中间代码生成实验报告

151220125 吴佳玮

邮箱：498279281@qq.com

一、 完成功能

实现了对C—语言的中间代码生成，能够翻译包含一维数组类型的变量可以作为函数参数，可以出现高维数组类型的变量的代码，修正了部分语义分析阶段遗留的错误

二、 实验环境

实验编写在 MacOS 系统下，使用flex 2.5.35和bison 2.3编译通过并运行。提交之前在ubuntu 16.04环境下编译通过并测试成功

//macOS下编译需要-ll参数而非-lfl参数，在提交前已将Makefile恢复为-lfl

三、 实现解释

1、操作值表示

```
typedef struct Operand_{
    enum{O_TEMPVAR,O_VARIABLE,O_CONSTANT,O_LABEL,O_FUNCTION,O_ADDRESS,O_NULL } kind;

    union{
        int var_no;
        int value;
        char name[32];
    }u;

    Operand *next;
} Operand_;
```

这是中间代码的变量的数据结构，其中类型分别为临时变量，变量，立即数，Label名，函数名，地址（用于函数中传入识别传入地址），空类型。

其中空类型是为了防止出现段错误而加入的，在优化中去去除包含该类型的代码。变量名和函数名和地址名均直接使用代码中的名称，临时变量为t0，t1 …., label名为l1,l2,l3…。

2、中间代码

```
typedef struct InterCode_{
    enum{I_LABEL,I_FUNCTION,I_ASSIGN,I_ADD,I_MINUS,I_STAR,I_DIV,I_GETADDR,I_GETVALUE,I_PUTVALUE,
        I_GOTO,I_IFGOTO,I_RETURN,I_DEC,I_ARG,I_CALL,I_PARAM,I_READ,I_WRITE}kind;

    union{
        struct{ Operand x; }sinop;

        struct{ Operand x,y; }binop;

        struct{ Operand x; Operand y; Operand z; }triop;

        struct{ Operand x; Operand y; Operand z; int relop;} forop;

        struct{ Operand op; int size; }dec;
    }u;

    InterCode* pre;
    InterCode* next;
} InterCode_;
```

中间代码的数据结构如上图，采用双向链表记录全部代码，所有类型与书上给的中间代码指令一致。

中间代码生成

在语义分析的同时进行中间代码生成，生成方式与书上的翻译模板一致，此外为了实现要求3.2，实现了额外操作：

1. 声明时加入数组声明的中间代码
2. 检测到多维数组时，对上一维度的数组空间*位置得到偏移量，在原地址基础上加上偏移量得到新的地址
3. 调用函数，压入变量时如果是数组变量，则直接压入数组的地址
4. 在函数内部，若数组变量是本地声明，则要先取地址再操作，而若是形参传入则无需再取地址

特殊思路

1） 为了屏蔽掉带有结构体变量的代码，在语义分析之前直接扫描语法树来判断

2） 实验中若遇到需要place = NULL的情况，则传入kind = O_NULL的变量，先加入进中间代码再通过优化去除

3）代码优化思路如下

i. 扫描全部代码，去掉所有包含O_NULL的代码

ii. 创建一个与临时变量数目相等的数组，扫描全部代码，若遇到形如

“临时变量 := 立即数” 或者 “临时变量 := 变量”

在数组中对应临时变量的位置中存入相应立即数或变量，删除此行代码

iii. 扫描全部代码，对每个临时变量

如果 数组中该临时变量的位置不为空，则将该临时变量改为对应立即数或变量

对所有算数表达式代码，如果其两个操作数均为立即数，则将该值计算出来并将原代码变为赋值代码

iv. 重复ii. iii.步骤直至代码中不存在上述形式的代码。

运行效果