

Swinburne University of Technology

Faculty of Science, Engineering and Technology

ASSIGNMENT COVER SHEET

Subject Code: COS30008
Subject Title: Data Structures and Patterns
Assignment number and title: 3, List ADT
Due date: November 4, 2024, 14:30
Lecturer: Dr. Ky Trung Pham
StudentName: Lau Ngoc Quyen
StudentId: 104198996

Check Tutorial	Mon 10:30	Mon 14:30	Tues 08:30	Tues 10:30	Tues 12:30	Tues 14:30	Tues 16:30	Wed 08:30	Wed 10:30	Wed 12:30	Wed 14:30
	X										

Marker's comments:

Problem	Marks	Obtained
1	48	
2	28	
3	26	
4	30	
5	42	
Total	174	

Extension certification:

This assignment has been given an extension and is now due on 4th November, 2024


Signature of Convener: _____

1

ListPS3.h

```
1  #pragma once
2
3  #include "DoublyLinkedList.h"
4  #include "DoublyLinkedListIterator.h"
5  #include <stdexcept>
6
7  template<typename T>
8  class List
9  {
10 private:
11     using Node = DoublyLinkedList<T>;
12     Node* fRoot;
13     size_t fCount;
14
15 public:
16     using Iterator = DoublyLinkedListIterator<T>;
17
18     ~List()
19     {
20         while (fRoot != nullptr)
21         {
22             if (fRoot != &fRoot->getPrevious())
23             {
24                 Node* lTemp = const_cast<Node*>(&fRoot->getPrevious());
25                 lTemp->isolate();
26                 delete lTemp;
27             }
28             else
29             {
30                 delete fRoot;
31                 break;
32             }
33         }
34     }
35
36     void remove(const T& aElement)
37     {
38         Node* lNode = fRoot;
39         while (lNode != nullptr)
40         {
41             if (*lNode == aElement)
42             {
43                 break;
44             }
45             if (lNode != &fRoot->getPrevious())
46             {
47                 lNode = const_cast<Node*>(&lNode->getNext());
48             }
49             else
50             {
51                 lNode = nullptr;
52             }
53         }
54         if (lNode != nullptr)
55         {
56             if (fCount != 1)
57             {
58                 if (lNode == fRoot)
59                 {
60                     fRoot = const_cast<Node*>(&fRoot->getNext());
61                 }
62             }
63             else
64             {
65                 fRoot = nullptr;
66             }
67             lNode->isolate();
68             delete lNode;
69             fCount--;
70         }
71     }
72 }
```


```
1 // Problem 1
2 List() : fRoot(nullptr), fCount(0) {}
3
4 bool empty() const
5 {
6     return fRoot == nullptr;
7 }
8
9 size_t size() const
10 {
11     return fCount;
12 }
13
14 void push_front(const I& aElement)
15 {
16     if (empty())
17     {
18         fRoot = new Node(aElement);
19     }
20     else
21     {
22         Node* lNode = new Node(aElement);
23         fRoot->push_front(*lNode);
24         fRoot = lNode;
25     }
26     ++fCount;
27 }
28
29 Iterator begin() const
30 {
31     return Iterator(fRoot).begin();
32 }
33
34 Iterator end() const
35 {
36     return Iterator(fRoot).end();
37 }
38
39 Iterator rbegin() const
40 {
41     return Iterator(fRoot).rbegin();
42 }
43
44 Iterator rend() const
45 {
46     return Iterator(fRoot).rend();
47 }
48
```



```

1 // Problem 2
2
3 void push_back(const I& aElement)
4 {
5     if (empty())
6     {
7         fRoot = new Node(aElement);
8     }
9     else
10    {
11        Node* lastNode = const_cast<Node*>(&fRoot->getPrevious());
12        lastNode->push_back(*new Node(aElement));
13    }
14    ++fCount;
15 }

```



```

1 // Problem 3
2 const I& operator[](size_t aIndex) const
3 {
4     if (aIndex > size() - 1)
5     {
6         throw std::out_of_range("Index out of bounds");
7     }
8     Iterator lIterator = Iterator(fRoot).begin();
9     for (size_t i = 0; i < aIndex; i++)
10    {
11        ++lIterator;
12    }
13    return *lIterator;
14 }

```

```

1 // Problem 4
2 List(const List& aOtherList) : fRoot(nullptr), fCount(0)
3 {
4     *this = aOtherList;
5 }
6
7 List& operator=(const List& aOtherList)
8 {
9     if (&aOtherList != this)
10    {
11        this->~List();
12        if (aOtherList.fRoot == nullptr)
13        {
14            fRoot = nullptr;
15        }
16        else
17        {
18            fRoot = nullptr;
19            fCount = 0;
20            for (auto& payload : aOtherList)
21            {
22                push_back(payload);
23            }
24        }
25    }
26    return *this;
27 }
28
29 // Problem 5
30 List(List&& aOtherList) : fRoot(nullptr), fCount(0)
31 {
32     *this = std::move(aOtherList);
33 }
34 List& operator=(List&& aOtherList)
35 {
36     if (&aOtherList != this)
37     {
38         this->~List();
39         if (aOtherList.fRoot == nullptr)
40         {
41             fRoot = nullptr;
42         }
43         else
44         {
45             fRoot = aOtherList.fRoot;
46             fCount = aOtherList.fCount;
47             aOtherList.fRoot = nullptr;
48             aOtherList.fCount = 0;
49         }
50     }
51     return *this;
52 }
53
54 void push_front(I&& aElement)
55 {
56     if (empty())
57     {
58         fRoot = new Node(std::move(aElement));
59     }
60     else
61     {
62         Node* lNode = new Node(std::move(aElement));
63         fRoot->push_front(*lNode);
64         fRoot = lNode;
65     }
66     ++fCount;
67 }
68
69 void push_back(I&& aElement)
70 {
71     if (empty())
72     {
73         fRoot = new Node(aElement);
74     }
75     else
76     {
77         Node* lastNode = const_cast<Node*>(&fRoot->getPrevious());
78         lastNode->push_back(*new Node(aElement));
79     }
80     ++fCount;
81 }
82 };

```

