学号：2022141410193　**姓名**：王枭越 **学院**：计算机学院

Assignment6

代码

`linkedlist.h`

```
#include<iostream>
#include <initializer_list>
class LinkedList {
public:
    class Node {
    friend class LinkedList;
    public:
        Node();
        Node(double );
        Node *next;
        Node *previous;
        double getValue();
        void setValue(double);
    private:
        double value;
    };
    LinkedList();
    LinkedList(const LinkedList &);
    LinkedList(std::initializer_list<double>);
    ~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();
    int getSize();
    void extend(const LinkedList &);
    double& operator[](int);
private:
    int N{0};
public:
    Node *head;
    Node *tail;
};

std::ostream& operator<<(std::ostream& out,LinkedList::Node x);
```

`linkedlist.cpp`

```cpp
#include "../h/linkedlist.h"
#include <iostream>
#include <iomanip>

//no-argument constuctor
LinkedList::Node::Node():next(nullptr),previous(nullptr),value(0){}

//set private data member
void LinkedList::Node::setValue(double val){
    value=val;
}

//obtain private data member
double LinkedList::Node::getValue(){
    return value;
}

//constructor
LinkedList::Node::Node(double val){
    next=new Node();
    previous=new Node();
    setValue(val);
}

void LinkedList::push_back(double val){
    Node *now=new Node(val);
    if(head==tail){
        //maintain the head's value being the same with the first node
        head->value=val;
    }
    //link with the tail
    now->next=tail->next;
    now->previous=tail;
    tail->next=now;
    tail=now;
    ++N;
}

void LinkedList::push_front(double val){
    Node *now=new Node(val);
    Node *tmp=head->next;
    //maintain the head's value being the same with the first node
    head->value=val;
    if(tmp==nullptr){
        //special process
        now->previous=head;
        head->next=now;
        tmp=tail=now;
        ++N;
        return;
    }
    //link with head and head->next
```

```
    now->previous=tmp->previous;
    now->next=tmp;
    tmp->previous=now;
    ++N;
}

double LinkedList::back(){
    if(tail==head){
        //logic error
        throw std::logic_error("Out of range!");
    }
    return tail->getValue();
}

double LinkedList::front(){
    if(tail==head){
        //logic error
        throw std::logic_error("Out of range!");
    }
    return head->next->getValue();
}

double LinkedList::pop_back(){
    double ret=back();
    //copy
    auto tail_=tail;
    auto now=tail->previous;
    if(now!=nullptr){
        now->next=nullptr;
        tail=now;
    }
    delete tail_;
    --N;
    return ret;
}

double LinkedList::pop_front(){
    double ret=front();
    auto head_=head->next;
    if(head_!=nullptr){
        if(head_->next==nullptr){
            tail=head;
            delete head_;
            return ret;
        }
        head_->next->previous=head;
        head->next=head_->next;
        //maintain the head's value being the same with the first node
        head->value=head->next->value;
    }
    delete head_;
    --N;
```

```
    return ret;
}

bool LinkedList::empty(){
    return !N;
}

void LinkedList::clear(){
    Node *tmp=nullptr;
    for(Node *now=head->next;now!=tail->next;now=now->next){
        //traverse delete
        if(tmp!=nullptr)
        delete tmp;
        tmp=now;
    }
    delete tmp;
    //maintain the size
    N=0;
    tail=head;
}

void LinkedList::show(){
    std::cout<<'[';
    for(Node *now=head->next;now!=tail;now=now->next){
        if(now==nullptr){
            //special process
            std::cout<<" ]"<<std::endl;
            return;
        }
        std::cout<<now->getValue()<<", ";
    }
    std::cout<<tail->getValue()<<']'<<std::endl;
}

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

void LinkedList::extend(const LinkedList & add){
    //copy
    LinkedList tmp(add);
    if(tmp.head==tmp.tail){
        return;
    }
    //append
    tail->next=tmp.head->next;
    tmp.head->next->previous=tail;
    tail=tmp.tail;
    //delete add's head
    delete tmp.head;
    //maintain the size
    N+=tmp.N;
```

```
}

//constructor
LinkedList::LinkedList(){
    head=new Node();
    tail=head;
}

//copy constructor
LinkedList::LinkedList(const LinkedList & tmp){
    N=tmp.N;
    head=new Node();
    tail=head;
    for(Node *now=tmp.head->next;now!=tail->next;now=now->next){
        push_back(now->getValue());
    }
}

//initializer_list constructor
LinkedList::LinkedList(std::initializer_list<double> tmp){
    N=0;
    head=new Node();
    tail=head;
    for(auto it:tmp){
        push_back(it);
    }
}

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

//overload []
double& LinkedList::operator[](int x){
    int cnt=0;
    for(Node* now=head->next;now!=tail->next;now=now->next,++cnt){
        if(cnt==x){
            return now->value;
        }
    }
    throw std::logic_error("Out of range");
}

//overload <<
std::ostream& operator<<(std::ostream& out,LinkedList::Node x){
    out<<x.getValue();
    return out;
}
```

解释

这里是对部分代码的具体解释，其他的看注释解释

1. 在我的代码中，`head`节点类似于`0`节点，是一个恒定的节点，但是在测试中要检测`head`节点的值，我只需要维持其为第一节点的值就好了
2. 对于考察的点，我新增了一些构造函数，例如`LinkedList`中有个`initailizer_list`初始化的构造函数，用于处理类似`LinkedList list{114,514,1919,810};`这种情况
3. 对于考察的点，我新增了重载`[]`和`<<`
4. 除此之外，关于`<<`的重载可以放在`Node`里声明，但是得作为友元函数加上关键字`friend`，这样子这个函数就不再是成员函数而也可以访问这个类的成员

其他的具体处理注释写的很详尽了，不赘述