语义分析程序

1. 题目及要求

编写语义分析和翻译程序，实现对算数表达式的类型检查和求值。要求所分析算术表达式由如下文法产生。

E->E+T | E-T | T

T->T\*F | T/F | F

F->(E) | num|num.num

1. 写出满足要求的语法制导定义或翻译方案。
2. 编写语义分析和翻译程序，实现对表达式的类型进行检查和求值，并输出：
3. 分析过程中所用产生式。
4. 识别出的子表达式的类型。
5. 识别出的子表达式的类型。
6. 程序设计说明
7. 拓广文法

(0)E’->E

(1)E->E+T

(2)E->E-T

(3)E->T

(4)T->T\*F

(5)T->T/F

(6)T->F

(7)F->num.num

(8)F->(E)

(9)F->num

1. 翻译方案

设计两个综合属性val和type，分别记录表达式的值和类型。Type属性有有两种取值，分别为integer（整型）和real（实型）。翻译方案如下：

E->E’+T {E.val = E’.val+T.val}

{if( E’.type == integer || T.type == integer ) E.type = integer; else E.type = real;}

E->E’-T {E.val = E’.val - T.val}

{if( E’.type == integer || T.type == integer ) E.type = integer; else E.type = real;}

E -> T {E.val = T.val} {E.type = T.type}

T->T’\*F {T.val = T’.val \* F.val}

{if(T’.type == integer || F.type == integer) T.type = integer; else T.type = real;}

T->T’/F {T.val = T’.val / F.val}

{if( T’.type == integer || F.type == integer ) T.type = integer; else T.type = real;}

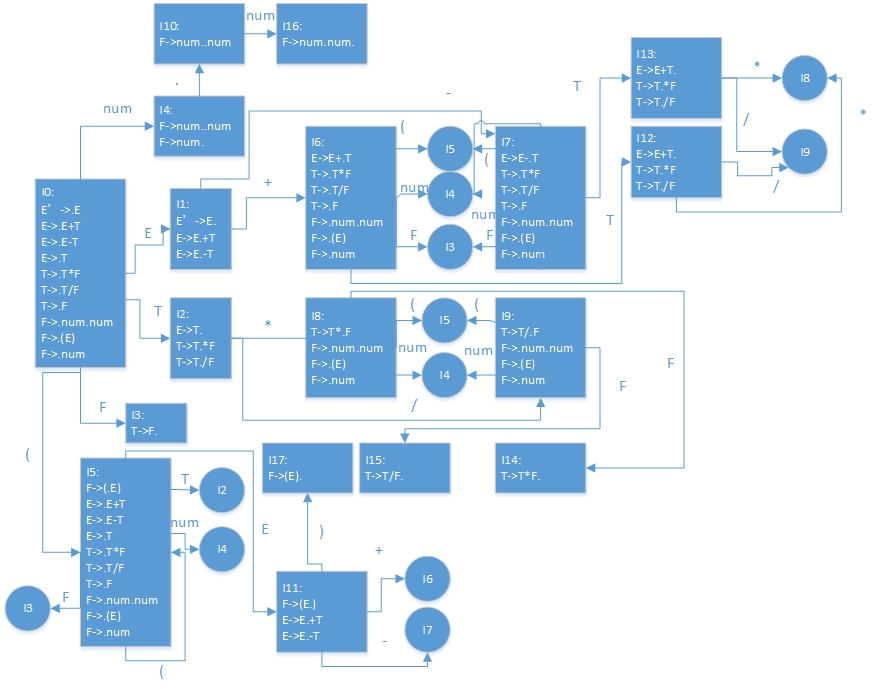
T -> F {T.val=F.val} {T.type = F.type}

F -> num1.num2 { F.val = num1.val + num2.val / 10} { F.type = real}

F -> (E) {F.val = E.val} {F.type = E.type}

F -> num {F.val = num.val} { F.type = integer}

1. 识别文法的所有活前缀的DFA



4．构造LR分析表（状态按照上面DFA定义，产生式标号按照上面拓广文法定义的标号）

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 状态 | + | - | \* | / | ( | ) | . | Num | $ | E | T | F |
| 0 |  |  |  |  | S5 |  |  | S4 |  | 1 | 2 | 3 |
| 1 | S6 | S7 |  |  |  |  |  |  | acc |  |  |  |
| 2 | R3 | R3 | S8 | S9 |  | R3 |  |  | R3 |  |  |  |
| 3 | R6 | R6 | R6 | R6 |  | R6 |  |  | R6 |  |  |  |
| 4 | R9 | R9 | R9 | R9 |  | R9 | S10 |  | R9 |  |  |  |
| 5 |  |  |  |  | S5 |  |  | S4 |  | 11 | 2 | 3 |
| 6 |  |  |  |  | S5 |  |  | S4 |  |  | 12 | 3 |
| 7 |  |  |  |  | S5 |  |  | S4 |  |  | 13 | 3 |
| 8 |  |  |  |  | S5 |  |  | S4 |  |  |  | 14 |
| 9 |  |  |  |  | S5 |  |  | S4 |  |  |  | 15 |
| 10 |  |  |  |  |  |  |  | S16 |  |  |  |  |
| 11 | S6 | S7 |  |  |  | S17 |  |  |  |  |  |  |
| 12 | R1 | R1 | S8 | S9 |  | R1 |  |  | R1 |  |  |  |
| 13 | R2 | R2 | S8 | S9 |  | R2 |  |  | R2 |  |  |  |
| 14 | R4 | R4 | R4 | R4 |  | R4 |  |  | R4 |  |  |  |
| 15 | R5 | R5 | R5 | R5 |  | R5 |  |  | R5 |  |  |  |
| 16 | R7 | R7 | R7 | R7 |  | R7 |  |  | R7 |  |  |  |
| 17 | R8 | R8 | R8 | R8 |  | R8 |  |  | R8 |  |  |  |

1. 源代码

#include<iostream>

#include<vector>

#include<string>

#include<cmath>

using namespace std;

int lex\_flag = 1;

string orig\_G[9] = {//原文法表达式

"E->E+T", "E->E-T", "E->T", "T->T\*F", "T->T/F", "T->F", "F->n.n", "F->(E)", "F->n"

};

char term\_table[9] = {//终结符表

'+', '-', '\*', '/', '(', ')', '.', 'n', '$'

};

char non\_term\_table[3] = {//非终结符表

'E', 'T', 'F'

};

string Action\_Table[18][9] = {//分析表ACTION部分

{"", "", "", "", "S5", "", "", "S4", ""},

{"S6", "S7", "", "", "", "", "", "", "ACC"},

{"R3", "R3", "S8", "S9", "", "R3", "", "", "R3"},

{"R6", "R6", "R6", "R6", "", "R6", "", "", "R6"},

{"R9", "R9", "R9", "R9", "", "R9", "S10", "", "R9"},

{"", "", "", "", "S5", "", "", "S4", ""},

{"", "", "", "", "S5", "", "", "S4", ""},

{"", "", "", "", "S5", "", "", "S4", ""},

{"", "", "", "", "S5", "", "", "S4", ""},

{"", "", "", "", "S5", "", "", "S4", ""},

{"", "", "", "", "", "", "", "S16", ""},

{"S6", "S7", "", "", "", "S17", "", "", ""},

{"R1", "R1", "S8", "S9", "", "R1", "", "", "R1"},

{"R2", "R2", "S8", "S9", "", "R2", "", "", "R2"},

{"R4", "R4", "R4", "R4", "", "R4", "", "", "R4"},

{"R5", "R5", "R5", "R5", "", "R5", "", "", "R5"},

{"R7", "R7", "R7", "R7", "", "R7", "", "", "R7"},

{"R8", "R8", "R8", "R8", "", "R8", "", "", "R8"}

};

int Goto\_Table[18][3] = {//分析表GOTO部分

{ 1, 2, 3 },

{ -1, -1, -1 },

{ -1, -1, -1 },

{ -1, -1, -1 },

{ -1, -1, -1 },

{ 11, 2, 3 },

{ -1, 12, 3 },

{ -1, 13, 3 },

{ -1, -1, 14 },

{ -1, -1, 15 },

{ -1, -1, -1 },

{ -1, -1, -1 },

{ -1, -1, -1 },

{ -1, -1, -1 },

{ -1, -1, -1 },

{ -1, -1, -1 },

{ -1, -1, -1 },

{ -1, -1, -1 }

};

int get\_term\_indx( char term )//终结符位置返回

{

for ( int i = 0; i < 9; i++ ){

if ( term\_table[i] == term )

return i;

}

}

int get\_nonterm\_indx( int nonterm )//非终结符位置返回

{

for ( int i = 0; i < 3; i++ ){

if ( non\_term\_table[i] == nonterm )

return i;

}

}

void lex\_error(int err\_kind)//词法分析过程错误信息输出

{

lex\_flag = 0;

if ( err\_kind == 1 ){

cout << "Invalid token." << endl;

}

else if ( err\_kind == 2 ){

cout << "Too much decimal number." << endl;

}

}

void grammar\_error()//语法分析过程错误信息输出

{

cout << "Unaccepted sentence."<< endl;

}

double calculate( int ptr, int num, vector<double> val, vector<char> type )//LR分析中计算表达式的值

{

double result;

int temp;

int length = 0;

int d = 10;

switch(num){

case 1:

result = val[ptr] + val[ptr-2];

break;

case 2:

result = val[ptr-2] - val[ptr];

break;

case 3:

result = val[ptr];

break;

case 4:

if ( type[ptr] == 'i' && type[ptr-2] == 'i')

result = (int)((int)val[ptr] \* (int)val[ptr-2]);

else

result = val[ptr] \* val[ptr-2];

break;

case 5:

if ( type[ptr] == 'i' && type[ptr-2] == 'i')

result = (int)((int)val[ptr-2] / (int)val[ptr]);

else{

result = val[ptr-2] / val[ptr];

}

break;

case 6:

result = val[ptr];

break;

case 7:

temp = (int)val[ptr];

while( temp != 0 ){

temp = temp/d;

length++;

}

result = val[ptr-2] + double(val[ptr])/pow((double)10, length);

break;

case 8:

result = val[ptr-1];

break;

case 9:

result = val[ptr];

break;

}

return result;

}

char kind ( int ptr, int num, vector<char> type )//LR分析中计算表达式类型

{

char result;

switch(num){

case 1:

if ( type[ptr] == 'i' && type[ptr-2] == 'i' )

result = 'i';

else

result = 'r';

break;

case 2:

if ( type[ptr] == 'i' && type[ptr-2] == 'i' )

result = 'i';

else

result = 'r';

break;

case 3:

result = type[ptr];

break;

case 4:

if ( type[ptr] == 'i' && type[ptr-2] == 'i' )

result = 'i';

else

result = 'r';

break;

case 5:

if ( type[ptr] == 'i' && type[ptr-2] == 'i' )

result = 'i';

else

result = 'r';

break;

case 6:

result = type[ptr];

break;

case 7:

result = 'r';

break;

case 8:

result = type[ptr-1];

break;

case 9:

result = 'i';

break;

}

return result;

}

int main()

{

string math\_sentence;

cout << "Please input the math sentence:"<< endl;

cin >> math\_sentence;

string buf;

vector<char> token;//保存词法分析中每个识别的字符

vector<char> bufa;//保存词法分析后的表达式结果

vector<int> buf\_val;//保存词法分析中数字的值

vector<int> state\_stack;//LR分析的状态栈

vector<double> val\_stack;//LR分析的数值栈

vector<char> type\_stack;//LR分析的类型栈

int ip = 0;

int top1 = 0;

int p\_num = 0;

int S;

char a;

buf = math\_sentence + "$";

cout << "Analysing process"<< endl;

int i = 0;

int state = 0;

int n1 = 0, n2 = 0;

int point = 0;

int start = 0;

while( i < buf.size() && lex\_flag == 1 ){//对算术表达式进行词法分析

switch(state){

case 0:

start = i;

if ( buf[i] >= '0' && buf[i] <= '9' ){//整数部分

bufa.push\_back('n');

token.push\_back(buf[i]);

state = 1;

}

else {

if ( buf[i] == '+' || buf[i] == '-' || buf[i] == '\*' || buf[i] == '/' || buf[i] == '(' || buf[i] == ')' || buf[i] == '$')

bufa.push\_back(buf[i]);

else{

lex\_error(1);

cout << "Error location: " << i+1 << endl;

}

state = 0;

}

break;

case 1:

if ( buf[i] >= '0' && buf[i] <= '9' ){

token.push\_back(buf[i]);

state = 1;

}

else if ( buf[i] == '.' ){

bufa.push\_back('.');

point = i - start;

state = 2;

}

else{

string num;

num.assign(token.begin(), token.end());

n1 = atoi(num.c\_str());

buf\_val.push\_back(n1);

i--;

token.clear();

state = 0;

}

break;

case 2:

if ( buf[i] >= '0' && buf[i] <= '9' ){//小数部分

bufa.push\_back('n');

token.push\_back(buf[i]);

state = 3;

}

else{

i--;

state = 0;

}

break;

case 3:

if ( buf[i] >= '0' && buf[i] <= '9' ){

token.push\_back(buf[i]);

state = 3;

}

else{

vector<char> num1;

vector<char> num2;

string num\_1;

string num\_2;

for ( int j = 0; j < point; j++ )

num1.push\_back(token[j]);

for ( int j = point; j < token.size(); j++ )

num2.push\_back(token[j]);

num\_1.assign(num1.begin(), num1.end());

num\_2.assign(num2.begin(), num2.end());

n1 = atoi(num\_1.c\_str());

n2 = atoi(num\_2.c\_str());

buf\_val.push\_back(n1);

buf\_val.push\_back(n2);

num1.clear();

num2.clear();

num\_1 = "";

num\_2 = "";

token.clear();

i--;

point = 0;

state = 0;

}

break;

}

i++;

}

cout << endl;

for ( int i = 0; i < bufa.size(); i++ ){

cout << bufa[i];

}

state\_stack.push\_back(0);

val\_stack.push\_back('-');

type\_stack.push\_back('\_');

do{//对词法分析后的表达式进行LR语法分析

S = state\_stack[top1];

a = bufa[ip];

if ( Action\_Table[S][get\_term\_indx(a)] == "" ){

grammar\_error();

break;

}

else if ( Action\_Table[S][get\_term\_indx(a)].at(0) == 'S' ){//移进动作

cout << "State: ";

for ( int i = 0; i < state\_stack.size(); i++ ){

cout << state\_stack[i]<< '|';

}

cout << '\t';

for ( int k = ip; k < bufa.size(); k++ ){

cout << bufa[k];

}

cout << '\t';

string entry = Action\_Table[S][get\_term\_indx(a)];

string next\_state = entry.substr(1);

int S1 = atoi(next\_state.c\_str());

cout << "Shift "<< S1;

cout << endl;

cout << endl;

state\_stack.push\_back(S1);

if ( bufa[ip] == 'n' ){

val\_stack.push\_back(buf\_val[p\_num]);

type\_stack.push\_back('i');

p\_num++;

}

else{

val\_stack.push\_back(0);

type\_stack.push\_back('#');

}

top1++;

ip++;

}

else if ( Action\_Table[S][get\_term\_indx(a)].at(0) == 'R' ){//规约动作

cout << "State: ";

for ( int i = 0; i < state\_stack.size(); i++ ){

cout << state\_stack[i]<< '|';

}

cout << '\t';

for ( int k = ip; k < bufa.size(); k++ ){

cout << bufa[k];

}

cout << '\t';

string entry = Action\_Table[S][get\_term\_indx(a)];

string regress\_sntns = entry.substr(1);

int R = atoi(regress\_sntns .c\_str());

string pop\_symble = orig\_G[R-1].substr(3);

int pop\_length = pop\_symble.length();

cout << "Reduce by "<< orig\_G[R-1];

cout << endl;

cout << endl;

double result;

char sen\_type;

result = calculate(top1, R, val\_stack, type\_stack);

sen\_type = kind(top1, R, type\_stack);

for ( int i = 0; i < pop\_length; i++ ){

state\_stack.pop\_back();

val\_stack.pop\_back();

type\_stack.pop\_back();

top1--;

}

//

int S2 = state\_stack[top1];

char next\_symble = orig\_G[R-1].at(0);

int newS = Goto\_Table[S2][get\_nonterm\_indx(next\_symble)];

state\_stack.push\_back(newS);

val\_stack.push\_back(result);

type\_stack.push\_back(sen\_type);

top1++;

}

else if ( Action\_Table[S][get\_term\_indx(a)] == "ACC" ){

cout << "State: ";

for ( int i = 0; i < state\_stack.size(); i++ ){

cout << state\_stack[i] << '|';

}

cout << '\t';

for ( int k = ip; k < bufa.size(); k++ ){

cout << bufa[k];

}

cout << '\t';

cout << "ACC";

cout << endl;

break;

}

}while(1);

cout << "The result of the math sentence is: "<< val\_stack[top1]<< endl;

cout << "The type of the math sentence is: " << type\_stack[top1]<< endl;

system("pause");

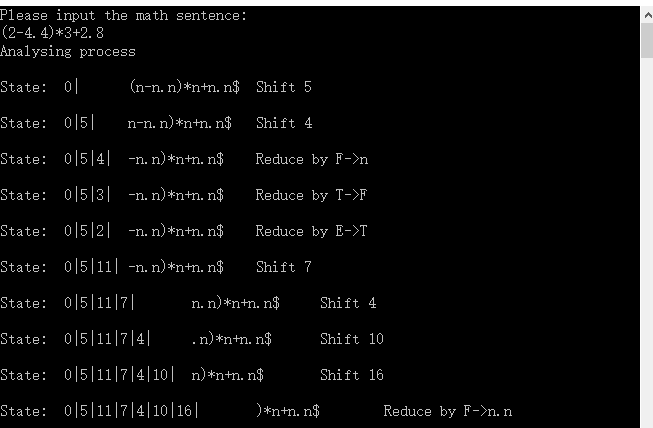
return 0;

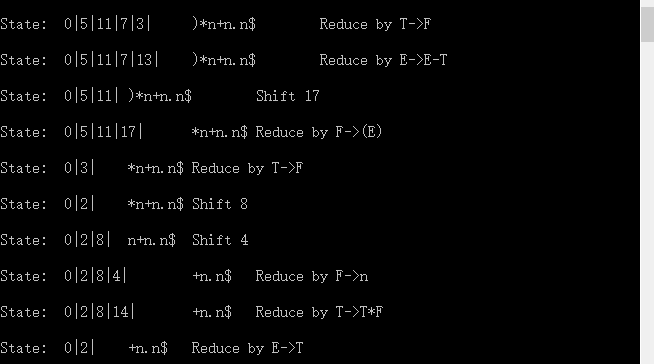
}

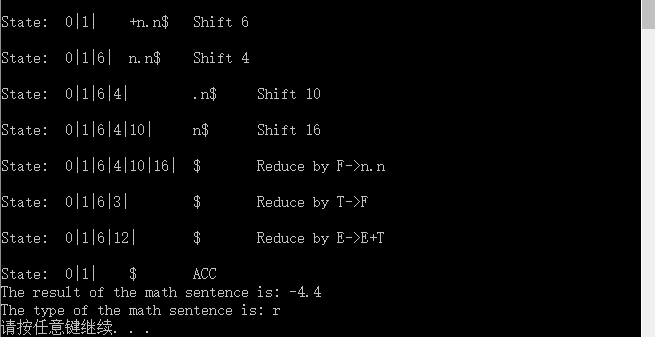
1. 测试及结果

输出结果为栈变化、输入串以及相应的分析动作。

1. 输入符号串合法的结果

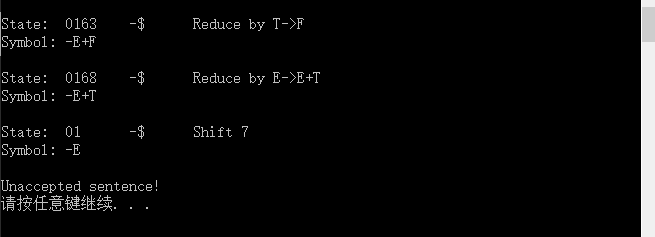






2．输入符号串不合法的结果

![C:\Users\lenovo\AppData\Roaming\Tencent\Users\2082445507\QQ\WinTemp\RichOle\_}S4](D](552${O0Q3{AJY0.png](data:image/png;base64,)



在分析错误时终止程序，输出不可接受的信息。

