Lab-1 实验报告

22371027-汤睿璟

一、参考编译器设计

编译器主要参考了 Wokron 学长为编译课设计的实例编译器 tolangc。

考虑到现在实验课和理论课的进度，该部分内容会在后续的实验中迭代完善。

1.1 tolangc

tolangc 的前端在 libs/tolang 文件夹下，主要由 lexer, parser, AST 和 visiter 四个部分构成，分别处理词法分析、语法分析、抽构建象语法树和中间代码生成四项任务。其符号表由 libs/tolang/include/symtable.h 文件中 SymbolTable 类来统一管理。前端的错误处理则由同目录下的 error.h 文件下的 ErrorReporter 完成错误的记录和输出。

前端的文件结构为：

```
tolang
├── CMakeLists.txt
├── include
│   └── tolang
│   ├── ast.h
│   ├── error.h
│   ├── lexer.h
│   ├── parser.h
│   ├── symtable.h
│   ├── token.h
│   ├── utils.h
│   └── visitor.h
└── src
├── ast.cpp
├── lexer.cpp
├── parser.cpp
└── visitor.cpp
```

后端的文件结构随实验和理论课推进后进行补充。

二、编译器总体设计

本人设计的编译器由前端和后端构成，前端包括词法分析、语法分析、符号表构建和管理、抽象语法树构建和中间代码生成。**后端的相关内容随实验和理论课推进后进行补充。**

2.1 接口设计

词法分析器提供方法 nextToken() 用于返回 Lex 到的下一个 Token，用以分析构建语法树。

三、词法分析设计

词法分析器采用自顶向下的分析方法对源代码进行解析，一遍完成文件的读入，并将输入的原文件在对外接口中暴露为 Token 流。 Lexer 在 Lex 标识符时会对关键字进行检查，并在可以在 nextToken() 接口中读取到 Lex 到的下一个 Token。

在编码前，原本预计使用 vector¡Token¿ 将所有读入的 Token 进行保存，再进行处理。后续发现这种处理方式会带来额外的内存开支，并在随后废弃了这种想法。