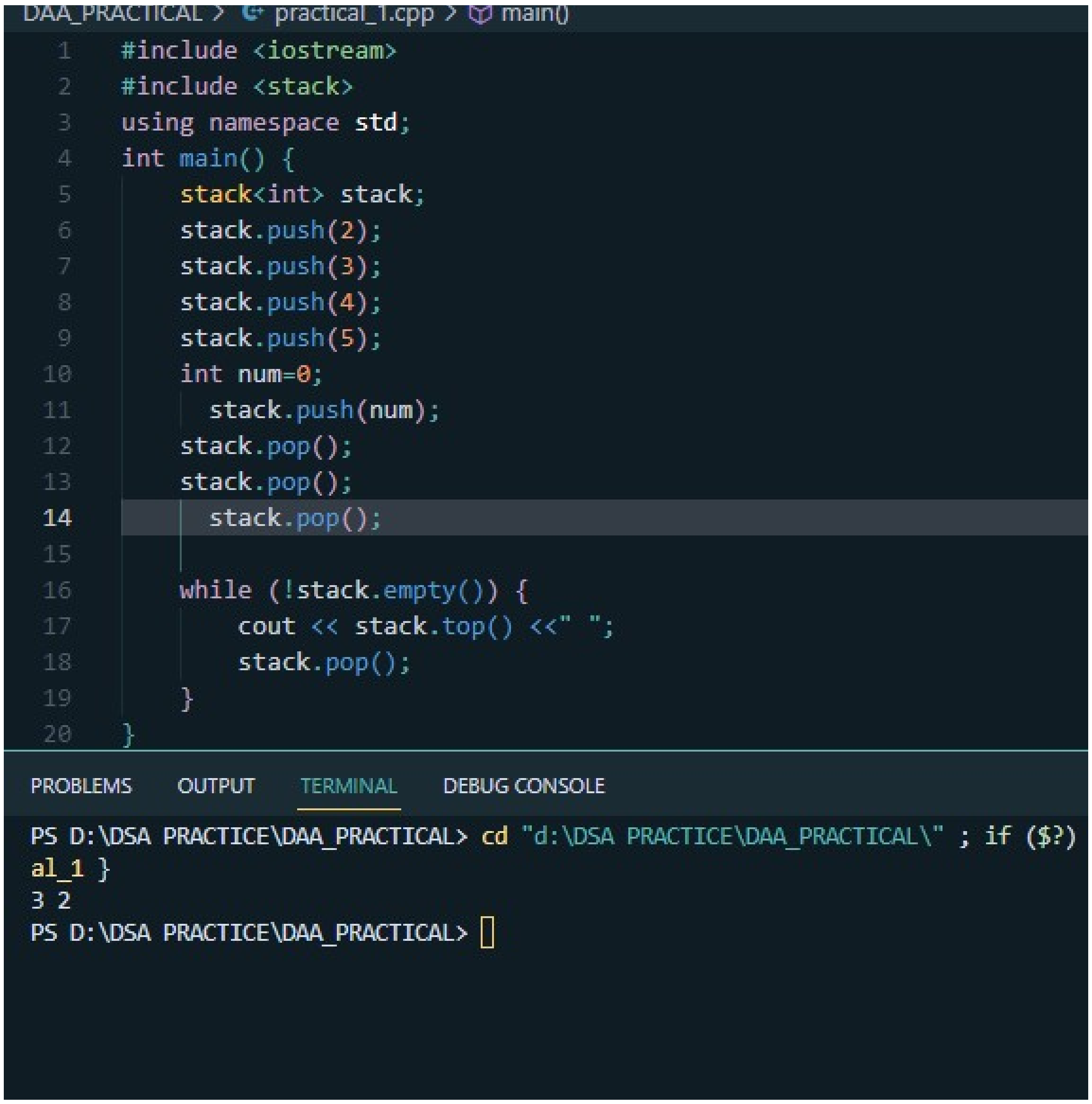
Design and Analysis of Algorithms

NAME : SHUBHAM SHIVRAJ SURYAWANSHI

ASSIGNMENT = 1

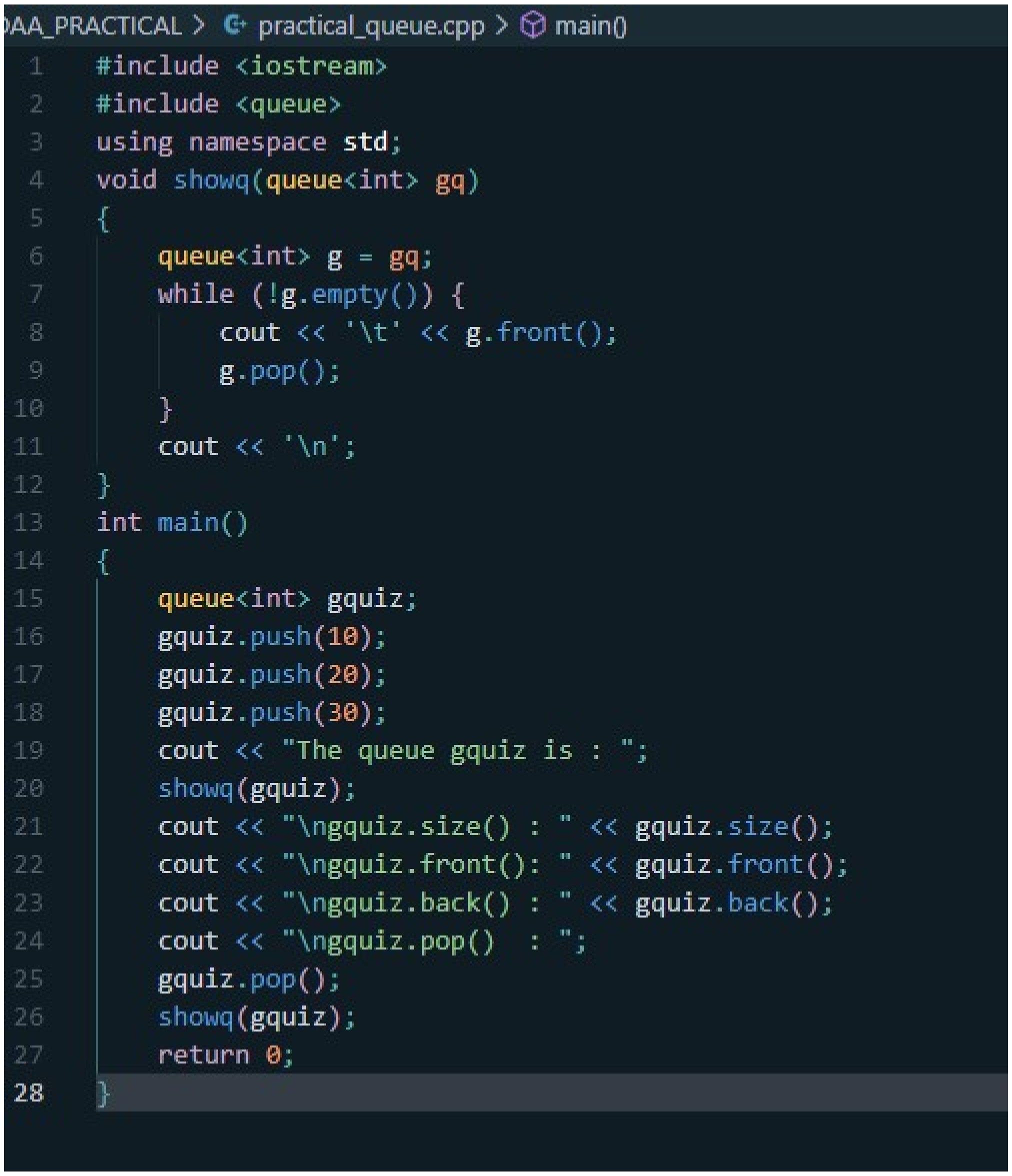
1) STACK



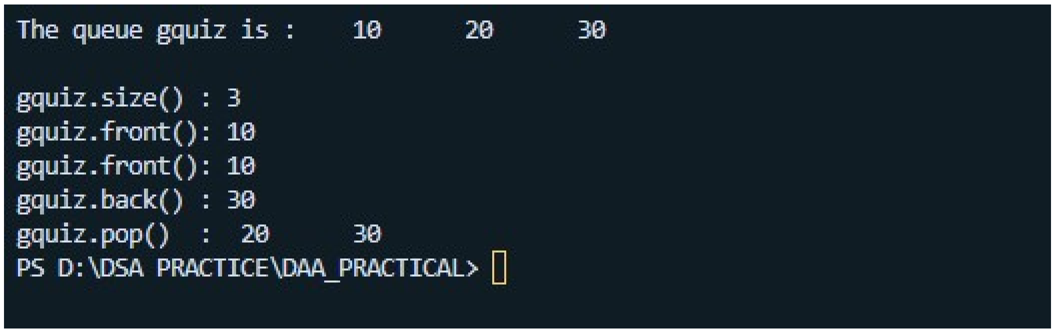
```cpp
#include <iostream>
#include <stack>
using namespace std;
int main() {
    stack<int> stack;
    stack.push(2);
    stack.push(3);
    stack.push(4);
    stack.push(5);
    int num=0;
      stack.push(num);
    stack.pop();
    stack.pop();
      stack.pop();

    while (!stack.empty()) {
        cout << stack.top() <<" ";
        stack.pop();
    }
}
```

```
PS D:\DSA PRACTICE\DAA_PRACTICAL> cd "d:\DSA PRACTICE\DAA_PRACTICAL\" ; if ($?)
al_1 }
3 2
PS D:\DSA PRACTICE\DAA_PRACTICAL>
```

2)Queue

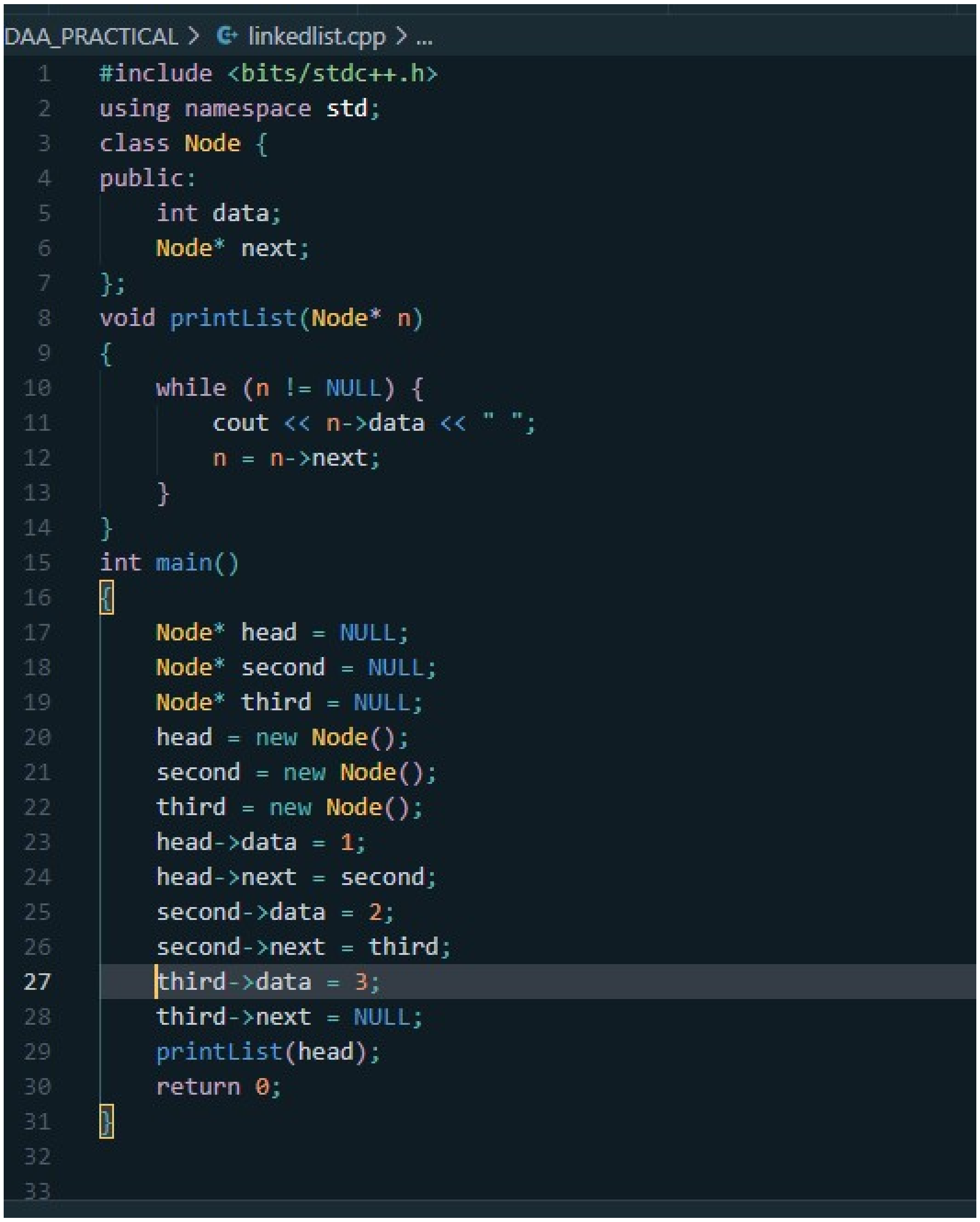

DAA_PRACTICAL > practical_queue.cpp > main()

```cpp
#include <iostream>
#include <queue>
using namespace std;
void showq(queue<int> gq)
{
    queue<int> g = gq;
    while (!g.empty()) {
        cout << '\t' << g.front();
        g.pop();
    }
    cout << '\n';
}
int main()
{
    queue<int> gquiz;
    gquiz.push(10);
    gquiz.push(20);
    gquiz.push(30);
    cout << "The queue gquiz is : ";
    showq(gquiz);
    cout << "\ngquiz.size() : " << gquiz.size();
    cout << "\ngquiz.front(): " << gquiz.front();
    cout << "\ngquiz.back() : " << gquiz.back();
    cout << "\ngquiz.pop()  : ";
    gquiz.pop();
    showq(gquiz);
    return 0;
}
```



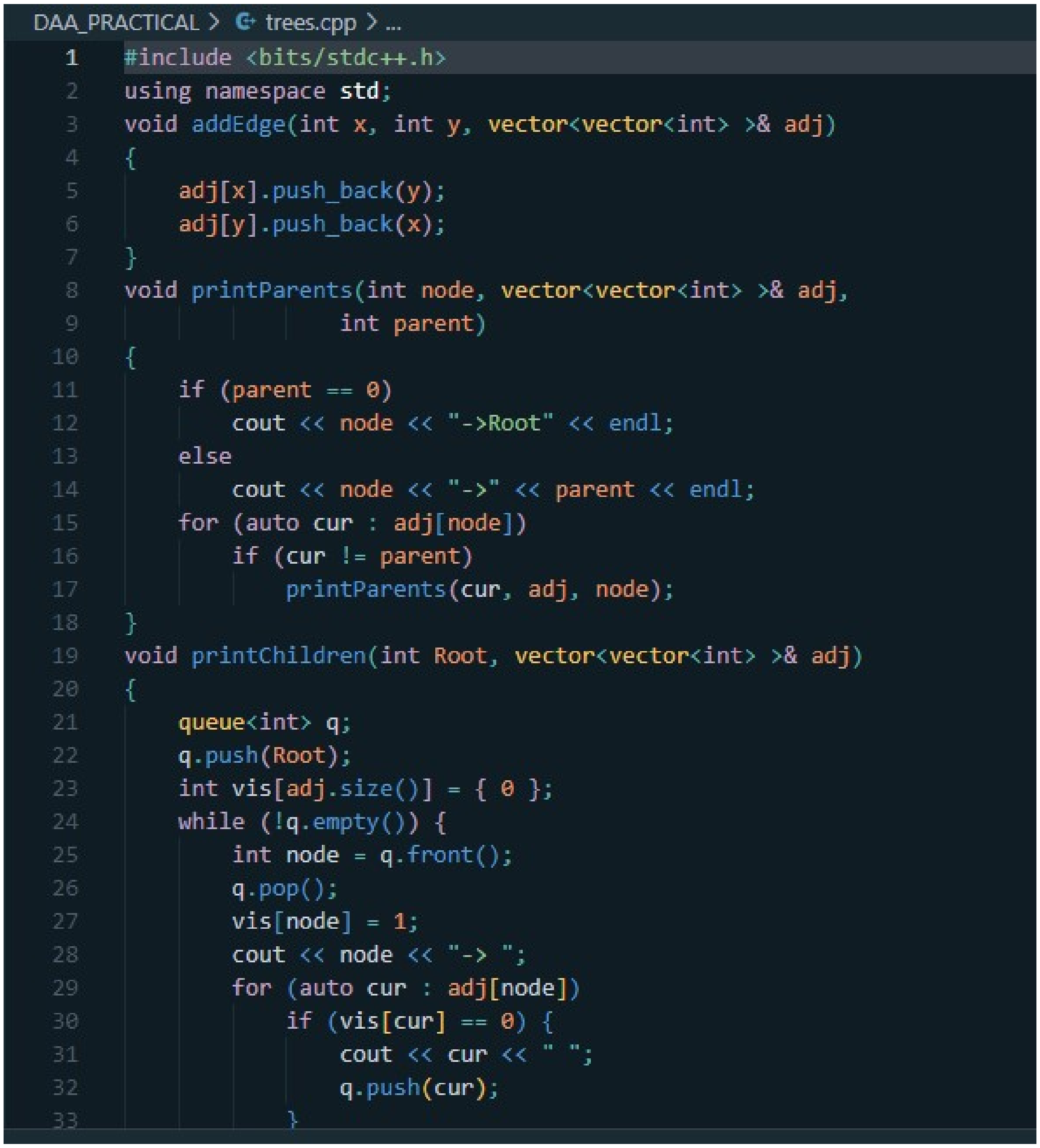
```
The queue gquiz is :    10      20      30

gquiz.size() : 3
gquiz.front(): 10
gquiz.front(): 10
gquiz.back() : 30
gquiz.pop()  :  20      30
PS D:\DSA PRACTICE\DAA_PRACTICAL> 
```

3) Linked list

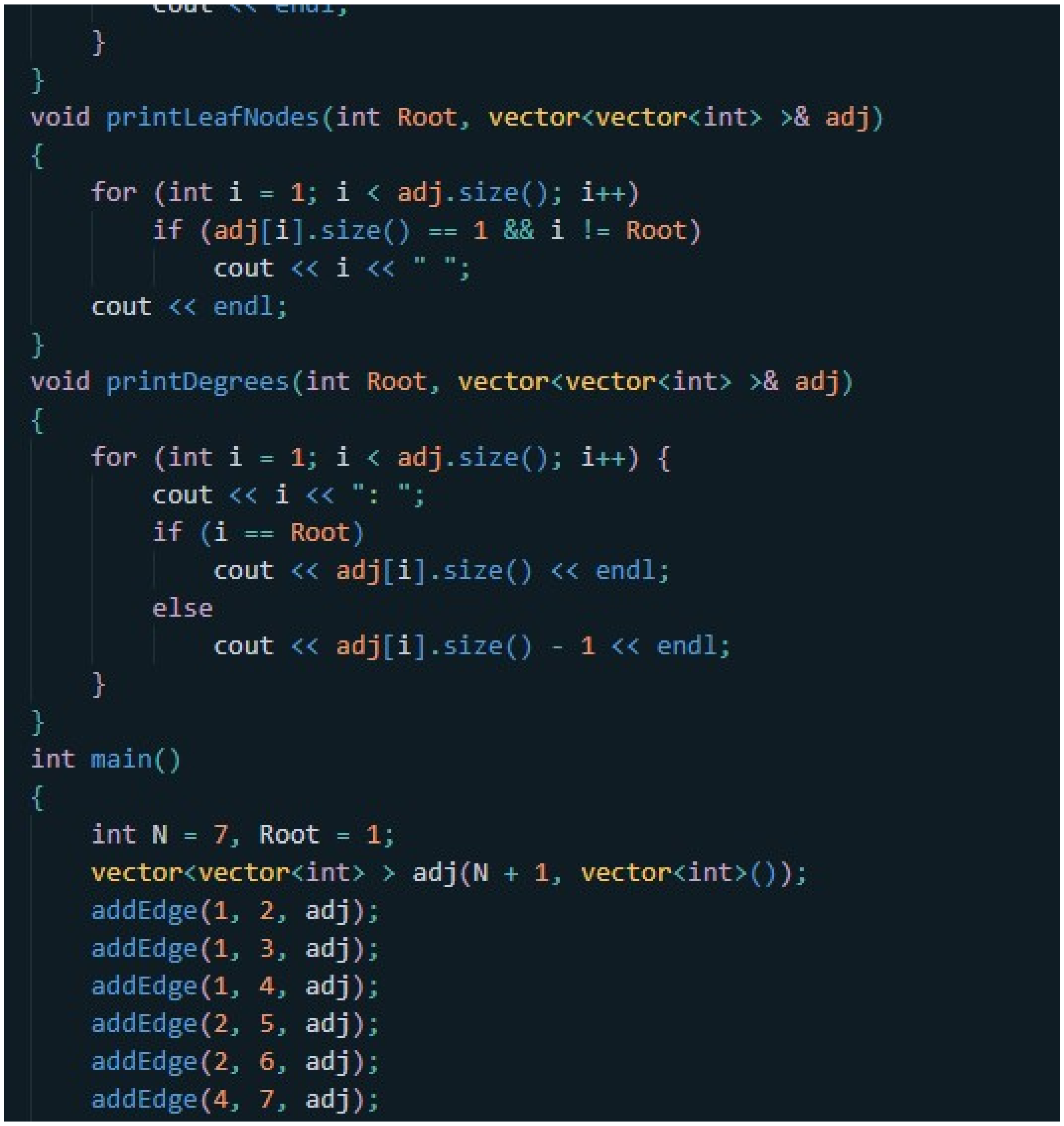

DAA_PRACTICAL > linkedlist.cpp > ...

```cpp
#include <bits/stdc++.h>
using namespace std;
class Node {
public:
    int data;
    Node* next;
};
void printList(Node* n)
{
    while (n != NULL) {
        cout << n->data << " ";
        n = n->next;
    }
}
int main()
{
    Node* head = NULL;
    Node* second = NULL;
    Node* third = NULL;
    head = new Node();
    second = new Node();
    third = new Node();
    head->data = 1;
    head->next = second;
    second->data = 2;
    second->next = third;
    third->data = 3;
    third->next = NULL;
    printList(head);
    return 0;
}
```

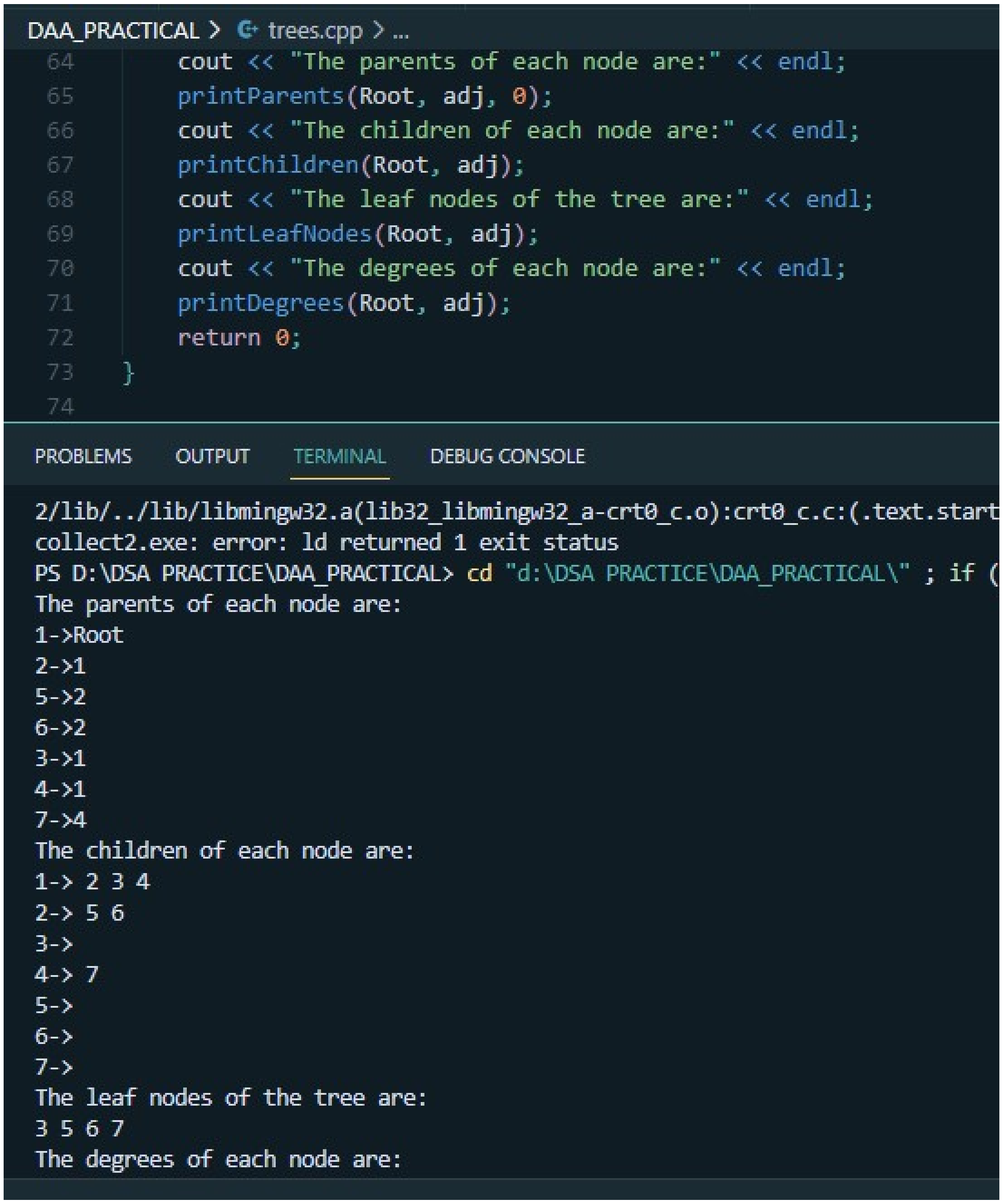
4)Trees


DAA_PRACTICAL > trees.cpp > ...

```cpp
#include <bits/stdc++.h>
using namespace std;
void addEdge(int x, int y, vector<vector<int> >& adj)
{
    adj[x].push_back(y);
    adj[y].push_back(x);
}
void printParents(int node, vector<vector<int> >& adj,
                  int parent)
{
    if (parent == 0)
        cout << node << "->Root" << endl;
    else
        cout << node << "->" << parent << endl;
    for (auto cur : adj[node])
        if (cur != parent)
            printParents(cur, adj, node);
}
void printChildren(int Root, vector<vector<int> >& adj)
{
    queue<int> q;
    q.push(Root);
    int vis[adj.size()] = { 0 };
    while (!q.empty()) {
        int node = q.front();
        q.pop();
        vis[node] = 1;
        cout << node << "-> ";
        for (auto cur : adj[node])
            if (vis[cur] == 0) {
                cout << cur << " ";
                q.push(cur);
            }
```



```cpp
        cout << endl;
    }
}
void printLeafNodes(int Root, vector<vector<int> >& adj)
{
    for (int i = 1; i < adj.size(); i++)
        if (adj[i].size() == 1 && i != Root)
            cout << i << " ";
    cout << endl;
}
void printDegrees(int Root, vector<vector<int> >& adj)
{
    for (int i = 1; i < adj.size(); i++) {
        cout << i << ": ";
        if (i == Root)
            cout << adj[i].size() << endl;
        else
            cout << adj[i].size() - 1 << endl;
    }
}
int main()
{
    int N = 7, Root = 1;
    vector<vector<int> > adj(N + 1, vector<int>());
    addEdge(1, 2, adj);
    addEdge(1, 3, adj);
    addEdge(1, 4, adj);
    addEdge(2, 5, adj);
    addEdge(2, 6, adj);
    addEdge(4, 7, adj);
```



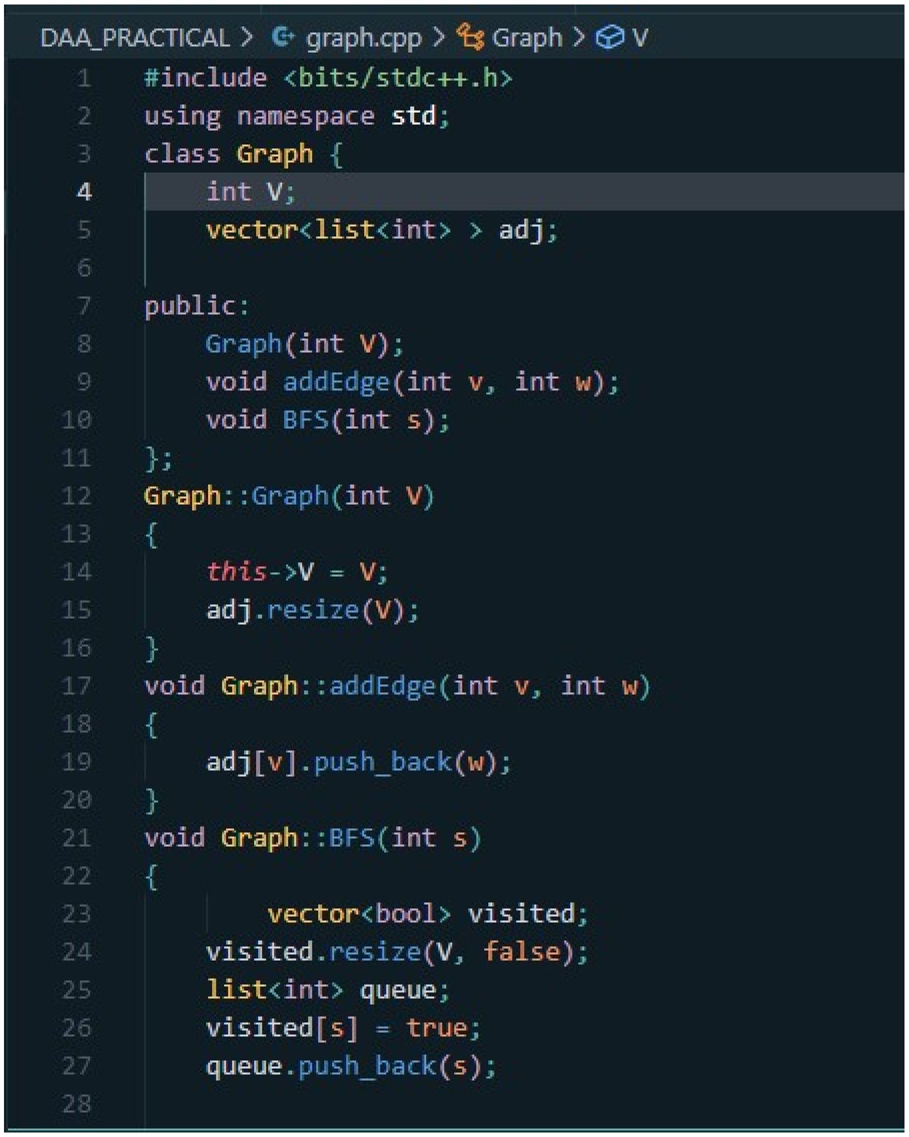
DAA_PRACTICAL > trees.cpp > ...

```cpp
        cout << "The parents of each node are:" << endl;
        printParents(Root, adj, 0);
        cout << "The children of each node are:" << endl;
        printChildren(Root, adj);
        cout << "The leaf nodes of the tree are:" << endl;
        printLeafNodes(Root, adj);
        cout << "The degrees of each node are:" << endl;
        printDegrees(Root, adj);
        return 0;
    }

```

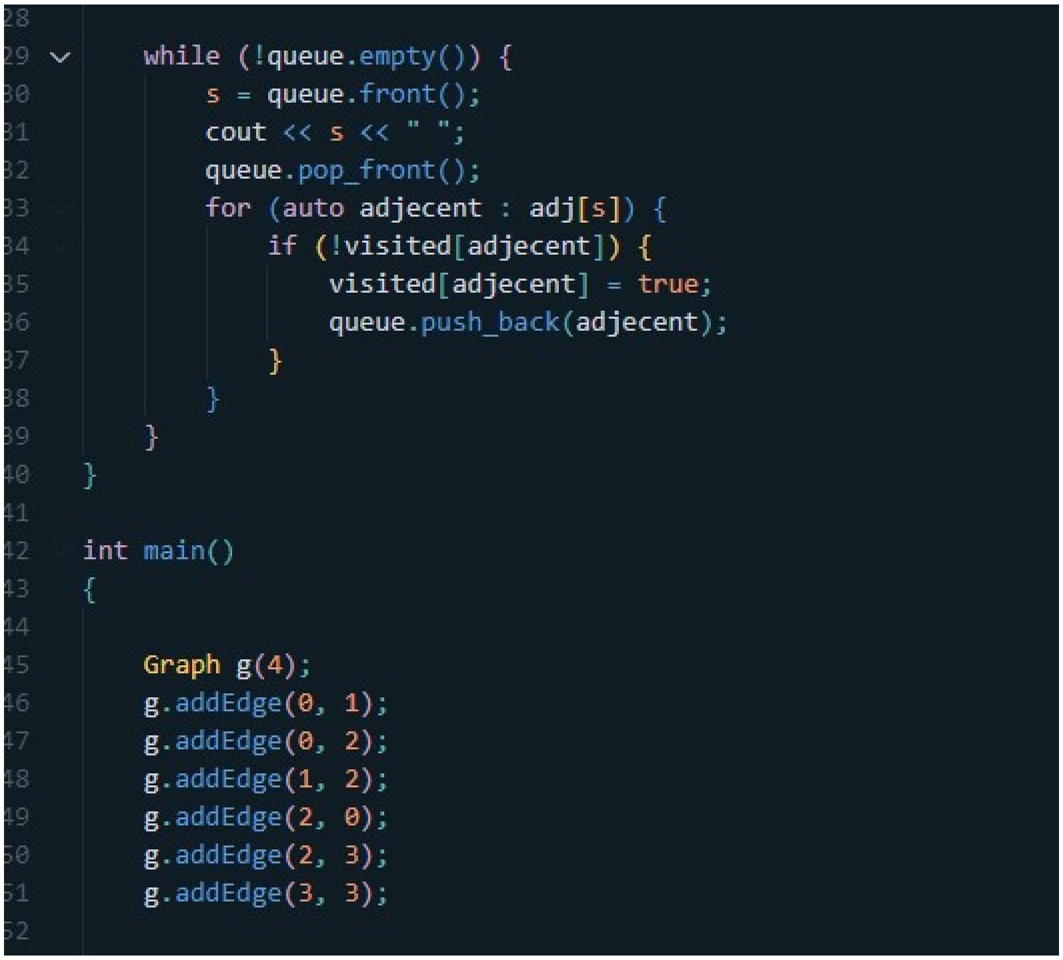
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

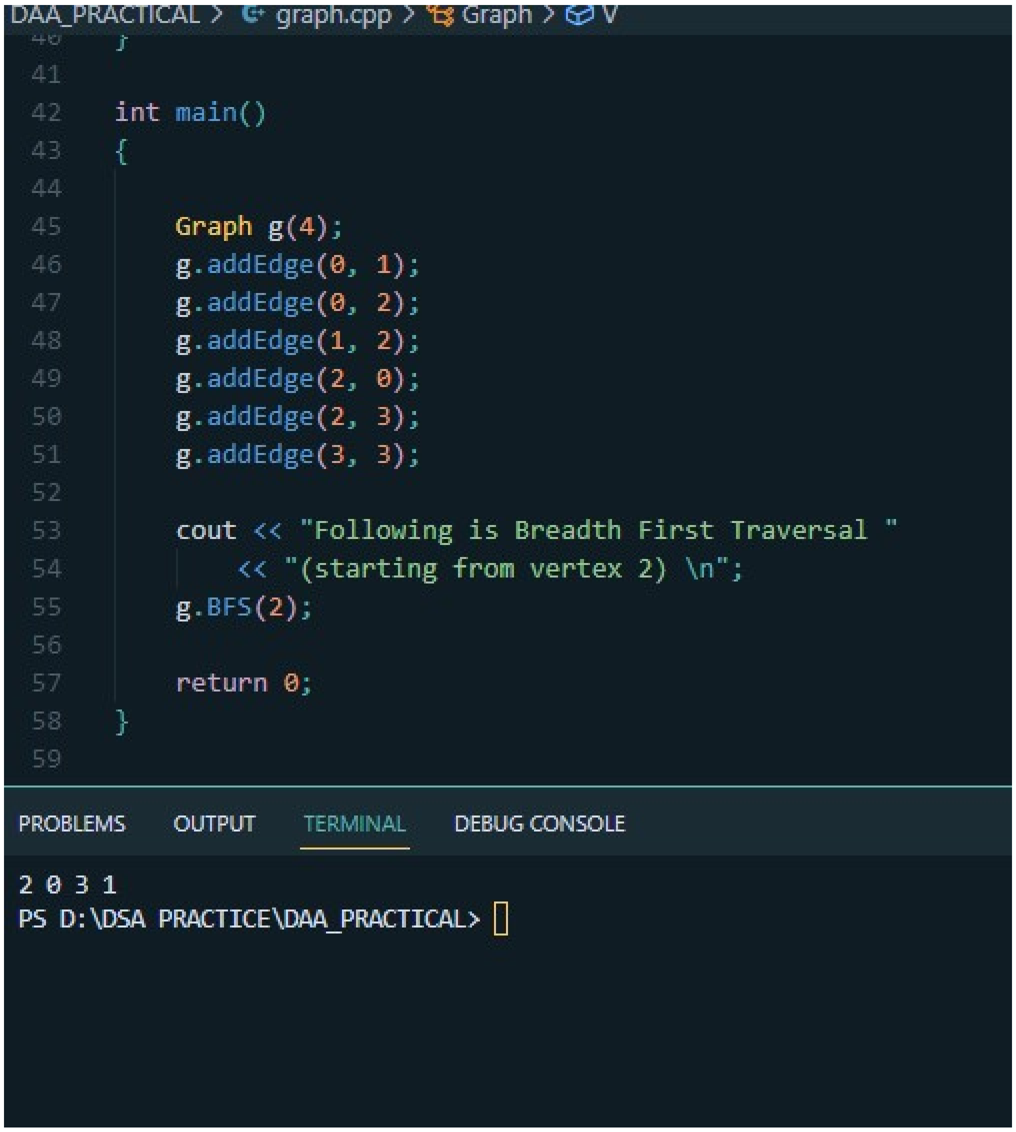
```
2/lib/../lib/libmingw32.a(lib32_libmingw32_a-crt0_c.o):crt0_c.c:(.text.start
collect2.exe: error: ld returned 1 exit status
PS D:\DSA PRACTICE\DAA_PRACTICAL> cd "d:\DSA PRACTICE\DAA_PRACTICAL\" ; if (
The parents of each node are:
1->Root
2->1
5->2
6->2
3->1
4->1
7->4
The children of each node are:
1-> 2 3 4
2-> 5 6
3->
4-> 7
5->
6->
7->
The leaf nodes of the tree are:
3 5 6 7
The degrees of each node are:
```

5) Graph


DAA_PRACTICAL > graph.cpp > Graph > V

```cpp
#include <bits/stdc++.h>
using namespace std;
class Graph {
    int V;
    vector<list<int> > adj;

public:
    Graph(int V);
    void addEdge(int v, int w);
    void BFS(int s);
};
Graph::Graph(int V)
{
    this->V = V;
    adj.resize(V);
}
void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w);
}
void Graph::BFS(int s)
{
        vector<bool> visited;
    visited.resize(V, false);
    list<int> queue;
    visited[s] = true;
    queue.push_back(s);

```


```
    while (!queue.empty()) {
        s = queue.front();
        cout << s << " ";
        queue.pop_front();
        for (auto adjecent : adj[s]) {
            if (!visited[adjecent]) {
                visited[adjecent] = true;
                queue.push_back(adjecent);
            }
        }
    }
}

int main()
{

    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

```



```cpp
}

int main()
{

    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    cout << "Following is Breadth First Traversal "
        << "(starting from vertex 2) \n";
    g.BFS(2);

    return 0;
}
```

```
2 0 3 1
PS D:\DSA PRACTICE\DAA_PRACTICAL>
```