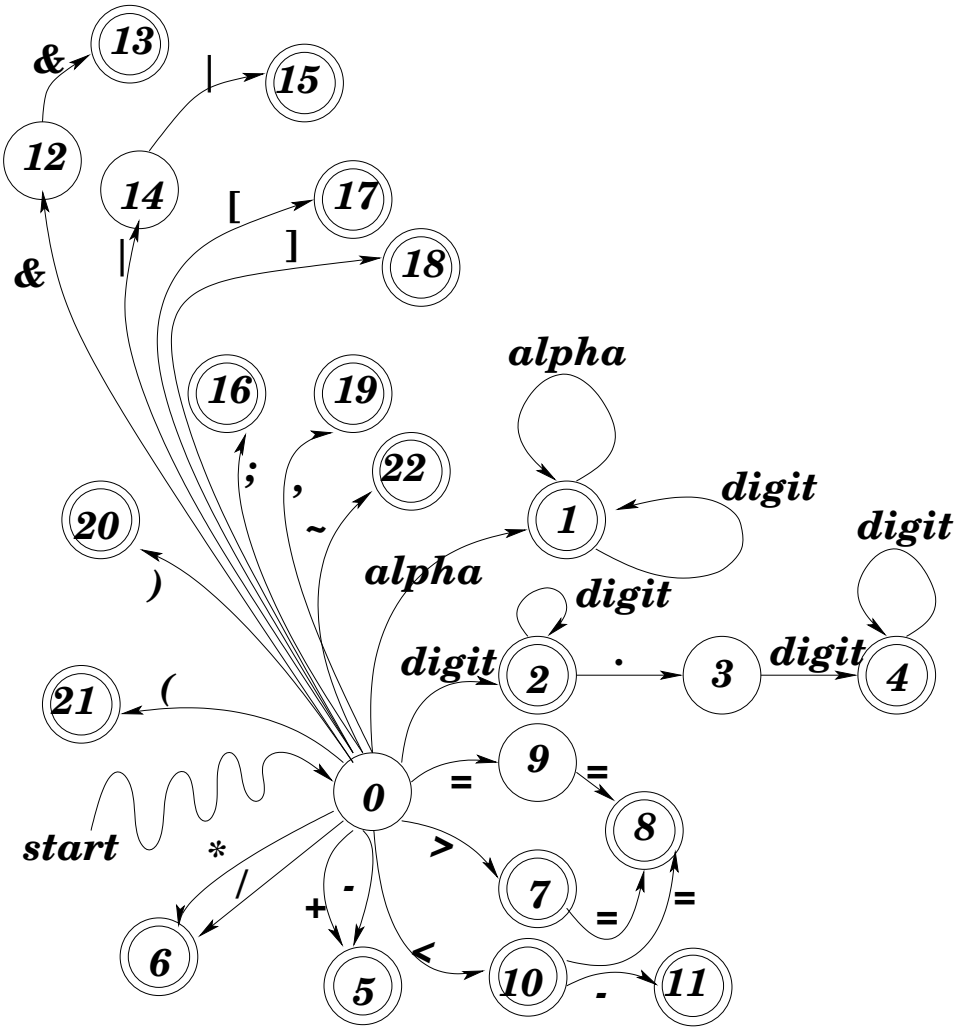
CSC 40800 Project #1
DUE: Monday, September 30, 2024, 11:59 PM

1 Objectives

- interpretation of an automata
- building a lexical analyzer
- using the `lex/flex/flex++` tool.

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:

You must use either the `lex`*,* `flex`*, or* `flex++` *tool for this project.*

Final State	Token Type	Final State	Token Type
1	ID_T	15	OR_T
2	NUM_INT_T	16	SEMICOLON_T
4	NUM_REAL_T	17	LBRACK_T
5	ADDOP_T	18	RBRACK_T
6	MULOP_T	19	COMMA_T
7,8,10	RELOP_T	20	RPAREN_T
11	ASSIGNOP_T	21	LPAREN_T
13	AND_T	22	NOT_T

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_T	if	IF_T
int	INTEGER_T	then	THEN_T
float	FLOAT_T	else	ELSE_T
begin	BEGIN_T	while	WHILE_T
end	END_T		

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 the `Token` class. This gives the ability to construct tokens and print them out (among other things.)
- the file `main.cpp`, which contains the `main` function. You *must* use this main function *AS IS* !!!
- Essentially, you are building a lex/flex file to implement the lexical analyzer.

4 Submission

You should post to Canvas your `.l` file and any additional `C++` source code files you created for this project. You should also include a plain (ASCII) text file (not an MS Word file) called `read.me`. Your submission should be contained within a foder that is submitted via a single 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	30%
Correct Execution	30%
Code Formatting, Comments, & read.me file	10%
Extra Credit (early submission)	5%

6 Final Notes

- 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 !**