WEEK 1

Lab Exercises:

1). 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.

Program:

```
#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 the option\n");
printf("1 Enter key to search/insert, 2 Display Preorder, 3 Display Inorder, 4 Display
Postorder\n");
scanf("%d", &op);
switch(op)
{
case 1: printf("Enter the key ");
int a;scanf("%d", &a);
root = search(root, a);
```

```
break;
case 2:printf("Preorder:\n");
preorder(root);
printf("\n");
break;
case 3:printf("Inorder:\n");
inorder(root);
printf("\n");
break;
case 4:printf("Postorder:\n");
postorder(root);
printf("\n");
break;
default: flag = 0;
}
}
return 0;
}
```

Output:

```
Enter the option
1 Enter key to search/insert, 2 Display Preorder, 3 Display Inorder, 4 Display Postorder
1
Enter the key 20
Element inserted
Enter the option
1 Enter key to search/insert, 2 Display Preorder, 3 Display Inorder, 4 Display Postorder
1
Enter the key 30
Element inserted
Enter the option
1 Enter key to search/insert, 2 Display Preorder, 3 Display Inorder, 4 Display Postorder
1
Enter the key 10
Element inserted
Enter the option
1 Enter key to search/insert, 2 Display Preorder, 3 Display Inorder, 4 Display Postorder
1
Enter the key 20
Element found
Enter the option
1 Enter key to search/insert, 2 Display Preorder, 3 Display Inorder, 4 Display Postorder
2
Preorder:
20 10 30
Enter the option
1 Enter key to search/insert, 2 Display Preorder, 3 Display Inorder, 4 Display Postorder
3
Inorder:
10 20 30
Enter the option
1 Enter key to search/insert, 2 Display Preorder, 3 Display Inorder, 4 Display Postorder
4
Postorder:
10 30 20
Enter the option
1 Enter key to search/insert, 2 Display Preorder, 3 Display Inorder, 4 Display Postorder
```

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

Program: i. Adjacency list

```
#include <stdio.h>
#include <stdlib.h>
struct adlistnode{
int dest;
struct adlistnode *next;
};
struct adlist
{
struct adlistnode *head;
};
struct Graph
{
int V;
struct adlist *array;
};
struct adlistnode *newAdjListNode(int dest)
{
struct adlistnode *newNode = (struct adlistnode *)malloc(sizeof(struct adlistnode));
newNode->dest = dest;
newNode->next = NULL;
return newNode;
}
struct Graph *createGraph(int V)
{
struct Graph *graph = (struct Graph *)malloc(sizeof(struct Graph));
graph->V = V;
graph->array = (struct adlist *)malloc(V * sizeof(struct adlist));
for (int i = 0; i < V; i++)
graph->array[i].head = NULL;
return graph;
}
void addEdge(struct Graph *graph, int src, int dest)
{
struct adlistnode *newNode = newAdjListNode(dest);
newNode->next = graph->array[src].head;
graph->array[src].head = newNode;
newNode = newAdjListNode(src);
```

```
newNode->next = graph->array[dest].head;
graph->array[dest].head = newNode;
}
void printGraph(struct Graph *graph)
{
int v;
for (v = 0; v < graph->V; ++v)
{
struct adlistnode *pCrawl = graph->array[v].head;
printf("\n Adjacency list of vertex %d\n head ", v);
while (pCrawl)
{
printf("-> %d", pCrawl->dest);
pCrawl = pCrawl->next;
}
printf("\n");
}}
int main()
{
int V = 5;
struct Graph *graph = createGraph(V);
addEdge(graph, 0, 2);
addEdge(graph, 0, 3);
addEdge(graph, 1, 0);
addEdge(graph, 1, 4);
addEdge(graph, 1, 1);
addEdge(graph, 2, 3);
addEdge(graph, 3, 4);
printGraph(graph);
return 0;
}
```

Output:

```
Adjacency list of vertex 0
head -> 1-> 3-> 2

Adjacency list of vertex 1
head -> 1-> 1-> 4-> 0

Adjacency list of vertex 2
head -> 3-> 0

Adjacency list of vertex 3
head -> 4-> 2-> 0

Adjacency list of vertex 4
head -> 3-> 1
```

Program: ii. Adjacency matrix

```
#include <stdio.h>
int n, m;
void createAdMatrix(int Adj[][n + 1], int arr[][2])
{
for (int i = 0; i < n + 1; i++)
{
for (int j = 0; j < n + 1; j++)
{
Adj[i][j] = 0;
}
}
for (int i = 0; i < m; i++)
{
int x = arr[i][0];
int y = arr[i][1];
Adj[x][y] = 1;
Adj[y][x] = 1;
}
}
void printAdMatrix(int Adj[][n + 1])
{
for (int i = 1; i < n + 1; i++)
{
for (int j = 1; j < n + 1; j++)
{
printf("%d ", Adj[i][j]);
}
printf("\n");
}
}
int main()
{
n = 5;
int arr[][2] = {{1, 3}, {2, 4}, {4, 3}, {2, 5}};
m = sizeof(arr) / sizeof(arr[0]);
int Adj[n + 1][n + 1];
createAdMatrix(Adj, arr);
printAdMatrix(Adj);
return 0;
}
```

Output:

```
0 0 1 0 0
0 0 0 1 1
1 0 0 1 0
0 1 1 0 0
0 1 0 0 0
```