程序设计 3 语义分析程序的设计与实现
实验报告

班级：-------- 姓名：-------- 学号：--------

一、 实验题目与要求

题目：语义分析程序的设计与实现。

实验内容：编写语义分析程序，实现对算术表达式的类型检查和求值。要求所分析算术表达式由如下的文法产生。

$$E \to E+T \mid E-T \mid T$$
$$T \to T*F \mid T/F \mid F$$
$$F \to num.num \mid (E) \mid num$$

实验要求：用自底向上的语法制导翻译技术实现对表达式的分析和翻译。

(1) 写出满足要求的语法制导定义或翻译方案。

(2) 编写分析程序，实现对表达式的类型进行检查和求值，并输出：

① 分析过程中所有产生式。

② 识别出的表达式的类型。

③ 识别出的表达式的值。

(3) 实验方法：可以选用以下两种方法之一。

① 自己编写分析程序。

② 利用 YACC 自动生成工具。

二、 实验分析

本次实验使用 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，在实验 2 的基础上进行扩展。首先文法中删去了 id，因此在 lex 语法中删去关于 id 的识别，同时在 yacc 语法中添加了对 num.num 这一文法的识别。由于要通过自底向上的语法制导翻译技术来实现对表达式类型的检查以及求值，因此必须对识别出的符号的属性进行扩充，由原先的 value 扩充到 value 和 type，即定义第二个综合属性 E.type。这需要将原先 YYSTYPE 的类型由 double 扩充为 struct，在 struct 中包含 value 和 type 两个属性。然后对表达式中的翻译方案进行修改，根据不同表达式决定产生式的类型。

因此首先要分析文法的翻译方案。

由于文法中只有 num 和 num.num，因此本文法涉及的类型只有 int 和 real。又因为 int 可以隐式类型转换成 real 类型，因而本文法中重点考虑的是类型转换而非之前定义 id 时需要考虑的判断类型是否合法；再加上文法全部是规约至标识符的语句，所以判断类型即判断标识符的类型。由分析可知，F->num.num 得到的是 real 类型，F->num 得到的是 int 类型，而在计算中，当两个标识符都是 int 型时，得到结果的标识符为 int 型，而存在一个标识符为 real 型，另一个标识符为 int 型时，后者需要隐式类型转换为 real 型，得到的结果也就是 real 型了。

得到的翻译方案如下：

$E \rightarrow E_1+T\{E.val=E_1.val+T.val;$

$print("E \rightarrow E+T");$

$if(E_1.type=real\ or\ T.type = real)E.type = real;$

$else\ E.type = int;\}$

$E\rightarrow E_1\text{-}T\{E.val=E_1.val\text{-}T.val;print("E\rightarrow E - T")\}$

$if(E_1.type=real\ or\ T.type = real)E.type = real;$

$else\ E.type = int;\}$

$E\rightarrow T\{E.val=T.val;print("E\rightarrow T");E.type=T.type\}$

$T\rightarrow T_1*F\{T.val=T_1.val*F.val;print("T\rightarrow T*F")$

$if(E_1.type=real\ or\ T.type = real)E.type = real;$

$else\ E.type = int;\}$

$T\rightarrow T_1/F\{T.val=T_1.val/F.val;print("T\rightarrow T/F")\}$

$if(E_1.type=real\ or\ T.type = real)T.type=real;$

$else\ T.type=int;\}$

$T\rightarrow F\{T.val=F.val;print("T\rightarrow F");T.type=F.type;\}$

$F\rightarrow num.num\{F.val=id.val;print("F\rightarrow id");F.type=real;\}$

$F\rightarrow(E)\{F.val=E.val;print("F\rightarrow(E)");F.type=E.type;\}$

$F \rightarrow num\{F.val=num.val;print("F \rightarrow num");F.type=int;\}$

通过将实验 2 中各标识符属性和翻译方案进行扩充，可以得到新的符合要求的翻译方案用于分析表达式的结果和类型。原先设计的文法中只涉及 E.value 属性，因此在 YACC 语法中属性的类型为 double；在本次设计的翻译方案中涉及 E.value 和 E.type 两个综合属性，因此属性的类型需设置为 struct，在 struct 中再定义 value 和 type 两个属性。Type 为方便起见直接设置为 int 型，0 代表整型(int)，1 代表实数(real)。

三、 源程序

YACC 程序：

```
%{
#include <ctype.h>
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <math.h>
struct exp{
   double value;
   int type;
};
int length;
#define YYSTYPE struct exp

int yylex(void);
void yyerror(char * s);
int yywrap(void);

%}

%token NUM
%left '+' '-'
%left '*' '/'
%right UMINUS
%%


lines : lines expr '\n' {if($2.type == 0){
                        printf("Type: Int\n");
                        printf("Result: %d\n",(int)$2.value);
                    }
                    else {
                        printf("Type: Real\n");
                        printf("Result: %f\n",$2.value);
                    }}
   |lines '\n'
   |      /* empty */
   |error '\n' {yyerror("Retry please.");
                yyerrok;}
   |'-' expr %prec UMINUS {$$.type = $2.type;
                           $$.value = -$2.value;
                           printf("E->-E\n");}
   ;
expr : expr '+' term {if($1.type == 0 && $3.type == 0)
                           $$.type = 0;
                    else
                           $$.type = 1;
                    $$.value = $1.value + $3.value;
```

```
                    printf("E->E+T\n");}
   |expr '-' term {if($1.type == 0 && $3.type == 0)
                       $$.type = 0;
                   else
                       $$.type = 1;
                   $$.value = $1.value - $3.value;
                   printf("E->E-T\n");}
   |term {$$.type = $1.type;
          $$.value = $1.value;
          printf("E->T\n");}
   ;
term : term '*' form {if($1.type == 0 && $3.type == 0)
                       $$.type = 0;
                   else
                       $$.type = 1;
                   $$.value = $1.value * $3.value;
                   printf("T->T*F\n");}
   |term '/' form {if($1.type == 0 && $3.type == 0){
                       $$.type = 0;
                       $$.value = (int)($1.value / $3.value);
                   }
                   else{
                       $$.type = 1;
                       $$.value = $1.value / $3.value;
                   }
                   printf("T->T/F\n");}
   |form {$$.type = $1.type;
          $$.value = $1.value;
          printf("T->F\n");}
   ;
form : NUM '.' NUM {$$.type = 1;
                   $$.value = $1.value + $3.value /
                              pow(10,(double)length);
                   printf("F->num.num\n");}
   |'(' expr ')' {$$.type = $2.type;
                  $$.value = $2.value;printf("F->(E)\n");}
   |NUM {$$.type = 0;
         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 "first.tab.h"
char* id[100];
double num[100];
int count=0;

double install_id();
int yywarp (void);

%}
space      [' ']*
wrap       [\n]
letter     [A-Za-z]
digit      [0-9]
num        {digit}+


%%
{num}         {yylval.value=atof(yytext);
               length=strlen(yytext);return (NUM);}
{space}       {printf(" ");}
{wrap}        {return '\n';}
.             {return yytext[0];}
%%

int yywarp (void){
   return 1;
}
```

四、 验证结果

输入：6+5/3
输出： F->num
T->F

```
E->T
F->num
T->F
F->num
T->T/F
E->E+T
Type: Int
Result: 7
```

输入：6+5.0/3.0
输出：

```
F->num
T->F
E->T
F->num.num
T->F
F->num.num
T->T/F
E->E+T
Type: Real
Result: 7.666667
```

当需要隐式类型转换时：
输入：75/(4+3*3.0)
输出：

```
F->num
T->F
F->num
T->F
E->T
F->num
T->F
F->num.num
T->T*F
E->E+T
F->(E)
T->T/F
E->T
Type: Real
Result: 5.769231
```

输入错误表达式时：
输入：5+4/(5+7
输出：

```
F->num
T->F
E->T
```

```
F->num
T->F
F->num
T->F
E->T
F->num
T->F
E->E+T
syntax error
```

五、 实验总结

本次实验通过编写语义分析程序，实现对算术表达式的类型检查和求值，加深了我们对于语法制导翻译和语义分析的理解。经测试，设计出的程序能够正确地按照文法识别相应表达式并对其进行表达式类型判断和求值。此程序还有可以继续推进的地方，例如可以加入第二次实验中涉及的 id，引入其他数据类型并加入类型判断。