LameCC

LameCC: 编译原理课程设计项目汇报

2050250 李其桐

Computer Science and Technology Department, College of Electronic and Information Engineering(CEIE), Tongji University.
同济大学 电子与信息工程学院 计算机科学与技术系

2023 年 6 月 24 日

项目开发任务目标

- 类 C 编译器程序架构设计与实现：使用高级程序语言作为实现语言，设计并实现一个类 C 语言的编译器，编码实现编译器的组成部分。
- 词法分析任务：对于词法分析任务，给出类 C 语言的单词子集及机内表示，输入为源程序字符串，输出为单词的机内表示序列。
- 语法分析任务：对于语法分析任务，通过 LR(1) 或者递归下降等语法分析方法设计并且构建语法分析器，同时在语法分析过程中一遍地调用词法分析器的 nextToken 方法获取下一个 token，推进语法分析过程。
- 中间代码生成任务：使用语法制导翻译技术，选择合适的中间代码表示形式（本项目中采用 LLVM IR），要求能够在语法分析的同时生成中间代码，并且将生成结果保存到文件中。
- 目标代码输出任务：编译器能够根据输入的类 C 语言源程序，还有运行时的参数选择，针对不同的处理器架构（target），输出例如 x86，x86-64，mips 等多种汇编代码，并且根据运行时参数进行不同等级的代码优化。生成后的汇编代码文件可以链接第三方编译环境提供的相关类库，进一步生成可执行文件。

项目开发任务目标

- 实现过程、函数调用、指针、数组和 GCC 风格内联汇编的代码编译：本实验中，我已经实现了包括过程、函数调用（支持递归）、指针、数组等等各种类 C 语言的文法扩展和编译能力，使得编译器的功能更加健全与完备，具体实现的扩展功能点将在报告下文中进行详细阐述。

预备知识

- 词法分析器设计原理
- 文法分析方法，包括 LR(1) 分析方法、递归下降分析法，ACTION 表和 GOTO 表的推导方法以及分析流程逻辑等等
- 类 C 语言语法规范文法的设计
- 语义分析与中间代码产生原理与技术
- LLVM IR Builder 库编译、集成与部署技术，用以发射规范的 LLVM IR 中间代码，这种中间代码表示形式相比于四元式而言可读性和规范性更强，并且更加容易进行代码优化。
- 中间代码优化与目标代码生成技术

开发环境

- OS：Windows 11 Pro/Mac OS/Ubuntu20.04(可跨平台)
- Language：CPP
- IDE：vscode + visual studio
- 编译环境：Cmake + MinGW64 + MSVC + Clang
- 报告绘图工具：StarUML + Doxygen
- LLVM 版本：version 17.0.0 git Optimized build.

依赖库

- json.hpp (https://github.com/nlohmann/json)：用于以 json 格式 dump tokens
- rang.hpp (https://github.com/agauniyal/rang)：用于修改控制台输出字体的颜色、样式等。
- LLVM version 17.0.0 (https://github.com/llvm/llvm-project)：使用 llvm::cl::opt 处理程序运行时的启动参数，使用 LLVM 提供的 LLVM IR Builder 相关接口生成 LLVM 中间代码，使用 LLVM Pass 进行代码优化和目标代码生成。

类设计

- File 类：用于储存读入的源文件内容，提供了操作文件的相关接口，包括数据获取、移动行号、信息记录等等
- Lexer 类：词法分析器实现主体类，在一遍过程中为语法分析器提供 nextToken() 接口用于推进分析过程
- Parser 和 LR1Parser 类：实现了递归下降和 LR1 分析两种语法分析方法的语法分析器实现主体类，输出相同的 AST 数据结构
- LLVMIRGenerator 和 IRGenerator 类：分别使用语法制导翻译技术，输出 LLVM IR 和四元式形式表示的中间代码序列
- CodeGenerator 类：使用 LLVM IR 中间代码序列作为输入，执行代码优化逻辑后输出可执行的目标文件
- 相关工具类：包括错误处理、日志记录、运行时参数解析等于项目实现相关的工具类，可在源码中查看，此处不再赘述
- AST 节点类：本实验中的 AST 节点类设计参考了 Clang 源码中的设计架构，为每种实现需求中的文法符号实现了对应的节点类型定义，此处重点阐述其中所有类型的具体含义