cpp2llvm词法分析与语法分析

林敏芝 王皓雯 郭嘉伟

选题与工具

源语言：C++

目标语言：LLVM（LLVM 架构的中间代码，可以使用LLVM工具编译成机器代码

使用python-lex-yacc

开发环境

操作系统：ubuntu 20.04

安装库：

```
sudo apt-get install clang
sudo apt-get install llvm
pip install PLY
```

并安装与已安装llvm版本对应的llvmlite。

运行方式

- 词法分析：对需要翻译的C++语言文件`test.cpp`，运行以下命令

```
python src/lex.py test.cpp
```

程序将输出token流，可以用定向符把它存储到文本文件中：

```
python src/lex.py test.cpp > token_test.txt
```

支持的功能

- 预处理`#include`和`#define`
- 词法分析的错误处理：遇到错误（无法匹配识别的部分）时不会阻塞，会打印报错信息并继续进行编译

难点与创新点

词法分析

预处理

在进行匹配token前，需要预处理C++中由`#`开头的预处理命令，主要是对`#include`、`#define`进行预处理。

首先，逐行读入需要处理的文件，去除原文件中所有的空格、换行符等C++语言中无意义字符。

```
line = line.strip(' ').strip('\t').strip('\r').strip('\n')
```

通过搜索由`#include`开头的行来处理头文件：对于头文件名被`""`修饰的自定义库，根据引号内的文件名构造绝对路径，并将其添加到需要读取的文件列表中；对于头文件名被`<>`修饰的标准库直接跳过，后续对库函数进行单独处理。处理后，将这些行从原始串中删去。

递归地预处理需要读取的文件列表，将所有函数和类的定义都粘贴进处理后的串。在递归时，通过**条件编译语句`#if、#ifdef、#ifndef`等判断是否重复处理**，防止进入死循环或进行冗余处理。

搜索由`#define`开头的行来处理：遍历所有定义宏语句，将其名称和定义的内容添加至定义列表中。在遍历匹配串中所有存在的宏名称，替换为定义的内容。

预处理后，将得到的串返回。

词法分析中正则表达式

见`lex.py`代码中注释，解释其功能和正确性。

对于c++代码中的存在的注释，通过正则表达式识别`//`和`/**/`两种，对其进行识别，期望能最终转化为llvm中的注释，提高转化出的代码的可读性。

错误处理

预处理中，对所有由`#`开头且不是`#include #define`或者条件编译语句`#if、#ifdef、#ifndef`等的行，打印错误提示并跳过，继续处理。

词法分析中，对所有无法与已存在token匹配的，打印错误提示并跳过，继续匹配token。

分工

	词法分析阶段
林敏芝	搭建框架、工具调研、词法分析正则表达式设计、注释处理
王皓雯	搭建框架、工具调研、测试代码编写
郭嘉伟	编写预处理代码`#include`、`#define`、`#ifndef`，查找C++标准

参考

C语言标准：https://www.dii.uchile.cl/~daespino/files/Iso_C_1999_definition.pdf

https://blog.csdn.net/keke193991/article/details/17566937

https://blog.csdn.net/qq_41687938/article/details/119349135