Step 11 Report

刘轩奇 2018011025

2020年10月8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

1 词法分析 没有改变。

2 句法分析 按照指导书上的要求修改了以下的树节点：

* `unary` 支持解地址和取地址，以及类型转换。
* `type` 支持指针类型。
* `assignment` 等号左边的树节点为`type`类型。

3 产生中间代码 首先添加了类型分析功能，类型附加在每个树节点上，在 parse 阶段不分析类型而是在生成中间代码步骤中产生类型。对于一般的控制过程语句（循环、判断节点等）其类型为空(void)，而表达式类型树节点（包括表达式、各种运算等节点）则生成对应的类型。

此阶段也分析表达式是否为左值。在遍历语法树的过程中，根据语法指出当前分析的节点应当是否为左值（例如生成 `assignment` 节点时，等号左边的对应节点应该为左值），而若产生的实际表达式不是左值则报错。

同时在生成中间代码的分析过程中也分析类型是否正确：函数调用参数类型是否匹配、是否有指针参与数学运算等、循环条件表达式是否为指针等等。

4 产生汇编代码 没有改变。

2 思考题

1 为什么类型检查要放到名称解析之后？

答 只有名称解析之后才知道函数调用的返回值类型是什么。若不知道函数返回值的类型则无法进行类型检查。

2 MiniDecaf 中一个值只能有一种类型，但在很多语言中并非如此，请举出一个反例。

答 例如 python 中的变量可赋予不类型的值，其具体类型根据赋值而改变，C++中union语法也可以支持一个变量中存储的值以不同类型的方式分别处理。

3 在本次实验中我们禁止进行指针的比大小运算。请问如果要实现指针大小比较需要注意什么问题？可以和原来整数比较的方法一样吗？

答 指针比较大小应当注意判断指针的类型是否相同，若指向的类型不相同则无法进行比较，若指向的类型相同，则可以比较指向地址的高低，此时等同于无符号整数的比较。