

**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:** \_\_\_\_\_ **Your student id:** \_\_\_\_\_

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: \_\_\_\_\_

```
1
2 // COS30008, Problem Set 3, 2023
3
4 #pragma once
5
6 #include "DoublyLinkedList.h"
7 #include "DoublyLinkedListIterator.h"
8
9 template<typename T>
10 class List
11 {
12 private:
13     using Node = typename DoublyLinkedList<T>::Node;
14
15     Node fHead;    // first element
16     Node fTail;    // last element
17     size_t fSize;  // number of elements
18
19 public:
20
21     using Iterator = DoublyLinkedListIterator<T>;
22
23     List() noexcept :
24         fHead(),
25         fTail(),
26         fSize(0)
27     {} // default constructor
28
29     // copy semantics
30     List(const List& aOther) :
31         fHead(aOther.fHead),
32         fTail(aOther.fTail),
33         fSize(aOther.fSize)
34     {} // copy constructor
35     List& operator=(const List& aOther) {
36
37         if (this != aOther) {
38
39             this->~List();
40
41             new (this) List(aOther);
42         }
43
44         return *this;
45     } // copy assignment
46
47     // move semantics
48     List(List&& aOther) noexcept :
49         List()
```

```
50     {
51         swap(aOther);
52     } // move constructor
53     List& operator=(List&& aOther) noexcept {
54
55         if (this != &aOther) {
56             swap(aOther);
57         }
58
59         return *this;
60     } // move assignment
61     void swap(List& aOther) noexcept {
62         std::swap(fHead, aOther.fHead);
63         std::swap(fTail, aOther.fTail);
64         std::swap(fSize, aOther.fSize);
65     } // swap elements
66
67     // basic operations
68     size_t size() const noexcept {
69         return fSize;
70     } // list size
71
72     template<typename U>
73     void push_front(U&& aData) {
74         Node aNode = DoublyLinkedList<T>::makeNode(aData);
75
76         aNode->fNext = fHead;
77         if (fHead != nullptr) fHead->fPrevious = aNode;
78
79         fHead = aNode;
80
81         if (fTail == nullptr) fTail = fHead;
82         fSize++;
83     } // add element at front
84
85     template<typename U>
86     void push_back(U&& aData) {
87         Node aNode = DoublyLinkedList<T>::makeNode(aData);
88
89         aNode->fPrevious = fTail;
90         if (fTail != nullptr) fTail->fNext = aNode;
91
92         fTail = aNode;
93         if (fHead == nullptr) fHead = fTail;
94         fSize++;
95     } // add element at back
96
97     void remove(const T& aElement) noexcept {
98         Node remove = fHead;
```

```
100     while (remove->fData != aElement && remove != nullptr) {
101         remove = remove->fNext;
102     }
103     if (remove->fData == aElement) {
104         (remove)->isolate();
105     }
106     fSize--;
107 }// remove element
108
109 const T& operator[](size_t aIndex) const {
110     assert(aIndex < fSize);
111
112     Node result = fHead;
113
114     for (size_t i = 1; i <= aIndex; i++) {
115         result = result->fNext;
116     }
117
118     return result->fData;
119 }// list indexer
120
121 // iterator interface
122 Iterator begin() const noexcept {
123     Iterator aBegin = Iterator(fHead, fTail);
124
125     return aBegin.begin();
126 }
127
128 Iterator end() const noexcept {
129     Iterator aEnd = Iterator(fHead, fTail);
130
131     return aEnd.end();
132 }
133
134 Iterator rbegin() const noexcept {
135     Iterator aRBegin = Iterator(fHead, fTail);
136
137     return aRBegin.rbegin();
138 }
139
140 Iterator rend() const noexcept {
141     Iterator aREnd = Iterator(fHead, fTail);
142
143     return aREnd.rend();
144 }
145
146 };
```