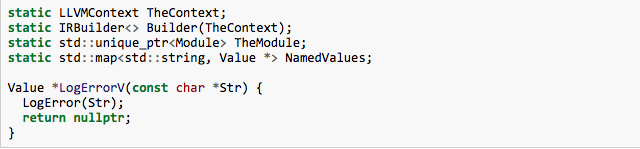
Github地址：

<https://github.com/Interpretion/Interpreter>

LLVM IR代码的生成，需要做的就是将得到的抽象语法树，转换成LLVM IR，而这需要一个转换器，也就是Code generation,用于实现代码生成。

在ExprAST中定义虚函数Codegen，这样之后每个类只需要重定义就可以用了。其次，还需要一个“error”方法，类似于报错函数，用于报告代码生成过程中发生的错误。



TheModule是LLVM中用于存放代码段中所有函数和全局变量的结构。从某种意义上讲，可以把它当作LLVM IR代码的顶层容器。Builder是用于简化LLVM指令生成的辅助对象。IRBuilder类模板的实例可用于跟踪当前插入指令的位置，同时还带有用于生成新指令的方法。NamedValues映射表用于记录定义于当前作用域内的变量及与之相对应的LLVM表示（换言之，也就是代码的符号表）。

在生成代码之前必须先设置好Builder对象，指明写入代码的位置。

表达式节点生成代码：新建并返回了一个ConstantFP对象。

二元表达式生成代码：和AST一样。这里也需要递归，也是先处理表达式左边，然后处理表达式右边，最后计算整个二元表达式的值；由于运算符简单，只用switch语句就行。

之后根据VSL语言来进行表达式的添加删改。

函数的代码生成：

函数原型代码生成：“函数原型”描述的是函数的对外接口（而不是某表达 式计算出的值），因此函数返回值类型是“Function\*”而不是“Value\*”。首先为给定的函数原型创建对应的FunctionType对象，包括参数类型；随后，FunctionType::get方法创建一个函数类型；最后创建与该函数原型相对应的函数。其中包含了类型、链接方式和函数名等信息。

函数定义的代码生成：生成函数原型的代码并进行校验。与此同时，需要清空NamedValues映射表，确保其中不会残留之前代码生成过程中的产生的内容。