# 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:** Rox Nguyen **Your student id:** 103500095

---

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  // COS30008, Problem Set 4, 2024
2
3  #pragma once
4
5  #include "DoublyLinkedList.h"
6  #include "DoublyLinkedListIterator.h"
7
8  template<typename T>
9  class List
10 {
11 private:
12     using Node = typename DoublyLinkedList<T>::Node;
13
14     Node fHead;     // first element
15     Node fTail;     // last element
16     size_t fSize;   // number of elements
17
18 public:
19
20     using Iterator = DoublyLinkedListIterator<T>;
21
22     // default constructor  (2)
23     List() noexcept : fHead(nullptr), fTail(nullptr), fSize(0)
24     {}
25
26     // copy semantics
27     List(const List& aOther) : fSize(aOther.fSize)
28     {
29         fHead = fTail = nullptr;
30         Node lNode = aOther.fHead;
31         while (lNode != nullptr)
32         {
33             push_back(lNode->fData);
34             lNode = lNode->fNext;
35         }
36     }
37
38     List& operator=(const List& aOther)
39     {
40         if (this != &aOther)
41         {
42             List lTemp(aOther);
43             Node lNode = fHead;
44             while (lNode)
45                 //List lTemp(aOther); // copy-and-swap idiom -> swap(lTemp)
46             {
47                 Node lNext = lNode->fNext;
48                 lNode = lNext;
49             }
50         }
51         return *this;
52     }
53
54     // move semantics
55     List(List&& aOther) noexcept : fHead(std::move(aOther.fHead)), fTail
         (std::move(aOther.fTail)), fSize(std::move(aOther.fSize))
```

```
56      {
57          aOther.fHead = nullptr;
58          aOther.fTail = nullptr;
59          aOther.fSize = 0;
60      }
61
62      List& operator=(List&& aOther) noexcept
63      {
64          if (this != &aOther)
65          {
66              fHead = std::move(aOther.fHead);
67              fTail = std::move(aOther.fTail);
68              fSize = aOther.fSize;
69              aOther.fHead = nullptr;
70              aOther.fTail = nullptr;
71              aOther.fSize = 0;
72          }
73          return *this;
74      }
75
76      void swap(List& aOther) noexcept
77      {
78          std::swap(fHead, aOther.fHead);
79          std::swap(fTail, aOther.fTail);
80          std::swap(fSize, aOther.fSize);
81      }
82
83      // basic operations
84      size_t size() const noexcept
85      {
86          return fSize;
87      }
88
89      // add element at front (24)
90      template<typename U>
91      void push_front(U&& aData)
92      {
93          Node lNode = DoublyLinkedList<T>::makeNode(std::forward<U>(aData));
94          if (!fHead)
95          { // If list is empty
96              fHead = fTail = lNode;
97          }
98          else
99          { // Non-empty list
100             lNode->fNext = fHead;
101             fHead->fPrevious = lNode;
102             fHead = lNode;
103         }
104         fSize++;
105     }
106
107     // add element at back  (24)
108     template<typename U>
109     void push_back(U&& aData)
110     {
111         Node lNode = DoublyLinkedList<T>::makeNode(std::forward<U>(aData));
```

```
112         if (!fTail)
113         { // If list is empty
114             fHead = fTail = lNode;
115         }
116         else
117         {
118             lNode->fPrevious = fTail;
119             fTail->fNext = lNode;
120             fTail = lNode;
121         }
122         fSize++;
123     }
124
125     // remove element       (36)
126     void remove(const T& aElement) noexcept
127     {
128         Node lNode = fHead;
129
130         while (lNode)
131         {
132             if (lNode->fData == aElement)
133             {
134                 lNode->isolate();
135             }
136             lNode = lNode->fNext;
137         }
138         fSize--;
139     }
140
141     // list indexer         (14)
142     const T& operator[](size_t aIndex) const
143     {
144         Node lNode = fHead;
145         for (size_t i = 0; i < aIndex; i++)
146         {
147             lNode = lNode->fNext;
148         }
149         return lNode->fData;
150     }
151
152     // iterator interface
153     Iterator begin() const noexcept
154     {
155         return Iterator(fHead, fTail);
156     }
157
158     Iterator end() const noexcept
159     {
160         return Iterator(fHead, fTail).end();
161     }
162
163     Iterator rbegin() const noexcept
164     {
165         return Iterator(fHead, fTail).rbegin();
166     }
167
```

```
168     Iterator rend() const noexcept
169     {
170         return Iterator(fHead, fTail).rend();
171     }
172 };
173
```