BFS Implementation in C++

```
#include <iostream>
#include <vector>
#include <queue>
#include <stack>
#include <climits>
#include <algorithm>

using namespace std;

vector<int> bfsShortestPath(const vector<vector<int>>& graph, int src, int target)
{
    int n = graph.size();
    vector<int> distance(n, INT_MAX); // Distance from src to each node
    vector<int> predecessor(n, -1);   // To track the path
    queue<int> q;

    // Initialize BFS
    distance[src] = 0;
    q.push(src);

    // Perform BFS
    while (!q.empty()) {
        int current = q.front();
        q.pop();

        // Explore neighbors
        for (int neighbor : graph[current]) {
            if (distance[neighbor] == INT_MAX) { // If neighbor not visited
                distance[neighbor] = distance[current] + 1;
                predecessor[neighbor] = current; // Track the predecessor
                q.push(neighbor);

                // Stop early if we reached the target
```

```
            if (neighbor == target) {
                break;
            }
        }
    }

    // Check if target is reachable
    if (distance[target] == INT_MAX) {
        return {}; // No path found
    }

    // Reconstruct the shortest path from src to target
    vector<int> path;
    for (int at = target; at != -1; at = predecessor[at]) {
        path.push_back(at);
    }
    reverse(path.begin(), path.end()); // Reverse to get path from src to target

    return path;
}

int main() {
    // Example graph as an adjacency list
    vector<vector<int>> graph = {
        {1, 2},    // Neighbors of node 0
        {0, 3, 4}, // Neighbors of node 1
        {0, 4},    // Neighbors of node 2
        {1, 5},    // Neighbors of node 3
        {1, 2, 5}, // Neighbors of node 4
        {3, 4}     // Neighbors of node 5
    };

    int src = 0, target = 5;
    vector<int> shortestPath = bfsShortestPath(graph, src, target);

    if (!shortestPath.empty()) {
```

```
        cout << "Shortest path from " << src << " to " << target << ": ";
        for (int node : shortestPath) {
            cout << node << " ";
        }
        cout << endl;
    } else {
        cout << "No path found from " << src << " to " << target << endl;
    }

    return 0;
}
```

Explanation of the Code

1. **Initialization**:
    - distance[src] = 0 to mark the start of the BFS.
    - The predecessor array is initialized to -1 for each node, meaning no predecessor at the start.
    - The queue is initialized with src.
2. **BFS Execution**:
    - For each node current, all of its neighbors are checked.
    - If a neighbor has not been visited (i.e., distance[neighbor] == INT_MAX), the algorithm:
        - Sets distance[neighbor] = distance[current] + 1.
        - Sets predecessor[neighbor] = current to keep track of the path.
        - Pushes neighbor to the queue.
    - This continues until the queue is empty or the target node is reached.
3. **Path Reconstruction**:
    - If distance[target] == INT_MAX, it means there's no path from src to target.
    - Otherwise, backtrack from target to src using the predecessor array and construct the path.
    - Finally, the path is reversed to display it from src to target.

Example Output

If we run the code on a graph with nodes 0 to 5, and with src = 0 and target = 5, it might output:

Shortest path from 0 to 5: 0 1 3 5