Flow Control

Stop and Wait:

CODE:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <time.h>

#define FRAME_SIZE 128
#define ACK 1
#define NAK 0

// Function to introduce random errors
void induce_error(char *data)
{
    int r1 = rand() % 2;
    if (r1 == 1)
    {
        int r2 = rand() % strlen(data);
        data[r2] = (data[r2] == '0') ? '1' : '0';
    }
}

// Function to calculate CRC
void calculate_crc(char *data, char *crc_generator, char *crc_result)
{
    strcpy(crc_result, data);

    while (strlen(crc_result) >= strlen(crc_generator))
    {
        for (int i = 0; i < strlen(crc_generator); i++)
        {
            crc_result[i] = (crc_result[i] == crc_generator[i]) ? '0' : '1';
        }

        if (crc_result[0] == '0')
            strcpy(crc_result, crc_result + 1);
        else
            strcpy(crc_result, crc_result);
    }
}

// Function to convert ASCII string to binary string
void ascii_to_bin(char *input, char *binary)
{
    int i, j;
    for (i = 0; i < strlen(input); i++)
    {
```

```
        for (j = 7; j >= 0; j--)
        {
            binary[(i * 8) + (7 - j)] = ((input[i] & (1 << j)) ? '1' : '0');
        }
    }
    binary[i * 8] = '\0';
}

// Function to append zero bits
void append_zero(char *binary, int num_zeros)
{
    int len = strlen(binary);
    for (int i = 0; i < num_zeros; i++)
    {
        binary[len + i] = '0';
    }
    binary[len + num_zeros] = '\0';
}

// Function to send data packet
void send_packet(char *data, char *crc_result, int *ack)
{
    printf("Sending data packet: %s%s\n", data, crc_result);

    // Randomly introduce error
    induce_error(data);
    induce_error(crc_result);

    // Simulate receiver's response
    int r = rand() % 2;
    if (r == 0)
    {
        printf("Acknowledgement received: OK\n");
        *ack = ACK;
    }
    else
    {
        printf("Acknowledgement received: NAK\n");
        *ack = NAK;
    }
}

// Function to receive data packet
void receive_packet(char *packet, char *crc_generator, char *ack)
{
    // Simulate packet corruption
    int r = rand() % 2;
    if (r == 0)
    {
        printf("Received packet: %s\n", packet);

        // Check for corruption
        char crc_result[FRAME_SIZE];
        calculate_crc(packet, crc_generator, crc_result);
        if (strcmp(crc_result, "000") == 0)
        {
            printf("Packet is error-free.\n");
```

```
            *ack = ACK;
        }
        else
        {
            printf("Packet is corrupted.\n");
            *ack = NAK;
        }
    }
    else
    {
        printf("Received corrupted packet.\n");
        *ack = NAK;
    }
}

// Function to send file
void send_file(char *filename, char *crc_generator)
{
    FILE *file = fopen(filename, "r");
    if (file == NULL)
    {
        printf("Error opening file.\n");
        return;
    }

    char input[FRAME_SIZE];
    char binary[FRAME_SIZE * 8];
    char data[FRAME_SIZE];
    char crc_result[FRAME_SIZE];
    int ack;

    while (fgets(input, FRAME_SIZE, file))
    {
        // Convert ASCII to binary
        ascii_to_bin(input, binary);

        // Append zero bits
        append_zero(binary, strlen(crc_generator) - 1);

        // Generate CRC
        calculate_crc(binary, crc_generator, crc_result);

        // Send packet and receive acknowledgement
        ack = NAK;
        while (ack != ACK)
        {
            strncpy(data, binary, FRAME_SIZE - strlen(crc_generator) + 1);
            send_packet(data, crc_result, &ack);
            if (ack == ACK)
            {
                break;
            }
        }

        if (ack == NAK)
        {
            printf("Resending packet: %s%s\n", data, crc_result);
```

```
            send_packet(data, crc_result, &ack);
        }
    }

    // Send end-of-file flag
    printf("Sending end-of-file flag.\n");
    send_packet("", "", &ack);

    fclose(file);
}

// Function to receive file
void receive_file(char *filename, char *crc_generator)
{
    FILE *file = fopen(filename, "w");
    if (file == NULL)
    {
        printf("Error creating file.\n");
        return;
    }

    char packet[FRAME_SIZE];
    char ack = NAK;

    while (ack != ACK)
    {
        receive_packet(packet, crc_generator, &ack);
        if (ack == ACK)
        {
            break;
        }
        else
        {
            printf("Resending acknowledgement: NAK\n");
        }
    }

    while (strlen(packet) > 0)
    {
        // Extract data bits
        char data[FRAME_SIZE - strlen(crc_generator) + 1];
        strncpy(data, packet, FRAME_SIZE - strlen(crc_generator));
        data[FRAME_SIZE - strlen(crc_generator)] = '\0';

        // Convert binary to ASCII
        char output[FRAME_SIZE / 8];
        for (int i = 0; i < strlen(data) / 8; i++)
        {
            char byte[9];
            strncpy(byte, data + (i * 8), 8);
            byte[8] = '\0';
            output[i] = strtol(byte, NULL, 2);
        }

        fprintf(file, "%s", output);

        // Send acknowledgement
```

```
        ack = ACK;
        printf("Sending acknowledgement: OK\n");

        if (ack != NAK)
        {
            receive_packet(packet, crc_generator, &ack);
            if (ack == NAK)
            {
                printf("Resending acknowledgement: NAK\n");
            }
        }
    }

    fclose(file);
}

int main()
{
    srand(time(NULL));

    char crc_generator[] = "1011"; // CRC-4

    // Sender
    printf("Sender:\n");
    send_file("input.txt", crc_generator);

    printf("\n-----------------\n\n");

    // Receiver
    printf("Receiver:\n");
    receive_file("output.txt", crc_generator);

    return 0;
}
```

OUTPUT:

```
PROBLEMS  OUTPUT  TERMINAL  COMMENTS  DEBUG CONSOLE                                   Code

PS C:\Users\Admin\Desktop\cn> cd "c:\Users\Admin\Desktop\cn\" ; if ($?) { gcc cn.c -o cn } ; if ($?) { .\cn }
Sender:
Sending data packet: 0100100001100101011011000110110001101111001011000010000001010111011011101110011001101100011001000010000100001010000001
Acknowledgement received: NAK
Sending data packet: 0100100001100101011011000110110001101111001011000010000001010111011011101110011001101100011001000010000100001010000011
Acknowledgement received: OK
Sending data packet: 0101010001101000011010010111001100100000011010010111001100100000011000010010000001110100011001010111001101110100001011000001011
Acknowledgement received: OK
Sending data packet: 0101001101100101011011100110010001101001011011100110011100100000011001000110000101110100011000010010000001101111011001100100
```

PURVA SHARMA

21BCE0169

C code for implementing the Go-Back-N and Selective Repeat protocols for reliable data transmission over a network.

```c
#include <stdio.h>
#include <stdbool.h>

#define WINDOW_SIZE 4
#define FRAME_COUNT 8

// Go-Back-N protocol
void gobackn() {
    int base = 0;
    int nextseqnum = 0;
    int i;

    // Simulating the network layer sending frames
    int frame_sent[FRAME_COUNT] = {0, 1, 2, 3, 4, 5, 6, 7};

    // Simulating the network layer receiving ACKs
    bool ack_received[FRAME_COUNT] = {false};

    while (base < FRAME_COUNT) {
        // Sending frames within the window
        for (i = base; i < base + WINDOW_SIZE && i < FRAME_COUNT; i++) {
            if (!ack_received[i]) {
                printf("Sending frame %d\n", frame_sent[i]);
            }
        }

        // Simulating the network layer receiving ACKs
        for (i = base; i < base + WINDOW_SIZE && i < FRAME_COUNT; i++) {
            if (!ack_received[i]) {
                // Simulating the ACK for frame i received
                printf("Received ACK for frame %d\n", frame_sent[i]);
                ack_received[i] = true;
                nextseqnum = i + 1;
            }
        }

        // Moving the window if the ACK for the base frame is received
        if (ack_received[base]) {
            base++;
        }
    }
}

// Selective Repeat protocol
void selectiverepeat() {
    int base = 0;
    int nextseqnum = 0;
    int i;
```

```
    // Simulating the network layer sending frames
    int frame_sent[FRAME_COUNT] = {0, 1, 2, 3, 4, 5, 6, 7};

    // Simulating the network layer receiving ACKs
    bool ack_received[FRAME_COUNT] = {false};

    while (base < FRAME_COUNT) {
        // Sending frames within the window
        for (i = base; i < base + WINDOW_SIZE && i < FRAME_COUNT; i++) {
            if (!ack_received[i]) {
                printf("Sending frame %d\n", frame_sent[i]);
            }
        }

        // Simulating the network layer receiving ACKs
        for (i = base; i < base + WINDOW_SIZE && i < FRAME_COUNT; i++) {
            if (!ack_received[i]) {
                // Simulating the ACK for frame i received
                printf("Received ACK for frame %d\n", frame_sent[i]);
                ack_received[i] = true;
            }
        }

        // Moving the window for the frames that have been acknowledged
        while (base < FRAME_COUNT && ack_received[base]) {
            base++;
        }
    }
}

int main() {
    printf("Go-Back-N Protocol:\n");
    gobackn();

    printf("\nSelective Repeat Protocol:\n");
    selectiverepeat();

    return 0;
}
```

In this code, the gobackn function implements the Go-Back-N protocol, and the selectiverepeat function implements the Selective Repeat protocol. Both protocols simulate the sending and receiving of frames and ACKs.

You can adjust the WINDOW_SIZE and FRAME_COUNT constants according to your requirements. The code prints the actions taken at each step, such as sending frames and receiving ACKs, to demonstrate the protocol's behaviour.

Note that this code only simulates the protocols and doesn't include actual network communication. It serves as a simplified example to illustrate the concepts of Go-Back-N and Selective Repeat.

OUTPUT:

```
PROBLEMS  OUTPUT  TERMINAL  COMMENTS  DEBUG CONSOLE                    Code

PS C:\Users\Admin\Desktop\CN> cd "c:\Users\Admin\Desktop\CN\" ; if ($?) { gcc goBack.c -o goBack } ; if ($?) { .\goBack }
Go-Back-N Protocol:
Sending frame 0
Sending frame 1
Sending frame 2
Sending frame 3
Sending frame 3
Received ACK for frame 0
Received ACK for frame 1
Received ACK for frame 2
Received ACK for frame 3
Sending frame 4
Sending frame 5
Sending frame 6
Sending frame 7
Received ACK for frame 4
Received ACK for frame 5
Received ACK for frame 6
Received ACK for frame 7
PS C:\Users\Admin\Desktop\CN>
```

AIM
Validate the class of an IP Addressing Schemes using Python

<u>Network Address and Mask:</u>
Example: Given IP address 132.6.17.85 and default class B mask, find the beginning address (network address).
Solution: The default mask is 255.255.0.0, which means that the only the first 2 bytes are preserved and the other 2 bytes are set to 0. Therefore, the network address is 132.6.0.0.

ip.py

```python
def validate_ip_class(ip_address):

    octets = ip_address.split('.')

    first_octet = int(octets[0])


    if 1 <= first_octet <= 126:

        return 'Class A'

    elif 128 <= first_octet <= 191:

        return 'Class B'

    elif 192 <= first_octet <= 223:

        return 'Class C'

    elif 224 <= first_octet <= 239:

        return 'Class D'

    elif 240 <= first_octet <= 255:

        return 'Class E'

    else:

        return 'Invalid IP address'
```

```python


def calculate_network_address(ip_address, subnet_mask):

    ip_octets = ip_address.split('.')

    mask_octets = subnet_mask.split('.')


    network_octets = []

    for i in range(4):

        network_octets.append(str(int(ip_octets[i]) & int(mask_octets[i])))


    network_address = '.'.join(network_octets)

    return network_address


# Example usage

ip_address = input("Enter the IP address: ")

subnet_mask = input("Enter the subnet mask: ")


ip_class = validate_ip_class(ip_address)

print("IP Address class:", ip_class)


network_address = calculate_network_address(ip_address, subnet_mask)

print("Network Address:", network_address)
```

```
PROBLEMS  DEBUG CONSOLE  TERMINAL  COMMENTS

python -u "/home/matlab/PRAC/ip.py"
harith_is_cool: python -u "/home/matlab/PRAC/ip.py"
Enter the IP address: 132.6.17.85
Enter the subnet mask: 255.255.0.0
IP Address class: Class B
Network Address: 132.6.0.0
harith_is_cool:
```

Classless Addressing

Some values calculated in subnetting:

1. Number of subnets: Given bits for mask – No. of bits in default mask

2. Subnet address: AND result of subnet mask and the given IP address

3. Broadcast address: By putting the host bits as 1 and retaining the network bits as in the IP address

4. Number of hosts per subnet: 2(32 – Given bits for mask) – 2

5. First Host ID: Subnet address + 1 (adding one to the binary representation of the subnet address)

16. Last Host ID: Subnet address + Number of Hosts

Example: Given IP Address – 172.16.0.0/25, find the number of subnets and the number of hosts per subnet. Also, for the first subnet block, find the subnet address, first host ID, last host ID and broadcast address.

Solution: This is a class B address. So, no. of subnets = 2(25-16) = 29 = 512.

No. of hosts per subnet = 2(32-25) – 2 = 27 – 2 = 128 – 2 = 126

For the first subnet block, we have subnet address = 0.0, first host id = 0.1, last host id = 0.126 and broadcast address = 0.127

ip.py ip2.py

ip2.py > ...

```python
import ipaddress

def calculate_subnet(ip):
    ip_network = ipaddress.ip_network(ip)

    subnet_bits = ip_network.prefixlen
    subnet_count = 2 ** (32 - subnet_bits)
    hosts_per_subnet = 2 ** (32 - subnet_bits) - 2

    first_subnet = next(ip_network.subnets(new_prefix=subnet_bits))
    subnet_address = str(first_subnet.network_address)
    first_host = str(first_subnet.network_address + 1)
    last_host = str(first_subnet.network_address + hosts_per_subnet)
    broadcast_address = str(first_subnet.broadcast_address)

    return subnet_count, hosts_per_subnet, subnet_address, first_host, last_host, broadcast_address


ip_input = input("Enter the IP adress:")
subnet_count, hosts_per_subnet, subnet_address, first_host, last_host, broadcast_address = calculate_subnet(ip_input)

print("Number of subnets:", subnet_count)
print("Number of hosts per subnet:", hosts_per_subnet)
print("Subnet address:", subnet_address)
print("First host ID:", first_host)
print("Last host ID:", last_host)
print("Broadcast address:", broadcast_address)

```

```
PROBLEMS  DEBUG CONSOLE  TERMINAL  COMMENTS

python -u "/home/matlab/PRAC/ip2.py"
harith_is_cool: python -u "/home/matlab/PRAC/ip2.py"
Enter the IP adress:172.16.0.0/25
Number of subnets: 128
Number of hosts per subnet: 126
Subnet address: 172.16.0.0
First host ID: 172.16.0.1
Last host ID: 172.16.0.126
Broadcast address: 172.16.0.127
harith_is_cool:
```