编译原理小组实验实验报告

1. 实验内容

实现了从C语言到Python语言的简单编译器，支持下列C语言特性：

1. `#include`,`#define`预处理指令
2. 简单的变量作用域（全局变量/局部变量）
3. 选择语句（if-else）、循环语句（for、while）
4. 结构体和数组（非指针）
5. 浮点数、科学计数法

且实现了KMP字符串匹配、回文检测、插入排序等示例程序的翻译。

2. 实验环境

操作系统：Windows
编程语言：Python
第三方工具：PLY(Python Lex-Yacc)

3. 实验原理

C语言源文件经过`lex.py`词法分析得到`token`流，再经过`yacc.py`语法分析得到抽象语法树，在`interpreter.py`程序中进行语义分析、中间代码生成，并最终输出目标代码Python源文件。

- **词法分析：**

  使用ply库中的lex模块实现，包含了C99标准文档中的保留字，对于同类的其它词素进行整理得到了合乎规范的tokens并用正则表达式书写了模式。同时，注释、换行和部分错误处理也在词法分析阶段实现。

- **语法分析：**

  使用ply库中yacc模块实现，整体参考了C99标准文档中的语法规则，自底向上的完成语法分析过程。该过程中会进行抽象语法树(`ast_.py`)的构造，每当一条文法产生式被匹配，便会调用`ASTInternalNode`为抽象语法树添加一颗内部节点，该节点包含了词法单元名和其子节点列表；终结符被匹配时会添加新的叶子节点。在完成语法分析后，会得到源代码的抽象语法树并作为下一步的参数。

- **语义分析：**

  TODO：不太清楚该部分实现细节

3. 功能说明

- 预处理

  `#define`类型预处理会在python代码中定义相应宏，`#include`类型预处理若参数为C标准库，则在python代码中包含预先使用python语言写好的C标准库替代文件，这些文件中实现了部分常用功能；若参数为自定义文件，则将对应文件的所有代码复制到源代码相应位置。

- 函数

  编译器可以识别源代码中的`main`函数，并将其作为程序入口点。对于其他函数，语义分析阶段会优先找到所有函数的声明和定义，并翻译成对应的python代码放到全局变量之后。

- 变量作用域

TODO：其它变量作用域不确定是否实现

编译器会记录下所有的全局变量，并在生成python代码时，于每个函数前声明这些全局变量，以确保变量拥有正确的作用域

- 分支/循环

  `if-else`格式和`while`格式在python中都容易翻译成对应格式，对于C语言中的`for`循环，由于其用法和Python有较大差异，因此被翻译成等价的`while`语句。

- 数组

  编译器支持不涉及指针的C语言数组定义，这些数组被翻译成了Python中的列表。数组的声明使用`name = [None]*length`形式翻译。数组下标的使用形式于Python列表保持一致。

- 结构体

  将C的结构体翻译为了Python的类，即`struct A{...}`被翻译成`class A: ...`。对于其中结构体的嵌套也正确翻译成了Python中语法格式。对于结构体数组，使用了`struct_list = [struct_name() for i in rage(list_length)]`的方式创建。

- 缩进

  使用嵌套列表来存储语法分析树的节点，每一层嵌套都代表了一层缩进。

4.难点与创新

1. C语言标准库的实现：使用预先写好的对应python文件解决。
2. 变量作用域问题：在每个函数中声明全局变量以确保正确。
3. 缩进问题：使用嵌套列表方式，以嵌套层数表示目标代码的缩进。
4. 目标代码风格：手写了`lint`函数以确保目标代码优美。
5. for循环问题：将C风格的for循环翻译为python中等价的while循环。

5. 小组分工

TODO