

Exercise A

“BE SURE TO DRINK YOUR OVALTINE”

Exercise B

:)

Exercise C

1. Although the use of `let` allows for more dynamic typing, Swift primarily uses static typing.
2. This means that type checking is done in the static context – before runtime. Errors about invalid typing are thrown before the program ever executes.

Exercise D

The grammar can be also represented with the regular expression $r o * z +$

The grammar contains the strings 1, 3, and 5.

Exercise E

This language contains all strings that start with n many pairs of a 's (aa), where n is greater than or equal to 0, followed by zero or more c 's, and ended by n many pairs of b 's (bb).

Exercise F

The regular expression $x * (ab \mid c) *$ contains the strings labeled

1, 3, 4, 5, 7.

Exercise G

This regular expression matches all strings that have any number of a 's, b 's, and c 's and also have one b , by matching the arbitrary set of $[abc]$ on either side of a b .

$(a \mid b \mid c) * b(a \mid b \mid c) *$

Exercise H

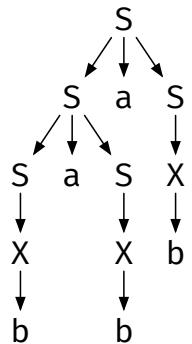
1. No, this grammar contains the rule $S \rightarrow SaS$ which is not one of the accepted forms of productions for a regular grammar.
2. Yes, this grammar is context-free as all of its production's left sides are non-terminals.

3. S
 SaS
 baS
 bab

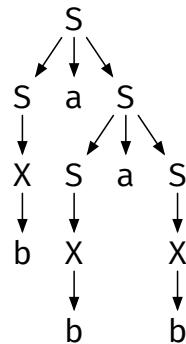
4. S
 SaS
 Sab
 bab

5. The string $babab$ has two different parse trees.

Parse tree one:



Parse tree two:

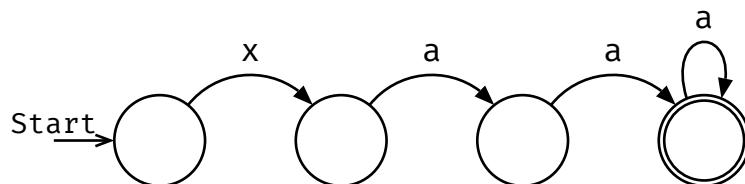


6. $S \rightarrow bA$
 $A \rightarrow aS \mid \epsilon$

Exercise E

1. $xa +$

2.



3. $S \rightarrow xaaA$
 $A \rightarrow a \mid \epsilon$

4. No, the grammar is not ambiguous as any derivation will always contain the first step, followed by n many additional of the second

steps for any number of a 's needed past two a 's. Since there are only two possible productions for A , and one of them is the empty string ϵ , there are no alternative ways to construct a string in the language from this grammar.

Exercise J

The provided statements are given (NOT IN BNF FORM!!!):

```
<uc-letter> = A-Z  
<lc-letter> = a-z
```

My solution is:

```
<name> = <real-name> <middle-name> <real-name>  
<real-name> = <uc-letter> <lc-letter> <lc-tail>  
    <lc-tail> = <lc-letter> <lc-tail> | ""  
<middle-name> = " " | " " <uc-letter> ". "
```

Note: " " is a single space character and "" is the empty string.