Identifying information will be printed here.

University of Waterloo
Midterm Examination
CS 145

Term: Fall Year: 2024

Date: October 28th, 2024
Time: 7:00 pm- 8:50 pm
Duration: 110 minutes
Sections: 001-002
Instructors: Vasiga

Student Signature: _____

UW Student ID Number: _____

Number of Exam Pages (including this cover sheet)	14 pages
Additional Material Allowed	UW-Approved Calculators

Question	Points
Q1	8
Q2	11
Q3	6
Q4	6
Q5	5

Question	Points
Q6	6
Q7	11
Q8	8
Q9	6

Total Points: 67

Instructions:

- All code is to use the Intermediate Student with lambda language level.
- Individual questions will indicate which design recipe components must be included for full marks. Unless otherwise specified, you can assume the following:
  - Helper functions **only** require a contract, a brief purpose, and the definition.
  - Examples and tests must use `check-expect`. Unless otherwise allowed, they must be different from any examples supplied in the question. Examples also count as test cases.
  - Contracts and data definitions are expected to include any necessary requires clauses.
- Unless otherwise specified, you may assume that all arguments provided to a function will obey the contract.
- Functions you write may use:
  - Any function you have written in another part of the same question.
  - Any function we asked you to write for a previous part of the same question (even if you didn't do that part).
  - Any other built-in function or special form **discussed in the course**, unless specifically noted in the question.
  - Any built-in **mathematical** function.
- Unless otherwise specified, for questions where you are required to provide a value, you may use either `cons` or `list` notation. For stepper questions, switching between these notations does not count as a "step".
- Throughout the exam, you should follow good programming practices as outlined in the course such as appropriate use of constants and meaningful identifier names.
- You are **not** allowed to use any built-in Racket functions or special forms not discussed in lecture (e.g., `symbol->string`, `local`, etc.).
- If you believe there is an error in the exam, notify a proctor. An announcement will be made if a significant error is found.
- It is your responsibility to properly interpret a question.
  - Do not ask questions regarding the interpretation of a question; it will not be answered and you will only disrupt your neighbours.
  - If there is a non-technical term you do not understand, you may ask for a definition.
  - If, despite your best efforts, you are still confused about a question, state your assumptions and proceed to the best of your abilities.
- If you require more space to answer a question, you may use the blank page(s) at the end of this exam, but you must **clearly indicate** in the provided answer space that you have done so.

Points on this page: 8

1. (8 points) Fill in the following blanks with the most appropriate words, numbers, or expressions:

(a) (1 point) `(first (rest (rest (rest (list 'a 'b 'c 'd)))))` will produce:

_____.

(b) (1 point) How many functions will Racket automatically create after processing

`(define-struct my-foobar (foo bar))`? _____.

(c) (1 point) Using list notation, `(cons 'h (cons 31 (cons "lo!" empty)))` can be written as

_____.

(d) (2 points) Without using `list` or quoting, `(list (list 8 true) (list "test"))` can be written as _____.

(e) (1 point) The most appropriate contract for the built-in function `equal?` is

_____.

(f) (1 point) `(length (append (list (list empty)) (list 1 2 3)))` will produce: _____.

(g) (1 point) `(or (and (= 1 1) (not false)) (> 0 (/ 0 0)))` will produce:

_____

Points on this page: 4

2. (11 points) As you may recall from science class, matter is commonly in one of three different states: solid, liquid, or gas. The temperature at which a solid becomes a liquid is called the *melting point (mp)*. The temperature at which a liquid becomes a gas is called the *boiling point (bp)*. Note that a substance exactly at its melting point temperature will be a liquid and, similarly, a substance exactly at its boiling point temperature will be a gas. You may know that the melting point of water is 0°C (zero degrees Celsius) and the boiling point of water is 100°C. Alternatively, alcohol has a melting point of -114.1°C and a boiling point of 78.37°C. For this question, all temperatures are in Celsius.

To represent this information about a substance, we use the following structures and data definitions:

```
(define-struct substance (name mp bp))
;; A Substance is a (make-substance Str Num Num)
;; Requires: mp < bp

;; A State is (anyof 'solid 'liquid 'gas)

(define-struct matterstate (name state))
;; A MatterState is a (make-matterstate Str State)
```

(a) (1 point) Define a constant `h2o`, that is a `Substance` with the given specifications for water and the `name "water"`.

(b) (3 points) Write a function, `substance->state` that consumes a `Substance` and a temperature (in that order) and produces the `State` that the `Substance` would be in at that temperature. **Provide a contract and the function definition.**

Points on this page: 7

(c) (3 points) Write a function, `filter-by-state`, that consumes a list of `Substance`s, a `State` and a temperature (in that order) and produces a list of the `Substance`s that are in the given state at that temperature. **Provide a contract and the function definition**. You may use abstract list functions here, but are not required to do so.

(d) (4 points) Write a function `substances->matterstates` that consumes a list of `Substance`s and a temperature (in that order) and produces a list of `MatterState`s. For each `Substance` in the consumed list, the consumed temperature is used to determine the corresponding `MatterState`. The position of each `MatterState` in the produced list corresponds to the position of the `Substance` with the same name in the consumed list. **Provide a contract and the function definition**. You must use abstract list functions to solve this problem.

Points on this page: 6

3. (6 points) Determine the first, second, and final substitution steps of each of the following expressions. If the evaluation would result in an error, describe the error. Assume that the following **`define`** and **`define-struct`** appear on the left of all the expressions in this question.

```
(define-struct fxy (f x y))
(define (f x y)
  (cond [(empty? x) y]
        [else (+ (first x) y (f (rest x) (add1 y)))]))
```

(a) (2 points) `(not (fxy? (fxy-x (make-fxy (make-fxy 1 true 3)`
`(make-fxy 'a true 'c) (not false)))))`

[1st] ⇒

[2nd] ⇒

[final] ⇒

(b) (2 points) `(reverse (rest (reverse (rest (list 1 2 3 4)))))`

[1st] ⇒

[2nd] ⇒

[final] ⇒

(c) (2 points) `(f (list 1 2) (add1 2))`

[1st] ⇒

[2nd] ⇒

[final] ⇒

Points on this page: 6

4. (6 points) A positive integer d is a divisor of n if dividing n by d leaves no remainder, or in other words, n is a multiple of d. A divisor of n is proper if it is not equal to n. A perfect number is a positive integer greater than 1 that is equal to the sum of all its proper divisors. For example, $1, 2$, and 3 are all the proper divisors of 6, and $1+2+3=6$, so 6 is a perfect number. In fact, 6 is the smallest perfect number.
Hint: In Racket, you can use `(remainder n d)` to find the remainder of `n` divided by `d`.

(a) (4 points) Write a function `divisor-list` that consumes an integer `n` greater than 1 and produces a sorted list of all the divisors of `n`, where `1` is the first item in the list and `n` is the last item in the list. For example,

```
(divisor-list 8) => (list 1 2 4 8)
```

Provide the function definition. Do not use or write your own version of `sort` or `reverse`, but you may use other abstract list functions. The running time of your function should be $O(n)$.

(b) (2 points) Write a function `is-perfect?` that consumes an integer `n` greater than 1 and produces `true` if `n` is a perfect number, and `false` otherwise. Your solution should run in $O(n)$ time. **Provide the function definition**.

```
(is-perfect? 6) => true   ; 1 + 2 + 3 = 6
(is-perfect? 8) => false  ; 1 + 2 + 4 = 7
```

Points on this page: 5

5. (5 points) A general linear recurrence relation can be described as:

$$\begin{array}{rcll} f(n) & = & \sum_{i=1}^{k} a_i \cdot f(n-i), & n \geq k \\ f(i) & = & b_i, & 0 \leq i < k \end{array}$$

Notice that for the Fibonacci recurrence relation, $k = 2$, $a_1 = a_2 = 1$, and $b_0 = 0, b_1 = 1$.

(a) (2 points) Write the Racket function `dotprod` that consumes two lists of numbers of the same length, and computes the dot product of the elements. The dot product is the sum of the product of the elements at index i in both lists. For example, `(dotprod (list 1 2 3) (list 2 -2 5)) => 13` since $1 \cdot 2 + 2 \cdot (-2) + 3 \cdot 5 = 2 - 4 + 15 = 13$.
Provide the function definition.

(b) (3 points) Write the function `f`, which consumes two lists of numbers, `A` and `B`, and a non-negative integer `n`, and computes $f(n)$. Note that the list `A` contains the coefficients in the recurrence relation in the order `(list` a_1 a_2 ... a_k `)`, and the list `B` contains the base case values in the order `(list` b_0 b_1 ... b_{k-1} `)`. The running time of your function must be $O(kn)$. **Provide the function definition.**

Points on this page: 6

6. (6 points) Consider the following Racket function

```
(define (go n i f)
    (cond
      [(zero? n) i]
      [else (f n (go (sub1 n) i f))]))
```

(a) (1 point) What does `(go 5 0 +)` produce?

(b) (1 point) What does `(go 7 empty cons)` produce?

(c) (1 point) Write the contract for `go`.

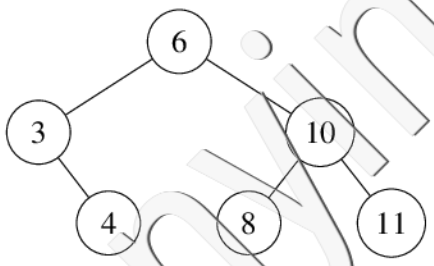
(d) (3 points) Using abstract list functions, and without explicit recursion, write the function `go2` which is equivalent to `go`. **Provide the function definition.**

Points on this page: 4

7. (11 points) Consider the following definition of binary list trees (BLTs).

```
;; A binary list tree (BLT) is either:
;;  * empty
;;  * (list key left right) where
;;    - key is an Int
;;    - left and right are BLTs
;;    - the keys in left are less than key
;;    - the keys in right are greater than key
```

(a) (2 points) Give the list representation of the following BLT:



(b) (2 points) Write the Racket function `height` that computes the height of a given BLT. For example, the height of the example BLT above is 3. **Provide the function definition.**

Points on this page: 7

(c) (2 points) Write the Racket function `find` that consumes an integer and a BLT (in that order), and produces `true` if that number is in the BLT, and `false` otherwise. The running time should be $O(h)$ where h is the height of the given BLT. **Provide the function definition.**

(d) (2 points) Write the Racket function `tree-min` that produces the smallest element in the given BLT. If the given tree contains no elements, `tree-min` should produce `false`. The running time should be $O(h)$ where h is the height of the given BLT. **Provide the function definition.**

(e) (3 points) Write the Racket function `preorder` that consumes a BLT, `T`, and produces list of the keys of `T` in preorder. For example, if `T` was the example BLT given earlier, then `(preorder T) => (list 6 3 4 10 8 11)`. The running time should be $O(n)$ where n is the number of keys of the given BLT. **Provide the function definition.**

Points on this page: 8

8. (8 points) **(a)** (2 points) State the formal definition of $f(n) \in O(g(n))$.

(b) (3 points) Is $2^n \in O(2^{\frac{n}{2}})$? Justify your answer using the formal definition of big-O.

(c) (3 points) Is $6n^2 + 19n + 10 \in O(n^2)$? Justify your answer using the formal definition of big-O.

Points on this page: 6

9. (6 points) For the following questions, you cannot use explicit recursion: that is, a function may not call itself, nor may you have any named helper functions. Use abstract list functions.

(a) (3 points) Write the Racket function `compose` which consumes a list of functions, `lof`, and a value `v`. Each function in the list consumes one value, which has the same type as `v`, and the value produced by each function is the same type as `v`. The produced value from `compose` should be the function composition $f_1(f_2(\cdots(f_n(v))\cdots)$. For example `(compose (list add1 sqr sub1) 3) => 5`. **Provide the contract and function body.**

(b) (3 points) Write the Racket function `fff`, which consumes a predicate `p` and list `L`, and does exactly what `(filter p L)` does. You may not use any other abstract list function other than `foldr` to solve this problem. **Provide the function body.**

This page is intentionally left blank for your use. Do not remove it from your booklet. If you require more space to answer a question, you may use this page, but you must **clearly indicate** in the provided answer space that you have done so.