A. Setting Up and Running *j--*

A.1 Introduction

This appendix describes where to obtain the *j--* compiler from, what is in the distribution, how to set it up for command-line execution, how to setup, run, and debug the compiler in Eclipse[1].

A.2 Obtaining j--

The zip file **`j--.zip`** containing the *j--* distribution can be downloaded from the following website:

`http://www.cs.umb.edu/j--`

A.3 What is in the Distribution?

The *j--* distribution includes the following files and folders:

`j--/`	*j--* root directory[2]
`src/`	
`jminusminus/`	
`*.java`	The compiler source files; J*.java files define classes representing AST nodes, CL*.java files supply backend code that is used by j-- to create JVM bytecode, and N*.java files implement the register allocation for SPIM.
`j--.jj`	JavaCC input file for generating a scanner and parser.
`package.html`	Describes the **`jminusminus`** package for *javadoc.*
`spim/`	
`*.s, *.java`	SPIM[3] runtime files and their Java wrappers
`package.html`	Describes the **`spim`** package for *javadoc.*
`overview.html`	Describes the *j--* project for *javadoc.*

[1] An open source IDE. http://www.eclipse.org.

[2] The directory which contains the *j--* root directory is referred to as **`$j`**.

[3] A self-contained simulator that runs MIPS32 programs. http://spimsimulator.sourceforge.net/.

`lib/`	Contains jar files for JavaCC[4], Junit[5], and Java2HTML[6]. Also contains the jar file generated for the compiler.
`bin/`	Contains UNIX and Windows scripts for running the compiler and the Java class file emitter.
`tests/`	
`clemitter/`	Contains Java programs that programmatically generate class files using the `CLEmitter` interface.
`pass/`	Contains *j--* test programs.
`fail/`	Contains erroneous *j--* test programs. These are negative tests to make sure that the compiler handles errors gracefully. None of these tests should compile successfully, i.e., the compiler shouldn't produce class files for any of these tests.
`junit/`	Contains JUnit test cases for compiling and running *j--* test programs.
`spim/`	Contains *j--* test programs that compile to SPIM.
`lexicalgrammar`	Lexical grammar for *j--*.
`grammar`	Syntactic grammar for *j--*.
`build.xml`	Ant[7] file for building and testing the compiler.

`.externalToolBuilders/`
`.classpath`
`.project`
	Contains project settings for Eclipse.

A.3.1 Scripts[8]

We provide three scripts. The first script, `$j/j--/bin/j--`, is a wrapper for `jminusminus.Main`. This is the *j--* compiler. It has the following command-line syntax:

```
Usage: j-- <options> <source file>
where possible options include:
  -t       Only tokenize input and print tokens to STDOUT
```

---

4 A lexer and parser generator for Java. http://javacc.dev.java.net/.

5 A regression testing framework. http://www.junit.org.

6 Converts Java source code into a colorized and browsable HTML representation. http://www.java2html.com.

7 A Java-based build tool. http://ant.apache.org.

8 For the scripts to run, *j--* must be compiled and `j--.jar` file must exist in `$j/j--/lib`.

```
  -p       Only parse input and print AST to STDOUT
  -pa      Only parse and pre-analyze input and print
           AST to STDOUT
  -a       Only parse, pre-analyze, and analyze
           input and print AST to STDOUT
  -s <naive|linear|graph> Generate SPIM code.
  -d <dir> Specify where to place generated class files;
           default = .
```

The second script, **`$j/j--/bin/javaccj--`**, is a wrapper for **`jminusminus.JavaCCMain`**. This is also the *j--* compiler, but uses the front end generated by JavaCC. Its command-line syntax is similar to the script discussed above.

The third script, **`$j/j--/bin/clemitter`**, is separate from the compiler, and is a wrapper for **`jminusminus.CLEmitter`**. This is the Java class file emitter. It has the following command-line syntax:

```
Usage: clemitter <file>
```

where **`file`** is a Java program that uses the **`CLEmitter`** interface to programmatically generate a JVM class file. The output is a class file for the program, and the class files the program was programmed to generate.

For example, running the following command:

```
$j/j--/bin/clemitter $j/j--/tests/clemitter/GenHelloWorld.java
```

produces **`GenHelloWorld.class`** and **`HelloWorld.class`** files. **`HelloWorld.class`** can then be run as,

```
java HelloWorld
```

producing as output,

```
Hello, World!
```

`$j/j--/bin` may be added to the environment variable **`PATH`** for convenience. This will allow invoking the scripts without specifying their fully qualified path, as follows:

```
j-- $j/j--/tests/pass/HelloWorld.java
```

A.3.2 Ant Targets

The Ant file **`$j/j--/build.xml`** advertises the following targets:

package	Makes a distributable package for the compiler which includes the source files, binaries, documentation, and

	JUnit test framework.
runCompilerTests	This is the default target. It firstly, compiles the *j--* test programs under **`$j/j--/tests/pass`** and **`$j/j--/tests/fail`** using the *j--* compiler. Secondly, it runs the *j--* pass tests.
runCompilerTestsJavaCC	Same as the above target, but using JavaCC scanner and parser.
testScanner	Tokenizes the *j--* test programs in **`$j/j--/tests/pass`**.
testJavaCCScanner	Same as the above target, but using JavaCC scanner.
testParser	Parses the *j--* test programs in **`$j/j--/tests/pass`**.
testJavaCCParser	Same as the above target, but using JavaCC scanner and parser.
testPreAnalysis	Pre-analyzes the *j--* test programs in **`$j/j--/tests/pass`**.
testAnalysis	Analyzes the *j--* test programs in **`$j/j--/tests/pass`**.
help	List main targets.

A.4 Setting Up j-- for Command-line Execution

We assume the following:

- J2SE[9] 6 or later (the latest version is preferred) has been installed and the environment variable **`PATH`** has been configured to include the path to the Java binaries. For example, on Windows, **`PATH`** might include **`C:\Program Files\Java\jdk1.6.0\bin`**.
- Ant 1.8.2 or later has been installed, the environment variable **`ANT_HOME`** has been set to point to the folder where it's installed, and environment variable **`PATH`** is configured to include **`$ANT_HOME/bin`**.

Now, the Ant targets listed in A.3.2 can be run as,

```
ant <target>
```

For example, the following command runs the target named **`testScanner`**:

```
ant testScanner
```

If no target is specified, the default (**`runCompilerTests`**) target is executed.

[9] Java Development Kit (JDK). http://www.oracle.com/technetwork/java/index.html.

A.5 Setting Up j-- in Eclipse

In order to be able to setup *j--* in Eclipse, we assume the following:

- J2SE 6 or later (the latest version is preferred) has been installed and the environment variable **`JAVA_HOME`** has been set to point to the installation folder. For example, on Windows, **`JAVA_HOME`** might be set to **`C:\Program Files\Java\jdk1.6.0`**.
- Eclipse 3.7 or later has been installed.

An Eclipse project for *j--* can be setup as follows:

1. Unzip the *j--* distribution into a temporary folder, say **`/tmp`**.
2. In Eclipse, click the *File→New→Project…* menu[10] to bring up the *New Project* dialog box. Select *Java Project* and click *Next*. In the *New Java Project* dialog box, type "j--" for *Project Name*, make sure JRE (Java Runtime Environment) used is 1.6 or later, and click *Finish*. This creates an empty Java project called "j--" in the current workspace.
3. In the *Project Explorer* pane on the left, select the "j--" project. Click *File→Import…* menu to bring up the *Import* dialog box. Under *General*, select *File System* as the *Import Source*, and click *Next*. Choose "/tmp/j--" as the *From directory*, select "j--" below, and click *Finish*. Answer *Yes to All* to the question on the *Question* pop-up menu. This imports the *j--* files into the "j--" project. Once the import is complete, Eclipse will automatically build the project. The automatic build feature can be turned off, if necessary, by clicking the *Project→Build Automatically* menu.

To build the *j--* project manually, select the project in the *Project Explorer* window and click the *Project→Build Project* menu. This runs the default (**`runCompilerTests`**) target in the Ant file **`$j/j--/build.xml`**. To run a different target, edit the following line in the Ant file:

```
<project default="runCompilerTests">
```

and change the value of the **`default`** attribute to the desired target.

A.5.1 Running/Debugging the Compiler

In order to run and debug the compiler within Eclipse, a *Launch Configuration* has to be

[10] Note that Eclipse menus have slightly different names under different operating systems

created, which can be done as follows:

1. Click the *Run→Run…* menu to bring up the *Run* dialog box. Select *Java Application*, and click the *New Launch Configuration* button.
2. Give a suitable name for the configuration. Select "j--" for *Project* and type "jminusminus.Main" for *Main class*.
3. In the *(x) = Arguments* tab, type appropriate values for *Program arguments*; these are the same as the arguments for the **`$j/j--/bin/j--`** script described in A.3.1. For example, type "-t tests/pass/HelloWorld.java" to tokenize the *j--* test program **`HelloWorld.java`** using the hand-written scanner.
4. Click *Apply* and *Close*.

You can have as many configurations as you like. To run or debug a particular configuration, click *Run→Run…* or *Run→Debug…* menu, select the configuration, and click *Run* or *Debug*. The output (**`STDOUT`** and **`STDERR`**) messages from the compiler are redirected to the *Console* pane.

A.6 Testing Extensions to j--

We test extensions to the *j--* language that compile to the JVM target using a JUnit test framework. The framework supports writing *pass* and *fail* tests in order to test the compiler; it also supports writing JUnit test cases to run the *pass* tests.

A *pass* test is a *j--* test program that is syntactically and semantically correct. When such a program is compiled using *j--*, the compiler should report no error(s) and should generate class file(s) for the program. A *fail* test is a *j--* test program that has syntactic and/or semantic error(s). When such a program is compiled using *j--*, the compiler should report the error(s) and exit gracefully. It should not produce class files for such programs.

The Ant targets, **`runCompilerTests`** and **`runCompilerTestsUsingJavaCC`**, attempt at compiling the *pass* and *fail* tests using *j--*, and running the test suite for the *pass* tests. Any *pass* test that compiles with errors or produces incorrect results would result in failed JUnit assertions. Any *fail* test that compiles successfully would also result in a failed JUnit assertion.

Chapter 1 describes how to add a simple extension (the division operator) to *j--* and how to test the extension using the JUnit test framework.

Appendix E describes the SPIM target and how to compile *j--* programs for that target.

Further Readings

Java Development User Guide in (Eclipse Documentation) for more on how to run, debug and test your programs using Eclipse.