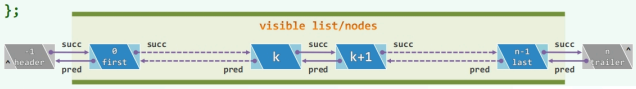
第2章 列表

List

```
#include "listNode.h" //引入列表节点类
template <typename T> class List { //列表模板类
   private:    Rank _size; ListNodePosi<T> header, trailer; //哨兵
               //头、首、末、尾节点的秩，可分别理解为-1、0、n-1、n
   protected:  /* ... 内部函数 */
   public:     /* ... 构造函数、析构函数、只读接口、可写接口、遍历接口 */
};
```



In [1]:
```
#include <iostream>
using namespace std;
```

一、节点定义

In [2]:
```
#define ListNodePointer(T) ListNode<T>*
template <class T>
struct ListNode{
    T data; ListNodePointer(T) pred; ListNodePointer(T) succ;
    ListNode(){}  // 什么也不做，可以创建一个空节点，针对后面列表的header和tailer的构造
    ListNode(T e, ListNodePointer(T) p = NULL, ListNodePointer(T) s = NULL);
    ListNodePointer(T) insertAsPred(const T e);
    ListNodePointer(T) insertAsSucc(const T e);
    T remove(ListNodePointer(T) p);
};
```

In [3]:
```
template <class T>
ListNode<T>::ListNode(T e, ListNodePointer(T) p , ListNodePointer(T) s ){
    data = e;
    pred = p;
    succ = s;
}
```

In [4]:
```
template <class T>
ListNodePointer(T) ListNode<T>::insertAsPred(const T e){
    ListNodePointer(T) x = new ListNode<T>(e,pred,this);
    pred -> succ = x;
    pred = x;
    return x;
}
```

In [5]:
```
template <class T>
ListNodePointer(T) ListNode<T>::insertAsSucc(const T e){
    ListNodePointer(T) x = new ListNode<T>(e,this,succ);
    succ -> pred = x;
    succ = x;
    return x;
}
```

In [6]:
```
template <class T>
T ListNode<T>::remove(ListNodePointer(T) p){
    T e = p -> data;
    p -> pred -> succ = p -> succ;
    p -> succ -> pred = p -> pred;
    delete p;
    return e;
}
```

任意初始化3个节点

In [7]:
```
auto node1 = ListNode<int>(5);
auto node2 = ListNode<int>(3);
auto node3 = ListNode<int>(8);
```

二、列表对象定义

In [8]:
```
template <class T>
class List{
```

```
    public:
        // 基本属性
        int size; ListNodePointer(T) header; ListNodePointer(T) tailer;

        // 构造和析构
        List();                   // 初始化一个空列表
        List(const List<T>& L);         // 从另一个列表复制

        // 插入删除
        ListNodePointer(T) insertA(ListNodePointer(T) p, T e);
        ListNodePointer(T) insertB(ListNodePointer(T) p, T e);
        T remove(ListNodePointer(T) p);

        // 访问
        T& getAndPut(int r);

        // 查找排序
        ListNodePointer(T) search(T e, int n, ListNodePointer(T) p);
        void insertSort(ListNodePointer(T) p, int n);        // 插入排序
        ListNodePointer(T) selectMax(ListNodePointer(T) p, int n);
        void selectSort(ListNodePointer(T) p, int n);        // 选择排序

        // 补充
        bool empty(){return !size;}
};
```

In [9]:
```
template <class T>
List<T>::List(){
    size = 0;
    header = new ListNode<T>;
    tailer = new ListNode<T>;
    header -> pred = NULL;
    header -> succ = tailer;
    tailer -> pred = header;
    tailer -> succ = NULL;
}
```

In [10]:
```
template <class T>
List<T>::List(const List<T>& L){
    size = L.size;
    ListNodePointer(T) p1 = L.header -> succ;
    ListNodePointer(T) p2 = header;
    for (int n = 0; n < size; n++){
        p2 -> insertAsSucc(p1 -> data);
        p2 = p2 -> succ;
        p1 = p1 -> succ;
    }
}
```

初始化一个空列表

1. 由于没有现成的列表，所以不可能调用列表复制；
2. 由于列表的插入算法还没写，所以空列表无法加入新节点；
3. 虽然可以调用header节点的插入算法来增加新节点，但是我们不这么做；

In [11]:
```
auto L1 = List<int>();
```

三、插入删除

插入：输入节点p和数值e，把e插入p的前驱或后继，返回插入后的节点；
删除：给定节点p，删除并返回其数值
因为哨兵节点的存在，所以插入删除算法会很安全，不会出现问题

In [12]:
```
template <class T>
ListNodePointer(T) List<T>::insertA(ListNodePointer(T) p, T e){
    size++;
    return p -> insertAsSucc(e);

}
```

In [13]:
```
template <class T>
ListNodePointer(T) List<T>::insertB(ListNodePointer(T) p, T e){
    size++;
    return p -> insertAsPred(e);
}
```

In [14]:
```
template <class T>
T List<T>::remove(ListNodePointer(T) p){
    size--;
    return p -> remove(p);
}
```

不停在L1的tailer节点前插入元素

In [15]:
```
L1.insertB(L1.tailer, 3);
L1.insertB(L1.tailer, 8);
L1.insertB(L1.tailer, 2);
L1.insertB(L1.tailer, 9);
L1.insertB(L1.tailer, 4);
```

列表销毁
列表的析构函数就是不停调用remove删除header的后继或者tailer的前驱，
最后删除header和tailer，这里就不详细说明了

四、call by link

由于重载操作符[]的话，在notebook里面只能写在对象定义里面，不便于这里讲解，
我们把这个运算符改名为getAndPut

In [16]:
```
template <class T>
T& List<T>::getAndPut(int r){
    ListNodePointer(T) p = header;
    for(int i = 0; i <= r; i++) p = p -> succ;
    return p -> data;
}
```

In [17]:
```
for(int i=0; i < L1.size; i++)
    cout << L1.getAndPut(i) << endl;
```

```
3
8
2
9
4
```

In [18]:
```
L1.getAndPut(3) = 100;
for(int i=0; i < L1.size; i++)
    cout << L1.getAndPut(i) << endl;
```

```
3
8
2
100
4
```

五、查找排序

查找
由于无法在O(1)的时间直接访问到中间元素，所以二分查找在这里没有用，
直接顺序查找，此处查找算法的定义是从p节点开始的n个节点中查找不小于e的第一个节点，
在插入排序中刚好插入这个节点的前驱中，类似打扑克时的插牌

In [19]:
```
template <class T>
ListNodePointer(T) List<T>::search(T e, int n, ListNodePointer(T) p){
    while(n > 0){
        if (p -> data >= e)
            return p;
        n--;
        p = p -> succ;
    }
    return p;
}
```

In [20]:
```
L1.search(5,3,L1.header) -> data  // 5在3和8之间，找到的节点一定是8
```

Out[20]:
```
8
```

插入排序
前缀序列从小到大有序，后缀序列无序，从后缀序列中拿出第一个元素插入前缀序列中
给定节点p，对从p开始的n个节点进行排序

In [21]:
```
template <class T>
void List<T>::insertSort(ListNodePointer(T) p, int n){
    ListNodePointer(T) p1 = p;
    ListNodePointer(T) p2 = p -> succ;
```

```
    ListNodePointer(T) temp;
    for (int r=1; r < n; r++) {
        temp = search(p2 -> data, r, p1);
        insertB(temp,p2 -> data);
        p2 = p2 -> succ;
        remove(p2 -> pred);
    }
}
```

In [22]:
```
L1.insertSort(L1.header -> succ,5);
```

In [23]:
```
for(int i=0; i < L1.size; i++)
    cout << L1.getAndPut(i) << endl;
```

```
2
3
4
8
100
```

选择排序

前缀序列无序，后缀序列有序，从前缀序列中找到最大元素，插入后缀序列的头

In [24]:
```
template <class T>
ListNodePointer(T) List<T>::selectMax(ListNodePointer(T) p, int n){
    ListNodePointer(T) p1 = p -> succ;
    ListNodePointer(T) max = p;
    while (n > 1) {
        if ((p1 -> data) > (max -> data)) {
            max = p1;
            p1 = p1 -> succ;
        }
        else {
            p1 = p1 -> succ;
        }
        n--;
    }
    return max;
}
```

In [25]:
```
template <class T>
void List<T>::selectSort(ListNodePointer(T) p, int n){
    ListNodePointer(T) p1 = p;
    ListNodePointer(T) p2 = tailer;
    ListNodePointer(T) temp;
    while (n > 0) {
        temp = selectMax(p1,n);
        insertB(p2,temp -> data);
        p2 = p2 -> pred;
        remove(temp);
        n--;
    }
}
```

In [26]:
```
auto L2 = List<int>();
L2.insertB(L2.tailer, 3);
L2.insertB(L2.tailer, 8);
L2.insertB(L2.tailer, 2);
L2.insertB(L2.tailer, 9);
L2.insertB(L2.tailer, 4);
for(int i=0; i < L2.size; i++)
    cout << L2.getAndPut(i) << endl;
```

```
3
8
2
9
4
```

In [27]:
```
L2.selectSort(L2.header -> succ,5);
for(int i=0; i < L2.size; i++)
    cout << L2.getAndPut(i) << endl;
```

```
2
3
4
8
9
```

In [ ]:
```
```