- Assignment Name: Lab5
- Description: Question based on Binary Search Tree
- Total Marks: 10
- Deadline: Depends on respective lab hrs

- Problem Title: **Develop a system to store SRNs using trees**
- Marks allotted: 10
- Description:

Create a Binary Search Tree to store SRNs of students where SRNs are strings. Implement the insert function of the BST by comparing strings.

You will have to implement these other functions also:
- A function to print all SRNs in ascending order
- A function to search an SRN

A Binary Search Tree is a node based non-linear data structure that stores values hierarchically which has the following properties:
- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and the right subtree of a node will be binary search trees themselves.

Your task is to implement an insert function, a search function and a function that prints values in an ascending order.

You are required to implement the following tree operations as a part of this lab exercise:
1) void insert(char*key, struct node**leaf,Compare cmp);
    - This function inserts the string "key" to its appropriate position in the tree.
    - It is mandatory in this function that you update check as -1 in case you encounter a duplicate insertion.
    - There can be multiple duplicate insertions, you just have to update check as -1 every time you see a duplicate value i.e. if a value has been duplicated once, check will be -1, and even if there are two duplicate values, check will be -1, so write the code accordingly.

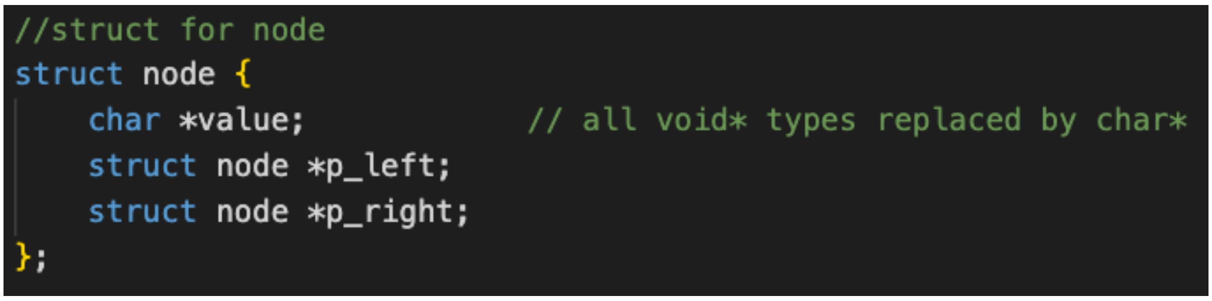
2) void asc_order(struct node*root)
    - This function prints the values stored in trees in the ascending order.

3) void search(char*key,struct node*leaf,Compare cmp)
    - This function is used to search for a key in the BST,set the found variable to 1 if found, else return;

The other string operations you're supposed to implement are my_strlen, my_strcmp, my_strcpy

You are supposed to use the code provided to you via the boilerplate code.

The skeleton of the tree is provided to you via the “struct” so you need not code that.



```
//struct for node
struct node {
    char *value;            // all void* types replaced by char*
    struct node *p_left;
    struct node *p_right;
};
```

USING THE BOILERPLATE CODE PROVIDED BY US IS MANDATORY.
YOU WILL FAIL MOST OF THE TEST CASES IN CASE YOU FAIL TO DO SO.

- Input format:

  Each test case starts with a digit ‘n’, this n denotes the number of values in the tree
  N is followed by the n strings to be inserted in the tree.

  After the n strings, one more string is scanned to see if the string is present in the tree or not.

- Output format:

  The values stored in the tree are printed in ascending order, followed by an integer to check if there have been any duplicate insertions or not,
  i.e. set check to -1 if there is **ANY** duplicate insertion, and 0 if not
  followed by the value returned by search i.e. 0 if the scanned element is not present in the tree and 1 if the element is present.

  You don’t need to print anything as a part of this lab exercise.
  Make sure to remove all the printfs and scanfs used for debugging.

- Sample test cases:

1. TC #1:
    - Input:

      ```
      2
      PES1UG30CS123
      PES1UG30CS143
      PES1UG30CS133
      ```
    - Output:

      ```
      PES1UG30CS123
      PES1UG30CS143
      0
      0
      ```

2. TC #2:
    - Input:

      ```
      3
      PES1UG19CS789
      PES1UG19CS123
      PES1UG19CS890
      PES1UG19CS123
      ```
    - Output:

      ```
      PES1UG19CS123
      PES1UG19CS789
      PES1UG19CS890
      PES1UG19CS123
      0
      1
      ```