Homework 4
CS 361

Jen McCall, John Mulcahy

Grammars

Exercise 1:

We consider the BNF grammar below:

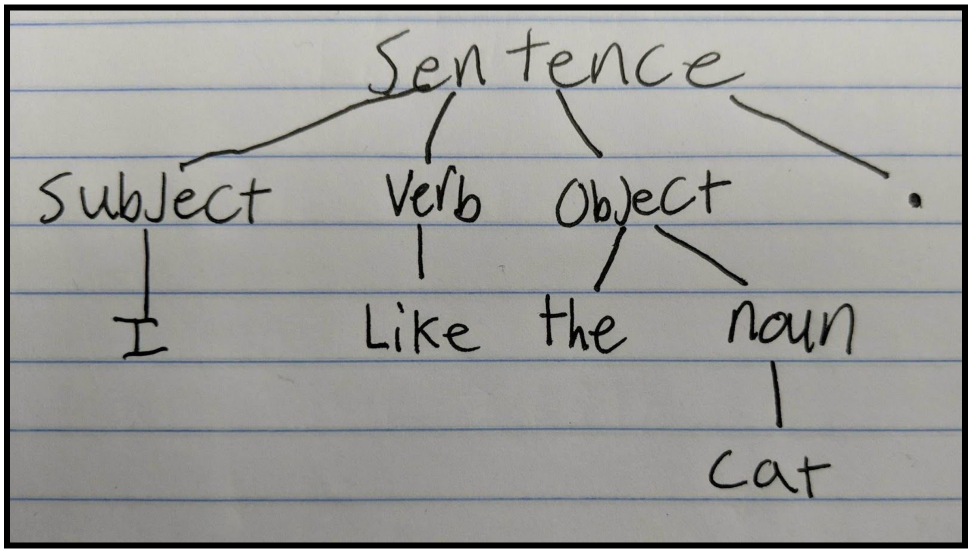
Sentence	::=	Subject Verb Object **.**
Subject	::=	**I** | a Noun | **the** Noun
Object	::=	**me** | **a** Noun | **the** Noun
Noun	::=	**cat** | **mat** | **rat**
Verb	::=	**like** | **is** | **see** | **sees**

a. Show that **I like the cat.** is recognized by this BNF grammar using a rightmost derivation and, then, a parse tree.

Rightmost Derivation:

Sentence => Subject Verb Object . => Subject Verb the Noun. => Subject Verb the cat. => Subject like the cat. => I like the cat.

Parse Tree:



b. Provide an expression that is NOT recognized by the grammar.

Given the requirements for 'Sentence' in the above BNF grammar, the expression "I am a cat." is not recognized. While this sentence is correct in our proper English grammar, in this BNF grammar it is not recognized. The problem lies in the "am" part of the sentence because it follows a 'Subject' and precedes an 'Object' but is not part of the defined 'Verb'.

Exercise 2:

We consider the following grammar:

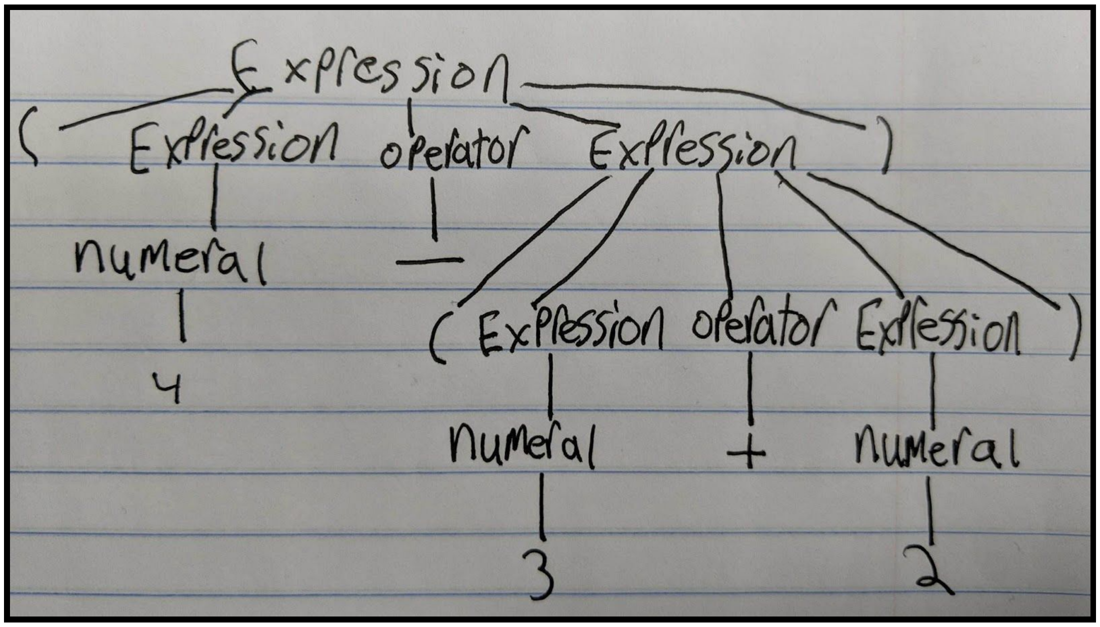
```
EXPRESSION ::= NUMERAL  |  ( EXPRESSION OPERATOR EXPRESSION
)
NUMERAL ::=  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
OPERATORS ::=  +  |  -
```

Show that (4 - (3 + 2)) is a legal EXPRESSION using a leftmost derivation, and,then, a parse tree.

Leftmost Derivation:

Expression => (Expression Operator Expression) => (Numeral Operator Expression) => (4 Operator Expression) => (4 - Expression) => (4 - (Expression Operator Expression)) => (4 - (Numeral Operator Expression)) => (4 - (3 Operator Expression)) => (4 - (3 + Expression)) => (4 - (3 + Numeral)) => (4 - (3 + 2))

Parse Tree:

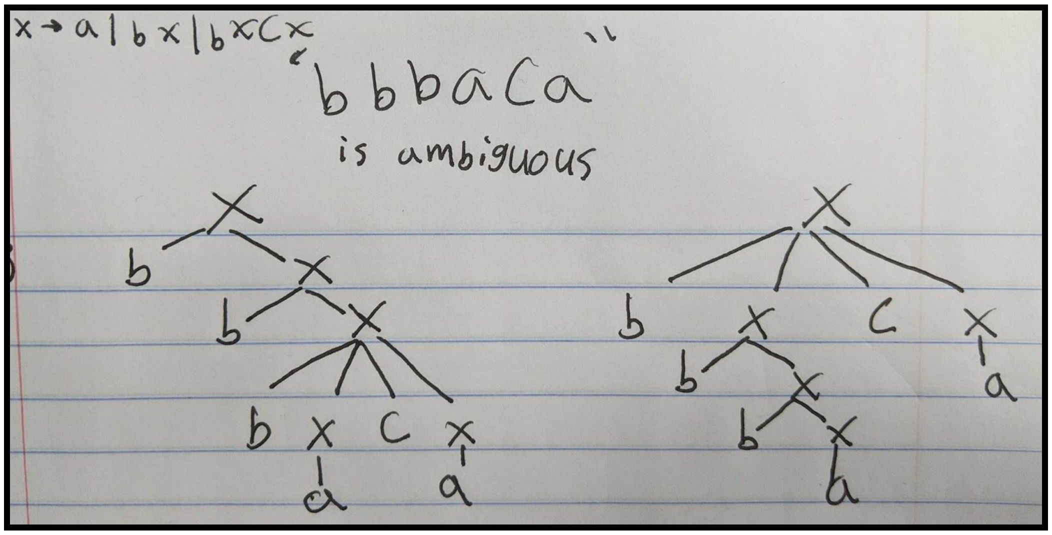


Exercise 3:

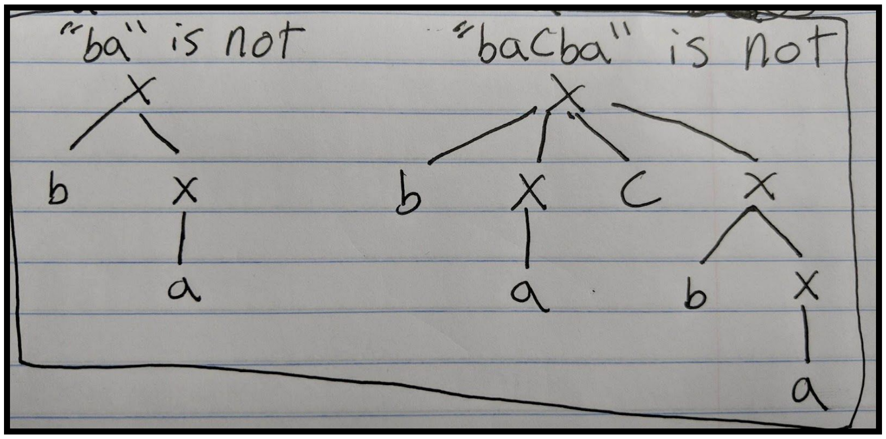
Show that the following grammar is ambiguous:

```
X -> a | bX | bXcX
where a,b,c are terminals.
```

In order for any grammar to be considered ambiguous, the grammar needs to allow any one string to be parsed into multiple parse trees. In the above grammar, there are multiple strings that could be parsed to yield multiple parse trees. For example, we see the string "bbbaca" and how, because of the ambiguity of the grammar, it can be parsed into these 2 completely different parse trees.



These parse trees are completely different, yet following them leads to the same string "bbbaca" outcome. This is not to say that some strings don't yield a singular parse tree. For example, the simple string "ba" or the more complex "bacba" string both have singular, unique parse trees. Ambiguity, however, exists in grammars no matter if 1 or 100 strings are permitted.



Any number of strings that would yield two or more parse trees makes any grammar ambiguous. Therefore, the above grammar is ambiguous.

Exercise 4:

a. Design a BNF grammar that recognizes expressions of the form Ai where A is in {a,b,c} and i is a digit.

```
Expression => A i
A => a | b | c
i => 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

b. Design a BNF grammar that recognizes lists of the form A1, A2, A3, …, An. **Use question a).**

```
List => Expression | List Separator Expression
Expression => A i
A => a| b | c
i => 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
Separator => ,
```

Exercise 5:

1. Write a JAY program that computes the sum of the *n* first numbers with a loop.

```
void main(){
     int n;
     int counter;
     int sum;
     sum = 0;

     if(counter>=m) {
          sum = sum + counter;
          counter = counter + 1;
     }
}
```

2. Write a JAY program that assigns the minimum of two numbers in a variable called min.

```
void main(){
     int one;
     int two;
     int min;
     if (one <= two )
          min == one;
     else
          min  ==  two;
}
```

3. Provide 2 examples of lexical errors in JAY.
    1. Trying to use a `for` loop will result in a lexical error in JAY. `for` in languages we know already, i.e. Java, is a keyword that can be used in reference to a token. In JAY, however, the only keywords defined are `boolean`, `else`, `if`, `int`, `main`, `void`, and `while`. There are keywords here that would start a loop, but you would use `if` or `while` and not `for`.

    2. You cannot use any numbers with decimal points in a JAY program. In order to represent a number with a decimal, we usually just put the "." sign in between the whole integer and the following decimal (i.e. 38.87, 64.42, etc.). In JAY, though, the "." sign is not designated anywhere. In other words, "." cannot be used anywhere in JAY, not just in regards to numbers with decimal places. Furthermore, the only variable types defined in JAY's *Literal* and *Keyword* tokens are boolean and integers. Integers are whole numbers only, further cementing the fact that decimal places and fractions have no place in JAY and will cause a lexical error.

4. Provide 2 examples of JAY programs with 2 different syntax errors.
    1.
```
void main(){
      int first;
      first = 42;
      int second;
}
```

This is wrong because JAY strictly requires *Declarations* first and then *Statements*. "i`nt first;`" and "`int second;`" are both examples of accepted *Declarations* but the fact that "`int second;`" comes after the "`first = 42;`" *Statement* is what makes this a syntax error. In order for this could to be functional, "`second`" must be declared before first is initialized.

```
2. void main(){
        int counter;
        int input;
        int sum;
        input= 10;
        sum = 0;
        while(counter < input){
             sum = sum + counter;
             counter++;

        }
   }
```

`counter++` is a shortcut provided in many modern programming languages, where a variable can be incremented by 1. Unfortunately, in the Jay syntax, this shortcut is not defined. It would have to be declared as "`counter= counter + 1;`".

5. Provide 2 examples of JAY programs with errors that are neither detected during the lexical analysis nor during the syntactic analysis.

1.
```
void main(){
     boolean sum;
     int counter;
     while(counter<10)
          sum = sum + counter;
}
```

Since an integer is being added to a boolean, the program will run but it will not logically make sense, therefore the program will fail.

2.
```
void main(){
     int number;
     int counter;
     number=1;
     counter = 0;
     while(number > counter)
          number = number + counter;
     counter = 0;
}
```

This program gets stuck in an infinite loop, as the loop lasts the duration of `number` is greater than 1, but number is always 1, and counter is always reinitiated to 0.