队列

回顾上节课所学知识点

- 堆栈
- 如何引出堆栈的概念呢?
- 具体操作有哪些?

若进栈序列为 1,2,3,4 ， 进栈过程中可以出栈，则下列不可能的一个出栈序列是（ ）

A 1,4,3,2

B 2,3,4,1

C 3,1,4,2

D 3,4,2,1

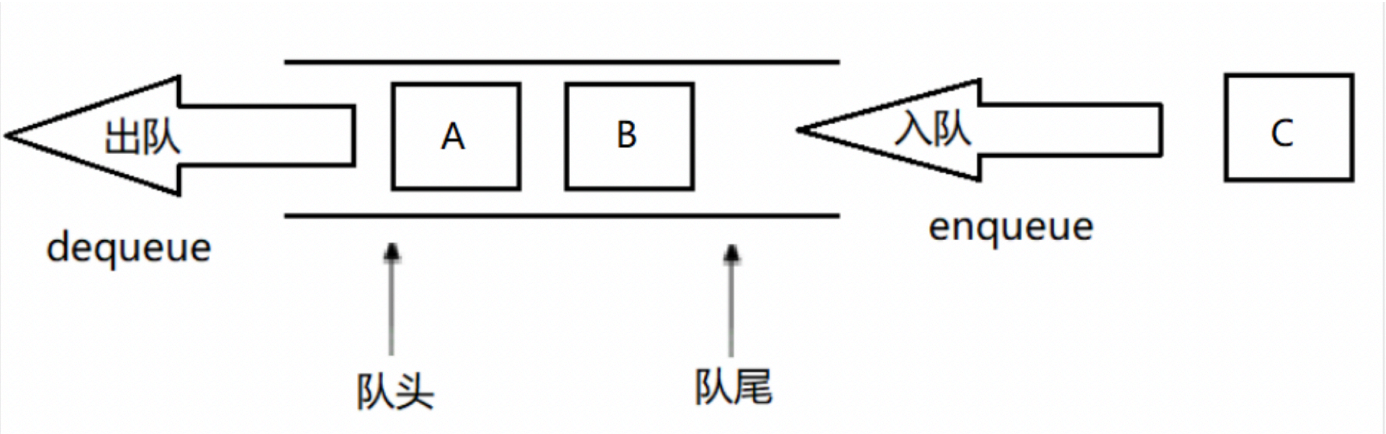
一.队列的概念和队列的2种结构

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。

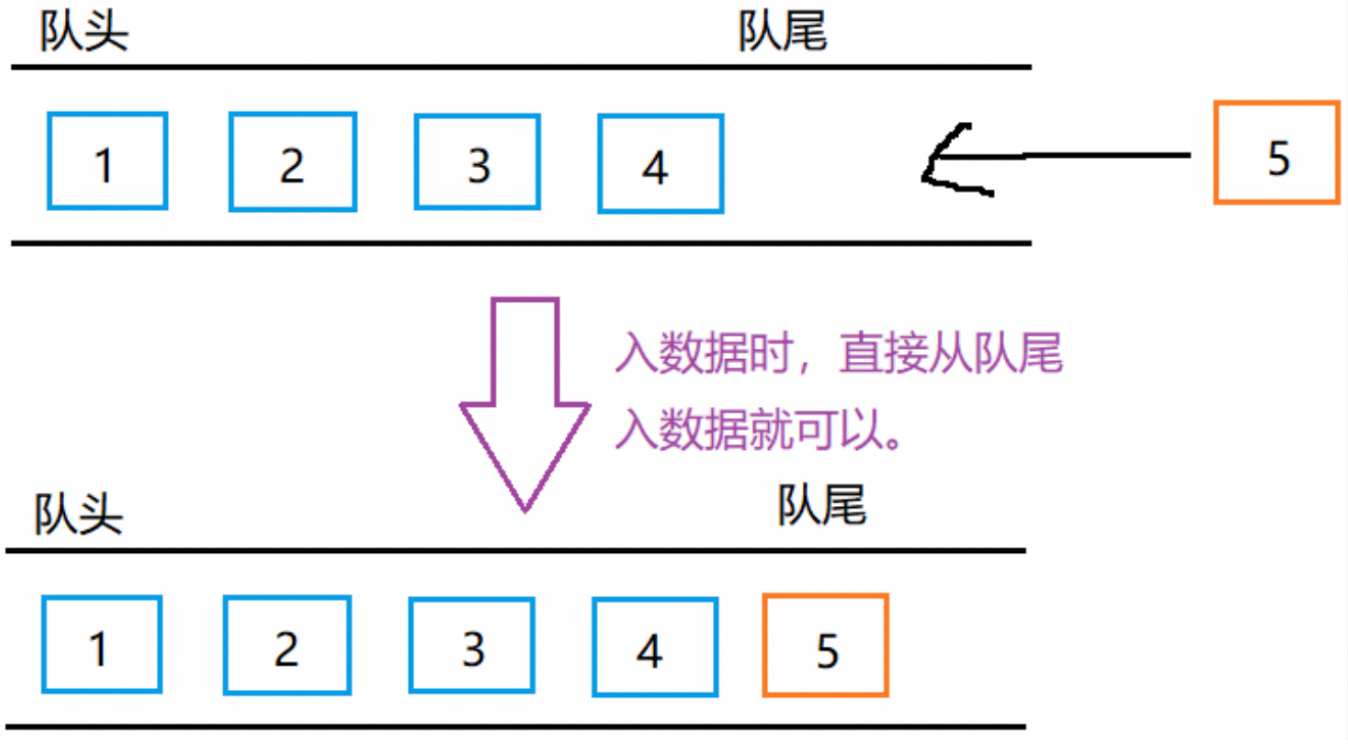
基本结构：

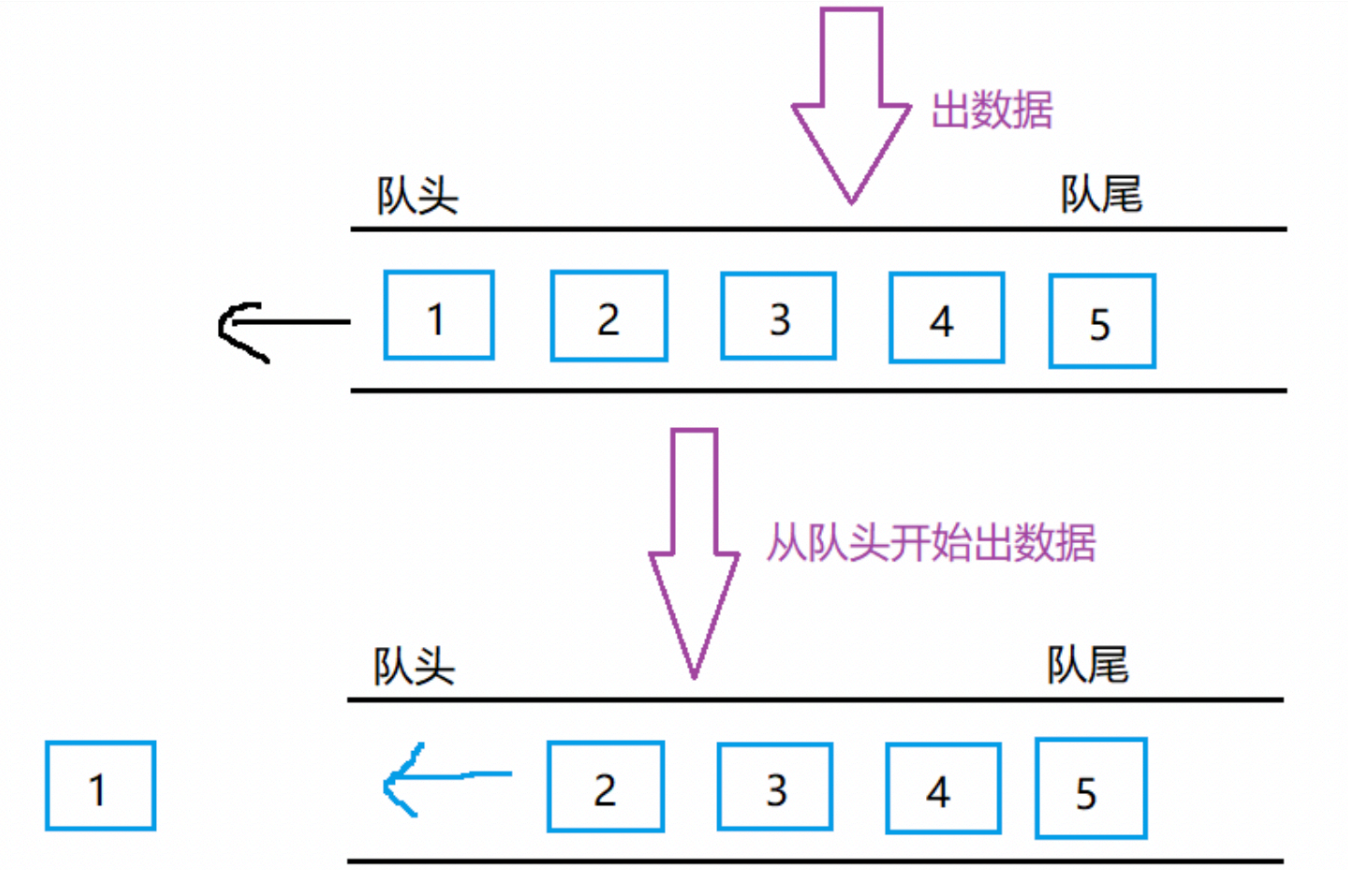


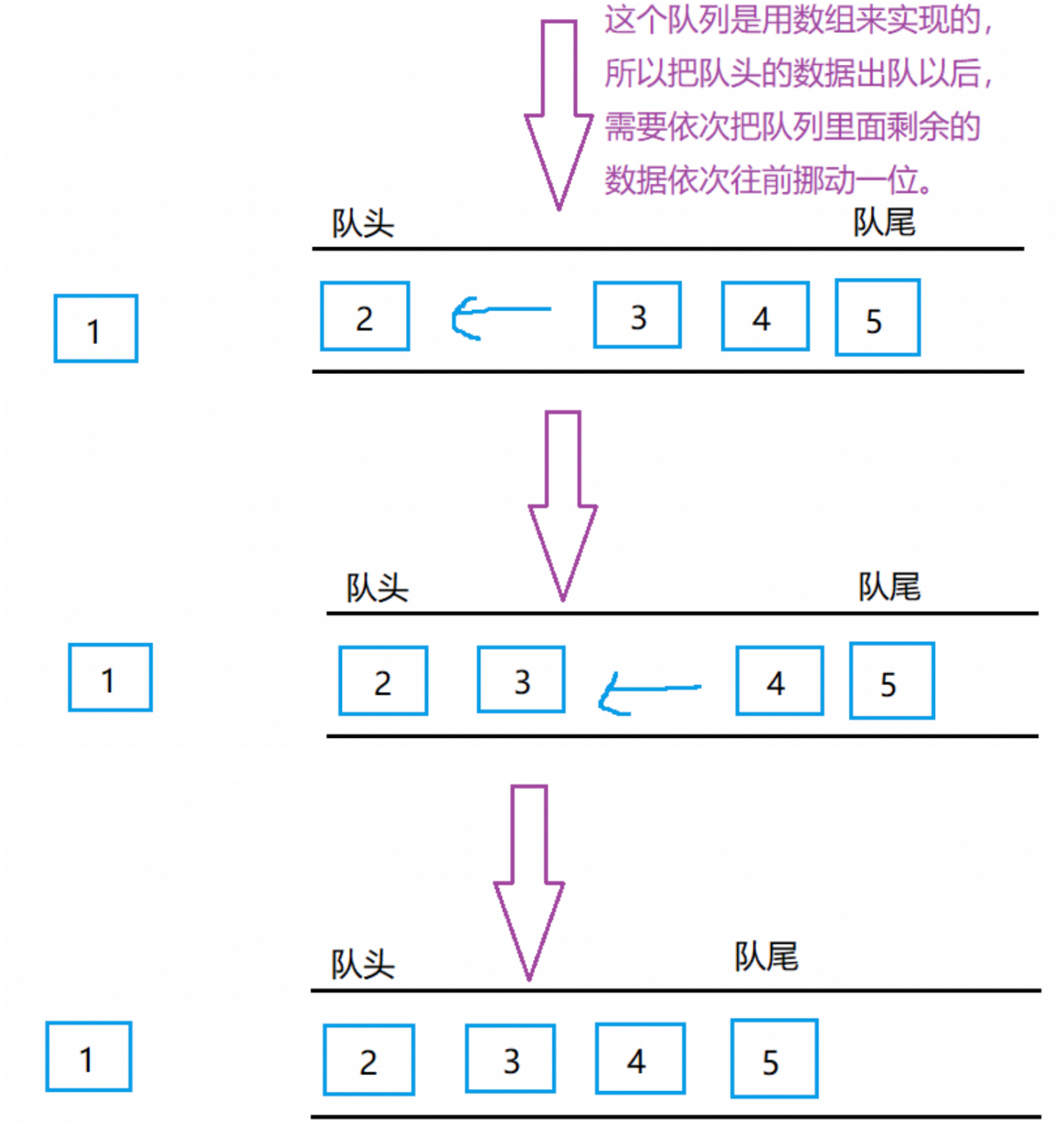
队列的实现可以用链表也可以用数组。（用链表更优，在使用数组来从数组头部出数据时，需要挪动大量的数据，效率较低；如果使用链表来实现，就可以让链表的头做队头，链表的尾做队尾，数据的插入和删除操作都比较方便）。

用数组实现队列：

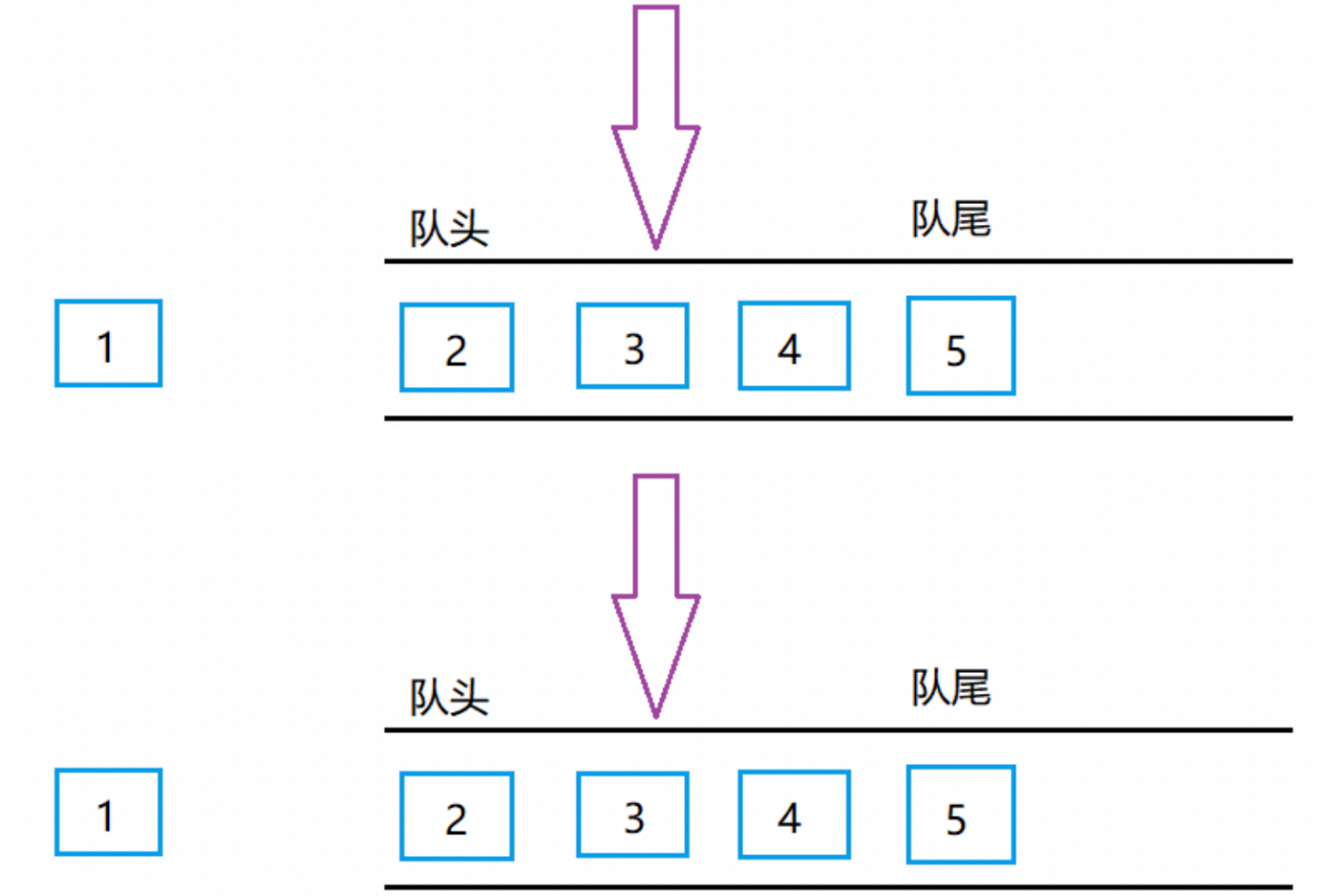
队列是先进先出的。	队尾入数据，队头出数据






这个队列是用数组来实现的，
所以把队头的数据出队以后，
需要依次把队列里面剩余的
数据依次往前挪动一位。
队头
队尾
1
2
3
4
5
队头
队尾
1
2
3
4
5
队头
队尾
1
2
3
4
5

用链表实现队列：



二.队列的各个函数接口及实现

1.队列的结构体定义（链式结构）

定义链式结构的队列，因为队列是先进先出的，(要实现它的功能需要头删和尾插）所以我们在这里分别定义了**链表的头指针(head)和尾指针(tail)**。

```
typedef int QDataType;
typedef struct QueueNode
{
```

```
  struct QueueNode* next;
  QDataType data;
}QueueNode;

typedef struct Queue
{
  QueueNode* head;
  QueueNode* tail;
}Queue;



typedef struct QueueNode
{
  struct QueueNode* next;
  QDataType data;

  QueueNode* head;
  QueueNode* tail;
}QueueNode;
```

2.队列的初始化

刚开始进行初始化的时候，队列里面还没有数据，所以应该先把头指针和尾指针先置空(NULL)。

```c
void QueueInit(Queue* pq)//初始化
{
  assert(pq);
  pq->head = pq->tail = NULL;
}
```

3.队列的销毁

我们实现的队列是用链表来实现的，所以要销毁队列就是要**销毁链表**，(所以和链表的销毁是一样的)，**要遍历队列，然后逐一的释放每个结点的空间**，最后再把**头指针和尾指针置空就可以了**。

```c
void QueueDestroy(Queue* pq)//销毁
{
  assert(pq);
  QueueNode* cur = pq->head;
  while (cur)
  {
    QueueNode* next = cur->next;
    free(cur);
    cur = next;
  }
  pq->head = pq->tail = NULL;
}
```

4.队尾数据入队列（尾插）

- 入队 和链表的尾插是一样的，
- 首先**申请一块新的空间**把要插入的数据放到新开辟的这块空间中
- 然后判断队列是否为空(head是否等于NULL),如果队列为空就是链表的头插，那么头指针和尾指针都要指向这个新的结点；
- 如果队列不为空就是链表的尾插，那么只需要把新节点链接到尾指针的后面，然后更新尾指针(让尾指针指向新的尾 --- 也就是所插入的新节点)。

```
void QueuePush(Queue* pq, QDataType x)//添加   尾插
{
  assert(pq);
  QueueNode* newnode = (QueueNode*)malloc(sizeof(QueueNode));
  if (newnode == NULL)
  {
    printf("malloc fail\n");
    exit(-1);
  }
  newnode->next = NULL;
```

```
    newnode->data = x;
    if (pq->head == NULL)
    {
        pq->head = pq->tail = newnode;
    }
    else
    {
        pq->tail->next = newnode;
        pq->tail = newnode;
    }
}
```

5.队头数据出队列（头删）

出队 和链表的头删是一样的，要判断一下队列是否为空，为空的话就不能删了。

要删除头结点，所以要先保存一下头结点的下一个结点的位置(next=head->next;)，然后把头结点释放掉（free),让头指针**指向新的头(next所指向的位置) (head=next)**,

还有一种情况需要进行特殊的考虑：

就是当删除头结点以后，队列如果为空了的话(head=NULL),这个时候要记得把尾指针也置空(tail=NULL)。

```
void QueuePop(Queue* pq)//删除  头删
{
  assert(pq);
  assert(!(QueueEmpty(pq)));
  QueueNode* next = pq->head->next;
  free(pq->head);
  pq->head = next;
  if (pq->head == NULL)
    // 这个情况是需要特殊来判断一下的，就是删除以后链表为空了，
    // 这时要记得把尾指针(tail)也置空一下
  {
    pq->tail = NULL;
  }
}
```

6.取队头的数据

取数据，需要首先判断一下队列**是否已经为空了**，如果已经为空了，队列里面就没有数据了，也就不能取数据了。

队头的数据：head->data

```
QDataType QueueFront(Queue* pq)//取队头的数据
{
  assert(pq);
  assert(!QueueEmpty(pq));
  return pq->head->data;
}
```

7.取队尾的数据

取数据，需要首先判断一下**队列是否已经为空了**，如果已经为空了，队列里面就没有数据了，也就不能取数据了。

队尾的数据：tail->data

```
QDataType QueueBack(Queue* pq)//取队尾的数据
{
  assert(pq);
  assert(!QueueEmpty(pq));
  return pq->tail->data;
}
```

8.计算队列的数据个数(队列的大小)

计算队列的数据个数：直接遍历链式队列的每一个结点，**然后返回size就可以了。**

```
int QueueSize(Queue* pq)//队列的大小
{
  QueueNode* cur = pq->head;
  int size = 0;
  while (cur)
  {
    cur = cur->next;
    size++;
  }
  return size;
}
```

9.判断队列是否为空
如果 队列为空 == 头结点为NULL

头结点为空(NULL)，则队列为空； 头结点不为空，则队列不为空。

```
bool QueueEmpty(Queue* pq)//判断队列是否为空
{
  assert(pq);
  return pq->head == NULL;
}
```

问题：