**TADs (tipo abstracto de dato) de las estructuras de datos**

1. **Tabla Hash**

|  |
| --- |
| ADT HashTable  {Size = <size>, Elements = <elements>, HashFunction = <hashFunction>}  { inv: TablaHash.Tamaño > 0, TablaHash.Elementos ≠ null } |
| Primitive Operations: |
| ▪ CreateHashTable: Size → HashTable  ▪ InsertIntoTable: HashTable x Key x Value → HashTable  ▪ GetValue: HashTable x Key → Value  ▪ RemoveFromTable: HashTable x Key → HashTable  ▪ ContainsKey: HashTable x Key → Boolean  ▪ IsEmpty: HashTable → Boolean |

|  |
| --- |
| CreateHashTable(size) "Creates a new hash table with the specified size" |
| { pre: {tamaño > 0} } |
| { post: HashTable.Size = size, HashTable.Elements is empty } |

|  |
| --- |
| InsertElement(table, key, value) "Inserts a key-value pair into the hash table" |
| { pre: {HashTable = table, key ≠ null, value ≠ null} } |
| { post: HashTable.Elements containing the new element added (key, value). Use of heapFunction(key, value) to return the hashValue of the element (key value pair), corresponding to the index of the element in the array } |

|  |
| --- |
| GetValue(table, key) "Obtains the value associated with a key in the hash table" |
| { pre: {HashTable = table, key ≠ null, value ≠ null} } |
| { post: If (key, value) is in HashTable.Elements, then value is returned; otherwise, a null value is returned } |

|  |
| --- |
| RemoveFromTable(table, key) "Removes and returns a key-value pair from the hash table" |
| { pre: {HashTable = table, key ≠ null, value ≠ null} } |
| { post: If the hashKey corresponding to the (key, value) is in HashTable.Elements, then it is returned and removed, and table.size = table.size - 1; otherwise, there is no change in the table } |

|  |
| --- |
| ContainsKey(table, key) "Checks if the hash table contains a key" |
| { pre: {HashTable = table, key ≠ null, value ≠ null} } |
| { post: Returns true if the table contains the key; otherwise, returns false } |

|  |
| --- |
| isEmpty(table) "Checks if the hash table is empty" |
| { pre: {HashTable = table} } |
| { post: Returns true if the hash table is empty; otherwise, returns false {if hash.getSize() == 0 : true, else : false}; } |

1. **Priority Queue**

|  |
| --- |
| ADT Priority Queue  {Elements = <elements>, Size = <size>, PriorityFunction = <priority\_function>}  { inv: PriorityQueue.Size >= 0, PriorityQueue.Elements ≠ null } |
| Primitive Operations: |
| ▪ CreatePriorityQueue: → PriorityQueue  ▪ InsertIntoQueue: PriorityQueue x Element → PriorityQueue  ▪ GetNext: PriorityQueue → Element  ▪ RemoveNext: PriorityQueue → PriorityQueue  ▪ IsEmpty: PriorityQueue → Boolean |

|  |
| --- |
| CreatePriorityQueue() "Creates a new empty priority queue" |
| { pre: NIL } |
| { post: PriorityQueue.Size = 0, PriorityQueue.Elements is empty } |

|  |
| --- |
| InsertIntoQueue(queue, element) "Inserts an element into the priority queue" |
| { pre: {PriorityQueue = queue, element ≠ null} } |
| { post: PriorityQueue.Size = queue.Size + 1, element is in PriorityQueue.Elements, following the priority criteria } |

|  |
| --- |
| GetMaxElement(queue) "Obtains the element of highest priority in the queue" |
| { pre: {PriorityQueue = queue} } |
| { post: element = queue.maxMaxElement. The element of highest priority in the queue is returned without removing it } |

|  |
| --- |
| DequeueMaxElement(queue) "Removes/dequeues the element of highest priority from the queue and returns it" |
| { pre: {PriorityQueue = queue} } |
| { post: PriorityQueue.Size = queue.Size - 1, the element of highest priority is removed and returned from PriorityQueue.Elements } |

|  |
| --- |
| IsEmpty(queue) "Checks if the priority queue is empty" |
| { pre: {PriorityQueue = queue} } |
| { post: Returns true if the priority queue is empty; otherwise, returns false {if queue.getSize() == 0 : true, else : false}} |

1. **Max Heap**

|  |
| --- |
| ADT Max Heap  MaxHeap = {Elements = <elements>, Size = <size>, ComparisonFunction = <comparison\_function> (Max Heap in this case)}  { inv: MaxHeap.Size >= 0, MaxHeap.Elements ≠ null } |
| Primitive Operations: |
| ▪ CreateMaxHeap: → MaxHeap  ▪ InsertIntoMaxHeap: MaxHeap x Element → MaxHeap  ▪ GetMaximum: MaxHeap → Element  ▪ RemoveMaximum: MaxHeap → MaxHeap  ▪ IsEmpty: MaxHeap → Boolean |

|  |
| --- |
| CreateMaxHeap() "Creates a new empty maximum criterion heap" |
| { pre: createHeap == true } |
| { post: MaxHeap.Size = 0, MaxHeap.Elements is empty } |

|  |
| --- |
| InsertIntoMaxHeap(maxHeap, element) "Inserts an element into the maximum heap" |
| { pre: {MaxHeap = maxHeap, element ≠ null} } |
| { post: MaxHeap.Size = maxHeap.Size + 1, element is in MaxHeap.Elements and maintains the maximum heap property } |

|  |
| --- |
| GetMaximum(maxHeap) "Obtains the maximum element of the maximum heap" |
| { pre: {MaxHeap = maxHeap, MaxHeap.Size > 0} } |
| { post: The maximum priority element in the maximum heap is returned without removing it } |

|  |
| --- |
| RemoveMaximum(maxHeap) "Removes and returns the maximum element from the maximum heap" |
| { pre: {MaxHeap = maxHeap, MaxHeap.Size > 0} } |
| { post: MaxHeap.Size = maxHeap.Size - 1, the maximum element is removed and returned, and the heap is reorganized to maintain the maximum heap property (use of heapify function for max values)} |

|  |
| --- |
| IsEmpty(maxHeap) "Checks if the maximum heap is empty" |
| { pre: {MaxHeap = maxHeap} } |
| { post: Returns true if the maximum heap is empty; otherwise, returns false } |

1. **Stack**

|  |
| --- |
| TAD Stack  Stack = {Elements = <elements>, Size = <size>}  { inv: Stack.Size >= 0, Stack.Elements ≠ null } |
| Primitive Operations: |
| ▪ CreateStack: → Stack  ▪ PushToStack: Stack x Element → Stack  ▪ PopFromStack: Stack → Element  ▪ PeekStack: Stack → Element  ▪ IsEmpty: Stack → Boolean |

|  |
| --- |
| CreateStack()"Creates a new empty stack" |
| { pre: createStack == true} |
| { post: Stack.Size = 0, Stack.Elements is empty } |

|  |
| --- |
| PushToStack(stack, element) "Pushes an element onto the stack" |
| { pre: {Stack = stack, element ≠ null, element.Exists = false} } |
| { post: Stack.Size = stack.Size + 1, element is on top of the Stack.Elements } |

|  |
| --- |
| PopFromStack(stack) "Pops and returns the top element from the stack" |
| { pre: {Stack = stack, Stack.Size > 0} } |
| { post: Stack.Size = stack.Size - 1, the top element is returned and removed from Stack.Elements } |

|  |
| --- |
| PeekStack(stack) "Returns the top element of the stack without removing it" |
| { pre: {Stack = stack, Stack.Size > 0} } |
| { post: The top element is returned from Stack.Elements } |

|  |
| --- |
| IsEmpty(stack) "Checks if the stack is empty" |
| { pre: {Stack = stack} } |
| { post: Returns true if the stack is empty; otherwise, returns false } |

1. **Queue**

|  |
| --- |
| ADT Queue  Queue = {Elements = <elements>, Size = <size>}  { inv: Queue.Size >= 0, Queue.Elements ≠ null } |
| Primitive Operations: |
| ▪ CreateQueue: → Queue  ▪ Enqueue: Queue x Element → Queue  ▪ Dequeue: Queue → Element  ▪ Front: Queue → Element  ▪ IsEmpty: Queue → Boolean |

|  |
| --- |
| CreateQueue()"Creates a new empty queue" |
| { pre: createQueue = true} |
| { post: Queue.Size = 0, Queue.Elements is empty } |

|  |
| --- |
| Enqueue(queue, element) "Adds an element to the end of the queue" |
| { pre: {Queue = queue, element ≠ null} } |
| { post: Queue.Size = queue.Size + 1, element is at the end of Queue.Elements } |

|  |
| --- |
| Dequeue(queue) "Removes and returns the element at the front of the queue" |
| { pre: {Queue = queue, Queue.Size > 0} } |
| { post: Queue.Size = queue.Size - 1, the element at the fron is returned and removed from Queue.Elements } |

|  |
| --- |
| Front(queue) "Returns the element at the front of the queue" |
| { pre: {Queue = queue, Queue.Size > 0} } |
| { post: The element at the front is returned from Queue.Elements } |

|  |
| --- |
| IsEmpty(queue) "Checks if the queue is empty" |
| { pre: {Queue = queue} } |
| { post: Returns true if the queue is empty; otherwise, returns false } |