Identifying information will be printed here.

University of Waterloo Midterm Examination #1 CS 135

Term: Fall Year: 2017

Date:	October 14, 2017
Time:	2:00 pm
Duration:	110 minutes
Sections:	001–011
Instructors:	Becker, Goldberg, Kaplan, Nijjar, Tompkins, Vasiga

Student Signature: ______________________________

UW Student ID Number: ____________________

Number of Exam Pages (including this cover sheet)	16 pages
Exam Type	Closed Book
Additional Material Allowed	NO ADDITIONAL MATERIALS ALLOWED

Instructions:

- All code is to use the Beginning Student language.
- 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 and a purpose.
  - Examples and tests must use *check-expect*, and unless otherwise allowed, must be different from any examples supplied in the question. Examples also count as test cases.
- Your functions do not have to check for values not specified by the contract unless we ask you to do so.
- 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 lecture**, unless specifically noted in the question.
  - Any built-in **mathematical** function.
- 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 **if** in any of your solutions.
- 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: 6

1. (6 points) You are probably familiar with the equation $E = m \cdot c^2$, made popular by physicist Albert Einstein (and later by non-physicist Mariah Carey), where c is the speed of light in a vacuum (299792458 metres per second). However, that formula only applies to objects *at rest*. For objects in motion, the kinetic energy is measured by:

$$\text{kinetic-energy}(m, v) = \frac{m \cdot c^2}{\sqrt{1 - \frac{v^2}{c^2}}} - m \cdot c^2$$

where v is the velocity in metres per second and m is the mass in kilograms.

(a) (1 point) Briefly explain why c is not a parameter of the function kinetic-energy, and provide Racket code to define c.

(b) (3 points) **Translate** the above formula for kinetic-energy from its mathematical description into Racket (as you did in Assignment 1). Note that *sqrt* and *sqr* are built-in mathematical functions in Racket. Give **only the definition**.

(c) (2 points) Provide a **contract** for kinetic-energy that would ensure your function produces a non-negative real number (and never generates an error).

Points on this page: 5

2. (10 points) Canadian travellers typically have to pay **duty** if they purchase any goods while they are travelling and return to Canada with those goods. However, it depends on the duration of the trip, the nature of the goods (*e.g.*, alcohol) and the profession of the traveller.

- Travellers whose *profession* is either `'ambassador` or `'diplomat` never pay duty.
- For durations less than 24 hours, duty must be paid on all purchases.
- For durations 24 hours or longer but less than 48 hours, duty is paid if the amount of goods exceeds $200 or the purchased goods includes alcohol.
- For durations 48 hours or longer, duty is only paid if the amount of the goods exceeds $800.

Your goal is to write **function definitions** for *pay-duty?* in two different ways. You are only allowed to use **define** once in each part (*i.e.*, no helper functions and you do not have to define any constants).

Note: the purpose and contract for *pay-duty?* is provided **at the top of the next page**.

(a) (5 points) Write the *pay-duty?* function using only **cond**. You may not use **and**, **or** or *not*.

Points on this page: 5

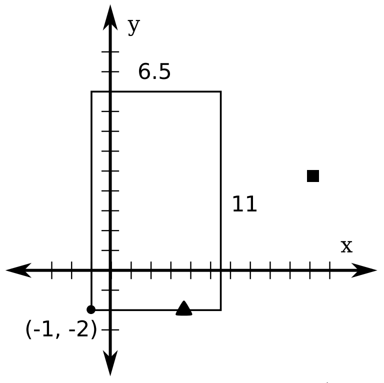
```
;; (pay-duty? duration goods alcohol? profession) produces true if the traveller needs to pay
;;     any amount of duty when returning to Canada, given the duration of the trip in hours,
;;     the amount of goods purchased in dollars, whether the goods includes any alcohol,
;;     and the profession of the traveller. It produces false otherwise.
;; pay-duty?: Nat Num Bool Sym → Bool
;; requires: goods ≥ 0 and must be > 0 if alcohol? is true
```

(b) (5 points) Write the *pay-duty?* function *without* **cond** (you may use **and**, **or** and *not*).

Points on this page: 5

3. (8 points) In this question you will need to "think outside the box." A box can be represented in the *x*-*y* plane by the coordinate of the lower-left corner and the width and height of the box (both of which are greater than zero).

The lower-left corner of this box is at $(-1, -2)$. It has a width of 6.5 and a height of 11.
Every point on the *x*-*y* plane is either located `'inside` this box, `'outside` it, or on the `'boundary`. The point indicated by the solid square is `'outside` the box, and the point indicated by the solid triangle is on the `'boundary`.



(a) (5 points) Write the **function definition** for the *point-location* function. For this question, helper functions with meaningful names and parameter names do not require a purpose or contract (you may find helper functions useful to break down the problem, but they are not required).

```
;; (point-location box-x box-y width height x y) determines the position of point (x,y) with respect
;;     to a box defined by with the lower-left corner at (box-x, box-y) with the given width and height.
;; point-location: Num Num Num Num Num Num → (anyof 'inside 'outside 'boundary)
;; requires: width, height > 0
```

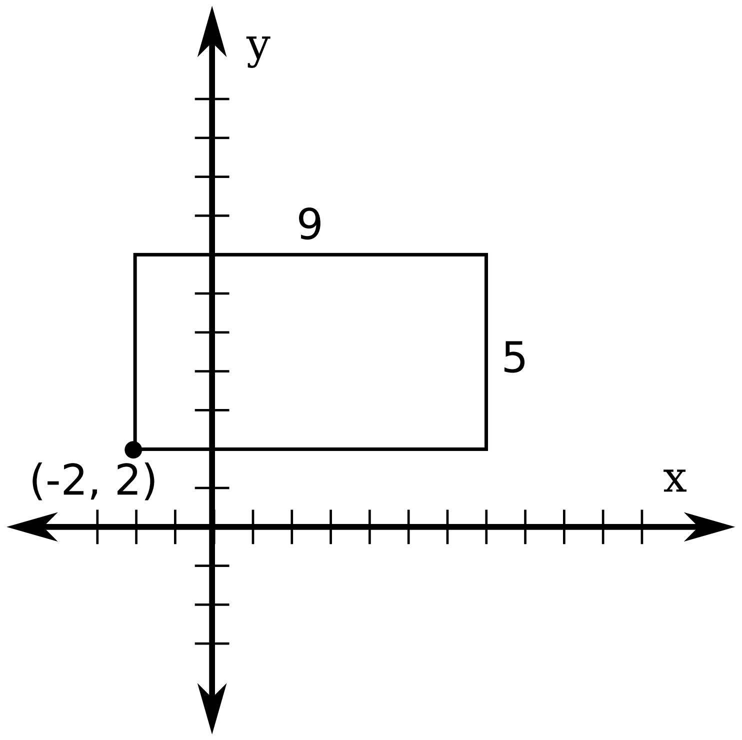
(This is more space for your function definition, if you need it.)

Points on this page: 3

(b) (2 points) Identify appropriate test cases for *point-location* by drawing points on the diagram below. (In practice you might want to test with boxes of varying sizes and locations, but for this question we will use this one box.)

The number of tests should be based on the complexity of your code: enough to test your code thoroughly, but without redundant cases that repeat conditions handled by other tests. (If you did not complete the previous part then you can still write test cases for the function in this part.)

Indicate test points 'outside the box with squares (□), points on the 'boundary with triangles (△), and points 'inside the box with hearts (♡).



(c) (1 point) Choose one of the 'inside (♡) test cases you drew above and write the test case in Racket using *check-expect*.

Points on this page: 4

4. (10 points) Assume that the following definitions have already been completely simplified in Beginning Student (*i.e.*, you have opened `DrRacket`, typed the following code, and clicked `Run`):

```
(define a 12)
(define b (/ a 2))

(define x1 (+ a 1))
(define y1 a)
(define x2 (/ a 3))
(define y2 6)

(define (mdist x1 y1 x2 y2)
  (+ (abs (- x1 x2))
     (abs (- y1 y2))))
```

For each of the following Racket expressions, provide the following three lines:

- The *first* expression that would result from exactly one substitution step, using the substitution rules as defined in lectures;
- The expression that would result from the *second* substitution step; and
- The *final value* that would result after executing all substitution steps. If the evaluation would result in an error, describe the error. You do not need to show any additional intermediate steps.

(a) (*mdist* 3 7 4 6)

[1st] $\Rightarrow$

[2nd] $\Rightarrow$

[final] $\Rightarrow$

(b) (*mdist y2 x1 x2 y1*)

[1st] $\Rightarrow$

[2nd] $\Rightarrow$

[final] $\Rightarrow$

Points on this page: 6

(c) (**or** (< 13 5) (> 2 3))

[1st] $\Rightarrow$

[2nd] $\Rightarrow$

[final] $\Rightarrow$

(d) (*make-posn* (*posn-x* (*make-posn* 7 3)) (*posn-y* (*make-posn* *y2* (+ 3 *a*))))

[1st] $\Rightarrow$

[2nd] $\Rightarrow$

[final] $\Rightarrow$

(e)
```
(cond
  [(symbol=? 'papaya 'bumblebee) 83]
  [(>= x1 (+ x2 x2)) x2]
  [else y1])
```

[1st] $\Rightarrow$

[2nd] $\Rightarrow$

[final] $\Rightarrow$

Points on this page: 2

5. (13 points) In their spare time, the CS 135 instructors have been working as vegetable farmers. They need to decide what types and quantities of seeds to purchase and plant in their fields. They have combined their love of CS 135 with their love of farming by using Racket to help plan seed purchases.

Seeds are planted in rows, and must be spaced a specified distance apart to ensure they grow properly. Seeds are purchased in packets, and farmer Paul has determined that a seed packet can be represented as a Racket structure containing the following pieces of information:

- the name for the seed's plant,
- the quantity (number) of seeds in each packet,
- the price of the seed packet (in dollars),
- the seed spacing (*i.e.*, how far apart each seed should be planted), in metres.

For example,
(*make-packet* "Red Bell Pepper" 25 5.25 0.5)
defines a packet of "Red Bell Pepper" seeds which contains 25 seeds and costs $5.25 per packet. Each Red Bell Pepper should be spaced 0.5 metres apart.

(a) (1 point) Use **define-struct** to define Farmer Paul's *packet* structure.

(b) (1 point) Farmer Craig always insists on a data definition. Write the data definition for *Packet* as a comment, including any necessary **requires** clauses.

Points on this page: 5

Given two different packets of seeds, Farmer Dave would like a Racket function to determine the cheaper packet per seed. For example,

(*cheaper-seed*
 (*make-packet* "Red Bell Pepper" 25 5.25 0.5)
 (*make-packet* "Ghost Pepper" 2 10.00 0.5))

would produce (*make-packet* "Red Bell Pepper" 25 5.25 0.5) because each Red Bell Pepper seed costs $0.21 and each Ghost Pepper seed costs $5.00.

If two seed packets have the same cost per seed then function should produce the first one.

(c) (2 points) Farmer Troy always insists that you write design recipe components before you define a function. Write the **purpose** and **contract** (including any necessary **requires** clauses) for the *cheaper-seed* function.

(d) (3 points) Write the **function definition** for Farmer Dave's *cheaper-seed* function.

Points on this page: 2

Farmer Byron wants to test the *cheaper-seed* function using *check-expect*.

(e) (2 points) Write three distinct **test cases** for the function *cheaper-seed*. Each test case should test a different aspect of the function.

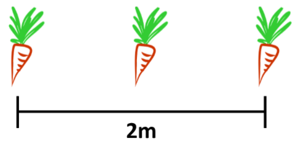
Here are some constants that might be helpful for your testing. You may define more if you think they are necessary.

```
(define red-pepper (make-packet "Red Bell Pepper" 25 5.25 0.5))
(define yellow-pepper (make-packet "Yellow Bell Pepper" 50 10.50 0.5))
(define parsnip (make-packet "Parsnip" 100 1.00 0.5))
(define cabbage (make-packet "Head Cabbage" 400 2.00 0.5))
```

Farmer Ian's fields are divided into rows, and he would like a Racket function *packets-needed* to determine how many packets of a seed he will need to fill a row, given the length of the row (as a positive number in metres). For example, consider the following definition:

(**define** *carrots* (*make-packet* "Carrots" 5 0.10 1))

Carrot seeds are spaced 1 metre apart. For a row 2 metres long, 3 carrot seeds can be planted.



However, each packet contains 5 carrot seeds, so only 1 packet would be needed.

(*packets-needed carrots* 2) ⇒ 1

The number of packets needed and the number of seeds per row must both be positive integers. You cannot plant fractional seeds or purchase a fraction of a packet, and you can always plant at least one seed, which will require at least one packet.

The built-in Racket functions *floor* and/or *ceiling* might be useful:

```
;; (floor x) produces the largest integer less than or equal to x ("rounding down").
;; floor: Num → Int
;; (ceiling x) produces the smallest integer greater than or equal to x ("rounding up").
;; ceiling: Num → Int
```

(f) (4 points) Write the **function definition** for Farmer Ian's *packets-needed* function.

```
;; (packets-needed seed length) determines the number of packets of the given seed type
;;     that are required to fill a row length (in metres).
```

(This is more space for your function definition, if you need it.)

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.