# CSCI 654-01—Foundations of Parallel Computing
# Programming Project 1

**Prof. Alan Kaminsky—Fall Semester 2018**
Rochester Institute of Technology—Department of Computer Science



---

## Overview

Write a sequential program and a parallel program in Java using the Parallel Java 2 Library. Run the programs on a multicore parallel computer to learn about multicore parallel programming and strong scaling.

**Help with your project:** I am willing to help you with the design of your project. I am willing to help you debug your project if the code isn't working. However, for help with design or debugging issues *you must come see me in person.* Either visit me during office hours or make an appointment. I will not help you with design or debugging issues via email. If it's the evening of the project deadline and I have gone home, you are on your own. Plan and work ahead so there will be plenty of time for me to help you if necessary.

---

## Lemoine's Conjecture

In 1894, French mathematician Émile Lemoine (1840–1912) made this conjecture:

**Every odd integer greater than 5 is the sum of a prime and twice a prime.**

In other words, if $n$ is odd and $n > 5$, then $n = p + 2q$ for some primes $p$ and $q$. The primes $p$ and $q$ might or might not be the same. Note that $p$ must be odd; $q$ might be odd or even. There might be more than one solution to the formula. Here are some examples:

7 = 3 + 2·2
9 = 3 + 2·3 = 5 + 2·2
11 = 5 + 2·3 = 7 + 2·2
13 = 3 + 2·5 = 7 + 2·3
15 = 5 + 2·5 = 11 + 2·2
17 = 3 + 2·7 = 7 + 2·5 = 13 + 2·2
19 = 5 + 2·7 = 13 + 2·3
99 = 5 + 2·47 = 13 + 2·43 = 17 + 2·41 = 37 + 2·31 = 41 + 2·29 = 53 + 2·23 = 61 + 2·19 = 73 + 2·13 = 89 + 2·5
199 = 5 + 2·97 = 41 + 2·79 = 53 + 2·73 = 113 + 2·43 = 137 + 2·31 = 173 + 2·13 = 193 + 2·3

The conjecture has been verified for $n$ up to $10^9$. However, no one has been able to prove or disprove it in general.

You will write sequential and multicore parallel versions of a program that verifies Lemoine's Conjecture for all odd integers in a given range. For purposes of this project, we will assume that Lemoine's Conjecture is true.

You may use class Prime (source code: Prime.java) in your programs. Class Prime contains an iterator over odd primes and a method to test whether an `int` is prime. You are not allowed to change class Prime in any way.

---

# Program Input and Output

The program's command line arguments are the lower bound and upper bound of a range of integers. The lower bound must be an odd integer greater than 5. The upper bound must be an odd integer greater than or equal to the lower bound.

The program must iterate over all odd integers $n$ from the lower bound to the upper bound inclusive. For each $n$, the program must find the *smallest* prime $p$ such that $n = p + 2q$, where $q$ is prime.

The program must print one and only one line of output, as in this example:

```
$ java pj2 LemoineSeq 1000001 1999999
1746551 = 1237 + 2*872657
```

The output line consists of the integer $n$, the string `" = "`, the integer $p$, the string `" + 2*"`, and the integer $q$, terminated with a newline.

The program must print the $n$, $p$, and $q$ values such that $p$ is the largest among the integers $n$ examined. If more that one integer yielded the same maximum $p$ value, the program must print the one with the largest $n$ value.

---

# Software Requirements

1. The sequential version of the program must be run by typing this command line:
2. `java pj2 LemoineSeq <lb> <ub>`
    a. `<lb>` is the lower bound integer to be examined (an odd integer greater than 5, type `int`).
    b. `<ub>` is the upper bound integer to be examined (an odd integer greater than or equal to `<lb>`, type `int`).

    *Note:* This means that the program's class must be named `LemoineSeq`, this class must not be in a package, and this class must extend class edu.rit.pj2.Task.

    *Note:* The above command will work on the `kraken` computer. On the `tardis` computer, you must include the `jar=` option. For further information, see Parallel Java 2 on the RIT CS Parallel Computers.

3. The parallel version of the program must be run by typing this command line:
4. `java pj2 cores=<K> LemoineSmp <lb> <ub>`
    a. `<K>` is the number of CPU cores to use; by default, this is also the number of threads in the parallel thread team.
    b. `<lb>` is the lower bound integer to be examined (an odd integer greater than 5).
    c. `<ub>` is the upper bound integer to be examined (an odd integer greater than or equal to `<lb>`).

    *Note:* This means that the program's class must be named `LemoineSmp`, this class must not be in a package, and this class must extend class edu.rit.pj2.Task.

    *Note:* The above command will work on the `kraken` computer. On the `tardis` computer, you must include the `jar=` option. For further information, see Parallel Java 2 on the RIT CS Parallel Computers.

5. If the command line does not have the required number of arguments, if any argument is erroneous, or if any other error occurs, the program must print an error message on the console and must exit. The error message must describe the problem. The wording of the error message is up to you.
6. The program must print on the console the information specified above under Program Input and Output.

    *Note: If your program's output does not conform **exactly** to Software Requirements 1 through 4, **you will lose credit** on your project.* See the Grading Criteria below.

---

# Software Design Criteria

1. The programs must follow the multicore parallel programming patterns studied in class.
2. The programs must be designed using object oriented design principles as appropriate.
3. The programs must make use of reusable software components as appropriate.
4. Each class or interface must include a Javadoc comment describing the overall class or interface.
5. Each constructor or method within each class or interface must include a Javadoc comment describing the overall constructor or method, the arguments if any, the return value if any, and the exceptions thrown if any.

   *Note:* See my Java source files which we studied in class for the style of Javadoc comments I'm looking for.

   *Note: If your program's design does not conform to Software Design Criteria 1 through 5,* ***you will lose credit*** *on your project.* See the Grading Criteria below.

---

# Submission Requirements

Your project submission will consist of a ZIP file containing the Java source file for every class and interface in your project. Put all the source files into a ZIP file named "`<username>.zip`", replacing `<username>` with the user name from your Computer Science Department account. On a Linux system, the command for creating a ZIP file is:

```
zip <username>.zip *.java
```

***DO NOT include the Parallel Java 2 Library in your ZIP file!***

Send your ZIP file to me by email at ark@cs.rit.edu. Include your full name and your computer account name in the email message, and include the ZIP file as an attachment.

When I get your email message, I will extract the contents of your ZIP file into a directory. I will copy the Prime.java source file into the directory where I extracted your files (overwriting any Prime.java source file in your ZIP file). I will set my Java class path to include the directory where I extracted your files, plus the Parallel Java 2 Library. I will compile all the Java source files using the JDK 1.7 compiler. I will then send you a reply message acknowledging I received your project and stating whether I was able to compile all the source files. If you have not received a reply within one business day (i.e., not counting weekends), please contact me. Your project is not successfully submitted until I have sent you an acknowledgment stating I was able to compile all the source files.

The submission deadline is Monday, September 24, 2018 at 11:59pm. The date/time at which your email message arrives in my inbox will determine whether your project meets the deadline.

You may submit your project multiple times up until the deadline. I will keep and grade only the most recent successful submission. There is no penalty for multiple submissions.

If you submit your project before the deadline, but I do not accept it (e.g. I can't compile all the source files), and you cannot or do not submit your project again before the deadline, the project will be late (see below). ***I STRONGLY advise you to submit the project several days BEFORE the deadline, so there will be time to deal with any problems that might arise in the submission process.***

---

# Grading Criteria

I will grade your project by:

- (10 points) Evaluating the design of your sequential and parallel programs, as documented in the Javadoc and as implemented in the source code.
  - All of the Software Design Criteria are fully met: 10 points.
  - Some of the Software Design Criteria are not fully met: 0 points.
- (10 points) Running your sequential and parallel programs. There will be ten test cases, each worth 1 point. For each test case, if both programs run using the command lines in Requirements 1 and 2 and both programs produce the correct output, the test case will get 1 point, otherwise the test case will get 0 points. "Correct output" means "output fulfills *all* the Software Requirements *exactly*."
- (10 points) Strong scaling performance. I will run your parallel program on one node of the `tardis` cluster, on a certain test case, with $K = 1$ core, three times, and take the smallest running time $T_1$. I will run the same test case, with $K = 4$ cores, three times, and take the smallest running time $T_2$. I will compute $Efficiency = T_1/4T_2$.
  - $Efficiency \geq 0.95$: 10 points.
  - $Efficiency \geq 0.90$: 8 points.
  - $Efficiency \geq 0.85$: 6 points.
  - $Efficiency \geq 0.80$: 4 points.
  - $Efficiency \geq 0.75$: 2 points.
  - Otherwise: 0 points.
- (30 points) Total.

When I run your programs, the Java class path will point to the directory with your compiled class files, plus the Parallel Java 2 Library. I will use JDK 1.7 to run your program.

I will grade the test cases based *solely* on whether your program produces the correct output as specified in the above Software Requirements. *Any* deviation from the requirements will result in a grade of 0 for the test case. This includes errors in the formatting (such as extra spaces), output lines not terminated with a newline, and extraneous output not called for in the requirements. The requirements state *exactly* what the output is supposed to be, and there is no excuse for outputting anything different. If any requirement is unclear, please ask for clarification.

If there is a defect in your program and that same defect causes multiple test cases to fail, I will deduct points for *every* failing test case. The number of points deducted does *not* depend on the

size of the defect; I will deduct the same number of points whether the defect is 1 line, 10 lines, 100 lines, or whatever.

After grading your project I will put your grade and any comments I have in your encrypted grade file. For further information, see the Course Grading and Policies and the Encrypted Grades.

**Test Cases**

The grading test case commands and correct printouts were as follows. For test cases 1–2, the wording of the error message was up to you, as long as it described the problem.

***IMPORTANT NOTE***
Running on a `tardis` cluster node, each test case 1–10 should have taken about 2 seconds, and the scaling test case should have taken about 60 seconds. I set a 10-second timeout on test cases 1–10 and a 120-second timeout on the scaling test case. If the program timed out, it indicates a bad design.

***IMPORTANT NOTE***
I have accurately recorded in your grade file what I observed during my testing. If you run the test cases on your machine and do not get the same results as I, then there is a flaw in your design somewhere that causes inconsistent results (perhaps due to incorrect parallelization), and I am not going to change your grade for the original submission. You may still resubmit your project.

```
1.  $ java pj2 timelimit=10 jar=p1.jar cores=1 LemoineSeq -1 1001
    LemoineSeq: Lower bound must be > 5
    Usage: java pj2 LemoineSeq <lb> <ub>
    $ java pj2 timelimit=10 jar=p1.jar cores=4 LemoineSmp -1 1001
    LemoineSmp: Lower bound must be > 5
    Usage: java pj2 [cores=<K>] LemoineSmp <lb> <ub>

2.  $ java pj2 timelimit=10 jar=p1.jar cores=1 LemoineSeq 2001 1001
    LemoineSeq: Upper bound must be >= lower bound
    Usage: java pj2 LemoineSeq <lb> <ub>
    $ java pj2 timelimit=10 jar=p1.jar cores=4 LemoineSmp 2001 1001
    LemoineSmp: Upper bound must be >= lower bound
    Usage: java pj2 [cores=<K>] LemoineSmp <lb> <ub>

3.  $ java pj2 timelimit=10 jar=p1.jar cores=1 LemoineSeq 100001 200001
    189947 = 829 + 2*94559
    $ java pj2 timelimit=10 jar=p1.jar cores=4 LemoineSmp 100001 200001
    189947 = 829 + 2*94559

4.  $ java pj2 timelimit=10 jar=p1.jar cores=1 LemoineSeq 200001 300001
    240719 = 733 + 2*119993
    $ java pj2 timelimit=10 jar=p1.jar cores=4 LemoineSmp 200001 300001
    240719 = 733 + 2*119993

5.  $ java pj2 timelimit=10 jar=p1.jar cores=1 LemoineSeq 300001 400001
    351251 = 829 + 2*175211
```

```
    $ java pj2 timelimit=10 jar=p1.jar cores=4 LemoineSmp 300001 400001
    351251 = 829 + 2*175211

6.  $ java pj2 timelimit=10 jar=p1.jar cores=1 LemoineSeq 400001 500001
    442307 = 1129 + 2*220589
    $ java pj2 timelimit=10 jar=p1.jar cores=4 LemoineSmp 400001 500001
    442307 = 1129 + 2*220589

7.  $ java pj2 timelimit=10 jar=p1.jar cores=1 LemoineSeq 500001 600001
    540263 = 937 + 2*269663
    $ java pj2 timelimit=10 jar=p1.jar cores=4 LemoineSmp 500001 600001
    540263 = 937 + 2*269663

8.  $ java pj2 timelimit=10 jar=p1.jar cores=1 LemoineSeq 600001 700001
    668783 = 937 + 2*333923
    $ java pj2 timelimit=10 jar=p1.jar cores=4 LemoineSmp 600001 700001
    668783 = 937 + 2*333923

9.  $ java pj2 timelimit=10 jar=p1.jar cores=1 LemoineSeq 700001 800001
    713687 = 1021 + 2*356333
    $ java pj2 timelimit=10 jar=p1.jar cores=4 LemoineSmp 700001 800001
    713687 = 1021 + 2*356333

10. $ java pj2 timelimit=10 jar=p1.jar cores=1 LemoineSeq 800001 900001
    827249 = 883 + 2*413183
    $ java pj2 timelimit=10 jar=p1.jar cores=4 LemoineSmp 800001 900001
    827249 = 883 + 2*413183

Scaling test case
    $ java pj2 timelimit=120 jar=p1.jar debug=makespan cores=1 LemoineSmp
2000001 6000001
    5056787 = 1669 + 2*2527559
    $ java pj2 timelimit=120 jar=p1.jar debug=makespan cores=4 LemoineSmp
2000001 6000001
    5056787 = 1669 + 2*2527559
```

---

# Late Projects

If I have not received a successful submission of your project by the deadline, your project will be late and will receive a grade of zero. You may request an extension for the project. There is no penalty for an extension. See the Course Policies for my policy on extensions.

---

# Plagiarism

Programming Project 1 must be entirely your own individual work. I will not tolerate plagiarism. If in my judgment the project is not entirely your own work, you will automatically receive, as a minimum, a grade of zero for the assignment. See the Course Policies for my policy on plagiarism.

## Resubmission

If you so choose, you may submit a revised version of your project after you have received the grade for the original version. However, if the original project was not successfully submitted by the (possibly extended) deadline or was not entirely your own work (i.e., plagiarized), you are not allowed to submit a revised version. Submit the revised version via email in the same way as the original version. I will accept a resubmission up until 11:59pm Wednesday 03-Oct-2018. You may resubmit your project multiple times up until the deadline; I will keep and grade only the most recent successful resubmission; there is no penalty for multiple resubmissions. I will grade the revised version using the same criteria as the original version, then I will subtract 3 points (10% of the maximum possible points) as a resubmission penalty. The revised grade will replace the original grade, even if the revised grade is less than the original grade.