**Project 5 Grading Guide**

|  |  |  |  |
| --- | --- | --- | --- |
| **Item #.** | **Description** | **Correctness** | **Style** |
|  | mailMerge | Total = 12 | Total = 6 |
| 1 | readSender, readRecipients, File2Cell and readZIP should be used to read the files named in the arguments (1S, all or nothing). |  | 1 |
| 2 | findZIP is called to find the city information for the sender and recipient (1S) with the first 5 digits of their respective ZIP codes (1C). | 1 | 1 |
| 3 | If the sender ZIP code is not found, an empty cell array should be returned (1C) and the following message is displayed (1S):  ‘Sender ZIP code ([5-digit ZIP code]) does not exist.’ | 1 | 1 |
| 4 | If a recipient ZIP code is not found, its letter should be skipped (1C) and the following message is displayed (1S):  ‘Recipient ZIP code ([5-digit ZIP code]) does not exist.’ | 1 | 1 |
| 5 | Exactly one letter is generated for each recipient (1C). | 1 |  |
| 6 | Substitutions are made to every placeholder in the template, even if multiple placeholders appear on the same line (2C, -1 for each mistake). | 2 |  |
| 7 | The correct fields are used when substituting placeholders (2C, -1 for each mistake). The %sender% and %recipient% substitutions are correctly formatted as multi-line addresses (1S).  **NOTE:** The %% placeholder should be substituted with %. Failure to do so or stating %% is undefined is a single mistake. | 2 | 1 |
| 8 | If an undefined placeholder is found, it is substituted with ‘??’ (1C) and the following message is displayed (1S):  ‘Undefined placeholder: [placeholder name].’ | 1 | 1 |
| 9 | Parts of the template that are not placeholders should be copied exactly into the new letters (1C). Line-breaks in the letters should match those in the template (1C). | 2 |  |
| 10 | All generated letters are returned in a cell array, with each cell containing a single string that is the whole letter (1C). | 1 |  |
|  | MakeZipcode | Total = 2 | Total = 0 |
| 11 | Z should contain the fields ‘ZIP’, ‘city’, ‘state’ and ‘county’ (1C). | 1 |  |
| 12 | The given arguments are put into the correct fields in Z (1C). | 1 |  |
|  | readZIP | Total = 4 | Total = 2 |
| 13 | File2Cell is called on the ZIP code filename to read the zip code data (1S). |  | 1 |
| 14 | The ZIP, city, state and county fields are extracted from each line read (2C, -1 for each mistake). | 2 |  |
| 15 | No trailing spaces in the final city field (1C). | 1 |  |
| 16 | MakeZipcode is called to form a struct from the read fields (1S). |  | 1 |
| 17 | All generated structures are returned correctly in a struct array (1C). | 1 |  |
|  | findZIP | Total = 2 | Total = 2 |
| 18 | A while loop is used to correctly find the position of the matching ZIP code in the given structure array (1C, 1S). If a for loop is used that always iterates to the end of the array, award 1C if the correct position is found but deduct the 1S point for inefficiency. | 1 | 1 |
| 19 | Comparisons of ZIP strings are correctly handled in one of two ways (1S):   1. str2double is used to convert ZIP strings to doubles and then == or ~= is used. 2. strcmp is used on ZIP strings.   Do not accept direct use of == or ~= on strings. |  | 1 |
| 20 | The structure at the correct position in the array or the empty vector is returned as appropriate (1C). | 1 |  |
| GENERAL |  |  | Total=10 |
| G1 | Script starts with a concise comment describing the program.  Function comment follows function header. |  | 1 |
| G2 | Code is sufficiently (but not excessively) commented. |  | 1 |
| G3 | Line lengths are not excessively long (80 columns).  **NOTE**: It's ok if a couple lines are a little too long, especially if it’s due to having to print a very long string**.** |  | 1 |
| G4 | No extra output (debugging output) produced |  | 1 |
| G5 | Proper indentation is always used. |  | 1 |
| G6 | Use meaningful variable names. Do not overwrite MATLAB keywords. |  | 1 |
| G7 | Name important parameters as variables (constants). |  | 1 |
| G8 | No superfluous code (e.g., an empty if or else branch or a useless loop). Of course some students will have code that is awkward or unclear or inefficient. This point is specifically for not having code that does literally nothing. |  | 1 |
| G9 | Reasonably efficient code. |  | 1 |
| G10 | Does NOT put semicolon at wrong places, e.g., at the end of these lines: "if", "elseif", "else"," for","while", "function". |  | 1 |
| TOTAL |  | 20 | 20 |

**Penalties**

|  |  |  |
| --- | --- | --- |
| P1 | Student's code does not execute (or student provides a script when a function is required and vice-versa) | -1 from final score |
| P2 | Student's code crashes or does not terminate (infinite loop) for normal cases. | -1 from final score |
| P3 | All function headers and file names match those specified in the project description exactly. All input and output variables should be of the correct type. | -1 from final score |

**Grade Calculation**

|  |  |
| --- | --- |
| Total Possible Correctness Points | TC = 20 |
| Total Possible Style Points | TS = 20 |
| Student Correctness Points | C = min( \_\_\_ + 1 freebie point, TC) |
| Student Style Points | S = min( \_\_\_ + 1 freebie point, TS) |

Exceptions: If any file is missing/unacceptable, no freebie points can be applied to that file and subtract 3 style points for each missing/unacceptable file.

|  |  |
| --- | --- |
| Student’s final score | ([(C/TC)+(S/TS)] X 5) - Penalties  (Out of 10; 1 decimal; no negative score; round to NEAREST) |