San José State University
Department of Computer Engineering

CMPE 180A

Data Structures and Algorithms in C++

Fall 2020

Instructor: Ron Mak

Assignment #9

Assigned:	Tuesday, October 20
Due:	Tuesday, October 27 at 5:30 PM
Canvas:	Assignment 9: Big Pi
Points:	100

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

The purpose of this assignment is to give you practice downloading, configuring, building, installing, and using a C++ library. You may be asked to perform such tasks in future classes, so these are important skills to master.

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)”.

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 MacOS X), MPIR is distributed in source form. Download the zip file for MPIR 3.0 and its documentation from http://mpir.org/downloads.html. After you unzip the file, you will have many C and assembly language source files and several shell scripts. Run the scripts on a Linux or Mac system in a bash terminal window.

If your platform is Windows 10, you should use the Windows Subsystem for Linux to install the Ubuntu distribution of Linux. See https://docs.microsoft.com/en-us/windows/wsl/install-win10 and https://www.microsoft.com/en-us/p/ubuntu-1804-lts/9n9tngvndl3q?rtc=1#activetab=pivot:overviewtab.

Build and install the libraries

MPIR uses a conventional sequence of commands to build and install its libraries. See the "Installing MPIR" chapter of the documentation.

First, you run the **`configure`** script which checks your system for necessary utility programs. It generates the makefiles if your system checks out OK. Be sure to specify the **`--enable-cxx`** option to build both the C and C++ libraries.

Next, you run **`make`** which uses the generated makefiles to compile the C source files and build the libraries. Then you run **`make check`** which compiles and runs test programs to verify that you properly built the libraries. Finally, you run **`make install`** to install the MPIR libraries **`libmpir*`** (several files) in the standard libraries directory **`/usr/local/lib`** and install the header files **`mpir.h`** (for the C API) and **`mpirxx.h`** (for the C++ API) in the standard include directory **`/usr/local/include`**.

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

```
g++ BigPi.cpp -lmpir -o BigPi
./BigPi
```

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

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

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

Useful tutorials

See http://www.cs.sjsu.edu/~mak/tutorials/index.html

- "Install Ubuntu on Windows 10 and on VirtualBox"
- "Configure Ubuntu for Software Development"
- "Install MPIR on Ubuntu and MacOS X"

C and C++ versions with timings

Be sure to build both the C and C++ libraries to get both the C and C++ APIs. (When you run the **`configure`** script during installation, be sure to specify the **`--enable-cxx`** option.)

Compute the thousand decimal digits of π twice:

- In your first version, make calls to the **`mpf_`** functions from the C library.
- In your second version, use the overloaded arithmetic operators from the C++ library.

Use **`std::chrono`** to time each version's computation. Don't include the time to print the results.

You may find it easier to write two separate versions of your program rather than to combine them into one program.

Sample output

Here is sample output. Note how the thousand digits are formatted.

```
C API: pi to 1,000 places:

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: 391,271 nsecs = 0.000391271 seconds


C++ API: pi to 1,000 places:

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: 151,405 nsecs = 0.000151405 seconds
```

Not for CodeCheck
Because of the MPIR library, you will not be able to compile and run your program in CodeCheck. Therefore, you must edit, compile, and debug outside of CodeCheck.

Submission into Canvas
Submit into Canvas a copy of your C++ program(s) and a text file of its output:
Assignment 9: Big Pi

Rubric
Your program will be graded according to these criteria:

Criteria	Maximum points
Correct output (the final four digits of the thousand should be 1989)	25
Good use of the MPIR libraries • Constant values computed outside of the iterations. • Good library calls inside the iterations (C version). • Good use of the overloaded arithmetic operators (C++ version).	50 • 10 • 20 • 20
Good program style • Descriptive variable names. • Good functional decomposition. • Meaningful comments. • Follow the coding style (formatting, braces, indentation, function declarations before the main, etc.) of the Savitch textbook.	25 • 5 • 10 • 5 • 5

A million digits?
Can you compute and print the first million decimal digits of π? You can check your results with Dr. Evil at
http://3.141592653589793238462643383279502884197169399375105820974944592.com/index314159.html

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.