编译实验LAB7：数组

这次实验是史上最最最最最难的一次！

我从周五早上开始做，做了一整天才写完了局部常量数组声明的部分，第二天又继续做，一直写到晚上十点才通过所有测试点，简直是折磨，这两天除了编译lab7之外什么事情都没干。之前的lab从来没有哪一次这么复杂，代码量这么大过。经过这次实验后我的程序结构已经变混乱而且我也无力维护了，包括但不限于使用了许多全局变量（例如一些flag），很多重复代码没有抽象出函数（例如局部常量数组和局部变量数组几乎是一样的代码被放在了两个不同的语法节点中），以及为了保证实现的方便加入了很多冗余代码。如果时间充裕的话这些地方是值得改进的。

局部数组声明

在局部数组的声明上，常量数组和变量数组并没有区别。下面简述流程：

1. 输出alloca指令，遍历数组各维长度的定义得到数组的类型（例如`[5 x [4 x i32]]`），并输出
2. 使用getelemenptr和memset指令给整个数组赋初值，相当于无论源程序有没有给初始值，都先全部赋为0，避免了访问未初始化区域的问题
3. 开始遍历initVal/constInitVal节点，在这个节点中使用类似深度优先搜索的算法，对每一个初始值按照维度的层次进行遍历。在访问到数值时同时也得到了它在每个维度的下标，此时再输出getelemenptr和store指令进行赋值。也就是说，每个元素单独赋值，有多少个初始化元素，执行多少次赋值

全局数组声明

在全局数组的声明上，常量数组和变量数组并没有区别。下面简述流程：

全局数组的声明在形式上更简洁（只需要一行指令就可以声明+初始化所有元素），但是实现的难度比局部数组更大。全局数组的声明形式如下：`@a = dso_local global [3 x i32] [i32 1, i32 2, i32 0]`

1. 先遍历数组各维长度的定义，得到数组的类型并输出
2. 存入符号表
3. 变量全局数组如果没有指定初值，就用zeroinitializer初始化

4. 遍历initVal/constInitVal，把所有需要初始化的元素加入 `globalArrayInitVals` ，通过计算出的元素在数组中的序号来定位
5. 定义并递归调用 `construct` 函数，生成高维数组的初始值表达式 `[[1 x i32] [i32 1], [1 x i32] [i32 3]]` 并输出

数组访问

首先访问语法结构，得到每一维度的下标并临时保存。根据给定的下标序号，调用getelemptr指令得到需要访问的元素的地址，然后用load指令加载即可。其中getelemptr后面参数的格式为， `i32 0, i32 第一维下标, i32 第二维下标, ...`

数组赋值

赋值时需要检查是否为常量数组，需要报错退出

然后和数组访问一样，得到getelemptr后用store指令保存即可。

其他bug处理

1. `list.size() != 0` 和 `list != null` 不等价，后者有些情况无法正常判断，在测试中会有bug
2. 计算getelemenptr中的每一维下标需要提前算好并临时保存，因为下标可能有+-*/等运算需要输出add/mul等指令才能得到结果，不能夹杂在指令getelemenptr的里面
3. Lab4遗留问题，条件语句的i32和i1类型不统一，导致and/or指令两个操作数类型不同，无法运行。解决方法是中间步骤统一计算icmp后用zext强制转换到i32，在and/or运算时先转成i1参与运算（ `and i1 xxx` / `or i1 xxx` ）因为不用i1的话结果有问题，brainfk测试点无法通过。最后在cond节点再转换到i1，返回给上层。