Ryan Lynch, z5416305
Version used: Python 3.9
Code is literally all in receiver and sender along with generated output files as per the spec.
No code was taken from the internet or books.
This code works in every circumstance which I could feasibly test. (may not end the receiver running on several dropped FIN packets)

Reciever:

Data structures:

The receiver globally (inside the class) stores everything used by the other functions (input values, logging values as well as packetBuffer (array of packets which couldnt be immediately added due to being a future packet) and lastACKNo (stores the next ACK expected).

Overall:

I was considering multithreading the receiver (a listener and a processor) which wouldve been a more effective design, however due to time constraints and what I'd already written, it is a single thread.
It has 3 main sections, the opening section which listens for a SYN and sends the initial ack (+1 seq num) back when it receives it (this also initialises the starttime for the program.
The Main section which receives packets and responds appropriately, either writing to the file, finishing or resetting depending on the packet.
The closing section waits two seconds after receiving a FIN packet, listening for more packets before logging the last few things and closing the program.
The only other important thing here is the processBuffer function which is called when a packet is successfully received and immediately added. This function goes through the list of future out of order packets and recursively adds them to the file while removing them from the buffer.

Sender:

Data structures:

As the sender is multithreaded, all shared data goes into the control block which is then locked with the control lock. These things are the seqNo and the sentPackets list.
The other global parameters are basically the same as reciever, with bad ACKs for out of order ACKs.
The seqNo tracks the next sequence number to send.
sentPackets is a list of sent packets, formatted as a list where each entry is another list of the format:
([time packet is sent, seqNo which it tries to match, actual message sent (includes seqno and type), number of times retransmitted, the length of the content, the string of its type for logging])

Overall:

The main program has start and close which just perform rudimentary starts and close operations along with the 3 threads, the main one (ptp send), listen and timeOut.
timeOut is the simplest. While the program is running, it constantly checks the oldest packet sent (always front of sentPackets) and resends it if it has timed out.
Send is actually the next simplest, It initially copies the file to send into an array of nextPackets which are 1000 bytes each (MSS) and then it idles until the first syn packet has been removed from packetsSend (this marks it has been ACK'ed). Then, while there are more packets to send, whenever there are less than the window packets sent (sentPackets has len < window), it will send another packet.
Listen is the beefiest, as it handles all of the out of order acks. While the program is running, listen waits for a message and then processes it. The processing starts by removing any packets from sentPackets which have an expected ACK which is older than the new received ACK. If the packet is the next expected one, it removes the first entry into sentPackets and then loops to check if the next packet in the list has also been ACK'd but it was out of order, removing it if it was. If the ACK wasnt the next expected one, we add it to the out of order acks list (badACKS) and if there is 3 of the same ACK there, it resends the required packet.