Step 7 Report

刘轩奇 2018011025

2020年10月6日

1 工作内容

本人选择不使用辅助工具 ANTLR 因此自己实现了 lexer 和 parser。

1.1 文件说明

`montLexer.h/cpp` 词法分析器；

`montParser.h/cpp` 语法分析器；

`montConceiver.h/cpp` 产生中间代码；

`montAssembler.h/cpp` 从中间代码产生汇编代码；

`montLog.h` 记录编译错误信息；

`montCompiler.cpp` MiniDecaf 编译器，包含主函数，编译成功返回 0 否则返回 -1，并将错误信息输出到 `std::cerr`。

1.2 本步骤完成的工作

什么也没做，就这么用 step6 的代码通过了测试。因为在 step5 中已经做完了有关工作。

2 思考题

1 请将下述 MiniDecaf 代码中的 `???` 替换为一个 32 位整数，使得程序运行结束后会返回 0。

```
int main() {
 int x = ???;
 if (x) {
     return x;
 } else {
     int x = 2;
 }
 return x;
}
```

答 取 0 即可，此时进入 `else` 分支后的声明的变量仅在此作用域中有效，跳出后 `x` 仍为初始的 0。

2 在实验指导中，我们提到“就 MiniDecaf 而言，名称解析的代码也可以嵌入 IR 生成里”，但不是对于所有语言都可以把名称解析嵌入代码生成。试问被编译的语言有什么特征时，名称解析作为单独的一个阶段在 IR 生成之前执行会更好？

答 若该语言允许先使用后声明/定义（例如不提前声明而直接相互调用的两个函数），或者允许不声明而直接使用（例如 Python 语言，若使用所谓编译器而非解释器），或者允许多文件的声明和定义分离（例如 C 的头文件和源文件），则将名称解析单独作为一个阶段更好。