Assignments: 2
Examples Due: Friday, September 22, 2023 8:00 am
Remainder Due: Tuesday, September 26, 2023 9:00 pm
Coverage: Module 3, Slide 27
Language level: Beginning Student
Files to submit: examples-a02.rkt, median.rkt, cond.rkt, blood.rkt, triads.rkt, bonus-a02.rkt

Assignment policies, Correctness and Craft

Please refer to the policies, correctness and craft sections at the top of A01.

Grading

• You will not receive marks for A02 unless you have earned full marks for A00 before the A02 due date. Assignments may be submitted as often as desired up to the due date.

• Your solutions for assignments will be graded on both correctness and on craft.

• For each function in Questions 2, 4a, and 5, you are required to submit examples that are illustrated with the use of check-expects. You should be providing enough examples to thoroughly show how the function works. You are required to submit these examples by Friday, September 22 at 8:00AM. Note: this is before the due date of assignment A02. This early submission component encourages you to start thinking about the assignment early. Place your examples in examples-a02.rkt. You do not need to have the functions themselves implemented, only the check-expects. This grading component will be worth approximately 5% of your grade for each of these questions.

Here are the assignment questions that you need to submit.

1. (20%): Complete the two required stepping problems in Module 3a and the three required problems in Module 3b at:

   https://www.student.cs.uwaterloo.ca/~cs135/assign/stepping/

   You should refer to the instructions from A01 Question 1 for the stepper question instructions.
2. (10%): The following function produces the median of 3 numbers:

\[
\text{define} \ (\text{median-of-3} \ a \ b \ c) \\
\text{(cond)} \\
[(\text{or} \ (\text{and} \ (<= \ b \ a) \ (<= \ a \ c)) \ (\text{and} \ (<= \ c \ a) \ (<= \ a \ b))) \ a] \\
[(\text{or} \ (\text{and} \ (<= \ a \ b) \ (<= \ b \ c)) \ (\text{and} \ (<= \ c \ b) \ (<= \ b \ a))) \ b] \\
[(\text{or} \ (\text{and} \ (<= \ b \ c) \ (<= \ c \ a)) \ (\text{and} \ (<= \ a \ c) \ (<= \ c \ b))) \ c])
\]

Simplify this function so that it reduces the maximum number of inequality comparisons that could be performed. Inequalities include >, >=, <, and <=. Racket has versions of the inequality operators that consume more than 2 arguments. You may not use them for this question.

In addition to the above inequality operators, you may only use define, cond, =, and, or, or not. Note: else is allowed whenever cond is allowed. Do not use any helper functions for this question.

The original function performs a maximum of 12 comparisons. A trivial transformation gets it down to 8. How much lower can you go? There may be several solutions depending on how you decide to write the function. Full marks will be granted if you can get the maximum number of possible comparisons down to at most 5 comparisons.

Name your function median-of-3-simple.

Hint: It is permissible to include median-of-3 in your code and use it to run check-expects.

Appropriate examples to submit by the Friday morning deadline for this function might include:

\[
(\text{check-expect} \ (\text{median-of-3-simple} \ 1 \ 2 \ 3) \ 2) \\
(\text{check-expect} \ (\text{median-of-3-simple} \ 2 \ 1 \ 3) \ 2)
\]

Examples using check-expect are all you need to submit by Friday morning for this function!

Place your solution code in the file median.rkt.
3. (30%): A **cond** expression can always be rewritten to produce equivalent expressions. These are new expressions that always produce the same answer as the original (given the same inputs, of course). For example, the following are all equivalent:

\[
\text{(cond \[(\text{> x } \text{ 0}) \text{'red}\] [\text{\leq x } \text{ 0}) \text{'blue}\])}
\]

\[
\text{(cond \[(\text{\leq x } \text{ 0}) \text{'blue}\] [\text{> x } \text{ 0}) \text{'red}\])}
\]

\[
\text{(cond \[(\text{> x } \text{ 0}) \text{'red}\] [\text{else 'blue}\])}
\]

(There is one more really obvious equivalent expression; think about what it might be.)

Many of the **cond** examples we’ve seen in class have followed the pattern

\[
\text{(cond \[\text{question1 answer1}\] \[\text{question2 answer2}\] \text{...} \[\text{questionk answerk}\])}
\]

where questionk might be **else**.

The questions and answers do not need to be simple expressions. In particular, either the question or the answer (or both!) can themselves be **cond** expressions. In this problem, you will practice manipulating these so-called “nested cond” expressions.

In some cases, having a single **cond** results in a simpler expression, and in others, having a nested **cond** results in a simpler expression. With practice, you will be able to simplify expressions even more complex than these.

Below are functions whose bodies are nested **cond** expressions. Write new versions of these functions subject to the following constraints:

- Each function uses exactly one **cond** (you may use **and**, **or**, and **not**).
- There may be several solutions depending on how you decide to write the function. For your chosen approach, you should have as few question/answer pairs as possible and it should not be possible to further simplify the questions.
- All of the **cond** questions are “useful”, that is, there exists no question that could never be asked or that would always answer **false**.
- The functions **q3a**, **q3b**, and **q3c** all consume either numerical or Boolean values. **q3a** produces a number and the rest produce symbols. The parameter names ending in a question mark are Boolean values.

(a) (Name this function **q3a-simplified**)

\[
\text{(define (q3a n a?)}
\text{(cond [a? (cond [(\text{\geq n 0}) (\text{+ n 1})]
\text{[else (- n 1)])]
\text{[else 0]])})}
\]

CS 135 — Fall 2023  Assignment 2
(b) (Name this function q3b-simplified)

\[
\text{(define (q3b a? b? c?)
 (cond
   [a? (cond [b? 'elm]
             [(not c?) 'birch]
             [else 'cedar]])
   [else (cond ; Yuck. else followed by cond!
             [b? 'pine]
             [(not c?) 'birch]
             [else 'cherry]]))}
\]

(c) (Name this function q3c-simplified)

\[
\text{(define (q3c a? b? c?)
 (cond
   [(cond [c? b?]
           [else (not a?)
               (cond
                 [b? 'spruce]
                 [c? 'fir]
                 [else 'larch])]
           [else (cond [a? 'hazel]
                        [else 'hickory]])]
   [else (cond [a? 'hazel]
                [else 'hickory]]))}
\]

Place your solution code in the file cond.rkt.

4. (20%): Everyone has a “blood type” that depends on many things, including the antigens they were exposed to early in life. There are many different ways to classify blood; one of the most common is by group: O, A, B, and AB. This is augmented by the “Rh factor” which is either “positive” or “negative”. This yields a set of eight relevant types. We’ll use the following symbols to represent them: 'O-, 'O+, 'A-, 'A+, 'B- 'B+, 'AB-, and 'AB+.

If a person needs a blood transfusion, the type of the donor’s blood is restricted to types which the recipient’s body can accept. In the following chart, a checkmark indicates which types of blood are acceptable to donate to a recipient. For example, a person with type '0+ can donate to a person with type 'A+, but not to someone with a type 'B-. You can observe that '0- is sometimes referred to as the universal donor and 'AB+ is sometimes referred to as the universal recipient.
(a) Write the function `can-donate-to/cond?` which consumes a symbol denoting the donor’s blood type as the first parameter and a symbol denoting the recipient’s blood type as the second parameter. Produce `true` if the donor’s blood type is acceptable for the recipient’s blood type, according to the above chart, and `false` otherwise. `can-donate-to/cond?` **must** use `cond` expressions **without** using `and`, `or`, or `not`. Your solution will be marked for style. You should not hard-code every possible combination of blood types in your solutions. Try to find patterns in the data and use those patterns to make your program easier to read and understand.

Hint: Use `symbol=?` to compare two symbols for equality.

Appropriate examples to submit by the Friday morning deadline for this function might include:

```racket
(check-expect (can-donate-to/cond? 'O- 'O-) true)
```

Examples using check-expect are all you need to submit by Friday morning for this function!

(b) Write the function `can-donate-to/bool?` which is identical to `can-donate-to/cond?` except that it uses **only** a Boolean expression. That is, it does **not** have a `cond` expression.

Make sure you type all symbols exactly as they are written in the question. Place your answers in the file `blood.rkt`.

5. **(20%)**: Western music is based on a set of twelve distinct pitches arranged in the *chromatic scale*. (The scale repeats after every *octave*, but pitches that differ only by their octaves “sound the same”.) We can refer to these pitches via the Racket symbols: `c`, `c#` ("C Sharp"), `d`, `d#`, `e`, `f`, `f#`, `g`, `g#`, `a`, `a#`, and `b`. Here they are on a piano keyboard:
The distance between any two consecutive pitches in this list is called a semitone. For example, there are nine semitones from ‘c’ to the next highest ‘a’, and five more semitones from ‘a’ to the ‘d’ above that (because the cycle repeats with a ‘c’ after every ‘b’).

If we choose any base pitch, called the root, there are four standard three-note chords called triads that can be built on top of it:

- The Major triad adds the notes four and seven semitones above the root;
- The Minor triad adds the notes three and seven semitones above the root;
- The Diminished triad adds the notes three and six semitones above the root; and
- The Augmented triad adds the notes four and eight semitones above the root.

Here is a set of four such triads, for the case where the root pitch is ‘c’:

Write a Racket function triad-type that consumes the three notes in a triad and produces one of the four symbols ‘Major’, ‘Minor’, ‘Diminished’, or ‘Augmented’, indicating the type of the triad. The function consumes exactly three symbols, which denote the name of the root pitch, the name of the second pitch in the triad, and the name of the third pitch in the triad. You can assume that the inputs will define a legal triad. So, for example, we would expect (triad-type ‘a ‘c ‘e) to produce ‘Minor. (If you’re stuck, consider writing a helper function that converts pitch names to natural numbers.)

Place your solution in the file triads.rkt.
Appropriate examples to submit by the Friday morning deadline for this function might include:

(check-expect (triad-type 'c 'e 'g) 'Major)

Examples using check-expect are all you need to submit by Friday morning for this function!

For fun, we’ve also created a Racket program `synth.rkt`, which you can download from the assignments web page. This program provides a function `play-triad` that uses your computer’s speakers to play triads as defined in this question. For example, try `play-triad 'a 'Minor` or `play-triad 'e 'Augmented`. This program is included purely for entertainment purposes—it isn’t needed to solve the assignment, and won’t give away any part of the solution.

This concludes the list of questions for you to submit solutions (but see the following pages as well). Don’t forget to always check the basic test results after making a submission.

Assignments will sometimes have additional questions that you may submit for bonus marks.

6. **5% Bonus**: Write a function `date->day-of-week` which consumes a natural number and produces a symbol corresponding to the day of the week, according to the Gregorian calendar. An integer encodes a date as follows:

- The final two digits correspond to the day of the month
- The two digits before that correspond to the month of the year
- The digits before that correspond to the year

The function should return one of seven symbols: 'Monday, 'Tuesday, 'Wednesday, 'Thursday, 'Friday, 'Saturday, or 'Sunday according to the day of the week the consumed date corresponds to.

For example:

(check-expect (date->day-of-week 20230926) 'Tuesday)
(check-expect (date->day-of-week 38781202) 'Monday)

You can assume that all dates are correct. This means that your function does not have to check for invalid input, such as, April 65, 1900 ((date->day-of-week 19000465)). You can also assume that no dates before January 1, 1753 AD, will be tested.¹

You may only use the Racket constructs we have discussed in lecture so far, and built-in mathematical functions. You may not use Racket’s date functions.

It is acceptable to consult rules or algorithms for computing the day of the week given a date, but it is not acceptable to copy and paste code you have found. You must cite any resources you use in your Racket file.

Put your solution in `bonus-a02.rkt`.

¹Fun fact: the Gregorian calendar was not fully established in the United Kingdom and her colonies before that date ([https://en.wikipedia.org/wiki/Adoption_of_the_Gregorian_calendar](https://en.wikipedia.org/wiki/Adoption_of_the_Gregorian_calendar), as retrieved on 2023-06-23).
**Challenges and Enhancements**: Reminder—enhancements are for your interest and are not to be handed in.

**check-expect** has two features that make it unusual:

1. It can appear before the definition of a function it calls (this involves a lot of sophistication to pull off).

2. It displays a window listing tests that failed.

However, otherwise it is a conceptually simple function. It consumes two values and indicates whether they are the same or not. Try writing your own version named **my-check-expect** that consumes two values and produces `'Passed if the values are equal and `'Failed otherwise. Test your function with combinations of values you know about so far: numbers (except for inexact numbers; see below), booleans, symbols, and strings.

Expecting two inexact numbers to be exactly the same isn’t a good idea. For inexact numbers we use a function such as **check-within**. It consumes the value we want to test, the expected answer, and a tolerance. The test passes if the difference between the value and the expected answer is less than or equal to the tolerance and fails otherwise. Write **my-check-within** with this behaviour.

The third check function provided by DrRacket, **check-error**, verifies that a function gives the expected error message. For example, `(check-error (/ 1 0) "/: division by zero")`

Writing an equivalent to this is well beyond CS135 content. It requires defining a special form because `/ 1 0` can’t be executed before calling **check-error**; it must be evaluated by **check-error** itself. Furthermore, an understanding of exceptions and how to handle them is required. If you’re interested, you might want to take a look at exceptions in DrRacket’s help desk.