Green University of Bangladesh

Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: 3rd - Fall semester / Year:2022

BSc in CSE (Day)

LAB REPORT NO - 05

Course Title: Data Structure Lab

Course Code: CSE 106 / Section: DE-221

Lab Experiment Tittle :

- Implement a BST with 3 traversal method
- Search an element in the tree and print the address of that element
- Print all the leaf element

Student Details

	Name	ID
1.	Khondokar Saim	221902353

Lab Date	: 07 / 12 / 2022
Submission Date	: 14 / 12 / 2022
Course Teacher's Name	: Farhana Akter Sunny.

[For Teachers use only: Don't Write Anything inside this box]

Lab Report Status

Marks: …………………………………… Signature: …………………

Comments: …………………………………… Date: …………………………

❑ Tittle of the Lab Experiment :

Binary Search Tree (BST) operations:

- Implement a BST with 3 traversal method.
- Search an element in the tree and print the address of that element.
- Print all the leaf element.

❑ Objectives :

- A binary search tree is a type of data structure that allows for efficient search and insertion operations.

- The primary objective of a binary search tree is to allow for efficient search operations. Because the tree is arranged in a specific way, search operations can be performed quickly by starting at the root of the tree and then traversing the tree in a specific manner.

- The main objective of traversing a binary search tree is to visit each node in the tree in some specific order. This can be useful for a number of different purposes

- There are three main ways to traverse a binary search tree:
  - ➢ In-order traversal
  - ➢ Pre-order traversal
  - ➢ Post-order traversal

- The primary objective of having leaf nodes in a binary search tree is to mark the end of a branch in the tree. This is important because it allows the tree to be traversed in a specific order, such as ascending order or descending order.

1. **Implement a BST with 3 traversal method.**

Algorithm :

Step – 1 = Create structure of a node

Step – 2 = Create function prototype

```
struct node *create_node(int);
void insert(int);
```

Step – 3 = Create menu based switch condition for operating BST operations

Step – 4 = creates a new node

```
struct node *create_node(int data)
{
    struct node *new_node = (struct node *)malloc(sizeof(struct node));

    if (new_node == NULL)
    {
        printf("\nMemory for new node can't be allocated");
        return NULL;
    }

    new_node->data = data;
    new_node->left = NULL;
    new_node->right = NULL;

    return new_node;
}
```

Step – 5 = inserts the data in the BST

Step – 6 = inorder traversal

```
void inorder(struct node *root)
{
    if (root == NULL)
    {
        return;
    }
    inorder(root->left);
    printf("%d ",  root->data);
    inorder(root->right);
}
```

Step – 7 = preorder traversal

```
void preorder(struct node *root)
{
    if (root == NULL)
    {
        return;
    }
    printf("%d ", root->data);
    preorder(root->left);
    preorder(root->right);
}
```

Step – 8 = postorder travsersal

```
void postorder(struct node *root)
{
    if (root == NULL)
    {
        return;
    }
    postorder(root->left);
    postorder(root->right);
    printf("%d ", root->data);
}
```

Step – 8 = getting data from the user

```
int get_data()
{
    int data;
    printf("\nEnter Data: ");
    scanf("%d", &data);
    return data;
}
```

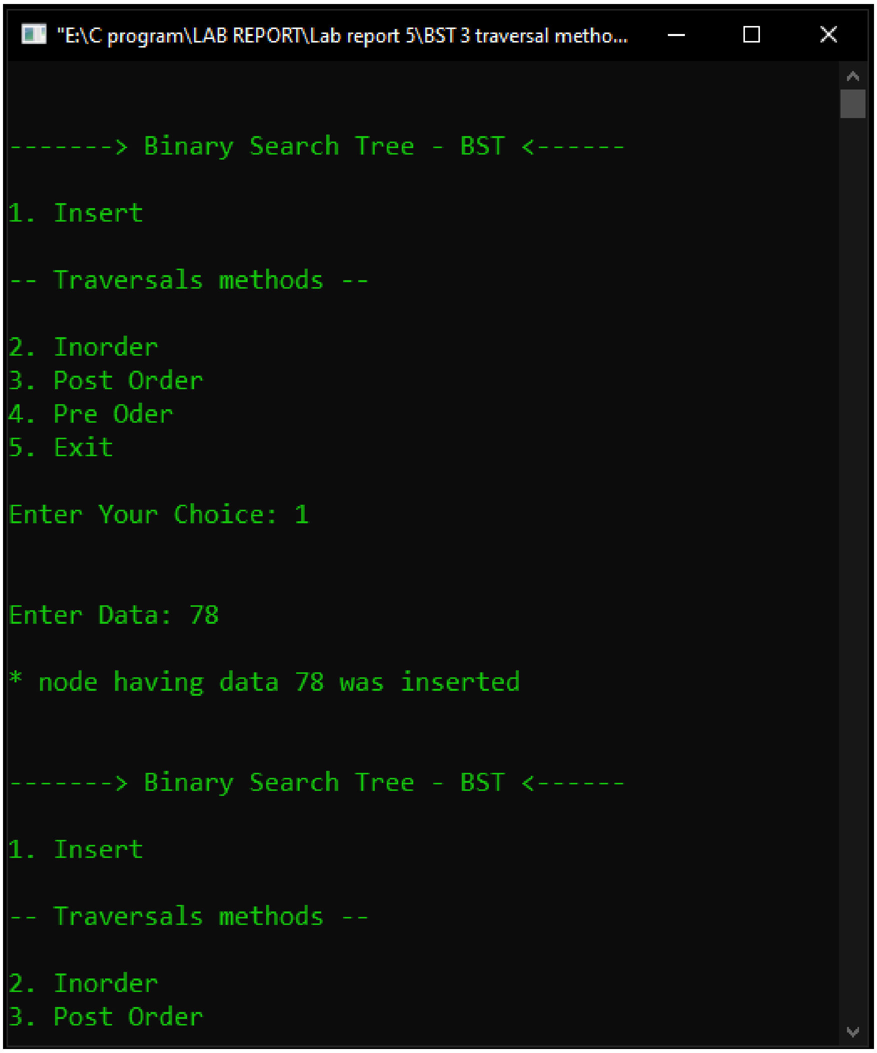
Code:

```
#include <stdio.h>
#include <stdlib.h>

// structure of a node
struct node
{
   int data;
   struct node *left;
   struct node *right;
};

// globally initialized root pointer
struct node *root = NULL;

// function prototyping
struct node *create_node(int);
void insert(int);

//int search(int);
void inorder(struct node *);
void postorder();
void preorder();

int get_data();

int main()
{
   int userChoice;
   int userActive = 'Y';
   int data;
   struct node* result = NULL;

   while (userActive == 'Y' || userActive == 'y')
   {
      printf("\n\n-------> Binary Search Tree - BST <------\n");
      printf("\n1. Insert");
      printf("\n\n-- Traversals methods --");
      printf("\n\n2. Inorder ");
      printf("\n3. Post Order ");
      printf("\n4. Pre Oder ");
      printf("\n5. Exit");
```

```
        printf("\n\nEnter Your Choice: ");
        scanf("%d", &userChoice);
        printf("\n");

        switch(userChoice)
        {
        case 1:
            data = get_data();
            insert(data);
            break;

        case 2:
            inorder(root);
            break;

        case 3:
            postorder(root);
            break;

        case 4:
            preorder(root);
            break;

        case 5:
            printf("\n\nProgram was terminated\n");
            break;

        default:
            printf("\n\tInvalid Choice\n");
            break;
        }


    }

    return 0;
}
// creates a new node
struct node *create_node(int data)
{
    struct node *new_node = (struct node *)malloc(sizeof(struct node));
    if (new_node == NULL)
    {
        printf("\nMemory for new node can't be allocated");
        return NULL;
    }

    new_node->data = data;
    new_node->left = NULL;
    new_node->right = NULL;

    return new_node;
}
// inserts the data in the BST
void insert(int data)
{
    struct node *new_node = create_node(data);

    if (new_node != NULL)
    {
        // if the root is empty then make a new node as the root node
        if (root == NULL)
        {
            root = new_node;
            printf("\n* node having data %d was inserted\n", data);
            return;
        }

        struct node *temp = root;
        struct node *prev = NULL;

        // traverse through the BST to get the correct position for insertion
        while (temp != NULL)
        {
            prev = temp;
            if (data > temp->data)
            {
                temp = temp->right;
            }
            else
            {
                temp = temp->left;
            }
        }
        // found the last node where the new node should insert
        if (data > prev->data)
        {
            prev->right = new_node;
        }
        else
        {
            prev->left = new_node;
        }

        printf("\n* node having data %d was inserted\n", data);
    }
}
// inorder traversal
void inorder(struct node *root)
{
    if (root == NULL)
    {
        return;
    }
    inorder(root->left);
    printf("%d ",  root->data);
    inorder(root->right);
}
// preorder traversal
void preorder(struct node *root)
{
    if (root == NULL)
    {
        return;
    }
    printf("%d ", root->data);
    preorder(root->left);
    preorder(root->right);
}
// postorder travsersal
void postorder(struct node *root)
{
    if (root == NULL)
    {
        return;
    }
```

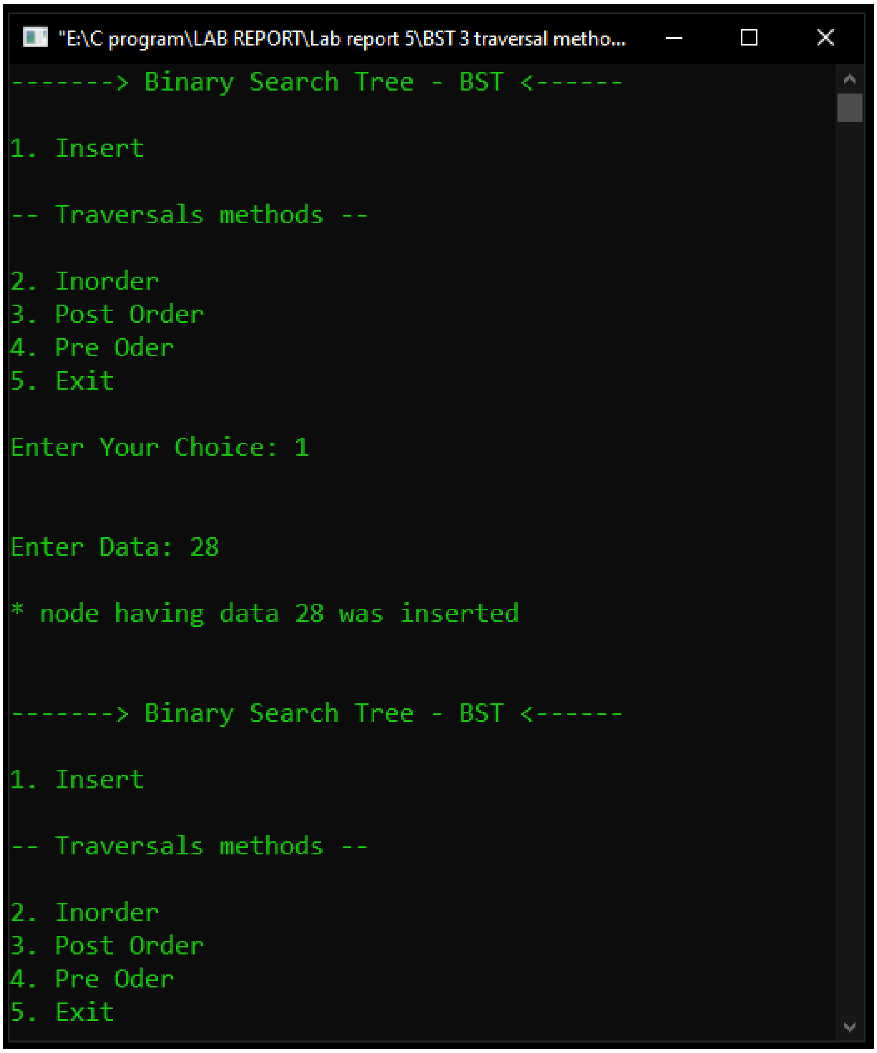
```
postorder(root->left);
    postorder(root->right);
    printf("%d ", root->data);
}
// getting data from the user
int get_data()
{
    int data;
    printf("\nEnter Data: ");
    scanf("%d", &data);
    return data;
}
```

Output:


"E:\C program\LAB REPORT\Lab report 5\BST 3 traversal metho...

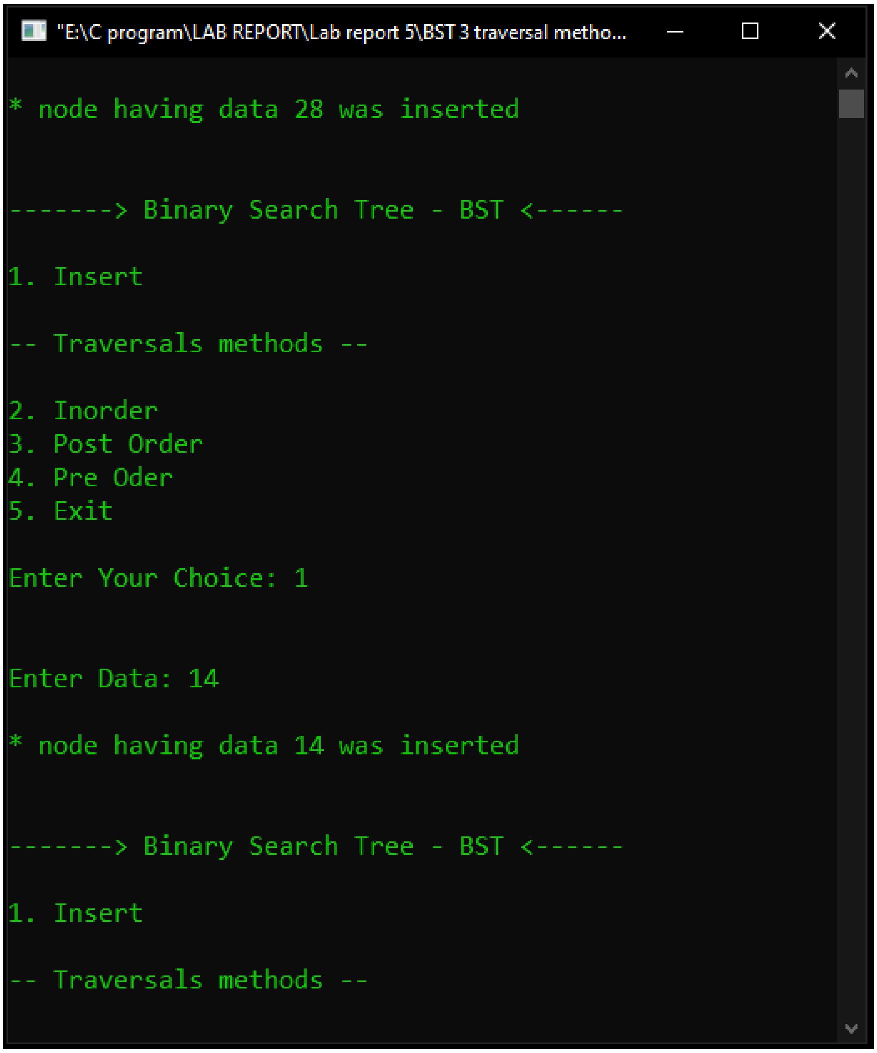
```
------->  Binary Search Tree - BST <------

1. Insert

-- Traversals methods --

2. Inorder
3. Post Order
4. Pre Oder
5. Exit

Enter Your Choice: 1


Enter Data: 78

* node having data 78 was inserted


------->  Binary Search Tree - BST <------

1. Insert

-- Traversals methods --

2. Inorder
3. Post Order
```

Case – 1 For inserted 78


"E:\C program\LAB REPORT\Lab report 5\BST 3 traversal metho...

```
-------> Binary Search Tree - BST <------

1. Insert

-- Traversals methods --

2. Inorder
3. Post Order
4. Pre Oder
5. Exit

Enter Your Choice: 1


Enter Data: 28

* node having data 28 was inserted


-------> Binary Search Tree - BST <------

1. Insert

-- Traversals methods --

2. Inorder
3. Post Order
4. Pre Oder
5. Exit
```

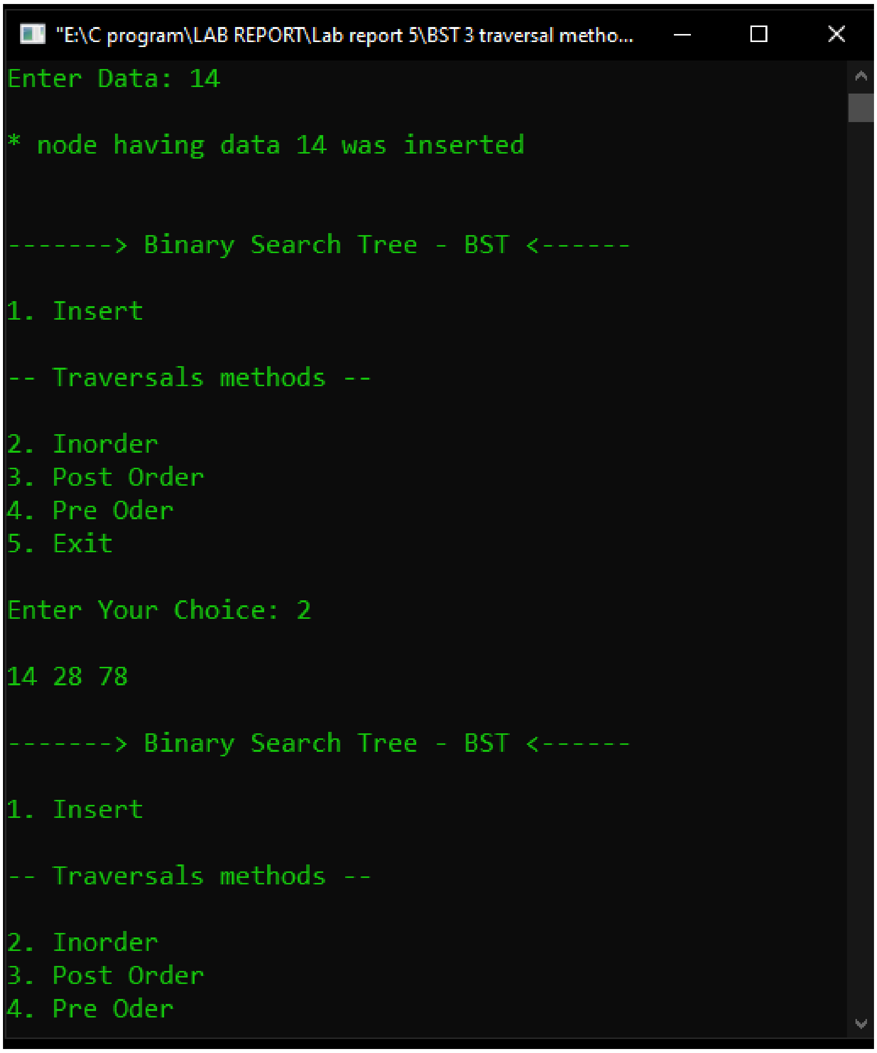
Case – 1 For inserted 28



"E:\C program\LAB REPORT\Lab report 5\BST 3 traversal metho...

```
* node having data 28 was inserted


-------> Binary Search Tree - BST <------

1. Insert

-- Traversals methods --

2. Inorder
3. Post Order
4. Pre Oder
5. Exit

Enter Your Choice: 1


Enter Data: 14

* node having data 14 was inserted


-------> Binary Search Tree - BST <------

1. Insert

-- Traversals methods --
```

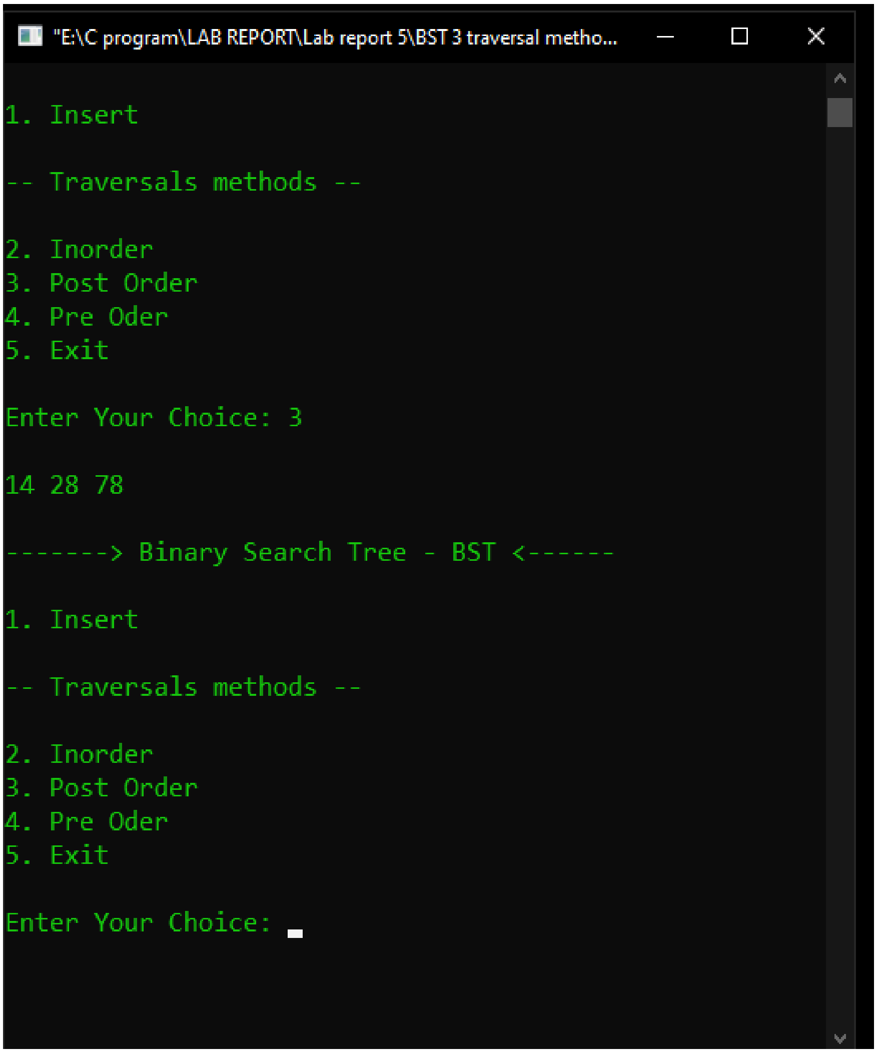
Case – 1 For inserted 14



"E:\C program\LAB REPORT\Lab report 5\BST 3 traversal metho...

```
Enter Data: 14

* node having data 14 was inserted


-------> Binary Search Tree - BST <------

1. Insert

-- Traversals methods --

2. Inorder
3. Post Order
4. Pre Oder
5. Exit

Enter Your Choice: 2

14 28 78

-------> Binary Search Tree - BST <------

1. Insert

-- Traversals methods --

2. Inorder
3. Post Order
4. Pre Oder
```

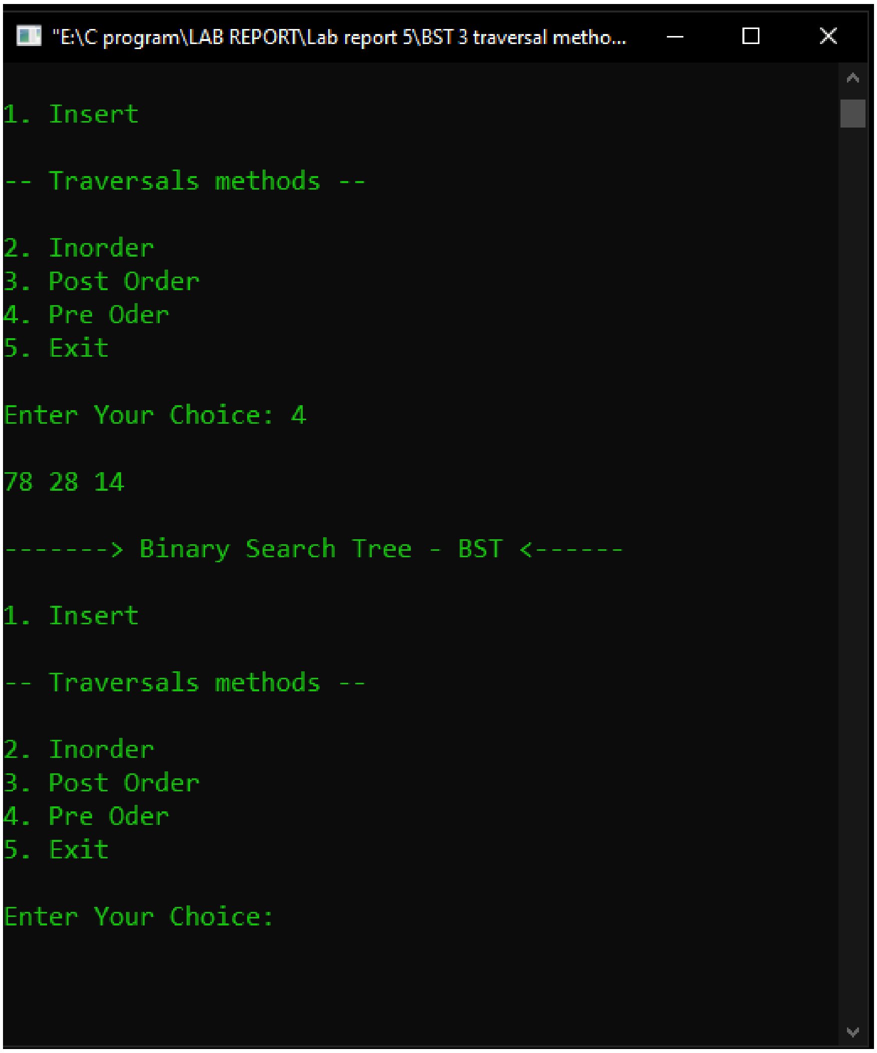
Case – 2 For Inorder



"E:\C program\LAB REPORT\Lab report 5\BST 3 traversal metho...

```
1. Insert

-- Traversals methods --

2. Inorder
3. Post Order
4. Pre Oder
5. Exit

Enter Your Choice: 3

14 28 78

-------> Binary Search Tree - BST <------

1. Insert

-- Traversals methods --

2. Inorder
3. Post Order
4. Pre Oder
5. Exit

Enter Your Choice:
```

Case – 3 For Postorder



"E:\C program\LAB REPORT\Lab report 5\BST 3 traversal metho...

```
1. Insert

-- Traversals methods --

2. Inorder
3. Post Order
4. Pre Oder
5. Exit

Enter Your Choice: 4

78 28 14

-------> Binary Search Tree - BST <------

1. Insert

-- Traversals methods --

2. Inorder
3. Post Order
4. Pre Oder
5. Exit

Enter Your Choice:
```

Case – 4 For Preorder

2. Search an element in the tree and print the address of that element.

Algorithm :

Step – 1 = Define a node in the binary tree

```
typedef struct node {
  int value;
  struct node *left;
  struct node *right;
} node;
```

Step – 2 = Function to search for an element in the binary tree

```
node* search(node* root, int value)
```

Step – 3 = If the value we are looking for is smaller than the current node's value, search in the left subtree

```
if (value < root->value) return search(root->left, value);
```

Step – 4 = If the value we are looking for is greater than the current node's value, search in the right subtree

```
if (value > root->value) return search(root->right, value);
```

Step – 5 = Create the binary tree in main function

Step – 6 = Search for the value 5 in the binary tree

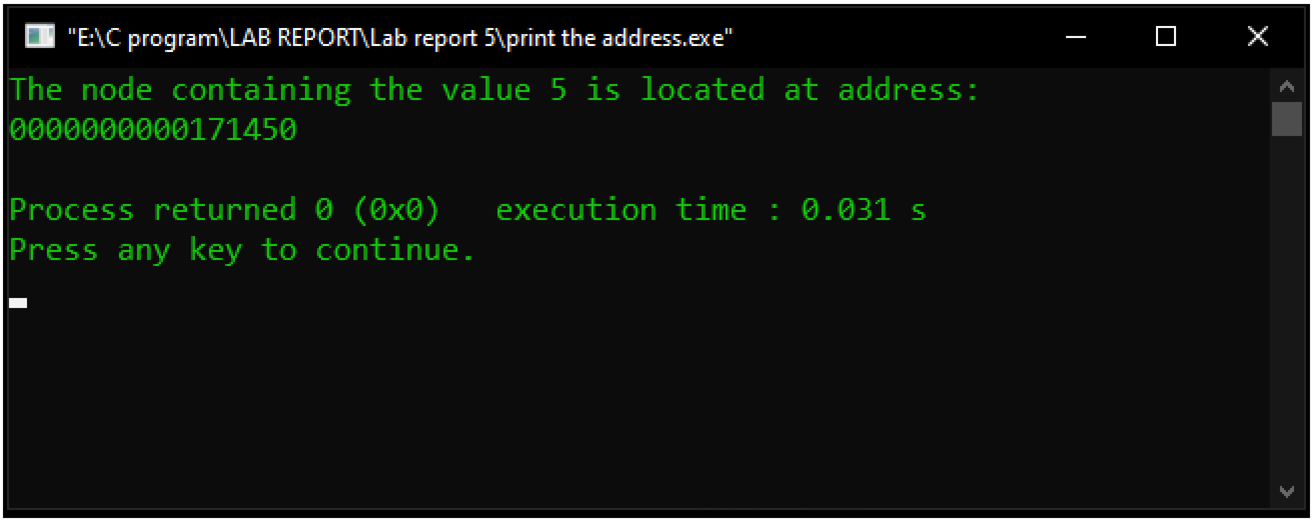
```
node *result = search(root, 3);
```

Step – 7 = Print the address of the node containing the value __

Code:

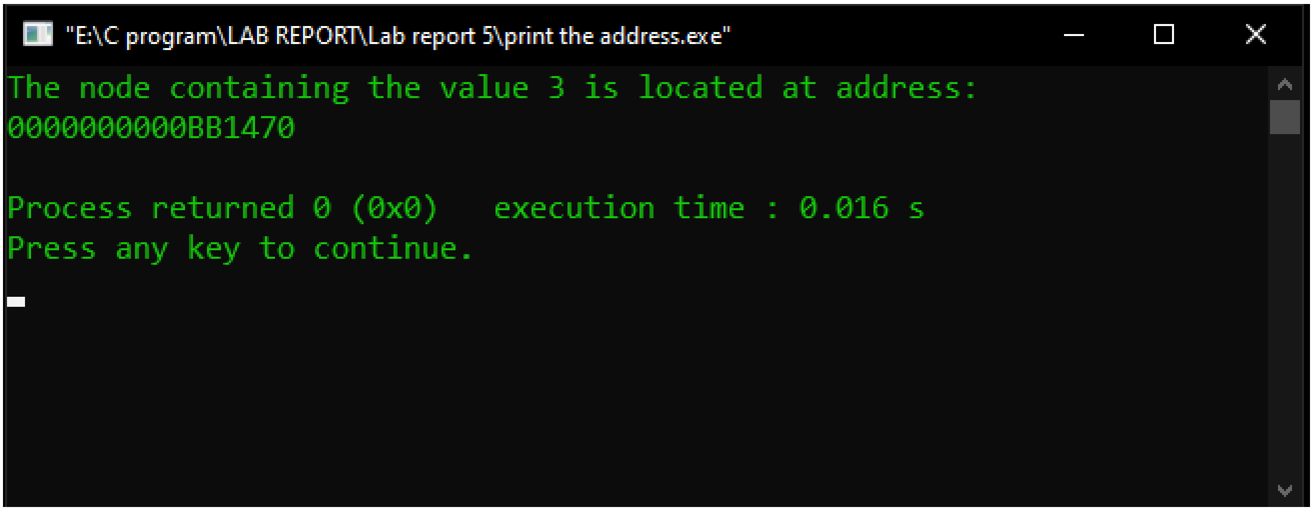
```
#include <stdio.h>
#include <stdlib.h>

// Define a node in the binary tree
typedef struct node {
  int value;
  struct node *left;
  struct node *right;
} node;

// Function to search for an element in the binary tree
node* search(node* root, int value) {
  // Return NULL if the tree is empty
  if (root == NULL) return NULL;

  // If the value we are looking for is smaller than the current node's value,
  // search in the left subtree
  if (value < root->value) return search(root->left, value);

  // If the value we are looking for is greater than the current node's value,
  // search in the right subtree
  if (value > root->value) return search(root->right, value);

  // If the value is equal to the current node's value, return the current
node
  return root;
}

int main() {
  // Create the binary tree
  node *root = malloc(sizeof(node));
  root->value = 5;

  root->left = malloc(sizeof(node));
  root->left->value = 3;

  root->right = malloc(sizeof(node));
  root->right->value = 7;

  root->left->left = NULL;
  root->left->right = NULL;
  root->right->left = NULL;
  root->right->right = NULL;
  // Search for the value 5 in the binary tree
  node *result = search(root, 3);

  // Print the address of the node containing the value 5
  printf("The node containing the value 5 is located at address %p\n",
result);

  return 0;
}
```

Output:


"E:\C program\LAB REPORT\Lab report 5\print the address.exe"

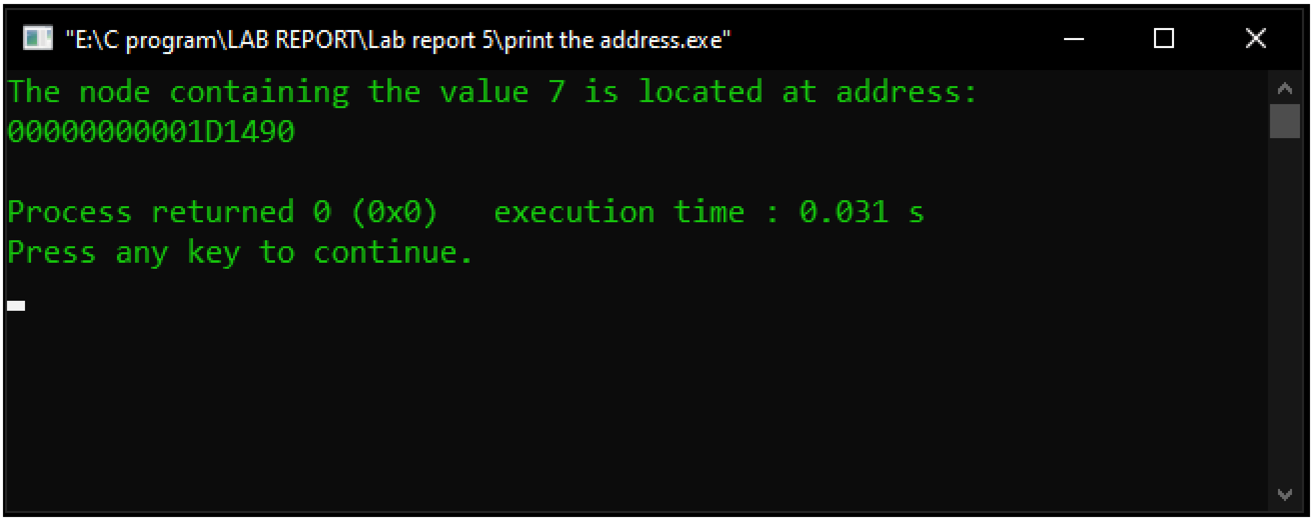
```
The node containing the value 5 is located at address:
0000000000171450

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

Case – 1 For 5 located Address


"E:\C program\LAB REPORT\Lab report 5\print the address.exe"

```
The node containing the value 3 is located at address:
0000000000BB1470

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

Case – 2 For 3 located address


"E:\C program\LAB REPORT\Lab report 5\print the address.exe"

```
The node containing the value 7 is located at address:
00000000001D1490

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

Case – 3 For 7 located address

3. Print all the leaf element.

Algorithm :

Step – 1 = Define a node in the binary tree

```
struct Node
{
   int data;
   struct Node *left, *right;
};
```

Step – 2 = Function to create a new binary tree node

```
struct Node* newNode(int data)
{
   struct Node* node = (struct Node*)malloc(sizeof(struct Node));
   node->data = data;
   node->left = node->right = NULL;
   return node;
}
```

Step – 3 = Function to print leaf nodes of a binary tree

```
void printLeafNodes(struct Node* root)
```

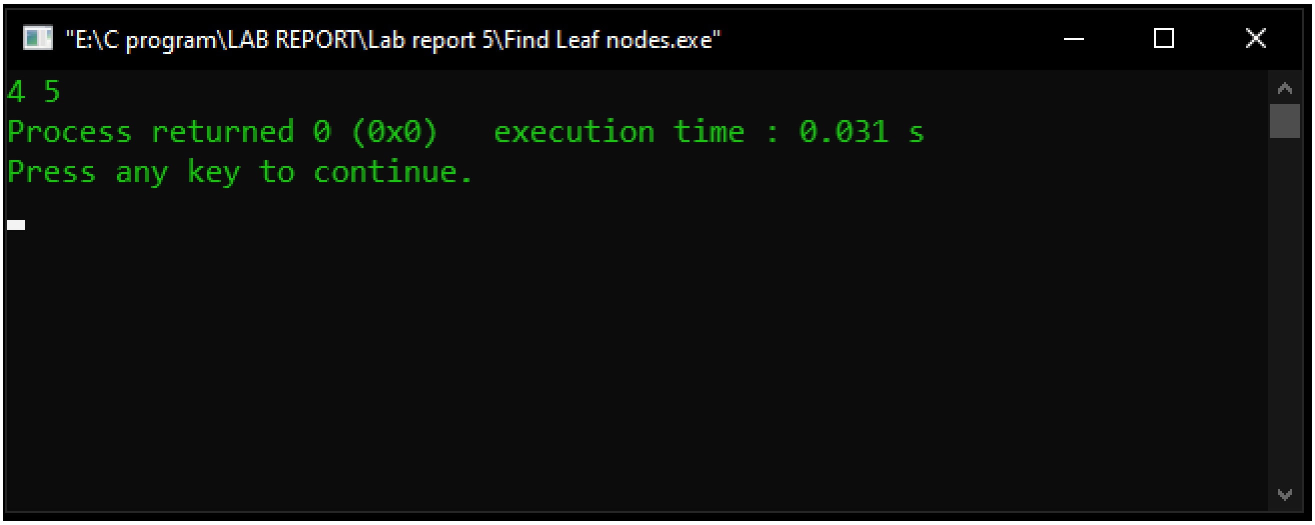
Step – 4 = construct a binary tree in main function

Step – 5 = print leaf nodes of the binary tree

Code:

```
#include <stdio.h>
#include <stdlib.h>

// A binary tree node
struct Node
{
    int data;
    struct Node *left, *right;
};

// Function to create a new binary tree node
struct Node* newNode(int data)
{
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

// Function to print leaf nodes of a binary tree
void printLeafNodes(struct Node* root)
{
    // base case
    if (root == NULL)
        return;

    // if current node is a leaf node, print its data
    if (root->left == NULL && root->right == NULL)
        printf("%d ", root->data);

    // recur for left and right subtrees
    printLeafNodes(root->left);
    printLeafNodes(root->right);
}

// main function
int main()
{
    // construct a binary tree
    struct Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
root->left->left = newNode(4);
    root->right->left = newNode(5);

    // print leaf nodes of the binary tree
    printLeafNodes(root);

    return 0;
}
```

Output:



```
4 5
Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

Case – 1 For 1,2,3,4,5, Element’s leaf nodes is 4 and 5

❑ Analysis and Discussion :

- We got the exact result on output. Sometimes the result was wrong but we found the right implementation.
- The problem of displaying anything in output is the easiest implementation. We solve that very easily.
- In this assignment, we faced some problems in this question but with the teacher's help we solve it.
- All program is easy to understand and these helped me a lot to remove my confusion about BST programming and BST traversing operations.
- I learnt display something in program, BST traversing operations , print element address, find leaf nodes from BST, switch statement and application of user-defined function etc on program and many basic things about c programming.