选择的题目：T4中缀表达式进行表达式求值（中等难度）

功能描述：(一级标题)

本程序可对输入的中序表达式求值。

具体功能可分为：

1. 读入表达式
2. 判断表达式是否有误
3. 如无误，则将表达式转化为后缀表达式
4. 对后缀表达式进行求值

问题分析与算法设计思路：（一级标题）

问题分析：（二级从属标题）

该问题可分为三部分：

1. 对读入的中缀表达式判断是否有误
2. 将中缀表达式转化为后缀表达式
3. 对后缀表达式进行求值运算

对应的算法设计思路：（二级从属标题）

判断读入的中缀表达式是否有误：（三级从属标题）

表达式的错误可分为四种：

1. 格式与题目给定不符合：未按照"#…………（数学表达式）#"的格式进行输入
2. 缺失操作数：两运算符之间没有数字，操作数缺失：出现了"6+-3","2\*/4"，"2/-4"等的情况
3. 非法算符：出现了除数字（0～9），运算符（‘+’‘—’‘—’‘\*’‘/’‘（’‘）’）以外的字符
4. 括号不配匹配：左括号或右括号多余

本题本别设计种算法，分别对对应的错误进行判断：

1.判断输入字符串的首尾元素是否为‘#’：

同时取读入字符串的首尾元素，若不通时等于‘#’则不符合输入规范

2.缺失运算数：设置int lose(char\*a)函数：

设置标志变量key=0，并从字符串首开始逐字遍历字符串：

若为运算符则key++；

若为数字则key=0；

若key超出1则发生运算数缺失

3.非法算符：设置illegal(char\*a)函数：

逐字遍历字符串，若发现非法字符则含有非法字符

4.括号不匹配：设置kuohao（char\*a）函数：

设置标志变量key=0，并从字符串首开始逐字遍历字符串：

若为'('则key++；

若为')'则key--；

若key<0则左括号缺失

遍历完后：

若key！=0则右括号缺失

将中缀表达式转化为后缀表达式（三级从属标题）

逐字遍历字符串，服从以下规则：

1、字符为 运算数 ：

直接送入后缀表达式（注：需要先分析出完整的运算数）。

2、字符为 左括号 ：

直接入栈（注：左括号入栈后优先级降至最低）。

3、字符为 右括号 ：

直接出栈，并将出栈字符依次送入后缀表达式，直到栈顶字符为左括号（左括号也要出栈，但不送入后缀表达式）。

总结：只要满足 栈顶为左括号 即可进行最后一次出栈。

4、字符为 操作符 ：

若栈空，直接入栈。

若栈非空，判断栈顶操作符，若栈顶操作符优先级低于该操作符，该操作符入栈；否则一直出栈，并将出栈字符依次送入后缀表达式，直到栈空或栈顶操作符优先级低于该操作符，该操作符再入栈。

对后缀表达式进行求值运算（三级从属标题）

从左至右依次遍历后缀表达式各个字符，服从以下规则：

1、字符为 运算数 ：

直接入栈（注：需要先分析出完整的运算数并将其转换为对应的数据类型）

2、字符为 操作符 ：

连续出栈两次，使用出栈的两个数据进行相应计算，并将计算结果入栈

3、重复以上步骤直至遍历完成后缀表达式，最后栈中的数据就是中缀表达式的计算结果。

说明本程序使用的数据结构：（一级标题）

数字栈：

定义两个整数指针top，base分别指向栈顶元素与栈底元素的后一元素

当出栈时：top后移

当入栈时：top前移

当top == base 时，栈为空

typedef struct {

int \*base; *//用于栈存储的基地址*

int \*top; *//指向该基地址的栈顶指针*

int stackSize; *//栈的大小*

}SqStackInt;

运算符栈：

定义两个字符指针top，base分别指向栈顶元素与栈底元素的后一元素

当出栈时：top后移

当入栈时：top前移

当top == base 时，栈为空

typedef struct {

char \*base; *//用于栈存储的基地址*

char \*top; *//指向该基地址的栈顶指针*

int stackSize; *//栈的大小*

}SqStackChar;

下面，对程序中使用的主要函数功能进行说明：（一级标题）

int InitStack\_Int(SqStackInt &S)

建立空栈（for数字）

int InitStack\_Char(SqStackChar &S)

建立空栈（for运算符）

int Push\_Int(SqStackInt &S,int e)

数字入数字栈

int Push\_Char(SqStackChar &S,char e)

运算符入运算符栈

int Pop\_Int(SqStackInt &S,int &e)

数字栈弹出首元素

int Pop\_Char(SqStackChar &S,char &e)

字符栈弹出首元素

int StackEmpty\_Int(SqStackInt S)

判断数字栈是否为空

int StackEmpty\_Char(SqStackChar S)

判断运算符栈是否为空

int ClearStack\_Int(SqStackInt S)

清空数字栈剩余的所有元素

int ClearStack\_Char(SqStackChar S)

清空运算符栈中的所有元素

int DestroyStack\_Int(SqStackInt &S)

运行结束后将数字栈销毁

int DestroyStack\_Char(SqStackChar &S)

运行结束后将运算符栈销毁

int isOper(char c)

判断该字符是否为运算符（还是数字）

char getStackTopPriority(char StackTop,char c)

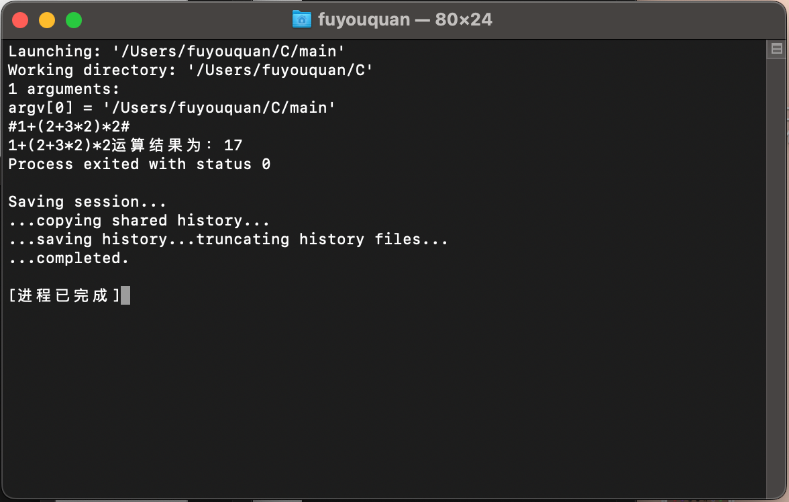
判断运算符栈顶元素与当前运算符的优先级大小

int operate(int a,char oper,int b)

根据得到的运算符与两数字，进行计算

运行截图：（一级标题）

正确运行截图：（含有括号，乘除，加减等操作）



格式错误响应截图：

1. 输入格式错误：

2.括号不匹配（左括号多余）：

3.括号不匹配（右括号多余）：

4.运算数缺失：

5.非法算符：

说明所采用的存储结构的优缺点，及采用该存储结构的理由：（一级标题）

本题主要采用了栈的存储结构，利用了栈“先入后出，后入先出”的存储性质。

由于本题主要采用了两大操作步骤：  
  
1.中缀表达式转后缀表达式

1. 后缀表达式的求值运算

在中缀表达式转后缀中，需要先将运算符存储，当遇到其他运算符时，根据运算符的优先级以“先入后出”的存取模式进行提取。因此栈的存储结构最符合该步骤要求。

在后缀表达式的求值运算中，需要根据表达式按序进行运算。当遇到运算符时，需要将距离运算符最近的两个数进行运算。栈的“先入后出，后入先出”的性质符合该操作的需求。

实验心得体会: （一级标题）

本程序解决了中缀表达式求值的问题。在编写程序中，多次运用了栈的相关操作，提高了对栈的存储模式，栈的性质的理解。熟练了出栈，入栈等相关操作。

在设计该算法时，遇到了操作符与数字字符性质不同，而均需以栈的模式进行存取的问题。为此本程序分别为操作符栈，数字字符栈设置对应栈结构与栈运算函数，分别对数字字符，运算符进行独立操作，互不干扰。

###注：中缀表达式进行表达式求值题目编程环境为: VSCode on MacOS，由于系统限制，使用了sys/malloc.h 包，如在Windows系统运行请将头文件中：  
#include<sys/malloc.h>

替换为：

#include<malloc.h>

并在程序中注意其他系统差异。

###（这儿确实不太好改，dev c++调试很困难，windows没装vscode，哎——，就这样写上吧）###