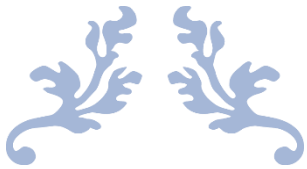

编译原理实验报告

语法分析

2019-6-2

北京理工大学计算机学院 07111606

谭超 1120161874

1. 实验目的:

（1） 掌握 C 语言的语法规则，了解编译器语法分析器的主要功能。
（2） 熟练掌握典型语法分析器构造的相关技术与方法，设计并实现具有一定分析能力的 C 语言语法分析器。
（3） 掌握编译器从前端到后端各个模块的工作原理，语法分析模块与其他模块之间的交互过程。

2. 实验内容:

（1） 该实验选择 C 语言的一个子集，基于 BIT-MiniCC 构建 C 语言子集的语法分析器该语法分析器能够读入 XML 文件形式的属性字符流，进行语法分析并进行错误处理，如果输入正确时输出 XML 形式的语法树，输入不正确时报告语法错误。
（2） 将分析树转换为抽象语法树，便于后续分析工作和代码生成工作的完成。

3. 实验的实现过程与步骤:

在本次实验中我采用了递归下降的方法来构建词法分析器，具体实现过程如下。
首先通过 ANTLRWorks 构建一个不含左递归与相同前缀的文法规则：

```
grammar grammerC;

program :   func_list;
func_list
:   TYPE ID (function |global_variable) func_list
|
;
function:    '(' arg_list ')' code_blocks;
global_variable
:   ('=' assignment_expression)? ';'
;
arg_list
:   argument (',' arg_list)?
|
;
argument:   TYPE ID;
code_blocks
:   '{' statement_list '}';
statement_list
:   declare_expression ';' statement_list
|   assignment_expression ';' statement_list
|   while_expression statement_list
|   for_expression statement_list
|   if_expression statement_list
|   return_expression ';'
|
;
```

```
declare_expression
:   TYPE ID ('='  assignment_expression)?
;
assignment_expression
:   ID (OPERATOR assignment_expression )?
|   constant (OPERATOR assignment_expression)?
;
while_expression
:   'while' '(' assignment_expression ')' code_blocks
;
for_expression
:   'for'   '('   declare_expression   ';'   assignment_expression';'
assignment_expression')'code_blocks
;
if_expression
:   'if' '(' assignment_expression ')' code_blocks ('else' code_blocks)?
;
return_expression
:   'return' assignment_expression;
constant:   'integer_constant'
|   'floating_constant'
|   'string_literal'
|   'character_constant'
;
TYPE :   'int'|'float'|'char'|'double'|'void'|'long'|'short';
ID   :   'identifier';
OPERATOR:   'operator';
KEYWORDS:   'keywords';
```

然后对于文法中的每个非终结符构造一个相应的函数，根据当前的状态和下一将要读入的终结符决定被调用的函数。在我的程序中，除了每个非终结符对应的函数之外还有一个用于存储全局变量的类，为了使用方便，我将所有全局变量都放在该类中。该类中包含有以列表形式存储的从.xml 文件中读入的词法分析后的单词和属性流、属性流的长度、当前位置、用于写入语法分析结果的.xml 的栈以及对词法分析词类更加细分的集合。由于之前的词法分析只分为了 8 类词，为了更好地拟合文法，我将一些较为特殊的终结符单独提出来作为一种词属性。同时，由于要将结果以树的形式写入到.xml 文件中，我使用一个栈来存储递归下降过程中的状态。

4. 运行效果截图:

对于以下 C 语言代码：

```c
int main ( )
{
    int a ;
    int b ;
    a = a + b ;
    while ( a < 10 )
    {
        if ( b = 1 )
        {
            return b ;
        }

    }
}
```

该源代码的部分属性流如下：

```xml
<?xml version="1.0" encoding="utf-8"?>
    <project name=".\test.c">
        <tokens>
            <token>
                <number>1</number>
                <value>int</value>
                <type>keyword</type>
                <line>1</line>
                <valid>true</valid>
            </token>
            <token>
                <number>2</number>
                <value>main</value>
                <type>identifier</type>
                <line>1</line>
                <valid>true</valid>
            </token>
            <token>
                <number>3</number>
                <value>(</value>
                <type>separator</type>
                <line>1</line>
                <valid>true</valid>
            </token>
            <token>
                <number>4</number>
                <value>)</value>
                <type>separator</type>
                <line>1</line>
                <valid>true</valid>
            </token>
            <token>
                <number>5</number>
                <value>{</value>
```

该源代码的属性流经过 ANTLRWorks 分析后得到的语法树如下：

经过语法分析的代码处理后得到的结果部分截图如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<parserTree>
    <program>
        <funclist>
            <datatype>int</datatype>
            <identifier>main</identifier>
            <function>
                <seperator>(</seperator>
                <arg_list/>
                <seperator>)</seperator>
                <code_blocks>
                    <separator>{</separator>
                    <statement_list>
                        <declare_expression>
                            <datatype>int</datatype>
                            <identifier>a</identifier>
                        </declare_expression>
                        <separator>;</separator>
                        <statement_list>
                            <declare_expression>
                                <datatype>int</datatype>
                                <identifier>b</identifier>
                            </declare_expression>
```

5. 实验心得体会：

递归下降方法是构建语法分析器中比较简单的一种，其思想十分简单，就是对于不同的非终结符构造各自的执行函数，然后跟着属性流一步一步走，当识别 wan3 所有的属性则识别成功，否则失败。在构建过程中需要注意的是文法不能存在左递归和公共前缀，对于左递归文法可以通过消除左递归修改文法，对于公共前缀可以通过消除回溯的方法。文法构建完成后工作就完成了一半，另外还需注意的是再写入到 XML 文件中时，由于整个过程是一个递归的过程，在每个函数中都需要将当前状态写入到 XML 树中，所以我用了一个栈来存储状态，每进入一个函数就将该状态压入栈中，同时，通过不断的 push 和 pop 来记录整个递归的过程。