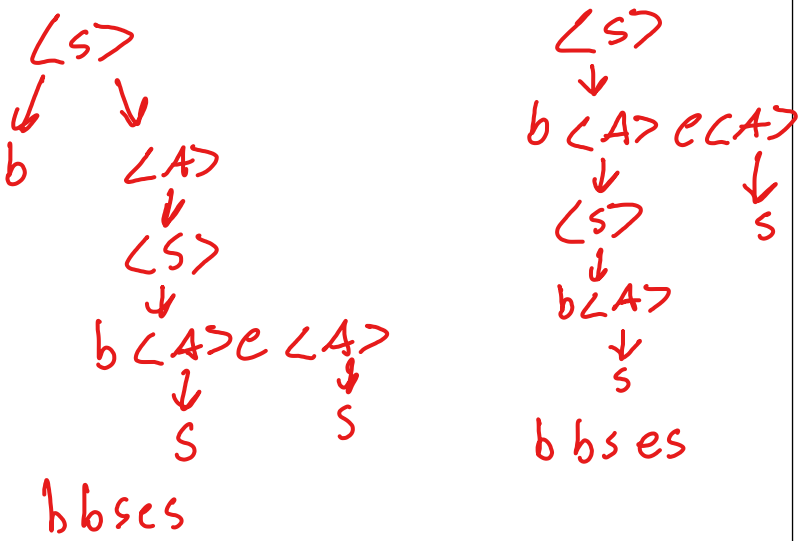
Homework #3

Nathan Petersen (Name) 001 (Sec)

Questions:	Answers:
1. Numbers can be real numbers, with a decimal point, as well as integers. Modify the BNF grammar below to allow reals as operands (i.e. define <number> as <integer> or <real> and then define integers and reals). <number> ::= <digit> | <number><digit> <digit> ::= 0|1|2|3|4|5|6|7|8|9	
2. Sometimes an expression can have two or more kinds of balanced parentheses. For example, Java expressions can have both round and square parentheses and both must be balanced; that is, every "(" must match ")", and every "[" must match"]". Write a grammar for strings of balanced parentheses of these two types. For example ([ ] ([( )])) is in the language but [( ]) is not.	<expression> ::= <left bracket> <inner> <right bracket> <inner> ::= <BLANK> | <expression> <left bracket> ::= (| [<right bracket> ::=) |] I think the book uses epsilon () to represent nothing/blank, here I just used <BLANK> but it represents the same thing.
3. Demonstrate that the following grammar is ambiguous. Create your own ambiguous case. (<S> is the start symbol.) <S> → b <A> <S> → b <A> e <A> <A> → <S> <A> → s	 the same string can be obtained using multiple parse trees, therefore it is ambiguous.

4. The following grammar for a fictitious operator '$' is ambiguous. <number> ::= 0\|1\|2\|3 <expression> ::= <number> <expression> ::= <expression>$<expression> Demonstrate the ambiguity by creating two parse trees for the expression 2 $ 3 $ 0.	2 $ 3 $ 0 -> 2 $ [3 $ 0] -> number $ [number $ number] -> expression $ [expression $ expression] -> expression $ expression -> expression OR 2 $ 3 $ 0 -> [2 $ 3] $ 0 -> [number $ number] $ number -> [expression $ expression] $ expression -> expression $ expression -> expression In our starter expression, one of the left or right branches has to be turned into a sub-expression, but it is ambiguous which one should be. As demonstrated above, either the right two numbers can become a subexpression, or the left two numbers. The associativity of the $ operator isn't defined, so it is ambiguous.
5. Fix the grammar in Problem #4 so that it is not ambiguous. Is your grammar left associative or right associative?	By defining the $ operator as left-associative it removes the ambiguity. We can accomplish this by defining the grammar in such a way that it can only recurse down the left side of the expression. <number> ::= 0|1|2|3 <expression> ::= <expression> $ <number> <expression> :: = <number> 2 $ 3 $ 0 -> (2 $ 3) $ 0 -> <expression> $ <number>
6. Suppose we want to implement a DDC (decimal digit calculator) compiler for the DDC language which performs arithmetic operations on integer arguments. The BNF grammar description below was written to describe the DDC language syntactically. Unfortunately, the grammar is ambiguous. <expr> ::= <term> | <expr> <op1> <expr> <term> ::= <decimal arg> | <term> <op2> <decimal arg> <decimal arg> ::= <digit> | <decimal arg> <digit> <digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 <op1> ::= + | - <op2> ::= * | /	