|  |
| --- |
| TAD <Max priority queue > |
| Max priority queue={size,comparator} |
| Inv: {comparator(a,b)= True} |
| Primitive Operations:  Createpriorityqueue(size): ->priorityqueue  enqueue(data) : priorityqueuexnodo -> void  Dequeue(): priorityqueue->Node  peek (): -> node  size(): -> data  Clear (): -> void |

|  |
| --- |
| TAD <Heap> |
| Heap={parent,left,right,size} |
| Inv: {parent>left ^ parent>right } |
| Primitive Operations:  CreateHeap(size): ->Heap  InsertHeap(dato) : Heapxnode -> void  DeleteHeap (): Heap->Node  Find (): -> node  sizeHeap(): -> data  isEmpty(): -> boolean |

|  |
| --- |
| TAD <Hash table > |
| Hash table={size,hashfunction, keyequalityfunction,table} |
| Inv: { key ≠ table(keys)} |
| Primitive Operations:  createHashTable(size, hashFunction, keyEqualityFunction): ->Hashtable  put( key, value): Hashtablexkey^value -> void  get(key): Hashtable->value  remove(key): Hashtablexkey->boolean  containsKey(key) :Hashtablexkey->boolean  sizeHeap(): -> data  isEmpty(): -> boolean  Clear (): -> void |

|  |
| --- |
| TAD <Queue > |
| Queue={add,poll,size} |
| Inv: {comparator(a,b)= True ^ (Q={x1,x2,x3,x4,x5…xn} ^ Q.poll()=x1)} |
| Primitive Operations:  CreateQueue(): ->Queue  add(Node) : add.Node -> void  poll(): Queue->Node  size(): -> data  isEmpty (): -> boolean |

**Max Priority Queque**

|  |
| --- |
| Createpriorityqueue(Size)  “Creates a new priority queque”  {pre: Size}  {pos: print priorityqueque data} |

|  |
| --- |
| enqueue(data)  “Adds data to queque”  {pre: data}  {pos: print priorityqueque data with enquque data } |

|  |
| --- |
| Dequeue()  “Removes data from queque ”  {pre: !=null}  {pos: print priorityqueque data without Dequeque data } |

|  |
| --- |
| peek ()  “print queque ”  {pre: !=null}  {pos: prints queque} |

|  |
| --- |
| size()  “Determines queque size”  {pre: !=null}  {pos: queque size} |

**Heap**

|  |
| --- |
| CreateHeap(size)  “Creates heap”  {pre: size }  {pos: prints created heap} |

|  |
| --- |
| InsertHeap(data)  “Inserts data in heap”  {pre: data}  {pos: prints Heap} |

|  |
| --- |
| DeleteHeap(data)  “Deletes data in heap”  {pre: data y !=null}  {pos: prints Heap } |

|  |
| --- |
| Find(daat)  “finds node in heap”  {pre: data y !=null}  {pos: prints node in heap} |

|  |
| --- |
| sizeHeap()  “calculates size of heap”  {pre: !=null}  {pos: prints heap size in Interger} |

**HashTable**

|  |
| --- |
| createqueue(size, hashFunction, keyEqualityFunction)  “Creates a new hash table”  {pre: size, hashFunction, keyEqualityFunction }  {pos: String “hashTable created”} |

|  |
| --- |
| put(key, value)  “places value in a key”  {pre: key, value y hashtable!=null}  {pos: String “placed”} |

|  |
| --- |
| Get(key)  “gets value from key”  {pre: key }  {pos: value |

|  |
| --- |
| Remove(key)  “removes value from key”  {pre: key y !=null }  {pos: prints “removed: ” + value} |

|  |
| --- |
| ContainsKey(key)  “finds if it has a key”  {pre: key y ¡=null}  {pos: boolean} |

**Queue:**

|  |
| --- |
| CreateHashTable(add,poll,size)  “Creates a new queue”  {pre add,poll,size }  {pos: String “queue created”} |

|  |
| --- |
| add(Node)  “adds a node to the queue”  {pre: Node!=null y queue !=null}  {pos: String “added”} |

|  |
| --- |
| poll ()  “takes the first element out of the queue”  {pre: queue ¡= null}  {pos: Node} |

|  |
| --- |
| size ()  “find and return the queue size”  {pre: queue !=null }  {pos: prints size} |

|  |
| --- |
| isEmpty()  “determines if the queue is empty”  {pre: queue! = null}  {pos: boolean} |