Problem:

1. Given an array of temperature readings (float), create three threads:
- One thread finds the minimum temperature.
- One thread finds the maximum temperature.
- One thread calculates the average temperature.

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

float readings[] = {23.5, 25.1, 19.8, 30.2, 27.4, 22.6, 24.9};
int total_readings = sizeof(readings) / sizeof(readings[0]);

float lowest, highest, average;

void* compute_lowest(void* args) {
   lowest = readings[0];
   for (int i = 1; i < total_readings; i++) {
      if (readings[i] < lowest) {
         lowest = readings[i];
      }
   }
 pthread_exit(NULL); }
void* compute_highest(void* args) {
   highest = readings[0];
   for (int i = 1; i < total_readings; i++) {
      if (readings[i] > highest) {
         highest = readings[i];
      }
   }
   pthread_exit(NULL);
}
void* compute_average(void* args) {
   float sum = 0;
   for (int i = 0; i < total_readings; i++) {
      sum += readings[i];
   }
   average = sum / total_readings;
   pthread_exit(NULL);
}

int main() {
   pthread_t thread1, thread2, thread3;
   pthread_create(&thread1, NULL, compute_lowest, NULL);
   pthread_create(&thread2, NULL, compute_highest, NULL);
   pthread_create(&thread3, NULL, compute_average, NULL);
```

```
Output

Lowest Temperature: 19.80°C
Highest Temperature: 30.20°C
Average Temperature: 24.79°C


=== Code Execution Successful ===
```

```
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    pthread_join(thread3, NULL);

    printf("Lowest Temperature: %.2f°C\n", lowest);
    printf("Highest Temperature: %.2f°C\n", highest);
    printf("Average Temperature: %.2f°C\n", average);

    return 0;
}
```

Problem:

2. Given an array of integers, create threads to:
• Find the most frequent number
• Find the least frequent number
• Count the number of unique elements
Display the results after threads finish.

Code 2:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define LENGTH 10

int data[LENGTH] = {1, 2, 2, 3, 4, 4, 4, 5, 6, 1};
int occurrences[LENGTH];

int mode_value, min_occurrence_value, single_occurrence_count;

void* find_mode(void* arg) {
    int highest = 0;
    for (int i = 0; i < LENGTH; i++) {
        if (occurrences[i] > highest) {
            highest = occurrences[i];
            mode_value = data[i];
        }
    }
    pthread_exit(NULL);
}

void* find_rare(void* arg) {
    int lowest = LENGTH + 1;
    for (int i = 0; i < LENGTH; i++) {
        if (occurrences[i] > 0 && occurrences[i] < lowest) {
            lowest = occurrences[i];
            min_occurrence_value = data[i];
        }
```

Output

```
Most Frequent Value: 4
Least Frequent Value: 3
Total Unique Values: 3


=== Code Execution Successful ===
```

```
    }
    pthread_exit(NULL);
}

void* count_distinct(void* arg) {
    int distinct = 0;
    for (int i = 0; i < LENGTH; i++) {
        if (occurrences[i] == 1) {
            distinct++;
        }
    }
    single_occurrence_count = distinct;
    pthread_exit(NULL);
}

int main() {
    for (int i = 0; i < LENGTH; i++) {
        occurrences[i] = 0;
        for (int j = 0; j < LENGTH; j++) {
            if (data[i] == data[j]) {
                occurrences[i]++;
            }
        }
    }

    pthread_t threadA, threadB, threadC;

    pthread_create(&threadA, NULL, find_mode, NULL);
    pthread_create(&threadB, NULL, find_rare, NULL);
    pthread_create(&threadC, NULL, count_distinct, NULL);

    pthread_join(threadA, NULL);
    pthread_join(threadB, NULL);
    pthread_join(threadC, NULL);

    printf("Most Frequent Value: %d\n", mode_value);
    printf("Least Frequent Value: %d\n", min_occurrence_value);
    printf("Total Unique Values: %d\n", single_occurrence_count);

    return 0;
}
```

Problem:

3. Given an array of integers, create three threads:
• One thread identifies and counts prime numbers
• One thread counts composite numbers
• One thread counts palindromic numbers (e.g., 121, 454)
Print the results after threads finish.

Code 3:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <math.h>

#define LENGTH 10

int numbers[LENGTH] = {2, 3, 4, 5, 121, 131, 22, 17, 9, 7};

int total_primes = 0, total_composites = 0, total_palindromes = 0;

int check_prime(int num) {
    if (num <= 1) return 0;
    if (num == 2) return 1;
    if (num % 2 == 0) return 0;
    for (int i = 3; i <= sqrt(num); i += 2) {
        if (num % i == 0) return 0;
    }
    return 1;
}

int check_palindrome(int num) {
    int original = num, reversed = 0;
    while (num > 0) {
        reversed = reversed * 10 + num % 10;
        num /= 10;
    }
    return original == reversed;
}

void* compute_primes(void* arg) {
    for (int i = 0; i < LENGTH; i++) {
        if (check_prime(numbers[i])) {
            total_primes++;
        }
    }
    pthread_exit(NULL);
}


void* compute_composites(void* arg) {
```

```
Output

Total Prime Numbers: 6
Total Composite Numbers: 4
Total Palindromic Numbers: 9


=== Code Execution Successful ===
```

```
    for (int i = 0; i < LENGTH; i++) {
        if (numbers[i] > 1 && !check_prime(numbers[i])) {
            total_composites++;
        }
    }
    pthread_exit(NULL);
}

void* compute_palindromes(void* arg) {
    for (int i = 0; i < LENGTH; i++) {
        if (check_palindrome(numbers[i])) {
            total_palindromes++;
        }
    }
    pthread_exit(NULL);
}

int main() {
    pthread_t thread1, thread2, thread3;

    pthread_create(&thread1, NULL, compute_primes, NULL);
    pthread_create(&thread2, NULL, compute_composites, NULL);
    pthread_create(&thread3, NULL, compute_palindromes, NULL);

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    pthread_join(thread3, NULL);

    printf("Total Prime Numbers: %d\n", total_primes);
    printf("Total Composite Numbers: %d\n", total_composites);
    printf("Total Palindromic Numbers: %d\n", total_palindromes);

    return 0;
}
```

Problem:

4. Write a program that performs matrix multi-
plication using threads:

• Each thread should compute one row (or cell) of the resulting matrix.

Code 4:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define N 3
#define M 3
```

```
int matrix1[N][M] = {
   {1, 2, 3},
   {4, 5, 6},
   {7, 8, 9}
};

int matrix2[M][N] = {
   {9, 8, 7},
   {6, 5, 4},
   {3, 2, 1}
};

int result[N][N];

void* process_row(void* arg) {
   int r = *(int*)arg;
   for (int c = 0; c < N; c++) {
      result[r][c] = 0;
      for (int k = 0; k < M; k++) {
         result[r][c] += matrix1[r][k] * matrix2[k][c];
      }
   }
   free(arg);
   pthread_exit(NULL);
}

int main() {
   pthread_t workers[N];
   for (int i = 0; i < N; i++) {
      int* idx = malloc(sizeof(int));
      *idx = i;
      pthread_create(&workers[i], NULL, process_row, idx);
   }
   for (int i = 0; i < N; i++) {
      pthread_join(workers[i], NULL);
   }

   printf("Matrix Multiplication Result:\n");
   for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) {
         printf("%d ", result[i][j]);
      }
      printf("\n");
   }

   return 0;
}
```

Output

```
Matrix Multiplication Result:
30 24 18
84 69 54
138 114 90


=== Code Execution Successful ===
```