代码翻译

-
- Hollow Man
-

实验原理分析

定义：

对递归下降分析器进行改造，使其能够一遍处理，同时完成语法分析和中间代码翻译。

输入：

词法分析器所输出单词符号输出成如下的二元式：

（单词内容，单词种别）

根据该二元式进行递归下降语法分析，边分析边翻译。

输出：

与输入对应的等价四元式序列，其格式如下：

行号：（操作符，参数 1，参数 2，目标）

每个函数内的四元式序列会类汇编形式地标出标签号，比如 main 函数，会标为：

main:

算法流程分析

在语法分析阶段，使用的文法如下：

Block -> { structDeclar | functionDeclar }

structDeclar -> struct identifier { {memberDeclar} }

memberDeclar -> varDeclar

varDeclar -> type identifier {, identifier} ;

type -> int | char | boolean | identifier

functionDeclar -> (type|void) identifier (paramList) functionBody

paramList -> type identifier {, type identifier} | ε

functionBody -> { {statement} }

statement -> varDeclarStatement | letStatemnt | ifStatement | whileStatement | doStatement | returnStatemnt | functionCall

varDeclarStatement -> var type identifier { , identifier } ;

letStatemnt -> let identifier [[expression]] = expression ;

ifStatement -> if (expression) { {statement} } [else { {statement} }]

whileStatement -> while (expression) { {statement} }

doStatement -> do { statement }

functionCall -> identifier [. identifier] (expressionList);

expressionList -> expression { , expression } | ε

returnStatemnt -> return [expression] ;

expression -> relationalExpression { (& | |) relationalExpression }

relationalExpression -> ArithmeticExpression { (= | > | <) ArithmeticExpression }

ArithmeticExpression -> term { (+ | -) term }

term -> factor { (* | /) factor }

factor -> (- | ~ | ε) operand

operand -> integerConstant | identifier [.identifier] [[expression] | (expressionList)] | (expression) | stringLiteral | true | false | null | this

程序在每个上下文无关文法的对应函数中，记录下该函数里递归调用的返回值，并统一在对应函数的结束位置进行代码的生成。对于二元运算符嵌套表达式的情况，则每次都会将递归调用的左值赋为 result1，右值赋值为 result2，结果存进 result 进行统一化。

程序关键部分分析

使用递归下降文法，*Test()函数来进行 First 集选择路径，PeekNextToken()方法来使用 Follow 集。在 if..else..中插入代码生成部分，如遇到错误程序立刻停止不再分析。CParser 类中的 line 属性记录当前的生成代码行号，generated 属性则记录当前生成的代码信息。myc 中 main 直接实例化该类进行代码翻译操作。

生成的四元式序列中，除了数学和逻辑表达式中的数学和逻辑符号对应的四元式 op，dec 代表赋值操作；if、else、return 则代表各自的语句含义，因为我的这些语句中括号内的表达式都已经计算成

了对应的布尔值或者要返回的值，且都是存在 result 中的，所以没有使用 j<, j>等类似的跳转表达方式。

运行 myc 时使用一个参数代表当前源代码所存放在的文件夹路径，如 input.c 存放于 test 文件夹中，则直接运行：

```
python input.c test
```

最后生成的代码将会存放于该文件夹目录下的 input-gen.txt 中。

实验总结

1. **编译环境**

Fedora 34

2. **语言环境**

Python 3.9