

## 实验报告 (week-03)

2020111235 马靖淳

### 一. 实验任务

1. 顺序栈
2. 栈的应用
3. 递归

### 二. 实验上机时间 7h

### 三. 知识点

#### 1. 顺序栈

- (1) 栈的初始化
- (2) 销毁栈
- (3) 清空栈
- (4) 判断空
- (5) 求栈长度
- (6) 入栈操作
- (7) 出栈操作
- (8) 取栈顶元素操作
- (9) 栈的遍历

#### 2. 栈的应用

- (1) 数制转换
- (2) 括号匹配的检验
- (3) 行编辑程序
- (4) 表达式求值

#### 3. 递归

### 四. 源代码及结果截屏

#### 1. 顺序栈

##### (1)stack.h

```
#ifndef STACK_H
#define STACK_H

#include <stdlib.h>
#include "Status.h"

#define STACK_INIT_SIZE 100//存储空间初始分配量
#define STACKINCREMENT 10//存储空间分配增量

typedef int SElemType;
typedef struct
{
    SElemType *base;//栈底指针
    SElemType* top;//栈顶指针
    int stacksize;//当前可使用最大容量，不是栈中元素个数
}SqStack;
```

```

Status InitStack(SqStack* S);
Status DestroyStack(SqStack *S);
Status ClearStack(SqStack* S);
Status StackEmpty(SqStack S);
Status Push(SqStack* S, SElemType e);
int StackLength(SqStack S);
Status Pop(SqStack* S, SElemType* e);
Status GetTop(SqStack S, SElemType* e);
visit(SElemType e);
Status StackTravers(SqStack *S);
#endif

```

## (2)stack.c

```

#include "Stack.h"
#include <stdio.h>
#include <malloc.h>

Status InitStack(SqStack* S)
{
    (*S).base = (SElemType*)malloc(STACK_INIT_SIZE * sizeof(SElemType));
    if(!(*S).base)
        exit(OVERFLOW);
    (*S).top = (*S).base;
    (*S).stacksize = STACK_INIT_SIZE;
    return OK;
}

Status DestroyStack(SqStack* S)
{
    if ((*S).base)
    {
        free((*S).base);
        (*S).base = NULL;
        (*S).top = NULL;
        (*S).stacksize = 0;
    }
    return OK;
}

Status ClearStack(SqStack* S)
{
    if ((*S).base)

```

```

        (*S).top = (*S).base;
    return OK;
}

Status StackEmpty(SqStack S)
{
    if (S.top == S.base)
        return TRUE;
    else
        return FALSE;
}

int StackLength(SqStack S)
{
    return S.top - S.base;
}

Status Push(SqStack* S, SElemType e)
{
    if ((*S).top - (*S).base >= (*S).stacksize)
    {
        //栈满，追加存储空间
        (*S).base = (SElemType*)realloc((*S).base, ((*S).stacksize + STACK_INIT_SIZE)
* sizeof(SElemType));
        if (!(*S).base)
            exit(OVERFLOW); //存储分配失败
        (*S).top = (*S).base + (*S).stacksize; //修改top
    }
    *(*S).top++ = e;
    return OK;
}

Status Pop(SqStack* S, SElemType* e)
{
    //若栈不空，则删除S的栈顶元素，用e返回其值，并返回OK；否则返回ERROR
    if ((*S).top == (*S).base)
        return ERROR;
    *e = *--(*S).top;
    return OK;
}

Status GetTop(SqStack S, SElemType* e)
{
    //若栈不空，则用e返回S的栈顶元素，并返回OK；否则返回ERROR

```

```

        if (S.top == S.base)
            return ERROR;
        *e = *(S.top - 1);
        return OK;
    }

Status visit(SElemType e)
{
    printf("%d ", e);
    return OK;
}

Status StackTravers(SqStack *S)
{
    if (!(*S).base)
        exit(OVERFLOW);
    if ((*S).top == (*S).base)
        printf("栈为空");
    SElemType *p = (*S).base;
    while (p < (*S).top)
    {
        visit(*p);
        p++;
    }
    printf("\n");
    return OK;
}

```

### (3)test.c

```

#include <stdio.h>
#include "Stack.h"

int main()
{

    SqStack S;
    //InitStack test
    if (InitStack(&S))
        printf("InitStack sucess!\n");
    else
        printf("InitStack unsucess!\n");

    //DestroyStack test
}

```

```

if (DestroyStack(&S))
    printf("DestroyStack sucess!\n");
else
    printf("DestroyStack unsucess!\n");

//ClearStack test
if (ClearStack(&S))
    printf("ClearStack sucess!\n");
else
    printf("ClearStack unsucess!\n");

//Push test
InitStack(&S);
for (int i = 1; i <= 5; i++)
{
    Push(&S, 100 + i);
}
printf("push后栈为: ");
StackTravers(&S);
printf("push后栈长度为:%d\n", StackLength(S));

//Pop test
SElemType* a = 0;
Pop(&S, &a);
printf("弹出的栈顶元素为:%d\n", a);
printf("pop后栈为: ");
StackTravers(&S);
printf("push后栈长度为:%d\n", StackLength(S));

//GetTop test
GetTop(S, &a);
printf("现在栈顶元素为:%d\n", a);
printf("gettop后栈为: ");
StackTravers(&S);
printf("push后栈长度为:%d\n", StackLength(S));

return 0;
}

```

```
Microsoft Visual Studio 调试控制台
InitStack sucess!
DestroyStack sucess!
ClearStack sucess!
push后栈为: 101 102 103 104 105
push后栈长度为:5
弹出的栈顶元素为:105
pop后栈为: 101 102 103 104
push后栈长度为:4
现在栈顶元素为:104
gettop后栈为: 101 102 103 104
push后栈长度为:4

E:\大二上\数据结构\代码保存\Stack\Debug\Stack.exe (进程 8900)已退出, 代码为 0。
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .
```

## 2. 栈的应用

### (1) 数制转换

#### Stack.h 和 Stack.c 文件同问题 1

##### test.c

```
#include <stdio.h>
#include "Stack.h"

void conversion(int N, int d);
void conversion(int N, int d)
{
    //对于输入的任意一个非负十进制整数, 打印输出与其等值的d进制数
    SElemType e=0;
    SqStack S;
    InitStack(&S);
    while (N)
    {
        Push(&S, N % d);
        N = N / d;
    }
    while (!StackEmpty(S))
    {
        int* e = 0;
        Pop(&S, &e);
        printf("%d", e);
    }
    printf("\n");
}

int main()
{
    int N = 0;
```

```

int d = 0;
printf("请输入一个非负十进制整数和想转化的进制: ");
scanf_s("%d%d", &N, &d);
printf("转化结果为: ");
conversion(N, d);
return 0;
}

```

选择Microsoft Visual Studio 调试控制台

请输入一个非负十进制整数和想转化的进制: 108 2  
转化结果为: 1101100

C:\Users\DELL\source\repos\Stackuse\Debug\Stackuse.exe (进程 21096) 已退出, 代码为 0。  
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。  
按任意键关闭此窗口. . .

Microsoft Visual Studio 调试控制台

请输入一个非负十进制整数和想转化的进制: 108 8  
转化结果为: 154

C:\Users\DELL\source\repos\Stackuse\Debug\Stackuse.exe (进程 12532) 已退出, 代码为 0。  
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。  
按任意键关闭此窗口. . .

## (2) 括号匹配的检验

### a.Stack.h

```

#ifndef STACK_H
#define STACK_H

#include <stdlib.h>
#include "Status.h"

#define STACK_INIT_SIZE 100//存储空间初始分配量
#define STACKINCREMENT 10//存储空间分配增量

typedef char SElemType;
typedef struct
{
    SElemType *base;//栈底指针
    SElemType* top;//栈顶指针
    int stacksize;//当前可使用最大容量, 不是栈中元素个数
}SqStack;

Status InitStack(SqStack* S);

Status StackEmpty(SqStack S);
Status Push(SqStack* S, SElemType e);
Status Pop(SqStack* S, SElemType* e);

```

```
#endif
```

### **b.Stack.c**

```
#include "Stack.h"
```

```
#include <stdio.h>
```

```
#include <malloc.h>
```

```
Status InitStack(SqStack* S)
```

```
{
    (*S).base = (SElemType*)malloc(STACK_INIT_SIZE * sizeof(SElemType));
    if(!(*S).base)
        exit(OVERFLOW);
    (*S).top = (*S).base;
    (*S).stacksize = STACK_INIT_SIZE;
    return OK;
}
```

```
Status StackEmpty(SqStack S)
```

```
{
    if (S.top == S.base)
        return TRUE;
    else
        return FALSE;
}
```

```
Status Push(SqStack* S, SElemType e)
```

```
{
    if ((*S).top - (*S).base >= (*S).stacksize)
    {
        //栈满，追加存储空间
        (*S).base = (SElemType*)realloc((*S).base, ((*S).stacksize + STACK_INIT_SIZE)
        * sizeof(SElemType));
        if (!(*S).base)
            exit(OVERFLOW); //存储分配失败
        (*S).top = (*S).base + (*S).stacksize; //修改top
    }
    *(*S).top++ = e;
    printf("入栈%c\n", e);
    return OK;
}
```

```
Status Pop(SqStack* S, SElemType* e)
```

```
{
```



```

//若栈不空，则删除S的栈顶元素，用e返回其值，并返回OK；否则返回ERROR
if ((*S).top == (*S).base)
    return ERROR;
*e = *--(*S).top;
printf("出栈%c\n", *e);
return OK;
}

```

### c.test.c

```

#include <stdio.h>
#include "Stack.h"

Status check();
Status check()
{
    char* ch = 0;
    SElemType e;
    SqStack S;
    InitStack(&S);
    while ((ch = getchar()) != '#')
    {
        if (ch == '(' || ch == '[' || ch == '{')
            Push(&S, ch);
        if (ch == ')')
        {
            if (StackEmpty(S))
                return FALSE;
            else
            {
                Pop(&S, &e);
                if (e != '(')
                    return FALSE;
            }
        }
    }
    if (ch == ']')
    {
        if (StackEmpty(S))
            return FALSE;
        else
        {
            Pop(&S, &e);
            if (e != '[')
                return FALSE;
        }
    }
}

```

```

    }
    if (ch == '}')
    {
        if (StackEmpty(S))
            return FALSE;
        else
        {
            Pop(&S, &e);
            if (e != '{')
                return FALSE;
        }
    }
}

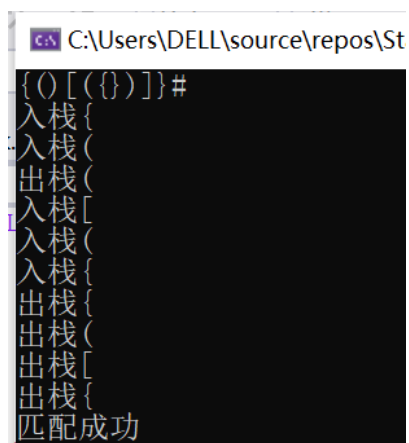
if (StackEmpty(S))
    return TRUE;
else
    return FALSE;
}

```

```

int main()
{
    if (check())
        printf("匹配成功");
    else
        printf("匹配失败");
    while (1);
    return 0;
}

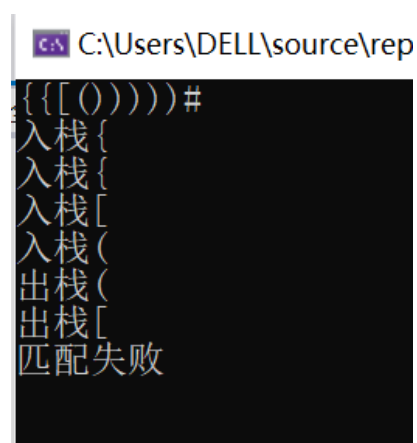
```



```

C:\Users\DELL\source\repos\St
{()[(())]}#
入栈{
入栈(
出栈(
入栈[
入栈(
入栈{
出栈{
出栈(
出栈[
出栈(
出栈{
匹配成功

```



```

C:\Users\DELL\source\rep
{{[(())]))#
入栈{
入栈{
入栈[
入栈(
出栈(
出栈[
匹配失败

```

### (3) 行编辑程序

#### a. Stack.h

```
#pragma once
```

```

#ifndef STACK_H
#define STACK_H

#include <stdlib.h>
#include "Status.h"

#define STACK_INIT_SIZE 100//存储空间初始分配量
#define STACKINCREMENT 10//存储空间分配增量

typedef char SElemType;
typedef struct
{
    SElemType* base;//栈底指针
    SElemType* top;//栈顶指针
    int stacksize;//当前可使用最大容量，不是栈中元素个数
}SqStack;

Status InitStack(SqStack* S);
Status DestroyStack(SqStack* S);
Status ClearStack(SqStack* S);
Status Push(SqStack* S, SElemType e);
Status Pop(SqStack* S, SElemType* e);
#endif

```

## b. Stack.c

```

#include "Stack.h"
#include <stdio.h>
#include <malloc.h>

Status InitStack(SqStack* S)
{
    (*S).base = (SElemType*)malloc(STACK_INIT_SIZE * sizeof(SElemType));
    if (!(*S).base)
        exit(OVERFLOW);
    (*S).top = (*S).base;
    (*S).stacksize = STACK_INIT_SIZE;
    return OK;
}

Status DestroyStack(SqStack* S)
{
    if ((*S).base)
    {
        free((*S).base);
        (*S).base = NULL;
    }
}

```

```

        (*S).top = NULL;
        (*S).stacksize = 0;
    }
    return OK;
}

Status ClearStack(SqStack* S)
{
    if ((*S).base)
        (*S).top = (*S).base;
    return OK;
}

Status Push(SqStack* S, SElemType e)
{
    if ((*S).top - (*S).base >= (*S).stacksize)
    {
        //栈满，追加存储空间
        (*S).base = (SElemType*)realloc((*S).base, ((*S).stacksize + STACK_INIT_SIZE)
* sizeof(SElemType));
        if (!(*S).base)
            exit(OVERFLOW); //存储分配失败
        (*S).top = (*S).base + (*S).stacksize; //修改top
    }
    *(*S).top++ = e;
    return OK;
}

Status Pop(SqStack* S, SElemType* e)
{
    //若栈不空，则删除S的栈顶元素，用e返回其值，并返回OK；否则返回ERROR
    if ((*S).top == (*S).base)
        return ERROR;
    *e = *(--(*S).top);
    return OK;
}

```

### c. test.c

```

#include <stdio.h>
#include "Stack.h"

void LineEdit();
void LineEdit()
{

```

```

//利用字符栈S，从终端接受一行并传送至调用过程的数据区
char ch =0;
char* temp=0;
SqStack S;
InitStack(&S); //构造空栈S
printf("请输入一行（#：空格；@：清行）：\n");
ch = getchar();
while (ch != EOF)
{
    //EOF全文结束符WIN: Ctrl+Z;MAC:Ctrl+D
    printf("输出结果为：\n");
    while (ch != EOF && ch != '\n')
    {
        switch (ch)
        {
            case '#':
                Pop(&S, &ch);
                break; //仅当栈非空时退栈
            case '@':
                ClearStack(&S);
                break; //重置S为空栈
            default:
                Push(&S, ch);
                break; //有效字符进栈，未考虑栈满情形
        }
        ch = getchar(); //从终端接收下一个字符
    }
    temp = S.base;
    while (temp != S.top)
    {
        printf("%c", *temp);
        ++temp; //将从栈底到栈顶的栈内字符传送至调用过程的数据区
    }
    ClearStack(&S); //重置S为空栈
    printf("\n");
    if (ch != EOF)
    {
        printf("请输入一行（#：退格；@：清行）：\n");
        ch = getchar();
    }
}
DestroyStack(&S);
}

```

```
Microsoft Visual Studio 调试控制台

请输入一行 (#: 空格; @: 清行):
while##ilr#e(s#*s)
输出结果为:
while(*s)
请输入一行 (#: 退格; @: 清行):
outchar@putchar(*s=##++);
输出结果为:
putchar(*s++);
请输入一行 (#: 退格; @: 清行):
'Z

E:\大二上\数据结构\代码保存\StackUse2\Debug\StackUse2.exe (进程 9008) 已退出, 代码为 0。
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .
```

```
#ifndef STACK_H
#define STACK_H

#include <stdlib.h>
#include "Status.h"

#define STACK_INIT_SIZE 100//存储空间初始分配量
#define STACKINCREMENT 10//存储空间分配增量

typedef char SElemType;
typedef float SElemType2;
typedef int Status;

typedef struct {
    SElemType* base;//在构造栈前和销毁栈后,base的值为NULL
    SElemType* top;//栈顶指针
    int stacksize;//当前分配的元素空间以元素为单位
} SqStack_OPTR;

typedef struct {
    SElemType2* base;//在构造栈前和销毁栈后,base的值为NULL
    SElemType2* top;//栈顶指针
    int stacksize;//当前分配的元素空间以元素为单位
} SqStack_OPND;
```

```

Status InitStack_OPTR(SqStack_OPTR* S);
Status InitStack_OPND(SqStack_OPND* S);
Status Push_OPTR(SqStack_OPTR* S, SElemType1 e);
Status Push_OPND(SqStack_OPND* S, SElemType2 e);
Status Pop_OPTR(SqStack_OPTR* S, SElemType1* e);
Status Pop_OPND(SqStack_OPND* S, SElemType2* e);
Status GetTop_OPTR(SqStack_OPTR S, SElemType1* e);
Status GetTop_OPND(SqStack_OPND S, SElemType2* e);
Status In(char e, char OP[]);
Status Precede(char m, char n);
Status Operate(SElemType2 a, char theta, SElemType2 b);

#endif

```

## b. Stack.c

```

#include "Stack.h"
#include <stdio.h>
#include <malloc.h>

Status InitStack_OPTR(SqStack_OPTR* S)
{
    (*S).base = (SElemType1*)malloc(STACK_INIT_SIZE * sizeof(SElemType1));
    if (!(*S).base)
        exit(OVERFLOW);
    (*S).top = (*S).base;
    (*S).stacksize = STACK_INIT_SIZE;
    return OK;
}

Status InitStack_OPND(SqStack_OPND* S)
{
    (*S).base = (SElemType2*)malloc(STACK_INIT_SIZE * sizeof(SElemType2));
    if (!(*S).base)
        exit(OVERFLOW);
    (*S).top = (*S).base;
    (*S).stacksize = STACK_INIT_SIZE;
    return OK;
}

Status Push_OPTR(SqStack_OPTR* S, SElemType1 e)
{
    if ((*S).top - (*S).base >= (*S).stacksize)
    {
        //栈满，追加存储空间
    }
}

```

```

        (*S).base = (SElemType1*)realloc((*S).base, ((*S).stacksize + STACK_INIT_SIZE)
* sizeof(SElemType1));
        if (!(*S).base)
            exit(OVERFLOW); //存储分配失败
        (*S).top = (*S).base + (*S).stacksize; //修改top
    }
    *(*S).top++ = e;
    return OK;
}

```

```

Status Push_OPND(SqStack_OPND* S, SElemType2 e)
{
    if ((*S).top - (*S).base >= (*S).stacksize)
    {
        //栈满，追加存储空间
        (*S).base = (SElemType2*)realloc((*S).base, ((*S).stacksize + STACK_INIT_SIZE)
* sizeof(SElemType2));
        if (!(*S).base)
            exit(OVERFLOW); //存储分配失败
        (*S).top = (*S).base + (*S).stacksize; //修改top
    }
    *(*S).top++ = e;
    return OK;
}

```

```

Status Pop_OPTR(SqStack_OPTR* S, SElemType1* e)
{
    //若栈不空，则删除S的栈顶元素，用e返回其值，并返回OK；否则返回ERROR
    if ((*S).top == (*S).base)
        return ERROR;
    *e = *((--(*S).top));
    return OK;
}

```

```

Status Pop_OPND(SqStack_OPND* S, SElemType2* e)
{
    //若栈不空，则删除S的栈顶元素，用e返回其值，并返回OK；否则返回ERROR
    if ((*S).top == (*S).base)
        return ERROR;
    *e = *((--(*S).top));
    return OK;
}

```

```

Status GetTop_OPTR(SqStack_OPTR S, SElemType1* e)

```



```

{
    //若栈不空，则用e返回S的栈顶元素，并返回OK；否则返回ERROR
    if (S.top == S.base)
        return ERROR;
    *e = *(S.top - 1);
    return OK;
}

```

```

Status GetTop_OPND(SqStack_OPND S, SElemType2* e)
{
    //若栈不空，则用e返回S的栈顶元素，并返回OK；否则返回ERROR
    if (S.top == S.base)
        return ERROR;
    *e = *(S.top - 1);
    return OK;
}

```

### c. test.c

```

#include <stdio.h>
#include "Stack.h"

char OP[7] = { '+', '-', '*', '/', '(', ')', '#' };
char SymbolPriority[8][8] =
{
    '+', '+', '-', '-', '*', '*', '/', '(', ')', '#',
    '+', '>', '>', '<', '<', '<', '>', '>',
    '-', '>', '>', '<', '<', '<', '>', '>',
    '*', '>', '>', '>', '>', '<', '>', '>',
    '/', '>', '>', '>', '>', '<', '>', '>',
    '(', '<', '<', '<', '<', '<', '=', ' ',
    ')', '>', '>', '>', '>', ' ', '>', '>',
    '#', '<', '<', '<', '<', '<', ' ', '='
};

Status In(char e, char OP[])
{
    int flag = 0;
    for (int i = 0; i < 7; i++)
    {
        if (e == OP[i]) flag = 1;
    }
    if (flag) return 1;
    else return 0;
}

```

Status Precede(char m, char n)

```
{  
    int mdata = 0;  
    int ndata = 0;  
    switch (m)  
    {  
        case '+':  
            mdata = 1;  
            break;  
        case '-':  
            mdata = 2;  
            break;  
        case '*':  
            mdata = 3;  
            break;  
        case '/':  
            mdata = 4;  
            break;  
        case '(':  
            mdata = 5;  
            break;  
        case ')':  
            mdata = 6;  
            break;  
        case '#':  
            mdata = 7;  
            break;  
    }  
    switch (n)  
    {  
        case '+':  
            ndata = 1;  
            break;  
        case '-':  
            ndata = 2;  
            break;  
        case '*':  
            ndata = 3;  
            break;  
        case '/':  
            ndata = 4;  
            break;  
        case '(':  
            ndata = 5;  
    }
```

```

        break;
    case ')':
        ndata = 6;
        break;
    case '#':
        ndata = 7;
        break;
}
return SymbolPriority[mdata][ndata];
}

```

```

Status Operate(SElemType2 a, char theta, SElemType2 b)
{
    SElemType2 result = 0;
    switch (theta)
    {
        case '+':
            result = a + b;
            break;
        case '-':
            result = a - b;
            break;
        case '*':
            result = a * b;
            break;
        case '/':
            result = a / b;
            break;
    }
    return result;
}

```

Status EvaluateExpression();

Status EvaluateExpression()

```

{
    //算术表达式求值的算符优先算法，设OPTR和OPND分别为运算符栈和操作数栈。如输入4+2*3-
    9/3#
    SqStack_OPTR OPTR; //寄存运算符
    SqStack_OPND OPND; //寄存操作符

    SElemType1 e = 0; //接收GetTop和Pop
    SElemType1 f = 0;
    SElemType1 x = 0; //接收退栈后的值
}

```

```

SElemType1 theta = 0; //在Precede函数判出>后,存放OPTR出栈的运算符
SElemType2 a = 0; //操作符左值
SElemType2 b = 0; //操作符右值
SElemType2 d = 0;

InitStack_OPTR(&OPTR); //OPTR元素类型char
Push_OPTR(&OPTR, '#');
InitStack_OPND(&OPND); //OPND元素类型float
char c = getchar();

GetTop_OPTR(OPTR, &e);
while (c != '#' || e != '#')
{
    if (!In(c, OP)) //不是运算符则进栈
    {
        Push_OPND(&OPND, c-48);
        c = getchar();
    }
    else
    {
        GetTop_OPTR(OPTR, &f);
        switch (Precede(f, c)) //优先级比较
        {
            case '<': //栈顶元素优先级低
                Push_OPTR(&OPTR, c);
                c = getchar();
                break;
            case '=': //脱括号并接收下一字符
                Pop_OPTR(&OPTR, &x);
                c = getchar();
                break;
            case '>':
                Pop_OPTR(&OPTR, &theta);
                Pop_OPND(&OPND, &b);
                Pop_OPND(&OPND, &a);
                Push_OPND(&OPND, Operate(a, theta, b));
                break;
        } //switch语句结束
    }
    GetTop_OPTR(OPTR, &e);
} //while语句结束
GetTop_OPND(OPND, &d);
return d;
}

```

```

int main()
{
    printf("输入表达式, 得出运算结果, 输入以#结束\n");
    printf("表达式结果为:%d\n", EvaluateExpression());
    return 0;
}

```

选择Microsoft Visual Studio 调试控制台

输入表达式, 得出运算结果, 输入以#结束  
4+2\*3-9/3#  
表达式结果为:7

E:\大二上\数据结构\代码保存\StackUse3\Debug\StackUse3.exe (进程 21308)已退出, 代码为 0。  
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。  
按任意键关闭此窗口. . .

Microsoft Visual Studio 调试控制台

输入表达式, 得出运算结果, 输入以#结束  
3+(4+5)\*2-1#  
表达式结果为:20

E:\大二上\数据结构\代码保存\StackUse3\Debug\StackUse3.exe (进程 22444)已退出, 代码为 0。  
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。  
按任意键关闭此窗口. . .

### 3. 递归

```
#include <stdio.h>
```

```

void move(char x, int n, char y)
{
    printf("%d号圆盘, %c-->%c\n", n, x, y);
}

```

```

void hanoi(int n, char x, char y, char z)
{
    if (n == 1)
        move(x, 1, z); //将编号1的圆盘从x移到z
    else
    {
        hanoi(n - 1, x, z, y); //将x上编号为1至n-1的圆盘移到y, z作辅助塔
        move(x, n, z); //将编号为n的圆盘从x移到z
        hanoi(n - 1, y, x, z); //将y上编号为1至n-1的圆盘移到z, x作辅助塔
    }
}

```

```

int main()
{
    int n;
    printf("请输入汉诺塔的层数: ");
    scanf_s("%d", &n);
}

```

```

printf("移动步骤如下: \n");
hanoi(n, 'x', 'y', 'z');

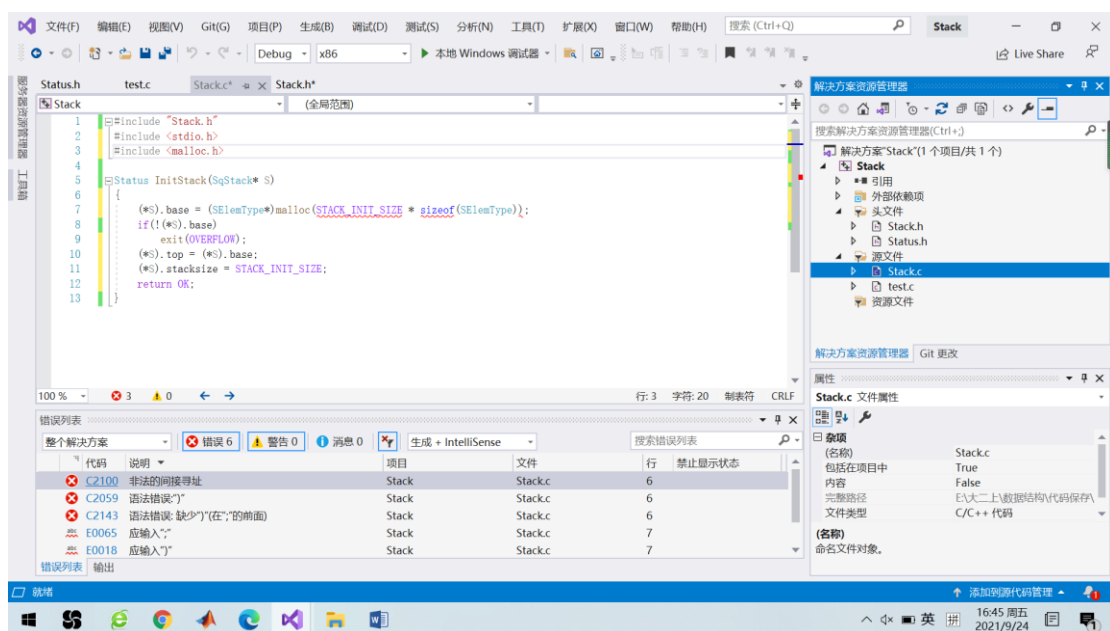
return 0;
}

```



## 五. 实验总结

### 1. 报错



#define STACK\_INIT\_SIZE 100//存储空间初始分配量

#define STACKINCREMENT 10//存储空间分配增量

发现多打分号导致

### 2. 写完遍历函数发现忘记将 base 指针返回首地址, 导致接下来进行操作时, 栈为空 错误代码

Status StackTravers(SqStack \*S)

```

{
    if (!(*S).base)
        exit(OVERFLOW);
    if ((*S).top == (*S).base)
        printf("栈为空");
    while ((*S).top > (*S).base)
    {
        visit((*S).base++);
    }
    printf("\n");
    return OK;
}

```

### 修改后

```

Status StackTravers(SqStack *S)
{
    if (!(*S).base)
        exit(OVERFLOW);
    if ((*S).top == (*S).base)
        printf("栈为空");
    SElemType *p = (*S).base;
    while (p < (*S).top)
    {
        visit(*p);
        p++;
    }
    printf("\n");
    return OK;
}

```

3. 出现错误 0x7A0628BC (ucrtbased.dll)处(位于 Stackuse.exe 中)引发的异常:  
 0xC0000005: 读取位置 0x0000007B 时发生访问冲突。  
 检查代码后发现, 整形被误操作, 以%s的形式打印。

### 错误代码

```

Status Push(SqStack* S, SElemType e)
{
    if ((*S).top - (*S).base >= (*S).stacksize)
    {
        //栈满, 追加存储空间
        (*S).base = (SElemType*)realloc((*S).base, ((*S).stacksize + STACK_INIT_SIZE)
        * sizeof(SElemType));
        if (!(*S).base)
            exit(OVERFLOW); //存储分配失败
        (*S).top = (*S).base + (*S).stacksize; //修改top
    }
}

```

```

    }
    *(*S).top++ = e;
    printf("入栈%s\n", e);
    return OK;
}

```

### 修改后

```

Status Push(SqStack* S, SElemType e)
{
    if ((*S).top - (*S).base >= (*S).stacksize)
    {
        //栈满，追加存储空间
        (*S).base = (SElemType*)realloc((*S).base, ((*S).stacksize + STACK_INIT_SIZE)
        * sizeof(SElemType));
        if (!(*S).base)
            exit(OVERFLOW); //存储分配失败
        (*S).top = (*S).base + (*S).stacksize; //修改top
    }
    *(*S).top++ = e;
    printf("入栈%c\n", e);
    return OK;
}

```

#### 4. 报错：getchar 重定义，不同的基类型

getchar 函数系统中已存在，再次定义发生报错

#### 5. 在编写表达式求解这一问题中，起初输入字符串后，无法得到结果

经过断点调试后发现 GetTop\_OPTR(OPTR, &e) 没有包含在 while 循环中，于是在循环末尾添加。

### 修改后代码如下：

```

GetTop_OPTR(OPTR, &e);
while (c != '#' || e != '#')
{
    if (!In(c, OP)) //不是运算符则进栈
    {
        Push_OPND(&OPND, c-48);
        c = getchar();
    }
    else
    {
        GetTop_OPTR(OPTR, &f);
        switch (Precede(f, c)) //优先级比较
        {

```



```

    case '<': //栈顶元素优先级低
        Push_OPTR(&OPTR, c);
        c = getchar();
        break;
    case '=': //脱括号并接收下一字符
        Pop_OPTR(&OPTR, &x);
        c = getchar();
        break;
    case '>':
        Pop_OPTR(&OPTR, &theta);
        Pop_OPND(&OPND, &b);
        Pop_OPND(&OPND, &a);
        Push_OPND(&OPND, Operate(a, theta, b));
        break;
} //switch语句结束
}
GetTop_OPTR(OPTR, &e); //添加
} //while语句结束

```