CS 4323
PROGRAMMING ASSIGNMENT #1
(SCANNER)

Due: March 6, 2018

This project builds the first component of a compiler, the lexical analyzer (or scanner), for TrumpScript++ language. The Internet explanation of the syntax of TrumpScript was incomplete, so I mixed it with other typical language features (thus the name TrumpScript++) and constructed context-free rules defining this modified (and simplified) TrumpScript. These rules will be further transformed later, if necessary, into the form with which LL(1) parsing can be done. A few funny features of this language include:

- Every program must start with the message "Make programming great again" and end with "America is great".
- There is no import statement, America doesn't need it.
- There is no floating-point number, all numbers are integers greater than a million.
- There is no input statement, Trump doesn't need it.
- Boolean values are denoted by fact and lie, meaning true and false, respectively.
- The language is case-insensitive.
- Error messages are maximally rude to programmers.

Syntax rules for TrumpScript++: Note that nonterminals are bracketed with < and >, [id] ([const] or [string]) is any terminal token recognized by the scanner as an identifier (constant or string), and \epsilon is the empty string.

1. <Trump> -> <first> <stmts> <last>
2. <first> -> Make programming great again
3. <last> -> America is great
4. <stmts> -> <stmt> <stmts> | \epsilon
5. <stmt> -> <decl> | <asmt> | <cond> | <loop> | <output>
6. <decl> -> make [ids] <type>
7. <type> -> number | boolean
8. <asmt> -> [id] is <expr>
9. <cond> -> if , <bool> ; : <stmts> ! else : <stmts> !
10. <loop> -> as long as , <bool> ; : <stmts> !
11. <output> -> tell <ids> | say [string]
12. <ids> -> [id] <more-ids>
13. <more-ids> -> [id] <more-ids> | \epsilon
14. <expr> -> <bool> | <arith>
15. <bool> -> fact <bool-tail> | lie <bool-tail> | not <bool> <bool-tail> | <arith> <test> <arith> ?
16. <bool-tail> -> and <bool> | or <bool> | \epsilon
17. <test> -> less | is | more
18. <arith> -> [id] <arith-tail> | [const] <arith-tail> | (<arith>) <arith-tail>
19. <arith-tail> -> plus <arith> | times <arith> | \epsilon

Lexical definition: There are five types of tokens. The lexical analyzer (scanner) is a DFA recognizing these tokens.

- **Keywords**: make, programming, great, again, america, is, great, number, boolean, if, as, long, tell, say, fact, lie, not, and, or, less, more, plus, times.
- **Identifiers**: any letter followed optionally by digits and/or letters.
- **Constants**: any sequence of digits whose corresponding value is greater than 1,000,000.
- **Strings**: any sequence of characters in a pair of " and ".
- **Special symbols**: , ; : ! ? ( ).

For your programming assignment #1, write a program (in Java or any programming language of your choice) with subprocedures/classes SCANNER(), BOOKKEEPER(), and ERRORHANDLER() that handle the five types of tokens defined above.

Construct SCANNER() from a DFA accepting these tokens. A blank (many consecutive blanks are the same as a single blank), line break or special symbol separates two tokens. Symbols following # (up to a line break) are comments; # and these comment symbols must be ignored by the scanner. The symbols " and " used to define a string must also be ignored by the SCANNER(). Call SCANNER() from the main body of the program, once for each token to be recognized, until all symbols in the given input program are consumed. Thus, the main body will contain a loop in which SCANNER() is called repeatedly, until all symbols of the source program are consumed. It is important that your SCANNER() consists of blocks of codes for states of the DFA recognizing the five types of tokens, as discussed in class. Using a method not implementing a DFA will result in zero credit for this project,

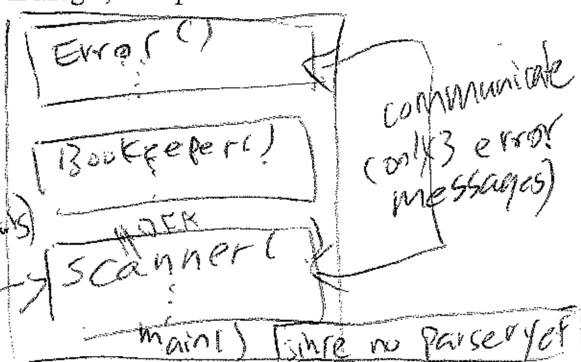
Call BOOKKEEPER() from SCANNER() when an identifier, constant or string is recognized. It is responsible for maintaining a symbol table SYMTAB (of size 100) to store tokens passed from SCANNER() and their attributes, i.e., classification as to identifier, constant or string. Each identifier, constant or string must appear exactly once in the SYMTAB.

Call ERRORHANDLER() from SCANNER() when an illegal token is recognized. It is responsible for producing appropriate error messages. There are three types of error messages; no information other than one of these three (such as the location of the error and/or possible error correction) should be printed.

- [id] error: This is a country where we speak English.
- [const] error: I'm really rich, part of the beauty of me is I'm very rich.
- Any other error: Trump doesn't want to hear it.

For output, print out the following:

- Print the input program exactly as you stored in your input file.
- For each token, if it is a legal one then print the token and its type. If illegal, then print the token and an error message.
- Print the content of SYMTAB.



output: source code
① / lexical output
② = # of tokens
③ / symbol table content

Your program must scan the whole source program, finding all legal and illegal tokens. Run your program on the following input:

↑ comment, skip the rest # routine

Make programming great again
main body begins
Make x number make y1 z2zz 1w numbers
make a b Boolean
X is 1000000 y1 is 2000000 z is 123456789
A is fact b is lie
As long as fact or lie :

Tell x y1 z2zz say "continue"
If, x plus (y) times 2000000 more z? ; : tell a b say "stop" ! else : make c boolean !
C is not not not fact and x less z ? or lie
Tell a b c x y z
Say "done" # say done
!
America is great

Submit a hard copy of your program and output; no electronic submission will be accepted. The following will be considered for grading purpose:

- Correctness as to whether your program has been designed as instructed above and whether your program runs as intended.
- Documentation, as is usual in any software design, and neatness.

Your work must be submitted on the due date in class before the class meeting begins. Penalty for late submission: 10% per calendar day. A partial credit of no more than half the full credit can be given for an incomplete work.

This project, as well as all other student activities in this course, is an individual (not group) assignment. Plagiarism will result in academic misconduct charges.

Assignment handed out on February 13, 2018.

Scanner() = DFA
one execution of scanner will find one token, and go back to the main program and print that token.
in main, print token + token type;
if token is invalid, token type is error message.