(Note for Grader) When testing my program that looks at the lexical and syntactical analysis. Please remember to place spaces between all the lexical tokens in the input file. The professor told me personally I could require this for my program. So for example, a while loop would look like: while ( expression ) { //code blocks ; } rather than: while(expression){//code blocks;}

Syntax I used for my statements

* Switch

The syntax I followed for my switch statement, was similar to C++ switch statement.

Switch ( <expression> ) {  
  case x:  
    *// code block*  
  case y:  
    *// code block*  
  default:  
    *// code block*  
}

* Foreach

The syntax I used for my foreach statement was similar to Java’s for each statement

foreach ( identifier identifier : identifier ) {

//code blocks

}

* For

The syntax I used for my for loop was similar to C++’s for loop statement

for ( <expression> ; <expression> ; <expression> ) {  
   *code blocks*  
}

* While

The syntax I used for my while loop was similar C++’s for loop statement

while ( <expression> ) {  
 *code block*  
}

* Do-while

The syntax I used for my do-while loop was similar c++’s do while statement

do {  
 *code block*  
} while ( <expression> )

* If
* The syntax I used for my if statement was C++’s if statement

if ( <expression> ) {  
   code blocks   
}

* Assignment

The syntax I used for assignment is similar to Java.

Identifier = <expression> ;

* Return

The syntax I used for the return statement is similar to Pythons return statement, so in ours we can just have keyword return with the semicolon, or you can have keyword return with an identifier, float, or integer followed by a semicolon

return ;

Or

return identifier ;

Or

return float ;

Or

return integer

3. Here I typed my explanations for why some of the input strings were not in the grammar, just in case you had a tough time reading my handwriting.

b.) If we begin by selecting S -> a C, we won’t be able to produce two lowercase c’s and a capital C with a lowercase d at the end. As seen, if we create two lowercase c’s then a d, we’ll be missing the capital C. We can’t use that last capital C to produce another ”c C” because then we’d then have 3 lowercase c’s

If we begin by selecting S->A C, we can’t do anything with the first non-terminal, if we make A -> A a, we’ll have too many a’s in our string, and if we make it “a B b”, we’ll have a lowercase b, which isn’t in our input string

Conclusion: This string is not in our grammar.

c.) We cant begin with S -> a C, because we wont ever be able to derive a lowercase b from the non-terminal C

We cant begin with “A C”, because if we convert A into “a B b”, we wont be able to get a capital ‘C’ into the 2nd spot for our string, and if we convert A -> A a, we run into a similar issue with getting a capital C into that 2nd spot.

Conclusion: This string is not in our grammar.

d.) We cant begin with S -> a C,because we wont ever be able to derive a lowercase ‘b from the non-terminal C.

When we take S-> A C, we can’t convert A into “A a”, because we will not be able to have a lowercase b Infront of our ‘d’ at the end (acdabd). But if we convert A into “a B b”, we won’t be able to get the lowercase a in the 4th spot ( acdabd), because it’s impossible to derive a lowercase ‘a’ from the non-terminal B

Conclusion: This string is not in our grammar

e.) We can’t start with S -> a C, because then we’ll never be able to derive a lowercase ‘b’. If we start with S -> A C, we’ll have to convert the non-terminal ‘C’ into d, because our string doesn’t have a lowercase ‘c’, so we can’t do C -> c C.

The other problem we run into is that we can’t create a capital ‘C’ from the non-terminal A, because we would then create a lowercase ‘c’. B can be converted into “c C”, and C can be converted into “c C”. So as seen, we can’t create capital ‘C’, without creating a lowercase c, which isn’t in our string.

Conclusion: Thus, this string is not in our grammar