

**数据结构实验报告**

**学 院** 信息工程学院

**专 业** 计算机类（大类招生）

**班 级** 2016240210

**姓 名** 瞿强鑫

**学 号** 2016902094

**指导教师** 史昕

**完成时间** 2017.4.14

**顺序栈：**

**一、实验目的**

1. 掌握栈的顺序存储结构；
2. 验证顺序栈及其基本操作的实现；
3. 验证栈的操作特性。

**二、实验设备（环境）及要求**

电脑一台，VC6.0环境下运行

**三、实验内容（步骤）与方法**

本程序通过建立一个空的栈，实现对顺序栈的插入、删除和取顶元素等操作。

//建立顺序栈类

const int StackSize=5;

**template <class DataType>**

**class SeqStack**

{

public:

SeqStack(){top=-1;} //构造函数，初始化一个空栈

~SeqStack(){} //析构函数为空

void Push(DataType x); //入栈操作

DataType Pop(); //出栈操作

void PrintStack(); //遍历栈

DataType GetTop(){return data[top];}//取栈顶元素

int Empty(){if(top==-1) return 1;

else return 0;} //判断栈是否为空

private:

DataType data[StackSize]; //存放栈元素的数组

int top; //栈顶指针

};

//入栈操作

**template <class DataType>**

**void SeqStack<DataType>::Push(DataType x)**

{

if(top==StackSize-1) throw"上溢";

data[++top]=x;

}

//出栈操作

**template <class DataType>**

**DataType SeqStack<DataType>::Pop()**

{

if(top==-1) throw"上溢";

int x=data[top--];

return x;

}

//遍历栈

**template <class DataType>**

**void SeqStack<DataType>::PrintStack()**

{

cout<<"栈遍历为:";

for(int i=top;i>-1;i--)

{

cout<<data[i]<<" ";

}

}

**四、实验结果与分析**

main（）函数：

int i;

SeqStack<int> s1;

int x;

cout<<"请依次插入五个栈元素：";

for(i=0;i<StackSize;i++)

{

cin>>x;

s1.Push(x);

}

s1.PrintStack();

cout<<endl;

cout<<"出栈删除元素为："<<s1.Pop()<<endl;

cout<<"再次遍历：";

s1.PrintStack();

cout<<endl;

cout<<"取栈顶元素："<<s1.GetTop()<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

return 0;

程序开始插入5，4，7，8，9五个元素，操作达到目的，如图1：



图1

出栈操作，操作达到目的，如图2：



图2

取栈顶元素，操作达到目的，如图3：



图3

**链队列：**

1. **实验目的**

1.掌握队列的链接存储结构；

2.验证链队列的存储结构和基本操作的实现；

3.验证队列的操作特性。

1. **实验设备（环境）及要求**

电脑一台，VC6.0环境下运行

1. **实验内容（步骤）与方法**

本程序建立一个空的对列，对该对列实现了插入、删除和取队头元素等操作。

//建立链队列类

**template <class DataType>**

**struct Node**

{

DataType data;

Node<DataType>\*next;

};

**template <class DataType>**

**class LinkQueue**

{

public:

LinkQueue(); //构造函数，初始化一个空队列

~LinkQueue(){} //析构函数

void EnQueue(DataType x); //入队操作

DataType DeQueue(); //出队操作

DataType GetQueue(); //取链队列的队头元素

int Empty() {if(rear==front) return 1;

else return 0;} //判断链队列是否为空

void PrintQueue(); //遍历队列

private:

Node<DataType> \*front,\*rear; //队头和队尾指针

};

//构造函数，初始化一个空队列

**template <class DataType>**

**LinkQueue<DataType>::LinkQueue()**

{

Node<DataType> \*s=new Node<DataType>;s->next=NULL;

front=rear=s;

}

//入队操作

**template <class DataType>**

**void LinkQueue<DataType>::EnQueue(DataType x)**

{

Node<DataType> \*s=new Node<DataType>;s->data=x;

s->next=NULL;

rear->next=s;

rear=s;

}

//遍历队列

**template <class DataType>**

**void LinkQueue<DataType>::PrintQueue()**

{

cout<<"遍历为：";

Node<DataType> \*p=front->next;

while(p!=NULL)

{

cout<<p->data<<" ";

p=p->next;

}

cout<<endl;

}

//出队操作

**template <class DataType>**

**DataType LinkQueue<DataType>::DeQueue()**

{

if(rear==front) throw "下溢";

Node<DataType> \*p=front->next;int x=p->data;

front->next=p->next;

if(p->next==NULL) rear=front;

delete p;

return x;

}

//取链队列的队头元素

**template <class DataType>**

**DataType LinkQueue<DataType>::GetQueue()**

{

return front->next->data;

}

1. **实验结果与分析**

main（）函数：

LinkQueue<int> l1;

cout<<"请依次插入五个队列元素：";

for(i=0;i<5;i++)

{

cin>>x;

l1.EnQueue(x);

}

l1.PrintQueue();

cout<<"删除队头元素后，";

l1.DeQueue();

l1.PrintQueue();

cout<<"取队头元素为："<<l1.GetQueue()<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

return 0;

程序开始运行，插入7，8，5，6，2五个元素，操作成功，如图1：



图1

删除队头元素，操作成功，遍历如图2：



图2

取队头元素，操作成功，如图3：



图3

**两个程序的实验心得与体会：**

在一番折腾后，终于在一个程序里写出了两个操作。其实回顾这两个操作，感觉是：顺序栈是在数组的基础上加了一些限制条件，如先进后出，只能对栈顶进行操作；而链队列其实也和链表的关系紧密，不过加了先进先出、只能对队头删除队尾插入操作。

在本次实验操作过程中，我发现了自己对for语句使用的生疏，限制条件和i++使用放反了，导致整个for语句无法实现，但这却不是error，仅仅是一个warning，如果不注意完全不会意识到自己的错误。这个错误值得我反省，我们在写程序执行时候，不能忽略任何一个警告，可能一个警告就会导致整个程序的错误。课本上的程序并不是完全正确，我们一次次的敲上课本源代码，会发现许多未定义的错误，这是我们需要注意的。

此外，我也从这次得到了些许心得体会。写程序时得保持心态的平稳，不能焦急担忧，必须放平心态，才能稳定发挥自己的水平，焦虑只会导致一个个小错误，对于程序毫无用处。还有，写程序得首先明确自己的目的，再规划程序实现的过程和操作进行，千万不能急功近利，从而忽略了程序的质量。