```c
/*
DS10: Design, Develop and Implement a menu driven Program in C for the following
      operations on Binary Search Tree(BST) of Integers.
a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2
b. Traverse the BST in Inorder, Preorder and Post Order
c. Search the BST for a given element(KEY) and report the appropriate message.
d. Exit
*/

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

/*
'typedef with structure' is used to define a new data type and
then use that data type to define structure variables.
If we use the typedef keyword followed by a new name, we can use the struct
by that name without writing the struct keyword.
Here we are creating a structure by name BST.
*/

struct BST
{
            int data;
            struct BST *lchild;
            struct BST *rchild;
};
typedef struct BST *NODE;

/*
The first step is to create NODES.
NODES are places where we enter the values.
Value of elements to be inserted is asked,along with creation of left & right trees.

temp represents temporary places where values will be inserted.
Initially all values will be NULL.
*/

NODE create()    //creation of nodes which in turn creates a tree.
{
    NODE temp;
    temp = (NODE) malloc(sizeof(struct BST)); //dynamic memory allocation.
    printf("\n Enter The value: ");          //Value of elements in BST.
    scanf("%d",&temp->data);

    temp->lchild = NULL;
    temp->rchild = NULL;
    return temp;
}

//Function calls.

void insert(NODE root, NODE newnode);          //Inserting element into BST.
void inorder(NODE root);                       //Tree Traversals.
void preorder(NODE root);
void postorder(NODE root);
void search(NODE root);                     //Searching for an element in BST

/*
For insertion of elements into the node, we check two conditions:
1.If element to be inserted is less than root value, we insert as left child.
2.If element to be inserted is greater than root value, we insert as right child.
*/

void insert(NODE root, NODE newnode)
{
    if(newnode->data < root->data)           //less than root element.
```

```
        {
            if(root->lchild == NULL)
            root->lchild = newnode;          //insert as left child
            else
            insert(root->lchild, newnode);
        }

        if(newnode->data > root->data)     //greater than root element.
        {
            if (root->rchild == NULL)
            root->rchild = newnode;       //insert as right child
        else
            insert(root->rchild, newnode);
        }
}

/*
3 Traversals. Root must not be NULL for the traversals to be performed.
All 3 functions are similar except for the definition.
*/

void inorder(NODE root)
{
    if(root!= NULL)
        {
            inorder(root->lchild);
            printf("%d\t",root->data);
            inorder(root->rchild);
        }
}

void preorder(NODE root)
{
    if (root!= NULL)
        {
            printf("%d\t",root->data);
            preorder(root->lchild);
            preorder(root->rchild);
        }
}

void postorder(NODE root)
{
    if (root!= NULL)
        {
            postorder(root->lchild);
            postorder(root->rchild);
            printf("%d\t",root->data);
        }
}

/*
In search function, we perform following opeartions:
1. First check whether root is empty(whether tree is empty).
2. If not empty, we search for the element.
3. If the element to be searched is at 'root', we say element found.
4. If element not found in root, we search 'left' and 'right' tree.
*/

void search(NODE root)
{
    int key;                    //represents key element to be searched.
    NODE x;                    // x is used as a proxy to root.
    if(root == NULL)
    {
        printf("\n BST is empty");
```

```
        return;
    }

    printf("\n Enter Element to be searched: ");
    scanf("%d",&key);

    x=root;                  //Initialising root as x
    while (x!=NULL)         //While root(tree) is not empty, we search element.
    {
        if (x->data == key)            //if key is equal to root
        {
            printf("\n Key element is present in BST");
            return;
        }

        if (key > x->data)              //if key is less than root
        x = x->rchild;
        else
        x = x->lchild;
    }

        printf("\n Key element is not found in the BST"); //element not found
}

/*
Main function will have the menu.
Initially the root node is initialised to NULL meaning tree is empty.
i=1 in for loop representing tree starts from first element(root).

In 'case 1',if no node is there, we create newnode or
we insert values into already craeted nodes.
*/

void main()
{
    int ch, i, n;
    NODE root = NULL, newnode;
    while(1)
        {
            printf("\n ~~~~BST MENU~~~~ ");
            printf("\n 1. Create a BST");
            printf("\n 2. BST Traversals:");
            printf("\n 3. Search an Element");
            printf("\n 4. Exit");

            printf("\n Enter your choice: ");
            scanf("%d",&ch);

            switch(ch)
            {
                case 1: printf("\n Enter the number of elements: ");
                        scanf("%d", &n);
                        for(i=1; i<=n; i++) //starting from 1st node/element
                            {
                                newnode = create();
                                if (root == NULL)
                                    root = newnode;
                                else
                                    insert(root,newnode);
                            }
                            break;

                case 2: if (root == NULL)
                        printf("\n Tree Is Not Created");
                        else
                            {
```

```
                                printf("\n The Preorder display: ");
                                preorder(root);
                                printf("\n The Inorder display: ");
                                inorder(root);
                                printf("\n The Postorder display: ");
                                postorder(root);
                            }
                            break;

                case 3: search(root);
                        break;

                case 4: exit(0);

            }

        }
}
```

OUTPUT 1:

```
 ~~~~BST MENU~~~~
 1. Create a BST
 2. BST Traversals:
 3. Search an Element
 4. Exit
 Enter your choice: 2
 Tree Is Not Created

 ~~~~BST MENU~~~~
 1. Create a BST
 2. BST Traversals:
 3. Search an Element
 4. Exit
 Enter your choice: 3
 BST is empty

~~~~BST MENU~~~~
 1. Create a BST
 2. BST Traversals:
 3. Search an Element
 4. Exit
 Enter your choice: 1

 Enter the number of elements: 12

 Enter The value: 6

 Enter The value: 9

 Enter The value: 5

 Enter The value: 2

 Enter The value: 8

 Enter The value: 15

 Enter The value: 24

 Enter The value: 14

 Enter The value: 7

 Enter The value: 8

 Enter The value: 5
```

```
Enter The value: 2

~~~~BST MENU~~~~
1. Create a BST
2. BST Traversals:
3. Search an Element
4. Exit
Enter your choice: 2

The Preorder display: 6        5       2       9       8       7       15      14
  24
The Inorder display:  2        5       6       7       8       9       14      15
  24
The Postorder display: 2       5       7       8       14      24      15      9
  6

~~~~BST MENU~~~~
1. Create a BST
2. BST Traversals:
3. Search an Element
4. Exit
Enter your choice: 3

Enter Element to be searched: 20

Key element is not found in the BST

~~~~BST MENU~~~~
1. Create a BST
2. BST Traversals:
3. Search an Element
4. Exit
Enter your choice: 3

Enter Element to be searched: 5

Key element is present in BST

~~~~BST MENU~~~~
1. Create a BST
2. BST Traversals:
3. Search an Element
4. Exit
Enter your choice:4
```

OUTPUT 2:

```
~~~~BST MENU~~~~
```

```
1. Create a BST
2. BST Traversals:
3. Search an Element
4. Exit
Enter your choice: 1

Enter the number of elements: 6

Enter The value: 2

Enter The value: 1

Enter The value: 3

Enter The value: 5

Enter The value: 7

Enter The value: 9

~~~~BST MENU~~~~
1. Create a BST
2. BST Traversals:
3. Search an Element
4. Exit
Enter your choice: 2

The Preorder display: 2        1       3       5       7       9
The Inorder display:  1        2       3       5       7       9
The Postorder display: 1       9       7       5       3       2

~~~~BST MENU~~~~
1. Create a BST
2. BST Traversals:
3. Search an Element
4. Exit
Enter your choice: 3

Enter Element to be searched: 3

Key element is present in BST

~~~~BST MENU~~~~
1. Create a BST
2. BST Traversals:
3. Search an Element
4. Exit
Enter your choice: 3

Enter Element to be searched: 8
```

```
 Key element is not found in the BST

~~~~BST MENU~~~~
 1. Create a BST
 2. BST Traversals:
 3. Search an Element
 4. Exit
 Enter your choice: 4
```

OUTPUT 3:

```
 ~~~~BST MENU~~~~
 1. Create a BST
 2. BST Traversals:
 3. Search an Element
 4. Exit
 Enter your choice: 1

 Enter the number of elements: 5

 Enter The value: 5

 Enter The value: 2

 Enter The value: 1

 Enter The value: 8

 Enter The value: 6

 ~~~~BST MENU~~~~
 1. Create a BST
 2. BST Traversals:
 3. Search an Element
 4. Exit
 Enter your choice: 3

 Enter Element to be searched: 4

 Key element is not found in the BST

~~~~BST MENU~~~~
 1. Create a BST
 2. BST Traversals:
 3. Search an Element
 4. Exit
 Enter your choice: 3
```

```
 Enter Element to be searched: 2

 Key element is present in BST

~~~~BST MENU~~~~
 1. Create a BST
 2. BST Traversals:
 3. Search an Element
 4. Exit
 Enter your choice: 2

 The Preorder display: 5        2       1       8       6
 The Inorder display:  1        2       5       6       8
 The Postorder display: 1       2       6       8       5

 ~~~~BST MENU~~~~
 1. Create a BST
 2. BST Traversals:
 3. Search an Element
 4. Exit
 Enter your choice: 4
```