SLIP 1

Q 1. Write a C program that accepts the vertices and edges of a graph and stores it as an adjacency matrix. Display the adjacency matrix. [15 Marks]

SOLUTION:

```
#include <stdio.h>

#define MAX_VERTICES 10

int main()

{

   int vertices, edges;

   printf("Enter the number of vertices: ");

   scanf("%d", &vertices);

if (vertices <= 0 || vertices > MAX_VERTICES) {

      printf("Invalid number of vertices. Please enter a positive integer less than or equal to %d.\n",
MAX_VERTICES);

      return 1;

   }

printf("Enter the number of edges: ");

 scanf("%d", &edges);


   if (edges < 0 || edges > vertices * (vertices - 1) / 2)

{

  printf("Invalid number of edges. Please enter a non-negative integer less than or equal to %d.\n",
vertices * (vertices - 1) / 2);

      return 1;

   }

   int adjMatrix[MAX_VERTICES][MAX_VERTICES] = {0};

   printf("Enter the edges (format: vertex1 vertex2):\n");

   for (int i = 0; i < edges; i++)

{

      int vertex1, vertex2;

      scanf("%d %d", &vertex1, &vertex2);

         if (vertex1 < 0 || vertex1 >= vertices || vertex2 < 0 || vertex2 >= vertices) {
```

```
            printf("Invalid edge. Vertex indices should be between 0 and %d.\n", vertices - 1);
            return 1;
        }
        adjMatrix[vertex1][vertex2] = 1;
        adjMatrix[vertex2][vertex1] = 1;
    }
    printf("Adjacency Matrix:\n");
    for (int i = 0; i < vertices; i++)
 {
        for (int j = 0; j < vertices; j++)
{
            printf("%d\t", adjMatrix[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Q 2. Implement a Binary search tree (BST) library (btree.h) with operations – create, insert, preorder. Write a menu driven program that performs the above operations . [15 Marks]

SOLUTION:

HEADER FILE

```
struct node
{
int data;
struct node *right;
struct node *left;
};
struct node *create(struct node *,int);
struct node *Insert (struct node *,int);
```

```
void preorder(struct node *);
struct node *create(struct node *root,int item)
{
if(root==NULL)
{
root=(struct node *)malloc(sizeof(struct node *));
root->left=root->right=NULL;
root->data=item;
return root;
}
else
{
if(item<root->data)
root->left=create(root->left,item);
else if(item>root->data)
    root->right=create(root->right,item);
else
   printf("duplicate element not allowed");
return(root);
}
}
struct node *Insert(struct node *root,int item)
{
if(root==NULL)
{
root=(struct node *)malloc(sizeof(struct node));
root->data=item;
root->left=root->right=NULL;
}
else
{
```

```
 if(item<root->data)
  root->left=Insert(root->left,item);
else
  root->right=Insert(root->right,item);
return(root);
}
}
void preorder(struct node *root)
{
 if(root!=NULL)
 {
  printf("\n %d",root->data);
  preorder(root->left);
  preorder(root->right);
 }
}
```

MAIN PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#include"btree.h"
void main()
{
struct node *root=NULL;
int ch,n,i,item;
printf("\n enter a choice to perform operation");
while(1)
{
printf("\n1.create BST\t 2.insert\t 3.preorder\t 4.exit");
scanf("%d",&ch);
switch(ch)
{
case 1:root=NULL;
```

```
        printf("\nenter number of nodes:");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
        printf("\n enter data for node %d",i);
        scanf("%d",&item);
        root=create(root,item);
        }
        break;
case 2:Insert(root,item);
        break;
case 3:preorder(root);
        break;
case 4:exit(0);
}
}
}
```

Q3.Viva [5Marks]

SLIP2

Q1. Write a C program for the implementation of Topological sorting. [15 Marks]

SOLUTION:

```
#include <stdio.h>

#include <stdlib.h>
#define MAX 100
int n, adj[MAX][MAX], visited[MAX], stack[MAX], top = -1;
void dfs(int v)
{
   int i;
   visited[v] = 1;
   for (i = 0; i < n; i++)
{
      if (adj[v][i] && !visited[i])
{
         dfs(i);
      }
   }
   stack[++top] = v;
}
void topologicalSort()
{
   int i;
   for (i = 0; i < n; i++)
{
      visited[i] = 0;
   }
   for (i = 0; i < n; i++) {
      if (!visited[i])
{
         dfs(i);
      }
   }
```

```
   while (top != -1)
 {
      printf("%d ", stack[top--]);
   }
}
int main()
 {
   int i, j;
   printf("Enter number of vertices: ");
   scanf("%d", &n);
   printf("Enter adjacency matrix:\n");
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         scanf("%d", &adj[i][j]);
      }
   }
   printf("Topological Sort: ");
   topologicalSort();
   return 0;
}
```

Q2.write a program that accepts the vertices and edges of a graph and stores it as an adjacency matrix. Display the adjacency matrix. [15 Marks]

SOLUTION:

```
#include <stdio.h>
#define MAX_VERTICES 10
int main()
{
   int vertices, edges;
   printf("Enter the number of vertices: ");
   scanf("%d", &vertices);
if (vertices <= 0 || vertices > MAX_VERTICES) {
      printf("Invalid number of vertices. Please enter a positive integer less than or equal to %d.\n",
MAX_VERTICES);
```

```
        return 1;
    }
printf("Enter the number of edges: ");
 scanf("%d", &edges);


    if (edges < 0 || edges > vertices * (vertices - 1) / 2)
 {
   printf("Invalid number of edges. Please enter a non-negative integer less than or equal to %d.\n",
vertices * (vertices - 1) / 2);
        return 1;
    }
    int adjMatrix[MAX_VERTICES][MAX_VERTICES] = {0};
    printf("Enter the edges (format: vertex1 vertex2):\n");
    for (int i = 0; i < edges; i++)
 {
        int vertex1, vertex2;
        scanf("%d %d", &vertex1, &vertex2);
           if (vertex1 < 0 || vertex1 >= vertices || vertex2 < 0 || vertex2 >= vertices) {
           printf("Invalid edge. Vertex indices should be between 0 and %d.\n", vertices - 1);
           return 1;
        }
        adjMatrix[vertex1][vertex2] = 1;
        adjMatrix[vertex2][vertex1] = 1;
    }
    printf("Adjacency Matrix:\n");
    for (int i = 0; i < vertices; i++)
 {
        for (int j = 0; j < vertices; j++)
{
           printf("%d\t", adjMatrix[i][j]);
        }
        printf("\n");
    }


    return 0;
}
```

Q 3. Viva [5 Marks]

SLIP 3

Q 1. Write a C program for the Implementation of Prim's Minimum spanning tree algorithm.

[15 Marks]

SOLUTION:

```
#include <stdio.h>
#include <stdbool.h>
#define INF 9999999
int main() {
   int n, i, j, u, v;
   printf("Enter the number of vertices: ");
   scanf("%d", &n);
   int cost[n][n], minCost = 0;
   int parent[n], key[n];
   bool visited[n];
   printf("Enter the cost matrix:\n");
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         scanf("%d", &cost[i][j]);
         if (cost[i][j] == 0)
            cost[i][j] = INF;
     }
   }

   for (i = 0; i < n; i++) {
      key[i] = INF;
      visited[i] = false;
   }
   key[0] = 0;
   parent[0] = -1;
   for (i = 0; i < n - 1; i++)
```

```
{
      int minKey = INF;
      for (j = 0; j < n; j++)
{
         if (visited[j] == false && key[j] < minKey)
{
            minKey = key[j];
            u = j;
         }
      }
     visited[u] = true;
     for (v = 0; v < n; v++)
{
         if (visited[v] == false && cost[u][v] < key[v])
{
            parent[v] = u;
            key[v] = cost[u][v];
         }
      }
   }
   printf("Minimum Spanning Tree:\n");
   for (i = 1; i < n; i++)
{
      printf("%d - %d\n", parent[i], i);
      minCost += cost[i][parent[i]];
   }
   printf("Minimum Cost: %d\n", minCost);
   return 0;
}
```

Q 2. Write a C program that accepts the vertices and edges of a graph and stores it as an adjacency matrix. Display the adjacency matrix. [15 Marks]

SOLUTION:

```
#include <stdio.h>

#define MAX_VERTICES 10

int main()

{

   int vertices, edges;

   printf("Enter the number of vertices: ");

   scanf("%d", &vertices);

if (vertices <= 0 || vertices > MAX_VERTICES) {

      printf("Invalid number of vertices. Please enter a positive integer less than or equal to %d.\n",
MAX_VERTICES);

      return 1;

   }

printf("Enter the number of edges: ");

 scanf("%d", &edges);


   if (edges < 0 || edges > vertices * (vertices - 1) / 2)

 {

  printf("Invalid number of edges. Please enter a non-negative integer less than or equal to %d.\n",
vertices * (vertices - 1) / 2);

      return 1;

   }

   int adjMatrix[MAX_VERTICES][MAX_VERTICES] = {0};

   printf("Enter the edges (format: vertex1 vertex2):\n");

   for (int i = 0; i < edges; i++)

 {

      int vertex1, vertex2;

      scanf("%d %d", &vertex1, &vertex2);

         if (vertex1 < 0 || vertex1 >= vertices || vertex2 < 0 || vertex2 >= vertices) {

         printf("Invalid edge. Vertex indices should be between 0 and %d.\n", vertices - 1);
```

```
            return 1;
        }
        adjMatrix[vertex1][vertex2] = 1;
        adjMatrix[vertex2][vertex1] = 1;
    }
    printf("Adjacency Matrix:\n");
    for (int i = 0; i < vertices; i++)
 {
        for (int j = 0; j < vertices; j++)
{
            printf("%d\t", adjMatrix[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Q 3. Viva [5 Marks]

SLIP 4

Q 1. Write a C program that accepts the vertices and edges of a graph. Create adjacency list.

[15 Marks]

SOLUTION:

```
#include  <stdio.h>
#include <stdlib.h>
struct Node
{
   int vertex;
   struct Node* next;
};
struct Node* createNode(int v)
 {
   struct Node* newNode = malloc(sizeof(struct Node));
   newNode->vertex = v;
   newNode->next = NULL;
   return newNode;
}
void addEdge(struct Node** adjList, int src, int dest)
 {
   struct Node* newNode = createNode(dest);
   newNode->next = adjList[src];
   adjList[src] = newNode;
}
void printAdjList(struct Node** adjList, int vertices)
 {
   for (int i = 0; i < vertices; i++)
 {
      struct Node* temp = adjList[i];
      printf("Adjacency list of vertex %d\n", i);
      while (temp) {
```

```
            printf("%d -> ", temp->vertex);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}
int main()
{
    int vertices, edges;
    printf("Enter the number of vertices: ");
    scanf("%d", &vertices);
    printf("Enter the number of edges: ");
    scanf("%d", &edges);
   struct Node** adjList = malloc(vertices * sizeof(struct Node*));
    for (int i = 0; i < vertices; i++)
{
        adjList[i] = NULL;
    }
    for (int i = 0; i < edges; i++)
{
        int src, dest;
        printf("Enter edge %d (source destination): ", i + 1);
        scanf("%d %d", &src, &dest);
        addEdge(adjList, src, dest);
    }
    printAdjList(adjList, vertices);
    return 0;
}
```

Q 2. Write a C program for the implementation of Topological sorting. [15 Marks]

SOLUTION:

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

```
#include <stdlib.h>

#define MAX 100

int n, adj[MAX][MAX], visited[MAX], stack[MAX], top = -1;

void dfs(int v)

{

   int i;

   visited[v] = 1;

   for (i = 0; i < n; i++)

 {

      if (adj[v][i] && !visited[i])

 {

         dfs(i);

      }

   }

   stack[++top] = v;

}

void topologicalSort()

{

   int i;

   for (i = 0; i < n; i++)

 {

      visited[i] = 0;

   }

   for (i = 0; i < n; i++) {
```

```
        if (!visited[i])
    {
            dfs(i);
        }
    }
    while (top != -1)
    {
        printf("%d ", stack[top--]);
    }
}
int main()
{
    int i, j;
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter adjacency matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &adj[i][j]);
        }
    }
    printf("Topological Sort: ");
    topologicalSort();
    return 0;
```

Q 3.Viva [5 Marks]

SLIP 6

Q 1. Write a C program for the Implementation of Prim's Minimum spanning tree algorithm.

[15 Marks]

SOLUTION:

```
#include <stdio.h>
#include <stdbool.h>
#define INF 9999999
int main() {
   int n, i, j, u, v;
   printf("Enter the number of vertices: ");
   scanf("%d", &n);
   int cost[n][n], minCost = 0;
   int parent[n], key[n];
   bool visited[n];
   printf("Enter the cost matrix:\n");
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         scanf("%d", &cost[i][j]);
         if (cost[i][j] == 0)
            cost[i][j] = INF;
      }
   }
   for (i = 0; i < n; i++) {
      key[i] = INF;
      visited[i] = false;
   }
   key[0] = 0;
   parent[0] = -1;
   for (i = 0; i < n - 1; i++) {
      int minKey = INF;
      for (j = 0; j < n; j++) {
```

```
            if (visited[j] == false && key[j] < minKey) {
                minKey = key[j];
                u = j;
            }
        }
        visited[u] = true;
        for (v = 0; v < n; v++) {
            if (visited[v] == false && cost[u][v] < key[v]) {
                parent[v] = u;
                key[v] = cost[u][v];
            }
        }
    }
    printf("Minimum Spanning Tree:\n");
    for (i = 1; i < n; i++) {
        printf("%d - %d\n", parent[i], i);
        minCost += cost[i][parent[i]];
    }
    printf("Minimum Cost: %d\n", minCost);
   return 0;
}
```

Q 2. Write aCprogram that acceptsthe vertices and edges of a graph and store sit as an adjacency matrix. Display the adjacency matrix. [15Marks]

SOLUTION:

```
#include <stdio.h>
#define MAX_VERTICES 10
int main()
{
    int vertices, edges;
    printf("Enter the number of vertices: ");
```

```
   scanf("%d", &vertices);
if (vertices <= 0 || vertices > MAX_VERTICES) {
      printf("Invalid number of vertices. Please enter a positive integer less than or equal to %d.\n",
MAX_VERTICES);
      return 1;
   }
printf("Enter the number of edges: ");
 scanf("%d", &edges);

   if (edges < 0 || edges > vertices * (vertices - 1) / 2)
 {
  printf("Invalid number of edges. Please enter a non-negative integer less than or equal to %d.\n",
vertices * (vertices - 1) / 2);
      return 1;
   }
   int adjMatrix[MAX_VERTICES][MAX_VERTICES] = {0};
   printf("Enter the edges (format: vertex1 vertex2):\n");
   for (int i = 0; i < edges; i++)
 {
      int vertex1, vertex2;
      scanf("%d %d", &vertex1, &vertex2);
         if (vertex1 < 0 || vertex1 >= vertices || vertex2 < 0 || vertex2 >= vertices) {
         printf("Invalid edge. Vertex indices should be between 0 and %d.\n", vertices - 1);
         return 1;
      }
      adjMatrix[vertex1][vertex2] = 1;
      adjMatrix[vertex2][vertex1] = 1;
   }
   printf("Adjacency Matrix:\n");
   for (int i = 0; i < vertices; i++)
 {
      for (int j = 0; j < vertices; j++)
```

```
{
            printf("%d\t", adjMatrix[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Q 3. Viva [5Marks]

SLIP 7

Q 1. Write a C program for the implementation of Floyd Warshall's algorithm for finding all pairs shortest path using adjacency cost matrix. [15 Marks]

SOLUTION:

```
#include <stdio.h>
#include <limits.h>
#define INF INT_MAX
#define V 4
void printSolution(int dist[][V]);
void floydWarshall(int graph[][V])
{
   int dist[V][V];
   for (int i = 0; i < V; i++)
      for (int j = 0; j < V; j++)
         dist[i][j] = graph[i][j];
   for (int k = 0; k < V; k++)
{
      for (int i = 0; i < V; i++)
{
         for (int j = 0; j < V; j++)
{
            if (dist[i][k] != INF && dist[k][j] != INF && dist[i][k] + dist[k][j] < dist[i][j])
               dist[i][j] = dist[i][k] + dist[k][j];
         }
      }
   }
   printSolution(dist);
}

void printSolution(int dist[][V])
{
```

```
    printf("Shortest distances between every pair of vertices:\n");
    for (int i = 0; i < V; i++)
 {
        for (int j = 0; j < V; j++)
{
            if (dist[i][j] == INF)
                printf("INF\t");
            else
                printf("%d\t", dist[i][j]);
        }
        printf("\n");
    }
}

int main()
 {
    int graph[V][V] = {
        {0, 5, INF, 10},
        {INF, 0, 3, INF},
        {INF, INF, 0, 1},
        {INF, INF, INF, 0}
    };
    floydWarshall(graph);
    return 0;
}
```

Q2. Write a program to sort n randomly generated elements using Heap sort method. [15 Marks]

SOLUTION:

```
#include <stdio.h>
void swap(int* a, int* b) {
    int temp = *a;
```

```
    *a = *b;
    *b = temp;
}
void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    if (left < n && arr[left] > arr[largest])
        largest = left;
    if (right < n && arr[right] > arr[largest])
        largest = right;
    if (largest != i) {
        swap(&arr[i], &arr[largest]);
        heapify(arr, n, largest);
    }
}
void heapSort(int arr[], int n) {
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);
    for (int i = n - 1; i > 0; i--) {
        swap(&arr[0], &arr[i]);
        heapify(arr, i, 0);
    }
}
void printArray(int arr[], int n) {
    for (int i = 0; i < n; ++i)
        printf("%d ", arr[i]);
    printf("\n");
}
int main() {
    int arr[] = { 12, 11, 13, 5, 6, 7 };
```

```
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Original array: \n");
    printArray(arr, n);

    heapSort(arr, n);

    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Q3.Viva [5Marks]

SLIP 9

Q 1. Write a C program that accepts the vertices and edges of a graph. Create adjacency list and display the adjacency list. [15 Marks]

SOLUTION:

```
#include <stdio.h>

#define MAX_VERTICES 10

int main()

{

   int vertices, edges;

   printf("Enter the number of vertices: ");

   scanf("%d", &vertices);

if (vertices <= 0 || vertices > MAX_VERTICES) {

      printf("Invalid number of vertices. Please enter a positive integer less than or equal to %d.\n",
MAX_VERTICES);

      return 1;

   }

printf("Enter the number of edges: ");

 scanf("%d", &edges);


   if (edges < 0 || edges > vertices * (vertices - 1) / 2)

{

  printf("Invalid number of edges. Please enter a non-negative integer less than or equal to %d.\n",
vertices * (vertices - 1) / 2);

      return 1;

   }

   int adjMatrix[MAX_VERTICES][MAX_VERTICES] = {0};

   printf("Enter the edges (format: vertex1 vertex2):\n");

   for (int i = 0; i < edges; i++)

{

      int vertex1, vertex2;

      scanf("%d %d", &vertex1, &vertex2);

         if (vertex1 < 0 || vertex1 >= vertices || vertex2 < 0 || vertex2 >= vertices) {
```

```
            printf("Invalid edge. Vertex indices should be between 0 and %d.\n", vertices - 1);
            return 1;
        }
        adjMatrix[vertex1][vertex2] = 1;
        adjMatrix[vertex2][vertex1] = 1;
    }
    printf("Adjacency Matrix:\n");
    for (int i = 0; i < vertices; i++)
 {
        for (int j = 0; j < vertices; j++)
{
            printf("%d\t", adjMatrix[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Q 2. Implement a Binary search tree (BST) library (btree.h) with operations – create, insert, postorder. Write a menu driven program that performs the above operations. [15 Marks]

SOLUTION:

HEADER FILE

```
struct node
{
int data;
struct node *right;
struct node *left;
};
struct node *create(struct node *,int);
struct node *insert(struct node *,int);
```

```
void preorder(struct node *);
struct  node *create (struct node *root,int item)
{
if(root==NULL)
{
root=(struct node *)malloc(sizeof(struct node *));
root->left=root->right=NULL;
root->data=item;
return root;
}
else
{
if(item<root->data)
root->left=create(root->left,item);
else if(item>root->data)
    root->right=create(root->right,item);
else
   printf("duplicate eklement not allowed");
return(root);
}
}
struct node *insert(struct node * root,int item)
{
if(root==NULL)
{
root=(struct node*)malloc(sizeof(struct node));
root->data=item;
root->left=root->right=NULL;
}
else
{
```

```
 if(item<root->data)
  root->left=insert(root->left,item);
else
  root->right=insert(root->right,item);
return(root);
}
}
void postorder(struct node *root)
{
 if(root!=NULL)
 {
  postorder(root->left);
  postorder(root->right);
  printf("\n%d",root->data);
 }
}
```

MAIN PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#include"btree.h"
void main()
{
struct node *root=NULL;
int ch,n,i,item;
printf("\n enter a choice to perform operation");
while(1)
{
printf("\n1.create BST\t 2.insert\t 3.postorder\t 4.exit");
scanf("%d",&ch);
switch(ch)
{
```

```
case 1:root=NULL;
      printf("\nenter number of nodes:");
      scanf("%d",&n);
      for(i=1;i<=n;i++)
      {
      printf("\n enter data for node %d",i);
      scanf("%d",&item);
      root=create(root,item);
      }
      break;
case 2:insert(root,item);
      break;
case 3:postorder(root);
      break;
case 4:exit(0);
}
}
}
```

Q 3. Viva [5 Marks]

SLIP 10

Q 1. Implement a Binary search tree (BST) library (btree.h) with operations – create, insert, inorder. Write a menu driven program that performs the above operations. [15 Marks]

SOLUTION:

HEADER FILE

```
struct node
{
int data;
struct node *right;
struct node *left;
};
struct node *create(struct node *,int);
struct node *insert(struct node *,int);
void inorder(struct node *);
struct  node *create(struct node *root,int item)
{
if(root==NULL)
{
root=(struct node *)malloc(sizeof(struct node *));
root->left=root->right=NULL;
root->data=item;
return root;
}
else
{
if(item<root->data)
root->left=create(root->left,item);
else if(item>root->data)
    root->right=create(root->right,item);
else
   printf("duplicate eklement not allowed");
```

```
return(root);
}
}
struct node *insert(struct node * root,int item)
{
if(root==NULL)
{
root=(struct node *)malloc(sizeof(struct node));
root->data=item;
root->left=root->right=NULL;
}
else
{
 if(item<root->data)
  root->left=insert(root->left,item);
else
  root->right=insert(root->right,item);
return(root);
}
}
void inorder(struct node *root)
{
 if(root!=NULL)
 {
  inorder(root->left);
  printf("\n%d",root->data);
  inorder(root->right);

 }
}
```

MAIN PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#include"btree.h"
void main()
{
struct node *root=NULL;
int ch,n,i,item;
printf("\n enter a choice to perform operation");
while(1)
{
printf("\n1.create BST\t 2.insert\t 3.inorder\t 4.exit");
scanf("%d",&ch);
switch(ch)
{
case 1:root=NULL;
      printf("\nenter number of nodes:");
      scanf("%d",&n);
      for(i=1;i<=n;i++)
      {
      printf("\n enter data for node %d",i);
      scanf("%d",&item);
      root=create(root,item);
      }
      break;
case 2:insert(root,item);
      break;
case 3:inorder(root);
      break;
case 4:exit(0);
}
}
```

}

Q 2. Write a C program that accepts the vertices and edges of a graph. Create adjacency list and display the adjacency list. [15 Marks]

SOLUTION:

```
#include <stdio.h>

#define MAX_VERTICES 10

int main()

{

   int vertices, edges;

   printf("Enter the number of vertices: ");

   scanf("%d", &vertices);

if (vertices <= 0 || vertices > MAX_VERTICES) {

      printf("Invalid number of vertices. Please enter a positive integer less than or equal to %d.\n",
MAX_VERTICES);

      return 1;

   }

printf("Enter the number of edges: ");

 scanf("%d", &edges);


   if (edges < 0 || edges > vertices * (vertices - 1) / 2)

 {

  printf("Invalid number of edges. Please enter a non-negative integer less than or equal to %d.\n",
vertices * (vertices - 1) / 2);

      return 1;

   }

   int adjMatrix[MAX_VERTICES][MAX_VERTICES] = {0};

   printf("Enter the edges (format: vertex1 vertex2):\n");

   for (int i = 0; i < edges; i++)

 {
```

```
        int vertex1, vertex2;
        scanf("%d %d", &vertex1, &vertex2);
            if (vertex1 < 0 || vertex1 >= vertices || vertex2 < 0 || vertex2 >= vertices) {
            printf("Invalid edge. Vertex indices should be between 0 and %d.\n", vertices - 1);
            return 1;
        }
        adjMatrix[vertex1][vertex2] = 1;
        adjMatrix[vertex2][vertex1] = 1;
    }
    printf("Adjacency Matrix:\n");
    for (int i = 0; i < vertices; i++)
 {
        for (int j = 0; j < vertices; j++)
{
            printf("%d\t", adjMatrix[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Q3.Viva [5Marks]

SLIP 12

Q 1. Implement a Binary search tree (BST) library (btree.h) with operations – create, insert, preorder. Write a menu driven program that performs the above operations. [15 Marks]

SOLUTION:

HEADER FILE

```
struct node
{
int data;
struct node *right;
struct node *left;
};
struct node *create(struct node *,int);
struct node *Insert (struct node *,int);
void preorder(struct node *);
struct node *create(struct node *root,int item)
{
if(root==NULL)
{
root=(struct node *)malloc(sizeof(struct node *));
root->left=root->right=NULL;
root->data=item;
return root;
}
else
{
if(item<root->data)
root->left=create(root->left,item);
else if(item>root->data)
    root->right=create(root->right,item);
else
   printf("duplicate element not allowed");
```

```
return(root);
}
}
struct node *Insert(struct node *root,int item)
{
if(root==NULL)
{
root=(struct node *)malloc(sizeof(struct node));
root->data=item;
root->left=root->right=NULL;
}
else
{
 if(item<root->data)
  root->left=Insert(root->left,item);
else
  root->right=Insert(root->right,item);
return(root);
}
}
void preorder(struct node *root)
{
 if(root!=NULL)
 {
  printf("\n %d",root->data);
  preorder(root->left);
  preorder(root->right);
 }
}
```

MAIN PROGRAM

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

```
#include<stdlib.h>
#include"btree.h"
void main()
{
struct node *root=NULL;
int ch,n,i,item;
printf("\n enter a choice to perform operation");
while(1)
{
printf("\n1.create BST\t 2.insert\t 3.preorder\t 4.exit");
scanf("%d",&ch);
switch(ch)
{
case 1:root=NULL;
      printf("\nenter number of nodes:");
      scanf("%d",&n);
      for(i=1;i<=n;i++)
      {
      printf("\n enter data for node %d",i);
      scanf("%d",&item);
      root=create(root,item);
      }
      break;
case 2:Insert(root,item);
      break;
case 3:preorder(root);
      break;
case 4:exit(0);
}
}
}
```

Q 2. Write a C program for the implementation of Topological sorting. [15 Marks]

SOLUTION:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
int n, adj[MAX][MAX], visited[MAX], stack[MAX], top = -1;
void dfs(int v)
{
   int i;
   visited[v] = 1;
   for (i = 0; i < n; i++)
{
      if (adj[v][i] && !visited[i])
{
         dfs(i);
      }
   }
   stack[++top] = v;
}
void topologicalSort()
{
   int i;
   for (i = 0; i < n; i++)
{
      visited[i] = 0;
   }
   for (i = 0; i < n; i++) {
      if (!visited[i])
{
         dfs(i);
      }
   }
   while (top != -1)
```

```
 {
       printf("%d ", stack[top--]);
    }
}
int main()
 {
    int i, j;
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter adjacency matrix:\n");
    for (i = 0; i < n; i++) {
       for (j = 0; j < n; j++) {
          scanf("%d", &adj[i][j]);
       }
    }
    printf("Topological Sort: ");
    topologicalSort();
    return 0;
}
```

Q 3. Viva [5Marks]

SLIP 18

Q 1. Write a C program that accepts the vertices and edges of a graph and stores it as an adjacency matrix. Display the adjacency matrix. [15 Marks]

SOLUTION:

```
#include <stdio.h>

#define MAX_VERTICES 10

int main()

{

   int vertices, edges;

   printf("Enter the number of vertices: ");

   scanf("%d", &vertices);

if (vertices <= 0 || vertices > MAX_VERTICES) {

      printf("Invalid number of vertices. Please enter a positive integer less than or equal to %d.\n",
MAX_VERTICES);

      return 1;

   }

printf("Enter the number of edges: ");

 scanf("%d", &edges);


   if (edges < 0 || edges > vertices * (vertices - 1) / 2)

{

  printf("Invalid number of edges. Please enter a non-negative integer less than or equal to %d.\n",
vertices * (vertices - 1) / 2);

      return 1;

   }

   int adjMatrix[MAX_VERTICES][MAX_VERTICES] = {0};

   printf("Enter the edges (format: vertex1 vertex2):\n");

   for (int i = 0; i < edges; i++)

{

      int vertex1, vertex2;

      scanf("%d %d", &vertex1, &vertex2);

         if (vertex1 < 0 || vertex1 >= vertices || vertex2 < 0 || vertex2 >= vertices) {
```

```
            printf("Invalid edge. Vertex indices should be between 0 and %d.\n", vertices - 1);
            return 1;
        }
        adjMatrix[vertex1][vertex2] = 1;
        adjMatrix[vertex2][vertex1] = 1;
    }
    printf("Adjacency Matrix:\n");
    for (int i = 0; i < vertices; i++)
 {
        for (int j = 0; j < vertices; j++)
{
            printf("%d\t", adjMatrix[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Q 2. Implement a Binary search tree (BST) library (btree.h) with operations – create, insert, inorder. Write a menu driven program that performs the above operations. [15 Marks]

SOLUTION:

HEADER FILE

```
struct node
{
int data;
struct node *right;
struct node *left;
};
struct node *create(struct node *,int);
struct node *insert(struct node *,int);
```

```
void inorder(struct node *);
struct  node *create(struct node *root,int item)
{
if(root==NULL)
{
root=(struct node *)malloc(sizeof(struct node *));
root->left=root->right=NULL;
root->data=item;
return root;
}
else
{
if(item<root->data)
root->left=create(root->left,item);
else if(item>root->data)
    root->right=create(root->right,item);
else
   printf("duplicate eklement not allowed");
return(root);
}
}
struct node *insert(struct node * root,int item)
{
if(root==NULL)
{
root=(struct node *)malloc(sizeof(struct node));
root->data=item;
root->left=root->right=NULL;
}
else
{
```

```
 if(item<root->data)
  root->left=insert(root->left,item);
else
  root->right=insert(root->right,item);
return(root);
}
}
void inorder(struct node *root)
{
 if(root!=NULL)
 {
  inorder(root->left);
  printf("\n%d",root->data);
  inorder(root->right);

 }
}
```

MAIN PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#include"btree.h"
void main()
{
struct node *root=NULL;
int ch,n,i,item;
printf("\n enter a choice to perform operation");
while(1)
{
printf("\n1.create BST\t 2.insert\t 3.inorder\t 4.exit");
scanf("%d",&ch);
switch(ch)
```

```
{
case 1:root=NULL;
      printf("\nenter number of nodes:");
      scanf("%d",&n);
      for(i=1;i<=n;i++)
      {
      printf("\n enter data for node %d",i);
      scanf("%d",&item);
      root=create(root,item);
      }
      break;
case 2:insert(root,item);
      break;
case 3:inorder(root);
      break;
case 4:exit(0);
}
}
}
```

Q 3. Viva [5Marks]