Lab 2: Bakhshali Approximation for Computing Square Roots

Finding the positive square root of a positive number (e.g., $\sqrt{5}$) is an example of a problem that requires a *numerical* solution. There is no formula that expresses the square root of an arbitrary number S in an exact way as a function of S. However, there are many methods for *numerically* computing square roots. A *numerical computation* is an approximation to the solution. Algorithms that compute successively better approximations to the solution are called *approximation methods*, or *approximation algorithms*.

In this lab exercise you will write a program that implements one particular square root approximation algorithm. It is a variation of a method known as the *Bakhshali approximation algorithm*. The Bakhshali method for finding an approximation to a square root was described in an ancient Indian mathematical manuscript called the *Bakhshali manuscript* (see the Wikipedia page on Methods of computing square roots for details). The Bakhshali method is really just two iterations of a more general, and ancient, algorithm dating back to the Babylonians. In this assignment, we will extend the Bakhshali method so that it can be iterated indefinitely, like the Babylonian method. The description of the algorithm follows.

The Bakhshali algorithm to compute the square root of a number S is defined by the following steps:

1. Make an initial *guess* at the square root by picking any number smaller than S.
2. Compute the difference d between the square of your *guess* and S:
$$d = S - guess^2.$$
3. Compute the ratio r of d and twice the *guess*:
$$r = \frac{d}{2 \cdot guess}.$$
4. Compute sum of the *guess* and the ratio r:
$$a = guess + r.$$
5. Compute your next *guess* as
$$guess = a - \frac{r^2}{2a}.$$
6. Go back to step 2 using the *guess* just computed.

Steps 2 through 6 are repeated as many times as the implementer sees fit. It can be proven mathematically that the *guess* gets closer and closer to the actual square root of n as the number of iterations increases[1]. But the process needs to stop somewhere for it to be useful.

[1] In fact, one can show mathematically how much more accurate it gets each time, but that fact will be a secret for now.

Exercises

Problem 1. Write a program that reads a positive integer for S from the user. You can assume that the user will always enter an integer, but the program does have to detect if the user entered a negative integer, and if she did, it should exit with an error message. The program then applies five iterations of the above Bakhshali algorithm to compute the square root of S. Your program should display the answer with ten decimal places (if there are fewer than ten digits after the decimal point, your program should display zeros). How will you choose the initial guess?

Problem 2. Modify the program from *Problem 1* so that the computed answer is more accurate. Execute as many iterations as are needed until the values computed in two consecutive iterations differ in absolute value by less than 0.0000001. For example, when oen iteration is 2.23606783 and the next is 2.23606791 the difference is 0.00000008. Add a counter to your program that keeps track of the number of iterations needed before the loop terminates. Output the number of iterations together with the answer. What happens as you choose to compute square roots of larger and larger integers? How can you make this second program more general? If you can think of a way, add that to a comment in the program. If not, do not worry about answering it.

Testing Your Work

Before you hand in your programs, you have to make sure that they work correctly. For these two programs, this means that when they are given a number, their output should be an approximation to its square root. One way to test is to manually square the answer to make sure it is close to the original number. Another way is to use a calculator to verify the answer. Still a third way is to pick perfect squares as input, so that their square roots are whole numbers. For example, when given the input 144, your program ought to produce something close to 12.

The second program is easier to test than the first, because it is supposed to be accurate to a specific number of decimal places. If your program's result is not that accurate, it is not correct. The first is harder, because the accuracy is not specified, only the number of iterations. In this case just make sure it really iterates five times as specified.

What to Submit

Submit both programs by the end of today's lab, i.e., before the end of the class at 2:00 P.M. To do this,

1. Create a directory in

   `/data/biocs/b/student.accounts/cs135_sw/cs136labs/lab02/submissions`

   whose name is your *username*. For example, I would create the directory `sweiss`.

2. Copy the two files, which should be named *username*`_lab02a.cpp` and *username*`_lab02b.cpp` to this directory. The first program is *username*`_lab02a.cpp` and the second is *username*`_lab02b.cpp`. It is critical that you name these properly. You will lose 5% of the grade if you misname either file!

3. Change the permission on the directory that you created so that no one else can read or modify it. You do this with the commands

```
$ cd /data/biocs/b/student.accounts/cs135_sw/cs136labs/lab02/submissions
$ chmod 700 username
where username is the name of the directory you created.
```

Do not submit executable files. (Make sure that your files have the .cpp extension.) Remember to document your code (both preamble and comments in the code) and make it easy to read. Your work will be judged using the rubric I outline in the Programming Rules document.

If you do not complete all the tasks, submit whatever you have by the end of class. There are absolutely no extensions to the deadline. You can submit a revised version of either program by 8:00 P.M. on Friday, Sept. 7. If you do, you must name it with a different name than what you first posted, e.g., *username*`_lab02a_v2.cpp`. and put it in the same directory as the original programs. Code submitted after the end of class will receive only partial credit and must look like a revision of the original file. If you do this, you must send email to me before 8:00 P.M. at myHunter address, telling me to look there!