Virginia Tech
Bradley Department of Electrical and Computer Engineering
ECE-3574: Applied Software Engineering Fall 2014

Homework 7

Submission Details
You must submit the solutions for this homework as an electronic submissions using Scholar (under ECE3574 → Assignments → Homework 7). The submission must be a gzipped tar file (.tar.gz) with your source code. Include all necessary project files, but no binary or compiled files. Your program will be run to evaluate its correctness, and the source code will be reviewed for adherence to the Qt programming style. Your program must run on Ubuntu 14.04.1 and compile/build using the GNU C/C++ compiler and the qmake/make tools. The following information must be included at the top of each of your source files as comments: your full name, your student ID number, your email address, class (ECE 3574), and the title of the assignment (Homework 7). The submitted file must be given a name in the following form: ***LAST_FIRST_hw7.tar.gz*** where LAST is your last or family name and FIRST is your first or given name. Paper, email or Drop Box submissions will not be accepted. All work must be submitted by the announced due date/time. **Late submissions will not be accepted!** (Don't do it! You have been warned!)

Questions
Use the Homework 7 forum in the Discussion Board area of the class web site to ask questions about this assignment. Do not post questions that contain specific information about the solution.

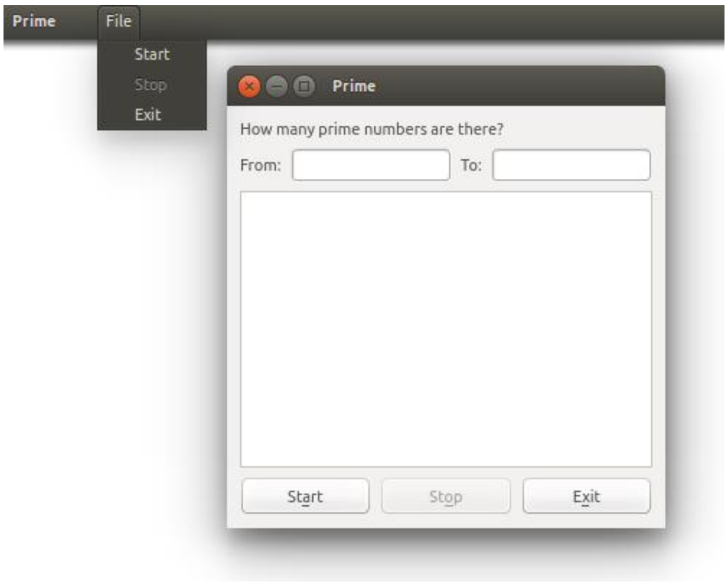
Honor Code
As stated in the syllabus, in working on homework and projects, discussion and cooperative learning are allowed. However, copying or otherwise using another person’s detailed solutions to assigned problems is an honor code violation. See syllabus for details.

Learning Objectives

The primary learning objectives of this assignment include event-driven programming abstractions, creation of graphical applications and concurrent programming abstractions including POSIX threading. Additional learning objectives include the use of generics, containers and synchronization abstractions.

Specifications

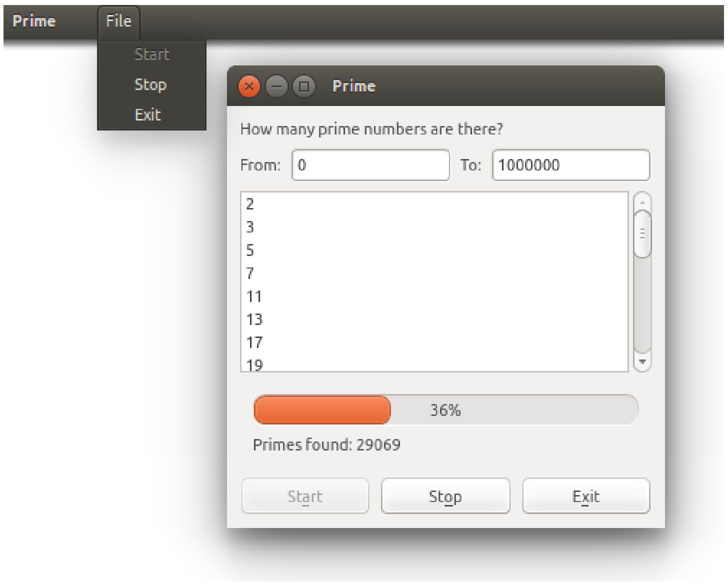
1. Your program must find the prime numbers between two numbers which must be read from the QLineEdit "From" and "To", as in Figure 1. Use quint64 as your data type to allow a larger range of prime numbers to be searched. Use QString::toULongLong to perform the conversion. The values in "From" and "To" must be included in the search.
2. Create a menu, "File", as a QMenu with the following actions as QActions in the QMenu, as in Figure 1, and functionality:
    a. "Start" - start the prime number search
    b. "Stop" - terminate the prime number search
    c. "Exit" - terminate the prime number search if running and cleanly exit the application
3. Create three buttons with the same text and functionality as the above actions, as in Figure 1.
4. When the program is not searching for prime numbers the "Stop" button and action must be disabled and the "Start" button and action must be enabled, as in Figure 1 and 2.
5. Likewise, when the program is searching for prime numbers the "Stop" button and action must be enabled and the "Start" button and action must be disabled, as in Figure 2.
6. While the prime search is being performed a QProgressBar must be shown and it must be regularly updated indicating how far the program has progressed through the search, as in Figure 2. This progress bar must be visible during a search, as in Figure 2, and not visible otherwise, as in Figure 3.
7. While the prime search is being performed, as in Figure 2, and until the next search starts a label must be shown, as in Figure 3, and it must be regularly updated indicating the number of primes found in the specified range during the search. This label must be visible during the search, must be visible after the search has completed with the final prime count and must not be visible before the first prime search after the application opened, as in Figure 1.
8. As the prime numbers are found they must be listed in a QListView, as in Figure 2, which uses a custom class which inherits from QAbstractListModel as its model. The prime numbers found must remain in the QListView until the start of the next prime search, as in Figure 3.
9. The search must be performed in a different thread from the QApplication's exec() call. The thread must be created using pthreads. You may use the Sieve of Eratosthenes algorithm to test for prime numbers.
10. When the search is terminated by the user the thread should be cancelled. Use deferred POSIX thread cancellation. You may use pthread_testcancel to cause the thread to cancel. Implement a cleanup routine, deallocate any memory that the thread dynamically allocated and print a message to the terminal (e.g. via stdout). A cancellation example is posted in Scholar.
11. Make sure that the search thread does not starve the GUI of processor time or cause the GUI to be otherwise blocked causing the interface to freeze. The progress bar, counter and the list of numbers must be updated as the search progresses. (This is a common use for threads in GUI applications)
12. After the search is stopped, whether by a normal exit or cancellation, if the user clicks on the "Start" button or action again then the search should start from the beginning. The QListView should be cleared, the counter reset and the progress bar start from 0% again.



How many prime numbers are there?

From: To:

Figure 1



How many prime numbers are there?

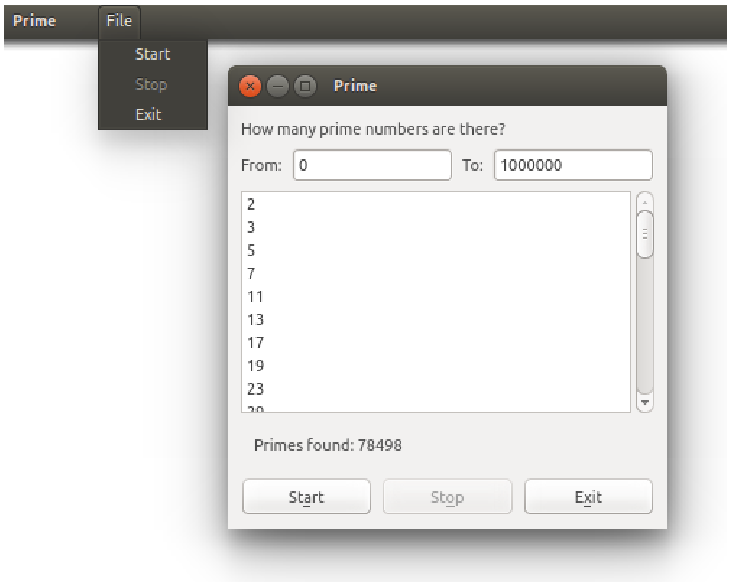
From: 0 To: 1000000

```
2
3
5
7
11
13
17
19
```

36%

Primes found: 29069

Figure 2



How many prime numbers are there?

From: 0 To: 1000000

```
2
3
5
7
11
13
17
19
23
```

Primes found: 78498

Figure 3

Notes

1. 0 and 1 are not prime but 2 is.
2. Once the user starts the search then the search itself, the displayed primes and the progress bar must not be affected by a change of the "From" and "To" values.
3. Be sure to read the most up-to-date documentation for QListView, QAbstractListModel and advisories in the documentation about subclassing QAbstractListModel.