CIS25 C++ Wireless Signal Database

By: Kirubel Essubalew

Data Members (all public):

- **double power - Signal power in watts**
- **double frequency - Frequency in MHz**
- **double distance - Distance in meters**
- **string deviceType - "WiFi", "Bluetooth", "Radio"**

Constructors:

- **Default: Sets safe default values (power=0.0, frequency=0.0, distance=0.0, deviceType="Unknown")**
- **Parameterized: WirelessSignal(double p, double f, double d, string type) - Sets all values immediately**

WirelessSignal - calculateSignalStrength()

Calculate how strong the signal is at a given distance

Code:

```
double calculateSignalStrength() {
    if (distance == 0) return power;
    return power / (distance * distance);
}
```

This is similar to the inverse square law which explains how intensity decreases as you move away from the source which is what signals do because they travel away from the source that created them.

WirelessSignal - saveToFile()

This method writes signal data to a file for storage

File Format:

- **deviceType power frequency distance**

Example Output:

- **WiFi 0.1 2400 10**

Code:

```
void saveToFile(ofstream& file) {
    file << deviceType << " " << power << " " << frequency << " " << distance << endl;
}
```

WirelessSignal - loadFromFile()

This method reads signal data from a file and also reads space-separated values in order

Code:

```
void loadFromFile(ifstream& file) {
    file >> deviceType >> power >> frequency >> distance;
}
```

SignalDatabase Class - Data Members & Constructor

Start of new class so there are now different data variables

Data Members (all public):

- **WirelessSignal* signals - Dynamic array of signals**
- **int currentSize - Current number of signals**
- **int maxSize - Maximum capacity**

There is a parameterized constructor for this class but what's new is the Destructor which helps clean up memory whenever I create a object that uses dynamic memory

Destructor code:

```
~SignalDatabase() {

    delete[] signals;  // Clean up memory

}
```

SignalDatabase - sortByFrequency()

This method sorts all signals by frequency (required for binary search)

Code: —------>

```
void sortByFrequency() {
    for (int i = 0; i < currentSize - 1; i++) {
        for (int j = 0; j < currentSize - i - 1; j++) {
            if (signals[j].frequency > signals[j + 1].frequency) {
                // Swap signals
                WirelessSignal temp = signals[j];
                signals[j] = signals[j + 1];
                signals[j + 1] = temp;
            }
        }
    }
}
```

How it Works:

- **Compare adjacent signals**
- **Swap if left > right frequency**
- **Repeat until all signals are in order**

This sorts each signal from lowest to highest frequency and gets it

SignalDatabase - findSignalByFrequency()

This method quickly finds a signal with a specific frequency using a BINARY SEARCH

Steps:

- **Start with middle signal**
- **If middle frequency = target → Found!**
- **If middle < target → Search right half**
- **If middle > target → Search left half**

Repeat until found or no more signals

Return Value:

Index number if found

-1 if not found

Code —>

```
int findSignalByFrequency(double targetFreq) {
    int left = 0;
    int right = currentSize - 1;

    while (left ≤ right) {
        int mid = left + (right - left) / 2;

        if (signals[mid].frequency == targetFreq) {
            return mid;  // Found the signal
        }
        else if (signals[mid].frequency < targetFreq) {
            left = mid + 1;
        }
        else {
            right = mid - 1;
        }
    }

    return -1;  // Signal not found
}
```

SignalDatabase - saveToFile()

Save entire database to a text file

File Format:

3 ← Number of signals

WiFi 0.1 2400 10 ← Signal 1

Bluetooth 0.001 2450 5 ← Signal 2

Radio 50 101.5 1000 ← Signal 3

The method write signal count first then write each signal using signal.saveToFile()

SignalDatabase - loadFromFile()

Load entire database from a text file (preferably from the file you saved)

Process:

- **Read signal count first**
- **Reset currentSize to 0**
- **Create temporary signal for each entry**
- **Load signal data and add to database**

Also

- **Checks if file opened successfully**
- **Won't exceed maxSize capacity**

SignalDatabase - findSignalsInPowerRange()

Find all signals within a specific power range and those parameters are

- **double minPower - Minimum power threshold**
- **double maxPower - Maximum power threshold**

Code —>

```
void findSignalsInPowerRange(double minPower, double maxPower) {
    cout << "Signals with power between " << minPower << " and " << maxPower <<
    bool found = false;

    for (int i = 0; i < currentSize; i++) {
        double power = signals[i].power;
        if (power ⩾ minPower && power ⩽ maxPower) {
            signals[i].displaySignal();
            found = true;
        }
    }

    if (!found) {
        cout << "No signals found in that power range." << endl;
    }
}
```