# 第三章 栈和队列

马媛

# 栈

- 栈是具有一定操作约束的线性表
- 限定仅在一端（表尾，Top）进行插入或删除操作
- 允许插入和删除的一端称为栈顶（表尾，Top），另一端称为栈底（表头，Bottom）
- 特点：后进先出，Last In First Out（LIFO）
- 插入数据：入栈（Push）
- 删除数据：出栈（Pop）

# 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

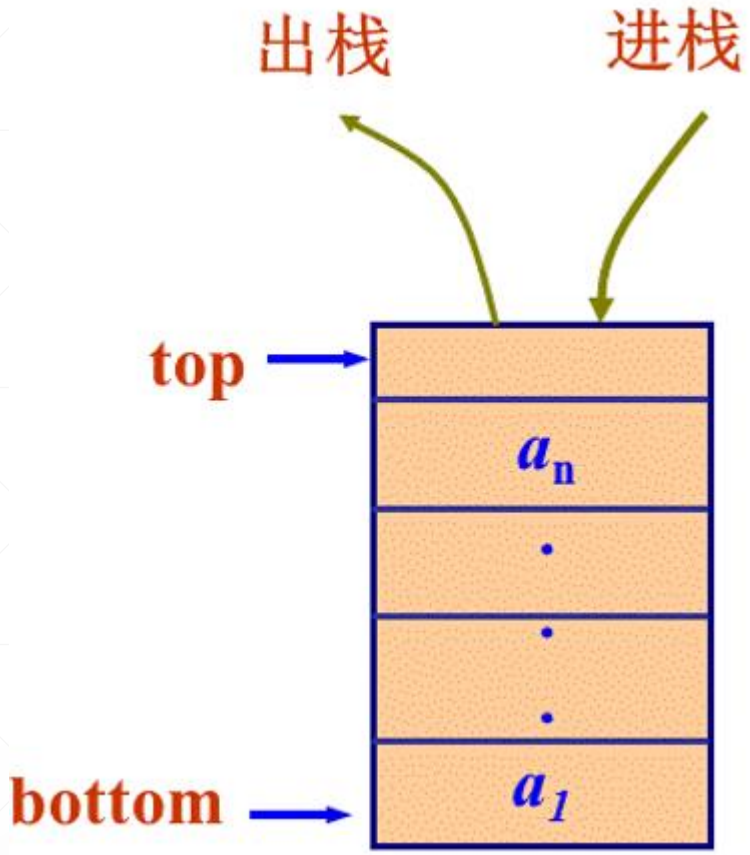
- ABC的所有排列都可能是出栈的序列吗？
- 可以产生CAB这样的序列吗？

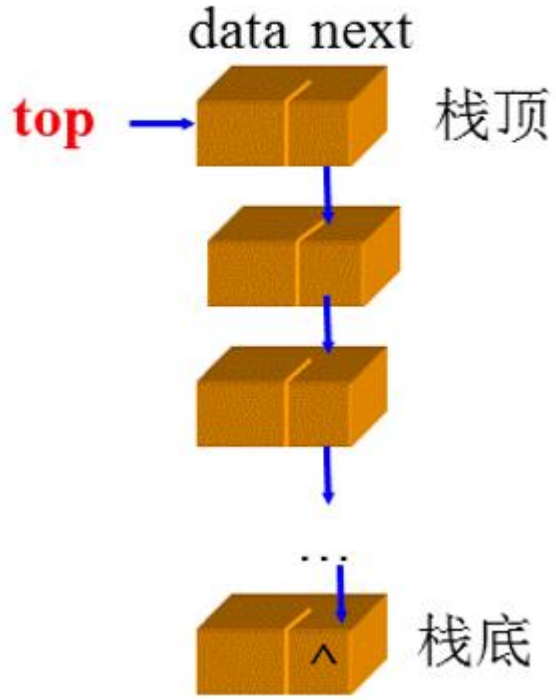
- 若1表示入栈操作，0表示出栈操作，若元素入栈顺序是ABCD，出栈顺序是ADCB，那么相应的1和0操作序列为？

10111000

# 栈的实现

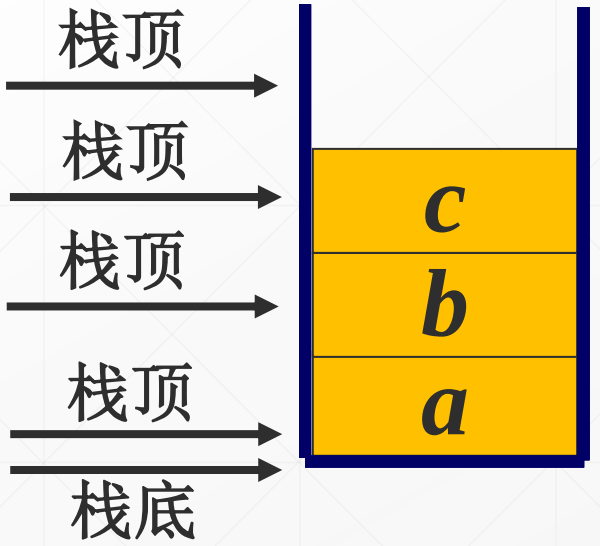
- 栈的存储结构主要有两种：
  - 顺序栈
  - 链式栈



# 顺序栈

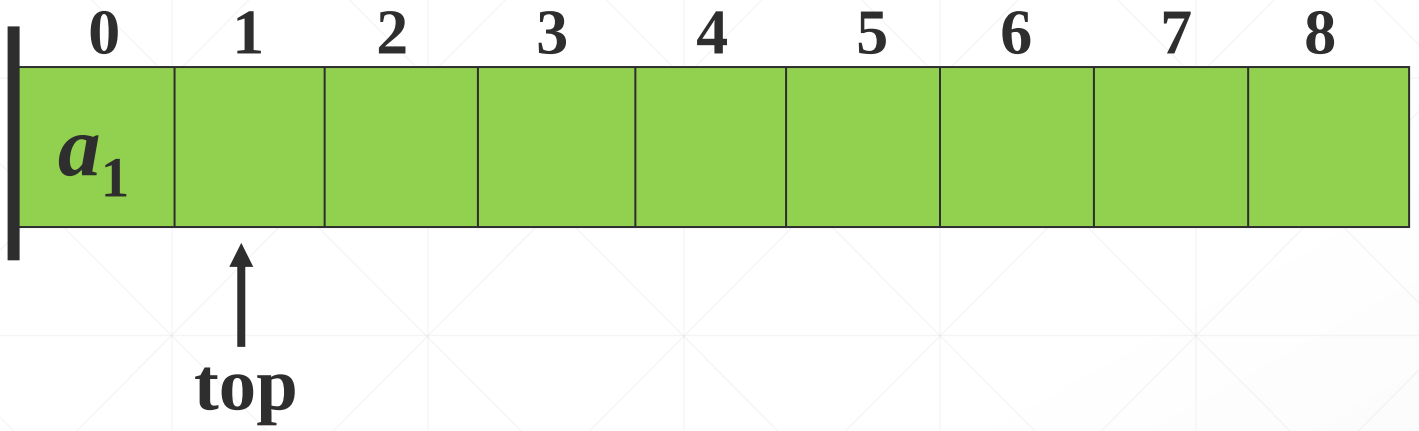
- 顺序栈是栈的顺序存储结构
- 利用一组地址连续的存储单元依次存放自栈底到栈顶的数据元素
- 指针top指向栈顶元素在顺序栈中的下一个位置
- base为栈底指针，指向栈底的位置




栈的特性：后进先出

# 顺序栈的顺序存储结构及实现

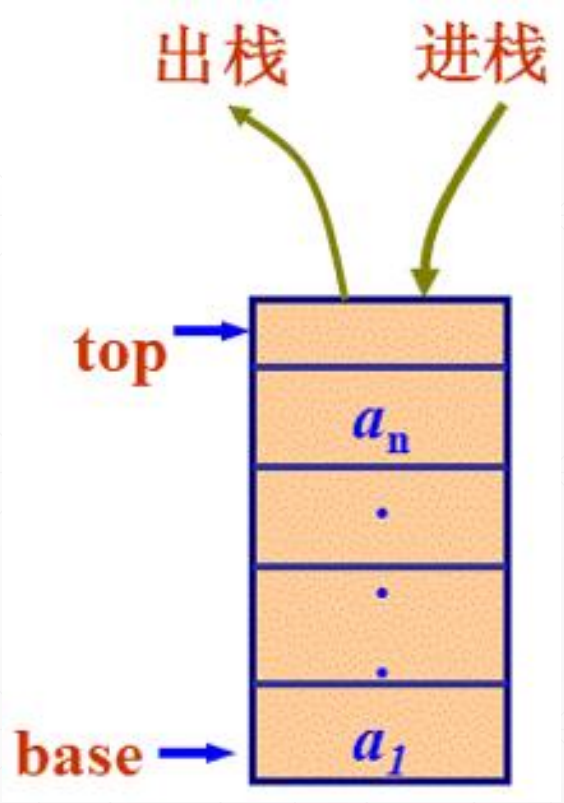
如何改造数组实现栈的顺序存储？




```
#define MAXSTACKSIZE    100     // 栈存储空间最大长度
struct SqStack{
        char base[MAXSTACKSIZE]; // 栈底指针，也是栈的基址
        char *top;                // 栈顶指针
};
```

# 顺序栈的特性

- top=base 表示空栈
- base=NULL 表示栈不存在
- 当插入新的栈顶元素时，指针 top + 1
- 删除栈顶元素时，指针 top - 1
- 当 top – base > MAXSTACKSIZE 时，栈满，溢出



# 顺序栈的抽象数据类型描述

- **ADT Stack {**

  数据对象： $D = \{a_i | a_i \in \text{ElemSet}, i = 1, 2, 3, \ldots, n, n \geq 0\}$

  数据关系： $R = \{< a_{i-1}, a_i > | a_{i-1}, a_i \in D \}$

  基本操作： **InitStack (&S)** // 构造一个空栈S

  **Push(&S, e)** // 进栈，插入元素e为新的栈顶元素；

  **Pop(&S, &e)** // 出栈，删除栈顶元素并用e返回其值；

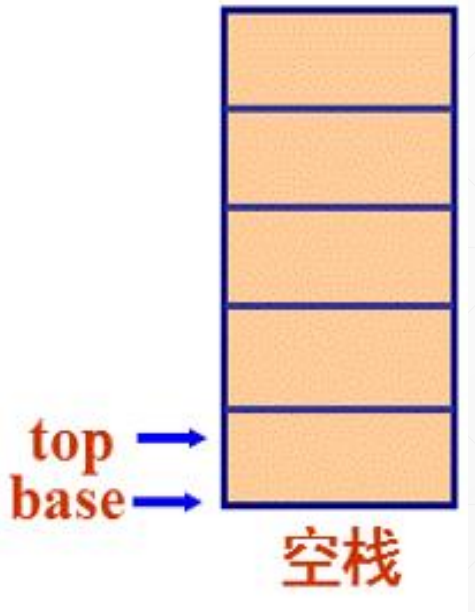
  **StackEmpty (&S)** // 栈是否为空

**} ADT Stack**

# 创建顺序栈

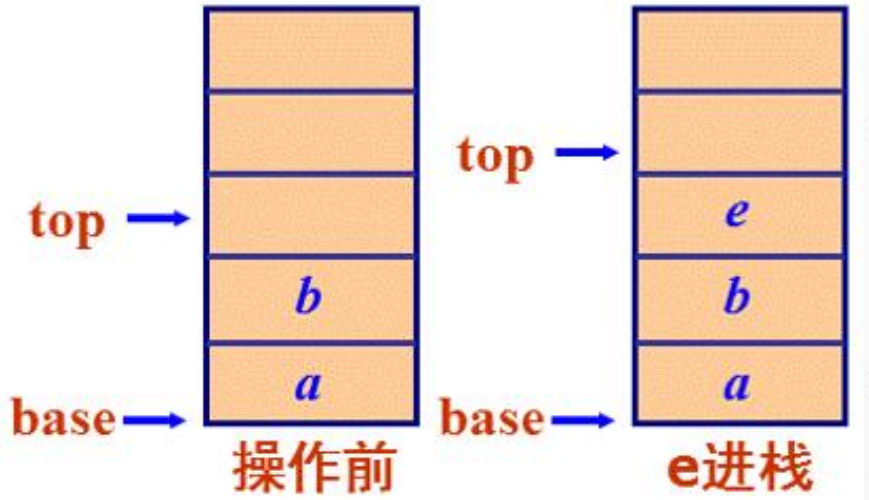
- 构造一个空栈S

```
int InitStack(SqStack &S){
if (S.base == NULL)   return {ERROR};
S. top=S. base;                    // 初始化堆栈（清空）
return (CORRECT) ;
}
```



# 进栈（插入新元素）

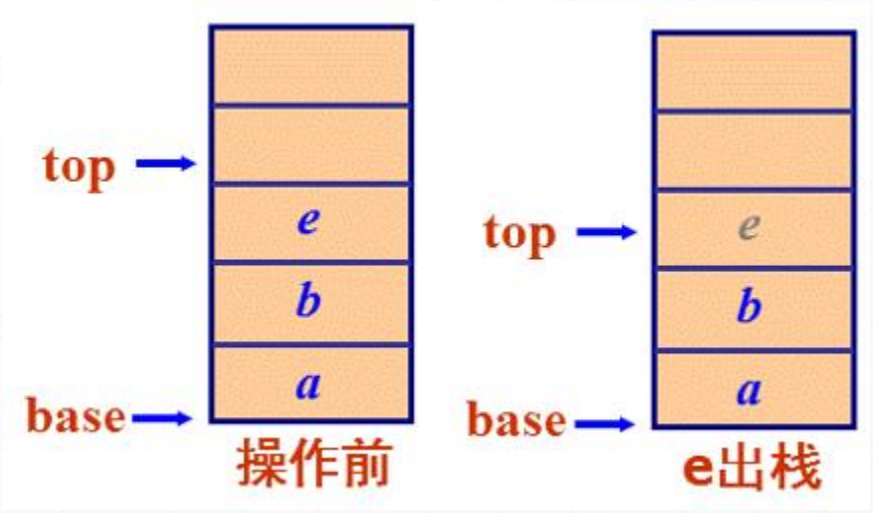
```
int Push(SqStack &S, char e)  //向栈中放入数据 [压栈]
{
   if ((S. top-S. base)  >=  MAXSTACKSIZE) return (ERROR);
   *S. top = e;
   S.top++;
   return (CORRECT);
}
```




}

# 出栈（删除元素）

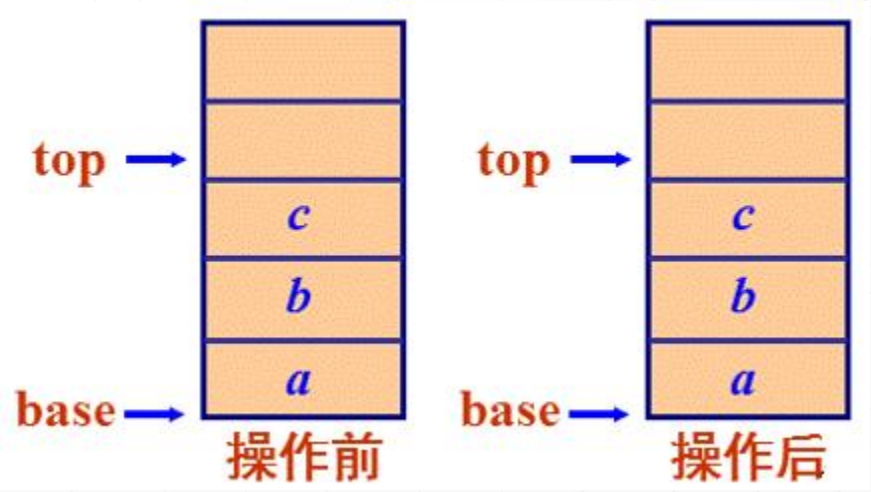
```
int Pop(SqStack &S, char &e)        //从栈中取数据[弹栈]
{
   if  (S. top <= S. base)  return (ERROR);
   S. top--;
   e=*S. top;
  return (CORRECT);
}
```



# 栈是否为空

- 如果栈为空，返回-1
- 否则，返回栈内元素数目

```
int StackEmpty (SqStack &S)
{
    if (S.  top <= S. base)
        return (ERROR) ;
     return (S. top-S. base);
}
```

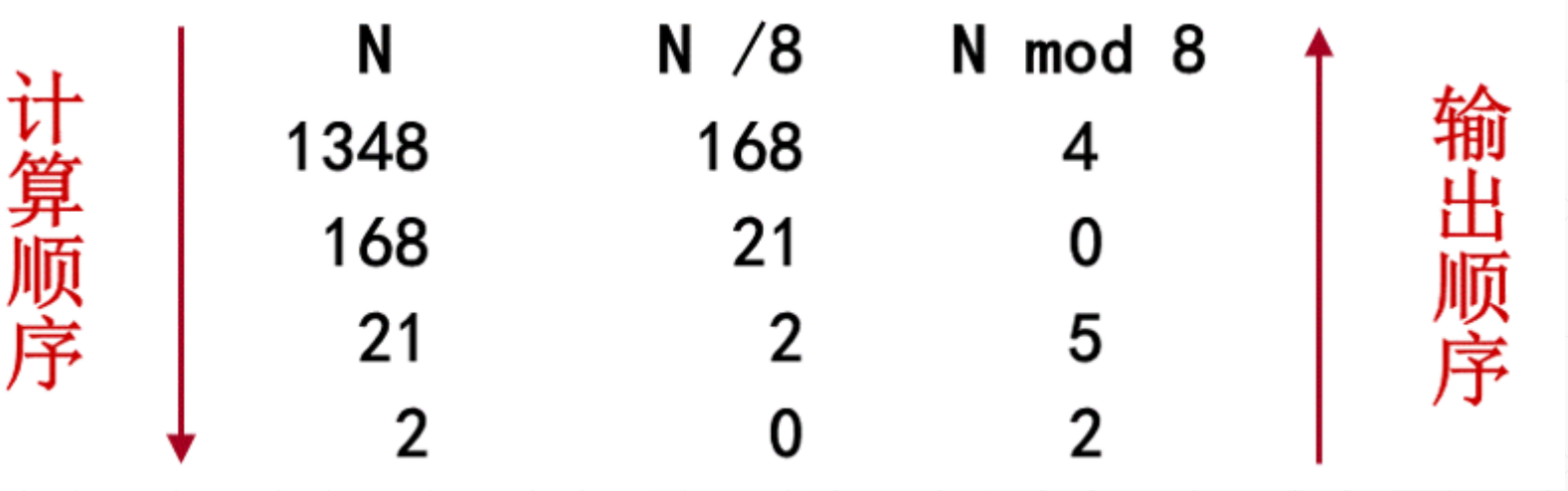


# 数值转换（八进制）

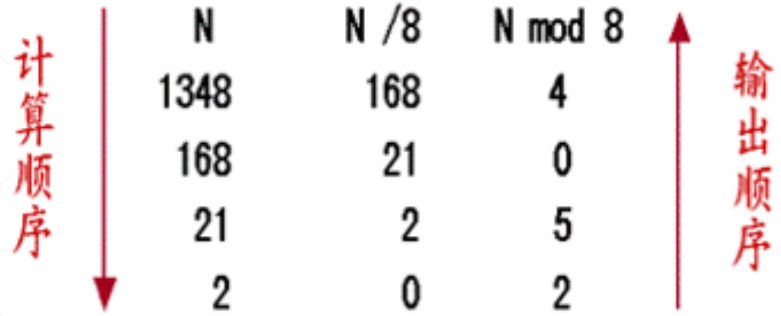
- 将十进制转换为其他进制（d），

  其原理为： N = （N/d）*d + N mod d

  例如： （1348）10=（2504）8



# 数值转换（八进制）




```
▪ void conversion () {
      initStack (S);                    // 创建空栈 S
      cin  >> N;                        // 输入一个十进制数 N
      while (N) {
          Push (S,  N %8)               // 将余数送入栈中
          N = N/8;                      // 求整除数
}
      while (StackEmpty(S) > -1){       // 如果栈不空
         Pop (S, e);                    // 将栈中数出栈
         cout  << e;
        }
} //conversion
```

# 行编辑程序

- 用户输入一行字符
- 允许用户输入出差错，并在发现有误时，可以用退格字符“#”及时更正
- 假设从终端接受一行字符:

**whli##ilr#e (s#*s)**

实际有效行为:

**while (*s)**

# 行编辑程序

- 对用户输入的一行字符进行处理，直到行结束（“\n”）

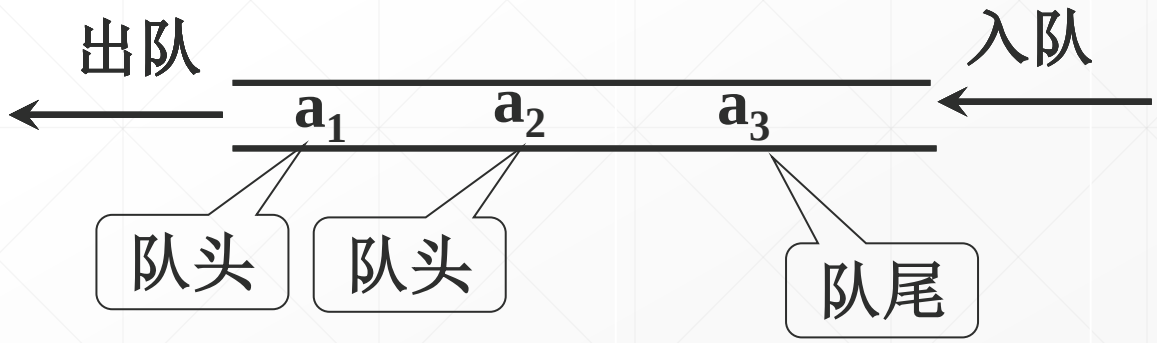
```
ch = getchar ();                                    // 从终端输入第一个字符
while (ch !=’\n’) {
    switch (ch) {
        case ’#’: Pop(S, c); break;        // 仅当栈非空时退栈
        default:     Push(S, ch); break;   // 有效字符进栈
    }
    ch = getchar();                                 // 从终端输入一个字符
}
```

将从栈底到栈顶的栈内字符传送到调用过程的数据区；

# 队列

- 队列是只允许在表的一端进行插入，而在另一端删除元素的线性表。
- 在队列中，允许插入的一端叫队尾（rear），

  允许删除的一端称为队头（front）。
- 特点：先进先出（FIFO）



# 顺序队列

- 顺序队列：采用一组地址连续的存储单元依次存储从队列头到队列尾的元素

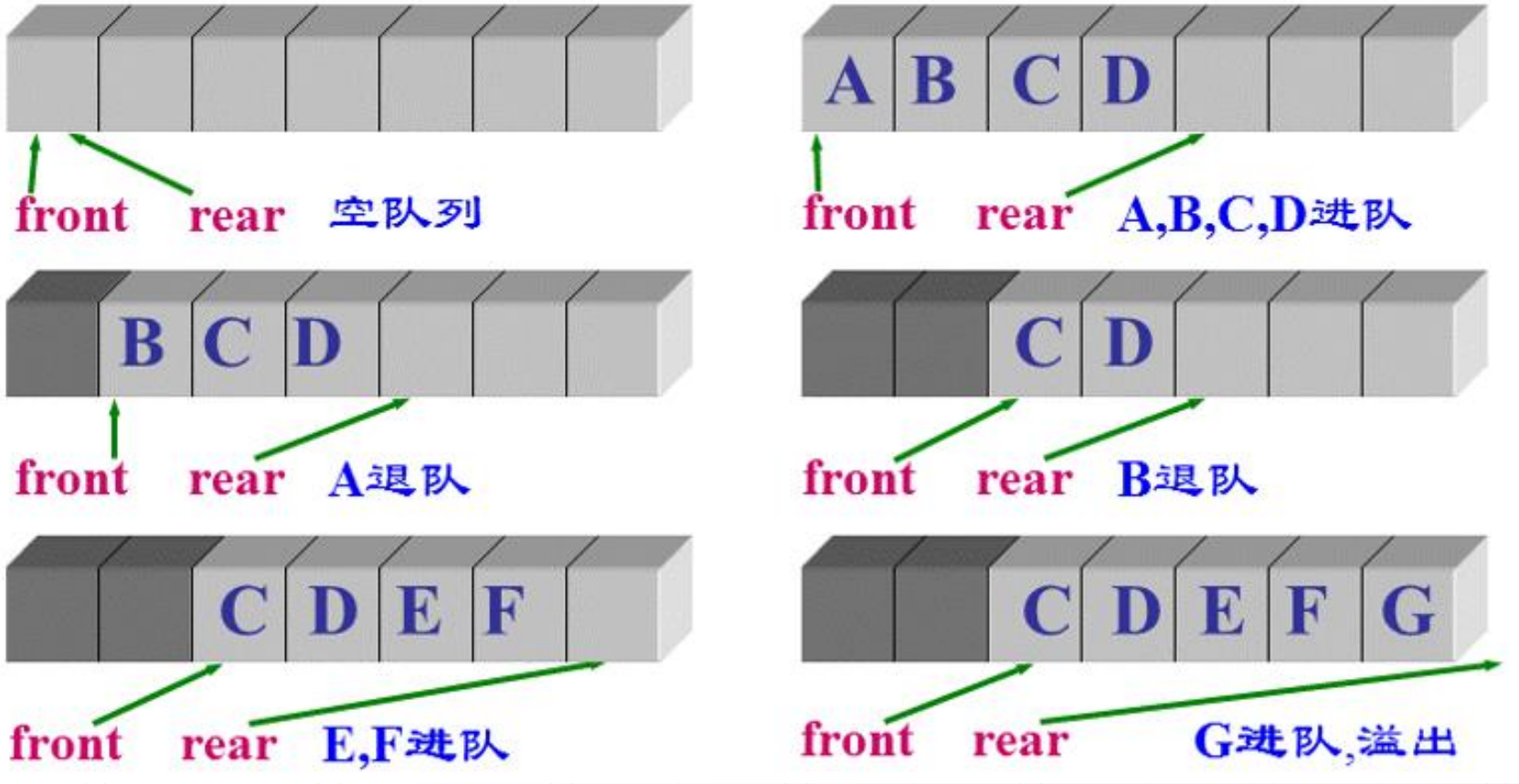
- 顺序队列有两个指针：
  - 队头指针 **front**
  - 队尾指针 **rear**

# 顺序队列的进队和出队原则

- 进队时，新元素按rear指针位置插入，然后队尾指针增一，即 rear = rear + 1
- 出队时，将队头指针位置的元素取出，然后队头指针增一，即 front = front + 1

- 队头指针始终指向队列头元素
- 队尾指针始终指向队列尾元素的下一个位置

# 顺序队列的进队和出队举例



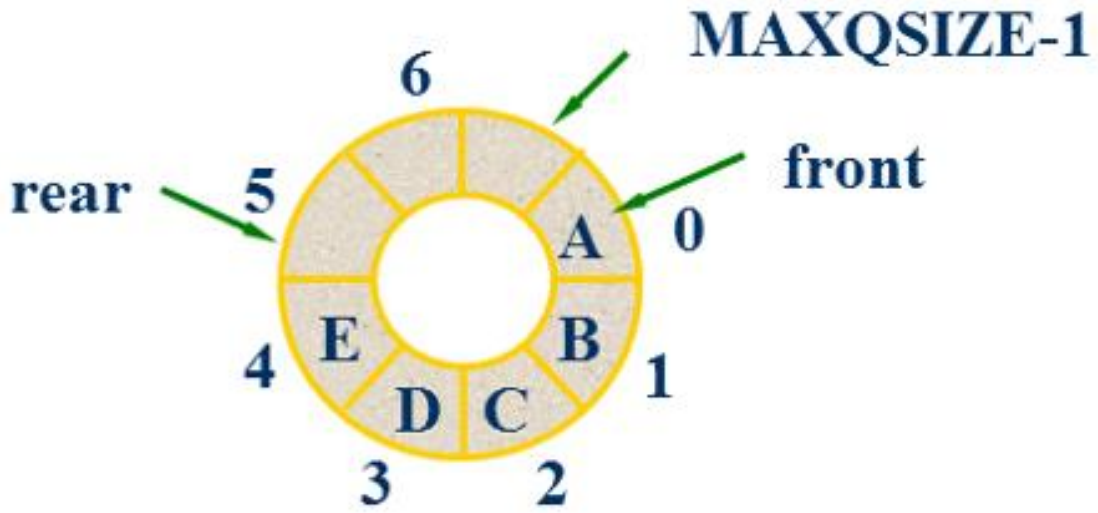
# 顺序队列存在的问题

- 当队尾指针指向队列存储结构中的最后单元时，如果再继续插入新的元素，则会产生溢出
- 当队列发生溢出时，队列存储结构中可能还存在一些空白位置（已被取走数据的元素）

- 解决办法之一：将队列存储结构首尾相连，形成循环（环形）队列

# 循环队列

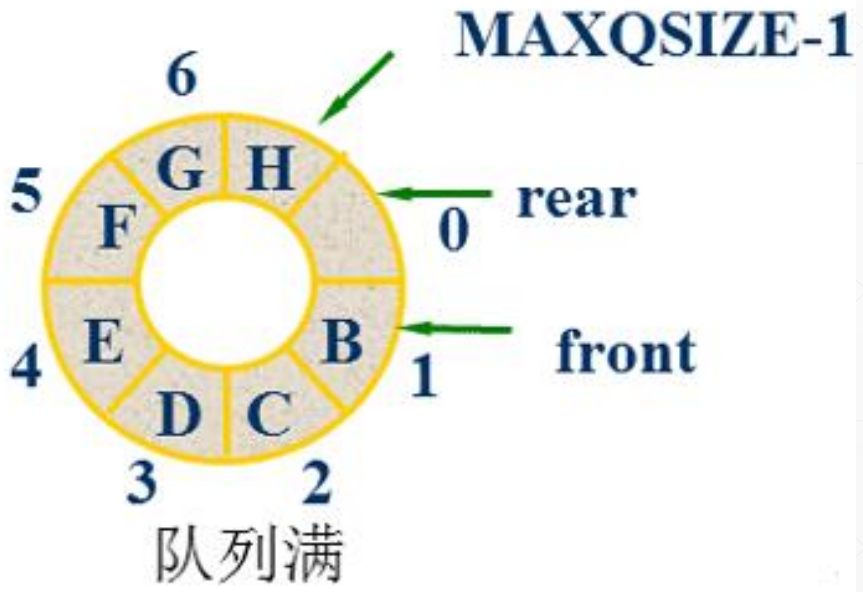
- 循环队列采用一组地址连续的存储单元
- 将整个队列的存储单元首尾相连



# 循环队列空与满

- front = rear, 循环队列空
- (rear+1) % MAXQSIZE = front, 循环队列满

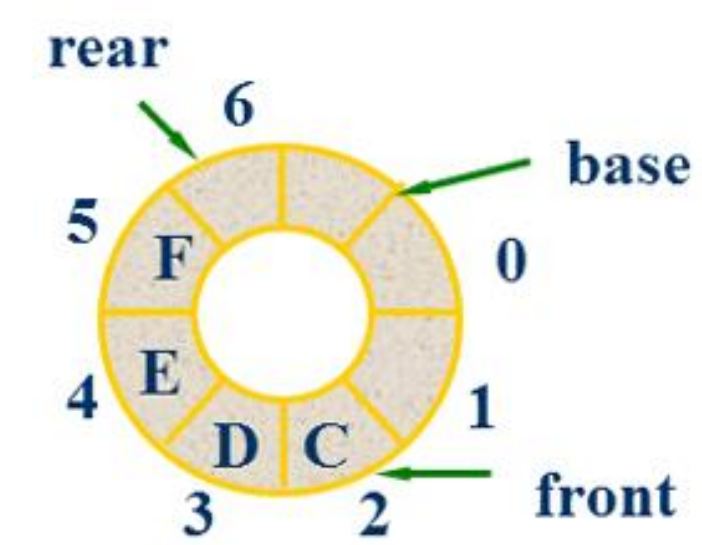
队列空



队列满

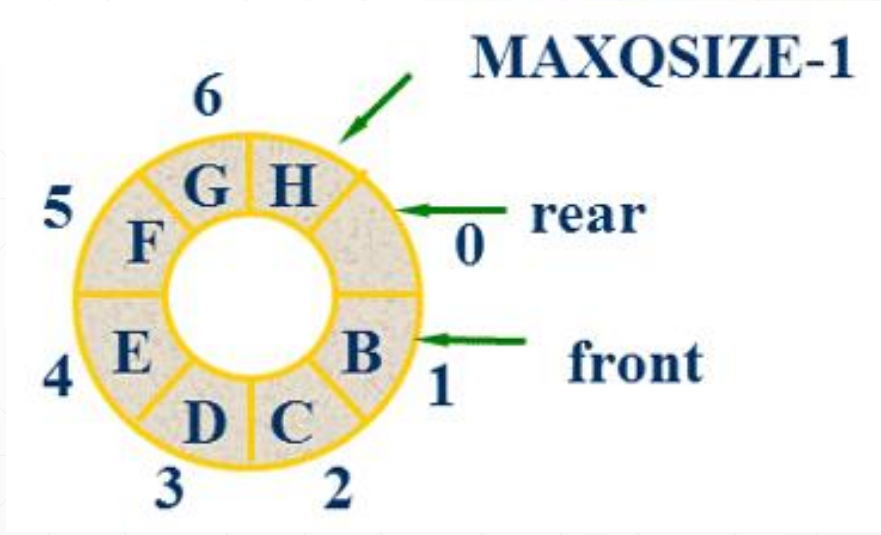
# 循环队列定义

```
#define MAXQSIZE 8
struct SqQueue {
    int   base[MAXQSIZE];
    int   front;
    int   rear;
};
```



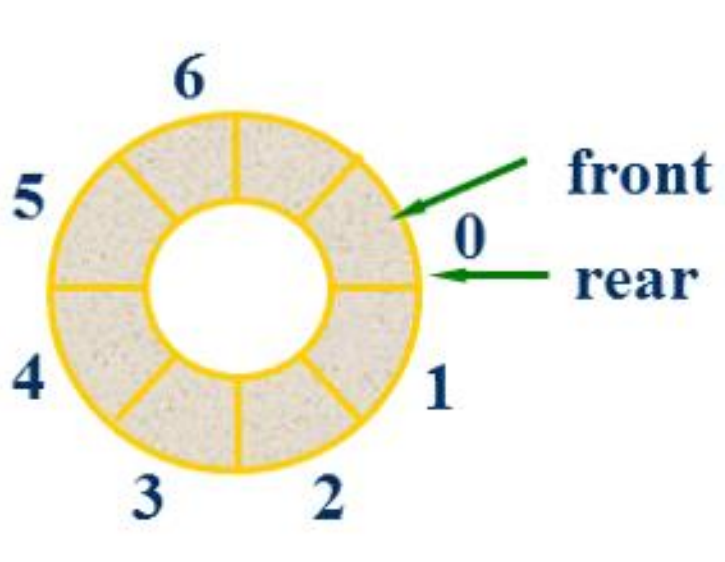
# 循环队列插入元素

```
int EnQueue(SqQueue &Q, int e) {
   if ((Q. rear + 1)%MAXQSIZE == Q.front)
        return (ERROR); // 队满
   Q. base [Q. rear] = e;
   Q. rear = (Q. rear + 1) % MAXQSIZE;
   return (CORRECT);
}
```



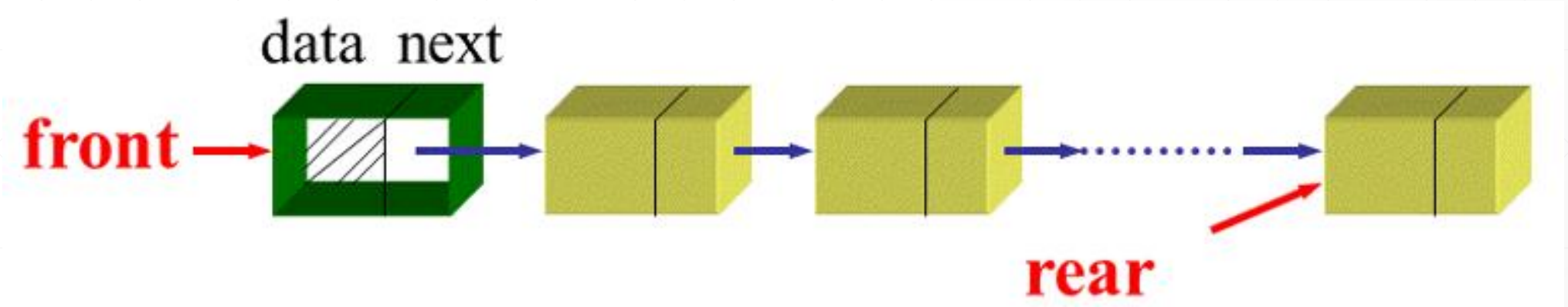
# 循环队列删除元素

```
int DeQueue(SqQueue &Q, int e) {
   if (Q. rear = Q. front)   return (ERROR);   // 队空
   e = Q. base [Q. front] ;
   Q.front = (Q. front + 1) %  MAXQSIZE;
   return (CORRECT);
}
```



# 链队列

- 链队列采用链表存储单元
- 链队列中，有两个分别指示队头和队尾的指针
- 链式队列在进队时无队满问题；但出队时，有队空问题



# 链队列指针变化情况

- 链队列是链表操作的子集