National Institute of Technology Calicut
Department of Computer Science and Engineering
Second Semester B.Tech (CSE)
CS1092E Program Design Laboratory
Assignment #5

Submission deadline (on or before): 23/03/2025 (Sunday), 11:59 PM

Naming Conventions for Submission

• Submit a single ZIP (.zip) file (do not submit in any other archived formats like .rar, .tar, .gz). The name of this file must be

ASSG<Number><ROLLNO><FIRST-NAME>.zip

(Example: ASSG5_ B240123CS_NEEMA.zip). DO NOT add any other files (like temporary f iles, input files, etc.) except your source code, into the zip archive.

• The source codes of each question must be named as

ASSG<Number><ROLLNO><PROGRAM-NUMBER>.c

(For example: ASSG5_ B240123CS_1.c).

Standard of Conduct

• Violation of academic integrity will be severely penalized. Each student is expected to adhere to high standards of ethical conduct, especially those related to cheating and plagiarism. Any submitted work MUST BE an individual effort. Any academic dishonesty will result in zero marks in the corresponding exam or evaluation and will be reported to the department council for record keeping and for permission to assign F grade in the course. The department policy on academic integrity can be found at: http://cse.nitc.ac.in/sites/default/files/Academic-Integrity new.pdf.

General Instructions

• Programs should be written in C language.
• Check your programs with sufficiently large values of inputs within the range as specified in the question.
• Global and/or static variables should not be used in your program.

PD Assignment-5 (Binary Search Tree (BST) operations and application)

S.No	Topic and Problem (Focus on problems 1 to 4)
1.	**Creation of BST from the parenthesis representation** A Binary Search Tree (BST) is represented using a parenthesis notation. Given a parenthesis representation of a BST, write a program to construct it and print its postorder traversal. Parenthesis Representation Rules: 1. Each node is represented as: ● (V(LeftSubtree)(RightSubtree)), where V is the value of the node. 2. If a node has no children, it is represented as V(). 3. The input is guaranteed to be a valid BST representation. **Input Specification:**

- The single string S is used to represent the BST in parenthesis notation.
- The value V of each node is a positive integer ($1 \leq V \leq 10^5$).
- The length of S does not exceed 10^5 characters.

Output Specification:

- Print the postorder traversal of the BST (space-separated values).

Note: The tree follows BST properties (the root node is bigger than its left subtree nodes and the root node is smaller than its right subtree nodes) with no duplicate values. The tree may be left-skewed, right-skewed, or balanced. Empty subtrees must be handled correctly (V()). The output must be in postorder traversal format (Left → Right → Root).

Sample Input 1:

(5(3(2)())(8(6)(10)))

Sample Output 1:

2 3 6 10 8 5

Sample Input 2:

(10(5(2(1)())())())

Sample Output 2:

1 2 5 10

Sample Input 3:

(1()(2()(3()(4()(5)))))

Sample Output 3:

5 4 3 2 1

Sample Input 4:

(50(25(10)(30))(75(60)(90)))

Sample Output 4:

10 30 25 60 90 75 50

	Sample Input 5: (100(5(25(15)(10))(20(6)(7)))(85(130)(135()(200)))) **Sample Output 5:** 15 10 25 6 7 20 5 130 200 135 85 100
2.	**<u>BST Traversal Menu-driven (Creation, Inorder, Preorder, Postorder, Level order travel) using structures</u>** Design a menu-driven C program for a Student Database Management System using a Binary Search Tree (BST)**.** Each student record has a Student ID (used as the BST key), Student Name, and Student Marks. The program should allow users to insert student records into the BST while maintaining the BST property. Users should be able to retrieve and display student records using different traversal methods: Inorder Traversal**,** Preorder Traversal**,** Postorder Traversal**,** and Level Order Traversal**.** Additionally, the program should provide a search function that allows users to find and display a specific student's details based on their Student ID. The menu should repeatedly allow the user to perform operations until they choose to exit the program. This application ensures efficient organization, searching, and retrieval of student data using BST properties. **Inp**ut Specification: ● Menu-driven options (1 Insert student, 2 Inorder, 3 Preorder, 4 Postorder, 5 Level Order Traversal and 6. Exit) ● In option 1 <**StudentID Name Marks**>: the given student's record is inserted into BST based on the StudentID. The student's record is in the format given below: ○ **StudentID Name Marks** **StudentID** (integer, 100 <= StudentID <= 250), **Name** (character string not more than 30), **Marks** (integer, 0 <= Marks <= 100). ● Options 2, 3, 4, and 5 are displaying the BST in the inorder, preorder, postorder, and level order traversals, respectively. ● Option 6 <StudentID>: Search if the given studentID is available in BST or not. ● Option 7: Exit from the program execution **Output Specifications:** ● Options 2, 3, 4, and 5 are displaying the BST in the inorder, preorder, postorder, and level order traversals, respectively. ● Option 6: Display the student's details with space seperated if the student's ID found in the non-empty BST; Otherwise, it prints "-1" ● Option 7: Exit from the program execution

Sample Input 1

1. Insert a student
2. Display Inorder Traversal
3. Display Preorder Traversal
4. Display Postorder Traversal
5. Display Level Order Traversal
6. Search for a student
7. Exit

1 105 John 85
1 102 Alice 90
1 108 Bob 78
2
3
4
5
6 108
6 103
7

Sample Output 1:

102 Alice 90
105 John 85
108 Bob 78

105 John 85
102 Alice 90
108 Bob 78

102 Alice 90
108 Bob 78
105 John 85

105 John 85
102 Alice 90
108 Bob 78

108 Bob 78
-1

7

3.	**BST (Creation, Search, Insertion, Deletion)** A cloud computing company maintains a database index as a Binary Search Tree (BST) to efficiently manage and retrieve customer transaction records. Each record is uniquely identified by a transaction ID (integer value). Your task is to implement the following BST operations: 1. **Insertion** – Insert a transaction ID into the BST. 2. **Search** – Search for a transaction ID in the BST. 3. **Deletion** – Delete a transaction ID from the BST. 4. **Inorder Traversal** – Print the transaction IDs in sorted order **Input Specifications:** 1. The first line contains N, the number of initial transaction IDs. 2. The second line contains N space-separated integers representing the initial transaction IDs to be inserted into BST (integer, 10^3 <= TranID <= 10^5). 3. I <TransID>: For inserting, the given <TransID> into the created BST. 4. Y <TransID>: For deleting, the given <TransID> from the non-empty BST. 5. Z <TransID>: Print "1" if the given <TransID> is found in the non-empty BST: Otherwise, print "-1". 6. K: To print the inorder traversal of the BST; Print "-1" if the BST is empty. 7. X : To exit from the program execution. **Output Specifications:** 1. Y <TransID>: Prints the TransID if it is deleted from the non-empty BST; Otherwise, print "-1". 2. Z <TransID>: Print "1" if the given <TransID> is found in the non-empty BST: Otherwise, print "-1". 3. K : Print the non-empty BST in inorder; print -1 if the BST if empty. 4. X: Exit from the program execution. **Sample Input 1:** 7 1005 1010 1002 1008 1015 1001 1006 I 1007 Y 1002

	Y 1012 Z 1010 K X **Sample Output 1:** 1002 -1 1 1001 1005 1006 1007 1008 1010 1015
4.	BST (Creation, Maximum, Minimum, Predecessor, Successor) In India, the Aadhaar Card is a resident's unique identification ID. Every Aadhaar number is a 12-digit unique number. Design a system to maintain Aadhaar card records effectively using **a Binary Search Tree (BST)**. The system must: 1. Store Aadhaar numbers and the corresponding name of the cardholder (not more than 50 characters including spaces). 2. Find the minimum and maximum Aadhaar numbers in the BST. 3. Locate the predecessor (Largest Aadhaar number which is smaller than the given aadhar number) and the successor (Smallest Aadhaar number which is Larger than the given aadhar number) of a particular Aadhaar number. **Input Specification** The program accepts a **command** (a string of characters not more than 10 characters) which is the operation to be performed (in capital letters including spaces), followed by optional arguments depending on the command. The commands and their formats are as follows: 1. **Insert Operations**: ○ **INSERT** <aadhaar_number> <name>: Inserts an Aadhaar record with the given <aadhaar_number> (12-digit unique integer) and <name> (string) into the BST. ■ Example: INSERT 123456789012 Rahul Sharma

2. **Find Operations**:
    - **MAXIMUM**: Finds and returns the Aadhaar record with the maximum Aadhaar number.
        - Example: MAXIMUM
    - **MINIMUM**: Finds and returns the Aadhaar record with the minimum Aadhaar number.
        - Example: MINIMUM
    - **PREDECESSOR** <aadhaar_number> <Ino/Pre/post>: Finds the predecessor of the given <aadhaar_number> based on the travel methods <Ino/Pre/post>.
        - Example: PREDECESSOR 456789123012 Ino
    - **SUCCESSOR** <aadhaar_number> <Ino/Pre/post>: Finds the successor of the given <aadhaar_number> based on the travel methods <Ino/Pre/post>
        - Example: SUCCESSOR 654321987012 Post
3. **Exit**:
    - EXIT: Terminates the program.
        - Example: EXIT

Each command should be entered on a new line.

Output Specification

The program will output results as follows:

1. **Insert Operations**: No output is required after the insertion operation.
2. **Find Operations**:
    - MAXIMUM: displays the maximum element available in the RST. Prints '-1' if RST is empty.
    - MINIMUM: Displays the maximum element in the LST from the base node. Prints -1 if the LST is empty.
    - PREDECESSOR: Displays the predecessor available based on the travel method mentioned. Prints -1 if there is no predecessor.
    - SUCCESSOR: Displays the successor available based on the travel method mentioned. Prints -1 if there is no successor.

SAMPLE INPUT 1

INSERT 123456789012 Rahul Sharma
INSERT 987654321012 Priya Patel

	INSERT 456789123012 Aarav Gupta INSERT 654321987012 Ananya Singh INSERT 321654987012 Isha Reddy MAXIMUM MINIMUM PREDECESSOR 456789123012 Ino SUCCESSOR 654321987012 Ino PREDECESSOR 123456789012 Post SUCCESSOR 987654321012 Pre EXIT **SAMPLE OUTPUT 1** 987654321012, Priya Patel 123456789012, Rahul Sharma 321654987012, Isha Reddy 987654321012 Priya Patel -1 -1
5.	Applications of Parenthesis representation of BST **Problem Statement: Library management system** A **library database system** maintains records of books using their **ISBN numbers** in a **Binary Search Tree (BST)**. The system **exports and imports** book data in **parenthesis representation** for efficient storage, searching, and retrieval. **Input Specification:** 1. The **BST is initially loaded from a file** named **bstparam.txt** containing the parameterized representation of BST. Given below is the contents of **bstparam.txt** **978(500(250)(700))(1500(1200)(1800(1750)(2000)))** 2. The following operations can be performed using menu-driven commands: ○ **p**: Print the BST using **pre-order traversal**. ○ **i**: Print the BST using **in-order traversal**. ○ **o**: Print the BST using **post-order traversal**. ○ **a <ISBN>**: Add a new book with the given ISBN to the BST and update the parenthesis representation. ○ **d <ISBN>**: Delete the book with the given ISBN from the BST and update the parenthesis representation.

- **x**: Export and update the current BST parenthesis representation to the file **bstparam.txt**.
- **e**: Exit the program.

Output Specification:

For traversal commands (p, i, o):

- Print the ISBN numbers in the specified traversal order, separated by commas.

For the insertion command (a <ISBN>):

- Print the updated parenthesis representation after inserting the new book.

For the delete command (d <ISBN>):

- Print the updated parenthesis representation after deleting the book.
- If the book is not found, print: ISBN not found

For the export command (x):

- Save the current BST parenthesis representation to **bstparam.txt**.
- Print: BST exported to filename.txt

For the exit command (e):

- Terminate the program.

Sample input 1:

```
p
i
o
a 1600
p
d 500
p
x
e
```

Sample output 1:

```
Pre-order Traversal: 978,500,250,700,1500,1200,1800,1750,2000
In-order Traversal: 250,500,700,978,1200,1500,1750,1800,2000
Post-order Traversal: 250,700,500,1200,1750,2000,1800,1500,978
Updated: 978(500(250)(700))(1500(1200)(1800(1750(1600))(2000)))
Pre-order Traversal: 978,500,250,700,1500,1200,1800,1750,1600,2000
After deletion:978(700(250))(1500(1200)(1800(1750(1600))(2000)))
Pre-order Traversal: 978,700,250,1500,1200,1800,1750,1600,2000
```

	BST exported to file 978(700(250))(1500(1200)(1800(1750(1600))(2000)))
6.	New features on BST (Creation, Is BST1 a subtree of BST2?) **Problem Statement:** You are designing a system for an online library where books are categorized using a **Binary Search Tree (BST)** based on their unique Book IDs. The library system needs to handle the following functionalities: 1. **Creation of a BST**: The system should allow adding book IDs to a BST. 2. **Subtree Check**: Given two BSTs representing different sections of the library, determine if one BST (BST1) is a subtree of another BST (BST2). A BST (**BST1)** is considered a subtree of **BST2** if there exists a node in **BST2** such that the subtree rooted at that node is identical to **BST1**. **Input Specification:** ● The first integer N1 represents the number of book IDs in **BST1**. ● The next N1 integers represent the book IDs inserted into **BST1** in the given order. ● The next integer N2 represents the number of book IDs in **BST2**. ● The next N2 integers represent the book IDs inserted into **BST2** in the given order. **Output Specification:** ● Print "BST1 is a subtree of BST2" if BST1 is found as a subtree of BST2. Otherwise, print "BST1 is NOT a subtree of BST2" **Sample Input 1:** 3 3 1 5 5 10 5 15 1 3 **Sample Output 1:** BST1 is NOT a subtree of BST2 **Sample Input 2:** 3 4 2 5 6 6 4 8 2 5 7

	Sample Output 2: BST1 is a subtree of BST2 **Sample Input 3:** 2 2 3 5 10 8 12 7 9 **Sample Output 3:** BST1 is NOT a subtree of BST2
7.	New features on BST (creation from the parentheses representation, applications of BST, infix, prefix, postfix) ## Scenario: Binary Search Trees (BSTs) are widely used in applications such as database indexing, expression evaluation, and search operations. In this problem, we introduce new features in BST, which include: 1. **Creation of BST1 from Parentheses Representation** ○ A BST will be provided in a **parentheses representation**, from which the BST needs to be constructed. ○ Example input: `"((5(3(2)(4)))(7(6)(8)))"` should be converted into a valid BST. 2. **Operations on BST1** ○ Supporting fundamental operations like **searching**, **finding min/max**, and **depth calculation** of the BST. 3. **Expression Conversions using BST2 created from the infix expression** ○ Given an **infix expression**, construct a BST and generate its **prefix** and **postfix** expressions. ## Input Specification 1. **First line:** A string representing the BST in **parentheses representation**. 2. **Second line:** A space-separated **infix expression** consisting of operands `{0-9}` and operators `{+,-,*,/}`.

3. **Third line:** An integer **Q**, denoting the number of queries.
4. **Next Q lines:** Queries, each of the form:
    - `"FIND x"` → Check if `x` exists in BST1 (`YES`/`NO`).
    - `"MIN"` → Return the minimum element in BST1.
    - `"MAX"` → Return the maximum element in BST1.
    - `"DEPTH"` → Return the depth of the BST1.
    - `"PREFIX"` → Return the **prefix notation** of the infix expression created from BST2.
    - `"POSTFIX"` → Return the **postfix notation** of the infix expression created from BST2.

Output Specification

For each query, output the result accordingly:

- `"YES"` or `"NO"` for `"FIND x"` query.
- Integer value for `"MIN"`, `"MAX"`, and `"DEPTH"`.
- **Prefix notation** for `"PREFIX"` query.
- **Postfix notation** for `"POSTFIX"` query.

Edge Cases

1. **Empty BST:**
    - Input: `""`
    - Output: Handle gracefully, return `"NO"` for search and `"N/A"` for MIN/MAX/DEPTH.
2. **Single Node BST:**
    - Input: `"(10)"`
    - Output: Handle `"FIND"`, `"MIN"`, `"MAX"` correctly.

Sample Input 1:

((5(3(2)(4)))(7(6)(8)))

3 + 5 * (2 - 4)

5

FIND 4

MIN

MAX

PREFIX

POSTFIX

Sample Output 1:

YES

2

8

+ 3 * 5 - 2 4

3 5 2 4 - * +

Sample Input 2:

((10(5(2)(7)))(15(12)(18)))

(8 + 2) * 5 - 6 / 3

4

FIND 20

DEPTH

PREFIX

POSTFIX

Sample Output 2:

NO

3

- * + 8 2 5 / 6 3

8 2 + 5 * 6 3 / -

Sample Input 3:

<Edge Case 1: Empty BST>

" "

3 + 5

4

FIND 3

MIN

MAX

DEPTH

Sample Output 3:

NO

N/A

N/A

0

Sample Input 4:

<Edge Case 2: Single Node BST>

(10)

10 + 2

5

FIND 10

FIND 5

MIN

MAX

DEPTH

Sample Output 4:

	YES NO 10 10 1