CS 131 - Exam 1 Study Suggestions

last modified: 09-30-10

* Remember: anything covered in lecture, in lab, or on a homework, is FAIR GAME.

* You are responsible for all of the material covered through Week 6 - Lecture 2 (09-30-10)

* These are simply suggestions of some especially important concepts, to help you in your studying.

* You are permitted to bring into the exam a single piece of paper (8.5" by 11") on which you have handwritten whatever you wish on one or both sides. This paper must include your name, it must be handwritten by you, and it will **not** be returned.

  Other than this piece of paper, the exam is closed-note, closed-book, and closed-computer.

* You will be writing and reading expressions and functions on this exam; you will be answering questions about concepts, expressions, and functions as well.

Programming Fundamentals

* What is meant by syntax? What is meant by semantics?

* In Racket, what is a simple expression? What is a compound expression? How do you write simple and compound expressions in Racket?
  * Given an expression, you should be able to tell if it is a "valid" Racket expression

* What Racket data types have we discussed so far? How can you write simple expressions that are literals (simply values) of each of these data types? What are the functions that we have discussed so far that are provided by Racket for each of these data types? How can predicates be used to determine the data type of an expression?

* how do you write a comment in Racket?

* What are some of the arithmetic operations/functions provided by Racket? Given a numeric operation in "algebraic" form, you should be able to write it as an equivalent Racket expression.

* When a variable represents an unchanging value within a program, we call it a "named constant"; why are these good to use within functions? What is a parameter variable? (a parameter is a variable defined in a function header, to stand in for an expression that this function will expect when it is called)
  * You could be asked to define a named constant and/or a parameter variable; you could be asked to use a named constant or a parameter variable that has been previously defined

* We used the term **identifier** to represent names that a programmer chooses (such as parameter variables, named constants, and function names). Can you have blanks in identifier names? What are the syntax rules for identifiers? What are the style guidelines for identifiers?
  * We know that an identifier (once defined) is considered to be a simple expression. Can you tell the difference between a simple expression that is an identifier, and a simple expression that is a literal?

  * Given a simple expression, can you tell if it is a "valid" Racket identifier? if it is a "valid" Racket literal?

* What is an auxiliary function? Within a Definitions file containing multiple functions, in what relative order

must the auxiliary functions and main function be?

* What is a parameter? Given a fragment of Racket code, could you name the parameters in that fragment?

* You should be very comfortable reading and writing Racket expressions, function definitions, named constant definitions, and function calls;
  * You should be comfortable with both simple and compound expressions

  * You should be comfortable with predicate functions

* Given a signature and purpose statement for a function, you should be able to make appropriate use of that function; you should be able to write compound expressions using that function

* You should be quite comfortable with the `2htdp/image` and `2htdp/universe` teachpacks, especially if you are provided with signatures and purpose statements for a selection of these functions.
  * You should be comfortable with how the `2htdp/universe` teachpack can be used to set up and animate a world, using `big-bang`

  * Be sure you are comfortable with the difference between a scene, an image, and a world

  * You should be able to read and create expressions involving images and scenes;

  * You should be comfortable with what kinds of functions need to be provided to `on-tick`, `on-draw`, `on-key`, and `on-mouse`;
    * You should be able to read and write such functions, and appropriate `big-bang` expressions;

    * Given such functions, named constant definitions, and `big-bang` call (or `main` function and call) involved, you should be able to describe the animation that should result, including being able to describe what should happen if the user presses certain keys or performs certain mouse events;

  * You should be comfortable reading and writing expressions involving `modulo` to make sure that some expression results in a value in a desired interval;
  * you should be comfortable reading and writing expressions involving `random`;
  * you should be comfortable with colors, whether represented as strings or as `make-color` expressions;

The Design Recipe

* You should be comfortable with the design recipe (since you should have been using it for all of the numerous functions that you have written up to this point).
  * There could be questions about the design recipe itself; you should also be comfortable with the expected order of the six steps in this process.

  * There could also be questions where you have to produce certain steps within the design recipe.

  * If the scenario for a problem is given, could you:
    * (step 1) perform data analysis and design data definition(s), if appropriate, for the kinds of data involved in that problem?

    * (step 2)
      * give the **signature** for each desired function involved in solving that problem?

    * write an appropriate **purpose statement** for each desired function involved? (...being sure to explicitly **describe** what the function **expects** and what it **produces**?)

    * give the **header** for each desired function involved?

* (step 3) develop **check-expect** calls with specific examples/test cases for each desired function involved?
    * Also remember to include examples for each major category/class of data, and for boundaries between those categories/classes of data if applicable

    * You should know the rules-of-thumb for how many specific examples/test cases are required for different situations/different types of data; expect some test questions in which you need to be able to determine reasonable numbers and varieties of specific examples/test cases

    * expect to have to write some specific examples/test cases!

* (step 4) determine if there are any **templates** that might be useful for helping to develop the body of a function involving the types of data involved in this function?
    * Why can this be helpful?

* (step 5) finish developing an appropriate **body**?

* (step 6) know how to then test your resulting code, and determine whether those tests succeeded?
    * How would you **debug** your code if any tests failed?

The `cond` expression

* You should be very comfortable with reading and writing the Racket **`cond`** (conditional) expression
    * You should be comfortable with interval notation for expressing ranges of numbers

    * You should be comfortable with boolean values, boolean operators (and relational operators), boolean expressions, and boolean functions

    * How can you see if one expression has the same value as another expression? How can you see if an expression is of a particular type?

    * expect to have to read, write some `cond` expressions!

structs (Structures)

* When might you decide to define and use a struct (structure)? How are they useful/beneficial?

* Be able to write and read **data definitions** for structures
* Be able to write and read Racket **`define-struct`** expressions

* Given a structure definition, what **constructor** function is created? what **selector** functions are created? what **predicate** is created?
    * How can these be used in making use of **instances** of your new structure?

    * How can you tell whether some expression is an instance of your new structure?

* You should be able to write and use a template for a function that expects a certain kind of struct as a

parameter, and a template for a function that produces a certain kind of struct instance

* You should be able to write functions that manipulate structures (functions that expect and use structure instances, and functions that create structure instances)

Enumerations and Itemizations

* You should know what kind of data each of these terms describes; you should be able to write a data definition for each of these kinds of data.

* You should know what template is appropriate for the body of a function that expects an Enumeration or Itemization type.

* Given such a data definition:
  ...what can you then include in a function's signature?
  ...how can you write functions handling such data?
  You should be able to read and write such functions.

Lists

* What is the data definition for a list? How do you write a list in Racket? What are the list-related built-in functions? What is the template for a function that expects a list as a parameter?
  * what is the more-complete template for a function that expects a list of structs as a parameter?

* I may ask you to give the value of expressions involving lists, or ask you to write an expression involving a list.

* I may ask you to give a data definition for a specialized list; I may ask you to give the template for a function that expects that specialized list type as a parameter.

* Expect to have to read a function involving a list; expect to have to write a function involving a list.

* When you are writing a recursive function - one that calls itself - what do we mean when we say that it needs one or more **base cases**? What do we mean when we say it has one or more **recursive cases**? What needs to be true about the recursive calls in the recursive cases?

File Input/Output

* What are the functions we discussed from the `2htdp/batch-io` teachpack? How can they be used to read data from and write data to a file?