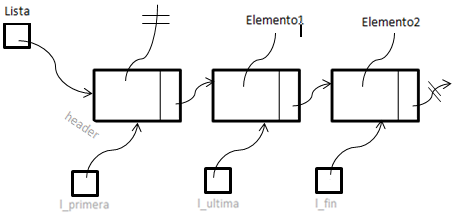
|  |
| --- |
| oRGANIZACIÓN DE COMPUTADORAS |
| TA-TE-TI |
| Proyecto N◦1 Programación en lenguaje C |
| **Segundo Cuatrimestre de 2019** |
|  |
|  |

|  |
| --- |
| Olaciregui Santiago – 121485 | Mayer Agustín – 121256 |

# TDA Lista

La lista está implementada mediante una estructura simplemente enlazada con celda centinela, utilizando el concepto de posición indirecta. En esta representación cada celda mantiene referencia a la celda siguiente y una referencia a un elemento genérico.  


***void crear\_lista(tLista \* l);*** Inicializa una lista vacía. Una referencia a la lista creada es referenciada en \*l.  
 *- Se reserva espacio en memoria para una nueva celda(header). Se le establece a esta, referencia a un elemento nulo, y referencia nula a la celda siguiente.*

***void l\_insertar(tLista l, tPosicion p, tElemento e);*** Inserta el elemento E, en la posición P, en L. Con L= A,B,C,D y la posición P direccionando C, luego: L' = A,B,E,C,D  
 - Se reserva espacio en memoria para una nueva celda. A la nueva celda se le establece el elemento e, y se actualizan las posiciones siguientes de la nueva celda, y de la posición p.

***void l\_eliminar(tLista l, tPosicion p, void (\*fEliminar)(tElemento));***  
 Elimina la celda P de L. El elemento almacenado en la posición P es eliminado mediante la función fEliminar parametrizada. Si P es fin(L), finaliza indicando LST\_POSICION\_INVALIDA.  
 - Si la posición p no es el fin de la lista, llamo a la función fEliminar parametrizada con el siguiente a p (por ser lista con posición indirecta). Actualizo el elemento a p, y la referencia a el siguiente en nulo. Por ultimo se libera el espacio en memoria.

***void l\_destruir(tLista \* l, void (\*fEliminar)(tElemento));*** Destruye la lista L, eliminando cada una de sus celdas. Los elementos almacenados en las celdas son eliminados mediante la función fEliminar parametrizada.

***tElemento l\_recuperar(tLista l, tPosicion p);***   
 Recupera y retorna el elemento en la posición P. Si P es fin(L), finaliza indicando LST\_POSICION\_INVALIDA.

***tPosicion l\_primera(tLista l);***  
 Recupera y retorna la primera posición de L. Si L es vacía, primera(L) = ultima(L) = fin(L).

***tPosicion l\_siguiente(tLista l, tPosicion p);***  
 Recupera y retorna la posición siguiente a P en L. Si P es fin(L), finaliza indicando LST\_NO\_EXISTE\_SIGUIENTE.

***tPosicion l\_anterior(tLista l, tPosicion p);***  
 Recupera y retorna la posición anterior a P en L. Si P es primera(L), finaliza indicando LST\_NO\_EXISTE\_ANTERIOR.

***tPosicion l\_ultima(tLista l);***  
 Recupera y retorna la última posición de L. Si L es vacía, primera(L) = ultima(L) = fin(L).

***tPosicion l\_fin(tLista l);***   
 Recupera y retorna la posición fin de L. Si L es vacía, primera(L) = ultima(L) = fin(L).

***int l\_longitud(tLista l);***  
 Retorna la longitud actual de la lista.

# TDA Árbol

El árbol está implementado mediante una estructura de nodos enlazados. En esta representación cada nodo mantiene referencia a otro nodo considerado padre del mismo dentro del árbol, una lista de nodos que representa los nodos hijos del mismo, y una referencia a un elemento genérico que representa el rótulo de dicho nodo.

***void crear\_arbol(tArbol \* a);***   
 Inicializa un árbol vaco. Una referencia al árbol creado es referenciado en \*A.

***void crear\_raiz(tArbol a, tElemento e);***   
 Crea la raíz de A. Si A no es vaco, finaliza indicando ARB\_OPERACION\_INVALIDA.

***tNodo a\_insertar(tArbol a, tNodo np, tNodo nh, tElemento e);*** Inserta y retorna un nuevo nodo en A. El nuevo nodo se agrega en A como hijo de NP, hermano izquierdo de NH, y cuyo rótulo es E. Si NH es NULL, el nuevo nodo se agrega como último hijo de NP. Si NH no corresponde a un nodo hijo de NP, finaliza indicando ARB\_POSICION\_INVALIDA. NP direcciona al nodo padre, mientras NH al nodo hermano derecho del nuevo nodo a insertar.

***void a\_eliminar(tArbol a, tNodo n, void (\*fEliminar)(tElemento));*** Elimina el nodo N de A. El elemento almacenado en el árbol es eliminado mediante la función fEliminar parametrizada. Si N es la raíz de A, y tiene un sólo hijo, este pasa a ser la nueva raíz del árbol. Si N es la raíz de A, y a su vez tiene ms de un hijo, finaliza retornando ARB\_OPERACION\_INVALIDA. Si N no es la raíz de A y tiene hijos, estos pasan a ser hijos del padre de N, en el mismo orden y a partir de la posición que ocupa N en la lista de hijos de su padre.

***void a\_destruir(tArbol \* a, void (\*fEliminar)(tElemento));*** Destruye el árbol A, eliminando cada uno de sus nodos. Los elementos almacenados en el árbol son eliminados mediante la función fEliminar parametrizada.

***tElemento a\_recuperar(tArbol a, tNodo n);***   
 Recupera y retorna el elemento del nodo N.

***tNodo a\_raiz(tArbol a);***   
 Recupera y retorna el nodo correspondiente a la raíz de A.

***tLista a\_hijos(tArbol a, tNodo n);***   
 Obtiene y retorna una lista con los nodos hijos de N en A.

***void a\_sub\_arbol(tArbol a, tNodo n, tArbol \* sa);***   
 Inicializa un nuevo árbol en \*SA. El nuevo árbol en \*SA se compone de los nodos del sub árbol de A a partir de N. El sub árbol de A a partir de N debe ser eliminado de A.

# Juego Ta-Te-Ti

Se implementó una aplicación de consola que permite visualizar la dinámica de una partida del juego Ta-Te-Ti. El programa inicia solicitando los nombres de los dos jugadores, el modo de juego que desea el usuario, y la indicación de qué jugador comienza la partida. Luego, inicia una nueva partida de forma tal que, en cada turno, se imprime por consola el estado actual del tablero, las indicaciones de quién debe jugar, se consulta el movimiento a realizar por los usuarios y se modifica la partida a partir de la recopilación de estos datos.   
Se permiten dos modos de juego: Usuario vs. Usuario y Usuario vs. Agente IA. A la hora de indicar qué jugador comienza la partida se permiten tres opciones: jugador 1, jugador 2 o jugador al azar. Bajo esta última alternativa, el programa (de forma aleatoria) selecciona qué jugador comienza la partida.