实验 3 语义分析程序分析报告

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

本次实验的 main 函数流程与实验 2 中流程基本一致，唯一的区别在于，本次实验在进行语法分析的过程中，同时进行了语义分析，所以在最后，多输出了进行语义分析的结果。

流程图如图 1 所示。

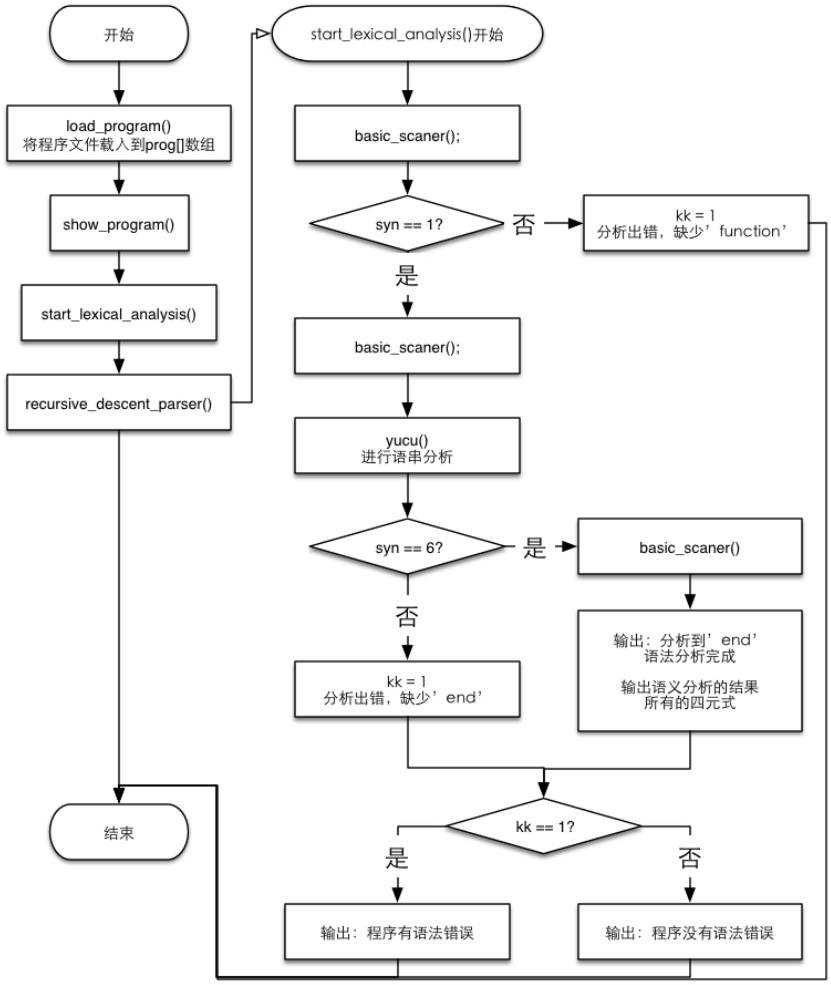


图 1 语义分析程序的 main()函数流程图

1.2 statement()函数流程分析

本次实验中的 yucu()函数和实验 2 中得 yucu 函数完全相同，故不再做详细分析。

但从 statement()函数开始，每个函数开始有细微变化。下面叙述其变化：

在实验 2 中，如果 syn==21，那么会接下来执行 expression()，本次实验中，expresion 将存在返回值，expression 会返回四元式中的 ag1，而 eplace 则负责接收。并调用 emit()函数，来完成一次语义分析。

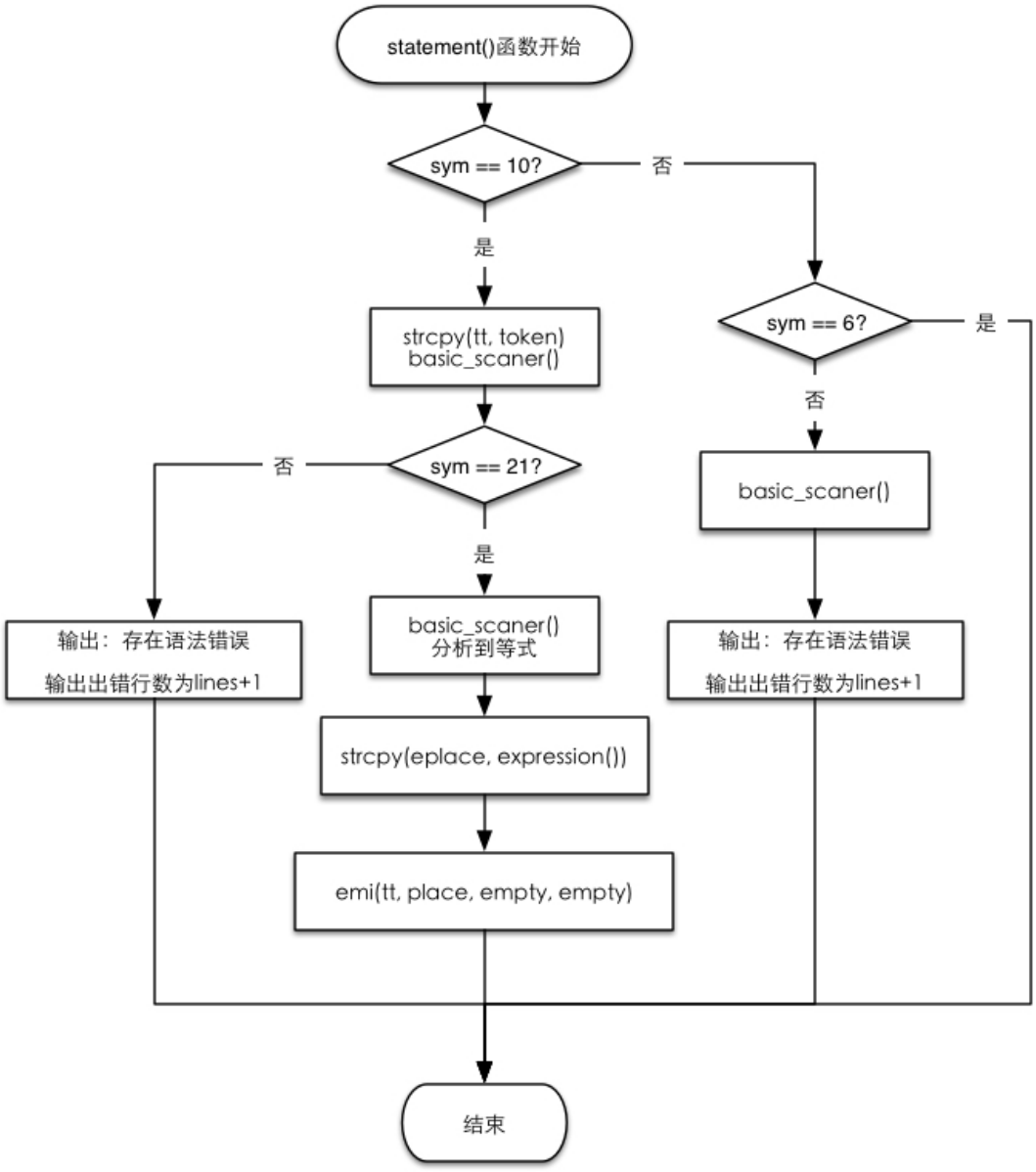
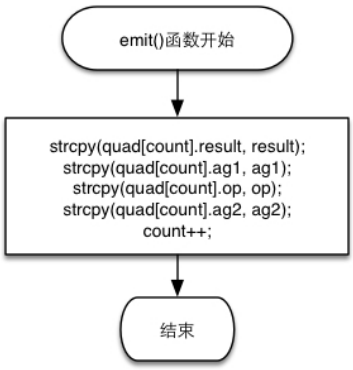


图 2 语义分析程序的 statement() 函数流程

1.3 emit()函数流程分析



emit 函数没有什么特别之处，emit 负责把分析出的四元式送到四元式表中进行保存。

图 3 语义分析程序的 emit()函数流程

1.4 expression()函数流程分析

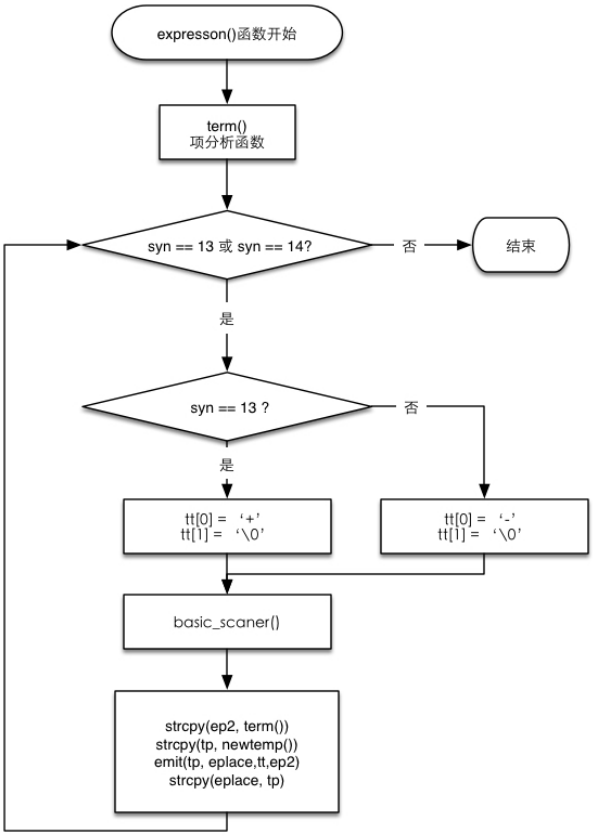


图 4 语法分析程序的 expression()函数流程分析

在实验 2 中，如果 syn==13 或 14，会根据两种情况分开处理 tt[]数组的值。接下来执行 term()，本次实验中，term 函数将存在返回值，term()会返回四元式中的 ag2，而 ep2 则负责接收。接收完成后，则会调用 newtemp()来信件临时变

量。然后再调用 emit()完成一次语义分析。并将 tp 的结果，传递给 eplace，返回给 statement()。由于 term()的过程和 expression 的过程几乎完全一致，故不再赘述。

1.5 newtemp()函数流程分析

newtemp 函数同样很简单，它负责把为四元式新建临时变量，然后把结果作为一个字符串进行返回。

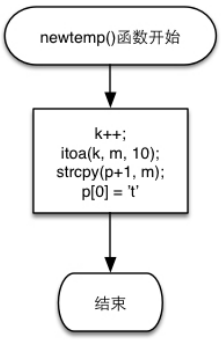


图 5 语义分析程序的 newtemp()函数流程

1.6 factor()函数流程分析

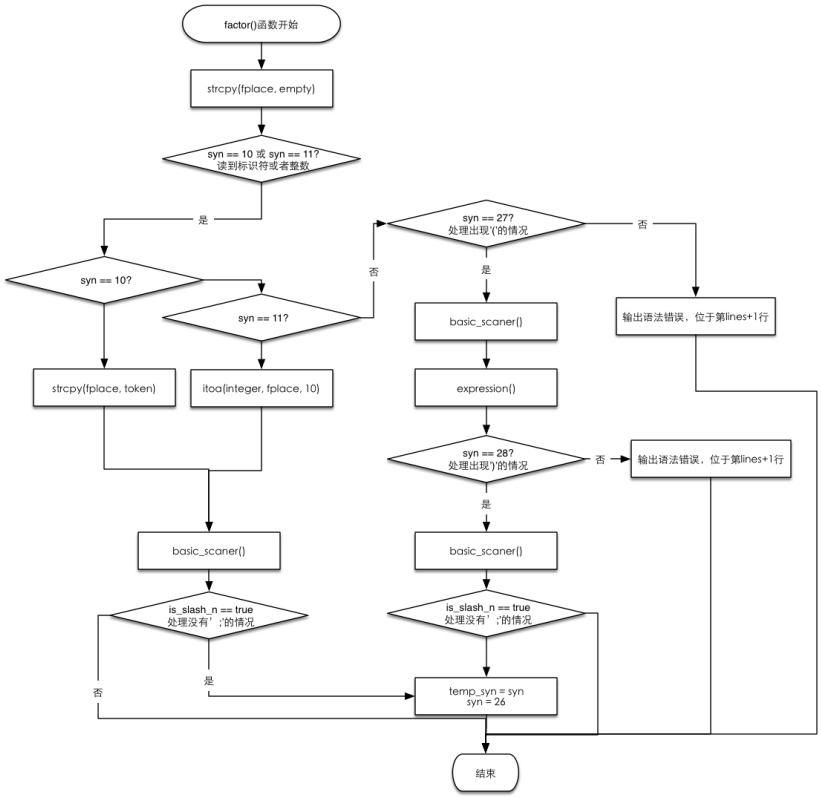


图 6 语法分析程序的 factor()函数流程分析

factor()函数的逻辑进行了很大的修改，首先会判断 syn==10，如果是标识符，则对 fplace 的设置，会设置对应的 token 值，而如果 syn==11，则 fplace 的值会使用 itoa 函数进行调整。如果两者都不是，则会判断 syn 的值是否等于 27，如果是，则说明出现了表达式，需要对括号进行处理，这时候 fplace 的值会处理为 expression 的返回值。出现右括号后，则会返回 fplace。

二、语义子程序的分析

实验中进行语义分析的文法产生式为：
<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>)
为了方面叙述，我们用 A 来表示赋值语句，用 i 来表示标识符，用 E 来表示表达式，用 T 来表示项，用 F 来表示因子，用 n 来表示数字。
则文法产生式简化为：
A → i = E
E → T { +T | -T }
T → F { *F | /F }
F → i | n | (E)
那么，其文法的每一个产生式的语义子程序可写为：
(1) A → I = E
```
{
    p = lookup( i.name )
    if (p == NULL)
        error();
     else
        emit(=, E.fplace, _, p);
}
```
(2) E → T(1) + T(2)
```
{
    E.fplace = newtemp();
    emit(+, T(1).fplace, T(2).fplace, E.fplace);
}
```
(3) E → T(1) – T(2)
```
{
    E.fplace = newtemp();
    emit(-, T(1).fplace, T(2).fplace, E.fplace);
}
```
(3) T → F(1) * F(2)
```
{
    T.fplace = newtemp();
    emit(*, F(1).fplace, F(2).fplace, T.fplace);
```

```
}
(4) T → F(1) / F(2)
{
    T.fplace = newtemp();
    emit(/, F(1).fplace, F(2).fplace, T.fplace);
}
(5) F → i
{
    p = lookup( i.name )
    if (p == NULL)
        error();
     else
        F.fplace = p;
}
(6) F → n
{
    p = lookup( n.name )
    if (p == NULL)
        error();
     else
        F.fplace = p;
}
(7) F → (E)
{
    F.fplace = E.fplace;
}
```

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. 源程序有冗余变量，在 statement 中定义了一个叫做 schain 的变量，于是在相关联的代码中，因此修改了返回值。但是在整个代码中，完全没有使用到 schain 的值，故对此做了简化，删除了 schain 变量。
2. 源程序在处理 factor()函数的时候逻辑混乱，是直接出错的原因，对此根据文法的产生式已经对整个函数进行了重写，分为标识符、数字和表达式三种情况进行了分别处理。

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，对于能够在一条语句完成时就输出四元式。

4. 修复了源程序的逻辑错误。

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 3 – 语义分析。