lab3 实验报告

学号 PB20081590 姓名 吕凯盛

实验要求

1. 修改 `src/cminusfc/cminusf_builder.cpp` 来实现 IR 自动产生的算法，使得它能正确编译任何合法的 cminus-f 程序。
2. 在 `report.md` 中解释你的设计，遇到的困难和解决方案

实验难点

1. 不知道从何下手

   一开始拿到题目的时候完全不知道怎么做，后面是从后往前做，先把几种基础的翻译出来，才有了一点灵感。

2. 不会怎么调试，对多文件编译后的调试一窍不通

   经过助教指导后才会怎么调试

实验设计

请写明为了顺利完成本次实验，加入了哪些亮点设计，并对这些设计进行解释。
可能的阐述方向有:

1. 如何设计全局变量
   1. `val` 变量，用于储存子结点访问的结果
   2. 作用域标识符 `in_fun_scope` ,用于标记是否属于函数作用域，若不是则代表不是函数声明作用域里面的复合语句，那就要再开一个作用域。

      若在函数里面，则不用新开作用域(因为已经在函数识别中设置了新的作用域)
   3. `left_hand` , `right_hand` ,用于标记是左值还是右值。这在赋值语句中使用
   4. `tmp_arg` ,这是一个临时储存函数调用变量的值。即在 `fun declare` 中，把参数传递给 `param` 的识别函数的变量
2. 遇到的难点以及解决方案
   1. 一开始没看懂复合语句需要处理的特殊情况是什么，后面看了实验文档才知道是可能产生新的作用域

      因此需要先判断是否属于是函数声明时的作用域，如果是就不用产生新的作用域，如果不是，则需要创建新的作用域(因为可能会在复合语句里面声明新的同名变量)
   2. `var` 结点的识别，不知道怎么处理指向数组首地址的指针。后面参考了第二次实验自己写的对于数组类型的处理

      `var` 结点分数组类型和非数组类型

      对于非数组类型，即(`node.expression` == `nullptr`)的，分为整型，浮点型，指向整型/浮点型的指针，一个指向指向整型/浮点型的指针的指针(类似于a[])。

      前三种可以直接加载，后面一种需要求得第一个数组的地址

      对于数组类型，(即 `node.expression` != `nullptr`)的，先求 `node.expression` 的值，再判断是否越界，越界则调用异常，不越界则继续识别。

      还是分为是否为指向指针的指针考虑，处理方法和前面相似
   3. 链式赋值的解决

      定义 `left_hand` , `right_hand` ,用于标记是左值还是右值。

      若是左值，就加载，否则只传值。注意都要预先初始化为 `false` 不然导致类型出问题
   4. 强制类型转换。

1. 主要是接口没看懂，不知道为什么会出现两个参数，明明一个参数就能解决了。。。
2. bool类型需要向i32转换，因为bool类型是i1，需要先做一个零拓展到32位

5. 几个简单表达式的翻译。

   需要考虑左右的类型问题，一样则可以直接算，不一样需要转浮点型。

   然后根据 `node.op` 去选择生成的指令即可

6. 函数调用的翻译，主要是接口好多好杂，搞不懂

   需要创建一个参数vector，对 `node.args` 的每一个进行分析。可以用**fun->get_function_type()->get_param_type(count)**去获取第count个的参数类型，然后进行比较，可以确定是否需要强制类型转换。

7. 自动补全和调试的配置

   发现vscode可以直接连接上容器进行实验，代码补全，调试配置的效果很好，就是docker太卡了

3. 如何降低生成 IR 中的冗余

   没考虑过。。。

实验总结

这次实验写了很久，很痛苦。一开始完全不知道怎么入手，看到那么多的接口完全不知道怎么操作。参考了计算器的实现，把几个简单表达式，常数识别做了出来。才弄清楚访问者模式到底是什么。之后才比较顺畅地做完了。不过调试上还是花了很多时间，各种样例也比较好地让我发现自己程序上的错误。完成以后，我感觉收获满满，对我们中间代码的翻译过程有了更多的理解。也更让我感觉到编译器的神奇和强大。

实验反馈 （可选 不计入评分）

1. 希望可以像cs61a那样分任务完成，然后每一个任务都有自动评判脚本，不然不知道怎么入手，这样也会更友好一些。