Student Name: 戈天齐
Student ID: 2022141460202

C++

**Assignment 6**

# linkedlist.h

```
#include <iostream>

class LinkedList {

public:
    class Node;

    friend std::ostream &operator<<(std::ostream &out, LinkedList::Node &node);

    LinkedList();

    LinkedList(const LinkedList &);

    LinkedList(std::initializer_list<double> list);

    ~LinkedList();

    void push_back(double);

    void push_front(double);

    double pop_back();

    double pop_front();

    double back();

    double front();

    bool empty();

    void clear();

    void show();

```

```
36      int getSize();
37
38      void extend(const LinkedList &);
39
40
41
42      double &operator[](int idx);
43
44      class Node {
45          friend double &LinkedList::operator[](int idx);
46      public:
47          Node();
48
49          explicit Node(double value);
50
51          ~Node();
52
53          double getValue();
54
55          void setValue(double);
56
57      private:
58          double value;
59      };
60
61  public:
62      Node *head;
63      Node *tail;
64  private:
65      int N{0};
66  };
```

## linkedlist.cpp

在重载 [] 操作符时，存在一个问题，我们要在外部去访问并修改 class Node 的一个 private 变量 value; 我的方法时，将 double & LinkedList::operator[](int i) 声明位 class Node 的友元函数，这样就实现访问和修改了。当然，除此之外还有很多方法，例如，在 class Node 里面声明一个 double & resetvalue()。又或者直接将 class LinkedList 声明为 class Node 的一个友元。

同时，在删除链表的时候，由于我们创立的是一个双向链表，我们可以采用头和尾处同时删除方法，也可以采用单向链表的删除方法

```
1   #include "../h/linkedlist.h"
2
3   LinkedList::Node::Node() :value(0),next(nullptr), previous(nullptr){}
```

```
4
5  LinkedList::Node::Node(double _value) : value(_value), next(nullptr), previous(nullptr){}
6
7  LinkedList::Node::~Node() = default;
8
9  double LinkedList::Node::getValue() {
10     return this->value;
11 }
12
13 void LinkedList::Node::setValue(double _value) {
14     this->value = _value;
15 }
16
17 LinkedList::LinkedList(): tail(nullptr),head(nullptr){};
18
19 LinkedList::LinkedList(const LinkedList & list){
20     this->head = nullptr;
21     this->tail = nullptr;
22     Node *p = list.head;
23      while(p != nullptr) {
24         push_back(p->getValue());
25         p = p->next;
26     }
27 }
28
29 LinkedList::LinkedList(std::initializer_list<double> list) {
30
31     this->head = nullptr;
32     this->tail = nullptr;
33     for(auto i:list){
34         this->push_back(i);
35     }
36 }
37
38 LinkedList::~LinkedList() = default;
39
40 void LinkedList::push_back(double _value) {
41     Node *temp = new Node(_value);
42     if (this->head == nullptr){
43         this->head = temp;
44         this->tail = temp;
45     }
46     else {
47         tail->next = temp;
48         temp->previous = tail;
49         temp->next = nullptr;
```

```
50          tail = temp;
51      }
52      N++;
53  }
54
55  void LinkedList::push_front(double _value) {
56      Node *temp = new Node(_value);
57      if(this->head == nullptr){
58          this->head = temp;
59          this->tail = temp;
60      }
61      else {
62          head->previous = temp;
63          temp->next = this->head;
64          temp->previous = nullptr;
65          this->head = temp;
66
67      }
68      N++;
69  }
70
71  double LinkedList::pop_back() {
72      if(this->tail == nullptr)
73      {
74          throw std::logic_error("Null");
75      }
76      else {
77          double _value = tail->getValue();
78          tail = tail->previous;
79          delete tail->next;
80          tail->next = nullptr;
81          N--;
82          return _value;
83      }
84  }
85
86  double LinkedList::pop_front() {
87      if(this->tail == nullptr){
88          throw std::logic_error("Null");
89      }
90      else {
91          double _value = tail->getValue();
92          head = head->next;
93          delete head->previous;
94          head->previous = nullptr;
95          N--;
```

```
        return _value;
    }
}

double LinkedList::back() {
    if(this->tail == nullptr){
        throw std::logic_error("Null");
    }
    else {
        return tail->getValue();
    }
}

double LinkedList::front() {
    if(this->head == nullptr){
        throw std::logic_error("Null");
    }
    else {
        return head->getValue();
    }
}

bool LinkedList::empty() {
    return N == 0;
}

void LinkedList::clear() {
    if(!this->empty()){
        Node *p = head;
        Node *temp;
        while (p != nullptr) {
            temp = p;
            p = p->next;
            delete temp;
        }
        N = 0;
    }
}

void LinkedList::show() {
    Node *p = head;
    while (p != nullptr) {
        std::cout << p->getValue() << " ";
        p = p->next;
    }
    std::cout << std::endl;
```

```
142 }
143
144 int LinkedList::getSize() {
145     return N;
146 }
147
148 void LinkedList::extend(const LinkedList &_extend) {
149     if (tail != nullptr) { tail->next = _extend.head; }
150     else {
151         head = _extend.head;
152         tail = _extend.tail;
153         this->N = _extend.N;
154         return;
155     }
156     if (_extend.head != nullptr) { _extend.head->previous = tail; }
157     else {
158         return;
159     }
160     tail = _extend.tail;
161     this->N += _extend.N;
162 }
163
164 std::ostream &operator<<(std::ostream &out, LinkedList::Node &node) {
165     out << node.getValue();
166     return out;
167 }
168
169
170 double & LinkedList::operator[](int i) {
171     if(i >= N || i < 0){
172         throw std::logic_error("beyond");
173     }
174     else
175     {
176         Node *p = this->head;
177         while (i-- && p != nullptr) {
178             p = p->next;
179         }
180         return p->value;
181     }
182 }
```