LogoCompiler的设计和原理

作者：陶天骅

前言

本文档并不是项目文档的一部分，我的两位（或者一位）队友负责文档的撰写，这部分工作依然由他们负责，但我觉得有一些内容值得一提，文档中却没有讲清楚，因此本文可以作为补充材料。

虽然LogoComplier是一门讲述面向过程编程的课程的大作业，但是在本项目中，面向对象编程（OOP）的技巧和思想一以贯之。事实证明，OOP的引入使LogoComplier变得更为强大、灵活、明了。而且OOP在本项目中并不是对功能函数的简单包装，而是使用了继承、多态，例如Op类是所有操作的基类，它派生出MoveOp、TurnOp、DefOp等，子类重写虚函数exec()，以表现不同的行为。

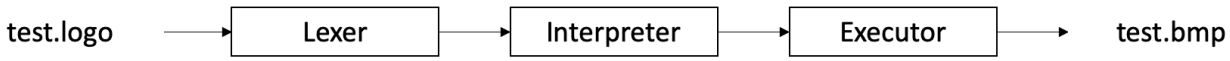
LogoComplier应该被视为一种logo语言的编译器，因此它的使用方法与大多数编译器类似：

```
./logoCompiler testcase1.logo
```

正常情况下，会在输入文件相同文件夹内生成一个testcase1.bmp文件。

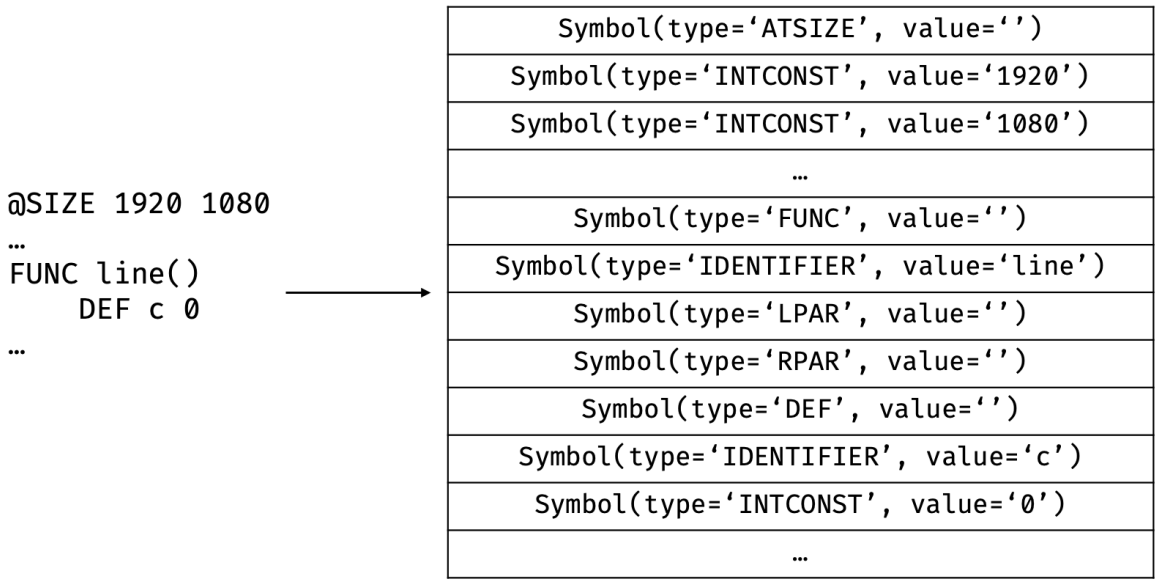
设计思路

LogoComplier的主要构成包括三部分：词法分析器（Lexer），解释器（Interpreter），执行器（Executor）。



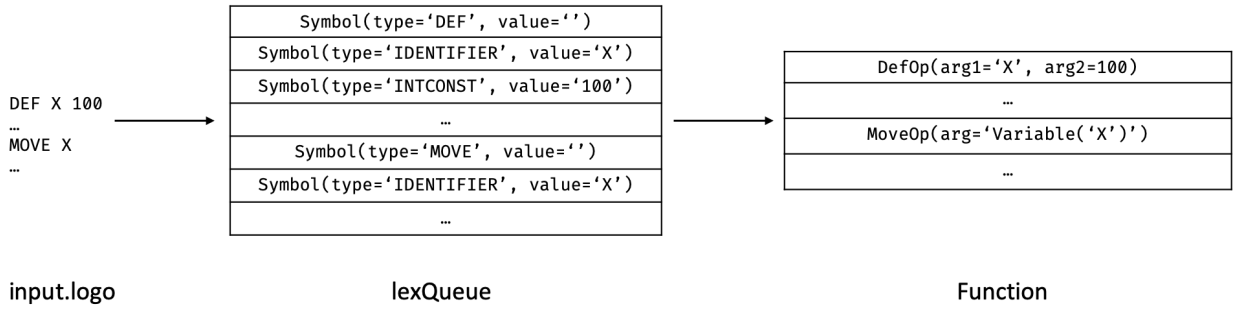
Lexer

Lexer的功能是词法分析，是使用我们平常构建编译器时用的lex和yacc中的lex实现的。我发现LogoComplier的语法实在太过简单，以至于不需要用yacc。Lexer把输入文件中的内容分为一个一个符号（Symbol），压入队列lexQueue，并将其传给Interpreter，开始第二阶段的分析。



Interpreter

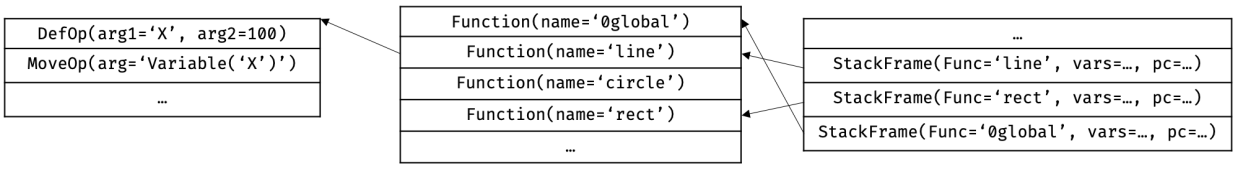
Interpreter会做一些简单的语法检查，然后通过分析lexQueue，调用Executor的接口，将Symbol序列转变为可执行的Op序列。之后调用Executor的run()方法，开始第三阶段的执行。



Executor

Executor负责实际的运算和绘画工作。通过Interpreter在第二阶段的处理，输入的符号序列已经变为了一连串的Op序列，Executor调用每个Op的exec()方法。一些的Op的集合组成一个Function，Executor内维护一个Function的列表和一个调用堆栈callStack。callStack由若干个栈帧StackFrame组成，一个StackFrame中存有当前执行的Function指针、返回地址、局部变量。

Function就相当于我们平时运行程序时的代码段，callStack就是运行堆栈。



`Function(name='line')` `allFunctions` `callStack`

一些feature和实现方式

函数调用

LogoCompiler运行时的最外层函数是"0global"，之所以起这个名字，是因为它不是用户可使用的合法标识符名，"0global"就相当于C语言中的main函数。

由于StackFrame和callStack的存在，实际上函数可以支持递归，但是由于语法中没有if支持，无法设置递归基，因此递归不可用。

变量作用域

作用域是以Function区分的。Function内的变量会遮蔽外部的变量。

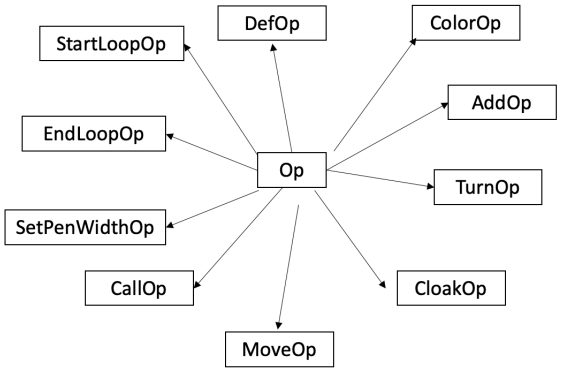
StackFrame中维护了一个Variable的数组，是在当前作用域中定义的局部变量。当要引用一个变量的时候，会优先在当前作用域中寻找，若找不到，会在外层作用域中继续寻找，若都找不到，则会报错。

VariableWrapper类

例如MOVE操作可以接受一个常数也可以接受一个变量，如MOVE 100和MOVE X。为了统一两者，我引入了VariableWrapper类，它可以将Variable和INTCONST字面量统一起来，并提供getValue()方法，在执行时再计算实际的值。这也算是一种lazy policy，它还可以配合变量作用域，优先查找当前作用域中的同名变量。

Op类

几乎每个Op类都是Executor的友元，可以直接修改其内部变量。



每个Op子类都重写了exec()函数。

注释

可以支持//开头的单行注释。但注释在最后一行的话，注意要以换行符结尾。

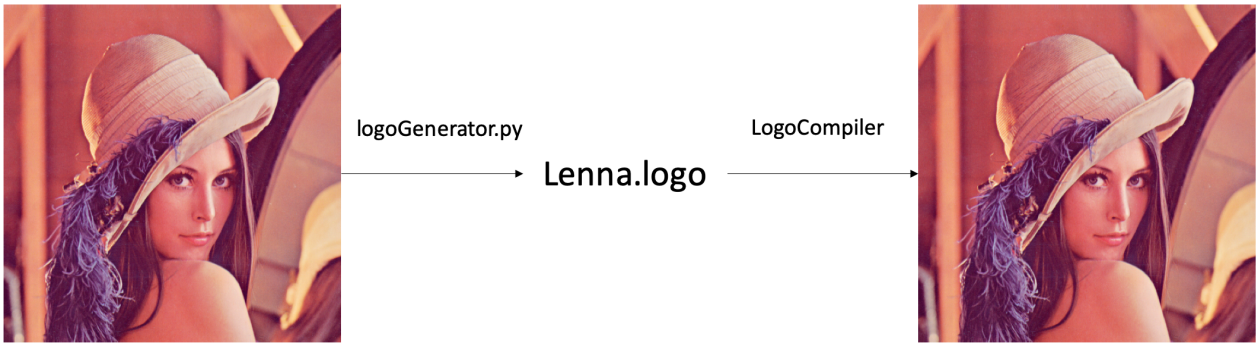
错误提示

我尽量使LogoCompiler支持可读的错误提示。一些例子如下:

- `Cannot open the file`
- `Error at line 1: \`
- `Error: The file is empty`
- `Error at line 6: Unexpected symbol: hackhacks`
- `Runtime Error at line 10: END LOOP not found`
- `Error: End of file in function definition, did you miss "END FUNC" for line()?`
- `Runtime Error at line 11: arguments do not match`
- `Runtime warning: Color value out of range, value larger than 255 will be set to 255, value smaller than 0 will be set 0`
- `Error at line 19: Unexpected symbol: END`

逆向生成logo文件

在`testcases/extended/logoGen`文件夹中，有一个辅助工具`logoGenerator.py`，它可以把任意一张图片，转变为合法的logo文件。把该logo文件作为输入，LogoCompiler可以生成完全相同的图片。logo文件可能很大，但是LogoCompiler可以高效地执行它。



Lenna.png　　Lenna.bmp

具体用法:

```
python3 logoGenerator.py Lenna.png > Lenna.logo
./logoCompiler Lenna.logo
```

输出展示