1. Pre order, In order, Post order

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node\*left;

struct node\*right;

}

void print preorder(struct node\*node)

{

if (node==null)

return;

printf("%d",node->data);

print preorder(node->left);

print preorder(node->right);

}

int main()

{

struct node\*root=newnode(5);

root->left=newnode(4);

root->right=newnode(3);

root->left->left=newnode(2);

root->left->right=newnode(1);

printf("\n preorder transversal of binary tree is \n");

print preorder(root);

getchar();

return 0;

}

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node\*left;

struct node\*right;

}

void print inorder(struct node\*node)

{

if (node==null)

return;

print inorder(node->left);

printf("%d",node->data);

print inorder(node->right);

}

int main()

{

struct node\*root=newnode(5);

root->left=newnode(4);

root->right=newnode(3);

root->left->left=newnode(2);

root->left->right=newnode(1);

printf("\n inorder transversal of binary tree is \n");

print postorder(root);

getchar();

return 0;

}

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node\*left;

struct node\*right;

}

void print postorder(struct node\*node)

{

if (node==null)

return;

print postorder(node->left);

print postorder(node->right);

printf("%d",node->data);

}

int main()

{

struct node\*root=newnode(5);

root->left=newnode(4);

root->right=newnode(3);

root->left->left=newnode(2);

root->left->right=newnode(1);

printf("\n postorder transversal of binary tree is \n");

print postorder(root);

getchar();

return 0;

}

1. Create or insert inorder transversal on binary search tree

# include <stdio.h>

# include <conio.h>

# include <stdlib.h>

typedef struct BST {

int data;

struct BST \*lchild, \*rchild;

} node;

void insert(node \*, node \*);

void inorder(node \*);

void preorder(node \*);

void postorder(node \*);

node \*search(node \*, int, node \*\*);

void main() {

int choice;

char ans = 'N';

int key;

node \*new\_node, \*root, \*tmp, \*parent;

node \*get\_node();

root = NULL;

clrscr();

printf("\nProgram For Binary Search Tree ");

do {

printf("\n1.Create");

printf("\n2.Search");

printf("\n3.Recursive Traversals");

printf("\n4.Exit");

printf("\nEnter your choice :");

scanf("%d", &choice);

switch (choice) {

case 1:

do {

new\_node = get\_node();

printf("\nEnter The Element ");

scanf("%d", &new\_node->data);

if (root == NULL) /\* Tree is not Created \*/

root = new\_node;

else

insert(root, new\_node);

printf("\nWant To enter More Elements?(y/n)");

ans = getch();

} while (ans == 'y');

break;

case 2:

printf("\nEnter Element to be searched :");

scanf("%d", &key);

tmp = search(root, key, &parent);

printf("\nParent of node %d is %d", tmp->data, parent->data);

break;

case 3:

if (root == NULL)

printf("Tree Is Not Created");

else {

printf("\nThe Inorder display : ");

inorder(root);

printf("\nThe Preorder display : ");

preorder(root);

printf("\nThe Postorder display : ");

postorder(root);

}

break;

}

} while (choice != 4);

}

node \*get\_node() {

node \*temp;

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

temp->lchild = NULL;

temp->rchild = NULL;

return temp;

}

void insert(node \*root, node \*new\_node) {

if (new\_node->data < root->data) {

if (root->lchild == NULL)

root->lchild = new\_node;

else

insert(root->lchild, new\_node);

}

if (new\_node->data > root->data) {

if (root->rchild == NULL)

root->rchild = new\_node;

else

insert(root->rchild, new\_node);

}

}

node \*search(node \*root, int key, node \*\*parent) {

node \*temp;

temp = root;

while (temp != NULL) {

if (temp->data == key) {

printf("\nThe %d Element is Present", temp->data);

return temp;

}

\*parent = temp;

if (temp->data > key)

temp = temp->lchild;

else

temp = temp->rchild;

}

return NULL;

}

void inorder(node \*temp) {

if (temp != NULL) {

inorder(temp->lchild);

printf("%d", temp->data);

inorder(temp->rchild);

}

}

void preorder(node \*temp) {

if (temp != NULL) {

printf("%d", temp->data);

preorder(temp->lchild);

preorder(temp->rchild);

}

}

void postorder(node \*temp) {

if (temp != NULL) {

postorder(temp->lchild);

postorder(temp->rchild);

printf("%d", temp->data);

}

}

1. Linear search algorithm

#include <stdio.h>

int main()  
{  
 int array[100], search, c, n;

printf("Enter number of elements in array**\n**");  
 scanf("%d", &n);

printf("Enter %d integer(s)**\n**", n);

for (c = 0; c < n; c++)  
 scanf("%d", &array[c]);

printf("Enter a number to search**\n**");  
 scanf("%d", &search);

for (c = 0; c < n; c++)  
 {  
 if (array[c] == search) */\* If required element is found \*/*  
{  
 printf("%d is present at location %d.**\n**", search, c+1);  
 **break**;  
 }  
 }  
 if (c == n)  
 printf("%d isn't present in the array.**\n**", search);

return 0;  
}

1. Binary search algorithm

#include <stdio.h>

int main()  
{  
 int c, first, last, middle, n, search, array[100];

printf("Enter number of elements**\n**");  
 scanf("%d", &n);

printf("Enter %d integers**\n**", n);

for (c = 0; c < n; c++)  
 scanf("%d", &array[c]);

printf("Enter value to find**\n**");  
 scanf("%d", &search);

first = 0;  
 last = n - 1;  
 middle = (first+last)/2;

while (first <= last) {  
 if (array[middle] < search)  
 first = middle + 1;  
 else if (array[middle] == search) {  
 printf("%d found at location %d.**\n**", search, middle+1);  
 **break**;  
 }  
 else  
 last = middle - 1;

middle = (first + last)/2;  
 }  
 if (first > last)  
 printf("Not found! %d isn't present in the list.**\n**", search);

return 0;  
}