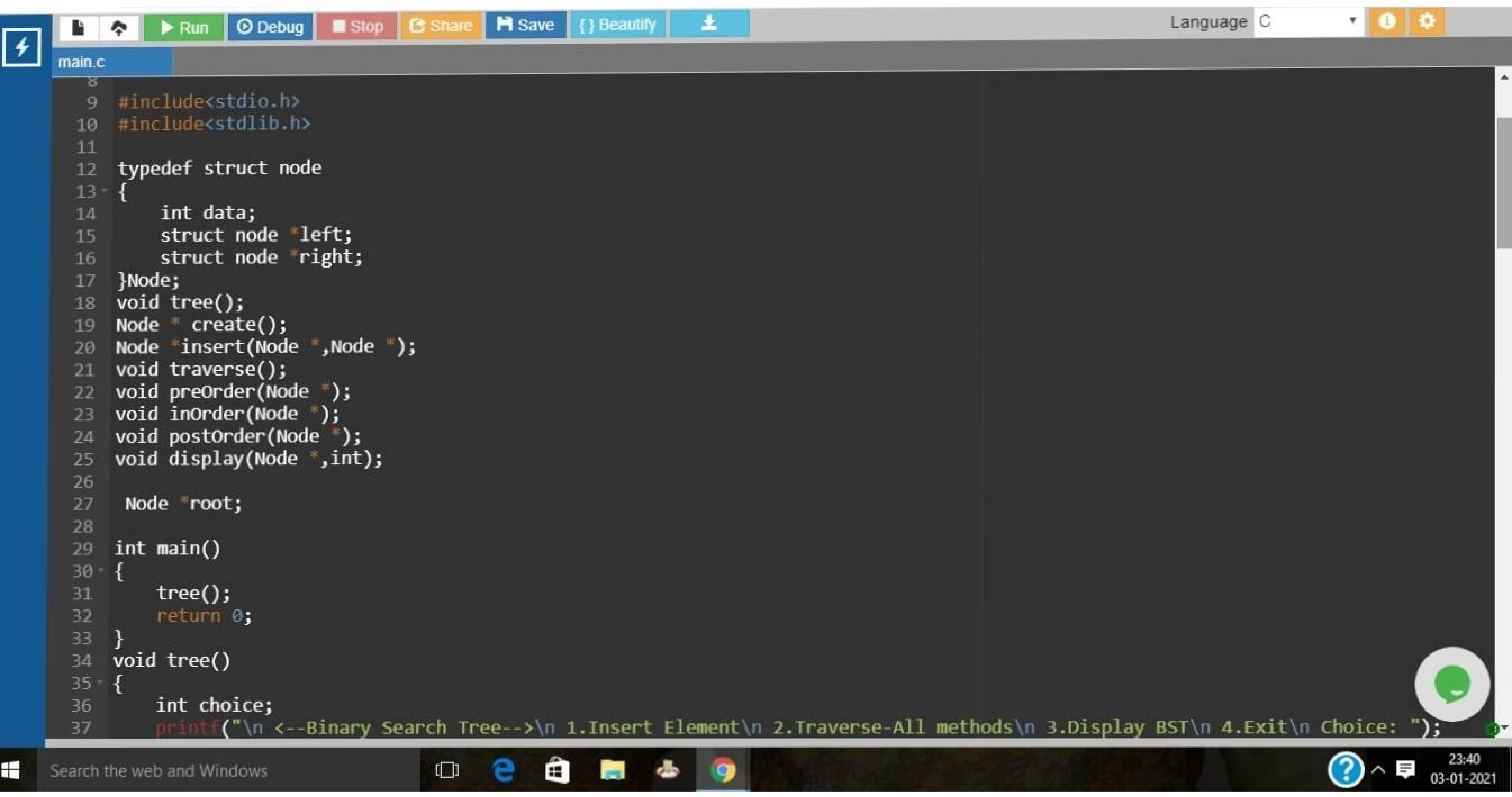
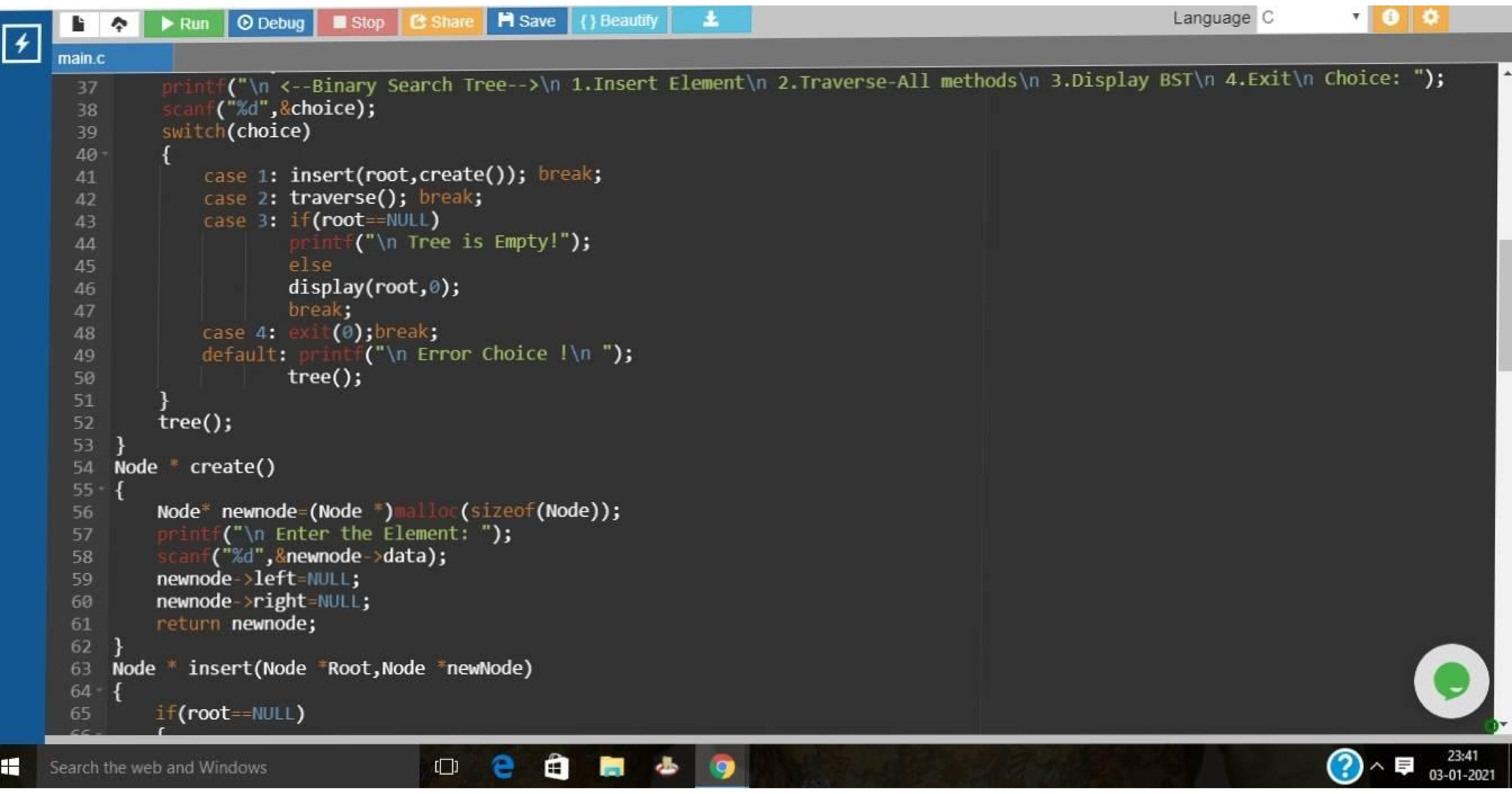
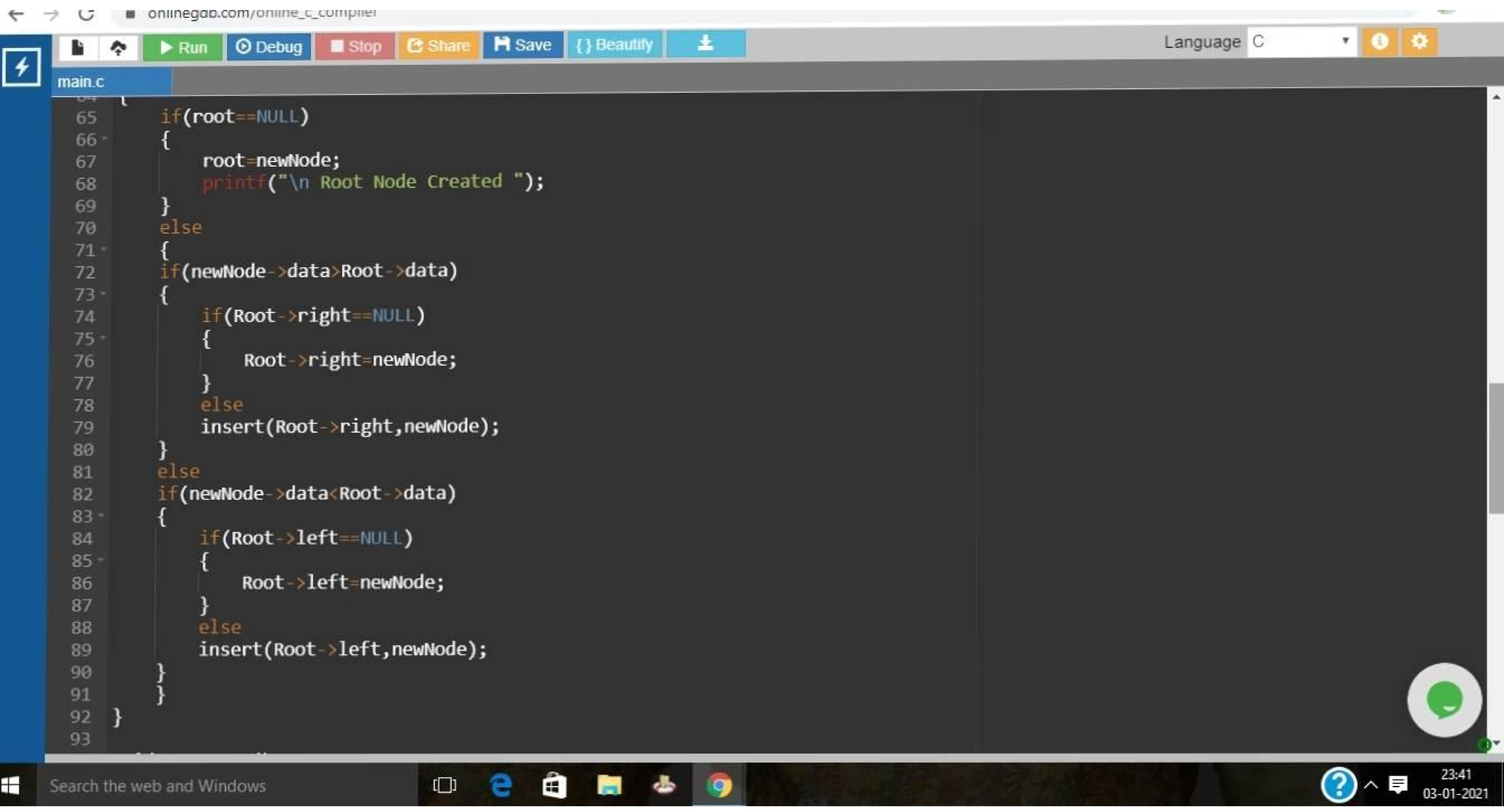


```c
#include<stdio.h>
#include<stdlib.h>

typedef struct node
{
     int data;
     struct node *left;
     struct node *right;
}Node;
void tree();
Node * create();
Node *insert(Node *,Node *);
void traverse();
void preOrder(Node *);
void inOrder(Node *);
void postOrder(Node *);
void display(Node *,int);

 Node *root;

int main()
{
     tree();
     return 0;
}
void tree()
{
     int choice;
     printf("\n <--Binary Search Tree-->\n 1.Insert Element\n 2.Traverse-All methods\n 3.Display BST\n 4.Exit\n Choice: ");
```

Scanned by TapScanner

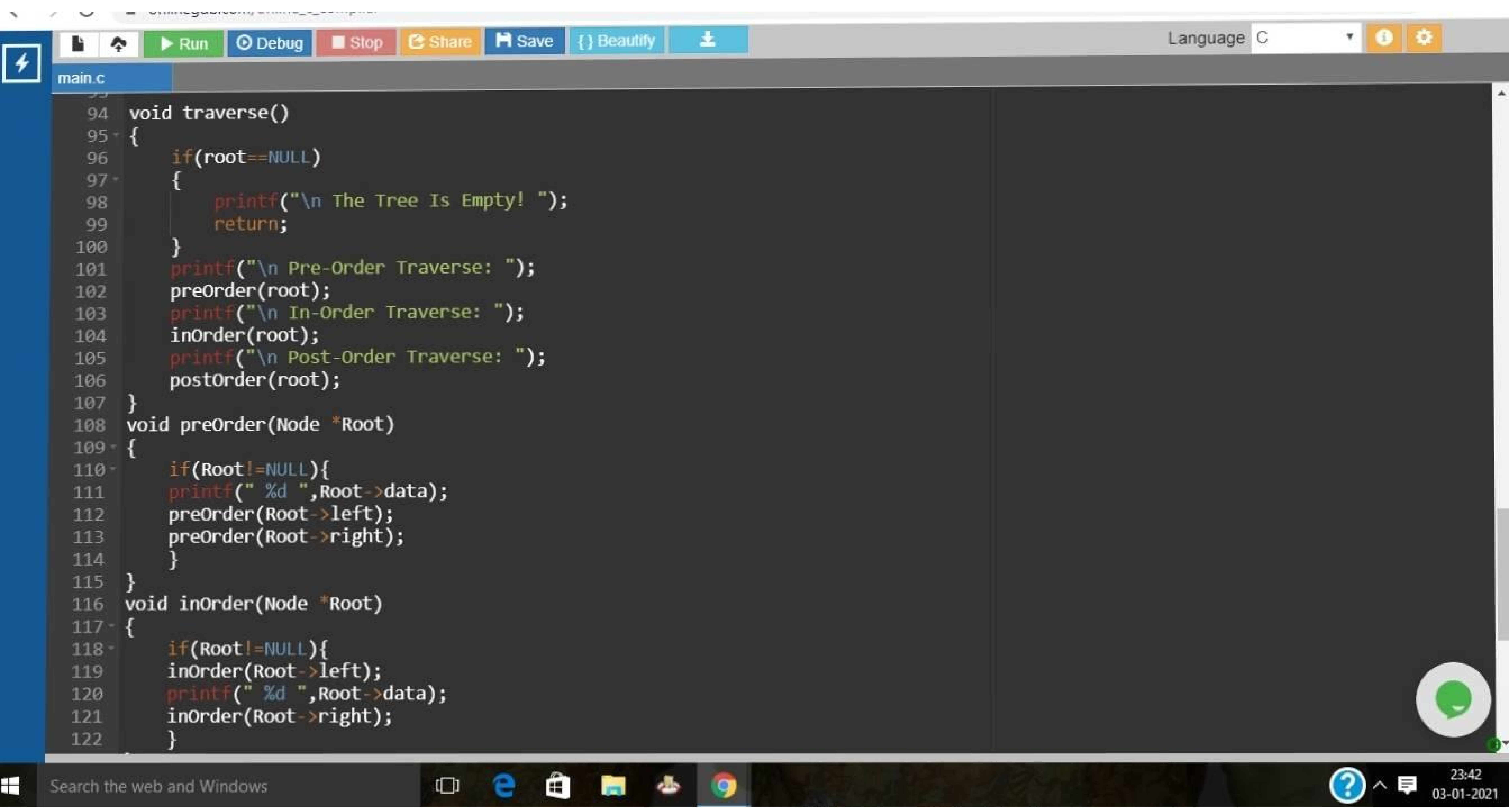


```
    printf("\n <--Binary Search Tree-->\n 1.Insert Element\n 2.Traverse-All methods\n 3.Display BST\n 4.Exit\n Choice: ");
    scanf("%d",&choice);
    switch(choice)
    {
        case 1: insert(root,create()); break;
        case 2: traverse(); break;
        case 3: if(root==NULL)
                printf("\n Tree is Empty!");
                else
                display(root,0);
                break;
        case 4: exit(0);break;
        default: printf("\n Error Choice !\n ");
                tree();
    }
    tree();
}
Node * create()
{
    Node* newnode=(Node *)malloc(sizeof(Node));
    printf("\n Enter the Element: ");
    scanf("%d",&newnode->data);
    newnode->left=NULL;
    newnode->right=NULL;
    return newnode;
}
Node * insert(Node *Root,Node *newNode)
{
    if(root==NULL)
```



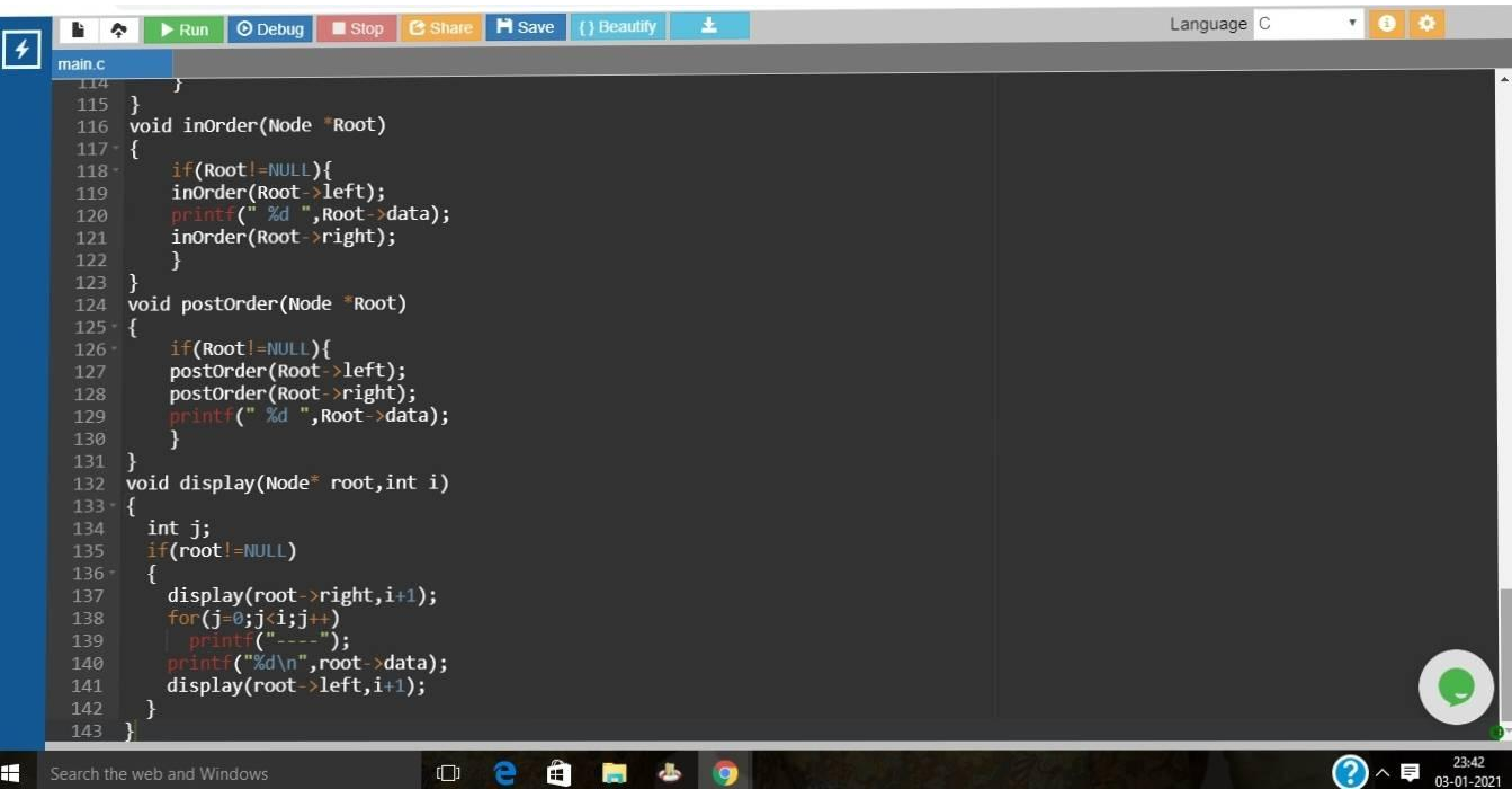
```c
    if(root==NULL)
    {
        root=newNode;
        printf("\n Root Node Created ");
    }
    else
    {
    if(newNode->data>Root->data)
    {
        if(Root->right==NULL)
        {
            Root->right=newNode;
        }
        else
        insert(Root->right,newNode);
    }
    else
    if(newNode->data<Root->data)
    {
        if(Root->left==NULL)
        {
            Root->left=newNode;
        }
        else
        insert(Root->left,newNode);
    }
    }
}
```

Scanned by TapScanner

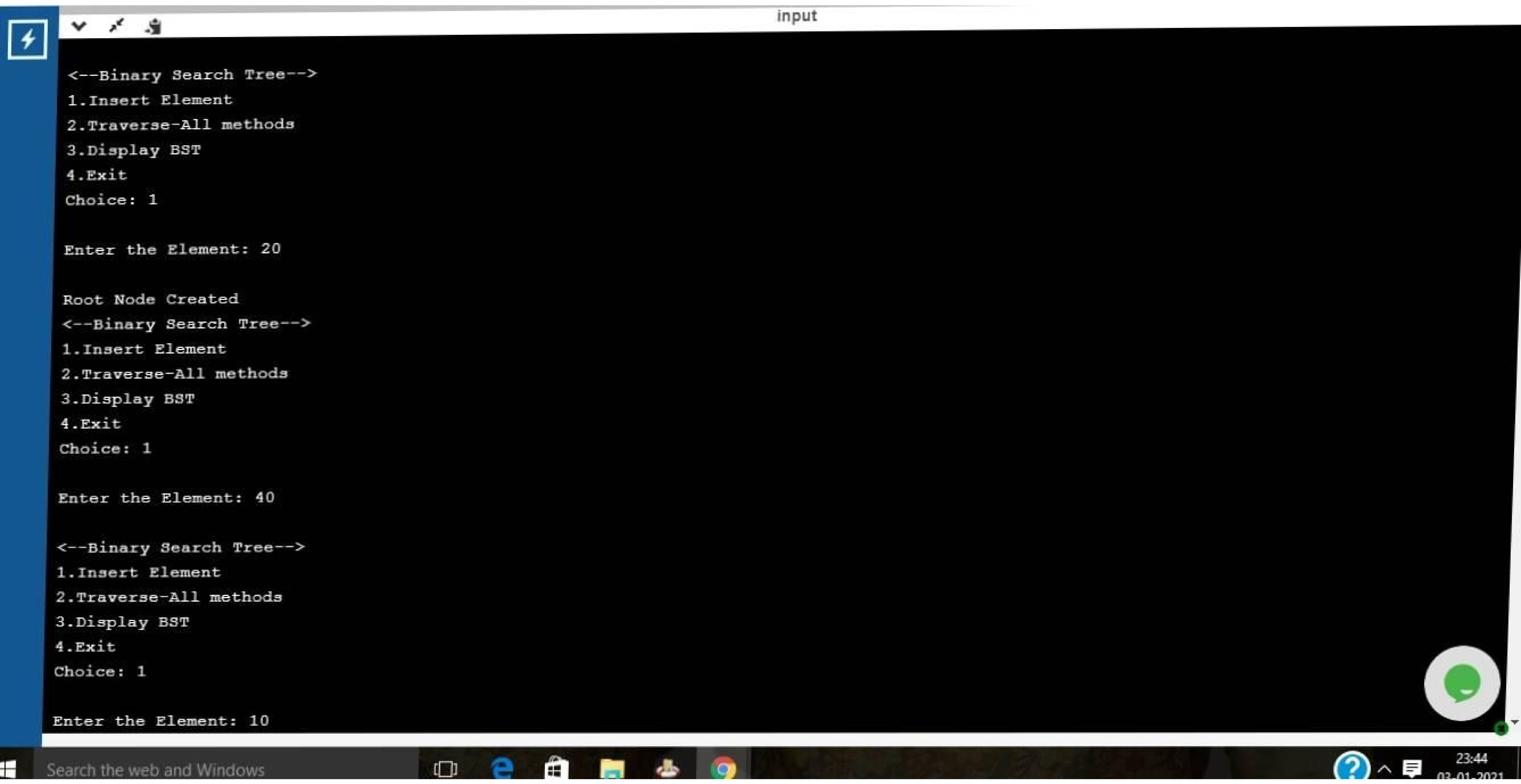


```c
void traverse()
{
    if(root==NULL)
    {
        printf("\n The Tree Is Empty! ");
        return;
    }
    printf("\n Pre-Order Traverse: ");
    preOrder(root);
    printf("\n In-Order Traverse: ");
    inOrder(root);
    printf("\n Post-Order Traverse: ");
    postOrder(root);
}
void preOrder(Node *Root)
{
    if(Root!=NULL){
    printf(" %d ",Root->data);
    preOrder(Root->left);
    preOrder(Root->right);
    }
}
void inOrder(Node *Root)
{
    if(Root!=NULL){
    inOrder(Root->left);
    printf(" %d ",Root->data);
    inOrder(Root->right);
    }
```

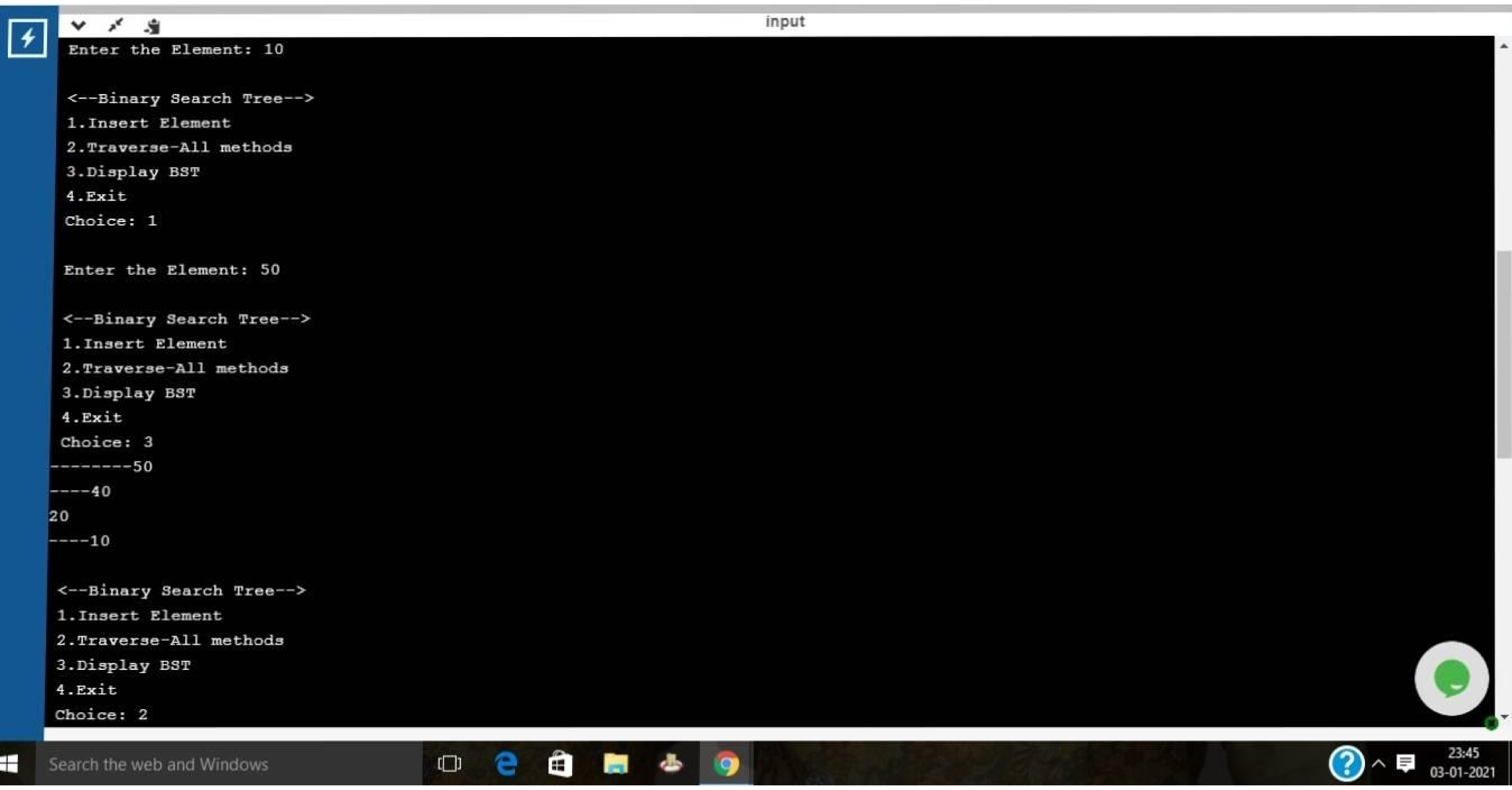
Scanned by TapScanner



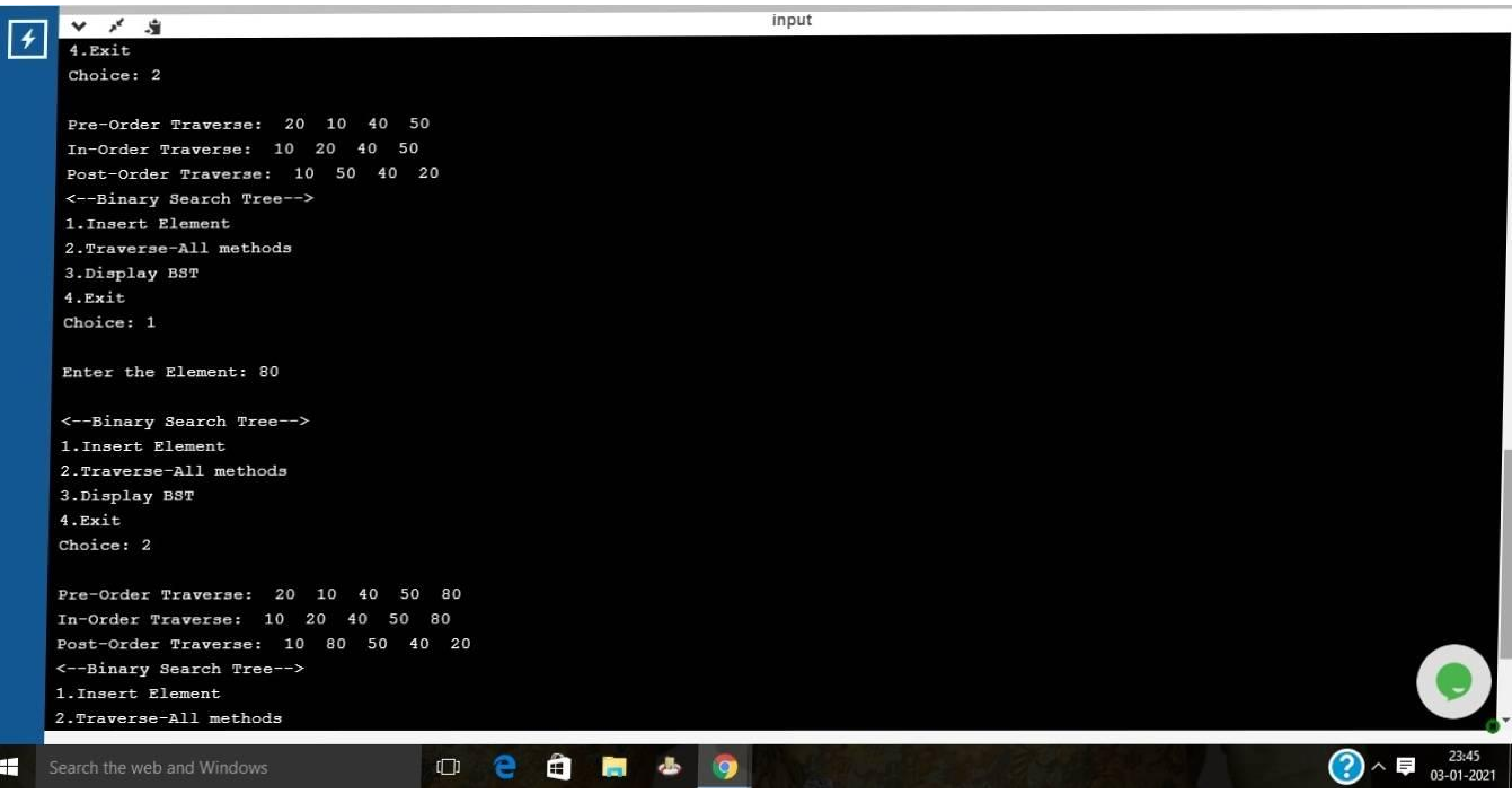
```c
    }
}
void inOrder(Node *Root)
{
    if(Root!=NULL){
    inOrder(Root->left);
    printf(" %d ",Root->data);
    inOrder(Root->right);
    }
}
void postOrder(Node *Root)
{
    if(Root!=NULL){
    postOrder(Root->left);
    postOrder(Root->right);
    printf(" %d ",Root->data);
    }
}
void display(Node* root,int i)
{
  int j;
  if(root!=NULL)
  {
    display(root->right,i+1);
    for(j=0;j<i;j++)
      printf("----");
    printf("%d\n",root->data);
    display(root->left,i+1);
  }
}
```



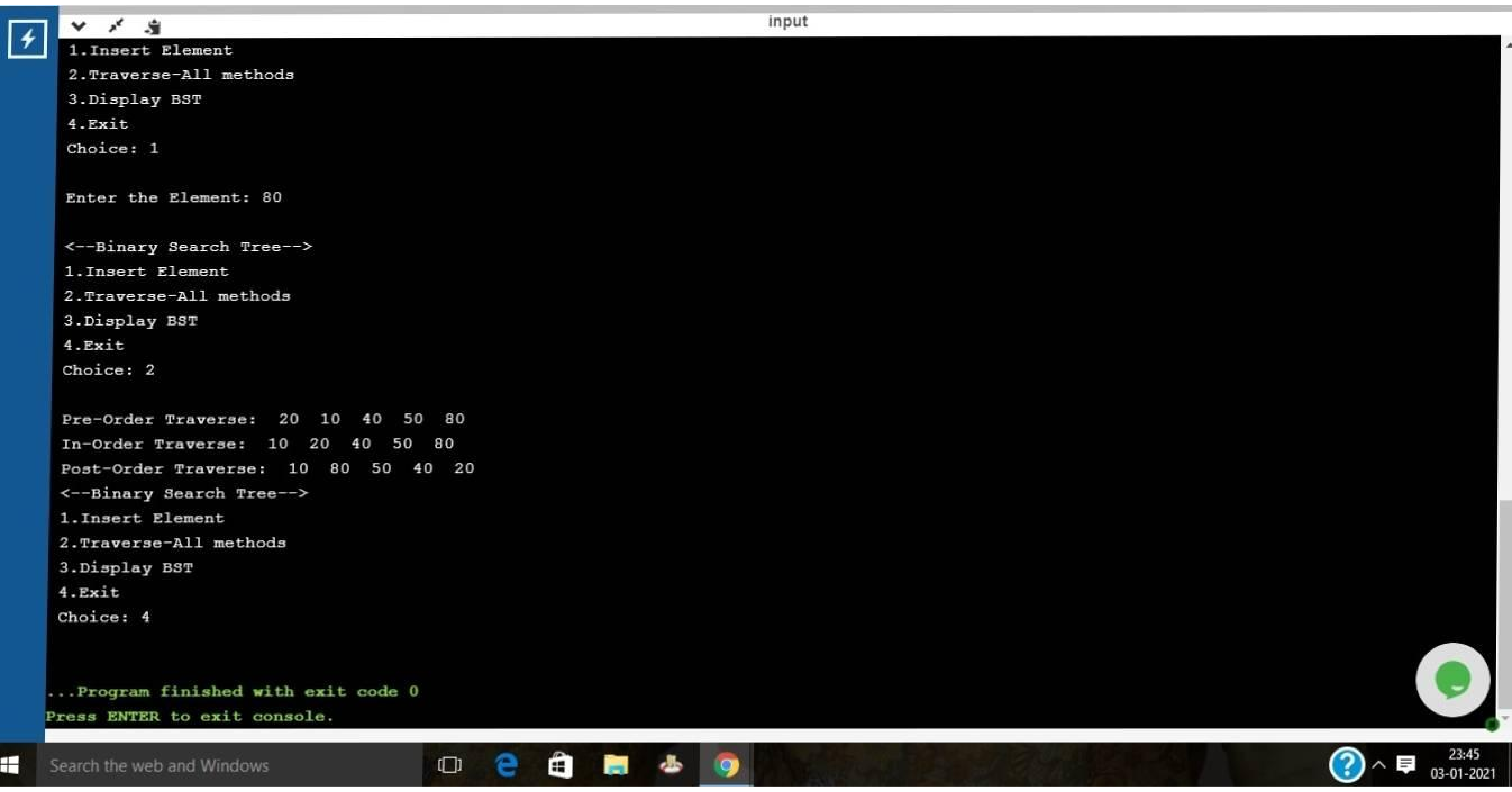
```
<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 1

Enter the Element: 20

Root Node Created
<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 1

Enter the Element: 40

<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 1

Enter the Element: 10
```



```
Enter the Element: 10

 <--Binary Search Tree-->
 1.Insert Element
 2.Traverse-All methods
 3.Display BST
 4.Exit
 Choice: 1

 Enter the Element: 50

 <--Binary Search Tree-->
 1.Insert Element
 2.Traverse-All methods
 3.Display BST
 4.Exit
 Choice: 3
--------50
----40
20
----10

 <--Binary Search Tree-->
 1.Insert Element
 2.Traverse-All methods
 3.Display BST
 4.Exit
 Choice: 2
```



```
4.Exit
Choice: 2

Pre-Order Traverse:  20  10  40  50
In-Order Traverse:  10  20  40  50
Post-Order Traverse:  10  50  40  20
<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 1

Enter the Element: 80

<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 2

Pre-Order Traverse:  20  10  40  50  80
In-Order Traverse:  10  20  40  50  80
Post-Order Traverse:  10  80  50  40  20
<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
```



```
1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 1

Enter the Element: 80

<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 2

Pre-Order Traverse:  20  10  40  50  80
In-Order Traverse:  10  20  40  50  80
Post-Order Traverse:  10  80  50  40  20
<--Binary Search Tree-->
1.Insert Element
2.Traverse-All methods
3.Display BST
4.Exit
Choice: 4


...Program finished with exit code 0
Press ENTER to exit console.
```