CS 6410: Compilers

Fall 2023

Part 1 – Scanner
Acknowledgment: Project assignment modified from an assignment developed by Hal Perkins, faculty member of the University of Washington, Allen School of Computer Science and Engineering

Posted on Wednesday, October 4, 2023
Instructor: Tamara Bonaci
Khoury College of Computer Science
Northeastern University – Seattle

Please submit your project by 11:59pm on Saturday, October 22, 2023. You should submit your project by pushing it to your Khoury GitHub repository, and providing a suitable tag. See the end of this writeup for details.

1 Assignment Overview

The purpose of this assignment is to construct a scanner for our MiniJava programming. You should use the **JFlex** scanner generator, which is a Java version of the famous **Lex** family of programs. The ideas behind **JFlex**, and the input language it supports, are taken directly from **Lex** and **Flex**, which are described in most compiler books, and have extensive documentation online.

These programs work with the **CUP** parser generator, which we will use for the next phase of the project. Although this phase of the project does not use the **CUP grammar**, it does require specifying the tokens in the **CUP input file**. You will need to update those definitions to ones appropriate for the full MiniJava language so they can be used by your scanner. Both JFlex and CUP are included in the starter code, and there is also a starter project that you should use to see how these tools work together.
Note: If you are using a different implementation language, you will need to use the starter project as a guide for how to set up the infrastructure for the project in your chosen language. The rest of this description is written in terms of **Java/JFlex/CUP**. Please make the appropriate adjustments to adapt to your language.

2 Your Assignment

1. To get started, you should clone the starter project from the course Github organization.
2. You will next need to examine the MiniJava source grammar to decide which symbols are terminals, and which are non-terminals *(hint: be sure to include operators, brackets, and other punctuation – but not comments and whitespace – in the set of terminal symbols).*
3. The starter code contains a TestScanner program that reads a file from standard input, and prints a readable representation of the tokens in that file to standard output. You should be able run it through command prompt with command `ant test-scanner`, or from the inside of your programming environments, such as Eclipse or IntelliJ, by focusing on the Ant built window. This test program is intended to show how to use a **JFlex** scanner, and you will want to study it to see how that works. **But for the compiler itself, you will need to create a more appropriate main program, and you will need to create an appropriate set of tokens for MiniJava.**
4. Specifically:
   (a) You will need to create a Java class named `MiniJava` with a `main` method that controls execution of your compiler. This method should examine its arguments (the `String` array parameter that is found in every Java `main` method) to discover compiler options, and the name of the file to be compiled. When this method is executed using the command

```
java MiniJava -S filename.java
```

the compiler should open the named input file, and read tokens from it by calling the scanner repeatedly until the end of the input file is reached. The tokens should be printed on a standard output (Java's `System.out`) using a format similar to the one produced by the `TestScanner` program in the starter code.

(b) The source code for `MiniJava.java` should be in the top-level project `src` folder, and Ant will compile it automatically along with all the other project files when needed.

(c) The actual details of running MiniJava's `main` method from a command prompt are a bit more complicated, because the Java virtual machine needs to know where the compiled classes and libraries are located. The following commands should recompile any necessary files, and run the scanner:

```
ant
java -cp build/classes:lib/java-cup-11b.jar MiniJava -S filename.java
```

Please note that if you set the CLASSPATH environment variable to point to the library jar file, and compiled classes directory, you should not need to provide the -cp argument on the java command. The build.xml file processed by ant already contains options to specify the class path, which is why you don't have to specify those things to run targets like test-scanner using ant. You can add similar targets to build.xml to run your MiniJava program or other test programs using ant, and you can use additional ant options in build.xml to specify program arguments like -S.

(d) When the compiler (i.e., just the scanner at this point) terminates, it should return an "exit" or status code indicating whether any errors were discovered when compiling the input program. In Java, the method call `System.exit(status)` terminates the program with the given status. The status value should be 0 (normal termination) if no errors are discovered. If an error is detected, the exit status value should be 1.

(e) To test your scanner, you should use a variety of input files, including some that contain legal MiniJava programs, and others that contain random input. Be sure your scanner does something reasonable if it encounters junk in the input file. (Crashing, halting immediately on the first error, or infinite looping is not reasonable; complaining, skipping the junk, and moving on is.) **Remember, it is up to the parser to decide if the tokens in the input make up a well-formed MiniJava program; the scanner's job is simply to deliver the next token whenever it is called.**

(f) **Remember, this assignment only asks you to implement the scanner part of the project. The parser, abstract syntax trees, and CUP grammar rules will come in the next part. You should use your Khoury CS 6410 GitHub repository to store the code for this, and remaining parts of the compiler project.**

3 If You Are Using a Different Implementation Language or Additional Libraries

If you are implementing the project in a language other than Java, or if your code requires libraries that are not part of the standard Java JDK, please be sure that your compiler works as similarly as possible to the specification above - options and input file name as command line arguments, proper return code from the compiler main program, etc. In addition, please be sure to include a README file at the top of your repository that explains:

1. How to compile, and build your project from source code, including descriptions of any necessary tools and libraries.
2. How to run your compiler, and supply command-line arguments or the equivalent, and, if possible.
3. Please, include information about how your compiler can be run from a script (bash, ant, etc.) to make it easier to run your compiler against multiple source programs for testing. An example would be most helpful.
4. Please be sure to include any non-standard libraries, or other resources needed by your compiler in your repository if you can. I will do my best to work with you if you are using different tools, but I do have to be able to compile, and run your code to provide useful feedback, and evaluate your work.

4 What to Submit

The main information I will examine for this phase of the project are your **JFlex** and **CUP** specification files, your `MiniJava` class and `main` program, and your test input files. Include example source files that demonstrate the abilities of your scanner, including at least one with an error in the middle of the file. You should not include the intermediate file(s) produced by JFlex or CUP – machine generated code is generally unenlightening, consisting of a bunch of tables and uncommented code, if it is readable at all. For the same reason, compiler output like .class or .o files should generally not be pushed to the repository.

Once you're done, "turning in" the assignment is simple – create an appropriate tag (**scanner-final**) in your git repository to designate the revision (commit) that I should examine for grading.