**编译原理大作业说明文档**

小组成员：熊伟伦 5120379076  
 郭彦昌 5111109017

王浩 5120379081

1. **简介**

此文档介绍了我们小组在编译原理大作业完成过程的一些情况，和实现的编译器的一些说明介绍。

此文档范围仅适用于我们完成的编译原理大作业——PsychoCompiler。

项目使用的版本控制软件是GitHub，地址是：<https://github.com/Azard/PsychoCompiler>。

在2014年12月30日前该项目权限设置为private project，大作业提交截止后设置为public project。

项目最终能够使用定义的MyLang语言，一键生成LLVM中间代码，无需修改。使用LLVM即可运行生成的LLVM中间代码。

1. **开发环境说明**

该项目主要使用Java进行开发，使用JavaCC进行词法语法分析，使用LLVM完成对生成的LLVM IR的最后执行。

* Java : JDK1.8 32bit
* IDE : Intellij IDEA 14
* LIB : JavaCC-5.0
* UI : AWT

1. **项目文件结构**

\说明文档.pdf ——即该文档

\demo ——demo程序的目录

\demo\quick\_sort.ml ——快速排序的MyLang源代码

\demo\quick\_sort.ll ——编译得到的快速排序LLVM中间代码

\demo\nQueens.ml ——N皇后的MyLang源代码

\demo\nQueens.ll ——编译得到的N皇后LLVM中间代码

\PsychoCompiler ——项目目录

\PsychoCompiler\src\main\java\jjt\MyLang.jjt ——传递给JavaCC使用的MyLang词法语法规则文件，该目录下其余文件为JavaCC所生成。

\PsychoCompiler\src\main\java\PsychoCompiler ——该文件夹包含了程序运行的主要源代码，包含对生成树的分析，符号表建立，静态分析，错误输出，LLVM中间代码翻译，UI等。该文件夹下所有文件都是我们小组手写的。

\PsychoCompiler\src\main\java\resources\test ——该文件夹包含了我们项目最终使用的测试代码和一些demo样例。

\PsychoCompiler\src\main\java\resources\error\_test ——包含了一系列错误测试的文件夹

\PsychoCompiler\src\main\java\resources\MyLang\_BNF.txt ——该文件为MyLang的词法语法定义，与MyLang.jjt文件的定义保持一致。

\PsychoCompiler\README.md ——项目完成的进度记录

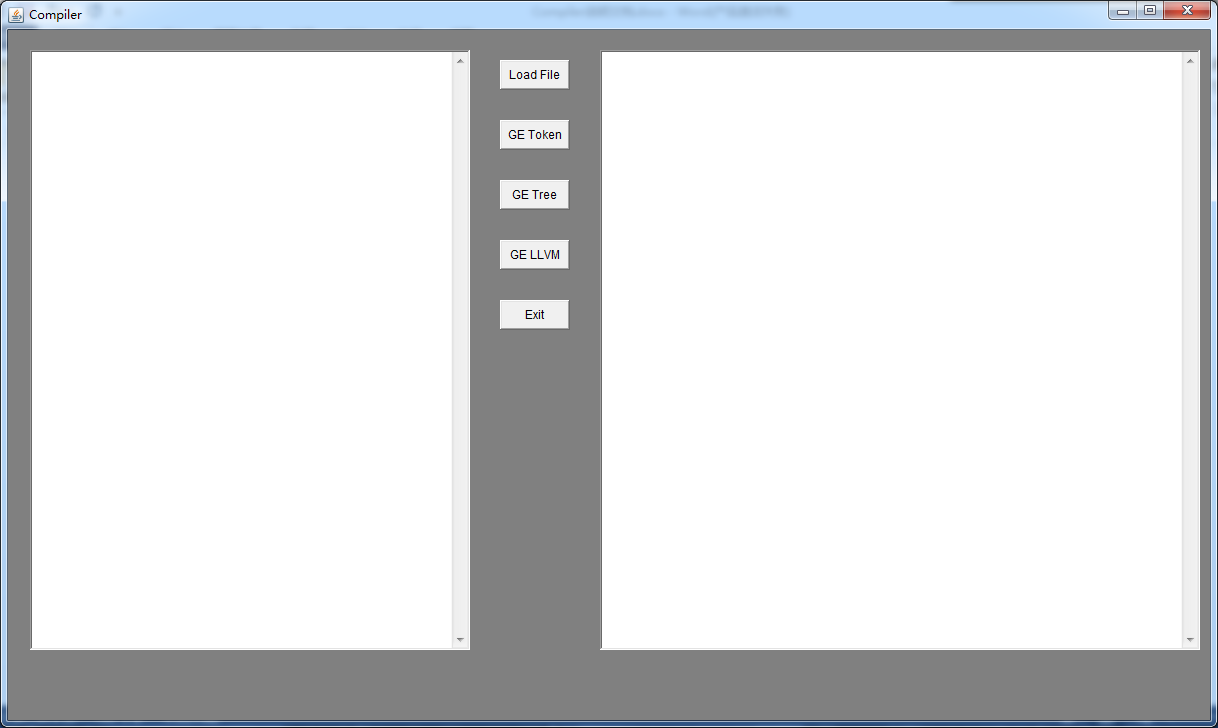
1. **功能介绍**

* 词法分析：根据MyLang的词法定义对MyLang源代码进行词法分解，不符合词法规定的部分会定位报错。
* 语法分析：根据MyLang的语法定义对MyLang源代码进行语法分析，并且生成语法树，不符合语法规定的部分会定位报错。
* 语义检查：能够在生成LLVM中间代码前，进行尽可能多的语义检查，包括类型匹配等。对在静态分析时发现的语义错误会定位报错，并终止LLVM中间代码的生成。
* LLVM中间代码生成：根据MyLang源代码，生成对应的LLVM中间代码，可使用LLVM直接运行，无需进行人工修改。

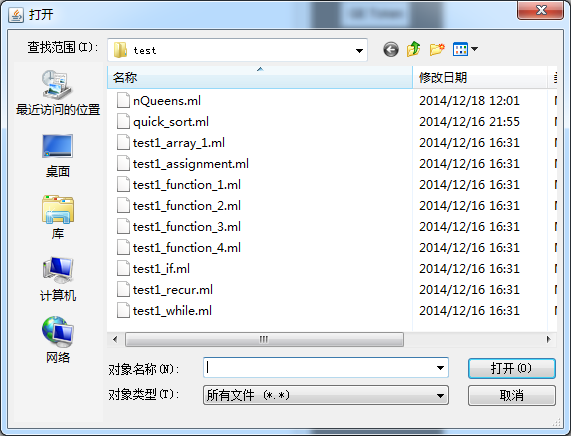
1. **使用方法**

程序的UI简洁明了，比较容易理解如何使用。

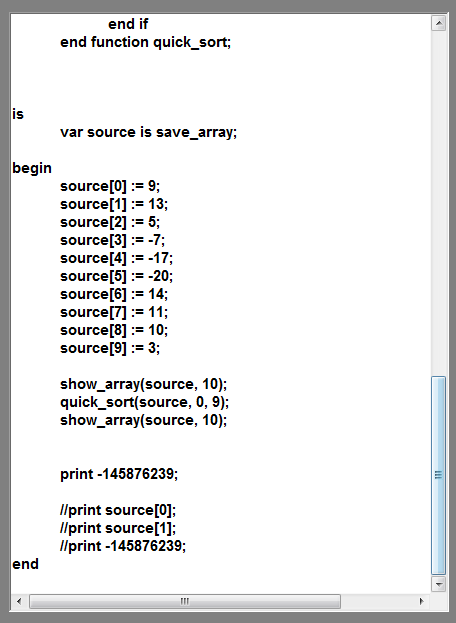
首先运行\PsychoCompiler\src\main\java\PsychoCompiler\GUI\_CompilerMain.java，出现如下程序窗口：



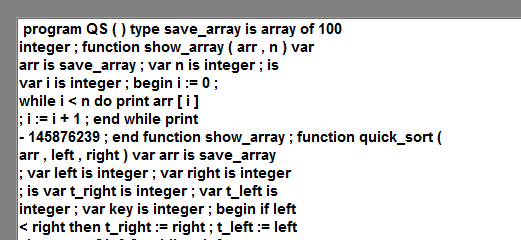
点击第一个按钮，也就是Load File按钮，选择MyLang源代码文件：



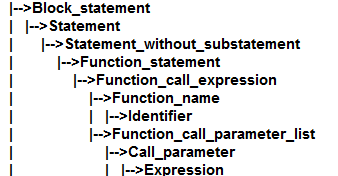
选择需要使用的源代码文件，比如quick\_sort.ml，会再程序的左边窗口显示文件的内容：



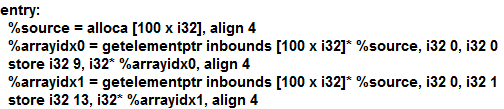
这里使用的是AWT的控件，相当于一个简易的文本编辑器，可以直接在左边窗口进行代码的修改。点击第二个按钮GE Token，在右边窗口会显示左边窗口源代码对应的词法分解输出，如果错误则会显示报错信息。



点击第三个按钮GE Tree会在右边窗口显示语法分析得到的语法树，如果有错会显示报错信息：



点击第四个按钮GE LLVM在右边窗口显示生成的LLVM中间代码，同时会在terminal中输出所有的变量名和变量类型，如果语义检查发现问题会显示报错信息：



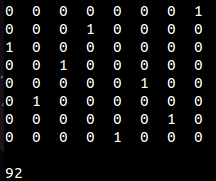
然后赋值生成的LLVM中间代码到Linux虚拟机中，使用LLVM的lli指令执行生成的LLVM中间代码的文本文件，就会在执行程序并在terminal中输出程序打印的内容：

如下图是使用demo中的MyLang编写的快速排序和八皇后得到的LLVM中间代码，使用LLVM运行的结果：

快速排序



八皇后



1. **测试用例**

如同文件结构中描述的，测试用例全在\PsychoCompiler\src\main\java\resources\test文件夹中，其中包括了快速排序和八皇后的demo和。

同时，我们也把这两个demo放在了\demo文件夹中。

\PsychoCompiler\src\main\java\resources\MyLang\_code中包含了词法和语法的错误测试用例。

\PsychoCompiler\src\main\java\resources\error\_test中包含了语义错误测试用例，也就是静态分析的时候的错误提示。

1. **开发流程**

简要说明下开发流程。

第一阶段是书写BNF定义。

第二阶段根据BNF定义编写相应的JavaCC使用格式的jjt文件，在这个过程中我们使用了LL(3)的语法分析方式构建语法树，并且对第一阶段的BNF定义做了一定的修改，最终能够对符合词法语法规则的MyLang代码构建语法树。

第三阶段根据第二阶段构建的语法树，首先分析程序所有的声明，包括函数声明，类型声明，变量声明，建立符号表，变量表，类型表并在读取声明的过程中做全部能做到的静态检查。  
 然后翻译begin和end之间的可执行代码变为LLVM中间代码，包括主函数之间的代码和函数的运行代码。当然在翻译过程中也需要根据前面建立的多个表的信息进行静态分析。稍微需要说明的是函数传参我们规定传入普通的类型例如integer和boolean是传值，而传定义的数组类型是传引用。