CS365 – Organization of Programming Languages
Program 2

Objective
Learn the basics of programming an ad-hoc language scanner

Due Date
2/21/2020

Assignment
Build the start of an ad hoc scanner to handle a simple language. The output for the scanner will be a list, and in order, of the type of token, a comma, and the actual value of the token (the token and the lexeme).

- This must be written in Java.
- If you create more than one Java file, compress all of your files into a single file for uploading.
- You must use file input. Pick up the source code file name from the command line. Exit **gracefully** with an error message if the input file does not exist.
- Write the output to the screen.
- There is no error checking required of file names OTHER than to ensure if they are present – your program will not be given an illegal file name when testing.
- There is no guarantee of whitespace except *where required* to separate tokens. A line feed is a form of whitespace. There will not be any tab characters in the source code.
- An `<id>` token is a string of alphabetic characters starting with an upper or lower case `‘a’-‘z’`, followed by either an upper or lower case `‘a’-‘z’` or `‘0’-‘9’` (standard identifier rules without the underscore character). An `<id>` cannot be one of the reserve words in the grammar.
- A `<number>` can be either an integer or a floating point value. A floating point value WILL have at least one digit both in front and after a decimal point.
- A `<rel_op>` token is either a `<`, `>`, `<=`, `>=`, `==`, or `!=`

- The `<assign>` token, the assignment operator, is the equal sign (“`=`”).
- The “reserved” words in the grammar are all lowercase and case-specific. They include: `input,` `print`, `begin`, `end`, `if`, and `else.` Their respective tokens are `<input>`, `<print`, `<begin>`, `<end>`, `<if>`, and `<else>.`
- A `#` symbol indicates a comment. Ignore the remainder of the current physical line. This will not generate a token.
- If your program encounters text that is not a valid token, write out an `<error>` token and the invalid text (not the remainder of the file). Stop processing the input file after dealing with an error.

For example, if the input file contains:

```
#sample “source code” for simple language
min = 0
input a
input b
if a < b begin
   min = a
end else
   min = b
print min
```

your output should contain:

```
<id>, min
<assign>, =
<number>, 0
<input>, input
<id>, a
<input>, input
<if>, if
<id>, a
<rel_op>, <
<id>, b
<begin>, begin
<id>, min
<assign>, =
<id>, a
<end>, end
<else>, else
<id>, min
<assign>, =
<id>, b
<print>, print
<id>, min
```