Student Name: 马越
Student ID: 2022141460097

C++ 面向对象程序设计

Assignment 6

声明LinkedList为Node的友元类，以便LinkedList中的函数访问以及修改Node中的私有成员的值。对链表进行操作时，注意特判链表是否为空。

Listing 1: linkedlist.h

```
#ifndef LINKEDLIST_H
#define LINKEDLIST_H

#include <iostream>

class LinkedList {
public:
  class Node {
  friend class LinkedList;
  public:
    Node();
    Node(double);

    Node *next;
    Node *previous;

    double getValue() const;
    void setValue(double);
    friend std::ostream &operator<<(std::ostream &, const Node &);

  private:
    double value;
  };

  LinkedList();
  LinkedList(const std::initializer_list<double>&);
  LinkedList(const LinkedList &);

  ~LinkedList();

  void push_back(double);
  void push_front(double);
  double pop_back();
  double pop_front();
  double back() const;
```

```
36    double front() const;
37
38    bool empty() const;
39    int getSize() const;
40
41    void clear();
42    void show() const;
43    void extend(const LinkedList &);
44
45    double &operator[](size_t);
46    const double &operator[](size_t) const;
47
48  private:
49    int N{0};
50
51  public:
52    Node *head;
53    Node *tail;
54  };
55
56  #endif //LINKEDLIST_H
```

Listing 2: linkedlist.cpp

```
1   #include "linkedlist.h"
2   #include <stdexcept>
3
4   LinkedList::Node::Node(): value(0), next(nullptr), previous(nullptr) { }
5
6   LinkedList::Node::Node(double val): value(val), next(nullptr), previous(nullptr) { }
7
8   double LinkedList::Node::getValue() const {
9     return value;
10  }
11
12  void LinkedList::Node::setValue(double val) {
13    value = val;
14  }
15
16  std::ostream &operator<<(std::ostream &os, const LinkedList::Node &node) {
17    os << node.value;
18    return os;
19  }
20
21  LinkedList::LinkedList(): head(nullptr), tail(nullptr), N(0) { }
22
23  LinkedList::LinkedList(const std::initializer_list<double> &list): head(nullptr), tail(nullptr), N(0) {
```

```
  for (auto &val: list)
    push_back(val);
}

LinkedList::LinkedList(const LinkedList &list): head(nullptr), tail(nullptr), N(0) {
  Node *cur = list.head;
  while (cur != nullptr) {
    push_back(cur->value);
    cur = cur->next;
  }
}

LinkedList::~LinkedList() {
  clear();
}

void LinkedList::push_back(double val) {
  Node *newNode = new Node(val);
  if (N == 0) {
    head = newNode;
    tail = newNode;
  } else {
    tail->next = newNode;
    newNode->previous = tail;
    tail = newNode;
  }
  ++N;
}

void LinkedList::push_front(double val) {
  Node *newNode = new Node(val);
  if (N == 0) {
    head = newNode;
    tail = newNode;
  } else {
    head->previous = newNode;
    newNode->next = head;
    head = newNode;
  }
  ++N;
}

double LinkedList::pop_back() {
  if (N == 0) {
    throw std::logic_error("pop_back(): empty list");
  }
```

```
70    double val = tail->value;
71    Node *newTail = tail->previous;
72    delete tail;
73    tail = newTail;
74    if (tail == nullptr) {
75      head = nullptr;
76    } else {
77      tail->next = nullptr;
78    }
79    --N;
80    return val;
81  }
82
83  double LinkedList::pop_front() {
84    if (N == 0) {
85      throw std::logic_error("pop_front(): empty list");
86    }
87    double val = head->value;
88    Node *newHead = head->next;
89    delete head;
90    head = newHead;
91    if (head == nullptr) {
92      tail = nullptr;
93    } else {
94      head->previous = nullptr;
95    }
96    --N;
97    return val;
98  }
99
100 double LinkedList::back() const {
101   if (N == 0) {
102     throw std::logic_error("back(): empty list");
103   }
104   return tail->value;
105 }
106
107 double LinkedList::front() const {
108   if (N == 0) {
109     throw std::logic_error("front(): empty list");
110   }
111   return head->value;
112 }
113
114 bool LinkedList::empty() const {
115   return N == 0;
```

```
}

int LinkedList::getSize() const {
  return N;
}

void LinkedList::clear() {
  while (head != nullptr) {
    Node *next = head->next;
    delete head;
    head = next;
  }
  head = tail = nullptr;
  N = 0;
}

void LinkedList::show() const {
  Node *cur = head;
  while (cur != nullptr) {
    std::cout << cur->value << ", ";
    cur = cur->next;
  }
  std::cout << std::endl;
}

void LinkedList::extend(const LinkedList &list) {
  Node *cur = list.head;
  while (cur != nullptr) {
    push_back(cur->value);
    cur = cur->next;
  }
}

double &LinkedList::operator[](size_t n) {
  if (n >= N) {
    throw std::logic_error("index out of range");
  }
  Node *cur = head;
  for (int i = 0; i < n; i++) {
    cur = cur->next;
  }
  return cur->value;
}

const double &LinkedList::operator[](size_t n) const {
  if (n >= N) {
```

```
    throw std::logic_error("index out of range");
  }
  Node *cur = head;
  for (int i = 0; i < n; i++) {
    cur = cur->next;
  }
  return cur->value;
}
```