实验题目：栈的操作

实验内容：实现对栈的创建、入栈、出栈操作

实验环境：

Win10

DevCpp

实验流程：

1.创建有N个元素的栈

2.将N个元素入栈

3.打印栈中元素

4.将栈顶元素出栈

5.销毁栈

代码实现：

#include <stdio.h>

#include <stdlib.h>

#define STACK\_INIT\_SIZE 100//储存空间初始分配量

#define STACKINCREMENT 10//存储空间分配增量

#define OK 1

#define ERROR 0

typedef int StackType; //栈元素类型

typedef struct {

StackType \*base; //在构造之前和销毁之后，base的值为NULL

StackType \*top; //栈顶指针

int stacksize; //当前已分配的存储空间，以元素为单位

}SqStack; //顺序栈

//栈的初始化

int InitStack(SqStack \*p) {

p->base = (StackType\*)malloc(STACK\_INIT\_SIZE \* sizeof(StackType));

if (p->base == NULL) return ERROR; //内存分配失败

p->top = p->base; //栈顶与栈底相同表示一个空栈

p->stacksize = STACK\_INIT\_SIZE;

return OK;

}

//判断栈是否为空

int EmptyStack(SqStack \*p) {

//若为空栈 则返回OK，否则返回ERROR

if (p->top == p->base) return OK;

else return ERROR;

}

//顺序栈的压入

int Push(SqStack \*p,StackType e) {

//插入元素e为新的栈顶元素

if ((p->top - p->base)>= p->stacksize) //栈满，追加储存空间

{

p->base = (StackType\*)realloc(p->base, (p->stacksize + STACKINCREMENT) \* sizeof(StackType));

if (p->base == NULL) return ERROR;// 储存空间分配失败

p->top = p->base + p->stacksize; //可能有人觉得这句有点多余(我当时也是这么想的 后面有解释)

p->stacksize += STACKINCREMENT;

}

\*(p->top) = e;

(p->top)++;

return OK;

}

// 顺序栈的弹出

int Pop(SqStack \*p,StackType \*e) {

//若栈不空，则删除p的栈顶元素，用e返回其值

if (p->top == p->base) return ERROR;

--(p->top);

\*e = \*(p->top);

return OK;

}

//顺序栈的销毁

int DestroyStack(SqStack \*p) {

//释放栈底空间并置空

free(p->base);

p->base = NULL;

p->top = NULL;

p->stacksize = 0;

return OK;

}

//将顺序栈置空 栈还是存在的，栈中的元素也存在，如果有栈中元素的地址任然能调用

int ClearStack(SqStack \*p) {

p->top= p->base;

return OK;

}

//返回顺序栈的元素个数

int StackLength(SqStack p) {

//栈顶指针减去栈底指针等于长度，因为栈顶指针指向当前栈顶元素的下一个位置

return p.top - p.base;

}

//返回顺序栈的栈顶元素

int GetTop(SqStack \*p, StackType \*e) {

//若栈不为空，则用e返回p的栈顶元素

if (p->top > p->base) {

\*e = \*(p->top - 1); return OK;

}

else return ERROR;

}

//从栈顶到栈底对每个元素调用某个函数

int StackTraverse(SqStack p,void (\*pfun)(StackType)/\*函数指针\*/){

//从栈底到栈顶依次对栈中的每个元素调用函数pfun()

while (p.top > p.base)

pfun(\*(p.base)++); //先调用后递增

printf("\n");

return OK;

}

//打印栈中元素

void print(StackType stack) {

printf("%d\n", stack);

}

//测试栈的各种操作

int main() {

int n,i;

StackType \*e,a;

SqStack \*pstack,stack;

pstack = &stack;

e=(StackType\*)malloc(sizeof(StackType)); //为指针e分配内存地址

InitStack(pstack); //初始化栈

if (EmptyStack(pstack) == 1) printf("-------栈为空-------\n");

printf("请输入栈的元素个数:");

scanf("%d", &n);

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

{

scanf("%d", &a);

Push(pstack, a);

}

if (EmptyStack(pstack) == 0) printf("-------栈不为空-----\n");

printf("栈的长度为:%d\n", StackLength(stack));

printf("--------------------\n");

printf("请输入一个入栈元素:");

scanf("%d", &a);

Push(pstack, a);

printf("--------------------\n");

printf("栈中的元素个数为:%d\n", StackLength(stack));

printf("--------------------\n");

GetTop(pstack, e);

printf("栈顶元素为:%d\n", \*e);

printf("--------------------\n");

printf("打印栈中的元素:\n");

StackTraverse(stack, print);

printf("---弹出栈顶元素---\n");

Pop(pstack, e);

printf("弹出的栈顶元素为:%d\n", \*e);

printf("--------------------\n");

GetTop(pstack, e);

printf("栈顶元素为:%d\n", \*e);

printf("--------------------\n");

printf("打印栈中的元素:\n");

StackTraverse(stack, print);

printf("--------------------\n");

printf("----------清空栈-------\n");

if (ClearStack(pstack) == 0) printf("已清空栈\n");

printf("----------销毁栈-------\n");

if (DestroyStack(pstack) == 0) printf("已销毁栈\n");

return 0;

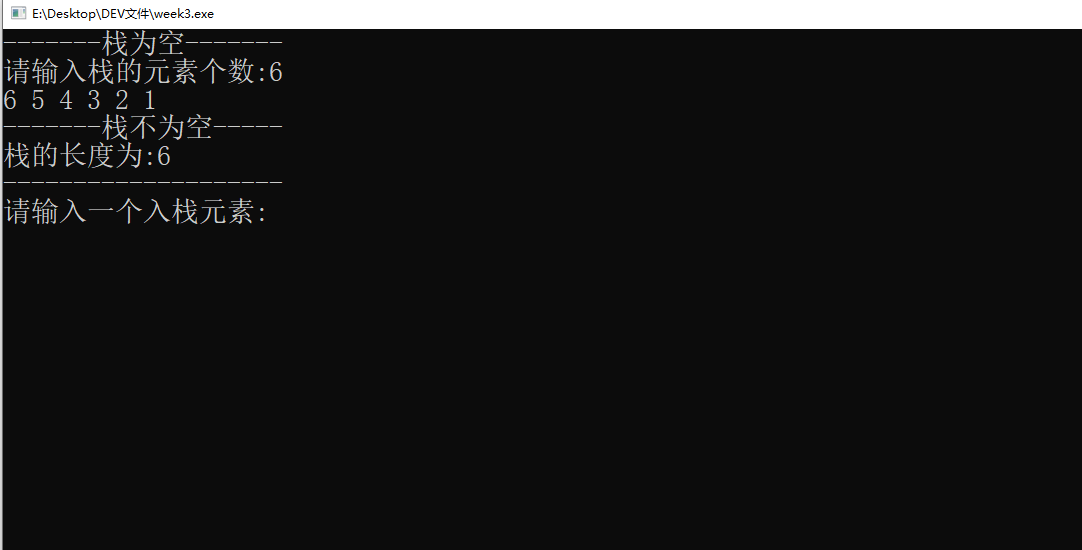
}

运行截图：

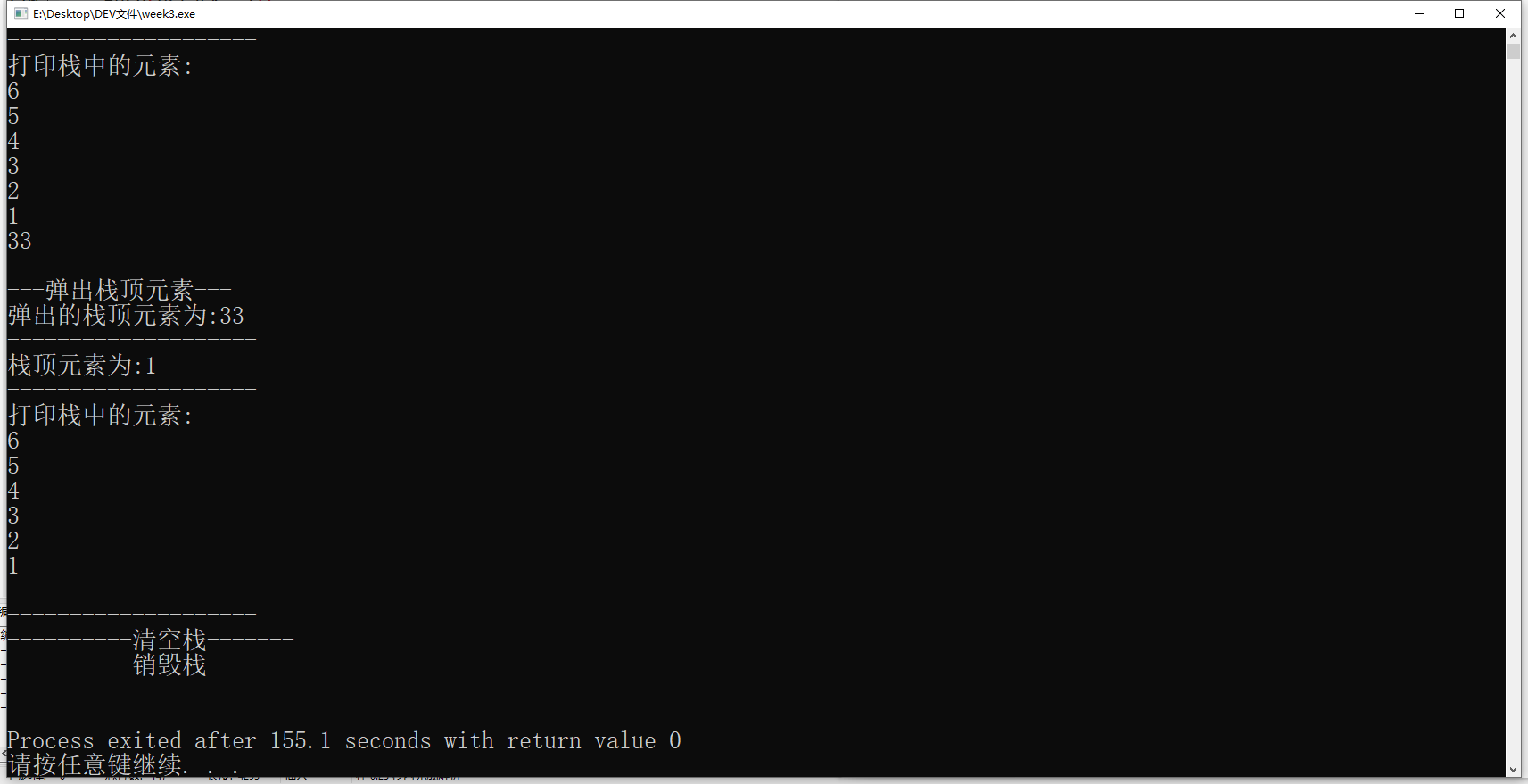
创建栈：



入栈：



对栈进行扩容后弹出栈顶元素并退出：



结果分析：

程序会对栈满后进行自动扩容

弹出栈顶元素后再次打印栈中元素，栈顶元素已经改变

程序结束后会将栈内元素清空，然后销毁