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

一． 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。

二． 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

三． 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

四． 功能介绍

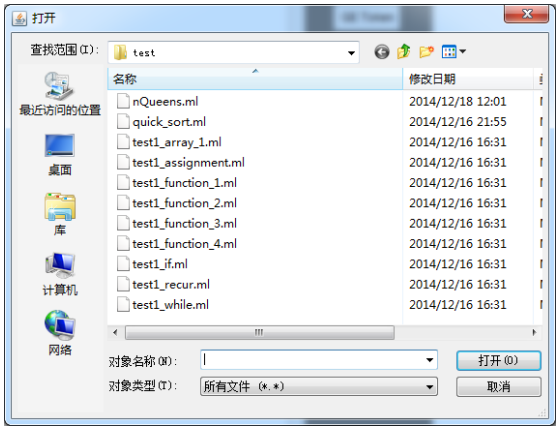
- 词法分析：根据 MyLang 的词法定义对 MyLang 源代码进行词法分解，不符合词法规定的部分会定位报错。
- 语法分析：根据 MyLang 的语法定义对 MyLang 源代码进行语法分析，并且生成语法树，不符合语法规定的部分会定位报错。
- 语义检查：能够在生成 LLVM 中间代码前，进行尽可能多的语义检查，包括类型匹配等。对在静态分析时发现的语义错误会定位报错，并终止 LLVM 中间代码的生成。
- LLVM 中间代码生成：根据 MyLang 源代码，生成对应的 LLVM 中间代码，可使用 LLVM 直接运行，无需进行人工修改。

五． 使用方法

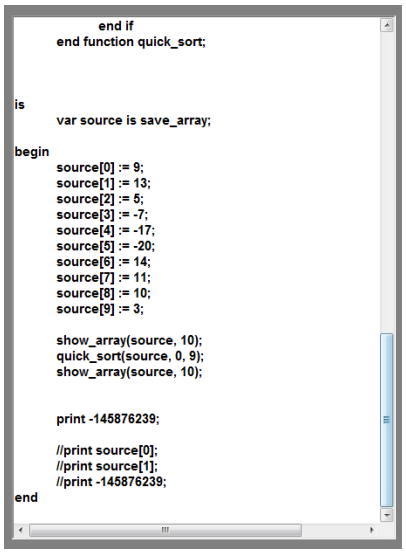
程序的 UI 简洁明了，比较容易理解如何使用。

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：

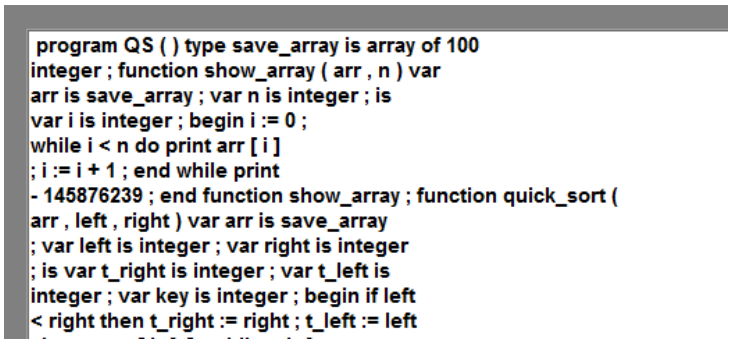


选择需要使用的源代码文件，比如 quick_sort.ml，会再程序的左边窗口显示文件的内容：



```
            end if
        end function quick_sort;



is
        var source is save_array;

begin
        source[0] := 9;
        source[1] := 13;
        source[2] := 5;
        source[3] := -7;
        source[4] := -17;
        source[5] := -20;
        source[6] := 14;
        source[7] := 11;
        source[8] := 10;
        source[9] := 3;

        show_array(source, 10);
        quick_sort(source, 0, 9);
        show_array(source, 10);


        print -145876239;

        //print source[0];
        //print source[1];
        //print -145876239;
end
```

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。



```
program QS ( ) type save_array is array of 100
integer ; function show_array ( arr , n ) var
arr is save_array ; var n is integer ; is
var i is integer ; begin i := 0 ;
while i < n do print arr [ i ]
; i := i + 1 ; end while print
- 145876239 ; end function show_array ; function quick_sort (
arr , left , right ) var arr is save_array
; var left is integer ; var right is integer
; is var t_right is integer ; var t_left is
integer ; var key is integer ; begin if left
< right then t_right := right ; t_left := left
```

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：

```
|-->Block_statement
|  |-->Statement
|     |-->Statement_without_substatement
|        |-->Function_statement
|           |-->Function_call_expression
|              |-->Function_name
|              |  |-->Identifier
|              |-->Function_call_parameter_list
|                 |-->Call_parameter
|                 |  |-->Expression
```

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：

```
entry:
  %source = alloca [100 x i32], align 4
  %arrayidx0 = getelementptr inbounds [100 x i32]* %source, i32 0, i32 0
  store i32 9, i32* %arrayidx0, align 4
  %arrayidx1 = getelementptr inbounds [100 x i32]* %source, i32 0, i32 1
  store i32 13, i32* %arrayidx1, align 4
```

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

```
azard@ubuntu:~/mylang$ lli quick_sort.ll
9  13  5  -7  -17  -20  14  11  10  3
-20  -17  -7  3  5  9  10  11  13  14
```

八皇后

```
0  0  0  0  0  0  0  1
0  0  0  1  0  0  0  0
1  0  0  0  0  0  0  0
0  0  1  0  0  0  0  0
0  0  0  0  0  1  0  0
0  1  0  0  0  0  0  0
0  0  0  0  0  0  1  0
0  0  0  0  1  0  0  0

92
```

六． 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。

七． 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。