1.

#include <stdio.h>

#include <stdlib.h>

{

Int data;

Struct node\* left;

Struct node\* right;

};

{

Struct node\* node = (struct node\*)malloc(sizeof(struct node));

node-> = data;

node->left = NULL;

node->right = NULL;

return(node);

}

{

If (node == NULL)

return;

void printInorder(struct node\* node)

{

If (node == NULL)

return;

}

void printPreorder(struct node\* node)

{

If (node == NULL)

return;

}

int main()

{

Struct node \*root = newNode(1);

root->left = newNode(2);

root->right = newNode(3);

root->left->left = newNode(4);

root->left->right = newNode(5);

printf(“\nPreorder traversal of binary tree is \n”);

printPreorder(root);

printf(“\nInorder traversal of binary tree is \n”);

printInorder(root);

printf(“\nPostorder traversal of binary tree is \n”);

printPostorder(root);

getchar();

return 0;

}

2.

//C program to demonstrate insert operation in binary search tree//

#include<stdio.h>

#include<stdlib.h>

struct node

{

int key;

struct node \*left, \*right;

};

struct node \*newNode(int item)

{

struct node \*temp = (struct node \*)malloc(sizeof(struct node));

temp->key = item;

temp->left = temp->right = NULL;

return temp;

}

void inorder(struct node \*root)

{

if (root != NULL)

{

inorder(root->left);

printf("%d \n", root->key);

inorder(root->right);

}

}

struct node\* insert(struct node\* node, int key)

{

/\* If the tree is empty, return a new node \*/

if (node == NULL) return newNode(key);

/\* Otherwise, recur down the tree \*/

if (key < node->key)

node->left = insert(node->left, key);

else if (key > node->key)

node->right = insert(node->right, key);

/\* return the (unchanged) node pointer \*/

return node;

}

int main()

{

struct node \*root = NULL;

root = insert(root, 5);

insert(root, 3);

insert(root, 2);

insert(root, 4);

insert(root, 7);

insert(root, 6);

insert(root, 8);

// print inoder traversal of the BST

inorder(root);

return 0;

}

Output:

2

3

4

5

6

7

8

3.

#include <stdio.h>

Int linear\_search(int arr[],int ele,int n);

Void main()

{

int n, ele;

printf(“enter the size of array:”);

scanf(“%d”,&ele);

int arr[n];

for(int i=0;i<n;i++)

{

printf(“enter any integer”);

scanf(“%d”,&ele);

linear\_search(arr,ele,n);

}

int linear\_search(int arr[],int ele,int n)

{

int found=0;

for(int i=0;i<n;i++)

{

if found=0;

for(int i=0;i<n;i++)

{

if(arr[i] == ele)

found = 1

}

if(found == 1)

printf(“element is found”);

else

printf(“element is not found”);

}

4.

#include <stdio.h>

int main()

{

int i,a[100],n, low, high, mid, x;

printf("Enter number of elements:");

scanf("%d", &n);

printf("Enter %d integers:", n);

for (i= 0; i< n; i++)

scanf("%d", &a[i]);

printf("Enter value to find:");

scanf("%d", &x);

low = 0;

high = n - 1;

mid = (low+high)/2;

while (low <= high) {

if (a[mid] < x)

low = mid + 1;

else if (a[mid] == x) {

printf("%d found at position %d", x, mid+1);

break;

}

else

high = mid - 1;

mid = (low + high)/2;

}

if (low>high)

printf("The number %d not found in the list", search);

return 0;

}