|  |
| --- |
| **TAD** **Stack** |
|  |
| { *inv: Los procedimientos de acceso permiten inserciones y eliminación de elementos solo en un extremo de la secuencia a excepción del ítem 1(top – last in first out)* } |
| Operaciones Primitivas:   * Push: *Stack x item* -> Stack * Pop: *Stack* -> Item * isEmpty: *Stack* -> Booleano * length: *Stack* -> Entero * top: *Stack* -> Item |

|  |
| --- |
| **Push(**Stack, item**)**  “Agrega un nuevo elemento al top de la pila”  { *pre:* TRUE }  { *post: Stack.top = item* } |

|  |
| --- |
| **Pop(**Stack**)**  “Devuelve el item en el top y al mismo tiempo lo elimina. Por lo tanto, modifica el Stack”  { *pre:* TRUE }  { *post: Si Stack.isEmpty != false, no es posible eliminar el objeto,*  *Si Stack.lenght > 1 o Stack.isEmpty = false el item en el top es eliminado de la pila* } |

|  |
| --- |
| **IsEmpty (**Stack**)**  “Informa si la pila está vacía.”  { *pre:* TRUE }  { *post: Si la Stack.lenght = 1 entonces Stack.isEmpty = true,*  *Si la Stack.lenght > 1 entonces Stack.isEmpty = false* } |

|  |
| --- |
| **Length(**Stack**)**  “Devuelve el tamaño de la pila o stack”  { *pre:* TRUE }  { *post: Determina el número de elementos que contiene el Stack* } |
| **Top(**Stack**)**  “Devuelve el item en el top del Stack”  { *pre:* TRUE }  { *post: Si Stack.isEmpty != true, el item en el top es retornado sin generar cambios en la pila,*  *Si Stack.isEmpty = true, no es posible devolver ningún item* } |

|  |
| --- |
| **TAD** **iList** |
|  |
| { *inv: si iList.size es cercano a iList.length entonces la iList crecera de tamaño*} |
| Operaciones Primitivas:   * AddFirst: *iList* -> iList * IsEmpty: *iList* -> Booleano * Length: *iList* -> Entero |

|  |
| --- |
| **addEnd(**iList**)**  “Agrega el nuevo elemento al final de la lista”  { *pre: el nuevo elemento que se va a agregar es nuevo*}  { *post: se agrega un nuevo jugador en la última posición*} |

|  |
| --- |
| **isEmpty(**iList**)**  “Informa si la lista es vacía”  { *pre:* TRUE }  { *post: Si la iList.lenght = 0 entonces Stack.isEmpty = true,*  *Si la iList.lenght > 0 entonces Stack.isEmpty = false* } |

|  |
| --- |
| **length(**iList**)**  “Devuelve el tamaño de la lista”  { *pre:* TRUE }  { *post: Determina el número de elementos que contiene la lista*} |

|  |
| --- |
| **TAD** **iHash** |
| C:\Users\usuario\Desktop\1.PNG |
| { *inv: La tabla de matriz se define de modo que cada tabla de ubicación [i] sea en sí misma una matriz, llamado cubo y cada ubicación [i] tiene una referencia a una linked list*} |
| Operaciones Primitivas:   * IsEmpty: *iHash* -> Booleano * length: *iHash* -> Entero * TableDelete: *iHash x Entero* -> iHash * TablaRetrive *iHash x Entero* -> iHash * HashFunction *iHash x Entero* -> Entero |

|  |
| --- |
| **isEmpty(**iHash**)**  “Informa si la iHash esta vacía”  { *pre:*  TRUE}  { *post: si iHash.length = 0 entonces iHash.isEmpty = true,*  *si iHash.lengh > 0 entonces iHash.isEmpty = false*} |

|  |
| --- |
| **lenght(**iHash**)**  “Devuelve el tamaño de la iHash”  { *pre:* TRUE }  { *post: Determina el número de elementos que contiene la iHash*} |

|  |
| --- |
| **tableDelete(**iHash, entero**)**  “Elimina un elemento de la tabla de acuerdo a la posición”  { *pre:*  *entero pertenece al iHash.length,*  *el elemento en la posición "entero" existe*}  { *post: Elimina un elemento de la iHash*} |

|  |
| --- |
| **tableRetrieve(**iHash, entero**)**  “Devuelve un elemento de la iHash en una posición determinada”  { *pre:*   *la posición o llave "entero" pertenece al iHash.length,*  *el elemento de la posición o llave "entero" existe*}  { *post: Devuelve el elemento en la posición o llave "entero"*} |

|  |
| --- |
| **hashFunction(**iHash, entero**)**  “Cálcula posición o llave que tendra el elemento en la iHash”  { *pre:*  TRUE}  { *post: devuelve una posición o llave (entero) que pertenece a la iHash.length* } |