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:	3, List ADT
Due date:	Monday, May 15, 2023, 10:30
Lecturer:	Dr. Markus Lumpe

Your name: Md Redwan Ahmed Zawad___ **Your student id: 103501849_____**

Check Tutorial	Tues 08:30	Tues 10:30	Tues 12:30 BA603	Tues 12:30 ATC627	Tues 14:30	Wed 08:30	Wed 10:30	Wed 12:30	Wed 14:30	Thurs 08:30	Thurs 10:30
			✓								

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 3, 2023

#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(),
        fTail(),
        fSize()
    {}// default constructor

      // copy semantics
    List(const List& aOther)                                // copy constructor
    {
        *this = aOther;
    }
    List& operator=( const List& aOther )          // copy assignment
    {
        if (this != &aOther)
        {
            fHead = aOther.fHead;
            fTail= aOther.fTail;
            fSize = aOther.fSize;

        }
        return *this;
    }
      // move semantics
    List(List&& aOther) noexcept                   // move constructor
    {
        swap(aOther);
    }
    List& operator=(List&& aOther) noexcept   // move assignment

    {
        if (this != &aOther)
        {
            swap(aOther);
        }
        return *this;
    }
    void swap(List& aOther) noexcept               // swap elements
    {
        std::swap(fHead, aOther.fHead);
        std::swap(fTail, aOther.fTail);
        std::swap(fSize, aOther.fSize);
    }
      // basic operations
    size_t size() const noexcept { return fSize; }                 // list size

    template<typename U>
    void push_front(U&& aData)                     // add element at front
    {
        Node lnode = DoublyLinkedList<T>::makeNode(aData);
        if (fHead)
    {
        lnode->fNext = fHead;
        fHead->fPrevious = lnode;

        fHead = lnode;


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

        fTail = lNode;

    }
    else
    {
        if (fHead)
        {
            lNode->fPrevious = fHead;
            fHead->fNext = lNode;
        }
        fTail = lNode;
    }
    fSize++;
}
void remove(const T& aElement) noexcept   // remove element
{
    Node lnode = fHead;
    while (lnode)
    {
        if (lnode->fData == aElement)
        {
            if (lnode->fPrevious.lock())
            {
                lnode->fPrevious.lock()->fNext = lnode->fNext;
            }
            else
            {
                fHead->fNext = lnode->fNext;
            }
            if (lnode->fNext)
            {
                lnode->fNext->fPrevious = lnode->fPrevious;
            }
            else
            {
                fTail->fPrevious = lnode->fPrevious;
            }
            fSize--;
            lnode->isolate();
            return;
        }
        lnode=lnode->fNext;
    }
}
const T& operator[](size_t aIndex) const // list indexer
```

```
    {
        Iterator lopera= Iterator(fHead, fTail);;
        if (aIndex < fSize && aIndex>0)
        {

            for (size_t i = 0; i != aIndex; i++)
            {
                lopera++;
            }
        }
        return *lopera;
    }
      // iterator interface
    Iterator begin() const noexcept
    {
        return Iterator(fHead, fTail).begin();
    }
    Iterator end() const noexcept
    {



        return Iterator(fHead, fTail).end();

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


        return Iterator(fHead, fTail).rend();
    }
};
```

1