prime_numbers.cpp

```
// prime_numbers.cpp
// Multithreaded prime enumerator with per-thread temp files + delayed
merge.
// Build: g++ -std=c++17 -O2 -pthread prime_numbers.cpp -o
prime_numbers
//
// Usage: ./prime_numbers <limit>
// Example: ./prime_numbers 50

// contributors :3 :
// Daniel Le and Keana De Padua


#include <algorithm>
#include <cctype>
#include <cmath>
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <thread>
#include <vector>
#include <limits>

/**
 *  Determines whether a number is prime or not
 *  @param longlongint Number to be checked
 *  @return True if prime, false if not prime
 */
static bool isPrime(long long n) {
    if (n < 2) return false;
    if (n == 2) return true;
    if (n % 2 == 0) return false;
    // Check odd divisors up to sqrt(n)
    for (long long d = 3; d * d <= n; d += 2) {
        if (n % d == 0) return false;
    }
    return true;
}

struct Range {
    long long start; // inclusive
    long long end;   // inclusive
```

```
};

/**
 *  Computes primes in [range.start, range.end]
 *  Write both to local vector and a temp file
 *  @param unsignedint Worker Id
 *  @param Range Range from starting to end point for worker to work
on
 *  @param vector Local vector for output to be stored
 *  @param string Name of file where output will be stored
 *  @return Nothing
*/
static void workerTask(unsigned int workerId,
                       Range range,
                       std::vector<long long>& localOut,
                       const std::string& fileName)
{
    localOut.clear();
    localOut.reserve((range.end >= range.start) ?
static_cast<size_t>(range.end - range.start + 1) : 0);

    for (long long x = range.start; x <= range.end; ++x) {
        if (isPrime(x)) {
            localOut.emplace_back(x);
        }
    }

    // write results to per-thread file
    std::ofstream ofs(fileName);
    if (ofs) {
        for (size_t i = 0; i < localOut.size(); ++i) {
            ofs << localOut[i] << (i + 1 == localOut.size() ? "" : "
");
        }
        ofs << "\n";
    } else {
        // If file can't be opened, fall back silently (merge will
still succeed from vectors if we kept them).
        // Keeping a message can help debugging:
        std::cerr << "[WARN] Thread " << workerId + 1 << ": failed to
open " << fileName << " for writing.\n";
    }
}

/**
 *  Strictly parse a non-negative integer that fits into long long
```

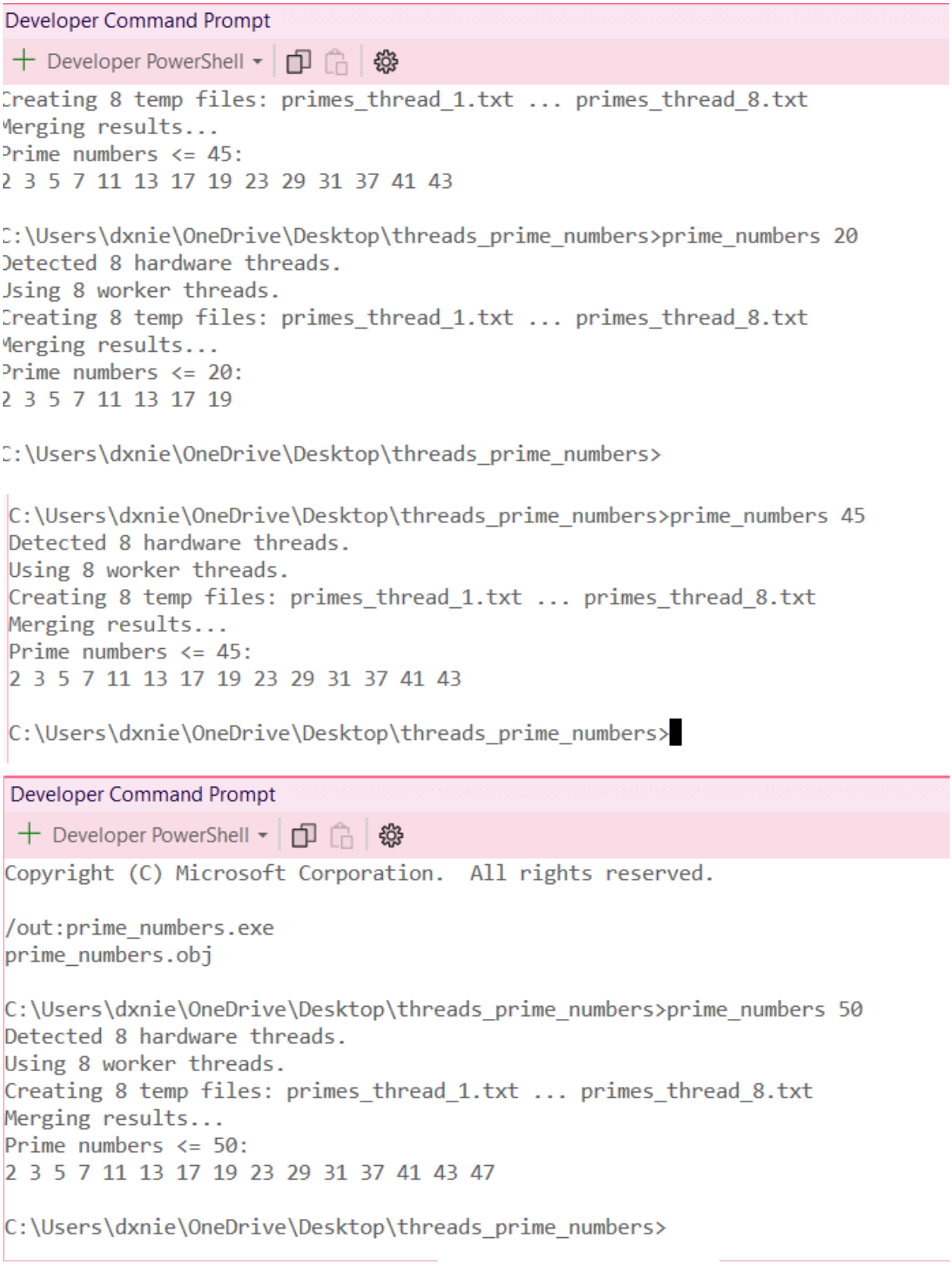
```
 *  Reject leading +/-, spaces inside, and non-digits
 *  @param char* Pointer to beginning of string we want to parse
 *  @param longlong Where we want the int to be stored
 */
static bool parseLimit(const char* s, long long& out) {
    std::string str(s);
    if (str.empty()) return false;
    for (char c : str) if (!std::isdigit(static_cast<unsigned
char>(c))) return false;
    std::istringstream iss(str);
    long long v;
    iss >> v;
    if (!iss || !iss.eof()) return false;
    out = v;
    return true;
}

int main(int argc, char* argv[]) {
    // Parse CLI
    if (argc != 2) {
        std::cerr << "Usage: " << argv[0] << " <limit>\n";
        std::cerr << "Example: " << argv[0] << " 50\n";
        return 1;
    }

    long long limit = 0;
    if (!parseLimit(argv[1], limit)) {
        std::cerr << "Error: limit must be a non-negative integer.\n";
        return 1;
    }

    // determine thread count
    unsigned int hw = std::thread::hardware_concurrency();
    if (hw == 0) hw = 2; // sensible default
    // Optional: don't spawn more threads than numbers to check
    unsigned int numThreads = static_cast<unsigned int>(std::min<long
long>(hw, std::max<long long>(1, limit)));
    // Special-case: if limit < 2, we still spawn 1 thread to keep
logic simple
    if (limit < 2) numThreads = 1;

    std::cout << "Detected " << hw << " hardware threads.\n";
    std::cout << "Using " << numThreads << " worker thread" <<
(numThreads == 1 ? "" : "s") << ".\n";

    // partition work using ceiling block size
```

```
    // We want to cover [1..limit], even though 1 is not prime;
simpler math.
    auto ceil_div = [](long long a, long long b) -> long long {
        return (a + b - 1) / b;
    };
    long long block = (numThreads > 0) ? ceil_div(limit,
static_cast<long long>(numThreads)) : limit;

    std::vector<Range> ranges;
    ranges.reserve(numThreads);
    for (unsigned int i = 0; i < numThreads; ++i) {
        long long start = i * block + 1;
        long long end   = std::min(limit, (static_cast<long long>(i) +
1) * block);
        if (start > end) { // happens when limit is small
            start = 1; end = 0; // empty range
        }
        ranges.push_back({start, end});
    }

    // Launch threads
    std::vector<std::thread> threads;
    std::vector<std::vector<long long>> locals(numThreads);
    std::vector<std::string> tempFiles(numThreads);

    // Creating tempFiles for each thread
    for (unsigned int i = 0; i < numThreads; ++i) {
        std::ostringstream oss;
        oss << "primes_thread_" << (i + 1) << ".txt";
        tempFiles[i] = oss.str();
    }

    if (!tempFiles.empty()) {
        if (numThreads == 1) {
            std::cout << "Creating 1 temp file: " << tempFiles.front()
<< "\n";
        } else {
            std::cout << "Creating " << tempFiles.size() << " temp
files: "
                      << tempFiles.front() << " ... " <<
tempFiles.back() << "\n";
        }
    }

    // Create threads
    threads.reserve(numThreads);
```

```
    for (unsigned int i = 0; i < numThreads; ++i) {
        threads.emplace_back(workerTask, i, ranges[i],
std::ref(locals[i]), std::ref(tempFiles[i]));
    }

    // Join all threads
    for (auto& t : threads) t.join();

    //  Merge from files (delayed merge). Read the temp files (as
required),
    //    but also append from locals as a fallback if a file was
missing.
    std::cout << "Merging results...\n";
    std::vector<long long> all;
    for (unsigned int i = 0; i < numThreads; ++i) {
        bool gotFromFile = false;
        {
            std::ifstream ifs(tempFiles[i]);
            if (ifs) {
                gotFromFile = true;
                long long v;
                while (ifs >> v) {
                    all.emplace_back(v);
                }
            }
        }
        if (!gotFromFile) {
            // Fallback: use in-memory results for this thread (still
correct).
            all.insert(all.end(), locals[i].begin(), locals[i].end());
        }
    }

    std::sort(all.begin(), all.end());
    all.erase(std::unique(all.begin(), all.end()), all.end());

    // Pretty final output
    if (limit < 2 || all.empty()) {
        std::cout << "No primes <= " << limit << ".\n";
        return 0;
    }

    std::cout << "Prime numbers <= " << limit << ":\n";
    for (size_t i = 0; i < all.size(); ++i) {
        std::cout << all[i] << (i + 1 == all.size() ? '\n' : ' ');
    }
```

```
    return 0;
}
```

Output:


Developer Command Prompt
Developer PowerShell
Creating 8 temp files: primes_thread_1.txt ... primes_thread_8.txt
Merging results...
Prime numbers <= 45:
2 3 5 7 11 13 17 19 23 29 31 37 41 43
C:\Users\dxnie\OneDrive\Desktop\threads_prime_numbers>prime_numbers 20
Detected 8 hardware threads.
Using 8 worker threads.
Creating 8 temp files: primes_thread_1.txt ... primes_thread_8.txt
Merging results...
Prime numbers <= 20:
2 3 5 7 11 13 17 19
C:\Users\dxnie\OneDrive\Desktop\threads_prime_numbers>
C:\Users\dxnie\OneDrive\Desktop\threads_prime_numbers>prime_numbers 45
Detected 8 hardware threads.
Using 8 worker threads.
Creating 8 temp files: primes_thread_1.txt ... primes_thread_8.txt
Merging results...
Prime numbers <= 45:
2 3 5 7 11 13 17 19 23 29 31 37 41 43
C:\Users\dxnie\OneDrive\Desktop\threads_prime_numbers>
Developer Command Prompt
Developer PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
/out:prime_numbers.exe
prime_numbers.obj
C:\Users\dxnie\OneDrive\Desktop\threads_prime_numbers>prime_numbers 50
Detected 8 hardware threads.
Using 8 worker threads.
Creating 8 temp files: primes_thread_1.txt ... primes_thread_8.txt
Merging results...
Prime numbers <= 50:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
C:\Users\dxnie\OneDrive\Desktop\threads_prime_numbers>

Evaluation Form

Group Members: Daniel Le & Keana De Padua

Daniel // Project Manager & Developer
- Tasks completed:
    - created ‘skeleton’ code → prime_numbers.cpp
    - completed code according to assignment requirements
        - Compute prime and write both vector and temp files.
        - Code to write results of the prime numbers (per-thread)
        - Parsing (parse CLI) and determining the thread count
    - ran tests for output
    - created repository
- Achievements:
    - completion of code in accordance with the assignment
    - created the repository to level the field.
- Contribution and Performance: 10

Keana // Developer
- Tasks completed:
    - Completed code required from the assignment,
        - Established the creation of threads and tempFiles.
        - Modularized code for a clean slate
        - Coded to write the results of prime numbers
        - Also completed the code necessary for parsing.
        - Added comments to distinguish significant points of code.
        - Ran tests for output
- Achievements:
    - completion of code in accordance with the assignment
    - Cleaned code to make it pretty, organized and informative (with comments)
- Contribution and Performance: 10