Experiment No-01: Introduction to Structure in C++

Objectives

- Introduce with structure in C++.
- Learn how to use pointers and structure together.
- Learn how to use the structure with functions.

Example 1: Write a C++ to define a structure.

```
#include<iostream>
#include <string>
using namespace std;


// create struct with person1 variable
struct Person {
  string name;
  int citNo;
  float salary;
} person1;

int main() {


   // assign value to name of person1
    getline(cin, person1.name);


  strcpy(person1.name, "Ronaldo");

  // assign values to other person1 variables
  person1.citNo = 1985;
  person1. salary = 2500;

  // print struct variables
  cout<<"Name: "<< person1.name<<endl;
  cout<<"Citizenship No.: "<< person1.citNo<<endl;
  cout<<"Salary: "<< person1.salary;

  return 0;
}
```

Example 2: Write a C++ program to access structure members using pointers.

```
#include<iostream>
```

```
using namespace std;

struct person
{
   int age;
   float weight;
};

int main()
{
    struct person *personPtr, person1;
    personPtr = &person1;

    cout<<"Enter age: ";
    cin>>personPtr->age;

    cout<<"Enter weight: ";
    cin>>personPtr->weight;


    cout<<"Displaying:\n";
    cout<<"Age: "<< (*personPtr).age<<endl;
    cout<<"weight: "<< personPtr->weight<<endl;

    return 0;

}
```

Example 3: Write a C++ program to pass structs to a function.

```
#include<iostream>
#include <string>
using namespace std;


struct student {
   string name;
   int age;
};

// function prototype
void display(struct student s);

int main() {
   struct student s1;

   cout<<"Enter name: "<<endl;
```

```
    // read string input from the user
    getline(cin,s1.name);


    cout<<"Enter age: "<<endl;
    cin>>s1.age;

    display(s1); // passing struct as an argument

    return 0;
}

void display(struct student s) {
    cout<<"Displaying information"<<endl;
    cout<<"Name: "<< s.name;
    cout<<"\nAge: "<< s.age;
}
```

*** For better understanding please feel free to search on internet because it is the best source of learning. ***

Practice Exercise

1. Write a C++ program to store and print the roll no., name, age, and marks of a student using structures.
2. Write a C++ program to store roll no. (starting from 1), name and age of 5 students and then print the details of the student with roll no. 2.
3. Enter the marks of 5 students in Chemistry, Mathematics, and Physics (each out of 100) using a structure named Marks having elements roll no., name, chem_marks, maths_marks, and phy_marks and then display the percentage of each student.
4. Write a C++ program to add two distances in inch-feet using structure.
5. Write a C++ program to subtract two complex numbers.

Experiment No-02: Introduction to Linked List.

Objectives

- Introduce with the single linked list.
- Create a singly linked list.
- Learn insertion operation in the singly linked list.

Prerequisite: [Function], [Pointer], and [Structure].

Example 1: Create a Single Linked List.

```
#include<iostream>
#include<stdlib.h>
#include<bits/stdc++.h>

using namespace std;

// Create a Node Data Type
struct Node{
    int data;
    Node *next;
};


int main()
{
    //Initialize three nodes with NULL pointer
    Node *a =NULL,*b=NULL,*c=NULL;

    // Allocate Memory for each node
    a = (Node*) malloc(sizeof(Node));
    b = (Node*) malloc(sizeof(Node));
    c = (Node*) malloc(sizeof(Node));

    // Insert Data and Connect the nodes
    a->data = 10;
    b->data = 20;
    c->data = 30;
    a->next = b;
    b->next = c;
    c->next = NULL;

    //Traverse the Linked list
    while (a!=NULL){
        cout<<a->data<<" ";
        a = a->next;
    }
}
```

Example 2: Create a linked list from an array and return the head.

```
#include<bits/stdc++.h>
using namespace std;

// Create a Node Data Type
struct Node
{
    int data;
    Node *next;

    // Set Node value and next pointer
    Node(int x)
    {
        data = x;
        next = nullptr;
    }
};
// Create Linked List Function
Node* constructLL(int arr[], int arrsize) {

    Node *head = new Node(arr[0]); // new work as a malloc function
    Node *current = head; // keep track of the new node

    for (int i = 1; i<arrsize; i++)
    {
        Node *temp = new Node(arr[i]); // new node
        current->next = temp;
        current = temp;
    }
    return head;
}
// Traverse Function
void TraverseList(Node *head)
{
    while (head!=nullptr)
    {
        cout<<head->data<<" ";
        head = head->next;
    }
}
// Main Function
int main()
{
    int arr [8] = {2,4,5,6};
    // Construct Linked List
    Node *head = constructLL(arr,4);
    // Print the List
    TraverseList(head);
}
```

Example 3: Insert a node at the beginning of the list.

```
#include<bits/stdc++.h>
using namespace std;
// This program only included the Function

Node* insertAtFirst(Node* head, int newValue) {

    Node* current = nullptr;
    // Edge Case: The list could be empty
    if (head==nullptr)
    {
        current = new Node(newValue);
        head = current;

        return head;
    }

    current = new Node(newValue);
    current->next = head;
    head = current;

    return head;

}
```

Example 3: Insert a node at the end of the list. [Assume the list already has two nodes.]

```
#include<bits/stdc++.h>
using namespace std;
// This program only included the Function

/* Possible Edge Cases:
1) The list could be empty
2) The list has only one node
3) The list has more than one node
*/
Node* insertAtLast(Node* head, int newValue) {
    Node *temp = head, *current;

    while (temp->next!= nullptr)
    {
           temp = temp->next;
    }
    current = new Node(newValue);
    temp->next = current;

    return head;

}
```

Practice Exercise

1. Write a C++ program to insert a new node at the end of a Singly Linked List [Consider all edge cases].
2. Write a C++ program to find the length of a singly linked list.
3. Write a C++ program to delete the first node of a Singly Linked List.
4. Write a C++ program to delete the last node of a Singly Linked List.

Resources (Link)

Please try to solve similar problems at an online Judge.

1. Create Linked List
2. Insert a Node
3. Delete a Node

Experiment No-03: Advanced Linked List.

Objectives

- Insert and delete at a particular position from a single linked list (SLL).
- Reverse a single linked list.
- Create a doubly linked list (DLL).

Prerequisite: [Function], [Pointer], and [Structure].

Example 1: Delete element from a particular position of the SLL.

```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;

// Create a Node Data Type
struct Node
{
    int data;
    Node *next;
    // Initialization
    Node(int x)
    {
        data = x;
        next = NULL;
    }
};
            // This program only includes the Function
// k is the position of the node in the linked list
Node* DeleteKthNode(Node *head, int k)
{
    Node *temp = head, *prev = NULL, *fr = NULL;
    int cnt = 0;
    while (temp!=NULL)
    {
        cnt++;
        if (cnt == k)
        {
            break;
        }
        prev = temp; // previous element of the kth node
        temp = temp->next; // kth node
    }
    fr = temp->next; // front element of the kth node
    prev->next = fr; // set the prev next pointer to kth node front node
    delete temp; // delete the node
    return head;
    }
```

Example 2: Reverse a SLL and return the new head.

```
#include<bits/stdc++.h>
using namespace std;

       // This program only includes the Function

Node* ReverseList(Node *head)
{
    Node *p = NULL,*c = NULL;

    while(head != NULL)
    {
        c = head->next;
        head->next = p;
        p = head;
        head = c;
    }

    head = p; // new head of the list
    return head;

}
```

Example 3: Create a doubly linked list from an array of values.

```
#include<bits/stdc++.h>
using namespace std;

//Create a Node Data Type for DLL
struct Node
{
    int data;
    Node *next;
    Node *bak;

    Node (int x)  // First Constructor
    {
        data = x;
        next = NULL;
        bak = NULL;
    }

    Node (int x, Node *f, Node *b) // Second Constructor
    {
        data = x;
        next = f;
        bak = b;
    }
};
```

```
                // This program only included the Function
Node* CreateDLL(int arr[], int arrsize)
{
    Node *head = NULL, *temp = NULL, *prev = NULL;

    head = new Node(arr[0]); // set the head pointer
    prev = head;

    for (int i = 1; i<arrsize; i++)
    {
        temp = new Node(arr[i], nullptr, prev); // insert new node
        prev->next = temp;
        prev = temp;
    }
    return head;
}
```

Practice Exercise

1. Write a C++ program to find the position of an element from a Singly Linked List [Linear Search].
2. Write a C++ program to insert an element at k^{th} position in a singly linked list. [Consider possible edge cases]
3. Write a C++ program to insert a node at the beginning of a DLL. [Consider possible edge cases]
4. Write a C++ program to insert a node at the end of a DLL. [Consider possible edge cases]
5. Write a C++ program to delete the first node of a DLL. [Consider possible edge cases]
6. Write a C++ program to delete the last node of a DLL. [Consider possible edge cases]

Resources (Link)

Try to solve similar problems at an online Judge.

1. Search in a SLL
2. Reverse a SLL
3. Construct a DLL
4. Insert a node in DLL
5. Delete a node in DLL

Experiment No-04: Vector, Stack, and Queue in C++.

Objectives

- Introduce with vector in C++.
- Introduce with stack and its operations in C++.
- Introduce with queue and its operations in C++.

Example 1: Vector in C++. [Vector]

```
/**
vector: Member Functions

1) push_back(element) -----> push_back() is used for inserting an
    element at the end of the vector

2) pop_back()       -----> pop_back() is used to remove the last
    element from the vector. It reduces the size of the vector by one.

8) clear()          ----- > This method clears the whole vector,
    removes all the elements from the vector but do not delete the
    vector.
9) size()           ------> returns the size of the vector

**/

#include<bits/stdc++.h>
using namespace std;

int main()
{
    vector<int>vec1; //int type vector declaration
    vector<string>vec2; // string type vector declaration

    // Push_back operatin on vec1

    for(int i=0;i<5;i++){
        vec1.push_back(i);
    }
    vec1.push_back(100);
    vec1.push_back(10);
    vec1.push_back(23);
    vec1.push_back(9);

// Print the elements of the vector
for(int i=0;i<vec1.size();i++){
    cout<<vec1[i]<<"\t";
}
}
```

Example 2: Stack in C++. [Stack]

```
#include<bits/stdc++.h>
using namespace std;

// Stack Container in C++

int main() {

stack<int>mystack; // variable declaration

mystack.push(42); // push operation
mystack.push(11);
mystack.push(5);
mystack.push(71);
mystack.push(43);


while(!mystack.empty()){
   cout<<mystack.top()<<" ";
   mystack.pop(); // pop operation
}

}
```

Example 3: Queue in C++. [Queue]

```
#include<bits/stdc++.h>
using namespace std;

// Queue Container in C++

int main ()
{
    queue <int> q; // creates an empty queue of integer q

    q.push(2); // pushes 2 in the queue , now front = back = 2
    q.push(3); // pushes 3 in the queue , now front = 2 , and back = 3
    q.push(8);
    q.push(45);
    q.push(60);
    q.push(80);

    while(!q.empty()){

        cout<<q.front()<<" ";

        q.pop();
    }

}
```

Practice Exercise

1. Take 5 integer values into a stack. Find the summation of all the stack elements.
2. Take 6 integer values (0 to 5) into a stack and then find the factorial of each stack element. Store the outputs in another stack. Print the output in the following way:

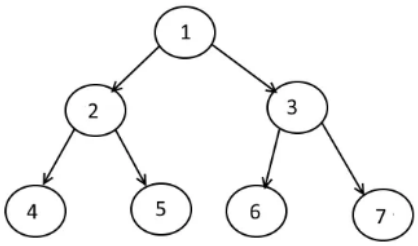
Factorial : 0 = 1
Factorial : 1 = 1
Factorial : 2 = 4
—-

Experiment No-05: Tree Representation and Traversal C++.

Objectives

- Construct a Tree.
- Find Inorder traversal using recursion.
- Learn Level Order Traversal.

Example 1: Tree representation in C++.




```
#include<bits/stdc++.h>
using namespace std;

struct Node
{
    int data;
    Node *left; // Left reference ptr to the node.
    Node *right; // Right reference ptr to the node.

    // Method to initialize the above values.
    Node(int val)
    {
        data = val;
        left = right = NULL;
    }
};

int main()
{

   Node* root = new Node(1);
   root -> left = new Node(2);
   root -> right = new Node(3);
   root -> left -> left = new Node(4);
   root -> left -> right = new Node(5);
   root -> right -> left = new Node(6);
   root -> right -> right = new Node(7);
}
```

Example 2: Inorder traversal using recursion.

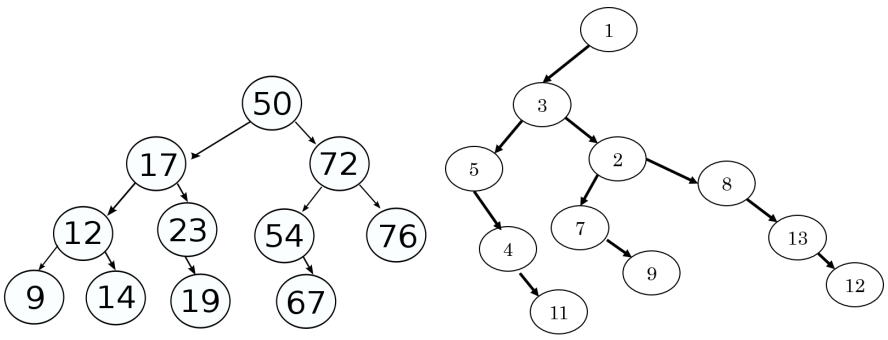
```
#include<bits/stdc++.h>
using namespace std;

struct Node
{
    int data;
    Node *left;
    Node *right;

    Node(int val)
    {
        data = val;
        left = NULL ;
        right = NULL;
    }
};

// Inorder Traversal Function

void InOrderTraversal(Node *temp)
{
    if (temp==NULL)
    {
        return;
    }

    InOrderTraversal(temp->left);
    cout<<temp->data<<" ";
    InOrderTraversal(temp->right);

}

int main()
{
   // Tree construction
   Node* root = new Node(1);
   root -> left = new Node(2);
   root -> right = new Node(3);
   root -> left -> left = new Node(4);
   root -> left -> right = new Node(5);
   root -> right -> left = new Node(6);
   root -> right -> right = new Node(7);


   court<<"Inorder Traversal:"<<endl;
   InOrderTraversal(root);

}
```

Example 2: Level-Order traversal in C++.

```
#include<bits/stdc++.h>
using namespace std;

// Level-Order Traversal Function
void LevelOrderTraversal(Node *root)
{
    if (root == NULL)
        cout<<"Tree is Empty."<<endl;

    queue<Node*> q;
    q.push(root);

    while(!q.empty()) {

        Node *temp = q.front();
        q.pop();

        if(temp->left != NULL)
            q.push(temp->left);
        if(temp->right != NULL)
            q.push(temp->right);

        cout<< temp->data<<" ";
    }

}
```

Practice Exercise

1. Write a C++ program to find the Inorder, Preorder, and Postorder traversals of the following trees.



(a) Tree 1 (b) Tree 2

Figure 1

Resources (Link)

Try to solve similar problems at an online Judge.

1. Preorder Traversal
2. Inorder Traversal
3. Postorder Traversal
4. Level Order Traversal

Experiment No-06: Important Problems on Binary Tree.

Objectives

- Find the height of a binary tree.
- Check whether a tree is balanced or not.
- Check whether a tree is BST or not.

Example 1: Find the height of a binary tree.

```
#include<bits/stdc++.h>
using namespace std;


// Function to find the tree height
int maxDepth(Node* root)
{
    if (root == NULL) return 0;

    int lh = 1+ maxDepth(root->left); // calculate height of left
        subtree
    int rh = 1+ maxDepth(root->right); // calculate height of right
        subtree

    return max(lh,rh); // return max between two numbers
}

int main()
{
   Node* root = new Node(1);
   root -> left = new Node(2);
   root -> right = new Node(3);
   root -> left -> left = new Node(4);
   root -> left -> right = new Node(5);
   root -> right -> left = new Node(6);
   root -> right -> right = new Node(7);
   root -> left -> left -> left = new Node(9);

   int h = maxDepth(root);

   cout<<"Height: "<<h<<endl;

}
```

Example 2: Check whether a tree is balanced or not.

```
// Height calculation function
int maxDepth(Node* root)
{
    if (root == NULL) return 0;

    int lh = 1+ maxDepth(root->left);

    if (lh == -1) return -1;

    int rh = 1+ maxDepth(root->right);

    if (rh == -1) return -1;

    if (abs(lh-rh)>1) // Check for imbalanced condition
        return -1;

    return max(lh,rh);
}


bool isbalanced(Node *root)
{
    // return 1 if true otherwise return 0
    return maxDepth(root)!=-1;
}

int main()
{

   Node* root = new Node(1);
   root -> left = new Node(2);
   root -> left -> left = new Node(4);

   int h = isbalanced(root);

   if(h==0)
   {
    cout<<"Tree in not balanced"<<endl;
   }
    else{
    cout<<"Tree is balanced"<<endl;
    }
}
```

Practice Exercise

1. Write a C++ program to find the height of the following tree (Figure 1).
2. Write a C++ program to check whether the following tree (Figure 1) is balanced.
3. Write a C++ program to check whether a given tree is BST.
4. Write a C++ program to determine whether a given tree is perfect. [**Hint:** height of left subtree and right subtree is equal]
5. Write a C++ program to find the sum of the left child of a given tree. [**Hint:** use level order traversal]

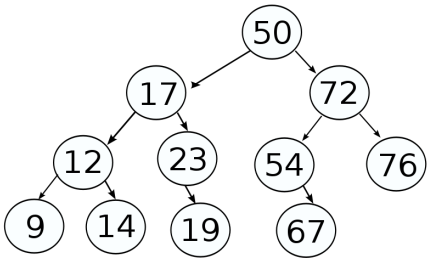


Figure 1

Resources (Link)

Try to solve similar problems at an online Judge.

1. Height of a Binary Tree
2. Balanced Tree

Experiment No-07: Graph Representation and Traversal.

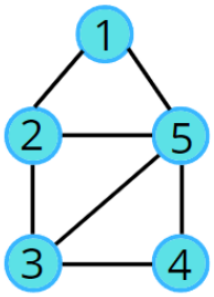
Objectives

- Represent a graph in C++.
- Traverse a graph using the breadth-first search technique.

Example 1: Graph Representation using Adjacency List.

```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;

/* Inputting Format

4 4      nodes  edges
0 1
0 2
0 3
1 2
*/

int main(){

vector<int>graph[5];    // initialize a vector of array
int nodes, edge, u, v;

cout<<"Enter Number of Nodes: ";
cin>>nodes;
cout<<"Enter Number of Edges: ";
cin>>edge;

for (int i = 0;i<edge;i++){
    cin>>u>>v;           // take input the edges connection
    graph[u].push_back(v);
    graph[v].push_back(u);
}

cout<<"Adjacency List of the Graph: "<<endl;

for (int j = 0; j<nodes; j++){
    cout<<j<<" --> ";
    for(auto it: graph[j]){
        cout<<it<<" ";
    }
    cout<<endl;
}

}
```

Example 2: Breadth First Search (BFS) Traversal.

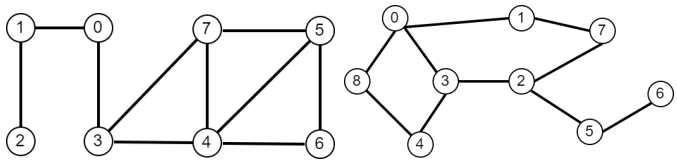


```
#include<bits/stdc++.h>
using namespace std;

vector<int> adj[100];
int visited[100]; // create an array with all zero values

// BFS function
vector<int>Bfs(int source) {
    vector<int>bfs;
    queue<int> q; // declare a empty queue
    visited[source] = 1;
    q.push(source); // push source node into queue

    while (!q.empty()) {
        int node = q.front(); // front element of the queue
        q.pop(); // pop the node
        bfs.push_back(node);
        for (auto it: adj[node]) {
            int nxt_node = it; // the neighbour node
            // if the neighbour has previously not been visited,
            if (visited[nxt_node]) continue;
            visited[nxt_node] = 1;
            q.push(nxt_node); // push into the queue
        }
    }
    return bfs;
}

int main() {

    int i, j, k;
    int n, e;
    vector<int>bfs;
    cout<< "No.of Nodes: ";
    cin >> n;
    cout<< "No.of Edges: ";
    cin >> e;
    cout<<"Enter the edge connections: "<<endl;
```

```
    // adjacency list
    for (i = 0; i < e; ++i) {
        int u, v;         // edge inputs
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    int source;
    cout<<"Enter the Source Node: "<<endl;
    cin >> source;
    // call the BFS method
    bfs = Bfs(source);
    // print the values
    for (auto it: bfs){
        cout<<it<<" ";
    }

}
```

Practice Exercise

1. Write a C++ program to Represent the following graphs using an adjacency matrix (Figure 1).
2. Write a C++ program to Represent the following graphs using an adjacency List (Figure 1).
3. Write a C++ program to find the traversal of the following graphs (Figure 1). [Choose a random node as a source]



(a) (b)

Figure 1

Resources (Link)

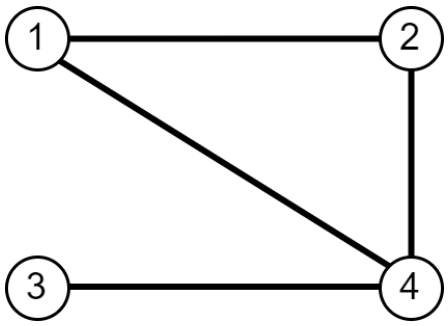
Try to solve similar problems at an online Judge.

1. Graph Representation
2. BFS Traversal

Experiment No-08: Shortest Path Finder using BFS Algorithm.

Objectives

- Find the shortest distances using BFS.
- Find the shortest path from a source to a destination node.



Example 1: Finding the shortest distance from the source to all the nodes.

```
#include<bits/stdc++.h>
using namespace std;

vector<int> adj[100];
int dis[100], visited[100];

// BFS function
void Bfs(int source) {
    queue<int> q; // declare a empty queue
    dis[source] = 0;
    visited[source] = 1;
    q.push(source); // push source node into queue
    while (!q.empty()) {
        int node = q.front(); // front element of the queue
        for (auto it: adj[node]) {
            int nxt_node = it;
            // already visited then skip
            if (visited[nxt_node]) continue;
            dis[nxt_node] = 1 + dis[node];
            visited[nxt_node] = 1;
            q.push(nxt_node); // push into the queue
        }
        // pop the node
        q.pop();
    }
}

int main() {
    int i, j, k;
    int n, m;
```

```
    cout<< "No.of Nodes: "<<endl;
    cin >> n;
    cout<< "No.of Edges: "<<endl;
    cin >> m;
    cout<<"Enter the edge connections: "<<endl;
    for (i = 0; i < m; ++i) {
        int u, v; // edge inputs
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    int source;
    cout<<"Enter the Source Node: "<<endl;
    cin >> source;
    // call the BFS method
    Bfs(source);
    for (i = 1; i <= n; ++i) {
        cout << "Distance " << source << " to " << i << " = " << dis[i]
            << endl;
    }
}
```

Example 2: Finding the shortest path from a source to the destination node.

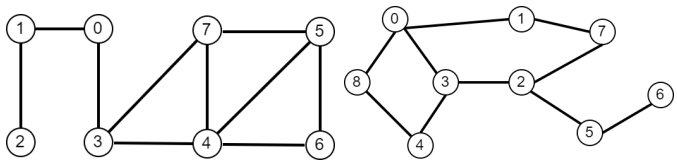
```
#include<bits/stdc++.h>
using namespace std;

vector<int> adj[100];
vector<int> path;
int parent[100], dis[100], visited[100];

// Function for finding the shortest path
void shortest_path(int d){
    if (d!=-1){
        int p = parent[d];
        path.push_back(d); // push the paths into a vector
        shortest_path(p); // recursively called
    }
}

// BFS function for finding the shortest distance
void Bfs(int source) {
    queue<int> q; // declare a empty queue
    dis[source] = 0;
    visited[source] = 1;
    parent[source] = -1;
    q.push(source); // push source node into queue
    while (!q.empty()) {
        int node = q.front(); // front element of the queue
        for (auto it: adj[node]) {
            int nxt_node = it;
```

```
            // already visited then skip
            if (visited[nxt_node]) continue;
            dis[nxt_node] = 1 + dis[node];
            visited[nxt_node] = 1;
            parent[nxt_node] = node;
            q.push(nxt_node); // push into the queue
        }
        // pop the node
        q.pop();
    }
}

int main() {
    int i, j, k;
    int n, m;
    cout<< "No.of Nodes: "<<endl;
    cin >> n;
    cout<< "No.of Edges: "<<endl;
    cin >> m;
    cout<<"Enter the edge connections: "<<endl;

    for (i = 0; i < m; ++i) {
        int u, v; // edge inputs
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    int source,dest;
    cout<<"Enter the Source Node: "<<endl;
    cin >> source;
    cout<<"Enter the Destination Node:"<<endl;
    cin>> dest;
    // call the BFS method
    Bfs(source);
    cout<<"Shortest Distance from "<<source<<" to "<<dest<<" =
        "<<dis[dest]<<endl;
    cout<<"Shortest Path is: ";
    shortest_path(dest); // call the shortest path function
    // Reverse the path vector
    reverse(path.begin(), path.end());
    // print the path
    for (auto it: path){
        cout<<it<<" ";
    }
}
```

Practice Exercise

Write C++ programs for the following graphs to -

1. Find the shortest distance from an arbitrary source to all the nodes.
2. Find the shortest distance and path from an arbitrary source to an arbitrary destination node.



(a) (b)

Figure 1