栈

LIFO 后进先出

顺序栈 ① typedef struct stack{
T elem[MAX_SIZE];
int top;
} Stack;

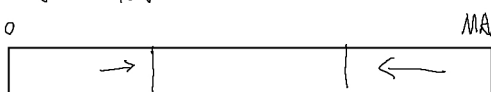
入栈 S.elem[++S.top] = value;
出栈 value = S.elem[S.top--];
栈空 S.top == -1
初始化 S.top = -1;

② typedef struct stack{
T *base;
T *top; //top 指向栈顶下一个位置
int size;
} Stack;

入栈 *S.top++ = value;
出栈 value = *--S.top;
栈空 S.top == S.base
初始化 S.top = S.base;

链栈 typedef struct SNode {
ElemType elem;
struct SNode *next;
} SNode, *LinkStack;
//只是栈结点受限，其余与链表一致

共享栈



0号栈 入栈 S.elem[++S.to] = e;
出栈 e = S.elem[S.to--];
初始化 S.to = -1;

1号栈 入栈 S.elem[--S.t1] = e;
出栈 e = S.elem[S.t1++];
初始化 S.t1 = MAX

栈满 S.t1 - S.to == 1

栈的应用

括号匹配

思路 顺序遍历一组括号
如果当前括号能与栈顶括号匹配，
则栈顶括号出栈，遍历下一个括号；
如不能匹配，则入栈，遍历下一个；
如果遍历完成后，栈空则全匹配

表达式求值（后缀表达式求值）

将中缀表达式转化为后缀表达式

例 ABCD-*+EF/-

当前元素	步骤	栈
		A
A	入栈	AB
B	入栈	ABC
C	入栈	ABCD
D	入栈	
-	D出栈，C出栈，计算C-D，结果为R_1入栈	ABR_1
*	R_1出栈，B出栈，计算B*R_1，结果为R_2入栈	AR_2
+	R_2出栈，A出栈，计算A+R_2，结果为R_3入栈	R_3
E	入栈	R_3E
F	入栈	R_3EF
/	F出栈，E出栈，计算E/F，结果为R_4入栈	R_3R_4
-	R_4出栈，R_3出栈，计算R_3-R_4，结果为R_5入栈	R_5

中缀转后缀表达式（比较运算符优先级）

例 A+B*((-D)-E/F

当前元素	步骤	栈	输出
A	输出	#	A
+	+># 入栈	#+	
B	输出		AB
*	*>+ 入栈	#+*	
(	(>* 入栈	#+*(	
(	输出		ABC
-	->(入栈	#+*(-	
D	输出		ABCD
)	)<- 出栈输出	#+*(	ABCD-
)	)=(出栈后移	#+*	
-	-<* 出栈输出	#+	ABCD-*
-	-<+ 出栈输出	#	ABCD-*+
-	-># 入栈	#-	
E	输出		ABCD-*+E
/	/>- 入栈	#-/	
F	输出		ABCD-*+EF
	出栈输出	#-	ABCD-*+EF/
	出栈输出	#	ABCD-*+EF/-

栈没中使用了栈