编译原理 Lab3 报告

171860659 吴紫航 联系邮箱：401986905@qq.com

实验目的

在词法分析、语法分析和语义分析程序的基础上，将 C 源代码翻译成中间代码。中间代码以线性结构输出到虚拟机小程序中运行。

实验要求

1）将符合假设的 c 源代码翻译成中间代码，代码规范按照讲义所给的表格。
2）选做 1 要求考虑结构体的定义、传参和使用。

整体思路

1）首先在语义分析阶段，把 write 和 read 两个外部接口函数加入符号表，保证语义分析顺利通过。
2）计划直接按线性结构存储中间代码
3）实现中间代码构造和打印的一系列数据结构和接口，放在 InterCode 模块中
4）测试保证 InterCode 模块的正确性
5）按照讲义的翻译模式，exp、stmt、cond 等翻译函数。
6）按照自己理解的翻译模式，完成变量定义、变量传参、变量使用、一维数组定义、使用、函数定义、函数调用、结构体定义、结构体传参、结构体使用等翻译函数
7）修改 exp 的赋值指令，对数组和结构体的赋值进行特殊的处理

数据结构

1）Operand
操作数结构，可以是变量、常量、临时变量、标号、函数名、临时变量所表示地址的内容、临时变量的地址、变量的地址
可以直接利用 Operand 构造中间代码

2）InterCode
表示一行中间代码

3）InterCodes
一个线性的双向链表，每个元素都是一行中间代码

4）ValueList
维护中间代码的变量和 c 源代码变量名的映射

5）ArgList
一个操作数的链表，为了解决实参参入指令的生成

关键步骤

为了精简报告，这里仅介绍在实验过程中比较关键的、或有体会的细节或难点。

1）语句和表达式

首先是对讲义给出的参考翻译模式的理解，讲义在思路和实际个人的实现思路略有区别。讲义在生成临时变量后，将临时变量传给子节点处理，子节点生成代码后返回代码给高层，由高层整合。这也就是讲义中 Place 的内涵。而实际实现的时候，直接在子节点生成临时变量、并直接生成线性代码，**去除了不必要的参数传递和返回。**Operand 操作数作为部分翻译函数的返回值，方便上层函数在子节点翻译后进行进一步的翻译。

2）变量生成

变量生成的时候要考虑几个特殊情况。比如：是否是参数定义？是否是结构体定义？是否是一维数组？是否是高位数组（需要报错）？变量的类型是什么？**这些信息需要存储到 ValueList 表中，方便后续的查询和使用。**因为不同的变量使用方法不同，例如结构体变量如果是参数，那么必然是地址的形式。

3）结构体的使用

直接在 exp 翻译模式中完善结构体的使用。即 Exp->Exp DOT ID。这里要分成两种情况，

一种是 Exp 的内容代表一个变量 v，即一个结构体变量内部的成员访问。此时只要在 ValueList 中查询到该变量，获取到变量的 Type 后结合 ID 得到成员的地址偏移量。最后生成临时变量 t1 := &v。t2 := t1+ #offset。然后返回这个地址变量，以后通过*t 的方式使用。当然如果 v 是一个结构体参数的话，则直接 t := v+#offset 即可，因为结构体参数变量就是地址。

另一种情况是 Exp 的内容是临时变量表示的地址所存储的内容。即*t。当访问的结构体变量是一维数组的元素或其他结构体的成员时，就会出现这种情况。此时只要把*t 当成 v，翻译模式跟前者基本一致。值得优化的地方是，t1 := &*t 是没必要的，直接返回 t+offset 这个地址变量即可。因为*t 所对应的 t 就是目标变量的首地址。

4）数组的使用

同样需要在 Exp 中完善数组的使用。即 Exp -> Exp1 LB Exp2 RB。同样要分为两种情况。

一种是 Exp1 的内容代表一个变量 v，即一个数组变量。此时只要在 ValueList 中查询到该变量，获取数组的基本类型的 Type 大小，生成常量#sz。Exp1 和 Exp2 翻译后返回一个临时变量 op1、op2。然后用 t1:=op2 x #sz 得到数组偏移地址。然后 t2 := &op1 得到数组首地址。最后生成临时变量 t3 := t1+t2。然后返回这个地址变量，以后通过*t 的方式使用。

另一种情况是 Exp 的内容是临时变量表示的地址所存储的内容。即*t。当访问的数组变量是其他结构体的成员时，就会出现这种情况。此时只要把*t 当成 v，翻译模式跟前者基本一致。值得优化的地方是，t2 := &*t 是没必要的，只需要 t3=op2 x #sz +op1 即可。

5）结构体和数组的使用的注意事项

有一个隐蔽的难点是，当结构体和数组中间变量从子节点返回后，高层节点如何判断这样一个地址所在的内容是什么类型的呢？也就是说，底层翻译分析，得到一个*t 变量，返回给高层，高层继续分析，如果高层有*t->member 这样的结构体翻译需求，如何知道*t 的类型，从而计算 member 成员的地址偏移量呢？这里个人的实现是在 Operand 数据结构中额外开一个空间 Type* tp,给*t 变量使用（Operand（）.kind=STAR 的情况）。当需要知道*t 变量的类型的时候，就可以直接在操作数结构中获取。当然，我们需要在生成*t 变量的时候，把*t 变量的类型也要传进构造 Operand 的过程之中。

6）结构体和数组的赋值

最后一个难点是结构体和数组的直接赋值。根据助教的要求，**中心思想是 memcpy(a,b,min(len(a),len(b)))。**具体的实现**在 tranlate.c 的 Exp -> ASSIGNOP Exp** 翻译过程中。总体分成**四种大的情况**考虑：目标操作数是 v 或*t，源操作数是 v 或*t。

如果目标操作数和源操作数都是变量 v。那么直接查询 ValueList 中变量的数组规格信息 dimension，如果不是数组，查询结果就是-1，计算的时候当 1 处理，看作一个元素的数组。比较 v1 和 v2 的 dimension，取较小的那个作为拷贝时候的 dim 参数。再通过 ValueList 的变量的类型信息获取单个变量的大小。然后乘以 dim 即获得总大小 totalSize。然后使用一个 for 循环，循环生成拷贝代码。拷贝的翻译模式就是 t1 := &op1 +#i。t2:=&op2+#i。*t1:=*t2。这里有两个小情况要区分，即变量是否是结构体的形参，如果是则 op 本身就是地址而不用再取地址。

如果目标操作数和源操作数都是临时变量地址*t 或一个为临时变量地址、一个为变量的三种情况，就不详细介绍了。具体参见 **tranlate.c 的 Exp -> ASSIGNOP Exp。**中心思想都是先计算两个结构体/数组的较小规格，然后 for 循环生成拷贝的中间代码。临时变量的类型获取前面已经介绍过了，在 Operand 中开辟了 Type*成员供使用。在类型结构中可以很方便的获取结构体的大小和数组的规模。

结果测试

测试的方式是直接用助教提供的 Makefile(没有任何修改)
输入 make clean
输入 make parser
输入 make test

得到测试结果。
注意：因为助教提供的原 makefile 里只对 test1.cmm 进行了测试，所以测试文件的名字是 test1.cmm。若想对其他命名的 cmm 进行测试就需要更改 makefile

补充说明

小组任务编号为 6、选作部分为 3.1。吴紫航 171860659 联系邮箱：401986905@qq.com