Swinburne University of Technology

School of Science, Computing and Emerging Technologies

ASSIGNMENT COVER SHEET

Subject Code: COS30008
Subject Title: Data Structures and Patterns
Assignment number and title: 4, List ADT
Due date: Sunday, June 1, 2025, 23:59
Lecturer: Dr. Markus Lumpe

Your name:_____ **Your student id:**_____

Marker's comments:

Problem	Marks	Obtained
1	134	
2	24	
3	21	
Total	179	

Extension certification:

This assignment has been given an extension and is now due on _____

Signature of Convener:_____

Problem Set 4: List ADT

Review the template classes `DoublyLinkedList` and `DoublyLinkedListIterator` developed in tutorial 11. Additionally, reviewing the lecture material on the construction of abstract data types and linked lists may be beneficial.

Use the header files provided on Canvas, as they have been fully tested. The classes have been revised to output protocol information about list node creation and destruction.

Using the template classes `DoublyLinkedList` and `DoublyLinkedListIterator`, implement the template class `List` as specified below:

```
#pragma once

#include "DoublyLinkedList.h"
#include "DoublyLinkedListIterator.h"

template<typename T>
class List
{
private:
  using node = typename DoublyLinkedList<T>::node;

  node fHead;
  node fTail;
  size_t fSize;

public:
  using iterator = DoublyLinkedListIterator<T>;

  List() noexcept;                                // default constructor     (2)
  ~List();                                        // destructor              (16)

  // copy semantics
  List( const List& aOther );                     // copy constructor        (10)
  List& operator=( const List& aOther );          // copy assignment         (14)

  // move semantics
  List( List&& aOther ) noexcept;                 // move constructor        (4)
  List& operator=( List&& aOther ) noexcept;      // move assignment         (8)
  void swap( List& aOther ) noexcept;             // swap elements           (9)

  // basic operations
  size_t size() const noexcept;                   // list size               (2)

  template<typename U>
  void push_front( U&& aData );                   // add element at front    (24)

  template<typename U>
  void push_back( U&& aData );                    // add element at back     (24)

  void remove( const T& aElement ) noexcept;      // remove element          (36)

  const T& operator[]( size_t aIndex ) const;     // list indexer            (14)

  // iterator interface
  iterator begin() const noexcept;                //                         (4)
  iterator end() const noexcept;                  //                         (4)
  iterator rbegin() const noexcept;               //                         (4)
  iterator rend() const noexcept;                 //                         (4)
};
```

Problem 1

Implement the basic list features.

The default list is empty. We require an explicitly defined destructor to avoid stack overflows while destroying smart pointers. The destructor runs a while loop over `fTail`. While `fTail` is not a **`nullptr`**, it is possible to reset its next element. In the following statement, `fTail` is set to the previous element, and the loop continues. This process allows all list elements to be destroyed from `fTail` to `fHead`. No stack overflow can occur.

The method `size()` returns the number of list elements.

To create new list elements, you need to use `DoublyLinkedNode`'s `makeNode()` method. For `push_front()` and `push_back()` to work properly, we need to use perfect forwarding when calling `makeNode()`. You need to insert the new element properly into the doubly-linked chain and update `fHead` and `fTail` (the list endpoints). Finally, adding an element to the list increases its size.

The method `remove()` deletes the first matching element from the list. If the list does not contain the specified element, it remains unchanged. Once found, the element to be removed must be isolated. It may also be necessary to adjust `fHead` and `fTail` when the removed element is the first or last element, respectively. The removed element is automatically destroyed when it goes out of scope. The `remove()` method deletes only the first occurrence.

The indexer has to return the corresponding element if the index is within bounds. The first element in the list has index 0. The last element has index `size()` – 1.

The iterator methods return the corresponding iterators. Refer to tutorial 11 for details on how to obtain these iterators.

To test your implementation of the basic features, uncomment **`#define`** `P1` and compile your solution. The test driver should produce the following output (the addresses will vary each time you run the program):

```
Test basic list functions:
Creating 'dddd' at address 0x6000009107e8
Creating 'cccc' at address 0x600000910838
Creating 'eeee' at address 0x600000910888
Creating 'bbbb' at address 0x6000009108d8
Creating 'ffff' at address 0x600000910928
Creating 'aaaa' at address 0x600000910978
List size: 6
5th element: eeee
Deleting 'eeee' at address 0x600000910888
Remove 5th element.
New 5th element: ffff
List size: 5
Forward iteration:
aaaa
bbbb
cccc
dddd
ffff
Backwards iteration:
ffff
dddd
cccc
bbbb
aaaa
Test basic list functions complete.
Deleting 'ffff' at address 0x600000910928
Deleting 'dddd' at address 0x6000009107e8
Deleting 'cccc' at address 0x600000910838
Deleting 'bbbb' at address 0x6000009108d8
Deleting 'aaaa' at address 0x600000910978
```

Problem 2

Implement the copy semantics.

To test your implementation of the basic features, uncomment **`#define`** `P2` and compile your solution. The test driver should produce the following output (the addresses will vary each time you run the program):

```
Test copy semantics:
Creating 'dddd' at address 0x6000014cc6f8
Creating 'cccc' at address 0x6000014cc748
Creating 'eeee' at address 0x6000014cc798
Creating 'bbbb' at address 0x6000014cc7e8
Creating 'ffff' at address 0x6000014cc838
Creating 'aaaa' at address 0x6000014cc888
Creating 'aaaa' at address 0x6000014cc8d8
Creating 'bbbb' at address 0x6000014cc928
Creating 'cccc' at address 0x6000014cc978
Creating 'dddd' at address 0x6000014cc9c8
Creating 'eeee' at address 0x6000014cca18
Creating 'ffff' at address 0x6000014cca68
Copied list iteration (source):
aaaa
bbbb
cccc
dddd
eeee
ffff
Copied list iteration (target):
aaaa
bbbb
cccc
dddd
eeee
ffff
Deleting 'ffff' at address 0x6000014cc838
Deleting 'eeee' at address 0x6000014cc798
Deleting 'dddd' at address 0x6000014cc6f8
Deleting 'cccc' at address 0x6000014cc748
Deleting 'bbbb' at address 0x6000014cc7e8
Deleting 'aaaa' at address 0x6000014cc888
Creating 'aaaa' at address 0x6000014cc888
Creating 'bbbb' at address 0x6000014cc7e8
Creating 'cccc' at address 0x6000014cc748
Creating 'dddd' at address 0x6000014cc6f8
Creating 'eeee' at address 0x6000014cc798
Creating 'ffff' at address 0x6000014cc838
Copied list iteration (source):
aaaa
bbbb
cccc
dddd
eeee
ffff
Copied list iteration (target):
aaaa
bbbb
cccc
dddd
eeee
ffff
Test copy semantics complete.
Deleting 'ffff' at address 0x6000014cca68
Deleting 'eeee' at address 0x6000014cca18
Deleting 'dddd' at address 0x6000014cc9c8
Deleting 'cccc' at address 0x6000014cc978
Deleting 'bbbb' at address 0x6000014cc928
Deleting 'aaaa' at address 0x6000014cc8d8
Deleting 'ffff' at address 0x6000014cc838
Deleting 'eeee' at address 0x6000014cc798
Deleting 'dddd' at address 0x6000014cc6f8
Deleting 'cccc' at address 0x6000014cc748
Deleting 'bbbb' at address 0x6000014cc7e8
Deleting 'aaaa' at address 0x6000014cc888
```

Problem 3

Implement the move semantics.

To test your implementation of the basic features, uncomment **`#define`** `P3` and compile your solution. The test driver should produce the following output (the addresses will vary each time you run the program):

```
Test move semantics:
Creating 'dddd' at address 0x6000036b47e8
Creating 'cccc' at address 0x6000036b4838
Creating 'eeee' at address 0x6000036b4888
Creating 'bbbb' at address 0x6000036b48d8
Creating 'ffff' at address 0x6000036b4928
Creating 'aaaa' at address 0x6000036b4978
Moved list iteration (source):
Moved list iteration (target):
aaaa
bbbb
cccc
dddd
eeee
ffff
Test move semantics complete.
Deleting 'ffff' at address 0x6000036b4928
Deleting 'eeee' at address 0x6000036b4888
Deleting 'dddd' at address 0x6000036b47e8
Deleting 'cccc' at address 0x6000036b4838
Deleting 'bbbb' at address 0x6000036b48d8
Deleting 'aaaa' at address 0x6000036b4978
```

Submission deadline: Sunday, June 1, 2025, 23:59.

Submission procedure:

Follow the instructions on Canvas. Submit electronically the PDF of the printed modified source file `List.h`. Upload the modified source file `List.h` to Canvas.

The sources will be assessed and compiled in the presence of the solution artifacts provided on Canvas.