

类定义

链表结点定义

```

1  template<class T>
2  struct Node {
3      T data;
4      Node<T>* next;
5      Node() : next(nullptr) {}
6      Node(T x) : data(x), next(nullptr) {}
7  };

```

一、顺序表定义

```

1  template<class T>
2  class SeqList {
3  private:
4      T* data;
5      int size;
6      void Renew();
7      void QuickSort(int left, int right);
8
9  public:
10     SeqList() : data(nullptr), size(0) {}
11     SeqList(T a[], int n);
12     SeqList(int n) : data(new T[n]), size(n) {}
13     ~SeqList();
14     void Output();
15     T DeleteMin(); // T1: delete and return min
16     void DeleteAllX(T x); // T2: delete all elements equal
    to x
17     void DeleteRange(T left, T right); // T3: delete ordered list's all
    elements range from s to t
18     void Unique(); // T4: unique
19     void Insert(int pos, T x);
20     void IncreaseInsert(T x); // T5: insert x to ordered list
21     void SortOnce(); // T9: sort once
22     void Sort(); // exp T1: quick sort
23     void Merge(SeqList<T>& obj); // exp T1: merge two ordered list
24     T& operator[](int idx) { return data[idx]; }

```

```

25     bool find(T x);
26 };

```

二、单链表定义

```

1  template<class T>
2  class LinkList {
3  private:
4      Node<T>* head;
5      Node<T>* Reverse(Node<T>* node);           // reverse a list
6
7  public:
8
9      LinkList() : head(nullptr) {}
10     LinkList(T a[], int n);
11     LinkList(LinkList<T>& obj);                 // T7: copy construct
12     ~LinkList();
13
14     void Output();                             // print to console
15     void OutputToFile(const string& path);      // print to file
16     void PushFront(T x);                       // push node to head
17     void Reverse();                             // public use to reverse
        a list
18     void Delete(T x);                          // delete item with
        value of x
19     bool Find(T x);                            // find item with value
        of x
20     void Split(LinkList<T>& odd, LinkList<T>& even); // T6: split into odd
        and even
21     void PrintInOrder();                       // T8: print list in
        order
22     void Merge(LinkList<T>& obj);               // T10: merge two
        ordered list
23 };

```

三、循环链表定义

```

1  template<class T>
2  class CircleList {
3  private:
4      Node<T>* head;
5
6  public:
7      CircleList() : head(nullptr) {}
8      CircleList(T a[], int n);
9      ~CircleList();
10
11     int CountNode();
12 };

```

作业

T1

试编写算法，从顺序表中删除具有最小值的元素并由函数返回最小值，空出的位置由最后一个元素填补，若顺序表为空则显示出错信息并退出运行。

查询再覆盖即可， $O(n)$

```

1  template<class T>
2  T SeqList<T>::DeleteMin() {
3      if (!size) {
4          cerr << "Error: empty seqlist!" << endl;
5          return 0;
6      }
7
8      int min_index = 0;
9      for (int i = 0; i < size; i++) {
10         if (data[i] < data[min_index]) {
11             min_index = i;
12         }
13     }
14
15     T temp = data[min_index];
16
17     if (size == 1) {
18         size--;

```

```

19     } else {
20         data[min_index] = data[size - 1];
21         size--;
22     }
23
24     return temp;
25 }

```

T2

试编写算法，从顺序表中删除具有给定值 x 的所有元素。

扫描到进行移位覆盖即可， $O(n^2)$

```

1  template<class T>
2  void SeqList<T>::DeleteAllX(T x) {
3      for (int i = 0; i < size; i++) {
4          if (data[i] == x) {
5              for (int j = i + 1; j < size; j++) {
6                  data[j - 1] = data[j];
7              }
8              size--;
9          }
10     }
11 }

```

T3

试编写算法，从有序表中删除其值在给定值 s 和 t (要求 s 小于 t) 之间的所有元素。

确定好范围边界后移位覆盖即可， $O(n)$

```

1  template<class T>
2  void SeqList<T>::DeleteRange(T left, T right) {
3      if (left > right) {
4          cerr << "Error: Wrong Range!";
5          return;
6      }
7
8      // find boundary of two points

```

```

9     int pl = 0, pr = size - 1;
10    while (data[pl] < left) pl++;
11    while (data[pr] > right) pr--;
12
13    for (int i = pr + 1, j = pl; i < size; i++, j++) {
14        data[j] = data[i];
15    }
16
17    size -= pr - pl + 1;
18 }

```

T4

试编写算法，从顺序表中删除所有其值重复的元素，使所有元素的值均不同。如对于线性表(2, 8, 9, 2, 5, 5, 6, 8, 7, 2)，则执行此算法后变为(2, 8, 9, 5, 6, 7)。注意：表中元素未必是排好序的，且每个值的第一次出现应当保留。

用哈希表存储首次出现的值，后续再覆盖即可， $O(n)$ 。当然也可以对于每一个数枚举后缀中出现的重复数并直接覆盖， $O(n^3)$

```

1  template<class T>
2  void SeqList<T>::Unique() {
3      // use stl to hash, we can also use bucket to hash if data is not big
    enough
4      unordered_map<T, bool> hash;
5      for (int i = 0; i < size; i++) {
6          hash[data[i]] = true;
7      }
8
9      // hashing logic looks like a stack, so it's reversed
10     int idx = hash.size() - 1;
11     for (auto& x: hash) {
12         data[idx] = x.first;
13         idx--;
14     }
15
16     size = hash.size();
17 }

```

T5

设表L用数组表示，且各元素值递增有序。试写一算法，将元素x插入到表L的适当位置，使得表中元素仍保持递增有序。

二分查到待插入的位置后先执行后缀后移操作，最后单点插入即可，二分是 $O(\log n)$ ，后缀后移是 $O(n)$ ，因此上限为 $O(n)$ 。当然也可以 $O(n)$ 查找待插入位置。值得注意的是，首次插入元素需要对顺序表动态扩容，仿照 `std::vector<T>` 的动态扩容机制。

```

1  template<class T>
2  void SeqList<T>::IncreaseInsert(T x) {
3      int l = 0, r = size - 1;
4      while (l < r) {
5          int m = (l + r) >> 1;
6          if (data[m] < x) l = m + 1;
7          else r = m;
8      }
9      Insert(r, x);
10 }
11
12 template<class T>
13 void SeqList<T>::Insert(int pos, T x) {
14     Renew();
15     for (int i = size - 1; i >= pos; i--)
16         data[i + 1] = data[i];
17     data[pos] = x;
18     size++;
19 }
20
21 template<class T>
22 void SeqList<T>::Renew() {
23     // dynamic expansion
24     int newsize = size + size;
25     T* newdata = new T[newsize];
26     for (int i = 0; i < size; i++)
27         newdata[i] = data[i];
28     delete[] data;
29     data = newdata;
30 }

```

T6

试编写算法，根据一个元素类型为整型的单链表生成两个单链表，使得第一个单链表中包含原单链表中所有元素值为奇数的结点，使得第二个单链表中包含原单链表中所有元素值为偶数的结点，原有单链表保持不变。

遍历链表时按照奇偶性进行头插法构造即可，如果想要顺序不变可以反转链表， $O(n)$

```

1  template<class T>
2  void LinkList<T>::Split(LinkList<T>& odd, LinkList<T>& even) {
3      Node<T>* p = head;
4      while (p) {
5          if (p->data % 2) odd.PushFront(p->data);
6          else even.PushFront(p->data);
7          p = p->next;
8      }
9      odd.Reverse();
10     even.Reverse();
11 }

```

至于如何反转链表，这是一个非常经典的问题了，下面罗列三种解法：

一、三指针迭代

```

1  template<class T>
2  Node<T>* LinkList<T>::Reverse(Node<T>* node) {
3      if (!node) return node;
4      Node<T>* pre = nullptr, * now = node;
5      while (now) {
6          Node<T>* temp = now->next;
7          now->next = pre;
8          pre = now;
9          now = temp;
10     }
11     return pre;
12 }

```

二、双指针+反向建表

```

1  template<class T>
2  Node<T>* LinkedList<T>::Reverse(Node<T>* node) {
3      if (!node) return node; // empty list
4      Node<T>* tail = nullptr;
5      Node<T>* p = node;
6      while (p) {
7          Node<T>* now = p;
8          p = p->next;
9          now->next = tail;
10         tail = now;
11     }
12     return tail;
13 }

```

三、递归翻转

```

1  template<class T>
2  Node<T>* LinkedList<T>::Reverse(Node<T>* node) {
3      if (!node || !node->next) return node; // empty list or tail node
4      Node<T>* tail = Reverse(node->next);
5      node->next->next = node;
6      node->next = nullptr;
7      return tail;
8  }

```

T7

已知一个单链表，设计一个复制单链表的算法。

头插法建表后翻转即可， $O(n)$


```

1  template<class T>
2  LinkList<T>::LinkList(LinkList<T>& obj) {
3      head = nullptr;
4      Node<T>* p = obj.head;
5      while (p) {
6          Node<T>* now = new Node<T>();
7          now->data = p->data;
8          now->next = head;
9          head = now;
10         p = p->next;
11     }
12     head = Reverse(head);
13 }

```

T8

已知一个无序单链表，表中结点的 data 字段为正整数。设计一个算法按递增次序打印表中结点的值。

显然我们可以双重循环 $O(n^2)$ 解决此问题，即对于每一个元素枚举比他小的元素即可。当然也可以离线法处理，遍历单链表并取出所有元素值后排序输出即可，以 STL 中的优先队列为例解决此问题， $O(n \log n)$

```

1  template<class T>
2  void LinkList<T>::PrintInOrder() {
3      Node<T>* p = head;
4      priority_queue<T, vector<T>, greater<T>> q;
5      while (p) {
6          q.push(p->data);
7          p = p->next;
8      }
9      while (q.size()) {
10         cout << q.top() << " ";
11         q.pop();
12     }
13 }

```

T9

试编写算法，将元素为整数的顺序表 (a_1, a_2, \dots, a_n) 重新排列为以 a_1 为界的两部分： a_1 前面的值均比 a_1 小， a_1 后面的值都比 a_1 大，要求时间复杂度为 $O(n)$ 。

可以理解为快速排序中分治一层的逻辑，或者可以理解为双指针算法， $O(n)$

```

1  template<class T>
2  void SeqList<T>::SortOnce() {
3      int i = -1, j = size, x = data[0];
4      while (i < j) {
5          while (data[++i] < x);
6          while (data[--j] > x);
7          if (i < j) {
8              swap(data[i], data[j]);
9          }
10     }
11 }
```

T10

给出一个算法，求循环链表中结点的个数。

关键在于这个循环链表是怎么构造的？我们假设构造方法和单链表一致，仅仅在构造完成后使最后一个结点指向头结点。于是在遍历时进行即可进行统计， $O(n)$

```

1  template<class T>
2  int CircleList<T>::CountNode() {
3      if (!head) {
4          return 0;
5      }
6
7      int res = 1;
8      for (Node<T>* p = head->next; p != head; p = p->next) {
9          res++;
10     }
11     return res;
12 }
```

实验

实验代码: https://github.com/Explorer-Dong/DataStructure/blob/main/Code/chapter2/Experiment_2.cpp

T1

顺序表的编程实现与测试。

1. 编写main()函数对class SeqList进行测试, 要求: 使用菜单选择各项功能。
2. 扩展顺序表class SeqList的功能 (增加成员函数或友元函数) 并进行测试:
 1. 排序;
 2. 归并两个有序顺序表。

1. 对于第一题, 使用 `while+switch` 语句即可测试 SeqList 的全部功能。
2. 对于第二题
 1. 排序使用 `QuickSort` 即可, $O(n \log n)$
 2. 归并有序顺序表可以使用双指针算法, $O(n)$

T2

用顺序表编程实现一个简易的商品管理系统, 完成报告。

商品信息包括: 商品代号、商品名称、价格、库存量等。对商品库存表的管理就是首先把它读入到线性表中, 接着对它进行必要的处理, 然后再把处理后的结果写回到文件中。对商品库存表的处理假定包括以下选项:

1. 打印 (遍历) 库存表;
2. 按商品代号修改记录的当前库存量, 若查找到对应的记录, 则从键盘上输入其修正量, 把它累加到当前库存量域后, 再把该记录写回原有位置, 若没有查找到对应的记录, 则表明是一条新记录, 应接着从键盘上输入该记录的商品名称、最低库存量和当前库存量的值, 然后把该记录追加到库存表中;
3. 按商品代号删除指定记录;
4. 按商品代号对库存表中的记录排序;
5. main函数中使用菜单选择各项功能。

构造下列商品类即可利用模板元编程的 SeqList 类:

```

1 struct item {
2     string id, name;
3     int price, cnt;
4     bool operator< (const item& t) const {
5         return this->id < t.id;
6     }
7     bool operator> (const item& t) const {
8         return this->id > t.id;
9     }
10    bool operator== (const item& t) const {
11        return this->id == t.id;
12    }
13    friend ostream& operator<< (ostream& os, const item& t) {
14        os << t.id << " " << t.name << " " << t.price << " " << t.cnt;
15        return os;
16    }
17 };

```

T3

单链表的编程实现与测试。

1. 编写main()函数对class LinkedList进行测试，要求：使用菜单选择各项功能。
2. 扩展单链表class LinkedList的功能（增加成员函数或友元函数）并进行测试：
 1. 原地逆置；
 2. 合并两个有序单链表。

这些功能均已内置实现在 `LinkedList` 类中，创建好类对象后直接调用即可。

T4

用单链表编程实现一个简易的高校学籍管理系统，完成报告。

1. 学生信息包括：学号、姓名、性别、专业、出生年月等，采用单链表存储方式。
2. 提供建立、查询、删除、增加、修改等功能。
3. main函数中使用菜单选择各项功能。

以学号为唯一关键字进行增删改查工作。并构造下列学生类即可利用模板元编程的 `LinkedList` 类：

```
1 struct stu {
2     string id, name, gender, major, birth;
3     bool operator==(const stu& t) const {
4         return this->id == t.id;
5     }
6     friend ostream& operator<< (ostream& os, const stu& t) {
7         os << t.id << " " << t.name << " " << t.gender << " " << t.major << "
" << t.birth << endl;
8         return os;
9     }
10 };
```