**Registration Number: 19BCE2119**

**Name: Gaurav Kumar Singh**

**Course: Network and Communication CSE1004**

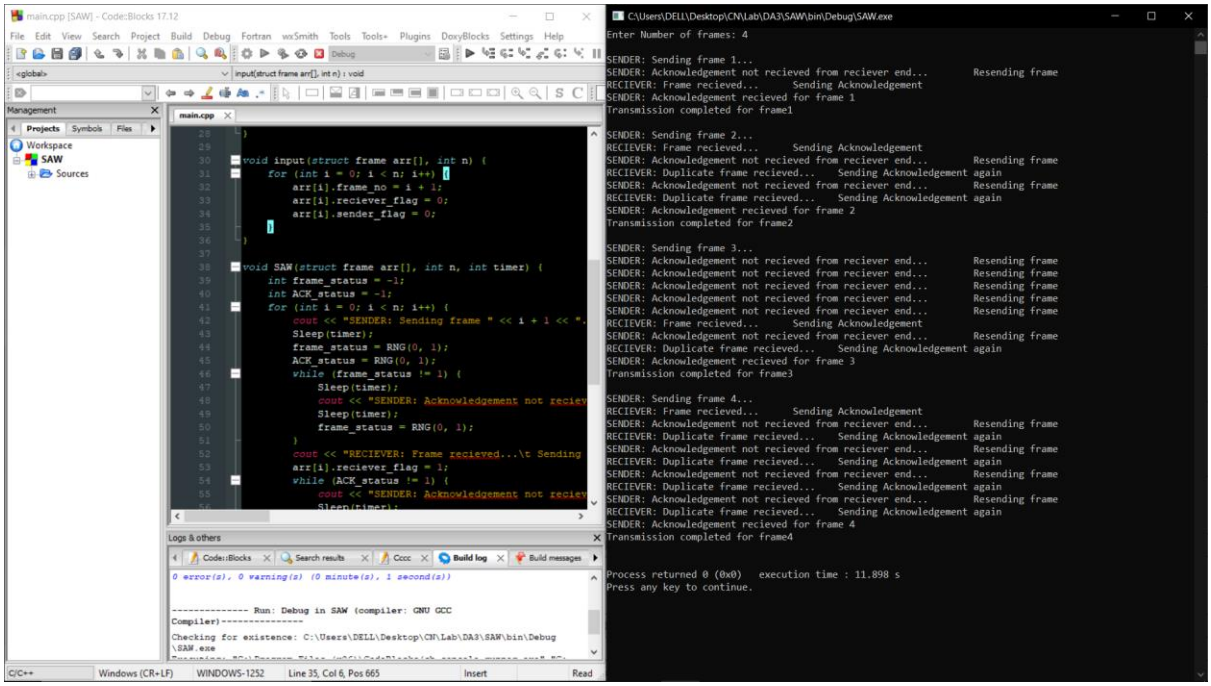
**Digital Assignment 3**

**Stop and Wait ARQ protocol, Go Back-N ARQ protocol, Selective Repeat ARQ protocol, IPv4 Classless Addressing**

# STOP AND WAIT

```
#include <iostream>

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include <time.h>

#include <windows.h>

using namespace std;

struct frame {
    int frame_no;
    int reciever_flag; //activates when frame recieved by reciever
    int sender_flag; //activates when ACK recieved by sender
};

int RNG(int lower, int upper) {
    int num = (rand() % (upper - lower + 1)) + lower;
    return num;
}

void init() {
    srand(time(NULL));
}

void input(struct frame arr[], int n) {
    for (int i = 0; i < n; i++) {
        arr[i].frame_no = i + 1;
        arr[i].reciever_flag = 0;
        arr[i].sender_flag = 0;
    }
}

void SAW(struct frame arr[], int n, int timer) {
    int frame_status = -1;
    int ACK_status = -1;
    for (int i = 0; i < n; i++) {
        cout << "SENDER: Sending frame " << i + 1 << "..." << endl;
        Sleep(timer);
        frame_status = RNG(0, 1);
        ACK_status = RNG(0, 1);
        while (frame_status != 1) {
            Sleep(timer);
```

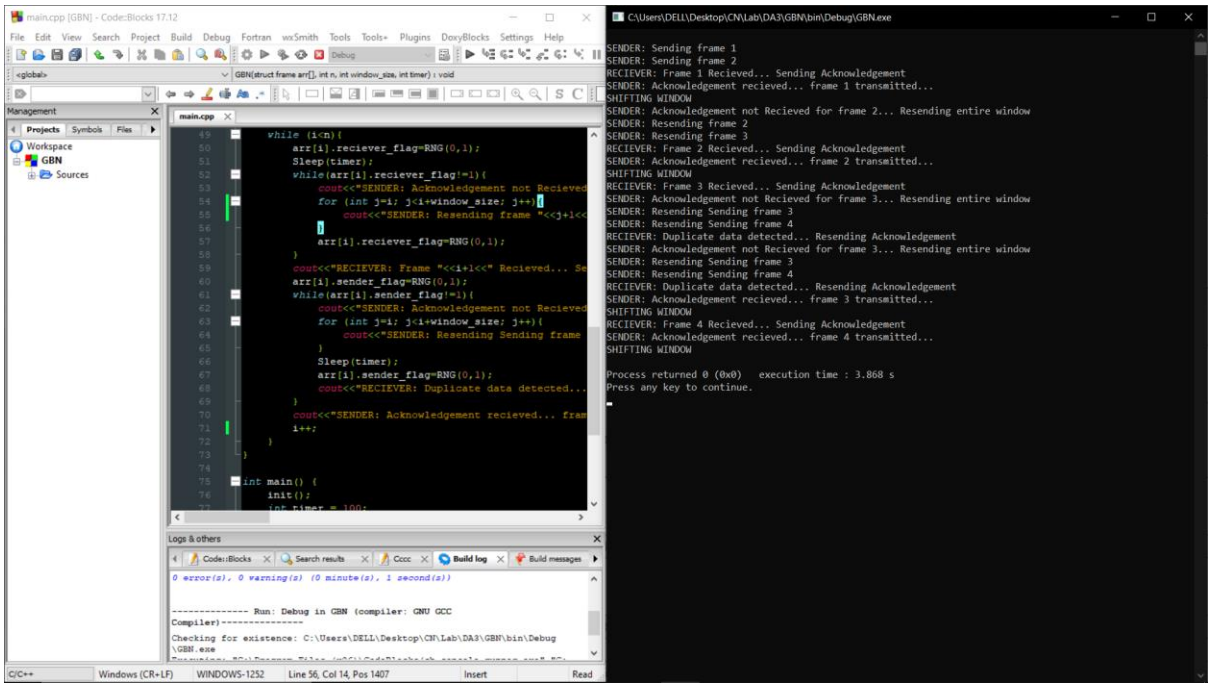
```
            cout << "SENDER: Acknowledgement not recieved from reciever end...\t
Resending frame\n";
            Sleep(timer);
            frame_status = RNG(0, 1);
        }
        cout << "RECIEVER: Frame recieved...\t Sending Acknowledgement\n";
        arr[i].reciever_flag = 1;
        while (ACK_status != 1) {
            cout << "SENDER: Acknowledgement not recieved from reciever end...\t
Resending frame\n";
            Sleep(timer);
            cout << "RECIEVER: Duplicate frame recieved...\t Sending Acknowledgement
again\n";
            Sleep(timer);
            ACK_status = RNG(0, 1);
        }
        arr[i].sender_flag = 1;

        cout << "SENDER: Acknowledgement recieved for frame " << i + 1 <<
"\nTransmission completed for frame" << i + 1 << "\n\n";
    }
}

int main() {
    init();
    int n;
    int timer = 100;
    cout << "Enter Number of frames: ";
    cin >> n;
    cout << "\n";
    struct frame arr[100];
    input(arr, n);
    SAW(arr, n, timer);

    return 0;
}
```




# GO BACK-N

```
#include <iostream>

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include <time.h>

#include <windows.h>

using namespace std;

struct frame {
    int frame_no;
    int reciever_flag; //activates when frame recieved by reciever
    int sender_flag; //activates when ACK recieved by sender
};

int RNG(int lower, int upper) {
    int num = (rand() % (upper - lower + 1)) + lower;
    return num;
}

void init() {
    srand(time(NULL));
}

void input(struct frame arr[], int n, int window_size) {
    for (int i = 0; i < n; i++) {
        arr[i].frame_no = i + 1;
        arr[i].reciever_flag = 0;
        arr[i].sender_flag = 0;
    }
```

```
    for (int i = n; i< n+window_size; i++){
        arr[i].frame_no = -1;
        arr[i].reciever_flag = 1;
        arr[i].sender_flag = 1;
    }
}

void GBN(struct frame arr[], int n, int window_size,int timer) {
    int i=0;
    for (int j=i; j<i+window_size; j++){
            cout<<"SENDER: Sending frame "<<j+1<<"\n";
        }
        Sleep(timer);
    while (i<n){
        arr[i].reciever_flag=RNG(0,1);
        Sleep(timer);
        while(arr[i].reciever_flag!=1){
            cout<<"SENDER: Acknowledgement not Recieved for frame "<<i+1<<"...
Resending entire window\n";
            for (int j=i; j<i+window_size; j++){
                cout<<"SENDER: Resending Sending frame "<<j+1<<"\n";
            }
            arr[i].reciever_flag=RNG(0,1);
        }
        cout<<"RECIEVER: Frame "<<i+1<<" Recieved... Sending Acknowledgement\n";
        arr[i].sender_flag=RNG(0,1);
        while(arr[i].sender_flag!=1){
            cout<<"SENDER: Acknowledgement not Recieved for frame "<<i+1<<"...
Resending entire window\n";
            for (int j=i; j<i+window_size; j++){
                cout<<"SENDER: Resending Sending frame "<<j+1<<"\n";
            }
            Sleep(timer);
            arr[i].sender_flag=RNG(0,1);
            cout<<"RECIEVER: Duplicate data detected... Resending Acknowledgement\n";
        }
        cout<<"SENDER: Acknowledgement recieved... frame "<<i+1<<"
transmitted...\nSHIFTING WINDOW\n";
        i++;
        if (arr[i+window_size].frame_no != -1){
            cout<<"SENDER: Sending frame "<<i+window_size<<"\n";
        }
    }
}

int main() {
    init();
    int timer = 100;
    int n;
    int window_size;
    cout << "Enter Window Size: ";
    cin>> window_size;
    cout << "\n";
    cout << "Enter Number of frames: ";
    cin >> n;
    cout << "\n";
    struct frame arr[100];
    input(arr, n, window_size);
    GBN(arr, n, window_size, timer);

    return 0;
}
```



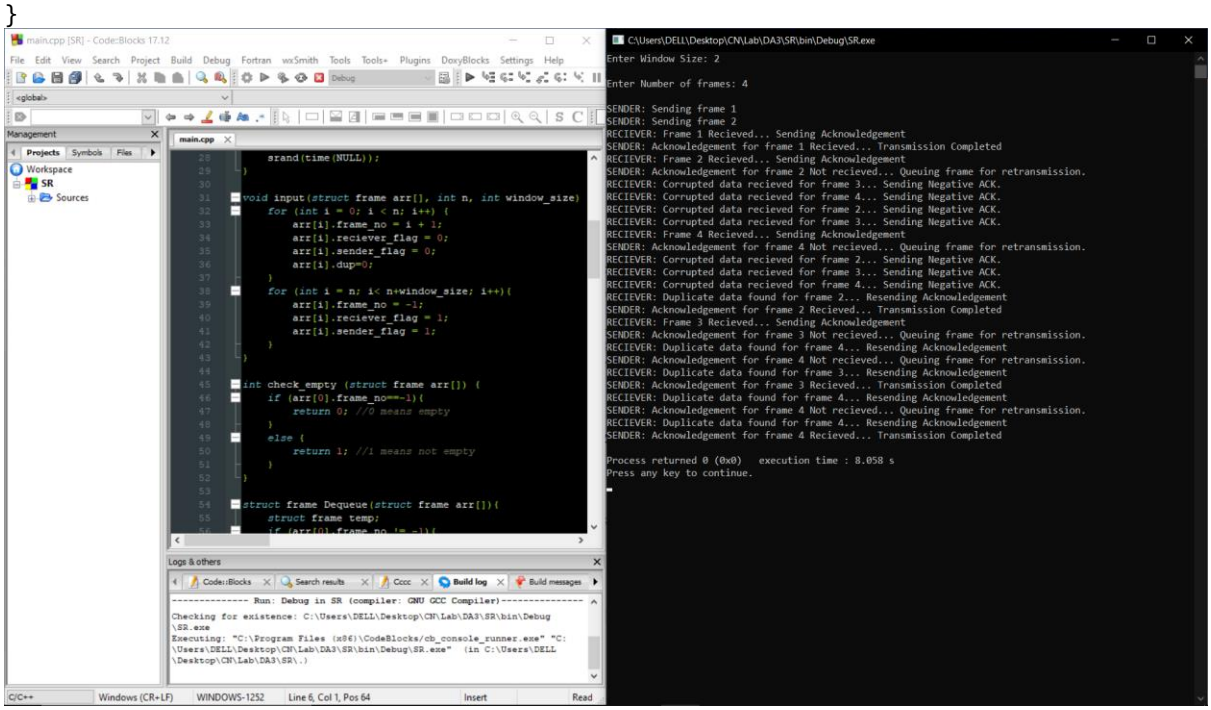

# SELECTIVE REPEAT

```
#include <iostream>

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include <time.h>

#include <windows.h>

using namespace std;

struct frame {
    int frame_no;
    int reciever_flag; //activates when frame recieved by reciever
    int sender_flag; //activates when ACK recieved by sender
    int dup;
};

int RNG(int lower, int upper) {
    int num = (rand() % (upper - lower + 1)) + lower;
    return num;
}

void init() {
    srand(time(NULL));
}

void input(struct frame arr[], int n, int window_size) {
    for (int i = 0; i < n; i++) {
        arr[i].frame_no = i + 1;
        arr[i].reciever_flag = 0;
```

```
        arr[i].sender_flag = 0;
        arr[i].dup=0;
    }
    for (int i = n; i< n+window_size; i++){
        arr[i].frame_no = -1;
        arr[i].reciever_flag = 1;
        arr[i].sender_flag = 1;
    }
}

int check_empty (struct frame arr[]) {
    if (arr[0].frame_no==-1){
        return 0; //0 means empty
    }
    else {
        return 1; //1 means not empty
    }
}

struct frame Dequeue(struct frame arr[]){
    struct frame temp;
    if (arr[0].frame_no != -1){
        temp=arr[0];
    }
    int i=0;
    while (arr[i].frame_no != -1){
        arr[i]=arr[i+1];
        i++;
    }
    return temp;
};

void Enqueue(struct frame buffer[], struct frame temp){
    int i=0;
    int j=0;

    while (buffer[i].frame_no != -1){
        i++;
    }
    buffer[i]=temp;
}

void SR(struct frame arr[], int n, int window_size,int timer) {
    int i=0;
    int counter=0;
    struct frame buffer[100];
    struct frame ready[window_size+1];
    input(ready, 0, window_size+1);
    struct frame temp;
    input(buffer, 0, 100);
    for (int j=0; j<window_size; j++){
            cout<<"SENDER: Sending frame "<<j+1<<"\n";
            ready[j]=arr[j];
            counter++;
        }
        Sleep(timer);
    while (check_empty(ready) || check_empty(buffer) || i<n){
        if (!check_empty(ready)){
            if(check_empty(buffer)){
                Enqueue(ready, Dequeue(buffer));
            }
            else if(counter!=n && !check_empty(buffer)){
```

```
                Enqueue(ready, arr[counter]);
                counter++;
            }

        }

        ready[0].reciever_flag=RNG(0,1);
        Sleep(timer);
        if (ready[0].reciever_flag){
            if (ready[0].dup==1){
                cout<<"RECIEVER: Duplicate data found for frame
"<<ready[0].frame_no<<"... Resending Acknowledgement\n";

            }
            else {
                cout<<"RECIEVER: Frame "<<ready[0].frame_no<<" Recieved... Sending
Acknowledgement\n";
                ready[0].dup=1;
            }
            temp=Dequeue(ready);
            if(check_empty(buffer)){
                Enqueue(ready, Dequeue(buffer));
            }
            else if(counter!=n && !check_empty(buffer)){
                Enqueue(ready, arr[counter]);
                counter++;
            }
            Sleep(timer);
            temp.sender_flag=RNG(0,1);
            if (temp.sender_flag){
                cout<<"SENDER: Acknowledgement for frame "<<temp.frame_no<<"
Recieved... Transmission Completed\n";
            }
            else{
                cout<<"SENDER: Acknowledgement for frame "<<temp.frame_no<<" Not
recieved... Queuing frame for retransmission.\n";
                Enqueue(buffer, temp);
            }
        }
        else {
            cout<<"RECIEVER: Corrupted data recieved for frame
"<<ready[0].frame_no<<"... Sending Negative ACK.\n";
            Enqueue(buffer, Dequeue(ready));
            if(check_empty(buffer)){
                Enqueue(ready, Dequeue(buffer));
            }
            else if(counter!=n && !check_empty(buffer)){
                Enqueue(ready, arr[counter]);
                counter++;
            }

        }
        i++;
    }
}

void display(struct frame arr[], int n){
    for (int i=0; i<n; i++){
        cout<<arr[i].frame_no<<"\t";
    }
}
```

```
int main() {
    init();
    int timer = 100;
    int n;
    int window_size;
    cout << "Enter Window Size: ";
    cin>> window_size;
    cout << "\n";
    cout << "Enter Number of frames: ";
    cin >> n;
    cout << "\n";
    struct frame arr[100];
    input(arr, n, window_size);
    SR(arr, n, window_size, timer);

    return 0;
}
```

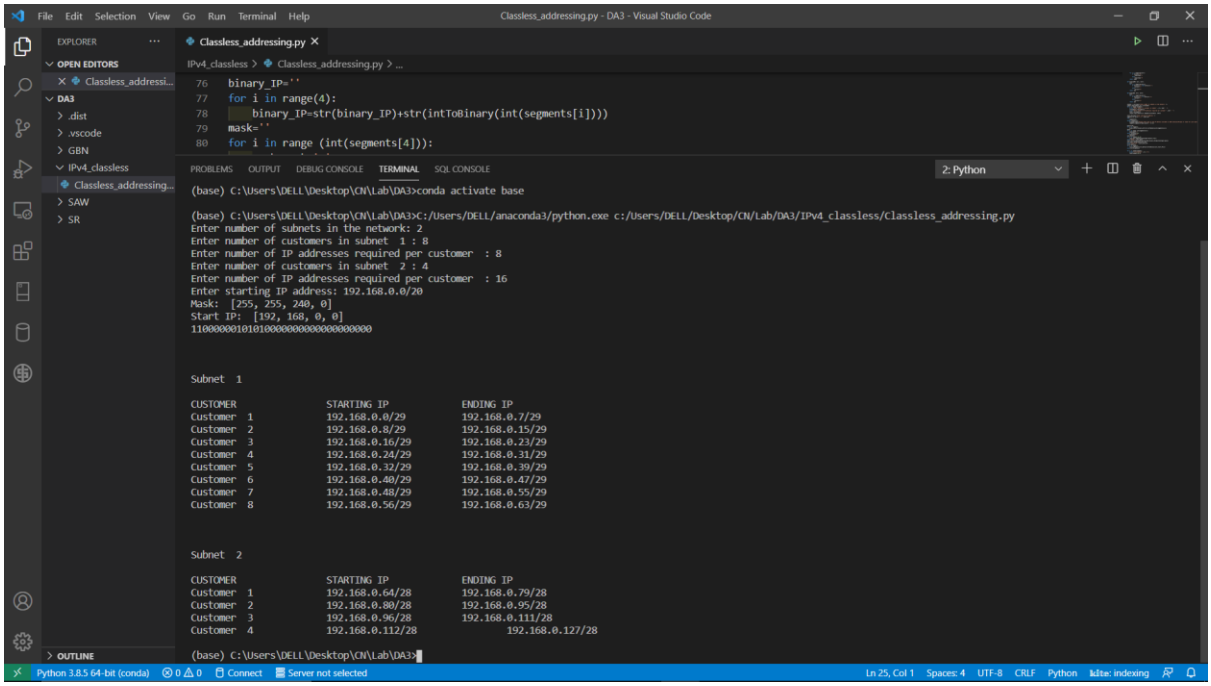



## IPV4 CLASSLESS ADDRESSING

```
import re
import numpy as np

def intToBinary(var):
    binary=bin(var).split("0b")[1]
    while(len(binary)<8):
        binary='0'+binary
    return binary

def intToBinary32(var):
    binary=bin(var).split("0b")[1]
    while(len(binary)<32):
        binary='0'+binary
```

```
    return binary

def binToInteger(var):
    return (int(var,2))

def display(temp):
    var=temp
    arr=[]
    for i in range(0,32,8):
        arr.append(binToInteger(var[i:i+8]))
    return arr

def IPv4Format(arr):
    ipv4=str(arr[0])
    for i in range(1,4,1):
        ipv4=ipv4+'.'+str(arr[i])
    return ipv4

def complement(var):
    temp=''
    for i in range(len(var)):
        if var[i]=='0':
            temp=temp+'1'
        else:
            temp=temp+'0'
    return temp

def binaryAND( var1, var2):
    ans=''
    for i in range(len(var1)):
        if var1[i]=='1' and var2[i]=='1':
            ans=ans+'1'
        else:
            ans=ans+'0'
    return ans

def binaryOR( var1, var2):
    ans=''
    for i in range(len(var1)):
        if var1[i]=='1' or var2[i]=='1':
            ans=ans+'1'
        else:
            ans=ans+'0'
    return ans

nsubnet = int(input("Enter number of subnets in the network: "))
subnet_requirements_arr =[]*nsubnet
for i in range (nsubnet):
    print("Enter number of customers in subnet ", i+1, end=" : ")
    ncustomers= int(input())
    print("Enter number of IP addresses required per customer ", end=" : ")
    nIPs= int(input())
    subnet_requirements_arr.append([ncustomers, nIPs])

start_IP=input("Enter starting IP address: ")
segments=re.split('\.|/', start_IP)
try:
    segments[4]
except IndexError:
    errmask=input("Please enter mask of the IP address provided in CIDR
notation(IP/mask or /mask for just mask): ")
    segments.append(errmask.split('/')[-1])
```

```
binary_IP=''
for i in range(4):
    binary_IP=str(binary_IP)+str(intToBinary(int(segments[i])))
mask=''
for i in range (int(segments[4])):
    mask=mask+'1'
while len(mask)<32:
    mask=mask+'0'
init_mask=[]
for i in range(0,32,8):
    init_mask.append(binToInteger(mask[i:i+8]))
print("Mask: ",init_mask)
start_IP=intToBinary32(binToInteger(binary_IP)&binToInteger(mask))
init_start_IP=display(start_IP)
print("Start IP: ",init_start_IP)
first_address=''
for i in range(4):
    first_address=first_address+intToBinary(init_start_IP[i])
print(first_address)

for i in range(nsubnet):
    print("\n\n\nSubnet ",i+1,"\n")
    nums=range(0,33,1)
    portbits=np.log2(subnet_requirements_arr[i][1])
    if portbits not in nums:
        portbits=int(portbits)+1
    nmask=int(32-portbits)
    mask=''
    for k in range(nmask):
        mask=mask+'1'
    while len(mask)<32:
        mask=mask+'0'
    print("CUSTOMER\t\tSTARTING IP\t\tENDING IP")
    for j in range(subnet_requirements_arr[i][0]):
        print("Customer ", j+1,end='\t\t')
        print(IPv4Format(display(binaryAND(mask,
first_address)))+'/'+str(nmask),end='\t\t')

print(IPv4Format(display(binaryOR(complement(mask),first_address)))+'/'+str(nmask))

first_address=intToBinary32(binToInteger(binaryOR(complement(mask),first_address))+1)
```


File Edit Selection View Go Run Terminal Help
Classless_addressing.py - DA3 - Visual Studio Code
EXPLORER
OPEN EDITORS
Classless_addressi...
DA3
.dist
.vscode
GBN
IPv4_classless
Classless_addressing...
SAW
SR
OUTLINE
Classless_addressing.py
IPv4_classless > Classless_addressing.py > ...
76 binary_IP=''
77 for i in range(4):
78 binary_IP=str(binary_IP)+str(intToBinary(int(segments[i])))
79 mask=''
80 for i in range (int(segments[4])):
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE
2: Python
(base) C:\Users\DELL\Desktop\CN\Lab\DA3>conda activate base
(base) C:\Users\DELL\Desktop\CN\Lab\DA3>C:/Users/DELL/anaconda3/python.exe c:/Users/DELL/Desktop/CN/Lab/DA3/IPv4_classless/Classless_addressing.py
Enter number of subnets in the network: 2
Enter number of customers in subnet 1 : 8
Enter number of IP addresses required per customer : 8
Enter number of customers in subnet 2 : 4
Enter number of IP addresses required per customer : 16
Enter starting IP address: 192.168.0.0/20
Mask: [255, 255, 240, 0]
Start IP: [192, 168, 0, 0]
11000000101010000000000000000000
Subnet 1
CUSTOMER STARTING IP ENDING IP
Customer 1 192.168.0.0/29 192.168.0.7/29
Customer 2 192.168.0.8/29 192.168.0.15/29
Customer 3 192.168.0.16/29 192.168.0.23/29
Customer 4 192.168.0.24/29 192.168.0.31/29
Customer 5 192.168.0.32/29 192.168.0.39/29
Customer 6 192.168.0.40/29 192.168.0.47/29
Customer 7 192.168.0.48/29 192.168.0.55/29
Customer 8 192.168.0.56/29 192.168.0.63/29
Subnet 2
CUSTOMER STARTING IP ENDING IP
Customer 1 192.168.0.64/28 192.168.0.79/28
Customer 2 192.168.0.80/28 192.168.0.95/28
Customer 3 192.168.0.96/28 192.168.0.111/28
Customer 4 192.168.0.112/28 192.168.0.127/28
(base) C:\Users\DELL\Desktop\CN\Lab\DA3>
Python 3.8.5 64-bit (conda)
0 0
Connect
Server not selected
Ln 25, Col 1
Spaces: 4
UTF-8
CRLF
Python
Idle: indexing