1. *\* C++ Program To Implement BST*
2. *\*/*
3. # include <iostream>
4. # include <cstdlib>
5. using namespace std;
6. */\**
7. *\* Node Declaration*
8. *\*/*
9. struct node
10. {
11. int info;
12. struct node \*left;
13. struct node \*right;
14. }\*root;
16. */\**
17. *\* Class Declaration*
18. *\*/*
19. class BST
20. {
21. public:
22. void find(int, node \*\*, node \*\*);
23. void insert(node \*, node \*);
24. void del(int);
25. void case\_a(node \*,node \*);
26. void case\_b(node \*,node \*);
27. void case\_c(node \*,node \*);
28. void preorder(node \*);
29. void inorder(node \*);
30. void postorder(node \*);
31. void display(node \*, int);
32. BST()
33. {
34. root = NULL;
35. }
36. };
37. */\**
38. *\* Main Contains Menu*
39. *\*/*
40. int main()
41. {
42. int choice, num;
43. BST bst;
44. node \*temp;
45. while (1)
46. {
47. cout<<"-----------------"<<endl;
48. cout<<"Operations on BST"<<endl;
49. cout<<"-----------------"<<endl;
50. cout<<"1.Insert Element "<<endl;
51. cout<<"2.Delete Element "<<endl;
52. cout<<"3.Inorder Traversal"<<endl;
53. cout<<"4.Preorder Traversal"<<endl;
54. cout<<"5.Postorder Traversal"<<endl;
55. cout<<"6.Display"<<endl;
56. cout<<"7.Quit"<<endl;
57. cout<<"Enter your choice : ";
58. cin>>choice;
59. switch(choice)
60. {
61. case 1:
62. temp = new node;
63. cout<<"Enter the number to be inserted : ";
64. cin>>temp->info;
65. bst.insert(root, temp);
66. case 2:
67. if (root == NULL)
68. {
69. cout<<"Tree is empty, nothing to delete"<<endl;
70. }
71. Else
72. {
73. cout<<"Enter the number to be deleted : ";
74. cin>>num;
75. bst.del(num);
76. }
77. break;
78. case 3:
79. cout<<"Inorder Traversal of BST:"<<endl;
80. bst.inorder(root);
81. cout<<endl;
82. break;
83. case 4:
84. cout<<"Preorder Traversal of BST:"<<endl;
85. bst.preorder(root);
86. cout<<endl;
87. break;
88. case 5:
89. cout<<"Postorder Traversal of BST:"<<endl;
90. bst.postorder(root);
91. cout<<endl;
92. break;
93. case 6:
94. cout<<"Display BST:"<<endl;
95. bst.display(root,1);
96. cout<<endl;
97. break;
98. case 7:
99. exit(1);
100. default:
101. cout<<"Wrong choice"<<endl;
102. }
103. }
104. }
106. */\**
107. *\* Find Element in the Tree*
108. *\*/*
109. void BST::find(int item, node \*\*par, node \*\*loc)
110. {
111. node \*ptr, \*ptrsave;
112. if (root == NULL)
113. {
114. \*loc = NULL;
115. \*par = NULL;
116. return;
117. }
118. if (item == root->info)
119. {
120. \*loc = root;
121. \*par = NULL;
122. return;
123. }
124. if (item < root->info)
125. ptr = root->left;
126. else
127. ptr = root->right;
128. ptrsave = root;
129. while (ptr != NULL)
130. {
131. if (item == ptr->info)
132. {
133. \*loc = ptr;
134. \*par = ptrsave;
135. return;
136. }
137. ptrsave = ptr;
138. if (item < ptr->info)
139. ptr = ptr->left;
140. else
141. ptr = ptr->right;
142. }
143. \*loc = NULL;
144. \*par = ptrsave;
145. }
147. */\**
148. *\* Inserting Element into the Tree*
149. *\*/*
150. void BST::insert(node \*tree, node \*newnode)
151. {
152. if (root == NULL)
153. {
154. root = new node;
155. root->info = newnode->info;
156. root->left = NULL;
157. root->right = NULL;
158. cout<<"Root Node is Added"<<endl;
159. return;
160. }
161. if (tree->info == newnode->info)
162. {
163. cout<<"Element already in the tree"<<endl;
164. return;
165. }
166. if (tree->info > newnode->info)
167. {
168. if (tree->left != NULL)
169. {
170. insert(tree->left, newnode);
171. }
172. else
173. {
174. tree->left = newnode;
175. (tree->left)->left = NULL;
176. (tree->left)->right = NULL;
177. cout<<"Node Added To Left"<<endl;
178. return;
179. }
180. }
181. else
182. {
183. if (tree->right != NULL)
184. {
185. insert(tree->right, newnode);
186. }
187. else
188. {
189. tree->right = newnode;
190. (tree->right)->left = NULL;
191. (tree->right)->right = NULL;
192. cout<<"Node Added To Right"<<endl;
193. return;
194. }
195. }
196. }
198. */\**
199. *\* Delete Element from the tree*
200. *\*/*
201. void BST::del(int item)
202. {
203. node \*parent, \*location;
204. if (root == NULL)
205. {
206. cout<<"Tree empty"<<endl;
207. return;
208. }
209. find(item, &parent, &location);
210. if (location == NULL)
211. {
212. cout<<"Item not present in tree"<<endl;
213. return;
214. }
215. if (location->left == NULL && location->right == NULL)
216. case\_a(parent, location);
217. if (location->left != NULL && location->right == NULL)
218. case\_b(parent, location);
219. if (location->left == NULL && location->right != NULL)
220. case\_b(parent, location);
221. if (location->left != NULL && location->right != NULL)
222. case\_c(parent, location);
223. free(location);
224. }
226. */\**
227. *\* Case A*
228. *\*/*
229. void BST::case\_a(node \*par, node \*loc )
230. {
231. if (par == NULL)
232. {
233. root = NULL;
234. }
235. else
236. {
237. if (loc == par->left)
238. par->left = NULL;
239. else
240. par->right = NULL;
241. }
242. }
244. */\**
245. *\* Case B*
246. *\*/*
247. void BST::case\_b(node \*par, node \*loc)
248. {
249. node \*child;
250. if (loc->left != NULL)
251. child = loc->left;
252. else
253. child = loc->right;
254. if (par == NULL)
255. {
256. root = child;
257. }
258. else
259. {
260. if (loc == par->left)
261. par->left = child;
262. else
263. par->right = child;
264. }
265. }
267. */\**
268. *\* Case C*
269. *\*/*
270. void BST::case\_c(node \*par, node \*loc)
271. {
272. node \*ptr, \*ptrsave, \*suc, \*parsuc;
273. ptrsave = loc;
274. ptr = loc->right;
275. while (ptr->left != NULL)
276. {
277. ptrsave = ptr;
278. ptr = ptr->left;
279. }
280. suc = ptr;
281. parsuc = ptrsave;
282. if (suc->left == NULL && suc->right == NULL)
283. case\_a(parsuc, suc);
284. else
285. case\_b(parsuc, suc);
286. if (par == NULL)
287. {
288. root = suc;
289. }
290. else
291. {
292. if (loc == par->left)
293. par->left = suc;
294. else
295. par->right = suc;
296. }
297. suc->left = loc->left;
298. suc->right = loc->right;
299. }
301. */\**
302. *\* Pre Order Traversal*
303. *\*/*
304. void BST::preorder(node \*ptr)
305. {
306. if (root == NULL)
307. {
308. cout<<"Tree is empty"<<endl;
309. return;
310. }
311. if (ptr != NULL)
312. {
313. cout<<ptr->info<<" ";
314. preorder(ptr->left);
315. preorder(ptr->right);
316. }
317. }
318. */\**
319. *\* In Order Traversal*
320. *\*/*
321. void BST::inorder(node \*ptr)
322. {
323. if (root == NULL)
324. {
325. cout<<"Tree is empty"<<endl;
326. return;
327. }
328. if (ptr != NULL)
329. {
330. inorder(ptr->left);
331. cout<<ptr->info<<" ";
332. inorder(ptr->right);
333. }
334. }
336. */\**
337. *\* Postorder Traversal*
338. *\*/*
339. void BST::postorder(node \*ptr)
340. {
341. if (root == NULL)
342. {
343. cout<<"Tree is empty"<<endl;
344. return;
345. }
346. if (ptr != NULL)
347. {
348. postorder(ptr->left);
349. postorder(ptr->right);
350. cout<<ptr->info<<" ";
351. }
352. }
354. */\**
355. *\* Display Tree Structure*
356. *\*/*
357. void BST::display(node \*ptr, int level)
358. {
359. int i;
360. if (ptr != NULL)
361. {
362. display(ptr->right, level+1);
363. cout<<endl;
364. if (ptr == root)
365. cout<<"Root->: ";
366. else
367. {
368. for (i = 0;i < level;i++)
369. cout<<" ";
370. }
371. cout<<ptr->info;
372. display(ptr->left, level+1);
373. }
374. }

$ **g++** bst.cpp

$ a.out

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 1

Enter the number to be inserted : 8

Root Node is Added

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 6

Display BST:

Root-**>**: 8

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 1

Enter the number to be inserted : 9

Node Added To Right

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 6

Display BST:

9

Root-**>**: 8

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 1

Enter the number to be inserted : 5

Node Added To Left

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 6

Display BST:

9

Root-**>**: 8

5

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 1

Enter the number to be inserted : 11

Node Added To Right

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 6

Display BST:

11

9

Root-**>**: 8

5

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 1

Enter the number to be inserted : 3

Node Added To Left

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 1

Enter the number to be inserted : 7

Node Added To Right

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 6

Display BST:

11

9

Root-**>**: 8

7

5

3

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 1

Enter the number to be inserted : 10

Node Added To Left

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 6

Display BST:

11

10

9

Root-**>**: 8

7

5

3

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 2

Enter the number to be deleted : 10

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 6

Display BST:

11

9

Root-**>**: 8

7

5

3

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 3

Inorder Traversal of BST:

3 5 7 8 9 11

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 4

Preorder Traversal of BST:

8 5 3 7 9 11

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 5

Postorder Traversal of BST:

3 7 5 11 9 8

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 2

Enter the number to be deleted : 8

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 6

Display BST:

11

Root-**>**: 9

7

5

3

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 1

Enter the number to be inserted : 10

Node Added To Left

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 6

Display BST:

11

10

Root-**>**: 9

7

5

3

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 1

Enter the number to be inserted : 15

Node Added To Right

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 6

Display BST:

15

11

10

Root-**>**: 9

7

5

3

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 4

Preorder Traversal of BST:

9 5 3 7 11 10 15

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 5

Postorder Traversal of BST:

3 7 5 10 15 11 9

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 6

Display BST:

15

11

10

Root-**>**: 9

7

5

3

-----------------

Operations on BST

-----------------

1.Insert Element

2.Delete Element

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 7