Lab3 中间代码生成

史子凡　151180115　vivianszf9@gmail.com

一、 实验进度。

完成了实验三必做和选做部分要求的全部内容，包括最普通的中间代码生成，可以出现结构体类型和高维数组类型的变量，结构体类型的变量和一维数组类型的变量可以作为函数参数。

二、 编译方式。

根目录下写有 Makefile 文件，提供以下编译方式：

1. make lab3: 即编译所有与 lab3 相关的文件并生成所需要的 parser 可执行文件；
2. make parser: 对 main.c, syntax.tab.c, treeop.c, symbol_table.c，semantic.c，intercode.c 还有 translate.c 进行联合编译，生成可执行文件 parser；
3. make clean: 删除 parser 文件；
4. make testcN：N 用 1 到 2 的数字替代，是对 test/compulsory/下的第 N 个必做样例进行测试，并输出文件 cN.ir 到 output 文件夹下。比如，make testc2 表示对 test/compulsory/2.cmm 进行测试，输出文件路径为 output/c2.ir。
5. make testoN：N 用 1 到 2 的数字替代，是对 test/others/下的第 N 个选做样例进行测试，并输出文件 oN.ir 到 output 文件夹下。比如，make testo2 表示对 test/others/2.cmm 进行测试，输出文件路径为 output/o2.ir。
6. make testmN: N 用 1 到 2 的数字替代，测试自己写的 test/my/N.cmm 这个测试样例，并输出文件 output/mN.ir。
7. make testall: 对所有测试样例（2 个必做+2 个选做+2 个自己的测试样例）进行测试，生成的文件都在 output 文件夹里。
8. ./parser input output: 读入 input 测试文件，生成 output 文件。

三、 实现细节。

1. 总体思路

由于之前在实验二中实现中离开作用域的时候会删除相关的符号表，所以我选择了一边进行语义检查一边生成中间代码。因为一些文法翻译和语义分析有区别，所以部分文法翻译的接口写在了 translate.c 中，其余的直接在 semantic.c 中。大体翻译方式与讲义提供的翻译模式一致。

2. 数据结构

Operand 的结构和讲义一致，不过多加了一个 adornot 是为了函数传参数的时候判断是不是要进行取地址操作：

```
struct Operand{
    enum{OVAR,OCONSTANT,OTEMP,OLABEL,OFUNC,OAD,OST,ONULL} kind;
    union{
        int value;
        char *name;
    };
    int adornot;
    int n;
};
```

中间代码的数据结构和讲义上基本一致，但因为是三地址代码，所以最多只有三个操作数，所以只要用 re,op1,op2 就足够了，不需要再用 union 结构。relop 用来记录比较符号是哪一个，size 记录结构体或者数组的大小：

```
struct Intercode{
    enum
{ILABEL,IFUNC,IASSIGN,IADD,ISUB,IMUL,IDIV,IAD,IST,IIF,IRETURN,IGOTO,IDEC,IARG,ICAI
kind;
    Operand *re,*op1,*op2;
    int relop,size;
};
```

除此之外还有两个双向链表结构用来分别连接 Operand 还有 Intercode。

对实验二的数据结构也做了一些修改，在 symbol 中加入了 Operand，在 type 中加入 mainornot 用于判断结构体或者数组是在函数里声明的还是作为参数传入的。

3. 选做内容（结构体和数组）

由于结构体和数组涉及到了取地址，所以专门写了一个 AS_translate 函数来对它们进行单独的处理。结构体和数组中的地址偏移的计算写在了 translate.c 中的 get_field_offset 和 get_size 两个函数中，用的方法就是讲义中提到的顺序计算的方法。比较麻烦的是将结构体和数组作为参数传入函数，因为传入的时候需要取地址，接受的函数需要取出传入的结构体或数组的内容，与在函数体里声明并使用结构体或数组有点区别。我用的方法是，在文法分析的时候如果这个结构体或数组是作为参数的那么给它一个标记，也就是 adornot 和 mainornot 两个标记（前面数据结构部分提到了），其中 adornot 用来记录作为 args 被传入函数的结构体或数组，mainornot 用来记录作为 params 的结构体或数组。在后面翻译的时候根据这两个的值进行特殊的处理。

4. 优化

主要做了两个优化，一个是对于立即数就不用产生一个临时变量，先赋值给临时变量再传给真正需要的 place 完全是多此一举，直接将立即数赋给最终需要的变量(位置)即可。第二个是对于常数的负值，不需要先用常数 0 减它，直接在遇到 MINUS Exp 的时候判断 Exp 如果是常数就直接返回常数的负值就可以了。

5. 中间代码打印

直接遍历由双向链表连接的 Intercode，因为在之前生成代码的时候已经记录了每种 Intercode 的类型，所以只需要从头开始遍历链表打印出对应的代码即可，其中 IPARAMAD 和 IARGAD 两种中间代码是为了特殊处理结构体和数组作为函数参数的情况的。

四、 实验总结。

这次实验照着实验讲义提供的模式写还是不是很麻烦的，不过由于一些翻译的过程需要对前面的实验的数据结构进行一些修改还是挺麻烦的，需要对前面涉及的这些结构的代码也进行相应的更改。

此外，这次实验最容易出错的地方仍然是和链表相关的操作，最开始我只是用了 Operand*作为 Exp_translate 函数的一个参数，结果编译的时候就报错一大堆，后来才发现应该用 Operand **才能进行后面对 Operand 的一系列操作。Segmentation fault 仍然是出现最多的 bug，最终原因都是忘了判断是否是空指针就开始进行操作或者忘记分配内存空间了。