

# 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

```
1 // #define _PRIVATE_DEBUG
2 #ifndef LINKED_LIST_HPP
3 #define LINKED_LIST_HPP
4
5 #ifdef _PRIVATE_DEBUG
6 #include <iostream>
7 #endif
8
9 namespace myDS
10 {
11     template<typename VALUE_TYPE>
12     class linkedList{
13     protected:
14         class linkedNode {
15         public:
16             VALUE_TYPE data = VALUE_TYPE();
17             linkedNode * next = nullptr;
18             linkedNode * priv = nullptr;
19
20             linkedNode() { }
21
22             linkedNode(VALUE_TYPE _data){
23                 next = nullptr;
24                 priv = nullptr;
25                 data = _data;
26             }
27
28             linkedNode(VALUE_TYPE _data,linkedNode * priv)
29             {
30                 next = nullptr;
31                 priv = priv;
32                 data = _data;
33             }
34
35             ~linkedNode() {
36 #ifdef _PRIVATE_DEBUG
37                 // if(this->next != nullptr)
38                 //     std::cout << "Unexpected Delete at :" << this->data
39                 //     << " with next:" << this->next->data << "\n";
40 #endif
41             }
42
43             linkedNode * linkNext(linkedNode * _next)
44             {
45                 next = _next;
46                 _next->priv = this;
47                 return this->next;
48             }
49         };
50     };
51 }
```

# 暨南大学本科实验报告专用纸(附页)

```
48     }
49     linkedNode * linkPriv(linkedNode * _priv)
50     {
51         priv = _priv;
52         _priv->next = this;
53         return this->priv;
54     }
55
56     void insertNext(linkedNode * _inst){
57         if(_inst == nullptr) return;
58         if(this->next == nullptr) linkNext(_inst);
59         else {
60             _inst->next = this->next;
61             this->next->priv = _inst;
62             _inst->priv = this;
63             this->next = _inst;
64         }
65     }
66
67     void deleteNext()
68     {
69         if(this->next == nullptr) return;
70         else {
71             linkedNode * tmp = this->next;
72             this->next = this->next->next;
73             this->next->priv = this;
74             tmp->next = nullptr;
75             delete tmp;
76         }
77     }
78 };
79
80 private:
81     class _iterator
82     {
83     private:
84         linkedNode * _ptr;
85
86     public:
87         enum __iter_dest_type
88         {
89             front,
90             back
91         };
92         __iter_dest_type _iter_dest;
93
94         _iterator(linkedNode * _upper , __iter_dest_type _d)
95         {
96             _ptr = _upper;
97             _iter_dest = _d;
98         }
```

# 暨南大学本科实验报告专用纸(附页)

```
99
100     VALUE_TYPE & operator*()
101     {
102         return _ptr->data;
103     }
104
105     VALUE_TYPE *operator->()
106     {
107         return _ptr;
108     }
109
110     myDS::linkedList<VALUE_TYPE>::_iterator operator++()
111     {
112         if (_iter_dest == front)
113             _ptr = _ptr->next;
114         else
115             _ptr = _ptr->priv;
116         return *this;
117     }
118
119     myDS::linkedList<VALUE_TYPE>::_iterator operator++(int)
120     {
121         myDS::linkedList<VALUE_TYPE>::_iterator old = *this;
122         if (_iter_dest == front)
123             _ptr = _ptr->next;
124         else
125             _ptr = _ptr->priv;
126         return old;
127     }
128
129     // myDS::linkedList<VALUE_TYPE>::_iterator operator+(size_t
130     _n)
131     // {
132     //     if (_iter_dest == front)
133     //         _upper_idx += _n;
134     //     else
135     //         _upper_idx -= _n;
136     //     _ptr = &((*_upper_pointer)[_upper_idx]);
137     //     return *this;
138     // }
139
140     bool operator==( myDS::linkedList<VALUE_TYPE>::_iterator _b)
141     {
142         if (&(*_b) == _ptr)
143             return 1;
144         else
145             return 0;
146     }
147
148     bool operator!=( myDS::linkedList<VALUE_TYPE>::_iterator _b)
149     {
150         if (&(*_b) == &(_ptr->data))
```

# 暨南大学本科实验报告专用纸(附页)

---

```
150         return 0;
151     else
152         return 1;
153     }
154 };
155
156 linkedNode * head = new linkedNode();
157 linkedNode * tail = new linkedNode();
158 int cap = 0;
159
160 public:
161     linkedList(){
162         head->linkNext(tail);
163     }
164
165     ~linkedList(){
166         clear();
167         delete head;
168         delete tail;
169     }
170
171     void push_back(VALUE_TYPE t) {
172         tail->data = t;
173         tail->linkNext(new linkedNode());
174         tail = tail->next;
175         cap ++;
176     }
177
178     void push_front(VALUE_TYPE t) {
179         head->data = t;
180         head = (head->linkPriv(new linkedNode()));
181         cap ++;
182     }
183
184     void clear() {
185         linkedNode * deletingObject;
186         while(tail->priv != head) {
187             deletingObject = tail;
188             tail = tail->priv;
189             delete deletingObject;
190         }
191         cap = 0;
192         delete head;
193         delete tail;
194         tail = new linkedNode();
195         head = new linkedNode();
196         head->linkNext(tail);
197     }
198
199     std::size_t erase(VALUE_TYPE p) {
200         linkedNode * ptr = head;
201         int ttl = 0;
```

# 暨南大学本科实验报告专用纸(附页)

```
202         while(ptr->next != tail) {
203             if(ptr->next->data == p){
204                 ptr->deleteNext();
205                 ttl ++;
206             } else ptr = ptr->next;
207         }
208         cap -= ttl;
209         return ttl;
210     }
211
212     std::size_t size() {return cap;}
213
214     bool erase(linkedList<VALUE_TYPE>::_iterator p) {
215         myDS::linkedList<VALUE_TYPE>::_iterator ptr = this-
>begin();
216         linkedNode * cur = head;
217         while(ptr != p) {
218             cur = cur->next;
219             ptr ++;
220             if(cur == tail) return 0;
221         }
222         cur->deleteNext();
223         cap --;
224         return 1;
225     }
226
227     myDS::linkedList<VALUE_TYPE>::_iterator begin() {
228         enum
myDS::linkedList<VALUE_TYPE>::_iterator::__iter_dest_type _FRONT =
myDS::linkedList<VALUE_TYPE>::_iterator::__iter_dest_type::front;
229         return myDS::linkedList<VALUE_TYPE>::_iterator(head-
>next,_FRONT);
230     }
231
232     myDS::linkedList<VALUE_TYPE>::_iterator rbegin() {
233         enum
myDS::linkedList<VALUE_TYPE>::_iterator::__iter_dest_type _BACK =
myDS::linkedList<VALUE_TYPE>::_iterator::__iter_dest_type::back;
234         return myDS::linkedList<VALUE_TYPE>::_iterator(tail-
>priv,_BACK);
235     }
236
237     myDS::linkedList<VALUE_TYPE>::_iterator end() {
238         enum
myDS::linkedList<VALUE_TYPE>::_iterator::__iter_dest_type _FRONT =
myDS::linkedList<VALUE_TYPE>::_iterator::__iter_dest_type::front;
239         return
myDS::linkedList<VALUE_TYPE>::_iterator(tail,_FRONT);
240     }
241
```

# 暨南大学本科实验报告专用纸(附页)

```
242         myDS::linkedList<VALUE_TYPE>::_iterator rend() {
243             enum
244 myDS::linkedList<VALUE_TYPE>::_iterator::__iter_dest_type _BACK =
245 myDS::linkedList<VALUE_TYPE>::_iterator::__iter_dest_type::back;
246             return myDS::linkedList<VALUE_TYPE>::_iterator(head,_BACK);
247         }
248         myDS::linkedList<VALUE_TYPE>::_iterator get(std::size_t p) {
249             linkedNode * ptr = head->next;
250             while(p --) ptr = ptr->next;
251             enum
252 myDS::linkedList<VALUE_TYPE>::_iterator::__iter_dest_type _FRONT =
253 myDS::linkedList<VALUE_TYPE>::_iterator::__iter_dest_type::front;
254             return myDS::linkedList<VALUE_TYPE>::_iterator(ptr,_FRONT);
255         }
256         VALUE_TYPE & operator[](std::size_t p) {
257             linkedNode * ptr = head;
258             while(p --) ptr = ptr->next;
259             return ptr->next->data;
260         }
261 #ifdef _PRIVATE_DEBUG
262         void innerPrint()
263         {
264             std::cout << "--Header[" << head << "]: " << head->data <<
265             "\n";
266             std::cout << "--Tail[" << tail << "]: " << tail->data <<
267             "\n";
268             std::cout << "-----\n";
269             std::cout << "cur:" << cap<< "\n";
270             auto ptr = head;
271             do {
272                 std::cout << "[" << ptr << "] ->next:" << ptr->next << "
273                 ->priv:" << ptr->priv << " ||data:" << ptr->data << "\n";
274                 ptr = ptr->next;
275             }while(ptr != nullptr);
276         }
277 #endif
278     };
279 }
```

## 3.2. \_PRIV\_TEST.cpp

```
1  #define DS_TOBE_TEST linkedList
2
3  #define _PRIVATE_DEBUG
```

# 暨南大学本科实验报告专用纸(附页)

---

```
4
5  #include "Dev\01\linkedList.h"
6
7  #include <iostream>
8  #include <math.h>
9  #include <vector>
10
11  using namespace std;
12
13  using TBT = myDS::DS_TOBE_TEST<int>;
14
15  void accuracyTest() { //结构正确性测试
16
17      TBT tc = TBT();
18      for(;;)
19      {
20          string op;
21          cin >> op;
22          if(op == "clr") { //清空
23              tc.clear();
24          } else if(op == "q") //退出测试
25          {
26              return;
27          } else if(op == "pb") //push_back
28          {
29              int c;
30              cin >> c;
31              tc.push_back(c);
32          } else if(op == "pf") //push_front
33          {
34              int c;
35              cin >> c;
36              tc.push_front(c);
37          } else if(op == "at") //随机访问
38          {
39              int p;
40              cin >> p;
41              cout << tc[p] << "\n";
42          } else if(op == "delEL") //删除所有等于某值元素
43          {
44              int p;
45              cin >> p;
46              cout << tc.erase(p) << "\n";
47          } else if(op == "delPS") //删除某位置上的元素
48          {
49              int p;
50              cin >> p;
51              cout << tc.erase(tc.get(p)) << "\n";
52          } else if(op == "iterF") //正序遍历
53          {
54              tc.innerPrint();
```



# 暨南大学本科实验报告专用纸(附页)

```
55         cout << "Iter with index:\n";
56         for(int i = 0;i < tc.size();i ++) cout << tc[i] << " ";cout
57         << "\n";
58         cout << "Iter with begin end\n";
59         for(auto x = tc.begin();x != tc.end();x ++) cout << (*x) <<
60         " ";cout << "\n";
61         cout << "Iter with AUTO&&\n";
62         for(auto x:tc) cout << x << " ";cout << "\n";
63     } else if(op == "iterB") //正序遍历
64     {
65         tc.innerPrint();
66         cout << "Iter with index:\n";
67         for(int i = 0;i < tc.size();i ++) cout << tc[tc.size()-1-i]
68         << " ";cout << "\n";
69         cout << "Iter with begin end\n";
70         for(auto x = tc.rbegin();x != tc.rend();x ++) cout << (*x)
71         << " ";cout << "\n";
72         // cout << "Iter with AUTO&&\n";." \n";
73     } else if(op == "mv")//单点修改
74     {
75         int p;
76         cin >> p;
77         int tr;
78         cin >> tr;
79         tc[p] = tr;
80     } else if(op == "")
81     {
82     }
83     }
84 }
85
86
87
88 void memLeakTest() { //内存泄漏测试
89     TBT tc = TBT();
90     for(;;){
91         tc.push_back(1);
92         tc.push_back(1);
93         tc.push_back(1);
94         tc.push_back(1);
95         tc.clear();
96     }
97 }
98
99 signed main()
100 {
101     // accuracyTest();
```

# 暨南大学本科实验报告专用纸(附页)

---

```
102     memLeakTest();  
103 }
```

## 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
```

# 暨南大学本科实验报告专用纸(附页)

---

```
[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
```

# 暨南大学本科实验报告专用纸(附页)

---

```
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	
 ...		0%	0.6 MB	