

# 苏州大学实验报告

院、系	计算机科学与技术学院	年级专业	21 计科	姓名	赵鹏	学号	2127405037
课程名称	编译原理实践					成绩	
指导教师	段湘煜	同组实验者	无	实验日期	2023.10.16		

实验名称 语法分析

## 一. 实验题目

TEST 语言的语法分析

基于 TEST 语言的语法规则，编写程序利用递归下降的方法实现对 TEST 语言的语法分析。

## 二. 实验原理及流程框

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。

本次实验为 TEST 语言编写自顶向下方法中的递归下降的分析方法。

TEST语言的语法规则如下：

- (1)  $\langle program \rangle \rightarrow \{ \langle declaration\_list \rangle \langle statement\_list \rangle \}$
- (2)  $\langle declaration\_list \rangle \rightarrow \langle declaration\_list \rangle \langle declaration\_stat \rangle \mid \varepsilon$
- (3)  $\langle declaration\_stat \rangle \rightarrow int\ ID;$
- (4)  $\langle statement\_list \rangle \rightarrow \langle statement\_list \rangle \langle statement \rangle \mid \varepsilon$
- (5)  $\langle statement \rangle \rightarrow \langle if\_stat \rangle \mid \langle while\_stat \rangle \mid \langle for\_stat \rangle \mid \langle read\_stat \rangle \mid \langle write\_stat \rangle \mid \langle compound\_stat \rangle \mid \langle expression\_stat \rangle$
- (6)  $\langle if\_stat \rangle \rightarrow if(\langle expression \rangle) \langle statement \rangle [else \langle statement \rangle]$
- (7)  $\langle while\_stat \rangle \rightarrow while(\langle expression \rangle) \langle statement \rangle$
- (8)  $\langle for\_stat \rangle \rightarrow for(\langle expression \rangle; \langle expression \rangle; \langle expression \rangle) \langle statement \rangle$
- (9)  $\langle write\_stat \rangle \rightarrow write \langle expression \rangle;$
- (10)  $\langle read\_stat \rangle \rightarrow read\ ID;$

- (11)  $\langle \text{compound\_stat} \rangle \rightarrow \{ \langle \text{statement\_list} \rangle \}$
- (12)  $\langle \text{expression\_stat} \rangle \rightarrow \langle \text{expression} \rangle ; | ;$
- (13)  $\langle \text{expression} \rangle \rightarrow ID = \langle \text{bool\_expr} \rangle | \langle \text{bool\_expr} \rangle$
- (14)  $\langle \text{bool\_expr} \rangle \rightarrow \langle \text{additive\_expr} \rangle | \langle \text{additive\_expr} \rangle ( > | < | > = | < = | = = | ! = )$   
 $\quad \quad \quad \langle \text{additive\_expr} \rangle$
- (15)  $\langle \text{additive\_expr} \rangle \rightarrow \langle \text{term} \rangle \{ ( + | - ) \langle \text{term} \rangle \}$
- (16)  $\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \{ ( * | / ) \langle \text{factor} \rangle \}$
- (17)  $\langle \text{factor} \rangle \rightarrow ( \langle \text{expression} \rangle ) | ID | NUM$

根据上述的语法的每条规则，分别设计其递归下降的分析方法。在实现完各个方法后，从 program 方法开始递归调用，利用每个函数的返回值表示分析状态，当返回值为 0 时表示没发现错误，返回值大于 0 表示发现错误，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。

### 三. 实验步骤

#### 1. 实现改写词法分析的程序

为了实现语法分析的过程中输出错误位置的功能，需要对词法分析的程序进行一定的改写，这一部分的修改较为简单，只需要在原来输出词法单元的基础上继续输出当前词法分析器记录的行号和列号即可。修改后的词法单元的输出结果的每一行是一个词法单元，依次是 TokenType TokenValue row column 四个信息，语法分析器在读取词法单元的过程中将每个词法单元的四个信息存储为一个 Token 对象。

#### 2. 实现语法分析的程序

这一部分的基本思路很简单。

首先定义一个 Token 类，包含两个 string 类型的成员变量 TokenType 和 TokenValue 和两个 int 类型的成员变量 row 和 col 用于记录这个词法单元出现的位置，以便在后续语法分析的过程中可以输出错误位置。从词法分析的输出文件读取词法单元，存储在 Token 对象中。

对于每个规则，编写一个函数进行递归下降的处理基本的思路如下图的伪代码所示：

```

void A() {
1)    选择一个 A 产生式,  $A \rightarrow X_1 X_2 \cdots X_k$ ;
2)    for ( i = 1 to k ) {
3)        if (  $X_i$  是一个非终结符号 )
4)            调用过程  $X_i()$ ;
5)        else if (  $X_i$  等于当前的输入符号 a )
6)            读入下一个输入符号;
7)        else /* 发生了一个错误 */;
    }
}

```

实际的实现过程也与上述伪代码中所述的“选择一个产生式不同”而是遍历当前非终结符号的所有产生式，并递归调用每一个产生式，判断是否存在一个产生式能够调用这个产生式的所有过程  $X_i$  后仍然没有任何错误。若存在，则选择这一产生式作为这个非终结符的语法分析结果。

对于规则(2)

需要改写为:

$$\langle R \rangle \rightarrow \langle \textit{declarationstat} \rangle \langle R \rangle | \varepsilon$$
$$\langle \text{statement\_list} \rangle \rightarrow \langle \text{statement\_list} \rangle \langle \text{statement} \rangle \mid \varepsilon$$
$$\langle \textit{statement list} \rangle \rightarrow \varepsilon \langle R \rangle$$
$$\langle R \rangle \rightarrow \langle statement \rangle \langle R \rangle | \varepsilon$$

对于递归下降的词法分析过程，也可以采用另一种实现策略，将所有产生式存储起来，然后用一个函数统一处理。大致方法如下图代码所示：

```
listGrammarItem(inList);
map::GrammarItem,vector<vector<GrammarItem>>> grammar{
    {GrammarItem("program",false),{{GrammarItem("{",true),GrammarItem("declaration_list",{false,GrammarItem("statement_list",{false,GrammarItem(";",true)})),
    GrammarItem("declaration_list",{false},{{GrammarItem("declaration_listR",{false}})},
    GrammarItem("declaration_listR",{false},{{GrammarItem("declaration_stat",{false,GrammarItem("declaration_listR",{false}),{}},{
    GrammarItem("declaration_stat",{false},{{GrammarItem("int",{true),GrammarItem("ID",{true),GrammarItem(";",true)})),
    GrammarItem("statement_list",{false},{{GrammarItem("statement_listR",{false}})},
    GrammarItem("statement_listR",{false},{{GrammarItem("statement",{false,GrammarItem("statement_listR",{false}),{}},{
    GrammarItem("statement",{false},{{GrammarItem("if_stat",{false},{{GrammarItem("while_stat",{false},{{GrammarItem("for_stat",{false},{{GrammarItem("read_stat",{false},{{Gramm
    GrammarItem("if_stat",{false},{{GrammarItem("if",{true),GrammarItem("expression",{false,GrammarItem(";",true),GrammarItem("statement",{false,Grammar
    GrammarItem("while_stat",{false},{{GrammarItem("while",{true),GrammarItem("(",{true),GrammarItem("expression",{false,GrammarItem(";",true),GrammarItem("statement",{false},G
    GrammarItem("for_stat",{false},{{GrammarItem("for",{true),GrammarItem("(",{true),GrammarItem("expression",{false,GrammarItem(";",true),GrammarItem("expression",{false,Gram
    GrammarItem("write_stat",{false},{{GrammarItem("write",{true),GrammarItem("expression",{false,GrammarItem(";",true)})),
    GrammarItem("read_stat",{false},{{GrammarItem("read",{true),GrammarItem("ID",{true),GrammarItem(";",true)})),
    GrammarItem("compound_stat",{false},{{GrammarItem("{",{true),GrammarItem("statement_list",{false,GrammarItem(";",true)})),
    GrammarItem("expression_stat",{false},{{GrammarItem("expression",{false,GrammarItem(";",true)},{{GrammarItem(";",true)})),
    GrammarItem("expression",{false},{{GrammarItem("ID",{true),GrammarItem("=",{true),GrammarItem("bool_expr",{false},{{GrammarItem("bool_expr",{false}})},
    GrammarItem("bool_expr",{false},{{GrammarItem("additive_expr",{false},{{GrammarItem("additive_expr",{false,GrammarItem(";",true),GrammarItem("additive_expr",{false},{{Gram
    GrammarItem("additive_expr",{false},{{GrammarItem("term",{false},{{GrammarItem("factor",{false,GrammarItem("*",{true),GrammarItem("factor",{false},{{GrammarItem("term",{false},G
    GrammarItem("term",{false},{{GrammarItem("factor",{false},{{GrammarItem("factor",{false,GrammarItem("*",{true),GrammarItem("factor",{false},{{GrammarItem("factor",{false},Gr
    GrammarItem("factor",{false},{{GrammarItem("(",{true),GrammarItem("expression",{false,GrammarItem(";",true)},{{GrammarItem("ID",{true},{{GrammarItem("NUM",{true})),
};
```

```

int prase(GrammarItem item)
{
    auto sentences=grammar[item];
    for(auto sentence:sentences)
    {
        for(auto word:sentence)
        {
            if(word.isTerminal&&word.name==currentToken.name)
            {
            }
            else if (word.isTerminal&&word.name!=currentToken.name)
            {
            }
            else if(!word.isTerminal)
            {
                if(prase(word)!=0)
                {
                }
                else
                {
                }
            }
        }
    }
}

```

## 四. 实验结果及分析

测试一:

源文件:

```

/*{
int i;
int n;
int j;
int abc;
j=1;
read n;
for(i=1; i<=n; i=i+1)
j=j*i;
write j;
}
*/

{
    int a;
    int b;
    int c;
    a = 10;
    a >= 10;
    a / 2000000;
    ads = 10;
/* asd */
    b = 1234567890;
    c = a == b;
    read c;
    write(a+b);
}

```

词法分析结果:

```

| { 14 1
int int 15 8
ID a 15 10
; ; 15 10
int int 16 8
ID b 16 10
; ; 16 10
int int 17 8
ID c 17 10
; ; 17 10
ID a 18 6
== 18 8
NUM 10 18 11
; ; 18 11
ID a 19 6
>= 19 8
NUM 10 19 12
; ; 19 12
ID a 20 6
/ / 20 8
NUM 2000000 20 16
; ; 20 16
ID ads 21 8
== 21 10
NUM 10 21 13
; ; 21 13
ID b 23 6
== 23 8
NUM 1234567890 23 19
; ; 23 19
ID c 24 6
== 24 9
ID a 24 10
== 24 12
ID b 24 15
; ; 24 15
read read 25 9
ID c 25 11
; ; 25 11
write write 26 10
( ( 26 10
ID a 26 12
+ + 26 12
ID b 26 14
) ) 26 14
; ; 26 15
} } 27 1

```

语法分析结果:

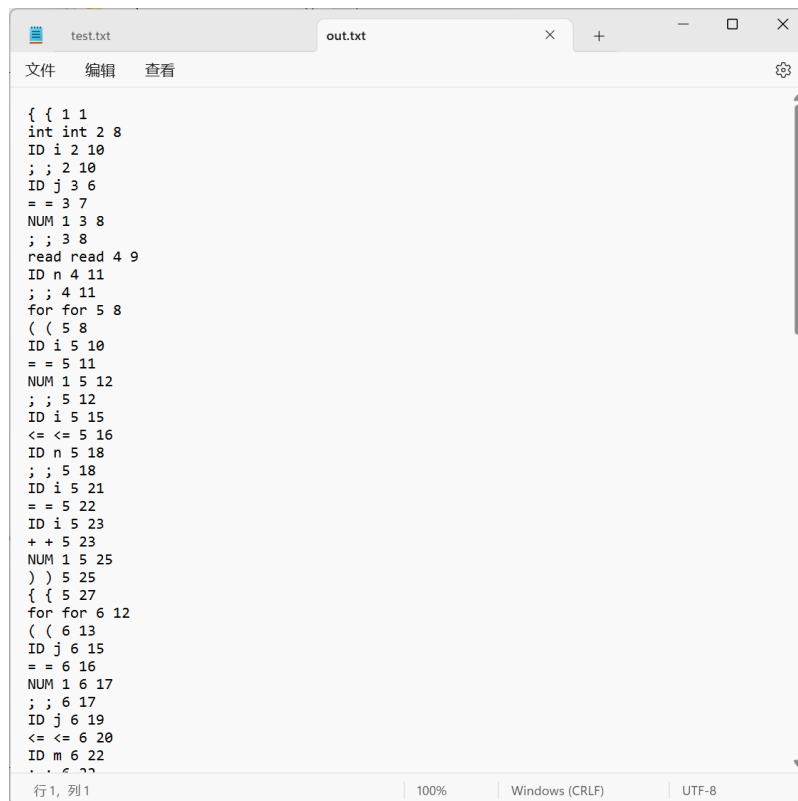
```
正在构建词法分析器所需的NFA:
构建完成
== Grammer prase result ==
Grammar parse success
语法分析正确
```

测试二:

源程序:

```
{
    int i;
    j=1;
    read n;
    for(i=1; i<=n; i=i+1) {
        for (j=1;j<=m;j=j+1) {
            write i+j;
            for (k=1;1==2;2<=3) {
                c=c+1;
                if (c==1) {
                    a=a+1;
                    b=b-1;
                }
            }
        }
    }
    i=1;
    k=j==2;
}
```

词法分析结果:



```
{ { 1 1
int int 2 8
ID i 2 10
; ; 2 10
ID j 3 6
= = 3 7
NUM 1 3 8
; ; 3 8
read read 4 9
ID n 4 11
; ; 4 11
for for 5 8
( ( 5 8
ID i 5 10
= = 5 11
NUM 1 5 12
; ; 5 12
ID i 5 15
<= <= 5 16
ID n 5 18
; ; 5 18
ID i 5 21
= = 5 22
ID i 5 23
+ + 5 23
NUM 1 5 25
) ) 5 25
{ { 5 27
for for 6 12
( ( 6 13
ID j 6 15
= = 6 16
NUM 1 6 17
; ; 6 17
ID j 6 19
<= <= 6 20
ID m 6 22
. . 6 22
```

语法分析结果:

```
正在构建词法分析器所需的NFA:
构建完成
== Grammer prase result ==
Grammar parse success
语法分析正确
```

测试三:

源程序:

```
{
    for (i=1;i<=13) { c=1;}
}
```

词法分析结果:

```
{ { 1 1
for for 2 5
( ( 2 6
ID i 2 8
= = 2 9
NUM 1 2 10
; ; 2 10
ID i 2 12
<= <= 2 13
NUM 13 2 16
) ) 2 16
{ { 2 18
ID c 2 21
= = 2 22
NUM 1 2 23
; ; 2 23
} } 2 24
} } 3 1
```

语法分析结果:

```
正在构建词法分析器所需的NFA:
构建完成
Error occur in forStat!row: 2 col:16
Error occur in program!
== Grammer prase result ==
Missing Semicolon Error
语法分析错误
```

测试四:

源程序:

```
{
    int a;
    int b;
    for i=1;i<=10;i=i+1)
    {
    }
}
```

词法分析结果:

```
{ { 1 1
int int 2 5
ID a 2 7
; ; 2 7
int int 3 5
ID b 3 7
; ; 3 7
for for 4 5
ID i 4 7
= = 4 8
NUM 1 4 9
; ; 4 9
ID i 4 11
<= <= 4 12
NUM 10 4 15
; ; 4 15
ID i 4 17
= = 4 18
ID i 4 19
+ + 4 19
NUM 1 4 21
) ) 4 21
{ { 5 2
} } 7 2
} } 8 1
```

语法分析结果:

```
正在构建词法分析器所需的NFA:
构建完成
Error occur in forStat!row: 4 col:7
Error occur in program!
== Grammer prase result ==
Missing Left Parenthesis
语法分析错误
```

## 五. 实验总结

通过本次实验,我对递归语法分析的过程有进一步的理解,了解并能够编写代码使用递归下降的方法进行语法分析,能够实现简单的错误类型以及错误位置的报告。

## 六. 代码

本次实验的代码涉及到 DFA.cpp、lexerNFA.cpp、main.cpp、NFA.hpp、parse.hpp、utility.h 等多个文件,代码已添加到附件。