

# 《数据结构》上机报告

2018 年 10 月 17 日

姓名：刘思源 学号：1651390 班级：电子三班 得分：\_\_\_\_\_

实验题目	队列	
问题描述	<p>队列是限定在表的一端插入，另一端删除的线性表。队列的特点是先进先出（FIFO）。循环队列是队列的顺序存储结构。本题练习循环队列的基本操作，包括入队、出队、判队空、判队满、队列的遍历。队列的链式存储结构需要两个分别指示队头和队尾的指针，保证入队和出队的操作时间复杂度都是 <math>O(1)</math></p>	
基本要求	<p>1. (p1) 循环队列是队列的顺序存储结构。本题练习循环队列的基本操作，包括入队、出队、判队空、判队满、队列的遍历。</p> <p>2. (p2) 本题练习链队列的基本操作，包括入队、出队、判队空、判队满、队列的遍历。</p>	
	已完成基本内容（序号）：	1, 2
选做要求		
	已完成选做内容（序号）	
数据结构设计	<p>循环队列：</p> <pre>typedef struct QNode{     ElemType *base;     int front;     int rear;     int QueueSize; } SqQueue;</pre> <p>链队列：</p> <pre>typedef struct QNode {     int data;     QNode *next; } QNode, *QueuePtr;</pre> <pre>struct LinkQueue {     QueuePtr front;     QueuePtr rear;</pre>	

	<pre>};</pre>
功能(函数)说明	<p>1、 循环队列的初始化</p> <p>若一开始没有指定队列的大小，则分配 MAXSIZE 的空间。将队列的头指针和尾指针同时指向初始分配的起始空间，将栈的大小定义为初始分配的 MAXSIZE 大小。若规定了要分配的空间大小，则使用 n 替换 MAXSIZE。值得一提的是，再分配空间时，本文多分配了一个空间。因为在后续的操作中，判满操作要求不能将头指针等于尾指针作为条件，否则在一开始就会判断为满；同时，再后续的判满时，因为以尾指针的下一位置与头指针作比较，如果分配恰好为 n 的空间，就会有一个位置永远无法赋值。所以多于分配一个空间出来。</p> <pre>Status InitQueue(SqQueue &amp;Q) {     Q.base = (ElemType *)malloc(MAXSIZE * sizeof(ElemType));     if (!Q.base)         exit(OVERFLOWED);     Q.front = Q.rear = 0;     Q.QueueSize = MAXSIZE;     return OK; }  Status InitQueue(SqQueue &amp;Q, int n) {     Q.base = (ElemType *)malloc((n+1) * sizeof(ElemType));     if (!Q.base)         exit(OVERFLOWED);     Q.front = Q.rear = 0;     Q.QueueSize = n+1;     return OK; }</pre> <p>2、 入队操作</p> <p>如果尾指针+1，也就是下一位置和头指针的位置相同的话，则队满，返回 ERROR。如果不是，就将该位置的数据更换为新的数据，再将尾指针向后移动。此处需要注意的是，由于整体的队列是一个环，所以前移后移的操作要在</p>

[0, MAXSIZE]的区间内进行。要进行模运算。

```
Status EnQueue(SqQueue &Q, ElemType elem)
{
    if ((Q.rear+1) % Q.QueueSize == (Q.front)) {
        return ERROR;
    }
    Q.base[Q.rear] = elem;
    Q.rear = (Q.rear + 1) % Q.QueueSize;
    return OK;
}
```

### 3、出队操作

如果尾指针等于头指针，则证明队是空队，返回 ERROR。否则，输出当前头指针指向的内容，将头指针后移。由于整体的队列是一个环，所以前移后移的操作要在[0, MAXSIZE]的区间内进行。也要进行模运算进行移动。

```
Status DeQueue(SqQueue &Q, ElemType &e)
{
    if (Q.front == Q.rear)
        return ERROR;
    e = Q.base[Q.front];
    Q.front = (Q.front + 1) % Q.QueueSize;
    return OK;
}
```

### 4、循环对列的读取

首先判断当前，尾指针是否小于头指针。如果小于，则证明尾指针已经走过至少一圈，那么将从当前的头指针开始，遍历到下一圈中的尾指针，即尾指针要加上队列的长度；如果头指针小于尾指针，则直接从头指针遍历到尾指针即可。

```
Status PrintQueue(SqQueue &Q)
{
    int i;
    if (Q.front < Q.rear) {
        for (i = Q.front; i < Q.rear; ++i) {
            cout << Q.base[i] << " ";
        }
    }
    else {
        for (i = Q.front; i < Q.rear+Q.QueueSize; ++i) {
            cout << Q.base[i%Q.QueueSize] << " ";
        }
    }
    cout << endl;
    return OK;
}
```

### 5、链队列的初始化

本文给出两种初始化方法：如果没有给出需要的大小，要在后续的操作中动态分配空间，就只初始化出头结点，将头指针和尾指针都指向头结点。如果

给定了大小 n，不应只分配 n 大小的空间，因为队列还要不断地出队入队，空间会不够。本文直接分配了 MAXSIZE 大小的空间。也可以在后续的入队中动态分配。

```
Status InitQueue(LinkQueue &Q) {
    Q.front = Q.rear = new QNode;
    if (!Q.front) {
        return ERROR;
    }
    Q.front->next = NULL;
    return OK;
}

Status InitQueue(LinkQueue &Q, int n) {
    n = MAXSIZE;
    Q.front = Q.rear = new QNode;
    if (!Q.front) {
        return ERROR;
    }
    Q.front->next = NULL;
    while (n>=1) {
        Q.rear->next = new QNode;
        Q.rear = Q.rear->next;
        n--;
    }
    Q.rear = Q.front;
    return OK;
}
```

## 6、链队列的销毁

当链队列的头指针不为空时，将尾指针指向头指针的下一位置。再释放头指针，释放完毕后将头指针指向尾指针，直到头指针为空。

```
void DestroyQueue(LinkQueue &Q) {
    while (Q.front) {
        Q.rear = Q.front->next;
        delete (Q.front);
        Q.front = Q.rear;
    }
}
```

## 7、链队列的判空

若链队列的头指针等于尾指针，则链队列为空。

```
Status EmptyQueue(LinkQueue Q)
{
    if (Q.front == Q.rear) {
        return TRUE;
    }
}
```

```
}  
return FALSE;  
}
```

#### 8、求链队列的长度

定义一个指针指向头指针的位置，遍历整个链队列直到尾指针，使用一个计数器进行计数，返回计数器的值。

```
Status GetQueueLength(LinkQueue Q)
```

```
{  
    int length=0;  
    QueuePtr p = Q.front;  
    while (p != Q.rear) {  
        p = p->next;  
        length++;  
    }  
    return length;  
}
```

#### 9、链队列的入队

如果当前链队列的长度大于要求的长度，则队列满，返回 ERROR。否则，将当前的尾指针位置赋予入队的值，将尾指针移至下一个节点。

```
Status EnQueue(LinkQueue &Q, int value, int n)
```

```
{  
    int t = GetQueueLength(Q);  
    if (n<=t) {  
        return ERROR;  
    }  
    Q.rear->data = value;  
    if (Q.rear->next) {  
        Q.rear = Q.rear->next;  
    }  
    else {  
        Q.rear = NULL;  
    }  
    return OK;  
}
```

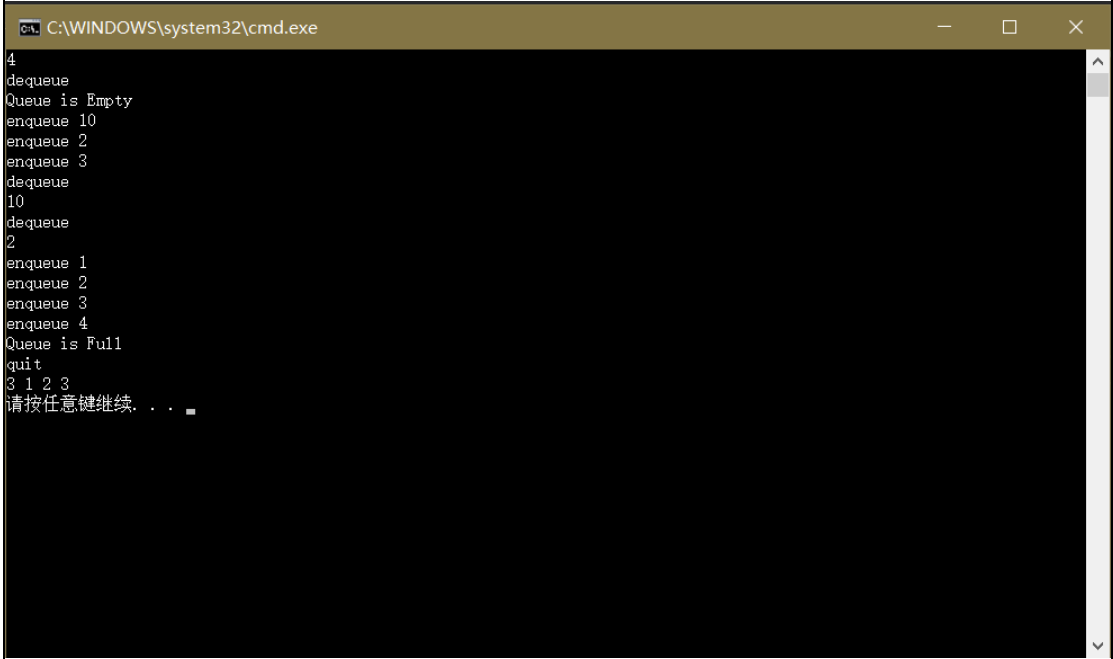
#### 10、链队列的出队

如果尾指针等于头指针，则队列为空。否则将当前头指针指向的值输出，同时头指针向后移动。

```
Status DeQueue(LinkQueue &Q, int &value)
```

```
{  
    if (Q.front == Q.rear) {  
        return ERROR;  
    }  
    value = Q.front->data;  
    Q.front = Q.front->next;
```

	<pre>        return OK;     }  11、    链队列的读取         如果尾指针等于头指针，则队列为空。否则，定义一个指针指向头指针，将         当前头指针指向的值输出，同时头指针向后移动。直到移动到尾指针处停止。     Status PrintQueue(LinkQueue Q)     {         if (Q.front == Q.rear) {             return ERROR;         }         QueuePtr p = Q.front;         while (p != Q.rear) {             cout &lt;&lt; p-&gt;data &lt;&lt; " ";             p = p-&gt;next;         }         cout &lt;&lt; endl;         return OK;     }</pre>
开发环境	Visual studio 2017
调试分析	

	 <pre> C:\WINDOWS\system32\cmd.exe 4 dequeue Queue is Empty enqueue 10 enqueue 2 enqueue 3 dequeue 10 dequeue 2 enqueue 1 enqueue 2 enqueue 3 enqueue 4 Queue is Full quit 3 1 2 3 请按任意键继续. . . </pre>
心得体会	<ol style="list-style-type: none"> <li>1、 循环列表注意初始分配空间 初始化循环列表时，本文多分配了一个空间。因为在后续的操作中，判满操作要求不能将头指针等于尾指针作为条件，否则在一开始就会判断为满；同时，再后续的判满时，因为以尾指针的下一位置与头指针作比较，如果分配恰好为 <math>n</math> 的空间，就会有一个位置永远无法赋值。所以多于分配一个空间出来。</li> <li>2、 循环列表所有的移动都是模运算 因为循环列表本身在逻辑上是一个环，所以所有的移动操作都涉及到模运算，并且注意判断头指针尾指针是在同一圈中还是不同圈。</li> <li>3、 链队列的初始化 如果给定了大小 <math>n</math>，不应只分配 <math>n</math> 大小的空间，因为队列还要不断地出队入队，空间会不够。本文直接分配了 <math>\text{MAXSIZE}</math> 大小的空间。也可以在后续的入队中动态分配。 当然，最合适的方法应该是先分配 <math>n</math> 大小的空间，然后再入队操作中动态分配空间。判满时根据队列的长度函数进行判满。这样做更能减少空间复杂度。</li> </ol>