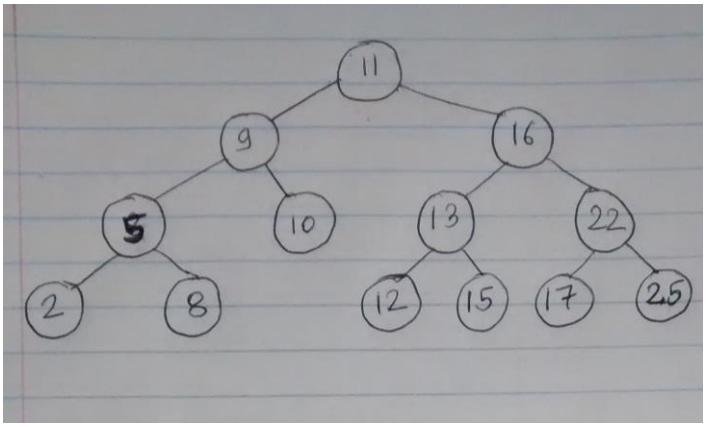
Pradip Lamsal

R11484918

Lab Session 10

1. **a.**
   The given binary tree can be restructured as a binary search tree as follows and the following binary search tree will be used as a reference for the rest of the questions:



Index	Info	Left	Right
0	11	11	12
1	2	-1	-1
2	8	-1	-1
3	12	-1	-1
4	15	-1	-1
5	17	-1	-1
6	25	-1	-1
7	5	1	2
8	10	-1	-1
9	13	3	4
10	22	5	6
11	9	7	8
12	16	9	10

b.

Code with functions:

```c
#include <stdio.h>
#include <stdlib.h>


struct node {
    int key;
    struct node *left;
    struct node *right;
};


struct node* tree[13] = {0};
int i = 0;


void insert(int key, int index, struct node* left, struct node* right) {
    struct node* temp = (struct node*) malloc(sizeof(struct node*));
    temp->key = key;
    temp->left = left;
    temp->right = right;
    tree[index] = temp;
}


int search(struct node* Node, int value) {
        if(Node == NULL) return 0;
    else if(Node->key == value) return 1;
    else if(Node->key >= value) return search(Node->left, value);
    else return search(Node->right, value);
}



struct node* MinValue(struct node* Node)
{
    while(Node->left != NULL) Node = Node->left;
    return Node;
}
```

```c
struct node* deletenode(struct node *Node, int value) {
    if(Node == NULL) return Node;
    else if(value < Node->key) Node->left = deletenode(Node->left, value);
    else if(value > Node->key) Node->right = deletenode(Node->right, value);

    else {
        if(Node->left == NULL && Node->right == NULL) {  //for leaf nodes
            free(Node);
            Node = NULL;
        }
        //Case 2: One child
        else if(Node->left == NULL) {                   //for nodes with single child
            struct node *temp = Node;
            Node = Node->right;
            free (temp);
        }


        else if(Node->right == NULL) {                   //for nodes with single child
            struct node *temp = Node;
            Node = Node->left;
            free (temp);
        }

        else {                                           //for nodes with two children
            struct node *temp = MinValue(Node->right);
            Node->key = temp->key;
            Node->right = deletenode(Node->right,temp->key);
        }
    }
    return Node;
}
```

```c
int main() {

    insert(2, 1, NULL, NULL);
    insert(8,2, NULL, NULL);
    insert(12, 3, NULL, NULL);
    insert(15, 4, NULL, NULL);
    insert(17, 5, NULL, NULL);
    insert(25, 6, NULL, NULL);
    insert(5, 7, tree[1], tree[2]);
    insert(10, 8, NULL, NULL);
    insert(13, 9, tree[3], tree[4]);
    insert(22, 10, tree[5], tree[6]);
    insert(9, 11, tree[7], tree[8]);
    insert(16, 12, tree[9], tree[10]);
    insert(11, 0, tree[11], tree[12]);


    printf("Node 16 found. The index of the node in the array is %d. \n", search(tree[0], 16));
    printf("Node 25 found. The index of the node in the array is %d. \n", search(tree[0], 25));
    printf("Node 9 found. The index of the node in the array is %d. \n", search(tree[0], 9));
    printf("Node 22 found. The index of the node in the array is %d. \n\n", search(tree[0], 22));


    printf("Node %d is successfully deleted\n", deletenode(tree[11], 9)->key);
    printf("Node %d is successfully deleted\n", deletenode(tree[3], 12)->key);
    printf("Node %d is successfully deleted\n\n\n", deletenode(tree[6], 25)->key);

    printf("The required BFS search is: ");
    queue[++rear] = Node->key;
    bfs(Node);
    for (int i = 0;i <= rear;i++) printf("%d ", queue[i]);
    printf("%d\n", Node->right->right->right->key);

    return 0;
}
```

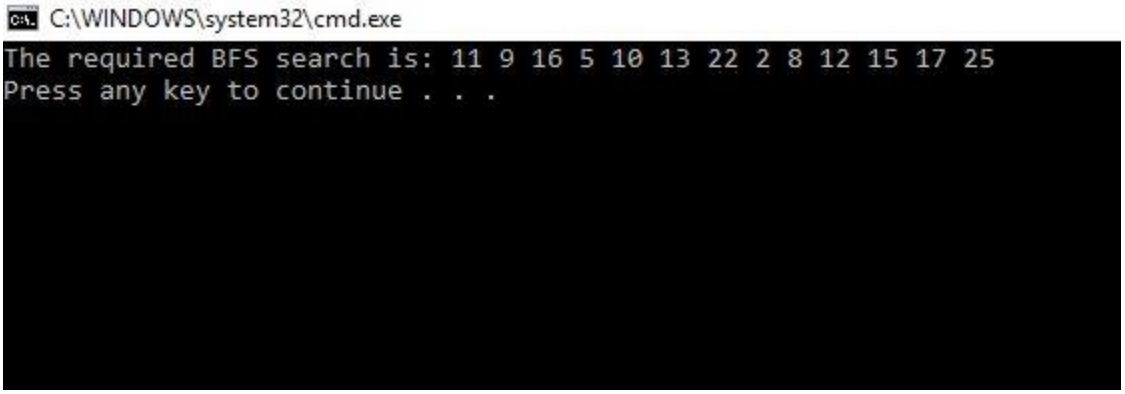
Output:

```
C:\WINDOWS\system32\cmd.exe
Node 16 found. The index of the node in the array is 12
Node 25 found. The index of the node in the array is 6
Node 9 found. The index of the node in the array is 11
Node 22 found. The index of the node in the array is 10

Node 9 is successfully deleted
Node 12 is successfully deleted
Node 25 is successfully deleted


Press any key to continue . . .
```

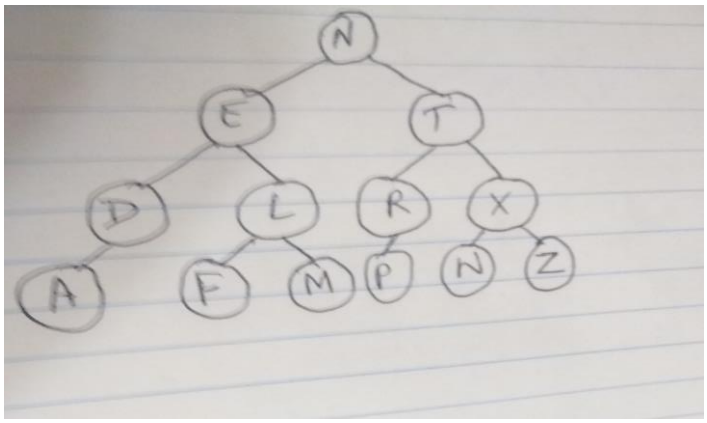
c. Breadth-first search code and functions:

```
void bfs(struct node *Node) {
    number = Node->key;
    if ((front <= rear)&&(Node->key == queue[front]))
    {
        if (Node->left != NULL) queue[++rear] = Node->left->key;
        if (Node->right != NULL || Node->right  ==  NULL) queue[++rear] = Node->right->key;
        front++;
    }
    if (Node->left != NULL) bfs(Node->left);
    if (Node->right != NULL) bfs(Node->right);

}
```

Result:



d. The question has asked to form a binary (search) tree. Based on the data given in the table, it will be a right-heavy binary search tree because 'A' seems to be the root of the tree and no element will be on left of 'A'. Therefore, I have restructured the table and the tree and will be using the following tree throughout this question:



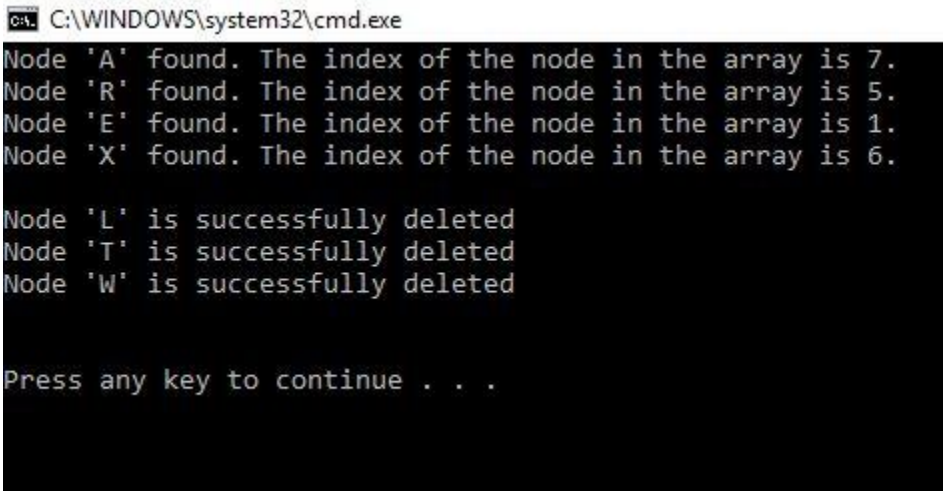
Codes and functions:

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
    int key;
    struct node *left;
    struct node *right;
};

 struct node* tree[13];
 struct node *Node = NULL;


void insert(int key, int index, struct node* left, struct node* right) {
    struct node* temp = (struct node*) malloc(sizeof(struct node*));
    temp->key = key;
    temp->left = left;
    temp->right = right;
    tree[index] = temp;
}


int search(struct node* Node, int value) {
        if(Node == NULL) return 0;
    else if(Node->key == value) return 1;
    else if(Node->key >= value) return search(Node->left, value);
    else return search(Node->right, value);
}


struct node* MinValue(struct node* Node)
{
    while(Node->left != NULL) Node = Node->left;
    return Node;
}
```

```
struct node* deletenode(struct node *Node, int value) {
    if(Node == NULL) return Node;
    else if(value < Node->key) Node->left = deletenode(Node->left, value);
    else if(value > Node->key) Node->right = deletenode(Node->right, value);

    else {
        if(Node->left == NULL && Node->right == NULL) {  //for leaf nodes
            free(Node);
            Node = NULL;
        }
        //Case 2: One child
        else if(Node->left == NULL) {                   //for nodes with single child
            struct node *temp = Node;
            Node = Node->right;
            free (temp);
        }


        else if(Node->right == NULL) {                   //for nodes with single child
            struct node *temp = Node;
            Node = Node->left;
            free (temp);
        }

        else {                                           //for nodes with two children
            struct node *temp = MinValue(Node->right);
            Node->key = temp->key;
            Node->right = deletenode(Node->right,temp->key);
        }
    }
    return Node;
}
```

```c
int main() {

    insert('E', 1, tree[3], tree[4]);
    insert('T',2, tree[5], tree[6]);
    insert('D', 3, tree[7], NULL);
    insert('L', 4, tree[8], tree[9]);
    insert('R', 5, tree[10], NULL);
    insert('X', 6, tree[11], tree[12]);
    insert('A', 7, NULL, NULL);
    insert('F', 8, NULL, NULL);
    insert('M', 9, NULL, NULL);
    insert('P', 10, NULL, NULL);
    insert('W', 11, NULL, NULL);
    insert('Z', 12, NULL, NULL);
    insert('N', 0, tree[1], tree[2]);


    printf("Node 'A' found. The index of the node in the array is %d. \n", search(tree[0], 'A'));
    printf("Node 'R' found. The index of the node in the array is %d. \n", search(tree[0], 'R'));
    printf("Node 'E' found. The index of the node in the array is %d. \n", search(tree[0], 'E'));
    printf("Node 'X' found. The index of the node in the array is %d. \n\n", search(tree[0], 'X'));


    printf("Node %c is successfully deleted\n", deletenode(tree[4], 'L')->key);
    printf("Node %c is successfully deleted\n", deletenode(tree[2], 'T')->key);
    printf("Node %c is successfully deleted\n\n\n", deletenode(tree[11],'W')->key);



    return 0;
}
```

Result:


C:\WINDOWS\system32\cmd.exe
Node 'A' found. The index of the node in the array is 7.
Node 'R' found. The index of the node in the array is 5.
Node 'E' found. The index of the node in the array is 1.
Node 'X' found. The index of the node in the array is 6.
Node 'L' is successfully deleted
Node 'T' is successfully deleted
Node 'W' is successfully deleted
Press any key to continue . . .