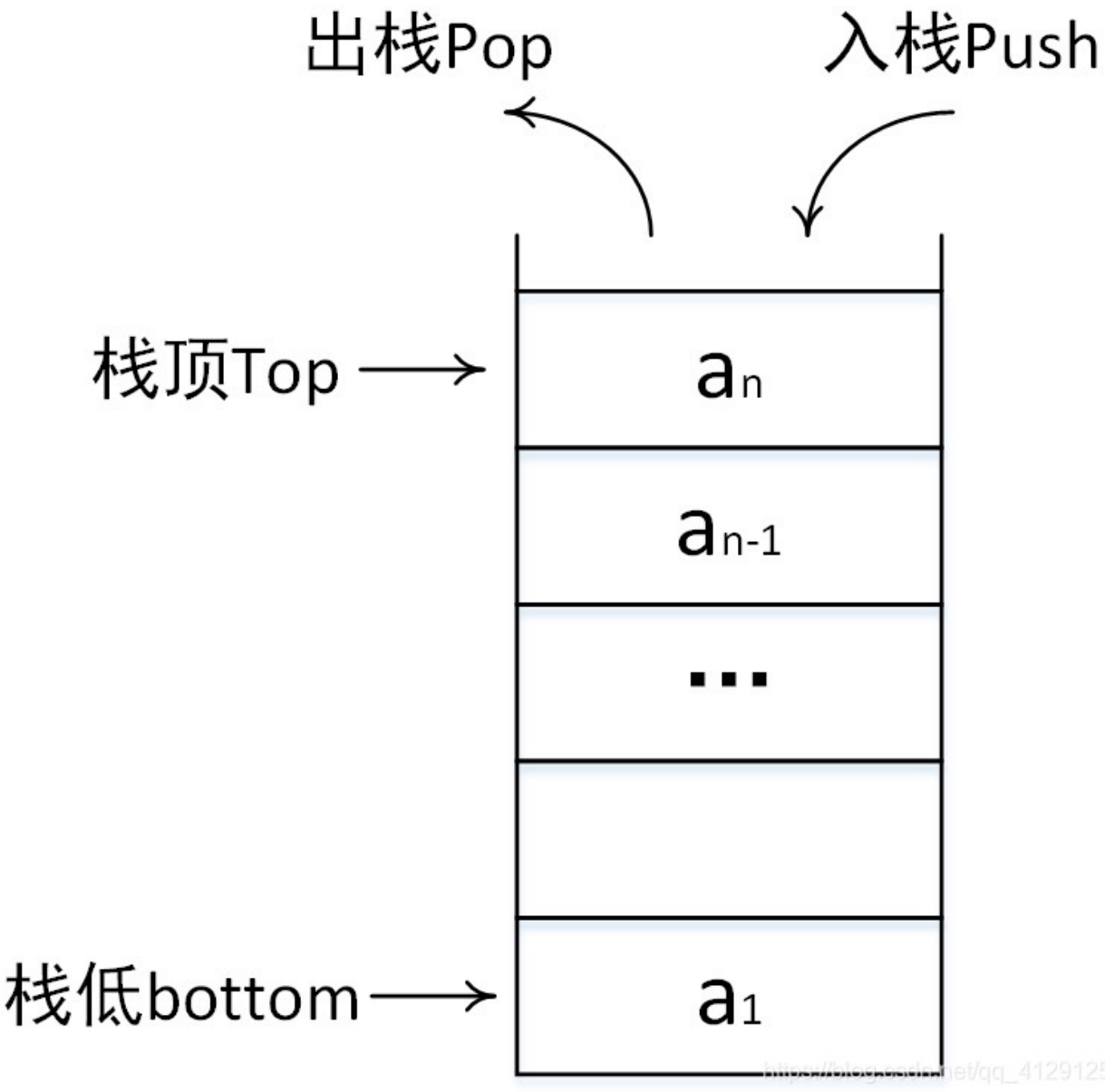
栈,队列和数组

栈

栈	只允许在一端进行插入或删除操作的线性表
栈顶	线性表允许进行插入删除的那一段
栈底	固定的，不允许进行插入删除的那一段
操作特性	后进先出，LIFO



数学性质

卡特兰数：当n个不同元素进栈，出栈元素有 $\frac{1}{n+1}C_{2n}^{n}$

栈的顺序存储结构

顺序栈

- 采用顺序存储的栈
- 利用一组地址连续的存储单元存放自栈底到栈顶的数据元素

```
#define MaxSize 50
typedef struct {
	ElemType data[MaxSize];
	int top;
}SqStack;
```

栈顶指针	**S.top，初始时设置S.top=-1；栈顶元素：S.data[S.top]**		
进栈操作	栈不满时，栈顶指针先加1，再送值到栈顶元素	出栈操作	栈非空时，先找栈顶元素值，再将栈顶指针减1
栈空条件	S.top==-1	栈满条件	S.top==Maxsize-1；栈长：S.top+1

基本操作

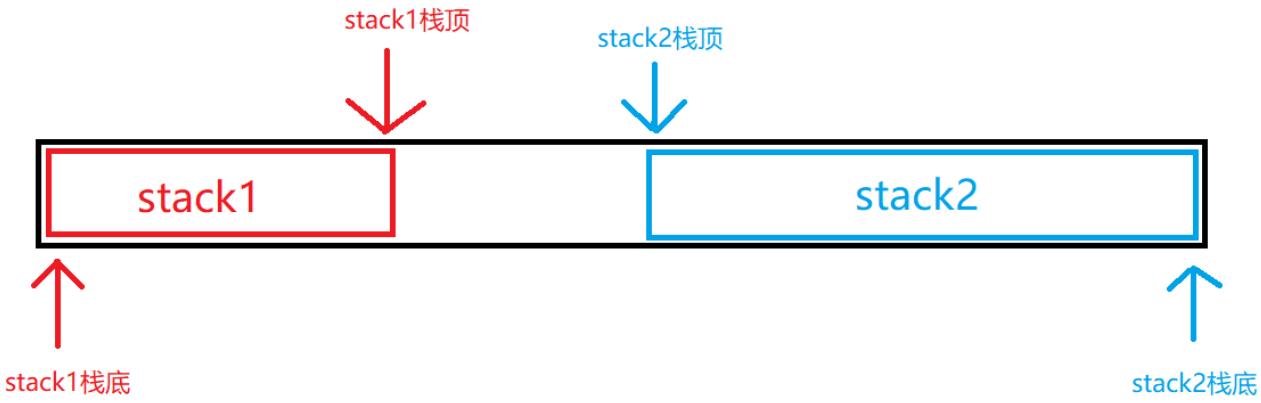
```
//初始化
void InitStack(SqStack &S)
{
	S.top= -1;
}
//判断栈空
bool StackEmpty(SqStack &S)
{
	if(S.top==-1)
		return true;
	else
		return false;
}
//进栈
bool Push(SqStack &S,ElemType x)
{
	if(S.top==MaxSize-1)
		return false;
	x=S.data[++S.top];
	return true;
}
//出栈
bool Pop(SqStack &S,ElemType x)
{
	if(S.top==-1)
```

```
            return false;
        else
            x=S.data[S.top--];
            return true;
}
// 读栈顶元素
bool GetTop(SqStack S, ElemType x)

{
    if (S.top == -1)
        return false;
    x = S.data[S.top];
      return ture;
}
```

共享栈

- 利用栈底位置相对不变的特性，可让两个顺序栈共享一个一维数组空间
- 将两个栈的栈底分别设置在共享空间的两端，两个栈顶向共享空间的中间延伸



链栈

便于多个栈共享存储空间和提高其效率，且不存在栈满上溢的情况，通常采用单链表实现，并规定所有操作都是在单链表的表头进行的，这里规定链栈没有头结点，Lhead指向栈顶元素。

```
typedef struct Linknode
{
        ElemType data;
        Struct Linknode *next;
}*LiStack;
```

易错题

1. 栈和队列具有相同的（ ）。

A. 抽象数据类型　B. 逻辑结构　C. 存储结构　D. 运算

栈是一种（ ）。

A. 顺序存储的线性结构　B. 链式存储的非线性结构

C. 限制存取点的线性结构　D. 限制存储点的非线性结构

【2013 统考真题】一个栈的入栈序列为 $1, 2, 3, \cdots, n$，出栈序列是 $P_1, P_2, P_3, \cdots, P_n$。若 $P_2=3$，则 P_3 可能取值的个数是（ ）。

A. $n-3$　B. $n-2$　C. $n-1$　D. 无法确定

代码题

1. 设单链表的表头指针为L,结点结构由data和next两个域构成，其中data域为字符型，试设计算法判断该链表的全部n个字符是否中心对称。例如xyx、xyyx都是中心对称。

```
int dc(LinkList L,int n)
{
	int i;
	char s[n/2];
	LNode *p=L->next;
	for(i=0;i<n/2;i++)
	{
		s[i]=p->data;
		p=p->next;
	}
	i--;
	if(n%2==1)
	{
		p=p->next;
	}
	while(p!=NULL&&s[i]==p->data)
	{
		i--;
		p=p->next;
	}
	if(i==-1)
		return 1;
	else
		return 0;
}
```

队列

FIFO

队列的顺序存储

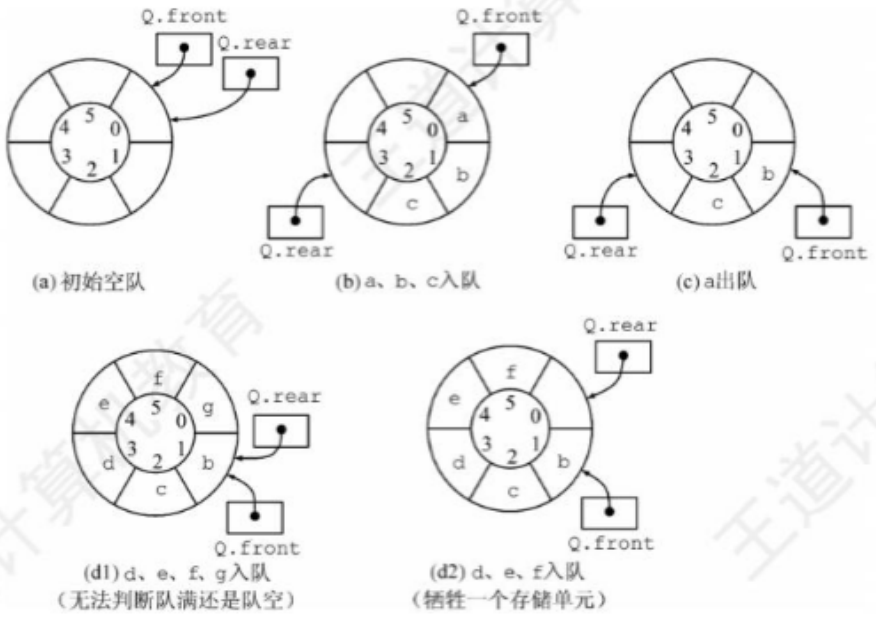
- 分配一块连续的存储单元存放队列中的元素
- 设两个指针
  - 队头指针front指向队头元素
  - 队尾指针rear指向队尾元素的下一个位置

```
#define MaxSize 50
typedef struct {
        ElemType data[MaxSize;
        int front ,rear;
}SqQueue;
```

初始状态	Q.front==Q.rear==0	队满操作	• Q.rear==MaxSize不能作为队列满的条件• 只有一个元素仍满足该条件（假溢出）
进队操作	队不满时，先送值到队尾元素，再将队尾指针加1	出队操作	队不空时，先取队头元素值，再将队头指针加1

循环队列

- 把存储队列元素的表从逻辑上视为一个环



循环队列

- 初始：Q.front=Q.rear=0;

- 入队：Q.rear=(Q.rear+1)%MaxSize
- 出队：Q.front=(Q.front+1)%MaxSize
- 队列长度；(Q.rear+MaxSize-Q.front)%MaxSize;
- 队空条件：Q.front==Q.rear
- 队满：(Q.rear+1)%MaxSize==Q.front;

判断队满还是队空

```
//方法一
//牺牲一个单元来区分队空和队满入队时少用一个队列单元约定以“队头指针在队尾指针的下一位置作为队满的标志
队空：Q.front==Q.rear;
队满：(Q.rear+1)%MaxSize==Q.front;
//方法二
//类型中增设表示元素个数的数据成员
队空：Q.size==0;
队满：Q.size==MaxSize;
//方法三
//类型中增设tag数据成员，以区分是队满还是队空
队空：tag=0且因删除导致Q.front==Q.rear
队满： tag=1且因插入导致Q.front==Q.rear
```

循环队列的操作

```
//初始化
void InitQueue (SqQueue &Q){
	Q.rear=Q.front=0;
}
// 判断队空
bool isEmpty(SqQueue Q)
{
	if(Q.rear==Q.front)
		return true;
	else
		return false;
}
// 入队
bool EnQueue(SqQueue &Q,ElemType x)
{
	if((Q.rear+1)%MaxSize==Q.front)
		return false;
	Q.data[Q.rear]=x;
	Q.rear=(Q.rear+1)%MaxSize;
```

```
        return true;
}
bool DeQueue(SqQueue &Q,ElemType &x){
        if(Q.front==Q.rear)
                return true;
        x=Q.data[Q.front];
        front=(front+1)%MaxSize;
        return true;
}
```

队列的链式存储

- 实质上是一个同时带有队头指针和队尾指针的单链表
- 头指针指向队头节点，尾指针指向队尾节点，即单链表的最后一个节点
- 删除操作时，通常仅需要修改头指针
- 当队列只有一个元素时，删除后队列为空，修改尾指针为rear=front

```
typedef struct LinkNode
{
        ElemType data;
        struct LinkNode *next;
}LinkNode;
typedef struct {
        LinkNode *front,*rear;
}LinkQueue;
```

基本操作

```
//初始化
void InitQueue(LinkQueue &Q)
{
        Q.front=Q.rear=(LinkNode*)malloc(sizeof(LinkLNode));
        Q.front->next=NULL;
}
//判断对空
bool IsEmpty(LinkQueue Q)
{
        if(Q.front==Q.rear)
        {
                return true;
        }
        else
                return false;
}
```

```
//入队
void EnQueue(LinkQueue &Q,ElemType x)
{
	LinkNode *s(LinkNode *)malloc(sizeof(LinkNode));
	s->data=x;
	s->next=NULL;
	Q.rear->next=s;
	Q.rear=s;
}
//出队
bool DeQueue(LinkQueue &Q,ElemType &x)
{
	if(Q.front==Q.rear)
	{
		return false;
	}
	LinNode *p=Q.front->next;
	x=p->data;
	Q.front->next = p->next;
	if(Q.rear==p)
		Q.rear=Q.front;
	free(p);
	return true;
}
```

双端队列

- 允许两端都可以进行入队和出队操作的队列
- 将队列的两端分别称为前端和后端
- 其元素的逻辑结构仍是线性结构

输出受限的双端队列	**• 允许在一段进行插入和删除，但在另一端只允许插入的双端队列**
输入受限的双端队列	• 允许在一段进行插入和删除，但在另一端只允许删除的双端队列

易错题

与顺序队列相比，链式队列（ ）。

A. 优点是队列的长度不受限制

B. 优点是进队和出队时间效率更高

C. 缺点是不能进行顺序访问

D. 缺点是不能根据队首指针和队尾指针计算队列的长度

用链式存储方式的队列进行删除操作时需要（ ）。

A. 仅修改头指针　　B. 仅修改尾指针

C. 头尾指针都要修改　　D. 头尾指针可能都要修改

代码题

1. 若希望循环队列中的元素都能得到利用，则需设置一个标志域tg,并以tag的值为0或1来区分队头指针front和队尾指针rear相同时的队列状态是“空”还是“满”.试编写与此结构相应的入队和出队算法

```
//入队
int EnQueue1(SqQueue &Q,ElemType x)
{
	if(Q.front==Q.rear&&Q.tag==1)
		return 0;
	Q.data[Q.rear]=x;
	Q.rear=(Q.rear+1)%MaxSize;
	Q.tag=1;
	return 1;
}
//出队
int DeQueue1(SqQueue &Q,ElemType &x)
{
	if(Q.front==Q.rear&&tag==0)
		return 0;
	x=Q.data[Q.front];
	Q.front=(Q.front+1)%MaxSize;
	Q.tag=1;
	return 1;
}
```

1. Q是一个队列，S是一个空栈，实现将队列中的元素逆置的算法。

```
void Inverse(Stack &S,Queue &Q)
{
```

```
    while(!QueueEmpty(Q))
    {
        x=DeQueue(Q);
        push(S,x);
    }
    while(!StackEmpty(S))
    {
        Pop(S,x);
        Enqueue(Q,x);
    }
}
```

利用两个栈 S1 和 S2 来模拟一个队列，已知栈的 4 个运算定义如下：

```
Push(S,x);                  //元素 x 入栈 S
Pop(S,x);                   //S 出栈并将出栈的值赋给 x
StackEmpty(S);              //判断栈是否为空
StackOverflow(S);           //判断栈是否满
```

如何利用栈的运算来实现该队列的 3 个运算（形参由读者根据要求自己设计）？

```
Enqueue;                    //将元素 x 入队
Dequeue;                    //出队，并将出队元素存储在 x 中
QueueEmpty;                 //判断队列是否为空
```

1）对 S2 的出栈操作用做出队，若 S2 为空，则先将 S1 中的所有元素送入 S2。

2）对 S1 的入栈操作用作入队，若 S1 满，必须先保证 S2 为空，才能将 S1 中的元素全部插入 S2 中。

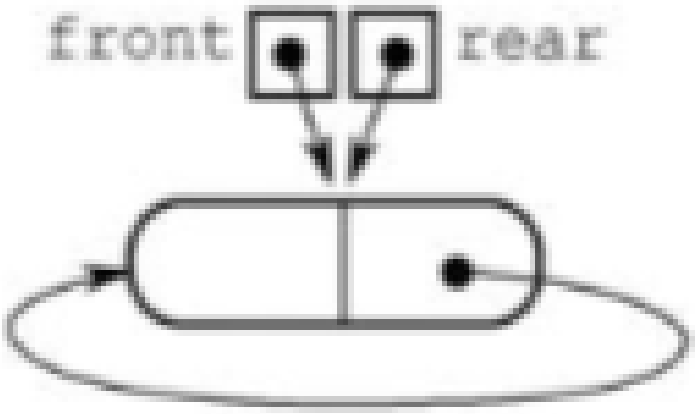
```
//入队
int EnQueue(Stack &S1,Stack &S2,ElemType e)
{
    if(!StackOverflow(s1))
    {
        Push(S1,e);
        return 1;
    }
    if(StackOverflow(S1)&&!StackEmpty(s2))
    {
        printf("队列满");
        return 0;
    }
    if(StackOverflow(S1)&&StackEmpty(s2)){
        while(!Stack(Empty(s1)))
        {
            Pop(S1,x);
            Push(S2,x);
        }
    }
    Push(S1,e);
```

```
        return 1;
}
//出队
void DeQueue(Stack &S1,Stack &S2,ElemType &x)
{
        if(!StackEmpty(S2))
        {
                Pop(S2,x);
        }
        else if(StackEmpty(s1))
                printf("队列空");
        else
        {
                while(!StackEmpty(S1))
                {
                        Pop(S1,x);
                        Push(S2,x);
                }
                Pop(S2,x)
        }
}
//判断为空
int QueueEmpty(Stack S1,Stack S2)
{
        if(StackEmpty(S1)&&StackEmpty(S2))
                return 1;
        else
                return 0;
}
```

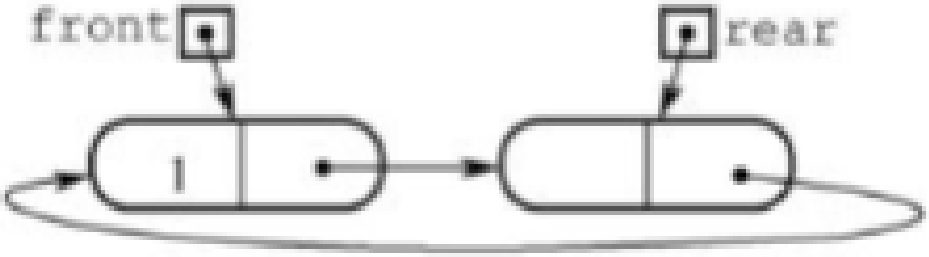
1. 【2019统考真题】请设计一个队列，要求满足：①)初始时队列为空；②入队时，允许增加队列占用空间：③出队后，出队元素所占用的空间可重复使用，即整个队列所占用的空间只增不减；④入队操作和出队操作的时间复杂度始终保持为O(1)。请回答：
   1)该队列是应选择链式存储结构，还是应选择顺序存储结构？

- 链式存储

   2)画出队列的初始状态，并给出判断队空和队满的条件。



3)画出第一个元素入队后的队列状态。



4)给出入队操作和出队操作的基本过程。

入队操作：
若（front==rear->next）　　//队满 　　则在rear后面插入一个新的空闲结点； 入队元素保存到rear所指结点中；rear=rear->next；返回。
出队操作：
若（front==rear）　　//队空 　　则出队失败，返回； 取front所指结点中的元素e；front=front->next；返回e。

栈和队列的应用

栈的应用

括号匹配

- 初始设置一个空栈，顺序读入括号
- 若是右括号，置于栈顶
- 若是左括号，压入栈中
- 算法结束时，栈为空，否则括号序列不匹配

算术表达式

中缀表达式转后缀表达式

> **Example**
>
> A+B*(C-D)-E/F的转化
>
> 1. 按运算顺序加括号：((A+③(B*②(C-①D)))-⑤(E/④F))。
> 2. 运算符后移：((A(B(CD)-①)*②)+③(EF)/④)-⑤
> 3. 去除括号：ABCD-①*②+③EF/④-⑤

1. 遇到操作数直接加入后缀表达式
2. 遇到界限符。若为“(”，则直接入栈；若为“)”，则依次弹出栈中的运算符，并加入后缀表达式，直到弹出“(”为止。注意，“(”直接删除，不加入后缀表达式。
3. 遇到运算符。若其优先级高于除“(”外的栈顶运算符，则直接入栈。否则，从栈顶开始，依次弹出栈中优先级高于或等于当前运算符的所有运算符，并加入后缀表达式，直到遇到一个优先级低于它的运算符或遇到“(”时为止，之后将当前运算符入栈。

步	待处理序列	栈内	后缀表达式	扫描项	说　明
1	A+B*(C-D)-E/F			A	A 加入后缀表达式
2	+B*(C-D)-E/F		A	+	+入栈
3	B*(C-D)-E/F	+	A	B	B 加入后缀表达式
4	*(C-D)-E/F	+	AB	*	*优先级高于栈顶，*入栈
5	(C-D)-E/F	+*	AB	(	(直接入栈
6	C-D)-E/F	+*(	AB	C	C 加入后缀表达式
7	-D)-E/F	+*(	ABC	-	栈顶为(，-直接入栈
8	D)-E/F	+*(-	ABC	D	D 加入后缀表达式
9	)-E/F	+*(-	ABCD	)	遇到)，弹出-，删除(
10	-E/F	+*	ABCD-	-	-优先级低于栈顶，依次弹出*、+，-入栈
11	E/F	-	ABCD-*+	E	E 加入后缀表达式
12	/F	-	ABCD-*+E	/	/优先级高于栈顶，/入栈
13	F	-/	ABCD-*+E	F	F 加入后缀表达式
14		-/	ABCD-*+EF		字符扫描完毕，弹出剩余运算符
			ABCD-*+EF/-		结束

栈的深度

栈中元素个数

后缀表达式求值

表 3.2　后缀表达式 ABCD-*+EF/-求值的过程

步	扫描项	项类型	动作	栈中内容
1			置空栈	空
2	A	操作数	进栈	A
3	B	操作数	进栈	A B
4	C	操作数	进栈	A B C
5	D	操作数	进栈	A B C D
6	-	操作符	D、C 退栈，计算 C-D，结果 R_1 进栈	A B R_1
7	*	操作符	R_1、B 退栈，计算 B×R_1，结果 R_2 进栈	A R_2
8	+	操作符	R_2、A 退栈，计算 A+R_2，结果 R_3 进栈	R_3
9	E	操作数	进栈	R_3 E
10	F	操作数	进栈	R_3 E F
11	/	操作符	F、E 退栈，计算 E/F，结果 R_4 进栈	R_3 R_4
12	-	操作符	R_4、R_3 退栈，计算 R_3-R_4，结果 R_5 进栈	R_5

递归

- 在递归调用的过程中，系统为每一层的返回点、局部变量、传入实参等开辟了递归工作栈来进行数据存储
- 递归次数过多容易造成栈溢出等
- 将递归算法转换为非递归算法，通常需要借助栈来实现这种转换,消除递归并不一定需要栈
- 效率不高：原因是递归调用过程中包含很多重复的计算
- 代码简单，容易理解

队列的应用

层次遍历

计算机系统中的应用

1. 解决主机与外部设备之间速度不匹配的问题（如打印机与主机，设置一个打印数据缓冲区）
2. 解决由多用户引起的资源竞争问题（如CPU资源的竞争）
3. 页面替换算法(FIFO算法)

数组和特殊矩阵

数组的存储结构

一维数组A[0...n-1]

$$LOC(a_i) = LOC(a_0) + i * L$$

二维数组A[0..h1][0...h2]

$$LOC(a_i) = LOC(a_{(0,0)}) + [i * (h2 + 1) + j] * L$$

特殊矩阵的存储与压缩

对称矩阵

下标对应关系

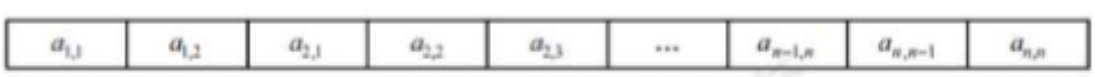
$$K = \begin{cases} \frac{i(i-1)}{2} + j - 1, i \geq j \\ \frac{j(j-1)}{2} + i - 1, i < j \end{cases}$$

三角矩阵

上三角矩阵全为常量开辟一个单元存储

$$K = \begin{cases} \frac{i(i-1)}{2} + j - 1, i \geq j \\ \frac{n(n+1)}{2}, i < j \end{cases}$$

三对角矩阵



$$k = 2i + j - 3$$

稀疏矩阵

$$M = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 0 & 9 & 0 & 0 \\ 0 & 23 & 0 & 0 \end{bmatrix}$$

对应的三元组

i	j	a_{ij}
0	0	4
1	2	6
2	1	9
3	1	23

易错题

下列关于矩阵的说法中，正确的是（ ）。

I. 在 n（$n>3$）阶三对角矩阵中，每行都有 3 个非零元

II. 稀疏矩阵的特点是矩阵中的元素较少

A. 仅 I　　B. 仅 II　　C. I 和 II　　D. 无正确项

【2023 统考真题】若采用三元组表存储结构存储稀疏矩阵 M，则除三元组表外，下列数据中还需要保存的是（ ）。

I. M 的行数　　II. M 中包含非零元素的行数

III. M 的列数　　IV. M 中包含非零元素的列数

A. 仅 I、III　　B. 仅 I、IV　　C. 仅 II、IV　　D. I、II、III、IV