

## DoublyLinkedList

Horfið vel á fyrirlestra um tvítengda lista, sér í lagi fyrirlestra í sal þar sem farið er vel í útfærslu með **sentinel nodes**. Skoðið einnig fyrirlestra og efni um **templates** og þá tilhögun að hafa allan kóða template-klasa í .h skrá og sleppa .cpp skrá fyrir það. Skoðið síðan forritstexta verkefnisins, þar eru lýsingar á þeim föllum þarf að útfæra, í commentum við hvert fall.

Verkefnið er að útfæra tvítengdan lista þar sem gögnin eru skilgreind með **template**. Það á að vera hægt að útbúa lista sem tekur við hvaða tegund af gögnum sem er, t.d. er bæði listi af string og listi af int prófaður í main.

Allar útfærslur sem bæta þarf við eru í *doublylinkedlist.h*. Allar útfærslur verða í .h skránni, vegna þess að við erum að vinna með **templates**. Þetta er útskýrt betur í fyrirlestri um template.

Það á að útfæra listann með svokölluðum **sentinel nodes**, þannig að *head* og *tail* eru ekki eiginlegir gagnahnútar, heldur eru þeir á undan og á eftir fyrsta og seinasta hnút. Þannig þarf aldrei að tékka hvort hnútur sé *NULL*, þar sem það eru alltaf a.m.k. tveir hnútar í listanum, jafnvel þegar hann er **tómur**. Þá bendir bara *head* beint á *tail* og *tail* beint á *head*.

Athugið að þegar þið smíðið nýjan hnúta gerið þið

**ListNode \*node = new ListNode(data, previousNodePointer, nextNodePointer);**

Ef þið gerið bara **ListNode \*node = new ListNode()** þá benda bæði next og prev á NULL.

Private breytur eru þegar skilgreindar og útskýrðar undir private í klasanum *DoublyLinkedList*.

ExpectedOutput.txt inniheldur rétt úttak miðað við núverandi main() fall. Þetta eru samt ekki nægjanlegar prófanir, heldur þurfið þið að útbúa nákvæmari prófanir fyrir öll föll og heildina.

### Verkefninu er skilað á Mooshak.

- Þið fáið **þrjár tilraunir** til að senda verkefnið inn í Mooshak. Þriðja tilraunin er lokaskil.
- Sendið ZIP skrá inn í Mooshak sem inniheldur allan forritstextann ykkar, .cpp og .h skrár.

### Þið verðið að gera virkilega góðar prófanir í main fallinu.

- Núverandi main() forrit inniheldur hjálparföll og dæmi um notkun þeirra.
- Þið verðið að sjá til þess að jaðartilvik séu prófuð.
  - Bætið í gagnagrindina og takið úr henni í mismunandi röð.
  - Bætið í og takið úr tómri gagnagrind.
- Verið alveg viss um að ykkar forritstexti virki og að þið skiljið hvernig og hvers vegna hann virkar áður en þið byrjið að reyna að skilja villuskilaboðin frá Mooshak.
- Verið viss um að forritið leki ekki minni.
  - Fyrir hvert new ætti að vera delete.
  - Eyðirinn ætti að eyða öllu minni sem hefur við úthlutað.

**insert()**

Þegar bætt er í listann bætist stakið fyrir framan (nær head) núverandi hnút (current node).

Næst þegar bætt er inn bætist aftur fyrir framan sama hnút.

Gildum sem bætt er inn í ákveðinni röð verða því í þeirri röð í listanum.

**append()**

Bætir aftast í listann.

Ef núverandi staðsetning er aftast í listanum heldur hún áfram að vera aftast í listanum. Í því tilfelli mun insert því halda áfram að bæta aftast í listann.

**remove()**

Fjarlægir núverandi stak.

Núverandi stak verður í framhaldinu næsta stak fyrir aftan, sem nú hefur sömu staðsetningu og hið fyrra núverandi stak.

Ef núverandi staðsetning er ekki á gildu staki þá er kastað *InvalidPositionException*.

**move\_to\_start()**

Færir núverandi stak á fremstu mögulegu staðsetningu.

Á þeim stað er bæði hægt að bæta í, sækja og fjarlægja stök.

**move\_to\_end()**

Færir núverandi stak á öftustu mögulegu staðsetningu.

Á þeim stað er bara hægt að bæta í listann, en ekki hægt að sækja eða fjarlægja.

**prev()**

Færir núverandi stak einu staki nær upphafi listans.

Ef núverandi stak er á upphafi listans gerist ekki neitt.

**next()**

Færir núverandi stak einu staki nær enda listans.

Ef núverandi stak er á öftustu mögulegu staðsetningu listans gerist ekki neitt.

**length()**

Skilar fjölda staka í listanum.

**curr\_pos()**

Skilar staðsetningu núverandi staks þar sem fremsta mögulega staðsetning hefur gildið 0.

**move\_to\_pos(int pos)**

Færir núverandi stak á staðsetninguna pos. Ef gildi pos er utan mengis mögulegra staðsetninga (sjá move\_to\_end og move\_to\_start) er kastað *InvalidPositionException*.

**get\_value()**

Skilar gildinu úr núverandi staki.

Ef núverandi staðsetning er ekki á gildu staki þá er kastað *InvalidPositionException*.

Búum til nýjan lista og köllum á `append()` og `insert()`.  
Að óbreyttu bæta bæði aftast í listann.

```
DoublyLinkedList<char> charList;  
charList.append('a');  
List: a  
charList.append('b');  
List: a b  
charList.append('c');  
List: a b c  
charList.insert('d');  
List: a b c d  
charList.insert('e');  
List: a b c d e  
charList.insert('f');  
List: a b c d e f
```

Höldum áfram og færum okkur á tiltekna staðsetningu og köllum nokkrum sinnum á `insert()`.  
Stökin koma inn fyrir framan núverandi staðsetningu í þeirri röð sem kallað er á `insert()`.

```
List: a b c d e f  
charList.move_to_pos(2);  
charList.insert('1');  
List: a b 1 c d e f  
charList.insert('2');  
List: a b 1 2 c d e f  
charList.insert('3');  
List: a b 1 2 3 c d e f
```

Ef við köllum á `append()` inn á milli hefur það engin áhrif á núverandi staðsetningu.

```
List: a b c d e f  
charList.move_to_pos(2);  
charList.insert('1');  
List: a b 1 c d e f  
charList.insert('2');  
List: a b 1 2 c d e f  
charList.append('x');  
List: a b 1 2 c d e f x  
charList.append('z');  
List: a b 1 2 c d e f x z  
charList.insert('3');  
List: a b 1 2 3 c d e f x z
```

Ef við aftur á móti færum núverandi staðsetningu á endann heldur hún áfram að vera á endanum, jafnvel þó að kallað sé á `append` á milli.

```
List: a b c d e f
charList.move_to_end();
charList.insert('1');
List: a b c d e f 1
charList.insert('2');
List: a b c d e f 1 2
charList.append('x');
List: a b c d e f 1 2 x
charList.append('z');
List: a b c d e f 1 2 x z
charList.insert('3');
List: a b c d e f 1 2 x z 3
```

Köllum svo á `remove()` á nokkrum mismunandi stöðum.

```
List: a b 1 2 3 c d e f x z
charList.move_to_pos(2);
charList.remove();
List: a b 2 3 c d e f x z
charList.remove();
List: a b 3 c d e f x z
charList.remove();
List: a b c d e f x z
charList.move_to_pos(4);
charList.remove();
List: a b c d f x z
charList.remove();
List: a b c d x z
charList.move_to_end();
charList.remove();
terminate called after throwing an instance of 'InvalidPositionException'
```