

**Data Structures and Algorithms**

**( CS09203 )**

**Lab Report**

Name: Wajeeh ul Hassan

Registration #: SEU-F16-151

Lab Report #: 09

Dated: 04-06-2018

Submitted To: Sir. Usman Ahmed

The University of Lahore, Islamabad Campus

Department of Computer Science & Information Technology

**Experiment # 09**

**Implementation of Binary Search Tree graph**

**Objective**

The objective of this session is to create the tree for binary search.

**Software Tool**

1. I use Code Blocks with GCC compiler.

# Theory

This section discusses how to create the graph and tell the number of edges and vertices . Graphs are used to model electrical circuits, chemical compounds, highway maps, and so on. They are also used in the analysis of electrical circuits, finding the shortest route, project planning, linguistics, genetics, social science, and so forth Undirected Edge - An undirected egde is a bidirectional edge. If there is a undirected edge between vertices A and B then edge (A , B) is equal to edge (B , A). Directed Edge - A directed egde is a unidirectional edge. If there is a directed edge between vertices A and B then edge (A , B) is not equal to edge (B , A). Weighted Edge - A weighted egde is an edge with cost on it.

# Task

## Procedure: Task 5

Write a C++ code using functions for the following operations. 1.Binary search tree

## 2.2

### #include*<*iostream*>* using namespace std ;

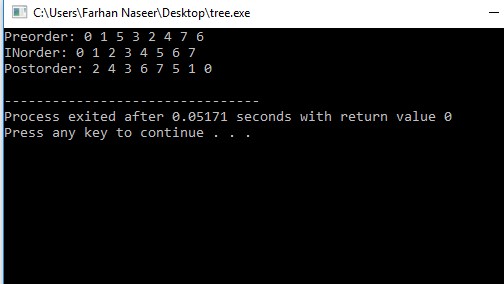


Figure 1: output

**struct** node{ **char** data ; **struct** node∗ l e f t ; **struct** node∗ right ;

};

**void** Preorder ( **struct** node ∗root ){ **if** ( root == NULL) **return** ;

cout*<<*root−*>*data*<<*”” ; Preorder ( root−*>*l e f t );

Preorder ( root−*>*right );

}

**void** Inorder ( **struct** node ∗root ){ **if** ( root == NULL)

**return** ; Inorder ( root−*>*l e f t ); cout*<<*root−*>*data*<<*”” ;

Inorder ( root−*>*right );

}

**void** Postorder ( **struct** node ∗root ){ **if** ( root == NULL)

### return ;

Postorder ( root−*>*l e f t ); Postorder ( root−*>*right ); cout*<<*root−*>*data*<<*”” ;

|  |  |  |  |
| --- | --- | --- | --- |
| } | |  |  |
| node∗ | | Insert (node ∗root , **char** data ){ **if** ( root == NULL){ root = **new** node (); root−*>*data = data ;  root−*>*l e f t = root−*>*right = NULL;  }  **else if** ( data *<*= root−*>*data ) root−*>*l e f t = Insert ( root−*>*left , data );  **else** root−*>*right = Insert ( root−*>*right , data );  **return** root ; |  |
| } |  |
| **int** | main(){ node∗ root = NULL; | |  |
|  | root = Insert ( root , ’0 ’ ); root = Insert ( root , | | ’1 ’ ); |
|  | root = Insert ( root , ’5 ’ ); root = Insert ( root , | | ’7 ’ ); |
|  | root = Insert ( root , ’6 ’ ); root = Insert ( root , | | ’3 ’ ); |
|  | root = Insert ( root , ’4 ’ ); root = Insert ( root , | | ’2 ’ ); |

cout*<<*”Preorder :” ; Preorder ( root ); cout*<<*”\n” ; cout*<<*”INorder :” ; Inorder ( root ); cout*<<*”\n” ; cout*<<*”Postorder :” ; Postorder ( root ); cout*<<*”\n” ;

}

# Conclusion

In today lab we have discussed how we can create a tree for binary search and how to display it on a screen by a code.