

main.c

```
1 // Implementing Red-Black Tree in C
2
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 enum nodeColor {
7     RED,
8     BLACK
9 };
10
11 struct rbNode {
12     int data, color;
13     struct rbNode *link[2];
14 };
15
16 struct rbNode *root = NULL;
17
18 // Create a red-black tree
19 struct rbNode *createNode(int data) {
20     struct rbNode *newnode;
21     newnode = (struct rbNode *)malloc(sizeof(struct rbNode));
22     newnode->data = data;
23     newnode->color = RED;
24     newnode->link[0] = newnode->link[1] = NULL;
25     return newnode;
26 }
27
28 // Insert an node
29 void insertion(int data) {
30     struct rbNode *curr, *parent, *newnode;
31 }
```

Share

Run

Output

```
/tmp/D4QjZAsCQ0.o
1. Insertion 2. Deletion
3. Traverse 4. Exit
Enter your choice:1 2 3 4 5 6 7
Enter the element to insert:
1. Insertion 2. Deletion
3. Traverse 4. Exit
Enter your choice:2
1. Insertion 2. Deletion
3. Traverse 4. Exit
Enter your choice:
== Code Execution Successful ==|
```



Type here to search



Badminton

13:45

```
main.c
>1 #include <stdio.h>, #include <stdlib.h>, #include <assert.h>
2# define MAX 100
3int arr[MAX], nL = 0, mIndex, ht = 0;
4struct Node {
5    int data;
6    struct Node *link;
7    enum {RED, BLACK} color;
8} *root;
9
10root = NULL;
11
12void printList() {
13    struct Node *ptr = root;
14    while (ptr != NULL) {
15        printf("%d ", ptr->data);
16        ptr = ptr->link;
17    }
18}
19
20int insert(int data) {
21    if (nL == MAX) {
22        printf("Stack Overflow\n");
23        return -1;
24    }
25    arr[nL] = data;
26    nL++;
27    return 1;
28}
29
30int delete(int index) {
31    if (index < 0 || index >= nL) {
32        printf("Index Out of Range\n");
33        return -1;
34    }
35    arr[index] = arr[nL - 1];
36    arr[nL - 1] = -1;
37    nL--;
38    return 1;
39}
40
41int search(int data) {
42    struct Node *ptr = root;
43    while (ptr != NULL) {
44        if (ptr->data == data) {
45            printf("Element Found at index %d\n", arr[ht]);
46            return 1;
47        }
48        ht++;
49    }
50    printf("Element Not Found\n");
51}
52
53int rotateLeft() {
54    struct Node *yPtr = stack[ht - 2];
55    stack[ht - 2] = yPtr;
56    yPtr->link = stack[ht - 1];
57    ht--;
58}
59
60int rotateRight() {
61    struct Node *yPtr = stack[ht - 1];
62    stack[ht - 1] = yPtr;
63    yPtr->link = stack[ht - 2];
64    ht--;
```

```
~ /tmp/D4QjZASCGQ.0.o
1. Insertion 2. Deletion
3. Traverse 4. Exit
Enter your choice:1 2 3 4 5 6 7
Enter the element to insert:
1. Insertion 2. Deletion
3. Traverse 4. Exit
Enter your choice:2
1. Insertion 2. Deletion
3. Traverse 4. Exit
Enter your choice:
==== Code Execution Successful ====
[
```

main.c

Run Share Output

62 xPtr = stack[ht - 1];

63 yPtr = xPtr->link[1];

64 xPtr->link[1] = yPtr->link[0];

65 yPtr->link[0] = xPtr;

66 stack[ht - 2]->link[0] = yPtr;

67 }

68 xPtr = stack[ht - 2];

69 xPtr->color = RED;

70 xPtr->link[0] = yPtr->link[1];

71 yPtr->link[1] = xPtr;

72 if (xPtr == root) {

73 root = yPtr;

74 } else {

75 stack[ht - 3]->link[dir[ht - 3]] = yPtr;

76 }

77 break;

78 }

79 }

80 }

81 yPtr = stack[ht - 2]->link[0];

82 if ((yPtr != NULL) && (yPtr->color == RED)) {

83 stack[ht - 2]->color = RED;

84 stack[ht - 1]->color = yPtr->color = BLACK;

85 ht = ht - 2;

86 }

87 }

88 if (dir[ht - 1] == 1) {

89 yPtr = stack[ht - 1];

90 } else {

91 xPtr = stack[ht - 1];

92 yPtr = xPtr->link[1];

/tmp/D4QJZAsCQ0.o

1. Insertion 2. Deletion

3. Traverse 4. Exit

Enter your choice:1 2 3 4 5 6 7

Enter the element to insert:

1. Insertion 2. Deletion

3. Traverse 4. Exit

Enter your choice:2

1. Insertion 2. Deletion

3. Traverse 4. Exit

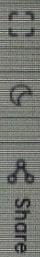
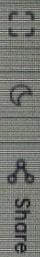
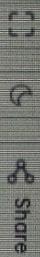
Enter your choice:

==== Code Execution Successful ===

Type here to search

## main.c

```
86* } else {
87*     if (dir[ht - 1] == 1) {
88*         yPtr = stack[ht - 1];
89*     } else {
90*         xPtr = stack[ht - 1];
91*         yPtr = xPtr->link[0];
92*         xPtr->link[0] = yPtr->link[1];
93*         yPtr->link[1] = xPtr;
94*         stack[ht - 2]->link[1] = yPtr;
95*     }
96*     xPtr = stack[ht - 2];
97*     yPtr->color = BLACK;
98*     xPtr->color = RED;
99*     xPtr->link[1] = yPtr->link[0];
100*    yPtr->link[0] = xPtr;
101*    if (xPtr == root) {
102*        root = yPtr;
103*    } else {
104*        stack[ht - 3]->link[dir[ht - 3]] = yPtr;
105*    }
106*    break;
107* }
108* }
109* }
110* root->color = BLACK;
111* }
112* }
113* // Delete a node
114* }
```



## Output

+ /tmp/D4QjZAsCQ0.o

1. Insertion 2. Deletion

3. Traverse 4. Exit

Enter your choice:1 2 3 4 5 6 7

Enter the element to insert:

1. Insertion 2. Deletion

3. Traverse 4. Exit

Enter your choice:2

1. Insertion 2. Deletion

3. Traverse 4. Exit

Enter your choice:

==== Code Execution Successful ===|

```
113 // Delete a node
114 void deletion(int data) {
115     struct rbNode *stack[98], *ptr, *xPtr, *yPtr;
116     struct rbNode *pPtr, *qPtr, *rPtr;
117     int dir[98], ht = 0, diff, i;
118     enum nodeColor color;
119
120     if (!root) {
121         printf("Tree not available\n");
122         return;
123     }
124
125     ptr = root;
126     while (ptr != NULL) {
127         if ((data - ptr->data) == 0)
128             break;
129         diff = (data - ptr->data) > 0 ? 1 : 0;
130         stack[ht] = ptr;
131         dir[ht--] = diff;
132         ptr = ptr->link[diff];
133     }
134
135     if (ptr->link[1] == NULL) {
136         if ((ptr == root) && (ptr->link[0] == NULL)) {
137             free(ptr);
138             root = NULL;
139         } else if (ptr == root) {
140             root = ptr->link[0];
141             free(ptr);
142         }
143     }
144 }
```

```
/tmp/D4QjZAsCQ0.o
1. Insertion 2. Deletion
3. Traverse 4. Exit
Enter your choice:1 2 3 4 5 6 7
Enter the element to insert:
1. Insertion 2. Deletion
3. Traverse 4. Exit
Enter your choice:2
1. Insertion 2. Deletion
3. Traverse 4. Exit
Enter your choice:
==== Code Execution Successful ===
```

Type here to search