```
/*1.Write a C program to print preorder, inorder, and postorder traversal on
Binary Tree. */

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

struct node {
    int data;
    struct node* left;
    struct node* right;
};
void inorder(struct node* root){
    if(root == NULL) return;
    inorder(root->left);
    printf("%d ->", root->data);
    inorder(root->right);
}
void preorder(struct node* root){
    if(root == NULL) return;
    printf("%d ->", root->data);
    preorder(root->left);
    preorder(root->right);
}
void postorder(struct node* root) {
    if(root == NULL) return;
    postorder(root->left);
    postorder(root->right);
    printf("%d ->", root->data);
}
struct node* createNode( int value){
    struct node* newNode = malloc(sizeof(struct node));
    newNode->data = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
struct node* insertLeft(struct node *root, int value) {
    root->left = createNode(value);
    return root->left;
}
struct node* insertRight(struct node *root, int value){
    root->right = createNode(value);
    return root->right;
}
```

```
int main(){
    struct node* root = createNode(1);
    insertLeft(root, 12);
    insertRight(root, 9);
    insertLeft(root->left, 5);
    insertRight(root->left, 6);
    printf("Inorder traversal \n");
    inorder(root);
    printf("\nPreorder traversal \n");
    preorder(root);
    printf("\nPostorder traversal \n");
    postorder(root);
}
```

Output:

```
Inorder traversal
5 ->12 ->6 ->1 ->9 ->
Preorder traversal
1 ->12 ->5 ->6 ->9 ->
Postorder traversal
5 ->6 ->12 ->9 ->1 ->
```

/*2.Write a C program to create (or insert) and inorder traversal on 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 (node == NULL) return newNode(key);
    if (key < node->key)
        node->left  = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

    return node;
}
int main()
{
    struct node *root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);

    inorder(root);

    return 0;
}
```

Output:

```
20
30
40
50
60
70
80
```

/*3.Write a C program depth first search (DFS) using array.*/

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

```
#include <stdlib.h>
#include <stdbool.h>
#define MAX 5

struct Vertex {
    char label;
    bool visited;
};
int stack[MAX];
int top = -1;
struct Vertex* lstVertices[MAX];

int adjMatrix[MAX][MAX];

int vertexCount = 0;

void push(int item) {
    stack[++top] = item;
}

int pop() {
    return stack[top--];
}

int peek() {
    return stack[top];
}

bool isStackEmpty() {
    return top == -1;
}
void addVertex(char label) {
    struct Vertex* vertex = (struct Vertex*) malloc(sizeof(struct Vertex));
    vertex->label = label;
    vertex->visited = false;
    lstVertices[vertexCount++] = vertex;
}
void addEdge(int start,int end) {
    adjMatrix[start][end] = 1;
    adjMatrix[end][start] = 1;
}

void displayVertex(int vertexIndex) {
    printf("%c ",lstVertices[vertexIndex]->label);
```

```
}
int getAdjUnvisitedVertex(int vertexIndex) {
    int i;

    for(i = 0; i < vertexCount; i++) {
        if(adjMatrix[vertexIndex][i] == 1 && lstVertices[i]->visited == false) {
            return i;
        }
    }

    return -1;
}

void depthFirstSearch() {
    int i;
    lstVertices[0]->visited = true;
    displayVertex(0);
    push(0);

    while(!isStackEmpty()) {
        int unvisitedVertex = getAdjUnvisitedVertex(peek());
        if(unvisitedVertex == -1) {
            pop();
        } else {
            lstVertices[unvisitedVertex]->visited = true;
            displayVertex(unvisitedVertex);
            push(unvisitedVertex);
        }
    }
    for(i = 0;i < vertexCount;i++) {
        lstVertices[i]->visited = false;
    }
}

int main() {
    int i, j;

    for(i = 0; i < MAX; i++)     {
        for(j = 0; j < MAX; j++)
            adjMatrix[i][j] = 0;
    }

    addVertex('S');
    addVertex('A');
    addVertex('B');
    addVertex('C');
    addVertex('D');

    addEdge(0, 1);
    addEdge(0, 2);
    addEdge(0, 3);
    addEdge(1, 4);
    addEdge(2, 4);
    addEdge(3, 4);

printf("Depth First Search: ")
    depthFirstSearch();

    return 0;
}
```

Output:

```
Depth First Search:S A D B C
```

/*4.Write a C program breath first search (BFS) using array.*/

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 40

struct queue {
     int items[SIZE];
     int front;
     int rear;
};

struct queue* createQueue();
void enqueue(struct queue* q, int);
int dequeue(struct queue* q);
void display(struct queue* q);
int isEmpty(struct queue* q);
void printQueue(struct queue* q);

struct node
{
     int vertex;
     struct node* next;
};
```

```
struct node* createNode(int);

struct Graph
{
    int numVertices;
    struct node** adjLists;
    int* visited;
};

struct Graph* createGraph(int vertices);
void addEdge(struct Graph* graph, int src, int dest);
void printGraph(struct Graph* graph);
void bfs(struct Graph* graph, int startVertex);

int main()
{
    struct Graph* graph = createGraph(6);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 2);
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 4);
    addEdge(graph, 1, 3);
    addEdge(graph, 2, 4);
    addEdge(graph, 3, 4);

    bfs(graph, 0);

    return 0;
}

void bfs(struct Graph* graph, int startVertex) {

    struct queue* q = createQueue();

    graph->visited[startVertex] = 1;
    enqueue(q, startVertex);

    while(!isEmpty(q)){
        printQueue(q);
        int currentVertex = dequeue(q);
        printf("Visited %d\n", currentVertex);

       struct node* temp = graph->adjLists[currentVertex];
```

```
        while(temp) {
            int adjVertex = temp->vertex;

            if(graph->visited[adjVertex] == 0){
                graph->visited[adjVertex] = 1;
                enqueue(q, adjVertex);
            }
            temp = temp->next;
        }
    }
}


struct node* createNode(int v)
{
    struct node* newNode = malloc(sizeof(struct node));
    newNode->vertex = v;
    newNode->next = NULL;
    return newNode;
}


struct Graph* createGraph(int vertices)
{
    struct Graph* graph = malloc(sizeof(struct Graph));
    graph->numVertices = vertices;

    graph->adjLists = malloc(vertices * sizeof(struct node*));
    graph->visited = malloc(vertices * sizeof(int));



    int i;
    for (i = 0; i < vertices; i++) {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0;
    }

    return graph;
}

void addEdge(struct Graph* graph, int src, int dest)
{
    // Add edge from src to dest
    struct node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

struct queue* createQueue() {
    struct queue* q = malloc(sizeof(struct queue));
    q->front = -1;
    q->rear = -1;
    return q;
}


int isEmpty(struct queue* q) {
    if(q->rear == -1)
        return 1;
    else
        return 0;
}

void enqueue(struct queue* q, int value){
    if(q->rear == SIZE-1)
        printf("\nQueue is Full!!");
    else {
        if(q->front == -1)
            q->front = 0;
        q->rear++;
        q->items[q->rear] = value;
    }
}

int dequeue(struct queue* q){
    int item;
    if(isEmpty(q)){
        printf("Queue is empty");
        item = -1;
    }
    else{
        item = q->items[q->front];
        q->front++;
        if(q->front > q->rear){
            printf("Resetting queue");
            q->front = q->rear = -1;
        }
    }
    return item;
}
void printQueue(struct queue *q) {
    int i = q->front;
    if(isEmpty(q)) {
        printf("Queue is empty");
    } else {
        printf("\nQueue contains \n");
        for(i = q->front; i < q->rear + 1; i++) {
                printf("%d ", q->items[i]);
        }
    }
}
```

```
Output:
Queue contains
0 Resetting queueVisited 0

Queue contains
2 1 Visited 2

Queue contains
1 4 Visited
Queue contains
4 3 Visited 4
Queue contains
3 Resetting queueVisited 3
```

/*5.Write a C program for 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;
}
```

```
Output:
Enter number of elements in array
5
Enter 5 integer(s)
1
2
12
01
76
Enter a number to search
12
12 is present at location 3.
```

/*6. Write a c program for Binary Search Algorithm*/

```
#include<stdio.h>
int main()
{
      int arr[10],i,n,x,flag=0,first,last,mid;

      printf("Enter size of array:");
      scanf("%d",&n);
      printf("\nEnter array element(ascending order)\n");
```

```
    for(i=0;i<n;++i)
        scanf("%d",&arr[i]);

    printf("\nEnter the element to search:");
    scanf("%d",&x);

    first=0;
    last=n-1;

    while(first<=last)
    {
        mid=(first+last)/2;

        if(x==arr[mid]){
            flag=1;
            break;
        }
        else
            if(x>arr[mid])
                first=mid+1;
            else
                last=mid-1;
    }

    if(flag==1)
        printf("\nElement found at position %d",mid+1);
    else
        printf("\nElement not found");

    return 0;
}
```

Output:

```
Enter size of array:4

Enter array element(ascending order)
1
12
13
24

Enter the element to search:12

Element found at position 2
```