山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

学号：202000130143	姓名： 郑凯饶	班级： 计科 20.1
实验题目：链式描述线性表		
实验学时：2	实验日期：10.23	

实验目的：

1、掌握线性表结构、链式描述方法（链式存储结构）、链表的实现。

2、掌握链表迭代器的实现与应用。

软件开发环境：

Windows 10 家庭中文版 64 位（10.0，版本 18363）

Dev-C++ IDE

1. 实验内容
2. 封装链表类、链表迭代器类，包含操作：在指定位置插入元素，删除指定元素，搜索链表中是否有指定元素，原地逆置链表，输出链表。
3. 使用 1> 中实现的链表类，迭代器，实现链表的排序、合并。

4. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法）

```
class chain {
    public:
        chain(int iniCap = 10);
        chain(const chain<T>&);
        ~chain();

        int indOf(const T& tar_ele) const;
        void erase(int ind);
        void insert(int tar_ind, const T& tar_ele);
        void reverse();
        bool empty() { return Size == 0; }
        int get_S() { return Size; }
//        chainNode<T>* get_F() { return fiNode; }
        void binSort(int r);
//      void clear();
        void Merge(chain<T> &c1, chain<T> &c2);     /
        int Cal();
        void M_sort() {
            fiNode = MergeSort(fiNode);
        }
        // 迭代器
        class iterator;
        iterator begin() { return iterator(fiNode); }
        iterator end() { return iterator(NULL); }

        class iterator {
    protected:
        chainNode<T>* fiNode;
        int Size;

};
```

这是最终的链表声明。

indOf() ： 返回待查找元素的索引，若元素不存在，返回-1。

Erase() ： 删除指定索引的元素。

Insert() ： 向指定位置插入指定元素。

Reverse() ： 原地逆置链表。

BinSort() ： 箱子排序，但题目输入数据包含负数，实际未使用。

Merge() ： 将参数部分的链表有序合并（归并）到该链表，默认当前链表为空。

Cal() ： 计算链表中元素与其索引的异或和。

M_sort() ： 递归实现的归并排序，由于要递归调用，主体部分 MergeSort()在类的外部实现，实际使用。

Class iterator ： 链表的迭代器。
数据部分包括链表的头结点 fiNode 和大小 Size。

5. 测试结果（测试输入，测试输出）
在 OJ 平台上成功提交。

6. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）
1>程序异常退出，返回 3221226356 或 3221225477。
3221225477 对应的是 STATUS_ACCESS_VIOLATION，基本上是访问了不该访问的内存。主要是对动态空间方法不熟悉，或者是操作指针的过程中细节出错了。
3221226356 在 NTSTATUS 中未找到对应的错误，但我两次出错都与空间析构有关，第一次是实验 3 数组描述线性表中在整体析构之前先调用析构回收了部分空间，第二次是本次实验我在复制链表时直接将一个链表的头节点赋值给另一个链表，并没有额外申请空间 hhh，可以想象这会造成空间的二次析构。
2>使用 binSort()在平台提交 RE。
平台的数据范围未指定，我思维惯性以为是[1,1e5]之类的数据，实际应该包含负数。

7. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）
1

```
#include<bits/stdc++.h>
using namespace std;

// 节点
template<class T>
struct chainNode {
    T ele;
    chainNode<T>* next;

    chainNode() { }
    chainNode(const T& ele) { this -> ele = ele; }
    chainNode(const T& ele, chainNode<T>* next) {
        this -> ele = ele;
        this -> next = next;
    }
};


template<class T>
class chain {
    public:
        chain(int iniCap = 10);
        chain(const chain<T>&);
        ~chain();
```

```
32.
33.         int indOf(const T& tar_ele) const;
34.         void erase(int ind);
35.         void insert(int tar_ind, const T& tar_ele);
36.         void reverse();
37.
38.         // 迭代器
39.     class iterator;
40.     iterator begin() { return iterator(fiNode); }
41.  iterator end() { return iterator(NULL); }
42.
43.  class iterator {
44.      public:
45.          // typedef
46.          // ......
47.
48.          iterator(chainNode<T>* theNode = NULL) { node = theNode; }
49.
50.          // 解引用操作符
51.          T& operator * () const { return node -> ele; }
52.          T* operator -> () const { return &node -> ele; }
53.
54.          // 迭代器加法操作
55.          iterator& operator ++ () { node = node -> next; return *this; }
56.          iterator operator ++ (int) { iterator old = *this; node = node -> next
   ; return old; }
57.
58.          // 相等检验
59.          bool operator != (const iterator r) const { return node != r.node; }
60.          bool operator == (const iterator r) const { return node == r.node; }
61.      protected:
62.          chainNode<T>* node;
63.  };
64.    protected:
65.        chainNode<T>* fiNode;
66.        int Size;
67.
68.};
69.
70.// chain 实现
71.template<class T>
72.chain<T>::chain(int iniCap) {
73.    fiNode = NULL;
74.    Size = 0;
75.}
76.
```

```cpp
template<class T>
chain<T>::chain(const chain<T>& theList) {
    Size = theList.Size;
    if (Size == 0) { fiNode = NULL; return;}

    chainNode<T>* s = theList.fiNode;
    fiNode = new chainNode<T>(s -> ele);
    s = s -> next;
    chainNode<T>* tarNode = fiNode;
    while (s != NULL) {
        tarNode -> next = new chainNode<T>(s -> ele);
        tarNode = tarNode -> next;
        s = s -> next;
    }
    tarNode -> next = NULL;
}

template<class T>
chain<T>::~chain() {
    while (fiNode != NULL) {
        chainNode<T>* nextNode = fiNode -> next;
        delete fiNode;
        fiNode = nextNode;
        }
    }

    template<class T>
    int chain<T>::indOf(const T& tar_ele) const { // 返回元素首次出现的索引
        chainNode<T>* curNode = fiNode;
        int ind = 0;
        while(curNode != NULL && curNode -> ele != tar_ele) {
            curNode = curNode -> next;
            ind ++;
        }
        if (curNode == NULL) return -1;
        return ind;
    }

    template<class T>
    void chain<T>::erase(int ind) {
        chainNode<T>* tar_;
        if (ind == 0) {
            tar_ = fiNode;
            fiNode = fiNode -> next;
        }
        else {
```

```
123.             chainNode<T>* p = fiNode;
124.             // 定位待删除节点的前驱
125.             for (int i = 0; i < ind - 1; i++) p = p -> next;
126.             tar_ = p -> next;
127.             p -> next = p -> next -> next;
128.         }
129.         Size--;
130.         delete tar_;
131.     }
132.
133.     template<class T>
134.     void chain<T>::insert(int tar_ind, const T& tar_ele) {
135.         if (tar_ind == 0) {
136.             fiNode = new chainNode<T>(tar_ele, fiNode);
137.         }
138.         else {
139.             chainNode<T>* p = fiNode;
140.             for (int i = 0; i < tar_ind - 1; i++) p = p -> next;
141.             p -> next = new chainNode<T>(tar_ele, p -> next);
142.         }
143.         Size++;
144.     }
145.
146.     template<class T>
147.     void chain<T>::reverse() { // 1 -> 2 -> 3 -> 4 -> 5 // 1 -> 2 -> NULL // 1
148.      if(Size <= 1) return;
149.     // chainNode<T>* cur = fiNode -> next, pre = fiNode, nxt = cur -> next;
150.     //invalid conversion from 'chainNode<int>*' to 'const int&' [-fpermissive]
151.      chainNode<T>* cur = fiNode -> next;
152.      chainNode<T>* pre = fiNode;
153.      chainNode<T>* nxt = cur -> next;
154.
155.      while (nxt != NULL) {
156.       cur -> next = pre;
157.       pre = cur;
158.       cur = nxt;
159.       nxt = cur -> next;
160.      }
161.      cur -> next = pre;
162.
163.      fiNode -> next = NULL;
164.      fiNode = cur;
165.     }
166.
167.     void op() {
168.
```

```
169.       chain<int> tt;
170.
171.       int n, q; cin >> n >> q;
172.       for (int i = 0, in; i < n; i++) {
173.           cin >> in;
174.           tt.insert(i, in);
175.       }
176.
177.       for (int i = 1, op, idx, val; i <= q; i++) {
178.        cin >> op;
179.        if (op == 1) {
180.         cin >> idx >> val;
181.         tt.insert(idx, val);
182.     }
183.     else if (op == 2) {
184.      cin >> val;
185.      int tarr = tt.indOf(val);
186.      if (tarr != -1) tt.erase(tarr);
187.      else cout << -1 << '\n';
188.     }
189.     else if (op == 3) {
190.      // 逆置 operation
191.      tt.reverse();
192.     }
193.     else if (op == 4) {
194.      cin >> val;
195.      cout << tt.indOf(val) << '\n';
196.     }
197.     else if(op == 5) {
198.      int ans = 0, i = 0;
199.      for (chain<int>::iterator it = tt.begin(); it != tt.end(); it++, i++) {
200.       ans = ans + (i ^ (*it));
201.      }
202.      cout << ans << '\n';
203.     }
204.    }
205.
206.    // check
207.   // for (chain<int>::iterator it = tt.begin(); it != tt.end(); it++) {
208.   //  cout << *it << ' ';
209.   // }
210.   // cout << '\n';
211.   }
212.
213.   int main(){
214.       op();
```

```
    return 0;
  }
```

2

```
#include<bits/stdc++.h>
using namespace std;

// 3221226356

// 节点
template<class T>
struct chainNode {
    T ele;
    chainNode<T>* next;

    chainNode() { }
    chainNode(const T& ele) { this -> ele = ele; }
    chainNode(const T& ele, chainNode<T>* next) {
        this -> ele = ele;
        this -> next = next;
    }
};


template<class T>
class chain {
    public:
        chain(int iniCap = 10);
        chain(const chain<T>&);
        ~chain();

        int indOf(const T& tar_ele) const;
        void erase(int ind);
        void insert(int tar_ind, const T& tar_ele);
        void reverse();
        bool empty() { return Size == 0; }
        int get_S() { return Size; }
//        chainNode<T>* get_F() { return fiNode; }
  void binSort(int r);
//  void clear();
        void Merge(chain<T> &c1, chain<T> &c2);  //
        int Cal();
        void M_sort() {
         fiNode = MergeSort(fiNode);
```

```
41.  }
42.         // 迭代器
43.      class iterator;
44.      iterator begin() { return iterator(fiNode); }
45.  iterator end() { return iterator(NULL); }
46.
47.  class iterator {
48.      public:
49.          // typedef
50.          // ......
51.
52.          iterator(chainNode<T>* theNode = NULL) { node = theNode; }
53.
54.          // 解引用操作符
55.          T& operator * () const { return node -> ele; }
56.          T* operator -> () const { return &node -> ele; }
57.
58.          // 迭代器加法操作
59.          iterator& operator ++ () { node = node -> next; return *this; }
60.          iterator operator ++ (int) { iterator old = *this; node = node -> n
    ext; return old; }
61.
62.          // 相等检验
63.          bool operator != (const iterator r) const { return node != r.node;
    }
64.          bool operator == (const iterator r) const { return node == r.node;
    }
65.      protected:
66.          chainNode<T>* node;
67.  };
68.    protected:
69.        chainNode<T>* fiNode;
70.        int Size;
71.
72.};
73.
74.// chain 实现
75.template<class T>
76.chain<T>::chain(int iniCap) {
77.    fiNode = NULL;
78.    Size = 0;
79.}
80.
81.template<class T>
82.chain<T>::chain(const chain<T>& theList) {
83.    Size = theList.Size;
```

```
84.     if (Size == 0) { fiNode = NULL; return;}
85.
86.     chainNode<T>* s = theList.fiNode;  //
87.     fiNode = new chainNode<T>(s -> ele);
88.     s = s -> next;
89.     chainNode<T>* tarNode = fiNode;
90.     while (s != NULL) {
91.         tarNode -> next = new chainNode<T>(s -> ele);
92.         tarNode = tarNode -> next;
93.         s = s -> next;
94.     }
95.     tarNode -> next = NULL;
96. }
97.
98. template<class T>
99. chain<T>::~chain() {
100.     while (fiNode != NULL) {
101.         chainNode<T>* nextNode = fiNode -> next;
102.         delete fiNode;
103.         fiNode = nextNode;
104.     }
105. }
106.
107. template<class T>
108. int chain<T>::indOf(const T& tar_ele) const { // 返回元素首次出现的索引
109.     chainNode<T>* curNode = fiNode;
110.     int ind = 0;
111.     while(curNode != NULL && curNode -> ele != tar_ele) {
112.         curNode = curNode -> next;
113.         ind ++;
114.     }
115.     if (curNode == NULL) return -1;
116.     return ind;
117. }
118.
119. template<class T>
120. void chain<T>::erase(int ind) {
121.     chainNode<T>* tar_;
122.     if (ind == 0) {
123.         tar_ = fiNode;
124.         fiNode = fiNode -> next;
125.     }
126.     else {
127.         chainNode<T>* p = fiNode;
128.         // 定位待删除节点的前驱
129.         for (int i = 0; i < ind - 1; i++) p = p -> next;
```

```
130.         tar_ = p -> next;
131.         p -> next = p -> next -> next;
132.     }
133.     Size--;
134.     delete tar_;
135. }
136. 
137. template<class T>
138. void chain<T>::insert(int tar_ind, const T& tar_ele) {
139.     if (tar_ind == 0) {
140.         fiNode = new chainNode<T>(tar_ele, fiNode);
141.     }
142.     else {
143.         chainNode<T>* p = fiNode;
144.         for (int i = 0; i < tar_ind - 1; i++) p = p -> next;
145.         p -> next = new chainNode<T>(tar_ele, p -> next);
146.     }
147.     Size ++;
148. }
149. 
150. template<class T>
151. void chain<T>::reverse() { // 1 -> 2 -> 3 -> 4 -> 5 // 1 -> 2 -> NULL // 1
152.  if(Size <= 1) return;
153. // chainNode<T>* cur = fiNode -> next, pre = fiNode, nxt = cur -> next;
154. //invalid conversion from 'chainNode<int>*' to 'const int&' [-fpermissive]
155.  chainNode<T>* cur = fiNode -> next;
156.  chainNode<T>* pre = fiNode;
157.  chainNode<T>* nxt = cur -> next;
158. 
159.  while (nxt != NULL) {
160.   cur -> next = pre;
161.   pre = cur;
162.   cur = nxt;
163.   nxt = cur -> next;
164.  }
165.  cur -> next = pre;
166. 
167.  fiNode -> next = NULL;
168.  fiNode = cur;
169. }
170. 
171. // 0. Cal
172. // 1. Sort
173. // 2. Merge
174. // 3. Cal
175. 
```

```
176.template<class T>
177.void chain<T>::binSort(int r) {
178. chainNode<T> **bm, **tp;
179. bm = new chainNode<T>* [r + 1];
180. tp = new chainNode<T>* [r + 1];
181. for (int b = 0; b <= r; b++) bm[b] = NULL;
182.
183. for (; fiNode != NULL; fiNode = fiNode -> next) {
184.  int Bin = fiNode -> ele;
185.  if (bm[Bin] == NULL) bm[Bin] = tp[Bin] = fiNode;
186.  else {
187.   tp[Bin] -> next = fiNode;
188.   tp[Bin] = fiNode;
189.  }
190. }
191.
192. chainNode<T>* y = NULL;
193. for (int Bin = 0; Bin <= r; Bin++) {
194.  if (bm[Bin] != NULL) {
195.   if (y == NULL) fiNode = bm[Bin];
196.   else y -> next = bm[Bin];
197.   y = tp[Bin];
198.  }
199. }
200. if (y != NULL) y -> next = NULL;
201.
202. delete [] bm;
203. delete [] tp;
204.}
205.
206.template<class T>
207.void chain<T>::Merge(chain<T> &c, chain<T> &c_) {
208. // 归并
209. // clear() != ~
210. // iterator != pointer
211.
212. fiNode = new chainNode<T>;
213. chainNode<T>* ptr;
214. if(c_.get_S() == 0) {
215.//  cout << "1" << '\n';
216.  Size = c.get_S();
217.  chain<int>::iterator it = c.begin();
218.  this -> fiNode -> ele = *it;
219.  ptr = fiNode;
220.  it++;
221.  while (it != c.end()) {
```

```
222.    chainNode<T>* newNode = new chainNode<T>;
223.    newNode -> ele = *it;
224.    ptr -> next = newNode;
225.    ptr = newNode;
226.    it++;
227.   }
228.   ptr -> next = NULL;
229.   // clear
230.   return;
231. }
232.
233. // error
234. chain<int>::iterator it = c.begin();
235. chain<int>::iterator it_ = c_.begin();
236.
237. if ((*it) < (*it_)) fiNode -> ele = *it, it++;
238. else{
239.   fiNode -> ele = *it_;
240.   it_++;
241. }
242. Size ++;
243.
244. // 维护链表尾节点
245. chainNode<T>* p = this -> fiNode;
246.
247. while (it != c.end() && it_ != c_.end()) {
248.//  cout << "2" << '\n';
249.   chainNode<T>* newNode = new chainNode<T>;
250.   if (*it < *it_) {
251.    newNode -> ele = *it;
252.    it ++;
253.   }
254.   else {
255.    newNode -> ele = *it_;
256.    it_ ++;
257.   }
258.   p -> next = newNode;
259.   p = newNode;
260.   Size ++;
261. }
262.
263. while (it != c.end()) {
264.   chainNode<T>* newNode = new chainNode<T>;
265.   newNode -> ele = *it;
266.   it ++;
267.   p -> next = newNode;
```

```
  p = newNode;
  Size ++;
 }
 while (it_ != c_.end()) {
  chainNode<T>* newNode = new chainNode<T>;
  newNode -> ele = *it_;
  it_ ++;
  p -> next = newNode;
  p = newNode;
  Size ++;
 }

// c.~chain();
// c_.~chain();

}

template<class T>
int chain<T>::Cal() {
 int ind = 0, res = 0;
 chainNode<T>* p = fiNode;
 while (p != NULL) {
  res += ind ^ (p -> ele);
  p = p -> next;
  ind ++;
 }
 return res;
}

template<class T>
chainNode<T>* MergeTwoList(chainNode<T>* l1, chainNode<T>* l2) {
 chainNode<T>* dummpy = new chainNode<T>;  // 哑节点
 chainNode<T>* tail = dummpy;
 while (l1 != NULL && l2 != NULL) {
  if ((l1 -> ele) < (l2 -> ele)) {
   tail -> next = l1;
   l1 = l1 -> next;
  }
  else {
   tail -> next = l2;
   l2 = l2 -> next;
  }
  tail = tail -> next;
 }

 while (l1 != NULL) {
```

```
  tail -> next = l1;
  tail = tail -> next;
  l1 = l1 -> next;
 }
 while (l2 != NULL) {
  tail -> next = l2;
  tail = tail -> next;
  l2 = l2 -> next;
 }

 return dummpy -> next;      // 不影响数据的组织
}


template<class T>
chainNode<T>* MergeSort(chainNode<T>* head) {
 if (head == NULL || head -> next == NULL) return head;
 chainNode<T>* slow = head;
 chainNode<T>* fast = head -> next;
 while (fast != NULL && fast -> next != NULL) {
  slow = slow -> next;
  fast = fast -> next -> next;
 }
 chainNode<T>* last = slow -> next;
 slow -> next = NULL;
 chainNode<T>* fi = MergeSort(head);
 chainNode<T>* se = MergeSort(last);
 chainNode<T>* res = MergeTwoList(fi, se);
 return res;
}

void solve() {
 chain<int> t1, t2, T;
 int n, m, in; cin >> n >> m;
 for (int i = 0; i < n; i++) {
  cin >> in;
  t1.insert(i, in);
 }
 for (int i = 0; i < m; i++) {
  cin >> in;
  t2.insert(i, in);
 }

 // insertSort
 // ......

```

```
360. // MergeSort
361. t1.M_sort();
362. t2.M_sort();
363.
364.// t1.binSort(1e7);
365.// t2.binSort(1e7);
366.// cout << "ok" << '\n';
367.
368. cout << t1.Cal() << '\n';
369. cout << t2.Cal() << '\n';
370.
371. T.Merge(t1, t2);
372.
373. cout << T.Cal() << '\n';
374. // check
375.// for (chain<int>::iterator it = T.begin(); it != T.end(); it++) {
376.//  cout << *it << ' ';
377.// }
378.// cout << '\n';
379.
380.}
381.
382.int main(){
383. solve();
384. return 0;
385.}
```