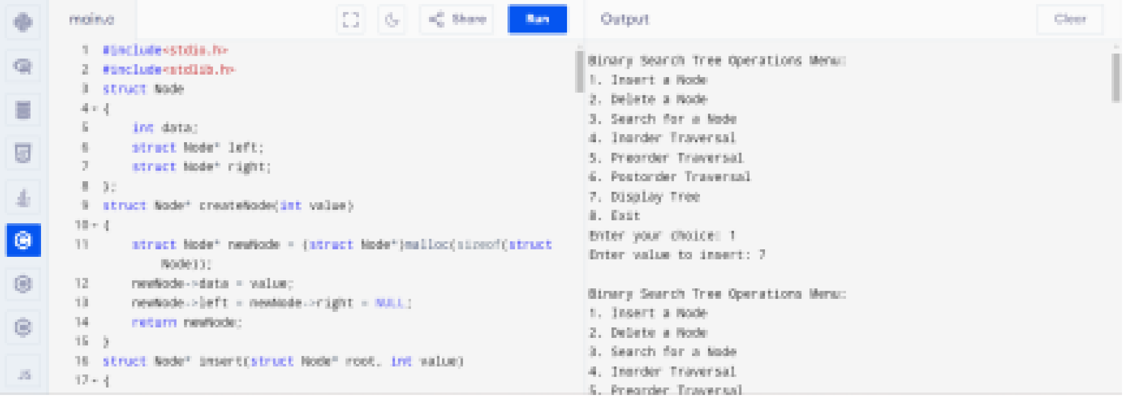
//Implemention of various operation using binary search Tree

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int data;
    struct Node* left;
    struct Node* right;
};
struct Node* createNode(int value)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}
struct Node* insert(struct Node* root, int value)
{
    if (root == NULL)
        return createNode(value);
    if (value < root->data)
        root->left = insert(root->left, value);
    else if (value > root->data)
        root->right = insert(root->right, value);
    return root;
}
struct Node* search(struct Node* root, int key)
{
    if (root == NULL || root->data == key)
        return root;
    if (key < root->data)
        return search(root->left, key);
    return search(root->right, key);
}
void inorder(struct Node* root)
{
    if (root != NULL)
    {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
```

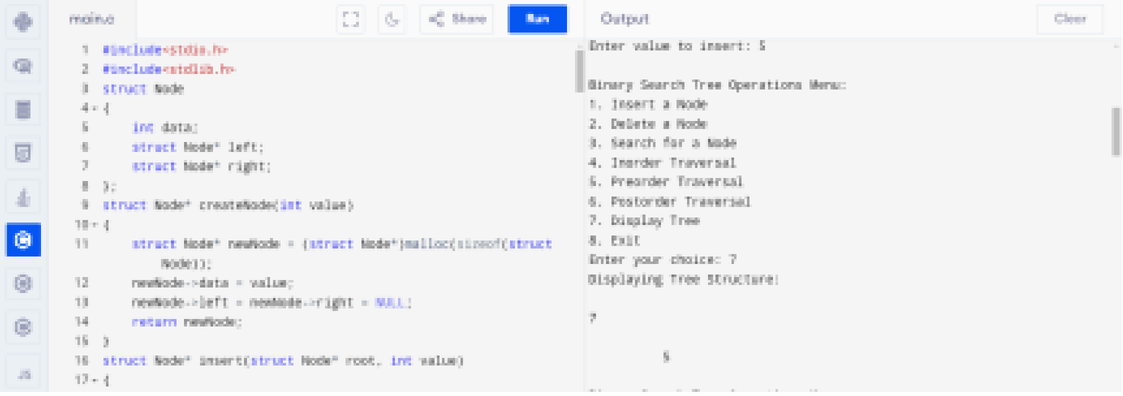
```
}
void preorder(struct Node* root)
{
    if (root != NULL)
    {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}
void postorder(struct Node* root)
{
    if (root != NULL)
    {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}
struct Node* findMin(struct Node* root)
{
    while (root && root->left != NULL)
        root = root->left;
    return root;
}
struct Node* deleteNode(struct Node* root, int key)
{
    if (root == NULL)
        return root;
    if (key < root->data)
        root->left = deleteNode(root->left, key);
    else if (key > root->data)
        root->right = deleteNode(root->right, key);
    else
    {
        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;
        }
        struct Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}
void display(struct Node* root, int space)
{
    int i;
    if (root == NULL)
        return;
    space += 10;
    display(root->right, space);
    printf("\n");
    for (i = 10; i < space; i++)
        printf(" ");
    printf("%d\n", root->data);
    display(root->left, space);
}
int main()
{
    struct Node* root = NULL;
    int choice, value, key;
    while (1)
    {
        printf("\nBinary Search Tree Operations Menu:");
        printf("\n1. Insert a Node");
        printf("\n2. Delete a Node");
        printf("\n3. Search for a Node");
        printf("\n4. Inorder Traversal");
        printf("\n5. Preorder Traversal");
        printf("\n6. Postorder Traversal");
        printf("\n7. Display Tree");
        printf("\n8. Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                root = insert(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", &key);
                if (search(root, key) != NULL)
                    printf("Node found!\n");
                else
                    printf("Node not found!\n");
                break;

            case 4:
                printf("Inorder Traversal: ");
                inorder(root);
                printf("\n");
                break;

            case 5:
                printf("Preorder Traversal: ");
                preorder(root);
                printf("\n");
                break;

            case 6:
                printf("Postorder Traversal: ");
                postorder(root);
                printf("\n");
                break;

            case 7:
                printf("Displaying Tree Structure:\n");
                display(root, 0);
                break;

            case 8:
                exit(0);

            default:
                printf("Invalid choice! Please enter a valid option.\n");
        }
    }
    return 0;
}
```

OUTPUT:



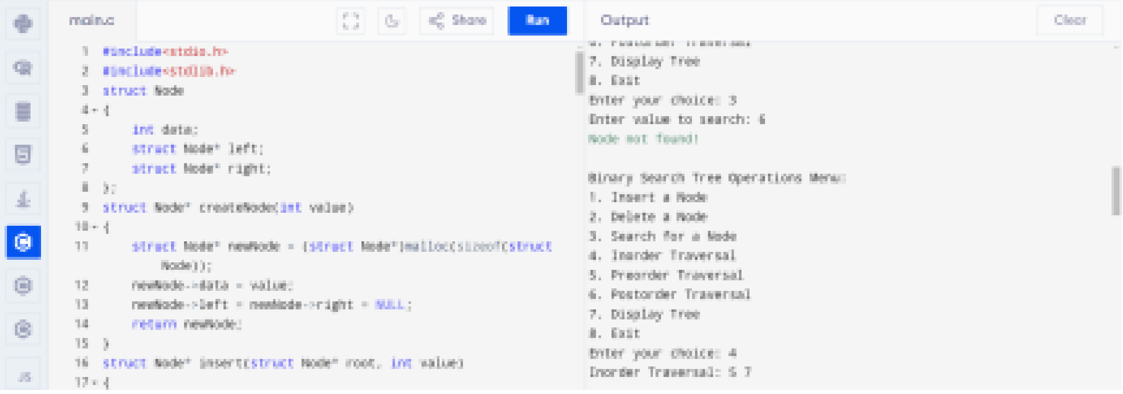
```c
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int data;
    struct Node* left;
    struct Node* right;
};
struct Node* createNode(int value)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}
struct Node* insert(struct Node* root, int value)
{
```

```
Binary Search Tree Operations Menu:
1. Insert a Node
2. Delete a Node
3. Search for a Node
4. Inorder Traversal
5. Preorder Traversal
6. Postorder Traversal
7. Display Tree
8. Exit
Enter your choice: 1
Enter value to insert: 7

Binary Search Tree Operations Menu:
1. Insert a Node
2. Delete a Node
3. Search for a Node
4. Inorder Traversal
5. Preorder Traversal
```



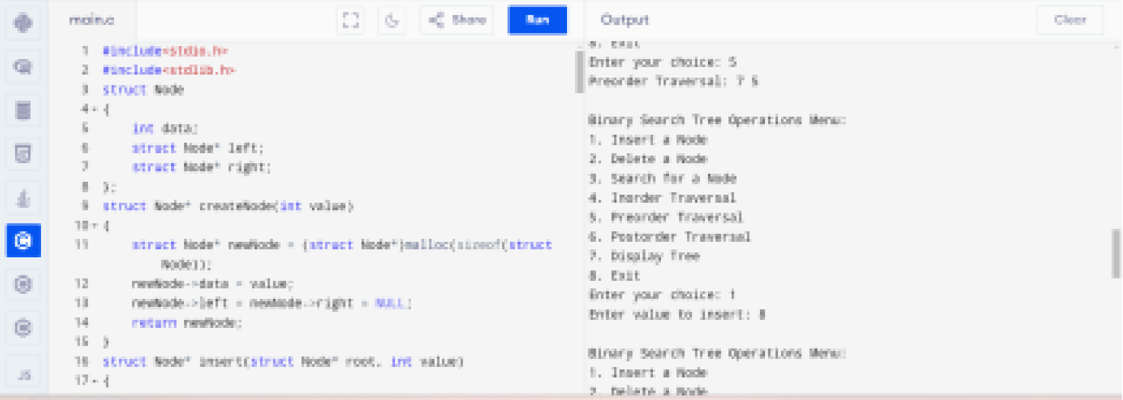
```c
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int data;
    struct Node* left;
    struct Node* right;
};
struct Node* createNode(int value)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}
struct Node* insert(struct Node* root, int value)
{
```

```
Enter value to insert: 5

Binary Search Tree Operations Menu:
1. Insert a Node
2. Delete a Node
3. Search for a Node
4. Inorder Traversal
5. Preorder Traversal
6. Postorder Traversal
7. Display Tree
8. Exit
Enter your choice: 7
Displaying Tree Structure:


7


          5
```



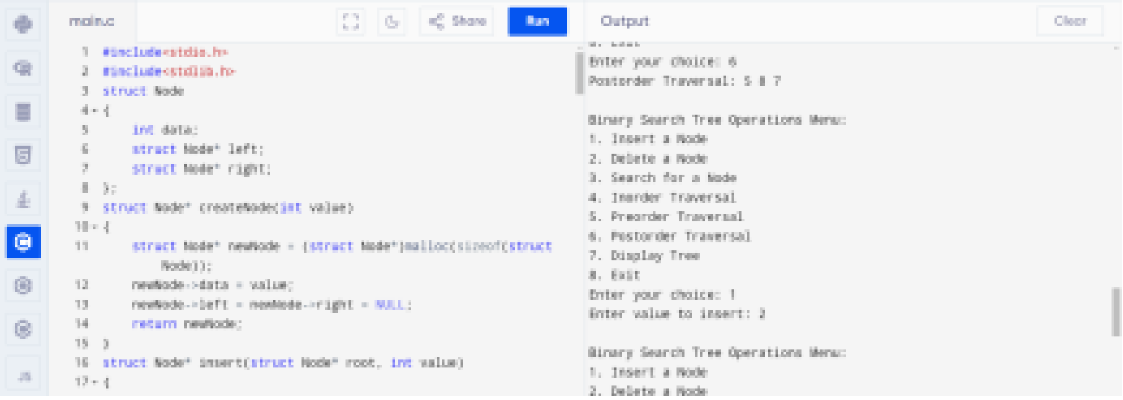
```c
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int data;
    struct Node* left;
    struct Node* right;
};
struct Node* createNode(int value)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}
struct Node* insert(struct Node* root, int value)
{
```

```
6. Postorder Traversal
7. Display Tree
8. Exit
Enter your choice: 3
Enter value to search: 6
Node not found!

Binary Search Tree Operations Menu:
1. Insert a Node
2. Delete a Node
3. Search for a Node
4. Inorder Traversal
5. Preorder Traversal
6. Postorder Traversal
7. Display Tree
8. Exit
Enter your choice: 4
Inorder Traversal: 5 7
```



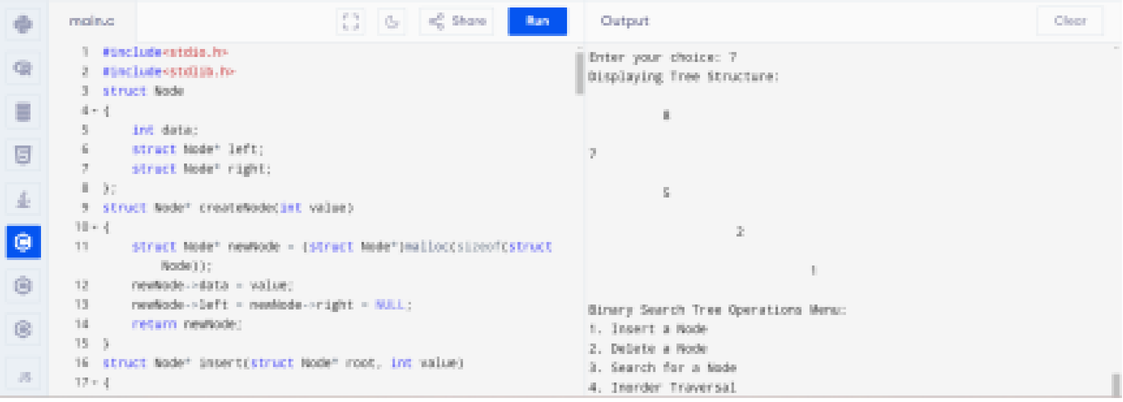
```c
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int data;
    struct Node* left;
    struct Node* right;
};
struct Node* createNode(int value)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}
struct Node* insert(struct Node* root, int value)
{
```

```
8. Exit
Enter your choice: 5
Preorder Traversal: 7 5

Binary Search Tree Operations Menu:
1. Insert a Node
2. Delete a Node
3. Search for a Node
4. Inorder Traversal
5. Preorder Traversal
6. Postorder Traversal
7. Display Tree
8. Exit
Enter your choice: 1
Enter value to insert: 8

Binary Search Tree Operations Menu:
1. Insert a Node
2. Delete a Node
```



```c
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int data;
    struct Node* left;
    struct Node* right;
};
struct Node* createNode(int value)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
        Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}
struct Node* insert(struct Node* root, int value)
{
```

```
8. Exit
Enter your choice: 6
Postorder Traversal: 5 8 7

Binary Search Tree Operations Menu:
1. Insert a Node
2. Delete a Node
3. Search for a Node
4. Inorder Traversal
5. Preorder Traversal
6. Postorder Traversal
7. Display Tree
8. Exit
Enter your choice: 1
Enter value to insert: 2

Binary Search Tree Operations Menu:
1. Insert a Node
2. Delete a Node
```



```c
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int data;
    struct Node* left;
    struct Node* right;
};
struct Node* createNode(int value)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
        Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}
struct Node* insert(struct Node* root, int value)
{
```

```
Enter your choice: 7
Displaying Tree Structure:

          8

7

          5

                    2

                              1

Binary Search Tree Operations Menu:
1. Insert a Node
2. Delete a Node
3. Search for a Node
4. Inorder Traversal
```