**实验三 栈与队列实验**

**-------基于栈结构的中缀表达式求值**

**计算接 166 班 李彪 6130116194**

1. **问题描述**

从键盘输入任意中缀表达式字符串，读字符串，利用栈结构实现表达式求值。

1. **输入与输出**

输入：从键盘中缀表达式如： 32＋5×（6－4）

输出：计算结果 42

1. **需求分析**

1.定义两个栈结构，数栈用于存放表达式中的数，符号栈用于存放表达式中的符号，实现栈的运算

2.在读数的时候考虑多位运算

3.实现表达式求值

1. **开发工具与环境**

硬件设备：微型计算机系统

系统：debian linux

编译器 g++

1. **概要设计**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参考结构定义 | |  |  |  |  |
| { | typedef struct | /\* 运算符栈 \*/ |  |  |  |
| char \*base,\*top; |  |  |  |  |
|  | int stacksize; |  |  |  |  |
| }SqStack; | |  |  |  |  |
| typedef struct | | /\* 运算数栈 \*/ |  |  |  |
| { | int \*base,\*top; |  |  |  |  |
|  | int stacksize; |  |  |  |  |
| }SqStack1; | |  |  |  |  |
| int priority[7][7]={{'>', '>', '<', '<', '<', '>', '>'}, | | | | // + |  |
|  | {'>', '>', '<', '<', '<', '>', '>'}, | | // - | |  |
|  | {'>', '>', '>', '>', '<', '>', '>'}, | | // \* | |  |
|  | {'>', '>', '>', '>', '<', '>', '>'}, | | // / | |  |
|  | {'<', '<', '<', '<', '<', '=', }, | | // ( | |  |
|  | {'>', '>', '>', '>', ' ', '>', '>'}, | | // ) | |  |
| }; | {'<', '<', '<', '<', '<', ' ', '='} | | // | # |  |
|  |  |  |  |  |

/\*用于比较符号优先级的全局二维数组\*/

2．各函数模块

void InitStack(SqStack \*s);

操作结果：初始化运算符栈

char GetTop(SqStack \*s);

操作结果：得到运算符栈的栈顶元素

void Push(SqStack \*s,char e);

操作结果：对运算符栈进行压栈操作

int IsNumber(char c);

操作结果：判断一个字符是否是数字

int MidExpression\_Eval(char Express[]);

操作结果：计算中缀表达式的值

int Operate (int a,char x,int b);

操作结果：计算表达式 axb，并返回结果

等等

1. **详细设计**

#include <iostream>

#include <cmath>

#include <cstring>

using namespace std;

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

* 定义栈类型

typedef struct /\* 运算符栈 \*/

{

char \*base, \*top;

int stacksize;

} SqStackSymb;

typedef struct /\* 运算数栈 \*/

{

float \*base, \*top;

int stacksize;

} SqStackNum;

* 使用的常量

const int STACK\_INIT\_SIZE = 100;

const int STACK\_INCREMENT = 10;

* 用于比较符号优先级的全局二维数组

const char priority[7][7] = {

{'>', '>', '<', '<', '<', '>', '>'}, // +

{'>', '>', '<', '<', '<', '>', '>'}, // -

{'>', '>', '>', '>', '<', '>', '>'}, // \*

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

{'<', '<', '<', '<', '<', '=', ' '}, // (

{'>', '>', '>', '>', ' ', '>', '>'}, // )

{'<', '<', '<', '<', '<', ' ', '='} // #

};

* 与比较符号优先级二维数组对应的操作符全集 char OP[] = {'+', '-', '\*', '/', '(', ')', '#'};

/\* -------------------初始化------------------------- \*/

* 初始化运算符栈

void InitStackSymb(SqStackSymb &s)

{

s.base = (char \*)malloc(STACK\_INIT\_SIZE \* sizeof(char)); s.top = s.base;

s.stacksize = STACK\_INIT\_SIZE;

}

* 初始化运算数栈

void InitStackNum(SqStackNum &s)

{

s.base = (float \*)malloc(STACK\_INIT\_SIZE \* sizeof(float)); s.top = s.base;

s.stacksize = STACK\_INIT\_SIZE;

}

/\* -------------------push ----------------------\*/

void Push(SqStackSymb &s, char e)

{ // 将运算符压入运算符栈

if (s.top - s.base >= s.stacksize)

{

s.base = (char \*)realloc(s.base, (s.stacksize + STACK\_INCREMENT) \* sizeof(char)); s.top = s.base + s.stacksize;

s.stacksize += STACK\_INCREMENT;

}

\*s.top++ = e;

}

void PushNum(SqStackNum &s, float e)

{ // 将运算数压入运算数栈

if (s.top - s.base >= s.stacksize)

{

s.base = (float \*)realloc(s.base, (s.stacksize + STACK\_INCREMENT) \* sizeof(float)); s.top = s.base + s.stacksize;

s.stacksize += STACK\_INCREMENT;

}

\*s.top++ = e;

}

/\* -----------------pop--------------------- \*/

void Pop(SqStackSymb &s, char &e)

{ // 将运算符栈顶元素出栈

if (s.top != s.base)

e = \*--s.top;

}

void PopNum(SqStackNum &s, float &e)

{ // 将运算数栈顶元素出栈

if (s.top != s.base)

e = \*--s.top;

}

/\* ----------------获取栈顶元素 -----------------\*/

* 获取运算符栈的栈顶元素

char GetTop(SqStackSymb &s)

{

if (s.top != s.base)

{

return \*(s.top - 1);

}

return '!'; // 出错返回感叹号

}

* 获取运算数栈的栈顶元素

float GetTopNum(SqStackNum &s)

{

if (s.top != s.base)

{

return \*(s.top - 1);

}

return FALSE; // 出错则返回 0

}

* 判断是否是数字

bool isNumber(char c)

{

if (c >= '0' && c <= '9')

{

return true;

}

elsereturn false;

}

* 计算表达式 axb，并返回结果

float Operate(float a, char x, float b)

{

float res = 0;

switch (x)

{

case '+':

res = a + b;

break;

case '\*':

res = a \* b;

break;

case '-':

res = a - b;

break;

case '/':

res = a / b;

break;

default:

break;

}

return res;

}

//C++里面好像没有获取数组元素下下标的函数？？？自己实现

int getIndex(char ops[], char e)

{

for (int i = 0; i < 7; i++)

{

if (e == ops[i])

{

return i;

}

}

return -1;

}

char Precede(char top, char c)

{

return priority[getIndex(OP, top)][getIndex(OP, c)];

}

float MidExpEval(char expstr[])

{

SqStackSymb stackSymb; // 声明运算符栈

SqStackNum stackNum; // 声明运算数栈

InitStackSymb(stackSymb); // 初始化运算符栈

InitStackNum(stackNum); // 初始化运算数栈

int i = 0; // 保存表达式的读取位置

char currChar; // 保存目前读取表达式的字符

Push(stackSymb, '#'); // 将'#'起始符压入操作符栈

currChar = expstr[i]; // 拿到表达式的第一个值

while (currChar != '#' || GetTop(stackSymb) != '#')

{

char numbers[40]; // 存储多位数字

int j = 0; // 存储数字数组下标，方便增加位数, 同时也可以记录数字数组的长度 while (isNumber(currChar) || currChar == '.')

{ // 若是数字或小数点，则循环读取

}

numbers[j] = currChar;

i++;

j++;

currChar = expstr[i];

if (j > 0)

{ // 若数字数组长度大于 0，则 2 循环数组求多位数字的值并压入运算数栈

float sum = 0;

// int dot = getIndex(numbers, '.'); // 若无小数点则返回-1

|  |  |  |
| --- | --- | --- |
| for (int k = 0; k < j; k++) |  |  |
| { |  |  |
| if (numbers[k] == '.') |  |  |
| { |  |  |
| continue; |  |  |
| } |  |  |
| else |  |  |
| { |  |  |
| sum += (numbers[k] - '0') \* pow(10, j - 1 - k); | |  |
| } |  |  |
| } |  |  |
| PushNum(stackNum, sum); |  |  |
| } |  |  |
| else |  |  |
| { |  |  |
| switch (Precede(GetTop(stackSymb), currChar)) | |  |
| { | // 栈顶元素优先级比此时的取值更低 |  |
| case '<': |  |
| Push(stackSymb, currChar); // 压入运算符栈 | |  |
| currChar = expstr[++i]; | // 使读取的位置后移，并取到下一个字符 |  |
| break; |  |  |
| case '=': |  |  |
| char x; |  |  |

Pop(stackSymb, x);

currChar = expstr[++i];

break;

case '>':

char theta;

float a, b;

Pop(stackSymb, theta);

PopNum(stackNum, b);

PopNum(stackNum, a);

PushNum(stackNum, Operate(a, theta, b));

default:

break;

}

}

}

return GetTopNum(stackNum);

}

int main()

{

}

char expstr[STACK\_INIT\_SIZE] = {}; // 存储中缀表达式 float exc\_result;

cout << "请输入中缀运算式: ";

cin >> expstr;

expstr[strlen(expstr)] = '#';//strlen CPP 库函数 获取字符串长度

exc\_result = MidExpEval(expstr);

expstr[strlen(expstr) - 1] = '\0';

cout << expstr << '=' << exc\_result << endl; return 0;

1. 调试结果（运行结果）



**实验心得与总结**

在代码中只是考虑了多位数的计算，忽略了小数的处理。。

<pre style="font-size:10px;padding:0 30px">

//在原函数中已经加了处理的入口。只需加上这段代码即可

if (dot != -1)

{

if (k < dot)

{

sum += (numbers[k] - '0') \* pow(10, dot - k - 1); //将字符转换成对应的

多位数字

}

else if (k > dot)

{ //若没有小数点，这个块不执行

sum += (numbers[k] - '0') \* pow(10, dot - k);

}

}

</pre>

运行结果：

