安 徽 新 华 学 院 实 验 报 告

姓 名 张卢 学号 1632104249专业班级 16软件2班

课程名称 数据结构 实验名称 栈与队列的应用 实验日期 2017.10.23

同组人员 指导老师 汪红霞 得分

**实 验 项 目 四**

**【实验名称】** 栈与队列的应用

**【实验类型】**基础性实验

**【实验目的与要求】**

* 掌握栈和队列这两种特殊的线性表，熟悉它们的特性，能在实际问题背景下灵活运用；
* 能够在两种存储结构上实现栈和队列的基本运算，特别注意栈满、栈空、队列满、队列空的判断条件及描述方法；
* 理解栈与队列的应用，借助栈与队列来解决某些实际应用问题，如数制转换、括号匹配的检验，杨辉三角的实现问题等。

**【实验主要仪器设备及台套数】**

一人一台安装Visual C++的电脑

**【实验学时】** 4学时

**【实验内容】**

1. 实现顺序栈和链栈的各种操作（入栈、出栈、取栈顶元素、判断栈空、栈置空等）

（1）顺序栈

#include <iostream.h>

typedef int elemtype;

const int maxsize=60; //定义栈的最大容量为maxlen

class seqstack

{

public:

elemtype stack[maxsize]; //将栈中元素定义为elemtype类型

int top; //指向栈顶位置的指针

public:

void inistack( seqstack &s); //将栈s置为一个空栈（不含任何元素）

void push(seqstack &s ,int n) ; //将元素x插入到栈s中，也称为“入栈”、“插入”、“压入”

void pop( seqstack &s) ; //删除栈s中的栈顶元素，也称为“退栈”、“删除”、“弹出”

elemtype gettop( seqstack s) ; //取栈s中栈的顶元素

bool empty( seqstack s) ; //判栈s是否为空

void print(seqstack s);

};

void seqstack::inistack( seqstack &s)//形参一定要写明类型个数

{

s.top=0;

}

void seqstack::push(seqstack &s,int n)

{

for(int i=1;i<=n;i++){

if (s.top==maxsize-1) cout<<"overflow";

else

{

int y;

cout<<"请放入第"<<i<<"个元素的值: ";

cin>>y;

s.top++;

s.stack[s.top]=y;

}

}

}

void seqstack::pop(seqstack &s )

{

if (s.top==0) cout<<"underflow";

else s.top--;

}

elemtype seqstack::gettop(seqstack s )

{

if (s.top==0) {cout<<"underflow";return 0;}

else return s.stack[s.top];

}

bool seqstack::empty(seqstack s)

{

if (s.top==0) return true;

else return false;

}

void seqstack::print(seqstack s)

{

for(int i=1;i<=s.top;i++)

cout<<s.stack[i]<<' ';

cout<<endl;

}

void main()

{

/\*

seqstack s1;

int x,y;

s1.inistack(s1);

cout<<"请选择放入元素个数: ";

cin>>x;

s1.push(s1,x);

s1.print(s1);

bool str=s1.empty(s1);

if(str=true)

cout<<"栈不为空"<<endl;

else

cout<<"栈为空"<<endl;

s1.gettop(s1);

s1.print(s1);

s1.pop(s1);

s1.print(s1);

s1.print(s1);

\*/

seqstack s1;

s1.inistack(s1);

cout<<"输入1为置栈空"<<endl;

cout<<"输入2为向栈插入元素"<<endl;

cout<<"输入3为退栈"<<endl;

cout<<"输入4为取栈顶元素"<<endl;

cout<<"输入5为判栈师是否为空"<<endl;

cout<<"输入6为查看栈"<<endl;

int q;

cin>>q;

while(q!=0)

{

switch(q)

{

case 1: {s1.inistack(s1);}break;

case 2: {

int x;

cout<<"请选择放入元素个数: ";

cin>>x;

s1.push(s1,x);}break;

case 3: {s1.pop(s1);}break;

case 4: {cout<<"栈顶元素为: "<<s1.gettop(s1)<<endl;}break;

case 5: {bool str=s1.empty(s1);

if(str=true)

cout<<" 栈不为空"<<endl;

else

cout<<" 栈为空"<<endl;}break;

case 6: {s1.print(s1);}break;

default:cout<<"您输入的位置有误！"<<endl;

}

cout<<"请输入数字: ";

cin>>q;

}

}

(2)链栈

#include <iostream.h>

typedef int elemtype;

class link

{

public:

elemtype data; //元素类型

link \*next;

public:

void inistack(link \*top);

void push(link \*top,int j);

void pop(link \*top);

elemtype gettop(link \*top);

bool empty(link \*top);

void print(link \*top);

};

void link::inistack(link \*top)

{

top->next=NULL;

}

void link::push(link \*top,int j)

{

elemtype x;

for(int i=0;i<j;i++)

{

cout<<"请输入元素: "<<endl;

cin>>x;

link \*s;

s=new link;

s->data=x;

s->next=top->next;

top->next=s;

}

}

void link::pop(link \*top)

{

link \*s;

s=top->next;

if(s!=NULL)

{

top->next=s->next;

delete(s);}

}

elemtype link::gettop(link \*top)

{

if(top->next!=NULL)

return top->next->data;

else

return NULL;

}

bool link::empty(link \*top)

{

if(top->next==NULL)

return(true);

else

return(false);

}

void link::print(link \*top)

{

link \*p;

p=top->next;

while(p->next!=NULL)

{

cout<<p->data<<"->"; //输出表中非最后一个元素

p=p->next;

}

cout<<p->data; //输出表中最后一个元素

cout<<endl;

}

void main()

{

link s1,\*p;

p=&s1;

s1.inistack(p);

cout<<"输入1为初始化栈"<<endl;

cout<<"输入2为进栈"<<endl;

cout<<"输入3为退栈"<<endl;

cout<<"输入4为取栈顶元素"<<endl;

cout<<"输入5为判栈是否为空"<<endl;

cout<<"输入6为查看栈"<<endl;

int q;

cin>>q;

while(q!=0)

{

switch(q)

{

case 1: {s1.inistack(p);}break;

case 2: {

int x;

cout<<"请选择放入元素个数: ";

cin>>x;

s1.push(p,x);}break;

case 3: {s1.pop(p);}break;

case 4: {cout<<"栈顶元素为: "<<s1.gettop(p)<<endl;}break;

case 5: {bool str=s1.empty(p);

if(str=true)

cout<<" 栈不为空"<<endl;

else

cout<<" 栈为空"<<endl;}break;

case 6: {s1.print(p);}break;

default:cout<<"您输入的位置有误！"<<endl;

}

cout<<"请输入数字: ";

cin>>q;

}

}

2、实现循环队列和链队的各种操作（入栈、出栈、取栈顶元素、判断栈空、栈置空等）…………

（1）顺序队列

#include <iostream.h>

typedef int elemtype;

const int maxsize= 60;

class seqqueue

{

public:

elemtype queue[maxsize]; //将队列中元素定为数组型，元素类型为elemtype

int front; //队头指针

int rear; //队尾指针

void INIQUEUE(seqqueue &Q); //将队列Q设置成一个空队列

void ENQUEUE (seqqueue &Q,int y); //将元素X插入到队尾中，也称“进队”，“插入”

void DLQUEUE(seqqueue &Q); //将队列Q的队头元素删除，也称“退队”、“删除”

elemtype GETHEAD(seqqueue Q ); //得到队列Q的队头元素之值

bool EMPTY(seqqueue Q );

void print(seqqueue Q); //判断队列Q是否为空，若为空返回真，否则返回假

};

void seqqueue::INIQUEUE(seqqueue &Q )

{

Q.front=Q.rear=maxsize-1;

}

void seqqueue::ENQUEUE (seqqueue &Q,int y)

{

for(int i=0;i<y;i++)

{

if ((Q.rear+1)%maxsize == Q.front)

cout<<"overflow";

else {

elemtype x;

cout<<"请输入要添加的元素: ";

cin>>x;

Q.rear=(Q.rear+1)%maxsize;

Q.queue[Q.rear]=x;

}

}

}

void seqqueue::DLQUEUE(seqqueue &Q )

{

if (Q.rear==Q.front) cout<<"underflow";

else

Q.front =(Q.front+1)%maxsize;

}

elemtype seqqueue::GETHEAD(seqqueue Q )

{

if (Q.rear==Q.front)

{ cout<<"underflow";return NULL;}

else{

return Q.queue[(Q.front+1)%maxsize];}

}

bool seqqueue::EMPTY(seqqueue Q )

{

if (Q.rear==Q.front)

return true;

else return false;

}

void seqqueue::print(seqqueue Q)

{

int i=Q.front;

for(int j=1;j<=(Q.rear-Q.front+maxsize)%maxsize;j++)

{

i=(i+1)%maxsize;

cout<<Q.queue[i]<<" ";

}

cout<<endl;

}

void main()

{

seqqueue s1;

s1.INIQUEUE(s1);

cout<<"输入1为队列初始化"<<endl;

cout<<"输入2为向队列插入元素"<<endl;

cout<<"输入3为删除对头元素"<<endl;

cout<<"输入4为得到队列队头元素的值"<<endl;

cout<<"输入5为判队列是否为空"<<endl;

cout<<"输入6为查看队列"<<endl;

int q;

cin>>q;

while(q!=0)

{

switch(q)

{

case 1: {s1.INIQUEUE(s1);}break;

case 2: {

int x;

cout<<"请选择放入元素个数: ";

cin>>x;

s1.ENQUEUE(s1,x);}break;

case 3: {s1.DLQUEUE(s1);}break;

case 4: {

cout<<"对头元素为: "<<s1.GETHEAD(s1)<<endl;}break;

case 5: {if(s1.EMPTY(s1)==true)

cout<<"队列为空"<<endl;

else

cout<<"队列不为空"<<endl;}break;

case 6: {s1.print(s1);}break;

default:cout<<"您输入的位置有误！"<<endl;

}

cout<<"请输入数字: ";

cin>>q;

}

}

（2）链式队列

#include <iostream.h>

typedef int elemtype;

class link //定义单链表数据类型

{ public:

elemtype data;

link \*next;

};

class linkqueue //定义链队列数据类型

{ public:

link \*front,\*rear; //定义头指针和尾指针

void iniqueue(linkqueue &s);

void enqueue(linkqueue &s, elemtype x);

bool empty(linkqueue s);

elemtype gethead(linkqueue s);

void dlqueue(linkqueue &Q);

void print(linkqueue &Q);

};

void linkqueue::iniqueue(linkqueue &s)

{

link \*p;

p=new link;

p->next=NULL;

s.front=p;

s.rear=p;

}

void linkqueue::enqueue(linkqueue &s, elemtype x)

{

link \*p;

p=new link;

p->data=x;

p->next=s.rear->next;

s.rear->next=p;

s.rear=p;

}

bool linkqueue::empty(linkqueue s)

{

if (s.front==s.rear) return true;

else return false;

}

elemtype linkqueue::gethead(linkqueue s)

{

if (s.front==s.rear) return NULL;

else return s.front->next->data;

}

void linkqueue::dlqueue(linkqueue &s)

{ link \*p;

p=s.front->next;

if(p->next==NULL)

{s.front->next=NULL;s.front=s.rear;}

else s.front->next=p->next;

delete p;

}

void linkqueue::print(linkqueue &Q)

{

link \*p;

// QElemType e;

p=Q.front->next;

// e=p->data;

while(p)

{

// printf("%d",e);

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

p=p->next;

}

cout<<endl;

}

void main()

{

linkqueue s1;

s1.iniqueue(s1);

cout<<"输入1为队列初始化"<<endl;

cout<<"输入2为向队列插入元素"<<endl;

cout<<"输入3为删除对头元素"<<endl;

cout<<"输入4为得到队列队头元素的值"<<endl;

cout<<"输入5为判队列是否为空"<<endl;

cout<<"输入6为查看队列"<<endl;

int q;

cin>>q;

while(q!=0)

{

switch(q)

{

case 1: {s1.iniqueue(s1);}break;

case 2: {

int x;

cout<<"请选择放入元素个数: ";

cin>>x;

s1.enqueue(s1,x);}break;

case 3: {s1.dlqueue(s1);}break;

case 4: {

cout<<"对头元素为: "<<s1.gethead(s1)<<endl;}break;

case 5: {if(s1.empty(s1)==true)

cout<<"队列为空"<<endl;

else

cout<<"队列不为空"<<endl;}break;

case 6: {s1.print(s1);}break;

default:cout<<"您输入的位置有误！"<<endl;

}

cout<<"请输入数字: ";

cin>>q;

}

}

**【实验心得】**

**深刻明白了队列（先进先出）和栈（先进后出）的含义；通过自己编写顺序队列，链式队列，顺序栈，链栈的增删改查等一系列操作后，加深的对顺序表和链表的理解，也学会了如何构建顺序表和链表的模板；体会到指针和引用的各种用法；过程中使用while循环和switch的配套使用，使得程序更加健壮，也体现的代码的复用性**