**Title:** Determining the class of a given IP address, calculate the default network mask for that class, and generate subnet IP addresses.

**Done by:** Thota Surya Vara Prasad Rao, AP22110011059,CSE-G.

**Submitted on:** 31-08-2024

**Objective**

The objective of this program is to determine the class of a given IP address, calculate the default network mask for that class, and generate subnet IP addresses based on the number of subnet bits specified by the user. The program helps in understanding IP address classification and subnetting, which are essential for efficient network management and design.

**How It Works**

1. **IP Address Classification**:
   * The program first determines the class of the IP address by analyzing its first octet.
   * The class can be A, B, C, D, or E, depending on the range of the first octet.
2. **Default Network Mask Calculation**:
   * Based on the IP class, the program assigns the default network mask:
     + Class A: 255.0.0.0
     + Class B: 255.255.0.0
     + Class C: 255.255.255.0
   * Classes D and E do not have standard network masks as they are reserved for special purposes.
3. **Subnet Mask and Subnet Generation**:
   * The user specifies the number of subnet bits, which are used to create a custom subnet mask.
   * The program calculates this subnet mask by extending the default mask with the specified number of subnet bits.
   * Finally, the program generates the possible subnet IP addresses using this subnet mask and displays them.

**Limitations**

1. **Class D and E Handling**:
   * The program does not generate subnets for Class D and Class E IP addresses, as these are reserved for multicast and experimental purposes, respectively.
2. **Limited to IPv4**:
   * The program is designed to handle only IPv4 addresses. It does not support IPv6 address formats.
3. **Manual Input Requirement**:
   * The program requires the user to manually input the IP address and the number of subnet bits, which may not be suitable for automated or large-scale operations.
4. **No Validation of IP Format**:
   * The program assumes that the inputted IP address is in a valid format. It does not include error handling for invalid IP address formats.

**Problem Statement**

Develop a program that accepts an IP address from the user, determines its class, and displays the default network mask. The program should also allow the user to input the number of subnet bits, calculate the corresponding subnet mask, and generate subnet IP address based on the provided information. The program should handle IP addresses from Class A, B, and C while appropriately handling reserved addresses in Classes D and E.

**CODE:**

#include <iostream>

#include <string>

#include <vector>

#include <sstream>

using namespace std;

// Function to split a string into parts based on a delimiter

vector<string> split(const string& str, char delimiter) {

    vector<string> tokens;

    stringstream ss(str);

    string token;

    while (getline(ss, token, delimiter)) {

        tokens.push\_back(token);

    }

    return tokens;

}

// Function to get the class of an IP address

char get\_class(const string& ip) {

    int first\_byte = stoi(split(ip, '.')[0]);

    if (first\_byte < 128) {

        return 'A';

    } else if (first\_byte < 192) {

        return 'B';

    } else if (first\_byte < 224) {

        return 'C';

    } else if (first\_byte < 240) {

        return 'D';

    } else {

        return 'E';

    }

}

// Function to get the network mask based on the class and subnet bits

string get\_network\_mask(char ip\_class, int subnet\_bits) {

    if (ip\_class == 'A') {

        return "255." + to\_string(256 - (1 << (16 - subnet\_bits))) + ".0.0";

    } else if (ip\_class == 'B') {

        return "255.255." + to\_string(256 - (1 << (8 - subnet\_bits))) + ".0";

    } else if (ip\_class == 'C') {

        return "255.255.255." + to\_string(256 - (1 << (8 - subnet\_bits)));

    }

}

// Function to generate the subnet IP address

string generate\_subnet(const string& ip, const string& subnet\_mask) {

    vector<string> ip\_parts = split(ip, '.');

    vector<string> subnet\_mask\_parts = split(subnet\_mask, '.');

    string subnet\_ip;

    for (int i = 0; i < 4; i++) {

        int ip\_part = stoi(ip\_parts[i]);

        int subnet\_mask\_part = stoi(subnet\_mask\_parts[i]);

        subnet\_ip += to\_string(ip\_part & subnet\_mask\_part) + ".";

    }

    return subnet\_ip.substr(0, subnet\_ip.size() - 1); // Remove the trailing dot

}

int main() {

    string ip;

    cout << "Enter IP address: ";

    cin >> ip;

    int subnet\_bits;

    cout << "Enter subnet bits: ";

    cin >> subnet\_bits;

    char ip\_class = get\_class(ip);

    cout << "Class of IP address: " << ip\_class << endl;

    string subnet\_mask = get\_network\_mask(ip\_class, subnet\_bits);

    cout << "Network mask: " << subnet\_mask << endl;

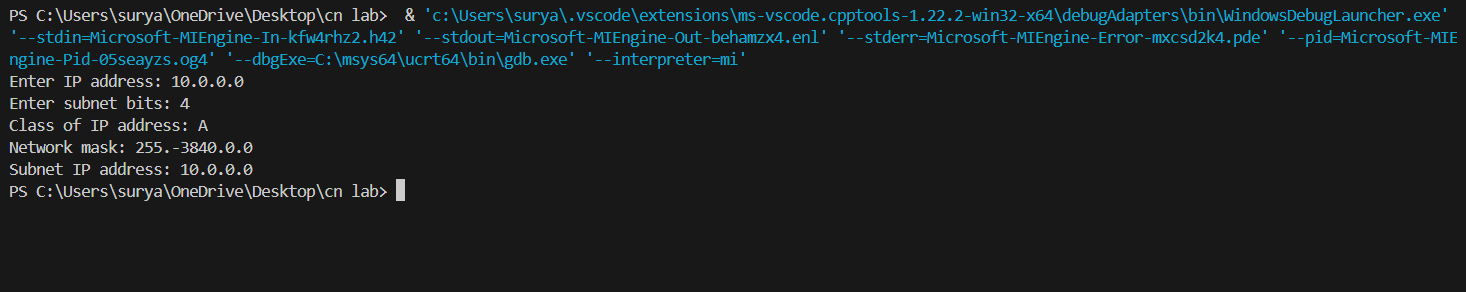
    string subnet\_ip = generate\_subnet(ip, subnet\_mask);

    cout << "Subnet IP address: " << subnet\_ip << endl;

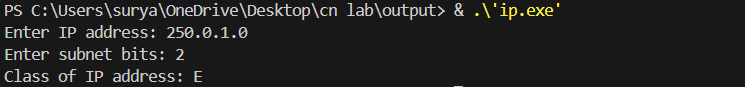
    return 0;

}

**OUTPUTS:**

 A screen shot of a computer program

Description automatically generated A screen shot of a computer program

Description automatically generated 

**Problems Faced**

1. **Handling Special IP Ranges**:
   * Managing IP addresses such as 0.0.0.0 and 127.0.0.1, which are reserved for specific purposes (e.g., default routes, loopback addresses) and do not fit neatly into the standard IP class categories. Properly classifying these addresses required additional logic and validation.

**Conclusion**

* **Knowledge**: Gained a thorough understanding of IP addressing, specifically the classification of IP addresses into Classes A, B, C, D, and E. Additionally, learned about special IP ranges like 0.0.0.0 (default route) and 127.0.0.1 (loopback address), and their significance in network configuration and troubleshooting.
* **Skill**: Enhanced C++ programming skills by implementing logic to determine and display the class of an IP address. This involved applying concepts such as bitwise operations, input validation, and conditional checks to accurately identify the class from the first octet of the IP address. The implementation also included handling special cases for reserved IP addresses, ensuring the program can correctly identify and process these non-standard cases. Additionally, the process of calculating subnet masks and generating subnet IP addresses reinforced the understanding of network segmentation and bitwise manipulation in practical scenarios.