Reverse directed graph in C++

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

class ReverseDirectedGraph {
public:
   static vector<vector<int>>
reverseDirectedGraph(const vector<vector<int>>& adj,
int V) {
       vector<vector<int>> reversedAdj(V + 1);

       for (int i = 0; i <= V; ++i) {
          for (int j : adj[i]) {
             reversedAdj[j].push_back(i);
          }
       }

       return reversedAdj;
   }

   static void printGraph(const vector<vector<int>>&
graph, int V) {
       for (int i = 1; i <= V; ++i) {
          for (int j : graph[i]) {
             cout << i << " -> " << j << endl;
          }
       }
   }
};

int main() {
   int V = 5;
   vector<vector<int>> adj(V + 1);

   adj[1].push_back(3);
   adj[1].push_back(2);
   adj[3].push_back(4);
   adj[4].push_back(5);

   vector<vector<int>> reversedAdj =
ReverseDirectedGraph::reverseDirectedGraph(adj, V);

   cout << "Reversed Graph:" << endl;
   ReverseDirectedGraph::printGraph(reversedAdj, V);

   return 0;
}
```

Input:

- **Number of nodes** (`V`) = 5
- **Edges** of the directed graph (adjacency list):
    - 1 → 3
    - 1 → 2
    - 3 → 4
    - 4 → 5

Step 1: Initialize Adjacency List

The adjacency list for the original graph (`adj`) is built as:

```
adj[1] = [3, 2]    // Node 1 has edges to 3 and 2
adj[2] = []        // Node 2 has no outgoing edges
adj[3] = [4]       // Node 3 has an edge to 4
adj[4] = [5]       // Node 4 has an edge to 5
adj[5] = []        // Node 5 has no outgoing edges
```

Step 2: Call `reverseDirectedGraph()` Function

Now, the function `reverseDirectedGraph()` will reverse the edges of the graph. We will iterate over the adjacency list and for each edge from `u → v`, we will add an edge `v → u` in the reversed graph.

Iterating through the adjacency list:

- **i = 1** (For node 1):
    - For edge `1 → 3`, reverse it to `3 → 1`
    - For edge `1 → 2`, reverse it to `2 → 1`
    - So, `reversedAdj[3]` becomes `[1]` and `reversedAdj[2]` becomes `[1]`.
- **i = 2** (For node 2):
    - Node 2 has no outgoing edges, so no change.
- **i = 3** (For node 3):
    - For edge `3 → 4`, reverse it to `4 → 3`
    - So, `reversedAdj[4]` becomes `[3]`.
- **i = 4** (For node 4):
    - For edge `4 → 5`, reverse it to `5`

→ 4

  - So, `reversedAdj[5]` becomes `[4]`.

- **i = 5** (For node 5):
  - Node 5 has no outgoing edges, so no change.

Reversed Graph:

After the reversal of the edges, the reversed adjacency list will be:

```
reversedAdj[1] = []       // No edges
coming into 1
reversedAdj[2] = [1]      // Node 2
has an edge coming from 1
reversedAdj[3] = [1]      // Node 3
has an edge coming from 1
reversedAdj[4] = [3]      // Node 4
has an edge coming from 3
reversedAdj[5] = [4]      // Node 5
has an edge coming from 4
```

Step 3: Print Reversed Graph Using `printGraph()` Function

Now, the `printGraph()` function will print the reversed adjacency list:

1. **For node 1**:
   - `reversedAdj[1] = []`, so no output for node 1.
2. **For node 2**:
   - `reversedAdj[2] = [1]`, so it will print `2 -> 1`.
3. **For node 3**:
   - `reversedAdj[3] = [1]`, so it will print `3 -> 1`.
4. **For node 4**:
   - `reversedAdj[4] = [3]`, so it will print `4 -> 3`.
5. **For node 5**:
   - `reversedAdj[5] = [4]`, so it will print `5 -> 4`.

Final Output:

The output of the program will be:

```
Reversed Graph:
2 -> 1
3 -> 1
4 -> 3
5 -> 4
```

Summary of the Dry Run:

1. **Original graph** has edges:
   - 1 → 3, 1 → 2, 3 → 4, 4 → 5
2. **Reversed graph** has edges:
   - 2 → 1, 3 → 1, 4 → 3, 5 → 4

Output:-

```
Reversed Graph:
2 -> 1
3 -> 1
4 -> 3
5 -> 4
```