编译原理 小组实验

简单编译器的实现

王景隆、陈书新、贾元昊

2018 年 1 月 11 日

一、概述

本实验实现了一个简单的 C 语言编译器，目标语言为 Python。其中，词法、语法分析部分利用了 Python 3 的 ply 库，ply 库的 lex 模块提供了词法分析功能，yacc 模块提供了语法分析功能。

本实验选用的测试程序为**四则运算计算（第四个）**，为代码同级目录下的 test.c 文件。

二、使用说明

环境配置：首先，确认本机上装有 Python 3.5 以上的环境。其次，使用

```
pip install ply
```

命令安装依赖的 ply 库。

运行说明：利用命令行进入代码文件夹，然后键入

```
python main.py [input file] [output file]
```

命令编译源文件。如果想要编译测试用的 test.c 文件，则可直接键入

```
python main.py test.c test.py
```

编译测试代码。注意由于设定，输入的文件不得超过 8192 个字符。

如果输入的代码文件没有语法错误，则稍等片刻后程序会退出，此时可在目录下找到编译好的 test.py 文件。运行 python test.py 即可在屏幕上看到：

```
113+(26-2)*4/(2+1)  = 145.000000
```

说明源文件已经被成功编译。

如果想直接运行编译好的 python 代码，则在命令行参数的最后加上 run，如：

```
python main.py test.c test.py run
```

则编译结束后，编译得到的 python 代码就可被直接运行。

运行结束后，还会在同级目录下找到 parsetab.py 文件及 parselog.txt 文件。前者是编译出来的 LALR 表的缓存，后者为语法分析过程中的调试信息，详细描述了分析栈的变化。

三、原理概述

本实验利用的 ply 库有两个主要模块：lex 与 yacc。

lexer.py 文件为本实验的词法分析模块。所有词法单元的名称均在 TOKENS 列表里以字符串的形式列出，每个词法单元的正则定义以 t_[TOKENNAME]的形式命名，以字符串或

函数的形式给出。最后使用 lex.lex()启动词法分析器，lex 模块就会以上述的命名规则找到所有词法单元的正则定义，并进行编译。

cparser.py 文件为本实验的语法分析模块。每一个以 p_开头的函数都描述了一个或多个产生式，这些产生式以函数的__doc__属性给出。随后，函数可能包含一条或多条语句，描述了当按照该产生式规约时应当采取的动作。

考虑到文法可能产生规约-规约冲突，precedence 元组描述了多个运算符之间的优先级与结合性，从上到下优先级依次递增。

ast_code.py 文件为本实验的语义分析模块。它构造了根据文法构建语法分析树的过程。所有语法分析树的节点均继承自 BaseNode 类。每一个 BaseNode 的子类均有一些属性值，这些属性值一般是在 cparser.py 中的函数中被求值。此外，每一 BaseNode 的子类都实现了 generate_code 方法，该方法（可能）调用子节点的 generate_code，用于生成目标代码。

symbols.py 文件为本实验的符号表模块。它构造了加入、查询符号表的过程，用于语法分析与语义分析阶段。

四、主要功能点

下面对本实验的主要功能点进行说明。

语法特性的支持：本实验利用的文法比较强大，支持的语法特性比较完全，包括表达式、多维数组、选择语句、循环语句、控制流语句、字符串、变量初始化、函数定义、函数调用等。由于 C 语言的指针在 Python 中没有很好的对应，经过取舍之后我们只好放弃了这一语法特性。

中间代码生成：尽管我们选取的目标语言为 Python，但为了体现编译原理的课程内容，生成的目标语言实际上是符合 Python 语言的三地址代码，每一复杂的表达式最终都拆成 a = x op y 的形式。同时，我们为每一用于三地址代码的临时变量都维护的独一无二的临时变量名，使得不同的临时变量不会互相干扰。这些临时变量以 temp_var_[NUM] 格式命名。

作用域管理：本实验在语义分析过程中维护了变量的块级作用域，函数、if、while 等拥有单独的作用域，使得不同作用域之间的同名变量不会被干扰。

类型检查：本实验执行了一定程度的类型检查。若用 float 类型的表达式给 int 类型变量赋值，则程序会报错。反之，若表达式中间出现了 int 与 double 之间运算，则运算结果将被提升为 double 类型。

函数参数检查：同样，本实验执行了一定程度的函数参数检查。若传递给函数的参数数量与函数定义中的数量不同，则程序会报错。如果参数的类型不匹配，也同样会报错。

多维数组：尽管 C 语言和 Python 都提供了多维数组的语法特性，但为了体现编译原理的课程内容，我们将所有 C 语言的多维数组都编译为一维数组，对多维数组 arr[X][Y] 的任何访问，均先查询数组的内情向量得到数组的大小，然后通过变换转换为一维数组的下标。

错误处理：本实验对几种常见的错误进行了处理。除前述的两点外，还包括未定义符号名的错误，以及多维数组访问下标不匹配的错误。

常量表达式求值：本实验执行了一定程度的代码优化，主要的一点就是常量表达式的求值。对于形如 X = Y * (2 + 3)的表达式，我们都将其先编译成 X = Y + 5 的形式，再输出成中间代码。

独立完成：本实验除了利用 Python 的 ply 库辅助词法分析与语法分析之外，并未参考任何开源的实现。

五、实验分工

本小组的分工如下：

王景隆	语义分析、中间代码生成
贾元昊	语义分析、文档
陈书新	词法分析、语法分析