Inicialmente a ideia proposta foi tratar os dados com structs, mas depois da aula com o monitor na sala de aula, ficou muito mais claro que seria melhor trabalhar com matriz. Dessa forma a ideia que pensei foi armazenar os equipamentos em matrizes que teriam basicamente 3 colunas e 5 linhas. 3 colunas pois teria que o espaço para os dois algarismos dos identificadores dos equipamentos e dos sensores, e um terceiro apenas para o "\n". 5 linhas pois a primeira obrigatoriamente teria o identificador do equipamento e também os identificadores dos possíveis 4 sensores que poderíamos associar a tal equipamento.

Criei também variáveis que armazenam de forma global a quantidade de sensores instalados em cada equipamento, de forma a ficar mais fácil de manipular.

Como forma de lógica agora para tratar os dados que eram passados como forma de mensagem via terminal após a conexão, foi criada inicialmente uma função chamada "direcionaOperacao" que recebe justamente o que é passado pelo usuário como forma de mensagem, sendo esta o buf. Pegamos no início dessa função o buf, e quebramos para formar um vetor, afim de ficar melhor de tratar os dados. O comando que fica logo no início das mensagens como "add sensor 01 in 01" fica logo na primeira posição e consegui pegá-lo usando o "strtok". Depois disso tem uma verificação para ver se os equipamentos passados estão certos, para o tratamento de erros, assim como o direcionamento para outras funções a depender do que foi passado. Vale lembrar que o comando KILL fecha a conexão, e também caso o user digite alguma mensagem errada como "ad sensor 01 in 01", ao invés de "add", vai fechar a conexão também, pois o server não vai entender o que está sendo solicitado.

Na função de adicionarSensores a um equipamento, foi inicialmente pego sensor que se quer trabalhar para fazer a mesma lógica para cada um só mudando as variáveis que se trabalha, dependendo de qual equipamento se passa. Pegamos os sensores da mensagem e os guardamos para fazermos as operações. Antes de tudo faz a verificação se os sensores passados não são inválidos, como pode ser visto abaixo:

```
while(auxSensor < numeroDeSensores) { // verifica se os sensores
    if (strncmp(sensor[auxSensor], "01", 2) != 0 &&
        strncmp(sensor[auxSensor], "02", 2) != 0 &&
        strncmp(sensor[auxSensor], "03", 2) != 0 &&
        strncmp(sensor[auxSensor], "04", 2) != 0) {
            strcat(auxSensorInvalidoMensagem, "invalid sensor");
            break;
    }
    auxSensor++;
}
```

E também verificamos se a quantidade de sensores que a estação remota trata máxima(15) não será ultrapassada. Caso esses erros não sejam "acionados", percorremos cada sensor que foi pego da mensagem comparando com os que existem dentro da matriz do equipamento, para fazer as operações de inserção e comparação, até mesmo para retornar mensagens de "já existe" ou de "sensor adicionado". A mesma lógica é aplicada para o sensor 02, 03 e 04 também e por fim, a depender das variáveis de mensagem que foram criadas, caso elas ao decorrer do percurso da lógica, caso tenham sido incrementadas com algo, vai retornar a informação referente ao sensor e ao equipamento. Podendo até mesmo retornar mensagens de que sensor já existe e que sensor foi adicionado, num mesmo retorno.

Já na função de listarSensores, foi um pouco mais fácil de lidar pelo fato de a quantidade de sensores em cada equipamento está sendo manipulada globalmente, fazendo com que apenas seja necessária uma validação para ver se tem algum sensor no equipamento, pois caso não tenha retorna para o cliente "none". E caso tenha, vamos retorná-los para o cliente como uma sequência de IDs.

A função de removerSensor percorre as matrizes da mesma forma que na função de adicionar, e fazendo as devidas verificações de se os sensores passados existem. Porém a principal diferença

é que agora temos que remover os sensores passados, e para isso foi criada uma convenção de ao remover tal sensor, substituir seu ID por ‘99’. Dessa forma, depois disso, foi feita uma sequência de condicionais para passar esse ‘99’ para a última posição do vetor, além do mais como ao remover vai ter a operação de decrementar em uma unidade a quantidade de sensores do respectivo equipamento, temos que “empurrar” esse sensor a ser removido para o final afim de que quando tiver a função list ou qualquer outra, este seja ignorado. Tal lógica pode ser observada abaixo:

```
int x = 0;
int y = 1;
int sensorExiste = 0;
if (strncmp(equipamento, "01", 2) == 0) {
    while(x < numeroDeSensores) {
    y = 1;
        while(y < 5) {
            if (strncmp(primeiroEquipamento[y], sensor[x], 2) == 0) { //esta função ordena os sensores deixando o apagado por ul
                sensorExiste++;
                strcpy(primeiroEquipamento[y], "99"); //99 valor que nao pode ser lido
                if(strcmp(primeiroEquipamento[y+1], "99" )!=0 || strcmp(primeiroEquipamento[y+1],"")!=0) {
                    strcpy(primeiroEquipamento[y],primeiroEquipamento[y+1]);
                    strcpy(primeiroEquipamento[y+1], "99");
                    if(strcmp(primeiroEquipamento[y+2], "99") !=0 || strcmp(primeiroEquipamento[y+2], "") !=0) {
                        strcpy(primeiroEquipamento[y+1], primeiroEquipamento[y+2]);
                        strcpy(primeiroEquipamento[y+2], "99");
                            if(strcmp(primeiroEquipamento[y+3], "99") != 0 || strcmp(primeiroEquipamento[y+3], "") != 0) {
                            strcpy(primeiroEquipamento[y+2], primeiroEquipamento[y+3]);
                            strcpy(primeiroEquipamento[y+3], "99");
                            }
                    }
                }
                strcat(auxRemoverSensorExistenteMensagem, sensor[x]);
                strcat(auxRemoverSensorExistenteMensagem, " ");
                quantidadeSensoresPrimeiroEquipamento--; //decai uma valor para a quantidade de sensores do equipamento que está
            }
            y++;
        }
```

OBS: houve uma dificuldade para fazer essa operação, além do mais não achei nenhuma função pronta para isso em C, dessa forma tive que fazer “na mão”, diferente do que poderia ser feito em JavaScript por exemplo, usando a função slice() que corta posições de dentro de um array.

Por fim, a função lerSensor que é acionada com o comando “read” faz com que retorne alguns valores aleatórios para serem exibidos como valores das variáveis que estes sensores medem. Desta forma, a lógica de percorrer as posições da matriz e os sensores passados na mensagem no terminal é a mesma das outras funções, porém a principal diferença é que caso exista este sensor dentro da matriz, vai imprimir o valor aleatório gerado usando a função rand() do C, e caso não exista vai retornar dizendo que tal sensor passado não existe na matriz do referente equipamento. Tal lógica pode ser analisada abaixo:

```
if (strncmp(equipamento, "04", 2) == 0) {
    while(x < numeroDeSensores) {
    y = 1;
        while(y < quantidadeSensoresQuartoEquipamento + 1) {
            if (strncmp(quartoEquipamento[y], sensor[x], 2) == 0) {
                sensorExiste++;
                float numeroAleatorio = rand() %10;
                char numeroAleatorioChar[6];
                sprintf(numeroAleatorioChar, "%.2f", numeroAleatorio);

                strcat(auxLerMensagem, numeroAleatorioChar);
                strcat(auxLerMensagem, " ");
            }
            y++;
        }
        if (!sensorExiste) {
            strcat(auxLerSensorNaoInstaladoMensagem, "sensor(s) ");
            strcat(auxLerSensorNaoInstaladoMensagem, sensor[x]);
            strcat(auxLerSensorNaoInstaladoMensagem, " ");
        }
```