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 and no token has invalid characters
    - lines may not be counted (**75 if counted**)
    - comments may be without spaces if it helps
2. 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 or 2.

If 2, then on the next line explain the location of your table if the code and attach your graph picture in the folder named GRAPH.
If 1, then on the next line explain if lines are counted and used in tokens.

- Implement a token as a triplet {tokenID, tokenInstance, line#} (line# if doing 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 another file with basename "testScanner". You need to implement your own tester and include as a part of the project. The tester will call 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*.fs19

    - Arguments are the same as P0
    - Wrong invocations may not be graded
- Graded **20 points** for style regardless of implementation method, the rest on performance
- You must have (C++ and other languages 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 P0) then calling testScanner() function with interface and preparation as needed.