编译原理 MiniDecaf 编译器实验
Step 7 实验报告

李祥泽
2018011331
lixiangz18@mails.tsinghua.edu.cn

实验内容

实现块语句和变量的作用域

按实验指导书的指导, 将名称解析与 IR 生成合并来做.

将符号表进行层叠, 形成一个栈. 其顶部为最内层的作用域, 底部为最外层的作用域. 在查找符号时从顶到底依次查找. 在进入块语句时, 向符号表栈中压入一个空的符号表; 在离开时将对应的符号表弹出.

内存分配使用了较紧凑的分配方式, 即反复使用同一片栈空间, 而在函数开头只分配足够 *同时存在的最多变量* 使用的空间. 为此, 需要更改变量计数的方式. 在定义变量时, 变量计数器 +1; 在离开作用域时, 变量计数器减去当前作用域的变量数. 另外维护一个计数器, 统计函数所需的栈空间.

严格来说, 这种能够访问内部的数据结构并不是栈 (按定义, 栈应该只能访问栈顶元素). 但是恰好 Java (由于性能原因) 推荐在不要求线程安全的情况下使用 `LinkedList` (双向链表, 也实现了栈的接口) 来代替栈, 且栈顶位于链表的头部. 这就天然允许我们使用迭代器 for 语法按我们所需的顺序遍历栈中元素.

思考题

请将以下 MiniDecaf 代码中的 `???` 替换为一个 32 位整数, 使程序运行的返回值是 0.

```
1  int main() {
2      int x = ???;
3      if (x) {
4          return x;
5      } else {
6          int x = 2;
7      }
8      return x;
9  }
```

注意到第 6 行的变量声明是包含于一个块语句, 也即位于内部作用域的. 那么该声明不会对第 2 行的变量的值产生影响. 因此取 `??? = 0` 即可.

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?

如果一种语言 *允许变量先使用, 再声明* (例如, JavaScript 的 var), 即一个变量可以在未被声明的情况下被使用, 只要保证在之后的代码中在同一作用域声明该变量即可, 那么将名称解析作为一个单独阶段是更好的.

因为, 如果名称解析与 IR 生成同时进行且不加额外处理, 那么在生成上述情况的 IR 时, 就会用一个未声明的符号去查表, 其结果或者导致出错, 或者引用了外层作用域的 (不该引用的) 符号.

相反, 如果名称解析在 IR 生成之前做, 就能提前对每个作用域生成一张合适且完整的符号表.

Honor Code

主要参考实验指导书实现.