#include<iostream>

#include<string>

using namespace std;

const int MAXSIZE = 10000;//最大的栈空间为1万

template<class ElemType>

class SqStack {

private:

ElemType\* top;//栈顶指针，在栈空的时候和栈底指针一起指向栈的第一个位置，栈为非空时指向栈顶元素的上面的一个位置

ElemType\* base;//栈底指针，始终指向栈的第一个位置，不管栈中有多少个元素

int maxSize;//用于标记栈中最多可以存放的元素数，同时也是判断栈满不满的标志之一，因为这是顺序栈，所以

//在使用它之前要先申请一块空间

public:

//构造函数:

SqStack() {

maxSize = MAXSIZE;

base = new ElemType[MAXSIZE];

top = base;

}

SqStack(int n) {

base = new ElemType[n];

top = base;

maxSize = n;

}

//析构函数:

~SqStack() {

StackDestory();

cout << "调用了析构函数" << endl;

}

//拷贝构造函数:

SqStack(const SqStack& SqS) {

int L = SqS.getLength();

maxSize = SqS.max\_size();

base = new ElemType[maxSize];

top = base;

ElemType\* bs=SqS.getBase();

int i;

for (i = 0; i < L; ++i) {

\*(base + i) = \*(bs + i);

}

top = base + i;

/\*cout << "调用了拷贝构造函数" << endl;

cout << "这个函数调用后，top-base的值为:" << top - base << endl;

cout << "base所指元素的值为:" << \*base<<endl;\*/

}

//拷贝赋值运算符:

SqStack<ElemType>& operator=(const SqStack<ElemType>& SqS);

//销毁栈:

void StackDestory() {

top = base;

delete[] base;

}

//得到栈底指针:

ElemType\* getBase() const { return base; }

//得到栈顶指针:

ElemType\* getTop\_pointer()const { return top; }

//返回目前栈的元素个数:

int getLength() const;

//返回栈的最大空间:

int max\_size()const { return maxSize; }

//判断栈是否为空:

bool isEmpty() const;

//判断栈是否为满栈:

bool isFull()const;

//用e返回栈顶元素:

bool getTop(ElemType& e) const;

//入栈:

bool push(ElemType& e);

//出栈:

bool pop(ElemType& e);

//遍历栈:

void traverse()const;

// 栈空间加倍:

void doubleSpace();

};

//拷贝赋值运算符:

template<class ElemType>

SqStack<ElemType>& SqStack<ElemType>::operator=(const SqStack<ElemType>& SqS) {

const int L = SqS.getLength();

maxSize = SqS.max\_size();

base = new ElemType[maxSize];

top = base;

ElemType\* bs = SqS.getBase();

int i;

for (i = 0; i < L; ++i) {

\*(base + i) = \*(bs + i);

}

top = base + i;

/\*cout << "调用了拷贝赋值运算符" << endl;

cout << "这个函数调用后，top-base的值为:" << top - base << endl;

cout << "base所指元素的值为:" << \*base << endl;\*/

return \*this;

}

//返回目前栈的元素个数:

template<class ElemType>

int SqStack<ElemType>::getLength() const {

return top - base;

}

//判断栈是否为空:

template<class ElemType>

bool SqStack<ElemType>::isEmpty() const {

if (base == top) return true;

else return false;

}

//判断栈是否为满栈:

template<class ElemType>

bool SqStack<ElemType>::isFull()const {

if (top - base == maxSize) { return true; }

else return false;

}

//用e返回栈顶元素:

template<class ElemType>

bool SqStack<ElemType>::getTop(ElemType& e) const {

if (base == top) { return false; }

else {

ElemType\* t = top;

e = \*(--t);

/\*cout << "调用了返回栈顶元素这个函数:" << endl;

cout << "这个函数调用后，top-base的值为:" << top - base << endl;

cout << "base所指元素的值为:" << \*base << endl;\*/

return true;

}

}

//入栈:

template<class ElemType>

bool SqStack<ElemType>::push(ElemType& e) {

if (top - base == maxSize) { return false; }

else {

\*top++ = e;

/\*cout << "调用了push这个函数:" << endl;

cout << "这个函数调用后，top-base的值为:" << top - base << endl;

cout << "base所指元素的值为:" << \*base << endl << endl;\*/

return true;

}

}

//出栈:

template<class ElemType>

bool SqStack<ElemType>::pop(ElemType& e) {

if (base == top) { return false; }

else {

e = \*(--top);

if (top - base == 0) {//为了使得当栈中的元素只有一个往出弹后，其实没有真正的弹出，只是指针移位，

//不过底部的指针是不会移位的，所以也就相当于这个值没有被弹出，这里选择采取将空间释放再

//重新申请空间的做法来实现这种特殊情况下的真正的弹出

delete[]base;

base = new ElemType[maxSize];

top = base;

//cout << "栈已被弹空" << endl;

}

/\*cout << "调用了pop这个函数:" << endl;

cout << "这个函数调用后，top-base的值为:" << top - base << endl;

cout << "base所指元素的值为:" << \*base << endl << endl;\*/

return true;

}

}

template<class ElemType>

void SqStack<ElemType>::traverse()const {//遍历顺序栈

if (top == base) { return; }

else {

const int L = this->getLength();

for (int i = 0; i < L; ++i) {

cout << \*(base + i) << ' ';

}

}

}

// 栈空间加倍:

template<class ElemType>

void SqStack<ElemType>::doubleSpace() {

ElemType\* tmp\_base = base;

int L = this->getLength();

base = new ElemType[maxSize \* 2];

int i;

for (i = 0; i < L; ++i) {

\*(base + i) = \*(tmp\_base + i);

}

top = base + i;

maxSize \*= 2;

delete[] tmp\_base;

/\*cout << "调用了doubleSpace这个函数:" << endl;

cout << "这个函数调用后，top-base的值为:" << top - base << endl;

cout << "base所指元素的值为:" << \*base << endl << endl;\*/

}

template<class ElemType>

void F(SqStack<ElemType>& S, const int& order) {

cout << "顺序栈S" << order << "的元素如下:" << endl;

S.traverse();

cout << endl;

cout << "顺序栈S" << order << "长度为:"<<S.getLength() << endl;

cout << "顺序栈S" << order << "最多可以放的元素个数:" << S.max\_size() << endl;

}