- Assignment Name: Lab 9
- Description: Hashing
- Total Marks: 10
- Deadline: Depends on respective lab hours

- Question Title: Collision Handling using Separate Chaining
- Marks allotted: 5
- Description:
  Insert values into a hash table using the hash function given below.
  Use separate chaining to resolve the collision i.e., handle collisions must
  by appending the collided values to the end of the linked list at its respective index.

  You will have to implement 1 function 'insertIntoHash' which takes the following parameters:
  - int size : size of the hash table
  - int value : the value that must be inserted
  - struct node** chain : the hash table which is stored as an array of linked lists.

  The function to be implemented is as follows:

```
void insertIntoHash(int size, int value, struct node** chain)
```

  - You must allocate memory for each value being inserted.
  - The hash function used is **key%size** to get the respective index.
  - The function is to **insert only ONE element** on function call.

  The skeleton of each node is as follows:

```
struct node
{
    int data;
    struct node *next;
};
```

- Input format:
  Each test case starts with a digit 's', which represents the size of the hash table created.
  The next input is a digit 'n', which is the number of values that have to be inserted, followed by the n line-separated numbers that must be inserted into the hash table.

- Output format:
  For this problem the display function has been handled in the boilerplate code as output formats could become unnecessarily confusing.

USING THE BOILERPLATE CODE PROVIDED BY US IS MANDATORY, ELSE YOU WILL FAIL MOST TEST CASES.

Sample Test Cases:

- **TC #1:**
  - **Input:**
    5
    10
    1
    2
    3
    4
    5
    6
    7
    8
    9
    10
  - **Expected output:**
    [0]->5->10->NULL
    [1]->1->6->NULL
    [2]->2->7->NULL
    [3]->3->8->NULL
    [4]->4->9->NULL
  - **Total Marks:** 0

- **TC #2:**
  - **Input:**
    3
    2
    6
    8
  - **Expected output:**
    [0]->6->NULL
    [1]->NULL
    [2]->8->NULL
  - **Total Marks:** 0

- Question Title: Collision Handling using Double Hashing
- Marks allotted: 5
- Description:
  Insert values into a hash table using the hash function given below. Use double hashing to resolve collisions i.e.,
  Collisions must be handled by calculating the index value using a second hash function by maintaining
  collision count.

  The hash table is of a constant size of 13. The first hash function is simply key%table_size. The second hash function is taken as, x-(key%x). The value of x is taken as the largest prime number less than the table size, in this case, 11. By default all values of the hash table is 0.

  You will have to implement two functions. One function to insert an element into the hash table and also a display function to print the hash table.

  Functions to be implemented are:

```
void insertIntoHash(int key, int *hashtable)
```

  - Takes in two parameters, int key (the value to be inserted) and int* hashtable (the hash table).
  - Inserts **only ONE element** per function call.
  - The first hash function is **key%table_size**. (In this case key%13)
  - The second hash function is **x-(key%x)**. (In this case 11-(key%11))
  - The index is to be calculated using the formula,
    (Hash1 + i*Hash2)%table_size
    Where 'i' is the collision count.
  - You must print -1 in a new line if there is no more space in the hash table.

```
void printHashTable(int *hashtable)
```

  - Takes in one parameter, the hash table.
  - Print all values of the hash table in a space separated manner.
  - Nothing else must be printed.
  - Any '-1' printed due to insufficient space will be printed before this function is called.

- Input format:
  Each test case starts with a digit 'n', which is the number of values that have to be inserted, followed by the n line-separated numbers that must be inserted into the hash table.

- Output format:
  The printHashTable function must be implemented which prints the values of the hash table in a space separated manner. '-1' must be printed in the insertIntoHash function if there is any insufficient space on the table. **Any number of '-1's will only be printed before the printHashTable function is called.**

USING THE BOILERPLATE CODE PROVIDED BY US IS MANDATORY, ELSE YOU WILL FAIL MOST TEST CASES.

Sample Test Cases:

- **TC #1:**
  - **Input:**
    ```
    10
    2
    4
    6
    8
    10
    12
    14
    16
    18
    20
    ```
  - **Expected output:**
    ```
    0 14 2 16 4 18 6 20 8 0 10 0 12
    ```
  - **Total Marks:** 0

- **TC #2:**
  - **Input:**
    ```
    14
    1
    2
    3
    4
    5
    6
    7
    8
    9
    10
    11
    12
    13
    14
    ```
  - **Expected output:**
    ```
    -1
    13 1 2 3 4 5 6 7 8 9 10 11 12
    ```
  - **Total Marks:** 0