Instructions:

- Complete the project in groups of 2-3.
- The grade for this project is weighted as follows:
  - 50% Implementation
  - 20% Performance Tests
  - 30% Write-up

Due Date: December 7, 1999 at the beginning of class.

Objective: Implementation and evaluation of the Go-back-N ARQ error control scheme.

Problem Description:

1. Add an error control scheme to the sliding window flow control mechanism that you implemented for Programming Assignment #2.

2. Sender and Receiver use the Go-back-N ARQ scheme for error control.

3. The error control scheme must deal with the following four types of errors:
   - lost data packets,
   - damaged data packets,
   - lost ACKs or NAKs,
   - damaged ACKs or NAKs.

4. The errors are artificially introduced at the Sender and Receiver. This allows you to control the rate of errors. Errors are introduced by replacing all `read()` and `write()` socket calls with “unreliable” socket calls. The following `read()` call:

   ```c
   read(sock, buff, buffsize);
   ```

   will be replaced with an `unreliable_read()` function call:

   ```c
   unreliable_read(sock, buff, buffsize);
   ```

   `write()` system calls will be replaced with `unreliable_write()` calls in a similar fashion.

5. 'C' code for the `unreliable_read()` function call is as follows:

   ```c
   int unreliable_read( int sock, char *buff, int buffsize ) {
   if( Rand() < P ) {
        /* Damaged Packet */
        return( DMGD_PKT );
   } else {
   ```
/* Successful Read Call */
return( read( sock, buff, buffsize ) );
}
}

where Rand() is a randomly generated number such that $0 \leq \text{Rand()} \leq 1$; $P$ is the probability of a damaged data packet, ACK, or NAK, and is fixed such that $0 \leq P \leq 1$; DMGD_PKT is a flag indicating that a damaged data packet, ACK, or NAK was received.

6. 'C' code for the unreliable_write() function call is as follows:

```c
int unreliable_write( int sock, char *buff, int buffsize ) {
    if( Rand() < Q ) {
        /* Lost Packet */
        return( buffsize );
    } else {
        /* Successful Write Call */
        return( write( sock, buff, buffsize ) );
    }
}
```

where Rand() is again a random number such that $0 \leq \text{Rand()} \leq 1$; $Q$ is the probability of a lost data packet, ACK, or NAK, and is fixed such that $0 \leq Q \leq 1$.

7. The sequence space and window size are identical for both sender and receiver. The sequence space is set to 8, while the window size is fixed at $W = 7$.

8. All data packets are fixed-length with a size of 100 bytes each. Select the size and format of the ACK and NAK packets any way you like.

Your Task:

1. Implement the Go-back-N ARQ error control scheme on Unix workstations using stream sockets. The implementation involves:

   (a) C/C++ code that implements the Sender process. Usage for the Sender process is as follows: Sender <dest_host> <dest_port> <spd_up> <P> <Q> <timeout> <max_time>, where:
       * dest_host is the name of the host with the Receiver process,
       * dest_port is a well-known port at which the Receiver accepts connections,
       * spd_up is the speedup factor described in Assignment #2,
       * P is the probability of a damaged data packet, ACK, or NAK as described above.
       * Q is the probability of a lost data packet, ACK, or NAK as described above.
       * max_time is the maximum running time in seconds.

   (b) C/C++ code that implements the Receiver process. Usage for the Receiver process is as follows: Receiver <port> <spd_up> <P> <Q> <timeout> <max_time>, where:
       * port is the port number at which the Receiver accepts connections,
* $\text{spd\_up}$, P, Q, timeout, and max\_time are as described in 1(a) above.

2. Measure the performance of your Go-back-N ARQ implementation with a set of experiments which are done over long periods of time. Please determine:

   - average time to recover a lost packet,
   - average time to recover a damaged packet,
   - overhead introduced due to packet retransmissions (i.e., the total number of retransmissions).

These quantities should be measured for different values of $\text{spd\_up}$, P, Q, and timeout. You may use graphs as necessary that show the impact of each parameter on system performance. Note that the values of P and Q should be the same for Sender and Receiver.

3. Deliver a write-up which includes:

   - your implementation approach,
   - a description of your test strategy that shows the correctness of your implementation,
   - performance tests described above,
   - a list of known bugs,
   - a summary,
   - lessons learned.

The write-up should not exceed 3 pages (excluding plots).

4. Submit your write-up has hard copy. Submit your source code using the online instructions on the web page.

Hints:

1. Use your code from Assignment #2 as a starting point for implementing both Sender and Receiver.

2. See the hints regarding fixed packet size and UNIX random number generators from Assignment #2.