编译原理小组作业实验报告

小组成员：刘明桥、段硕勋、周和平

一、实验概述

- **开发语言**：Python
- **辅助工具**：Lex/Yacc
- **实验内容**：实现一个简单的从C向Python的编译器

二、实验原理

1.词法分析

lex.py当中的词法分析器主要利用了Python的ply.lex模块来将源代码分解成一系列的词法单元。文件当中的reserved与tokens定义了保留字列表和token流。分析器通过一系列的正则表达式来匹配输入的字符，并将匹配到的字符组成词法单元。这个过程还会对空白字符和注释进行过滤，并在出错时报告错误以便调试。

```
# 例如这里列出了reserved当中有关控制语句的部分内容
reserved = {
    'switch': 'SWITCH',
    'break': 'BREAK',
    'return': 'RETURN',
    'goto': 'GOTO',
    'while': 'WHILE',
    'case': 'CASE',
    'for': 'FOR',
    'default': 'DEFAULT',
    'continue': 'CONTINUE',
    'do': 'DO',
    'else': 'ELSE',
    'if': 'IF',
}
```

生成的token流将进入语法分析器进行语法分析。

2.语法分析

根据grammer.txt中定义的规则生成yacc.py文件。yacc.py当中的语法分析器主要利用了Python的ply.yacc模块来将词法单元流转换成语法树。在yacc当中，通过定义推导符号与产生式，即可自底向上的完成语法分析过程。这里定义了一个ASTEncoder类用于处理 internode 和 exnode 类，分别对应非终结符和终结符。这两个类定义在tree.py当中。

```
# 例如这里列出了yacc.py中关于type_spec的推导规则
def p_type_spec(p):
    ''' type_spec : VOID
                  | CHAR
```

```
                    | SHORT
                    | INT
                    | LONG
                    | FLOAT
                    | DOUBLE
                    | SIGNED
                    | UNSIGNED
                    | BOOL
                    | struct_or_union_spec
                    | enum_spec '''
    p[0] = internode('type_spec', p[1:])
```

分析完毕语法后，会建立抽象语法树，以供下一步生成Python代码。

3.语义分析与代码生成

在 `compile.py` 文件中，语义分析与代码生成主要通过 `Compiler` 类来实现。以下是该部分的详细说明：

1. 语义分析

语义分析的主要任务是检查语法树中的语义错误，并为代码生成做准备。在 `Compiler` 类中，语义分析通过以下几个方法来实现：

- 计算结构体或联合体的标志。
- 将叶子节点转换为相应的字符串。
- 遍历语法树，生成中间代码列表和标志列表。
- 判断节点是否为函数声明。
- 提取声明中的标识符，并进行变量表的更新。
- 替换标识符名称，避免命名冲突。

2. 代码生成

代码生成的主要任务是将语法树转换为目标代码。在 `Compiler` 类中，代码生成通过以下方法来实现：

- 根据语法树节点的类型，生成相应的目标代码。
- 格式化并缩进生成的代码。

`Compiler` 类的主要方法：

```
class Compiler:
    def __init__(self):
        self.funcs = []
        self.decls = []
        self.globals = []
        self.var_table = {}
        self.head = "".join(some_functions) + '\n'
        self.tail = tail
```

```
    def compilec(self, input_file_name, output_file_name):
...
```

三、功能说明

1.预处理指令

- 支持#include/#ifdef/#ifndef/#endif/#define等预处理指令。
- 对于#include跟随的文件，会根据是否是标准库文件决定是跳过处理还是返回完整的路径
- 对于#define指令，程序将利用正则表达式自动进行宏替换

2.数据类型

- 支持int/float/char/string等基本数据类型
- 支持const，inline，volatile等修饰性关键字
- 支持struct/union/enum等复杂数据类型
- 支持指针类型，但考虑到Python并没有相关的机制，所以指针类型只会当作一般的类型处理

3.运算

- 支持+-等基础数学运算，支持&！等基础逻辑运算

4.控制语句

- 支持分支结构if-else/switch-case
- 支持循环结构while/for/do-while
- 支持跳转语句goto/break/continue/return

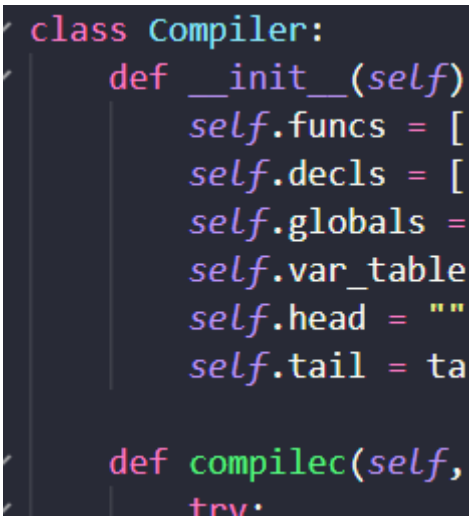
5.函数

- 支持函数声明/定义/调用

6.其他

- 将printf/atoi/atof/system等几个标准库函数直接与python代码段对应
- 缩进长度由节点的code属性的嵌套层数决定，很好的兼容自下而上的语义处理。

四、难点与创新点



1、语法树的生成与展示 此处自定义node类于tree.py中并在yacc中当ply库生成语法树时同时调用该类完成构造树的构造。 同时为了打印出语法树的具体构造又定义了ASTEncoder类用于解读该数据结构使其以json方式打印出

2、对作用域的解析 将变量（包括函数）其进行合理的重命名，使其能类似于c中的机制

3、对python格式的处理 frmat_single_item 是一个递归函数，用于处理每个代码项的缩进。 如果 item 是字符串，则在其前面添加相应级别的缩进。 如果 item 是列表，则对列表中的每个子项递归调用 format_single_item，并增加缩进级别。 如果 item 是其他类型，则直接转换为字符串 通过分类完成了格式的处理

五、分工

- 刘明桥：词法分析、语法分析、代码生成、项目管理统筹
- 段硕勋：词法分析、语法分析、文档编写
- 周和平：测试样例编写

使用方法详见readme