**第3章习题**

**3.1对一个栈的输入序列a1,a2,a3,… ,an,，称由此栈依次出栈后所得到的元素序列为栈的合法输出序列。例如，假设栈S的一个输入序列为1,2,3,4,5，则可得到多个输出序列，例如，1,2,3,4,5就是一个合法的输出序列，同理，5,4,3,2,1和3,2,1,4,5也分别是其合法的输出序列。分别求解下列问题：  
（1）判断序列1,3,4,5,2是否是合法的输出序列。  
（2）对输入序列1,2,3,4,5，求出其所有的合法的输出序列。  
（3\*）设计算法以判断对输入序列1,2,3,… ,n，序列a1,a2,a3,… ,an是否是该栈的合法的输出序列（假设输出序列在数组A中）。**

void legalSequence(Stack<>)

void legalSequence(Stack<int,100>S,elementType In[],elementType Out[],int lebn,int i,int j)

{

//In[]入栈序列，也可以用栈来保存，但要注意顺序。还可用 C++的 vector 保存。

//Out[]出栈序列，也可以用栈来保存，但要注意输出循序。还可用 C++的 vector保存。

//len 序列长度

//i 入栈序列元素指针，指示当前处理的入栈序列元素

//j 出栈序列元素指针，指示当前获取的出栈序列元素

//每次操作有可能有 2 中操作，要么出栈，要么入栈，且 2 种操作是或的关系。但，

//出栈、入栈递归返回后要恢复递归前状态

elementType x;

if(S.empty()&&j>=len)//递归出口，获得了一个出栈序列

{

seqNum++; //序列加1

seqPrint(Out,len);//打印序列

}

else if(!S.empty()&&i<len)//栈不空，入栈序列中还有数据

{

//选择出栈

S.getTop(x);

S.pop();

Out[j]=x;

j++;

legelSequence(S,In,Out,i,j);

j--;//递归返回，恢复到出栈钱的状态

S.push(x);

//选择入栈

S.push(In[i]);

i++;

legalSequence(S,In,Out,i,j);

i--;

S.pop();

}

else if(!S.empty()&&i>=len)//栈不空，入栈序列数据已经处理结束

{

//此时只能拿选择出栈操作

S.getTop(x);

S.pop();

Out[j]=x;

j++;

legelSequence(S,In,Out,i,j);

j--;//恢复到出栈钱的状态

S.push(x);

}

else if(S.empty()&&i<len)//栈空，入栈序列未处理结束

{

//此时，只能选择入栈操作

S.push(In[i]);

i++;

legalSequence(S,In,Out,i,j);

i--; //恢复到入栈前的状态

S.pop();

}

**}**

**（**3）用数组 In[]保存入栈序列，Out[]保存输出序列，分别用指针 i，j 指示。用一个

栈模拟进出站的操作。对每个输出元素 Out[j]做如下操作：

如果 In[i]<Out[j]，将 In[i]入栈，指针 i 后移，即 i++；

如果 In[i]==Out[j]，2 个指针同时后移，即 i++，j++；

如果栈顶元素==Out[j]，栈顶元素出栈，j 后移，即 j++；

循环结束后，如果栈空，且入栈序列处理完毕，则输出序列是一个合法的出栈序列，否

则不是bool legalSequence(elementType In[], elementType Out[], int len)

{

//In[]保存入栈序列

//Out[]保存输出序列

int i=0,j=0; //i 为入栈序列指针；j 为输出序列指针

elementType x;

Stack<int,100> S; //初始化顺序栈

while(j<len)

{

S.getTop(x);。if(In[i]<Out[j] && i<len) //入栈序列当前元素小于输出序列当前元素

{

S.push(In[i]); //输入序列当前元素入栈

i++; //入栈序列指针后移

}

else if(In[i]==Out[j]) //入栈序列当前元素等于输出序列当前元素

{

i++; //指针同时后移

j++;

}

else if(x==Out[j]) //栈顶元素等于输出序列当前元素

{

S.pop(); //出栈

j++; //输出序列指针后移

}

else

break;

}

if(S.empty() && i>=len) //栈空，且入栈序列处理完毕，则输出序列为合法出

栈序列，返回 true

return true;

else

return false;

}

**3.4对一个合法的数学表达式来说，其中的各大小括号“{”，“}”，“[”，“]”，“(”和“)”应是相互匹配的。设计算法对以字符串形式读入的表达式S，判断其中的各括号是否是匹配的。**

bool bracketMatch2(string str)

{

Stack<char,100> S; //初始化一个字符串顺序栈

int len;

int i=0;

int result=false;

char x; //保存栈顶字符（某种左括号）

bool tag=true;

len=str.length(); //求字符串长度

while(i<len && tag==true)

{

switch(str[i])

{

case '(':

case '[':

case '{':

S.push(str[i]);

break;

case ')':

if(S.empty())

{

tag=false;

result=false;

break;

}

S.getTop(x); //取当前栈顶

if(x=='(')

{

S.pop();

break;

}

else

{

tag=false;

result=false;

break;

}

case ']':

if(S.empty())

{

tag=false;

result=false;

break;

}

S.getTop(x); //取当前栈顶

if(x=='[')

{

S.pop();

break;

}

else

{

tag=false;

result=false;

break;

}

case '}':

if(S.empty())

{

tag=false;

result=false;

break;

}

S.getTop(x); //取当前栈顶

if(x=='{')

{

S.pop();

break;

}

else

{

tag=false;

result=false;

break;

}

default:

break;

}

i++;

}

if(S.empty() && tag==true)

{

result=true;

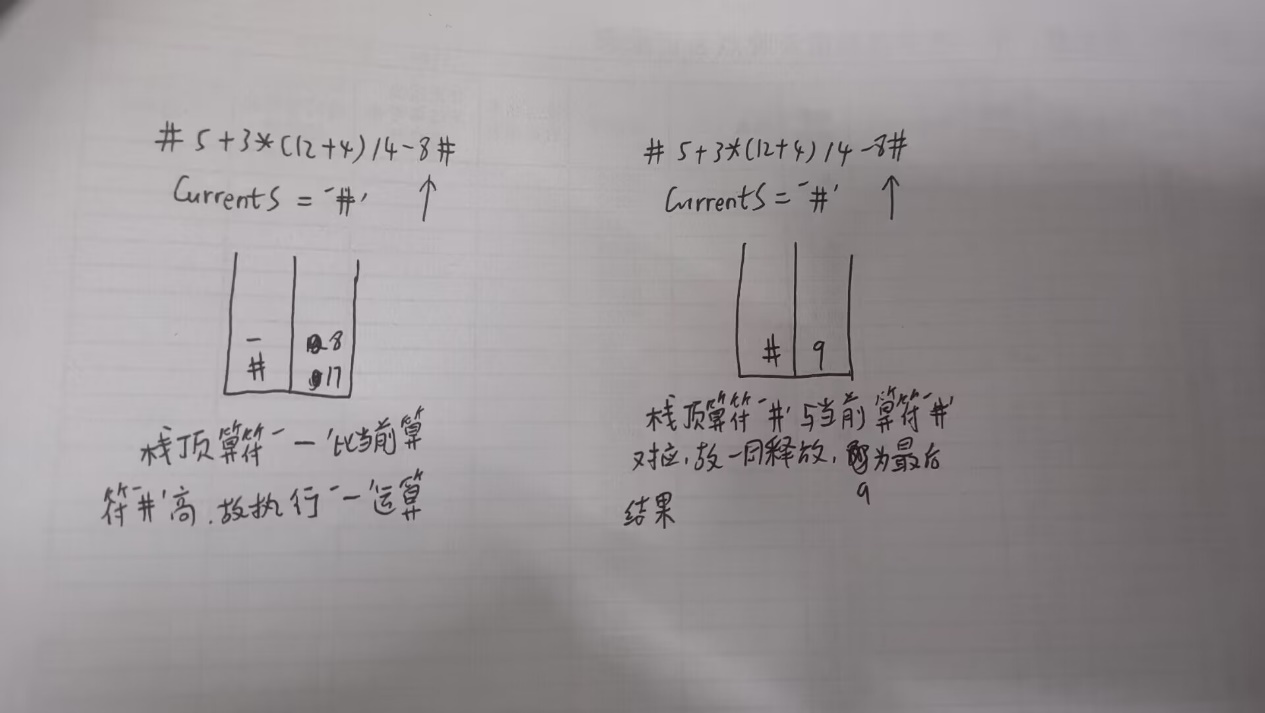
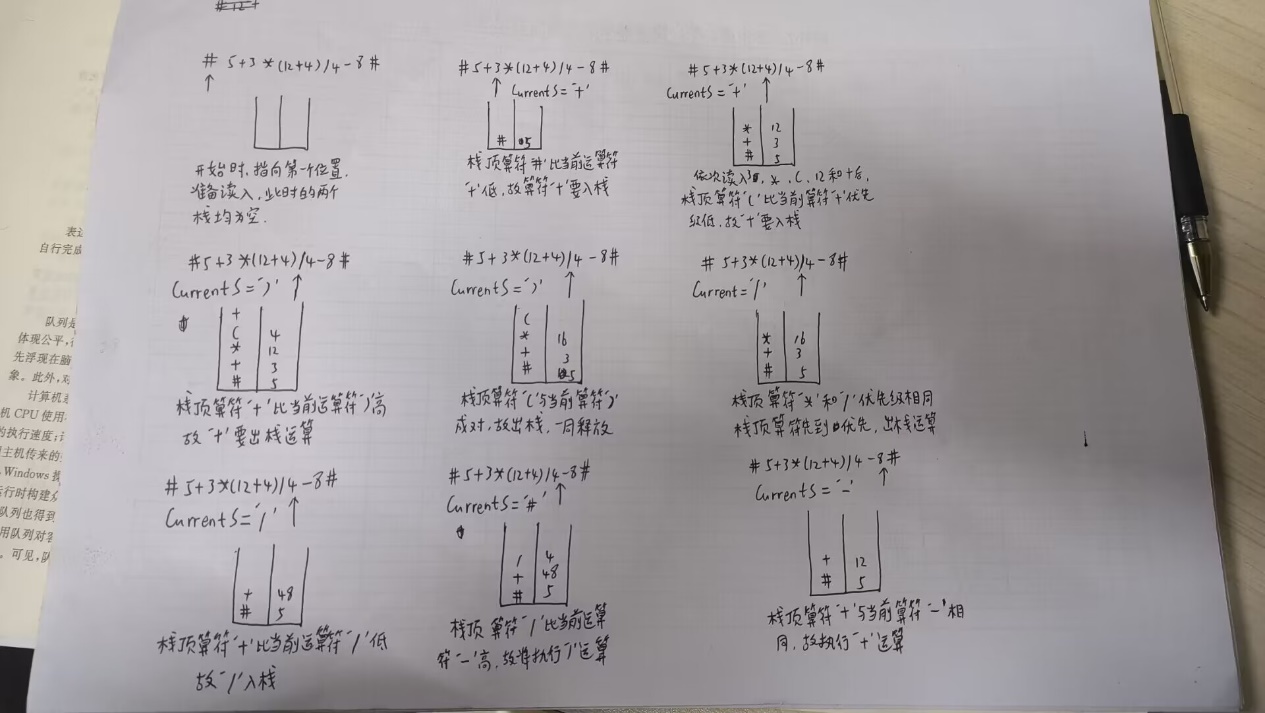
}

else

result=false;

return result;

} **3.5对表达式5+3\*(12+4)/4-8，依次画出在求解过程中的各步骤中的栈的状态。**



**3.9对教材中所讨论的循环队列及其约定，给出求解队列中元素个数的表达式。**

设循环队列为 Q，数组最大长度为 MaxLen，头指针和尾指针分别为 front 和 rear 元素个数=(Q.rear-Q.front+MaxLen) % MaxLen

**3.11如果采用带尾指针的单循环链表（带头结点）作为队列的存储结构，设计算法以实现队列的各运算。**

//带尾指针单循环链表实现链队列，以下 Q 为单循环链表尾指针

#include "..\scrLinkedList.h" //带尾指针单循环链表

typedef linkedList linkedQueue;

//--1.链队初始化

void initialQueue(linkedQueue &Q)

{

Q=new node; //产生头结点，指针为 R;

Q->next=Q;

}

//--2.判断队空

bool queueEmpty(linkedQueue &Q)

{

return (Q==Q->next);

}

//--3.取队头元素

bool getFront(linkedQueue &Q, elementType &x)

{

if(queueEmpty(Q))

return false; //空队列，无法取队头元素

else

{

x=(Q->next->next)->data;

return true;

}

}

//--4.入队

void enQueue(linkedQueue &Q, elementType x)

{

node\* p=new node; //申请内存，产生新节点

p->data=x;

p->next=Q->next;

Q->next=p;

Q=p;

}

//--5.出队

bool outQueue(linkedQueue &Q, elementType &x)

{

node\* u; //用以指向删除节点

if(queueEmpty(Q))

return false; //队空，无法执行出队操作

else

{

x=Q->next->next->data; //取出队头元素

u=Q->next->next; //u 指向队头

Q->next->next=u->next;

if(Q==u) //删除最后一个节点，成为空队列

Q=Q->next;

delete u; //删除原队头，释放内存

return true;

}

}