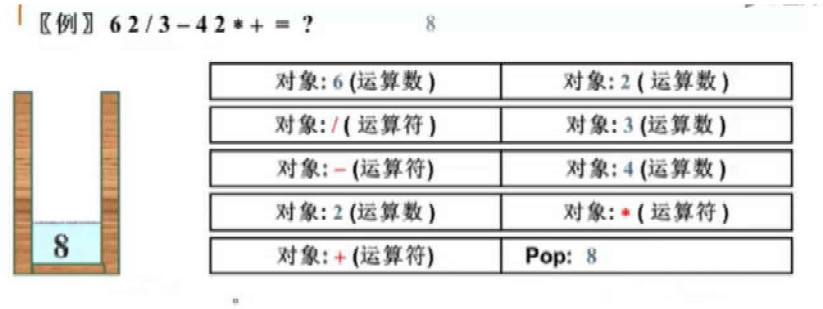
栈

2019年6月25日 20:05

后缀表达式：
6 2 / 3-4 2 * + =8
后缀表达式求值策略：从左向右“扫描”，逐个处理运算数和运算符号
1. 遇到运算数怎么办？如何“记住”目前还不未参与运算的数？
2. 遇到运算符号怎么办？对应的运算数是什么？
启示：需要有种存储方法，能顺序存储运算数， 并在需要时“倒序”输出！
(先进后出)将运算数入栈



堆栈（Stack）：具有一定操作约束的线性表 · 只在一端（栈顶，Top）做 插入、删除

插入数据：入栈（Push） · 删除数据：出栈（Pop） · 后入先出：Last In First Out（LIFO）

抽象数据类型描述
类型名称: 堆栈（Stack）
数据对象集：一个有0个或多个元素的有穷线性表。 操作集：长度为MaxSize的堆栈S · Stack，堆栈元素item · ElementType
1、Stack CreateStack(int MaxSize)： 生成空堆栈，其最大长度为MaxSize；
2、int IsFull(Stack S, int MaxSize)：判断堆栈S是否已满；
3、void Push(Stack S, ElementType item)：将元素item压入堆栈；
4、int IsEmpty (Stack S)：判断堆栈S是否为空；
5、ElementType Pop(Stack S)：删除并返回栈顶元素；

Push 和 Pop 可以穿插交替进行：

按照操作系列

(1)Push(S,A), Push(S,B),Push((S,C),Pop(S),Pop(S),Pop(S)

堆栈输出是？ CBA

(2) 而Push(S,A), Pop(S),Push(S,B),Push((S,C),Pop(S),Pop(S)

堆栈输出是？ ACB

[例] 如果三个字符按ABC顺序压入堆栈

- ABC的所有排列都可能是出栈的序列吗？
- 可以产生CAB这样的序列吗？

栈的顺序存储

栈的顺序存储结构通常由一个一维数组和一个记录 栈顶元素位置的变量组成。

```
#define MaxSize <储存数据元素的最大个数>
typedef struct SNode* Stack;
struct SNode {
        ElementType Data[MaxSize];
        int Top;//栈顶
};
```

入栈：

```
void Push(Stack PtrS, ElementType item)
{
        if (PtrS->Top == MaxSize - 1) {
                printf("堆栈满"); return;
        }
        else {
                PtrS->Data[++(PtrS->Top)] = item;//把item放在top+1的位置，同时将top+1
                return;
        }
}
```

出栈：

```
ElementType Pop(Stack PtrS)
{
        if (PtrS->Top == -1) {
                printf("堆栈空");
                return ERROR; /* ERROR是ElementType的特殊值，标志错误
          */
        }
        else
                return (PtrS->Data[(PtrS->Top)--]);
}
```

实例：

请用一个数组实现两个堆栈，要求最大地利用数组空间，使 数组只要有空间入栈操作就可以成功。

【分析】 一种比较聪明的方法是使这两个栈分别从数组的两头开始 向中间生长；当两个栈的栈顶指针相遇时，表示两个栈都满了。

【栈】

```
#define MaxSize <存储数据元素的最大个数>
struct DStack {
        ElementType Data[MaxSize];
        int Top1; /* 堆栈 1 的栈顶指针 */
        int Top2; /* 堆栈 2 的栈顶指针 */
} S;
S.Top1 = -1;
S.Top2 = MaxSize;//2 为空
```

具体操作

```
void Push(struct DStack* PtrS, ElementType item, int Tag)//tag=1第一个
{ /* Tag作为区分两个堆栈的标志，取值为1和2 */
        if (PtrS->Top2 – PtrS->Top1 == 1) { /*堆栈满*/
                printf(“堆栈满”); return;
        }
        if (Tag == 1) /* 对第一个堆栈操作 */
                PtrS->Data[++(PtrS->Top1)] = item;
        else /* 对第二个堆栈操作 */
                PtrS->Data[--(PtrS->Top2)] = item;
}

ElementType Pop(struct DStack* PtrS, int Tag)
{ /* Tag作为区分两个堆栈的标志，取值为1和2 */
        if (Tag == 1) { /* 对第一个堆栈操作 */
                if (PtrS->Top1 == -1) { /*堆栈1空 */
                        printf(“堆栈1空”); return NULL;
                }
                else return PtrS->Data[(PtrS->Top1)--];
        }
        else { /* 对第二个堆栈操作 */
                if (PtrS->Top2 == MaxSize) { /*堆栈2空 */
                        printf(“堆栈2空”); return NULL;
                }
                else return PtrS->Data[(PtrS->Top2)++];
        }
}
```

栈的链式存储

栈的链式存储结构实际上就是一个单链表，叫做链栈。插入和删 除操作只能在链栈的栈顶进行。

栈顶指针Top应该在链表的哪一头：在表头

结构：

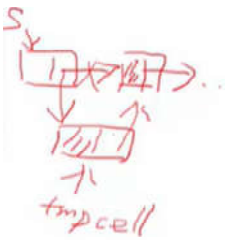
```
typedef struct SNode* Stack;
struct SNode {
        ElementType Data;
```

```
        struct SNode* Next;
};
```

创建堆栈头结点

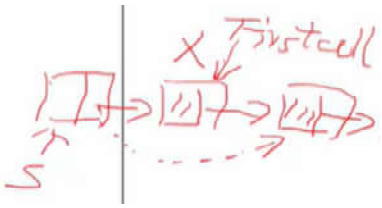
```
Stack CreateStack()
{ /* 构建一个堆栈的头结点，返回指针 */
        Stack S;
        S = (Stack)malloc(sizeof(struct SNode));
        S->Next = NULL;
        return S;
}
int IsEmpty(Stack S)
{ /*判断堆栈S是否为空，若为空函数返回整数1，否
则返回0 */
        return (S->Next == NULL);
}
```

push操作：链表就不存在需要判断栈满不满了




```
void Push(ElementType item, Stack S)
{ /* 将元素item压入堆栈S */
        struct SNode* TmpCell;
        TmpCell = (struct SNode*)malloc(sizeof(struct SNode));
        TmpCell->Element = item;
        TmpCell->Next = S->Next;
        S->Next = TmpCell;
}
```

pop操作：




```
ElementType Pop(Stack S)
{ /* 删除并返回堆栈S的栈顶元素 */
        struct SNode* FirstCell;
        ElementType TopElem;
        if (IsEmpty(S)) {
                printf("堆栈空"); return NULL;
        }
        else {
                FirstCell = S->Next;
                S->Next = FirstCell->Next;
                TopElem = FirstCell->Element;
                free(FirstCell);
                return TopElem;
        }
}
```

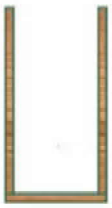
栈的运用：中缀表达式求值

先把中转后，再求值

运算符号的存储，优先级

表达式转换：将运算符入栈

〖例〗 $a*(b+c)/d=?$ $abc+*d/$



输入对象:a (操作数)	输入对象:* (乘法)
输入对象:((左括号)	输入对象:b (操作数)
输入对象:+ (加法)	输入对象:c (操作数)
输入对象:) (右括号)	输入对象:/ (除法)
输入对象:d (操作数)	

$T(N)=O(N)$

· 从头到尾读取中缀表达式的每个对象，对不同对象按不同的情况处理。

① 运算数：直接输出；

② 左括号：压入堆栈；

③ 右括号：将栈顶的运算符弹出并输出，直到遇到左括号（出栈，不输出）；

④ 运算符： • 若优先级大于栈顶运算符时，则把它压栈； • 若优先级小于等于栈顶运算符时，将栈顶运算符弹出并输出；再比 较新的栈顶运算符，直到该运算符大于栈顶运算符优先级为止，然 后将该运算符压栈；

⑤ 若各对象处理完毕，则把堆栈中存留的运算符一并输出。

❖中缀转换为后缀示例： （ 2*（9+6/3-5）+4）

步骤	待处理表达式	堆栈状态 （底←→顶）	输出状态
1	2*（9+6/3-5）+4		
2	*（9+6/3-5）+4		2
3	（9+6/3-5）+4	*	2
4	9+6/3-5）+4	*（	2
5	+6/3-5）+4	*（	2 9
6	6/3-5）+4	*（+	2 9
7	/3-5）+4	*（+	2 9 6
8	3-5）+4	*（+/	2 9 6
9	-5）+4	*（+/	2 9 6 3
10	5）+4	*（-	2 9 6 3 / +
11	）+4	*（-	2 9 6 3 / + 5
12	+4	*	2 9 6 3 / + 5 -
13	4	+	2 9 6 3 / + 5 - *
14		+	2 9 6 3 / + 5 - * 4
15			2 9 6 3 / + 5 - * 4 +