1. List the lexemes in the following snippet of C code (5):

```c
char array[ ] = "Hello";
```

2. How could you write a Flex regular expression for matching C string constants? (You may disregard the possibility of escaped quote characters in the string.) (5)

3. What is the difference between a statically typed language, and a dynamically typed language? What are some benefits and down-sides of each? (5)

4. What is the difference between top-down and bottom-up parsing, and what is the trade-off between the two? (5)
5. What is the difference between an expression and a statement? What are the effects of making a concept like “if” an expression (as in Racket) vs. making it a statement (as in C)? (5)

6. Racket, along with other Lisp languages, is homoiconic meaning that the code is written in the same structure (S-expressions) as is used for data. What are some benefits of this? (5)

7. Some languages include operator overloading, while others do not. What are the possible effects of this feature on a language’s readability, writeability and reliability? (5)
8. Some languages include pointers, while others do not. What are the possible effects of this feature on a language's readability, writeability and reliability? (5)

9. Write a racket expression that is equivalent to the following expression in a typical infix language such as C or Python (5):

\[(x + 12 * y) / 9 + b\]

10. What do the terms weakly typed and strongly typed refer to? (5)

11. Is the following grammar ambiguous? Justify your answer. (10)

```
<arrayliteral>: { <elements> }
<elements>: <elements> , <elements> | <element>
<element>: constant
```
12. Racket uses short-circuit evaluation for ‘and’ and ‘or’ expressions. What is short-circuit evaluation, and why are ‘and’ and ‘or’ implemented as macros in Racket? (5)

13. Consider a higher order function “zip” that takes a function f and two lists. The function zip evaluates to a new list consisting of f applied to each corresponding item in the two lists. For example:

> (zip + '(1 2 3 4) '(5 6 7 8))
'(6 8 10 12)

zip applies the function it's passed (in this case +) to each respective pair of the elements in the two lists. More examples:

> (zip max '(1 2 3 4 5) '(5 4 3 2 1))
'(5 4 3 4 5)

> (zip string-append '("a" "b" "c") '("d" "e" "f"))
'("ad" "be" "cf")

Write an implementation of zip in Racket. You can assume the two lists are the same length (10).
14. Given the following context free grammar (10):

<code>: <statement> <code>
    | <statement>

<statement>: <variable> = <expression>;
           | print <variable>;

<expression>: <expression> + <term>
           | <expression> - <term>
           | <term>

<term>: <term> * <variable>
       | <term> / <variable>
       | <variable>

<variable>: a | b | c | d

And the following code:

a = b * c + d;
b = e + e + a;
print a;

Draw the resulting parse tree.