Study Guide

Bookmark this page

Module 05 terminology

- **Auxiliary parameter:** In a recursive function, a parameter that "goes along for the ride", passed unchanged along a chain of recursive calls
- **Base case:** In a recursive function, a path through the function that terminates without requiring any more recursion
- **Box-and-pointer visualization:** A list visualization in which list items are arranged in a linked chain
- **Condensed trace:** Highlights from the trace of a function application, showing only the recursive calls and any final non-recursive steps
- **Data analysis:** A preliminary step in the Design Recipe where new types are defined based on the problem domain, and data definitions and templates are created for them
- **Data definition:** A comment that defines a new type name and describes the complete structure of values of that type
- **Data-directed design:** The principle that the form of a function should mirror the form of the data it processes
- **Filtering a list:** A list processing idiom where a sub-list is extracted from a list, consisting of all items that have a certain property
- **Folding a list:** A list processing idiom where a list is reduced down to a single value by repeatedly combining elements with a base value
- **Homogeneous list:** A list whose elements are all of a given type
- **Idiom:** In programming, a pattern or style of code that is useful in a wide range of contexts
- **Mapping a list:** A list processing idiom where every item of a list is transformed in a consistent way to produce a new list
- **Nested visualization:** A list visualization in which successive list items are shown nested within the items that refer to them
- **Recursion:** The property of a function that's defined in terms of itself. See also ***Recursion***
- **Self-reference:** A case in a data definition where a value is built from another value of the type currently being defined
- **Structural recursion:** A style of recursion in which recursive calls exactly follow the self-references in a data definition
- **Wrapper function:** A simple, non-recursive function that initiates a large computation, often preparing data for recursive processing and polishing up the result

Module 05 types

- (anyof t_1 ... t_n v_1 ... v_m)
- Char
- cons cell
- empty list
- (listof τ)

Module 05 functions, constants, and syntax

- cons
- cons?
- empty
- empty?
- first
- list->string
- list?
- member?
- rest
- string->list

Extra Practice Problems

Here are a few additional programming problems that you can use for practice with lists. They are not worth any marks, but they might help you hone your skills if you need it. No automated tests are provided for these problems. You are welcome to get help with them during office hours.

1. Write a function before-tea that consumes a list of symbols and produces the list of symbols that occur before the first occurrence of the symbol 'tea.
2. Write a function after-tea that consumes a list of symbols and produces the list of symbols that occur after the first occurrence of the symbol 'tea.
3. Write a function after-tea that consumes a list of symbols and produces the list of symbols that occur after the *last* occurrence of the symbol 'tea.
4. Write a function sixy that consumes a list of integers and produces a sub-list consisting of those integers that are divisible by 6.
5. Write a function total-string-length that consumes a list of strings and produces the total number of characters in all the strings in the list.
6. Write a function initials that consumes a list of strings and produces a single string formed from the first character of each string in the list. For example, (initials (cons "Computer" (cons "Science" empty))) produces "CS".
7. Write a function how-true that consumes a list of Booleans and produces the difference between the number of true values and the number of false values in the list.
8. Write a function how-true that consumes a list of *any* values and produces the difference between the number of true values and the number of false values in the list, ignoring any values that are not Booleans.
9. Write a function collatz-list that consumes a list of integers and produces the next Collatz number for each integer. That is, if the integer is even, divide by 2; if the integer is odd and larger than one, multiply the number by three and add 1; if the integer is one, it remains as one. For example, (collatz-list (cons 5 (cons 10 (cons 2 (cons 1 (cons 9 empty)))))) produces (cons 16 (cons 5 (cons 1 (cons 1 (cons 28 empty))))).
10. Write a function pair-sum that consumes a list of numbers of length at least two, and computes the list which is the sum of each pair of consecutive numbers. For example, (pair-sum (cons 1 (cons 2 (cons 3 (cons -4 (cons 5 empty)))))) should produce (cons 3 (cons 5 (cons -1 (cons 1 empty)))).
11. Write a function replace-all that consumes a list lst and two values a and b. The function produces a new list that's identical to lst, except that every occurrence of a has been replaced by b. For example, (replace-all (cons 'fish (cons 'dog (cons 'apple (cons 'dog (cons true empty))))) 'dog "pancakes") would produce (cons 'fish (cons "pancakes" (cons 'apple (cons "pancakes" (cons true empty))))).
12. Write a function eval-poly to evaluate a polynomial. A polynomial can be represented as a list of coefficients, from smallest power to largest. The function should consume such a list together with a number x at which to evaluate. For example, (eval-poly (cons 2 (cons 0 (cons 1 (cons 3 empty)))) 5) should produce the number $2 + 0 \cdot 5 + 1 \cdot 5^2 + 3 \cdot 5^3$.
13. Write a function get-divisible that consumes a list of natural numbers lon together with a natural number n. The function produces a new list containing all those elements of lon that are divisible by n.
14. Using the function defined in the previous question, write a function divisible23 that consumes a list of natural numbers lon and produces a new list containing all those elements of lon that are divisible by both 2 and 3.
15. Write a function swap-pairs that consumes a list of even length. The function produces a new list in which consecutive pairs of values been swapped (positions 0 and 1, 2 and 3, 4 and 5, and so on). For example, (swap-pairs (cons 1 (cons "hello" (cons 'apple (cons true empty))))) produces (cons "hello" (cons 1 (cons true (cons 'apple empty)))).
16. Write a function count-char that consumes a string str and a character ch and computes how many times ch occurs in str.
17. Write a function remove-char that consumes a string str and a character ch and produces a new string in which every copy of ch has been removed. For example, (remove-char "Now is the winter of our discontent" #\o) produces "Nw is the winter f ur discntent".
18. Write a function remove-chars that consumes a string str and a second string rm, and produces a new string in which *all* of the characters in rm are removed from str. For example, (remove-chars "Now is the winter of our discontent" "aeiou") produces "Nw s th wntr f r dscntnt". Use the function from the previous question as a helper.
19. Write a function replace-char that consumes a string str together with two characters a and b. It produces a new string in which every occurrence of a has been replaced by b. For example, (replace-char "computer science" #\c #\q) produces "qomputer sqienqe". An earlier function in this list may be useful as a helper.
20. Write a predicate has-char? that consumes a string str and a character ch and determines whether str contains any occurrences of ch. Do not always perform recursion all the way to the end of str: if you discover an occurrence of ch, stop immediately and produce true.
21. Write a function tser that consumes a non-empty list and produces a new list that has all the same elements as the original, except for the last one. For example, (tser (cons 2 (cons 8 (cons -3 empty)))) produces (cons 2 (cons 8 empty)).
22. Write a predicate non-decreasing? that consumes a list of numbers and determines whether the numbers in the list are in non-decreasing order (i.e., each number in the list is greater than or equal to the one before it). Experiment with modifying your function to produce the related predicates strictly-increasing?, non-increasing?, and strictly-decreasing?.
23. In a certain sports league, a team gets two season points for a win, one point for a tie, and zero points for a loss. Write a function standing that consumes a list of symbols, each one of 'win, 'tie, or 'loss, and calculates the total number of points the team has earned for the season. For example, (standing (cons 'win (cons 'loss (cons 'loss (cons 'tie))))) produces 3.
24. Using recursion on lists of characters, write string->nat, which behaves like string->number but only works on natural numbers. For example, (string->nat "5238") produces 5238. This one is a bit challenging!

Previous Next