**Assignment 1**

**Report**

Through this report, I am documenting my approaches towards the problems that I faced while solving the Assignment 1. This report is divided into three sections, for the three tasks.

**Task 1:**

**Description**

This task required us to parse the names.csv file and store it in a hash table with linear probing and calculate the number of unique terms and the number of collisions.

**Approach**

I used the parser provided in the skeleton code, since the parser from my previous code was continuously giving me segmentation faults and bus errors. I did not create a file loader function and instead, loaded the file in the main function.

To calculate the number of collisions, I created a global variable for it and simply incremented it by 1 every time I faced a non NULL pointer in the hash table while inserting a name.

To calculate the total number of terms, I simply created a count\_terms() function, that looped over the hash table and incremented the total number of terms every time it hit a non NULL pointer. This was achieved in O(n) complexity.

**Problem Faced**

I faced multiple segmentation errors while copying the name using strcpy(), as read by the parser, onto the name property of my Element.

**Solution**

I noticed that I wasn’t allotting memory to the name attribute of my Element before copying some text into it. Hence I modified my code to first allocate memory to name attribute using malloc() and then copy any text into the name attribute.

**Task 2:**

**Description**

This task required us to come up with our own Hash Function to reduce the number of collisions.

**Approach**

I came up with a very simple hash function. It added the ascii value of every letter and/or any spaces in the name, and mod it with the size of the hash table.

With this, the number of collisions reduced from 39 to 17.

**Task 3:**

**Description**

This task required us to do double hashing.

**Approach**

I used the hash3 function given in the assignment. Combining it with the hash1 function, I computed the hash value using Hash(k) = (Hash1(k) + j \* Hash3(k)) mod (table\_size), where j is an iterator that starts from 0 and increases by 1 after each collision.

This reduced the number of collisions from 39 to 25.