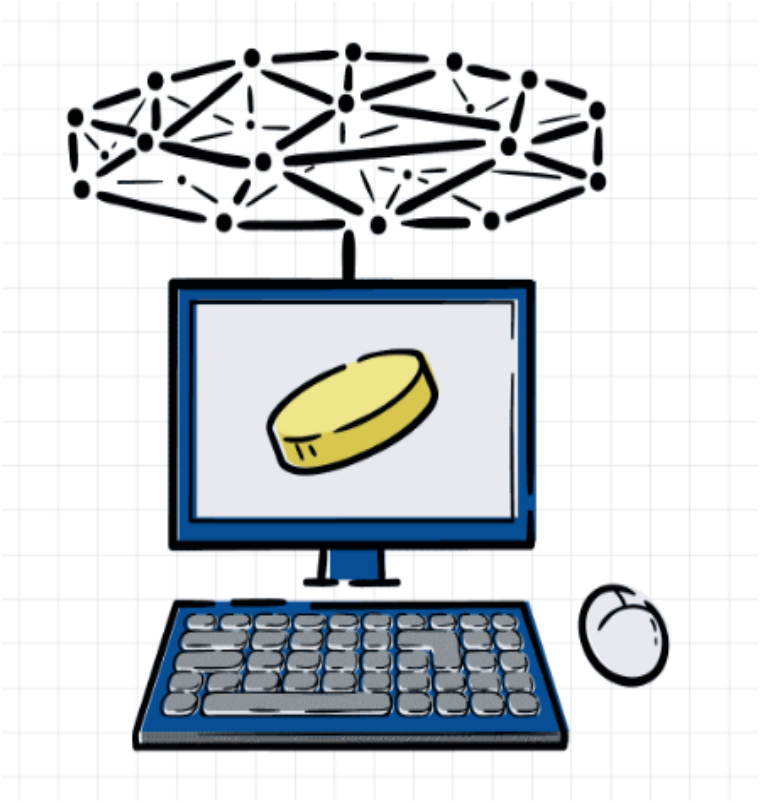
Universidad del Chubut

Tecnicatura en Desarrollo de Software

Estructura de Datos y Algoritmos

Trabajo Práctico grupal.

TDA Lista



Profesor:

Aldo Rafael ANA

Integrantes - Grupo 4:

Agustin Teresa Garcia

Leonardo Victor Paz

Rocio Guadalupe Manquillan

Ludmila Araceli Pereyra

Diego Maximiliano Orias

Universidad del Chubut

Enunciado:

Desarrollemos pseudocódigos de funciones de un TDA Lista dinámico y doblemente enlazado que nos permitan:

1. Crear una lista doblemente enlazada.
2. Determinar si una lista se encuentra vacía.
3. Agregar un nodo en la cabeza de la lista.
4. Agregar un nodo en una posición determinada.
5. Agregar un nodo en la cola de la lista.
6. Obtener el nodo de la cabeza de la lista.
7. Obtener el nodo de una posición determinada.
8. Obtener el nodo de la cola de la lista.
9. Eliminar el nodo de la cabeza de la lista.
10. Eliminar el nodo de una posición determinada.
11. Eliminar el nodo de la cola de la lista.
12. Destruir una lista doblemente enlazada.

Finalizado el desarrollo del inciso anterior, implementemos esas funciones en el lenguaje de programación C en un archivo de cabecera (.h) y un archivo con el código fuente (.c).

Luego, desarrollamos un programa que permita probar el TDA Lista implementado.

Desarrollo:

Punto 1:

Crear una lista doblemente enlazada:

```
//Funcion lista = CrearLista()       ---    Crear una lista doblemente enlazada.

   lista.cabeza = nulo
   lista.cantidad_nodos = 0

//FinFuncion
```

En este punto se define una función llamada **CrearLista** que inicializa una lista doblemente enlazada con su cabeza apuntando **NULO**, indicando que está vacía, y con una variable que cuenta la cantidad de nodos, comenzando en 0.

Punto 2:

verificar si una lista está vacia

```
// Precondicion: la lista debe ser valida
// Entrada: una lista
// Salida: un booleano indicando el exito del proceso
// Postcondicion: se determina si la lista esta vacia o no

//Funcion vacia = listaVacia(lista: Lista)                              ---    Determinar si una lista se encuentra vacía.

   Si lista.cantidad_nodos = 0 y lista.cabeza == nulo entonces
      vacia = Verdadero
    Sino
      vacia = Falso
   FinSi

//FinFuncion
```

La función `listaVacia` determina si una lista está vacía verificando si el contador de nodos es igual a 0 y si la cabeza de la lista es igual a **NULO**, si ambas condiciones se cumplen, la función retorna **Verdadero**, de lo contrario **Falso**

Punto 3:

Agregar un nodo en la cabeza de la lista

```
// Precondicion: la lista recibida debe ser valida
// Entrada: lista, nodo
// Salida: un booleano indicando el exito del proceso
// Postcondicion: se agrega un nodo en la cabeza de la lista.

 Subproceso ok=agregar_cabeza(lista: Lista, nodo)         ----    Agregar un nodo en la cabeza de la lista.

       Si listaVacia(lista) Entonces
           lista.cabeza <- nodo
       Sino

           aux = lista.cabeza
           nodo.siguiente = aux
           aux.anterior = nodo
           lista.cabeza = nodo

       FinSi

       lista.cantidad_nodos = lista.cantidad_nodos + 1
       ok=Verdadero

  FinSubproceso
```

La función **agregar_cabeza** agrega un nodo al inicio de una lista. Si la lista está vacía, el nodo se convierte en la cabeza. Si no está vacía, el nodo se enlaza al nodo actual de la cabeza (asignando los punteros **siguiente** y **anterior**), y luego el nuevo nodo se convierte en la nueva cabeza. Finalmente se incrementa la cantidad de nodos

Punto 4:

Agregar un nodo en una posición determinada

```
// Precondicion: la lista debe ser valida
// Entrada: lista, nodo, posicion
// Salida: un booleano indicando el exito del proceso
// Postcondicion: agregar un nodo en determinada posicion de la lista

 Subproceso ok=agregar_posicion(lista: Lista, nodo, posicion)    ----  Agregar un nodo en una posicion de la lista

    Si listaVacia(lista) Entonces

            agregar_cabeza(lista, nodo)
            retorno = verdadero
    FinSi

    Si posicion > lista.cantidad_nodos Entonces

        agregar_cola(lista, nodo)
        retorno = verdadero

    FinSi
        nodo_aux = lista.cabeza
        Para i = 1 hasta posicion hacer

          nodo_aux = nodo_aux.siguiente

        FinPara

        nodo.siguiente = nodo_aux.siguiente
        nodo.anterior = nodo_aux

    Si nodo_aux.siguiente ≠ Nulo Entonces
      nodo_aux.siguiente.anterior = nodo
    FinSi

    nodo_aux.siguiente = nodo
    lista.cantidad_nodos = lista.cantidad_nodos + 1

    retorno = Verdadero

 FinSubproceso
```

En la función **agregar_posicion** agrega un nodo en una posición específica de la lista. Primero verifica si la lista está vacía, en cuyo caso agrega el nodo en la cabeza. Si la posición es mayor que el número de nodos, agrega el nodo al final. De lo contrario, recorre la lista hasta la posición indicada, ajusta los punteros de los nodos **anterior** y **siguiente** para insertar el nuevo nodo en esa posición y finalmente incrementa el contador.

Punto 5:

Agregar un nodo en la cola de la lista

```
// Precondicion: la lista recibida debe ser valida
// Entrada: una lista, nodo
// Salida: un booleano indicando el exito del proceso
// Postcondicion: se agrega un nodo en la cola de la lista

 Subproceso ok=agregar_cola(lista: Lista, nodo)       ------    Agregar un nodo en la cola de la lista

       Si listaVacia(lista) Entonces
           agregar_cabeza(lista, nodo)
       Sino
           aux = lista.cabeza

           Mientras aux.siguiente = Nulo hacer
             aux = aux.siguiente
           FinMientras

           nodo.anterior = aux
           aux.siguiente = nodo
           lista.cantidad_nodos = lista.cantidad_nodos + 1

       FinSi

       ok=Verdadero
  FinSubproceso
```

La función **agregar_cola** se encarga de agregar un elemento nuevo al final de la lista, si la lista está vacía agrega el nodo al inicio de la lista, convirtiéndose en el único elemento. Si no está vacía se busca el último nodo de la lista, se enlaza el nuevo nodo al último nodo encontrado. y por último se actualiza la cantidad de nodos en la lista.

Punto 6:

Obtener el nodo de la cabeza de la lista

```
// Precondición: la lista debe ser válida.
// Entrada: lista (Lista).
// Salida: el nodo en la cabeza de la lista o Nulo si la lista está vacía.
// Postcondición: se obtiene el nodo en la cabeza de la lista, si existe.

Subproceso imprimir_cabecera(lista: Lista)   // obtener nodo de la cabeza de la lista

  Si listaVacia(lista) Entonces
    return nulo;
  sino
    return lista.cabecera;
  finSino

FinSubproceso
```

En la función **imprimir_cabecera** verifica si la lista está vacía y, si no lo está, devuelve el primer elemento (la cabeza) de la lista, de lo contrario, devuelve nulo.

Punto 7 :

Obtener el nodo de una posición determinada.

```
// Precondición: la lista recibida debe ser válida y no estar vacia
// Entrada: una lista, posicion
// Salida: nodo a obtener
// Postcondición: obtener el nodo de la posicion determianda

Subproceso obtener_posicion_determinada(lista: Lista,posicion)  //obtener nodo de una posicion determinada

  Si listaVacia(lista) entonces
          retorno = nulo;
  FinSi

  Si posicion > lista.cantidad_nodos Entonces
      retorno = Falso
  FinSi

  Si posicion == lista.cantidad_nodos Entonces
    imprimir_cola(lista)
    retorno = Verdadero

  Sino

    nodo_aux = lista.cabeza

    Para i = 1 hasta posicion hacer

      nodo_aux = nodo_aux.siguiente

    FinPara

    retorno = nodo_aux

  FinSin

finProceso
```

Explicación:

obtener_posicion_determinada, esta función permite acceder a cualquier elemento de una lista enlazada mediante su índice, verificando previamente que la lista no esté vacía y que el índice proporcionado sea válido.

Punto 8:

Obtener el nodo de la cola de la lista.

```
// Precondición: la lista debe ser válida.
// Entrada: lista (Lista).
// Salida: el nodo de la cola de la lista o Nulo si la lista está vacía.
// Postcondición: se obtiene el nodo de la cola, si existe.

Subproceso imprimir_cola(lista: Lista)  //obtener nodo final de la lista

  Si listaVacia(lista) Entonces
    return nulo;
  sino

  Mientras aux.siguiente = Nulo hacer
      aux = aux.siguiente
  FinMientras

  returno = aux

  finSino

FinSubproceso
```

Explicación:

imprimir_cola, esta función permite acceder al último elemento de una lista enlazada. Comienza verificando si la lista tiene elementos. Si es así, recorre la lista nodo por nodo hasta llegar al final y devuelve el último nodo encontrado. En caso contrario, indica que la lista está vacía devolviendo nulo.

Punto 9:

Eliminar el nodo de la cabeza de la lista.

```
48 // Precondición: la lista recibida debe ser válida y no estar vacía
49 // Entrada: una lista
50 // Salida: un booleano indicando el éxito del proceso
51 // Postcondición: el nodo en la cabeza de la lista se elimina
52
53 Subproceso retorno = eliminar_cabeza(lista: Lista)   // eliminar nodo de la cabeza de la lista
54
55   Si listaVacia(lista) Entonces
56
57     retorno = Falso;
58
59   Finsi
60
61   nodo_aux = lista.cabeza.siguiente
62
63   nodo_aux.anterior = nulo
64
65   lista.cabeza = nodo_aux;
66
67   lista.cantidad_nodos = lista.cantidad_nodos -1
68
69   retorno = Verdadero
70
71 FinSubproceso
72
```

Explicación:

La **funcion_eliminar_cabeza**, toma como entrada una lista, si la lista no tiene elementos devuelve falso, ya que no hay nada para eliminar. Si la lista no está vacía, se hace que el nuevo primer elemento de la lista sea el que actualmente sigue al primer elemento. Esto lo hacemos asignando el valor del puntero siguiente del nodo actual de la cabeza a la variable aux y luego asignando el valor de aux al atributo cabeza de la lista. Luego se decrementa en uno el contador de nodos y por último retorna verdadero.

Punto 10:

Eliminar el nodo de una posición determinada.

```
// Precondición: la lista debe ser válida y no estar vacía.
// Entrada: lista (Lista), posicion (Entero).
// Salida: un valor booleano indicando el éxito del proceso.
// Postcondición: el nodo en la posición determinada es eliminado.

Subproceso retorno = eliminar_posicion_determinada(lista: Lista, posicion: Entero)  //eliminar nodo de una posicion determinada

  Si listaVacia(lista) entonces
     retorno = Falso;
  FinSi

  Si posicion > lista.cantidad_nodos Entonces
      retorno = Falso
  FinSi

  Si posicion == 1 Entonces
    eliminar_cabeza(lista)
    retorno = Verdadero
  FinSi

  Si posicion == lista.cantidad_nodos Entonces
    eliminar_cola(lista)
    retorno = Verdadero
  FinSin

  nodo_aux = lista.cabeza
  Para i = 1 hasta posicion hacer

      nodo_aux = nodo_aux.siguiente

  FinPara

  nodo_aux.anterior.siguiente = nodo_aux.siguiente
  nodo_aux.siguiente.anterior = nodo_aux.anterior

  nodo_aux.siguiente = nulo
  nodo_aux.anterior = nulo

  lista.cantidad_nodos = lista.cantidad_nodos - 1

  retorno = Verdadero

FinProceso
```

Explicación:

eliminar_posicion_determinada, en esta funcion, como su nombre lo indica, se encarga de eliminar un nodo en una posicion específica dentro de una lista. Si la lista está vacía devuelve un booleano falso, si la posicion indicada es mayor que la cantidad de nodos, tampoco se puede realizar la eliminación y retorna falso. Si la posicion es 1, se llama a la funcion **eliminar_cabeza** para eliminar el primer elemento, lo mismo si la posicion es igual a la cantidad de nodos en la lista, se llama a la funcion **eliminar_cola** para eliminar el último elemento.

Si la posición a eliminar no es la primera ni la última, se realiza un recorrido de la lista hasta llegar al nodo anterior al nodo a eliminar. Una vez localizado el nodo anterior, se actualizan los punteros **siguiente** y **anterior** de los nodos involucrados para eliminar el nodo de la lista. Se decrementa el contador ya que se eliminó un elemento y por último retorna verdadero.

Punto 11:

Eliminar el nodo de la cola de la lista.

```
// Precondición: la lista recibida debe ser válida y no estar vacía
// Entrada: una lista
// Salida: un booleano indicando el éxito del proceso
// Postcondición: el nodo en la cola de la lista se elimina

Subproceso retorno = eliminar_cola(lista: Lista)  //eliminar nodo final /cola de la lista

  Si listaVacia(lista) Entonces
    retorno = Falso;
  finsi

  nodo_aux = lista.cabeza
  Mientras nodo_aux.siguiente != nulo Hacer
    nodo_aux = nodo_aux.siguiente
  FinMientras

  nodo_aux.anterior.siguiente = nulo
  lista.cantidad_nodos = lista.cantidad_nodos - 1

FinSubproceso
```

Explicación:

eliminar_cola, esta función se encarga de eliminar el último elemento de la lista, primero verifica si la lista está vacía y retorna un valor booleano falso. Se inicia un bucle que recorre la lista desde principio hasta encontrar el penúltimo nodo, iterando hasta que el nodo actual **nodo_aux** apunte a un nodo cuyo siguiente es nulo (el último nodo). Una vez encontrado, se establece su puntero siguiente a nulo. Esto desconecta el último nodo de la lista y lo elimina.

Punto 12:

Destruir una lista doblemente enlazada.

```
27 // Precondicion: la lista recibida debe ser valida
28 // Entrada: una lista
29 // Salida: un booleano indicando el exito del proceso
30 // Postcondicion: se eliminan todos los nodos de la lista
31
32 Subproceso retorno = eliminar_lista_completa(lista: Lista)                    //eliminar lista completa
33
34   Si listaVacia(lista) entonces
35         retorno Falso;
36   FinSi
37
38   Mientras ~listaVacia(lista) hacer
39     eliminar_cabecera(lista)
40   finMentras
41
42   retorno Verdadero
43
44 FinSubproceso
45
```

Explicación:

eliminar_lista_completa, en esta función verifica si la lista está vacía y retorna falso si no hay nada que eliminar, si la lista no está vacía, se entra en un bucle mientras que se ejecutará hasta que la lista quede vacía. Dentro del bucle, se llama a la función eliminar cabecera para eliminar el primer nodo de la lista en cada iteración, lo que hace que lo elimine nodo por nodo ya que siempre va apuntando al primer elemento hasta eliminar la lista. Por último retorna un booleano verdadero para indicar que se eliminaron todos los elementos correctamente.