作业四

1. 现有中缀表达式 E=((A-B)/C+D*(E-F))*G　（注：此题在纸上练习，不用提交）

（1）写出与 E 等价的后缀表达式。

（2）用一个操作符栈来模拟表达式的转换过程，画出在将 E 转换成后缀表达式的过程中，栈内容的变化图。

（3）用一个操作数栈来模拟后缀表达式的求值过程，画出对（2）中所得到的后缀表达式求值时，栈中内容的变化图。

2. 假设以带头结点的循环链表表示队列，并且只设一个表尾指针，试编写相应的置队列空、入队和出队操作。

```
template <class T>
void LinkQueue<T>::EnQueue(T x)
{
    s = new Node<T>;
    s->data = x;
    s->next = rear->next;
    rear->next = s;
    rear = s;
}

template <class T>
T LinkQueue<T>::DeQueue()
{
    if (rear == rear->next) { cerr << "下溢"; exit(1); }
    p = rear->next->next;
    x = p->data;
    rear->next->next = p->next;
    if (p->next == rear->next) rear = rear->next;
    delete p;
    return x;
}

template <class T>
void LinkQueue<T>::ClearQueue()
{
    while (rear->next != rear)
    {
        p = rear->next;
        rear = p->next;
        delete p;
        p = rear;
    }
}
```

3. 假设以一维数组 data[m]存储循环队列的元素，若要使这 m 个分量都得到应用，则另设一辅助标志变量 flag 判断队列的状态为“空”还是“满”。编写入队和出队算法。

```
(bool flag;)
```

```
template <class T, int MaxSize>
void SeqQueue<T, MaxSize>::EnQueue(T x)
{
    if (flag)
    {
        cerr << "上溢"; exit(1);
    }
    rear = (rear + 1) % MaxSize;
    if (rear == front)
        flag = 1;
    data[rear] = x;
}

template <class T, int MaxSize>
T SeqQueue<T, MaxSize>:: DeQueue()
{
    if (!flag)
    {
        cerr << "下溢"; exit(1);
    }
    front = (front + 1) % MaxSize;
    if (front == rear)
        flag = 0;
    return data[front];
}
```

4. 假设以一维数组 data[m]存放循环队列的元素，同时设变量 num 表示当前队列中元素的个数，以判断队列的状态为“空”还是“满”。试给出此循环队列满的条件，并编写入队和出队算法。

队满条件：num==MaxSize

```
template <class T, int MaxSize>
void SeqQueue<T, MaxSize>::EnQueue(T x)
{
    num = (rear - front + MaxSize)  % MaxSize;
    if (num == MaxSize)
    {
        cerr << "上溢"; exit(1);
    }
    rear = (rear + 1) % MaxSize;
    data[rear] = x;
}
template <class T, int MaxSize>
T SeqQueue<T, MaxSize>::DeQueue()
{
    num = (rear - front + MaxSize)  % MaxSize;
    if (num == 0)
    {
        cerr << "下溢"; exit(1);
    }
    front = (front + 1) % MaxSize;
    return data[front];
    }
```

5. 如何用两个栈来实现队列？并写出队列基本操作的算法。

```
template <class T>
class StackQueue<T>
{
public:
    void  EnQueue(T);
    T  DeQueue();
private:
    stack<T> inStack;
    stack<T> outStack;
};

template <class T>
void StackQueue<T>::EnQueue(T x)
{
    inStack.push(x);
}

template <class T>
T StackQueue<T>::DeQueue()
{
    T tmp;
    if (outStack.size() > 0)
    {
        tmp = outStack.top();
        outStack.pop();
    }
    else
    {
        if (inStack.size() > 0)
        {
            int size = inStack.size();
            for (int i = 0; i < size; i++)
            {
                outStack.push(inStack.top());
                inStack.pop();
            }
            tmp = outStack.top();
            outStack.pop();
        }
         else
        {
            cerr << "下溢"; exit(1);
        }
    }
    return tmp;
}
```