CS100M Fall 2007

Grading Guide: Project 5

April 14, 2008

The coded items below (e.g., c1e, s2a) indicate what a student's solution should accomplish. Codes that begin with the letter "c" deals with correctness; codes that begin with "s" deals with style.

Grader

If a student's solution does not accomplish task c1a, for example, then write the task code "c1a" along with any diagnostic remarks you can give. Count the number of correctness and style errors separately.

Items marked with ** count as two errors. In the table below, the top row lists the possible scores (1 to 5). The next row lists the number of correctness errors corresponding to every score category. The style score is determined similarly. Enter the total score (maximum of 10) in CMS as the project score. If there are bonus questions, enter any bonus points separately in the "Bonus Bucket", separate from the project score.

Student

Read the grading guide for every project, even if you get a perfect score! Notice from the table below that we often give one or two "freebies", i.e., mistakes that don't cost you any points. Learn from working on the project, and learn from any mistakes.

Scores

- c ands stand for correctness and style; see table below.
- parts with ** next to them means that they are double the value, *** for triple, etc.

Score	5	4	3	2	1	0
#correctness errors	0-2	3-5	6-8	9-12	13-16	17+
#style errors	0-1	2-3	4-6	7-9	10-15	16+

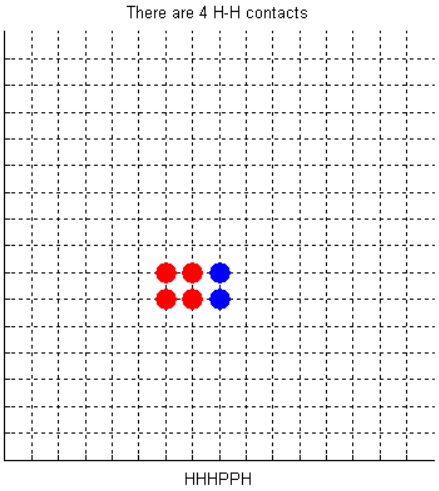
General

- (s0a) Use meaningful variable names
- (s0b) Appropriate indentation
- (s0c) Appropriate and concise comments throughout.
- (s0d) [up to 1* per m-file] Comment header is *required*.
- (s0e) Reasonable line lengths; no horizontal scrolling
- (s0f) [up to **] No superfluous code
- (s0g) [up to **] Reasonably efficient and concise code; a little inefficient is OK
- (s0h) No debugging output

- (c0a) [2* max]Program successfully executes without crashing.(*for occasional,**for persistent)
- (c0b) [up to 1* per m-file] Functions are defined as specified by the project (parameters are of correct type in correct order)

1 Protein Folding

- s1a : Code is neatly organized (easy for reader to understand what happens where). If subfunctions are used, these are appropriately defined. If no subfunctions are used, code is broken up into easily understood segments.
- s1b : Lattice/grid is displayed in an appropriate manner (no axis markings, messages are easily readable, amino acids of different types are easily distinguishable) (see figure below)
- s1c : ALL intended messages are displayed somewhere on the grid, as opposed to the command window.
- s1d : Student has chosen appropriate data structures for the problem (does not have to match the solution).
- c1a : Displays a lattice/grid of size at least 15x15.
- c1b : Accepts a mouse click from the user using function ginput.
- c1c : [up to 4*] Correctly identifies legal/illegal mouse click (1. the placement of the first amino acid is always legal, 2. the placement of any consecutive amino acid must be exactly one square away in a cardinal direction (NESW, no diagonals), 3. must not be on top of an already placed amino acid, 4. a click outside the grid is always considered illegal, but should not crash the program).
- s1e : Rounds mouse clicks to the nearest integer appropriately, so that the user does not have to click exactly on the lattice point.
- c1d : Upon a legal mouse click: places an appropriately colored/shaped/whatever item indicating an amino acid of the appropriate type has been placed there. (For this correctness point, let "legal" mean whatever the student has considered legal, do not double penalize from c1c).
- c1e : Upon an illegal mouse click: displays a message indicating to the user that this is so. (For this correctness point, let "illegal" mean whatever the student has considered illegal, do not double penalize from c1c).
- s1f : This message lets the user know what went wrong and what to do next (something as simple as "illegal click: try again" is ok, though we can encourage more informative error messages!)
- c1f : Indicates which kind of amino acid is next to be placed on the graph. Updates this correctly upon mouse click.
- c1g : Upon placing all amino acids in original string: program no longer accepts clicks (or if it does, displays a message to the effect of "No more amino acids" or something similar). Correctly identifies end of input string.
- c1h : Upon placing all amino acids in original string: Original amino acid string is displayed.
- c1i : Correctly calculates the number of HH contacts. A simple test case: hplayout('HHHPPH') layed out in a clockwise rectangular manner should have 4 HH contacts (see figure below).
- An example figure is displayed below. Your solution may not look exactly like it, but should be as readable as it.



2 From DNA To Protein

- s2a : Code is neatly organized (easy for reader to understand what happens where). If subfunctions are used, these are appropriately defined. If no subfunctions are used, code is broken up into easily understood segments.
- s2b : Student has chosen appropriate data structures for the problem (does not have to match the solution).
- c2a : MakeProtein('pdata8.txt') returns a structure with the following fields: id, def, dna, and amino.
- c2b : Correctly finds the protein id (line one, 13:25). It is ok if the student truncates this, provided they do so correctly. Example test: using MakeProtein('fantasyP.txt') should return 'Somewhere'.
- c2c : [up to 2* - one for correct starting position and one for correct ending position] Correctly obtains the protein definition (def). This occurs beginning at line 2 (13) and continues for a variable number of lines. (May identify end of def "until keyword ACCESSION" or as "while line begins with whitespace", other correct solutions are also acceptable). Make sure the code handles a variable number of lines. You may do so by trying it on pdata8.txt (def: 2 lines - WT1=Wilms' tumor ... 521 nt].), pdata7.txt (def: 1 line - Homo sapiens ... complete cds.), and fantasyP.txt (def: many lines - 686 chars - may vary depending on storage of def).
- s2c : Student should store def in a reasonable manner (best as a single line, but if they keep the line boundaries they should do so in a way that def would be easy to modify/print).
- c2d : Correctly obtains the DNA sequence as all lines after ORIGIN until end of document (marked by //).
- c2e : DNA sequence is stored with no numbers, spaces, or the //.
- s2d : File parsing is done in a reasonably efficient manner - one pass through the file.
- c2f : Correctly translates the DNA sequence into an amino acid sequence. Example test: fantasyP.txt should return a sequence of 80 'P's. (If student has left spaces/numbers as in c2e, do not double penalize).
- c2g : Utilizes the given function conversionTable.m correctly in doing the translation.