Swinburne University of Technology

School of Science, Computing and Engineering Technologies

ASSIGNMENT COVER SHEET

Subject Code:	COS30008
Subject Title:	Data Structures and Patterns
Assignment number and title:	4, List ADT
Due date:	Friday, May 24, 2024, 10:30
Lecturer:	Dr. Markus Lumpe

Your name: Dang Khoa Le **Your student id:** 103844421

Marker's comments:

Problem	Marks	Obtained
1	118	
2	24	
3	21	
Total	163	

Extension certification:

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

Signature of Convener:____________

```
// COS30008, Problem Set 4, 2024

#pragma once

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

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

  Node fHead;  // first element
  Node fTail;  // last element
  size_t fSize;     // number of elements

public:

  using Iterator = DoublyLinkedListIterator<T>;

    List() noexcept : fHead(nullptr), fTail(nullptr), fSize(0) {}         // default
constructor  (2)

  // copy semantics
    List(const List& aOther) : fHead(nullptr), fTail(nullptr), fSize(0) { // copy constructor
(10)
        for (auto& item : aOther) {
            push_back(item);
        }
    }

    List& operator=(const List& aOther) {              // copy assignment  (14)
        if (this != &aOther) {
            List temp(aOther);
            swap(temp);
        }
        return *this;
    }

  // move semantics
    List( List&& aOther ) noexcept : fHead(nullptr), fTail(nullptr), fSize(0) { // move
constructor (4)
        swap(aOther);
    }

    List& operator=( List&& aOther ) noexcept{       // move assignment  (8)
    if (this != &aOther) {
                swap(aOther);
            }
            return *this;
        }

    void swap( List& aOther ) noexcept{        // swap elements  (9)
    std::swap(fHead, aOther.fHead);
            std::swap(fTail, aOther.fTail);
            std::swap(fSize, aOther.fSize);
        }

  // basic operations
    size_t size() const noexcept {     // list size  (2)
        return fSize;
    }

  template<typename U>
    void push_front(U&& aData) {                     // add element at front  (24)
        Node newNode = DoublyLinkedList<T>::makeNode(std::forward<U>(aData));
```

```cpp
        if (!fHead) {
            fHead = fTail = newNode;
        } else {
            newNode->fNext = fHead;
            fHead->fPrevious = newNode;
            fHead = newNode;
        }
        fSize++;
    }

  template<typename U>
    void push_back(U&& aData) {                    // add element at back  (24)
            Node newNode = DoublyLinkedList<T>::makeNode(std::forward<U>(aData));
            if (!fTail) {
                fHead = fTail = newNode;
            } else {
                newNode->fPrevious = fTail;
                fTail->fNext = newNode;
                fTail = newNode;
            }
            fSize++;
        }

    void remove(const T& aElement) noexcept {     // remove element  (36)
            Node current = fHead;
            while (current) {
                if (current->fData == aElement) {
                    if (current == fHead) {
                        fHead = current->fNext;
                        if (fHead) {
                            fHead->fPrevious.reset();
                        } else {
                            fTail = nullptr;
                        }
                    } else if (current == fTail) {
                        fTail = current->fPrevious.lock();
                        if (fTail) {
                            fTail->fNext.reset();
                        } else {
                            fHead = nullptr;
                        }
                    } else {
                        Node prev = current->fPrevious.lock();
                        Node next = current->fNext;
                        if (prev) {
                            prev->fNext = next;
                        }
                        if (next) {
                            next->fPrevious = current->fPrevious;
                        }
                    }
                    current->isolate();
                    fSize--;
                    break;
                }
                current = current->fNext;
            }
        }

    const T& operator[](size_t aIndex) const {      // list indexer  (14)
            if (aIndex >= fSize) {
                throw std::out_of_range("Index out of range");
            }
            Node current = fHead;
            for (size_t i = 0; i < aIndex; i++) {
                current = current->fNext;
            }
            return current->fData;
        }

// iterator interface
  Iterator begin() const noexcept {                 // (4)
            return Iterator(fHead, fTail);
        }

        Iterator end() const noexcept {               // (4)
            return Iterator(fHead, fTail).end();
        }

        Iterator rbegin() const noexcept {            // (4)
            return Iterator(fHead, fTail).rbegin();
        }

        Iterator rend() const noexcept {              // (4)
            return Iterator(fHead, fTail).rend();
        }
  };
```