C++ implementation of Prim's Algorithm using a **Min Heap (priority queue)** and **adjacency list**, which works for **weighted undirected graphs**:

Prim's Algorithm in C++

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

typedef pair<int, int> pii; // (weight, vertex)

void primMST(int V, vector<vector<pii>>& adj) {
    vector<int> key(V, INT_MAX);       // Store minimum weight to
reach each vertex
    vector<bool> inMST(V, false);      // To track visited vertices
(included in MST)
    priority_queue<pii, vector<pii>, greater<pii>> pq;  // Min Heap

    key[0] = 0;           // Start from vertex 0
    pq.push({0, 0});      // (weight, vertex)
    int totalWeight = 0;

    while (!pq.empty()) {
        int u = pq.top().second;
        int wt = pq.top().first;
        pq.pop();

        if (inMST[u]) continue;  // Skip if already added to MST

        inMST[u] = true;
        totalWeight += wt;
```

```
        for (auto& [v, weight] : adj[u]) {
            if (!inMST[v] && weight < key[v]) {
                key[v] = weight;
                pq.push({weight, v});
            }
        }
    }

    cout << "Total weight of MST: " << totalWeight << endl;
}

int main() {
    int V = 4; // Number of vertices

    // Adjacency list: adj[u] contains pairs {v, weight}
    vector<vector<pii>> adj(V);

    // Example graph:
    // 0--1 (1), 0--3 (3), 1--2 (4), 2--3 (2)
    adj[0].push_back({1, 1});
    adj[1].push_back({0, 1});

    adj[0].push_back({3, 3});
    adj[3].push_back({0, 3});

    adj[1].push_back({2, 4});
    adj[2].push_back({1, 4});

    adj[2].push_back({3, 2});
    adj[3].push_back({2, 2});

    primMST(V, adj);
```

```
    return 0;
}
```

Output:

```
Total weight of MST: 6
```

Code Concepts:

- Uses priority queue for selecting the minimum weight edge.
- Uses adjacency list for efficient neighbor lookup.
- Tracks visited vertices with `inMST[]`.
- Maintains a `key[]` array for the current best weights.