CMPT 383 Comparative Programming Languages
Quiz 2

This quiz has 15 questions in total
- 10 questions worth 1 point
- 5 questions worth 2 points

Quiz time: 40 minutes

Student name: ______________________________

1. (1 pt) Which of the following about lambda calculus is NOT correct? C
   A. Lambda calculus is a Turing-complete programming language
   B. There are exactly three kinds of lambda terms: variables, function abstractions, and function applications
   C. Every lambda term has a normal form
   D. A redex refers to a lambda term that can be beta-reduced

2. (1 pt) Which of the following conversion in lambda calculus is NOT correct? B
   A. Eta conversion: λx. (λy.λz. z y) x → λy.λz. z y
   B. Eta conversion: λx. (λy. x y z) x → λy. x y z
   C. Alpha conversion: λx. (λy.λz. z y) x → λx. (λw.λz. z w) x
   D. Alpha conversion: λx. (λy. x y z) x → λw. (λy. w y z) w

3. (2 pts) Which of the following is NOT a string in the language defined by the following grammar? D
   ```
   S ::= S S | 'a' S 'b' | 'a' 'b'
   ```
   where S is the start symbol
   A. aababb
   B. ababab
   C. aabbab
   D. abbaab
   E. abaabb

4. (1 pt) Which of the following is NOT correct? A
   A. To evaluate the FUN expression "app (lambda x. 1 == x) (3 + 6)", we first evaluate "3+6" before function application under the call-by-name semantics
   B. The evaluation environment (environment in operational semantics) maps identifiers to values
   C. The evaluation of a program gets stuck if no inference rule in the operational semantics can apply
   D. For the multiplication operator *, small-step operational semantics must specify which operand to evaluate first

5. (1 pt) Which of the following is NOT correct? D
    A. To prove a programming language is type safe, we need to prove progress and preservation of its type system with respect to its operational semantics
    B. The typing environment maps identifiers to their types
    C. Int can be viewed as an abstract value for -1
    D. Int → Int can be viewed as an abstract value for 100

6. (2 pts) Given the following definitions
   W = 𝜆x.𝜆y. x y y
   K = 𝜆x.𝜆y. x
   What is the normal form of the lambda term **W (W K)**? If it is difficult to type the lambda symbol 𝜆, you can use backslash \ to represent 𝜆.

   𝜆y. y y
   (y can be alpha-renamed, e.g., 𝜆x. x x)

7. (2 pts) Consider the grammar
   `E ::= E '-' E | E '*' E | c`
   where E is the start symbol, and c is an integer constant. Rewrite the grammar to enforce that the * operator has higher precedence than the - operator.

   E ::= E '-' E | T
   T ::= T '*' T | c
   (Symbols E and T can be renamed)

8. (1 pt) Consider the grammar
   `E ::= E '-' E | c`
   where E is the start symbol, and c is an integer constant. Rewrite the grammar to enforce that the - operator is left-associative.

   E ::= E '-' c | c
   (Symbol E can be renamed)

9. (1 pt) What is the evaluation result of the following FUN expression?
   `let f = (lambda x. lambda y. x + y) in (app (app f 10) 20)`

   30

10. (2 pts) Consider a FUN expression
    `let x = 2 in (let x = 3 in x)`
    (1pt) Can we evaluate this expression based on the operational semantics of FUN?
    (1pt) If yes, what is the evaluation result? If not, what is the reason?

    Yes (1pt), the result is 3 (1pt)

11. (1 pt) Consider the following FUN expression with type annotations
    `let f: Int -> Bool -> Int = (lambda x: Int. lambda y: Bool. 10) in (app f 100)`
    Does this expression type check? If yes, what is the type of the expression?

    Yes, the type is Bool → Int

12. (1 pt) Consider the following FUN expression without type annotations
    `let f = (lambda x. lambda y. 10) in (app f 100)`
    What is the most general type of this expression inferred by the Hindley-Milner type inference?

    X1 → Int
    (X1 can be renamed to an arbitrary type variable)

13. (1 pt) Given two type variables X1, X2, what is the result of applying substitution **[X1 ↦ X2 → X2, X2 ↦ Int]** to type **X1 → X2**? If it is difficult to type, you can use -> to represent →.

    (X2 → X2) → Int
    (X2 cannot be renamed. The parentheses are required.)

14. (1 pt) Given type variables X1, …, X5, What is the result of the substitution composition
    [X2 ↦ X1 → X1, X3 ↦ Int] o [X4 ↦ X3 → X2, X5 ↦ X3]
    If it is difficult to type, you can use colon : to represent the mapsto symbol ↦ and use -> to represent →.

    [X4 ↦ Int → X1 → X1, X5 ↦ Int, X2 ↦ X1→X1, X3 ↦ Int]
    (The order of entries in the map does not matter)

15. (2 pts) Given type variables X1, …, X5, find a most general unifier for type constraints
    { X1 → X2 = X2 → X3 → Int, X1 = X4 → X5 }
    If it is difficult to type, you can use colon : to represent the mapsto symbol ↦ and use -> to represent →.

    [X1 ↦ X3 → Int, X2 ↦ X3 → Int, X4 ↦ X3, X5 ↦ Int]

    There are other correct most general unifiers like
    [X1 ↦ X4 → Int, X2 ↦ X4 → Int, X3 ↦ X4, X5 ↦ Int]