**American International University Bangladesh (AIUB)**

****

**Faculty of science & Technology**

**Department of Computer Science**

**LAB MANUAL DFS**  
CSC 2211 Algorithms

|  |
| --- |
| **TITLE** |

**STL (Standard Template Library) in C++**

**Vector**

Vectors are same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container. Vector elements are placed in contiguous storage so that they can be accessed and traversed using iterators.

#include<iostream>

#include<vector>

using namespace std;

void printVector(vector<int> vec){

cout<<"Size of vector: "<<vec.size()<<endl;

for(int i=0;i<vec.size();i++){

cout<<vec[i]<<endl;

}

}

int main(){

// initialising the vector

//vector<int> v = { 10, 20, 30, 40 };

vector<int> v;

printVector(v);

v.push\_back(10);

v.push\_back(20);

v.push\_back(30);

printVector(v);

return 0;

}

**Vector printing using iterator and insert value in any position**

#include<iostream>

#include<vector>

#include<iterator>

using namespace std;

int main(){

vector<int> vec={10,20,30,40,50};

vector<int>::iterator it;

//insert 500 in the front

vec.insert(vec.begin(),500);

//insert 100 in 2 index position

vec.insert(vec.begin()+2,100);

//printing vector using iterator

for(it=vec.begin();it!=vec.end();it++){

cout<<\*it<<" ";

}

return 0;

}

**Insert and delete from vector**

#include<iostream>

#include<vector>

#include<iterator>

using namespace std;

void printVector(vector<string> vec){

cout<<"Size of vector: "<<vec.size()<<endl;

for(int i=0;i<vec.size();i++){

cout<<vec[i]<<" ";

}

cout<<endl;

}

int main(){

vector<string> v={"Richard","Aaron","Florence"};

//insert beginning

v.insert(v.begin(),"Supta");

//insert position 2

v.insert(v.begin()+2,"Philip");

//insert end

v.insert(v.end(),"Mr. Hi");

printVector(v);

//delete from end

// v.erase(v.end());

//delete from begin

// v.erase(v.begin());

//delete elements from 3 to end

v.erase(v.begin()+3,v.end());

printVector(v);

//delete all elements

v.clear();

printVector(v);

return 0;

}

**Vector of objects(class or struct)**

#include<iostream>

#include<vector>

using namespace std;

class student{

private:

int id;

string name;

public:

student(int id, string name){

this->id=id;

this->name=name;

}

int getId(){

return id;

}

string getName(){

return name;

}

};

void printVector(vector<student> vec){

cout<<"Size of vector: "<<vec.size()<<endl;

for(int i=0;i<vec.size();i++){

cout<<vec[i].getId()<<" "<<vec[i].getName()<<endl;

}

}

int main(){

vector<student> v;

printVector(v);

student s1(100,"Richard");

student s2(200,"Philip");

v.push\_back(s1);

v.push\_back(s2);

printVector(v);

return 0;

}

**Stack**

Stacks are a type of container adaptors with LIFO(Last In First Out) type of working, where a new element is added at one end and (top) an element is removed from that end only**.**

#include <bits/stdc++.h>

using namespace std;

void printStack(stack <int> s)

{

while (!s.empty())

{

cout <<" "<< s.top();

s.pop();

}

cout << '\n';

}

int main ()

{

stack <int> s;

s.push(10);

s.push(30);

s.push(20);

s.push(5);

s.push(1);

cout << "The stack is : ";

printStack(s);

cout << "\ns.size() : " << s.size();

cout << "\ns.top() : " << s.top();

cout << "\ns.pop() : ";

s.pop();

printStack(s);

return 0;

}

**Queue**

Queues are a type of container adaptors which operate in a first in first out (FIFO) type of arrangement. Elements are inserted at the back (end) and are deleted from the front.

#include<iostream>

#include<queue>

using namespace std;

int main(){

queue<int> q;

q.push(10);

q.push(20);

q.push(30);

while(!q.empty()){

int x=q.front();

cout<<x<<" ";

q.pop();

}

return 0;

}

**Graph Algorithm in C++**

DFS Using Adjacency List Recursive

#include<bits/stdc++.h>

using namespace std;

vector<int> graph[5];

int visited[5];

void dfs(int source){

visited[source]=1;

for(int i=0;i<graph[source].size();i++){

int next = graph[source][i];

if(visited[next]==0){

dfs(next);

}

}

}

int main(){

int node,edge,u,v;

cin>>node>>edge;

for(int i=0;i<edge;i++){

cin>>u>>v;

graph[u].push\_back(v);

graph[v].push\_back(u);

}

dfs(0);

for(int i=0;i<node;i++){

if(visited[i]==1){

cout<<"Node "<<i <<" is visited"<<endl;

}

}

return 0;

}

/\*

5 4

0 1

1 2

1 3

3 4

\*/

DFS Using Adjacency List Iterative

#include<bits/stdc++.h>

using namespace std;

vector<int> graph[5];

int visited[5];

void dfs(int source){

stack<int> st;

st.push(source);

visited[source]=1;

while(!st.empty()){

int v=st.top();

st.pop();

for(int i=0;i<graph[v].size();i++){

int next = graph[v][i];

if(visited[next]==0){

st.push(next);

visited[next]=1;

}

}

}

}

int main(){

int node,edge,u,v;

cin>>node>>edge;

for(int i=0;i<edge;i++){

cin>>u>>v;

graph[u].push\_back(v);

graph[v].push\_back(u);

}

dfs(0);

for(int i=0;i<node;i++){

if(visited[i]==1){

cout<<"Node "<<i <<" is visited"<<endl;

}

}

return 0;

}

/\*

5 4

0 1

1 2

1 3

3 4

\*/

DFS Using Adjacency Matrix Recursive

#include<bits/stdc++.h>

using namespace std;

int graph[5][5];

int visited[5];

void dfs(int source){

visited[source]=1;

for(int i=0;i<5;i++){

if(graph[source][i]==1){

if(visited[i]==0){

dfs(i);

}

}

}

}

int main(){

int node,edge,u,v;

cin>>node>>edge;

for(int i=0;i<edge;i++){

cin>>u>>v;

graph[u][v]=1;

graph[v][u]=1;

}

dfs(0);

for(int i=0;i<node;i++){

if(visited[i]==1){

cout<<"Node "<<i <<" is visited"<<endl;

}

}

return 0;

}

/\*

5 4

0 1

1 2

1 3

3 4

\*/

Graph Using Adjacency Matrix Iterative

#include<bits/stdc++.h>

using namespace std;

int graph[5][5];

int visited[5];

void dfs(int source){

stack<int> st;

st.push(source);

visited[source]=1;

while(!st.empty()){

int v=st.top();

st.pop();

for(int i=0;i<5;i++){

if(graph[v][i]==1){

if(visited[i]==0){

st.push(i);

visited[i]=1;

}

}

}

}

}

int main(){

int node,edge,u,v;

cin>>node>>edge;

for(int i=0;i<edge;i++){

cin>>u>>v;

graph[u][v]=1;

graph[v][u]=1;

}

dfs(0);

for(int i=0;i<node;i++){

if(visited[i]==1){

cout<<"Node "<<i <<" is visited"<<endl;

}

}

return 0;

}

/\*

5 4

0 1

1 2

1 3

3 4

\*/