

实验报告 (week-04)
2020111235 马靖淳

一. 实验任务

1. 链队列
2. 循环队列

二. 实验上机时间 4h

三. 知识点

1. 链队列
 - (1) 初始化队列
 - (2) 销毁队列
 - (3) 判队列是否空
 - (4) 求队列长度
 - (5) 取队头元素
 - (6) 清空队列
 - (7) 入队列
 - (8) 出队列
 - (9) 遍历队列
2. 循环队列

四. 源代码及结果截屏

1. 链队列

(1) LinkQueue.h

```
#pragma once
#ifndef LINKQUEUE_H
#define LINKQUEUE_H

#include <stdlib.h>
#include "Status.h"

typedef int Status;
typedef int QElemType;
/*链队列结点实现*/
typedef struct QNode
{
    QElemType data;
    struct QNode* next;
}QNode, *QueuePtr;
/*链队列数据类型实现*/
typedef struct//链队列类型
{
    QueuePtr front;//队头指针
    QueuePtr rear;//队尾指针
}LinkQueue;
```

```

Status InitQueue(LinkQueue* Q); //初始化队列
Status DestroyQueue(LinkQueue* Q); //销毁队列
Status QueueEmpty(LinkQueue Q); //判队列是否空
int QueueLength(LinkQueue Q); //求队列长度
Status GetHead(LinkQueue Q, QElemType* e); //取队头元素
Status ClearQueue(LinkQueue* Q); //清空队列
Status EnQueue(LinkQueue* Q, QElemType e); //入队列
Status DeQueue(LinkQueue* Q, QElemType* e); //出队列
Status visit(QElemType e);
Status QueueTraverse(LinkQueue* Q, Status(*visit)(QElemType*e)); //遍历队列

#endif

```

(2) LinkQueue.c

```

#pragma once
#pragma warning(disable:4996)
#include "LinkQueue.h"
#include <stdio.h>

Status InitQueue(LinkQueue* Q)
{
    //构造一个带头节点的空队列
    (*Q).front = (*Q).rear = (QueuePtr)malloc(sizeof(QNode));
    if (!(*Q).front)
        exit(OVERFLOW); //存储分配失败
    (*Q).front->next = NULL;
    return OK;
}

Status DestroyQueue(LinkQueue* Q)
{
    while ((*Q).front)
    {
        (*Q).rear = (*Q).front->next;
        free((*Q).front);
        (*Q).front = (*Q).rear;
    }
    (*Q).front = NULL;
    (*Q).rear = NULL;
    return OK;
}

Status QueueEmpty(LinkQueue Q)

```

```

{
    // 若Q为空队列, 则返回TRUE, 否则返回FALSE
    if (Q.front == Q.rear)
        return TRUE;
    else
        return FALSE;
}

int QueueLength(LinkQueue Q)
{
    //求队列的长度
    int i = 0;
    QueuePtr p;
    p = Q.front;
    while (Q.rear != p)
    {
        i++;
        p = p->next;
    }
    return i;
}

Status GetHead(LinkQueue Q, QElemType* e)
{
    //若队列不空, 则用e返回Q的队头元素, 并返回OK, 否则返回ERROR
    QueuePtr p;
    if (Q.front == Q.rear)
        return ERROR;
    p = Q.front->next;
    *e = p->data;
    return OK;
}

Status ClearQueue(LinkQueue* Q)
{
    // 将Q清为空队列
    QueuePtr p, q;
    (*Q).rear = (*Q).front;
    p = (*Q).front->next;
    (*Q).front->next = NULL;
    while (p)
    {
        q = p;
        p = p->next;
    }
}

```

```

        free(q);
    }
    return OK;
}

Status EnQueue(LinkQueue* Q, QElemType e)
{
    //插入元素e为Q的新的队尾元素
    QueuePtr p = 0;
    p = (QueuePtr)malloc(sizeof(QNode));
    if (!p)
        exit(OVERFLOW); //存储分配失败
    p->data = e;
    p->next = NULL;
    (*Q).rear->next = p;
    (*Q).rear = p;
    return OK;
}

Status DeQueue(LinkQueue* Q, QElemType *e)
{
    //若队列不空，则删除Q的队头元素，用e返回其值，并返回OK；否则返回ERROR
    if ((*Q).front == (*Q).rear)
        return ERROR;
    QueuePtr p = 0;
    p = (*Q).front->next;
    *e = p->data;
    (*Q).front->next = p->next;
    if ((*Q).rear == p)
        (*Q).rear = (*Q).front;
    free(p);
    return OK;
}

Status visit(QElemType e)
{
    printf("%d ", e);
    return OK;
}

Status QueueTraverse(LinkQueue *Q, Status(*visit)(QElemType*))
{
    //从对头到队尾依次对队列Q中的每个元素调用函数visit()，返回OK，一旦visit失败则操作失败

```

```

//是空队列则返回ERROR
if ((*Q).front == (*Q).rear)
    return ERROR;
QueuePtr p = 0;
p = (*Q).front->next;
while (p)
{
    visit(p->data);
    p = p->next;
}
printf("\n");
return OK;
}

```

(3) test.c

```

#include <stdio.h>
#include "LinkQueue.h"

int main()
{
    LinkQueue Q;
    //InitQueue test
    if (InitQueue(&Q))
        printf("InitQueue success!\n");
    else
        printf("InitQueue unsucess!\n");

    //DestroyQueue test
    if (DestroyQueue(&Q))
        printf("DestroyQueue sucess!\n");
    else
        printf("DestroyQueue unsucess!\n");

    //EnQueue test
    InitQueue(&Q);
    for (int i = 1; i <= 5; i++)
    {
        EnQueue(&Q, 100 + i);
    }
    printf("EnQueue后队列为: ");
    QueueTraverse(&Q, visit);

    //QueueLength test
    printf("队列长度为: %d\n", QueueLength(Q));
}

```

```

//GetHead test
QElemType* e = 0;
GetHead(Q, &e);
printf("队头元素为: %d\n", e);


//DeQueue test
QElemType* f = 0;
DeQueue(&Q, &f);
printf("删除元素为: %d\n", f);
printf("DeQueue后队列为: ");
QueueTraverse(&Q, visit);

//ClearQueue test
ClearQueue(&Q);
printf("ClearQueue后\n");

//QueueEmpty test
if (QueueEmpty(Q))
    printf("队列为空\n");
else
    printf("队列非空\n");

system("pause");
return 0;
}

```

 E:\大二上\数据结构\代码保存\LinkQueue\Debug\

```

InitQueue success!
DestroyQueue sucess!
EnQueue后队列为: 101 102 103 104 105
队列长度为: 5
队头元素为: 101
删除元素为: 101
DeQueue后队列为: 102 103 104 105
ClearQueue后
队列为空
请按任意键继续. . .

```

2. 循环队列

(1) SequenceQueue.h

```

#pragma once
#ifndef SEQUENCEQUEUE_H
#define SEQUENCEQUEUE_H

```

```

#include <stdlib.h>
#include "Status.h"

#define MAXQSIZE 100//最大队列长度

typedef int Status;
typedef int QElemType;
typedef struct
{
    QElemType* base;//动态分配存储空间
    int front;//头指针，若队列不空，指向队列头元素
    int rear;//尾指针，若队列不空，指向队列尾元素的下一个位置
}SqQueue;

Status InitQueue(SqQueue* Q);//初始化循环队列
Status EnQueue(SqQueue* Q, QElemType e);//入队列
Status DeQueue(SqQueue* Q, QElemType* e);//出队列
#endif

```

(2) Sequence.c

```

#pragma once
#pragma warning(disable:4996)
#include "SequenceQueue.h"
#include <stdio.h>

Status InitQueue(SqQueue* Q)
{
    (*Q).base = (QElemType*)malloc(MAXQSIZE * sizeof(QElemType));
    if (!(*Q).base)
        exit(OVERFLOW);//存储分配失败
    (*Q).front = (*Q).rear = 0;
    return OK;
}

Status EnQueue(SqQueue* Q, QElemType e)
{
    //插入元素e为Q的新的队尾元素
    if (((*Q).rear + 1) % MAXQSIZE == (*Q).front)
        return ERROR;//队列满
    (*Q).base[(*Q).rear] = e;
    (*Q).rear = ((*Q).rear + 1) % MAXQSIZE;
    return OK;
}

```

```

Status DeQueue(SqQueue* Q, QElemType* e)
{
    //若队列不空，则删除Q的队头元素，用e返回其值，并返回OK；否则返回ERROR
    if ((*Q).front == (*Q).rear)
        return ERROR;
    *e = (*Q).base[(*Q).front];
    (*Q).front = ((*Q).front + 1) % MAXQSIZE;
    return OK;
}

```


(3) test.c

```

#include <stdio.h>
#include "CircleQueue.h"
int main()
{
    SqQueue Q;
    //InitQueue test
    if (InitQueue(&Q))
        printf("InitQueue success!\n");
    else
        printf("InitQueue unsuccess!\n");
    //EnQueue test & DeQueue test
    for (int i = 1; i <= 5; i++)
    {
        EnQueue(&Q, 100 + i);
    }
    QElemType* x = 0;
    DeQueue(&Q, &x);
    if (x==101)
        printf("EnQueue & DeQueue success\n");
    else
        printf("EnQueue & DeQueue unsuccess\n");

    system("pause");
    return 0;
}

```

 E:\大二上\数据结构\代码保存\CircleQueue\Debug\CircleQueue.exe

```

InitQueue success!
EnQueue & DeQueue success
请按任意键继续. . .

```


五. 实验总结

1. 学会了队列的创建
2. 考虑队列空间中单元闲置问题，在空间大小为 100 的情况下，根据设置，只能插入 100 个元素

CS E:\大二上\数据结构\代码保存\CircleQueue\Debug\CircleQueue.exe

```
InitQueue success!  
插入的值为101  
插入的值为102  
插入的值为103  
插入的值为104  
插入的值为105  
插入的值为106  
插入的值为107  
插入的值为108  
插入的值为109  
插入的值为110  
插入的值为111  
插入的值为112  
插入的值为113  
插入的值为114  
插入的值为115  
插入的值为116  
插入的值为117  
插入的值为118  
插入的值为119  
插入的值为120  
插入的值为121  
插入的值为122  
插入的值为123  
插入的值为124  
插入的值为125  
插入的值为126  
插入的值为127  
插入的值为128  
插入的值为129
```

.....

CS 选择E:\大二上\数据结构\代码保存\CircleQueue\Debug\CircleQueue.exe

```
插入的值为175  
插入的值为176  
插入的值为177  
插入的值为178  
插入的值为179  
插入的值为180  
插入的值为181  
插入的值为182  
插入的值为183  
插入的值为184  
插入的值为185  
插入的值为186  
插入的值为187  
插入的值为188  
插入的值为189  
插入的值为190  
插入的值为191  
插入的值为192  
插入的值为193  
插入的值为194  
插入的值为195  
插入的值为196  
插入的值为197  
插入的值为198  
插入的值为199  
队满
```