# 数据结构实验报告——实验五

## 学号： 20201060330 姓名： 胡诚皓 得分：\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### 一、实验目的

1. 复习栈的逻辑结构、存储结构及基本操作；
2. 掌握顺序栈、链栈。

### 二、实验内容

1. （课堂完成）顺序栈的基本实现

假设栈中数据元素类型是字符型，请采用顺序栈实现栈的以下基本操作：

1. Status InitStack (&S)

//构造空栈S；

1. Status Push(&S, e)

//元素e入栈S；

1. Status Pop(&S, &e)

//栈S出栈，元素为e。

1. （必做题）括号匹配

请实现：对于一个可能包括括号{}、[]、()的表达式，判定其中括号是否匹配。

1. （选做题）算术表达式的计算

请实现：

采用算符优先分析法分析输入的算术表达式语法是否正确，若表达式语法正确，请输出运算结果；否则输出提示“表达式错误！”。

### 三、数据结构及算法描述

1. （课堂完成）顺序栈的基本实现
2. 数据结构

使用结构体SqStack存储和表示顺序栈，其中的base始终指向顺序栈的栈底，top始终指向顺序栈栈顶的下一个位置，即当有新元素需要插入时，直接放在top的位置，再将top后移一个位置。

通过宏定义定义了栈的初始容量为100，当栈的容量不足时，每次扩充的容量为10。stacksize表示当前顺序栈的容量（而不是栈中实际有的元素个数），求取当前顺序栈元素个数可以使用top-base的指针运算实现。

1. 算法描述

Status InitStack(SqStack \*S)

* 1. 初始化STACK\_INT\_SIZE个SElemType类型元素的空间，并把起始位置赋给S->base作为栈底；
  2. 将S的容量stacksize设为STACK\_INT\_SIZE（即100）；
  3. 由于此时栈为空，栈顶与栈底相同，即把S->top指向栈底S->base的位置。

Status Push(SqStack \*S, SElemType e)

* 1. 通过S->top - S->base计算目前栈中元素的个数length；
  2. 若length + 1 == S->stacksize，即把当前要放入的元素e放入后栈就满了，此时将栈扩充STACKINCREMENT个位置；
  3. 将新元素e放在S->top所指的位置上，再把S->top移向下一个位置。

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

* 1. 判断栈是否为空，若栈为空，无法弹出，直接返回ERROR；
  2. 将弹出的元素赋值到e所指的位置；
  3. 将S->top前移一个位置。

1. （必做题）括号匹配
   1. 数据结构

仍使用与第1题相同的结构体来存储和表示顺序栈。

* 1. 算法描述

多了一个判断栈是否为空的辅助函数Status Empty(SqStack S)，若作为参数的栈为空则返回OK，若不为空则返回ERROR。

另外还有Status leftJudge(char ch)、Status rightJudge(char ch)、Status match(char ch1, char ch2)分别用于判断ch是否为左括号、ch是否为右括号、ch1与ch2是否为匹配的左右括号，若是则返回OK，若不是则返回ERROR。

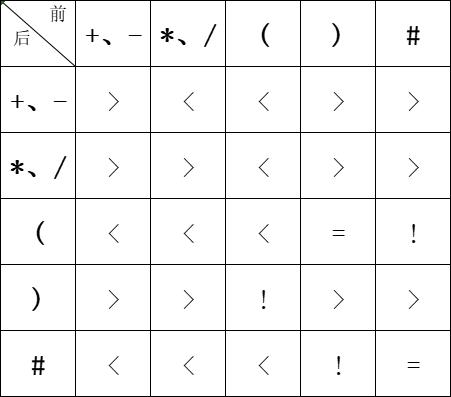
* + 1. 声明、初始化变量，并调用InitStack初始化顺序栈；
    2. 读取一整行输入作为一个表达式
    3. 不断处理当前的一个字符inputString[p]（若inputString[p]为空字符，则转到④）若为左括号，直接入栈，继续测试下一个字符；若为右括号，弹出栈顶的元素尝试与之匹配，若不匹配则输出错误信息并直接退出程序，若匹配则继续测试下一个字符；若不是括号，直接继续测试下一个字符（由于只需要知道括号是否匹配，无需关心其他字符）。
    4. 若栈为空，说明所有的括号都匹配到了，输出“All brackets matched.”；若栈不为空，说明有左括号没有对应的右括号与之匹配，输出“Fail to match all brackets.”

1. （选做题）算术表达式的计算

（1）数据结构

两个与第1题中相似的结构体用来存储和表示顺序栈，charStack用于存放char类型的操作符，intStack用于存放int类型的操作数。char类型的二维数组rules用于存储各个运算符之间的优先级比较，需要注意的是，运算符在前在后对其优先级是有影响的。下标0代表'+'、'-'，1代表'\*'、'/'，2代表'('，3代表')'，4代表开始/结束符'#'，例如+与(的优先级比较为rules[0][2]，值为'<'，即+的优先级低于(。

优先级表如下：



（2）算法描述

设计了辅助函数char frontCharStack(charStack S)、int cal(int a, int b, char op)、Status isNum(char ch)、char compareOp(char firstOp, char secondOp)分别用于读取charStack的栈顶元素、计算a op b的值、判断ch是否为数字（即处于'0'~'9'之间）、比较firstOp与secondOp的优先级（返回'<'表示secondOp优先级较高；返回'>'表示firstOp优先级较高；返回'='表示优先级相当，用于匹配消去括号；返回'!'表示有语法错误）

* + 1. 初始化局部变量，初始化操作符栈opEr、操作数栈opNum；
    2. 读入一行字符串作为输入的表达式，并在输入的表达式末尾添加一个'#'作为结束符以便后续算法中进行判断，在操作符栈中先压入'#'，作为开始符；
    3. 判断当前处理的字符inputString[p]是否为'#'，即是否读取到了输入表达式的末尾；判断操作符栈的栈顶是否为'#'，即是否将所有操作符运算完毕。若这两项都处理完毕，说明整个表达式计算完毕。
    4. 若inputString[p]为数字，则使用tmp作为临时变量读取这一个操作数，读取完成后入opNum操作数栈；

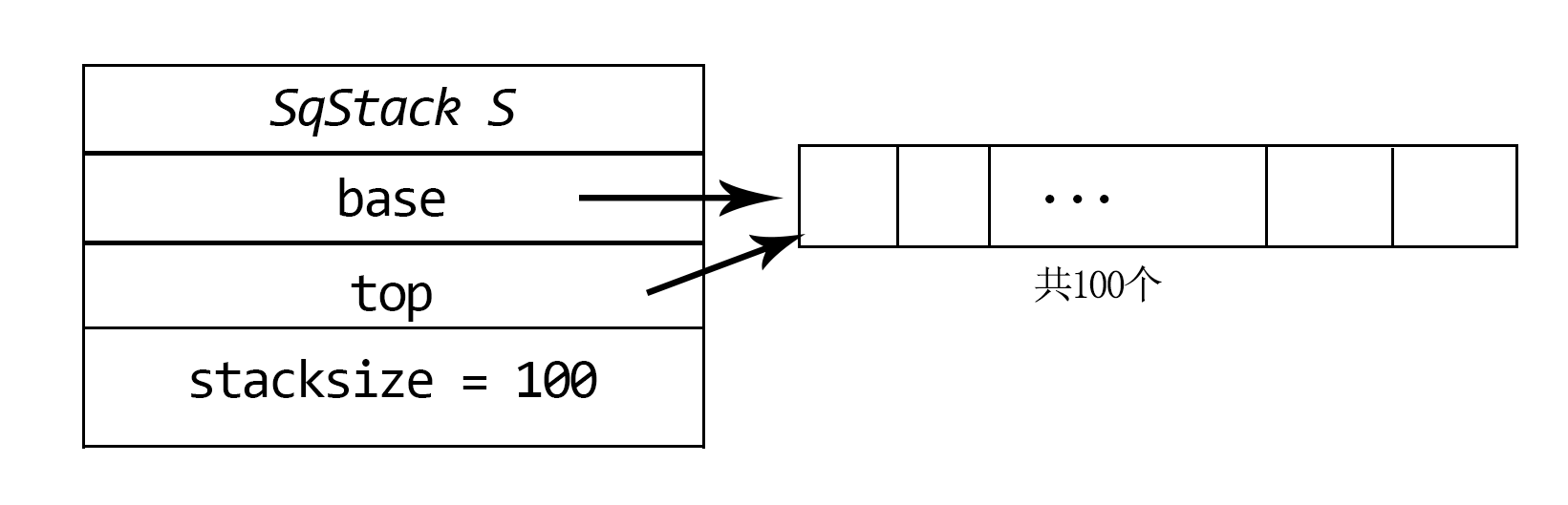
1. 若inputString[p]不为数字，调用compareOp将操作符栈栈顶的操作符与inputString[p]进行优先级比较：若返回'<'，说明inputString[p]优先级较高，直接将inputString[p]加入操作符栈，p往后移；若返回'='，说明优先级相同，说明inputString[p]作为右括号碰到了左括号，弹出操作符栈栈顶的左括号进行消去操作，p往后移；若返回'>'，说明inputString[p]优先级较低，从操作数栈中弹出两个操作数进行计算，将计算出的结果再压入操作数栈中，若操作数栈中的操作数只有一个不足以进行计算，说明表达式中有语法错误，输出错误信息并直接退出程序；若返回'!'，说明有无效的字符或无效的表达式格式，输出错误信息并直接退出程序。返回③
   * 1. 操作数栈中必然只剩下一个数字，即表达式最终的运算结果，输出即可。

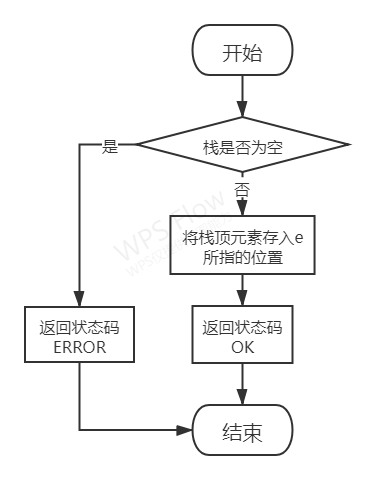
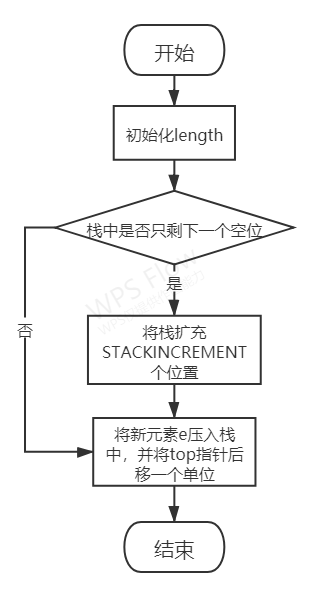
### 详细设计

1. （课堂完成）顺序栈的基本实现

题5-1

InitStack初始化的顺序栈





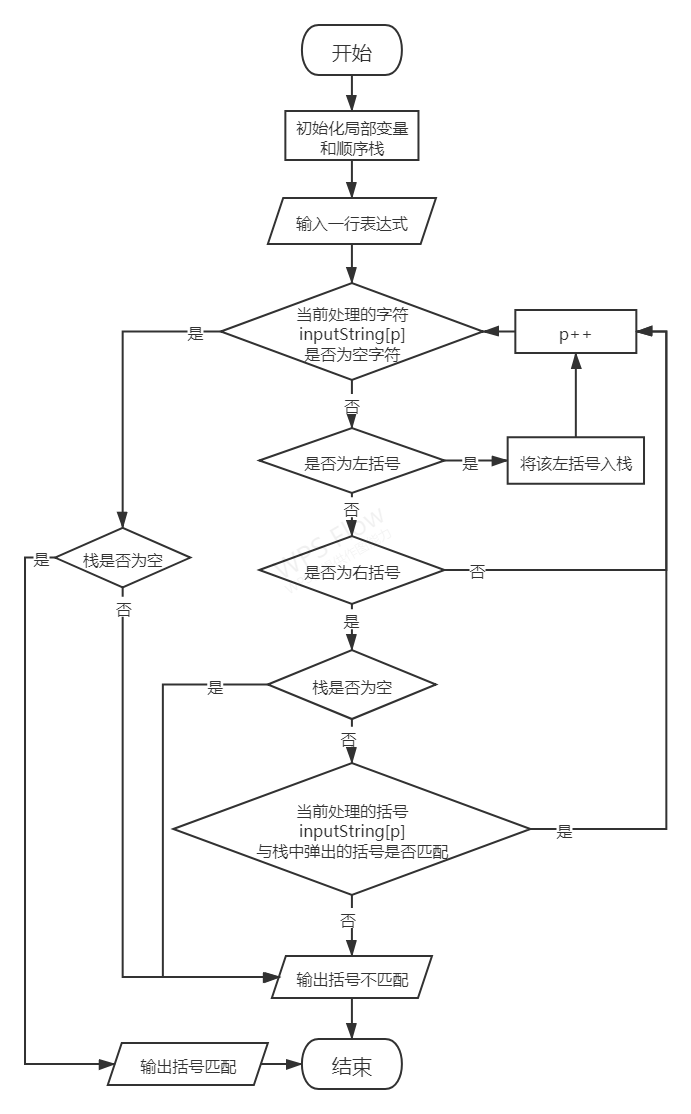
题5-1

Pop函数示意图

题5-1

Push函数示意图

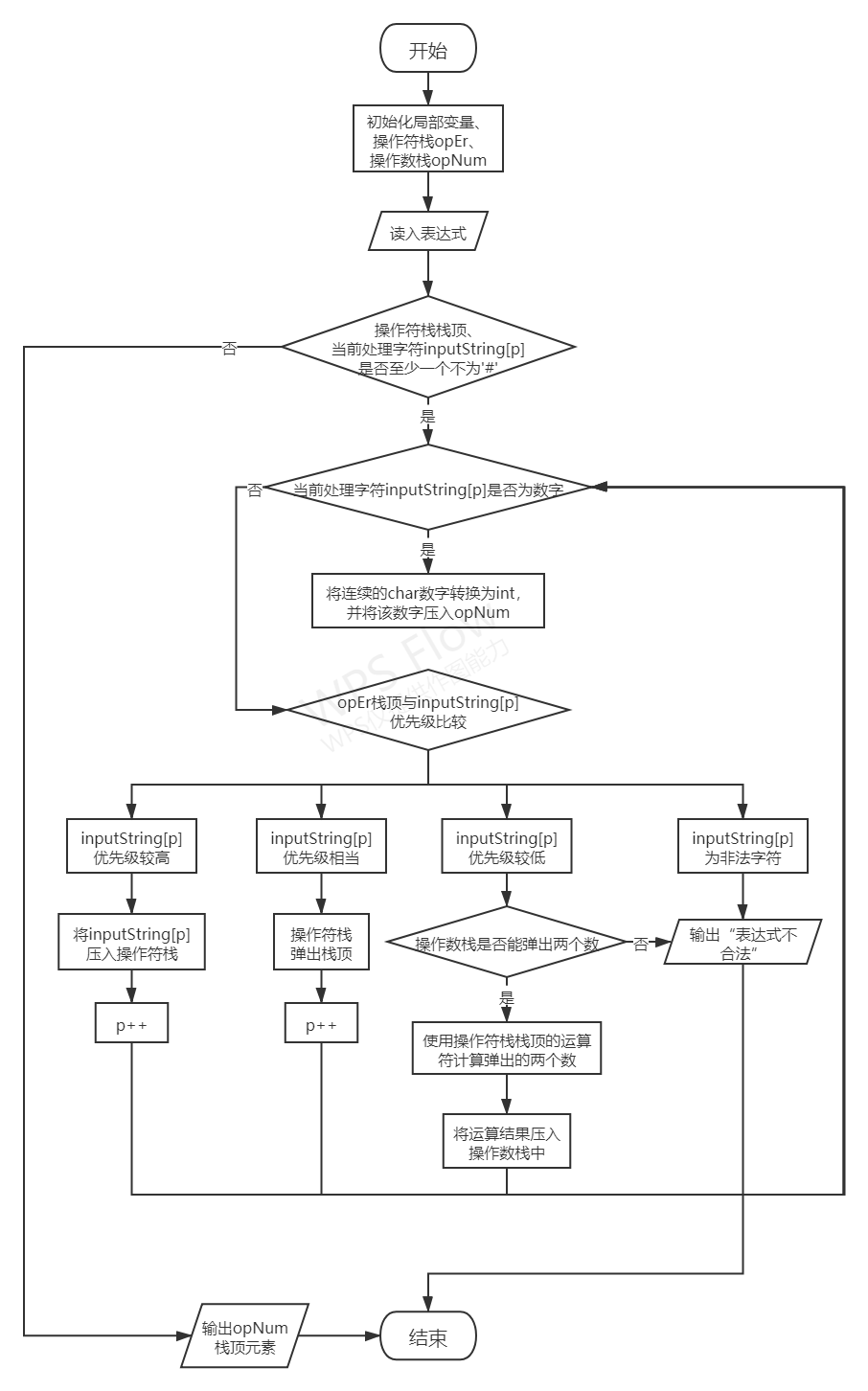
1. （必做题）括号匹配



题5-2

主函数示意图

1. （选做题）算术表达式的计算



题5-3

主函数示意图

### 五、程序代码

1. （课堂完成）顺序栈的基本实现



#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#define Status int

#define OK 1

#define ERROR 0

#define STACK\_INT\_SIZE 100

#define STACKINCREMENT 10

typedef char SElemType;

typedef struct Node {

SElemType \*base;

SElemType \*top;

int stacksize;

} SqStack;

Status InitStack(SqStack \*);

Status Push(SqStack \*, SElemType);

Status Pop(SqStack \*, SElemType \*);

int main() {

SqStack t;

SElemType \*num;

num = (SElemType \*) malloc(sizeof(SElemType));

char ch;

InitStack(&t);

printf("input some chars separate with space or enter(ending with #)\n");

//以“#”作为输入的结尾，以此判断输入结束

while (1) {

scanf("%c", &ch);

if (ch == '#')

break;

if (isspace(ch))

continue;

Push(&t, ch);

}

while (Pop(&t, num) == OK)

printf("%c ", \*num);

printf("\n");

system("pause");

return 0;

}

Status InitStack(SqStack \*S) {

S->base = (SElemType \*) malloc(sizeof(SElemType) \* STACK\_INT\_SIZE);

if (S->base == NULL)

return ERROR;

S->stacksize = STACK\_INT\_SIZE;

S->top = S->base;

return OK;

}

Status Push(SqStack \*S, SElemType e) {

int length = (int) (S->top - S->base);

//把e放入就满了，需要多申请STACKINCREMENT个位置

if (length + 1 == S->stacksize) {

S->base = realloc(S->base, sizeof(SElemType) \* (S->stacksize + STACKINCREMENT));

if (S->base == NULL)

return ERROR;

S->stacksize += STACKINCREMENT;

}

\*(S->top) = e;

S->top++;

return OK;

}

Status Pop(SqStack \*S, SElemType \*e) {

//栈为空，无法弹出

if (S->base == S->top)

return ERROR;

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

return OK;

}

1. （必做题）括号匹配



#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define Status int

#define OK 1

#define ERROR 0

#define STACK\_INT\_SIZE 100

#define STACKINCREMENT 10

typedef char SElemType;

typedef struct Node {

SElemType \*base;

SElemType \*top;

int stacksize;

} SqStack;

Status InitStack(SqStack \*);

Status Push(SqStack \*, SElemType);

Status Pop(SqStack \*, SElemType \*);

Status Empty(SqStack);

Status leftJudge(char);

Status rightJudge(char);

Status match(char, char);

int main() {

SqStack t;

SElemType \*tmpElem;

char inputString[500] = {'\0',};

int p = 0;

tmpElem = (SElemType \*) malloc(sizeof(SElemType));

\*tmpElem = '\0';

InitStack(&t);

printf("input an expression:\n");

//读取一整行输入

gets(inputString);

while (p < 500 && inputString[p] != '\0') {

//由于只需要知道括号是否匹配，只要对括号进行处理即可

//处理的是左括号，直接入栈

if (leftJudge(inputString[p])) {

Push(&t, inputString[p]);

p++;

} else if (rightJudge(inputString[p])) {

//处理的是右括号，先把栈顶取出以进行匹配

//栈空，当前右括号没有东西可以匹配了

if (Pop(&t, tmpElem) != OK) {

printf("Fail to match all brackets.\n");

system("pause");

return 0;

}

//若不匹配，输出错误信息直接退出程序

if (!match(\*tmpElem, inputString[p])) {

printf("Fail to match all brackets.\n");

system("pause");

return 0;

} else {

//若匹配，继续判断下一个

p++;

continue;

}

} else {

//处理的不是括号，直接跳过看下一个字符

p++;

}

}

//栈为空，说明括号全部匹配完成

//栈不为空，说明有括号没有匹配上

if (Empty(t))

printf("All brackets matched.\n");

else

printf("Fail to match all brackets.\n");

system("pause");

return 0;

}

//判断是否为左括号

Status leftJudge(char ch) {

return ch == '(' || ch == '[' || ch == '{';

}

//判断是否为右括号

Status rightJudge(char ch) {

return ch == ')' || ch == ']' || ch == '}';

}

//判断ch1与ch2是否为匹配的括号

Status match(char ch1, char ch2) {

return (ch1 == '(' && ch2 == ')') ||

(ch1 == '[' && ch2 == ']') ||

(ch1 == '{' && ch2 == '}');

}

Status InitStack(SqStack \*S) {

S->base = (SElemType \*) malloc(sizeof(SElemType) \* STACK\_INT\_SIZE);

memset(S->base, '\0', STACK\_INT\_SIZE);

if (S->base == NULL)

return ERROR;

S->stacksize = STACK\_INT\_SIZE;

S->top = S->base;

return OK;

}

Status Push(SqStack \*S, SElemType e) {

int length = (int) (S->top - S->base);

//把e放入就满了，需要多申请STACKINCREMENT个位置

if (length + 1 == S->stacksize) {

S->base = realloc(S->base, sizeof(SElemType) \* (S->stacksize + STACKINCREMENT));

if (S->base == NULL)

return ERROR;

}

\*(S->top) = e;

S->top++;

return OK;

}

Status Pop(SqStack \*S, SElemType \*e) {

//栈为空，无法弹出

if (S->base == S->top)

return ERROR;

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

return OK;

}

Status Empty(SqStack S) {

return S.base == S.top ? OK : ERROR;

}

1. （选做题）算术表达式的计算



#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define Status int

#define OK 1

#define ERROR 0

#define STACK\_INT\_SIZE 100

#define STACKINCREMENT 10

typedef struct {

char \*base;

char \*top;

int stacksize;

} charStack;

typedef struct {

int \*base;

int \*top;

int stacksize;

} intStack;

Status initCharStack(charStack \*);

Status pushCharStack(charStack \*, char);

Status popCharStack(charStack \*, char \*);

char frontCharStack(charStack);

Status initIntStack(intStack \*);

Status pushIntStack(intStack \*, int);

Status popIntStack(intStack \*, int \*);

char compareOp(char, char);

Status isNum(char);

int cal(int, int, char);

//下标0代表+、-，1代表\*、/，2代表(，3代表)，4代表开始/结束符#

char rules[5][5] = {{'>', '<', '<', '>', '>'},

{'>', '>', '<', '>', '>'},

{'<', '<', '<', '=', '!'},

{'>', '>', '!', '>', '>'},

{'<', '<', '<', '!', '='}};

int main() {

char inputString[200] = {'\0'};

int p = 0, tmp;

char res, op;

int num1, num2;

intStack opNum;

charStack opEr;

initCharStack(&opEr);

initIntStack(&opNum);

printf("input an expression:\n");

gets(inputString);

//#作为垫底和结尾元素,即作为开始符和结束符

pushCharStack(&opEr, '#');

inputString[strlen(inputString)] = '#';

while (inputString[p] != '#' || frontCharStack(opEr) != '#') {

//操作数直接进栈

if (isNum(inputString[p])) {

tmp = 0;

while (isNum(inputString[p])) {

tmp = tmp \* 10 + (inputString[p] - '0');

p++;

}

pushIntStack(&opNum, tmp);

} else {

res = compareOp(frontCharStack(opEr), inputString[p]);

if (res == '<') {

//当前操作符优先级高，将当前操作符加入操作符栈

pushCharStack(&opEr, inputString[p]);

p++;

} else if (res == '=') {

//右括号碰到左括号，消去

popCharStack(&opEr, &op);

p++;

} else if (res == '>') {

//当前操作符优先级低于栈顶的，弹出两个操作数进行计算

//通过操作数栈中元素是否够以判断是否有形如2++5的无效表达式

if (popIntStack(&opNum, &num1) != OK || popIntStack(&opNum, &num2) != OK) {

printf("Expression invalid.\n");

system("pause");

return 0;

}

popCharStack(&opEr, &op);

pushIntStack(&opNum, cal(num2, num1, op));

} else if (res == '!') {

printf("Expression invalid.\n");

system("pause");

return 0;

}

}

}

popIntStack(&opNum, &num1);

printf("= %d\n", num1);

system("pause");

return 0;

}

//比较firstOp和secondOp的优先级

//返回'<'表示secondOp优先级较高

//返回'>'表示firstOp优先级较高

//返回'='表示优先级相当，用于匹配消去括号

//返回'!'表示有语法错误

char compareOp(char firstOp, char secondOp) {

int i, j;

switch (firstOp) {

case '+':

case '-':

i = 0;

break;

case '\*':

case '/':

i = 1;

break;

case '(':

i = 2;

break;

case ')':

i = 3;

break;

case '#':

i = 4;

break;

default:

return '!';

}

switch (secondOp) {

case '+':

case '-':

j = 0;

break;

case '\*':

case '/':

j = 1;

break;

case '(':

j = 2;

break;

case ')':

j = 3;

break;

case '#':

j = 4;

break;

default:

return '!';

}

return rules[i][j];

}

Status isNum(char ch) {

return ch <= '9' && ch >= '0' ? OK : ERROR;

}

int cal(int a, int b, char op) {

switch (op) {

case '+':

return a + b;

case '-':

return a - b;

case '\*':

return a \* b;

case '/':

return a / b;

default:;

}

}

Status initCharStack(charStack \*S) {

S->base = (char \*) malloc(sizeof(char) \* STACK\_INT\_SIZE);

memset(S->base, '\0', STACK\_INT\_SIZE);

if (S->base == NULL)

return ERROR;

S->stacksize = STACK\_INT\_SIZE;

S->top = S->base;

return OK;

}

Status initIntStack(intStack \*S) {

S->base = (int \*) malloc(sizeof(int) \* STACK\_INT\_SIZE);

memset(S->base, '\0', STACK\_INT\_SIZE);

if (S->base == NULL)

return ERROR;

S->stacksize = STACK\_INT\_SIZE;

S->top = S->base;

return OK;

}

Status pushCharStack(charStack \*S, char e) {

int length = (int) (S->top - S->base);

//把e放入就满了，需要多申请STACKINCREMENT个位置

if (length + 1 == S->stacksize) {

S->base = (char \*) realloc(S->base, sizeof(char) \* (S->stacksize + STACKINCREMENT));

if (S->base == NULL)

return ERROR;

}

\*(S->top) = e;

S->top++;

return OK;

}

Status pushIntStack(intStack \*S, int e) {

int length = (int) (S->top - S->base);

//把e放入就满了，需要多申请STACKINCREMENT个位置

if (length + 1 == S->stacksize) {

S->base = (int \*) realloc(S->base, sizeof(int) \* (S->stacksize + STACKINCREMENT));

if (S->base == NULL)

return ERROR;

}

\*(S->top) = e;

S->top++;

return OK;

}

Status popCharStack(charStack \*S, char \*e) {

//栈为空，无法弹出

if (S->base == S->top)

return ERROR;

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

return OK;

}

char frontCharStack(charStack S) {

if (S.base == S.top)

return '\0';

return \*(S.top - 1);

}

Status popIntStack(intStack \*S, int \*e) {

//栈为空，无法弹出

if (S->base == S->top)

return ERROR;

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

return OK;

}

### 六、测试和结果

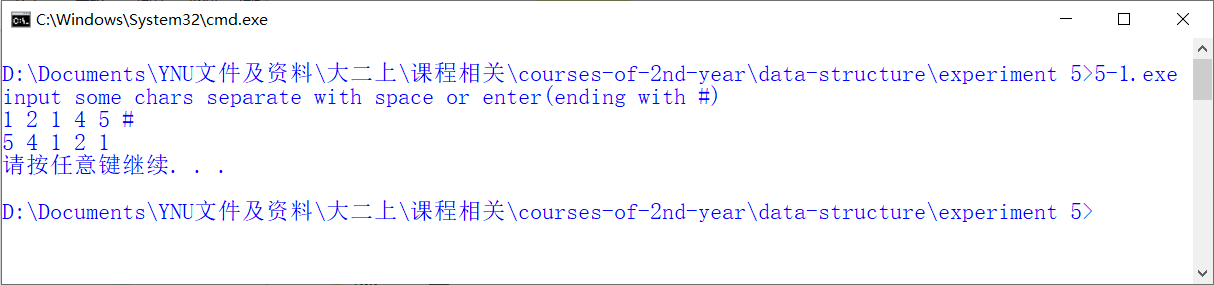
1. （课堂完成）顺序栈的基本实现

**Input:**

1 2 1 4 5 #

**Output:**

5 4 1 2 1



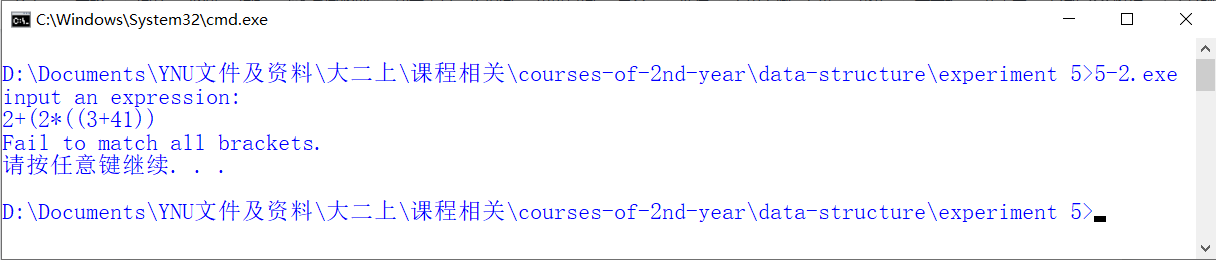
1. （必做题）括号匹配

**Input:**

2+(2\*((3+41))

**Output:**

Fail to match all brackets.



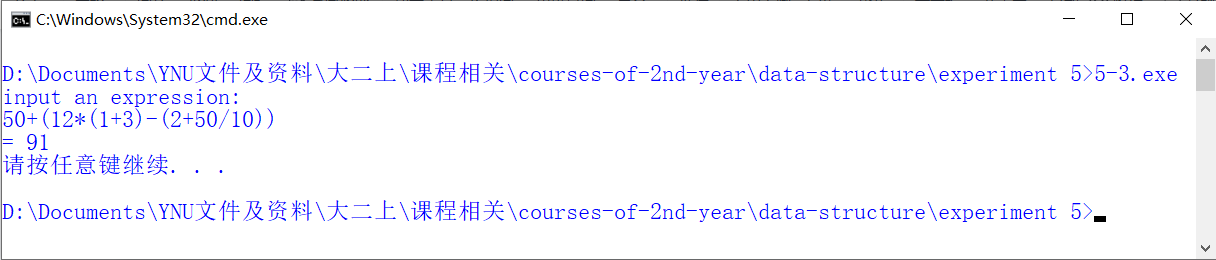
1. （选做题）算术表达式的计算

**Input:**

50+(12\*(1+3)-(2+50/10))

**Output:**

= 91



### 用户手册

* 1. （课堂完成）顺序栈的基本实现

输入的数据均被作为字符型处理，输入的元素之间以空格分隔，输入必须以#结尾作为结束符。输出的数据是按链表从头往尾的顺序的。

* 1. （必做题）括号匹配

输入的表达式长度最大为500，程序只针对小括号、中括号和大括号是否匹配来判断表达式是否有效，表达式的其他无效情况均无法检测。

* 1. （选做题）算术表达式的计算

输入的表达式必须保证：

- 各操作数必须为正数（表达式的开头也不能是负号）；

- 操作符只支持加、减、乘、整除、小括号；

- 计算中的任何一步的结果都不超过int的范围。

需要注意的是，表达式中所有的除号“/”均视为整除