Light C Internal

TianXiao Xia

LightC is a ANSI C compiler. implement with modern compiler-kits: yacc lex llvm.
为了实践编译原理，解决实际实现中会遇到的问题；希望能做到结构清晰，即使会牺牲些性能

1 Architecture

取舍:

- 构造出 AST 后再生成中间代码，而不是一边 parse 一边生成
- 构造 AST 时不使用 llvm::Type 子系统，一是练习，也是方便以后替换掉 llvm 的东西
- 构造 AST 时完成声明和符号表的处理，AST 中的 decl 节点含有中间代码的类型信息
- 为单条声明语句作准备
- 引入轻量级的 string 代替 std::string，类似 llvm::StringRef

2 Grammar of C

- declaration 一条声明语句，可以是变量声明、函数声明、for 循环声明或其他复杂声明
- primary 包括变量、整数、字符串等
- declarator 声明中得变量，被声明对象

3 Type subsystem

4 Memory

5 AST internal

three kind of AST: Type Decl Stmt
although could just use llvm's Type system,but attemp approach another way

5.1 SymbolTable

符号表由链表链接

- 在 {} 的进入和退出时执行 push 和 pop，由词法解析器完成,symboltable 放在 compound_statement 里
- function_definition 结束时 pop
- function proto 结束时 push
- declaration in forloop 时处理和函数处理相同
- declaration in block_item 直接加入当前符号表
- declaration 同上（此处是函数外的全局声明）

5.2 Implement

词法分析器在'{、}' 时，分别执行 push 和 pop 操作
借助于一个标志变量，proto 在第一个参数开始的时候 push，直到整个函数结束的时候再 Pop，对于 abstract 声明或纯声明未测试
全局有一个 context 存放全局变量

6 中间代码

6.1 How to represent stack variables

declaration 分为 global var 和 function var，有不同的生成代码

- mutable local variables, same as clang and llvm-gcc

7 Error Recovery

8 Testing

生成的 IR 代码与 clang 的 IR 代码对比，优化选项全部关闭
根据 llvm 的 cpp 后端，生成 IR 产生代码，进行调试

9 Index

- SSA Single Static Assignment subset of CPS
- CPS Continuation Pass Style for functional language