```
#include <vector> // --> Includes the vector container for dynamic arrays
#include <iostream> // --> Includes input/output stream functionality
#include <algorithm> // --> Includes algorithms like sort, find, etc.
#include <windows.h> // --> Includes Windows-specific APIs for multithreading

struct Query // --> Defines a structure to store a query with ID and conditions
{
   int id; // --> Query identifier
   std::vector<int> conditions; // --> List of conditions for the query
};

std::vector<int> data; // --> Global data vector shared across threads
HANDLE mutex; // --> Mutex handle for synchronizing thread access
int total_matches = 0; // --> Shared variable to store total matches

DWORD WINAPI thread_function(LPVOID lpParam); // --> Declaration of thread function

struct ThreadData // --> Structure to pass data to threads
{
   Query *query; // --> Pointer to a Query object
};

int main() // --> Entry point of the program
{
   int n; // --> Variable to store number of data elements
   std::cout << "Enter total number of data elements: "; // --> Prompt user for data size
   std::cin >> n; // --> Take input for data size

   data.resize(n); // --> Resize data vector to hold n elements
   std::cout << "Enter data elements:\n"; // --> Prompt for data input
   for (int i = 0; i < n; i++) // --> Loop to input each data element
```

```
    {
        std::cin >> data[i]; // --> Take input for each data element
    }

    int num_queries; // --> Variable to store number of queries
    std::cout << "\nEnter number of queries: "; // --> Prompt user for number of queries
    std::cin >> num_queries; // --> Take input for number of queries

    std::vector<Query> queries(num_queries); // --> Create vector to store all queries
    for (int i = 0; i < num_queries; i++) // --> Loop over all queries
    {
        int k; // --> Variable to store number of conditions
        std::cout << "Enter number of conditions for Query " << i + 1 << ": "; // --> Prompt for number of conditions
        std::cin >> k; // --> Input number of conditions

        std::cout << "Enter " << k << " condition values:\n"; // --> Prompt for condition values
        queries[i].id = i; // --> Assign query ID
        queries[i].conditions.resize(k); // --> Resize condition vector
        for (int j = 0; j < k; j++) // --> Loop to input each condition
        {
            std::cin >> queries[i].conditions[j]; // --> Input each condition value
        }
    }

    mutex = CreateMutex(NULL, FALSE, NULL); // --> Create a mutex for thread synchronization

    std::vector<HANDLE> threads(num_queries); // --> Vector to store thread handles
    std::vector<ThreadData> thread_data(num_queries); // --> Vector to store data for each thread

    for (int i = 0; i < num_queries; i++) // --> Loop to create threads for each query
```

```
    {
        thread_data[i].query = &queries[i]; // --> Set query pointer for each thread
        threads[i] = CreateThread( // --> Create a new thread
            NULL, 0, thread_function, &thread_data[i], 0, NULL); // --> Pass thread function and data
    }

    WaitForMultipleObjects(num_queries, threads.data(), TRUE, INFINITE); // --> Wait for all threads to finish

    for (auto &th : threads) // --> Loop through all thread handles
    {
        CloseHandle(th); // --> Close each thread handle
    }
    CloseHandle(mutex); // --> Close the mutex handle

    std::cout << "\n\nFinal total matches found: " << total_matches << std::endl; // --> Print final total matches

    return 0; // --> End of program
}

DWORD WINAPI thread_function(LPVOID lpParam) // --> Thread function to process query
{
    ThreadData *data_ptr = (ThreadData *)lpParam; // --> Cast parameter to ThreadData pointer
    Query *query = data_ptr->query; // --> Get the query pointer

    int local_count = 0; // --> Local counter for matches
    for (int record : data) // --> Loop over all data elements
    {
        for (int cond : query->conditions) // --> Loop over query conditions
        {
            if (record % cond == 0) // --> Check if record satisfies condition
```

```
        {
            local_count++; // --> Increment local match count
            break; // --> Stop checking other conditions for this record
        }
      }
    }

    WaitForSingleObject(mutex, INFINITE); // --> Lock mutex before updating shared variable
    total_matches += local_count; // --> Safely add local count to global total
    std::cout << "Thread for Query " << query->id + 1 << " found " << local_count << " matches.
Total so far: " << total_matches << std::endl; // --> Print thread result
    ReleaseMutex(mutex); // --> Unlock mutex

    return 0; // --> End of thread
}
```