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: Dhanveer Ramnauth **Your student id:** 103866373

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:____________________________

Figure 1: SortablePair.h

```cpp

// COS30008, Problem Set 4, 2025

#pragma once

#include "DoublyLinkedList.h"
#include "DoublyLinkedListIterator.h"
#include <cassert>
#include <utility>

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
        : fHead(nullptr), fTail(nullptr), fSize(0)
    {}

    ~List()
    {
        while (fTail)
        {
            node previous = fTail->previous.lock();
            fTail->next.reset();
            fTail->previous.reset();
            fTail = previous;
        }
        fHead.reset();
    }

    // copy constructor
    List( const List& aOther )
        : fSize(aOther.fSize)
    {
        for (node cur = aOther.fHead; cur; cur = cur->next)
        {
            push_back(cur->data);
        }
    }

    // Copy assignment
    List& operator=( const List& aOther )
    {
        if (this != &aOther)
        {
            this->~List();

            new (this) List(aOther);
        }
        return *this;
    }

    // move constructor
    List( List&& aOther ) noexcept
        : fHead(aOther.fHead), fTail(aOther.fTail), fSize(aOther.fSize)
    {
        // by setting all these things to nullptr
        // the destructor won't be able to destroy the aOther list.
        aOther.fHead = nullptr;
        aOther.fTail = nullptr;
        aOther.fSize = 0;
    }

    // Move assignment
    List& operator=( List&& aOther ) noexcept
    {
        if (this != &aOther)
        {
            this->~List();
            swap(aOther);
        }
        return *this;
    }

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

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

    template<typename U>
    void push_front( U&& aData )
    {
        node newNode = DoublyLinkedList<T>::makeNode(std::forward<U>(aData));
        newNode->previous.reset();
        newNode->next = fHead;

        if (fHead)
            fHead->previous = newNode;

        fHead = newNode;

        if (!fTail) // empty list
            fTail = newNode;

        fSize++;
    }

    template<typename U>
    void push_back( U&& aData )
    {
        node newNode = DoublyLinkedList<T>::makeNode(std::forward<U>(aData));
        newNode->next.reset();
        newNode->previous = fTail;

        if (fTail)
            fTail->next = newNode;

        fTail = newNode;

        if (!fHead) // empty list
            fHead = newNode;

        fSize++;
    }

    void remove( const T& aElement ) noexcept
    {
        node cur = fHead;

        while (cur)
        {
            if (cur->data == aElement)
            {
                if (cur == fHead)
                    fHead = cur->next;

                if (cur == fTail)
                    fTail = cur->previous.lock();

                cur->isolate();

                fSize--;
                return;
            }
            cur = cur->next;
        }
    }

    const T& operator[]( size_t aIndex ) const
    {
        assert(aIndex < fSize);

        size_t i = 0;
        for (node cur = fHead; cur; cur = cur->next)
        {
            if (i++ == aIndex)
                return cur->data;
        }
    }

    // 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();
    }
};
```