程序设计2 语法分析程序的设计与实现

实验报告

一，实验题目与要求

题目：语法分析程序的设计与实现

实验内容：编写语法分析程序，实现对算数表达式的语法分析。要求所分析的算数表达式由如下的文法产生。

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

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

F -> (E) | num

实验要求：在对输入表达式进行分析的过程中，输入所采用的产生式。

方法1：编写递归调用程序实现自顶向下的分析。

方法2：编写LL(1)语法分析程序，要求如下。

（1）编程实现算法4.2，为给定文法自动构造预测分析表

（2）编程实现算法4.1，构造LL(1)预测分析程序

方法3：编写语法分析程序实现自底向上的分析，要求如下。

（1）构造识别所有活前缀的DFA

（2）构造LR分析表

（3）编程实现算法4.3，构造LR分析程序。

方法4：利用YACC自动生成语法分析程序，调用LEX自动生成的词法分析程序

二，实验分析

本次实验使用了方法1和方法4：

方法1：编写递归调用程序实现自顶向下的分析：

源代码：

#include<stdio.h>

#include<string.h>

char Exp[30],gra[30],prod[30]="",chExp='#';

int expSize=0,graSize=0,step=0;

/\*

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

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

F -> (E)|num

带有左递归，将左递归消除之后得到

E -> T E'

T -> F T'

F -> ( E ) | num

E' -> epsilon | + T E' | - T E'

T' -> epsilon | \* F T' | / F T'

\*/

void printGrammar()

{

printf("\t\t\t递归下降分析程序\n\t\t\n");

printf("-----------------------------------------------------------\n");

printf("\t\t\t (1)E->E+T|E-T|T\n");

printf("\t\t\t (2)T->T\*F|T/F|F\n");

printf("\t\t\t (3)F->(E)|num\n");

printf("将上述产生式消除左递归之后的产生式如下：\n");

printf("\t\t\t (1)E->TA\n");

printf("\t\t\t (2)T->FB\n");

printf("\t\t\t (3)F->(E)|num\n");

printf("\t\t\t (4)A->epsilon|+TA|-TA\n");

printf("\t\t\t (5)B->epsilon|\*FB|/FB\n");

printf("-----------------------------------------------------------\n");

}

void GetExp()

{

printf("请输入表达式:（以#结束）\t");

gets(Exp);

expSize=strlen(Exp);

chExp=Exp[0];

printf("-----------------------------------------------------------\n");

}

void printHead()

{

printf("步骤：\t 语法栈：\t\t输入串：\t产生式：\n");

}

void printStep()

{

printf("%d\t%-20s %10s\t \t%-15s\n",step,gra,Exp,prod);

strcpy(prod,"");

step++;

if(chExp=='#'&&gra[graSize-1]=='#')

{

printf("\n表达式分析成功！\n");

}

}

void pushGraStack(char \* ch)

{

int i;

for(i=0;i<strlen(ch);i++)

{

gra[graSize]=ch[strlen(ch)-1-i];

graSize++;

}

}

char popGraStack()

{

char ch;

ch=gra[graSize-1];

gra[graSize-1]='\0';

graSize--;

return ch;

}

void nextChar()

{

int i=0;

for(i=0;i<expSize-1;i++)

{

Exp[i]=Exp[i+1];

}

Exp[expSize-1]='\0';

expSize--;

chExp=Exp[0];

printf("当前chExp=:%c\n",chExp);

}

void InitGra()

{

gra[graSize]='#';

graSize++;

gra[graSize]='E';

graSize++;

printStep();

}

void printError()

{

printf("\n表达式不匹配！\n");

}

/\*

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

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

F -> (E)|num

带有左递归，将左递归消除之后得到

E -> T A

T -> F B

F -> ( E ) | num

A -> epsilon | + T A | - T A

B -> epsilon | \* F B | / F B

\*/

int E()

{

popGraStack();

char graE[]="TA";

pushGraStack(graE);

strcpy(prod,"E->TA");

printStep();

T();

A();

return 1;

}

int T()

{

popGraStack();

char graT[]="FB";

pushGraStack(graT);

strcpy(prod,"T->FB");

printStep();

F();

B();

return 1;

}

int F()

{

if(chExp=='(')

{

popGraStack();

char graF[]="(E)";

pushGraStack(graF);

strcpy(prod,"F->(E)");

printStep();

popGraStack();

nextChar();

strcpy(prod,"匹配");

printStep();

E();

if(chExp==')')

{

popGraStack();

nextChar();

strcpy(prod,"匹配");

printStep();

return 1;

}

else

{

printError();

return 0;

}

}

else if(chExp<='9' && chExp>='0')

{

popGraStack();

char graF[]={chExp,'\0'};

pushGraStack(graF);

strcpy(prod,"F->");

strcat(prod,graF);

printStep();

popGraStack();

nextChar();

strcpy(prod,"匹配");

printStep();

return 1;

}

else

{

printError();

return 0;

}

}

/\*

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

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

F -> (E)|num

带有左递归，将左递归消除之后得到

E -> T A

T -> F B

F -> ( E ) | num

A -> epsilon | + T A | - T A

B -> epsilon | \* F B | / F B

\*/

int A()

{

if(chExp=='+'||chExp=='-')

{

popGraStack();

char graA[]={chExp,'T','A','\0'};

pushGraStack(graA);

strcpy(prod,"A->");

strcat(prod,graA);

printStep();

popGraStack();

nextChar();

strcpy(prod,"匹配");

printStep();

T();

A();

return 1;

}

else

{

strcpy(prod,"A->epsilon");

printStep();

popGraStack();

strcpy(prod,"匹配");

printStep();

return 1;

}

}

int B()

{

if(chExp=='\*'||chExp=='/')

{

popGraStack();

char graB[]={chExp,'T','B','\0'};

pushGraStack(graB);

strcpy(prod,"B->");

strcat(prod,graB);

printStep();

popGraStack();

nextChar();

strcpy(prod,"匹配");

printStep();

T();

B();

return 1;

}

else

{

strcpy(prod,"B->epsilon");

printStep();

popGraStack();

strcpy(prod,"匹配");

printStep();

return 1;

}

}

int main()

{

printGrammar();

GetExp();

printHead();

InitGra();

E();

return 0;

}

说明：因为原文法存在左递归，无法进行语法分析，因此人为的将文法消除左递归，编写出上述源代码。

程序运行结果：

测试结果一：

递归下降分析程序

-----------------------------------------------------------

(1)E->E+T|E-T|T

(2)T->T\*F|T/F|F

(3)F->(E)|num

将上述产生式消除左递归之后的产生式如下：

(1)E->TA

(2)T->FB

(3)F->(E)|num

(4)A->epsilon|+TA|-TA

(5)B->epsilon|\*FB|/FB

-----------------------------------------------------------

请输入表达式:（以#结束） (5+6)/(9-2)#

-----------------------------------------------------------

步骤： 语法栈： 输入串： 产生式：

0 #E (5+6)/(9-2)#

1 #AT (5+6)/(9-2)# E->TA

2 #ABF (5+6)/(9-2)# T->FB

3 #AB)E( (5+6)/(9-2)# F->(E)

当前chExp=:5

4 #AB)E 5+6)/(9-2)# 匹配

5 #AB)AT 5+6)/(9-2)# E->TA

6 #AB)ABF 5+6)/(9-2)# T->FB

7 #AB)AB5 5+6)/(9-2)# F->5

当前chExp=:+

8 #AB)AB +6)/(9-2)# 匹配

9 #AB)AB +6)/(9-2)# B->epsilon

10 #AB)A +6)/(9-2)# 匹配

11 #AB)AT+ +6)/(9-2)# A->+TA

当前chExp=:6

12 #AB)AT 6)/(9-2)# 匹配

13 #AB)ABF 6)/(9-2)# T->FB

14 #AB)AB6 6)/(9-2)# F->6

当前chExp=:)

15 #AB)AB )/(9-2)# 匹配

16 #AB)AB )/(9-2)# B->epsilon

17 #AB)A )/(9-2)# 匹配

18 #AB)A )/(9-2)# A->epsilon

19 #AB) )/(9-2)# 匹配

当前chExp=:/

20 #AB /(9-2)# 匹配

21 #ABT/ /(9-2)# B->/TB

当前chExp=:(

22 #ABT (9-2)# 匹配

23 #ABBF (9-2)# T->FB

24 #ABB)E( (9-2)# F->(E)

当前chExp=:9

25 #ABB)E 9-2)# 匹配

26 #ABB)AT 9-2)# E->TA

27 #ABB)ABF 9-2)# T->FB

28 #ABB)AB9 9-2)# F->9

当前chExp=:-

29 #ABB)AB -2)# 匹配

30 #ABB)AB -2)# B->epsilon

31 #ABB)A -2)# 匹配

32 #ABB)AT- -2)# A->-TA

当前chExp=:2

33 #ABB)AT 2)# 匹配

34 #ABB)ABF 2)# T->FB

35 #ABB)AB2 2)# F->2

当前chExp=:)

36 #ABB)AB )# 匹配

37 #ABB)AB )# B->epsilon

38 #ABB)A )# 匹配

39 #ABB)A )# A->epsilon

40 #ABB) )# 匹配

当前chExp=:#

41 #ABB # 匹配

42 #ABB # B->epsilon

43 #AB # 匹配

44 #AB # B->epsilon

45 #A # 匹配

46 #A # A->epsilon

47 # # 匹配

表达式分析成功！

测试结果二：

递归下降分析程序

-----------------------------------------------------------

(1)E->E+T|E-T|T

(2)T->T\*F|T/F|F

(3)F->(E)|num

将上述产生式消除左递归之后的产生式如下：

(1)E->TA

(2)T->FB

(3)F->(E)|num

(4)A->epsilon|+TA|-TA

(5)B->epsilon|\*FB|/FB

-----------------------------------------------------------

请输入表达式:（以#结束） (5+7)/5+#

-----------------------------------------------------------

步骤： 语法栈： 输入串： 产生式：

0 #E (5+7)/5+#

1 #AT (5+7)/5+# E->TA

2 #ABF (5+7)/5+# T->FB

3 #AB)E( (5+7)/5+# F->(E)

当前chExp=:5

4 #AB)E 5+7)/5+# 匹配

5 #AB)AT 5+7)/5+# E->TA

6 #AB)ABF 5+7)/5+# T->FB

7 #AB)AB5 5+7)/5+# F->5

当前chExp=:+

8 #AB)AB +7)/5+# 匹配

9 #AB)AB +7)/5+# B->epsilon

10 #AB)A +7)/5+# 匹配

11 #AB)AT+ +7)/5+# A->+TA

当前chExp=:7

12 #AB)AT 7)/5+# 匹配

13 #AB)ABF 7)/5+# T->FB

14 #AB)AB7 7)/5+# F->7

当前chExp=:)

15 #AB)AB )/5+# 匹配

16 #AB)AB )/5+# B->epsilon

17 #AB)A )/5+# 匹配

18 #AB)A )/5+# A->epsilon

19 #AB) )/5+# 匹配

当前chExp=:/

20 #AB /5+# 匹配

21 #ABT/ /5+# B->/TB

当前chExp=:5

22 #ABT 5+# 匹配

23 #ABBF 5+# T->FB

24 #ABB5 5+# F->5

当前chExp=:+

25 #ABB +# 匹配

26 #ABB +# B->epsilon

27 #AB +# 匹配

28 #AB +# B->epsilon

29 #A +# 匹配

30 #AT+ +# A->+TA

当前chExp=:#

31 #AT # 匹配

32 #ABF # T->FB

表达式不匹配！

33 #ABF # B->epsilon

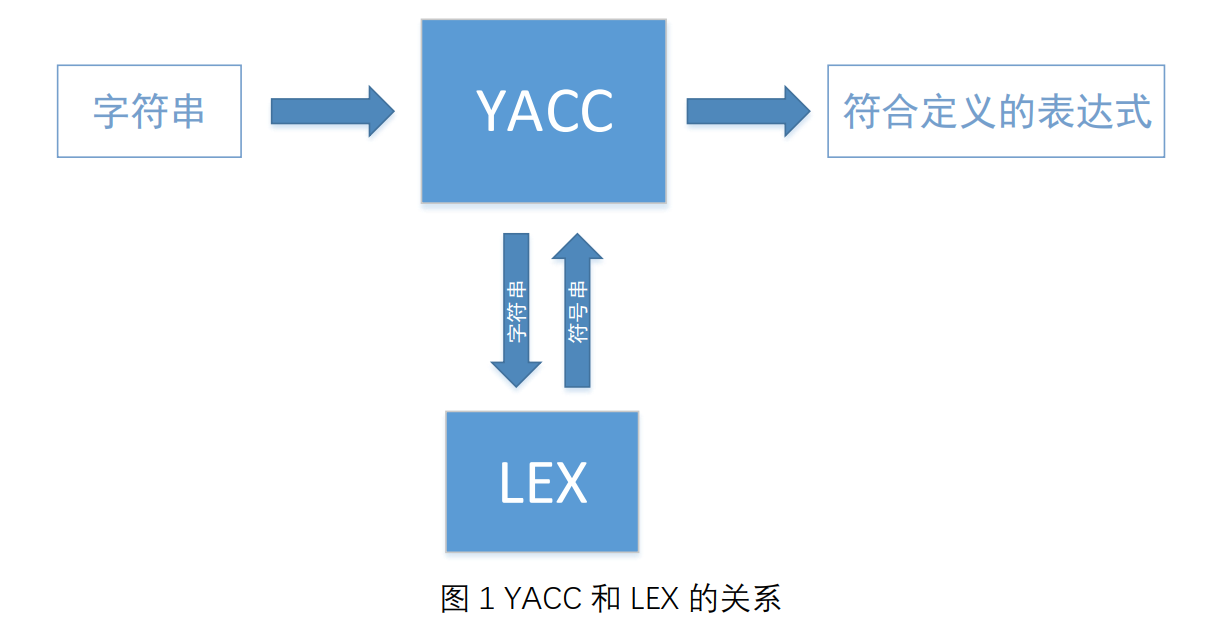
34 #AB # 匹配

35 #AB # A->epsilon

36 #A # 匹配

表达式分析不成功！

方法4：使用YACC语法分析程序生成器，其中词法分析调用了LEX词法分析程序。通过定义翻译规则，YACC使用LALR文法进行规约。通过LEX将读取到的字符串分析为符号串，再对符号串按照定义的表达式进行规约，判断输入的字符串是否符合表达式定义。



源程序：

YACC程序：

%{

#include <ctype.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define YYSTYPE double /\*double type for YACC stack\*/

int yylex(void);

void yyerror(char \* s);

int yywrap(void);

%}

%token NUM

%token ID

%left '+' '-'

%left '\*' '/'

%right UMINUS

%%

lines : lines expr '\n' {printf("Result: %f\n",$2);}

|lines '\n'

| /\* empty \*/

|error '\n' {yyerror("Retry please.");

yyerrok;}

;

expr : expr '+' term {$$ = $1 + $3;printf("E->E+T\n");}

|expr '-' term {$$ = $1 - $3;printf("E->E-T\n");}

|term {printf("E->T\n");}

;

term : term '\*' form {$$ = $1 \* $3;printf("T->T\*F\n");}

|term '/' form {$$ = $1 / $3;printf("T->T/F\n");}

|form {printf("T->F\n");}

;

form : '(' expr ')' {$$ = $2;printf("F->(E)\n");}

|NUM {printf("F->num\n");}

;

%%

#include "lex.yy.c"

void yyerror(char \* s){

printf("%s\n",s);

}

int main (){

return yyparse();

}

LEX程序：

%option noyywrap

%{

#include "y.tab.h"

char\* id[100];

double num[100];

int count=0;

extern double yylval;

double install\_id();

int yywarp (void);

%}

space [' ']\*

wrap [\n]

letter [A-Za-z]

digit [0-9]

id {letter}({letter}|{digit})\*

num {digit}+(\.{digit}+)?

%%

{id} {yylval=install\_id();return (ID);}

{num} {yylval=atof(yytext);return (NUM);}

{space} {printf(" ");}

{wrap} {return '\n';}

. {return yytext[0];}

%%

double install\_id(){

int i=0;

for(i=0;i<count;i++){

if(strcmp(yytext,id[i])==0){

return num[i];

}

}

printf("Found an unassigned id:\n%s = ",yytext);

id[count]=(char\*)malloc(sizeof(char)\*yyleng);

strcpy(id[count],yytext);

scanf("%lf", &num[count]);

count++;

return num[count-1];

}

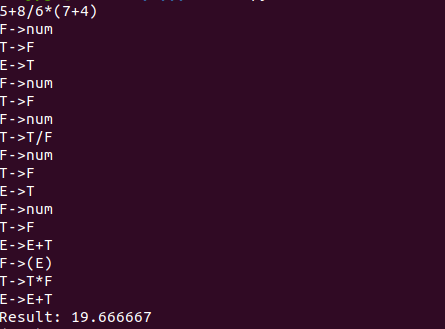
int yywarp (void){

return 1;

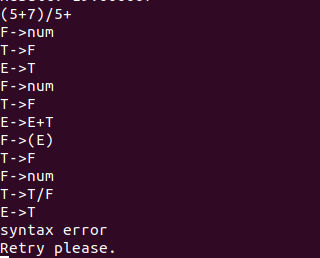
}

程序运行结果：

测试结果一：



测试结果二：



三，实验总结：

这次实验通过编写两种方法实现的语法分析程序以完成对算数表达式的语法分析。经测试，设计出的程序能够准确识别表达式设定的语法，并对错误表达式进行反馈。这次实验加强了我对预测分析程序和LALR文法的理解，同时由于YACC输出语法实际上是通过在翻译方案中设计输出语句进行输出，因此这次实验也涉及到了语法制导翻译的部分知识，设计出的程序还有需要改进的地方，例如在遇到错误表达式时进行一定的错误恢复并提示错误类型，扩充更多表达式等等。