编译实验LAB8：函数

本次实验是除挑战实验之外的最后一个实验，整体上的实现不是很复杂，但和数组相关的调用才是这次的难点。我函数声明和调用昨晚几个小时就完成了，但是从今天中午到现在（晚上七点半）一直都在反复修改函数调用时传递的参数是数组的情况，这里的情况很多很复杂，我花了很久才逐渐整理好思路。

函数的声明

函数声明上没有太大难度，只需要修改和新建语法分析中的funcdef、funcparams等节点。在funcdef中分析并输出函数名、返回类型、函数形参，然后把参数添加到这个函数自己的符号表，并分析函数体和返回值。最后在当前符号表（全局变量符号表）中加入这个函数的信息。

这里再次解释我的符号表设计：函数和全局变量放在同一层（最外层）的表中，因为函数只允许在全局区定义，不能在函数中定义函数。其次因为上面的函数需要能够在下面的函数中访问到，而且函数间不能重名，于是放在全局符号表中是最方便的选择。

函数声明时的参数的处理

我们需要分别分析并输出每个参数的类型和寄存器（需要给每个参数提供一个寄存器），并加入到这个函数的符号表。额外的，如果参数是数组类型，就在遍历数组每一维的同时记录每一维度的定义长度，并且用字符串拼接生成出数组的类型，例如`[2 x [2 x i32]]`。特别的，我对每个参数中的数组的维度数进行了+1，加了一个空维度，因为函数参数里的数组比在当前block声明的数组少了一维，不补上的话后面维数判断会出错。

```
else { // 参数是数组
                    // 拿到arrayType和arrayDim
                    StringBuilder arrayType = new StringBuilder();
                    ArrayList<Integer>arrayDim = new ArrayList<>();
                    for(SysYParser.ExpContext e:
ctx.funcFParam(i).exp()) {
                        String len = visit(e); // 数组的各维长度必须是
编译时可求值的【非负】【常量表达式】。
```

```
                    if(len.startsWith("-")) {
                        System.exit(-1);
                    }
                    arrayDim.add(Integer.parseInt(len));
                    arrayType.append("[").append(len).append("
x ");
                    // 例如：[2 x [2 x i32]]
                }
                arrayType.append("i32");
                for(int j=0;j<arrayDim.size();j++) {
                    arrayType.append("]");
                }
                System.out.print(arrayType + "* " + param_reg);
                arrayDim.add(0, 0);
                addToAssignMapWhenBlock.add(new Variable(name,
param_reg, false, arrayDim, arrayType.toString(), true));
            }
```

另外关于参数的一点，函数的参数虽然分配了寄存器，但i32类型（整型）的参数是通过值传递的，分配的寄存器中直接保存着数值，而不是这个变量所在的地址。这就造成了这个参数只可读，不可修改，因为它就不是变量。我的解决方法是对所有整型参数，创建一个变量并把参数值赋给这个变量，用其地址作为这个参数的地址。

```
String ptr_reg = "%r" + regId++;
                System.out.println("    " + ptr_reg + " =
alloca i32");
                System.out.println("    store i32 " + v.reg +
", i32* " + ptr_reg);
                v.isParam = false;
                v.reg = ptr_reg;
```

函数的调用

首先考虑库函数的调用，我采用特判这6个库函数，就不需要把他们加入符号表了。对于一般的函数，首先从符号表中根据函数名找到对应的表项，然后判断参数列表的长度是否和函数定义时的参数长度相等，如果不等就报错退出。

然后分情况处理有返回值的函数和没有返回值的函数，分别生成不同的中间代码，其中需要分析并遍历函数调用时的参数，得到每个参数的类型和寄存器。参数类型我通过一个全局变量`funcCallingType`来实现跨函数的传递，避免了表达式遍历函数的多重情况下使用的冲突。

函数调用中数组作为参数

数组作为参数来调用函数时，数组需要降一维（ir的规则是这样设定的）才能传参，例如如果原来的数组类型是`[99999 x [99999 x i32]]`，那么需要先将类型转换到`[99999 x i32]`，然后才能作为参数传递而保持维度不变。

数组作为参数时，如果没有下标（例如`func(arr)`），代表传递完整的数组；如果有一个下标（例如`func(arr[0])`），代表传递这一维度固定了具有剩余维度的数组，也就是维数-1；如果有多个下标，以此类推，最多可以有和数组维数相同的下标，代表传递的是一个整型元素，而不是数组（需要单独处理）。

处理完下标和维度的变换后，输出`getelementptr`指令获取数组的地址，需要根据给定的下标数和维数共同决定指令的指针类型和取多少个维度的下标。

需要注意的是，对于本身就是当前函数参数的数组，它的维度已经比正常数组少了1，因此在`getelementptr`时的访问维度需要少一维`i32 0`，也就是说不需要和普通数组一样，降一维再传递。

```
if(val.arrayDim.size() != ctx.exp().size()) { // 如果数组最后还是数组
                    if(val.isParam) System.out.println(", i32 0");
                    else System.out.println(", i32 0, i32 0");
                    return elm_reg;
                }
                else { // 如果数组退化成了int
                    System.out.println(", i32 0");
                    String int_reg = "%r" + regId++;
                    System.out.println("    " + int_reg + " = load i32, i32* " + elm_reg);
                    return int_reg;
                }
```

其他值得注意的点

- if-else和while语句可能会出现：函数全部在语句中间return，而语句后面没有return的情况，这时候中间代码生成出来最后一个label后就是空的（直接跟着代表函数结束的`}`），执行时会报错没有返回语句。我采用了最简单的办法解决，每个函数最后强制加一个ret语句，如果是有返回值的函数就加一个`ret i32 0`，如果是void函数就加一个`ret void`，保证执行时函数末尾一定有一个返回语句，并永远不会被执行到。