Actividad 2.2
Juan Pablo Montoya

Reflexión:

- Create:
  - push(T data): Este método agrega un valor al inicio de la lista. La complejidad es O(1) ya que solo tiene que crear un nuevo nodo y ponerle el resto de la lista al apuntador next
  - add(int index, T data): esta función tiene una complejidad de Tiempo : O(N) y Espacio : O(1), ya que no crea memoria adicional, y le conlleva n cantidad de movimientos llegar al index. En este algoritmo, se recorre hasta el index y se le asigna un nuevo nodo a la variable next de ese nodo y se le asigna la variable next del anterior al nuevo nodo.
  - insert_mid: esta función se utiliza para insertar un valor en el medio de la lista enlazada. En este caso podemos utilizar una técnica llamada slow pointer fast pointer. Con esta técnica lo que se hizo fue crear dos apuntadores en la lista enlazada que se movieran a distintas velocidades. Uno iba doblemente rápido que la otra. De esta manera cuando el apuntador rápido llega al final, el apuntado pequeño se queda en el medio. Así obtenemos el nodo de enmedio y procedemos a agregar el nuevo nodo enfrente.
- Read:
  - get_node(int index): En este caso se va moviendo la cabeza de la lista index veces. Retorna la cabeza en ese momento
  - get(int index): Estor retorna get_node(index)->Data. El tiempo es O(N) ya que le toma N movimientos llegar al index, y O(1)

- Update:

  - update(int index, T value): Se obtiene el nodo a actualizar con get_node(index) y se le agrega el valor con curr->value = value. La complejidad es la misma que leer un dato de cierto index O(N) y O(1)

- Delete:
  - delete(int index) : Para eliminar este nodo, se usa la funcion curr = get_node(index-1), despues se asigna next = curr->next, y se apunta curr->next a next->next, después se elimina next. La complejidad de esto es O(N) y O(1), la misma que es leer un dato de cierto index

```
LinkedList<int> list;
    // list.add_mid(10);
    // Cuando no tengo nada :
    list.push(1);
    // Cuando tengo 1
    list.push(2);
    list.update(1, 3);
    // Cuando tengo muchos
    list.push(4);
    list.push(5);
```

Actividad 2.2
Juan Pablo Montoya

```
    list.push(6);
    list.push(7);
    list.push(8);
    list.del(0);
    LinkedList<int> list2;
    // Cuando no tengo nada :
    list2.insert_mid(100);
    // Cuando tengo 1 :
    list2.push(101);
    list2.insert_mid(100);
    // Cuando tengo muchos
    list2.push(101);
    list2.push(102);
    list2.push(103);
    list2.insert_mid(100);
    list2.update(1, 20);
    list2.del(1);
    list2.add(10, 10);
```

Se utilizaron estos casos de prueba, y se obtuvo el siguiente resultado:
Push : 1
Current List : 1 -> nullptr
Push : 2
Current List : 2 -> 1 -> nullptr
Update index : 1 To value : 3
Current List : 2 -> 3 -> nullptr
Push : 4
Current List : 4 -> 2 -> 3 -> nullptr
Push : 5
Current List : 5 -> 4 -> 2 -> 3 -> nullptr
Push : 6
Current List : 6 -> 5 -> 4 -> 2 -> 3 -> nullptr
Push : 7
Current List : 7 -> 6 -> 5 -> 4 -> 2 -> 3 -> nullptr
Push : 8
Current List : 8 -> 7 -> 6 -> 5 -> 4 -> 2 -> 3 -> nullptr
Delete index : 0
Current List : 7 -> 6 -> 5 -> 4 -> 2 -> 3 -> nullptr
Insert Mid : 100
Current List : 100 -> nullptr
Push : 101
Current List : 101 -> 100 -> nullptr
Insert Mid : 100
Current List : 101 -> 100 -> 100 -> nullptr
Push : 101
Current List : 101 -> 101 -> 100 -> 100 -> nullptr

Actividad 2.2
Juan Pablo Montoya

Push : 102
Current List : 102 -> 101 -> 101 -> 100 -> 100 -> nullptr
Push : 103
Current List : 103 -> 102 -> 101 -> 101 -> 100 -> 100 -> nullptr
Insert Mid : 100
Current List : 103 -> 102 -> 101 -> 100 -> 101 -> 100 -> 100 -> nullptr
Update index : 1 To value : 20
Current List : 103 -> 20 -> 101 -> 100 -> 101 -> 100 -> 100 -> nullptr
Delete index : 1
Current List : 103 -> 101 -> 100 -> 101 -> 100 -> 100 -> nullptr
Add : 10 at index : 10
Current List : 103 -> 101 -> 100 -> 101 -> 100 -> 100 -> 10 -> nullptr