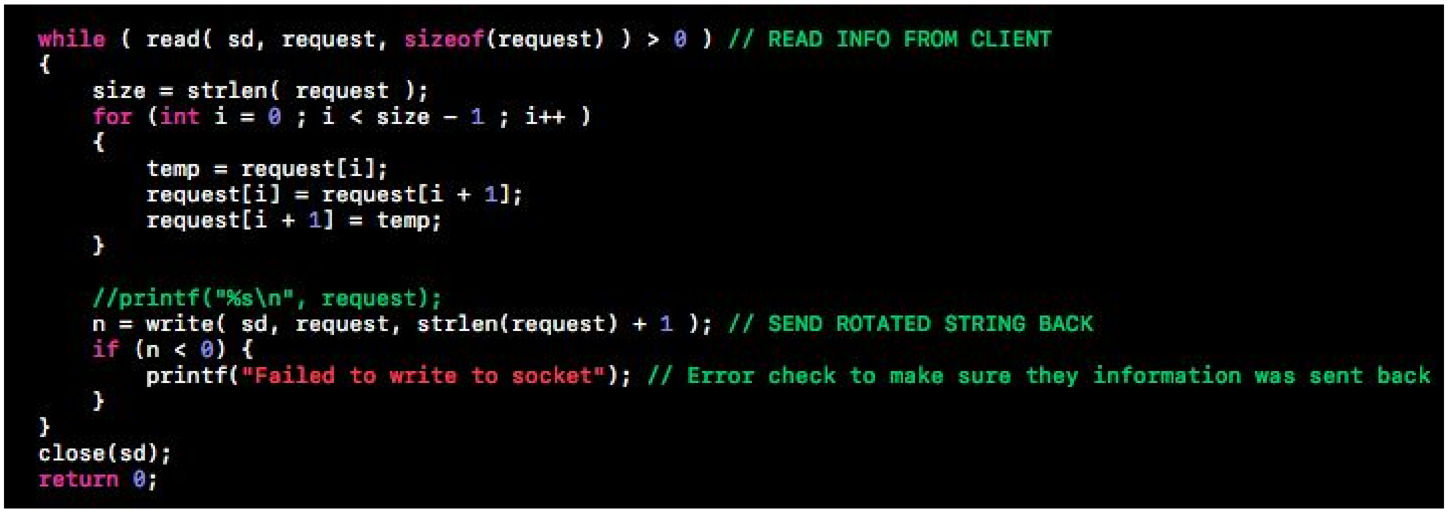
I started out with the client3 and sever3 code that was provided. Let's take a look at the server code first. The first thing the server needs to do is bind to a port. In this case the "claim_port" method is called. I used "SOCK_STREAM" which means we are working on TCP, "AF_INET" which means we are using IPv4. We perform various checks as we try to bind to socket 8675. If we are successful we print out "Bind to port 8675 succeeded". This let's us know that the server is up and running on our local machine on the port we wanted. The server then sits there passively waiting for a connection. The following code is especially relevant to this assignment.

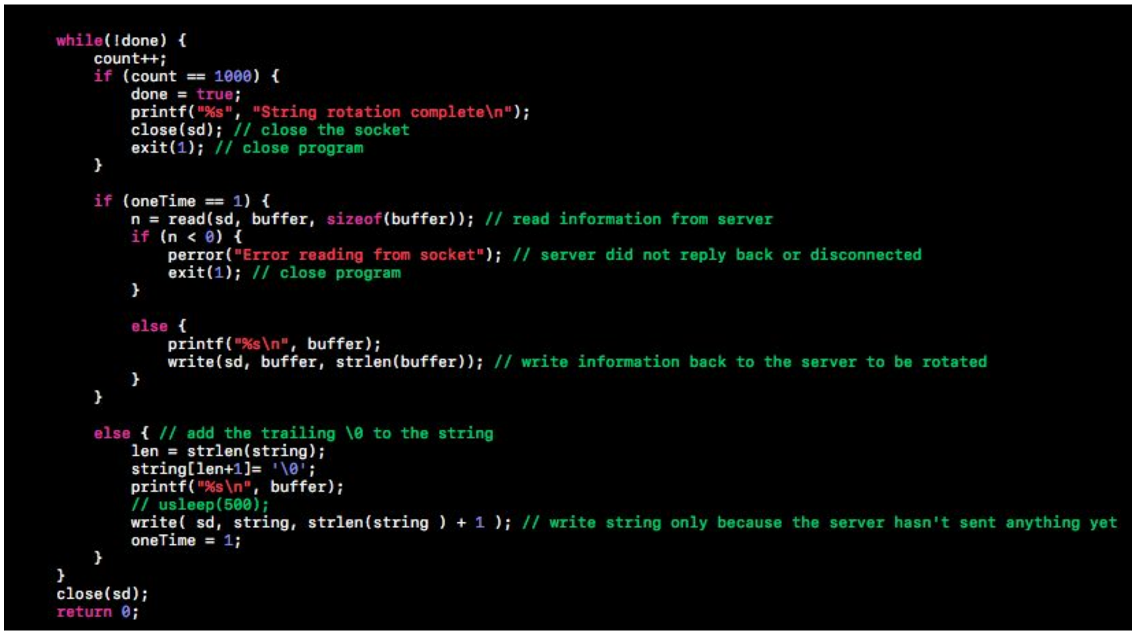


```
while ( read( sd, request, sizeof(request) ) > 0 ) // READ INFO FROM CLIENT
{
    size = strlen( request );
    for (int i = 0 ; i < size - 1 ; i++ )
    {
        temp = request[i];
        request[i] = request[i + 1];
        request[i + 1] = temp;
    }

    //printf("%s\n", request);
    n = write( sd, request, strlen(request) + 1 ); // SEND ROTATED STRING BACK
    if (n < 0) {
        printf("Failed to write to socket"); // Error check to make sure they information was sent back
    }
}
close(sd);
return 0;
```

This while loop executes as long as the client is sending us something. The string is then rotated by performing a series of swaps. Then we write the rotated string back to the client. If the value of n is less than zero then we know the writing failed and we print out an error. The server remains open in this scenario.

Now we take a look at the client stub. First the client stub checks that it was given two arguments a server name and a port number. If it is unable to connect to that server or port it will print out an error message. If the connection is successful the client will print Connected to server + the server name provided by the user. The following code fragments is especially relevant to the assignment.



```
while(!done) {
    count++;
    if (count == 1000) {
        done = true;
        printf("%s", "String rotation complete\n");
        close(sd); // close the socket
        exit(1); // close program
    }

    if (oneTime == 1) {
        n = read(sd, buffer, sizeof(buffer)); // read information from server
        if (n < 0) {
            perror("Error reading from socket"); // server did not reply back or disconnected
            exit(1); // close program
        }

        else {
            printf("%s\n", buffer);
            write(sd, buffer, strlen(buffer)); // write information back to the server to be rotated
        }
    }

    else { // add the trailing \0 to the string
        len = strlen(string);
        string[len+1]= '\0';
        printf("%s\n", buffer);
        // usleep(500);
        write( sd, string, strlen(string ) + 1 ); // write string only because the server hasn't sent anything yet
        oneTime = 1;
    }
}
close(sd);
return 0;
```

The loop executes 1,000 string rotations. The initial string is hardcoded as “ABCDEF” into the client stub. After all string rotations have been completed the client prints “String rotation complete”, the socket is closed, and the program exits. There is also an error check in there in the event the client is unable to write to the socket. A static variable “oneTime” is used to mark the first execution of the loop by the client. In this event the server just needs to write the string to the server. Every other time the client reads the string sent from the server, prints it, and then sends it back to be rotated again.