1 Syntax and Semantics—25 points

1. (Robert Sebesta–5 points) Describe, using a single English sentence, the language defined by the following grammar:

\[
\begin{align*}
\mathit{S} & : = \mathit{A}\mathit{B}\mathit{C} \\
\mathit{A} & : = \text{a}\mathit{A} \mid \text{a} \\
\mathit{B} & : = \text{b}\mathit{B} \mid \text{b} \\
\mathit{C} & : = \text{c}\mathit{C} \mid \text{c}
\end{align*}
\]

\textbf{Answer:} strings consisting of one or more a followed by one or more b followed by one or more c.

2. (2 points) What does it mean for a context-free grammar to be \textit{ambiguous}?

\textbf{Answer:} The grammar generates a sentence with two (or more) parse trees.

3. (8 points) The grammar of the original definition of Algol 60 contained the following production rules:

\[
\begin{align*}
\text{<statement>} & \rightarrow \text{<conditional-statement>} \mid \text{begin <statement> end} \\
\text{<conditional-statement>} & \rightarrow \text{if <condition> then <statement>} \mid \text{if <condition> then <statement> else <statement>}
\end{align*}
\]

Show that any grammar containing these production rules is ambiguous.

\textbf{Answer:} See slide 41 in the slides on syntax for the answer for the Clite language, which has very similar production rules.

4. (6 points)

\[
\text{if 5 < 6 orelse (5 div 0) < 6 then 7 else 8;}
\]

(a) ML is a functional programming language. Is the above ML statement an expression or a command? (Choose one). \textbf{Answer:} an expression

(b) The above ML statement does not result in an exception. Explain why. \textbf{Answer:} The second argument of orelse is evaluated only if the first argument evaluates to false.

(c) What is the result of the ML statement above? \textbf{Answer:} 7

5. (4 points) Match:

(a) Command
(b) Declaration
(c) Expression

with

(a) Is evaluated to yield a value.
(b) Is executed to change the value of a variable or to change the input or output streams.
(c) Is elaborated to produce a binding, usually to allocate memory, and sometimes to initialize variables.

Answer: c-a, b-c, a-b.
2 FP–28 points

1. (2 points) Match the FP combining forms to their examples:

   (a) composition
   (b) construction
   (c) apply-to-all (map)

   (a) & %1
   (b) [id, %5]
   (c) tl @ [%1, id]

   Answer: 1-3, 2-2, 3-1

2. (1 point) What is !+:<1,2,3>?  
   Answer: 6

3. (1 point) Composition and construction (in FP) are examples of

   (a) primitive functions
   (b) control structures
   (c) combining forms

   (Choose one)

   Answer: combining forms (3)

4. (1 point) Combining forms are also called higher-order functions, because

   (a) they are closer to the way programmers think than normal functions
   (b) they take other functions as arguments
   (c) their domains and ranges have high dimension

   (Choose one)

   Answer: they take other functions as arguments (2)

5. (4 points) Write a function that multiplies its argument by seven. Call it timesseven. So, for example, timesseven:5 is 35.0. (The “.0” appears if you use Carter Bays’s FP interpreter.)  
   Answer: {timesseven * @ [id, %7]}

6. (4 points) Write a function that applies timesseven to all elements of a sequence and give an example of its application to a sequence of three numbers. Do not give a name to the function.  
   Answer: & timesseven

7. (2 points) What is !+: <1 2 3>? What do you call ! in FP?  
   Answer: 6; insert.
8. (7 points) Write a function that computes the length of a sequence. Do not use recursion. Do not use 
while. Use composition. (Hint: What is \& %1 : <1 2 3>) Answer: \!+ \& \& 1

9. (5 points) Call the function you wrote in the previous exercise length. (So, for example, length: <2 3 4> is 3.) Write a function that computes the average of a sequence of numbers. Call the function avg. For example, avg: <1 4 4> is 3.0. (The “.0” appears if you use Carter Bays’s FP interpreter.) Answer: / \@ \[!+, length\]
3 Haskell—68 points

1. (1 point) In Haskell, \([1,2,3]\) is an abbreviation for \(1 : (2 : (3 : [])\)). True or false?

   **Answer:** True.

2. (1 point) Here are signatures for two Haskell functions. Which one is curried?

   (a) \(\text{add\_a} :: \text{Int, Int} \to \text{Int}\)
   
   (b) \(\text{add\_b} :: \text{Int} \to \text{Int} \to \text{Int}\)

   **Answer:** The second one

3. (2 point) What is the domain of the type \((\text{[a]}, \text{[a]})\) in Haskell?

   **Answer:** Tuples of two lists of elements of the same type. (Note: a tuple of two is also called a pair.)

4. (4 points) A recursive function has two parts, the *basis* and the *inductive step*.

   (a) The basis computes the result for sufficiently small arguments, without making any recursive call.

   (b) The inductive step calls the function recursively, with smaller arguments.

   The following recursive function (which is intended to reverse a list) breaks one of these two rules. Which one? In which way?

   \[
   \text{reverse} :: \text{[a]} \to \text{[a]}
   \]

   \[
   \text{reverse}(L) = \begin{cases} 
   [] & \text{if } L = [] \\
   \text{reverse}(L) ++ [\text{head}(L)] & \text{else}
   \end{cases}
   \]

   **Answer:** The second (because the recursive call does not have a smaller argument)

5. (15 points total) Define functions \texttt{fact} of one argument that compute the factorial of a non-negative integer in four different ways:

   (a) (2 points) a non-recursive function using \texttt{product}. Do not use a loop.

   (Name this \texttt{fact1}.)

   (b) (5 points) a recursive function with a conditional expression. (Name this \texttt{fact2}.)

   (c) (4 points) guarded equations. (Name this \texttt{fact3}.)

   (d) (4 points) pattern matching. (Name this \texttt{fact4}.)

   **Answer**
--fact in four different ways

--using product

\[
\text{fact1} :: \text{Int} -> \text{Int} \\
\text{fact1} \ n = \text{product} \ [1..n] \\
\]

--recursive, with conditionals

\[
\text{fact2} :: \text{Int} -> \text{Int} \\
\text{fact2} \ n = \begin{cases} 
  1 & \text{if } n == 0 \\
  n \ast \text{fact2} \ (n - 1) & \text{otherwise}
\end{cases} \\
\]

--with guarded equations

\[
\text{fact3} :: \text{Int} -> \text{Int} \\
\text{fact3} \ n \ = \begin{cases} 
  1 & n == 0 \\
  n \ast \text{fact3} \ (n - 1) & \text{otherwise}
\end{cases} \\
\]

--with patterns

\[
\text{fact4} :: \text{Int} -> \text{Int} \\
\text{fact4} \ 0 = 1 \\
\text{fact4} \ n = n \ast \text{fact4} \ (n - 1) \\
\]

\newpage

\item
(10 points)
Define a function \texttt{rev1} of one argument that reverses a list. Use patterns.
\begin{verbatim}
\textbf{Answer:}
--reverse
\texttt{rev1} :: [a] -> [a] \\
\texttt{rev1} [] = [] \\
\texttt{rev1} (x:xs) = (\texttt{rev1} \ xs) ++ [x]
\end{verbatim}

6. (20 points total)

(a) (8 points) Define a function \texttt{count} of two arguments that counts the number of occurrences of the first argument in the second argument, which is a list of elements of the type of the first argument, using a list comprehension.

(b) (2 points) What is the type of this function? (Hint: a class constraint is needed because of equality testing.)

(c) (8 points) Define a function that counts the number of occurrences of the character 'a' in a string. Name your function \texttt{countA}. Here is an example of use:
Main> countA "CSCE330 is a great course!"
2 :: Int

countA must be defined as a partial application of count.

(d) (2 points) What is the type of countA?

Answer:

```
--count
count :: Eq a => a -> [a] -> Int
count x xs = length [x' | x' <- xs, x == x']

--countA
countA :: [Char] -> Int
countA as = count 'a' as
```

7. (10 points) The combinatorial function choices used in the countdown
problem gives all sublists of a list. For example, choices \([1,2,3]\) is the
list (not necessarily in the order given): \([[],[3],[2],[2,3],[3,2],[1],[1,3],[3,1],[1,2],[2,1],\]
\([1,2,3],[2,1,3],[2,3,1],[1,3,2],[3,1,2],[3,2,1]\]). Define choices using a
list comprehension and the functions subs, which computes a list contain-
ing all subsets of a given list (e.g., \(\text{subs}[1,2] = [\[],[1],[2],[1,2]]\)), and perms,
which computes a list containing all permutations of a given list (e.g.,
\(\text{perms}[1,2,3] = [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]\)). Assume
that subs and perms are given to you; you only need to define choices
using them.

Answer: choices xs = [zs | ys <- subs, zs <- perms ys]