Swinburne University of Technology

School of Science, Computing and Engineering Technologies

FINAL EXAM COVER SHEET

Subject Code: COS30008
Subject Title: Data Structures & Patterns
Due date: June 14, 2024, 12:00 AEST
Lecturer: Dr. Markus Lumpe

Your name: Nur E Siam **Your student id:** 103842784

Marker's comments:

Problem	Marks	Obtained
1	34	
2	130	
3	114	
4	68	
Total	346	

This test requires approx. 2 hours and accounts for 50% of your overall mark.

```
 1
 2  // COS30008, Final Exam, 2024
 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;
16      Node fTail;
17      size_t fSize;
18
19  public:
20
21      using Iterator = DoublyLinkedListIterator<T>;
22
23      List() noexcept :
24          fSize(0)
25      {}
26
27      // Problem 1
28      ~List() noexcept
29      {
30          Node lCurrent = fTail;
31          fTail.reset();
32
33          while (lCurrent)
34          {
35              Node lPrevious = lCurrent->fPrevious.lock();
36              lCurrent->fPrevious.reset();
37              lCurrent->fNext.reset();
38              lCurrent = lPrevious;
39          }
40          fHead.reset();
41      }
42
43
44      List( const List& aOther ) :
45          List()
46      {
47          for ( auto& item : aOther )
48          {
49              push_back( item );
```

```
50          }
51      }
52
53      List& operator=( const List& aOther )
54      {
55          if ( this != &aOther )
56          {
57              this->~List();
58
59              new (this) List( aOther );
60          }
61
62          return *this;
63      }
64
65      List( List&& aOther ) noexcept :
66          List()
67      {
68          swap( aOther );
69      }
70
71      List& operator=( List&& aOther ) noexcept
72      {
73          if ( this != &aOther )
74          {
75              swap( aOther );
76          }
77
78          return *this;
79      }
80
81      void swap( List& aOther ) noexcept
82      {
83          std::swap( fHead, aOther.fHead );
84          std::swap( fTail, aOther.fTail );
85          std::swap( fSize, aOther.fSize );
86      }
87
88      size_t size() const noexcept
89      {
90          return fSize;
91      }
92
93      template<typename U>
94      void push_front( U&& aData )
95      {
96          Node lNode = DoublyLinkedList<T>::makeNode( std::forward<U>
              (aData) );
97
```

```
 98          if ( !fHead )                                   // first element
 99          {
100              fTail = lNode;                              // set tail to first
                   element
101          }
102          else
103          {
104              lNode->fNext = fHead;                       // new node becomes
                   head
105              fHead->fPrevious = lNode;                   // new node previous
                   of head
106          }
107
108          fHead = lNode;                                  // new head
109          fSize++;                                        // increment size
110      }
111
112      template<typename U>
113      void push_back( U&& aData )
114      {
115          Node lNode = DoublyLinkedList<T>::makeNode( std::forward<U>
               (aData) );
116
117          if ( !fTail )                                   // first element
118          {
119              fHead = lNode;                              // set head to first
                   element
120          }
121          else
122          {
123              lNode->fPrevious = fTail;                   // new node becomes
                   tail
124              fTail->fNext = lNode;                       // new node next of
                   tail
125          }
126
127          fTail = lNode;                                  // new tail
128          fSize++;                                        // increment size
129      }
130
131      void remove( const T& aElement ) noexcept
132      {
133          Node lNode = fHead;                             // start at first
134
135          while ( lNode )                                 // Are there still
               nodes available?
136          {
137              if ( lNode->fData == aElement )             // Have we found the
                   node?
```

```
138             {
139                 break;                              // stop the search
140             }
141
142             lNode = lNode->fNext;                   // move to next node
143         }
144
145         if ( lNode )                                // We have found a
              first matching node.
146         {
147             if ( fHead == lNode )                   // remove head
148             {
149                 fHead = lNode->fNext;               // make lNode's next
                      head
150             }
151
152             if ( fTail == lNode )                   // remove tail
153             {
154                 fTail = lNode->fPrevious.lock();    // make lNode's
                      previuos tail, requires std::shared_ptr
155             }
156
157             lNode->isolate();                       // isolate node,
                  automatically freed
158             fSize--;                                // decrement count
159         }
160     }
161
162     const T& operator[]( size_t aIndex ) const
163     {
164         assert( aIndex < fSize );
165
166         Node lNode = fHead;
167
168         while ( aIndex-- )
169         {
170             lNode = lNode->fNext;
171         }
172
173         return lNode->fData;
174     }
175
176     Iterator begin() const noexcept
177     {
178         return Iterator( fHead, fTail );
179     }
180
181     Iterator end() const noexcept
182     {
```

```
183         return begin().end();
184     }
185
186     Iterator rbegin() const noexcept
187     {
188         return begin().rbegin();
189     }
190
191     Iterator rend() const noexcept
192     {
193         return begin().rend();
194     }
195 };
196
```

```
1  // COS30008, Final Exam, 2024
2
3  // DynamicQueue.h
4  #pragma once
5
6  #include <optional>
7  #include <cassert>
8
9  #include <iostream>
10
11 template<typename T>
12 class DynamicQueue
13 {
14 private:
15     T* fElements;
16     size_t fFirstIndex;
17     size_t fLastIndex;
18     size_t fCurrentSize;
19
20     void resize(size_t aNewSize) {
21         T* lNewElements = new T[aNewSize];
22         size_t j = 0;
23         for (size_t i = fFirstIndex; i < fLastIndex; ++i, ++j) {
24             lNewElements[j] = std::move(fElements[i]);
25         }
26         delete[] fElements;
27         fElements = lNewElements;
28         fFirstIndex = 0;
29         fLastIndex = j;
30         fCurrentSize = aNewSize;
31     }
32
33     void ensure_capacity() {
34         if (fLastIndex >= fCurrentSize) {
35             resize(fCurrentSize * 2);
36         }
37     }
38
39     void adjust_capacity() {
40         if ((fLastIndex - fFirstIndex) <= fCurrentSize / 4 && fCurrentSize
           > 1) {
41             resize(fCurrentSize / 2);
42         }
43     }
44
45
46 public:
47     DynamicQueue() : fElements(new T[1]), fFirstIndex(0), fLastIndex(0),
       fCurrentSize(1) {}
```

```
48
49      ~DynamicQueue() {
50          delete[] fElements;
51      }
52
53      DynamicQueue(const DynamicQueue&) = delete;
54      DynamicQueue& operator=(const DynamicQueue&) = delete;
55
56      std::optional<T> top() const noexcept {
57          if (fFirstIndex == fLastIndex) {
58              return std::nullopt;
59          }
60          return fElements[fFirstIndex];
61      }
62
63      void enqueue(const T& aValue) {
64          ensure_capacity();
65          fElements[fLastIndex++] = aValue;
66      }
67
68      void dequeue() {
69          if (fFirstIndex < fLastIndex) {
70              ++fFirstIndex;
71              adjust_capacity();
72          }
73      }
74
75  };
```

```cpp
// PalindromeStringIterator.cpp
#include "PalindromeStringIterator.h"

void PalindromeStringIterator::moveToNextIndex()
{
    while (fIndex < static_cast<int>(fString.length()) && !std::isalpha
      (fString[fIndex]))
    {
        ++fIndex;
    }
}

void PalindromeStringIterator::moveToPreviousIndex()
{
    while (fIndex >= 0 && !std::isalpha(fString[fIndex]))
    {
        --fIndex;
    }
}

PalindromeStringIterator::PalindromeStringIterator(const std::string&
  aString)
    : fString(aString), fIndex(0)
{
    moveToNextIndex();
}

char PalindromeStringIterator::operator*() const noexcept
{
    return std::toupper(fString[fIndex]);
}

PalindromeStringIterator& PalindromeStringIterator::operator++() noexcept
{
    ++fIndex;
    moveToNextIndex();
    return *this;
}

PalindromeStringIterator PalindromeStringIterator::operator++(int) noexcept
{
    PalindromeStringIterator old = *this;
    ++(*this);
    return old;
}

PalindromeStringIterator& PalindromeStringIterator::operator--() noexcept
{
    --fIndex;
```

```
48      moveToPreviousIndex();
49      return *this;
50  }
51
52  PalindromeStringIterator PalindromeStringIterator::operator--(int) noexcept
53  {
54      PalindromeStringIterator old = *this;
55      --(*this);
56      return old;
57  }
58
59  bool PalindromeStringIterator::operator==(const PalindromeStringIterator&
      aOther) const noexcept
60  {
61      return fIndex == aOther.fIndex;
62  }
63
64  bool PalindromeStringIterator::operator!=(const PalindromeStringIterator&
      aOther) const noexcept
65  {
66      return !(*this == aOther);
67  }
68
69  PalindromeStringIterator PalindromeStringIterator::begin() const noexcept
70  {
71      PalindromeStringIterator iter(*this);
72      iter.fIndex = 0;
73      iter.moveToNextIndex();
74      return iter;
75  }
76
77  PalindromeStringIterator PalindromeStringIterator::end() const noexcept
78  {
79      PalindromeStringIterator iter(*this);
80      iter.fIndex = fString.length();
81      return iter;
82  }
83
84  PalindromeStringIterator PalindromeStringIterator::rbegin() const noexcept
85  {
86      PalindromeStringIterator iter(*this);
87      iter.fIndex = static_cast<int>(fString.length()) - 1;
88      iter.moveToPreviousIndex();
89      return iter;
90  }
91
92  PalindromeStringIterator PalindromeStringIterator::rend() const noexcept
93  {
94      PalindromeStringIterator iter(*this);
```

```
95      iter.fIndex = -1;
96      return iter;
97  }
98
```