CS4120/4121/5120/5121—Spring 2016
Programming Assignment 1
Lexical Analysis
Due: Monday, February 8, 11:59PM

This programming assignment requires you to implement a *lexer* (also called a *scanner* or a *tokenizer*) for the Xi programming language. As discussed in Lecture 2, a lexer provides a stream of *tokens* (also called *symbols* or *lexemes*) given a stream of characters.

0 Changes

- None yet; watch this space.

1 Instructions

1.1 Grading

Solutions will be graded on design, correctness, and style. A good design makes the implementation easy to understand and maximizes code sharing. A correct program compiles without errors or warnings, and behaves according the requirements given here. A program with good style is clear, concise, and easy to read.

A few suggestions regarding good style may be helpful. You should use brief but mnemonic variable names and proper indentation. Keep your code within an 80-character width. Methods should be accompanied by Javadoc-compliant specifications, and class invariants should be documented. Other comments may be included to explain nonobvious implementation details.

1.2 Partners

You will work in a group of 3–4 students for this assignment. Find your partners as soon as possible, and set up your group on CMS so we know who has partners and who does not. Piazza also has support for soliciting partners. If you are having trouble finding partners, ask the course staff, and we will try to find you a group in a fair way.

Remember that the course staff is happy to help with problems you run into. Read all Piazza posts and ask questions that have not been addressed, attend office hours, or set up meetings with any course staff member for help.

1.3 Package names

Please ensure that all Java code you submit is contained within a package whose name contains the NetID of at least one of your group members. Subpackages under this package are allowed; they can be named however you would like.

1.4 Tips

This assignment is much smaller than future assignments will be: it is intended primarily as a warmup assignment that gives your group the chance to practice working together. The later assignments will test your ability to work effectively as a group. This is a good time to set up the infrastructure that you will use for the rest of the semester. Some tips:

- Meet with your partners as early as possible to work out the design and to discuss the responsibilities for the assignment. Keep meeting and talking as the project progresses. Be prepared for your meetings. Be ready to present proposals to your partners for what to do, and to explain the work you have done. Good communication is essential.
- One way to partition an assignment into parts that can be worked on separately is to agree on, first, what the different modules will be, and further, exactly what their interfaces are, including detailed specifications.

2 Design overview document

We expect your group to submit an overview document. The Overview Document Specification outlines our expectations. Writing a clear document with good use of language is important.

These are key topics to include in your design overview document:

- Have you thought about the key data structures in this assignment?
- Have you thought through the key algorithms and identified implementation challenges?
- Have you thought about your implementation strategy and division of responsibilities between the group members?
- Do you have a testing strategy that covers the possible inputs and the different kinds of functionality you are implementing?

3 Version control

Working with group members effectively is a key learning goal for this project. To facilitate this goal, we would like you to use version control in managing your partnership. You may choose to use any system you like; common industry standards include Git, Subversion, and Mercurial. You must submit file `pa1.log` that lists your commit history from your group. This is not extra work, as version control systems already provide this functionality. While it may require some learning, using version control is a valuable skill to have. In the short term, you will reap the benefits as you delve further into the project. In the long run, any large piece of modern software is always managed with version control.

4 Lexer

We encourage you to use a lexer generator such as JFlex in your implementation, but this is not required. If you do use a lexer generator, you may wish to consider using the adapter pattern to aid

you in your implementation.

5 Command-line interface

A command-line interface is the primary channel for users to interact with your compiler. As your compiler matures, your command-line interface will support a growing number of possible options.

A general form for the command-line interface is as follows:

```
xic [options] <source files>
```

For this assignment, two options are possible:

- `--help`: Print a synopsis of options.
  A synopsis of options lists all possible options along with brief descriptions. No source files are required if this option is specified. Invoking `xic` without any source files should also print a synopsis. To see an example of a synopsis, run `javac` from your command line.
- `--lex`: Generate output from lexical analysis.
  For each source file named `filename.xi`, an output file named `filename.lexed` is generated to provide the result of lexing the source file. Each line in the output file corresponds to each token in the source file in the following format:

  ```
  <line>:<column> <token-type>
  ```

  where <line> and <column> indicate the beginning position of the token, and <token-type> is one of the following:

  - `id` <name> for an identifier
  - `integer` <value> for an integer constant
  - `character` <value> for a character constant, where `value` excludes enclosing quotes
  - `string` <value> for a string constant, where `value` excludes enclosing quotes
  - <symbol> for a symbol such as parentheses, punctuation, and operators
  - <keyword> for a keyword, including names and values such as `int` and `true`

  Non-printable and special characters in character and string literal constants should be escaped in the output, but ordinary printable ASCII characters (e.g., "d") should not be. Comments and whitespace should not appear in the output.

  A lexical error should result in the following line in the output file:

  ```
  <line>:<column> error:<description>
  ```

  where <description> details the error. All valid tokens prior to the location of the error should be reported as above.

  Table 1 shows a few examples of expected results.

Content of input file	Content of output file

`use io`
 	
`main(args: int[][]) {`
`  print("Hello, Worl\0x64!\n")`
`  c3po: int = 'x' + 47;`
`  r2d2: int = c3po   //  No Hans Solo`
`}`
	`1:1 use`
	`1:5 id io`
	`3:1 id main`
	`3:5 (`
	`3:6 id args`
	`3:10 :`
	`3:12 int`
	`3:15 [`
	`3:16 ]`
	`3:17 [`
	`3:18 ]`
	`3:19 )`
	`3:21 {`
	`4:3 id print`
	`4:10 (`
	`4:11 string Hello, World!\n`
	`4:26 )`
	`5:3 id c3po`
	`5:7 :`
	`5:9 int`
	`5:13 =`
	`5:15 character x`
	`5:19 +`
	`5:21 integer 47`
	`5:23 ;`
	`6:3 id r2d2`
	`6:7 :`
	`6:9 int`
	`6:13 =`
	`6:15 id c3po`
	`7:1 }`

`x:bool = 4all`
`x = ''`
`this = does not matter`
	`1:1 id x`
	`1:2 :`
	`1:3 bool`
	`1:8 =`
	`1:10 integer 4`
	`1:11 id all`
	`2:1 id x`
	`2:3 =`
	`2:5 error:Invalid character constant`

Table 1: Examples of running `xic` with `--lex` option

6 Test harness

We will provide a test harness, along with sample test cases, that you can use to test your implementation. Watch for updates in this section.

7 Submission

You should submit these items on CMS:

- `overview.txt/pdf`: Your overview document for the assignment. It should also include descriptions of any extensions you implemented.
- A `zip` file containing these items:
  - *Source code*: You should include all source code required to compile and run the project. Please ensure that the directory structure of your source files is maintained within the archive so that your code can be compiled upon extraction. If your code depends on any third-party libraries, please include compilation instructions in your overview document.
    If you use a lexer generator, please include the lexer input file, e.g., `*.flex`, as well as the generated code.
  - *Tests*: You should include all your test cases and test code that you used to test your program. Be sure to mention where these files are in your overview document.

  Do not include any non-source files or directories such as `.class`, `.classpath`, `.project`, `.git`, and `.gitignore`.
- `pa1.log`: A dump of your commit log from the version control system of your choice.