

**Swinburne University of Technology**

School of Science, Computing and Emerging Technologies

**ASSIGNMENT COVER SHEET**

---

**Subject Code:** COS30008  
**Subject Title:** Data Structures and Patterns  
**Assignment number and title:** 4, List ADT  
**Due date:** Sunday, June 1, 2025, 23:59  
**Lecturer:** Dr. Markus Lumpe

---

**Your name:** Dhanveer Ramnauth      **Your student id:** 103866373

---

Marker's comments:

Problem	Marks	Obtained
1	134	
2	24	
3	21	
Total	179	

---

**Extension certification:**

This assignment has been given an extension and is now due on \_\_\_\_\_

Signature of Convener: \_\_\_\_\_

Figure 1: SortablePair.h

```
1
2 // CDS30008, Problem Set 4, 2025
3
4 #pragma once
5
6 #include "DoublyLinkedList.h"
7 #include "DoublyLinkedListIterator.h"
8 #include <cassert>
9 #include <utility>
10
11 template<typename T>
12 class List
13 {
14 private:
15     using node = typename DoublyLinkedList<T>::node;
16
17     node fHead;
18     node fTail;
19     size_t fSize;
20
21 public:
22
23     using iterator = DoublyLinkedListIterator<T>;
24
25     List() noexcept
26         : fHead(nullptr), fTail(nullptr), fSize(0)
27     {}
28
29     ~List()
30     {
31         while (fTail)
32         {
33             node previous = fTail->previous.lock();
34             fTail->next.reset();
35             fTail->previous.reset();
36             fTail = previous;
37         }
38         fHead.reset();
39     }
40
```

```

41 // copy constructor
42 List( const List& aOther )
43     : fSize(aOther.fSize)
44 {
45     for (node cur = aOther.fHead; cur; cur = cur->next)
46     {
47         push_back(cur->data);
48     }
49 }
50
51 // Copy assignment
52 List& operator=( const List& aOther )
53 {
54     if (this != &aOther)
55     {
56         this->~List();
57
58         new (this) List(aOther);
59     }
60     return *this;
61 }
62
63 // move constructor
64 List( List&& aOther ) noexcept
65     : fHead(aOther.fHead), fTail(aOther.fTail),
↪ fSize(aOther.fSize)
66 {
67     // by setting all these things to nullptr
68     // the destructor won't be able to destroy the aOther
↪ list.
69     aOther.fHead = nullptr;
70     aOther.fTail = nullptr;
71     aOther.fSize = 0;
72 }
73
74 // Move assignment
75 List& operator=( List&& aOther ) noexcept
76 {
77     if (this != &aOther)
78     {

```

```

79         this->~List();
80         swap(aOther);
81     }
82     return *this;
83 }
84
85 void swap( List& aOther ) noexcept
86 {
87     std::swap(fHead, aOther.fHead);
88     std::swap(fTail, aOther.fTail);
89     std::swap(fSize, aOther.fSize);
90 }
91
92 // basic operations
93 size_t size() const noexcept
94 {
95     return fSize;
96 }
97
98 template<typename U>
99 void push_front( U&& aData )
100 {
101     node newNode =
↪ DoublyLinkedList<T>::makeNode(std::forward<U>(aData));
102     newNode->previous.reset();
103     newNode->next = fHead;
104
105     if (fHead)
106         fHead->previous = newNode;
107
108     fHead = newNode;
109
110     if (!fTail) // empty list
111         fTail = newNode;
112
113     fSize++;
114 }
115
116 template<typename U>
117 void push_back( U&& aData )

```

```

118     {
119         node newNode =
↪     DoublyLinkedList<T>::makeNode(std::forward<U>(aData));
120         newNode->next.reset();
121         newNode->previous = fTail;
122
123         if (fTail)
124             fTail->next = newNode;
125
126         fTail = newNode;
127
128         if (!fHead) // empty list
129             fHead = newNode;
130
131         fSize++;
132     }
133
134     void remove( const T& aElement ) noexcept
135     {
136         node cur = fHead;
137
138         while (cur)
139         {
140             if (cur->data == aElement)
141             {
142                 if (cur == fHead)
143                     fHead = cur->next;
144
145                 if (cur == fTail)
146                     fTail = cur->previous.lock();
147
148                 cur->isolate();
149
150                 fSize--;
151                 return;
152             }
153             cur = cur->next;
154         }
155     }
156

```

```
157     const T& operator[]( size_t aIndex ) const
158     {
159         assert(aIndex < fSize);
160
161         size_t i = 0;
162         for (node cur = fHead; cur; cur = cur->next)
163         {
164             if (i++ == aIndex)
165                 return cur->data;
166         }
167     }
168
169     // iterator interface
170     iterator begin() const noexcept
171     {
172         return iterator(fHead, fTail).begin();
173     }
174
175     iterator end() const noexcept
176     {
177         return iterator(fHead, fTail).end();
178     }
179
180     iterator rbegin() const noexcept
181     {
182         return iterator(fHead, fTail).rbegin();
183     }
184
185     iterator rend() const noexcept
186     {
187         return iterator(fHead, fTail).rend();
188     }
189 };
```