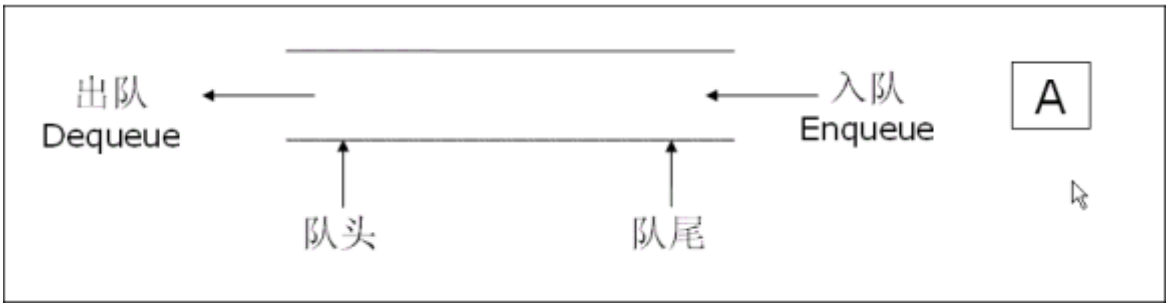
数据结构——队列

1.队列的结构及概念

队列：只允许在一端进行插入数据操作，在另一端进行删除数据操作的特殊线性表，队列具有先进先出FIFO(First In First Out)

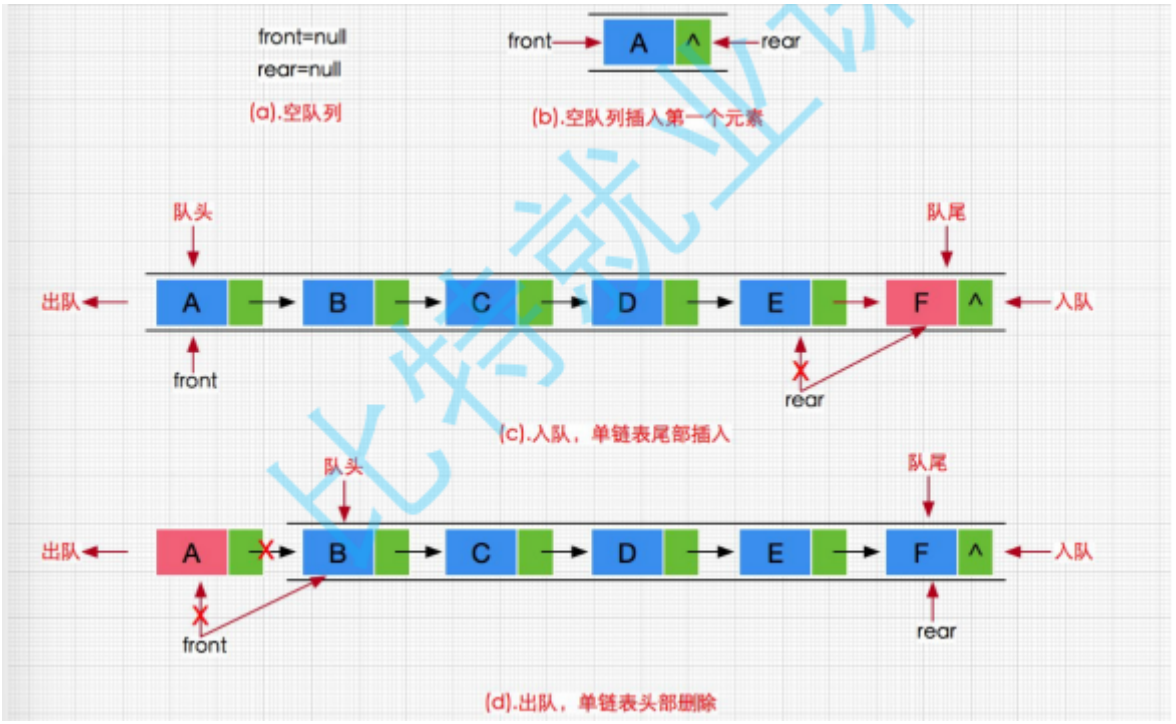
入队列：进行插入操作的一端称为**队尾**

出队列：进行删除操作的一端称为**队头**



2.队列的实现

队列也可以由**数组和链表**的结构实现，使用链表的结构更优一些，因为数组结构在出队列上**是在对头出数据的**，需要挪动数据，效率较低。



队列的头文件——申明

```
//Queue.h
#include<stdio.h>
#include<stdlib.h>
#include<assert.h>
#include<stdbool.h>
typedef int QueueDataType;
//创建队列节点
typedef struct QueueNode
```

```
{
    struct QueueNode* next;
    QueueDataType val;

}QNode;
//创建队列
typedef struct Queue
{
    QNode* head;//创建头节点方便头删
    QNode* tail;//创建尾节点方便尾插
    int size;//计队列的节点数

}Queue;
//队列的初始化和销毁
void QueueInit(Queue* q);
void QueueDestroy(Queue* q);
//入队列和出队列
void QueuePush(Queue* q,QueueDataType x);
void QueuePop(Queue* q);
//获取队列的队头元素和队尾元素
QueueDataType QueueFront(Queue* q);
QueueDataType QueueBack(Queue* q);
//队列的判空
bool QueueEmpty(Queue* q);
//队列中有效元素个数
int QueueSize(Queue* q);
```

```
//Queue.c
#include"Queue.h"
void QueueInit(Queue* q)
{
    q->head = q->tail = NULL;
    q->size = 0;
};
void QueueDestroy(Queue* q)
{
    QNode* cur = q->head;
    while (cur != NULL)
    {
        QNode* next = cur->next;
        free(cur);
        cur = next;
    }
    q->head = q->tail = NULL;
    q -> size = 0;

};
bool QueueEmpty(Queue* q)
{
    return q->head == NULL &&
        q->tail == NULL;
};
```

```c
void QueuePush(Queue* q, QueueDataType x)
{
    if (QueueEmpty(q))
    {
        QNode* newnode = (QNode*)malloc(sizeof(QNode));
        newnode->val = x;
        newnode->next = NULL;
        if (newnode == NULL)
        {
            perror("malloc failed");
            return;
        }
        q->head = newnode;

        q->tail = newnode;
        q->size++;
    }
    else {
        QNode* newnode = (QNode*)malloc(sizeof(QNode));
        newnode->val = x;
        newnode->next = NULL;
        if (newnode == NULL)
        {
            perror("malloc failed");
            return;
        }
        q->tail->next = newnode;
        q->tail = newnode;
        q->size++;
    }
};
void QueuePop(Queue* q)
{
    assert(!QueueEmpty(q));
    QNode* next = q->head->next;
    q->head = next;
    q->size--;
    //只有单一节点时
    if (q->head == NULL)
    {
        q->tail = NULL;
    }
}
QueueDataType QueueFront(Queue* q)
{
    assert(!QueueEmpty(q));

    return q->head->val;
};
QueueDataType QueueBack(Queue* q)
{
    assert(!QueueEmpty(q));
    return q->tail->val;
}
int QueueSize(Queue* q)
{
```

```
    return q->size;
}
```