Important notes about this examination

1. This exam has 2 separate parts. You have 90 minutes to complete both parts. When you are done with part A raise your hand and we will collect that and give you part B. We recommend that you save yourself 20 minutes for part B.

2. Except for questions 1 and 2, this exam will be graded largely on how well you follow the design recipes. You have been given a copy of the Recipe Exam Sheet. Use it!

3. Put away books, papers, laptops, cell phones... everything but pens, pencils, erasers and this exam.

4. Good luck!

Rules Governing Formal Examinations

1. Each candidate must be prepared to produce, upon request, a UBC card for identification.

2. Candidates are not permitted to ask questions of the invigilators, except in cases of supposed errors or ambiguities in examination questions. No questions will be answered in this exam. If you see text you feel is ambiguous, make a reasonable assumption, write it down, and proceed to answer the question.

3. No candidate shall be permitted to enter the examination room after the expiration of one-half hour from the scheduled starting time, or to leave during the first half hour of the examination.

4. Candidates suspected of any of the following, or similar, dishonest practices shall be immediately dismissed from the examination and shall be liable to disciplinary action:
   ○ having at the place of writing any books, papers or memoranda, calculators, computers, sound or image players/recorders/transmitters (including telephones), or other memory aid devices, other than those authorized by the examiners;
   ○ speaking or communicating with other candidates; and
   ○ purposely exposing written papers to the view of other candidates or imaging devices. The plea of accident or forgetfulness shall not be received.

5. Candidates must not destroy or mutilate any examination material; must hand in all examination papers; and must not take any examination material from the examination room without permission of the invigilator.

6. Candidates must follow any additional examination rules or directions communicated by the instructor or invigilator.
Problem 1 - Mechanisms

[3 points - 1 each. all or nothing; exact correct syntax required]

(A) What is the value of

\[
\left\lfloor \frac{\left( 2 \times 6 \right) \times 2}{2} \right\rfloor
\]

3

(B) Given

\[
\text{(define (foo x y)}\n\text{ (cond \[(= x y) (* x y)]\n[ (> x y) (- x y)]\n[else (- y x)])\n}\text{)}
\]

what is the value of:

\[
\text{(foo 4 6)}
\]

2

(C) Given

\[
\text{(define-struct student (n h))}
\]

\[
\text{(define S1 (make-student "Harry" "Gryffindor"))}
\text{(define S2 (make-student "Tom" "Slytherin"))}
\]

what is the value of:

\[
(> \text{(string-length (student-h S1)) \text{(string-length (student-h S2))}}
\]

true
Problem 2 - Mechanisms

Given

\(\text{(define (foo x)}\)
\(\text{ (if (>= x 0)}\)
\(\text{ x} \)
\(\text{ (* -1 x))})\)

Show the step-by-step evaluation of the following expression. Including the original expression and the final result there should be 6 steps.

\(\text{(foo (- 2 5))}\)

[5 points -
1 for each correct step, each step is all or nothing,
small parentheses errors in expressions ok,
but parens in results make that step wrong
-1 for extra steps]

\(\text{(foo -3)}\)

\(\text{(if (> = -3 0)}\)
\(\text{ -3} \)
\(\text{ (* -1 -3))}\)

\(\text{(if false}\)
\(\text{ -3} \)
\(\text{ (* -1 -3))}\)

\(\text{(* -1 -3)}\)

3
Problem 3 - Form of Data

In this problem you will be given small fragments of problem descriptions. Each fragment
describes some information in a problem domain that must be represented using data in a
program. In each case you need to choose the form of data that would be appropriate to
represent this information. Mark your choice by circling the most appropriate form of data.

[4 points - 1 each. all or nothing]

(A) A person’s age.

[simple atomic] [interval] [enumeration] [itemization] [compound] [arbitrary-sized]

(B) The first name of everyone in a course.

[simple atomic] [interval] [enumeration] [itemization] [compound] [arbitrary-sized]

(C) A student’s name, id number and major.

[simple atomic] [interval] [enumeration] [itemization] [compound] [arbitrary-sized]

(D) An olympic medal which is gold, silver or bronze.

[simple atomic] [interval] [enumeration] [itemization] [compound] [arbitrary-sized]
**Problem 4 - Data Definitions**

You are working on a program to rank scores on a national exam. The scores range from 0 to 600. The exam is designed so that scores less than 300 indicate insufficient knowledge of the material; scores from 300 to 499 inclusive indicate competence; and scores 500 and above indicate excellence. Be sure to do both part A and part B of this question.

(A) Design a data definition called Score to represent these scores. Use an itemization of intervals. Include all the parts of a data definition the recipe calls for, including the template rules used.

[11 points as follows:
[ 3 - 1 each if the type has 3 well-formed interval subclasses
[ 6 - 1 each for correct upper and lower bounds of intervals
[ if Number is used instead of Integer/Natural then bounds have to be different, Number\([300, 500)\) for example.
[ 1 for reasonable interp
[ 1 for 1 good example
[ -1 for case errors
[ -2 if use Number in intervals

[13 points as follows:
[ 3 - 1 each for 3 case cond (-1 for each extra case)
[ 3 - 1 each if cond questions correctly match type comment
[ 3 - 1 each if cond answers are (... s)
[ 4 - 1 each for correct template rules used, part after ‘:’ is optional
[ -2 if order of cases in template does not match type comment

[3 points if answer to part B is 9

;; Score is one of:
;; - Integer\([0, 300)\)
;; - Integer\([300, 499]\)
;; - Integer\([500, 600]\)
;; interp. national exam scores
(define S1 20)

(define (fn-for-score s)
  (cond [(< s 300) (... s)]
        [(<= 300 s 499) (... s)]
        [else (... s)]))
(B) You do not need to design a function operating on Score. But in general, how many test cases would you expect a function operating on your Score data definition to have.
**Problem 5 - Designing Functions**

As part of your summer job working at a fitness center you are working on the customer profile system. Some of the customers are enrolled in martial arts classes, and as a result they have martial arts belts. The following data definition is in the program you are working on:

```scheme
;; BeltLevel is one of:
;; - "black"
;; - "red"
;; - "blue"
;; - "white"
;; interp. the level of a martial arts student
;; "black" is highest, "white" is lowest
;; <examples are redundant for enumerations>
```

```scheme
(define (fn-for-belt-level b)
  (cond [(string=? b "black") (...)]
        [(string=? b "red") (...)]
        [(string=? b "blue") (...)]
        [(string=? b "white") (...)]
  )
)
```

For compatibility with another system your boss would like a function that consumes a belt level and produces its numeric equivalent in that system. White is 0, blue is 1, red 2 and black 3. Please provide the complete design for the function. Include a copy of the stub. You can save a little time by abbreviating check-expect to c-e.
[13 points as follows:
[2 - 1 each for proper types in Signature, note Integer is OK for result, but Number is not
[2 for a purpose that says in some specificity what the function does
[4 - 1 each for all four tests
[3 - function clearly follows template
[2 - function is correct
[Note, allow 4 for black as well as 3, problem description changed
[-1 extra arg to fn in c-e, uppercase colours in c-e, no quotes around strings
[-1 no rename of functions, handling other colours, returning strings
[-1 missing arg in string=?]
[-2 no fn call in c-e

;; BeltLevel -> Natural
;; produce numeric equivalent of belt level
(check-expect (numeric-equiv "black") 3)
(check-expect (numeric-equiv "red") 2)
(check-expect (numeric-equiv "blue") 1)
(check-expect (numeric-equiv "white") 0)

(define (numeric-equiv b)
  (cond [(string=? b "black") 3]
        [(string=? b "red") 2]
        [(string=? b "blue") 1]
        [(string=? b "white") 0])))
Problem 6A - Domain Analysis for a World Program - Part 1

In this problem you will start the design of a world program that displays a blade of grass growing. When the program starts the screen should be blank. As time goes by a green line should grow up in the middle of the screen.

All you need to do for this part of the problem is the domain analysis part of the HtDW recipe, specifically:
- draw several sketches
- identify constant information
- identify changing information and
- identify big-bang options required.

Be sure to make your diagrams clear and neat. Be sure to clearly distinguish constant information from changing information. Do not worry about the exact names of the big bang options, just put something that makes it clear you know which options are needed.

[ 15 points as follows:
  [ 3 - 1 each for three good pictures
  [ 4 - 1 each for reasonable constants including:
    [ width, height, mts, grass color, grass width, speed
    [ 1/2 off if constant definitions instead of identification
  [ 4 - identifying height of grass as changing,
  [ 2 for identifying on-tick option (allow reasonable names for it)
  [ 2 for identifying to-draw option (allow reasonable names for it)