

Experiments\binary_search_tree..c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4
5  //Represent a node of binary tree
6  struct node{
7      int data;
8      struct node *left;
9      struct node *right;
10 };
11
12 //Represent the root of binary tree
13 struct node *root = NULL;
14
15 //createNode() will create a new node
16 struct node* createNode(int data){
17     //Create a new node
18     struct node *newNode = (struct node*)malloc(sizeof(struct node));
19     //Assign data to newNode, set left and right child to NULL
20     newNode->data = data;
21     newNode->left = NULL;
22     newNode->right = NULL;
23
24     return newNode;
25 }
26
27 //Represent a queue
28 struct queue
29 {
30     int front, rear, size;
31     struct node* *arr;
32 };
33
34 //createQueue() will create a queue
35 struct queue* createQueue()
36 {
37     struct queue* newQueue = (struct queue*) malloc(sizeof( struct queue ));
38
39     newQueue->front = -1;
40     newQueue->rear = 0;
41     newQueue->size = 0;
42
43     newQueue->arr = (struct node**) malloc(100 * sizeof( struct node* ));
44
45     return newQueue;
46 }
47
48 //Adds a node to queue
```

```
49 void enqueue(struct queue* queue, struct node *temp){
50     queue->arr[queue->rear++] = temp;
51     queue->size++;
52 }
53
54 //Deletes a node from queue
55 struct node *dequeue(struct queue* queue){
56     queue->size--;
57     return queue->arr[++queue->front];
58 }
59
60
61 //insertNode() will add new node to the binary tree
62 void insertNode(int data) {
63     //Create a new node
64     struct node *newNode = createNode(data);
65     //Check whether tree is empty
66     if(root == NULL){
67         root = newNode;
68         return;
69     }
70     else {
71         //Queue will be used to keep track of nodes of tree level-wise
72         struct queue* queue = createQueue();
73         //Add root to the queue
74         enqueue(queue, root);
75
76         while(true) {
77             struct node *node = dequeue(queue);
78             //If node has both left and right child, add both the child to queue
79             if(node->left != NULL && node->right != NULL) {
80                 enqueue(queue, node->left);
81                 enqueue(queue, node->right);
82             }
83             else {
84                 //If node has no left child, make newNode as left child
85                 if(node->left == NULL) {
86                     node->left = newNode;
87                     enqueue(queue, node->left);
88                 }
89                 //If node has left child but no right child, make newNode as right child
90                 else {
91                     node->right = newNode;
92                     enqueue(queue, node->right);
93                 }
94                 break;
95             }
96         }
97     }
98 }
```

```
99
100 //inorder() will perform inorder traversal on binary search tree
101 void inorderTraversal(struct node *node) {
102     //Check whether tree is empty
103     if(root == NULL){
104         printf("Tree is empty\n");
105         return;
106     }
107     else {
108
109         if(node->left != NULL)
110             inorderTraversal(node->left);
111         printf("%d ", node->data);
112         if(node->right != NULL)
113             inorderTraversal(node->right);
114
115     }
116 }
117
118 int main(){
119
120     //Add nodes to the binary tree
121     insertNode(1);
122     //1 will become root node of the tree
123     printf("Binary tree after insertion: \n");
124     //Binary after inserting nodes
125     inorderTraversal(root);
126
127     insertNode(2);
128     insertNode(3);
129     //2 will become left child and 3 will become right child of root node 1
130     printf("\nBinary tree after insertion: \n");
131     //Binary after inserting nodes
132     inorderTraversal(root);
133
134     insertNode(4);
135     insertNode(5);
136     //4 will become left child and 5 will become right child of node 2
137     printf("\nBinary tree after insertion: \n");
138     //Binary after inserting nodes
139     inorderTraversal(root);
140
141     insertNode(6);
142     insertNode(7);
143     //6 will become left child and 7 will become right child of node 3
144     printf("\nBinary tree after insertion: \n");
145     //Binary after inserting nodes
146     inorderTraversal(root);
147
148     return 0;
```

```
149 }
150 /*
151 Output:
152 Binary tree after insertion:
153 1
154 Binary tree after insertion:
155 2 1 3
156 Binary tree after insertion:
157 4 2 5 1 3
158 Binary tree after insertion:
159 4 2 5 1 6 3 7
160 */
```