Name: Stephen Allegri
GTID: 903207717

Explanation of program structure: This assignment required a bit of preliminary research to understand how the K-means algorithm works, but once everything was clarified, the code itself was relatively straightforward. I made sure to give my script flexibility to deal with different numbers for both k (number of centroids) and n (dimensions). Both of these are determined by the user. This flexibility required some slightly more sophisticated for() loops, but overall, the entire algorithm could be created systematically. Each block of code works as a “step” in the algorithm, meaning that no functions were really necessary (I didn’t want to add complexity to the code). The lack of functions makes the script a bit messy to read, but otherwise more algorithmic/linear in nature.  Finally, to implement the z-score dataset, I created an empty array of size [nRows][n] that can hold both the original data and the z-score calculated data. This empty array is what is used in the iteration loop.

```
// K_switch indicates whether the loops will be using normal data or z-scores.
for (int k_switch = 0; k_switch < 2; k_switch = k_switch + 1) {
```

Testing procedure: For debugging purposes, I wrote the code in blocks and tested each block for expected outcomes with printf() statements. Some of these statements might be seen residually throughout the code, but I deleted most of them. On top of this, I made sure to add printf() functionality at the end of the iteration loop to display the original centroids and new, calculated centroids to see if they make sense. They do, unless the program is pushed beyond its limit.

```
for (int i = 0; i < k; i = i + 1) {
    random_indices[i] = (rand() % nRows) + 1;
    // printf("%d \n", random_indices[i]);
}
```

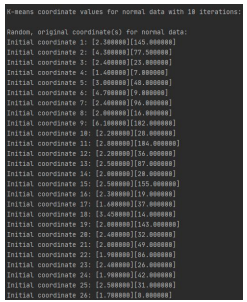
Example of a residual printf() statement testing for reasonability.

```
K-means coordinate values for normal data with 10 iterations:

Random, original coordinate(s) for normal data:
Initial coordinate 1: [2.300000][99.000000]
Initial coordinate 2: [5.500000][18.000000]

Coordinate for Centroid 1, Iteration 1:
2.904063
90.097865
Coordinate for Centroid 2, Iteration 1:
2.398832
29.748270
```

First few lines of program output (k = 2, n = 2).




When the program is run with an absurdly large k (like 200+ in the example on the left), the program broke. I make note of some tested-for limits in the script.

```
K-means coordinate values for normal data with 10 iterations:

Random, original coordinate(s) for normal data:
Initial coordinate 1: [2.400000][23.000000][11.000000]
Initial coordinate 2: [1.500000][13.000000][10.100000]
Initial coordinate 3: [2.600000][70.000000][9.900000]
Initial coordinate 4: [1.800000][34.000000][10.500000]
```

First few lines of program output (k = 3, n = 3).

Purpose of normalization: In a K-means algorithm, it's important to know if the new, learned centroids are actually good or bad measurements. This can be measured for by calculating the z-score. Z-score indicates how many standard deviations specific points are from the data mean. This is useful in the sense that, essentially, a minimized z-score indicates the "best fit" centroid. My program essentially calculates K-means for the z-score data (the normal data transformed into z-scores) on top of the normal dat, giving it additional functionality. By pinpointing z-score centroids, these results could be translated back to the original data to describe how well the K-means fit is (Though my program does not make this translation, as it wasn't specified in the problem statement). On a side note, even with randomized initial centroids, the z-score centroids, after being run through the K-means algorithm enough times, usually converged into the same few values.

```
Coordinate for Centroid 1, Iteration 20:
-21.304815
-39.987498
Coordinate for Centroid 2, Iteration 20:
-13.414747
39.987498
```

Purpose of randomized centroids: I'm not sure quite how this data is laid out, but if there was some sort of list or iterative pattern to the data, having the initial k data points as the initial centroids might not be indicative of a good first choice. Having a randomized set of initial centroids is more true to the algorithm and screens out any, potentially unseen, bias within the data.

```
// Generates seed for rand() to use- time based, as current time is always different.
srand(time(0));

for (int i = 0; i < k; i = i + 1) {
    random_indices[i] = (rand() % nRows) + 1;
    // printf("%d \n", random_indices[i]);
}

// Finding k random centroids.
for (int i = 0; i < k; i = i + 1) {
    for (int j = 0; j < n; j = j + 1) {
        initial_centroids[i][j] = data_array[random_indices[i]][j];
    }
}
```

Summary and result interpretation: As far as useful k-values, I found that the range of 2-4 was most informative. These values generally produced pretty consistent results after enough iterations (for normal data, usually 10 iterations, but for z-score data, more like 20 iterations), but anything past 4 messed with the results. I would guess that a z-score of 3 or 4 is best. To understand my results and see if the program is working correctly, I look for convergence. Essentially, after enough iterations, the algorithm will find the “best” centroid and continue to repeatedly iterate it. These convergences aren’t always the optimal solution, though, which is better determined with the help of a z-score. Overall, I think the clusters produced by the algorithm (for both .txt files) were useful. I am really happy with how the code turned out, and the process of generating these clusters was satisfying.

```
Coordinate for Centroid 1, Iteration 10:
2.395763
30.661913
Coordinate for Centroid 2, Iteration 10:
2.957740
92.759214
```

```
Coordinate for Centroid 1, Iteration 20:
-21.304815
-39.987498
Coordinate for Centroid 2, Iteration 20:
-13.414747
39.987498
```

I found that these coordinates were convergent (for dataset 1, k = 2).
This effect was seen with other k and n values.