2.2 堆栈

什么是堆栈

计算机如何进行表达式求值？

[例] 算术表达式5+6/2-3*4。正确理解：

5+6/2-3*4 = 5+3-3*4 = 8-3*4 = 8-12 = -4

- 由两类对象构成的：
  - 运算数，如2、3、4
  - 运算符号，如+、-、*、/
- 不同运算符号优先级不一样

后缀表达式

- 中缀表达式：运算符号位于两个运算数之间。如，$a+b*c-d/e$
- 后缀表达式：运算符号位于两个运算数之后。如，$abc*+de/-$

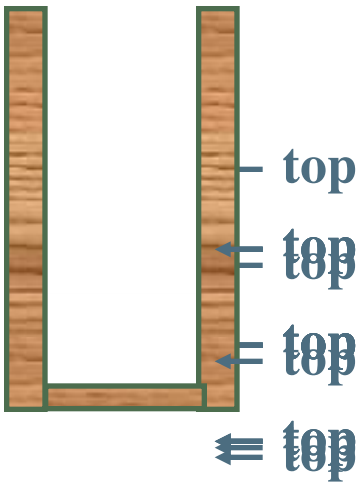
〖例〗$62/3-42*+=?$

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！

〖例〗 6 2 / 3 – 4 2 * + = ?　　8



对象: 6 (运算数)	对象: 2 (运算数)
对象: / (运算符)	对象: 3 (运算数)
对象: – (运算符)	对象: 4 (运算数)
对象: 2 (运算数)	对象: * (运算符)
对象: + (运算符)	Pop: 8

$T(N) = O(N)$

堆栈的抽象数据类型描述

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）

堆栈的抽象数据类型描述

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

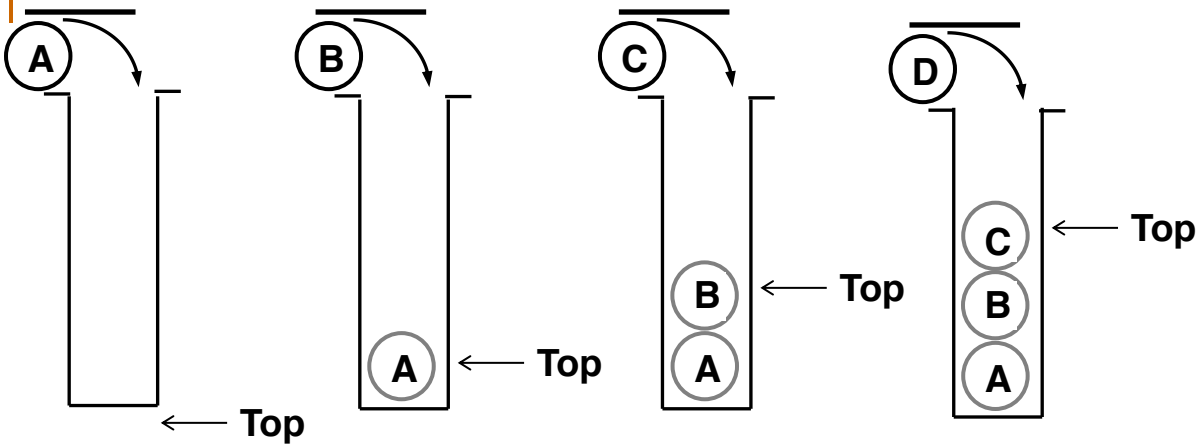
1、Stack CreateStack(int MaxSize)： 生成空堆栈，其最大长度为MaxSize；

2、int IsFull(Stack S, int MaxSize)：判断堆栈S是否已满；

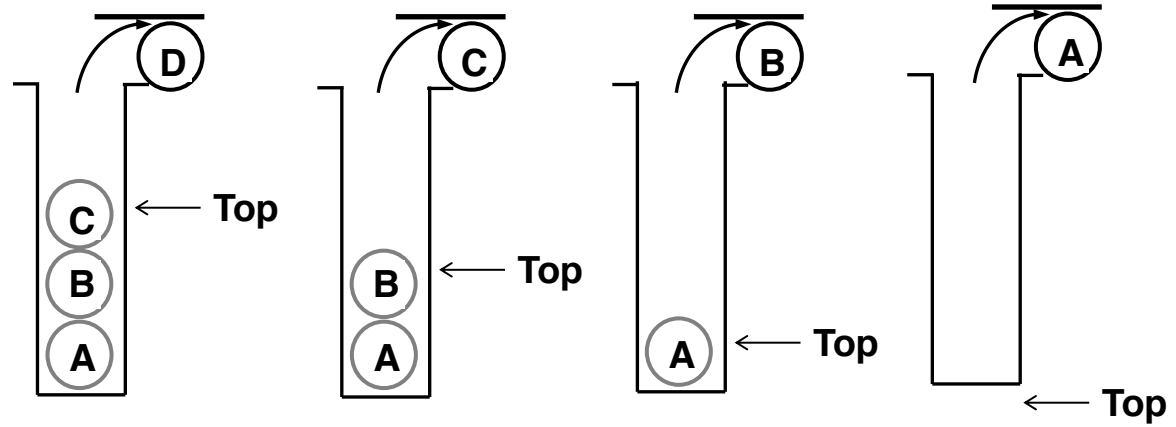
3、void Push(Stack S, ElementType item)：将元素item压入堆栈；

4、int IsEmpty (Stack S)：判断堆栈S是否为空；

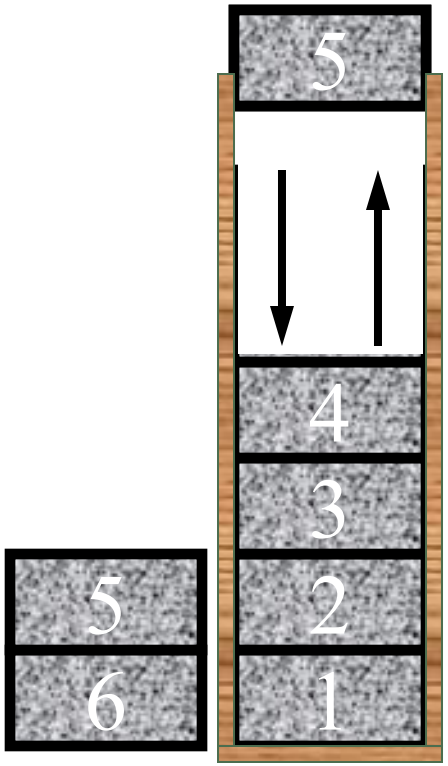
5、ElementType Pop(Stack S)：删除并返回栈顶元素；



CreatStack( ); Push(S,A); Push(S,B); Push(S,C);



x=Pop(S); x=Pop(S); x=Pop(S); IsEmpty (S)



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

压栈的顺序必须是ABC，但是可以压栈后在没有完全进栈就进行出栈

- 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？

栈的顺序存储实现

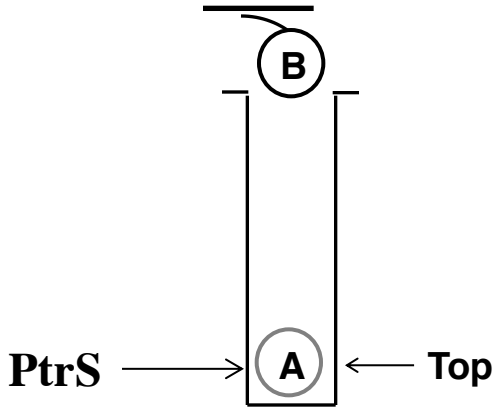
栈的顺序存储结构通常由一个一维数组和一个记录栈顶元素位置的变量组成。

正确理解这句话，只有“一个数组”，也就是说，这个堆栈只有一个用于存储数据的数组，比如下面定义的 SNode结构体，在使用中只声明一个 SNode的结构体对象，而后所有栈中的数据都是通过操作这个结构体对象中的Data[]数组进行进栈-出栈操作

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

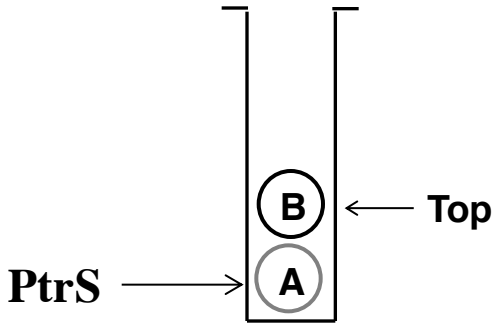
(1) 入栈

```
void Push( Stack PtrS, ElementType item )
{
    if ( PtrS->Top == MaxSize-1 ) {
       printf(“堆栈满”);  return;
    }else {
       PtrS->Data[++(PtrS->Top)] = item;
       return;
    }
}
```



(2)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)--] );
}
```



[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。

```
#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;
```

```
void Push( struct DStack *PtrS, ElementType item, int Tag )
{  /* 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;
}
```

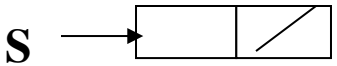
```
ElementType Pop( struct DStack *PtrS, int Tag )
{  /* Tag作为区分两个堆栈的标志，取值为1和2   */
    if ( Tag == 1 ) {  /* 对第一个堆栈操作   */
      if ( PtrS->Top1 == -1 ) { /*堆栈1空 */
         printf("堆栈1空");    return NULL;
      } else return PtrS->Data[(PtrS->Top1)--];
   } else {              /* 对第二个堆栈操作 */
    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

堆栈的链式存储实现

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？

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

(1) 堆栈初始化（建立空栈）
(2) 判断堆栈S是否为空




```
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 );
}
```

```
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;
}
```

```
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;
   }
}
```

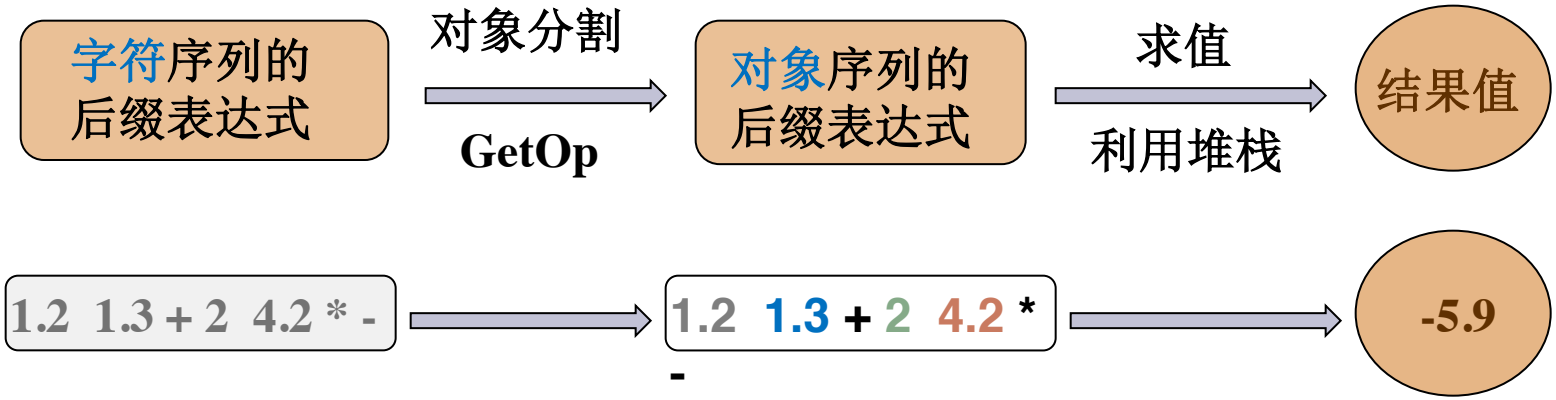
浙江大学计算机科学与技术学院

堆栈应用：表达式求值

➢ 回忆：应用堆栈实现后缀表达式求值的基本过程：

从左到右读入后缀表达式的各项（运算符或运算数）；

1. 运算数：入栈；
2. 运算符：从堆栈中弹出适当数量的运算数，计算并结果入栈；
3. 最后，堆栈顶上的元素就是表达式的结果值。



中缀表达式求值

基本策略：将中缀表达式转换为后缀表达式，然后求值

如何将中缀表达式转换为后缀？

观察一个简单例子： 2+9/3-5 → 2 9 3 / + 5 -

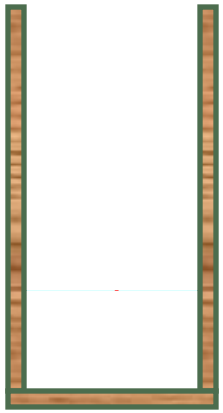
1. 运算数相对顺序不变
2. 运算符号顺序发生改变
   - 需要存储“等待中”的运算符号
   - 要将当前运算符号与“等待中”的最后一个运算符号比较

堆栈！

有括号怎么办？

〖例〗 $a*(b+c)/d$ = ?　　$abc+*d/$

输出:　a　b　c　$+$　$*$　d　$/$

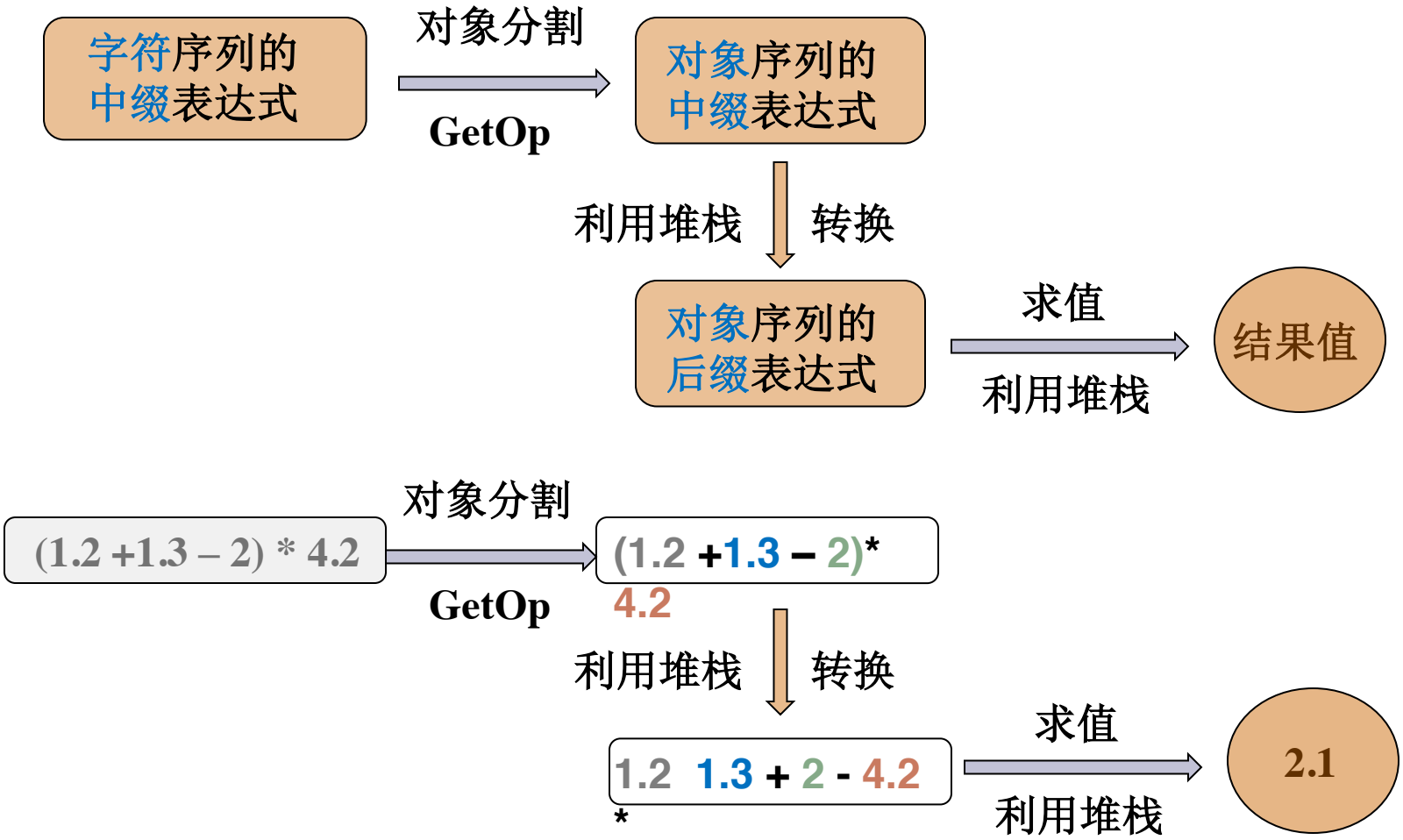


← top

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)	

$$T(N)=O(N)$$

中缀表达式求值


字符序列的
中缀表达式
对象分割
GetOp
对象序列的
中缀表达式
利用堆栈
转换
对象序列的
后缀表达式
求值
利用堆栈
结果值
(1.2 +1.3 – 2) * 4.2
对象分割
GetOp
(1.2 +1.3 – 2)*
4.2
利用堆栈
转换
1.2 1.3 + 2 - 4.2
*
求值
利用堆栈
2.1

中缀表达式如何转换为后缀表达式

➢ 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 +

堆栈的其他应用：

- 函数调用及递归实现
- 深度优先搜索
- 回溯算法
- 。 。 。