编译原理实验报告

实验项目三：中间代码生成

• 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--源代码翻译为中间代码，并将中间代码在虚拟机小程序中运行，检验代码生成是否正确。

• 实验设计

（一）设计思路

我在实验中的思路基本是在理解实验手册上讲解基本思想后进行的，首先按照实验手册上给出的例子分析了一下本次实验的实验要点。分析后我发现主要任务依然是要做语法翻译工作，我依然采用首先遍历语法树的方式进行递归的分析。然后将翻译分析得到的中间代码逐一按顺序记录在一个数组中，当语法树遍历结束之后，把数组中记录的所有中间代码输出到文件中即完成了本次实验中间代码的生成。

（二）数据结构

这次实验数据结构的重点就是每条中间代码的数据结构，我首先在结构中设计了一个枚举类型表示该中间代码的类型，例如赋值语句，运算语句，IF 语句等不同的中间代码类型，然后我用一个联合结构表示每个对应类型需要的各个操作数等信息。中间代码结构如下：

```
typedef struct InterCode
{
    enum { ASSIGN, ADD, SUB, MUL, DIV, LABEL, GOTO, READ, WRITE, IF, RETURN,
    FUNCTION, CALL, ARGS,PARAM, DEC,GETADDR,LEFTADDR,RIGHTADDR,GAP,ARRAYGAP,
    TEMPGAP} kind;
    union {
        struct { Operand right, left; } assign;
        struct { Operand result, op1, op2; } binop;
        struct { Operand op; char func_name[30];}callop;
        struct { Operand right, left; int type; int label;} ifGotoOp;
        Operand op;
        char funcName[30];
        char varName[30];
        int label;
        struct{int size; Operand op;}decop;
        struct{int gap; Operand left,right;}getaddrop;
    } u;
    int sign;
}InterCode;
```

图 1 中间代码结构

在每个中间代码中需要的操作数同样需要一个数据结构来表示，所以我设计了一个操作数 Operend 结构，用来记录每条中间代码中需要的操作数，这个结构中首先需要包含一个枚举类型 kind，用来表示操作数的类型来区分常量，变量，临时变量等操作数类型，之后用一个结构记录其各个操作数需要的名字，数值等信息，其结构体如下图：

```
typedef struct Operand_ {
    enum { VARIABLE, CONSTANT, ADDRESS, TEMP,ARRAY, STRUCTURE} kind;
    struct {
        int var_no;
        int value;
        char var_name[30];
        structCode s;
        int is_param;
        int array_size;
        Operand addr;
        /*, … */
    } u;
}Operand_;
```

图 2 操作数类型

（三）功能实现

（1）在定义好数据结构之后，就要开始中间代码生成的工作了。这一工作说起来有些部分要比第二次试验简单一些，我们在翻译语法树时不需要再记录继承属性，也就是需要翻译的部分减少了；但是他也有更加复杂的部分，因为操作数的种类很多，每一种操作数创建的方式都有所不同，需要记录的信息也不同，所有在创建一条中间代码里包含的操作数时，需要对更多种情况进行讨论。

（2）对于中间代码的记录，我采用的最简单的线性数组 codes[n]的方式，这种方式表示起来简单，添加时只需要对中间代码的数量进行考虑即可，但最需要注意的就是中间代码添加的位置，避免出现代码的顺序错误。

（3）本次实验的一般的运算语句实现起来比较简单，只需要在翻译 EXP 时，在各个运算符情况出现时，进行判断，并将运算符信息和操作数信息保存，顺次放入中间代码的数组中即可。而对于赋值和判断语句需要处理的就多了一些，因为往往需要等待处理完每个表达式得到临时变量的信息之后才可以进行下一步操作。

（4）在条件语句的翻译上也和实验 2 有了很大的不同，因为在 IF 或 While 语句中需要创建 label 代码和跳转代码，所以这里我的处理方式是直接在处理 stmt 语句时，直接创建 label 语句以及跳转语句，并将后面所有的会发生的判断操作都在这里进行分析。然后按照课件中给出的翻译模式按顺序向中间代码数组里添加中间代码，即完成了对条件语句和循环语句的处理。

（四）代码优化

1.首先，我优化了操作数是一个常数的情况，按照最原始的情况，程序会首先创建一个临时变量来记录常量的值，但是这样的方式会增加一行代码。经过改进，可以将这行代码删掉，直接在其他语句中直接用#n 来使用常量值。

2.同理，我将变量 v 的表示同样用上述方式直接在语句中引用，不需要再创建一条语句将变量赋给一个临时变量。

3.我在记录中间代码时，会对是否有重复代码进行检验，与之前记录的所有代码进行对照，如果重复则不会将这条代码记录进去，从而减少代码数量。

• 实验中遇到问题及解决办法

（1） 本次实验总体思路和上一次实验类似，所以上手比较快，可以很快的进行设计。但是，做到实验后期发现有些地方需要考虑的地方很多，所以有的一些之前的设计需要进行修改，比如中间代码中地址类型的数据，添加了地址类型的操作数之后需要对一些运算里的保存信息进行调整，这些都增加了我完成实验的难度。

（2） 本次实验的结构体和数组中间代码的生成是实验难点，因为这两部分的代码生成都涉及到地址的赋值和使用。这里我在最后开始设计的时候没有理解中间代码中取地址和取值的使用。在理解了取值和取地址的使用后，设计了几种中间代码类型分别表示地址偏移的中间代码，取地址和取值的中间代码。

（3） 因为我的选做任务是选做部分 1，所以在数组的翻译上只需要考虑一维数组，因此我在翻译时只考虑了一次数组的情况，其他情况如果出现直接输出错误信息。我的选做任务中需要完成结构体的设计，这部分的设计主要是要复杂一些，因为我在设计结构体之前都没有使用到任何的继承属性，所以当一个 Specifier 是一个定义好的结构体时，我无法将结构体的信息传到变量中去，所以这里我没有采用之前设计好的 VarDec 等语句的翻译，而是单独重新设计了 struct 的变量翻译，从而解决这选做问题。

• 实验总结

经过本次实验，了解了编译器中语义分析和类型检查实现的基本原理，虽然我设计的中间代码生成肯定还存在着不足，但我从中确实学习到了很多。从自己设计一个系统的数据结构，到递归的生成中间代码，其中都让我在对原有知识的理解更加深入。尤其是对以前写代码中常有的指针取值取地址的问题有了更深的认识，从原理上进行学习之后，更有助于我们平时高级语言的代码设计。

• 编译方法

（1） 进入 Lab/Code 目录

（2） 在命令行中输入：make parser，生成可执行文件 parser

（3） 在命令行中输入：./parser <测试文件路径> <输出文件路径>，对测试文件进行中间代码生成输出到输出文件中。