P1

Due Sunday by 11:59pm **Points** 100 **Submitting** on paper

Scanner

```
/accounts/classes/janikowc/submitProject/submit_cs4280_P1 SubmitFileOrDirectory
```

Implement scanner for the provided lexical definitions.

The scanner is embedded and thus it will return one token every time it is called. Since the parser is not available yet, we will use a tester program to call the scanner.

The scanner could be implemented as

1. Plain string reader - read strings separated by spaces - (**70 points**) assuming
   - all tokens must be separated by spaces
   - lines may not be counted (**75 if counted**)
   - comments may be without spaces (will be tested without spaces in this option)
2. Module generated by lex (**75 points**)
3. FSA table + driver (**100 points**)

- You must have the README.txt file with your submission stating on the first line which option you are using: 1, 2, or 3, and if 3 then include information where the FSA table is and which function is the driver. If this information is missing, the project will be graded under option 1
- Implement a token as a triplet {tokenID, tokenInstance, line#} (if option with line numbers)
- Dont forget EOFtk token
- Implement the scanner in a separate file with basename "scanner"
- For testing purposes, the scanner will be tested using a testing driver implemented in file with basename "testScanner". You need to implement your own tester and include as a part of the project. This tester will ask the scanner for one token at a time and display the token to the screen, one per line, including information (descriptive) on what token class, what token instance, and what line, if applicable.
- Invocation:

  ```
  scanner [file]
  ```

  to read from stdin or file *file*.sp2020
  - Arguments are the same as P0

    - Wrong invocations may not be graded
    - Dont confuse executable name with file name with function name
- Graded **20 points** for style regardless of implementation method
- You must have (C++ can be accordingly different)
    - types including token type in token.h
    - implement scanner in scanner.c and scanner.h
    - implement the tester in another file testScanner.c and testScanner.h
    - main.c processing the arguments (as in P0) then calling testScanner() function with interface and preparation as needed.