**信息科学与工程学院**

**2020－2021学年第一学期**

实 验 报 告

课程名称： 数据结构与数据库技术

实验名称： 实验二

专 业 班 级 通信工程 二班

学 生 学 号 201800121050

学 生 姓 名 孟麟芝

实 验 时 间 2020年10月9日

实验报告

## 【实验目的】

1.复习C语言相关内容，熟练掌握结构体、指针的使用。

2.加深对抽象数据类型ADT栈和ADT队列的理解。

## 【实验要求】

1.编写程序实现ADT栈的定义，及常用操作（数组或指针实现）：生成栈；Push和

Pop

2.编写程序实现ADT队列的定义，及常用操作：生成队列；Enqueues入列；Isempty判断是否队列为空。

## 【实验具体内容】

## 【ADT栈的实现】

1.实验库函数stackar.c源代码：

#include "stackar.h"

#include "fatal.h"

#include <stdlib.h>

#define EmptyTOS (-1)

#define MinStackSize (5)

struct StackRecord

{

    int Capacity;

    int TopOfStack;

    ElementType \*Array;

};

/\* START: fig3\_48.txt \*/

int IsEmpty(Stack S)

{

    return S->TopOfStack == EmptyTOS;

}

/\* END \*/

int IsFull(Stack S)

{

    return S->TopOfStack == S->Capacity - 1;

}

/\* START: fig3\_46.txt \*/

Stack CreateStack(int MaxElements)

{

    Stack S;

    /\* 1\*/ if (MaxElements < MinStackSize)

        /\* 2\*/ Error("Stack size is too small");

    /\* 3\*/ S = malloc(sizeof(struct StackRecord));

    /\* 4\*/ if (S == NULL)

        /\* 5\*/ FatalError("Out of space!!!");

    /\* 6\*/ S->Array = malloc(sizeof(ElementType) \* MaxElements);

    /\* 7\*/ if (S->Array == NULL)

        /\* 8\*/ FatalError("Out of space!!!");

    /\* 9\*/ S->Capacity = MaxElements;

    /\*10\*/ MakeEmpty(S);

    /\*11\*/ return S;

}

/\* END \*/

/\* START: fig3\_49.txt \*/

void MakeEmpty(Stack S)

{

    S->TopOfStack = EmptyTOS;

}

/\* END \*/

/\* START: fig3\_47.txt \*/

void DisposeStack(Stack S)

{

    if (S != NULL)

    {

        free(S->Array);

        free(S);

    }

}

/\* END \*/

/\* START: fig3\_50.txt \*/

void Push(ElementType X, Stack S)

{

    if (IsFull(S))

        Error("Full stack");

    else

        S->Array[++S->TopOfStack] = X;

}

/\* END \*/

/\* START: fig3\_51.txt \*/

ElementType

Top(Stack S)

{

    if (!IsEmpty(S))

        return S->Array[S->TopOfStack];

    Error("Empty stack");

    return 0; /\* Return value used to avoid warning \*/

}

/\* END \*/

/\* START: fig3\_52.txt \*/

void Pop(Stack S)

{

    if (IsEmpty(S))

        Error("Empty stack");

    else

        S->TopOfStack--;

}

/\* END \*/

/\* START: fig3\_53.txt \*/

ElementType

TopAndPop(Stack S)

{

    if (!IsEmpty(S))

        return S->Array[S->TopOfStack--];

    Error("Empty stack");

    return 0; /\* Return value used to avoid warning \*/

}

/\* END \*/

2.实验头文件stackar.h源代码：

typedef int ElementType;

/\* START: fig3\_45.txt \*/

#ifndef \_Stack\_h

#define \_Stack\_h

struct StackRecord;

typedef struct StackRecord \*Stack;

int IsEmpty(Stack S);

int IsFull(Stack S);

Stack CreateStack(int MaxElements);

void DisposeStack(Stack S);

void MakeEmpty(Stack S);

void Push(ElementType X, Stack S);

ElementType Top(Stack S);

void Pop(Stack S);

ElementType TopAndPop(Stack S);

#endif /\* \_Stack\_h \*/

/\* END \*/

3.实验主文件main.c源代码：

#include "stackar.h"

#include "fatal.h"

#include <stdlib.h>

void main()

{

    int m = 0;

    Stack myStack = CreateStack(6);

    printf("下面将六个0~9的随机数入栈\n");

    for (int i = 0; i <= 5; i++)

    {

        Push(rand() % 10, myStack);

        printf("第%d个数入栈后，栈顶为：%d\n", i + 1, Top(myStack));

    }

    printf("注意：这时再进行入栈操作将返回栈满提示并退出程序\n");

    //Push(3, myStack);

    printf("下面将六个数出栈\n");

    for (int i = 0; i <= 5; i++)

    {

        m = TopAndPop(myStack);

        printf("第%d个出栈的数为：%d\n", i + 1, m);

    }

    printf("判断这时栈是否为空（1代表是，0代表否）：%d\n", IsEmpty(myStack));

}

## 【栈实现实验记录与说明分析】

此处由于课本中的库函数简洁而强大，本次栈的实现完全调用了课本中的库函数。

程序的核心在于其定义了一个结构体变量来表示栈，即：

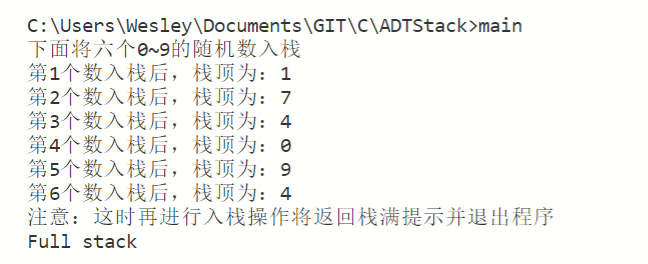
1. **struct** StackRecord
2. {
3. //栈最大能容纳的元素个数
4. **int** Capacity;
5. //栈目前的顶部
6. **int** TopOfStack;
7. //存放栈元素的数组
8. ElementType \*Array;
9. };

要注意，课本中的库函数缺乏对结构体中TopOfStack的初始化（置0），其实这是不合适的。

对栈的一切操作围绕这个结构体来展开，如Push操作：

1. **void** Push(ElementType X, Stack S)
2. {
3. //若栈满则调用Error()报错并退出程序
4. **if** (IsFull(S))
5. Error("Full stack");
6. **else**
7. //否则先将TopOfStack，即栈顶指针自增，再将新元素放入
8. S->Array[++S->TopOfStack] = X;
9. }

其效果如下：

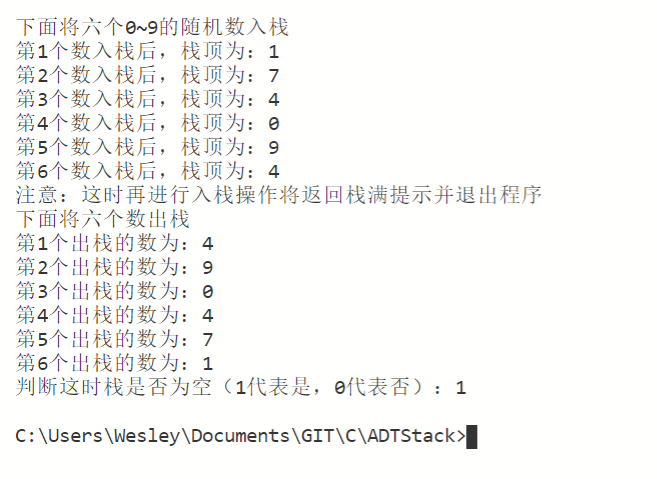


Pop操作则相反：

1. **void** Pop(Stack S)
2. {
3. **if** (IsEmpty(S))
4. Error("Empty stack");
5. **else**
6. S->TopOfStack--;
7. }

这两个操作都对栈满/栈空进行了检验，较为完善。

下面是对入栈出栈操作的整体演示：



这样也可以看出栈的规则，即“先入后出，后入先出“。

## 【ADT队列的实现】

1.实验库函数queue.c源代码：

#include "queue.h"

#include "fatal.h"

#include <stdlib.h>

#define MinQueueSize (5)

struct QueueRecord

{

    int Capacity;

    int Front;

    int Rear;

    int Size;

    ElementType \*Array;

};

/\* START: fig3\_58.txt \*/

int IsEmpty(Queue Q)

{

    return Q->Size == 0;

}

/\* END \*/

int IsFull(Queue Q)

{

    return Q->Size == Q->Capacity;

}

Queue CreateQueue(int MaxElements)

{

    Queue Q;

    /\* 1\*/ if (MaxElements < MinQueueSize)

        /\* 2\*/ Error("Queue size is too small");

    /\* 3\*/ Q = malloc(sizeof(struct QueueRecord));

    /\* 4\*/ if (Q == NULL)

        /\* 5\*/ FatalError("Out of space!!!");

    /\* 6\*/ Q->Array = malloc(sizeof(ElementType) \* MaxElements);

    /\* 7\*/ if (Q->Array == NULL)

        /\* 8\*/ FatalError("Out of space!!!");

    /\* 9\*/ Q->Capacity = MaxElements;

    /\*10\*/ MakeEmpty(Q);

    /\*11\*/ return Q;

}

/\* START: fig3\_59.txt \*/

void MakeEmpty(Queue Q)

{

    Q->Size = 0;

    Q->Front = 1;

    Q->Rear = 0;

}

/\* END \*/

void DisposeQueue(Queue Q)

{

    if (Q != NULL)

    {

        free(Q->Array);

        free(Q);

    }

}

/\* START: fig3\_60.txt \*/

static int

Succ(int Value, Queue Q)

{

    if (++Value == Q->Capacity)

        Value = 0;

    return Value;

}

void Enqueue(ElementType X, Queue Q)

{

    if (IsFull(Q))

        Error("Full queue");

    else

    {

        Q->Size++;

        Q->Rear = Succ(Q->Rear, Q);

        Q->Array[Q->Rear] = X;

    }

}

/\* END \*/

ElementType

Front(Queue Q)

{

    if (!IsEmpty(Q))

        return Q->Array[Q->Front];

    Error("Empty queue");

    return 0; /\* Return value used to avoid warning \*/

}

void Dequeue(Queue Q)

{

    if (IsEmpty(Q))

        Error("Empty queue");

    else

    {

        Q->Size--;

        Q->Front = Succ(Q->Front, Q);

    }

}

ElementType

FrontAndDequeue(Queue Q)

{

    ElementType X = 0;

    if (IsEmpty(Q))

        Error("Empty queue");

    else

    {

        Q->Size--;

        X = Q->Array[Q->Front];

        Q->Front = Succ(Q->Front, Q);

    }

    return X;

}

2.实验头文件queue.h源代码：

typedef int ElementType;

/\* START: fig3\_57.txt \*/

#ifndef \_Queue\_h

#define \_Queue\_h

struct QueueRecord;

typedef struct QueueRecord \*Queue;

int IsEmpty(Queue Q);

int IsFull(Queue Q);

Queue CreateQueue(int MaxElements);

void DisposeQueue(Queue Q);

void MakeEmpty(Queue Q);

void Enqueue(ElementType X, Queue Q);

ElementType Front(Queue Q);

void Dequeue(Queue Q);

ElementType FrontAndDequeue(Queue Q);

#endif /\* \_Queue\_h \*/

/\* END \*/

3.实验主文件main.c源代码：

#include "queue.h"

#include "fatal.h"

#include <stdlib.h>

#include "queue.c"

void main()

{

    Queue myQueue;

    myQueue = CreateQueue(6);

    printf("下面将六个0~9的随机数入队：\n");

    for (int i = 0; i < 6; i++)

    {

        int randNum;

        randNum = rand() % 10;

        Enqueue(randNum, myQueue);

        printf("第%d个入队的数为：%d\n", i + 1, randNum);

    }

    printf("注意：这时再进行入队操作将返回队列已满提示并退出程序\n");

    //Enqueue(3, myQueue);

    printf("下面将六个数出队：\n");

    for (int i = 0; i < 6; i++)

    {

        printf("第%d个出队的数为：%d\n", i + 1, FrontAndDequeue(myQueue));

    }

    printf("判断这时队列是否为空（1代表是，0代表否）：%d\n", IsEmpty(myQueue));

    return (0);

}

## 【队列实现实验记录与说明分析】

该部分由于课本的实现十分完善，故完全调用了课本中提供的库函数，所有的函数围绕下面这个结构体展开：

1. **struct** QueueRecord
2. {
3. //总容量
4. **int** Capacity;
5. //头位置
6. **int** Front;
7. //尾位置
8. **int** Rear;
9. //目前队列的长度
10. **int** Size;
11. //存放数据的数组
12. ElementType \*Array;
13. };

首先是空队检验程序的实现：

1. **int** IsEmpty(Queue Q)
2. {
3. **return** Q->Size == 0;
4. }

这一程序较为简单，只需要检验Size（目前队列长度）是否为0即可。

课本的程序中，使用Succ()函数实现了循环数组，即Front或Rear到达尾端，就要绕回到开头：

1. Succ(**int** Value, Queue Q)
2. {
3. **if** (++Value == Q->Capacity)
4. Value = 0;
5. **return** Value;
6. }

这一函数在本次的实验中并没有用到，不过它解决了Front和Rear不断自增自减导致溢出的问题。

入队的实现较为简单：

1. **void** Enqueue(ElementType X, Queue Q)
2. {
3. **if** (IsFull(Q))
4. Error("Full queue");
5. **else**
6. {
7. Q->Size++;
8. Q->Rear = Succ(Q->Rear, Q);
9. Q->Array[Q->Rear] = X;
10. }
11. }

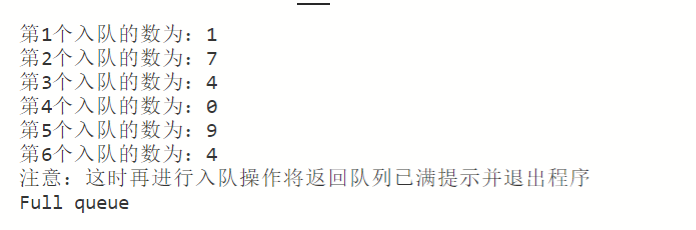
首先对队是否为空、Rear是否需要绕回进行了检验，最后将数据放进Array数组中Rear代表的位置即可。

出队的操作正好是与入队相对应的，程序如下，要注意，这一函数在对元素出队的同时还获取了这个元素。

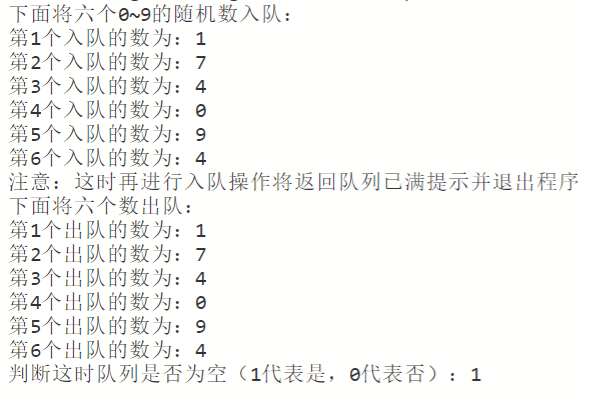
1. FrontAndDequeue(Queue Q)
2. {
3. ElementType X = 0;
5. **if** (IsEmpty(Q))
6. Error("Empty queue");
7. **else**
8. {
9. Q->Size--;
10. X = Q->Array[Q->Front];
11. Q->Front = Succ(Q->Front, Q);
12. }
13. **return** X;
14. }

对队是否为空和Front是否需要绕回作检验后，将队首的元素取出即可。

下面是对程序的一些测试：



下面是对队的特性的测试：



可见结果是符合预期的。

## 【实验心得】

本次实验与上次实验的思路是一致的，利用了结构体这一数据类型，构造出了我们想要的“对象”，通过对它各个属性的定义、操作，即可实现不同的功能，如本次实验中实现了队列和栈的功能。这两个要比链表简单许多，队列就像一个“水管”，这边进那边出，栈则像一个网球筒。如此简单的模型，却能够发展出深刻、应用广泛的理论，如队列把人们引向了对排队论的思考，栈在计算机中大放异彩，实在令人啧啧称奇。