Programming Languages - Spring 2016

Exam 2

Code: 97267

1. Which of the following statements are true with respect to the lambda calculus?

   (i) integers are a feature
   (ii) variadic functions are a feature
   (iii) identifiers are a feature

   (A) *ii*
   (B) *i, ii*
   (C) *i*
   (D) none are true
   (E) *iii*
   (F) all are true
   (G) *ii, iii*

2. With this implementation for *cons*:

   ```
   (define (cons a b) (lambda (f) ((f a) b)))
   ```

   which of the following would be a valid function body for *car*, assuming the formal parameter is named *c*?

   (A) `(c (lambda (x) a))`
   (B) `(c (lambda (y x) x))`
   (C) `(c (lambda (x y) x))`
   (D) `(c (lambda (y) b))`
   (E) `(c (lambda (y) (lambda (x) y)))`
   (F) `(c (lambda (x) (lambda (y) y)))`
   (G) `(c (lambda (y) (lambda (x) b)))`
   (H) `(c (lambda (x) (lambda (y) a)))`

3. Given the following incrementer and base:

   ```
   (define (inc x) (list x x))
   (define base 0)
   ```

   and the evaluation of the Church number *three*:

   ```
   (define x ((three inc) base))
   ```

   What is the value of x?

   (A) (((0 0) (0 0) (0 0)))
   (B) 3
   (C) (((0 0) (0 0)) ((0 0) (0 0)))
   (D) (3 3)
   (E) (((0 0))))
   (F) ((0 0) (0 0) (0 0))

4. Given this definition of *add*:

   ```
   (define (add a)
       (lambda (b)
           (lambda (f)
              (lambda (x)
                  ((a f) ((b f) x))
                  )
               )
           )
       )
   ```

   Which of the following functions for multiplying two Church numerals are correct?

   ```
   (define (mul a b)
      (lambda (f)
         (lambda (x)
            ((((a (add b)) zero) f) x)
            )
         )
      )
   ```

   ```
   (define (mul a b)
      (lambda (f)
         (lambda (x)
            ((((a add) b) f) x)
            )
         )
      )
   ```

   (A) the one on the left
   (B) the one on the right
   (C) both
   (D) neither

5. How many cons cells make up the structure with the print value of `(1 (2 3 4))`?

   (A) 4
   (B) 5
   (C) 6
   (D) 3

6. What is a better print value that more truly represents the structure that could be represented as `(1 2 . 3 4)`?

   (A) (1 2 3 4)
   (B) (1 2 3 . 4)
   (C) (1 2 . 3 . 4)
   (D) (1 2 (3 4))

7. How many cons cells are needed, at a minimum, to represent a doubly-linked list node that holds *two* pieces of information. Consider both pieces of information and pointers to other nodes.

   (A) 7
   (B) 5
   (C) 8
   (D) 4
   (E) 6
   (F) 3

8. Consider a minimal cons cell structure for representing a doubly-linked list node that holds *two* pieces of information. How many cons cells are there in a list composed of these nodes if there have been three insertions into an empty list *and* the list is circular. Assume the list is of minimal size (e.g. no dummy nodes).

   (A) 7
   (B) 10
   (C) 9
   (D) 12
   (E) 11
   (F) 8

9. Suppose you wish to flatten a list m that contains either atoms or lists of atoms (an atom is anything that is not a cons cell).

   ```
   (flatten '(a b (c d) e)) ; should evaluate to (a b c d e)
   ```

   Which of the following expressions accomplishes that task? Choose the most simple answer. These helper functions might be useful:

   ```
   (define h1 (lambda (x) (if (pair? x) append cons)))
   (define h2 (lambda (a b) ((if (pair? a) append cons) a b)))
   ```

   (A) `(map h2 m)`
   (B) `(accumulate h1 nil m)`
   (C) `(accumulate h2 nil m)`
   (D) `(map h1 m)`
   (E) `(accumulate append nil m)`
   (F) `(map append m)`

10. Suppose you have a filtering function named *keep* that collects elements for which a given predicate *p?* is true. Which of the following expressions removes those elements instead? Choose the most simple answer.

    (A) `(keep (lambda (x) ((not p?) x)) items)`
    (B) `(not (keep p? items))`
    (C) `(keep (not p?) items)`
    (D) `(keep (lambda (x) (not (p? x))) items)`

11. Suppose you wish to use *accumulate* to perform mapping a function f onto a list of items m. Which of the following expressions accomplishes that task?

    (A) `(accumulate (lambda (x y) (cons (f x) y)) nil m)`
    (B) `(accumulate (lambda (x) (cons (f x) m)) nil m)`
    (C) `(accumulate (lambda (x) (f x)) nil m)`
    (D) none of these expressions work
    (E) `(accumulate (lambda (x y) (cons x (f y))) nil m)`

12. Suppose you wish to use *accumulate* to count the number of items in a list m. Which of the following expresssions accomplishes that task?

    (A) `(accumulate (lambda (x) 1) 0 m)`
    (B) none of these expressions work
    (C) `(accumulate (lambda (x y) (+ 1 (cdr y))) 0 m)`
    (D) `(accumulate (lambda (x y) (+ 1 y)) 0 m)`
    (E) `(accumulate (lambda (x) (+ 1 (cdr m))) 0 m)`

13. Suppose you wish to use *map* to count the number of items in a list *m*. Which of the following expresssions accomplishes that task?

(A) `(map (lambda (x) (+ 1 (apply + (cdr m)))) m)`

(B) none of these expressions work

(C) `(map (lambda (x y) (+ 1 y)) m)`

(D) `(map (lambda (x y) (+ 1 (cdr y))) m)`

(E) `(map 1 m)`

(F) `(map (lambda (x) 1) m)`

14. Suppose you are tasked to count the number of cons cells in a given argument. What is the minimum number of cases that need to be processed, including any base cases? Assume the existence of the *pair?* predicate, which returns true if its given argument is a cons cell. Here are some examples:

```
(countCons 5)                    ; should evaluate to zero
(countCons (list 1 2 3))         ; should evaluate to 3
(countCons (list (cons 1 2) 3 4)) ; should evaluate to 4
```

(A) 5

(B) 3

(C) 2

(D) 4

15. What is the print value of the expression `'''a`?

(A) `(quote (quote a))`

(B) `(quote quote quote a)`

(C) none, it's an invalid expression

(D) `(quote (quote (quote a)))`

16. Consider implementing the *equal?* predicate, without the use of `and` or `or` in the base cases. How many base cases are needed at a minimum, if the predicate is to be used to compare numbers and lists of numbers (recursive)? Remember, the *eq?* predicate can be used for both pointer and numeric equality. Assume the existence of the *pair?* predicate, which returns true if its given argument is a cons cell. Here are some examples:

```
(equal? 3 5)               ; should evaluate to #f
(equal? (list 1 2) '(1 2)) ; should evaluate to #t
(equal? 3 '(1 2))          ; should evaluate to #f
```

(A) 3

(B) 4

(C) 2

(D) 5

(E) 1

(F) 6

17. Consider re-implementing *apply-generic* (from the text) for just binary operations on tagged data. If no error checking is to be performed, what would be a valid function body? Assume the formal parameters are *op*, *left*, and *right* and that functions installed in the operator table work on tagged operands. Choose the correct answer with the fewest number of printable characters.

(A)
```
(apply
    (get op (map type-tag (list left right)))
    (list left right))
```

(B)
```
(op
    left
    right)
```

(C)
```
(apply
    (get op (type-tag left) (type-tag right))
    (list left right))
```

(D)
```
((get op)
    left
    right)
```

(E)
```
((get op (type-tag left) (type-tag right))
    left
    right)
```

(F)
```
((get op (map type-tag (list left right)))
    left
    right)
```

18. Consider the following grammar rule, using the notation that all caps indicates terminals:

```
alpha : BETA gamma
      | delta
      | BETA EPSILON iota
```

The *alphaPending* function consists of how many calls to *check* and how many calls to pending functions, respectively?

(A) two and zero
(B) one and two
(C) one and one
(D) two and two
(E) zero and one
(F) one and zero
(G) two and one
(H) zero and two

19. Consider the following grammar rule, using the notation that all caps indicates terminals:

```
alpha : BETA gamma
      | delta
      | BETA EPSILON iota
```

A recognizing function for this grammar rule would minimally contain how many logical tests?

(A) 3
(B) 4
(C) 2
(D) 1

20. Consider the following grammar rule, using the notation that all caps indicates terminals:

```
alpha : beta GAMMA
      | delta KAPPA
      | beta EPSILON iota
      | delta beta
```

and the recognizing function that follows. What code should immediately follow a call to *delta*? Assume the style given in lecture.

(A) `if (betaPending()) { beta(); match(KAPPA); }`
(B) `beta(); check(KAPPA);`
(C) `if (check(beta)) beta(); else match(KAPPA);`
(D) `if (check(KAPPA)) match(KAPPA); else beta();`
(E) `check(KAPPA); beta();`
(F) `beta(); match(KAPPA);`
(G) `if (betaPending()) match(beta); else match(KAPPA);`
(H) `if (pending(KAPPA)) beta(); else match(KAPPA);`