队列

一种先进先出的运算受限的线性表，只在表的一端插入，另一端删除

队首（front）：只允许删除
队尾（rear）：只允许插入

基本操作：

- Create(); //创建一个空队列
- EmptyQue(); //判空
- InsertQue(x); //向队尾插入元素
- DeleteQue(x); //删除队首元素

顺序队列

利用一维数组（连续存储单元）存储队列，和线性表一样由着动、静态之分

静态顺序队列

```
#define MAX_QUEUE_SIZE 100

/*初始化*/
typedef struct queue{
    int queueArray[MAX_QUEUE_SIZE];
    int front;  //始终指向头元素
    int rear;   //始终指向队尾元素的下一位
}sqQueue;
```

入队：
  先放元素
  rear++
出队：
  先出元素
  front++

此操作容易假溢出，解决方案：

- 引入length：
  - 入队：length++
  - 出队：length--
  - 此时，length==0为空；length==n为满。入队和出队共享资源，需要互斥访问，需要相关的进程管理机制，对结构进行控制
- 引入一个flag：
  - 当队列满的时候flag=1
  - 反之flag=0

注意：**以上两种方法容易造成更复杂的管理方式**

- 将队列看成一个首尾相连的队列，形成**循环队列**

循环队列【重点】

循环队列舍弃一个空间用于判断队空、队满

```
if(i+1 == MAX_QUEUE_SIZE) i = 0;
else i++;

//初始化
rear = front = 0

//循环队列为空
front == rear

//循环队列满
front == (rear+1) % MAX_QUEUE_SIZE

//入队
rear = (rear+1) % MAX_QUEUE_SIZE

//出队
rear = (rear+1) % MAX_QUEUE_SIZE

//计算元素个数
len = (rear - front + MAX_QUEUE_SIZE) % MAX_QUEUE_SIZE

/*
如果非空状态下，front指向队首元素的前一个空位，rear指向队尾
入队、出队需要分别让rear、fornt先++，然后再执行操作
同时初始化时，需要rear = front = n-1
*/

/*
如果非空状态下，front指向队首，rear指向队尾，需要引入length字段
判空：front == rear；length = 0
判满：front == rear；length = n
入队时先rear++，再入队
出队时先出队，再front++
*/
```

循环队列的初始化，入队、出队、计算长度

```
/*循环队列初始化*/
sqQueue initCirQueue(void){
    sqQueue Q;
    Q.front = Q;
    Q.rear = 0;
```

```
    return(Q);
}

/*入队*/
int insertCirQueue(sqQuene Q, int e){
    if((Q.rear+1) % MAX_QUEUE_SIZE == Q.front)  return 0;    //判满
    Q.queueArray[Q.rear] = e;    //插入e
    Q.rear = (Q.rear+1) % MAX_QUEUE_SIZE     //入队指针移动
    return 1;
}

/*出队*/
int deleteCirQueue(sqQueue Q, int *x){
    if(Q.front == Q.rear) return 0;      //判空
    *x = Q.queueArray[Q.front];      //取出队首元素
    Q.front = (Q.front+1) % MAX_QUEUE_SIZE; //移动首指针
    return 1;
}
```

链式队列

限制表头删除、表尾插入的单链表

头出：头指针始终指向头结点
尾插：尾指针始终指向尾节点

定义

```
/*定义数据指针节点*/
typedef struct QNode{
    int data;
    struct Qnode *next;
}qNode;

/*定义首尾指针结点*/
typedef struct LinkQueue{
    qNode *fornt, *rear;
}linkQueue;
```

链队运算以及指针变化

- 若带头结点：
    1. 插入：尾改，头不改
    2. 删除：
        1. **当删除最后一个节点时，头尾均要改**
        2. 其它情况头改，尾不改
- 若不带头结点：
    1. 插入：
        1. **插入第一个结点，头尾均要改**

2. 插入其他结点时，尾改，头不改

2. 删除：

1. 当删除最后一个结点，头尾均要改
2. 其它情况头改，尾不改

链队列的初始化，入队、出队

```
/*初始化*/
linkQueue *initLinkQueue(void){
    linkQueue *q;
    qNode *p;
    p = (qNode *)malloc(sizeof(qNode)); //开辟头结点
    p->next = NULL;
    q = (linkQueue *)malloc(sizeof(linkQueue)); //开辟联队指针结点
    q.front = q.rear = p
    return(Q);
}

/*入队*/
int insertCirQueue(linkQueue *q, int e){
    p = (qNode *)malloc(sizeof(qNode));
    if(!p) return 0;  //结点申请失败
    p->data = e;
    p->next = null;
    q.rear->next = p;
    q.rear = p; //在队尾插入新节点
    return 1;
}

/*出队*/
int deleteLinkQueue(linkQueue *q, int *x){
    qNode *p;
    if(q.front == q.rear) return 0;    //判空
    p = q.front->next;  //取队首结点
    *x = p->data;
    q.front->next = p->next;     //修改队首指针
    if(p == q.rear) q.rear = q.front;    //队列只有一个结点时，防止丢失队尾指针
    free(p);
    return 1;
}
```

双端队列（2010、2021）

栈和队列的组合体，分为两种类型：

- 输出受限：两端均可插入，限制一端删除
- 输入受限：两端均可删除，限制一端插入

队列的应用

- 排队业务、打印机服务、挂号系统、etc.
- 树的层次遍历
- 图的广度优先遍历

补充

- 用单链表的队列的几种形式：
    1. 带有头指针和尾指针，最简便
    2. 带尾指针的单循环链表
    3. 双向链表：
        1. 带尾指针的双向循环列表
        2. 带头指针的双向循环列表
- 对于第一个进入队列的存储位置在A[0]的循环队列中（数组A[0..n-1]用于存储），初始时front=0，rear=n-1
- 不带头结点的链队列在出队操作时，修改尾指针的情况发生在出队后队列为空的时候