S.No: 1	Exp. Name: ***Project Module***	**Date:2024-06-14**

Aim:

Project Module

Source Code:

hello.c

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

// Define the structure of a node in the binary search tree
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

// Function to insert a node into the BST
struct Node* insertNode(struct Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insertNode(root->left, data);
    } else if (data > root->data) {
        root->right = insertNode(root->right, data);
    }
    return root;
}

// Function to search for a node in the BST
struct Node* searchNode(struct Node* root, int data) {
    if (root == NULL || root->data == data) {
        return root;
    }
    if (data < root->data) {
        return searchNode(root->left, data);
    }
    return searchNode(root->right, data);
}

// Function to find the minimum value node in the BST
struct Node* findMin(struct Node* root) {
    while (root->left != NULL) {
        root = root->left;
    }
    return root;
}

// Function to delete a node from the BST
struct Node* deleteNode(struct Node* root, int data) {
    if (root == NULL) {
        return root;
    }
    if (data < root->data) {
        root->left = deleteNode(root->left, data);
    } else if (data > root->data) {
        root->right = deleteNode(root->right, data);
    } else {
        // Node with only one child or no child
        if (root->left == NULL) {
            struct Node* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct Node* temp = root->left;
            free(root);
            return temp;
        }

        // Node with two children: Get the inorder successor (smallest in the
right subtree)
        struct Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

// Function to print the tree structure in a readable format
void printTree(struct Node* root, int space) {
    if (root == NULL) {
        return;
    }

    // Increase distance between levels
    space += 5;

    // Process right child first
    printTree(root->right, space);

    // Print current node after space count
    printf("\n");
    for (int i = 5; i < space; i++) {
        printf(" ");
    }
    printf("%d\n", root->data);

    // Process left child
    printTree(root->left, space);
```

```
}

int main() {
    struct Node* root = NULL;
    int n, value, choice;

    printf("Enter the number of nodes to insert: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        printf("Enter value for node %d: ", i + 1);
        scanf("%d", &value);
        root = insertNode(root, value);
    }

    printf("\nBinary Search Tree Structure:\n");
    printTree(root, 0);

    while (1) {
        printf("\nMenu:\n");
        printf("1. Insert a node\n");
        printf("2. Delete a node\n");
        printf("3. Search for a node\n");
        printf("4. Print the tree\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                root = insertNode(root, value);
                break;
            case 2:
                printf("Enter value to delete: ");
                scanf("%d", &value);
                root = deleteNode(root, value);
                break;
            case 3:
                printf("Enter value to search: ");
                scanf("%d", &value);
                struct Node* result = searchNode(root, value);
                if (result != NULL) {
                    printf("Value %d found in the tree.\n", value);
                } else {
                    printf("Value %d not found in the tree.\n", value);
                }
                break;
            case 4:
                printf("\nBinary Search Tree Structure:\n");
                printTree(root, 0);
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
            }
        }

        return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
`Hello World`