CPS506 – Comparative Programming Languages – Current Questions

– All current questions as of March 2, 2019

Purpose The purpose of this document is to assist students in preparing for tests and exams.

It includes all questions previously used in the course. Many of the questions on this year's evaluations will come from this list (possibly with some modifications).

How to use The goal of the evaluations is not memorization, but rather understanding. If you have attended lectures, done the labs and assignments, and are prepared to think, rather than regurgitate, you will very likely do well on the evaluation.

The questions below are provided so that you may verify your understanding of the various concepts and languages, and where necessary refresh/fill-in your nowledge. Answers are not provided, because this should not be an exercise in memorization and regurgitation. As you review, you have a computer available to you, so all the answers are available to you.

For each question, you should evaluate what you believe to be a correct answer, and then use the computer to verify. The questions fall into 3 categories:

1. Programming questions. For this I encourage you to first program a solution on paper - emulating the test environment - and then enter and debug it on the computer to verify it.

2. Tracing questions. Work through the code associated with several questions and write down your answers to the questions. Only then, evaluate the provided code to verify your answers. Simply evaluating the code and memorizing the answers will not help you nearly as much!

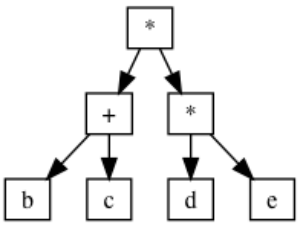
3. Theory questions. Answer the questions, then review your notes or the provided slides to verify the answer.

If a question says "Not applicable" it means we did not cover the material in sufficient detail to make the question reasonable to include on the evaluation. See the allQuestions file which includes everything.

If you find a question ambiguous, please let the professor know, so it can be made clearer.

---

ast-mc



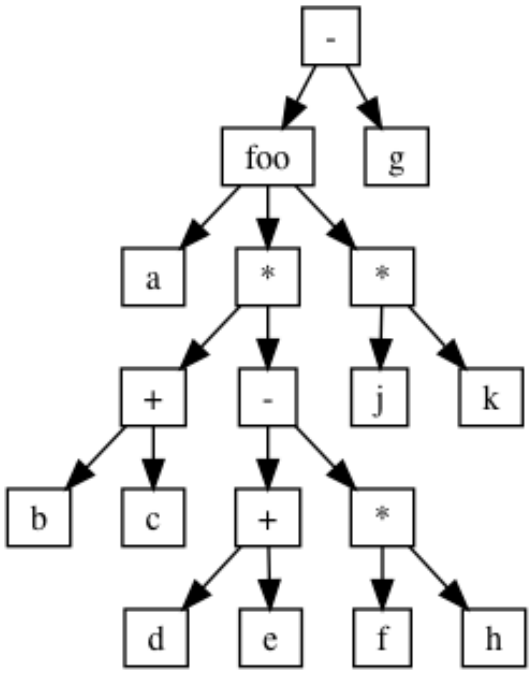
The abstract syntax tree for an expression is shown to the right. Select the correct concrete syntax for this tree in each of the following expression syntaxes, using no more parentheses than required.

Q1. prefix
Q2. postfix
Q3. Java
Q4. APL
Q5. Smalltalk

---

ast

Q6. (4 marks) The abstract syntax tree for an expression is shown to the right. Show the concrete syntax for this tree in each of the following expression syntaxes, using no more parentheses than required.

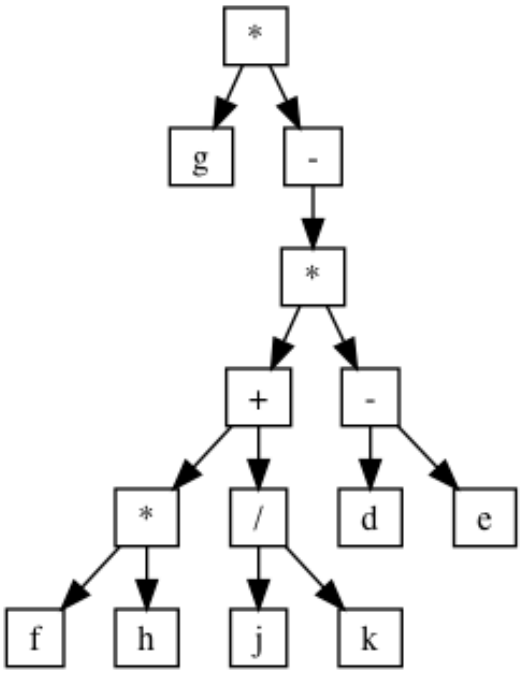


Prefix: Postfix: Infix, Java precedence rules: infix, left-to-right precedence:

---

ast2

Q7. (4 marks) The abstract syntax tree for an expression is shown to the right. Show the concrete syntax for this tree in each of the following expression syntaxes, using no more parentheses than required.

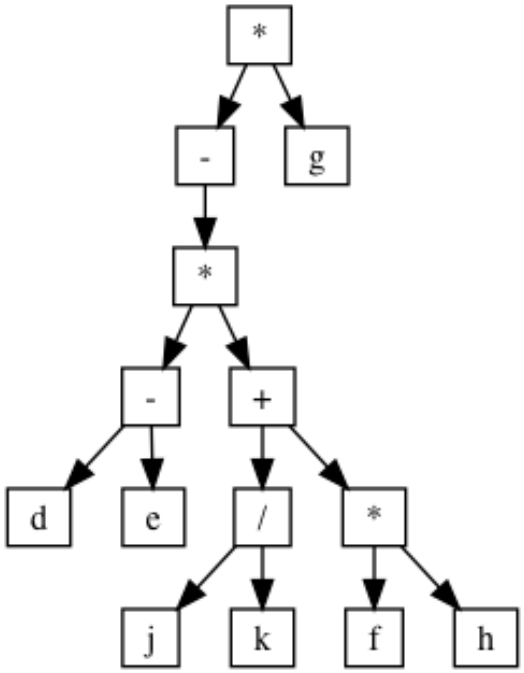


Prefix: Postfix: Infix, Java precedence rules: infix, left-to-right precedence:

---

ast3

Q8. (4 marks) The abstract syntax tree for an expression is shown to the right. Show the concrete syntax for this tree in each of the following expression syntaxes, using no more parentheses than required. Where required, use "negated" for unary negation.



Prefix: Postfix: Infix, Java precedence rules: infix, Smalltalk precedence:

---

ast4

Q9. (4 marks) For the following expressions, show the abstract syntax tree.

Prefix:

```
(/ (- (+ a b) (* c d (- e)) (+ f g)) h)
```

Postfix: (_ is unary minus)

```
a b + c d + * e f + _ * g h - *
```

Infix , Java precedence rules:

```
- (a * b + c * (d - (e * f + (g * h))))
```

Infix, left-to-right precedence rules:

```
- (a * b + c * (d - (e * f + (g * h))))
```

---

categories

Languages can be in several categories, including: OO (Object-Oriented), Imperative, Functional, Parallel, and Declarative. Langauges can also have statements or everything can be an expression. Languages can be statically type, dynamically typed, or untyped Finally, languages can have properties such as functions, methods, reflection, and closures.

In each of the following questions choose the single best answer. There may be more than one correct answer, and there may be answers that will give part marks. If the same question is asked twice, there are at least 2 correct answers and you must answer a different property for the two questions.

Q10. Smalltalk is
Q11. Smalltalk has
Q12. Smalltalk has
Q13. Elixir is
Q14. Elixir has
Q15. Elixir has
Q16. Haskell is
Q17. Haskell has
Q18. Haskell has
Q19. Rust is
Q20. Rust has
Q21. Rust has

---

elixir-moveLeft

Q22. (4 marks) Extend the following code, to define the function `canMoveLeft` which takes a Tetris piece and returns true if the piece could move left. The only consideration is whether the piece would fall off the left side of the board (the board has columns 1..n). Also define the function `moveLeft` that returns the same piece or the piece moved left one step if that's legal (using the `canMoveLeft` function.

A piece is a tuple composed of: the atom `piece`, a pair for the centre point, a list of pairs for the individual cells as offsets from the centre, and a colour.

```
defmodule Tetris do
  def canMoveLeft(
```

---

elixir-objects

Q23. (5 marks) In Elixir objects can be emulated by processes, where the object loops receiving messages and replying to them. For example:

```
    p1=Pawn.new(),
    Obj.call(p1,{:goto,1,2}),
    1=Obj.call(p1,:x),
    2=Obj.call(p1,:y),
    Obj.call(p1,{:moveDelta,3,1}),
    4=Obj.call(p1,:x),
    3=Obj.call(p1,:y).
```

Add the necessary code to the following to support the API used above for the object `pawn`:

```
defmodule Obj do
  def call(obj,msg) do
    send obj,{self(),msg}
```

```
    receive do
      Response -> Response
    end
  end
end
defmodule Pawn do
  def new(), do: spawn(__MODULE__,:init,[]).

  def init() do
```

---

elixir-output
In the following code,

```
defmodule Output do
  defp abc(x) do
    cond do
      x > 0 -> 1
      x < 0 -> -1
      true -> 0
    end
  end
  defp fgh(x,y), do: x * y
  defp n(x,x), do: []
  defp n(x,y), do: [y|n(x,y+1)]
  defp n(x), do: n(x,-x)
  def ghi() do
    for x<-1..4, y<-n(4), y<3, x>1, do: x+y
  end
  def hij() do
    ghi() |>
      Enum.take(7) |>
      Enum.map(fn x -> fgh(abc(x),x) end)
  end
end
[x1,x2,x3|x4]=Output.ghi()
[y1,y2,y3|y4]=Output.hij()
```

Q24. What is `x1`?
Q25. What is `x2`?
Q26. What is `length(x4)`?
Q27. What is `y1`?
Q28. What is `y2`?
Q29. What is `length(y4)`?
Q30. What language is this?

---

elixir-output2
In the following code,

```
defmodule Output2 do
  def abc(x) do
    y = spawn_link(__MODULE__,:n,[self()])
    send y,x
    receive do
        z -> z
    end
  end
  def n(z) do
    receive do
        v -> send z,n(v*v,v)
    end
  end

  defp n(x,x), do: [x]
  defp n(x,y), do: [y|n(x,y+y)]

end
[x1,y1|z1]=Output2.abc(2)
[x2,y2|z2]=Output2.abc(16)
```

Q31. What is `x1`?
Q32. What is `y1`?
Q33. What is `length(z1)`?
Q34. What is `x2`?
Q35. What is `y2`?
Q36. What is `length(z2)`?
Q37. What language is this?

---

elixir-server

Q38. (4 marks) Extend the following code, defining the loop function so that the process receives a pair of values and replies with the difference between them and the sum of them to the originating process.

```
defmodule ElixirServer do
  def new(), do: spawn(loop)
  def init() do
    x = new()
    send x,{self(),4,2}
    {2,6}=receive do x -> x end
    send x,{self(),5,5}
    {0,10}=receive do x -> x end
  end
  def loop() do
```

---

extensibility

Q39. (8 marks)

Explain the primary form(s) of extensibility for each of the following languages:

Smalltalk: Elixir: Haskell: Rust:

---

fold
Q40. (12 marks)

For each of the languages we studied, write a function called `fold` that takes a function `f`, and initial value `it` and a list `l` and returns the result that is produced by applying the function to the initial value and the first element of the list, then to that result and the second element of the list, and so on. For example, the code (in some made-up Javascript-like language):

```
function add(a1,a2) {return a1+a2*a2;}
fold(add,5,[2,3,4])
```

would produce the value: `34`. If the language doesn't have lists, use arrays.

Smalltalk: Elixir: Haskell: Rust:

---

haskell-func1
Q41. (3 marks) Write the simplest Haskell function that has the following type signature:

```
pairList :: (a -> b) -> (b -> c) -> [a] -> [(b,c)]
```

---

haskell-func2
Q42. (5 marks) Given the following definition, define the `instance` of `Eq` that has the natural semantics with the addition that `A` is equal to `C 0` and `D []`, and `B` is equal to `C 1`. Remember that equality is symmetric (i.e. $x == y \iff y == x$).

```
data Abc = A | B | C Integer | D [Abc]
```

---

haskell-monad
Q43. (1 marks) What is a monad?

---

haskell-monad2
Q44. (2 marks) Why doesn't a language like Java have monads?

---

haskell-output
In the following code,

```
let x = head (drop 3 [(x,y) | x <- [1..], y <- [3..4]])
let y0:z0 = drop 4 [(x,y) | x <- [1..], y <- [3..4]]
let y1:z1 = drop 4 [(x,y) | y <- [1..4], x <- [1..3]]
let (y2,_):z2 = (take 2 (drop 4 [(x,y) | x <- [1..3], y <- [1..4]]))
let (y3,_):z3 = (take 2 (drop 4 [(x,y) | x <- [1..3], y <- [1..4], x<y]))
```

Q45. What is `x`**?**
Q46. What is `y0`**?**
Q47. What is `length z0`**?**
Q48. What is `y1`**?**
Q49. What is `length z1`**?**
Q50. What is `y2`**?**
Q51. What is `z2`**?**
Q52. What is `y3`**?**
Q53. What is `z3`**?**
Q54. What language is this?

---

haskell-output2
In the following code,

```
let d x = x+x::Int
let f x = x : f (d x)
let x0:x1:x2 = f 3
let y0:_ = x2
let g x y = (x,y x) : g (y (y x)) y
let z0:z1:z2 = take 7 (g 3 (+2))
```

Q55. What is `x0`**?**
Q56. What is `x1`**?**
Q57. What is `length x2`**?**
Q58. What is `y0`**?**
Q59. What is `z0`**?**
Q60. What is `z1`**?**
Q61. What is `length z2`**?**
Q62. What language is this?
Q63. What is the type of `f`**?**
Q64. What is the type of `g`**?**

---

haskell-types
Q65. (5 marks) What is type inference, and why would you want it in a language? Use type inference to determine the type of:

```
abc f g w = f w g

ghi f g = map (\y -> y g) f

jkl f g w = w (map g (map (\(x,y) -> y) f))
```

---

haskell-typesmc
For each of the definitions below, what is the type of z?

Q66.
```
let z =
 \x -> x
```

Q67.
```
let z =
 \x -> x:[]
```

Q68.
```
let z x =
 [x]
```

Q69.
```
let z x y =
 x:y
```

Q70.
```
let z y x =
 x:y
```

Q71.
```
let z x y =
 [x y]
```

---

lang-mc
In each of the following questions choose the single best answer. There may be more than one correct answer, and there may be answers that will give part marks. If the same question is asked twice, there are at least 2 correct answers and you must answer a different language for the two questions.

Which language ...

Q72. allows unrestricted mutation?
Q73. can generate native code?
Q74. can generate native code?
Q75. doesn't have garbage collection?
Q76. doesn't have list comprehensions?
Q77. doesn't have list comprehensions?
Q78. dynamic dispatch for polymorphic ops?
Q79. generates code for polymorphic ops?
Q80. has control structures as statements?
Q81. has syntactic features to support DSLs?
Q82. has type classes?
Q83. has traits?
Q84. has traits?
Q85. is built on a process model?
Q86. is designed to support multiple cores?
Q87. is pure OO?
Q88. has pure functional functions?
Q89. is pure functional?
Q90. is statically typed?
Q91. is statically typed?
Q92. recognizes tail recursion?
Q93. recognizes tail recursion?
Q94. runs only on JVM/CLR?
Q95. simulates multiple-inheritance for code?
Q96. uses lazy evaluation?
Q97. uses monads for mutation?
Q98. uses prefix expression?
Q99. uses type inference?
Q100. uses type inference?

---

lang-preferred
Q101. (4 marks) Write a short essay (in the space provided) explaining which of the languages we studied – Smalltalk, Elixir, Haskell, and Rust – is your favourite and give comparisons with the others and with Java and C. There is no "right" answer; marks will be assigned for the quality of the reasoning and the understanding of the trade-offs.

---

map2
Q102. (12 marks)

For each of the languages we studied, write a function called `map2` that takes a function `f` and 2 lists `l1` and `l2` and returns the list that is produced by applying the function to one element from each of the lists in turn. For example, the code (in some made-up Javascript-like language):

```
function add(a1,a2) {return a1+a2;}
map2(add,[1,2,3],[4,5,6])
```

would produce the list: `[5,7,9]`. If the language doesn't have lists, use arrays.

Smalltalk: Elixir: Haskell: Rust:

---

multi-func1
Q103. (8 marks) Write the simplest function that has the following type signature (or equivalent) and works for all sizes of list:

```
pairList :: (a -> b) -> (b -> c) -> [a] -> [(b,c)]
```

In Smalltalk: In Elixir: In Haskell: In Rust:

---

multiprocessing
Q104. (6 marks) Not applicable

---

open-classes
Q105. (3 marks) Smalltalk is a "pure object-oriented" language. What is meant by this term? In pure OO languages, you often want to add methods to standard types, including `Object`. Why?

---

parallelism
Q106. (4 marks) One of the dominant issues of the next few years will be the end of faster processors and the rise of multiple cores. Expain the problem this presents at the processor/language level. Describe various approaches programming languages might adopt to address the problem.

---

rank-extensibility
Q107. (4 marks) Rank the languages we studied, with respect to extensibility.

Smalltalk **Elixir** **Haskell** **Rust**

Justify that ranking (note that "It's more like what I'm used to" is not a justification).

---

rank-orthogonality
Q108. (4 marks) Rank the languages we studied, with respect to orthogonality.

Smalltalk **Elixir** **Haskell** **Rust**

Justify that ranking (note that "It's more like what I'm used to" is not a justification).

---

rank-simplicity
Q109. (4 marks) Rank the languages we studied, with respect to simplicity.

Smalltalk **Elixir** **Haskell** **Rust**

Justify that ranking (note that "It's more like what I'm used to" is not a justification).

---

rust-dice
Q110. (5 marks) Given the following struct definition from the assignment, define a minimal implementation with a function called `new` that takes 1 parameter - a `Vec` of unsigned dice rolls - and returns a new object, and a method called `next` that, each time it is called with no parameters, it returns a `usize` value for the next dice roll for the object, wrapping around as necessary. For example, if created with 1,2,3,2 it would produce the sequence 1,2,3,2,1,2,3,2,1,2,3,2,1,2,3,2,...

```
struct Dice {
    rolls : Vec<u8>,
    roll : usize,
}
```

---

rust-output
In the following code,

```
let s = "Hello World!";
let mut r = String::from(s);
r.push_str(" from me");
let mut p = (&r).split_whitespace();
let q = p.next().expect("no keyword");
let o = p.collect::<Vec<_>>();
let mut x : Vec<_> = (3..6).collect();
// x.push(-5);
let z = x.remove(1);
let s = z+x.len();
let w : Vec<_>= x.iter().map(|x| (*x,s)).collect();
let (p,q)=w[0];
```

what do the following expressions evaluate to if printed or what type are they? (It may help you to know that the commented line would cause an error).

Q111. type of `s`
Q112. type of `x`
Q113. type of `q`
Q114. `w.len()`
Q115. `o.len()`
Q116. `x`
Q117. What language is this?

---

rust-safety
Q118. (5 marks) Compare and contrast Rust and C, explaining C issues that are addressed by Rust. Explain how Rust handles moving, borrowing, and copying to fulfil its safety promise.

---

semantic
Q119. (6 marks) Explain the difference between syntax, semantics, and pragmatics in programming languages. Is it possible to have 2 languages with different syntax, but the same semantics? If not, explain why not; if so, explain and give an example of two such languages. Is it possible to have 2 languages with different semantics, but the same syntax? If not, explain why not; if so, explain and give an example of two such languages.

---

smalltalk-func2
Q120. (10 marks) Define the Smalltalk classes to mimic the Haskell declaration:

```
data Abc = A | B | C Integer | D [Abc]
```

and define the necessary = functions that have the natural semantics with the addition that A is equal to C(0) and D(List()), and B is equal to C(1). Remember that equality is symmetric (i.e. x==y y==x). Here is a start on the code:

```
Object subclass: #Abc
    instanceVariableNames: ''
    classVariableNames: ''
  isZero
    ↑ false
  isOne
    ↑ false
  isEmpty
    ↑ false
  contains: otherValue
    ↑ false

Abc subclass: #A
    instanceVariableNames: ''
    classVariableNames: ''
  isZero
    ↑ true
  = other
    (other isKindOf: Abc) ifFalse: [↑ false].
    other isZero ifTrue: [↑ true].
    ↑ other isEmpty
```

---

smalltalk-output
In the following code,

```
Object subclass: #Abc
    instanceVariableNames: 'abc def ghi'
    classVariableNames: ''
initialize
    self abc: 39.
    self def: 17.
abc
    ↑ abc
abc: x
    abc := x
def
    ↑ self abc + 3
def: x
    def := x

Abc class
    instanceVariableNames: 'operator'
abc
    ↑ self new abc + 6
def
    | temp |
    temp := OrderedCollection new.
    self new def timesRepeat: [ temp add: temp size - 10].
    ↑ temp
ghi: a
    | i |
    i := self new.
    ↑ {i abc. i def} collect: a
```

what do the following expressions evaluate to if printed?

Q121. `Abc new`
Q122. `Abc new abc`
Q123. `Abc abc`
Q124. `Abc def size`
Q125. `Abc def last:2`
Q126. `Abc ghi: [:x| x+3]`
Q127. What language is this?

---

smalltalk-pieces
Q128. (8 marks) Write the code for the method `height` which returns the height of the piece (from the pivot point to the top of the piece, including the pivot row), and the method `rotate:` which is passed the number of 90-degree clockwise turns to make. Assume the points are represented as Point values relative to the pivot point (such as {`-1@0. 0@0. 1@0. 2@0`} for the horizontal bar. Also assume that points have methods `x` and `y` that return the x and y values, a method `@` which constructs points, and a method `rotatedClockwise` that will return the new point with the coordinates of the original point rotated 90-degrees clockwise.

```
Object subclass: #Piece
    instanceVariableNames: 'squares'
```

---

smalltalk-write

Q129. (5 marks) In a cell-based game, such as LightsOut, MineSweeper, or the GameOfLife, we need to know how many neighbour cells are occupied. Write the code for the method `countNeighbours` which uses `neighboursOfCell` (which you also must write), to provide the answer, using the following code as a start:

```
Object subclass: #Cell
    instanceVariableNames: 'neighbours below above left right occupied'
add: aCell
    aCell ifNotNil: [ neighbours add: aCell ]
above
    ↑ above
below
    ↑ below
left
    ↑ left
right
    ↑ right
occupied
    ↑ occupied " true if occupied, false otherwise "
```

---

type-inference

Q130. (5 marks) What is type inference? What is the advantage of type inference? What is the type signature of the following function?

```
abc f x = f (1 + x)
```

Show reasonable inference steps. What is the type signature of the following function?

```
ghi f g x = f (g (f x))
```

Show reasonable inference steps. What is the type signature of the following function?

```
jkl x f y = f (map (+y) x)
```

Show reasonable inference steps.

---

typing

Q131. (5 marks) Explain the difference between static and dynamic type sysems in languages. Provide and explain a context where dynamic types are better, and explain why. Provide and explain a context where static types are better, and explain why.

---

typing2
For each of these languages that you've studied at Ryerson, say whether it is untyped, has a dynamic type system, or has a weak, medium or strong static type system.
Q132. C
Q133. Smalltalk
Q134. Haskell
Q135. Java
Q136. Elixir
Q137. Rust

---

typing3
Q138. (3 marks) Explain the difference between static and dynamic type systems in languages.

---

typing4
Q139. (4 marks) Explain why Haskell and Rust have such a complicated type system compared to C, Java or Elixir, with particular reference to type-inference and type classes/traits.
