

**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: \_\_\_\_\_

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

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

```
96         fHead = newNode;
97     }
98
99     fTail = newNode;
100    fSize++;
101 }
102
103 void remove(const T& aElement) noexcept {
104     for (Node current = fHead; current; current = current->fNext) {
105         if (current->fData == aElement) {
106             if (current == fHead) {
107                 fHead = current->fNext;
108             }
109             if (current == fTail) {
110                 fTail = current->fPrevious.lock();
111             }
112             current->isolate();
113             fSize--;
114             break;
115         }
116     }
117 }
118
119 const T& operator[](size_t aIndex) const {
120     if (aIndex >= fSize) {
121         throw std::out_of_range("Index out of bounds");
122     }
123
124     Node current = fHead;
125     for (size_t i = 0; i < aIndex; ++i) {
126         current = current->fNext;
127     }
128     return current->fData;
129 }
130
131 // iterator interface
132 Iterator begin() const noexcept {
133     return Iterator(fHead, fTail).begin();
134 }
135
136 Iterator end() const noexcept {
137     return Iterator(fHead, fTail).end();
138 }
139
140 Iterator rbegin() const noexcept {
141     return Iterator(fHead, fTail).rbegin();
142 }
143
144 Iterator rend() const noexcept {
```

```
145         return Iterator(fHead, fTail).rend();  
146     }  
147 };
```