**成绩:**

**《编译原理》课程实验报告**

**专 业：**计算机科学与技术

**班 级：**1720541

**学 号：**172054110

**姓 名：**段佳琪

**日期： 2020 年 6 月 12 日**

目录

[实验一 词法分析 3](#_Toc41033082)

[实验二 递归下降分析法 4](#_Toc41033083)

[实验三 LL(1)分析法 5](#_Toc41033084)

[实验四 逆波兰式的产生的计算 6](#_Toc41033085)

# 实验一 词法分析

1. **实验题目**

词法分析器

1. **实验目的**

编制一个读单词过程，从输入的源程序中，识别出各个具有独立意义的单词，即基本保留字、标识符、常数、运算符、分隔符五大类。并依次输出各个单词的内部编码及单词符号自身值

1. **实验内容**

要求输出如下二元式序列：

|  |  |  |
| --- | --- | --- |
| （ 4， ”>=”） | （2， ”b”） | （5，”）”） |
| （ 2， ”j”） | （4， ”=”） | （ 2， ”a”） |
| （ 4， ”-”） | （2， ”b”） | （ 5， ”;”） |
| （ 1， ”else”） （2， ”j”） | | （4， ”=”） |
| （ 2， ”b”） | （ 4， ”-”） | （ 2， ”a”） |
| （ 5， ”;”） | （5， ”}”） |  |

其中程序语言的单词符号一般分为五种：

1. 关键字（保留字 / 基本字） if 、while 、 begin等
2. 标识符：常量名、变量名 等
3. 常数： 34 、 56.78 、 true 、,a? 、 等
4. 运算符： + 、 - 、 \* 、/ 、〈 、 and 、or 、 等
5. 界限符：，；（ ） { } /\*
6. **实验步骤**
   1. 问题分析

要求从输入的一段源程序中，识别出各个具有独立意义的单词，即上述的五类，首先构造一个保留字表，然后每输入一个字符就检测应该进入什么状态，一直循坏，最后根据所在的接受状态以及保留字表识别单词。

* 1. 算法描述（可以包含使用的数据结构、算法、算法流程图等）

首先输入一段字符串代码，从首字符开始识别

当识别字符为字母时，依次识别，字符长度依次增加，当遇到空格符时，停止识别，将识别到的字符串与数组相比较，相同即为关键字，不同即为标识符。

当识别字符为数字时，依次识别，字符长度依次增加，当遇到空格时，停止识别，取到的即为常数。

当识别的字符为+，-，=，\*，/，即取到的为运算符；当识别的字符为；，>,<,(,),{,},”,即取到的为界限符；直接输出。

* 1. 程序源代码

#include<iostream>

#include<fstream>

#include<cstring>

using namespace std;

#define MAX 10

#define LIST\_MAX\_LENGTH 10

char ch = ' ';

char strToken[LIST\_MAX\_LENGTH];

int len = 0;

int s = 0;

Char key[30][10]={ "if","else","for","while","do","bool","int","char","double","friend","class","struct","return","delete","new","public",

"private","endl","break","const","static","include","endl","auto","iostream","cstring","define","void","cin","cout" };

char GetChar()

{

ch = getchar();

if (ch != '/n')

return ch;

return ' ';

}

char GetBC()

{

while (ch == ' ')

ch = GetChar();

return ch;

}

void Concat()

{ strToken[len] = ch; len++; }

int Reserve(char strToken[])//保留字

{

for(int i=0;i<(sizeof(key)/sizeof(key[0]));i++)

if (strcmp(key[i], strToken) == 0)

{ return 1; }

return 0;

}

int IsLetter()

{

if ((((ch <= 'z') && (ch >= 'a')) || (ch <= 'Z') && (ch >= 'A')))

return 1;

else

return 0;

}

int IsDigit()

{

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

return 1;

else

return 0;

}

void InsertId()

{

ofstream outfile("symbol.text", ios::app);

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

{ outfile << strToken[i]; }

outfile << endl; outfile.close();

}

void InsertConst()

{

ofstream outfile("number.text", ios::app);

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

{ outfile << strToken[i]; }

outfile << endl; outfile.close();

}

int main()

{

cout << "请输入字符串代码：" << endl;

if (s != 1)

{ GetChar(); }

while (ch != 10)

{

GetBC();

if (IsLetter())

{

while (IsLetter())

{

Concat(); GetChar();

}

s = 1;

if (Reserve(strToken))

{

cout << '('<< 1<<','<<strToken <<')'<< endl;

len = 0; memset(strToken, 0, sizeof(strToken));

}

else

{

cout <<'('<< 2<<','<<strToken<<')'<< endl;

InsertId(); len = 0;

memset(strToken, 0, sizeof(strToken));

}

}

else if (IsDigit())

{

while (IsDigit())

{

Concat(); GetChar();

}

s = 1;

cout <<'('<<3<<','<< strToken<<')'<< endl;

InsertConst(); len = 0;

memset(strToken, 0, sizeof(strToken));

}

else if (ch == '#') { cout <<'#'<< " 警告" << endl; s = 0; }

else if (ch == '+') { cout <<'('<< 4 <<','<<" +" <<')'<<endl; s = 0; }

else if (ch == '-') { cout <<'('<< 4 <<','<<" -" <<')'<< endl; s = 0; }

else if (ch == '=') { cout <<'('<< 4 <<','<<" =" <<')'<<endl; s = 0; }

else if (ch == '\*') { cout <<'('<< 4 <<','<<" \*" <<')'<<endl; s = 0; }

else if (ch == '/') { cout <<'('<< 4 <<','<< "/" <<')'<< endl; s = 0; }

else if (ch == ';') { cout <<'('<< 5 <<','<< ";" <<')'<< endl; s = 0; }

else if (ch == '>') { cout <<'('<< 5 <<','<< ">" <<')'<< endl; s = 0; }

else if (ch == '<') { cout <<'('<< 5 <<','<< "<" <<')'<< endl; s = 0; }

else if (ch == '(') { cout <<'('<< 5 <<','<< "(" <<')'<< endl; s = 0; }

else if (ch == ')') { cout <<'('<< 5 <<','<< ")" <<')'<< endl; s = 0; }

else if (ch == '{') { cout <<'('<< 5 <<','<< "{" <<')'<< endl; s = 0; }

else if (ch == '}') { cout <<'('<< 5 <<','<< "}" <<')'<< endl; s = 0; }

else if (ch == '"') { cout <<'('<< 5 <<','<< '"' <<')'<< endl; s = 0; }

else cout << ch << " 错误处理" << endl;

if (s != 1)

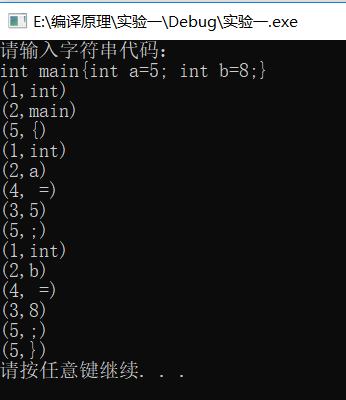
{

GetChar();

}}

system("pause"); return 0;}

* 1. 运行结果



1. **出现的问题及解决的方法**

# 实验二 递归下降分析法

**一．实验题目**

递归下降分析法

**二．实验目的**

根据某一文法编制调试递归下降分析程序，以便对任意输入的符号串进行分析。本次实验的目的主要是加深对递归下降分析法的理解。

**三．实验内容**

实验规定对下列文法G【E】，用递归下降分析法对任意输入的符号串进行分析：

E→E+T|E-T | T

T→T\*F|T/F | F

F→(E) | i

消除左递归、提取公共左因子后的等价文法：

E →TE’

E’→(+T|-T)E’ | ε

T →FT’

T’→(\*F|/F)T’ | ε

F→(E) | i

输入：输入算术表达式以“#”结束，比如：（i+i）\*i#, (i-i)/i#, i+i\*i#，(i+i\*i#等。

输出：是否分析成功

**四．实验步骤**

（1）问题分析

对于递归下降分析，每个非终结符都有一个对应的函数，程序从开始符号的对应函数开始执行，如果程序成功扫描了整个输入字符串，就代表语法分析成功

在调用非终结符对应的函数时就会遇见两种情况：

* 遇到终结符，因为终结符本质上是token，所以直接把这个终结符和句子中对应位置的token进行比较，判断是否符合即可；符合就继续，不符合就返回
* 遇到非终结符，此时只需要调用这个非终结符对应的函数即可。在这里函数可能会递归的调用，这也是算法名称的来源。

递归下降分析法的原理是利用函数之间的递归调用模拟语法树自上而下的构造过程，具体实现方法概述如下：

* 1）每个非终结符对应一个解析函数；
* 2）产生式右侧为该产生式左侧非终结符所对应解析函数的“函数体”；
* 3）产生式右侧终结符对应从输入串中“消耗”该终结符的操作；
* 4）产生式中的‘|’对应函数体中的“if-else”语句；

（2）算法描述（可以包含使用的数据结构、算法、算法流程图等）

1. 输入字符串，将其归约为T和E\_
2. T又可归约为F和T\_，F则判断字符是否为i，是则分析成功，否则判断字符是否为（或者），是则分析成功，否则分析失败。
3. T\_则判断字符是否为\*或/，是则返回T继续归约
4. E\_则判断字符是否为+或-，是则继续看下一个字符，否则失败。

（3）程序源代码

#include<iostream>

#include<cstring>

using namespace std;

int k=0;

char sentence[50];

int flag = 1;

void E(char ch);

void E\_(char ch);

void T(char ch);

void T\_(char ch);

void F(char ch);

void E\_(char ch) {

if (ch == '+'||ch=='-') {

k++; ch = sentence[k]； T(ch);

}

else { flag = 0; }

}

void T\_(char ch) {

if (ch == '\*' || ch == '/'){

k++; ch = sentence[k]; T(ch);}

}

void F(char ch) {

if (ch == 'i') {

k++; ch = sentence[k]; E\_(ch);}

else if (ch == '(') {

k++; ch = sentence[k]; E(ch);}

else if (ch == ')') { k++; }

else { flag = 0;}

}

void T(char ch) {

F(ch); T\_(ch);}

void E(char ch) {

T(ch); E\_(ch);}

int main() {

do {

cout << "请输入字符串,以#结尾" << endl;

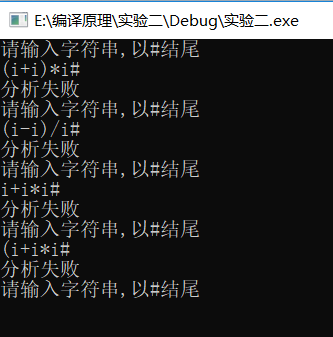
cin >> sentence; E(sentence[k]);

if (flag == 1) {cout << "分析成功"<<endl;}

else { cout << "分析失败"<<endl;}

} while (1);}

（4）运行结果



**五．出现的问题及解决的方法**

# 实验三 LL(1)分析法

**一．实验题目**

LL（1）分析法

**二．实验目的**

根据某一文法编制调试 LL(1) 分析程序， 以便对任意输入的符号串进行分析。 本次实验的目的主要是加深对预测分析 LL(1) 分析法的理解。

**三．实验内容**

实验规定对下列文法G【E】，用LL （1）分析法对任意输入的符号串进行分析：

E→E+T | T

T→T\*F | F

F→(E) | i

消除左递归后的等价文法：

E →TE’

E’→+TE’ | ε

T →FT’

T’→\*FT’ | ε

F→(E) | i

若输入串为i+i#，则输出分析过程：

|  |  |  |  |
| --- | --- | --- | --- |
| 步骤 | 分析栈 | 输入串 | 所用产生式 |
| 1 | #E | i+i# | E->TE’ |
| 2 | #E’T | i+i# | T->FT’ |
| 3 | #E’T’F | i+i# | F ->i |
| 4 | #E’T’i | i+i# |  |
| 5 | #E’T’ | +i# | T’->ε |
| 6 | #E’ | +i# | E’->+TE’ |
| 7 | #E’T+ | +i# |  |
| 8 | #E’T | i# | T -> FT’ |
| 9 | #E’T’F | i# | F -> i |
| 10 | #E’T’i | i# |  |
| 11 | #E’T’ | # | T’ ->ε |
| 12 | #E’ | # | E’ ->ε |
| 13 | # | # | 分析成功 |

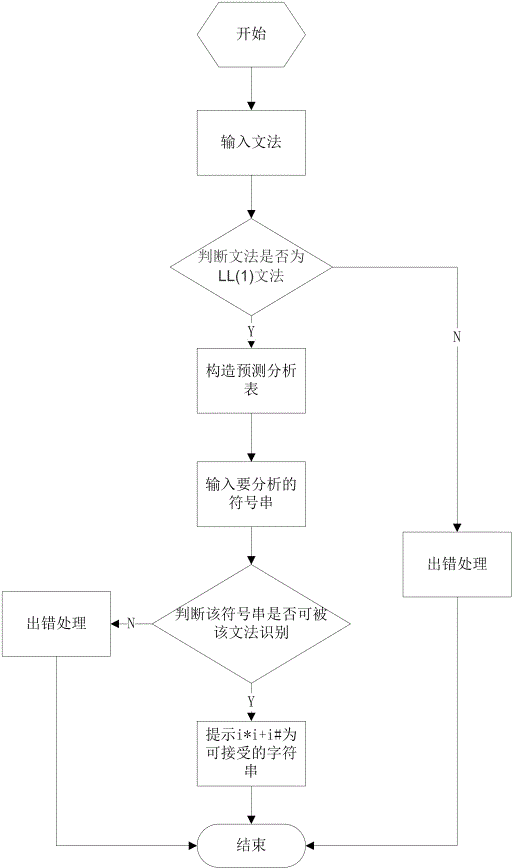
**四．实验步骤**

（1）问题分析

1.首先将‘ # ’压入堆栈中, 将开始符号S 也压入堆栈中，读取第一个输入符号到a。

2.栈顶符号弹出放入X 中,如果X 为终结符号:当X = = a ! =‘ # ’时,表明成功匹配a 符号,然后读取下一个符号到a ,否则出错;当X = = a =‘ # ’时,分析结束,接受句子,跳出循环。  
3.如果X ! = a ,进行出错处理。  
4.如果X 为非终结符号, 则查分析表M :如果M[ X , a ]为空, 进行出错处理; 如果M [X , a ]=‘X1 X2 X3...Xn’, 则将右部Xn...X2 X1 反序压入堆栈中。

（2）算法描述（可以包含使用的数据结构、算法、算法流程图等）



（3）程序源代码

#include<iostream>

#include<cstring>

#include<cstdio>

#include<algorithm>

#include<stack>

using namespace std;

char LL1[50][50][100] = { {"->AT","->AT" ,"null","null" ,"null","null" },

{"->BU","->BU" ,"null","null" ,"null", "null"},

{"null","null" ,"->$" ,"->+AT","null","->$" },

{"null","null" ,"->$" ,"->$" ,"->\*BU","->$" },

{"->i" ,"->(S)","null","null" ,"null","null" } };

char H[200] = "SATUB";

char L[200] = "i()+\*#";

stack<char>cmp;

int findH(char a)

{

for (int i = 0; i < 5; i++){

if (a == H[i]){ return i;}

}

return -1;

}

int findL(char b)

{

for (int i = 0; i < 6; i++)

{

if (b == L[i]){ return i; }

}

return -1;

}

int error(int i, int cnt, int len, char p[], char str[])

{

printf("%d\t%s\t", cnt, p);

for (int q = i; q < len; q++)

{ cout << str[q];}

printf("\t报错\n"); return len;

}

void analyze(char str[], int len)

{

int cnt = 1; int i = 0;

char p[200] = "#S"；int pindex = 2;

printf("Step\tStack\tString\tRule\n");

while (i < len)

{ int x, y; char ch = cmp.top();

if (ch >= 'A'&&ch <= 'Z')

{ cmp.pop(); x = findH(ch)； y = findL(str[i]);

if (x != -1 && y != -1)

{ int len2 = strlen(LL1[x][y]);

if (strcmp(LL1[x][y], "null") == 0)

{ i = error(i, cnt, len, p, str); continue;}

printf("%d\t%s\t", cnt, p);

if (p[pindex - 1] != '#')

{ p[pindex] = '\0'; pindex--;}

if (LL1[x][y][2] != '$')

{

for (int q = len2 - 1; q > 1; q--)

{ p[pindex++] = LL1[x][y][q];

cmp.push(LL1[x][y][q]);}

}

else

{ p[pindex] = '\0'; pindex--;}

for (int q = i; q < len; q++)

{ cout << str[q];}

printf("\t%c%s\n", ch, LL1[x][y]);}

else

{ i = error(i, cnt, len, p, str); continue;}

}

else

{

if (ch == str[i])

{

cmp.pop(); printf("%d\t%s\t", cnt, p);

if (ch == '#'&&str[i] == '#')

{ printf("#\t接受\n"); return;}

for (int q = i; q < len; q++)

{ cout << str[q];}

printf("\t%c匹配\n", ch); pindex--; p[pindex] = '\0'; i++; }

else

{ i = error(i, cnt, len, p, str); continue;}

} cnt++;}

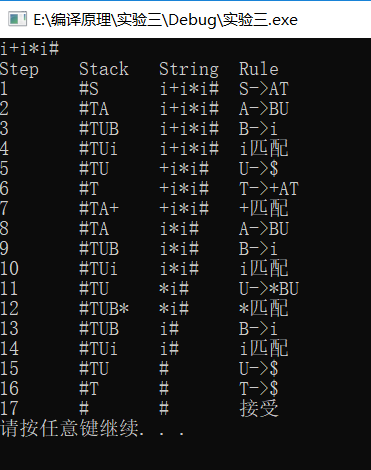
}

int main(）{

char str[200]; cin >> str; int len = strlen(str); cmp.push('#');

cmp.push('S'); analyze(str, len + 1); system("pause"); return 0;}

（4）运行结果



**五．出现的问题及解决的方法**

# 实验四 逆波兰式的产生的计算

**一．实验题目**

逆波兰式的产生及计算

**二．实验目的**

将用中缀式表示的算术表达式转换为用逆波兰式(后缀式)表示的算术表达式，并计算用逆波兰式来表示的算术表达式的值

**三．实验内容**

如输入如下： 21+（（ 42-2） \*15+6 ） -18#

输出为：

原来表达式： 21+ （（ 42-2 ） \*15+6 ） - 18#

后缀表达式： 21&42&2&-15&\*6&++18&-

计算结果： 609

**四．实验步骤**

（1）问题分析

1.设定运算符栈

2.假设表达式的结束符为"#"，我们需要预设运算符栈底元素为"#"

3.扫描表达式，若当前字符是操作数，则直接发送给后缀表达式；

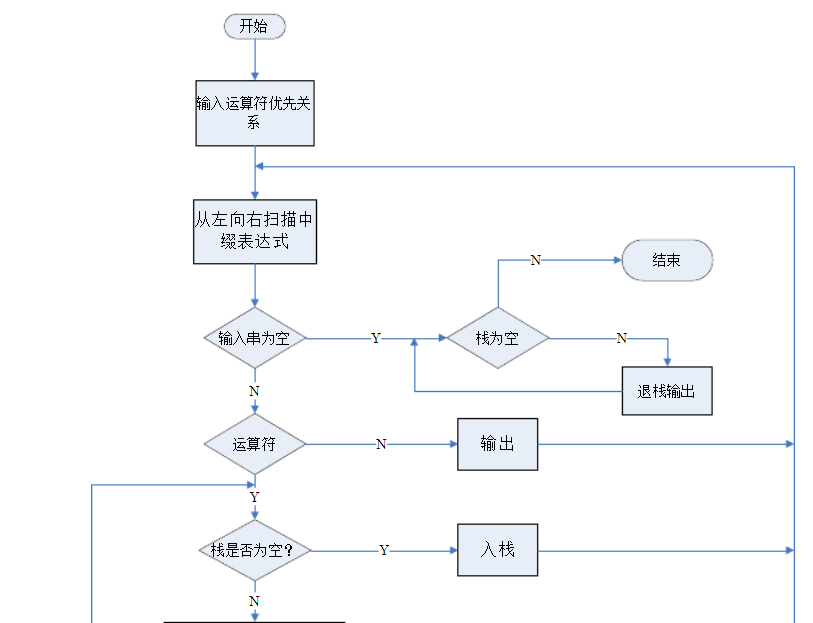
4.若当前字符为运算符且优先级大于栈顶运算符，则进栈，否则退出栈顶运算符并将其发送给后缀式。然后将当前运算符放入栈中。

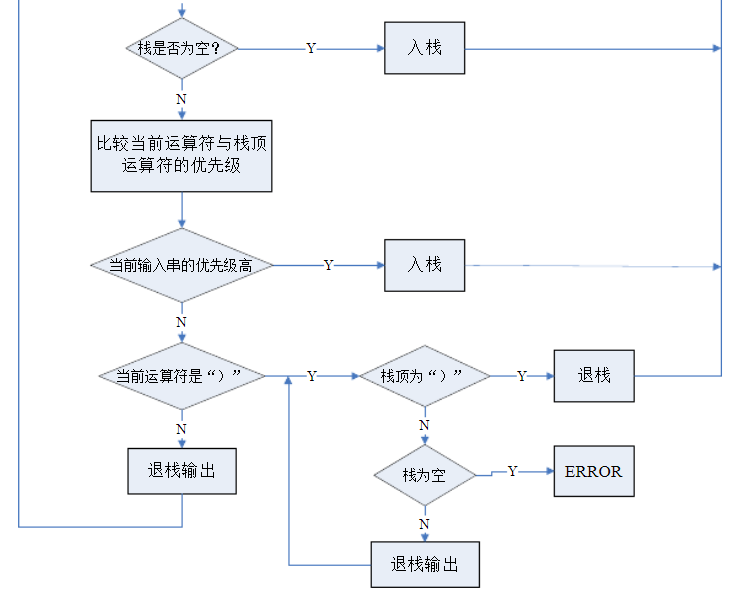
5.若当前字符是结束符，则将栈中的全部运算符依次发送给后缀式。

6.若当前字符为"("，进栈。

7.若当前字符为")"，则从栈顶起，依次将栈中运算符出栈发送给ie后缀式，直到碰到"("。将栈中"("出栈，不需要发送给后缀式。然后继续扫描表达式。

（2）算法描述（可以包含使用的数据结构、算法、算法流程图等）





（3）程序源代码

#include<iostream>

using namespace std;

#define max 100

char ex[max];

void trans()

{

char str[max]; char stack[max]; char ch; int sum, i, j, t, top = 0;

cout<<"请输入一个求值的表达式，以#结束。"<<endl;

cout<<"算术表达式："<<endl; i = 0;

do

{ i++; cin>> str[i];

} while (str[i] != '#'&&i != max);

sum = i; t = 1; i = 1; ch = str[i]; i++;

while (ch != '#')

{

switch (ch)

{

case '(':///判定为左括号

top++; stack[top] = ch; break;

case ')':///判定为右括号

while (stack[top] != '(')

{ ex[t] = stack[top]; top--; t++; }

top--; break;

case '+':///判定为加减号

case '-':

while (top != 0 && stack[top] != '(')

{ ex[t] = stack[top]; top--; t++; }

top++; stack[top] = ch; break;

case '\*':

case '/':

while (stack[top] == '\*' || stack[top] == '/')

{ ex[t] = stack[top]; top--; t++; }

top++; stack[top] = ch; break;

case ' ':

break;

default:

while (ch >= '0'&&ch <= '9')

{ ex[t] = ch; t++; ch = str[i]; i++; }

i--; ex[t] = '&'; t++; break; }

ch = str[i];i++; }

while (top != 0)

{ ex[t] = stack[top]; t++; top--; }

ex[t] = '#'; cout<<"原来表达式："<<endl;

for (j = 1; j < sum; j++)

cout<<str[j];

cout<<"\n后缀表达式："<< ex<<endl;

for (j = 1; j < t; j++)

cout<<ex[j];

}

void compvalue()

{

float stack[max], d; char ch; int t = 1, top = 0;

ch = ex[t]; t++;

while (ch != '#')

{

switch (ch)

{

case '+':

stack[top - 1] = stack[top - 1] + stack[top];

top--; break;

case '-':

stack[top - 1] = stack[top - 1] - stack[top];

top--; break;

case '\*':

stack[top - 1] = stack[top - 1] \* stack[top];

top--; break;

case '/':

if (stack[top] != 0)

stack[top - 1] = stack[top - 1] / stack[top];

else

{ cout << "出现错误" << endl; exit(0); }

top--; break;

default:

d = 0;

while (ch >= '0'&&ch <= '9')

{ d = 10 \* d + ch - '0'; ch = ex[t]; t++; }

top++; stack[top] = d;

}

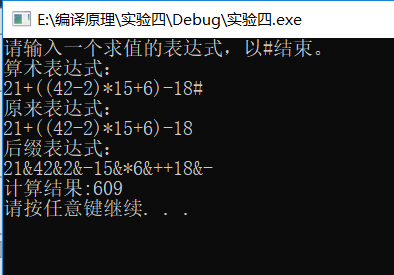
ch = ex[t]; t++;

} cout << "\n计算结果:" << stack[top] << endl;}

int main()

{ trans(); compvalue(); system("pause"); return 0; }

（4）运行结果



**五．出现的问题及解决的方法**