CMPS-112 • Comparative Programming Languages • Fall 2015 • Test #2

$Id: cmps112-2015q4-exam2.mm,v 1.13 2015-11-12 15:02:50-08 - - $

page 1	page 2	page 3	Total / 32

Please print clearly :

Name :
Login : @ucsc.edu

No books ; No calculator ; No computer ; No email ; No internet ; No notes ; No phone. Neatness counts ! Do your scratch work elsewhere and enter only your final answer into the spaces provided.

1. Name two kinds of ***universal*** polymorphism, and give a ***brief*** example of each. Do not use more than a few lines of code. **[2✔]**

   (a)

   (b)

2. Name two kinds of ***ad hoc*** polymorphism, and give a ***brief*** example of each. Do not use more than a few lines of code. **[2✔]**

   (a)

   (b)

3. ***Scheme.*** Define the function `eval` for arbitrarily nested arithmetic expressions. Use `map` and `apply`. Assume the `car` of each list and sublist is a function, and any operand that is not a `number?` is a subexpression. **[2✔]**

```
> (map even '(1 2 3 4 5 6))
(#t #f #f #f #f #f)
> (apply + '(1 2))
3
> (eval `(,+ (,* 3 4) (,/ (,- 2 3) 4)))
47/4
```

4. ***Smalltalk.*** Define a block called `sum` which when sent the `value:` message with an array argument, returns the sum of the elements of the array. **[2✔]**

```
st> sum value: #(1 2 3 4 5).
15
```

5. ***Ocaml.*** Define the function `ip` (inner product) which is the sum of pairwise products of two lists. Use tail recursion. Raise an exception if the lists are of different lengths. The formula is given mathematically here. **[2✔]**

$$p = \sum_{i=0}^{n-1} u_i v_i$$

```
# ip;;
- : float list -> float list -> float = <fun>
# ip [1.;2.;3.] [4.;5.;6.];;
- : float = 32.
```

6. ***Ocaml.*** Define the function **`zip`** which takes two lists as arguments and returns a single list of pairwise tuples with the same data. Raise an exception of the lengths of the lists are different. **[2✓]**

```
# zip;;
- : 'a list -> 'b list -> ('a * 'b) list = <fun>
# zip [1;2;3] [4;5;5];;
- : (int * int) list = [(1, 4); (2, 5); (3, 5)]
```

7. ***Smalltalk.*** Define a class **`List`**. It has instance variables **`car`** and **`cdr`**, and functions of the same name which return those values. It has class methods **`new`** which returns **`nil`** and **`car:cdr:`** which create a new **`List`** containing the two operands in the appropriate fields. It has instance methods **`car:cdr:`** which update the **`car`** and **`cdr`** fields, and instance methods **`car`** and **`cdr`** which return them. **[4✓]**

```
st> a := List car:1 cdr: (List car:2 cdr: (List car:3 cdr: (List new))).
st> a car.
1
st> a cdr car.
2
st> a cdr cdr car.
3
st> a cdr cdr cdr.
nil
st> a car:6 cdr:8.
a List
st> a car. a cdr.
6
8
```

8. ***Smalltalk.*** Define classes **`Num`** and **`Mul`** that can be used as a basis for expression trees. **`Num`** has an instance value number, a class method **`new:`** which sets the number ; and instance methods **`set:`** which updates the number, and **`value`** which returns the number. **`Mul`** has two instance values which point at left and right subexpressions, a class method **`left:right:`** which creates a new instance with pointers to other **`Num`**s and **`Mul`**s, and instance methods **`left:right:`** which updates the left and right children of the expression tree, and **`value`** which evaluates the expression. **[4✓]**

```
st> a:= Num new: 6.
a Num
st> b:= Num new: 8.
a Num
st> a value.
6
st> b value.
8
st> c:= Mul left:a right:b.
a Mul
st> c value.
48
st> b set: 99.
a Num
st> c value.
594
```

Multiple choice. To the ***left*** of each question, write the letter that indicates your answer. Write ***Z*** if you don't want to risk a wrong answer. Wrong answers are worth negative points. **[12✔]**

number of correct answers		× 1 =	= a
number of wrong answers		× ½ =	= b
number of missing answers		× 0 =	0
column total $c = \max(a-b, 0)$	12		= c

1. Backus-Naur Form, used to define syntax, was first used in the definition of:
   (A) ALGOL
   (B) BASIC
   (C) COBOL
   (D) FORTRAN

2. What is the Smalltalk expression for $\sqrt{2}$?
   (A) `sqrt (2)`
   (B) `2 sqrt`
   (C) `Number::sqrt 2`
   (D) `2 ^ .5`

3. What is `((lambda (x) x) (+ 2 3))` ?
   (A) `(+ 2 3)`
   (B) `+`
   (C) `10`
   (D) `5`

4. In a garbage collected language like Java, with no `free` function or its equivalent, if M = memory leaks and D = dangling pointers or references, which is possible ?
   (A) D but not M
   (B) M but not D
   (C) both M and D
   (D) neither M nor D

5. What is the running time of `let rec f n = if n <= 1 then n else f (n - 1) + f (n - 2)` ?
   (A) $O(\log_2 n)$
   (B) $O(n)$
   (C) $O(2^n)$
   (D) $O(n^2)$

6. The Java idea of an interface is implemented in Smalltalk as:
   (A) abstract classes
   (B) duck typing
   (C) multiple inheritance of fields
   (D) single inheritance

7. For a list of length n, how much function call stack space is used by fold left and fold right ?
   (A) fold left $O(1)$ and fold right $O(1)$
   (B) fold left $O(1)$ and fold right $O(n)$
   (C) fold left $O(n)$ and fold right $O(1)$
   (D) fold left $O(n)$ and fold right $O(n)$

8. In Smalltalk what is the meaning of:
   `foo bar + foo set: 3 + 4 next`
   (A) `((foo bar) + foo) set: (3 + (4 next))`
   (B) `(foo (bar + foo)) set: ((3 + 4) next)`
   (C) `(foo bar) + (foo set: 3) + (4 next)`
   (D) `(foo bar) + (foo set: 3) + (4 next)`

9. What is 7 in Smalltalk ?
   (A) `(+) 3 4.`
   (B) `(3+4) value.`
   (C) `[3+4] value.`
   (D) `{3+4} value.`

10. Which function can be implemented using a constant amount of stack space ?
    (A) filter
    (B) fold_left
    (C) fold_right
    (D) map

11. The PL/1 language allows a non-local `goto` directly from a function to a label in a function deeper down in the function call stack, thus returning past several levels of function calls. In Java, something similar can be accomplished by what statement ?
    (A) `goto`
    (B) `implements`
    (C) `synchronized`
    (D) `throw`

12. If we define the block `sum := [:i :j| i + j]` in Smalltalk, how might we obtain the number 7 ?
    (A) `3 4 sum`
    (B) `3 sum: 4`
    (C) `sum 3 value 4 value`
    (D) `sum value: 3 value: 4`