Annotated Code: Routing Table Simulator in C++

# // 🧩 Includes & Setup

#include <iostream>  
#include <vector>  
#include <sstream>  
#include <bitset>  
#include <algorithm>  
#include <iomanip>  
#include <string>  
#include <arpa/inet.h>  
  
using namespace std;

# // 📦 RouteEntry Structure

struct RouteEntry {  
 string destNetwork;  
 string subnetMask;  
 string nextHop;  
 string interfaceName;  
 int metric;  
};

# // 🔁 Convert IP to 32-bit binary string

string ipToBinary(const string &ip) {  
 struct in\_addr ip\_addr;  
 inet\_pton(AF\_INET, ip.c\_str(), &ip\_addr);  
 bitset<32> binary(ntohl(ip\_addr.s\_addr));  
 return binary.to\_string();  
}

# // 🧠 Longest Prefix Match Algorithm

RouteEntry\* findBestRoute(const vector<RouteEntry> &table, const string &destIP) {  
 string destBin = ipToBinary(destIP);  
 int maxMatch = -1;  
 RouteEntry\* bestMatch = nullptr;  
  
 for (const auto &entry : table) {  
 string netBin = ipToBinary(entry.destNetwork);  
 string maskBin = ipToBinary(entry.subnetMask);  
 string maskedDest = destBin;  
 string maskedNet = netBin;  
  
 for (int i = 0; i < 32; ++i) {  
 maskedDest[i] = (maskBin[i] == '1') ? destBin[i] : '0';  
 maskedNet[i] = (maskBin[i] == '1') ? netBin[i] : '0';  
 }  
  
 if (maskedDest == maskedNet) {  
 int matchLen = count(maskBin.begin(), maskBin.end(), '1');  
 if (matchLen > maxMatch) {  
 maxMatch = matchLen;  
 bestMatch = const\_cast<RouteEntry\*>(&entry);  
 }  
 }  
 }  
 return bestMatch;  
}

# // 🚀 Main Function (Input & Execution)

int main() {  
 vector<RouteEntry> routingTable = {  
 {"89.2.225.0", "255.255.255.0", "192.168.0.209", "eth0", 7},  
 {"250.237.99.0", "255.255.255.0", "192.168.77.102", "eth3", 6},  
 {"105.113.65.0", "255.255.255.0", "192.168.97.29", "eth1", 14},  
 {"16.237.196.0", "255.255.255.0", "192.168.110.65", "eth1", 15},  
 {"8.81.31.0", "255.255.255.0", "192.168.163.206", "eth0", 2}  
 };  
  
 string ip;  
 cout << "Enter destination IP: ";  
 cin >> ip;  
  
 RouteEntry\* match = findBestRoute(routingTable, ip);  
  
 if (match) {  
 cout << "Matched Route:" << endl;  
 cout << "Network: " << match->destNetwork << "/" << match->subnetMask  
 << " via " << match->nextHop << " on " << match->interfaceName  
 << " (Metric: " << match->metric << ")" << endl;  
 } else {  
 cout << "No matching route found." << endl;  
 }  
  
 return 0;  
}