编译原理lab3中间代码生成实验报告

201840009田永上

实验内容

生成中间代码，满足以下七条假设

\1) **假设**1：**不会出现注释、八进制或十六进制整型常数、浮点型常数或者变量。

\2) **假设**2：**不会出现类型为结构体或高维数组（高于1维的数组）的变量。

\3) **假设**3：**任何函数参数都只能为简单变量，也就是说，结构体和数组都不会作为参数传入函数中。

\4) **假设**4：**没有全局变量的使用，并且所有变量均不重名。

\5) **假设**5：**函数不会返回结构体或数组类型的值。

\6) **假设**6：**函数只会进行一次定义（没有函数声明）。

\7) **假设**7：**输入文件中不包含任何词法、语法或语义错误（函数也必有return语句）。

此外，选做任务3.1要求如下

要求3.1**：修改前面对C--源代码的假设2和3，使源代码中：

a) 可以出现结构体类型的变量（但不会有结构体变量之间直接赋值）。

b) 结构体类型的变量可以作为函数的参数（但函数不会返回结构体类型的值）。

数据结构使用

```
//Operand
struct Operand_
{
    enum
    {
        OP_VARIABLE,
        OP_CONSTANT,
        OP_ADDRESS,
        OP_LABEL,
        OP_FUNCTION,
        OP_ARRAY,
        OP_STRUCTURE,
        OP_TMP,
    } kind;
    union
    {
        int no_val;
        int value;
        char name[50];
        int no_addr;
    } u;
    Type type;          // 存储结构体或者数组
    int size;           // 如果存储数组,那么就存储大小
    bool is_structure; // 表示是不是结构体
};
```

```
//IR
struct InterCode_
{
    enum
    {
        IR_LABEL,
        IR_FUNCTION,
        IR_ASSIGN,
        IR_ADD,
        IR_SUB,
        IR_MUL,
        IR_DIV,
        IR_ADDR,
        IR_STAREQUAL, //*x=y
        IR_EQUALSTAR, // x=*y
        IR_GOTO,
        IR_RETURN,
        IR_ARG,
        IR_PARAM,
        IR_READ,
        IR_WRITE,
        IR_DEC,
        IR_CALL,
        IR_IFGOTO,
    } kind;
    union
    {
        struct
        {
            Operand right, left;
        } assign;
        struct
        {
            Operand res, op1, op2;
        } binop;
        struct
        {
            Operand op;
        } sinop;
        struct
        {
            Operand x, y, z;
            char relop[50];
        } gotorelop;
        struct
        {
            Operand op;
            int size;
        } decop;
    } u;
};
```

```
//IR链表
struct InterCodes_
{
    InterCode code;
    InterCodes prev, next;
};
```

辅助函数创建

主要如下

```
Operand new_temp();
Operand new_addr();
Operand new_label();
void make_assign(int ir_kind, Operand op1, Operand op2);
void make_binop(int ir_kind, Operand res, Operand op1, Operand op2);
void make_sinop(int ir_kind, Operand op);
void make_gotorelop(int ir_kind, Operand op1, Operand op2, Operand op3, char *name);
void make_decop(int ir_kind, Operand op, int size);
InterCodes make_intercodes(InterCode ir_code, InterCodes prev_codes, InterCodes
next_codes);
void addtocodes(InterCodes root_head, InterCode ir_code);
Operand make_operand(int op_kind, int val, int number, char *name);
InterCode make_ir(InterCodes root_head, int ir_kind, ...);
```

前三个主要是为了切合实验指导中的代码而创建的，其实并没有必要

中间五个是针对五种IR类型创建中间代码的函数

make_intercodes与addtocodes配合使用，用于把IR加入双向链表中

make_operand和make_ir顾名思义创建operand和IR

代码特点

没有特点，一切都是按实验指导上来的，实验指导上没有的就生成最为朴素的中间代码

编译方式

```
make//编译整个项目,生成parser可执行文件
./parser [testfile.cmm] [savepath.ir]
make lab3test//我修改了makefile文件,以进行测试
```

```
lab3test:
    ./parser Test/M-1.cmm m1.ir
    ./parser Test/M-2.cmm m2.ir
    ./parser Test/O-1.cmm o1.ir
    ./parser Test/O-2.cmm o2.ir
```

实验感悟

这一次实验有实验手册中translate_Exp,translate_stmt,translate_cond,translate_args的示例,降低了设计的难度,除去这几个函数外其余函数中间代码生成部分都不太复杂。

但是debug依然是一个非常痛苦的过程，时常会出现段错误，有多种原因，可能是历史遗留问题，如实验1，实验2中没有存储对应内容，需要不上，另外还有可能if else条件设置不当，导致没有进入正确的分支，而且在某些情况下中间代码还能正确运行，对于这些情况，我只能写一个简单的样例，然后手动生成一下大概的中间代码，然后比对程序生成的结果判断可能是哪里出错了。

在代码中需要不断添加printf打印现有信息来判断究竟是前面的代码没有正确传递数据还是实现错误。