**T2W3 – Practical Revision Worksheet**

Please submit your solutions in Coursemology.

Use Python comments to indicate the start of each task, eg #Task 1, #Task 2 …etc

**1** John finds a collection of old text files with the following documentation.

|  |  |
| --- | --- |
| **File** | **File Data** |
| CHARACTERS.TXT | Saved character data in the following format:  <NAME>, <CLASS NAME>, <STR>, <DEX>, <INT>, <CURRENT XP>  <ITEM 1 ID>, <ITEM 2 ID>, …, <ITEM N ID>  Thus, the data for each character is saved on 2 lines of this text file. |
| ITEMS.TXT | Saved item data in the following format:  <ITEM ID>, <ITEM NAME> |

However, all data is stored in Hexadecimal format (including formatting data – i.e., commas, but excluding end line characters). For example, a string: "ABC,1,15" would be stored as: "41 42 43 2C 31 2C 31 35". With the conversion performed as follows.A

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CHARACTER** | A | B | C | , | 1 | , | 1 | 5 |
| **ASCII (DEC)** | 65 | 66 | 67 | 44 | 49 | 44 | 49 | 53 |
| **ASCII (HEX)** | 41 | 42 | 43 | 2C | 31 | 2C | 31 | 35 |

|  |
| --- |
| **Task 1.1**  Write the function hex2Str(data), that returns the original string when given its hexadecimal representation, data.  **Evidence 1.1**  The code for the function hex2Str(data). **[4]** |

|  |
| --- |
| **Task 1.2**  Write the code to read and store the contents of the file ITEMS.TXT. The data stored should be decoded using hex2Str(data) such that the original data is stored – i.e., store the original strings. Your code should also ensure that each element of data is stored separated – i.e., ITEM ID and ITEM NAME are stored separately for each entry.  **Evidence 1.2**  The code to read and store the contents of the file ITEMS.TXT as specified in Task 1.2. **[3]** |

|  |
| --- |
| **Task 1.3**  Write the function mergeSort(items), which takes the item data stored in Task 1.2, items, and sorts it by ITEM ID using the merge sort algorithm.  **Evidence 1.3**  The code for the function mergeSort(items). **[5]** |

|  |
| --- |
| **Task 1.4**  Write the code to read and store the contents of the file CHARACTERS.TXT. Note that ITEM IDS should be stored as an array of integers.  **Evidence 1.4**  The code to read and store the contents of the file CHARACTERS.TXT as specified in Task 1.4. **[3]** |

|  |
| --- |
| **Task 1.5**  Write the code that requests a character name from the user, and then uses the stored data (from CHARACTERS.TXT and ITEMS.TXT) to search for the requested character. If the character is found, the characters statistics and list of items are printed to the terminal, else if the character is not found, then a simple message stating that the character has not been found is instead printed.  This program should repeatedly loop until some exit character or string is input.  **Evidence 1.5**  The code corresponding to Task 1.5. **[5]** |

|  |  |  |
| --- | --- | --- |
| **2** | Mike would like your assistance in implementing a particular Hash Table. You are NOT to implement this Hash Table using object-oriented programming.  The Hash Function:  Mike would like this Hash Table implementation is to store STRINGs. To compute the Hash Values of these STRINGs, you are to use the following Hash Function:  CHAR-1.ASCII \* 1 + CHAR-2.ASCII \* 2 + CHAR-3.ASCII \* 3 + … + CHAR-N.ASCII \* N,  where there are N CHARs in the given STRING, and **CHAR-1** refers to the **right-most** CHAR in the STRING, while **CHAR-N** refers to the **left-most** CHAR in the STRING. For example, the Hash Value for the STRING “ABC” would be 67 \*1 + 66 \* 2 + 65 \* 3.  You may assume that all strings stored in this Hash Table will not be empty. |  |
|  | **Task 2.1**  Write the code to implement the function strHash(data), where data corresponds to a STRING. This function returns an INTEGER value corresponding to the Hash Value for the parameter data. This function should work as described above.  **Evidence 2.1**  The code for the function strHash(data). | [2] |
|  | Having implemented the Hash Function, your next step is to implement the Hash Table itself. To do this, you should implement to the following:   * Use a global array, hashTable, to store the contents of the Hash Table, and a global INTEGER, size, to store size of the Hash Table. * An insert(data) function, which insert a given STRING, data. Note that you should use the Hash Function to perform this insertion. * Utilise the Linear Probing method for collusion resolution. * A search(data) function, which returns the index of the STRING, data, within the Hash Table, or else returns -1 if the object cannot be found. * A delete(data) function to delete the given STRING, data, from the Hash Table. This function will return TRUE if the STRING is successfully deleted, or else returns FALSE.   You should practice modularity in your implementation to ensure maximal code-reuse. |  |
|  | **Task 2.2**  Write the code to the Hash Table as described above.  **Evidence 2.2**  The code for the Hash Table. | [10] |
| **3** | The file LISTINGS.CSV contains a directory listing of all files on a computer.  Each row contains the file name, last modified time stamp and the size of the file.  **Task 3.1**  Write Python code to read the contents of the file and change the time stamps on the file listings such that the last modified time stamp of each file is 100 days from the original time stamp.  Create an output file LISTINGS\_UPDATE.CSV such that the modified file listings are stored in descending order of time stamps, ie most recently updated files first.  **Evidence 3.1**  Python code | [6] |