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 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;

Status InitStack(SqStack* S);
Status Push(SqStack* S, SElemType e);
Status Pop(SqStack* S, SElemType* e);
Status GetTop(SqStack S, SElemType* e);
Status In(char e, char OP[]);
Status Precede(char m, char n);
Status Operate(SElemType a, char theta, SElemType b);


#endif
```

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 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;
}
```

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(SElemType a, char theta, SElemType b)
{
    SElemType 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;//寄存运算符
    SqStack OPND;//寄存操作符
    char e = 0;//接收GetTop和Pop
    char f = 0;
    char x = 0;//接收退栈后的值
    char theta = 0;//在Precede函数判出>后,存放OPTR出栈的运算符
    float a = 0;//操作符左值
    float b = 0;//操作符右值
    int d = 0;

    InitStack(&OPTR);//OPTR元素类型char
    Push(&OPTR, '#');
    InitStack(&OPND);//OPND元素类型float
    char c = getchar();


    if (c >= 48 && c <= 57)
        c -= 48;
    GetTop(OPTR, &e);
    while (c != '#' ||e != '#')
    {
        if (!In(c,OP))//不是运算符则进栈
        {
            if (c >= 48 && c <= 57)
```

```
                c -= 48;
            Push(&OPND, c);
            c = getchar();
            if (c >= 48 && c <= 57)
                c -= 48;
        }
        else
            GetTop(OPTR, &f);
            switch (Precede(f, c))//优先级比较
            {
            case '<'://栈顶元素优先级低
                Push(&OPTR, c);
                c = getchar();
                if (c >= 48 && c <= 57)
                    c -= 48;
                break;
            case '='://脱括号并接收下一字符
                Pop(&OPTR, &x);
                c = getchar();
                if (c >= 48 && c <= 57)
                    c -= 48;
                break;
            case '>':
                Pop(&OPTR, &theta);
                Pop(&OPND, &a);
                Pop(&OPND, &b);
                Push(&OPND, Operate(a, theta, b));
                break;
            }//switch语句结束
    }//while语句结束
    GetTop(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;
}
```