INFORMATION
TECHNOLOGY
UNIVERSITY

Software Engineering Department - ITU
SE200L: Data Structures & Algorithms Lab

Course Instructor: Usama Bin Shakeel	**Dated: 30/10/2024**
Teaching Assistant: Zainab Bashir & Ryan Naveed	**Semester: Fall 2024**
Lab Engineer: Sadia Ijaz	**Batch: BSSE2023B**

Lab 8. Hashing

Name	**Roll number**	**Report (out of 35)**

Checked on: ______________________________

Signature: ___________________________________

1.1 Objective

The objective of this lab is to practice problems related to hashing..

1.2 Equipment and Component

Component Description	Value	Quantity
Computer	Available in lab	1

1.3 Conduct of Lab

1. Students are required to perform this experiment individually.
2. In case the lab experiment is not understood, the students are advised to seek help from the course instructor, lab engineers, assigned teaching assistants (TA) and lab attendants.

1.4 Theory and Background

A **hash table** is a data structure that implements an associative array, mapping keys to values efficiently. It provides an average-case time complexity of O(1) for insertion, deletion, and search operations. The core principle of hash tables is a **hash function**, which transforms a key into an index to determine where the corresponding value is stored in the array.

Key Concept:

1. **Hash Function**:
   - A hash function takes an input (or 'key') and returns an integer (the hash value) that maps to a specific index in the hash table. An effective hash function minimizes collisions and uniformly distributes entries across the table.
   - The formula is: $index = key \, \% \, tableSize$
2. **Collisions**:
   - Collisions occur when two keys hash to the same index. Managing collisions is crucial for hash table efficiency.
   - Common strategies include:
     - **Separate Chaining**: Each bucket holds a linked list of entries that hash to the same index, adding new entries to the list upon collision.
     - **Open Addressing**: If a collision occurs, the algorithm searches for the next available index in the array.
3. **Load Factor**:
   - The load factor is the ratio of the number of entries to the number of buckets (table size). It significantly impacts performance; a higher load factor increases the likelihood of collisions, while a lower load factor ensures more empty buckets, enhancing performance but using more memory.

Lab Tasks

Task 1:

Implement the **Node** class, Represents a single entry

Data Members:

- **int value**: The value stored in the node.
- **Node* next:** Pointer to the next node in the list.

Member Functions to Implement:

- **Node(int v):** Initializes the node with the given value and sets the next pointer to nullptr
- **~Node():** Destructor to clean up if needed.
- **int getValue():** Returns the value stored in the node.
- **Node* getNext():** Returns the pointer to the next node in the list.
- **void setNext(Node* nextNode):** Sets the pointer to the next node.

Implement the **List** class.

Data Members:

- **Node* head:** Pointer to the first node in the linked list.
- **int size:** The number of nodes in the linked list

Member Functions to Implement:

- **List():** Initializes the linked list with head set to nullptr and size set to 0
- **~List():** Destructor to clean up and deallocate all nodes in the linked list.
- **void append(int value):** Inserts a new node with the specified value at the end of the linked list.
- **bool remove(int value):** Removes the first node that contains the specified value. Returns true if the node was found and removed; otherwise, returns false
- **bool search(int value):** Searches for a node with the specified value. Returns true if found; otherwise, returns false.
- **int getSize():** Returns the number of nodes in the linked list.
- **void display():** Displays the values of all nodes in the linked list.

Task 2:

Class: <u>HashTable</u>

Data Members:

- **List* table:** An array of List objects representing the hash table, where each index in the array corresponds to a bucket.
- **int tableSize:** The size of the hash table (number of buckets).
- **int elementCount:** Count of current elements in the hash table.

Member Functions to Implement:

- **HashTable(int size):** Initializes the hash table with a given size. Allocates memory for the table array and initializes each List at each index.
- **~HashTable():** Destructor to clean up and deallocate memory for all List objects in the hash table, ensuring proper memory management.
- **int hashFunction(int key):** Computes the hash value for a given key using the modulo operation with tableSize. This determines the index at which the key-value pair will be stored.
- **void insert(int value):** Inserts a key-value pair into the hash table. If a collision occurs (the bucket is already occupied), it adds the new node to the corresponding linked list in that bucket. utilizes hashFunction to store value.
- **bool remove(int key):** Removes a key-value pair from the hash table. If the key is found in the corresponding linked list, it deletes the node and returns true. If the key is not found, it returns false
- **bool search(int key):** Check whether a given key exists in the hash table by using the hash function to find the appropriate index in the array and then searching through the list at that index.
- **int getSize():** Returns the number of key-value pairs currently stored in the hash table.
- **void display():** Displays the contents of the hash table, showing each bucket (index) and the linked list of nodes stored in it.

Please read the following instructions carefully:

1. ***Do Not Modify test.cpp:*** *You are strictly prohibited from making any changes to the test.cpp file. This file is designed to test your implementation and any modifications will lead to the assignment being graded as zero.*
2. ***Class Definitions:*** *All class definitions and implementations must be provided solely within the files functions.h and functions.cpp. You are not allowed to create any additional files for your class definitions or implementations.*

Any deviation from these rules, including creating additional files or modifying the test.cpp file, will result in your assignment receiving a grade of zero.

Assessment Rubric for Lab

Performance metric	**CLO**	**Able to complete the task over 80% (4-5)**	**Able to complete the task 50-80% (2-3)**	**Able to complete the task below 50% (0-1)**	**Marks**
1. Realization of experiment	1	Executes without errors excellent user prompts, good use of symbols, spacing in output. The testing has been completed.	Executes without errors, user prompts are understandable,minimum use of symbols or spacing in output. Some testing has been completed.	Does not execute due to syntax errors, runtime errors, user prompts are misleading or non- existent. No testing has been completed.	
2. Conducting experiment	1	Able to make changes and answer all questions.	Partially able to make changes and few incorrect answers.	Unable to make changes and answer all questions.	
3. Computer use	2	Document submission timely.	Document submission late.	Document submission not done.	
4. Teamwork	3	Actively engages and cooperates with other group member(s) in an effective manner.	Cooperates with other group member(s) in a reasonable manner but conduct can be improved.	Distracts or discourages other group members from conducting the experiment	
5. Laboratory safety and disciplinary rules	3	Code comments are added and do help the reader to understand the code.	Code comments are added and do not help the reader to understand the code.	Code comments are not added.	
6. Data collection	3	Excellent use of white space, creatively organized work, excellent use of variables and constants, correct identifiers for constants, No line-wrap.	Includes name, and assignment, white space makes the program fairly easy to read. Title, organized work, good use of variables.	Poor use of white space (indentation, blank lines) making code hard to read, disorganized and messy.	
7. Data analysis	4	Solution is efficient, easy to understand, and maintain.	A logical solution that is easy to follow but it is not the most efficient.	A difficult and inefficient solution.	
Total (out of 35):					