**Institute for Advancing Intelligence, TCG CREST**
(TCG Centres for Research and Education in Science and Technology)

# Introduction to Programming and Data Structures
Ph.D. Coursework: First year, First Semester (Session: 2024-25)
**Assignment #08**

Full Marks: 200 | Instructor: Dr. Laltu Sardar
Clarification Deadline: **2024-Nov-14** | Submission Deadline: **2024-Nov-17**

Instructions:

1. The program should be as fault-tolerant as possible, handling potential input errors gracefully.
2. The test file should include a menu, and input matrices must be provided via files only.

## Problem #0701: Hash Table

Implement each hash table operation in C as described below.

1. **Insert into the Hash Table**
   Insert a key-value pair into the hash table. If the key already exists, update its value.

   **Function Definition:**

   - `void insertHashTable(HashTable *table, int key, int value);`
   - **Inputs:**
     - `table`: Pointer to the hash table.
     - `key`: Integer key to insert.
     - `value`: Integer value associated with the key.
   - **Output:** Updates the hash table with the new key-value pair.

   **Example:**

   - **Input:** Insert key = 5, value = 20 into a hash table with existing keys [2, 4, 7].
   - **Output:** Hash table contents after insertion.

```
Key: 2, Value: 15
Key: 4, Value: 10
Key: 5, Value: 20
Key: 7, Value: 25
```

2. **Search for a Key in the Hash Table**
   Search for a key in the hash table and return its associated value if found.

   **Function Definition:**

   - `int searchHashTable(HashTable *table, int key);`
   - **Inputs:**
     - `table`: Pointer to the hash table.
     - `key`: Integer key to search for.
   - **Output:** Returns the value associated with the key if found, otherwise returns -1.

   **Example:**

- **Input:** Search for key = 4 in a hash table with keys [2, 4, 7].
- **Output:** `Value found: 10`

3. **Delete a Key from the Hash Table**
   Remove a key from the hash table. Ensure that the hash table remains functional after deletion, especially if using linear probing.

   **Function Definition:**

   - `void deleteHashTable(HashTable *table, int key);`
   - **Inputs:**
     - `table`: Pointer to the hash table.
     - `key`: Integer key to delete.
   - **Output:** Updates the hash table after removing the key.

   **Example:**

   - **Input:** Delete key = 4 from a hash table with keys [2, 4, 7].
   - **Output:** Hash table contents after deletion.

     ```
     Key: 2, Value: 15
     Key: 7, Value: 25
     ```

4. **Display the Hash Table Contents**
   Traverse and print all key-value pairs in the hash table.

   **Function Definition:**

   - `void displayHashTable(HashTable *table);`
   - **Input:** `table`: Pointer to the hash table.
   - **Output:** Prints the key-value pairs currently stored in the hash table.

   **Example:**

   - **Output:** Hash table contents.

     ```
     Key: 2, Value: 15
     Key: 5, Value: 20
     Key: 7, Value: 25
     ```

**Other details:**

1. Use both linear probing or chaining to resolve collisions.
2. Use two distinct header file "`hashTableLP.h`" and "`hashTableC.h`" for linear probing and chaining respectively.
3. Use two distinct test files to test them.
4. take inputs from files only.

[70+130]