Ayush Goyal

CSE D, Roll 62, Reg: 190905522

DAA Lab 1 (Week 1)

Q1) Write a program to construct a binary tree to support the following operations. Assume no duplicate elements while constructing the tree.

i. **Given a key, perform a search in the binary search tree. If the key is found then display "key found" else insert the key in the binary search tree.**

ii. **Display the tree using in order, pre order and post order traversal methods**

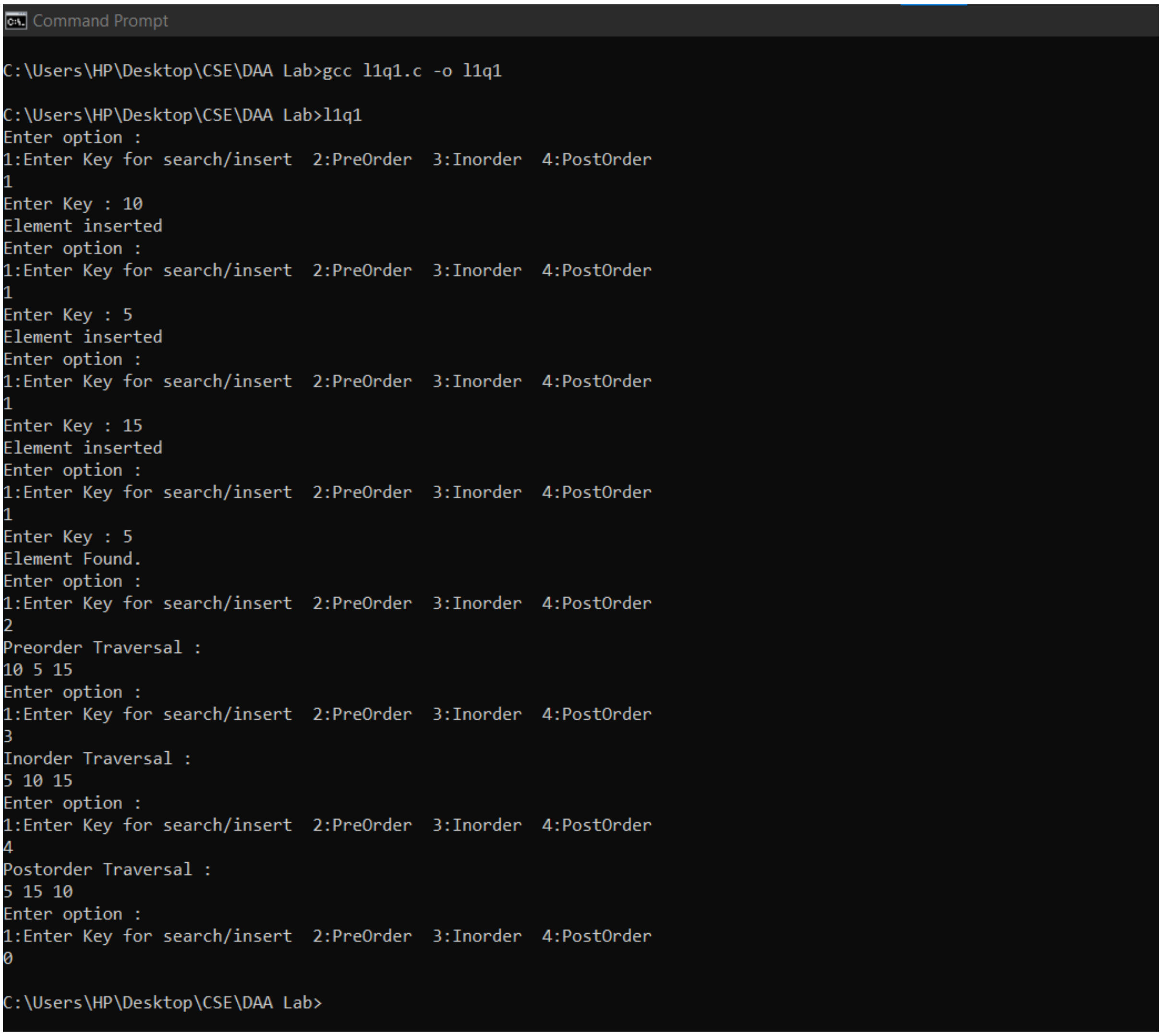
CODE :

```
#include<stdio.h>

#include<stdlib.h>


typedef struct Node* Nodeptr;


typedef struct Node{

   int data;

   Nodeptr left;

   Nodeptr right;

}Node;


Nodeptr search(Nodeptr root, int key){

   if(!root){

      Nodeptr temp = (Nodeptr)malloc(sizeof(Node));

      temp->data = key;

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

      printf("Element inserted\n");

      return temp;

   }
```

```
    if(root->data == key){
        printf("Element Found.\n");
    }
    else if(root->data > key)
        root->left = search(root->left, key);
    else
        root->right = search(root->right, key);
    return root;
    }

void preorder(Nodeptr root){
    if(root){
        printf("%d ",root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

void inorder(Nodeptr root){
    if(root){
        inorder(root->left);
        printf("%d ",root->data);
        inorder(root->right);
    }
}

void postorder(Nodeptr root){
    if(root){
        postorder(root->left);
        postorder(root->right);
        printf("%d ",root->data);
    }
}

int main(){
    int op;
    Nodeptr root = NULL;
    int flag = 1;
    while(flag){
        printf("Enter option : \n");
        printf("1:Enter Key for search/insert  2:PreOrder  3:Inorder  4:PostOrder\n");
        scanf("%d",&op);
        switch(op){
            case 1 : printf("Enter Key : ");
                    int a;
                    scanf("%d",&a);
                    root = search(root, a);
                    break;
            case 2 : printf("Preorder Traversal : \n");
                    preorder(root);
                    printf("\n");
                    break;
            case 3 : printf("Inorder Traversal : \n");
                    inorder(root);
                    printf("\n");
                    break;
            case 4 : printf("Postorder Traversal : \n");
                    postorder(root);
```

```
                printf("\n");
                break;
            default : flag = 0;
        }
    }
    return 0;
}
```

OUPUT :



```
C:\Users\HP\Desktop\CSE\DAA Lab>gcc l1q1.c -o l1q1

C:\Users\HP\Desktop\CSE\DAA Lab>l1q1
Enter option :
1:Enter Key for search/insert  2:PreOrder  3:Inorder  4:PostOrder
1
Enter Key : 10
Element inserted
Enter option :
1:Enter Key for search/insert  2:PreOrder  3:Inorder  4:PostOrder
1
Enter Key : 5
Element inserted
Enter option :
1:Enter Key for search/insert  2:PreOrder  3:Inorder  4:PostOrder
1
Enter Key : 15
Element inserted
Enter option :
1:Enter Key for search/insert  2:PreOrder  3:Inorder  4:PostOrder
1
Enter Key : 5
Element Found.
Enter option :
1:Enter Key for search/insert  2:PreOrder  3:Inorder  4:PostOrder
2
Preorder Traversal :
10 5 15
Enter option :
1:Enter Key for search/insert  2:PreOrder  3:Inorder  4:PostOrder
3
Inorder Traversal :
5 10 15
Enter option :
1:Enter Key for search/insert  2:PreOrder  3:Inorder  4:PostOrder
4
Postorder Traversal :
5 15 10
Enter option :
1:Enter Key for search/insert  2:PreOrder  3:Inorder  4:PostOrder
0

C:\Users\HP\Desktop\CSE\DAA Lab>
```

Q2). Write a program to implement the following graph representations and display them.
i. Adjacency list   ii. Adjacency matrix

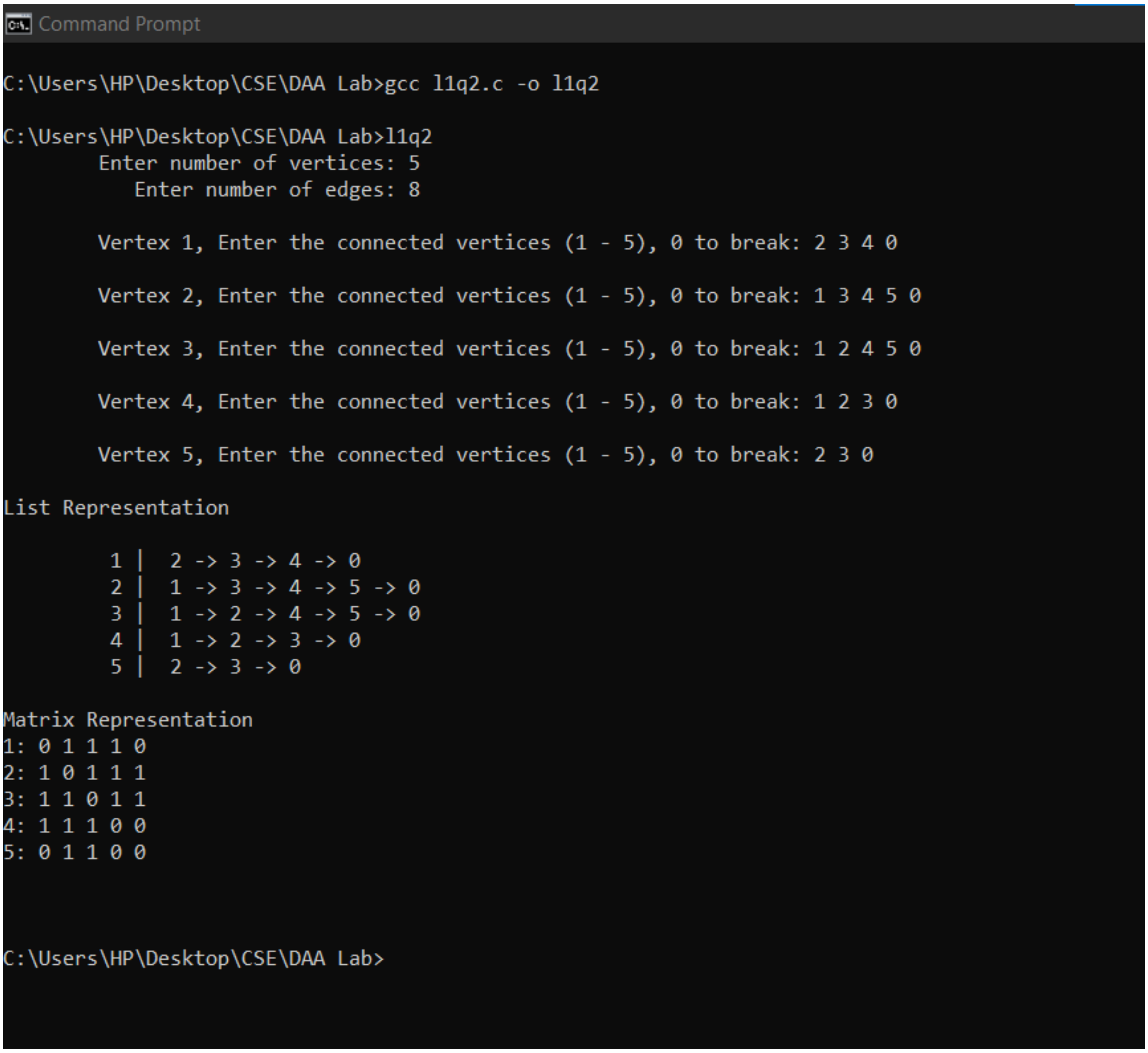
CODE:

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



// Initializing the matrix to zero
void init(int arr[][10], int v) {
 int i, j;
 for (i = 0; i < v; i++)
  for (j = 0; j < v; j++)
   arr[i][j] = 0;
}

// Adding edges
void addEdge(int arr[][10], int i, int j) {
 arr[i][j] = 1;
 arr[j][i] = 1;
}

// Printing the matrix
void printAdjMatrix(int arr[][10], int v) {
 int i, j;

 for (i = 0; i < v; i++) {
  printf("%d: ", i+1);
  for (j = 0; j < v; j++) {
      printf("%d ", arr[i][j]);
    }
    printf("\n");
  }
}

typedef struct AdjNode {
        int vertex;
        struct AdjNode *next;
} ADJ_NODE_t, *ADJ_NODE_p_t;

typedef struct AdjListNode {
        int count;
        ADJ_NODE_p_t head;
} ADJ_LIST_NODE_t, *ADJ_LIST_NODE_p_t;

ADJ_NODE_p_t createAdjNode (int value) {
        ADJ_NODE_p_t adjNode = (ADJ_NODE_p_t)malloc(sizeof(ADJ_NODE_t));
        adjNode->vertex = value;
        adjNode->next = NULL;
        return adjNode;
}

ADJ_LIST_NODE_p_t createAdjListNode () {
        ADJ_LIST_NODE_p_t adjListNode = (ADJ_LIST_NODE_p_t)malloc(sizeof(ADJ_LIST_NODE_t));
        adjListNode->count = 0;
        adjListNode->head = NULL;
        return adjListNode;
}
```

```
void insertAdjNode (ADJ_NODE_p_t *head, int value) {
        if (*head == NULL) {
                *head = createAdjNode(value);
                return;
        }
        ADJ_NODE_p_t temp = *head;
        while (temp->next != NULL)
                temp = temp->next;
        temp->next = createAdjNode(value);
}

ADJ_LIST_NODE_p_t * inputAdjList (int arr[][10], int v) {

        int i, vertex;

        ADJ_LIST_NODE_p_t *listHeadArr = (ADJ_LIST_NODE_p_t *)calloc(v, sizeof(ADJ_LIST_NODE_p_t));

        ADJ_LIST_NODE_p_t temp;
        for (i = 0; i < v; ++i) {
                *(listHeadArr + i) = createAdjListNode();
                temp = *(listHeadArr + i);
                printf("\n\tVertex %d, Enter the connected vertices (1 - %d), 0 to break: ", i+1, v);
                do {
                        scanf(" %d", &vertex);
                        if (vertex != 0)
     addEdge(arr, i, vertex-1);
                                insertAdjNode(&temp->head, vertex);
        } while (vertex != 0);
    }

    return listHeadArr;
}

void printAdjList (ADJ_LIST_NODE_p_t *listHeadArr) {
    int i = 0;
    ADJ_LIST_NODE_p_t temp = *(listHeadArr + i);
    while (temp != NULL) {
        printf("\n\t %d | ", i+1);
        temp = *(listHeadArr + i);
        ADJ_NODE_p_t p = temp->head;
        while (p->next != NULL) {
            printf(" %d ->", p->vertex);
            p = p->next;
        }
        printf(" %d ", p->vertex);
        temp = *(listHeadArr + (++i));
    }
}

int main (int argc, const char * argv []) {

 int adjMatrix[10][10];
    int v, e;
    printf("\tEnter number of vertices: ");
    scanf(" %d", &v);
    printf("\t   Enter number of edges: ");
```

```
        scanf(" %d", &e);

 init(adjMatrix, v);

 ADJ_LIST_NODE_p_t *list = inputAdjList(adjMatrix, v);

 printf("\nList Representation\n");
 printAdjList(list);

 printf("\n\nMatrix Representation\n");

 printAdjMatrix(adjMatrix, v);



        printf("\n\n");

        return 0;
}
```

OUTPUT :



```
C:\Users\HP\Desktop\CSE\DAA Lab>gcc l1q2.c -o l1q2

C:\Users\HP\Desktop\CSE\DAA Lab>l1q2
        Enter number of vertices: 5
           Enter number of edges: 8

        Vertex 1, Enter the connected vertices (1 - 5), 0 to break: 2 3 4 0

        Vertex 2, Enter the connected vertices (1 - 5), 0 to break: 1 3 4 5 0

        Vertex 3, Enter the connected vertices (1 - 5), 0 to break: 1 2 4 5 0

        Vertex 4, Enter the connected vertices (1 - 5), 0 to break: 1 2 3 0

        Vertex 5, Enter the connected vertices (1 - 5), 0 to break: 2 3 0

List Representation

         1 |  2 -> 3 -> 4 -> 0
         2 |  1 -> 3 -> 4 -> 5 -> 0
         3 |  1 -> 2 -> 4 -> 5 -> 0
         4 |  1 -> 2 -> 3 -> 0
         5 |  2 -> 3 -> 0

Matrix Representation
1: 0 1 1 1 0
2: 1 0 1 1 1
3: 1 1 0 1 1
4: 1 1 1 0 0
5: 0 1 1 0 0



C:\Users\HP\Desktop\CSE\DAA Lab>
```

THE END