# 数据结构实验报告——实验六

## 学号： 20201060330 姓名： 胡诚皓 得分：\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### 一、实验目的

1. 复习队列的逻辑结构、存储结构及基本操作；
2. 掌握链队列、循环队列。

### 二、实验内容

1. （必做题）链队列的基本实现

假设队列中数据元素类型是字符型，请采用链队列实现队列的以下基本操作：

1. Status InitQueue(&Q)

//构造空队列Q；

1. Status EnQueue(&Q, e)

//元素e入队列Q；

1. Status DeQueue (&Q, &e)

//队列Q出队列，元素为e。

1. （必做题）循环队列的基本实现

假设队列中数据元素类型是字符型，请采用循环队列实现队列的以下基本操作：

1. Status InitQueue(&Q)

//构造空队列Q；

1. Status EnQueue(&Q, e)

//元素e入队列Q；

1. Status DeQueue (&Q, &e)

//队列Q出队列，元素为e。

### 三、数据结构及算法描述

1. （必做题）链队列的基本实现
2. 数据结构

使用定义的QNode作为链队列中的结点元素，其中的data作为数据域、next作为指向下一个结点的指针域，同时定义了指向QNode的指针类型QueuePtr。

LinkQueue作为链队列类型，队列需要从队尾入队、从队首出队，因此需要同时记录链表中头结点和尾结点的位置。为了操作方便，用头插法处理链表，即把链表头作为队尾、将链表尾作为队头。front为指向队头（链表尾）的指针，rear为指向队尾（链表头）的指针。

在链队列中，rear始终指向下一个可以直接放入元素的位置，也就是说，链表的结点个数始终会比实际队列中的元素个数多一。

设计了Status empty(LinkQueue Q)辅助函数，用于判断队列是否为空。若队列为空，返回OK；若队列不为空，返回ERROR。

1. 算法描述

Status InitQueue(LinkQueue \*Q)

* 1. 申请1个QNode结点tmp作为链表中的首个结点，若申请失败，直接返回ERROR，同时为tmp数据域赋空字符（便于调试时的观察），指针域赋NULL；
  2. 将链表队列Q的队首队尾都指向tmp。

Status EnQueue(LinkQueue \*Q, QElemType e)

* 1. 申请1个QNode结点tmp作为链表中的首个结点，若申请失败，直接返回ERROR，同时为tmp数据域赋空字符（便于调试时的观察），指针域赋NULL；
  2. 将指向要插入元素的位置上的QNode的数据域赋值为e，并使其指针域指向下一个要插入元素的位置tmp；
  3. 将tmp作为队尾，即下一个要插入元素的位置。

Status DeQueue(LinkQueue \*Q, QElemType \*e)

* 1. 判断队列是否为空，若队列为空，无法弹出，直接返回ERROR；
  2. 用tmp临时存储要弹出的结点；
  3. 将S->top前移一个位置；
  4. 将弹出的元素的数据域赋值到e所指的位置；
  5. 释放弹出的结点。

1. （必做题）循环队列的基本实现
   1. 数据结构

循环队列以线性表的形式来存储数据，SqQueue作为循环队列的实际结构体，其中的指向QElemType的指针base作为指向线性表基址的指针；front存储队首位置对基址的偏移量，rear存储队尾位置对基址的偏移量。与上一题同样的，rear始终表示下一个要添加的新元素的位置。

在本题中规定了队列最大元素个数MAXQSIZE为100。设计了Status empty(SqQueue Q)、Status full(SqQueue Q)辅助函数分别用于判断队列是否为空、是否为满，为空/为满返回OK，不为空/不为满返回ERROR。

* 1. 算法描述

1. Status InitQueue(SqQueue \*Q)
   1. 对循环队列Q进行初始化，申请MAXQSIZE+1（由于要最多存储100个数据，使用一个元素的空间以区别队空/队满）大小的一段空间，并把首地址存于Q->base中，若申请失败，直接返回ERROR；
   2. 将循环队列的队首和队尾的位置都设为0

Status EnQueue(SqQueue \*Q, QElemType e)

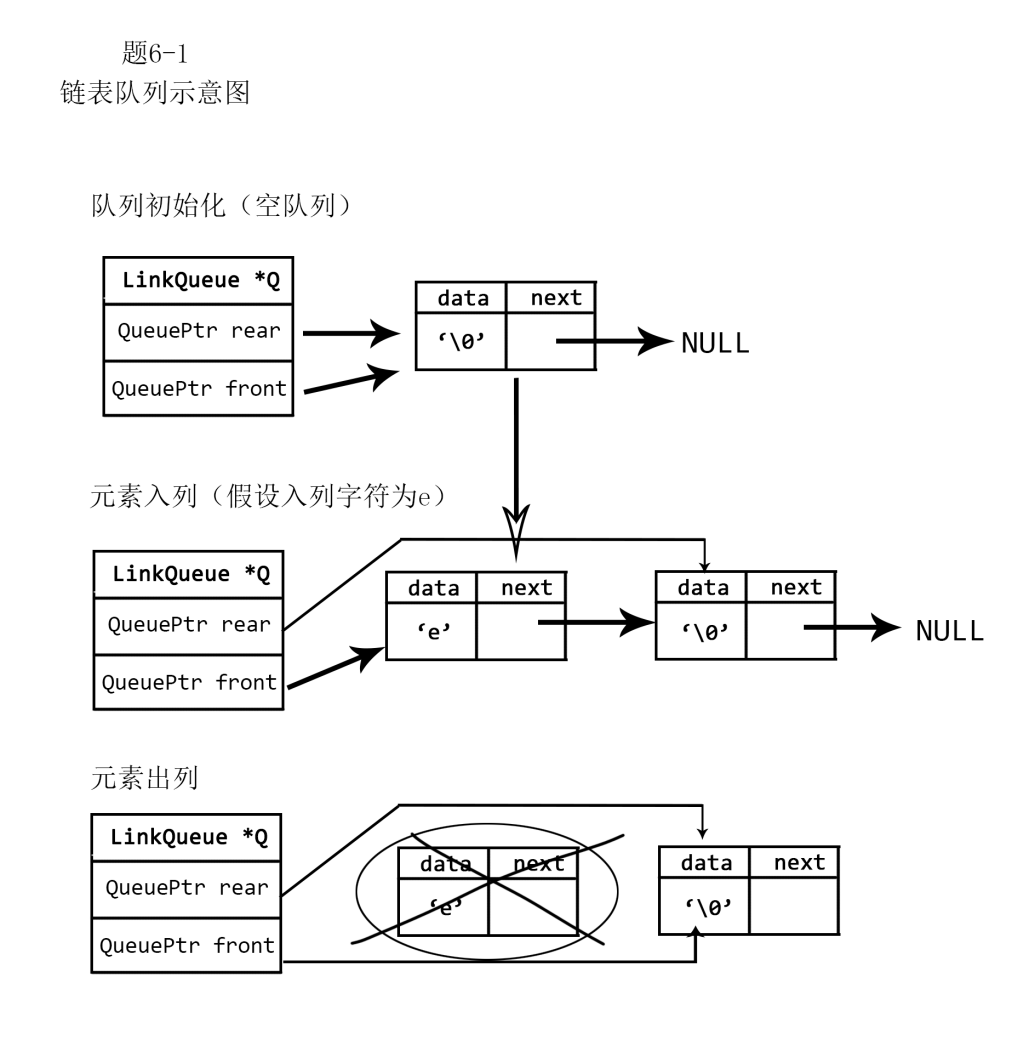
* 1. 若循环队列Q已满，直接返回ERROR；
  2. 从队尾将新数据e入队；
  3. 将队尾位置+1并对总空间数MAXQSIZE+1取余，以达到循环效果。

Status DeQueue(SqQueue \*Q, QElemType \*e)

* 1. 若循环队列Q为空，直接返回ERROR；
  2. 从队头出列一个元素，将数据存储到e所指的地址；
  3. 将队头位置+1并对总空间数MAXQSIZE+1取余，以达到循环效果。

### 详细设计

1. （必做题）链队列的基本实现



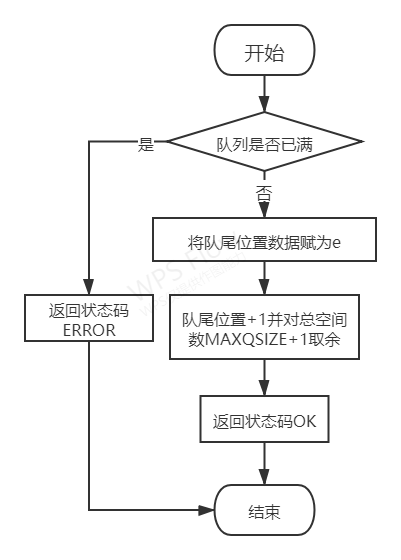
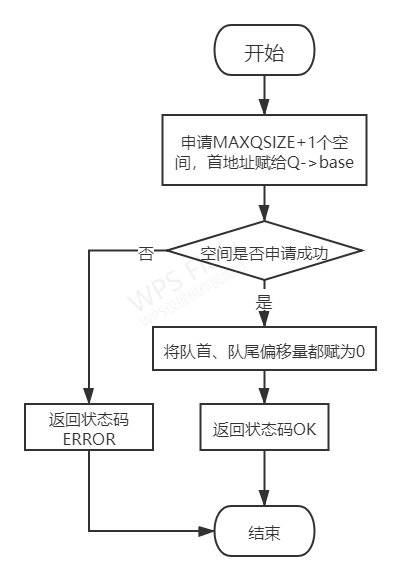
题5-1

Pop函数示意图

题5-1

Push函数示意图

1. （必做题）循环队列的基本实现

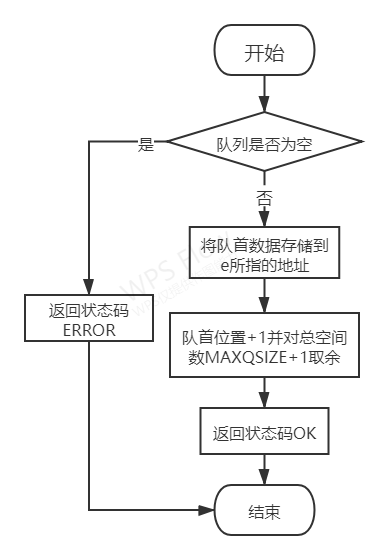


题5-1

EnQueue函数

题5-1

InitQueue函数



题5-1

DeQueue函数

### 五、程序代码

1. （必做题）链队列的基本实现



#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#define QElemType char

#define Status int

#define OK 1

#define ERROR 0

typedef struct QNode {

QElemType data;

struct QNode \*next;

} QNode, \*QueuePtr;

typedef struct {

QueuePtr rear;

QueuePtr front;

} LinkQueue;

Status InitQueue(LinkQueue \*);

Status EnQueue(LinkQueue \*, QElemType);

Status DeQueue(LinkQueue \*, QElemType \*);

Status empty(LinkQueue);

int main() {

LinkQueue queue;

InitQueue(&queue);

char ch;

printf("input some chars separate with space or enter(ending with #)\n");

//以“#”作为输入的结尾，以此判断输入结束

while (1) {

scanf("%c", &ch);

if (ch == '#')

break;

if (isspace(ch))

continue;

EnQueue(&queue, ch);

}

printf("DeQueue:\n");

while (DeQueue(&queue, &ch) == OK)

printf("%c ", ch);

printf("\n");

system("pause");

return 0;

}

Status InitQueue(LinkQueue \*Q) {

QNode \*tmp = (QNode \*) malloc(sizeof(QNode));

if (tmp == NULL)

return ERROR;

tmp->data = '\0';

tmp->next = NULL;

Q->rear = tmp;

Q->front = tmp;

return OK;

}

Status EnQueue(LinkQueue \*Q, QElemType e) {

QNode \*tmp = (QNode \*) malloc(sizeof(QNode));

if (tmp == NULL)

return ERROR;

tmp->data = '\0';

tmp->next = NULL;

Q->rear->data = e;

Q->rear->next = tmp;

Q->rear = tmp;

return OK;

}

Status DeQueue(LinkQueue \*Q, QElemType \*e) {

QNode \*tmp=Q->front;

if (empty(\*Q) == OK)

return ERROR;

Q->front = Q->front->next;

\*e = tmp->data;

free(tmp);

return OK;

}

Status empty(LinkQueue Q) {

return Q.rear == Q.front ? OK : ERROR;

}

1. （必做题）循环队列的基本实现



#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#define QElemType char

#define Status int

#define OK 1

#define ERROR 0

#define MAXQSIZE 100

typedef struct {

QElemType \*base;

int front;

int rear;

} SqQueue;

Status InitQueue(SqQueue \*);

Status EnQueue(SqQueue \*, QElemType);

Status DeQueue(SqQueue \*, QElemType \*);

Status empty(SqQueue);

Status full(SqQueue);

int main() {

SqQueue queue;

InitQueue(&queue);

char ch;

printf("input some chars separate with space or enter(ending with #)\n");

printf("input character should be fewer than 100\n");

//以“#”作为输入的结尾，以此判断输入结束

while (1) {

scanf("%c", &ch);

if (ch == '#')

break;

if (isspace(ch))

continue;

if (EnQueue(&queue, ch) == ERROR){

printf("Too many chars!\n");

break;

}

}

printf("DeQueue:\n");

while (DeQueue(&queue, &ch) == OK)

printf("%c ", ch);

printf("\n");

system("pause");

return 0;

}

Status InitQueue(SqQueue \*Q) {

Q->base = (QElemType \*) malloc(sizeof(QElemType) \* (MAXQSIZE+1));

if (Q->base == NULL)

return ERROR;

Q->front = Q->rear = 0;

return OK;

}

Status EnQueue(SqQueue \*Q, QElemType e) {

if (full(\*Q))

return ERROR;

(Q->base)[Q->rear] = e;

Q->rear = (Q->rear + 1) % (MAXQSIZE+1);

return OK;

}

Status DeQueue(SqQueue \*Q, QElemType \*e) {

if(empty(\*Q))

return ERROR;

\*e = (Q->base)[Q->front];

Q->front = (Q->front + 1) % (MAXQSIZE+1);

return OK;

}

Status empty(SqQueue Q) {

return Q.front == Q.rear ? OK : ERROR;

}

Status full(SqQueue Q) {

return (Q.rear + 1) % (MAXQSIZE+1) == Q.front ? OK : ERROR;

}

### 六、测试和结果

1. （必做题）链队列的基本实现

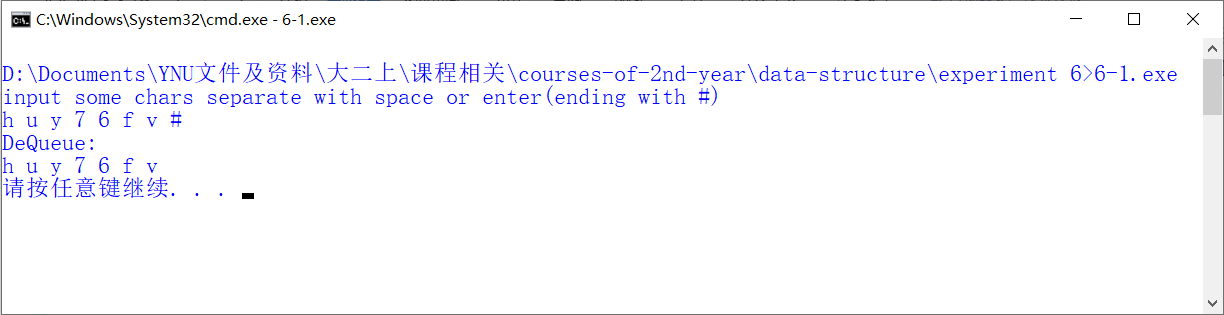
**Input:**

h u y 7 6 f v #

**Output:**

DeQueue:

h u y 7 6 f v



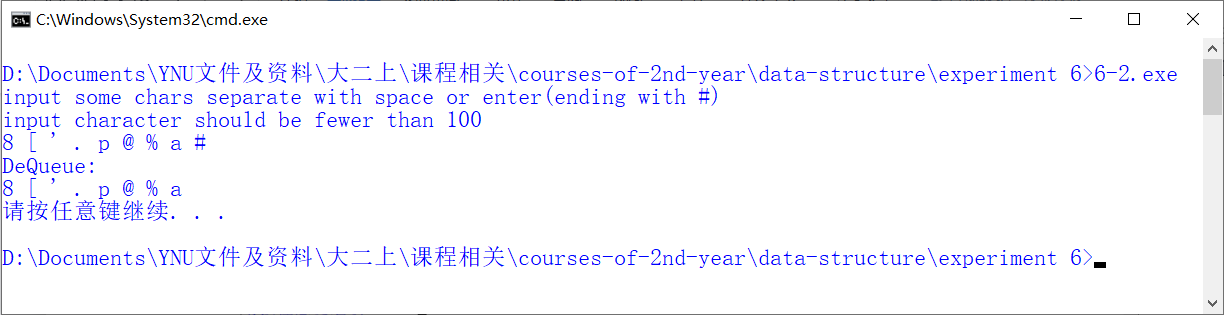
1. （必做题）循环队列的基本实现

**Input:**

8 [ ' . p @ % a #

**Output:**

8 [ ' . p @ % a



### 用户手册

* 1. （必做题）链队列的基本实现

使用“#”作为结束符，因此不能把“#”作为要入列的字符。

* 1. （必做题）循环队列的基本实现

使用“#”作为结束符，因此不能把“#”作为要入列的字符。同时规定队列最大容量为100，不能入列超过100个元素。