1.

a) Explain why it is important for the designers/implementers of a programming language to define both a *concrete syntax* and an *abstract syntax* for the language. **(2)**

*The concrete syntax of a language is how human programmers interact with the language; it's important because it's usually very inconvenient and unintuitive to interact directly with abstract syntax.*

*The abstract syntax of a language is what the backend of the compiler operates on to perform static analysis and compilation/interpretation; it's important because it's usually very inconvenient to define these backend phases using concrete syntax.*

*It's important to give robust definitions of both concrete and abstract syntax because these definitions act as guidelines for developers working on implementations of a language, especially the implementation of syntax analysis.*

b) For each of the following language features, identify whether the feature falls under *static semantics* or *dynamic semantics* and give a brief explanation why. **(2 each)**

i) The C++ specification requires that all function calls in a program have the correct number of arguments in order for a program to be considered legal C++.

*Static semantics, because the property must be checked at compile time.*

ii) The Haskell specification defines an error mechanism that triggers when a program attempts to access an out-of-bounds index in a list at runtime.

*Dynamic semantics, because the error mechanism must act at runtime.*

iii) The Python specification defines an error mechanism that triggers when a program attempts to perform a function call with an incorrect number of arguments at runtime.

*Dynamic semantics, because the error mechanism must act at runtime.*

2.

Imagine you're given a buggy compiler ("86toARM") that translates from x86 bytecode to ARM bytecode. (ARM is another processor architecture, with a different instruction set than x86).

a) What does it mean to claim that 86toARM is *incorrect*? **(2)**

*There is at least one valid input program for which 86toARM fails to produce valid output or produces output whose semantics differ from the semantics of the input in an x86 interpreter (e.g. an x86 processor).*

b) What kind of evidence could you construct to prove this claim? **(1)**

*An input program that has valid semantics in an x86 interpreter, for which 86toARM fails to produce valid output or produces output whose semantics differ from the semantics of the input in the interpreter.*

Now imagine you're given another compiler ("HSto86") that translates from Haskell to x86 bytecode, and you construct a compiler ("HStoARM") that translates from Haskell to ARM bytecode by feeding the output of HSto86 into 86toARM.

c) Assume HSto86 has been proven *correct*, and 86toARM is still incorrect. What can be said about the correctness of HStoARM? Justify your answer. **(3)**

*HStoARM is probably incorrect: the output of HSto86 is guaranteed to be valid, but this output is fed into 86toARM, which may produce invalid output from this valid input. It's possible that HStoARM is correct if the output of HSto86 is guaranteed to only use some subset of x86 instructions and 86toARM only produces invalid output for input that includes instructions outside that subset.*

3.

Consider the following Haskell code:

```
reverse :: [Int] -> [Int]
reverse [] = []
reverse (x:xs) = reverse xs ++ [x]
```

For each of the following modifications to the code (highlighted in gray), identify which stage of compilation would first report an error and give a brief justification why. **(2 each)**

```
a) reverse :: [Int] -> [Int]
   reverse [] = []
   reverse (x:xs) = reverse xs && [x]
```

*Static analysis: this is a type error, because `(&&)` has type `Bool → Bool → Bool` but `reverse xs` and `[x]` both have type `[Int]`, not `Bool`. The program passes source input because all the characters are valid, passes lexical analysis because all the tokens are valid, and passes parsing because there is a valid AST.*

```
b) reverse :: [Int] -> [Int]
   reverse [] == []
   reverse (x:xs) = reverse xs ++ [x]
```

*Parsing: the modified line is a syntactically valid expression, but an expression is not valid in the middle of a function definition. The program passes source input because all the characters are valid and passes lexical analysis because all the tokens are valid.*

```
c) reverse :: [Int] -> [Int]
   reverse [] = []
   reverse (x:xs) = reverse 'xs' ++ [x]
```

*Lexical analysis: single quotes are used for single character literals, but `'xs'` is not a single character literal or any other kind of token. The program passes source input because all the characters are valid.*

```
d) reverse :: [Int] -> [Int]
   reverse [] = []
   reverse (x:xs) = reverse xs ++ [y]
```

*Static analysis: this is a scope error, because there is no binding for `y` at the location of the reference to it. The program passes source input because all the characters are valid, passes lexical analysis because all the tokens are valid, and passes parsing because there is a valid AST.*

4.

For each of the following regular expressions, give a short English description of the language specified by the regex and identify whether each of the given strings is a member of that language. **(4 each)**

```
a) a* • b* • c*
```

*Zero or more occurrences of `a`, followed by zero or more occurrences of `b`, followed by zero or more occurrences of `c`.*

|  |  |  |
|---|---|---|
| i) | "aabbcc" | **yes** \| no |
| ii) | "acba" | yes \| **no** |
| iii) | "" | **yes** \| no |

```
b) (a • b+)* • (c | d)+
```

*Zero or more occurrences of (one occurrence of `a` followed by one or more occurrences of `b`), followed by one or more occurrences of `c` or `d`.*

|  |  |  |
|---|---|---|
| i) | "abbabc" | **yes** \| no |
| ii) | "abcddc" | **yes** \| no |
| iii) | "aacdcd" | yes \| **no** |

5.

Give regular expressions for the following languages. **(2 each)**

a) Non-negative *even* integer literals with any number of leading zeros (e.g. "0", "4", "0038", "123456")

```
[0-9]* • [0|2|4|6|8]
```

b) Boolean literals separated by & (e.g. "true", "false & true", "true & true & false")

```
(true | false) • (& (true | false))*
```

6.

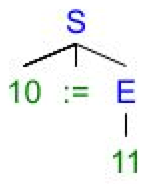
Consider the following context-free grammar for a tiny imperative language L, where `S` is the start symbol, `c` is any lower-case letter, `n` is any non-negative integer, and the other terminals are `:=`, `print`, `;`, and `+`.

```
S → c := E
S → print E
S → S ; S
E → c
E → n
E → E + E
```

For each of the following strings, identify whether the string is in L; if it is in L give a parse tree for it, and if it isn't in L give an argument that there can't be a parse tree for it. **(2 each)**

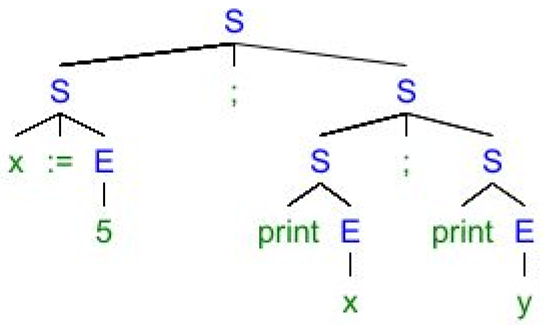
a) `"10 := 11"`

*Not in L: the parse tree would have to have the following form, but there's no rule* *`S → n := E`.*



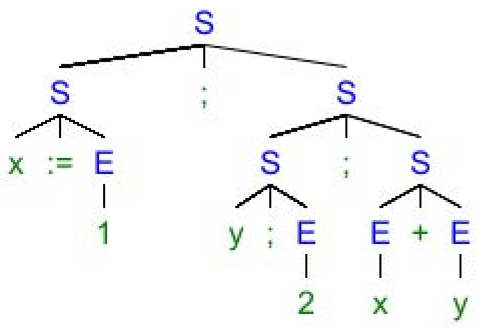
b) `"x := 5 ; print x ; print y"`

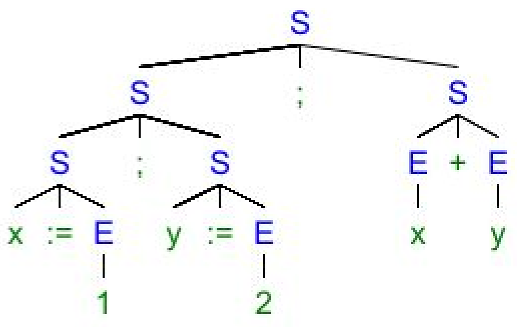
*In L: the following is one valid parse tree.*



c) `"x := 1 ; y := 2 ; x + y"`

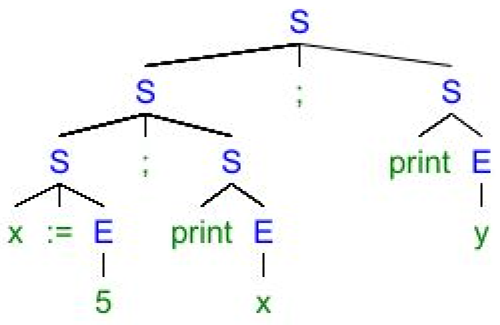
*Not in L: the parse tree would have to have one of the following forms, but there's no rule* $S \rightarrow E + E$.





d) Is this grammar for L ambiguous? Justify your answer. **(3)**

*Yes: the following parse tree is also valid for the string in b), so there is at least one string with two different parse trees.*



6.

For each of the following scenarios, argue in a couple sentences whether a class-based object-oriented programming language or a functional programming language would be more appropriate for writing the program in question.

Focus on the paradigms, not on specific languages/implementations - for example, don't say that an OOP language is more appropriate for a task because GCC compiles C++ to very efficient code and the task requires high performance, or that an FP language is more appropriate because there are high-quality Haskell libraries that are relevant to the task.

(Hint: recall the discussion in week 5 about the expression problem, and the relative strengths and weaknesses of OOP and FP when extending a program with new code.)

These are subjective questions with no strictly right or wrong answers - your responses will be graded on the strength of your justifications.

**(2 each)**

a) A video game company is building a racing game with a variety of cars to choose from. Each car has the same set of functionality (such as accelerating and steering), but unique values for several parameters (such as weight and horsepower). The team plans to add more cars to the game over time.

*Object-oriented programming is a natural fit because there will be new kinds of data added to the program over time, but not new kinds of functionality. There could be a class for the general notion of cars with a subclass for each individual car, or possibly a more structured hierarchy to categorize cars by common parameters (like manual vs. automatic transmission).*

b) A team is implementing a compiler for FORTRAN 77, an old language with a published formal specification, in order to enable legacy code to run more efficiently on modern platforms. The code will deal mostly with a single AST type, which will never change: the team does not plan to support newer versions of FORTRAN. However, they do plan to extend the codebase in the future to support additional features such as automatic code formatting and static analysis to detect memory leaks.

*Functional programming is a natural fit because there will be new kinds of functionality added to the program over time, but not new kinds of data. The AST type can be modeled with an algebraic data type, like the definition of the Prop AST type we've been using in lectures and homeworks, and features like formatting and static analysis can be implemented as functions defined by pattern-matching on this AST type.*

c) A web development startup is contracted to build a web store for a retail corporation that sells a wide variety of products. The application will initially only allow users to search for products by name and make purchases, but the retail corporation has indicated that they may want to expand the capabilities of the website in several different ways if the initial launch is a success. For a couple examples, they're interested in allowing customers to track the availability of selected products on a wishlist; categorizing the products into a structured hierarchy that can be browsed visually; and offering a rewards system that tracks the purchases each customer makes and gives them discounts.

*This is the expression problem in a nutshell: either paradigm may run into some challenges, since the full set of modifications that may be made to the program over time over time is not known in advance. One natural implementation might be with a relational database, which is a concept outside the scope of this course but can be used in either OOP or FP languages with different tradeoffs in either paradigm.*