readme

代码结构:

一共包含词法分析、语法分析、语义分析、中间代码生成、JIT 初始配置几部分。在我们的实现中，拥有一个主控循环每次读取用户输入的代码，依次进行词法、语法、语义、中间代码生成、装载 JIT 并运行操作。

1. token 列表：

```
enum Token {
  tok_eof = -1, //结尾
  // commands
  tok_def = -2, //定义
  // let definition
  tok_let = -3,
  // primary
  tok_identifier = -4, //标识符
  tok_real = -5,        // double
  tok_int = -6,         // int
  tok_string = -7,      // string
  tok_char = -8,        // char
  // control,控制流
  tok_if = -9,
  tok_then = -10,
  tok_else = -11,
  tok_while = -12,
  tok_do = -13,
  tok_in = -16,
  tok_end = -17,
  // operators
  tok_binary = -14,
  tok_unary = -15,
  // 变量声明符
```

```
  tok_val = -18,
  //分号
  tok_semi = -19,
  // print
  tok_print = -20,
  tok_change = -21
};
```

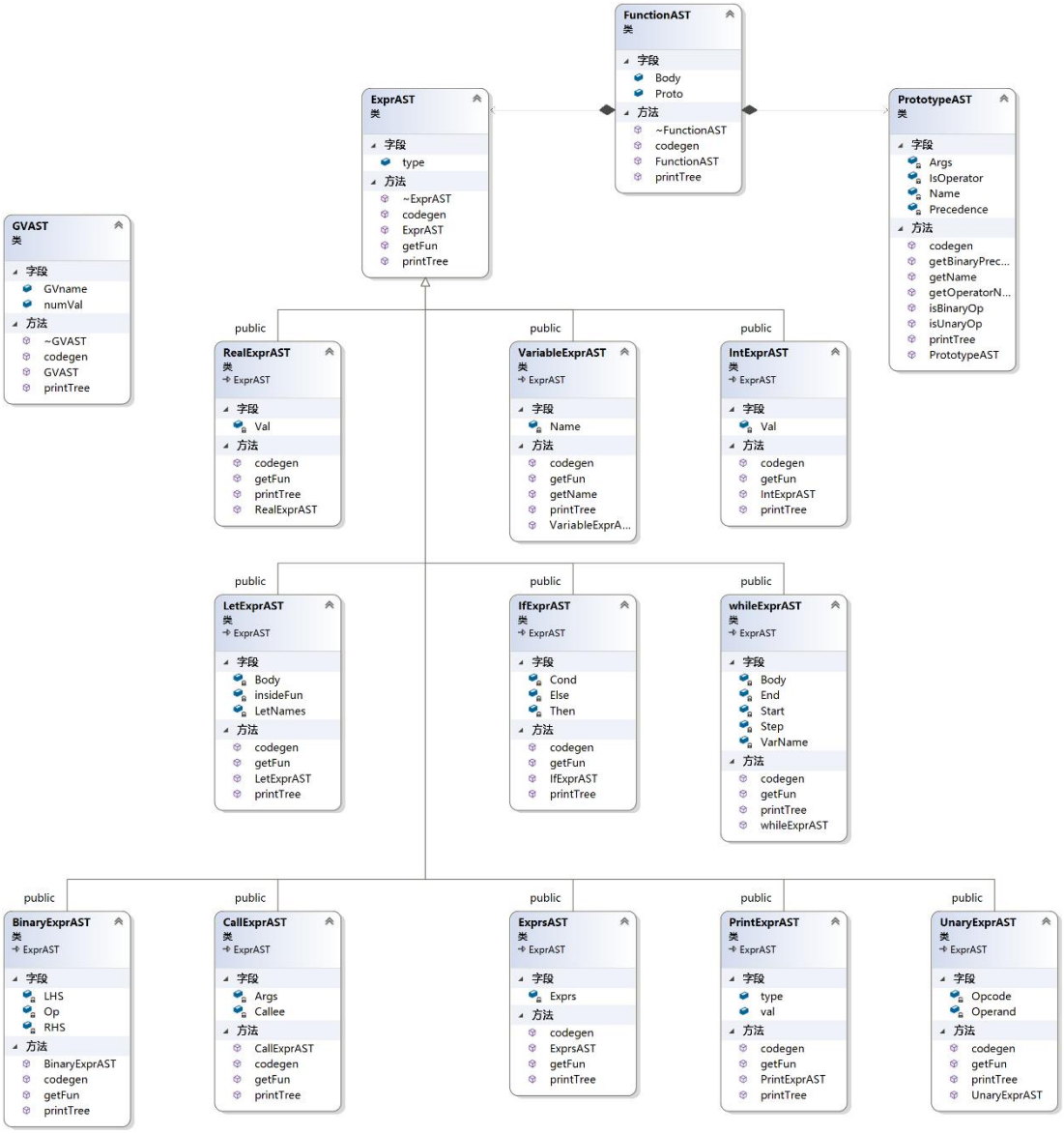
2. 词法分析部分

主要包含以下函数：

```
static int gettok();
static int getNextToken();
```

3. 语法分析部分

主要包含各个 AST 节点类的 Parse()函数，这些函数的作用是根据 SML 的语法规则解析代码，生成相应的 AST 节点。



4. 语义分析部分

这一部分的代码是和语法分析杂糅在一块的，包括函数调用时对参数个数的匹配等等。

5. 中间代码生成

每一个 AST 类都重写基类的 codegen()函数，递归调用该函数得到 IR 代码。

6. JIT 配置

修改的地方:

1. token 列表针对 SML 语言增加了 token：例如 tok_val、tok_let、tok_in、tok_end、tok_if、tok_then、tok_else、tok_while、tok_do、tok_semi、tok_print、tok_change 等。

2. 根据 token 的增加，gettok()方法也进行了相应的增加：例如对'let'的识别判断 if (IdentifierStr == "let") return tok_let;添加了对小数点的判断，即只允许一个小数点（万花筒官方文档中没有解决 1.3.1 数值的问题）
3. 添加了基本运算符的数量，增加了%、/等运算，同时也设置了优先级。
4. 根据新增的文法，增加了表达式的类型，如 ParseIfExpr(处理 if then else 条件语句)、ParseWhileExpr(处理 while do()循环语句)、ParseLetExpr(处理 let in end 局部变量语句)、ParsePrint(处理 print 函数)等表达式类，相应的也会要求我们额外的写相应的 codegen()方法等与 IR 有关的函数。
5. 针对 SML 语言文法的要求修改了原 LLVM 代码 AST 类中的绝大部分的 parse 函数（根据文法对于下一个标识符的识别来判断是否出现语法错误）
6. 增加了函数重载和函数嵌套定义的功能，以及各种控制流的功能。
7. 增加了全局变量功能。并扩充了相对应的词法语法中间代码生成代码。
8. 增加了一些新的全局结构。使用 double condNum(判断是否参数为常数)、bool LineFun(判断是否是包含|的函数)、std::unique_ptr<PrototypeAST>P(用来存放并传递函数的名字和参数)等。
9. 其他为功能扩充而改动较大的函数有例如 ParsePrototype()、ParseExpression()等。同时为了实现参数匹配的功能也增加了 ParsePattern()等函数，相应的 IR 部分的例如 codegen()方法代码也会做较大的修改。
10. 增加了 AST 树的命令行绘制功能。为每一个 AST 节点增加 PrintTree()函数。