element of the Hash table given the key and returns the value of the element

pre: hashtable != NIL$\wedge\ value \in T$

post: if element != NIL → return list.value, else → return NIL

AddElement(key, value)

Add an element to the hashtable using the key and value.

pre: hashTable !=NIL$\wedge\ value \in T$

post: newElement = {Key : key, value : value) element hashTable$\wedge\in$

Delete(key)

Remove element from hashtable given a key

pre: hashtable != NIL key K$\wedge\in$

post: element = {Key : key, value : value)
$\wedge$**element hashTable**$\notin$

Hash(String)

Calculates the position where the element should be inserted in the hash table.

pre: hashtable != NIL key K$\wedge\in$

post : A fixed-length string