For this course project, please choose *a group of up to four members*. Each group member will receive the same grade. So choose your group members carefully.

CPSC 3780
Course Project
Due: TBA
Total marks: 50

This project will give you an introduction on socket programming, which is the foundation of the *client-server computing model*. Due to the limited class time in one semester, we cannot cover many topics at the application layer. It is thus hoped that this project will practice you at this layer, providing complementary materials to our course topics.

Your task is to transfer big flat ASCII file(s) from the server (A) to the client (B). Each time only one line in the file is transferred. However, each transfer can only handle a frame of at most 64 characters. If one line contains too many characters, you need to chop it down into several frames and transfer each of them. You also need to simulate, should an error happen, retransmissions.

There are two simplex channels you simulate between A and B. While A's task is to send an ASCII file to B using one simplex channel, B uses the other channel to tell A whether a transfer is correct or not and a retransmission is necessary.

A. Project background

Sockets are a mechanism for exchanging data between processes. The processes can be on the same host or on different hosts via a network connection. Once a socket connection is established, data can be exchanged in both directions until one of the two parties closes the connection.

Usually the process which provides a service (to be made concrete later on) is called the *server* while the process which requests the service is called the *client*. Both the server and client are implemented as processes.

A typical client-server connection consists of the following steps. First, the server creates a listening socket and waits for connection attempts from clients. The client creates a socket on its side, and attempts to connect with the server. The server then accepts the connection. After that, the data exchange begins. Once all data has been passed through the socket connection, either party can close the connection. This is summarized in the table as follows.

Server	Client
1. Establish a listening socket and wait for connections from	

the client.	
	2. Create a client socket and attempt to connect to server.
3. Accept the client's connection attempt.	
4. Send and receive data.	4. Send and receive data.
5. Close the connection.	5. Close the connection.

Read the tutorial posted on the course website for socketing.

B. Project description

In the following, A refers to the server while B refers to one client. Essentially, A needs to set up two sockets, one for transferring data while the other for accepting ACK/NAK. So you need to change the general procedure in the above table between the server and client for this purpose.

Before sending out a frame, A calculates the sum of the ASCII values of all characters in it and generates an even parity bit. A needs to send the parity bit, along with the frame, to B. For every five frames that A sends out, there is one frame that has a bad parity bit. Such a frame could be the first frame or could be the fifth frame. You decide this using a random number. For B, upon receiving a frame from A, it needs to check the parity bit with the frame. If the parity bit is good, B sends an ACK to A. Otherwise, it sends an NAK to B and asks for a retransmission of the frame.

Once the client receives a correct entire line, it needs to print it out on its screen.

The data file(s) in your program should be ASCII based and contain more than 1000 lines. Some lines can have only a few characters while others can have, say, more than 500 characters, to simulate the deassembly and reassembly operations. One possible way to do this is to download document(s) from the Internet and then reformat them. Note that the file content is not important. When you reformat a file, please add a line number to each line such that the marker can trace it.

Both A and B should be floating, in the sense when A is installed on *host1*, through *host1* B can connect and get service from A. When A changes to *host2*, then B needs to make a request to *host2*. So when B makes a request to connect to A, its connection request should contain the host information of A.

You are required to design the protocol between A and B. There are two scenarios for you to design and implement.

(S1) There is one A and one B. B makes a request and A serves the request. This process continues until a data file is finished transferring.

(S2) There are multiple big ASCII files to transfer from A to multiple Bs. i.e., there are one server and multiple clients (up to 5). B makes a request for transferring a particular data file (suppose that the data file names are known to all clients in advance) and the server responds it by creating a new thread. The new thread is responsible for transferring the data file to the client. The server then goes back and waits for requests from other clients. There could be that multiple Bs are transferring data files at the same time.

C. Project tasks and submission

(1) Read the tutorial on the course website.
(2) Design a solution to each of the above two scenarios and implement it using C++.
(3) For each scenario, write a small tutorial (hardcopy) as how to compile and use it.
You can use the *make file* posted on the course website. Your marker will check your program based on your tutorial.
(4) Write a summary (hardcopy and at most three (3) pages) of your design and implementation for each scenario, such as the protocol, e.g., the data format you have designed, what difficulties you have encountered and how you have handled them, special data structures you have used, etc. You can put two summaries together into one document.
(5) Submit your program (all the source code files, make file, etc.) in a CD. On the CD, structure your files as follows.

In the root directory, you create two sub directories, called *S1* and *S2*, for Scenario 1 and Scenario 2, respectively. In each sub directory, put your test data file(s) there and create two sub directories called *server* and *client*, which contains that source codes for the server and client, respectively. For subdirectories *server* and *client*, copy the corresponding makefile there.

D. Project hints

(1) Finish Part (S1) first. Part (S2) is just a generalization of Part (S1) by including threading in your program. For multithreading in C++, please see the link on the course website.
(2) Design the data format first, and then create functions that can operate on it.