Write a C program to print preorder, in order and postorder traversal on Binary Tree.

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

typedef struct BST
{
   int data;
   struct BST *left;
   struct BST *right;
} node;

node *create();
void insert(node *, node *);
void preorder(node *);
void postorder(node *);
void inorder(node *);

int main()
{
   int ch;
   node *root = NULL, *temp, *current;
   printf("Enter the number of Nodes you want :    ");
   scanf("%d", &ch);
   printf("Enter %d Nodes data :", ch);
   do
   {
      temp = create();
      if (root == NULL)
         root = temp;
      else
         insert(root, temp);
      ch--;

   } while (ch != 0);

   printf("\n\nPreorder Traversal\t");
   preorder(root);

   printf("\n\nInorder Traversal\t");
   inorder(root);
```

```
    printf("\n\nPostorder Traversal\t");
    postorder(root);

    printf("");

    return 0;
}

node *create()
{
    node *temp;

    temp = (node *)malloc(sizeof(node));
    scanf("%d", &temp->data);
    temp->left = temp->right = NULL;
    return temp;
}

void insert(node *root, node *temp)
{
    if (root == NULL)
    {
        root = temp;
    }
    else
    {

        if (temp->data < root->data)
        {
            if (root->left != NULL)
                insert(root->left, temp);
            else
                root->left = temp;
        }

        if (temp->data > root->data)
        {
            if (root->right != NULL)
                insert(root->right, temp);
            else
                root->right = temp;
        }
```

```
    }
}

void preorder(node *root)
{
   if (root != NULL)
   {
      printf("%d  ", root->data);
      preorder(root->left);
      preorder(root->right);
   }
}

void postorder(node *root)
{
   if (root != NULL)
   {
      postorder(root->left);
      postorder(root->right);
      printf("%d  ", root->data);
   }
}

void inorder(node *root)
{
   if (root != NULL)
   {
      inorder(root->left);
      printf("%d  ", root->data);
      inorder(root->right);
   }
}
```

Write a C program to create (or insert) and inorder traversal on Binary Search Tree.

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

typedef struct BST
{
   int data;
   struct BST *left;
```

```
    struct BST *right;
} node;

node *create();
void insert(node *, node *);
void inorder(node *);

int main()
{
    int ch;
    node *root = NULL, *temp, *current;
    printf("Enter the number of Nodes you want :     ");
    scanf("%d", &ch);
    printf("Enter %d Nodes data :", ch);
    do
    {
        temp = create();
        if (root == NULL)
            root = temp;
        else
            insert(root, temp);
        ch--;

    } while (ch != 0);


    printf("\n\nInorder Traversal\t");
    inorder(root);


    printf("");

    return 0;
}

node *create()
{
    node *temp;

    temp = (node *)malloc(sizeof(node));
    scanf("%d", &temp->data);
    temp->left = temp->right = NULL;
```

```
    return temp;
}

void insert(node *root, node *temp)
{
    if (root == NULL)
    {
        root = temp;
    }
    else
    {

        if (temp->data < root->data)
        {
            if (root->left != NULL)
                insert(root->left, temp);
            else
                root->left = temp;
        }

        if (temp->data > root->data)
        {
            if (root->right != NULL)
                insert(root->right, temp);
            else
                root->right = temp;
        }
    }
}

void inorder(node *root)
{
    if (root != NULL)
    {
        inorder(root->left);
        printf("%d  ", root->data);
        inorder(root->right);
    }
}
```

Write a C program depth first search (DFS) using array.

```
#include<stdio.h>
int a[20][20],reach[20],n;
void dfs(int v)
{
 int i;
 reach[v]=1;
 for(i=1;i<=n;i++)
  if(a[v][i] && !reach[i])
  {
   printf("n %d->%d",v,i);
   dfs(i);
  }
}
int main()
{
 int i,j,count=0;
 printf("n Enter number of vertices:");
 scanf("%d",&n);
 for(i=1;i<=n;i++)
 {
  reach[i]=0;
  for(j=1;j<=n;j++)
   a[i][j]=0;
 }
 printf("n Enter the adjacency matrix:n");
 for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
   scanf("%d",&a[i][j]);
 dfs(1);
 printf("n");
 for(i=1;i<=n;i++)
 {
  if(reach[i])
   count++;
 }
 if(count==n)
  printf("n Graph is connected");
 else
  printf("n Graph is not connected");
 return 0;
```

}

Write a C program breath first search (BFS) using array.

```
#include<stdio.h>
int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1;

void bfs(int v) {
 for(i = 1; i <= n; i++)
 if(a[v][i] && !visited[i])
 q[++r] = i;
 if(f <= r) {
 visited[q[f]] = 1;
 bfs(q[f++]);
 }
}

void main() {
 int v;
 printf("\n Enter the number of vertices:");
 scanf("%d", &n);

 for(i=1; i <= n; i++) {
 q[i] = 0;
 visited[i] = 0;
 }
 printf("\n Enter graph data in matrix form:\n");
 for(i=1; i<=n; i++) {
 for(j=1;j<=n;j++) {
 scanf("%d", &a[i][j]);
 }
 }

 printf("\n Enter the starting vertex:");
 scanf("%d", &v);
 bfs(v);
 printf("\n The node which are reachable are:\n");

 for(i=1; i <= n; i++) {
 if(visited[i])
 printf("%d\t", i);
 else {
```

```
 printf("\n Bfs is not possible. Not all nodes are reachable");
 break;
 }
 }
}
```

Write a C program for linear search algorithm.

```
#include <stdio.h>

int main()
{
   int a[20],i,n,x;
   printf("Enter the array size\n");
   scanf("%d",&n);
   printf("\n enter the array elements");
   for(i=0;i<n;i++)
     scanf("%d",&a[i]);
   printf("\n enter the search element");
   scanf("%d",&x);
   for(i=0;i<n;i++)
   {
      if(a[i]==x)
      {
         printf("%d element is found at %d location",x,i+1);
         break;
      }
   }
  if(i==n)
  printf("\n element not found");
  return 0;
}
```

Write a C program for 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;
}
```