Coding Manual for Rainfall Study in Racket/Pyret/ML/Java

Kathi Fisler (kfisler@cs.wpi.edu)

Accompanies ICER 2014 paper, "The Recurring Rainfall Problem"

Problem Statement (to be given to students)

Design a program called rainfall that consumes a list of numbers representing daily rainfall amounts as entered by a user. The list may contain the number -999 indicating the end of the data of interest. Produce the average of the non-negative values in the list up to the first -999 (if it shows up). There may be negative numbers other than -999 in the list.

At the top of your file, please include a comment describing your programming experience prior to this course. Do NOT include your name.

Evaluation

In line with the existing studies, we will view this as a plan-composition problem. We will be computing data on (a) which plans students implement correctly, and (b) how they compose the plans.

Plans to Track

(adapted from Ebrahimi's metric, omitting his read and print plans, and adding the filter negative and stop at sentinel plans)

- Sum the (non-negative) numbers (SumPlan)
- Filter out the negative numbers (NegativePlan)
- Stop at the sentinel, if any (SentinelPlan)
- Count the numbers (CountPlan)
- Check division by zero (CheckDivZeroPlan)
- Compute the average (AveragePlan)

Each solution will compose these plans through a combination of sequential ordering ("appended" in Ebrahimi), interleaving, and branching.

Data to code per solution

- student-id : an anonymous id for the student
- school : factor variable
- course : factor variable within schools
- language-used : factor variable
- time-taken : numeric, estimate in minutes (use a+b format if students separated out coding and doc/test times)
- prior-experience : none, (prior | self) (bit | lot) lang [use these tags for whatever they say]
- how many test cases they wrote : number (main/rainfall only, no helpers)
- tests correct against the spec : number of correct test cases
- any tests incl negative nums beyond sentinel : boolean
- code structure summary : string indicating plan-composition style (code list later in doc)

- per required plan
    a. which function contains the plan : rainfall, own, helper, multiple, missing, assumedaway (use "own" if plan is in separate function primarily designed for that plan, such as a separate sum function which may also handle negs and sentinel). Use helper if plan is one of many computations in a function other than "rainfall" (or the "main" function of the solution). "Assumed-away" for cases when stated assumption obviates the plan (such as assuming the list has a positive rainfall amount)
    b. which, if any, built-in function was used to implement the plan
    c. correctness summary : y (meaning correct) or error codes from following, hyphen separated (ie X-B). Use semicolon to separate multiple error codes
        i. category of difference : missing (**X**), misplaced (**P**), malformed (**F**), spurious (**S**), inconsistent with tests (**I**)
        ii. component of difference : init (**I**), base(**B**), update (**U**), guard (**G**), input(**P**), header(**H**)
        iii. extent of difference: full (**F**), partial (**P**) [F is default]
- rainfall Remarks: blank or one of "Recursive", "SentSplit" [cases based on whether sentinel exists in list], "TemplateOnly", "TemplateListed" (but non-template code also provided), "AssumedData" if simplified problem through data assumption, "ExpectedSentinel" if errored on lack of sentinel in list

Coding Examples for Low-Level Task-Implementation Errors

(adapted from Ebrahimi, removing issues that do not arise without I/O or in Racket/Pyret programs)

On components "init" vs "base": "init" refers to the output value in the base case, whereas "base" refers to the guard and structure of the input base case (usually the empty list).

Component "input" refers to the input data (actual parameter) being off – this covers cases where the connections between plans are off (expect to use in conjunction with "malformed" most of the time).

Component "header" refers to the function name and formal parameter list at the top of the function.

The following tables show examples of patterns in the code that correspond to various codes (formed by col-row)

SumPlan : Sum the Numbers

	Plan Components		
Plan differences	**init**	**update**	**guard**
Missing	no init of 0	not incrementing sum	no check for end of list
Misplaced	init sum wrong place	increment outside loop	'() test after first
Malformed	wrong init for sum	wrong sum formula	check other than empty
Spurious	init some other var	update addl var	check more than empty

NegativePlan : Filter Out the Negative Numbers

	Plan Components		
Plan differences	**init**	**update**	**guard**
Missing	no init of empty	not keeping any nums	no check for +/-/0
Misplaced	init ‘() wrong place	keeping in wrong place	check after use
Malformed	wrong init for list	keep neg/elim pos,0	checking wrong item
Spurious	init some other var	update addl var	check more than +/-/0

SentinelPlan : Stop at the Sentinel (if any)

	Plan Components		
Plan differences	**init**	**update**	**guard**
Missing	no init of empty	no non-sentinel data	no check for sentinel
Misplaced	init ‘() wrong place	keeping in wrong place	check after use
Malformed	wrong init for list	keep sentinel or later; lose non-sentinel	wrong check for value
Spurious	init other var	update addl var	check more than sentnl

* use “base” if wrong element is compared to -999 in the base case

CountPlan : Count the Numbers

	Plan Components		
Plan differences	**init**	**update**	**guard**
Missing	no init of 0	no add1	no check end of nums
Misplaced	init count wrng place	add1 in wrong place	check after access
Malformed	wrong init for count	wrong incr formula	check other than ‘()
Spurious	init some other var	update addl var	check more than ‘()

CheckDivZeroPlan : Check Division by Zero

	Plan Components		
Plan differences	**init**	**update**	**guard**
Missing	NA	NA	no check for 0 denom*
Misplaced	NA	NA	check 0 after /
Malformed	NA	NA	wrong check
Spurious	NA	NA	guard beyond /0

* enter “X-G-P” if only missing the guard on some control paths (eg, after cleaning out neg/sentinel)

AveragePlan : Compute the Average

	Plan Components			
Plan differences	**init/base case**	**update**	**guard**	**input**
Missing	NA	avg not computed	NA	function with no input
Misplaced	NA	avg before data ready	NA	wong input position
Malformed	NA	wrong / formula	NA	wrong input to plan
Spurious	NA	more than /	NA	extra input to plan

Coding High-Level Plan Compositions

Legend: & = weave together | = in either order ; = sequential -> = branch/conditional

T = Sentinel N = Negative D = CheckDivZero C = Count S = Sum A = Average

See the spreadsheet "PlanCodes" in the same directory for the codes and their clusterings.

Miscellaneous Notes

- Omitting the negative plan simplifies the CheckDivZero plan, since checking whether orig list is empty or just has the sentinel then covers CheckDivZero. Those who handle negatives potentially need those checks again after filtering out negatives.
- Coding DivZero as correct if init list checked for empty and no cleaned list gets created
- even when students crossed work out, marked it in this assessment (unless crossed out work was replaced by other code for the same function name or computation). Usually happened when students crossed out a start at a function
- left open spec when there were no nums in the list; accepted any handling of that case (error, return 0, etc)
- Repeated sub-compositions (like T&N&C twice) often means calling a helper for the sub-composition multiple times
- Classifying average as correct even if sum is horribly wrong if code suggests that student knew that avg was sum/count
- Used error code P-I when inputs to / swapped in computation of average
- For solutions that started with a cond on whether sentinel is in the list, ignoring that test in the plan code and capturing with "SentSplit" in RainfallCorrect summary instead
- marking a plan as "correct" if its core computation is conceptually correct relative to the other plans. If a plan (like average) correctly divs sum by count, but sum/count are off, average is marked correct and sum/count are marked erroneous as appropriate
- If student assumed only one sentinel, no clear place to record that in the coding (might show up as an input problem to average)
- not recording whether tests were syntactically malformed. Assess tests based on correctness of intent relative to spec
- in rare case of student giving multiple sols, graded first one
- if D occurs before T/N, divzero should not be correct