暨南大学本科实验报告专用纸(附页)

基于双向链表的`linkedList`

课程名称＿＿＿数据结构＿＿＿成绩评定＿＿＿＿
实验项目名称＿基于双向链表的`linkedList`＿指导老师＿＿干晓聪＿＿
实验项目编号＿01＿实验项目类型＿设计性＿实验地点＿数学系机房＿
学生姓名＿＿郭彦培＿＿学号＿＿2022101149＿＿
学院＿信息科学技术学院＿系＿数学系＿专业＿信息管理与信息系统＿
实验时间＿2024 年 6 月 13 日上午＿~＿2024 年 7 月 13 日中午＿

1. 实验目的

实现一个双向列表类，在类中实现增、删、改、查的方法并完成测试

2. 实验环境

计算机：PC X64

操作系统：Windows + Ubuntu20.0LTS

编程语言：C++：GCC std20

IDE：Visual Studio Code

3. 程序代码

3.1. `linkedList.h`

```
// #define _PRIVATE_DEBUG
#ifndef LINKED_LIST_HPP
#define LINKED_LIST_HPP

#ifdef _PRIVATE_DEBUG
#include <iostream>
#endif

namespace myDS
{
    template<typename VALUE_TYPE>
    class linkedList{
    protected:
        class linkedNode {
        public:
            VALUE_TYPE data = VALUE_TYPE();
            linkedNode * next = nullptr;
            linkedNode * priv = nullptr;

            linkedNode() { }

            linkedNode(VALUE_TYPE _data){
                next = nullptr;
                priv = nullptr;
                data = _data;
            }

            linkedNode(VALUE_TYPE _data,linkedNode * priv)
            {
                next = nullptr;
                priv = priv;
                data = _data;
            }

            ~linkedNode() {
    #ifdef _PRIVATE_DEBUG
            // if(this->next != nullptr)
            //     std::cout << "Unexpected Delete at :" << this->data
            //             << " with next:" << this->next->data << "\n";
    #endif
            }

            linkedNode *  linkNext(linkedNode * _next)
            {
                next = _next;
                _next->priv = this;
                return this->next;
```

```cpp
            }
            linkedNode *  linkPriv(linkedNode * _priv)
            {
                priv = _priv;
                _priv->next = this;
                return this->priv;
            }

            void insertNext(linkedNode * _inst){
                if(_inst == nullptr) return;
                if(this->next == nullptr) linkNext(_inst);
                else {
                    _inst->next = this->next;
                    this->next->priv = _inst;
                    _inst->priv = this;
                    this->next = _inst;
                }
            }

            void deleteNext()
            {
                if(this->next == nullptr) return;
                else {
                    linkedNode * tmp = this->next;
                    this->next = this->next->next;
                    this->next->priv = this;
                    tmp->next = nullptr;
                    delete tmp;
                }
            }
        };

    private:
        class _iterator
        {
        private:
            linkedNode *_ptr;

        public:
            enum __iter_dest_type
            {
                front,
                back
            };
            __iter_dest_type _iter_dest;

            _iterator(linkedNode * _upper ,__iter_dest_type _d)
            {
                _ptr = _upper;
                _iter_dest = _d;
            }

            VALUE_TYPE & operator*()
            {
                return _ptr->data;
            }

            VALUE_TYPE *operator->()
            {
                return _ptr;
            }

             myDS::linkedList<VALUE_TYPE>::_iterator operator++()
            {
                if (_iter_dest == front)
                    _ptr = _ptr->next;
                else
                    _ptr = _ptr->priv;
                return *this;
            }

             myDS::linkedList<VALUE_TYPE>::_iterator operator++(int)
            {
                 myDS::linkedList<VALUE_TYPE>::_iterator old = *this;
                if (_iter_dest == front)
                    _ptr = _ptr->next;
                else
                    _ptr = _ptr->priv;
                return old;
            }

            //  myDS::linkedList<VALUE_TYPE>::_iterator operator+(size_t _n)
            // {
            //     if (_iter_dest == front)
            //         _upper_idx += _n;
            //     else
            //         _upper_idx -= _n;
            //     _ptr = &((*_upper_pointer)[_upper_idx]);
            //     return *this;
            // }

            bool operator==( myDS::linkedList<VALUE_TYPE>::_iterator _b)
            {
                if (&(*_b) == _ptr)
                    return 1;
                else
                    return 0;
            }

            bool operator!=( myDS::linkedList<VALUE_TYPE>::_iterator _b)
            {
                if (&(*_b) == &(_ptr->data))
```

```
                    return 0;
                else
                    return 1;
            }
        };

        linkedNode * head = new linkedNode();
        linkedNode * tail = new linkedNode();
        int cap = 0;

    public:
        linkedList(){
            head->linkNext(tail);
        }

        ~linkedList(){
            clear();
            delete head;
            delete tail;
        }

        void push_back(VALUE_TYPE t) {
            tail->data = t;
            tail->linkNext(new linkedNode());
            tail = tail->next;
            cap ++;
        }

        void push_frount(VALUE_TYPE t) {
            head->data = t;
            head = (head->linkPriv(new linkedNode()));
            cap ++;
        }

        void clear() {
            linkedNode * deletingObject;
            while(tail->priv != head) {
                deletingObject = tail;
                tail = tail->priv;
                delete deletingObject;
            }
            cap = 0;
            delete head;
            delete tail;
            tail = new linkedNode();
            head = new linkedNode();
            head->linkNext(tail);
        }

        std::size_t erase(VALUE_TYPE p) {
            linkedNode * ptr = head;
            int ttl = 0;
            while(ptr->next != tail) {
                if(ptr->next->data == p){
                    ptr->deleteNext();
                    ttl ++;
                } else ptr = ptr->next;
            }
            cap -= ttl;
            return ttl;
        }

        std::size_t size() {return cap;}

        bool erase(linkedList<VALUE_TYPE>::_iterator p) {
             myDS::linkedList<VALUE_TYPE>::_iterator ptr = this->begin();
            linkedNode * cur = head;
            while(ptr != p) {
                cur = cur->next;
                ptr ++;
                if(cur == tail) return 0;
            }
            cur->deleteNext();
            cap --;
            return 1;
        }

         myDS::linkedList<VALUE_TYPE>::_iterator begin() {
            enum myDS::linkedList<VALUE_TYPE>::_iterator::__iter_dest_type _FRONT = myDS::linkedList<VALUE_TYPE>::_iterator::__iter_dest_type::front;
            return  myDS::linkedList<VALUE_TYPE>::_iterator(head->next,_FRONT);
        }

         myDS::linkedList<VALUE_TYPE>::_iterator rbegin() {
            enum myDS::linkedList<VALUE_TYPE>::_iterator::__iter_dest_type _BACK = myDS::linkedList<VALUE_TYPE>::_iterator::__iter_dest_type::back;
            return  myDS::linkedList<VALUE_TYPE>::_iterator(tail->priv,_BACK);
        }

         myDS::linkedList<VALUE_TYPE>::_iterator end() {
            enum myDS::linkedList<VALUE_TYPE>::_iterator::__iter_dest_type _FRONT = myDS::linkedList<VALUE_TYPE>::_iterator::__iter_dest_type::front;
            return myDS::linkedList<VALUE_TYPE>::_iterator(tail,_FRONT);
        }

```

```cpp
         myDS::linkedList<VALUE_TYPE>::_iterator rend() {
            enum myDS::linkedList<VALUE_TYPE>::_iterator::__iter_dest_type _BACK = myDS::linkedList<VALUE_TYPE>::_iterator::__iter_dest_type::back;
            return  myDS::linkedList<VALUE_TYPE>::_iterator(head,_BACK);
        }

         myDS::linkedList<VALUE_TYPE>::_iterator get(std::size_t p) {
            linkedNode * ptr = head->next;
            while(p --) ptr = ptr->next;
            enum myDS::linkedList<VALUE_TYPE>::_iterator::__iter_dest_type _FRONT = myDS::linkedList<VALUE_TYPE>::_iterator::__iter_dest_type::front;
            return  myDS::linkedList<VALUE_TYPE>::_iterator(ptr,_FRONT);
        }

        VALUE_TYPE & operator[](std::size_t p) {
            linkedNode * ptr = head;
            while(p --) ptr = ptr->next;
            return ptr->next->data;
        }

#ifdef _PRIVATE_DEBUG
        void innerPrint()
        {
            std::cout << "--Header[" << head << "]: " << head->data << "\n";
            std::cout << "--Tail[" << tail << "]: " << tail->data << "\n";
            std::cout << "-----------\n";
            std::cout << "cur:" << cap<< "\n";
            auto ptr = head;
            do {
                std::cout << "[" << ptr << "] ->next:" << ptr->next << " ->priv:" << ptr->priv << " ||data:" << ptr->data << "\n";
                ptr = ptr->next;
            }while(ptr != nullptr);
        }
#endif

    };
}
#endif
```

3.2. `_PRIV_TEST.cpp`

```cpp
#define DS_TOBE_TEST linkedList

#define _PRIVATE_DEBUG
```

```cpp

#include "Dev\01\linkedList.h"

#include <iostream>
#include <math.h>
#include <vector>

using namespace std;

using TBT =  myDS::DS_TOBE_TEST<int>;

void accuracyTest() {//结构正确性测试

    TBT tc = TBT();
    for(;;)
    {
        string op;
        cin >> op;
        if(op == "clr") { //清空
            tc.clear();
        } else if(op == "q") //退出测试
        {
            return;
        } else if(op == "pb")//push_back
        {
            int c;
            cin >> c;
            tc.push_back(c);
        } else if(op == "pf")//push_frount
        {
            int c;
            cin >> c;
            tc.push_frount(c);
        } else if(op == "at")//随机访问
        {
            int p;
            cin >> p;
            cout << tc[p] << "\n";
        } else if(op == "delEL")//删除所有等于某值元素
        {
            int p;
            cin >> p;
            cout << tc.erase(p) << "\n";
        } else if(op == "delPS")//删除某位置上的元素
        {
            int p;
            cin >> p;
            cout << tc.erase(tc.get(p)) << "\n";
        } else if(op == "iterF") //正序遍历
        {
            tc.innerPrint();
            cout << "Iter with index:\n";
            for(int i = 0;i < tc.size();i ++) cout << tc[i] << " ";cout << "\n";
            cout << "Iter with begin end\n";
            for(auto x = tc.begin();x != tc.end();x ++) cout << (*x) << " ";cout << "\n";
            cout << "Iter with AUTO&&\n";
            for(auto x:tc) cout << x << " ";cout << "\n";
        } else if(op == "iterB") //正序遍历
        {
            tc.innerPrint();
            cout << "Iter with index:\n";
            for(int i = 0;i < tc.size();i ++) cout << tc[tc.size()-1-i] << " ";cout << "\n";
            cout << "Iter with begin end\n";
            for(auto x = tc.rbegin();x != tc.rend();x ++) cout << (*x) << " ";cout << "\n";
            // cout << "Iter with AUTO&&\n";."\n";
        } else if(op == "mv")//单点修改
        {
            int p;
            cin >> p;
            int tr;
            cin >> tr;
            tc[p] = tr;
        } else if(op == "")
        {

        } else {
            op.clear();
        }
    }
}




void memLeakTest() {//内存泄漏测试
    TBT tc = TBT();
    for(;;){
        tc.push_back(1);
        tc.push_back(1);
        tc.push_back(1);
        tc.push_back(1);
        tc.clear();
    }
}

signed main()
{
    // accuracyTest();
    memLeakTest();
}
```

4. 测试数据与运行结果

运行上述`_PRIV_TEST.cpp`测试代码中的正确性测试模块，得到以下内容:

```
  pb 1
  pb 2
  pb 3
  pb 4
  pf 3
  pb 3
  iterF
  iterB
  delEL 3
  iterF
  delPS 1
  clr
  pb 1
  pb 2
  iterF
  delPS 0
  delEL 2
  iterF


  pb 1
  pb 2
  pb 3
  pb 4
  pf 3
  pb 3
  iterF
--Header[0x662720]: 0
--Tail[0x662770]: 0
-----------
cur:6
[0x662720] ->next:0x662540 ->priv:0 ||data:0
[0x662540] ->next:0x662590 ->priv:0x662720 ||data:3
[0x662590] ->next:0x6625e0 ->priv:0x662540 ||data:1
[0x6625e0] ->next:0x662630 ->priv:0x662590 ||data:2
[0x662630] ->next:0x662680 ->priv:0x6625e0 ||data:3
```

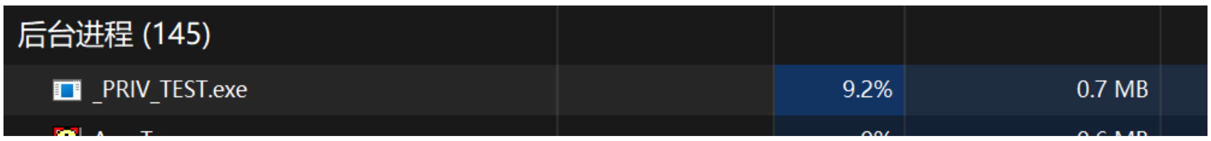
```
[0x662680] ->next:0x6626d0 ->priv:0x662630 ||data:4
[0x6626d0] ->next:0x662770 ->priv:0x662680 ||data:3
[0x662770] ->next:0 ->priv:0x6626d0 ||data:0
Iter with index:
3 1 2 3 4 3
Iter with begin end
3 1 2 3 4 3
Iter with AUTO&&
3 1 2 3 4 3
  iterB
--Header[0x662720]: 0
--Tail[0x662770]: 0
-----------
cur:6
[0x662720] ->next:0x662540 ->priv:0 ||data:0
[0x662540] ->next:0x662590 ->priv:0x662720 ||data:3
[0x662590] ->next:0x6625e0 ->priv:0x662540 ||data:1
[0x6625e0] ->next:0x662630 ->priv:0x662590 ||data:2
[0x662630] ->next:0x662680 ->priv:0x6625e0 ||data:3
[0x662680] ->next:0x6626d0 ->priv:0x662630 ||data:4
[0x6626d0] ->next:0x662770 ->priv:0x662680 ||data:3
[0x662770] ->next:0 ->priv:0x6626d0 ||data:0
Iter with index:
3 4 3 2 1 3
Iter with begin end
3 4 3 2 1 3
  delEL 3
3
  iterF
--Header[0x662720]: 0
--Tail[0x662770]: 0
-----------
cur:3
[0x662720] ->next:0x662590 ->priv:0 ||data:0
[0x662590] ->next:0x6625e0 ->priv:0x662720 ||data:1
[0x6625e0] ->next:0x662680 ->priv:0x662590 ||data:2
[0x662680] ->next:0x662770 ->priv:0x6625e0 ||data:4
[0x662770] ->next:0 ->priv:0x662680 ||data:0
Iter with index:
1 2 4
Iter with begin end
1 2 4
Iter with AUTO&&
1 2 4
```

```
  delPS 1
1
  clr
Unexpected Delete at :4 with next:16187728
  pb 1
  pb 2
  iterF
--Header[0x6625e0]: 0
--Tail[0x662680]: 0
-----------
cur:2
[0x6625e0] ->next:0x662540 ->priv:0 ||data:0
[0x662540] ->next:0x662630 ->priv:0x6625e0 ||data:1
[0x662630] ->next:0x662680 ->priv:0x662540 ||data:2
[0x662680] ->next:0 ->priv:0x662630 ||data:0
Iter with index:
1 2
Iter with begin end
1 2
Iter with AUTO&&
1 2
  delPS 0
1
  delEL 2
1
  iterF
--Header[0x6625e0]: 0
--Tail[0x662680]: 0
-----------
cur:0
[0x6625e0] ->next:0x662680 ->priv:0 ||data:0
[0x662680] ->next:0 ->priv:0x6625e0 ||data:0
Iter with index:

Iter with begin end

Iter with AUTO&&
```

可以看出，代码运行结果与预期相符，可以认为代码正确性无误。

运行`_PRIV_TEST.cpp`中的内存测试模块，在保持 CPU 高占用率运行一段时间后内存变化符合预期，可以认为代码内存安全性良好。

暨南大学本科实验报告专用纸(附页)



后台进程 (145)

_PRIV_TEST.exe 9.2% 0.7 MB