3. 语法分析

语法分析程序以词法分析输出的符号串作为输入，在分析过程中检查这个符号串是否为该程序语言的句子。若是，则输出该句子的分析树；否则就表示源程序存在语法错误，需要报告错误的性质和位置。

常用的语法分析大体上可以分成自顶向下和自底向上两大类：

（1） 自顶向下方法。语法分析从顶部（分析树的根节点）到底部（语法树的叶节点），为输入的符号串建立分析树。该方法又进一步分为递归下降的方法和非递归的 LL 分析方法。

（2） 自底向上方法。语法分析从底部到顶部为输入的符号串建立分析树，该方法通常也叫做 LR 分析方法。

本章将为 TEST 语言编写自顶向下方法中的递归下降方法。

3.1 TEST 语言的递归下降分析实现

TEST 语言的语法规则如下：

(1) <program>→{ <declaration_list><statement_list> }

(2) <declaration_list> → <declaration_list><declaration_stat> | ε

(3) <declaration_stat> → int ID;

(4) <statement_list>→<statement_list><statement> | ε

(5) <statement> → <if_stat> | <while_stat> | <for_stat> | <read_stat>
| <write_stat> | <compound_stat> | <expression_stat>

(6) <if_stat> → if(<expression>) <statement> [else <statement>]

(7) <while_stat> → while(<expression>) <statement>

(8) <for_stat>→ for(<expression>;<expression>;<expression>) <statement>

(9) <write_stat>→write <expression>;

(10) <read_stat>→read ID;

(11) <compound_stat>→{<statement_list>}

(12) <expression_stat>→<expression>; | ;

(13) <expression>→ID=<bool_expr> | <bool_expr>

(14) <bool_expr>→<additive_expr>

| <additive_expr>(>|<|>=|<=|==|!=)<additive_expr>

(15) <additive_expr>→<term>{(+|-)<term>}

(16) <term>→<factor>{(*|/)<factor>}

(17) <factor>→(<expression>) | ID | NUM

其中,规则(1)和规则(11)中的符号 “{” 和 “}” 为终结符号,不是元符号,而规则(6)-(8)和(17)中出现的符号 “(” 和 “)” 也是终结符号,不是元符号。

分析程序设计如下：

针对每一条规则,分别来设计其递归下降分析过程。TEST 语言的语法规则基本符合递归下降的要求,但对于规则(13)

<expression>→ ID=<bool _expr> | <bool_expr>

因为<boolexpr>的首符号可能是 ID,即存在首符号集相交问题,而此时,不可能将布尔表达式规则代入,所以,在程序设计时,通过超前读一个符号来解决。方法是:如果识别出标识符的符号 ID 后,再读一个符号,如果这个符号是=,说明选择的

是赋值表达式;如果不是=,则说明选择的是布尔表达式。其实现程序如下:

```
//<表达式> → <标识符>=<布尔表达式><布尔表达式>
//<expr> → ID=<bool _expr> | <bool_expr>
int expression()
{
    int es=0,fileadd;
    char token2 [20], token3[40];
    if (strcmp (token, "ID")==0)
    {
        fileadd=ftell(fp); //记住当前文件位置
        fscanf(fp,"%s %s\n", &token2, &token3);
        printf("%s %s \n", token2, token3);
        if(strcmp (token2, "=")==0) //赋值表达式
        {
            fscanf (fp, "%s %s\n", &token, &token1);
            printf("%s %s\n", token, token1);
            es=bool_expr();
            if(es>0)return(es);
        }else
        {
            fseek(fp, fileadd, 0); //若非=，则文件指针回到=前的标识符
            printf("%s $s\n", token, token1);
```

```
                es=bool expr();
                if(es>0) return(es);
            }else es=bool_expr();
        }
        return(es);
    }
```

在语法分析程序的实现中,对应每条规则的分析函数的取名与规则中的符号同名。语法分析程序的名为 TESTparse(),在这个函数里,调用对应于规则(1)的分析函数 program()开始进行语法分析,其他规则的分析函数会从函数 program()中递归调用。每个分析函数都有返回值,当返回值为 0 时,表示这个函数的分析没发现错误;如果返回值大于 0,则有错误。该语法分析程序没有进行错误处理,一旦发现错误立即返回,并报告错误信息。

语法分析程序运行,首先调用词法分析程序,请求输入 TEST 源程序的文件名以及词法分析输出文件名,接着执行语法分析。如果输入的 TEST 源程序没有语法错误,则显示语法分析成功;如果有错误,则该语法分析程序遇到错误时立即停止分析,并报告错误信息。

该递归下降的分析程序可以按如下开始（也可以按自己的方式写）:

```
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
int TESTparse();
```

```
int program();
int compound_stat();
int statement();
int expression_stat();
int expression();
int bool_expr();
int additive_expr();
int term();
int factor();
int if_stat();
int while_stat();
int for_stat();
int write_stat()_;
int read_stat();
int declaration_stat();
int declaration_list();
int statement_list();
int compound_list();
char token[20], token1[40];
extern char Scanout[300];
FILE * fp;
//语法分析程序
```

```
int TESTparse()
{
    int es = 0;
    if( (fp = fopen(Scanout, “r” ))==NULL)
    {
        printf( “\n Can not open %S\n” , Scanout);
        es = 10;
    }
if(es == 0) es = program();
printf( “==语法分析结果==\n” );
switch(es)
{
    case 0: printf( “语法分析成功\n” ); break;
    case 10: printf( “打开文件%s 失败” , Scanout); break;
    case 1: printf( “\n” ); break;
    case 2: printf( “\n” ); break;
    case 3: printf( “\n” ); break;
    case 4: printf( “\n” ); break;
    case 5: printf( “\n” ); break;
    case 6: printf( “\n” ); break;
    case 7: printf( “\n” ); break;
}
```

```
fclose(fp);
return(es);
//<program>→{<declaration_list><statement_list>}
int program()
{ … }
…
//主程序
#include<stdio.h>
#include<ctype.h>
extern int TESTscan();
extern int TESTparse();
char Scanin[300], Scanout[300];
FILE * fin, * fout;
void main() {
    int es =0;
    es = TESTscan();
    if(es > 0) printf( “词法分析有错，编译终止！\n” );
    else printf( “词法分析成功！\n” ) ;
    if(es == 0)
    {
        es = TESTparse();
        if(es > 0) printf( “语法分析有错，编译终止！\n” );
```

```
        else printf( “语法分析成功！\n” ) ;
    }
}
```