Batch: D-2	**Roll No.: 16010123325**
Experiment / assignment / tutorial No.__6_____	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Experiment No.:6

TITLE: IP classes and Implementation of Subnet mask concept.

AIM: To study IP classes and Implementation of Subnet mask concept.

An IP (Internet Protocol) address is a unique identifier for a node or host connection on an IP network. Subnetting an IP Network can be done for a variety of reasons, including organization, use of different physical media (such as Ethernet, FDDI, WAN, etc.), preservation of address space, and security. The most common reason is to control network traffic. In an Ethernet network, all nodes on a segment see all the packets transmitted by all the other nodes on that segment. Performance can be adversely affected under heavy traffic loads, due to collisions and the resulting retransmissions. A router is used to connect IP networks to minimize the amount of traffic each segment must receive.

This experiment enables student for identifying the class of the IP address and design particular subnets as per user requirements.

Expected Outcome of Experiment:
CO:

Books/ Journals/ Websites referred:

1. A. S. Tanenbaum, “Computer Networks”, Pearson Education, Fourth Edition
2. B. A. Forouzan, “Data Communications and Networking”, TMH, Fourth Edition

Pre Lab/ Prior Concepts: IP Address, Classes, Subnet concept

New Concepts to be learned: Subnet mask calculation, Subnet address calculation

Stepwise-Procedure:

Applying a subnet mask to an IP address allows to identify the network and node parts of the address. The network bits are represented by the 1s in the mask, and the node bits are represented by the 0s. Performing a bitwise logical AND operation between the IP address and the subnet mask

results in the *Network Address* or Number.

Default subnet masks:

Class A - 255.0.0.0 - 11111111.00000000.00000000.00000000
Class B - 255.255.0.0 - 11111111.11111111.00000000.00000000
Class C - 255.255.255.0 - 11111111.11111111.11111111.00000000

Additional bits can be added to the default subnet mask for a given Class to further subnet, or break down, a network. When a bitwise logical AND operation is performed between the subnet mask and IP address, the result defines the *Subnet Address* (also called the *Network Address* or *Network Number*). There are some restrictions on the subnet address. Node addresses of all "0"s and all "1"s are reserved for specifying the local network (when a host does not know its network address) and all hosts on the network (broadcast address), respectively. This also applies to subnets. A subnet address cannot be all "0"s or all "1"s. This also implies that a 1 bit subnet mask is not allowed. This restriction is required because older standards enforced this restriction. Recent standards that allow use of these subnets have superseded these standards, but many "legacy" devices do not support the newer standards. If you are operating in a controlled environment, such as a lab, you can safely use these restricted subnets.

CIDR -- **C**lassless **I**nter **D**omain **R**outing:

The "classful" system of allocating IP addresses can be very wasteful; Under supernetting, the classful subnet masks are extended so that a network address and subnet mask could, for example, specify multiple Class C subnets with one address.

For example, If about 1000 addresses are required, it could be possible to supernet 4 Class C networks together:

192.60.128.0(11000000.00111100.10000000.00000000) Class C subnet address
192.60.129.0(11000000.00111100.10000001.00000000) Class C subnet address
192.60.130.0(11000000.00111100.10000010.00000000) Class C subnet address
192.60.131.0(11000000.00111100.10000011.00000000) Class C subnet address
---
192.60.128.0(11000000.00111100.10000000.00000000)Supernetted subnet address
255.255.252.0(11111111.11111111.11111100.00000000)SubnetMask 192.60.131.255
(11000000.00111100.10000011.11111111) Broadcast address

In this example, the subnet 192.60.128.0 includes all the addresses from 192.60.128.0 to 192.60.131.255. In the binary representation of the subnet mask, the Network portion of the address is 22 bits long, and the host portion is 10 bits long. Under CIDR, the subnet mask notation is reduced to simplified shorthand. Instead of spelling out the bits of the subnet mask, it is simply listed as the number of 1s bits that start the mask. In the above example, instead of writing the address and subnet mask as 192.60.128.0, Subnet Mask 255.255.252.0 .the network address would be written simply as: 192.60.128.0/22 Which indicates starting address of the network, and number of 1s bits (22) in

the network portion of the address. Subnet mask in binary

11111111.11111111.11111100.00000000.

The use of a CIDR notated address is the same as for a Classful address. Classful addresses can easily be written in CIDR notation as Class A = /8, Class B = /16, and Class C = /24
To calculate the number of subnets or nodes,

No. of Nodes/ Subnets = 2^n-2

Where n = number of bits in either field.

Multiplying the number of subnets by the number of nodes available per subnet gives you the total number of nodes available for your class and subnet mask. Also, note that although subnet masks with non-contiguous mask bits are allowed, they are not recommended.

Example:

10001100.10110011.11011100.11001000	140.179.220.200	IP Address
11111111.11111111.**111**00000.00000000	255.255.**224**.000	Subnet Mask
10001100.10110011.11000000.00000000	140.179.192.000	Subnet Address
10001100.10110011.11011111.11111111	40.179.223.255	Broadcast Address

1. Program starts with taking IP address from user and the number of subnets from the user.
2. Then the calculation for subnet mask is done as specified in methodology.
3. Then with AND ing with subnet mask the subnet addresses are calculated.

IMPLEMENTATION:

1) Classful-

```cpp
#include <sstream>
#include <vector>
#include <string>
using namespace std;

vector<string> split(const string& str, char delimiter) {
    vector<string> parts;
    string part;
    stringstream ss(str);
    while (getline(ss, part, delimiter)) {
        parts.push_back(part);
    }
    return parts;
}

int main() {
    string ip;
    cout << "Enter IP Address: ";
    cin >> ip;

    vector<string> parts = split(ip, '.');
    if (parts.size() != 4) {
        cout << "Invalid IP Address!" << endl;
        return 0;
    }

    int firstOctet = stoi(parts[0]);
    string ipClass = "";
    string defaultMask = "";

    // Determine Class
    if (firstOctet >= 1 && firstOctet <= 127) {
        ipClass = "A";
        defaultMask = "255.0.0.0";
    } else if (firstOctet >= 128 && firstOctet <= 191) {
        ipClass = "B";
        defaultMask = "255.255.0.0";
    } else if (firstOctet >= 192 && firstOctet <= 223) {
        ipClass = "C";
        defaultMask = "255.255.255.0";
    } else if (firstOctet >= 224 && firstOctet <= 239) {
        ipClass = "D (Multicast)";
        defaultMask = "N/A";
    } else if (firstOctet >= 240 && firstOctet <= 255) {
        ipClass = "E (Experimental)";
```

```cpp
        defaultMask = "N/A";
    } else {
        cout << "Invalid IP Address!" << endl;
        return 0;
    }

    cout << "Class: " << ipClass << endl;
    cout << "Default Mask: " << defaultMask << endl;

    // First and Last Address (only for Class A, B, C)
    if (ipClass == "A") {
        cout << "First Address: " << parts[0] << ".0.0.0" << endl;
        cout << "Last Address: " << parts[0] << ".255.255.255" << endl;
    } else if (ipClass == "B") {
        cout << "First Address: " << parts[0] << "." << parts[1] << ".0.0" << endl;
        cout << "Last Address: " << parts[0] << "." << parts[1] << ".255.255" << endl;
    } else if (ipClass == "C") {
        cout << "First Address: " << parts[0] << "." << parts[1] << "." << parts[2] << ".0" << endl;
        cout << "Last Address: " << parts[0] << "." << parts[1] << "." << parts[2] << ".255" << endl;
    } else {
        cout << "First/Last Address: Not applicable" << endl;
    }

    // Only for Class A, B, C
    if (ipClass == "A" || ipClass == "B" || ipClass == "C") {
        vector<int> ipOctets(4), maskOctets(4);
        vector<string> maskParts = split(defaultMask, '.');

        for (int i = 0; i < 4; i++) {
            ipOctets[i] = stoi(parts[i]);
            maskOctets[i] = stoi(maskParts[i]);
        }

        // Network Address: ip & mask
        vector<int> network(4);
        for (int i = 0; i < 4; i++) {
            network[i] = ipOctets[i] & maskOctets[i];
        }
        cout << "Network Address: "
             << network[0] << "." << network[1] << "." << network[2] << "." << network[3] << endl;

        // Broadcast Address: ip | ~mask
        vector<int> broadcast(4);
        for (int i = 0; i < 4; i++) {
            broadcast[i] = ipOctets[i] | (~maskOctets[i] & 0xFF);
        }
        cout << "Broadcast Address: "
             << broadcast[0] << "." << broadcast[1] << "." << broadcast[2] << "." << broadcast[3] << endl;
    } else {
        cout << "Network/Broadcast Address: Not applicable" << endl;
    }
```

Output-

```
Enter IP Address: 188.16.22.22
Class: B
Default Mask: 255.255.0.0
First Address: 188.16.0.0
Last Address: 188.16.255.255
Network Address: 188.16.0.0
Broadcast Address: 188.16.255.255
```

```
Enter IP Address: 245.22.12.9
Class: E (Experimental)
Default Mask: N/A
First/Last Address: Not applicable
Network/Broadcast Address: Not applicable
```

2) Classless-

```
}

// Convert octets to string
string octetsToString(const vector<int> &o) {
    return to_string(o[0]) + "." + to_string(o[1]) + "." + to_string(o[2]) + "." + to_string(o[3]);
}

int main() {
    string ipStr;
    cout << "Enter IP Address: ";
    cin >> ipStr;

    vector<string> parts = split(ipStr, '.');
    if (parts.size() != 4) {
        cout << "Invalid IP Address!" << endl;
        return 0;
    }

    vector<int> ipOct(4);
    try {
        for (int i = 0; i < 4; i++) ipOct[i] = stoi(parts[i]);
    } catch (...) {
        cout << "Invalid IP octet!" << endl;
        return 0;
    }

    cin.ignore();
    cout << "Enter subnet mask (dotted) OR prefix (e.g. /24 or 24): ";
    string maskInput;
    getline(cin, maskInput);
    maskInput.erase(remove(maskInput.begin(), maskInput.end(), ' '), maskInput.end());

    vector<int> maskOct;
    try {
        if (maskInput[0] == '/') maskInput = maskInput.substr(1);

        if (maskInput.find('.') != string::npos) {
            maskOct = dottedMaskToOctets(maskInput);
        } else {
            int prefix = stoi(maskInput);
            if (prefix < 0 || prefix > 32) {
                cout << "Prefix must be 0..32" << endl;
                return 0;
            }
            maskOct = prefixToMaskOctets(prefix);
        }
    } catch (...) {
        cout << "Invalid mask or prefix!" << endl;
        return 0;
    }

    cout << "Mask: " << octetsToString(maskOct) << endl;
```

```
#include <sstream>
#include <vector>
#include <string>
#include <cmath>
#include <bitset>
using namespace std;

vector<string> split(const string &str, char delimiter) {
    vector<string> parts;
    string part;
    stringstream ss(str);
    while (getline(ss, part, delimiter)) parts.push_back(part);
    return parts;
}

vector<int> prefixToMaskOctets(int prefix) {
    unsigned int mask = (prefix == 0) ? 0 : 0xFFFFFFFF << (32 - prefix);
    vector<int> oct(4);
    oct[0] = (mask >> 24) & 0xFF;
    oct[1] = (mask >> 16) & 0xFF;
    oct[2] = (mask >> 8) & 0xFF;
    oct[3] = mask & 0xFF;
    return oct;
}

vector<int> dottedMaskToOctets(const string &maskStr) {
    vector<string> parts = split(maskStr, '.');
    if (parts.size() != 4) throw invalid_argument("Invalid dotted mask");
    vector<int> oct(4);
    for (int i = 0; i < 4; i++) oct[i] = stoi(parts[i]);
    return oct;
}

vector<int> networkFromIpAndMask(const vector<int> &ip, const vector<int> &mask) {
    vector<int> net(4);
    for (int i = 0; i < 4; i++) net[i] = ip[i] & mask[i];
    return net;
}

vector<int> broadcastFromIpAndMask(const vector<int> &ip, const vector<int> &mask) {
    vector<int> b(4);
    for (int i = 0; i < 4; i++) b[i] = ip[i] | (~mask[i] & 0xFF);
    return b;
}

vector<int> addToOctets(const vector<int> &oct, long add) {
    unsigned long val = 0;
    for (int i = 0; i < 4; i++) val = (val << 8) | (oct[i] & 0xFF);
    val = (val + add) & 0xFFFFFFFF;
    vector<int> res(4);
    res[0] = (val >> 24) & 0xFF;
    res[1] = (val >> 16) & 0xFF;
    res[2] = (val >> 8) & 0xFF;
```

```
    vector<int> network = networkFromIpAndMask(ipOct, maskOct);
    vector<int> broadcast = broadcastFromIpAndMask(ipOct, maskOct);

    cout << "Network Address: " << octetsToString(network) << endl;
    cout << "Broadcast Address: " << octetsToString(broadcast) << endl;

    // Calculate prefix length
    unsigned int maskInt = (maskOct[0] << 24) | (maskOct[1] << 16) | (maskOct[2] << 8) | maskOct[3];
    int prefixLen = 0;
    for (int i = 31; i >= 0; i--) {
        if ((maskInt >> i) & 1) prefixLen++;
        else break;
    }

    unsigned long total = (prefixLen == 32) ? 1UL : (1UL << (32 - prefixLen));
    unsigned long usable = (prefixLen >= 31) ? 0UL : (total - 2);

    cout << "Total: " << total << endl;
    cout << "Usable hosts: " << usable << endl;

    if (usable > 0) {
        vector<int> first = addToOctets(network, 1);
        vector<int> last = addToOctets(broadcast, -1);
        cout << "First Usable: " << octetsToString(first) << endl;
        cout << "Last Usable:  " << octetsToString(last) << endl;
    } else if (prefixLen == 31) {
        cout << "Addresses (point-to-point): "
             << octetsToString(network) << " , " << octetsToString(broadcast) << endl;
    } else if (prefixLen == 32) {
        cout << "Single host: " << octetsToString(network) << endl;
    }

    return 0;
}
```

Output-

```
Enter IP Address: 124.55.12.2
Enter subnet mask (dotted) OR prefix (e.g. /24 or 24): 16
Mask: 255.255.0.0
Network Address: 124.55.0.0
Broadcast Address: 124.55.255.255
Total: 65536
Usable hosts: 65534
First Usable: 124.55.0.1
Last Usable:  124.55.255.254
```

Subnetting-

```
unsigned int ipToInt(const string &ip) {
    stringstream ss(ip);
    string part;
    unsigned int result = 0;
    for (int i = 0; i < 4; i++) {
        getline(ss, part, '.');
        result = (result << 8) | (stoi(part) & 0xFF);
    }
    return result;
}

// Convert 32-bit unsigned int to dotted IP
string intToIp(unsigned int val) {
    stringstream ss;
    ss << ((val >> 24) & 0xFF) << "."
       << ((val >> 16) & 0xFF) << "."
       << ((val >> 8) & 0xFF) << "."
       << (val & 0xFF);
    return ss.str();
}

// Convert prefix (e.g. /24) to subnet mask
unsigned int prefixToMask(int prefix) {
    if (prefix == 0) return 0;
    return (0xFFFFFFFFu << (32 - prefix));
}

// Convert mask to prefix length
int maskToPrefix(unsigned int mask) {
    int count = 0;
    for (int i = 31; i >= 0; i--) {
        if ((mask >> i) & 1) count++;
        else break;
    }
    return count;
}

int main() {
    string ipStr, maskStr;

    cout << "IP (dotted): ";
    getline(cin, ipStr);

    cout << "Mask (dotted) or prefix (e.g. /24 or 24): ";
    getline(cin, maskStr);

    try {
        unsigned int ip = ipToInt(ipStr);
        int prefix;
```

```
        if (maskStr.find('.') != string::npos) {
            unsigned int mask = ipToInt(maskStr);
            prefix = maskToPrefix(mask);
        } else {
            prefix = stoi(maskStr);
        }

        unsigned int mask = prefixToMask(prefix);
        unsigned int network = ip & mask;
        unsigned int broadcast = network | (~mask);

        unsigned long total = (prefix == 32) ? 1UL : (1UL << (32 - prefix));
        unsigned long usable = (prefix >= 31) ? 0 : (total - 2);

        // Determine default prefix based on IP class
        int firstOctet = stoi(ipStr.substr(0, ipStr.find('.')));
        int defaultPrefix;
        if (firstOctet >= 1 && firstOctet <= 126) defaultPrefix = 8;      // Class A
        else if (firstOctet >= 128 && firstOctet <= 191) defaultPrefix = 16; // Class B
        else if (firstOctet >= 192 && firstOctet <= 223) defaultPrefix = 24; // Class C
        else defaultPrefix = prefix; // Other classes

        cout << "Enter desired number of subnets: ";
        int desiredSubnets;
        cin >> desiredSubnets;

        int subnetBits = ceil(log2(desiredSubnets));
        int newPrefix = defaultPrefix + subnetBits;

        if (newPrefix > 32) {
            cout << "Cannot create that many subnets with this IP class.\n";
            return 0;
        }

        unsigned int newMask = prefixToMask(newPrefix);
        unsigned int newNetwork = ip & newMask;
        unsigned int newBroadcast = newNetwork | (~newMask);

        unsigned long newTotal = (newPrefix == 32) ? 1UL : (1UL << (32 - newPrefix));
        unsigned long newUsable = (newPrefix >= 31) ? 0 : (newTotal - 2);

        cout << "\n--- Subnet Info (User Input) ---\n";
        cout << "IP:           " << ipStr << endl;
        cout << "Prefix:       /" << newPrefix << endl;
        cout << "Mask:         " << intToIp(newMask) << endl;
        cout << "Network:      " << intToIp(newNetwork) << endl;
        cout << "Broadcast:    " << intToIp(newBroadcast) << endl;
        cout << "Total:        " << newTotal << endl;
        cout << "Usable hosts: " << newUsable << endl;
        cout << "Number of subnets: " << (1UL << subnetBits) << endl;

        if (newUsable > 0) {
            cout << "First usable: " << intToIp(newNetwork + 1) << endl;
            cout << "Last usable:  " << intToIp(newBroadcast - 1) << endl;
        }
    }
    catch (...) {
        cout << "Invalid input!" << endl;
    }

    return 0;
}
```

Output-

```
IP (dotted): 192.168.32.2
Mask (dotted) or prefix (e.g. /24 or 24): 26
Enter desired number of subnets: 4

--- Subnet Info (User Input) ---
IP:           192.168.32.2
Prefix:       /26
Mask:         255.255.255.192
Network:      192.168.32.0
Broadcast:    192.168.32.63
Total:        64
Usable hosts: 62
Number of subnets: 4
First usable: 192.168.32.1
Last usable:  192.168.32.62
```

CONCLUSION:

The above experiment demonstrates implementation IP classless like Classless and Classful along with Subnetting in Java.

Post Lab Questions

1. Which of the following is private IP address?
   A. 12.0.0.1 B. 168.172.19.39
   C. 172.15.14.36 **D. 192.168.24.43**
2. Which class of IP address provides a maximum of only 254 host addresses per network ID?
   A. Class A
   B. Class B
   C. Class C
   D. Class D
3. What is the address range of a Class B network address in binary?
   A. 01xxxxxx
   B. 0xxxxxxx
   C. 10xxxxxx
   D. 110xxxxx
4. Which two statements describe the IP address 10.16.3.65/23?
   1.The subnet address is 10.16.3.0 255.255.254.0.
   2.The lowest host address in the subnet is 10.16.2.1 255.255.254.0.
   3.The last valid host address in the subnet is 10.16.2.254 255.255.254.0.
   4.The broadcast address of the subnet is 10.16.3.255 255.255.254.0.
   A. 1 and 3
   B. 2 and 4
   C. 1, 2 and 4
   D. 2, 3 and 4
5. What is the maximum number of IP addresses that can be assigned to hosts on a local subnet that uses the 255.255.255.224 subnet mask?

   A. 14 B. 15
   C. 16 **D. 30**

6. You need to subnet a network that has 5 subnets, each with at least 16 hosts. Which classful subnet mask would you use?

   A. 255.255.255.192 B. 255.255.255.224
   C. 255.255.255.240 D. 255.255.255.248

7. You have a network that needs 29 subnets while maximizing the number of host addresses available on each subnet. How many bits must you borrow from the host field to provide the correct subnet mask?

A. 2 B. 3
C. 4 **D. 5**

Date :_____ **Signature of Faculty In-charge**