191098328 计算机科学与技术系 张世茂

邮箱：191098328@smail.nju.edu.cn

在本次实验中，我借助在之前两次实验中进行词法分析、语法分析、语义分析得到的语法树以及符号解析的基础上进一步实现翻译方案，实现了将 C—源代码程序按照一定的规则转化为线性中间代码，并按照实验要求完成了一系列附加要求，包括选做 3.1，添加了对结构体变量的支持。最终的中间代码输出到指定的 ir 文件中，并可以在虚拟机小程序中运行，得到正确的结果。

要实现对 C—程序由源代码到中间代码的转换，需要遵照既定的翻译模式。与实验二类似，由于都是基于语法树遍历的自顶向下分析，因此从整体来看与实验二的函数定义框架是基本一致的。首先根据实验手册上定义出的几种基本语句翻译模式实现针对基本 Exp、条件分支、Stmt 语句等等一系列翻译函数，这些函数将是在翻译其它高层单元时的基石。由于在程序中存在定义变量和临时变量，因此分别使用 new_var 函数和 new_temp 函数对他们进行创建，同时由于定义的变量会被多次使用，因此在创建新的变量之前还将先通过 FindVar 函数尝试进行寻找。同理的还有创建新 label 标识的 new_label 函数。

因为本次实验中约定不出现全局变量的使用，这就意味着最终拿到的源代码是以函数为单位的，因此在翻译中我将 ExtDef 作为翻译的单位，并将最终所有得到的中间代码进行拼接。当然在其他的翻译过程中也有中间代码产生，但此处采取直接调用函数将中间代码加入全局的双向链表中的做法，而其他的翻译过程中则将中间代码链表头指针作为函数返回值返回。实验中的中间代码采用双向链表存储，并通过相关的函数进行维护。

在实现数组和结构体相关的中间代码表示时，由于注意到在函数调用中数组和结构体是以引用的形式传参，因此在 Operand_型结构体中添加了用于判断是否是函数参数的域。同时由于实际程序中定义的变量与中间代码中需要重复使用的 v 型变量的需要，建立了一张两者之间的映射表，方便随时检查和获取对应的中间代码变量。

针对于数组和结构体的赋值取值语句翻译较为特殊，由于已经在语义分析时已经对它们的类型相关信息进行了分析和存储，因此一方面可以通过数组名或结构体变量对应的结构体类型名来获取相关类型所占空间的大小，进而在节点中获取数字索引，通过中间代码取出基地址后结合得到的偏移量计算最终的目标地址。

为增加代码的易读性，增强代码的逻辑性，我在实现翻译程序的过程中对大多数常用的方法进行了较好的封装。包括可以创建 ASSIGN 相关类型的函数 NewAssignInterCodes，可

以对不同的中间代码链表进行连接的函数 ConnectCodes，通过递归求某一特定结构体类型大小的函数 GetStructTypeSize 等。这些封装的方法在一定程度上为我实现和调试中间代码翻译程序提供了帮助。

实验中由于在翻译时对象是碎片化的，因而不仅需要将这些得到的中间代码组装起来，还需要进行值的传递，使得一个语法单元可以获得其需要的与其相关的另一个语法单元计算得到的目标值。通过借助函数的指针类型的 place 参数可以将计算值传给 place 所指向的变量，实现翻译过程中值的传递。

本次实验的调试相比于之前的实验都困难不少，原因主要在于本次实验需要生成真的“能跑起来”并可以得到准确结果的中间代码，这导致之前一些隐藏的错误都暴露出来，且一些实现上的工程细节很难仅仅通过模糊处理解决困难。且本次实验代码量比较大，因此在调试的过程中面对生成的上百行中间代码难以下手，在这个过程中我尝试通过对函数单元进行测试的方式调试，并对不同类型的源程序代码进行总结，使得我可以较为准确地编写想要的测试代码。

编译该程序的方法为：

1.将文件解压后进入到 Code 文件夹下；

2.在终端中输入 make clean 命令并回车，防止有冗余残留的多余文件；

3.在终端中输入 make 命令并回车，对程序进行编译；

4.生成了最终的可执行文件。