

**CCC 204 Data Structures and Algorithms LABORATORY**  
**REPORT :**  
**LAB 9# - BST**  
Samano, Rajel Johann  
Information Technology Program, CABECS  
Colegio San Agustin – Bacolod

**I. INTRODUCTION**

For Laboratory Activity Number Nine is to follow the objectives of the exercises and answer questions. The Three main objectives as follows:

- **Define Binary Search Trees (BST)**
- **Recognize characteristics and applications of BST**
- **Observe, analyze and run BST implementations in code**

**II. IMPLEMENTATION / APPROACH**

Figure 1-7. LA9\_codeTasks.c with output

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct node {
5      int key;
6      struct node *left, *right;
7  };
8
9  // Create a node
10 struct node *newNode(int item) {
11     struct node *temp = (struct node *)malloc(sizeof(struct node));
12     temp->key = item;
13     temp->left = temp->right = NULL;
14     return temp;
15 }
16
17 // Function Prototype
18 void inorder(struct node *root);
19 struct node *insert(struct node *node, int key);
20 struct node *insert(struct node *node, int key);
21 struct node *minValueNode(struct node *node);
22 struct node *deleteNode(struct node *root, int key);
23 struct node* search(struct node *root, int key);
24 void BSTmenu();
25 // Driver code
26 int main() {
27     int key;
28     int generate = 1;
29     char input[2];
30     struct node *root = NULL;
31
32     while (generate) {
33         char BSTselection;
34         do {
35             BSTmenu();
36             printf("Input: ");
37             scanf(" %c", &BSTselection);
38
39             switch (BSTselection) {
40                 case '1':
41                     printf("\nInsert Value to BST: ");
42                     scanf("%d", &key);
43                     root = insert(root, key);
44                     break;
45
46                 case '2':
47                     printf("\nSearch Value in BST: ");
48                     scanf("%d", &key);
49                     struct node *result = search(root, key);
50                     printf("\n");
51
52                     if (result != NULL){
53                         printf("\n[=====]\n");
54                         printf("\nElement with Key %d: Found", result->key);
55                         printf("\n[=====]\n");
56                     }else{
57                         printf("\n[=====]\n");
58                         printf("\nElement with Key %d: Not Found", result->key);
59                         printf("\n[=====]\n");
60                     }
61                     break;
62
63                 case '3':
64                     printf("\nDelete Value in BST: ");
65                     scanf("%d", &key);
66                     root = deleteNode(root, key);
67                     printf("\n");
68                     printf("\n[=====]\n");
69                     printf("\nAfter Deleting: ");
70                     inorder(root);
71                     printf("\n[=====]\n");
72                     break;
73
74                 case '4':
75                     printf("\n[=====]\n");
76                     printf("\nInorder traversal: ");
77                     inorder(root);
78                     printf("\n[=====]\n");
79                     break;
80
81                 case '5':
82                     break;
83
84                 default:
85                     printf("Invalid choice. Please enter a valid option.\n");
86                     break;
87             } while (BSTselection != '5');
88
89             printf("\nPress any key to regenerate again. Press 'x' to quit: ");
90             scanf(" %s", input);
91
92             if (input[0] == 'x' || input[0] == 'X') {
93                 generate = 0;
94                 printf("\n");
95             }
96         }
97     }
98     return 0;
99 }
```

```

101 // Inorder Traversal
102 void inorder(struct tnode *root) {
103     if (root != NULL) {
104         // Traverse left
105         inorder(root->left);
106
107         // Traverse root
108         printf("-> %d ", root->key);
109
110         // Traverse right
111         inorder(root->right);
112     }
113 }
114
115 // Insert a node
116 struct tnode *insert(struct tnode *node, int key) {
117     // Return a new node if the tree is empty
118     if (node == NULL) return newNode(key);
119
120     // Traverse to the right place and insert the node
121     if (key < node->key)
122         node->left = insert(node->left, key);
123     else
124         node->right = insert(node->right, key);
125
126     return node;
127 }
128
129 // Find the inorder successor
130 struct tnode *minValueNode(struct tnode *node) {
131     struct tnode *current = node;
132
133     // Find the leftmost leaf
134     while (current && current->left != NULL)
135         current = current->left;
136
137     return current;
138 }
139
140 // Deleting a node
141 struct tnode *deleteNode(struct tnode *root, int key) {
142     // Return if the tree is empty
143     if (root == NULL) return root;
144
145     // Find the node to be deleted
146     if (key < root->key)
147         root->left = deleteNode(root->left, key);
148     else if (key > root->key)
149         root->right = deleteNode(root->right, key);
150     else {
151         // If the node is with only one child or no child
152         if (root->left == NULL) {
153             struct tnode *temp = root->right;
154             free(root);
155             return temp;
156         } else if (root->right == NULL) {
157             struct tnode *temp = root->left;
158             free(root);
159             return temp;
160         }
161
162         // If the node has two children
163         struct tnode *temp = minValueNode(root->right);
164
165         // Place the inorder successor in position of the node to be deleted
166         root->key = temp->key;
167
168         // Delete the inorder successor
169         root->right = deleteNode(root->right, temp->key);
170     }
171
172     return root;
173 }
174
175 //search
176 struct tnode* search(struct tnode *root, int key){
177     printf("\n[=====]\n");
178     printf("\n Visiting elements: ");
179     while(root->key != key) {
180         if(root != NULL) {
181             printf("-> %d ",root->key);
182
183             //go to left tree
184             if(root->key > key) {
185                 root = root->left;
186             } //else go to right tree
187             else {
188                 root = root->right;
189             }
190
191             //not found
192             if(root == NULL) {
193                 return NULL;
194             }
195         }
196     }
197     printf("\n[=====]\n");
198     return root;
199 }
200
201 void BSTmenu() {
202     printf("\n=====");
203     printf("\n||<<<< LA9 Task Code >>>>||\n");
204     printf("\n|| [1] Insert || [2] Search ||\n");
205     printf("\n|| [3] Delete || [4] Display ||\n");
206     printf("\n|| [5] Exit ||\n");
207     printf("\n||<<<< Select Command >>>>||\n");
208     printf("\n=====");
209 }
210
211 int main() {
212     struct tnode *root = NULL;
213     int key;
214     int choice;
215     int cmd;
216     int searchKey;
217     int insertValue;
218     int displayCount;
219     int searchCount;
220     int deleteCount;
221     int exitCount;
222     int selectCommand;
223     int input;
224     int searchValue;
225     int insertValue;
226     int displayCount;
227     int searchCount;
228     int deleteCount;
229     int exitCount;
230     int selectCommand;
231     int input;
232     int searchValue;
233     int insertValue;
234     int displayCount;
235     int searchCount;
236     int deleteCount;
237     int exitCount;
238     int selectCommand;
239     int input;
240     int searchValue;
241     int insertValue;
242     int displayCount;
243     int searchCount;
244     int deleteCount;
245     int exitCount;
246     int selectCommand;
247     int input;
248     int searchValue;
249     int insertValue;
250     int displayCount;
251     int searchCount;
252     int deleteCount;
253     int exitCount;
254     int selectCommand;
255     int input;
256     int searchValue;
257     int insertValue;
258     int displayCount;
259     int searchCount;
260     int deleteCount;
261     int exitCount;
262     int selectCommand;
263     int input;
264     int searchValue;
265     int insertValue;
266     int displayCount;
267     int searchCount;
268     int deleteCount;
269     int exitCount;
270     int selectCommand;
271     int input;
272     int searchValue;
273     int insertValue;
274     int displayCount;
275     int searchCount;
276     int deleteCount;
277     int exitCount;
278     int selectCommand;
279     int input;
280     int searchValue;
281     int insertValue;
282     int displayCount;
283     int searchCount;
284     int deleteCount;
285     int exitCount;
286     int selectCommand;
287     int input;
288     int searchValue;
289     int insertValue;
290     int displayCount;
291     int searchCount;
292     int deleteCount;
293     int exitCount;
294     int selectCommand;
295     int input;
296     int searchValue;
297     int insertValue;
298     int displayCount;
299     int searchCount;
300     int deleteCount;
301     int exitCount;
302     int selectCommand;
303     int input;
304     int searchValue;
305     int insertValue;
306     int displayCount;
307     int searchCount;
308     int deleteCount;
309     int exitCount;
310     int selectCommand;
311     int input;
312     int searchValue;
313     int insertValue;
314     int displayCount;
315     int searchCount;
316     int deleteCount;
317     int exitCount;
318     int selectCommand;
319     int input;
320     int searchValue;
321     int insertValue;
322     int displayCount;
323     int searchCount;
324     int deleteCount;
325     int exitCount;
326     int selectCommand;
327     int input;
328     int searchValue;
329     int insertValue;
330     int displayCount;
331     int searchCount;
332     int deleteCount;
333     int exitCount;
334     int selectCommand;
335     int input;
336     int searchValue;
337     int insertValue;
338     int displayCount;
339     int searchCount;
340     int deleteCount;
341     int exitCount;
342     int selectCommand;
343     int input;
344     int searchValue;
345     int insertValue;
346     int displayCount;
347     int searchCount;
348     int deleteCount;
349     int exitCount;
350     int selectCommand;
351     int input;
352     int searchValue;
353     int insertValue;
354     int displayCount;
355     int searchCount;
356     int deleteCount;
357     int exitCount;
358     int selectCommand;
359     int input;
360     int searchValue;
361     int insertValue;
362     int displayCount;
363     int searchCount;
364     int deleteCount;
365     int exitCount;
366     int selectCommand;
367     int input;
368     int searchValue;
369     int insertValue;
370     int displayCount;
371     int searchCount;
372     int deleteCount;
373     int exitCount;
374     int selectCommand;
375     int input;
376     int searchValue;
377     int insertValue;
378     int displayCount;
379     int searchCount;
380     int deleteCount;
381     int exitCount;
382     int selectCommand;
383     int input;
384     int searchValue;
385     int insertValue;
386     int displayCount;
387     int searchCount;
388     int deleteCount;
389     int exitCount;
390     int selectCommand;
391     int input;
392     int searchValue;
393     int insertValue;
394     int displayCount;
395     int searchCount;
396     int deleteCount;
397     int exitCount;
398     int selectCommand;
399     int input;
400     int searchValue;
401     int insertValue;
402     int displayCount;
403     int searchCount;
404     int deleteCount;
405     int exitCount;
406     int selectCommand;
407     int input;
408     int searchValue;
409     int insertValue;
410     int displayCount;
411     int searchCount;
412     int deleteCount;
413     int exitCount;
414     int selectCommand;
415     int input;
416     int searchValue;
417     int insertValue;
418     int displayCount;
419     int searchCount;
420     int deleteCount;
421     int exitCount;
422     int selectCommand;
423     int input;
424     int searchValue;
425     int insertValue;
426     int displayCount;
427     int searchCount;
428     int deleteCount;
429     int exitCount;
430     int selectCommand;
431     int input;
432     int searchValue;
433     int insertValue;
434     int displayCount;
435     int searchCount;
436     int deleteCount;
437     int exitCount;
438     int selectCommand;
439     int input;
440     int searchValue;
441     int insertValue;
442     int displayCount;
443     int searchCount;
444     int deleteCount;
445     int exitCount;
446     int selectCommand;
447     int input;
448     int searchValue;
449     int insertValue;
450     int displayCount;
451     int searchCount;
452     int deleteCount;
453     int exitCount;
454     int selectCommand;
455     int input;
456     int searchValue;
457     int insertValue;
458     int displayCount;
459     int searchCount;
460     int deleteCount;
461     int exitCount;
462     int selectCommand;
463     int input;
464     int searchValue;
465     int insertValue;
466     int displayCount;
467     int searchCount;
468     int deleteCount;
469     int exitCount;
470     int selectCommand;
471     int input;
472     int searchValue;
473     int insertValue;
474     int displayCount;
475     int searchCount;
476     int deleteCount;
477     int exitCount;
478     int selectCommand;
479     int input;
480     int searchValue;
481     int insertValue;
482     int displayCount;
483     int searchCount;
484     int deleteCount;
485     int exitCount;
486     int selectCommand;
487     int input;
488     int searchValue;
489     int insertValue;
490     int displayCount;
491     int searchCount;
492     int deleteCount;
493     int exitCount;
494     int selectCommand;
495     int input;
496     int searchValue;
497     int insertValue;
498     int displayCount;
499     int searchCount;
500     int deleteCount;
501     int exitCount;
502     int selectCommand;
503     int input;
504     int searchValue;
505     int insertValue;
506     int displayCount;
507     int searchCount;
508     int deleteCount;
509     int exitCount;
510     int selectCommand;
511     int input;
512     int searchValue;
513     int insertValue;
514     int displayCount;
515     int searchCount;
516     int deleteCount;
517     int exitCount;
518     int selectCommand;
519     int input;
520     int searchValue;
521     int insertValue;
522     int displayCount;
523     int searchCount;
524     int deleteCount;
525     int exitCount;
526     int selectCommand;
527     int input;
528     int searchValue;
529     int insertValue;
530     int displayCount;
531     int searchCount;
532     int deleteCount;
533     int exitCount;
5
```

My approach towards the problem was first copy that code that our teacher has provided to us and modify it by copying the code that I used for stacks and queues menu from the past activities while changing some functions output and moving the functions in the code to the bottom and creating function prototypes so that it would run properly.

### **III. EXPERIMENTAL FINDINGS / DISCUSSIONS**

What I found through various scanning from websites was that BST is very efficient as it does not use additional memory for pointers or other data structures. While compared to others in terms of searching, inserting, and deleting. (Depends on the specific characteristics of the data and the way the tree is balanced.)

### **IV. CONCLUSIONS**

To conclude this laboratory activity, I have somewhat grasped on how to use the searching, inserting, and deleting of BST or Binary Search Tree. While also kind of knowing how to make a BST from scratch.

### **V. References:**

<https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-binary-search-tree/>

<https://www.javatpoint.com/binary-search-tree>

[https://www.tutorialspoint.com/data\\_structures\\_algorithms/binary\\_search\\_tree.htm](https://www.tutorialspoint.com/data_structures_algorithms/binary_search_tree.htm)