CMSC 15100 Midterm Exam 1

Ellyn Liu

TOTAL POINTS

88.5 / 100

QUESTION 1

1 expressions 8 / 8

✓ + 0.5 pts 1) -4

✓ + 0.5 pts 2) error, cannot add strings

✓ + 0.5 pts 3) #f

✓ + 0.5 pts 4) #t

✓ + 0.5 pts 5) 4/3

✓ + 0.5 pts 6) "b!"

✓ + 0.5 pts 7) error, cannot apply string-append to a symbol

✓ + 0.5 pts 8) gray circle in square frame

✓ + 0.5 pts 9) gray sqaure

✓ + 0.5 pts 10) squares on top of one another; "wedding cake"

✓ + 0.5 pts 11) error, cannot overlay a string on an image

✓ + 0.5 pts 12) (list #f #t #f #t)

✓ + 0.5 pts 13) (list 2 4)

✓ + 0.5 pts 14) error, cannot apply odd? to strings

✓ + 0.5 pts 15) error, cannot apply image-width to integers

✓ + 0.5 pts free half point!

QUESTION 2

2 types 13 / 14

✓ + 0 pts claimed

✓ + 2 pts starts-with? : Char String -> Boolean

- 0.5 pts starts-with? first type must be Char.

+ 2 pts average : (Listof Integer) -> (U Integer 'CannotCompute)

(Note: number types other than integer are OK.)

- 1 pts Result type of average must include possibility of 'Cannot Compute.

✓ + 2 pts sum-with: All (a) (a -> Integer) (Listof a) -> Integer

(Note: number types other than integer are OK.)

- 1 pts Function type with sum-with type must have result Integer (or some other number type).

- 1 pts sum-with must be able to work with non-integers in the input list.

✓ + 2 pts count-if: All (a) (a -> Boolean) (Listof a) -> Integer

✓ + 2 pts sum-if : (Integer -> Boolean) (Listof Integer) -> Integer

(note: Number types other than integer are also OK.)

✓ + 2 pts reject: All (a) (a -> Boolean) (Listof a) -> (Listof a)

✓ + 0 pts Element type of reject is overspecified to Integer.

✓ + 2 pts contains?: All (a) (a a -> Boolean) a (Listof a) -> Boolean

- 0.5 pts contains? has one too few argument types.

- 2 pts "All (...)" prefixes omitted from polymorphic types.

+ 1 Point adjustment

average is partially correct: 1.0/2.0

QUESTION 3

3 Date functions 24 / 24

+ 0 pts Claimed

✓ + 5 pts The implementation of format is correct, subject to any deductions enumerated below

- 1 pts Incorrect zero-padding

- 2 pts No zero padding

✓ + 3 pts The implementation of first-of is correct, subject to any deductions enumerated below

✓ + 5 pts The implementation of same-year? is

correct, subject to any deductions enumerated below

✓ **+ 6 pts** **The implementation of before? is correct, subject to any deductions enumerated below**

✓ **+ 5 pts** **The implementation of autumn? is correct, subject to any deductions enumerated below**

- 2 pts Conditions out of order resulting in true when should be false

+ 0 pts Response for format is essentially incorrect

+ 0 pts Response for first-of is essentially incorrect

+ 0 pts Response for same-year? is essentially incorrect

+ 0 pts Response for before? is essentially incorrect.

+ 0 pts Response for autumn? is essentially incorrect.

+ 0 pts No response for format

+ 0 pts No response for first-of

+ 0 pts No response for same-year?

+ 0 pts No response for before?

+ 0 pts No response for autumn?

QUESTION 4

4 tree functions 19 / 19

✓ **+ 0 pts** **Claimed**

✓ **+ 6 pts** **The implementation of add-to is correct, subject to any deductions enumerated below.**

- 1 pts In add-to, the recursive call leaves out the integer argument

- 2 pts Add-to does not build a tree.

✓ **+ 6 pts** **The implementation of change is correct, subject to any deductions enumerated below.**

- 1 pts In change, the recursive call leaves out the function argument.

- 2 pts Change does not build a tree.

✓ **+ 7 pts** **The implementation of replace is correct, subject to any deductions enumerated below.**

- 1 pts In replace, the recursive call does not have all 4 arguments

+ 0 pts Response for add-to is essentially incorrect

+ 0 pts Response for change is essentially incorrect

+ 0 pts Response for replace is essentially incorrect

+ 0 pts No response for add-to

+ 0 pts No response for change

+ 0 pts No response for replace

QUESTION 5

5 list functions 23 / 25

+ 0 pts claimed

✓ **+ 6 pts** **Remove is correct, subject to any enumerated deductions.**

- 1 pts In remove, use symbol=? instead of = or some other equality test.

- 1 pts In remove, you must not cons '() onto the result in any case. The output is a list of symbols.

- 1 pts In remove, the item is not removed if it is the sought symbol.

- 1 pts In remove, the item is not kept if it is not the sought symbol.

✓ **- 1 pts** **In remove, the recursive call leaves out the symbol argument.**

✓ **+ 6 pts** **Autumn is correct, subject to any enumerated deductions.**

✓ **+ 7 pts** **Clip is correct, subject to any enumerated deductions.**

✓ **- 1 pts** **In clip, recursive calls must have three arguments: both bounds and the list.**

- 2 pts In clip, out-of-bounds items are removed rather than clipped. Please look more closely at the examples.

✓ **+ 6 pts** **Majority? is correct, subject to any enumerated deductions.**

- 0.5 pts A majority is strictly greater than 50%

+ 0 pts There is no response for remove.

+ 0 pts There is no response for autumn.

+ 0 pts There is no response for clip.

+ 0 pts There is no response for majority.

+ 0 pts All responses are essentially incorrect

QUESTION 6

6 emulate the computer 1.5 / 10

✓ **+ 0 pts** **claimed**

✓ **+ 3 pts** **The type of f is**

Integer (Integer -> Image) (Integer -> Image) Integer -> Image

(Number types other than Integer are also OK.)

✓ **- 1.5 pts Types of i1 and i2 are not function types. They are both (Integer -> Image).**

- 0.5 pts Missing final output type -> Image

+ 0 pts Incorrect response

+ 7 pts Correct image: two squares, two circles, a dot.

✓ **+ 0 pts The expression *can* be evaluated.**

- 2 pts Too few or too many squares or circles.

- 2 pts Improper alternation between squares and circles.

+ 0 pts Incorrect response.

+ 0 pts No response.

Ellyn Liu

Midterm Exam 1
CMSC 15100 Autumn 2018
Monday, October 29, 2018

Please write your name here:

Please also write your initials in the upper right corner of each page.

We do not answer questions from students once the exam has begun. Please read the directions carefully and follow them as best you can. If you have trouble interpreting a question, you can write us a note about your interpretation of it on the test itself along with your response.

You may use the functions you write on this test anywhere on this test. Wherever you design your own helper function, write its purpose and type above its definition. You may not refer to functions you may have written at some earlier time (such as a homework exercise) without rewriting them here.

We will be scanning your exams and grading digital versions of them. Please do your best to write all responses in the given spaces. This will simplify our work considerably. Having said that, all your exams will be read by actual people and we can read outside the designated spaces if we must.

Some common built-in operations and their types are as follows:

```
symbol=? : Symbol Symbol -> Boolean
string=? : String String -> Boolean

cons     : All (A) A (Listof A) -> (Listof A)

first    : All (A) (Listof A) -> A
rest     : All (A) (Listof A) -> (Listof A)

empty?   : All (A) (Listof A) -> Boolean

length   : All (A) (Listof A) -> Integer
reverse  : All (A) (Listof A) -> (Listof A)
```

For each of the following expressions, either evaluate the expression or state why it is not possible to evaluate it. If the result is an image, sketch the image (images need not be exactly to scale):

```
(+ (* 1 2 3) (- 10))
```
→ −4 (+ 6 −10)

```
(+ "10" "11")
```
not possible, + expects numbers, not strings

```
(> (- 7 8) (- 8 7))
```
→ #f (−1 > 1)

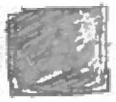
```
(>= (- 7 7) (- -7 -7))
```
→ #t (0 ≥ 0)

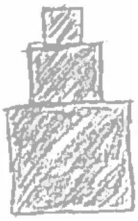
```
(+ 1/3 1)
```
→ $\frac{4}{3}$

```
(string-append (if (<= 1/3 1/4) "a" "b") "!")
```
→ "b!" (0.3 ≤ 0.25)

```
(string-append (if (>= 1/5 1/6) 'x 'y) "!")
```
→ not possible to append symbol to string

```
(overlay (circle 10 "solid" "gray") (square 30 "outline" "black"))
```
→ 

```
(overlay (square 30 "solid" "gray") (circle 8 "solid" "orange"))
```
→ 

```
(above (square 10 "solid" "gray")
       (square 20 "solid" "gray")
       (square 30 "solid" "gray"))
```
→ 

```
(overlay "???" (circle 80 "outline" "black"))
```
→ not possible, overlay expects an image, not a string

```
(map even? (list 1 2 3 4))
```
→ '(#f #t #f #t)

```
(filter even? (list 1 2 3 4))
```
→ '(2 4)

```
(filter odd? (map number->string (list 1 2 3 4)))
```
→ not possible, map number->string will return a list of strings and odd? will cause an error on strings

```
(map image-width (list 1 2 3 4))
```
not possible, image-width expects an image, not a list of integers

EL

The following operations are suggested by a set of examples.

For each sketched operation, write the best type that accords with the examples of its use. Some, but not all, of the correct answers include type variables. To be clear: the type only!

```
(: starts-with? : Char String -> Boolean)

ex: (starts-with? #\a "abcde") --> #t
ex: (starts-with? #\z "abcde") --> #f
ex: (starts-with? #\z "zebra") --> #t
```

```
(: average : (Listof Integer) -> Real)

ex: (average (list 10 20)) --> 15
ex: (average (list 10))    --> 10
ex: (average '())          --> 'CannotCompute
```

```
(: sum-with : All (a) (a -> Integer) (Listof a) -> Integer)

ex: (sum-with string-length (list "a" "bb" "c")) --> 4
ex: (sum-with sqr (list 1 2 3)) --> 14
```

```
(: count-if : All (a) (a -> Boolean) (Listof a) -> Integer)

ex: (count-if even? (list 1 2 3 4))     --> 2
ex: (count-if odd?  (list 1 2 3 4))     --> 2
ex: (count-if positive? (list 1 2 3 4)) --> 4
ex: (count-if empty? (list '() (cons 'q '()) '())) --> 2
```

```
(: sum-if : (Integer -> Boolean) (Listof Integer) -> Integer)

ex: (sum-if even? (list 1 2 3 4)) --> 6
ex: (sum-if odd?  (list 1 2 3 4)) --> 4
```

```
(: reject : (Integer -> Boolean) (Listof Integer) -> (Listof Integer))

ex: (reject even? (list 1 2 3)) --> (list 1 3)
ex: (reject odd?  (list 1 2 3)) --> (list 2)
```

```
(: contains? : All (a) (a -> Boolean) a (Listof a) -> Boolean)

ex: (contains? = 2 (list 1 2 3)) --> #t
ex: (contains? = 5 (list 1 2 3)) --> #f
ex: (contains? symbol=? 'x (list 'a 'b 'c)) --> #f
```

Implement the specified functions. Do not write check-expects.

You need not verify that the date is correct (i.e., is a real, sensible date) in any of these functions.

```
(struct Date
  ([month : Integer] ;; 1 means January, 2 February, ..., 12 December
   [day   : Integer]
   [year  : Integer]))

;; Return a string of the form "MM/DD/YYYY"
;; Both month and year must be two digits long in all cases.
(: format : Date -> String)

;; Given a month and a year, return the date representing
;; the first of that month.
(: first-of : Integer Integer -> Date)

;; Return true if the two dates are in the same year.
(: same-year? : Date Date -> Boolean)

;; Return true if the first date is before the second (on the calendar).
(: before? : Date Date -> Boolean)

;; A date is in autumn if it is on or after Sept 21 and on or before
;; Dec 21. (This may vary year to year, but, on this exam, assume it
;; does not.)
(: autumn? : Date -> Boolean)
```

next page →

```
(: two-digits : Integer -> String)
(define (two-digits n)
  (if (< n 10) (string-append "0" (number->string n))
      (number->string n)))
```

```
(: format : Date -> String)
(define (format d)
  (match d
    [(Date month day year)
     (string-append (two-digits month)
                    "/"
                    (two-digits day)
                    "/"
                    (number->string year))]))
```

```
(: first-of : Integer Integer -> Date)
(define (first-of m y)
  (Date m 1 y))
```

```
(: same-year? : Date Date -> Boolean)
(define (same-year? x y)
  (match* (x y)
    [((Date _ _ year1) (Date _ _ year2)) (= year1 year2)]))
```

EL

(more space for Date functions)

```
(: autumn? : Date -> Boolean)
(define (autumn? dt)
  (match dt
    [(Date m d _) (cond
                    [(= m 9) (>= d 21)]
                    [(or (= m 10)(= m 11)) #t]
                    [(= m 12)(<= d 21)]
                    [else #f])))
(: before? : Date Date -> Boolean)
(define (before? x y)
  (match* (x y)
    [((Date m1 d1 y1)(Date m2 d2 y2))
      (cond
        [(< y1 y2)  #t]
        [(> y1 y2)  #f]
        [(< m1 m2) #t]
        [(> m1 m2) #f]
        [(< d1 d2) #t]
        [else #f ])]))
```

Implement the specified functions. Don't write check-expects.

```
(define-type (Treeof A)
  (U 'E (Node A)))

(define-struct (Node A)
  ([root : A]
   [lsub : (Treeof A)]
   [rsub : (Treeof A)]))

;; Add the given integer to every integer in the tree.
(: add-to : Integer (Treeof Integer) -> (Treeof Integer))

;; Use the given function to transform all items in the tree.
(: change : All (A B) (A -> B) (Treeof A) -> (Treeof B))

;; Replace takes four arguments: an equality function,
;; a value a, a value b, and a tree. The function must
;; return a new tree such that every occurrence of a has
;; been replaced by b.
(: replace : All (X) (X X -> Boolean) X X (Treeof X) -> (Treeof X))
```

if = equal to a, return b

```
(: add-to : Integer (Treeof Integer) -> (Treeof Integer))
(define (add-to n tree)
  (match tree
    ['E 'E]
    [(Node root 'E 'E) (Node (+ n root) 'E 'E)]
    [(Node root l r) (Node (+ n root) (add-to n l) (add-to n r))]))

(: change : All (A B) (A -> B) (Treeof A) -> (Treeof B))
(define (change func tree)
  (match tree
    ['E 'E]
    [(Node root 'E 'E) (Node (func root) 'E 'E)]
    [(Node root l r) (Node (func root) (change func l) (change func r))]))

(: replace : All (X) (X X -> Boolean) X X (Treeof X) -> (Treeof X))
(define (replace test a b tree)
  (match tree
    ['E 'E]
    [(Node root 'E 'E) (if (test root a) (Node b 'E 'E) (Node root 'E 'E))]
    [(Node root l r) (if (test root a) (Node b (replace test a b l)
                                               (replace test a b r))
                                       (Node root (replace test a b l)
                                                  (replace test a b r)))]))
```

EL

(more space for tree functions)

Implement the specified list functions. Don't write check-expects.

```
;; Remove the given symbol where it appears.
;; ex: (remove 'y (list 'x 'y 'z 'y 'x)) --> (list 'x 'z 'x)
(: remove : Symbol (Listof Symbol) -> (Listof Symbol))

;; Return the list of dates that are in the autumn.
;; The definition of autumn appears several pages earlier.
(: autumn : (Listof Date) -> (Listof Date))

;; Change the numbers so they are all within the given lower and upper
;; bounds. You may *assume* the given lower bound is less than or
;; equal to the given upper bound.
;; ex: (clip 0 9 (list -1 1 10 8 11 0 -1)) -> (list 0 1 9 8 9 0 0)
(: clip : Real Real (Listof Real) -> (Listof Real))

;; Return true if a majority of items pass the test.
;; ex: (majority? even? (list 1 2 3 -1)) --> #f
;; ex: (majority? positive? (list 1 2 3 -1)) --> #t
(: majority? : All (T) (T -> Boolean) (Listof T) -> Boolean)
```

```
(: remove : Symbol (Listof Symbol) -> (Listof Symbol))
(define (remove s ss)
  (match ss
    ['() '()]
    [(cons hd tl) (if (symbol=? hd s) (remove tl)
                      (cons hd (remove tl)))]))

(: autumn : (Listof Date) -> (Listof Date))
(define (autumn list)
  (filter autumn? list))

(: clip : Real Real (Listof Real) -> (Listof Real))
(define (clip l u list)
  (match list
    ['() '()]
    [(cons hd tl) (cond
                    [(< hd l) (cons l (clip tl))]
                    [(> hd u) (cons u (clip tl))]
                    [else (cons hd (clip tl))])]))
```

EL

(more space for list functions)

```
(: majority? : All (T) (T -> Boolean) (Listof T) -> Boolean)
(define (majority? test list)
  (> (/ (track-majority test list) (length list)) 1/2))

;; tracks how many in list pass test
(: track-majority : All (T) (T -> Boolean) (Listof T) -> Integer)
(define (track-majority test list)
  (match list
    ['() 0]
    [(cons hd tl) (+ (if (test hd) 1 0) (track-majority tl))]))
```

n - Integer d - Integer

i1 - Image

Please write the omitted type of function f.

(: f : Integer Image Image Integer -> Image)

```
(define (f d i1 i2 n)
  (if (<= n 0)
      (circle 1 "solid" "black")
      (overlay (f d i2 i1 (- n d))
               (i1 n))))
```

Assuming these definitions of a and b (types omitted; assume they are well-typed and checked):

```
(define (a n)
  (square n "outline" "black"))

(define (b n)
  (circle (exact-floor (/ n 2)) "outline" "black"))
```

and the definition of f above, either evaluate the following expression and sketch the result, or explain why it cannot be evaluated:

```
(f 60 a b 240)
```

(overlay (f 60 b a 180))
(a 240)

This will not be evaluated as it will enter an infinite loop because (<= n 0) will never evaluate to true. and the function will continue to call itself recursively.