CS 144
Advanced C++ Programming

Spring 2019
Instructor: Ron Mak

Assignment #6

Assigned:	Tuesday, March 5
Due:	Tuesday, March 12 at 8:30 AM
CodeCheck:	http://codecheck.it/files/19030421167mnnd4i5xoqm3a6fk0tu02pcw
Canvas:	Assignment 6. Big π
Points:	100

Big π
You will compute and print the first thousand decimal places of pi.

The purpose of this assignment is to give you practice downloading, configuring, building, installing, and using a C++ library, the Multiple Precision Integers and Rationals Library (MPIR). These are important skills to master for future classes and for work in industry.

The MPIR has both a functional and an object-oriented API (application programming interface). You will first use the functional API and then use the object-oriented API.

The Borwein algorithm
Your program will use the algorithm first published by mathematician brothers Jonathan and Peter Borwein in 1985:

Set $a_0 = 6 - 4\sqrt{2}$ and $y_0 = \sqrt{2} - 1$. Then iterate

$$y_i = \frac{1 - \sqrt[4]{1 - y_{i-1}^4}}{1 + \sqrt[4]{1 - y_{i-1}^4}}$$

$$a_i = a_{i-1}(1 + y_i)^4 - 2^{2i+1} y_i (1 + y_i + y_i^2)$$

and the a_i will converge quartically towards $1/\pi$.

This algorithm is quartic – each iteration increases the number of correct digits by a factor of 4. See https://en.wikipedia.org/wiki/Borwein's_algorithm "Quartic algorithm (1985)". Five iterations are sufficient to compute a thousand digits of π.

The Multiple Precision Integers and Rationals (MPIR) library
From the introduction in the library's documentation:

> MPIR is a portable library written in C for arbitrary precision arithmetic on integers, rational numbers, and floating-point numbers. It aims to provide the fastest possible arithmetic for all applications that need higher precision than is directly supported by the basic C types.

Like many C++ software meant for Linux systems (and similarly for Mac OS and Windows), MPIR is distributed in source form. Download the zip file for MPIR 3.0.0 and its documentation from http://mpir.org/downloads.html. After you unzip the file, you will have many source files and several shell scripts to configure, build, and install the header files and library. You run the scripts on a Linux or Mac system in a terminal window. If you are on a Windows system, you must run the scripts in the Cygwin terminal window, not the standard Windows command terminal.

Configure, build and install the MPIR header files and library
MPIR uses a conventional sequence of commands to configure, build, and install the library. See the "Installing MPIR" chapter of the documentation. First, you run the **`configure`** script which checks your system for necessary utility programs and generates the makefiles. Next, you run **`make`** which uses the generated makefiles to compile and assemble the source files and build the library. Then you run **`make check`** to compile and run test programs to verify that you properly built the library. Finally, you run **`make install`** to install the MPIR library **`libmpir*`** (several files) and its header file **`mpir.h`** in the standard locations **`/usr/local/lib`** and **`/usr/local/include`**, respectively.

The build is straightforward on Linux and Mac OS. To compile and run your program **`BigPi.cpp`** on the command line:

```
g++ -std=c++11 -lmpir -o BigPi BigPi.cpp
./BigPi
```

The **`-lmpir`** specifies using the MPIR library.

On some systems, you first need to set the environment variable **`LD_LIBRARY_PATH`** before running the compiled program

```
export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
```

If you put this line in your **`.bashrc`** file (note the leading dot) in your home directory, you won't have to do set this every time. After adding the line, either restart your terminal window or execute

```
source ~/.bashrc
```

Please work together to properly configure, build, and install the MPIR library on your computers. However, as always, your program must be individual work.

MPIR on Windows
Windows users should consider installing the VirtualBox virtual machine manager, and then installing Ubuntu as a virtual machine. See the tutorials at http://www.cs.sjsu.edu/~mak/tutorials/index.html : InstallVirtualBox.pdf, InstallUbuntu.pdf, InstallEclipse.pdf, and InstallMPIR.pdf. After installing Ubuntu, be sure to download and install the development tools (which include the C++ compiler). See "Install developer tools" on page 7 of InstallUbuntu.pdf.

With Windows 10, you can install and run Ubuntu without VirtualBox. See https://www.microsoft.com/en-us/p/ubuntu/9nblggh4msv6?activetab=pivot:overviewtab After installing Ubuntu, be sure to download and install the development tools (which include the C++ compiler). See "Install developer tools" on page 7 of InstallUbuntu.pdf.

If you insist on staying and working in Windows, you must do everything in the Cygwin terminal window in order to run the MPIR bash scripts. You must first download and install the utility programs that the build scripts need. These include **`make`** and the **`m4`** macro processor. You get them from the Cygwin website similarly to the way you got the GNU C++ compiler. If you are missing other utilities, **`configure`** will tell you. Then you must go back to the Cygwin website to download and install the missing utilities. You may need to do this several times before **`configure`** is finally happy.

When you run **`configure`** on Windows, you must include two arguments to tell it that you want to create a Windows DLL (dynamic-link library):

```
./configure --disable-static --enable-shared
```

See p. 13 of the MPIR manual. Then run the **`make`** commands the same way as described above for Linux and Mac OS.

When you compile and run a program on Windows, you may also need the **`-L`** option to specify the location of the library directory, for example:

```
g++ -std=c++11 -L"D:\mpir-3.0.0\.libs" -lmpir -o BigPi BigPi.cpp
BigPi
```

MPIR and Eclipse
To compile and run **`BigPi.cpp`** in Eclipse, you must specify using the MPIR library for the project. Right-click on the project and select Properties. Under Linker, select Libraries. Add **`mpir`** to the Libraries box.

The functions API and the object-oriented API
You will compute π twice using the Borwein algorithm.

First use the **`mpf_`** functions API as described in the chapter "Floating-point functions" of the MPIR manual. These functions are declared in the header file **`mpir.h`**.

Then use the object-oriented API as described in the chapter "C++ Class Interface". Instead of calling functions to perform arithmetic operations, you will use the overloaded arithmetic operators with MPIR objects. This API is declared in the header file **`mpirxx.h`**.

In both cases, use the chrono functions to time the algorithm. Do not include printing π in the timings.

Use **`const`** and **`inline`** as appropriate.

Expected output
Your output should be similar to the following. Print the thousand digits of π in blocks of ten digits, ten blocks per line, and two groups of five lines each. In both cases, the last block of digits should be **`2164201989`**. Your timings will be different, of course.

```
pi to 1000 places calculated with functions:

3.1415926535 8979323846 2643383279 5028841971 6939937510 5820974944 5923078164 0628620899 8628034825 3421170679
  8214808651 3282306647 0938446095 5058223172 5359408128 4811174502 8410270193 8521105559 6446229489 5493038196
  4428810975 6659334461 2847564823 3786783165 2712019091 4564856692 3460348610 4543266482 1339360726 0249141273
  7245870066 0631558817 4881520920 9628292540 9171536436 7892590360 0113305305 4882046652 1384146951 9415116094
  3305727036 5759591953 0921861173 8193261179 3105118548 0744623799 6274956735 1885752724 8912279381 8301194912

  9833673362 4406566430 8602139494 6395224737 1907021798 6094370277 0539217176 2931767523 8467481846 7669405132
  0005681271 4526356082 7785771342 7577896091 7363717872 1468440901 2249534301 4654958537 1050792279 6892589235
  4201995611 2129021960 8640344181 5981362977 4771309960 5187072113 4999999837 2978049951 0597317328 1609631859
  5024459455 3469083026 4252230825 3344685035 2619311881 7101000313 7838752886 5875332083 8142061717 7669147303
  5982534904 2875546873 1159562863 8823537875 9375195778 1857780532 1712268066 1300192787 6611195909 2164201989

  Elapsed time: 0.00165475 seconds

pi to 1000 places calculated with operators:

3.1415926535 8979323846 2643383279 5028841971 6939937510 5820974944 5923078164 0628620899 8628034825 3421170679
  8214808651 3282306647 0938446095 5058223172 5359408128 4811174502 8410270193 8521105559 6446229489 5493038196
  4428810975 6659334461 2847564823 3786783165 2712019091 4564856692 3460348610 4543266482 1339360726 0249141273
  7245870066 0631558817 4881520920 9628292540 9171536436 7892590360 0113305305 4882046652 1384146951 9415116094
  3305727036 5759591953 0921861173 8193261179 3105118548 0744623799 6274956735 1885752724 8912279381 8301194912

  9833673362 4406566430 8602139494 6395224737 1907021798 6094370277 0539217176 2931767523 8467481846 7669405132
  0005681271 4526356082 7785771342 7577896091 7363717872 1468440901 2249534301 4654958537 1050792279 6892589235
  4201995611 2129021960 8640344181 5981362977 4771309960 5187072113 4999999837 2978049951 0597317328 1609631859
  5024459455 3469083026 4252230825 3344685035 2619311881 7101000313 7838752886 5875332083 8142061717 7669147303
  5982534904 2875546873 1159562863 8823537875 9375195778 1857780532 1712268066 1300192787 6611195909 2164201989

  Elapsed time: 0.00139462 seconds
```

Not for CodeCheck
Because of the MPIR library, you will not be able to compile and run your program in CodeCheck. Go to the CodeCheck URL above to obtain the suggested skeleton of your program. But otherwise you must edit, compile, and debug outside of CodeCheck.

Submission into Canvas
Submit into Canvas a copy of your C++ program and a text file of its output:
Assignment 6. Big Pi

Rubric
Your program will be graded according to these criteria:

Criteria	Max points
Correct output • Functions API • Object-oriented API (overloaded arithmetic operators)	**30** • 15 • 15
Good use of the MPIR library • Constant values computed outside of the iterations. • Calls to the functions API. • Use of the overloaded arithmetic operators.	**50** • 10 • 20 • 20
Good program style • Good functional decomposition with functions declared before the main and defined after the main. • Use of `const` and `inline`. • Descriptive variable names and meaningful comments.	**20** • 5 • 5 • 10

A million digits of π?
If it takes five iterations to compute pi accurately to a thousand digits, what is the fewest number of iterations to compute π accurately to a million digits? Remember that this is a quartic algorithm. How long did it take to compute those million digits on your computer?

To verify your results, see https://www.exploratorium.edu/pi/numbers-of-pi
Checking only the last ten digits should be sufficient!

Academic integrity

You may study together and discuss the assignments, but what you turn in must be your individual work. Assignment submissions will be checked for plagiarism using Moss (http://theory.stanford.edu/~aiken/moss/). **Copying another student's program or sharing your program is a violation of academic integrity.** Moss is not fooled by renaming variables, reformatting source code, or re-ordering functions.

Violators of academic integrity will suffer severe sanctions, including academic probation. Students who are on academic probation are not eligible for work as instructional assistants in the university or for internships at local companies.