Assignment 1: Hash Tables

CS3D5A, Trinity College Dublin

Deadline: 22:00 12/10/2022

Grading: The assignment will be **graded partly automatically, partly by the demonstrators** looking at your code, and via your answers to the questions **during the lab hours** on **15/10/2021.**

Note that some of the automatic tests are used merely to see that your code is working and to see its performance - if you have better performance than the test, even though the test will fail the demonstrators will give you the marks!

Submission: **Submit via submitty.** Include a separate .c file for each task, and **the short assignment report in pdf**, detailing your results for each part.

Goals:

- Learn how to implement a hash table in C
- Consider how to choose a hash function
- Learn how to evaluate the performance of a hash table based on collisions
- Use a hash table to extract information from a dataset of historical figures

Task 1: Getting Started - 5 Marks

For the first part of this assignment, you should write a hash table to store **the frequency** of names in an (unordered) list of Irish surnames. It will therefore use `char` arrays as keys and store `int` as values.

Use the hash function `hash_function` that adds the integer value of the `chars` in the string:

```
int hash_function(char* s){ int
    hash = 0; while(*s){
        hash = hash + *s;
        s++;
    }
    Return (hash % ARRAY_SIZE);
}
```

You should use a linear probing strategy to handle collisions. For simplicity, you do not need to worry about a dynamically growing hash table. You can use an array to store each key (assuming that there is a known MAX_STRING_SIZE) as well as its count, and a fixed size for the hash table (ARRAY_SIZE) and focus on implementing hashing and linear probing.
A skeleton implementation is provided, though its use is completely optional (if you can, try to avoid using it as it makes the assignment more challenging!). If you do not use the skeleton, or use it but improve it, document this in your report.

Set up your program so that it takes a CSV file as an argument and test your hash table by loading the test data provided in names.csv. Extend the solution to take in a name as input from a user and returning its frequency.

NB: Load the data using a cvs parser (yours from assignment 0 or the one in the solution on blackboard – see skeleton if required).

Update your code to count the number of collisions obtained and display it.

Upload your code in Task 1 on Submitty.

Listing 1: Sample output

```
names.csv loaded!
        Capacity   : 59
        Num Terms  : 42
        Collisions : 46
        Load       : 0.71%
Enter term to get frequency or type "quit" to escape
>>> Synnot
Synnot 2
>>> bleb
bleb not in table
>>> quit
```

Task 1 Mark Allocations	
Correct implementation of hash table	4 marks
Take a string as input and print the number of times the string occurred in data	1 mark

Task 2: Choosing a hash function - 3 Marks

Now find a better hash function `hash2` for the data considered. Feel free to consult online resources, but make a reference to them in your report. Note, the sample data is only a sample! Do not overfit your function to the sample provided, it should work well with any lists of Irish surnames. Justify your choice of hash function in your report (half a page max). Test your function on the sample data, indicate in your report how many collisions occur, is it better than the result from task 1? (Note that it is expected that you will get less collisions than the automatic tests here!)

Upload your task 2 on Submitty.

Task 2 Mark Allocations	
Justify your choice of hashing algorithm	1 mark
Implement and evaluate your hashing algorithm	2 marks

Task 3: Twice the Fun - 2 Marks

Using the hashing function `hash3` below, augment your solution to Task 1 such that it uses double hashing instead of linear probing. Use this to demonstrate the improvement of double hashing over linear probing. Document in the report whether hash3 is a good choice to implement double hashing and explain why.

Upload your task 3 on submitty.

```c
int hash3(char* s){
    int hash = 0;
    while(*s){
        hash = 1+ (hash + *s) % (ARRAY_SIZE-1);
        s++;
    }
    return hash;
}
```

Listing 2: Sample output. Input from names.csv

```
File names.csv loaded
 Capacity: 59
 Num Terms: 42
 Collisions : 22
 Load: 0.711864
Enter term to get frequency or type "quit" to escape
>>> Stafford - 4
>>> Murphy - 0
>>> Wagstaffe - 2
>>> Doe - 0
>>> sghhj - 0
>>>
```

Task 3 Mark Allocations	
Successful implementation of the second hashing function	2 marks

Task 4: A More Interesting Application - 5 Marks

Hash tables have numerous applications in computer science. One domain where they can find great use is that of Information Retrieval. Search terms extracted from a collection of documents are used as the keys and lists of documents are stored as values in the table. When a user issues a query, the list of all documents which may be of interest to them can be rapidly retrieved, ranked and presented.

For this task you have been given a file containing a list of people. The data which you have been given is real data produced with great effort and expense by a number of Trinity historians. The people mentioned are all individuals who were in some way involved with the 1641 Irish rebellion. Learning about these people is of great interest to historians, but the challenging nature of working with 17th century data can make this difficult.

For this task you should expand (a copy of) your solution to the previous tasks. Instead of storing term counts at each index, you should store a list of people with a given surname. The keys of the hash table will be surnames. The values will be linked lists of people.

As before you should provide a way to search for information in the hash table. Allow a user to enter a surname and get a list of people with the given surname as a search result. Test on the truncated data and then on the full dataset.

Upload task 4 on Submitty – briefly document on the report your approach and the results you achieved.

Remember to submit your report on Submitty also!

Listing 3: Sample output.

```
File people.csv loaded!
 Capacity : 99991
 Num Terms : 14963
 Collisions : 333666226
 Load : 0.150
Enter term to get frequency or type "quit" to escape
>>> Person ID Deposition ID   Surname   Forename    Age Person Type  Gender  Nationality Religion Occupation
      509    815275r350       Wagstaffe Thomas      0   Victim       Male    Unknown    Unknown   Unknown
      508    815275r350       Wagstaffe Elizabeth   0   Deponent     Female  Unknown    Unknown   Unknown
>>> Person ID Deposition ID   Surname   Forename    Age Person Type  Gender  Nationality Religion Occupation
    32290    830138r107       Doe       Morroghoe   0   Unknown      Unknown Unknown     Unknown  Unknown
>>> sghhj not in table
>>>
```

Task 4 Mark Allocations	
Alter (a copy of) your previous solution so that it now stores lists of people as values instead of word counts. This alteration should include freeing any dynamically allocated memory required for the lists.	3 marks
Provide a means to search across your hash table of people	2 marks

Extra Bits - Not Graded

- As previously mentioned, the people in this dataset are real people. Do a search for your family name and see if anything comes up. If you find someone you would like to read about, the text of the depositions is available at https://1641.tcd.ie/.
- You may have noticed that some people's names are spelled "wrong", e.g. Maguire spelled "Magwire". The English language was not standardised until some time in the mid-18th century. What you are seeing is the early Anglicization of Irish names coupled with the fact that the rules of the English language were not yet well established. Can you think of a way to provide more tolerant search across the hash table so that people searching for "Maguire" might also get results for "Magwire"? You can document this in your report.