CMPS-112 · Comparative Programming Languages · Fall 2015 · Test 11of 1

$Id: cmps112-2015q4-exam1.mm,v 1.52 2015-10-21 14:42:06-07 - - $

.PS

examboxes(3)

.PE

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.

.EQ

delim $$

.EN

1. What are the very general possibilities that a function might

exhibit when called? [2pt]

(a)

(b)

(c)

(d)

2. Define the function filter whose first argument is a predicate and

whose second argument is a list. It returns a list consisting of

all elements of the argument list which satisfy the predicate. Do

not use a higher-order function.

(a) Scheme. [2pt]

> (filter (lambda (x) (> x 0)) '(1 -1 2 -3 5 -99 8))

(1 2 5 8)

> (filter even '(1 2 3 4 5 6 7 8 9))

(2 4 6 8)

(b) Ocaml. [2pt]

# filter;;

- : ('a -> bool) -> 'a list -> 'a list = <fun>

# filter (fun x -> x > 0) [1;-1;2;-3;5;-99;8];;

- : int list = [1; 2; 5; 8]

# filter even [1;2;3;4;5;6;7;8;9];;

- : int list = [2; 4; 6; 8]

3. Define the function length, which returns the length of a list.

Use tail-recursion: the function must use $ O ( 1 ) $ stack. Do

not use a higher-order function.

(a) Scheme. [1pt]

> (length '(1 2 3 4 5))

5

(b) Ocaml. [1pt]

# length;;

- : 'a list -> int = <fun>

# length [1;2;3;4;5];;

- : int = 5

4. Code sub' according to the specifications of the project. Assume

that the number of larger magnitude is the first argument, and the

carry is the third argument. Assume sub has taken care of the

signs so that sub' does not need to do so. [2pt]

val sub' : int list -> int list -> int -> int list

5. Define the function fold\_left: the first argument is a function to

use to fold the list, the second argument is a unit value used to

fold the first element, the third argument is a list. Use tail

recursion: the function must use $ O ( 1 ) $ stack.

(a) Scheme. [2pt]

> (define (length list) (fold\_left (lambda (n \_) (+ n 1)) 0

list))

> (define (sum list) (fold\_left + 0 list))

> (length '(1 2 3 4 5))

5

> (sum '(1 2 3 4 5))

15

(b) Ocaml. [2pt]

# fold\_left;;

- : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a = <fun>

# let length list = fold\_left (fun n \_ -> n + 1) 0 list;;

val length : 'a list -> int = <fun>

# let sum list = fold\_left (+) 0 list;;

val sum : int list -> int = <fun>

# length [1;2;3;4;5];;

- : int = 5

# sum [1;2;3;4;5];;

- : int = 15

6. Define the function reverse which produces a list in reverse order

to that of its argument. The function must use $ O ( 1 ) $ stack.

You may use a tail-recursive function, or make it very simple by a

call to fold\_left.

Bonus points: These two ``reverse'' questions are worth 3 points

each, not 2 points, if you correctly define them in terms of fold\_

left instead of writing a recursive function.

(a) Scheme. [2pt] (3 points if you use fold\_left correctly.)

> (reverse '(1 2 3 4 5))

(5 4 3 2 1)

> (reverse '())

()

(b) Ocaml. [2pt] (3 points if you use fold\_left correctly.)

# reverse;;

- : 'a list -> 'a list = <fun>

# reverse [1;2;3;4;5];;

- : int list = [5; 4; 3; 2; 1]

# reverse [];;

- : 'a list = []

7. C or C++. Code the function in C or C++ to reverse a list. Do not

allocate or free any memory. Do not cause memory leak or use

uninitialized memory. Assume the nodes are properly initialized as

a valid linked list. Use $ O ( 1 ) $ stack space. [2pt]

|

typedef struct node node; | node\* reverse (node\* head) {

struct node { |

int value; |

node\* link; |

}; |

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. [12pt]

+--------------------------+------+------+------------+

|number of | |× 1 = | $= a$ |

|correct answers | | | |

+--------------------------+------+------+------------+

|number of | |× ½ = | $= b$ |

|wrong answers | | | |

+--------------------------+------+------+------------+

|number of | |× 0 = | 0 |

|missing answers | | | |

+--------------------------+------+------+------------+

|column total | 12 | | $= c$ |

|$ c = max ( a - b , 0 ) $ | | | |

+--------------------------+------+------+------------+

1. What kind of polymorphism is exhibited by generic classes in Java

and template classes in C++?

(A) conversion

(B) inheritance

(C) overloading

(D) parametric

2. What is ((lambda (f x) (f x)) + 3)?

(A) '(+ 3)

(B) '(f x)

(C) 3

(D) 6

3. What is type of (+) in Ocaml?

(A) int \* int \* int

(B) int \* int -> int

(C) int -> int \* int

(D) int -> int -> int

4. What is?

(car (cdr (cons '(1 2 3) '(4 5 6))))

(A) '(1 2 3)

(B) '(4 5 6)

(C) 1

(D) 4

5. In Ocaml, what is the type of [1;2;3;4]?

(A) (list int)

(B) int list

(C) list->int

(D) list<int>

6. The type system in Scheme are:

(A) strong and dynamic

(B) strong and static

(C) weak and dynamic

(D) weak and static

7. The type system in Ocaml are:

(A) strong and dynamic

(B) strong and static

(C) weak and dynamic

(D) weak and static

8. In C, C++, and Java, which operator is lazy?

(A) ++

(B) --

(C) //

(D) ||

9. What is 2?

(A) (caar '(1 2 3))

(B) (cadr '(1 2 3))

(C) (cdar '(1 2 3))

(D) (cddr '(1 2 3))

10. Lisp and Scheme, in general form, are based on a form of

mathematics first formulated by Alonzo Church.

(A) \lambda-calculus

(B) \mu-calculus

(C) \pi-calculus

(D) \psi-calculus

11. Which feature of imperative languages\* is missing from Scheme?

(A) conditionals

(B) functions

(C) loops

(D) variables

12. In 1968, Edsger W. Dijkstra published a paper entitled ``\_\_\_

statement considered harmful''.

(A) call

(B) goto

(C) switch

(D) throw

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*EWD498: How do we tell truths that might hurt? Prof. Dr. Edsger W.

Dijkstra, June 1975.

· FORTRAN, ``the infantile disorder'', by now nearly 20 years old, is

hopelessly inadequate for whatever computer application you have in

mind today: it is now too clumsy, too risky, and too expensive to

use.

· PL/I, ``the fatal disease'', belongs more to the problem set than

to the solution set.

· It is practically impossible to teach good programming to students

that have had a prior exposure to BASIC: as potential programmers

they are mentally mutilated beyond hope of regeneration.

· The use of COBOL cripples the mind; its teaching should,

therefore, be regarded as a criminal offence.