```cpp
#pragma once

#include <stdexcept>
#include "DoublyLinkedList.h"
#include "DoublyLinkedListIterator.h"

template <typename T>
class List {
private:
    using Node = typename DoublyLinkedList<T>::Node;

    Node fHead;     // first element
    Node fTail;     // last element
    size_t fSize;   // number of elements

public:
    using Iterator = DoublyLinkedListIterator<T>;

    // default constructor
    List() noexcept : fHead(nullptr), fTail(nullptr), fSize(0) {}

    // copy constructor
    List(const List& aOther) : fHead(nullptr), fTail(nullptr), fSize(0) {
        for (const auto& item : aOther) {
            push_back(item);
        }
    }

    // copy assignment
    List& operator=(const List& aOther) {
        if (this != &aOther) {
            List temp(aOther);
            swap(temp);
        }
        return *this;
    }

    // move constructor
    List(List&& aOther) noexcept : fHead(std::move(aOther.fHead)), fTail
      (std::move(aOther.fTail)), fSize(aOther.fSize) {
        aOther.fHead = nullptr;
        aOther.fTail = nullptr;
        aOther.fSize = 0;
    }

    // move assignment
    List& operator=(List&& aOther) noexcept {
        if (this != &aOther) {
            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 };
```