A certain university has decided to get in on the blockbuster film game by creating a set of inter-connected movies which will be collectively called the HOTNCU (Harvard of the North Cinematic Universe). Each movie will focus on the gripping adventures of one or more super-heroes who happen to be students, staff or faculty at the mysterious University Q, situated in the far-away and not-cold-at-all land of Notario. (The University Administration is very proud of having come up with this clever disguise for the actual setting of the movies.) The projected number of movies in the series will be at least 2000 and not more than 2500.

Each movie under development has been assigned a project code name to preserve secrecy. Each code name is an 8-letter English word. A sample set of code names is provided in the file HOTNCU_codenames_2018.txt. There is no way this is the actual set of movie code names, no, not at all.

You have been assigned the task of creating a data structure that can

- contain up to 2500 items
- support insert and search operations
- provide access to each item with an average of \( \leq 5 \) steps (that is, the average number of table addresses examined during an insert operation can be no more than 5). For more information on this, see below.

Your hard-earned data structures expertise has convinced you that neither a sorted array nor a binary tree can meet this requirement, so you have settled on using a hash table.

The HOTNCU Project Director was previously a Computer Science professor and she has taken an interest in your project. She has already decided that you are required to use some form of open addressing. She is aware that your table will need to be \( > 2500 \) in size but she wants you to try to minimize it.

She wants you to explore at least two forms of open addressing: quadratic probing and double hashing. For each method she wants you to experiment with different hashing functions and details of the collision resolution methods to determine a table size that lets you achieve the required performance standard. See below for a discussion of how to compute the necessary information.
The Director has also set you an interesting challenge: if you can achieve the required performance level with a table size no greater than 5000 she will promote you to CCH (Chief Code Head). (Note: this has no bearing on your grade for this assignment.)

Part 1:

Decide how you will convert the code names into usable key values. This may involve converting each code name to an integer, or simply treating each code name as a bit string. You will also find a wealth of ideas on the Internet. Whatever method you decide on, explain why you chose it and remember to cite your source if it is not your own creation.

You may wish to take advantage of the fact that all the code names have exactly 8 characters.

Part 2:

Implement a hash table where collisions are resolved by quadratic probing.

Use an \( h'(k) \) hashing function of your own choice. You must implement the algorithm yourself. Using downloaded code from external sources is not permitted – but writing your own code based on a published algorithm is fine (remember to cite your sources). You may wish to experiment with different \( h'(k) \) functions to minimize the number of collisions.

Try at least three combinations of \( c_1 \) and \( c_2 \):

1) \( c_1 = 1, c_2 = 1 \),

2) \( c_1 = 2, c_2 = \frac{1}{2} \) (remember that you will have to convert the result to an integer),

3) other combinations of your choice.

For each combination, find a table size that lets you achieve the requirement on average probe sequence length. Use experimentation to get close to the minimum table size that satisfies the requirement.

A table size must be rejected if there is any code name in the set for which the insert operation simply fails on that table size.
Part 3:

Repeat Part 2 but with Double Hashing instead of Quadratic Probing. Try at least three combinations of $h'(k)$ and $h''(k)$. For each combination, find a table size that achieves the required performance.

You are free to choose any hashing functions you like for $h'(k)$ and $h''(k)$, but as with Part 2 you must implement them yourself.

Part 4:

Discuss the results of your experiments. Do they support the hypothesis that Double Hashing allows us to use smaller tables than Quadratic Probing does, when trying to achieve a particular level of performance?
Computing the Average Number of Steps

Every time your program looks at the content of a table address, that counts as a step. So if you are inserting a value and you try addresses 17, 5, and 83 before finally inserting the value in address 36, that counts as **four** steps.

Since we don't know which keys are most likely to be searched for, we can assume that each key is equally likely to be the target of a search. This means that the average number of steps in a search operation will be exactly the same as the average number of steps in an insertion. To compute that average we can add up the number of steps made during all the insertions and divide by the number of values that were inserted.
Logistics:

You may complete the programming part of this assignment in Python, Java, C or C++. You must submit your source code, properly formatted and documented. You must also submit a PDF file summarizing the results of your experiments and containing your conclusions. All files must contain your name and student number, and must contain the following statement: “I confirm that this submission is my own work and is consistent with the Queen’s regulations on Academic Integrity.” Combine your files into a .zip archive for uploading.

You are required to work individually on this assignment. You may discuss the problem in general terms with others and brainstorm ideas, but you may not share code. This includes letting others read your code or your written conclusions. The course TAs will be available to advise and assist you regarding this assignment.

The due date for this assignment is 11:59 PM, March 16, 2018.