编译原理实验3实验报告

功能及实现方式

核心逻辑：ir代码的生成

我们在`cmm2ir.h/.c`中实现了生成中间代码的接口和具体实现，其中用到的模块参见 模块介绍

- 代码的核心逻辑位于`cmm2ir.h/.c`中，同样是调用`ast_walk`函数来dfs遍历ast树，生成中间代码
    - 其中前序遍历处理函数定义
    - 表达式、赋值、函数调用等语句位于后序遍历处理
- 需要转换成ir的节点类型有：
    - `AST_NODE_Exp`：表达式
    - `AST_NODE_Stmt`：语句
    - `AST_NODE_StmtList`：语句列表，其`ir_code_block_t`存储了其stmt的ir
    - `AST_NODE_VarDec`：变量声明
    - `AST_NODE_Dec`：
    - `AST_NODE_DecList`：变量声明列表
    - `AST_NODE_DefList`：声明列表
    - `AST_NODE_CompSt`：复合语句
    - `AST_NODE_ExtDef`：外部定义
    - `AST_NODE_ExtDefList`：外部定义列表，需要对ExtDef的ir进行连接
    - `AST_NODE_Args`：函数参数
- 如何创建ir并连接呢?
    - 在语义分析时收集信息、例如ID的name、type、size等信息，用于后面转换成ir的相关信息
    - 最**底层**的基本的非终结符申请`ir_code_t`/`ir_code_block_t`，**高层**的非终结符根据产生式体申请`ir_code_block_t`并连接/添加产生式体中非终结符的`ir_code_t`/`ir_code_block_t`

      > e.g.
      > DecList -> Dec
      > | Dec COMMA DecList
      >
      > - 当匹配产生式为`DecList->Dec`时直接将DecList的`ir_code_block_t`指向Dec的`ir_code_block_t`，而不是申请新的`ir_code_block_t`
      > - 当匹配产生式为`DecList->Dec COMMA DecList1`时，申请一个新的`ir_code_block_t`，并将其指向Dec的`ir_code_block_t`，然后将DecList1的`ir_code_block_t`连接到新申请的`ir_code_block_t`上
    - 在`ir_code_block_t`中实现了链表数据结构，来存储一个非终结符的所有`ir_code_t`
- 最后交给`ir_dump`函数输出`AST_NODE_Program`的ir代码，即整个程序的ir代码

if-else和while的翻译

- 因与课本略有不同，故详细说明，我们的实现方案为布尔表达式节省一个label，但是多使用了一个临时变量
- **布尔表达式的处理**
  - 我们的写法为将一个布尔表达式作为一个值来求值处理，并且申请一个**新的临时变量来代表布尔表达式的t/f**，同时我们不使用true/false的label跳转
  - 实现方式为在条件跳转`relop_goto`的ir前为临时变量赋值1
  - 如果条件成立，则条件跳过临时变量赋值为0的语句，直接跳转到后面的ir块
  - 如果不成立，则临时变量赋值为0，赋值之后继续执行后面的ir块
- **if语句的翻译**
  - 同样的，我们先对布尔表达式进行翻译，得到存储布尔表达式值的临时变量
  - 无else的情况
    - 我们为false的情况申请一个label，为`false_branch`
    - 最终本非终结变量`stmt`的ir代码按顺序列出如下：
      - 布尔表达式的ir代码
      - 子Stmt的ir代码
      - label false_branch
    - 从而实现了if语句的翻译
  - 有else的情况
    - 我们为false的情况（此处即指向else分支）申请一个label，为`false_branch`，为离开if-else结构申请一个label，为`go_out`
    - 最终本非终结变量`stmt`的ir代码按顺序列出如下：
      - 布尔表达式的ir代码
      - if分支Stmt的ir代码
      - 无条件跳转至 go_out 的ir代码(因不用执行else分支)
      - label false_branch
      - else分支Stmt的ir代码
      - label go_out
    - 从而实现了if-else语句的翻译
- **while语句的翻译**
  - 我们为while的代码最开头申请一个label `very_begin`，为离开while结构申请一个label `go_out`.
  - 最终本非终结变量`stmt`的ir代码按顺序列出如下：
    - label very_begin
    - 布尔表达式的ir代码
    - 条件跳转至 go_out 的ir代码
    - 子Stmt的ir代码
    - 无条件跳转至 very_begin 的ir代码(因需要重新判断while条件)
    - label go_out

模块介绍

`ir模块(ir.h/.c)`

- 提供了描述具体一句ir的`ir_code_t`以及描述一个非终结符的`ir_code_block_t`
  - 其中`ir_code_t`包含了该语句的操作码、操作数、目标寄存器、函数名、返回值等信息，同时还实现了链表数据结构。`ir_code_block_t`即使用链表数据结构管理一个非终结符包含的所有`ir_code_t`
- 提供了申请ir变量的接口

    - identifier `struct ir_variable_t* ir_get_id_variable(const char *id);`
    - 引用 `struct ir_variable_t* ir_get_ref_variable(struct ir_variable_t *var);`
    - 整形变量、浮点变量 `struct ir_variable_t* ir_get_int_variable(int i);`
    - 临时变量等等 `struct ir_variable_t* ir_new_temp_variable();`
- 提供了申请`ir_code_t`的接口，为函数声明、return、变量声明等语句生成相应的ir
    - 基本声明语句
        - 函数声明 `struct ir_code_t* ir_new_code_fundec(const char *fun_name)`
        - return `struct ir_code_t* ir_new_code_return(struct ir_variable_t *var)`
        - 声明（用于数组、结构体等需要申请空间的数据结构） `struct ir_code_t* ir_new_code_dec(const char *dec_name, int dec_size)`
    - relop、if-else、while相关
        - label `struct ir_code_t* ir_new_code_label()`
        - 跳转 `struct ir_code_t* ir_new_code_goto(struct ir_code_t *goto_dest)`
        - 条件跳转 `struct ir_code_t* ir_new_code_relop_goto(...)`
- 提供了输出一个`ir_code_block_t`的接口 `ir_dump(FILE* file, struct ir_code_block_t *block)`

编译相关

环境要求

- ubuntu 20.04
- flex 2.6.4
- bison 3.5.1
- build-essential

编译指令

```
cd Code
make parser
```

最后生成的可执行文件为`parser`，可以通过`./parser /path/to/input_file /path/to/output_file`运行