|  |
| --- |
| **ADT Hash Table** |
| HT = {size=<size>, table=<table>}  table =<Node1, Node2, Node3, Noden>  Node = <K, V, next> |
| {inv: HT.table.length = HT.size ^ ∀ x, y ∈ HT.table, x ≠ y ⇒ hash(x) ≠ hash(y) } |
| Constructor operations:   * HT: <size> -> HT   Analysis operations:   * getValue HT x K - -> V * hash K -> Integer   Modification operations:   * add HT x K x V -> String * remove HT x K - -> String |

|  |
| --- |
| **HashTable()**  "Creates a new HashTable"  { pre: TRUE }  { post: HashTable = {table = <table>} } |

|  |
| --- |
| **put(HashTable, k, v)**  "Adds a new node to the hash table"  { pre: TRUE}  {post: If there is no collision, Table<newNode, …, …>a new node is added to the table. If there is a collision, the new node is added at the end Table<Node->newNode, …, …> . Return a message with the operation result } |

|  |
| --- |
| **get(HashTable, K)**  "Returns the value associated with the given key"  { pre: TRUE }  { post: Returns the value associated with the given key or null if it is not found } |

|  |
| --- |
| **remove(HashTable, K)**  "Removes the node associated with the given key from the hash table"  { pre: TRUE }  { post: Returns a message indicating if the node was found and removed or if it was not found } |

|  |
| --- |
| **hash(K)**  "Calculates the index of the key in the hash table"  { pre: key is not null }  { post: non-negative integer less than the size of the hash table } |

|  |
| --- |
| **ADT Priority Queue** |
| PQ= {heap=<heap>}  heap = {<Node1, Node2, Node3, Noden>, size=<n>}  Node = <T> |
| {inv: PQ.Element.length == PQ.Size} |
| Constructor operations:   * PQ: -> PQ   Analysis operations:   * peek PQ -> V * isEmpty PQ -> Boolean * size PQ -> Integer   Modification operations:   * insert PQ x T -> String * remove PQ -> V |

|  |
| --- |
| **PriorityQueue()**  "Creates a new PriorityQueue"  { pre: TRUE }  { post: { post: PQ = {heap = <heap>} } |

|  |
| --- |
| **remove(PQ)**  “Removes and returns the element with the highest priority from the priority queue”  { pre: PQ is not empty }  { post: PQ.heap = <Node2, Node3, …, Noden> where ∀ i ∈ [2,n], Node1 ≥ Nodei } |

|  |
| --- |
| **isEmpty(PQ)**  “Checks if the priority queue is empty”  { pre: TRUE }  { post: Returns true if n = 0, false otherwise } |

|  |
| --- |
| **Insert (T)**  “Adds a new element to the priority”  { pre: TRUE }  { post: PQ.heap = <T, Node1, Node2, …, Noden> where ∀ i ∈ [1,n], T ≥ Nodei } |

|  |
| --- |
| **increasePriority(PQ, V, P)**  “Increases the priority of an element in the priority queue”  { pre: TRUE }  { post: Returns Node1 where ∀ i ∈ [1,n], Node1 ≥ Nodei } |