问题求解与实践
——栈

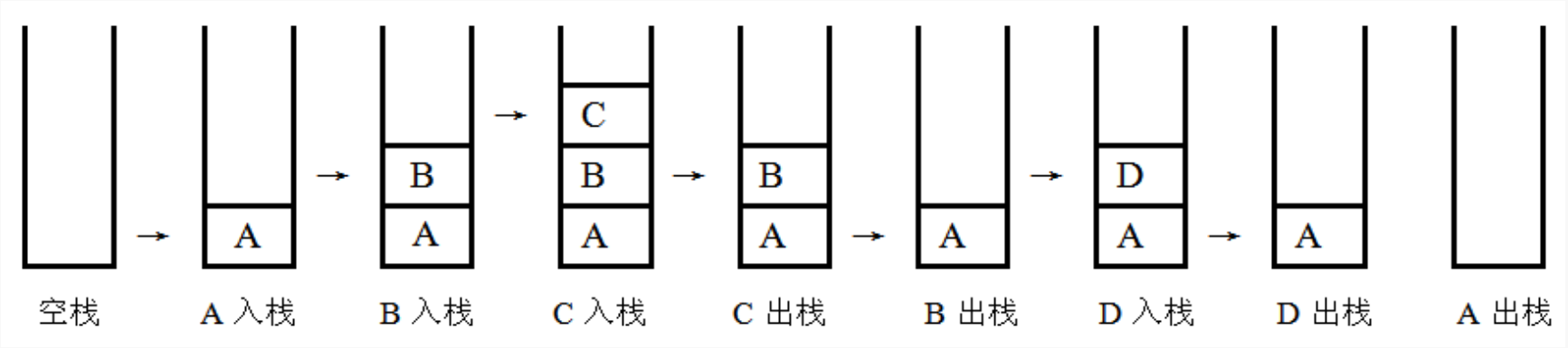
主讲教师：陈雨亭、沈艳艳

栈

- **栈**(Stack)
  - 只能在一端进行插入和删除操作的特殊线性表
  - 允许进行插入和删除操作的一端称为**栈顶**，另一端称为**栈底**
- 你举出一些栈的生活实例吗？

栈

- 栈的特点：
  - **后进先出**（LIFO, last in, first out ）或**先进后出**（FILO）



栈的主要操作

- **创建空栈**
- **入栈**（push）：也称为进栈、压栈，在栈顶加入新的元素
- **出栈**（pop）：也称退栈或弹栈，将栈顶元素删除
- **读栈顶元素**：只读出栈顶元素，但不出栈

栈的应用

- 进制转换(除2取余法)
- 括号匹配检查
- 引号匹配检查
- 递归算法

顺序栈的定义

```
struct SqStack{
        ElemType *data;     //存储元素的变量
        int top;            //栈顶指针，存储栈顶元素的下标
        int stacksize;      //堆栈最大可分配空间数量，以元素为单位
};
SqStack  s;                 //定义一个堆栈s
```

初始化栈

```
void InitStack(SqStack *s, int size )
{
        if( size > 0 )
        {
                s->stacksize = size;
                s->top = -1;
                s->data = new  ElemType[size];          // 申请空间
        }else
                cout<<"堆栈初始化长度错误";
}
```

入栈

```
void Push(SqStack *s, ElemType x)
{
        if(s->top < s->stacksize-1)
        {
                s->top++;
                s->data[top]=x;
        }else
                cout<<“栈满”;
}
```

出栈

```
void Pop (SqStack *s, ElemType &e)
{
        if(s->top > -1)
        {
                e= s->data[s->top];
                s->top--;
        }else
                cout<<“栈空”;
}
```

取栈顶

```
void GetTop(SqStack *s, ElemType &e)
{
        if(s->top > -1)
                e= s->data[s->top];
        else
                cout<<“栈空”;
}
```

链式栈

- 只能在头部插入删除元素的单链表;
- 一般很少用链式结构来表示栈(为什么？)

问题求解与实践
——队列

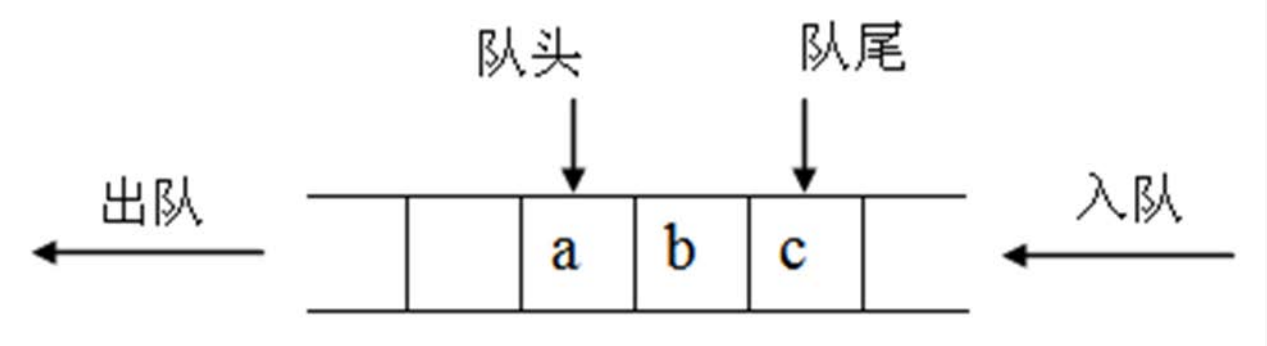
主讲教师： 陈雨亭、沈艳艳

队列

- 在日常生活中，购物交款时，总会遇到排队的问题，在排队时要遵守的原则是
  - 后来的人只能排在队伍的后面（增加元素）
  - 排在前面的人得到服务后就可以离开了（删除）
  - 即：先进来的先出去

队列

- **队列**（Queue）
  - 只能在表的一端进行插入操作、在另一端进行删除操作的特殊线性表
  - 允许删除元素的一端称为**队头**，允许插入元素的一端称为**队尾**
- 特点
  - **先进先出**（FIFO）或**后进后出**（LILO）



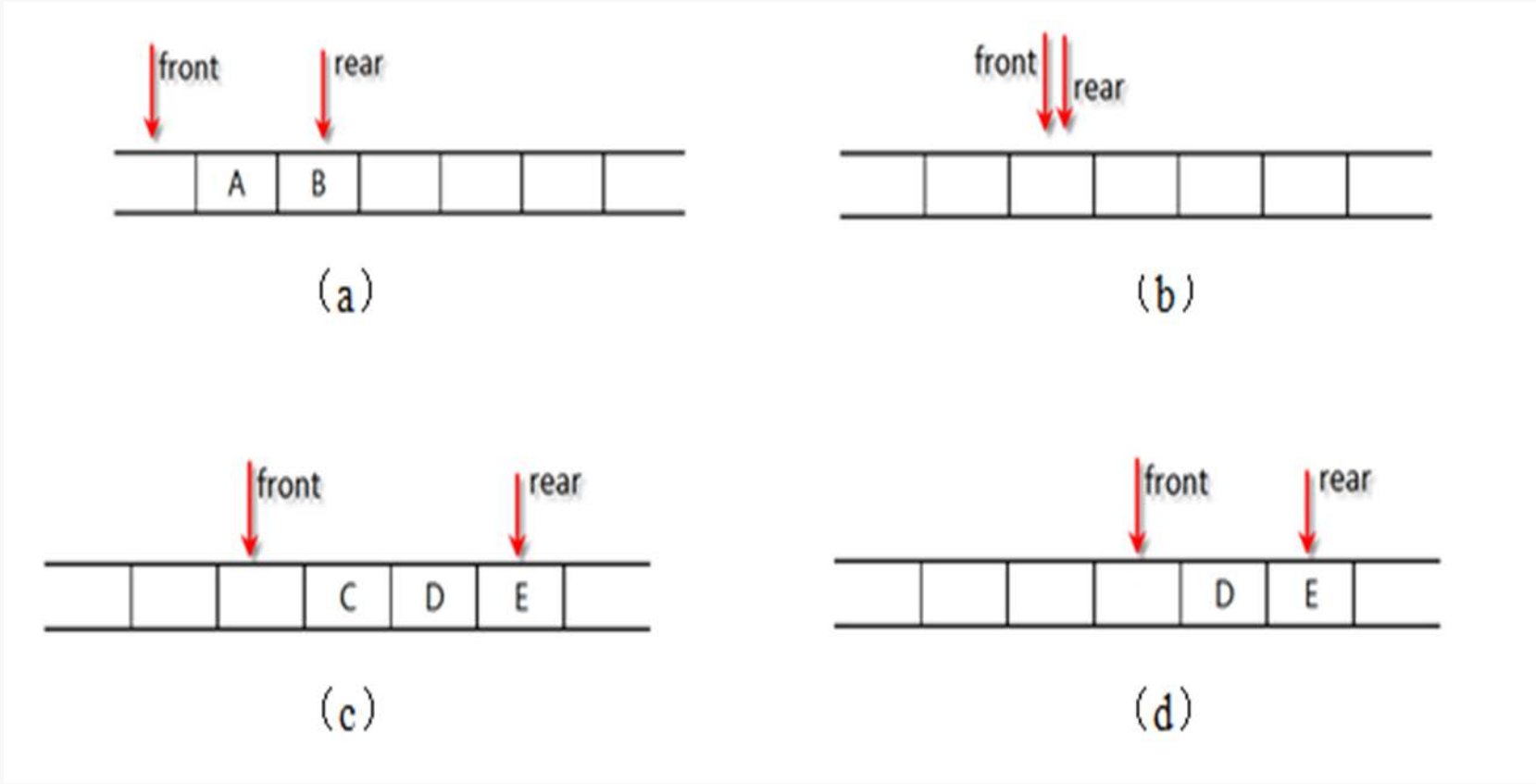
队列的主要操作

- 队列的主要操作
  - 判断队列是否满
  - 判断队列是否为空
  - **入队**：在队尾插入元素
  - **出队**：在队头删除元素
  - 取队头元素

队列的主要应用

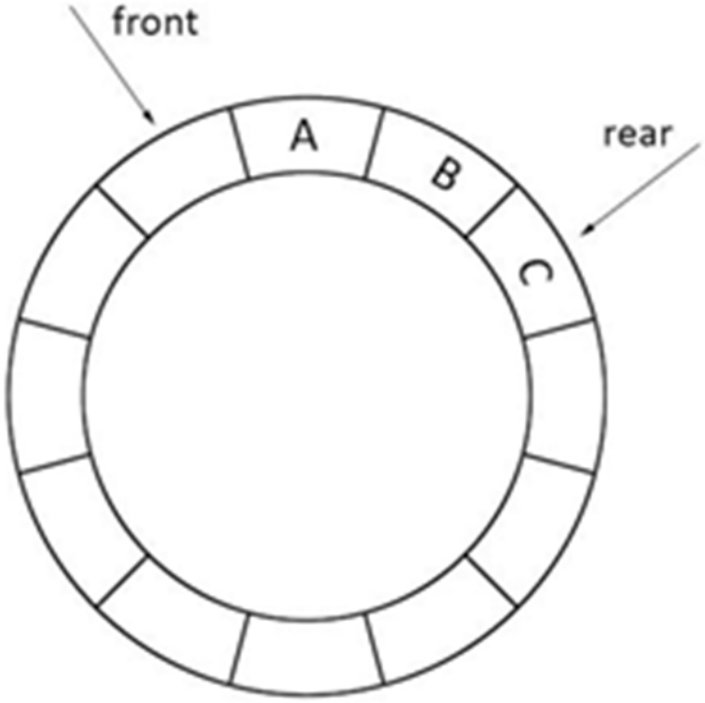
- 缓存
  - 打印队列
  - 发送手机短信
  - 发送电子邮件

队列



循环队列

- 将队列的头尾相连形成一个圆圈
- 当队尾和队头重叠时，队列为空还是满呢?
- 约定：当队头和队尾相等时，队空。当队尾加1后等于队头时，队满



队列的定义

```
const int MAX=100;
struct SqQueue{
        int data[MAX];          // 存放元素的数组
        int front;              // 队头指针
        int rear;               // 队尾指针
};
SqQueue Q;                      // 定义队列q
Q.front=Q.rear=0;               // 指针初始化
```

入队

```
void  EnQueue(SqQueue &Q , ElemType x)
{
        if((Q.rear+1)% MAX == Q.front)  cout<<"队列已满";
        else {
                Q.rear = (Q.rear+1)% MAX;
                Q.data[Q.rear]=x;
        }
}
```

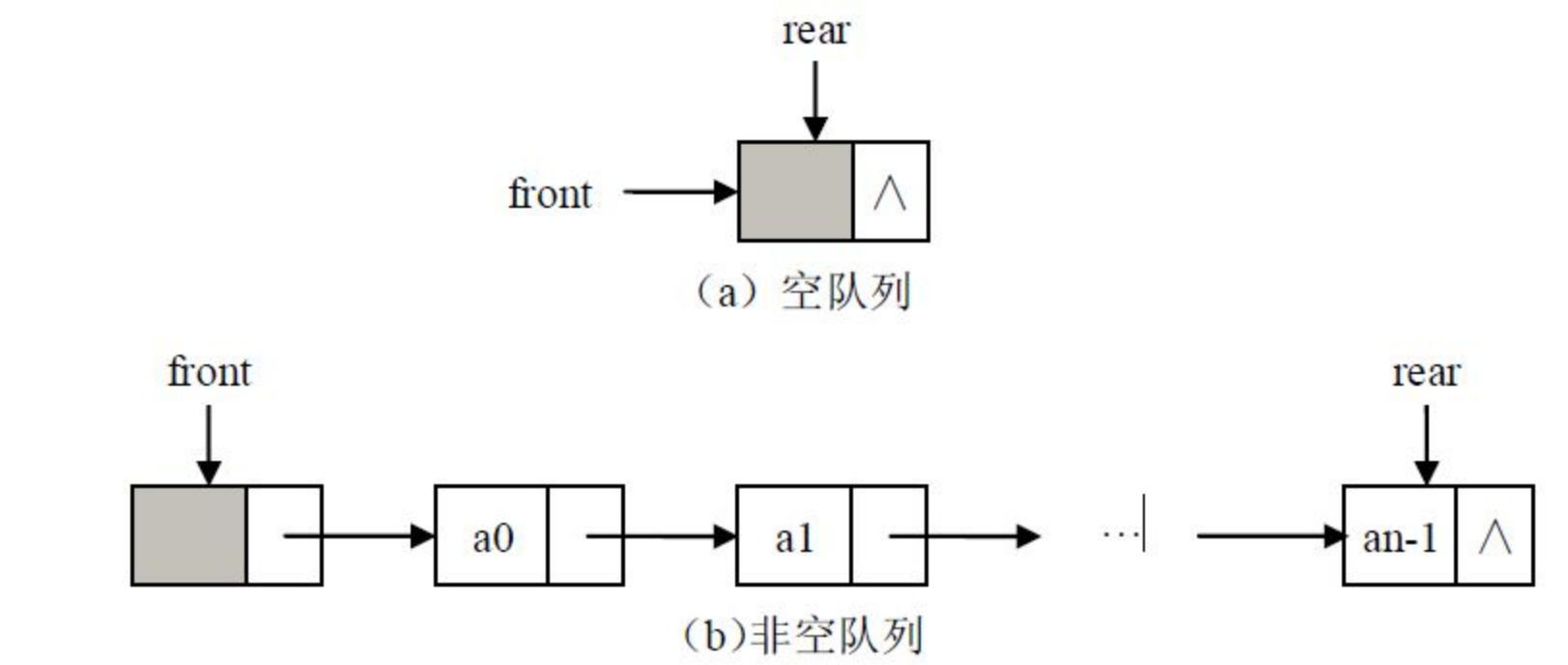
出队

```
ElemType  DeQueue(SqQueue &Q)
{
        if(Q.rear==Q.front)  cout<<"队列已空";
        else {
                Q.front = (Q.front +1)% MAX;
        }
        return Q.data[Q.front];
}
```

取队头元素

```
ElemType  GetHead(SqQueue &Q)
{
int i;
        if(Q.rear==Q.front)  cout<<"队列已空";
        else {
                int i=(Q.front +1)% MAX;
        }
        return Q.data[i];
}
```

链式队列



链式队列的定义

```
struct QNode {
        ElemType data;
        struct QNode *next;
};
struct LinkQueue {
        QNode *front;                    // 队头指针
        QNode *rear;                     // 队尾指针
};
```

链式队列初始化

```
void InitQueue(LinkQueue &Q)
{
        Q.front = new QNode;     //建立头结点
        Q.front->next=NULL;
        Q.rear = Q.front;        //尾指针也指向头结点
}
```

链式队列求长度函数

```
int QueueLength(LinkQueue &Q)
{
        QNode *p=Q.front;
        int len=0;
        while(p!=Q.rear){
                len++;
                p= p->next;
        }
        return len;
}
```

链式队列入队

```
void EnQueue(LinkQueue &Q, ElemType x)
{
        QNode *s=new QNode;  //建立新结点s
        s->data = x;
        s->next =NULL;
        Q.rear ->next = s;          //在队尾插入结点s
        Q.rear = s;                 //修改队尾指针
}
```

链式队列出队

```
void DeQueue (LinkQueue &Q, ElemType &e)
{
        QNode *p;
        if(Q.front==Q.rear) cout<<"队列已空";
        else {
                p= Q.front->next;
                e=p->data;
                Q.front->next=p->next;
                if(Q.rear==p) Q.rear=Q.front;  //删除最后一个元素时修改尾指针
        delete p;
}
```

链式队列取队头元素

```
void GetHead(LinkQueue &Q, ElemType &e)
{
        QNode *p;
        if(Q.front==Q.rear) cout<<"队列已空";
        else {
                p= Q.front->next;
                e=p->data;
        }
}
```

思考

- 和栈的链式形式相比，链式队列还是有一定的应用的，为什么？