

```
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     void push_back( const T& item )
54     {
55         Node lNewNode = Node( item );
56
57         if ( fTail )
58         {
59             lNewNode->fPrevious = fTail;
60             fTail->fNext = lNewNode;
61             fTail = lNewNode;
62         }
63         else
64         {
65             fHead = lNewNode;
66             fTail = lNewNode;
67         }
68         fSize++;
69     }
70
71     void push_front( const T& item )
72     {
73         Node lNewNode = Node( item );
74
75         if ( fHead )
76         {
77             lNewNode->fNext = fHead;
78             fHead->fPrevious = lNewNode;
79             fHead = lNewNode;
80         }
81         else
82         {
83             fTail = lNewNode;
84             fHead = lNewNode;
85         }
86         fSize++;
87     }
88
89     void pop_back()
90     {
91         if ( !fTail )
92             return;
93
94         Node lCurrent = fTail;
95         fTail = fTail->fPrevious;
96         fTail->fNext = nullptr;
97
98         fSize--;
99     }
100
101    void pop_front()
102    {
103        if ( !fHead )
104            return;
105
106        Node lCurrent = fHead;
107        fHead = fHead->fNext;
108        fHead->fPrevious = nullptr;
109
110        fSize--;
111    }
112
113    void clear()
114    {
115        fHead.reset();
116        fTail.reset();
117        fSize = 0;
118    }
119
120    size_t size() const
121    {
122        return fSize;
123    }
124
125    Iterator begin() const
126    {
127        return Iterator( fHead );
128    }
129
130    Iterator end() const
131    {
132        return Iterator( fTail );
133    }
134
135    void swap( List &other )
136    {
137        std::swap( fHead, other.fHead );
138        std::swap( fTail, other.fTail );
139        std::swap( fSize, other.fSize );
140    }
141}
```

```
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>
97          (aData) );
```

```
98     if ( !fHead )                                // first element
99     {
100         fTail = lNode;                           // set tail to first ↵
101         element
102     }
103     else
104     {
105         lNode->fNext = fHead;                  // new node becomes ↵
106         head
107         fHead->fPrevious = lNode;             // new node previous ↵
108         of head
109     }
110 }
111
112 template<typename U>
113 void push_back( U&& aData )
114 {
115     Node lNode = DoublyLinkedList<T>::makeNode( std::forward<U>      ↵
116         (aData) );
117
118     if ( !fTail )                                // first element
119     {
120         fHead = lNode;                           // set head to first ↵
121         element
122     }
123     else
124     {
125         lNode->fPrevious = fTail;              // new node becomes ↵
126         tail
127         fTail->fNext = lNode;                // new node next of ↵
128         tail
129     }
130
131     fTail = lNode;                            // new tail
132     fSize++;                                // increment size
133 }
134
135 void remove( const T& aElement ) noexcept
136 {
137     Node lNode = fHead;                      // start at first
138
139     while ( lNode )                         // Are there still ↵
140         nodes available?
141     {
142         if ( lNode->fData == aElement )       // Have we found the ↵
143             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      ↵
146         first matching node.
147     {
148         if ( fHead == lNode )                // remove head
149         {
150             fHead = lNode->fNext;          // make lNode's next    ↵
151             head
152         }
153
154         if ( fTail == lNode )              // remove tail
155         {
156             fTail = lNode->fPrevious.lock(); // make lNode's        ↵
157             previous tail, requires std::shared_ptr
158         }
159
160         lNode->isolate();               // isolate node,
161         automatically freed
162         fSize--;                      // decrement count
163     }
164
165     const T& operator[]( size_t aIndex ) const
166     {
167         assert( aIndex < fSize );
168
169         Node lNode = fHead;
170
171         while ( aIndex-- )
172         {
173             lNode = lNode->fNext;
174         }
175
176         return lNode->fData;
177     }
178
179     Iterator begin() const noexcept
180     {
181         return Iterator( fHead, fTail );
182     }
183
184     Iterator end() const noexcept
185     {
```

```
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
```