编译实验LAB5：作用域和全局变量

这次实验的内容是增加了对变量作用域的区分，以及全局变量的支持。

这次的改动基本都在语义分析上，顾词法和语法分析不再赘述。

作用域

作用域的实现必不可少的需要对符号表加以管理。我之前的符号表是只有一个map，对应实现的是单一作用域（所有符号都在一个表里），或者说没有考虑作用域。如果想实现多作用域，我使用了栈式结构，每进入新的一个内层，就创建一个新的符号表（map）压入栈中；每从一个内层退出到外层，就把栈顶的符号表弹出。这样的压栈出栈操作就对应了作用域的变化。

```
private Stack<Map<String,Variable>>assignStack = new Stack<>();
```

在变量声明时，只需访问并检查当前栈顶的符号表，也就是当前作用域的符号表。因为实验要求是变量名可以和外层变量重名（覆盖外层的），但当前层的变量名要求唯一性。

```
Map<String,Variable> assignMap = assignStack.peek();
```

在变量赋值和取值时，需要从符号表栈的顶层开始，一层一层遍历，直到找到这个变量为止，如果到最底层也没找到，就报错退出。这样的遍历顺序保证了内层同名变量可以覆盖掉外层的变量。

```
for(int i=assignStack.size()-1;i>=0;i--) {
                Map<String,Variable> assignMap =
assignStack.get(i);
                if(assignMap.containsKey(name)) {
                    val = assignMap.get(name);
                    break;
                }
            }
```

全局变量

本实验中全局变量和普通变量的区别如下：

1. 全局变量的作用域是整个程序，即任何地方都可以访问
2. 全局变量定义时不需要 `alloca` 和 `store` ，而是 `dso_local global i32 xxx`

对于第一个区别，我的实现方式是：把所有全局变量放在单独的最外层的一个作用域（map），相当于在所有函数和Block 的外层，绝对任何地方都可以访问到。

对于第二个区别，我定义了 `boolean globalFlag` 指示当前声明的变量是不是全局变量，如果是的话就输出全局变量声明时对应的指令语句。

其他几个重要的细节

最后是一些一开始没注意到，评测时才发现的要点，我后续主要时间都花在处理这些小细节上了：

1. Java中的Stack在用foreach遍历时的顺序是从栈底到栈顶，与我的需求相反，因此我放弃foreach而是手写for循环来遍历Stack
2. 全局变量声明时需要对表达式压缩，即需要把赋值表达式计算出一个数字，直接赋给全局变量，而不能是一步一步打印计算指令
3. 全局变量的赋值表达式中不能含有变量（只能含有数字或上面声明过的全局常量），有的话要报错退出