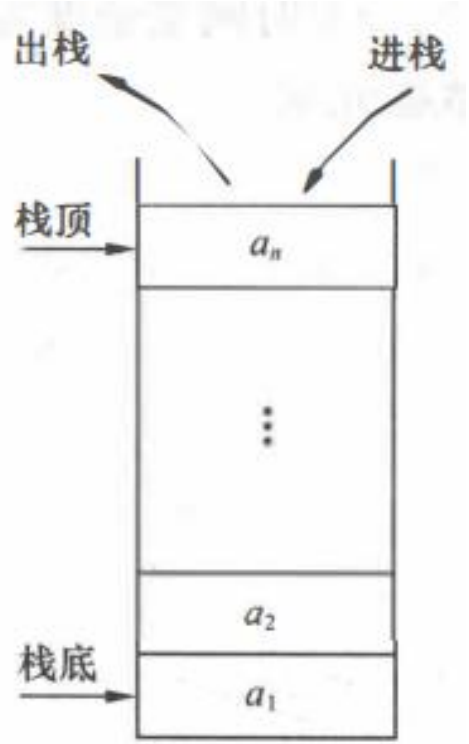
栈和队

栈

线性表是具有相同数据类型的n个数据元素的有限序列。n为表长
栈是限定仅在表尾进行插入或删除操作的线性表。
表尾端称为栈顶，表头端称为栈底，不含元素的空表称为空栈。

洗干净的盘子总是逐个往上叠放在已经洗好的盘子上面,而用时从上往下逐个取用。栈的操作特点正是上述实际应用的抽象。　　后进先出

栈的数学性质：n个不同元素进栈，出栈元素不同排列的个数为 $\frac{1}{n+1}C_{2n}^{n}$



（a）栈的示意图

例题

(1) 3个不同元素依次进栈，能得到____种不同的出战序列.

(2) 13. 用 S 表示进栈操作，用 X 表示出栈操作，若元素的进栈顺序是 1234，为了得到 1342 的出栈顺序，相应的 S 和 X 的操作序列为（ ）。

A. SXSXSSXX　　B. SSSXXSXX　　C. SXSSXXSX　　D. SXSSXSXX

(3) 26. 【2017 统考真题】下列关于栈的叙述中，错误的是（ ）。

I. 采用非递归方式重写递归程序时必须使用栈

II. 函数调用时，系统要用栈保存必要的信息

III. 只要确定了入栈次序，即可确定出栈次序

IV. 栈是一种受限的线性表，允许在其两端进行操作

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

(4) 15. 【2018 统考真题】若栈 S1 中保存整数，栈 S2 中保存运算符，函数 F() 依次执行下述各步操作：

1）从 S1 中依次弹出两个操作数 a 和 b。

2）从 S2 中弹出一个运算符 op。

3）执行相应的运算 b op a。

4）将运算结果压入 S1 中。

假定 S1 中的操作数依次是 5, 8, 3, 2（2 在栈顶），S2 中的运算符依次是*、-、+（+在栈顶）。调用 3 次 F() 后，S1 栈顶保存的值是（ ）。

A. -15　　B. 15　　C. -20　　D. 20

栈的基本操作

2. 栈的基本操作

各种辅导书中给出的基本操作的名称不尽相同，但所表达的意思大致是一样的。这里我们以严蔚敏编写的教材为准给出栈的基本操作，希望读者能熟记下面的基本操作。

InitStack(&S)：初始化一个空栈 S。

StackEmpty(S)：判断一个栈是否为空，若栈 S 为空则返回 true，否则返回 false。

Push(&S,x)：进栈，若栈 S 未满，则将 x 加入使之成为新栈顶。

Pop(&S,&x)：出栈，若栈 S 非空，则弹出栈顶元素，并用 x 返回。

GetTop(S,&x)：读栈顶元素，若栈 S 非空，则用 x 返回栈顶元素。

DestroyStack(&S)：销毁栈，并释放栈 S 占用的存储空间（“&”表示引用调用）。

在解答算法题时，若题干未做出限制，则可直接使用这些基本的操作函数。

销毁栈写不写？

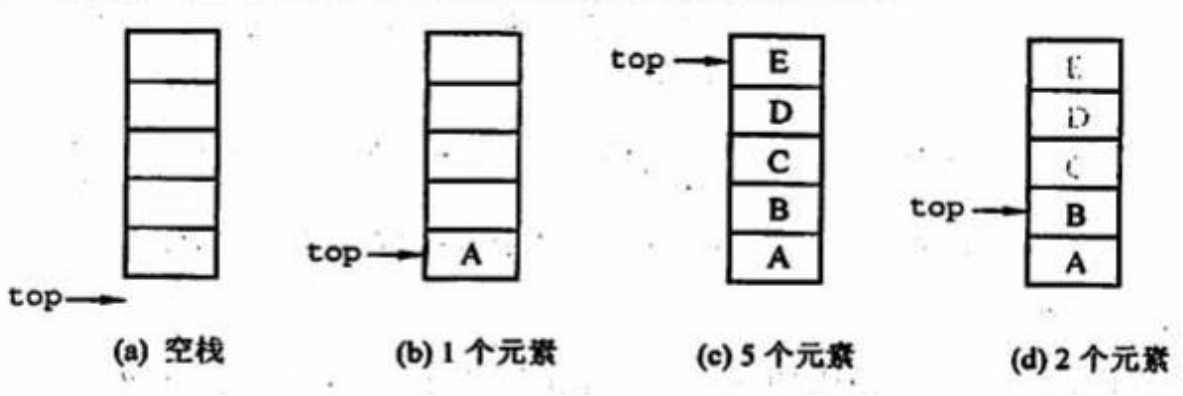
顺序栈结构体定义

1. 顺序栈定义

```
typedef struct
{
    int data[maxSize];          //存放栈中元素，maxSize 是已定义的常量
    int top;                    //栈顶指针
} SqStack;                      //顺序栈类型定义
```

栈顶指针：S.top，初始时设置 S.top=-1①；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，先取栈顶元素值，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。

顺序栈的操作



初始化

判空

进栈

出栈

读栈顶

注意：这里 top 指向的是栈顶元素，所以进栈操作为 S.data[++S.top]=x，出栈操作为 x=S.data[S.top--]。若栈顶指针初始化为 S.top=0，即 top 指向栈顶元素的下一位置，则入栈操作变为 S.data[S.top++]=x；出栈操作变为 x=S.data[--S.top]。相应的栈空、栈满条件也会发生变化。请读者仔细体会其中的不同之处，做题时要灵活应变。

共享栈

3．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，如图 3.3 所示。

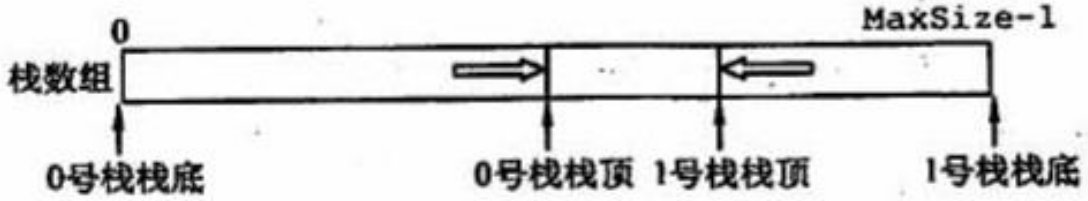


图 3.3　两个顺序栈共享存储空间

两个栈的栈顶指针都指向栈顶元素，top0=-1 时 0 号栈为空，top1=MaxSize 时 1 号栈为空；仅当两个栈顶指针相邻（top1-top0=1）时，判断为栈满。当 0 号栈进栈时 top0 先加 1 再赋值，1 号栈进栈时 top1 先减 1 再赋值；出栈时则刚好相反。

共享栈是为了更有效地利用存储空间，两个栈的空间相互调节，只有在整个存储空间被占满时才发生上溢。其存取数据的时间复杂度均为 $O(1)$，所以对存取效率没有什么影响。

共享栈的实现

```
#define MaxSize 10                  //定义栈中元素的最大个数
typedef struct{
    ElemType data[MaxSize];         //静态数组存放栈中元素
    int top0;                       //0号栈栈顶指针
    int top1;                       //1号栈栈顶指针
} ShStack;

//初始化栈
void InitStack(ShStack &S){
    S.top0=-1;                      //初始化栈顶指针
    S.top1=MaxSize;
}
```

栈满的条件：top0 + 1 == top1

链栈的结构体定义

2. 链栈结点定义

```
typedef struct LNode
{
    int data;                    //数据域
    struct LNode *next;          //指针域
}LNode;                          //链栈结点定义
```

链栈就是采用链表来存储栈。这里用带头结点的单链表来作为存储体

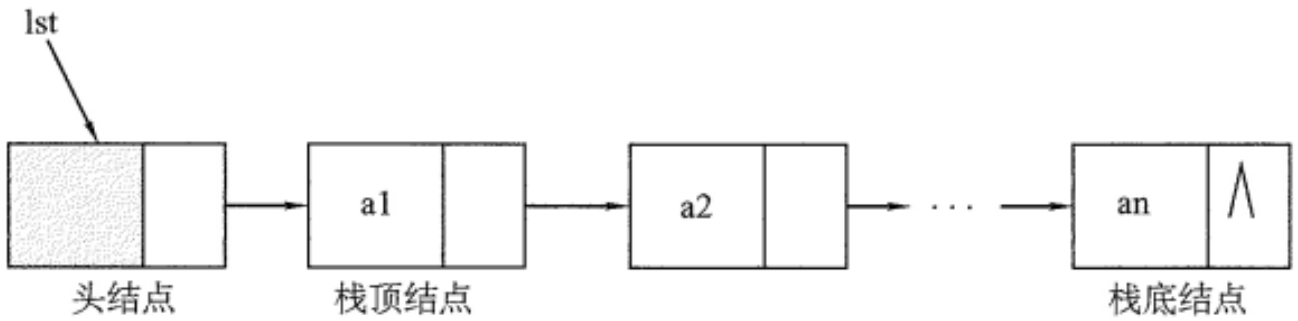


图 3-2　链栈示意图

链栈的操作

- 入栈——push(LinkStack &S, int x) （头插法）
- 出栈——Pop(LinkStack S, int &x)
- 判空
- 取栈顶元素

队列

线性表是具有相同数据类型的n（n≥0）个数据元素的有限 序列，其中n为表长，当n = 0时线性表是一个空表。若用L命名线性表，则其一般表示为

$$L = (a_1, a_2, \dots, a_i, a_{i+1}, \dots, a_n)$$

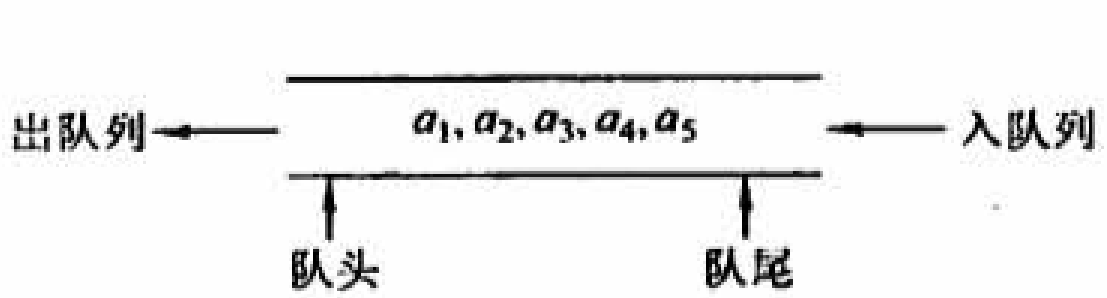
进栈　出栈

栈（Stack）是只允许在一端进行插入或删除操作的线性表

队列（Queue）是只允许在一端进行插入，在另一端删除的线性表

入队　出队

先进先出



队头（Front）。允许删除的一端，又称队首。

队尾（Rear）。允许插入的一端。

空队列。不含任何元素的空表。

队列的基本操作

InitQueue(&Q)：初始化队列，构造一个空队列 Q。

QueueEmpty(Q)：判队列空，若队列 Q 为空返回 true，否则返回 false。

EnQueue(&Q,x)：入队，若队列 Q 未满，将 x 加入，使之成为新的队尾。

DeQueue(&Q,&x)：出队，若队列 Q 非空，删除队头元素，并用 x 返回。

GetHead(Q,&x)：读队头元素，若队列 Q 非空，则将队头元素赋值给 x。

需要注意的是，栈和队列是操作受限的线性表，因此不是任何对线性表的操作都可以作为栈和队列的操作。比如，不可以随便读取栈或队列中间的某个数据。

顺序队的结构体定义

3．顺序队列定义

```
typedef struct
{
```


```
    int data[maxSize];
    int front;                  //队首指针
    int rear;                   //队尾指针
}SqQueue;                       //顺序队类型定义
```

初始状态（队空条件）：Q.front==Q.rear==0。

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。

顺序队的实现

初始化

判空

进队

出队

读队头

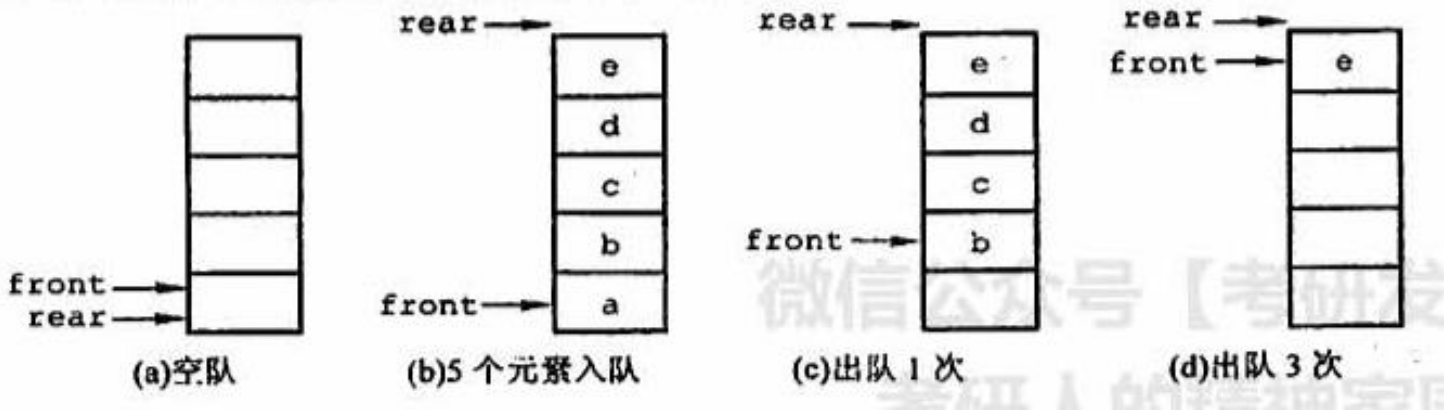


图 3.6(a)所示为队列的初始状态，有 Q.front==Q.rear==0 成立，该条件可以作为队列判空的条件。但能否用 Q.rear==MaxSize 作为队列满的条件呢？显然不能，图 3.6(d)中，队列中仅有一个元素，但仍满足该条件。这时入队出现“上溢出”，但这种溢出并不是真正的溢出，在 data 数组中依然存在可以存放元素的空位置，所以是一种“假溢出”。

循环队列

把顺序存储的队想象成一个环状。即把存储队列元素的表从逻辑上视为一个环,称为循环队列，当指针指向MaxSize-1时，再前进一个位置就自动到0,这里使用取余运算来实现。

初始时：Q.front=Q.rear=0。

队首指针进 1：Q.front=(Q.front+1)%MaxSize。

队尾指针进 1：Q.rear=(Q.rear+1)%MaxSize。

队列长度：(Q.rear+MaxSize-Q.front)%MaxSize。

判空判满

牺牲一个单元来区分队空和队满，这是普遍的做法。

队满条件:

(Q.rear+1)%MaxSize==Q.front

队空条件:

Q.front==Q.rear

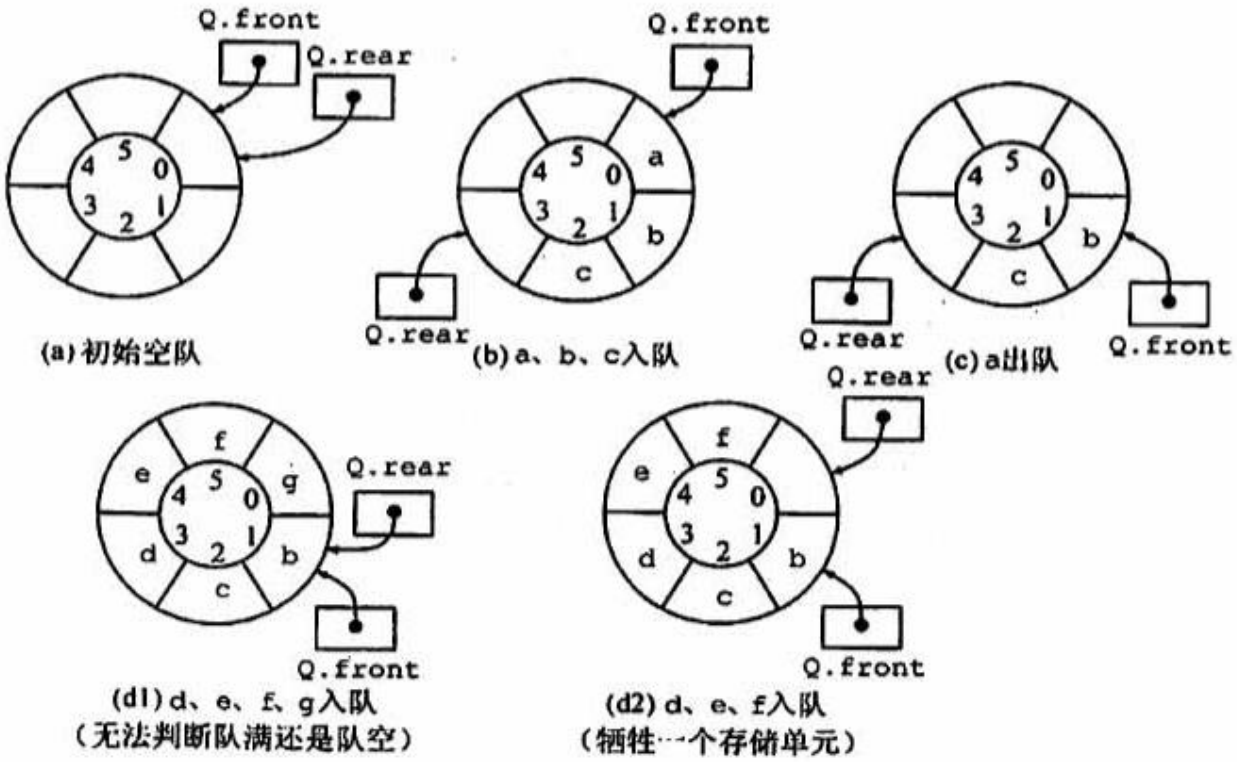


图 3.7 循环队列出入队示意图

第二种做法
自行实现

2）类型中增设表示元素个数的数据成员。这样，队空的条件为 Q.size==0；队满的条件为 Q.size==MaxSize。这两种情况都有 Q.front==Q.rear。

3）类型中增设 tag 数据成员，以区分是队满还是队空。tag 等于 0 时，若因删除导致 Q.front==Q.rear，则为队空；tag 等于 1 时，若因插入导致 Q.front==Q.rear，则为队满。

循环队列的实现

- InitQueue
- isEmpty
- EnQueue
- DeQueue
- Front()

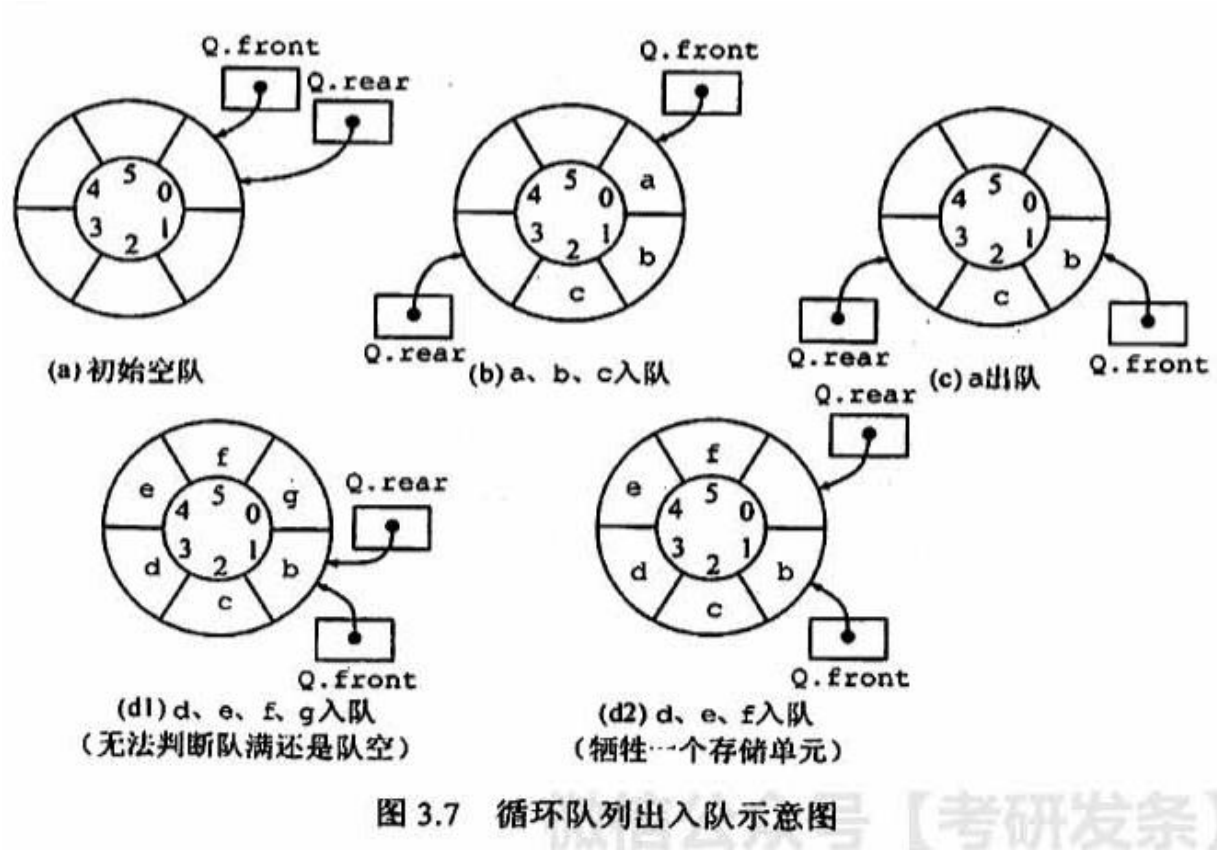
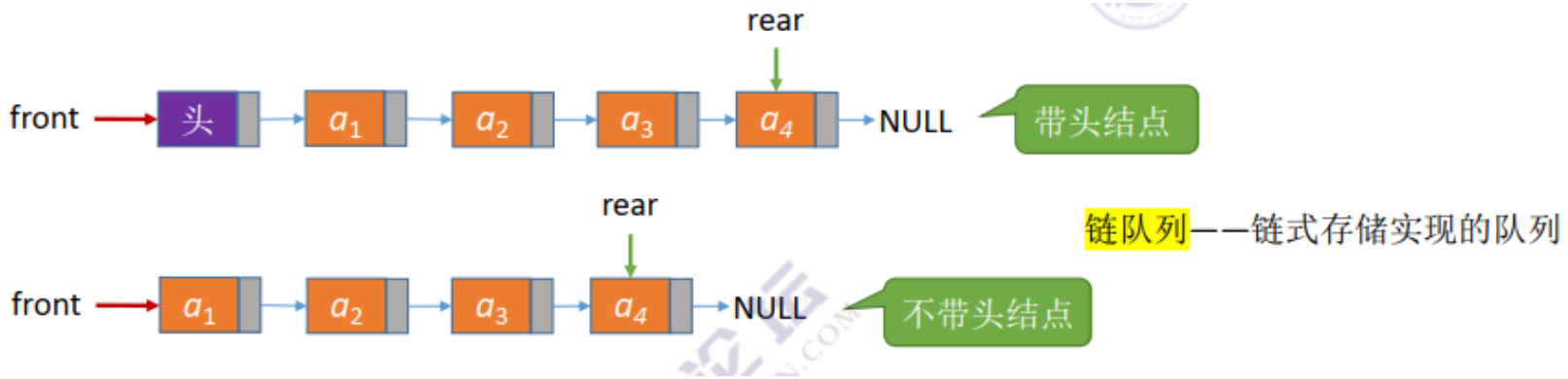


图 3.7 循环队列出入队示意图

链队的结构体定义

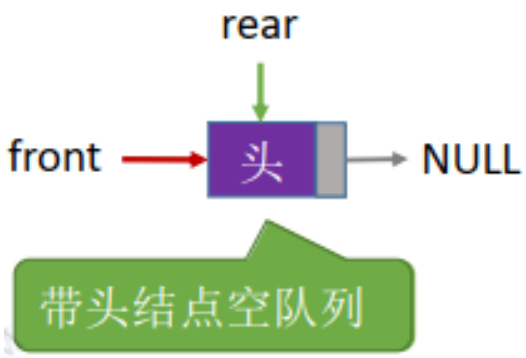
```
typedef struct LinkNode{     //链式队列结点
    ElemType data;
    struct LinkNode *next;
}LinkNode;
typedef struct{              //链式队列
    LinkNode *front,*rear;   //队列的队头和队尾指针
}LinkQueue;
```



链队列——链式存储实现的队列

链队的操作

```
// 含有头节点
class LinkedQueue {
private:
    QNode *front_;
    QNode *rear_;

public:

    LinkedQueue() : front_(new QNode( e: 0)), rear_(front_) {

    }

    bool front(T &e);

    bool back(T &e);

    bool push(T e);

    bool pop();

    int size();

    bool empty();
```



双端队列


双端队列
1
2
3
4
5
插入
删除
栈：只允许从一端插入和删除的线性表
删除
1
2
3
4
5
插入
队列：只允许从一端插入、另一端删除的线性表
若只使用其中一端的插入、删除操作，则效果等同于栈
插入
删除
1
2
3
4
5
插入
删除
双端队列：只允许从两端插入、两端删除的线性表

递归的小例子

- 使用递归实现div2
- 将一个数除以2 直到他小于等于1

- 上一次作业题目 ListIsEqual递归实现

栈的应用——进制转换

将十进制数转为二进制。
对除2取余法进行模拟。

递归?

十进制转n进制？ n<10
实现convert_recursion

栈的应用——括号匹配

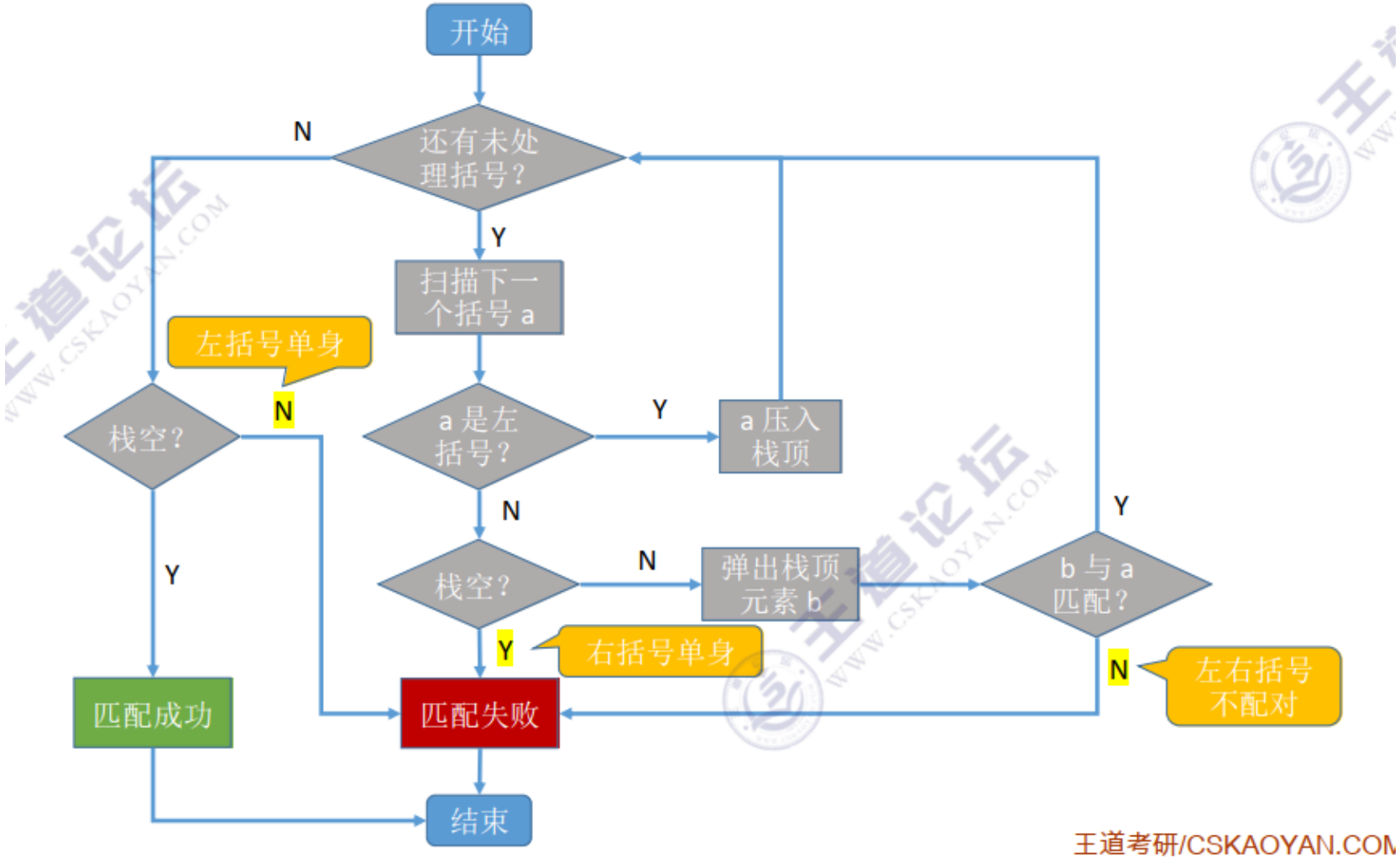
检查一个括号序列是否合法，例如“[]()[()]”

合法的括号序列满足：当前括号为左括号时，下一个是左括号，或者是对应状态的右括号。

练习：{}[]()

随便写一个

括号匹配



回文串　迭代+递归

课本　（2）回文是指正读反读均相同的字符序列，如“abba”和“abdba”均是回文，但“good”不是回文。试写一个算法判定给定的字符序列是否为回文。（提示：将一半字符入栈）

栈 空间复杂度O(n)

验证回文串

https://leetcode.cn/problems/valid-palindrome/solution/yan-zheng-hui-wen-chuan-by-leetcode-solution/

“123”转123(int)

中缀转后缀

中缀表达式转后缀表达式（机算）

初始化一个栈，用于保存暂时还不能确定运算顺序的运算符。
从左到右处理各个元素，直到末尾。可能遇到三种情况：

① 遇到操作数。直接加入后缀表达式。
② 遇到界限符。遇到“(”直接入栈；遇到“)”则依次弹出栈内运算符并加入后缀表达式，直到弹出“(”为止。注意：“(”不加入后缀表达式。
③ 遇到运算符。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，若碰到“(”或栈空则停止。之后再把当前运算符入栈。

* / 优先级高于 + -

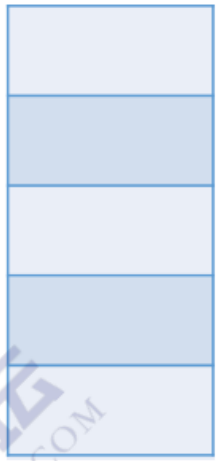
按上述方法处理完所有字符后，将栈中剩余运算符依次弹出，并加入后缀表达式。

A + B - C * D / E + F

① ④ ② ③ ⑤

A B + C D * E / - F +

① ② ③ ④ ⑤



A

后缀表达式求值

后缀表达式求值

postexp= “ 2 1 3 + * 4 – ”

栈

- 初始化栈从左至右读取后缀表达式
- 遇到数字，入栈
- 遇到运算符，弹出两个元素进行运算，并把结果压入栈
- 后缀表达式计算读取完毕，表达式运算结果为栈中的唯一元素

中缀表达式求值

中缀表达式的计算（用栈实现）

用栈实现中缀表达式的计算：
初始化两个栈，操作数栈和运算符栈
若扫描到操作数，压入操作数栈
若扫描到运算符或界限符，则按照“中缀转后缀”相同的逻辑压入运算符栈（期间也会弹出运算符，每当弹出一个运算符时，就需要再弹出两个操作数栈的栈顶元素并执行相应运算，运算结果再压回操作数栈）

A + B - C * D / E + F

① ④ ② ③ ⑤

A B + C D * E / - F +

① ② ③ ④ ⑤

表达式求值

1.表达式类型：抽象or具体

2.表达式怎么存?

char* or SeqList

边转边求值

栈的应用——递归

通常用头指针来标识一个单链表，如单链表 L，头指针为 NULL 时表示一个空表。此外，为了操作上的方便，在单链表第一个结点之前附加一个结点，称为头结点。头结点的数据域可以不设任何信息，也可以记录表长等信息。头结点的指针域指向线性表的第一个元素结点，如图 2.4 所示。

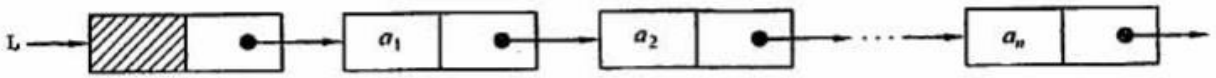


图 2.4　带头结点的单链表

头结点和头指针的区分：不管带不带头结点，头指针都始终指向链表的第一个结点，而头结点是带头结点的链表中的第一个结点，结点内通常不存储信息。

1.函数调用中的栈

2.

阿克曼函数 递归

（9）已知 Ackermann 函数定义如下：

$$Ack(m,n)=\begin{cases} n+1 & \text{当}m=0\text{时} \\ Ack(m-1,1) & \text{当}m\neq 0,n=0\text{时} \\ Ack(m-1,Ack(m,n-1)) & \text{当}m\neq 0,n\neq 0\text{时} \end{cases}$$

① 写出计算 $Ack(m, n)$的递归算法，并根据此算法给出 $Ack(2, 1)$的计算过程。

② 写出计算 $Ack(m, n)$的非递归算法。

（10）已知 f 为单链表的表头指针，链表中存储的都是整型数据，试写出实现下列运算的递归算法：

① 求链表中的最大整数；

② 求链表的结点个数；

③ 求所有整数的平均值。

作业

天勤

（3）假设以 I 和 O 分别表示入栈和出栈操作。若栈的初态和终态均为空，入栈和出栈的操作序列可表示为仅由 I 和 O 组成的序列，则称可以操作的序列为合法序列，否则称为非法序列。

1）试指出判别给定序列是否合法的一般规则。

2）两个不同的合法序列（对两个具有同样元素的输入序列）能否得到相同的输出元素序列？如能得到，请举例说明。

3）写出一个算法，判定所给的操作序列是否合法。若合法，返回 1，否则返回 0（假定被判定的操作序列已存入一维 char 型数组 ch[]中，操作序列以“\0”为结束符）。

课本

（5）假设以 I 和 O 分别表示入栈和出栈操作。栈的初态和终态均为空，入栈和出栈的操作序列可表示为仅由 I 和 O 组成的序列，称可以操作的序列为合法序列，否则称为非法序列。

① 下面所示的序列中哪些是合法的?

A. IOIIOIOO　　B. IOOIOIIO　　C. IIIOIOIO　　D. IIIOOIOO

② 通过对①的分析，写出一个算法，判定所给的操作序列是否合法。若合法，返回 true，否则返回 false（假定被判定的操作序列已存入一维数组中）。

作业

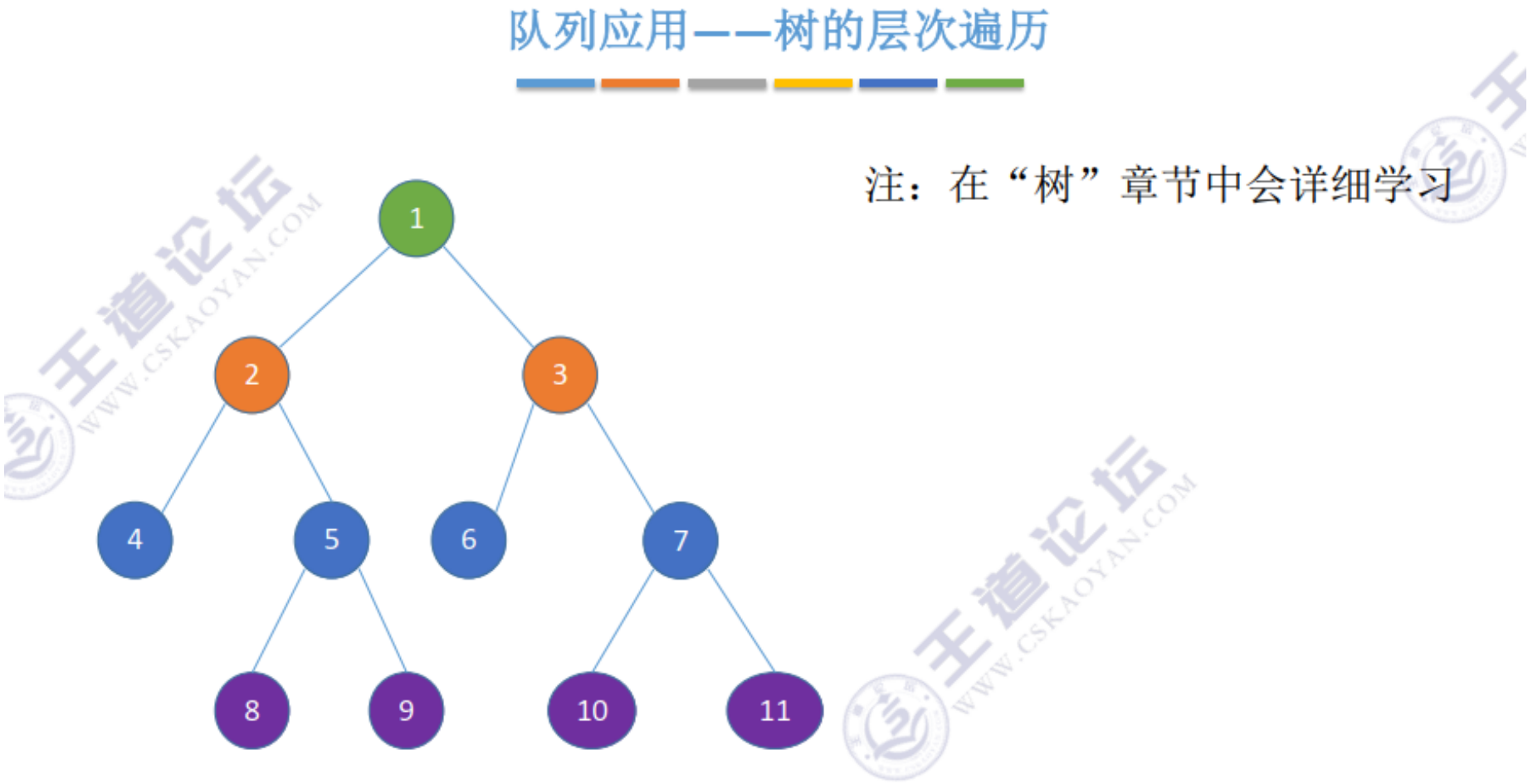
迭代+递归

（1）求解二次方根 $\sqrt{A}$ 的迭代函数定义如下：

$$sqrt(A,p,e)=\begin{cases} p & |p^2-A|<e \\ sqrt\left(A,\dfrac{\left(p+\dfrac{A}{p}\right)}{2},e\right) & |p^2-A|\geqslant e \end{cases}$$

式中，p 是 A 的近似二次方根；e 是结果允许误差。试写出相应的递归算法和非递归算法（假设取绝对值函数 fabs()可以直接调用）。

队列的应用——二叉树的层次遍历



队列在计算机系统中的应用

- 进程调度算法：例如FCFS
- 进程的就绪队列
- 打印数据缓冲区