NATIONAL INSTITUTE OF TECHNOLOGY PUDUCHERRY

(An Institution of National Importance under MHRD, Govt. of India)

KARAIKAL – 609 609

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Roll Number:	CS19B1009	**Name**	ARUN KUMAR R
Semester:	II	**Class:**	B TECH - CSE
Subject Code:	CS106	**Subject Name:**	DATA STRUCTURES LABORATORY

1.BINARY SEARCH TREE

DATE: 22/03/20

AIM:

To implement binary search trees algorithms using c language.

ALGORITHM:

1. Start the program.
2. Declare the variables and function for insert, delete, search.
3. In binary search tree, while inserting, the smaller value is transferred to left and larger value transferred to right of the root node.
4. Get the desired action from the user.
5. Get the input from the user.
6. Output the result.
7. End the program.

PROGRAM:

```
#include <stdio.h>

#include <stdlib.h>

struct node

{

   int value;

   struct node *left, *right;
```

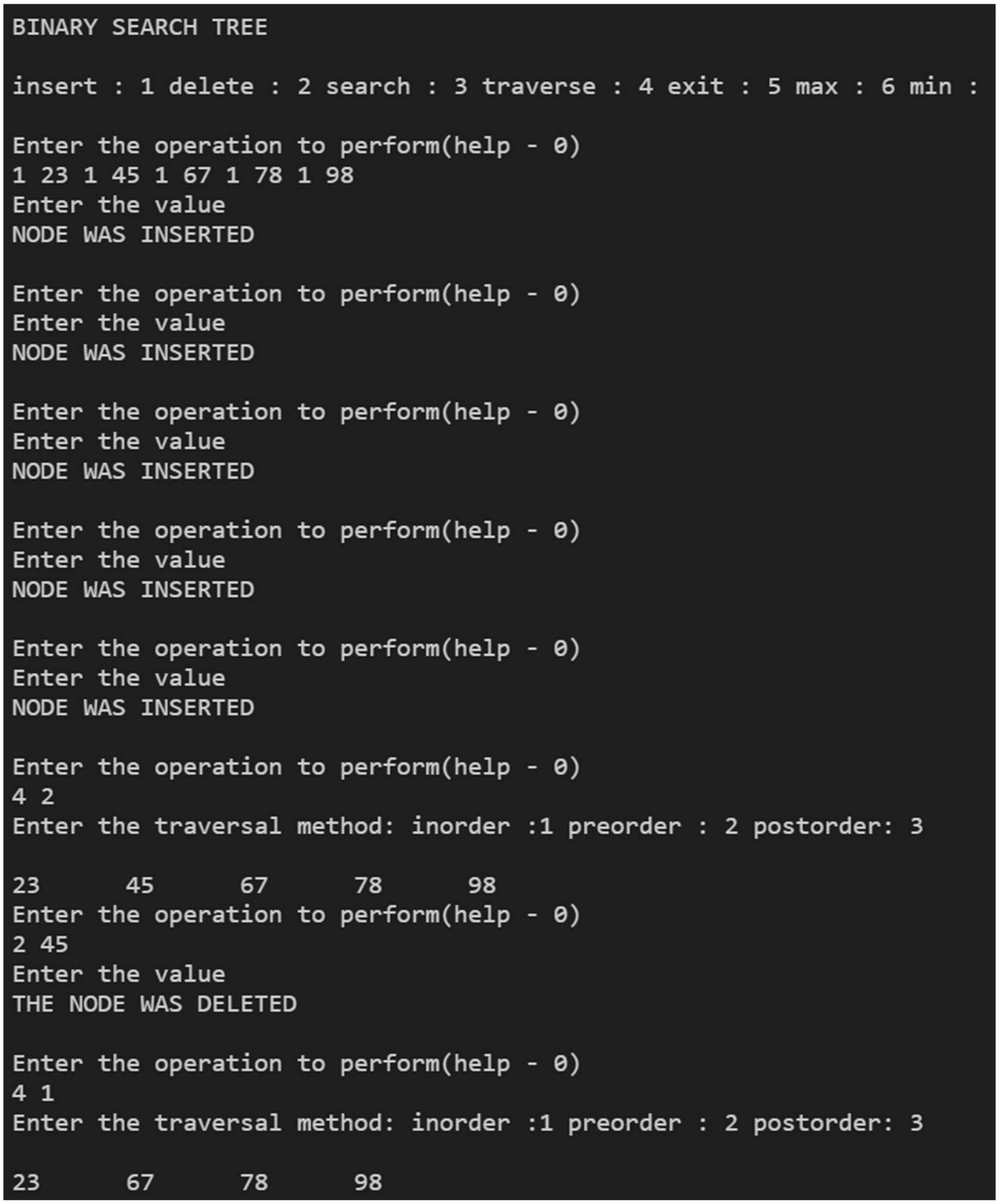
```
    int sub_height ;//TREE NODE WITH HEIGHT MEMBER FOR AVL TREE
};
typedef struct node NODE;
NODE *ROOT = NULL;
NODE *create_node(int value)
{
    NODE *temp = (NODE *)malloc(sizeof(NODE));
    temp->left = temp->right = NULL;
    temp->value = value;
    temp->sub_height = 0;
    return temp;
}
int getInput(int *value, int arr[], int flag)
{
    if (flag == 0)
        printf("Enter the value\n");
    if (flag == 1)
        printf("\nEnter the operation to perform(help - 0)\n");
    if (flag == 2)
    {
        int n;
        printf("Enter the no of elements:\n");
        scanf("%d", &n);
        printf("Enter the elements:\n");
        for (int i = 0; i < n; i++)
            scanf("%d", &arr[i]);
        arr[n] = '\0';
        return 1;
    }
    if (flag == 3)
        printf("your option :\n");
    scanf("%d", value);
    return 1;
}
int *delete_all(NODE **root)
{
    free((*root));
    *root = NULL;
}
int tree_to_array(NODE **root, int arr[], int flag, int count)
{
    if ((*root) == NULL)
    {
        return 1;
    }
    tree_to_array(&((*root)->left), arr, flag + 1,count);
    tree_to_array(&((*root)->right), arr, flag + 2, count);
    arr[flag] = (*root)->value;
    return 1;
}
int *print(NODE **root)
{
    printf("%d\t", (*root)->value);
}
//TREE TRAVERSAL
int in_order(NODE **root, int *(*callback)(NODE **))
{
    if (*root == NULL)
        return 0;
    in_order(&(*root)->left, callback);
    callback(root);
    in_order(&(*root)->right, callback);
    return 1;
}
int pre_order(NODE **root, int *(*callback)(NODE **))
{
    if (*root == NULL)
        return 0;
    //Node_count = callback(root);
    callback(root);
    pre_order(&(*root)->left, callback);
    pre_order(&(*root)->right, callback);
    return 1;
}
int post_order(NODE **root, int *(*callback)(NODE **))
{
    if (*root == NULL)
        return 0;
    post_order(&(*root)->left, callback);
    post_order(&(*root)->right, callback);
    callback(root);
    return 1;
}
int traverse(NODE **root)
{
    if (*root == NULL)
    {
        printf("ROOT IS NULL\n");
        return 0;
    }
    int i;
    printf("Enter the traversal method: inorder :1 preorder : 2 postorder: 3\n");
    while (1)
    {
        scanf("%d", &i);
        printf("\n");
        switch (i)

        {

        case 1:

            in_order(root, print);

            return 1;

        case 2:

            pre_order(root, print);

            return 1;

        case 3:

            post_order(root, print);

            return 1;

        default:

            printf("incorrect input, please try again.\n");

            break;

        }

    }

}




//BINARY SEARCH TREE

int insert_bst(NODE **root, int value)

{
    if (*root == NULL)
    {
        *root = create_node(value);
        return 1;
    }
    if ((*root)->value > value)
    {
        if ((*root)->left == NULL)
            (*root)->left = create_node(value);
        else
            insert_bst(&(*root)->left, value);
    }
    else if ((*root)->value < value)
    {
        if ((*root)->right == NULL)
            (*root)->right = create_node(value);
        else
            insert_bst(&(*root)->right, value);
    }
    return 1;
}
NODE *min_bst(NODE *root)
{
    NODE *NEXT = root;
    while (NEXT->left != NULL)
        NEXT = NEXT->left;
    return NEXT;
}
NODE *max_bst(NODE *root)
{
    NODE *NEXT = root;
    while (NEXT->right != NULL)
        NEXT = NEXT->right;
    return NEXT;
}
NODE *delete_bst(NODE *root, int value)
{
    if (root == NULL)
        return root;
    if (root->value > value)
        root->left = delete_bst(root->left, value);
    else if (root->value < value)
        root->right = delete_bst(root->right, value);
    else if( root->value == value)
    {
        NODE *temp;
        if (root->left == NULL)
        {
            temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL)
        {
            temp = root->left;
            free(root);
            return temp;
        }
        else
        {
            temp = min_bst(root->right);
            root->value = temp->value;
            root->right = delete_bst(root->right, root->value);
            return root;
        }
    }
    else
    {
        return root;
    }
```

```
}
NODE *search_bst(NODE *root, int value)
{
    if (root == NULL || root->value == value)
        return root;
    if (root->value > value)
        return search_bst(root->left, value);
    else if (root->value < value)
        return search_bst(root->right, value);
}
int convert_to_bst(NODE **root)
{


    int arr[100], flag;
    getInput(&flag, NULL, 3);
    if (flag == 0)
        getInput(NULL, arr, 2);
    if (flag == 1)
        tree_to_array(root, arr, 0, 0);// doubt at 4th argument
    post_order(root, delete_all);
    for (int i = 0; arr[i] != '\0'; i++)
        insert_bst(root, arr[i]);
    printf("CONVERTED TO BST\n");
    return 1;
}
int binary_search_tree(NODE **root) //BST DRIVER
{
    printf("\n\nBINARY SEARCH TREE\n\n");
    int input, value;
    NODE *result;
     printf("insert : 1 delete : 2 search : 3 traverse : 4 exit : 5 max : 6 min : 7 convert to BST : 8 delete tree : 9\n");
    while (1)
    {
        getInput(&input, NULL, 1);
        switch (input)
        {
        case 0:
             printf("insert : 1 delete : 2 search : 3 traverse : 4 exit : 5 max : 6 min : 7 convert to BST : 8 delete tree : 9\n");
            break;
        case 1:
            getInput(&value, NULL, 0);
            if (insert_bst(root, value))
                printf("NODE WAS INSERTED\n");
            else
                printf("THE NODE WAS NOT INSERTED\n");
            break;
```

```
case 2:
    getInput(&value, NULL, 0);
    if (delete_bst(ROOT, value) != NULL)
        printf("THE NODE WAS DELETED\n");
    else
        printf("ELEMENT WAS NOT FOUND\n");
    break;
case 3:
    getInput(&value, NULL, 0);
    result = search_bst(ROOT, value);
    if (result != NULL)
        printf("THE NODE WAS FOUND\n");
    else
        printf("THE ELEMENT WAS NOT FOUND\n");
    break;
case 4:
    traverse(root);
    break;
case 5:
    printf("\nExited from BST\n\n");


    return 1;
case 6:
    result = max_bst(ROOT);
```

```
            printf("THE LARGEST NUMBER IS %d\n", result->value);
            break;
        case 7:
            result = min_bst(ROOT);
            printf("THE SMALLEST NUMBER IS %d\n", result->value);
            break;
        case 8:
            convert_to_bst(&ROOT);
            break;
        case 9:
            if (post_order(&ROOT, delete_all))
                printf("THE TREE WAS DELETED");
            else
                printf("THE TREE WAS NOT DELETED");
            break;
        default:
            printf("incorrect input, try again\n");
            break;
        }  }
}
int main()
{   binary_search_tree(&ROOT);
    return 1;
}
```

OUTPUT:



```
BINARY SEARCH TREE

insert : 1 delete : 2 search : 3 traverse : 4 exit : 5 max : 6 min :

Enter the operation to perform(help - 0)
1 23 1 45 1 67 1 78 1 98
Enter the value
NODE WAS INSERTED

Enter the operation to perform(help - 0)
Enter the value
NODE WAS INSERTED

Enter the operation to perform(help - 0)
Enter the value
NODE WAS INSERTED

Enter the operation to perform(help - 0)
Enter the value
NODE WAS INSERTED

Enter the operation to perform(help - 0)
Enter the value
NODE WAS INSERTED

Enter the operation to perform(help - 0)
4 2
Enter the traversal method: inorder :1 preorder : 2 postorder: 3

23      45      67      78      98
Enter the operation to perform(help - 0)
2 45
Enter the value
THE NODE WAS DELETED

Enter the operation to perform(help - 0)
4 1
Enter the traversal method: inorder :1 preorder : 2 postorder: 3

23      67      78      98
```

RESULT:

The program was executed successfully.