CS 431 – Assignment 1 – Spring 2025

Due: Tuesday, Feb. 18^{th} **before 8:00 AM**

Total points: 100

You **must** work in three-member teams on this assignment. If the total number of students in the class is not a multiple of three, then exactly one team must have either 2 or 4 members. Make sure to join a team as soon as possible.

All members of a team will receive the same grade unless I am made aware **before the due date** of significant collaboration issues within the team, e.g., if some team member(s) did not complete a fair share of the work.

Learning outcomes

- Define the lexical structure of a programming language using regular expressions
- Implement a lexical analyzer using a lexer generator

Each assignment will carry a weight between 1 and 5 when it gets factored into your overall project grade for this course. The weight of this assignment is 1. Later assignments will often carry a larger weight, since they get more and more involved. Be aware that each and every assignment in this course requires that all previous assignments be completed successfully.

Preparation

- You should already have read chapter 3 of the Bantam Java manual. If not, do it now.
- Read chapter 7 of the Bantam Java manual, skipping sections 7.2 and 7.4.
- Review your notes on lexical analysis, regular expressions, and JavaCC.
- Make sure that you have `Java` and Apache `ant` installed on your computer.
- Read this handout carefully right away, and again just before submitting your work. **It would be a shame to lose points for failing to follow the directions given in this document.**
- Download the `a1.zip` file from Canvas and expand it to yield a directory called `a1` containing the Bantam Java compiler shell.
- The code you downloaded should now compile. Try it by typing `ant src` in the top-level (i.e., `a1`) directory.
- Run the lexer on a given test file by typing: `./bin/bantamc ./tests/EmptyProgram.btm`, which should produce an error, since the lexer is incomplete.
- Study and fully understand the file `a1/src/lexer/lexer.jj` that you will have to complete.
- Carefully study the `.tokens.ref` files in the `a1/tests/` directory. They contain the expected output of your lexer on all provided test cases.

Problem statement

Your task is to complete the lexical analyzer for the Bantam Java language using the lexer generator JavaCC. You must handle all of the Bantam Java language features listed in the manual (see chapter 3 and section 7.3). After you have studied and fully understood the lexical structure of the Bantam Java language, implement the corresponding regular expressions in the `src/lexer/lexer.jj` file. This is the only file that you may modify for this assignment.

Your lexer must also detect ALL of the lexical errors listed on page 100 of the manual. The `tests` directory contains two source files that together exhibit at least one instance of each error.

You should spend a significant amount of your time on error handling. There are several ways to handle some of the errors. You should pick the simplest one. In general, you'll have to decide what language corresponds to each token. In some cases, this language might contain lexemes that are not valid tokens. This is OK as long as the bad tokens are detected (and registered as errors) in the Java code associated with each TOKEN or SKIP rule.

Since the provided `lexer.jj` code already declares an error handler and automatically invokes it upon recognition of the `EOF` token, your main responsibility pertaining to error handling is to add Java code in some lexical actions to detect and register each error. The error handler is described at the beginning of Section 4.2 of the manual and its API documentation is linked off of the course web page.

Since your lexer's output must match mine exactly down to the level of each character (I will look for mismatches between your output and mine with the `diff` utility), I provided the expected output for all test cases in the `tests` directory in the form of `.tokens.ref` files, which contain the output of my reference compiler.

Here are some additional requirements for this assignment:

1. While you should fully understand the `scanAndPrintTokens()` method, you are not allowed to modify it. In particular, note that it calls the following method to handle the string constants as special tokens.

2. You must complete the `computeLengthAndCheck()` method to meet its specification, as given in the `.jj` file. When implementing this method, remember that each escape sequence counts as ONE character in the Java string. In particular, note the difference between the single character `'\n'` (which is a lexical error within a string) and the two-character substring `"\n"`, which is a valid escape sequence. Finally, the quotes surrounding the string constant are not part of it and thus not counted in its length.
3. Your regular expressions must be correct and as concise as possible.
4. Remember that you are not allowed to use lexical states (except for the default one) nor `MORE` and `SPECIAL_TOKEN` blocks.
5. Your code may not use any "magic numbers", e.g., 5000.
6. All lexical errors must be registered with the handler. Scanning MUST continue until the end of the file in all cases. As many errors as possible must be reported. For example, if a string contains several newline characters, then this error must be reported for each one of them. The same is true for unsupported characters outside of comments and string constants. Inside the `computeLengthAndCheck()` method, you must access the error handler using `token_source.errorHandler.register(...)`. However, in the Java code spliced into the token rules, you must refer to it using `errorHandler.register(...)`. The same difference applies when accessing the constant used as the first argument of the `register()` method.

Testing your compiler

The following instructions assume that your current directory is the top-level `a1` directory. To build your lexer type: `ant src`
To run it on a Bantam Java program in the `tests` directory and have its output sent to a text file in the same directory, type:
`./bin/bantamc tests/file_name.btm > tests/file_name.btm.tokens`

Furthermore, when you are done with this assignment, you should 'cd' to the `tests` directory and type `ant`, which will run your compiler on all test files, produce a `.tokens` file for each one of them, and then compare its contents to the corresponding `.tokens.ref` file (the output of my reference compiler). For full credit, you must pass all tests, that is, all the `.diff` files MUST contain 0 bytes.

Grading

If ALL of your token files match the reference token files exactly and you have met ALL of the other specifications and other submission requirements listed in this handout, then you will earn 100 points. For every mismatch in the output or any other unmet requirement that my testing uncovers, I will take a certain number of points (to be determined at the time of grading) off based on my estimate of the severity of the error.

Submission procedure

Before the deadline, you must upoad to the Canvas A1 dropbox a single file called `lexer.jj`.

In order for all team members to get full credit, all of their names, except for the one who did the Canvas submission, must be listed in full as part of a comment posted on the Canvas assignment submission page.

Points will be deducted if your lexer does not work properly, and/or if you do not follow the naming/formatting/submisssion requirements listed in this handout.

The late-submission penalty stated in the syllabus applies to this and following assignments.

For full credit, your submitted file(s) must be professionally formatted. This includes logical use of indentation and white space, ensuring consistent vertical alignment, and making sure that no single line exceeds 80 characters.

Since formatting will affect grading, do not hesitate to ask me for clarification before the due date.

Good luck and have fun!