1. Doubly linked list

Code:

.cpp file

roblem1

```cpp
// Sooji Kim
// CS5008 HW5
// 6 March 2024

#include <iostream>
#include <cassert>
#include "doublyLinkedList.h"

using namespace std;

// constructor
doublyLinkedList::doublyLinkedList() {
    head = nullptr;
    tail = nullptr;
    len = 0;
}

// destructor
doublyLinkedList::~doublyLinkedList() {}

// return info of first element
int doublyLinkedList::front() const {
    assert(head != nullptr);
    return head->info;
}

// return info of last element
int doublyLinkedList::back() const {
    assert(tail != nullptr);
    return tail->info;
}

// gets the length of the list
int doublyLinkedList::getLength() const {
    return len;
}

// return if node with given info exists
bool doublyLinkedList::search(int searchItem) const {
    nodeType* curr = head;
    while (curr->next != nullptr) {
        if (curr->info == searchItem) {
            return true;
        }
        curr = curr->next;
    }
    return false;
}
```

```
49
50      // adds node with given info to the end of the list
51      void doublyLinkedList::append(int addItem) {
52          nodeType* newNode;
53          newNode = new nodeType;
54          newNode->info = addItem;
55          newNode->next = nullptr;
56          newNode->prev = nullptr;
57          newNode->next = nullptr;
58          if (head == nullptr) {
59              newNode->prev = nullptr;
60              head = newNode;
61              tail = newNode;
62          }
63          else {
64              tail->next = newNode;
65              newNode->prev = tail;
66              tail = newNode;
67          }
68      }
69
70
71      // adds node with given info to the start of the list
72      void doublyLinkedList::prepend(int addItem) {
73          nodeType* newNode;
74          newNode = new nodeType;
75          newNode->info = addItem;
76          newNode->next = nullptr;
77          newNode->prev = nullptr;
78          newNode->next = nullptr;
79          if (head == nullptr) {
80              newNode->prev = nullptr;
81              head = newNode;
82              tail = newNode;
83          }
84          else {
85              newNode->next = head;
86              head->prev = newNode;
87              head = newNode;
88          }
89      }
90
```

```
90
91  // insert node with given info after the given node
92  void doublyLinkedList::insertAfter(int insertItem, nodeType* currNode) {
93      if (search(currNode->info)) {
94          nodeType* curr;
95          nodeType* trailCurr;
96          nodeType* newNode;
97          bool found;
98          newNode = new nodeType;
99          newNode->info = insertItem;
100         newNode->next = nullptr;
101         newNode->prev = nullptr;
102         // case 1: list empty
103         if (head == nullptr) {
104             head = newNode;
105             tail = newNode;
106         }
107         // case 2: insert after tail
108         else if (currNode == tail) {
109             tail->next = newNode;
110             newNode->prev = tail;
111             tail = newNode;
112         }
113         else {
114             nodeType* next = currNode->next;
115             newNode->next = next;
116             newNode->prev = currNode;
117             currNode->next = newNode;
118             next->prev = newNode;
119         }
120         len += 1;
121     }
122 }
123
```

```
124     // delete node with given info
125     void doublyLinkedList::deleteNode(nodeType* curr) {
126         if (search(curr->info)) {
127             nodeType* next = curr->next;
128             nodeType* prev = curr->prev;
129             if (next != nullptr) {
130                 next->prev = prev;
131             }
132             if (prev != nullptr) {
133                 prev->next = next;
134             }
135             if (curr == head) {
136                 head = next;
137             }
138             if (curr == tail) {
139                 tail = prev;
140             }
141             len -= 1;
142         }
143     }
144
145     // prints the linkedlist
146     void doublyLinkedList::print() const {
147         nodeType* curr = head;
148         while (curr != nullptr) {
149             cout << curr->info;
150             curr = curr->next;
151         }
152     }
153
154     // prints the linkedlist in reverse
155     void doublyLinkedList::printR() const {
156         nodeType* curr = tail;
157         while (curr->prev != nullptr) {
158             cout << curr->info;
159             curr = curr->prev;
160         }
161     }
```

.h file:

problem1

```
1   // Sooji Kim
2   // CS5008 HW5
3   // 6 March 2024
4
5   struct nodeType {
6       int info;
7       nodeType* next;
8       nodeType* prev;
9   };
10
11  class doublyLinkedList
12  {
13
14  public:
15      // constructor
16      doublyLinkedList();
17      // destructor
18      ~doublyLinkedList();
19      // return info of first element
20      int front() const;
21      // return info of last element
22      int back() const;
23      // return if node with given info exists
24      bool search(int searchItem) const;
25      // adds node with given info to the end of the list
26      void append(int addItem);
27      // adds node with given info to start of the list
28      void prepend(int addItem);
29      // insert node with given info after the given node
30      void insertAfter(int insertItem, nodeType* currNode);
31      // delete given node from the list
32      void deleteNode(nodeType* curr);
33      // prints the list
34      void print() const;
35      // prints the linkedlist in reverse
36      void printR() const;
37      // gets the length of the list
38      int getLength() const;
39
40  private:
41      nodeType* head;
42      nodeType* tail;
43      int len;
44
45  };
46
47
```

Main.cpp

```cpp
#include "doublyLinkedList.h"
#include <iostream>

using namespace std;

int main() {
    doublyLinkedList myList;
    // add an element
    for (int i = 1; i <= 5; i++) {
        myList.append(i);
    }
    // print the list
    cout << "My linked list: " << endl;
    myList.print();
    cout << endl;
    // remove an element
    nodeType* myHead = myList.front();
    myList.deleteNode(myHead);
    cout << "My linked list after removing the head: " << endl;
    myList.print();
    cout << endl;
    // seach for an elemnent
    cout << "Is 1 in my linked list? " << endl;
    cout << myList.search(1) << endl;
    cout << "Is 2 in my linked list? " << endl;
    cout << myList.search(2) << endl;
    return 0;
}
```

Output:

```
Microsoft Visual Studio Debug Console
My linked list:
12345
My linked list after removing the head:
2345
Is 1 in my linked list?
0
Is 2 in my linked list?
1
```