编译原理第三次实验

实验报告

171860691 王辛有 171860005 宋斯涵

一、 实现功能

1） 生成三地址代码：输入 C--源代码，输出三地址代码文件
2） 指选内容：允许 C--源代码中有结构体，以及结构体变量作为函数的参数
3） 拓展内容：允许内部以数组作为域的结构体以及结构体数组的存在

二、 重要的数据结构、函数和变量说明

（1）中间代码的结构：

```
typedef struct Operand_{
  enum{TEMP_VAR,VARIABLE,CONSTANT,LABEL,MYFUNCTION,VAR_ADDR,TEMP_ADDR,MYSTAR,NONE}kind;
  union{
    int var_no;//CONSTANT
    char value[32];
    Operand addr;
  }u;
} Operand_;

typedef struct InterCode_{
  enum{W_LABEL,W_FUNCTION,W_ASSIGN,W_ADD,W_SUB,W_MUL,W_DIV,W_GETADDR,W_GETVALUE,W_VALUEGOT,W_GOTO,
  W_IFGOTO,W_RETURN,W_DEC,W_ARG,W_CALL,W_PARAM,W_READ,W_WRITE}kind;
  union{
    struct{ Operand op; }Single;
    struct{ Operand op1,op2,result; }Double;
    struct{ Operand x; Operand y; Operand label; char relop[32]; }Three;
    struct{ Operand left,right; }Assign;
    struct{ Operand op; int size; }Dec;
  }u;
  InterCode pre;
  InterCode next;
} InterCode_;
```

InterCode_表示单条的中间代码，将中间代码分成了 19 种（与书上的划分一致），每一种的操作数有所不同，依据操作数个数和种类的不同又划分为 Singl，Double，Assign 等类型；
Operand_表示一个操作数，不同类型的操作数有不同的属性，用 union 来表示不同的属性；
中间代码的结构用双向链表实现，我们采用了在语义检查完成之后再生成中间代码的方式并且为每条中间代码的插入和删除分别定义了函数：
void Inter_delete(InterCode a);
void Inter_insert(InterCode a);

三、 实现方法

（一）翻译模式

对于基本表达式，条件表达式的翻译，采用了书上给出的翻译模式，此处不再赘述，而对于结构体和数组的翻译，我们对 Operand 增加了新类型：MYSTAR 和 VAR_ADDR（TEMP_ADDR），VAR_ADDR（TEMP_ADDR）表示对变量进行取地址操作，MYSTAR 表示对指针进行取值操作。
此外，还需要注意当结构体变量是函数内部变量和形参变量时的情况不同，当结构体 a 是形参变量时，采用了引用传递的方式，a 表示一个地址，而当结构体 a 是函数内部变量，a 表示一个变量的名字，&a 才是取地址操作。

（二）中间代码优化

我们对中间代码分别进行了以下几个方面的优化：

1．去除同一位置的 LABEL：

```
WRITE #31
LABEL label33 :
LABEL label30 :
LABEL label21 :
LABEL label3 :
RETURN #0
```

对于图中所示的这种多个 LABEL 指示的是同一位置的情况，我们仅保留了第一个 label，将下面的 label 都删除，并且将 GOTO 目标是下面几条 label 的 goto 指令的 GOTO 目标改为第一个 label。

2.去除多余的跳转指令：

对 relop 取反，将[IF op1 relop op2 GOTO label_true GOTO label_false LABEL label_true:] 转化为 [IF !op1 relop op2 GOTO label_false LABEL label_true:]

3. 去除多余的 LABEL 指令：

遍历所有中间代码，如果存在某些 LABEL 不是任一 goto 指令的跳转目标时，这些 LABEL 是多余的，就将这些 LABEL 去掉。

在测试中发现，若 2,3 结合会使得执行的中间代码条数减少，但若只保留一个，则执行的代码条数不会减少，可能还会增多；

4. 常量折叠：

对于预先可以计算出的表达式，在优化时我们会直接计算出其值，并在用到这些表达式的时候直接用计算出的值来替换；

5. 消除公共子表达式：

在消除公共子表达式时，首先将程序划分为基本块，我们只在基本块的内部进行了公共子表达式的消除，因为在 if-else 等语句中可能会使得公共子表达式的值改变，所以并没有在全局消除公共子表达式；此外，由于数组元素的赋值与使用也存在着不确定性，我们没有对数组的取地址操作进行公共子表达式的消除；

四、 编译指令

1. bison -d -v syntax.y
2. flex lexical.l
3. gcc main.c SymbolTable.c SemanticAnalysis.c HelpFunc.c tree.c syntax.tab.c Intercode.c -lfl -ly -o parser