CSC 45500 Project #1

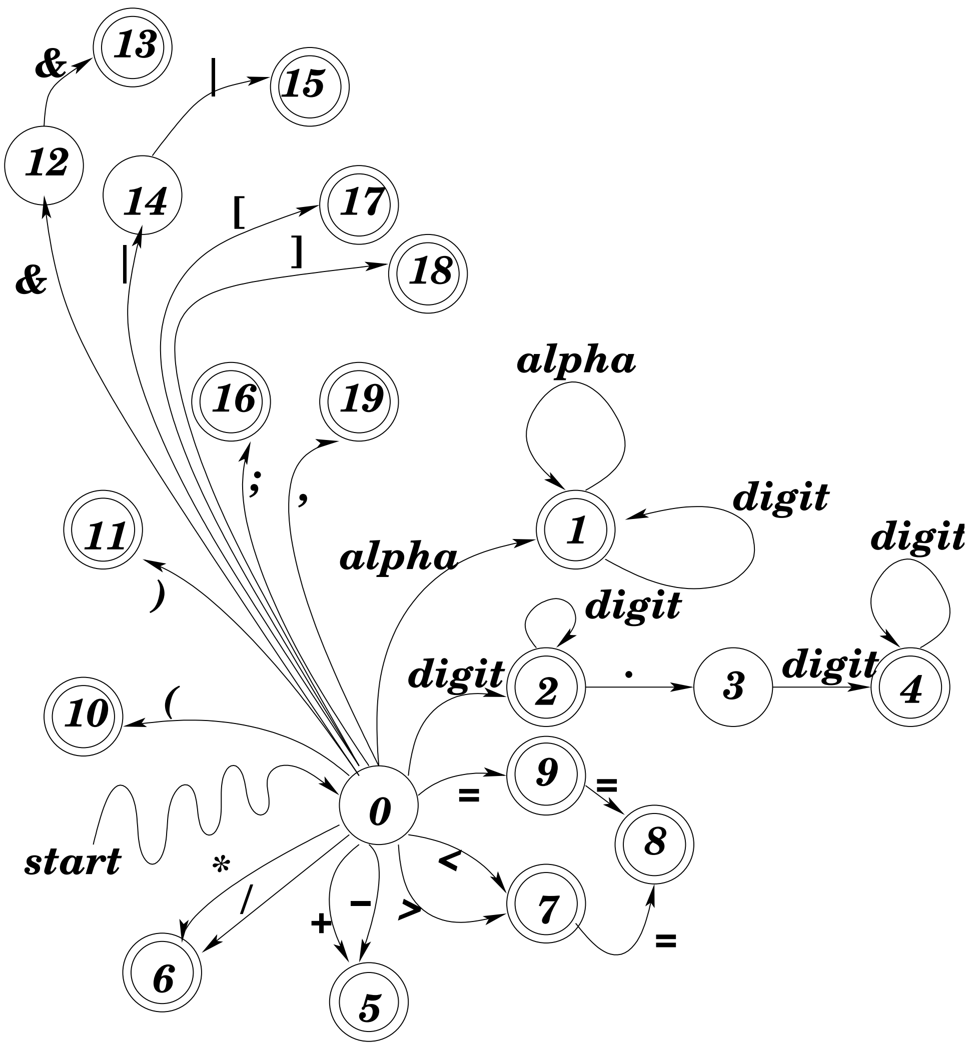
DUE: Tuesday, January 23, 2024, 11:59 PM

1 Objectives

- building a lexical analyzer
- implementation of an automata

2 Problem Statement

You are to build a lexical analyzer for a programming language. Note that you are *not* (yet) to build a parser for the language, seeing as you have not been given a language specification (i.e. grammar). Valid tokens in the language can be determined through utilization of the following automata:



This automata "returns" different token types based on which final state has been reached. The following table summarizes which token type should be "returned" when in a particular final state:

Final State	Token Type	Final State	Token Type
1	ID	10	LPAREN
2	NUM_INT	11	RPAREN
4	NUM_REAL	13	AND
5	ADDOP	15	OR
6	MULOP	16	SEMICOLON
7,8	RELOP	17	LBRACK
9	ASSIGNOP	18	RBRACK
		19	COMMA

Note that as specified in the automata, all keywords will be accepted as an ID. However, you are to return keywords with a token type equivalent to their keyword. The following table lists the keywords and their associated token types, noting that keywords should be case *sensitive* (i.e. they must be lower case in the input file):

Keyword	Token Type	Keyword	Token type
void	VOID	if	IF
int	INTEGER	then	THEN
float	FLOAT	else	ELSE
begin	BEGIN	while	WHILE
end	END		

All comments and whitespace (' ','`\t`','`\n`') should be ignored. Comments are specified as beginning with a '#' and continue to the end of that line.

3 Problem Details

You are to use the provided code as described below:

- the files `Token.hpp` and `Token.cpp`, which give the member function prototype `get` and the class definition it is found in. *You MUST implement the* `get` *function and use its associated class* **AS IS** *!!!*
- the file `main.cpp`, which contains the `main` function. You *must* use this main function *AS IS* !!!
- You may add code to the `Token.cpp` file (in fact, you will likely need to do so), but you may not modify any of the existing code. You are also not allowed to change the class definition.

4 Submission

You should post to Canvas both your `C++` source code file and a plain text file (not an MS Word file) called `read.me` in a zip or tgz file. Make sure the "root" of your submission is a folder (not just a collection of files.)

The read.me file should contain:

- your name and any other identifying information.
- What platform you developed your solution on (i.e Linux, OS/X, Windows (God forbid!), ...)
- any special details on how to compile your project
- any special details on how to run your project - note that you cannot circumvent the project specifications here!
- any known bugs your project has, and possible fixes to those bugs (partial credit abounds here).
- an overview of how you solved the project.
- You may put any other information you like in this file, although you are not required to do so - one common additional set of entries is a "software engineering log" that indicates what you have done every time you sat down and worked on the project. Many programmers find that such actually helps you to finish projects faster!

deadline, and you get a free 5 points.

5 Grading Breakdown

Correct Submission	10%
Code Compiles	20%
Following Directions	20%
Correct Execution	40%
Code Formatting, Comments, & read.me file	10%
Extra Credit (early submission)	5%

6 Final Notes

- start this project now.
- You may *not* use the program `lex`, `flex`, or `flex++` (found in most Unix distributions) to solve this problem.
- Have you started this project yet? If not, start NOW.
- Do NOT try and implement more than a lexical analyzer at this point in time!! DO NOT WASTE TIME BY DOING UNNECESSARY WORK!
- If you haven't yet started this project, you'd better start **now**.
- If you have *any* questions about *anything* regarding this project, ASK ME IMMEDIATELY !!!!!
- START **NOW !**