编译原理实习 3

中间代码生成

组长：金厦涛
邮箱：jkingxt@gmail.com
手机：15996254843
学号：101220047

组员：赖亨
邮箱：134lhforever@gmail.com
手机：15996272472
学号：101220050

一、 实现的功能及方法

1. 基本实现了将 C--代码翻译成中间代码的功能。
   实现包含但不限于赋值、加减乘除、IF 和 WHILE 语句、数组定义及使用、函数定义及调用、参数列表以及 read 和 write 函数等功能。能够生成对应 C--语法的中间代码，并且能够用虚拟机对中间代码进行执行操作。
2. 支持 C--代码中出现结构体类型，并且结构体变量可以作为函数参数
   我们的代码可以支持结构体定义及使用。如果代码中出现结构体，我们的编译器能够很好地将结构体变量翻译成中间代码，实现要求的 3.1 功能。

为中间代码以及中间代码的操作数都设计了数据结构。

特别地，我们组为每种中间代码类型都设计了结构，因此，输出中间代码时只需要遍历中间代码表，根据表中中间代码的类型进行输出，就能很好地完成任务。

我们也为每个语法单元都设计了翻译函数，因此如果想要生成中间代码，只需要传入语法树的根节点，这些函数就能够相互调用，最后将中间代码链表返回。我们还为中间代码的输出特地写了输出函数。最后将中间代码输出至指定文件。

二、 编译方法

直接使用 Makefile 进行编译，然后./parser op1 op2 执行程序，其中 op1 为输入文件名，op2 为输出文件名。如果能够成功翻译，则能够将输入文件翻译成中间代码，将中间代码翻译至目标文件；如果不成功，则会在控制台中输出提示信息。

三、 实验亮点

1. 将中间代码操作数的名称统一
   因为中间代码的操作数名称包含 t1、v1 以及 label1 等，如果将这些数据都分开，则每次都需要很多操作。我们的解决方案就是将所有的操作数名称都当成是字符串，就能够对这些数据进行统一处理。
2. 中间代码的类型添加了 NONE 类型
   我们设计了空中间代码类型。因为有的函数翻译会返回空代码。之前我们是直接返回 NULL 指针，但是我们发现，NULL 指针的调用以及访问会产生很多不稳定的结果。为了能够统一化处理，我们想到了返回一个空代码类型。这样就能够安全地用链表将中间代码链接起来，而不需要判断是否为空指针。能够更加高效地编写代码。

3. 根据情况对攻略上的翻译部分进行微笑改动
   在攻略上，翻译 Exp→ID 的语法是生成中间代码[place := #value]，但是我们组考虑到上层结构的变量名称其实就是 place，那么我们是否可以少生成一条中间代码，而是直接将更改 place 中的内容，直接把 value 的值传递给上层？于是，在我们的代码中，在这步的翻译中并没有生成中间代码，而是直接将 place 的类型改成 VARIABLE，并且将其名称替换为 ID 节点的名称。这个方法不仅使用在这个翻译语句中，还使用在所有存在 place 生成代码的语句中。
4. 翻译数组时判断是否是多维数组
   我们在语句 VarDec→ID 的翻译的过程中，查表以后判断 ID 的类型，如果类型是 structure，则生成 Dec 指令，而如果类型是多维数组，则在控制台上输出提示信息。
5. 函数定义部分不
   将函数定义分成两个部分，一个是不包含参数、一个是包含参数。如果是不包含参数的函数，则生成 FUNCTION 的中间代码；如果是包含参数的函数定义，除了生成 FUNCTION 代码外，还要翻译参数列表。我们的做法就是扫描符号表，获得名称，最后生成 PARAM 代码。
6. 指针以及引用参数定义
   因为在生成函数的中间代码的时候，传入的数组以及结构体需要是以指针的形式输入，而在函数体内部使用这些变量时，就要使用引用的形式。所以，需要解决指针已经引用的类型表示。我们在操作数的结构里添加了 ADDRESS 以及 REFERENCE 类型。如果是指针类型的使用，则操作数的类型就被设置成 ADDRESS，在输出时，除了输出变量名称，还会在其前面添加&符号；如果是引用类型的使用，则操作数的类型就被设置成 REFERENCE，在输出时，除了输出变量名称，还会在其前面添加*符号。这样就解决这两个结构的表示已经输出问题。
7. 对于 new_temp()函数实现，直接用 static int 型变量存储了申请的 no，然后使用 sprintf 将其和 “t”，一起装入一个 char * 中，返回该数组。new_label()函数也通过类似功能进行实现。

四、 实验遇到的问题及解决方法

1. a > 1 && b > 2 问题。
   语法分析中 如 a > 1 && b > 2 ，期望得到规约：

   Exp -> Exp AND Exp

   因为没有&& 与+-*/的优先级设定，所以会出现如下规约：

   Exp -> Exp (Exp(a > 1) AND(&&) Exp(b)) **RELOP**(>) **Exp**(2)

   解决方案是 需要在 && 前后的表达式上加()，如 (a>1) && (b>2)。
2. 结构体题中成员变量问题。
   本次我们的代码中，对于 struct 变量 能够正确计算其需要的空间(包括成员中有一维数组，结构体)，但是不能对 struct 变量的非 basic 成员进行访问。
3. 起初对攻略中的各个 translate 函数不能理解。
   最开始的时候还不能完全明白 translate 函数的含义，所以读完整个攻略部分，对说明的内容都是半知半解。攻略中说到的 place 的含义就不太清楚。然后，我就花了很长时间，把书上中间代码生成的部分重新看了一遍。懂了生成中间代码的大致流程，这时再来看攻略，就能读懂其中的大部分内容，再来写代码就能够比较轻松。

感悟就是写代码和磨刀砍柴是一个道理，磨刀不误砍柴工，这是有用的真理。正是因为前期花了大量时间弄懂问题的本质，所以后期才能比较正确、成功地完成任务。如果一开始就不顾一切编写代码，可能最后更改的时候就会比较困难以及痛苦。