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: Xuan Tuan Minh Nguyen **Your student id:** 103819212

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:_____

```
 1
 2  // COS30008, Problem Set 4, 2024
 3
 4  #pragma once
 5
 6  #include "DoublyLinkedList.h"
 7  #include "DoublyLinkedListIterator.h"
 8  #include <cstddef> // for size_t
 9  #include <iostream>
10
11  template <typename T>
12  class List
13  {
14  private:
15      using Node = typename DoublyLinkedList<T>::Node;
16
17      Node fHead;   // first element
18      Node fTail;   // last element
19      size_t fSize; // number of elements
20
21  public:
22      using Iterator = DoublyLinkedListIterator<T>;
23
24      List() noexcept : fHead(nullptr), fTail(nullptr), fSize(0) {} //
          default constructor
25
26      // copy constructor
27      List(const List &aOther) : fHead(nullptr), fTail(nullptr), fSize(0)
28      {
29          for (Node nodeIndex = aOther.fHead; nodeIndex; nodeIndex =
              nodeIndex->fNext)
30          {
31              push_back(nodeIndex->fData);
32          }
33      }
34
35      // copy assignment
36      List &operator=(const List &aOther)
37      {
38          if (this != &aOther)
39          {
40              List temp(aOther);
41              swap(temp);
42          }
43          return *this;
44      }
45
46      // move constructor
47      List(List &&aOther) noexcept : fHead(std::move(aOther.fHead)), fTail
```

```
          (std::move(aOther.fTail)), fSize(aOther.fSize)
48      {
49          aOther.fSize = 0;
50          aOther.fHead = nullptr;
51          aOther.fTail = nullptr;
52      }
53
54      // move assignment
55      List &operator=(List &&aOther) noexcept
56      {
57          if (this != &aOther)
58          {
59              fHead = std::move(aOther.fHead);
60              fTail = std::move(aOther.fTail);
61              fSize = aOther.fSize;
62              aOther.fSize = 0;
63              aOther.fHead = nullptr;
64              aOther.fTail = nullptr;
65          }
66          return *this;
67      }
68
69      void swap(List &aOther) noexcept
70      {
71          std::swap(fHead, aOther.fHead);
72          std::swap(fTail, aOther.fTail);
73          std::swap(fSize, aOther.fSize);
74      }
75
76      size_t size() const noexcept
77      {
78          return fSize;
79      }
80
81      template <typename U>
82      void push_front(U &&aData)
83      {
84          Node tempNode = DoublyLinkedList<T>::makeNode(std::forward<U>
              (aData));
85          if (fHead)
86          {
87              tempNode->fNext = fHead;
88              fHead->fPrevious = tempNode;
89          }
90          else
91          {
92              fTail = tempNode;
93          }
94          fHead = tempNode;
```

```
 95          ++fSize;
 96      }
 97
 98      template <typename U>
 99      void push_back(U &&aData)
100      {
101          Node tempNode = DoublyLinkedList<T>::makeNode(std::forward<U>
              (aData));
102          if (fTail)
103          {
104              tempNode->fPrevious = fTail;
105              fTail->fNext = tempNode;
106          }
107          else
108          {
109              fHead = tempNode;
110          }
111          fTail = tempNode;
112          ++fSize;
113      }
114
115      void remove(const T &aElement) noexcept
116      {
117          for (Node nodeIndex = fHead; nodeIndex; nodeIndex = nodeIndex-
              >fNext)
118          {
119              if (nodeIndex->fData == aElement)
120              {
121                  if (nodeIndex == fHead)
122                  {
123                      fHead = nodeIndex->fNext;
124                      if (fHead)
125                      {
126                          fHead->fPrevious.reset();
127                      }
128                      else
129                      {
130                          fTail.reset();
131                      }
132                  }
133                  else if (nodeIndex == fTail)
134                  {
135                      fTail = nodeIndex->fPrevious.lock();
136                      if (fTail)
137                      {
138                          fTail->fNext.reset();
139                      }
140                      else
141                      {
```

```
142                             fHead.reset();
143                         }
144                     }
145                     else
146                     {
147                         nodeIndex->isolate();
148                     }
149                     --fSize;
150                     return;
151                 }
152             }
153         }
154
155         const T &operator[](size_t aIndex) const
156         {
157             Node tempNode = fHead;
158             for (size_t i = 0; i < aIndex; ++i)
159             {
160                 tempNode = tempNode->fNext;
161             }
162             return tempNode->fData;
163         }
164
165         Iterator begin() const noexcept
166         {
167             return Iterator(fHead, fTail).begin();
168         }
169
170         Iterator end() const noexcept
171         {
172             return Iterator(fHead, fTail).end();
173         }
174
175         Iterator rbegin() const noexcept
176         {
177             return Iterator(fHead, fTail).rbegin();
178         }
179
180         Iterator rend() const noexcept
181         {
182             return Iterator(fHead, fTail).rend();
183         }
184     };
185
```