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:_____   **Your student id:**_____

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:_____

```cpp
 1
 2  // COS30008, Problem Set 4, 2024
 3
 4  #pragma once
 5
 6  #include "DoublyLinkedList.h"
 7  #include "DoublyLinkedListIterator.h"
 8
 9  #include <cassert>
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
23      using Iterator = DoublyLinkedListIterator<T>;
24
25      List() noexcept :                                // default      ⏎
          constructor (2)
26          fHead(nullptr), fTail(nullptr), fSize(0)
27      {}
28
29      // copy semantics
30      List(const List& aOther) : fSize(aOther.fSize)              // ⏎
          copy constructor        (10)
31      {
32          for (Node lCurrent = aOther.fHead; lCurrent != nullptr;     ⏎
            lCurrent = lCurrent->fNext)
33          {
34              push_back(lCurrent->fData);
35          }
36      }
37
38      List& operator=(const List& aOther)     // copy assignment      ⏎
          (14)
39      {
40          if (this != &aOther)
41          {
42              List lTemp(aOther);
43              swap(lTemp);
44          }
45          return *this;
46      }
47
48      // move semantics
49      List(List&& aOther) noexcept :                  // move constructor     ⏎
```

```
                (4)
50          fHead(std::move(aOther.fHead)), fTail(std::move(aOther.fTail)), ⮰
                fSize(aOther.fSize)
51      {
52          aOther.fSize = 0;
53      }
54
55      List& operator=(List&& aOther) noexcept // move assignment      (8)
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          }
64          return *this;
65      }
66
67      void swap(List& aOther) noexcept              // swap elements      ⮰
                (9)
68      {
69          std::swap(fHead, aOther.fHead);
70          std::swap(fTail, aOther.fTail);
71          std::swap(fSize, aOther.fSize);
72      }
73
74      // basic operations
75      size_t size() const noexcept                        // list size      ⮰
                (2)
76      {
77          return fSize;
78      }
79
80      template<typename U>
81      void push_front(U&& aData)                    // add element at front ⮰
                (24)
82      {
83          Node lNode = DoublyLinkedList<T>::makeNode(std::forward<U>   ⮰
                (aData));
84          if (!fHead)
85          {
86              fHead = lNode;
87              fTail = lNode;
88          }
89          else
90          {
91              lNode->fNext = fHead;
92              fHead->fPrevious = lNode;
93              fHead = lNode;
94          }
95          ++fSize;
96      }
```

```cpp
 97
 98        template<typename U>
 99        void push_back(U&& aData)                        // add element at back  ⮑
               (24)
100        {
101            Node lNode = DoublyLinkedList<T>::makeNode(std::forward<U>        ⮑
                   (aData));
102            if (!fHead)
103            {
104                fHead = lNode;
105                fTail = lNode;
106            }
107            else
108            {
109                fTail->fNext = lNode;
110                lNode->fPrevious = fTail;
111                fTail = lNode;
112            }
113            ++fSize;
114        }
115
116        void remove(const T& aElement) noexcept // remove element              ⮑
             (36)
117        {
118            Node lCurrent = fHead;
119            while (lCurrent)
120            {
121                if (lCurrent->fData == aElement)
122                {
123                    if (lCurrent == fHead)
124                    {
125                        fHead = lCurrent->fNext;
126                        if (fHead)
127                        {
128                            fHead->fPrevious.reset();
129                        }
130                        else
131                        {
132                            fTail.reset();
133                        }
134                    }
135                    else if (lCurrent == fTail)
136                    {
137                        fTail = lCurrent->fPrevious.lock();
138                        if (fTail)
139                        {
140                            fTail->fNext.reset();
141                        }
142                        else
143                        {
144                            fHead.reset();
145                        }
146                    }
```

```
147                 else
148                 {
149                     lCurrent->isolate();
150                 }
151                 --fSize;
152                 break;
153             }
154             lCurrent = lCurrent->fNext;
155         }
156     }
157
158     const T& operator[](size_t aIndex) const     // list indexer      ⏎
            (14)
159     {
160         assert(aIndex < fSize);
161         Node lCurrent = fHead;
162         for (size_t i = 0; i < aIndex; ++i) {
163             lCurrent = lCurrent->fNext;
164         }
165         return lCurrent->fData;
166     }
167
168     // iterator interface
169     Iterator begin() const noexcept             //                    ⏎
            (4)
170     {
171         return Iterator(fHead, fTail).begin();
172     }
173
174     Iterator end() const noexcept               //                    ⏎
            (4)
175     {
176         return Iterator(fHead, fTail).end();
177     }
178
179     Iterator rbegin() const noexcept                //                ⏎
            (4)
180     {
181         return Iterator(fHead, fTail).rbegin();
182     }
183
184     Iterator rend() const noexcept              //                    ⏎
            (4)
185     {
186         return Iterator(fHead, fTail).rend();
187     }
188
189 };
190
```