9 月 14 日～9 月 26 日作业

1. 给定一个顺序存储的线性表 L，请设计一个时间和空间上尽可能高效的算法删除所有值大于 min 而且小于 max 的元素。

解.

设两个指针index = 0和elem = 0，如果遇到不需要删除的元素，那么index和elem同时前移一格。如果遇到需要删除的元素，则index前移一格继续遍历，elem停在原位置不动，直到index遍历到下一个不需要删除的元素时，把这个元素直接存到elem处，顶替掉要删除的元素。这样就可以删除掉所有值 $\min \leq x \leq \max$ 的元素，且这个算法的**时间复杂度是 $O(n)$，空间复杂度是 $O(1)$**。代码如下所示：

```
void MinMaxDelete(SqList *L, int min, int max){
        int index = 0;
        int elem = 0;
        while (index < L->length){
                if (L->data[index] <= min || L->data[index] >= max){
                        L->data[elem]=L->data[index];
                        elem++;
                }
                index++;
        }
        L->length = elem + 1;
}
```

2. 请设计两个算法分别实现一个顺序表和一个单链表的就地逆置，即利用原表的存储空间将原表中元素的顺序逆转过来。例如，(4, 2, 7, 3, 0) 逆置后是（0, 3, 7, 2, 4）。

解.

顺序表的就地逆置思路如下：将顺序表的元素轴对称的交换（第一个与最后一个、第二个与倒数第二个，...），**时间复杂度为 $O(n)$，空间复杂度为 $O(1)$**

```
SqList* ReverseSqList(SqList *L){
        int start = 0;
        int end = L->length - 1;
        ElemType temp;
        while (start < end){
                std::swap(L->data[start], L->data[end]);
                start++;
                end--;
        }
        return L;
}
```

链表的就地逆置思路如下：把 L 的头结点与后面元素断开，然后再用**头插法**把原本元素一个个插入表头，从而自动实现逆置，且**时间复杂度为 $O(n)$，空间复杂度为 $O(1)$**

```
LNode* ReverseLinkList(LNode *L){
        LNode *p = new LNode;
        LNode *q = new LNode;
        p = L->next;// p 记住原链表第一个结点
        L->next = NULL;//断开原链表头结点与第一个结点
        while (p != NULL){
                q = p->next; //q 记住 p 的下一个结点
                p->next = L->next;
                L->next = p;//p 头插法插入原表头
                p = q;//p 移动到下一个结点，准备把下一个结点插入
        }
        return L;
}
```

3. 试以循环链表作稀疏多项式的存储结构，编写求其导数的算法，要求利用原多项式的结点空间存放其导函数（多项式），同时释放所有被删结点。

解.

对稀疏多项式 $f(x) = a_n x^{e_n} + a_{n-1} x^{e_{n-1}} + ... + a_1 x^{e_1}$ 求导的结果是

$$f'(x) = (a_n e_n) x^{e_n - 1} + (a_{n-1} e_{n-1}) x^{e_{n-1} - 1} + ... + (a_1 e_1) x^{e_1 - 1}$$

如果稀疏多项式中存在常数项，则求导后该项为 0。用链表存储导函数时将该项删除并释放。该算法的时间复杂度为 $O(n)$，空间复杂度为 $O(1)$，代码如下：

```
void Derivative(PNode *P){
        PNode *cursor = P->next;//从表头结点的下一个开始遍历多项式每一项
        PNode *temp = new PNode;
        if(cursor->exp == 0){//特殊情况：如果输入为一个常数
                cursor->coef = 0;//直接令其导数为 0
        }
        else{
                while (cursor != P){//循环链表遍历终止条件
                        if (cursor->next->exp == 0){
                        //如果某一项为常数，导数为 0，需要把这项从链表中删除
                                temp = cursor->next;
                                cursor->next = cursor->next->next;
                                delete temp;
```

```
                        }
                        cursor->coef *= cursor->exp;//求导过程 1
                        cursor->exp--;//求导过程 2
                        cursor = cursor->next;
                }
        }
}
```

4. 请设计算法将一个循环链表表示的稀疏多项式分解成两个多项式，使这两个多项式中各自仅含奇次项或偶次项，要求算法的空间复杂度为 $O(1)$。

解.

创建一个新链表odd用来存放奇次项多项式。我们将原多项式Polyn中的奇次项全部放入链表odd后，Polyn的剩余项即为原多项式的全部偶次项。该算法的时间复杂度为 $O(n)$，空间复杂度为 $O(1)$，

```
void Decompose(PNode *Poly){
        PNode *odd = new PNode;//奇次项头指针
        PNode *odd_tail;//奇次项尾指针
        PNode *p = Poly;//原多项式头指针
        PNode *q = Poly->next;
        odd->next = NULL;
        odd_tail = odd;
        while (q != Poly){//循环链表遍历条件
                if (q->exp % 2 == 0){//偶次项保留在原多项式 Poly 中
                        p = q;
                        q = q->next;
                }
                else{//奇次项尾插入 odd 这个多项式中
                        p->next = q->next; //先从 Poly 中删除这个节点
                        //接下来用尾插法把 q 这个新结点插入 odd 这个链表里
                        q->next = NULL;//赋新结点的指针域为 NULL
                        odd_tail->next = q;//让上一个尾巴指向新结点（尾插）
                        odd_tail = q;//新结点成为了新尾巴
                        q = p->next;//更新 q 的位置，继续遍历原多项式 Poly
                }
        }
        odd_tail->next = odd;//让奇次项多项式形成一个循环链表
        std::cout << " 输出奇次项多项式: ";
        PrintPolyn(odd);
        std::cout << " 输出偶次项多项式: ";
        PrintPolyn(Poly);
```

```
}
```

5. 给定顺序表 L，请设计一个时间和空间上尽可能高效的算法将该线性表循环右移指定的 m 位。例如，(1, 2, 5, 7, 3, 4, 6, 8) 循环右移 3 位（$m = 3$）后的结果是（4, 6, 8, 1, 2, 5, 7, 3）。

解.

设顺序表的长度为 n，我们将线性表的后 m 位记作 B，前 $n - m$ 位记作 A，$m < n$。我们需要做的是将序列 AB 转化为 BA。记 $\overline{\mathsf{A}}$ 是序列 A 的倒置，则这样的转化可以通过如下操作实现：

1. 先分别将 A, B 倒置，得到 $\overline{\mathsf{A}}, \overline{\mathsf{B}}$。
2. 再将 $\overline{\mathsf{AB}}$ 整体倒置，得到 BA。

该算法的时间复杂度为 $O(n)$，空间复杂度为 $O(1)$，代码实现如下所示：

```
void ReverseSqList(SqList *L, int start, int end){
        //start, end 分别为下标, [start,end] 表示倒置范围
        ElemType temp;
        while (start < end){
                std::swap(L->data[start], L->data[end]);
                start++;
                end--;
        }
}

void MoveSqList(SqList *L, int m){
        ReverseSqList(L, 0, L->length - m - 1);
        ReverseSqList(L, L->length - m, L->length - 1);
        ReverseSqList(L, 0, L->length - 1);
}
```

6. 给定一个顺序存储的线性表 L，请设计一个算法查找该线性表中最长递增子序列。例如，(1, 9, 2, 5, 7, 3, 4, 6, 8, 0) 中最长的递增子序列为（3, 4, 6, 8）。请分析这个算法的时间和空间复杂度。

解.

首先遍历线性表 L 的每一个元素。对于每一个元素，考察以这个元素为起点的最长连续递增子序列，并将这个子序列的长度、起始位置和终止位置分别记录在`MaxLength`, `final_start`, `final_end`。
最后通过不断倾轧最大值的方法，能够得到最大连续递增子序列的长度`MaxLength`，以及该子序列对应的起始、终止下标`final_start`, `final_end`。**该算法的时间复杂度为 $O(n)$，空间复杂度为 $O(1)$。**

```
void MaxAscendingSubsequence(SqList *L){
        int start = 0;
```

```
        int end = 0;
        int final_start = 0;
        int final_end = 0;
        int MaxLength = 1;
        for (int i = 0 ; i < L->length ; i++){
                if (L->data[i]<L->data[i + 1]){
                        end++;
                }
                else{
                        if (MaxLength < end - start + 1){
                                MaxLength = end - start + 1;
                                final_start = start;
                                final_end = end;
                        }
                        start = i + 1;
                        end = i + 1;
                }
        }
        std::cout << " 最长连续递增子序列为: ";
        PrintSqList(L, final_start, final_end);
}
```

7.**(约瑟夫环问题)** 编号为 $1, 2, ..., n$ 的 n 个人按照顺时针方向围坐一圈。从第一个人开始按照顺时针方向从 1 开始报数，当报到指定的数 m 时，报 m 的人出列。再从他顺时针方向的下一位人开始重新从 1 开始报数，报到 m 的人出列，如此下去，直到所有人都出列。请设计算法用单向循环链表模拟约瑟夫环问题的出列过程，输出出列的顺序。

解.

用单向循环链表模拟如下：将报到 m 的人对应的编号记到一个数组里，然后将这个人从链表中删除。以此类推，直到链表中只剩最后一个人为止，输出最后这个人的编号，从而得到出列顺序。该算法的时间复杂度为 $O(n^2)$，空间复杂度为 $O(n)$（开了一个数组sequence[]）：

```
void JosephRing(int m, int n, LNode *ring){
        LNode *pos = new LNode;
        pos = ring->next;
        int member = n;
        int i = 0;

        if (m%n == 1){
                while (member > 0){
                        sequence[i]=pos->data;
```

```
                        member--;
                        i++;
                        pos = pos->next;
                }
        }
        else{
                while (member > 1){
                        for (int i = 0 ; i < m%n-2 ; i++){//找到出列人的前面一个人
                                pos = pos->next;
                        }
                        sequence[i]=(pos->next)->data;
                        LNode *temp = pos->next;
                        pos->next = temp->next;
                        free(temp);
                        member--;
                        i++;
                        pos = pos->next;
                }
                sequence[i]=(pos->next)->data;
        }

        std::cout << " 出列顺序为: " << "[";
        for (int i = 0 ; i < n-1 ; i++){
                std::cout << sequence[i] << " ";
        }
        std::cout << sequence[n-1] << "]" << "\n";
}
```

8. 假定采用带头结点的单链表保存单词，当两个单词有相同的后缀时，则可共享相同的后缀存储空间，例如，“loading”和“being”，如图所示。

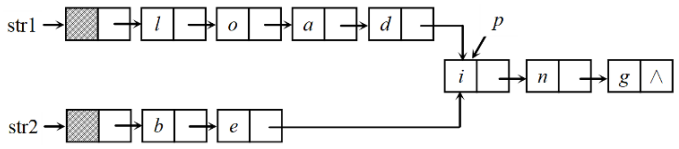

图 2.2　两个单词的后缀共享存储结构

设 str1 和 str2 分别指向两个单词所在单链表的头结点，链表结点结构为（data，next）。请设计一个时间上尽可能高效的算法，找出由 str1 和 str2 所指向两个链表共同后缀的起始位置（如图中字符 i 所在结点的位置 p）。

解.

首先，我们写一个函数int ListLength()获得单词str1，str2的长度，并用两个指针p，q分别指向str1，str2的首个字母。其次，我们将单词按尾部对齐，对于更长的单词（这里是str1），我们用指针p找到单词str1中的某个字母，使得该字母后str1和str2的长度相等（在这里，p指向“loading”这个单词的‘a’字母处）。最后，让指针p，q同时移动，直到两指针指向字母相等。此时指向的位置就是两单词相同后缀的第一个字母。该算法的时间复杂度为 $O(\max\{\text{Length(str1)}, \text{Length(str1)}\})$，空间复杂度为 $O(1)$，

```
int ListLength(CharacterNode *word){
        int length = 0;
        CharacterNode *p = word->next;
        while (p != NULL){
                length++;
                p = p->next;
        }
        return length;
}

CharacterNode* CommonSuffix(CharacterNode *str1, CharacterNode *str2){
        CharacterNode *p = str1->next;
        CharacterNode *q = str2->next;
        int str1_length = ListLength(str1);
        int str2_length = ListLength(str2);
        if (str1_length > str2_length){
                while (str1_length > str2_length){
                        p = p->next;
                        str1_length--;
                }
        }
        else{
                while (str2_length > str1_length){
                        q = q->next;
                        str2_length--;
                }
        }
        while (p != NULL && p->character != q->character){
                p = p->next;
                q = q->next;
        }
        std::cout << p->character;
        return p;
}
```