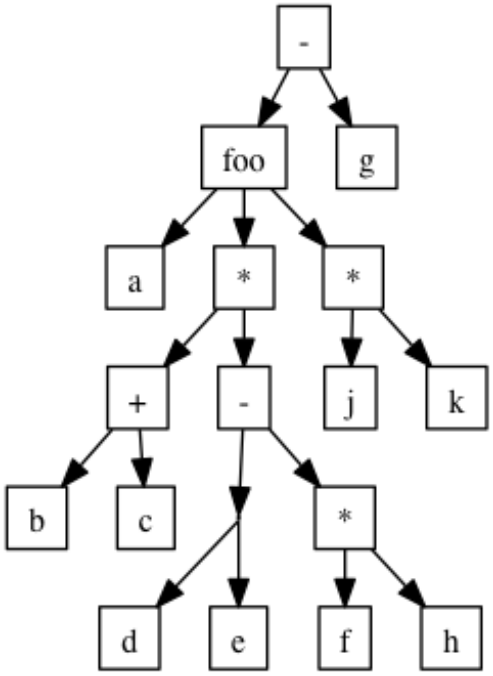
CPS506 – Comparative Programming Languages – All Questions

– All questions as of February 26, 2016

ast

Q1. (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.

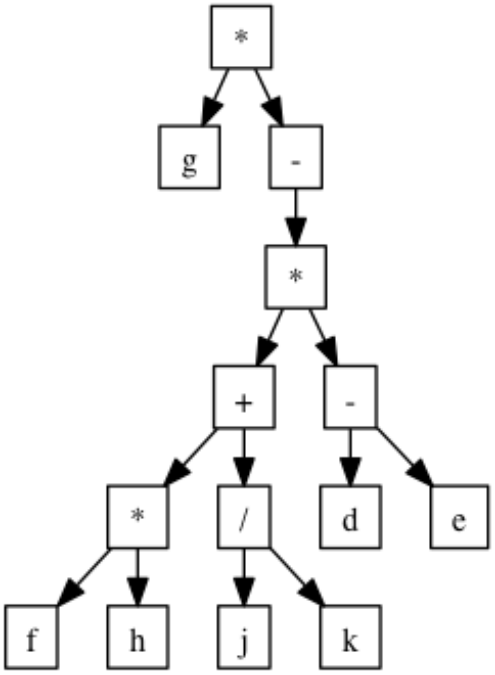


Prefix: Postfix: Infix, Java precedence rules: infix, left-to-right precedence:

---

ast2

Q2. (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.

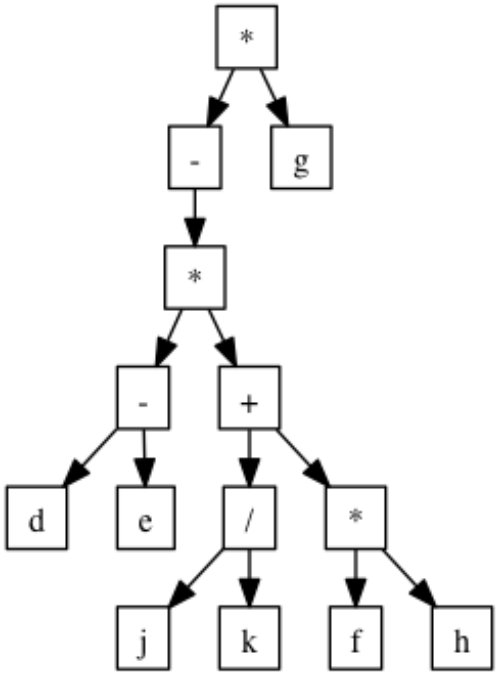


Prefix: Postfix: Infix, Java precedence rules: infix, left-to-right precedence:

---

ast3

Q3. (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:

---

ast4

Q4. (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))))
```

---

clojure-output
In the following code for the visitor pattern,

```
;;; handles orange and green baskets
;;;    (:orange n-kumquats weight)
;;;    (:green basket basket)
;;; e.g. (:green (:orange 15 701.126) (:green (:orange 5 10) (:orange 7)))
;;;              - 5 nodes, 27 kumquats, 711.126 kilos
;;;       (:green (:orange 10))
;;;              - 2 nodes. 10 kumquats, 0 kilos
(defn visit [ofield gfield]
   (fn v [[tag l r :as basket]]
      (case tag
        :green  (+ (or (gfield basket)) (v l) (v r))
        :orange (or (ofield basket) 0)
        0)))

(def zero (fn [_] 0))
(def one (fn [_] 1))
(def f1 (fn [[_ x]] x))
(def f2 (fn [[x]] x))
(def f3 (fn [[_ _ x]] x))
(def node-count   (visit n1 n2)
(def kums-count   (visit k1 k2))
(def weight-count (visit w1 w2))
```

the `ofield` and `gfield` parameters must be definted appropriately for the particular visitor. You must figure them out for `node-count`, `kums-count` and `weight-count`.
Q5. What is `n1`
Q6. What is `n2`
Q7. What is `k1`
Q8. What is `k2`
Q9. What is `w1`
Q10. What is `w2`
Q11. What language?

---

clojure-output2
In the following code,

```
(defn f [g h]
   (fn v [[tag l & rest]]
      (case tag
        :g (g l (h (v rest)))
        :h (g (h l) (v rest))
        1)))
(def f0 (f + +))
(def f1 (f + -))
(def x0 (f0 [1 2 3 4]))
(def x1a (f0 [:g 1 :h 2 :h 3 :g 4]))
(def x1b (f0 [:g 1 :g 2 :h 3 :g 4]))
(def x2a (f1 [:g 1 :h 2 :h 3 :g 4]))
```

```
(def x2b (f1 [:g 1 :g 2 :h 3 :h 4]))
(def x3 ((f * +) [:g 1 :h 2 :h 3 :g 4]))
```

What will be the following resulting values?
Q12. What is `x0`?
Q13. What is `x1a`?
Q14. What is `x1b`?
Q15. What is `x2a`?
Q16. What is `x2b`?
Q17. What is `x3`?
Q18. What language is this?

---

combinator2
In the Scala expression `"S" ~> "'" ~> """[a-zA-Z ]+""".r <~ "'"`, what are:
Q19. things with quotes?
Q20. things with tildes?
Q21. the whole expression?

---

combinators
Q22. (2 marks) In the Scala expression `"S" ~> "'" ~> """[a-zA-Z ]+""".r <~ "'"`, all the things with double-quotes are parsers. What do the things with tildes (~, ~>, <~ and similar) do? What is the resulting expression?

---

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

```
Call=fun pawn:call/2,
P1=pawn:new(),
Call(P1,{goto,1,2}),
1=Call(P1,x),
2=Call(P1,y),
Call(P1,{moveDelta,3,1}),
4=Call(P1,x),
3=Call(P1,y).
```

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

```
-module(pawn).
-export([new/0,init/0,call/2,test/0]).
new() -> spawn(pawn,init,[]).
call(Obj,Msg) -> Obj ! {self(),Msg},
                 receive
                     Response -> Response
                 end.
init() ->
```

---

erlang-output
In the following code,

```
-module(output).
-export([hij/0,ghi/0,n/1,abc/1]).
abc(X) -> if
          X > 0 -> 1;
          X < 0 -> -1;
           true -> 0
      end.
fgh(X,Y) -> X * Y.
n(X,X) -> [];
n(X,Y) -> [Y|n(X,Y+1)].
n(N) -> n(N,-N).
ghi() -> [X+Y||X<-[1,2,3,4],Y<-n(4),Y<3,X>1].
hij() -> lists:map(fun(X) -> fgh(abc(X),X) end,
             [X+Y||X<-[1,2,3,4],Y<-n(4),Y<3,X>1]).
[X1,X2,X3,X4|_] = output:ghi().
[Y1,Y2,Y3,Y4|_] = output:hij().
```

Q24. What is `X1`**?**
Q25. What is `X2`**?**
Q26. What is `X3`**?**
Q27. What is `X4`**?**
Q28. What is `Y1`**?**
Q29. What is `Y2`**?**
Q30. What is `Y3`**?**
Q31. What is `Y4`**?**
Q32. What language is this?

---

erlang-output2
In the following code,

```
-module(output2).
-export([n/1,abc/1]).

abc(X) ->
        Y = spawn_link(output2,n,[self()]),
        Y ! X,
        receive
           Z -> Z
         end.
n(X,X) -> [X];
n(X,Y) -> [Y|n(X,Y+Y)].
n(Z) -> receive N -> Z ! n(N*N,N) end.

[X1,Y1|Z1]=output2:abc(2).
[X2,Y2|Z2]=output2:abc(8).
```

Q33. What is `X1`?
Q34. What is `Y1`?
Q35. What is `length(Z1)`?
Q36. What is `X2`?
Q37. What is `Y2`?
Q38. What is `length(Z2)`?
Q39. What language is this?

---

extensibility
Q40. (8 marks)
Explain the primary means to extend each of the following languages:

Ruby: Haskell: Clojure: Erlang:

---

fold
Q41. (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.

Ruby: Haskell: Clojure: Erlang:

---

haskell-func1
Q42. (3 marks) Write the function that has the following type signature:

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

---

haskell-func2
Q43. (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
Q44. (1 marks) What is a monad?

---

haskell-monad2
Q45. (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]))
```

Q46. What is `x`**?**
Q47. What is `y0`**?**
Q48. What is `length z0`**?**
Q49. What is `y1`**?**
Q50. What is `length z1`**?**
Q51. What is `y2`**?**
Q52. What is `z2`**?**
Q53. What is `y3`**?**
Q54. What is `z3`**?**
Q55. 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))
```

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

---

haskell-types
Q65. (3 marks) What is type inference, and why would you want it in a language?

---

haskell-types

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. is pure functional?
Q74. is pure functional?
Q75. is pure OO?
Q76. is pure OO?
Q77. has mixins?
Q78. runs only on JVM/CLR?
Q79. runs only on JVM/CLR?
Q80. has syntax based on Prolog?
Q81. uses type inference?
Q82. uses type inference?
Q83. is used in major web servers?
Q84. is designed to support multiple cores?
Q85. uses monads for mutation?
Q86. can generate native code?
Q87. simulates multiple-inheritance for code?
Q88. simulates multiple-inheritance for code?
Q89. has syntactic features to support DSLs?
Q90. has syntactic features to support DSLs?
Q91. has traits?
Q92. uses lazy evaluation?
Q93. supports futures?
Q94. is built on a process model?
Q95. recognizes tail recursion?
Q96. recognizes tail recursion?
Q97. is statically typed?
Q98. is statically typed?
Q99. has control structures as statements?
Q100. doesn't have list comprehensions?
Q101. doesn't have list comprehensions?
Q102. uses prefix expression?

---

lang-mc2
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 ...

Q103. allows unrestricted mutation?
Q104. is pure functional?
Q105. is pure functional?
Q106. is pure OO?
Q107. has mixins?
Q108. runs only on JVM/CLR?
Q109. has syntax based on Prolog?
Q110. uses type inference?
Q111. is designed to support multiple cores?
Q112. uses monads for mutation?
Q113. can generate native code?
Q114. simulates multiple-inheritance for code?
Q115. has syntactic features to support DSLs?
Q116. uses lazy evaluation?
Q117. supports futures?
Q118. is built on a process model?
Q119. recognizes tail recursion?
Q120. recognizes tail recursion?
Q121. is statically typed?
Q122. has control structures as statements?
Q123. doesn't have list comprehensions?
Q124. uses prefix expression?

---

lang-mc3
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 ...

Q125. allows unrestricted mutation?
Q126. is pure functional?
Q127. is pure OO?
Q128. runs only on JVM/CLR?
Q129. runs only on JVM/CLR?
Q130. has syntax based on Prolog?
Q131. uses type inference?
Q132. uses type inference?
Q133. uses monads for mutation?
Q134. can generate native code?
Q135. simulates multiple-inheritance for code?
Q136. has traits?
Q137. uses lazy evaluation?
Q138. supports futures?
Q139. is built on a process model?
Q140. recognizes tail recursion?
Q141. recognizes tail recursion?
Q142. is statically typed?
Q143. is statically typed?
Q144. has control structures as statements?
Q145. doesn't have list comprehensions?
Q146. uses prefix syntax?

---

lang-preferred
Q147. (6 marks) Write a short essay (in the space provided) explaining which of the languages we studied – Haskell, Clojure, Erlang, or Scala – 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
Q148. (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.

Ruby: Haskell: Clojure: Erlang:

---

multi-func1
Q149. (8 marks) Write the simplest function that has the following type signature (or equivalent):

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

In Haskell: In Clojure: In Erlang: In Scala:

---

open-classes
Q150. (4 marks) Smalltalk, Ruby, and Scala are all "pure object-oriented" languages. What is meant by this term? Give an example other than those languages. In pure OO languages, you often want to add methods to standard types, including `Object`. Why? In Smalltalk, you can use the browser and go to any class and add methods. How do you get equivalent capability in Ruby and Scala?

---

open-classes2
Q151. (3 marks) Smalltalk and Ruby are "pure object-oriented" languages. What is meant by this term? In pure OO languages, you often want to add methods to standard types, including `Object`. Why? In Smalltalk, you can use the browser and go to any class and add methods. How do you get equivalent capability in Ruby?

---

parallelism
Q152. (10 marks) One of the dominant issues of the next few years will be the end of faster processors and the rise of multiple cores. Each of the languages we studied has its own approach to effective use of multiprocessing. Explain the approach taken by each of the following languages:

Ruby: Java: Haskell: Clojure: Erlang:

---

rank-extensibility
Q153. (4 marks) Rank the languages we studied, with respect to extensibility.

Ruby Scala Haskell Clojure Erlang

Justify that ranking.

---

rank-orthogonality
Q154. (4 marks) Rank the languages we studied, with respect to orthogonality.

Ruby Scala Haskell Clojure Erlang

Justify that ranking.

---

rank-simplicity
Q155. (4 marks) Rank the languages we studied, with respect to simplicity.

Ruby Scala Haskell Clojure Erlang

Justify that ranking.

---

ruby-output
For the following code:

```
def abc(x)
  y = 0
  t = 0
  while t < x
    y = y + yield(t)
    y = yield(y+1)
    t = t + 1
  end
  return y
end
```

What do the following expressions produce?
Q156. `abc 1 do |x| x+1 end`
Q157. `abc 2 do |x| x+1 end`
Q158. `abc 1 do |x| x+2 end`
Q159. `abc 2 do |x| x+2 end`
Q160. `abc 1 do |x| x*2 end`
Q161. `abc 2 do |x| x*2 end`
Q162. What language is this?

---

ruby-yield
Q163. (5 marks) In ruby there is a powerful expression called `yield`. What does it do? Show a possible implementation of `Array>>each` that uses `yield`.

---

scala-func2
Q164. (5 marks) Define the Scala case classes to mimic the Haskell declaration:

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

and define the necessary `equals` 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. $x == y \iff y == x$). Here is a start on the code:

```
object Classes {
   abstract class Abc
```

---

scala-output
Q165. (10 marks) What does the following code do?

```
import scala.language.postfixOps
import scala.util.parsing.combinator._

// try: java visitor 'O(10,12.5)' 'G(O(15,701.125)G(O(5,10),O(7)))' 'G(O(10))

object visitor {
  sealed abstract class Basket
```

```
case class GreenBasket(b1:Basket = null, b2:Basket = null)     extends Bas
case class OrangeBasket(nKumquats: Int = 0,kilos:Double = 0.0) extends Bas

class BasketParsers extends RegexParsers {
  lazy val integer: Parser[Int] = """\d+""".r ^^ { _.toInt }
  lazy val floatNum: Parser[Double] = """\d+(\.\d*)?""".r ^^ { _.toDouble }
  lazy val green: Parser[Basket] = 'G' ~> '(' ~> (basket ~ (((',')?) ~> bask
    y => y match {
      case Some(x) => x._2 match {
        case Some(z) => new GreenBasket(x._1,z)
        case None => new GreenBasket(x._1)
      }
      case None => new GreenBasket()
    }}
  lazy val orange: Parser[Basket] = 'O' ~> '(' ~> (integer ~ (((',')? ) ~> f
    y => y match {
      case Some(x) => x._2 match {
        case Some(z) => new OrangeBasket(x._1,z)
        case None => new OrangeBasket(x._1)
      }
      case None => new GreenBasket()
    }}
  lazy val basket: Parser[Basket] = orange | green
  def parse(text : String):ParseResult[Basket] = {
    parseAll(basket, text)
  }
}

def nodeCount(b:Basket):Int = b match {
  case GreenBasket(b1,b2) => nodeCount(b1)+nodeCount(b2)+1
  case OrangeBasket(_,_) => 1
}
def kumquatCount(b:Basket):Int = b match {
  case GreenBasket(b1,b2) => kumquatCount(b1)+kumquatCount(b2)
  case OrangeBasket(n,_) => n
}
def weight(b:Basket):Double = b match {
  case GreenBasket(b1,b2) => weight(b1)+weight(b2)
  case OrangeBasket(_,w) => w
}

def main(args: Array[String]) {
  args.foreach { arg =>
    println("For: "+arg);
    val parse = (new BasketParsers).parse(arg)
    if (parse successful) {
      val basket:Basket = parse.get
      println("   Node count     is "+nodeCount(basket))
      println("   Kumquat count  is "+kumquatCount(basket))
      println("   Weight         is "+weight(basket))
```

```
      }
      else {
        println("    Can't parse")
      }
    }
  }
}
val i = 3
```

Q166. What language is this?

---

scala-output2
In the following code,

```
def g2[a,b](f:(a,b)=>Boolean,pairs:List[(a,b)],init:List[(a,b)]):List[(a,b)]
  pairs match {
    case ((x,y)::rest) =>
      if (f(x,y)) pairs.head::g2(f,rest,init) else g2(f,rest,init)
    case _ => init
  }
def g[a,b](fs:List[(a,b)=>Boolean],pairs:List[(a,b)],init:List[(a,b)]=List()):
                                        List[(a,b)] =
  fs match {
    case List() => init
    case (f::fs) => g2(f,pairs,g(fs,pairs,init))
  }
val nums=List((1,2),(2,2),(3,2),(3,4),(2,3))
val less=(_:Int) < (_:Int)
val lesseq=(_:Int) <= (_:Int)
val grt=(_:Int) > (_:Int)
val q::r::s::_=g(List(less,grt),nums)
val w::x::y::z=g(List(grt,lesseq,less),nums)
```

Q167. What is `q`?
Q168. What is `r`?
Q169. What is `s`?
Q170. What is `w`?
Q171. What is `x`?
Q172. What is `y`?
Q173. What is length `z`?
Q174. What language is this?

---

semantic
Q175. (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, why not; if so, give an example of two such languages. Is it possible to have 2 languages with different semantics, but the same syntax? If not, why not; if so, give an example of two such languages.

---

type-inference
Q176. (10 marks) What is type inference? What is the advantage of type inference? What is the type signature of the following function?

```
let abc f g x = f (g (f x))
```

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

```
let ghi x f y = f (map (+y) x)
```

Show reasonable inference steps.

---

typing
Q177. (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.
Q178. C
Q179. Smalltalk
Q180. Prolog
Q181. Haskell
Q182. Java
Q183. Erlang

---

typing3
Q184. (3 marks) Explain the difference between static and dynamic type systems in languages.

---

typing4
Q185. (4 marks) Explain why Haskell and Scala have such a complicated type system compared to C, Java or Erlang, with particular reference to type-inference and type classes.
