# SENG1050 – DATA STRUCTURES

## MAJOR ASSIGNMENT 2 - COMPARE SORTED LINKED LISTS WITH HASH TABLES

### OVERVIEW

- For this assignment, you will compare the efficiency of searching a sorted linked list and a hash table. You should use a variation of your code for Focused Assignment 3 for this.

### OBJECTIVES

- Create and use linked lists.
- Create and use hash tables.
- Explain about the differences in efficiency amongst different algorithms.
- Use common algorithms to enhance software development.
- Use best practices to effectively produce quality software.

### ACADEMIC INTEGRITY AND LATE PENALTIES

- Link to Academic Integrity Information
- Link to Late Policy

### EVALUATION

- The evaluation of this assignment will be done as detailed in the Marking lecture in Week 2 of the C course.

### PREPARATION

- Understand how to implement sorted linked lists.
- Understand how to implement hash tables.

### REQUIREMENTS

#### Changed Requirements

- Instead of getting words to load into the hash table from the user, you must load at least 2000 names from a file called names.txt as input (using file I/O). Names will have spaces within the strings. It is OK if you share the names with classmates for this. Make

sure that your names are in random order (you can use Excel for this). The names.txt file must be in the same directory as your source code. All names should be in lowercase.

- It's OK to use more than 2000 names but don't have more than 10000.
- The hash table's buckets must point to sorted linked lists. When you load a name into the hash table, also load the names into one very long sorted linked list.
    - You must call the same function for inserting into the sorted linked list as for inserting into the hash table bucket's linked list. Do not duplicate code for linked list insertion.
        - This implies that you must be consistent with whether you are using a singly-linked or doubly-linked list in both the standalone list and the hash table bucket.
        - Since I'm not mandating whether you use a singly-linked or doubly-linked list, your results might be different from any sample executable that is provided.
- Make the hash table 127 buckets in size.
- You must create a function called searchLinkedList() that searches a sorted linked list for a name and returns a pointer to the node containing the name (if found) or NULL.
    - This function must return NULL immediately after you pass the point in the linked list where the name would have been found if it was in the linked list.
    - The function takes three parameters:
        - char * searchName (or you can use a string object): name to search for (entered by the user)
        - struct linkedList * linkedList: linked list to search (in your program, you can call the linked list node struct anything that makes sense)
        - int * comparisonCount: pointer to int filled in by the function with the count of strcmp comparisons done in searching the linked list
    - Again, you can assume that all names will be in lowercase.
    - Unlike Focused Assignment 3, do **not** display the words that you compare against in the search.
- Create a search function called searchForNameTwice(). It returns nothing and will have the following parameters:
    - char * searchName (or you can use a string object): name to search for (entered by the user)
    - struct linkedList * linkedList: linked list to search
    - struct linkedList * hashTable[]: hash table to search
    - int comparisonCount[2]: array containing the count of strcmp comparisons done in searching the extremely-long sorted linked list (element 0) and in searching the hash table (element 1)

    - It must call your linked list search function. It then displays one of the following messages once the search is done:
        - "name was found in the list in number comparisons", where name is the name being searched for, list is either "linked list" or "hash table bucket" and number equals the number of times that strcmp was called
        - "name was NOT found in the list in number comparisons"
        - You will use this search function to search the hash table bucket and the sorted linked list.
- Don't worry about the grammatical inconsistency of possibly displaying "1 comparisons".
- Indent the message displayed by one TAB ('\t').
- Once you are finished the loop, display the total number of searches, the total number of comparisons done by searching the sorted linked list, and the total number of comparisons done by searching the hash table.
- Design your linked list code so that you do not have to duplicate code unnecessarily. This is very important.
- Clean up all allocated memory before exiting.

## PROOF OF COMPARISON REQUIREMENTS

- Create a JPG file called compare.jpg that contains a screenshot of your program running with a full screen of sample output (so that I can see the difference in comparison count between the methods for at least two successful and three unsuccessful searches (do not skimp on the number of searches)). Make sure that it includes the final summary output. Put this file in the top directory of your project (so that it is submitted with your submission). A sample of this file is provided for you. Please follow the output format and contents exactly. Not providing this file as required could produce an automatically-failing mark in this assignment.list.

## OTHER REQUIREMENTS

- The field containing the names must be dynamically allocated to an appropriate size as in Focused Assignment 1.
- Do not get user input except as indicated in these requirements.
- Do not display output except as indicated in these requirements.
- Do not clear the screen at any time in this program.
- You must do error checking where necessary.
    - If you detect an error, you should display an appropriate error message and take appropriate action.
    - You can assume that all statements above that have the phrase "will be" in them will be true when testing is done.

- It must not use global variables.
- It must not use goto.
- Be aware that you still cannot use C++ strings in this assignment because malloc() does not support them.
- The SET Coding Standards must be adhered to.

## GIT REQUIREMENTS

- Use GitHub Classroom for revision control, similar to Focused Assignment 1.
- It is expected that you have a reasonable number of commits with meaningfully descriptive commit comments.

## CHECKLIST REQUIREMENTS

- Create a requirements checklist. This should contain the specific requirements from this assignment as well as any relevant requirements that have been covered in lecture or that are found in the SET Coding Standards or SET Submission Standards. Do it in whatever form you wish. Hand in your completed checklist in PDF form as checklist.pdf. Not having this checklist will result in a cap of 80 on your mark.

## FILE NAMING REQUIREMENTS

- You must call your source file m2.cpp.
- You must call your checklist checklist.pdf.
- You must call your screenshot file compare.jpg as previously stated.

## SUBMISSION REQUIREMENTS

- Do not hand in any other files.
- Submit your files to the *DS: Major Assignment 2* Assignment Submission Folder.
- Once you have submitted your file, make sure that you've received the eConestoga e-mail confirming your submission. Do not submit that e-mail (simply keep it for your own records until you get your mark).

## HINTS

- I will have my own names.txt file for testing.
- Unit testing your functions is a really good idea. You should not leave your unit testing code in your submitted source.
- Don't forget that malloc() does not support C++ string objects so you must use new instead if you have strings as part of a dynamically-allocated struct or class.

- You can use the examples provided on the DS website and C website. You must attribute credit for the djb2 hash function (credit the real author Dan Bernstein and state that you got it from lecture).