COSC 4316 Lab Spring 2020 Burris

Due: The scanner/Lexical Analyzer is be complete by February 28, 2020. ***You will receive a single grade for the semester project***. A grade of "C" or higher for the project requires you successfully translate and execute specific programs designed to test your translator implementation. **The complete project must be submitted no later than April 24, 2020.**

Cheating:

Possession of program listings, machine readable code, or any other material associated with the construction of a language will be construed as cheating. This assignment has been designed to provide students with the minimum number of constructs to develop functional competence combined with the required theoretical basis with a minimum amount of work. You must do your own work. *Any verifiable instance of cheating will result in course failure for all concerned*. It will be one calendar year before the course can be repeated to raise your grade as COSC 4316 is only taught in the spring semester. Please help each other in the sense of answering questions or help in debugging a problem. *You may not work together to produce any part of the project*. All work must be accomplished individually!

Alternative #1

Semester Project (Part 1):

Write a lexical analyzer as a table-driven problem solver to construct tokens using the "Scanner" grammar for Java 0 on page 61 of the hymnal **one character at a time.** Use the sample program on page 63 of the hymnal as your test data. Each time the scanner is called, it should return the next token and its class. You may create your own class names. I recommend you consider the class names listed below. In addition, your program should create a symbol table as described in class. Submit your working scanner, a print out of symbols with class designations, and a printed copy of the symbol table. You must also submit a copy of the state diagram used to create the table that drives your solution. It should be easy to map your state diagram to the table values in your program. I recommend you write the tokens and corresponding classification returned by the scanner to a file, one line per token. This output can be used latter as input for a second pass by your language translator. You should also consider writing the symbol table to a text file for use in future passes of the translator. In general, translators make multiple passes to complete their task using a large number of intermediate files. The higher the quality of the resulting code generated by the translator the more likely it is the translator will use a substantial number of passes and intermediate files.

You are being graded more on program technique than correctness of the answers. I assume you will get the correct answers. Your program must reflect

the table-driven techniques we have learned based of finite state automata's (FSA). You may use the example on page 47 of the hymnal as a starting point. Use of simplistic techniques such as the "StringTokenizer" in Java to create tokens is unacceptable and will result in a grade of zero. **Specifically, you must process the input one character at a time in your scanner to create tokens**. This may be accomplished by reading one character at a time from the input or by reading a line at a time from the input and then processing the line one character at a time.

Suggested classes (actual program code on next page):

Token	**Classification**
CLASS	$CLASS
LargestCommonDenominator	<var>
{	$LB
M	<var>
=	<assign>
7	<integer>
,	<comma>
N	<var>
=	<assign>
85	<integer>
;	<semi>
VAR	<$var>
◦◦◦	◦◦◦
}	$RB
CONST	$CONST
PROCEDURE	$PROCEDURE
WHILE	$WHILE
DO	$DO
<	<relop>
<=	<relop>
>	<relop>
>=	<relop>
==	<relop>
!=	<relop>
IF	IF
THEN	$THEN
ELSE	$ELSE
/	<mop>
*	<mop>
+	<addop>
-	<addop>
CALL	$CALL
(	$LP
)	$RP

/*	$LComment
*/	$RComment

Assume the following program:

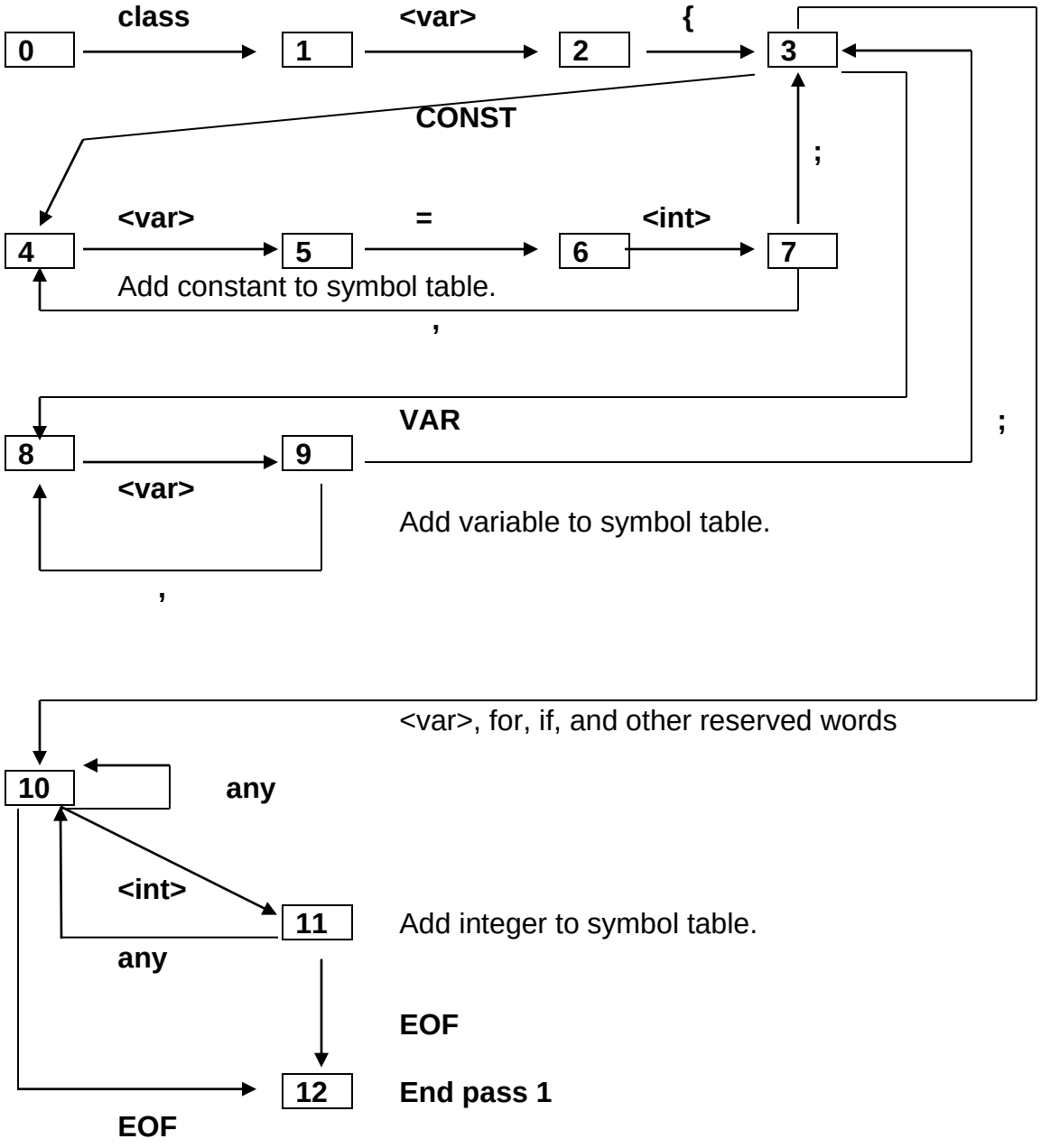
```
CLASS Pgm1 {
        CONST M = 13, N = 56;
        VAR X, Y, Z;

        Y = 97;
        X = M * N + 18 – y;
}
```

Symbol Table

Token	Class	Value	Address	Segment
Pgm1	<Program Name>		0	CS
M	$CONST	13	0	DS
N	$CONST	56	2	DS
X	<var>		4	DS
Y	<var>		6	DS
Z	<var>		8	DS
97	$NumLit	97	10	DS
18	$NumLit	18	12	DS

May I suggest the following strategy as a basis for the first pass to build the symbol table and token/class list.



We assume correct programs only. We will not be ready to handle errors efficiently until a later date.

Note both the scanner and logic for the first pass are based on our knowledge of FSA and Chomsky Type 3 grammars.

"C" Option:

You must implement all of the following: CONST, VAR, compound statements [{ and }], integer I/O, generate a source listing, and a code listing. In addition, your translator must process all arithmetic assignment statements. Implement either the "IF" or "WHILE" statement. You have to implement the ">" and ">=" relational operators though you are encouraged to implement all relational operators. Nesting is not required. **Your code must assemble and execute to be acceptable**. Assume all programs will be correct. Hence error messages are not required. Write several sample programs to demonstrate the features of your translator. Source code and executables for these programs as well as the translator must be supplied in machine readable format. Supply a printed copy of all program results with the source code***. For all grading options: You have not completed the grading option successfully unless all your programs compile, assemble, and execute with appropriate I/O!***

"B" Option:

Complete the "C" option but implement both the "IF" and "WHILE" statements. One must allow for reasonable nesting (at least 5 deep). Some error messages are expected. Write several sample programs to demonstrate the features of your translator. Source code and executables for these programs as well as the translator should be supplied in machine readable format as well as appropriate listings. Supply a printed copy of all program results with the source code.

"A" Option:

Complete the "B" option. Implement procedure calls (methods). To score above 95, you must allow for recursion. Improved error messages and some code optimization is expected. You are encouraged to include subscripted variables. Write several sample programs to demonstrate the features of your translator. Source code and executables for these programs as well as the translator should be supplied in machine readable format as well as appropriate listings. Supply a printed copy of all program results with the source code.

The final lab is due prior to April 24, 2020!
You will be required to make arrangements outside class to demonstrate "A" or "B" grading options.

Alternative #2

Rather than implementing a translator for Java 0, you have the flexibility of implementing a translator for a subset of any popular programming language. The grading restrictions for each grading option in Alternative 1 must be met. You are encouraged to consider implementing arithmetic operations but I would be happy to accept a reasonable set of string operations as an alternative to arithmetic operations or in addition to arithmetic operations. It is simpler to implement static string declarations such as “char A[2]” than dynamic such as “char A*” (using “C” as an example). You need not restrict yourself to languages requiring strict declarations. You may wish to consider something different such as Python, html, xml, or pdf interpreter.

Regardless of your selection, you will be held to the restrictions of each grading option, e.g., writing the parser as a table driven DFSA using one character at a time from the input stream. If the language you select does not support the constructs of Java 0, then we will negotiate suitable replacements consistent with your language selection. Your selection must produce executable assembly language! You must demonstrate your translator executes properly.

Grading Guide for both Alternatives.

Lab Grading Guideline Sheet Spring 2020 (modified to fit the language if you do not select Java 0 by me)

Minimum options for each level.

“C” Option:

Criteria	**Yes**	**No**	**All**
Working “C” option			
FSA, table driven			
PDA, table drive.			
Symbolic addresses in code			
Numeric addresses in code and symbol table			
CONST			
VAR			
compound statements			
integer I/O			
source listing			
arithmetic assignment statements			
“IF” or “WHILE” statement			
“>” and “>=”			
Error checking.			
Executable programs demonstrating success!			

"B" Option:

Criteria	Yes	No	All
Both the "IF" and "WHILE"			
Nested "if" or "while"			
Error checking.			
Executable programs demonstrating success!			

"A" Option:

Criteria	Yes	No	All
Procedure calls.			
Recursion			
Subscripted variables.			
Error checking.			
Executable programs demonstrating success!			

COSC 4316 Lab Final Presentation:
We are approaching the point you will wish to submit your translator for grading. When you demonstrate your translator, I will give you a program or programs to type and execute. Please have each of the following programs ready to compile and run based on the grading option you select. Provide me with program listings and screen shots of their execution. Proper declaration of variables, constants, and other language constructs defined by the grammar is required.

"C" Option:

1) For the "C" option compute ans = a * ((Bob +Jane -10) /2 * 4)/ (b + c). Read values for a, b, c, Bob, and Jane from the terminal and print the result.
2) For a simple, read two integers from the terminal and print the largest value.
3) For the nested if, read three integers from the terminal and print the largest.
4) Compute N! using a "while" or "for" loop. Read the value of N from the terminal and print the result. If you nest for loops, place your factorial calculation in a loop for M iterations. M should be read from the terminal. N! is defined as N*(N-1)! For N > 1. 0! And 1! both equal 1 by definition.

"B" Option Territory:

5) Repeat #4 for the while and nested while statements.
6) Calculate N! using a function. Use a for or while loop to do the actual calculation. All input should be from the keyboard and results placed on the display device in the main program.

"A" Option Territory:

7) Calculate N! using a recursive function. All input should be from the keyboard and results placed on the display device in the main program.

8) Exhibit any programs required to show special features of your translator such as CBV or CBR.

Order of submission:

1) A cover sheet with your name and grading option completed. A Table of Contents is helpful.
2) Your test programs:
   A) Program in our language to be translated.
   B) Assembly language equivalent produced by your translator. It is helpful to have the symbol table.
   C) Prior to this point you should be using a high-level language. I will accept screen shots for the following.
      a) Compile and link the program.
      b) Show the programs execution.

   Repeat step 2 as many times as necessary to exhibit what your program can do.

3) The source code for your translator and any additional information you deed desirable. You must convenience me your translator works to receive a grade of "C" or higher!

Dr. B