编译原理Lab3实验报告

匡亚明学院
181240035 刘春旭
181240035@smail.nju.edu.cn

实验目标和结果

此时的实验目标是实现中间代码的翻译，已完成全部实验要求以及选做要求3.1和3.2，并且能实现3.1和3.2的结合不会有冲突（即结构体数组和数组结构体）。

代码框架

> 相似的部分不再赘述，下面凸出一些和其他人不同的点

1. 代码结构：

```
.
├──── common.h  // 包含所有结构体的定义和必要的函数声明
├──── main.c    // main函数
├──── lexical.l // Lab1 的语法检查
├──── syntax.y  // Lab1的语法检查
├──── hash.c    // Lab2的语义检查
├──── ir.c      // [新增] Lab3的主体，负责翻译中间代码
└──── struct_symtable.c // [新增] Lab3的辅助代码，会在下面进行说明
```

2. 没有注释掉Lab2中实现的作用域：

   - 从上面的代码结构中可以看出，我比较希望能够保证代码的模块化，所以开始时就尝试了从Lab1已经建好的语法树递归地向下翻译。

- 比较简单的一个思路是直接注释掉Lab2的，这样所的符号都会留在符号表中，然后就可以在Lab2递归一遍树结构之后，在Lab3中重新递归一遍树结构，以达到模块化。但是由于Lab2中已经完成了作用域，丢掉有些可惜。 所以在Lab3中，我选择保留作用域，同时实现一些模块化设计，不需要在Lab2上做太多修改，可以实现代码分开放置，非常整齐。
- 我的思路是这样的：
  1. 每次在删除特定深度的符号表中符号前都对这一部分 `CompSt` 进行翻译，并且把翻译后的 `Intercodes` 存进 `CompSt` 的符号表。
  2. 到外层 `CompSt` 后，将其中间代码和内层 `CompSt` 翻译好的代码连接起来（到了外层说明内层的 `CompSt` 一定已经完成翻译了），这里有一些小细节，在内层翻译为空时需要特殊处理，以防指针跑丢。
  3. 在处理数组和结构体的时候， `translate_Exp()` 能看到的东西只是类似于$Exp_1[Exp_2]$以及$Exp.ID$这样的形式。思路都是先用 `translate_Exp()` 来翻译好前面的Exp_1作为基地址$base$，然后再翻译后面的部分（Exp_2或ID）得到$offset$，最后相加得到一个地址。$base$的翻译是递归，但是$offset$如何得到，这里面有一些只有作用域会引入的细节。分开来分析：
     - 数组类型：
       - 增加Lab2部分符号表记录的信息，记录下数组类型的每一维的长度、大小。
       - 递归地拆开 Exp_1 ，直到它要被拆成 $ID[Exp]$ 或者是 $Exp.ID$ 的形式，前者代表高维数组，后者代表结构体中的数组。总之我们都可以拿到 ID ，这个 ID 一定是数组类型。然后我们根据其类型信息中记录下来的长度和大小就可以求到现在当前数组位置的 $offset$
     - 结构体类型：
       - 这里的难度就是： Exp 当前只能看到 $Exp.ID$ ，能确切知道的内容只有ID，但是在Lab2中，这些ID只有在结构体定义的 `CompSt` 中才会出现，在离开时删除。但是问题在于，我们翻译IR时，是不会翻译到结构体的定义中的。所以必须要做的额外操作就是要把所有的域名以及从结构体变量到域名的 $offset$ 都记录下来，这样我们在查表时就可以看到 ID 就知道 $offset$ 是多少。
       - 在 `struct_symbol_table.c` 中定义了一系列关于上述符号表的操作，在结构体定义时插入，这张表基本就是一张可以实现Lab2基本要求的符号表。这样我们在看到 ID 时就可以直接翻译出 $offset$
  4. 正因为上述处理，我们不仅可以处理数组、结构体，也可以处理结构体数组和数组结构体，互相嵌套的形式也都可以处理。

3. 数组的赋值：类似于 `memcpy()`

```
iter = 0
LABEL loop:
temp = *r
*l = temp
r += 4
l += 4
iter += 4
if iter < size GOTO loop
```

这里$size$的获取复用了上面数组$offset$翻译中的部分流程，同样的原理就可以拿到当前维度的大小信息。

4. 小型优化操作:
   - 当 `translate_Exp` 遇到 `INT` 或 `ID` 时，如果是可以直接返回值的情况，那么将会把传入的 `OP_TEMP` 类型的place直接换成 `OP_VARIABLE` 或者 `OP_CONSTANT` 。
   - 当翻译运算表达式（ `translate_Exp` ）时（除了数组、函数），若 `place` 为空，说明没有地方会用到它的结果，所以运算结果可以不生成（ `place = t (result)` ），这样可以减少一条语句。
5. 像书中一样，函数名称用原名称代替，变量名称换为 `v` ，临时变量名称换为 `t` 。
6. 关于翻译结果是否要返回地址还是返回值，会根据这个地址指向的内容进行判断。如果这个地址指向的内容是BASIC类型，那么就生成一条取值的中间代码；如果指向的内容还是ARRAY或STRUCTURE，那么就直接返回地址。
7. 函数中的参数PARAM，如果是数组、结构体，那么默认传入的是地址，这一点当时是没有注意到的，在报告中记录一下。

测试结果

再次感谢学长学姐们的测试代码，帮助我找到了很多bug，其中又包括指针未初始化带来的麻烦。最终的结果是基本全部可以通过。这里的“基本”是指，对于一条样例 `impossible.cmm` 无法通过，和周围同学的讨论结果也是觉得这份代码impossible。其余（包括超长代码以及结构体和数组的复杂嵌套）均可以通过（除了一份代码 `zzw-1.cmm` 中有违反实验二假设7的部分无法进行处理）。