实验 2 语法分析程序分析报告

西南民族大学 计算机科学与技术 1202 欧长坤 201231102123

一、流程分析

1.1 main()函数流程分析

本流程对实验原本提供的流程进行了一些改动，使得程序可以从文本文件中读入需要进行词法分析的程序，并将词法分析的部分独立出来，置于 scaner.h 和 scanner.cpp 中，并统一封装在了 start_lexical_analysis () 中，而语义分析统一置于 parser.h 和 parser.cpp 中。

流程图如图 1 所示。

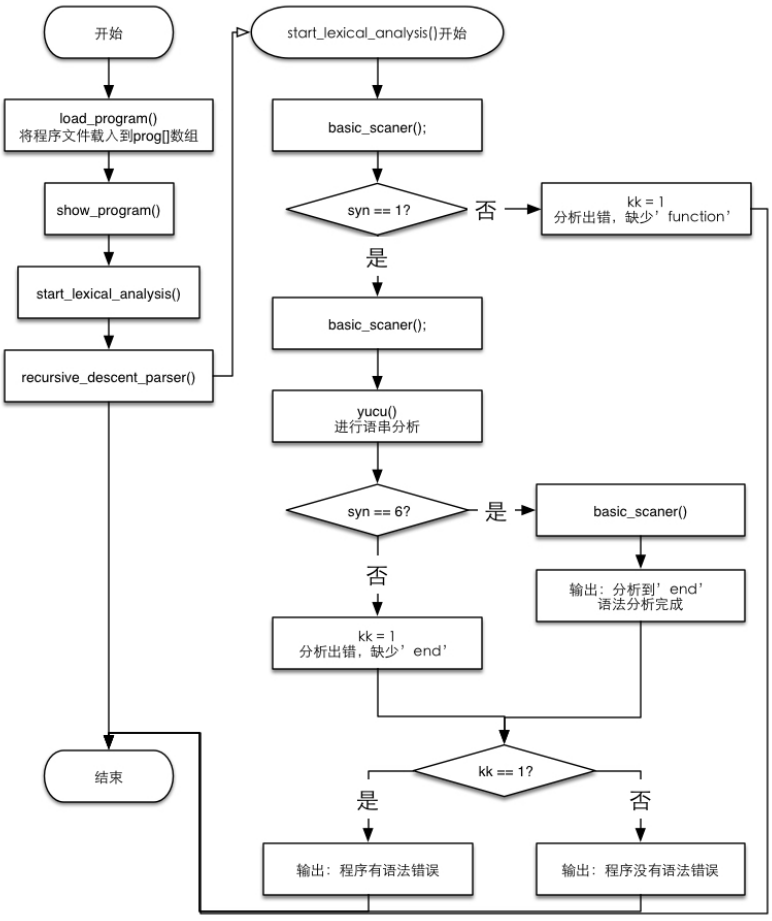

图 1 语法分析程序的 main()函数流程图

1.2 yucu()函数流程分析

yucu()函数用于分析这段文法产生式：
<语句串> → <语句>;{语句}

流程与分析此段文法产生式流程是一致的，首先会进入 statement()语句分析函数，进而判断是否存在';'，接下来进一步进行 statement()语句分析，直到判断到没有再出现';'为止。

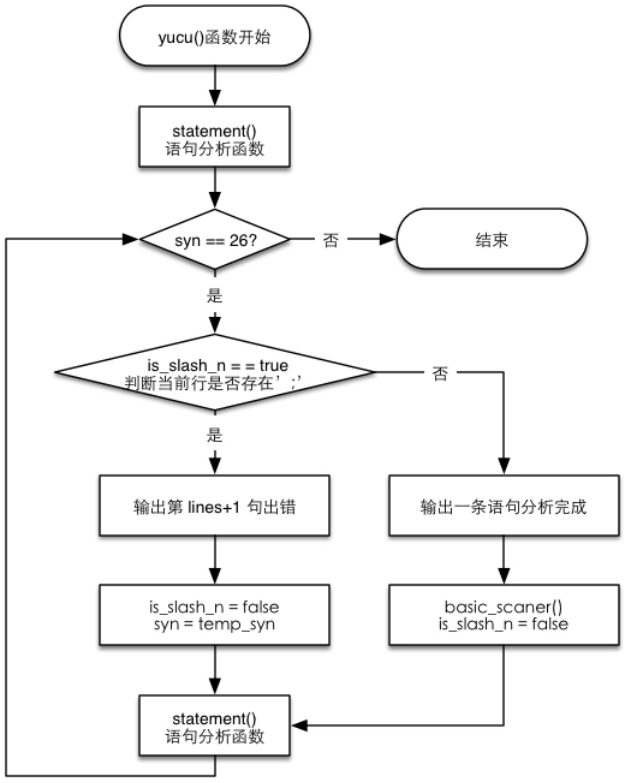

图 2 语法分析程序中 yucu()函数流程图

1.3 statement()函数流程分析

statement()函数用于分析这段文法产生式：
<语句> → <赋值语句>
<赋值语句> → 标识符 = <表达式>
流程与分析此段文法产生式流程是一致的，首先会进入判断当前语句是否存在'='，如果存在'='则对其进行相应的表达式分析。
这是因为，在这两句文法产生式中，'='前面只可能是标识符，不可能是表达式，因此，编译器可以对程序串进行依次读取，读取到'='后，接下来读取的一定是表达式，不需要考虑'='前面的情况。

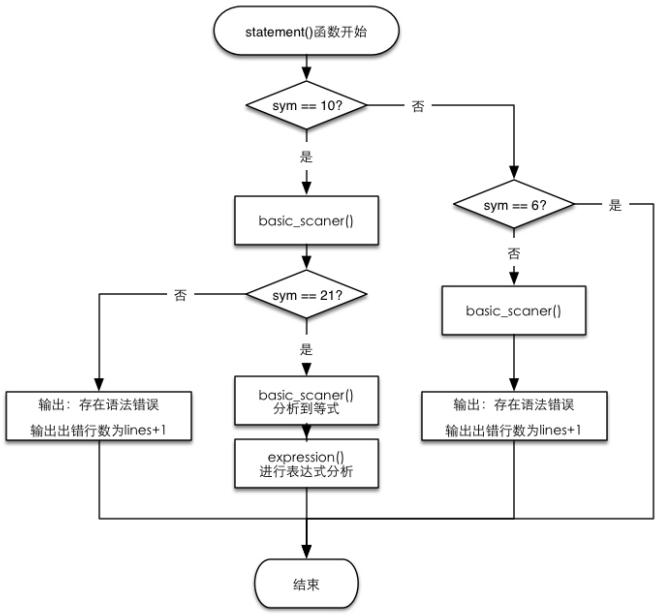

图 3 语法分析程序的 statement() 函数

1.4 expression()函数流程分析

expression()函数用于分析这段文法产生式：
<表达式> → <项>{+<项> | -<项>}
流程与分析此段文法产生式流程是一致的，首先会进入 term()函数分析项的组成部分，然后判断是否出现'+'或'-'，如果出现了，则根据文法产生式的规定，后边必然还需要进行项分析，因此进而再次进入 term()进行项分析，重复；如果不存在；则结束表达式分析。

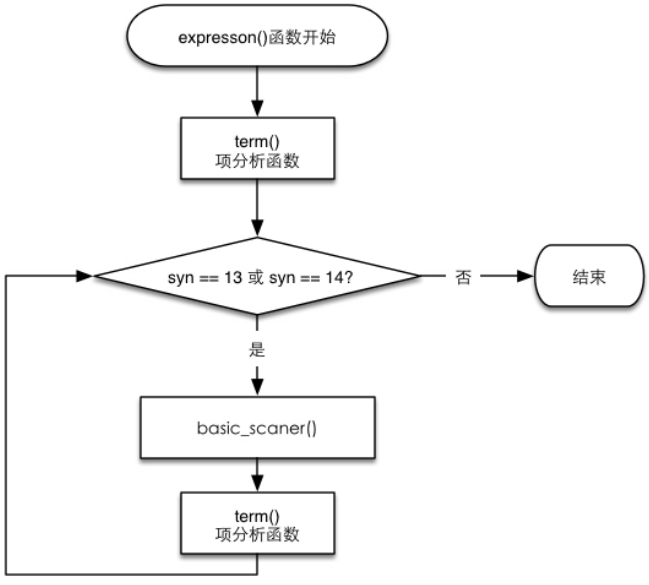

图 4 语法分析程序的 expression()函数流程分析

1.5 term()函数流程分析

term()函数用于分析这段文法产生式：
<项> → <因子>{*<因子> | /<因子>}
流程与分析此段文法产生式流程是一致的，首先会进入 facotr()函数分析因子的组成部分，然后判断是否出现'*'或'/'，如果出现了，则根据文法产生式的规定，后边必然还需要进行因子分析，因此进而再次进入 factor()进行项分析，重复；如果不存在；则结束因子分析。这个过程和 expression()的表达式分析是基本一致的。

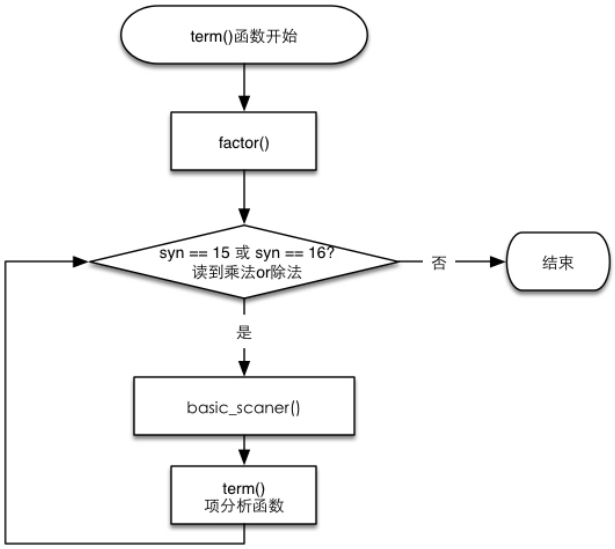

图 5 语法分析程序的 term()函数流程分析

1.6 factor()函数流程分析

factor()函数用于分析这段文法产生式：
<因子> → <标识符> | <数字> | (<表达式>)
流程与分析此段文法产生式流程是一致的，首先会判断因子是否为标识符或数字，如果是，则处理是否出现某行没有';'的情况，并继续读取，结束本次因子分析；如果发现了'('，则根据文法的产生式则说明后面还有表达式，因此需要进入 expression()函数进行进一步的表达式分析，分析完成后，则需要判断时候存在')'，如果不存在，则说明因子存在语法错误，如果存在，则可以完成本次的因子分析，进而返回 term()函数；如果这三者都不是，则说明存在语法错误。

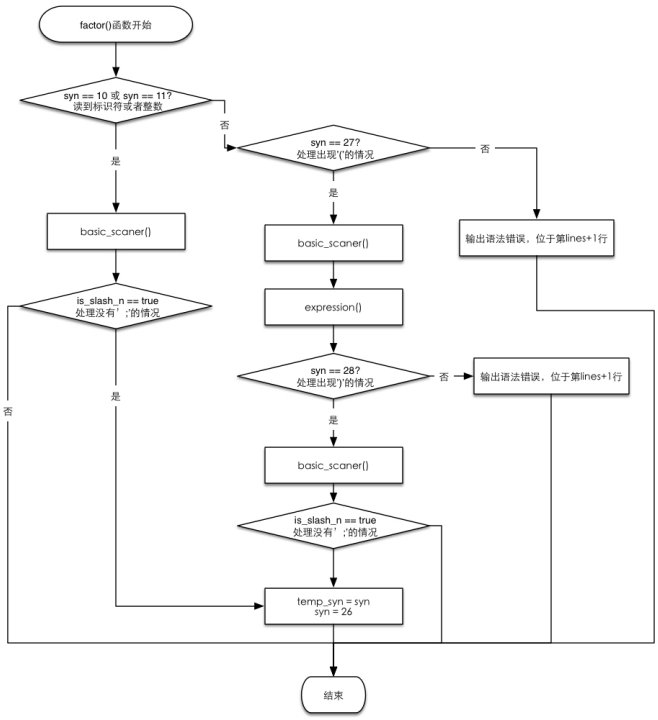

图 6 语法分析程序的 factor()函数流程分析

1.7 basic_scaner()函数流程分析

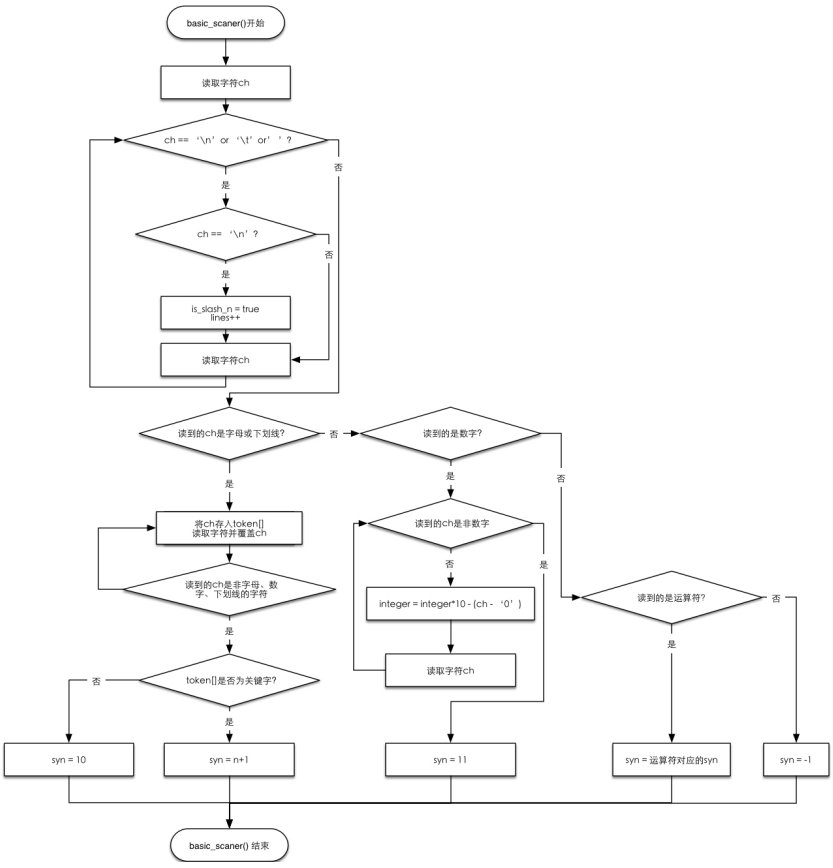

图 7 语法分析程序的 basic_scaner()函数流程分析

basic_scaner()函数流程和词法分析程序中的 basic_scaner()基本一致，唯一 iGetter 改动就是在跳过空格、回车和制表符的同时，增加了对回车符号的记录，主要用于处理某行不出现';'的情况，并用于提醒出现错误的行数。

二、文法的产生式

本实验实现的递归下降分析文法的产生式如下：

<程序> → function <语句串> end
<语句串> → <语句> ;{ <语句> }
<语句> → <赋值语句>
<赋值语句> → 标识符 = <表达式>
<表达式> → <项> { +<项> | -<项> }
<项> → <因子>{ *<因子> | /<因子> }
<因子> → 标识符 | 数字 | (<表达式>)

三、调试过程简述

实验所提供的源程序出现了以下几个问题：

1. 实验所要求的文法产生式的错误，对于

<语句串> → <语句> { ; <语句> }

存在错误，这是因为{}内的内容表示里面的内容可有可无，可以想象设计产生式的人员希望让语言支持';'的可有可无，但是这样的产生式是不能够产生多条无存在';'的语句。从而导致了错误的文法。我给出的解决方案是，将文法修改为：

<语句串> → <语句> ;{ <语句> }

2. 源程序的 statement()函数存在处理问题不完整的情况，statement()首先会判断当前读取的 syn 值是否为标识符，如果不是标识符，源程序直接将这种行为人为是语句串出现错误，但事实上如果 syn=6，则表明已经分析玩语句串，这时便应该直接返回，结束 statement()的执行。

四、功能扩展描述

本次实验一共扩展了四个功能：

1. 能够处理某行没有';'的情况；
2. 能够输出出错语法所在的行数；
3. 详细输出了分析过程；
4. 修复了源程序不会成功的 bug。

五、程序源代码

程序源程序使用 Makefile 进行编译运行。
代码已经开源至：http://github.com/euryugasaki/compiler-of-training
本实验报告涉及的源码位于：实验 2 – 语法分析。