

# 实验内容

---

## Step 9

在文法中增加 `parameter_list` 和 `expression_list` 完成对函数定义和函数调用的解析。

增加 `CallExpr` 结点，即函数调用表达式，`CallExpr` 中包含 `ExprList` 即调用时其传递的每一个参数为一个表达式构成。

TAC 中增加 `CALL` 和 `PARAM`，`CALL` 表示函数调用，`PARAM` 表示参数，调用函数时一定为如下结构：

```
PARAM T?  
...  
PARAM T?  
T? = CALL ...
```

即 `CALL` 之前一定为若干连续的 `PARAM`。

修改对应的数据流，增加 `genCall` 等。

在生成目标代码时，先将活跃变量逐个入栈，获取到 `TAC::PARAM` 时无任何操作，获取到 `TAC::CALL` 时，顺着链往前查看 `PARAM`，依次入栈，并计数为 `count`，执行 `call` 指令，根据 `count` 修改栈，然后将活跃变量逐个出栈，最后把返回值赋值给相应寄存器，即完成 `TAC::CALL`。

## Step 10

增加一个 `kind: GLOBAL`，`as: GlobalVar globalVar`，`TransHelper` 中增加 `genGlobalVariable`。同时每次对于变量使用前都需要通过 `isGlobalVariable()`，来判断是否为全局变量。

新增 `LOAD_SYMBOL`，`LOAD`，`STORE`，分别用于加载符号所代表的地址、根据加载地址中的数据、存储数据到特定地址，在目标代码中对应 `la`、`lw`、`sw`。

最关键的部分就是要根据定义的全局变量来修改 `.data` 字段。每遇到一个 `Piece::GLOBAL`，先输出一行 `.data`，然后 `.global` 以及对应的全局变量名，`.word` 以及对应的初始值。

还要修改相应的数据流、词法解析。

## 思考题

---

## Step 9

```
int a = 1, b = 2;  
func(a = b, b = a);
```

## Step 10

```
auipc v0, a[31:12]  
addi v0, v0, a[11:0]
```

```
lui v0, a[31:12]  
addi v0, v0, a[11:0]
```

Non-PIC vs PIC