	University of Waterloo
Term:	Spring 2019
Course:	CS 115
Exam:	Midterm Sample Examination
Time:	,
Duration:	110 minutes
Sections:	001–002
Instructors:	Morland
Permitted Aids:	Closed book; Reference Sheet Provided

Student Signature: _____

UW Student ID Number: _____

1. Ensure that this booklet contains 13 pages, and that you have a double-sided Reference Sheet.
2. There are 10 questions on this exam, worth a total of 47 points.
3. Do not write on or near the QR code in the top corner of each page. Doing so may result in loss of marks.
4. The Racket language level is *Intermediate Student with Lambda*.
   Do not write any recursive code unless a specific question requires it.
5. Supply exactly the parts of the design recipe requested in each question.
6. Unless otherwise indicated, "complete" means you should provide just the body of the function.
7. You may use helper functions and constants where you feel it is needed. For each helper function, you are only required to write the function header and body (definition). Helper functions may be written after the primary function if you wish.
8. You may use any function defined in the exam as a helper function for any other function.
9. If you need more space to complete an answer, you are likely writing too much. However, if you need more space, use the last page, and indicate that you have done so in the original question.
10. Only use functions discussed on the slides, allowed on assignments, or that are described in the String documentation on the course website, or on the Reference Sheet.

1. (4 points) For each of the following, determine what would happen if you opened DrRacket and tried to evaluate the given expression. If there is an error, briefly explain what is wrong; be specific. If there is no error, give the fully-simplified value of the expression.

Assume the following have been defined:

```
(define x 4)
(define stuff (list 2 3 4 4 5))
(define rope "nylon")
```

(a) (1 point) `(filter (lambda (x) (= x x)) stuff)`

(b) (1 point) `(and (= x 4) (< 3 4) (> x 5) (even? x) (= 0 (string-length x)))`

(c) (1 point) `(first (first stuff))`

(d) (1 point) `(substring rope (sqrt x) x)`

2. (4 points) Full marks will be given for the correct answer. Part marks may be given if you show your work, even if your final answer is incorrect.

(a) (2 points) Given the following definitions:

```
(define a 5)
(define b 7)
(define c 8)
(define (f a b) (g (* a b) 5))
(define (g a b) (+ (- c b)
                   (* a b)))
```

Evaluate `(f 2 3)`.

(b) (2 points) Given the following definition:

```
(define (h k)
  (cond [(= k 7) 11]
        [else
         (+ 3 (h (- k 1)))]))
```

Evaluate `(h 10)`.

3. (6 points) The function `(find-parts s L n)` consumes three values: a `Str`, a `(listof Nat)`, and a `Nat`.

The function returns the `(listof Str)` containing the substrings of `s` of length `n`, which start at each value in `L`.

Any values in `L` which are too close to the end are ignored.

For example,

```
(find-parts "foobar" (list 0 2 3) 3) => (list "foo" "oba" "bar")
(find-parts "foobar" (list 0 2 3) 4) => (list "foob" "obar")
```

(a) (2 points) Write the contract, including any requirements, for the function `find-parts`.

(b) (4 points) Complete the function:

```
(define (find-parts s L n)
```

4. (4 points) John McCarthy designed the Lisp language, which is the basis for Racket. For testing, he created the *McCarthy 91 function*. It is a recursive function described by the following equation:

$$M(n) = \begin{cases} n - 10, & \text{if } n > 100 \\ M(M(n+11)), & \text{if } n \leq 100 \end{cases}$$

(a) (2 points)

Complete the function using recursion:

```
;; (McC n) compute the McCarthy-91 function on n.
;; McC: Nat -> Int
;; Example:
(check-expect (McC 100) 91)

(define (McC n)
```

(b) (2 points) Evaluate, and show your work:

```
(M 98)
```

5. (6 points) In this question you will create a function `(divisibles L D)` that consumes two `(listof Nat)` and returns a list containing all the values in `L` that are divisible by one or more values in `D`.

You require a helper function `(has-divisor? n D)`, that consumes a `Nat` and a `(listof Nat)`. The function returns `#true` if `n` is divisible by one or more values in `D`.

(a) (2 points) Write the contract, including any requirements, for the function `has-divisor?`.

(b) (2 points) Complete the function `has-divisor?`.

```
(define (has-divisor? n D)
```

(c) (2 points) Complete the function `(divisibles L D)`.

```
(define (divisibles L D)
```

6. (6 points) The function `(join-even-str L)` consumes a `(listof Str)`. It returns the `Str` created by appending of all the values with an even length.

For example, `(join-even-str (list "when" "shall" "WE" "3" "meet" "AGAIN?") => "whenWEmeetAGAIN?")`

> **!** *Do not use `lambda`. For full marks, use exactly one helper function.*

(a) (2 points) Write the purpose, contract, and two examples for your helper function.

(b) (4 points) Complete your helper function and `join-even-str`.
(Design recipe is **not** required for `join-even-str`.)

7. (4 points) Recall that the function `sort` consumes a list `L`, and a predicate function `P`. `P` consumes two arguments; if it returns `#true`, the first argument will appear in the final list to the left of the second argument.

For example:

- `(sort (list 1 3 2 4) <) => (list 1 2 3 4)` since `(< 1 2)`, `(< 2 3)`, etc.

You will create the predicate function `(compare-length-alpha s1 s2)` that will cause `sort-length-alpha` to behave correctly. You may assume `L` contains only lowercase `Str`.

```
;; (sort-length-alpha L) sort L in increasing string length. For equal length,
;;   use alphabetic order.
;; sort-length-alpha: (listof Str) -> (listof Str)
;; Example:
(check-expect (sort-length-alpha (list "able" "was" "i" "ere" "i" "saw" "elba"))
              (list "i" "i" "ere" "saw" "was" "able" "elba"))

(define (sort-length-alpha L)
  (sort L compare-length-alpha))
```

(a) (1 point) Write two examples for `compare-length-alpha`.

```
;; (compare-length-alpha s1 s2) Return #true if s1 comes before s2, else #false.
;; compare-length-alpha: Str Str -> Bool
;; Examples:
```

(b) (3 points) Complete `compare-length-alpha`.

8. (4 points) The function `(remove-first values)` consumes a nonempty list of integers and returns a similar list, but without all occurrences of the first value. For example,

```
(remove-first (list 2 3 -1 2 8 2)) => (list 3 -1 8)
```

Another programmer gives you the following code, but it does not work.

```
(define (is-first n v1)
  (not (equal? n (first v1))))

(define (remove-first values)
  (filter is-first values))
```

(a) (2 points) Explain why this code does not work.

(b) (2 points) Explain in words (**not code**) how you would change this code to make it work.

> **!** *Do not write code. You must explain what you would change.*

9. (4 points) Write a function `(mul-end-distance L)` that consumes a `(listof Int)` and returns the `(listof Int)` which contains the values in `L`, where each value has been multiplied by its distance to the end.

For example,

```
(mul-end-distance (list 5 7 11 13))
=> (list (* 3 5) (* 2 7) (* 1 11) (* 0 13)) => (list 15 14 11 0)
```

10. (5 points) This question concerns the function `(replace-zeros L)`.

```
;; (replace-zeros L) replace each zero in L with the value after it.
;; replace-zeros: (listof Int) -> (listof Int)
;; Requires: the last value in L is not 0.
;; Examples:
(check-expect (replace-zeros (list 0 2 3 0 4 5 6)) (list 2 2 3 4 4 5 6))
(check-expect (replace-zeros (list 0 4 4 0 5)) (list 4 4 4 5 5))

(define (replace-zeros L)
  (foldr da0 '() L))
```

The helper function `da0` has the contract:

```
;; da0: Int (listof Int) -> (listof Int)
;; Requires: n is not zero if acc is empty.
```

(a) (1 point) Write a purpose statement for `da0`.

```
;; (da0 n acc)
```

(b) (1 point) Create two tests for `da0`.

(c) (3 points) Complete the function `da0`.

```
(define (da0 n acc)
```

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. If this page is used for continuing an answer, the marks for work done here will be included in that question's page mark.

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. If this page is used for continuing an answer, the marks for work done here will be included in that question's page mark.