实验报告—step4

2018011340 计 83 郭峥岩

一、 实验内容

1. 整体框架

本次实验中 step4 的整体架构与 step3 相差不大，改动要点与 step3 相比相差不多，甚至不需要用到新的 IR 类，沿用 step3 的二元运算符类，唯一注意的一点就是如此实现的逻辑运算'&&','||'没有短路性质，但是在本次实验中对此性质并无要求，因此可以沿用 step3 的框架和类。

2. Step4 的功能实现

（1） 实现程序的语法规范的改动：

增加了比较和逻辑运算表达式，总体来说与 step3 相差不大。

（2） IR 类的更改和产生：

沿用 step3 中设计的二元运算符类，因为各种比较运算符以及逻辑操作运算符本质上可以理解为二元运算符，只需要写好其对应的汇编代码即可。

（3） 汇编代码的产生：

具体的生成过程同 step3，根据上述的 IR 代码，设定好比较和逻辑运算需要用到的汇编代码，之后按照 AST 的遍历顺序正常运行即可。

二、 思考题

1. 在表达式计算时，对于某一步运算，是否一定要计算出所有的操作数的结果才能进行运算？

解答：不一定，比如在计算表达式 $0 \&\& (1+1)$ 的时候，由于左边已经是 0，因此无论最终右操作数的结果是什么，对于最终结果都不会有什么影响，此时可以不需要计算出该操作数的结果。

综上：对于某一步运算，不一定要计算出所有结果的操作数的结果才能进行运算。

2. 在 MiniDecaf 中，我们对于短路求值未做要求，但在包括 C 语言的大多数流行的语言中，短路求值都是被支持的。为何这一特性广受欢迎？你认为短路求值这一特性会给程序员带来怎样的好处？

解答：因为短路求值能够给我们带来很多好处：

（1） 编译器中使用短路来消除节省文件大小和执行速度的死代码。在运行时，如果短路的结果不影响答案，则短路不会执行逻辑表达式中的其余子句，从而加快了公式的求值速度。

（2） 如果有一个调用开销较大的函数，该函数在右侧返回了一个布尔值，而我们只想在另一个条件为 true(或 false)时才调用该布尔值。在这种情况下，短路特性可以节省许多 CPU 周期。同时也能够提高我们的编程效率。

（3） 短路求值还可以防止其他括号内的缩进，从而使代码更具可读性，并且括号之间有累加的趋势。

给程序员带来的好处：

（1） 同上面的好处，程序员使用短路求值的特性能够使得表达式的形式更加简单与简洁，使得代码具有更好的可读性。

（2） 在某些具体情况下，短路求值的特性可能会对程序员编写程序有所帮助：

假设有一个可能为 NULL 或可能不是 NULL 的指针，并且想要检查它不是 NULL，并且它指向的对象不是 0。但是，如果指针为 NULL，则不能取消引用。没有短路求值特性，我们必须执行以下操作：

```
if (a != NULL) {
 if (*a != 0) {
 }
}
```

但是，使用短路求值可以写出如下代码：

```
if (a != NULL && *a != 0) {}
```

能够使代码更加紧凑。

三、 参考资料

助教所写的代码：md-dzy branch

四、 总结

本次编译原理的实验 step4 的重点以及操作步骤大致与 step3 相同。step2-step4 中，我们扩充了二元操作运算符，使得程序的返回值看的更多样化了一些。

能够完成了一些基础的二元运算，总体来说难度不是很大。