Assignment week 3: Approximating the square root

Course 'Imperative Programming' (IPC031)

Make sure to start working on the assignment **before the lab** session, otherwise you will be too late for the deadline

1 Background

In the first two assignments you have developed algorithms for Charles-related problems, using control-structures to coordinate the function-units that have been developed in a top-down or bottom-up way. Starting with this assignment, we create console input/output-based applications.

2 Learning objectives

After doing this assignment you are able to:

- create and use void-functions that have parameters;
- work with standard elementary data types (`bool`, `int`, `double`, `char`, `string`);
- work with console based input (`cin`, `>>`) and output (`cout`, `<<`);
- create and compile such an application.

3 Assignment

Computing the square root of a number is usually a built-in function in most programming languages. C++ is no exception, and uses `sqrt` for this purpose. In this assignment you implement two methods yourself that approximate the square root $\sqrt{v}$ of a given positive floating point value v (represented as a `double`).

> You use only the basic arithmetic operations `+` , `-`, `*`, `/`, `abs`, comparison operations `<`, `<=`, `==`, `>`, `>=`, functions `min` and `max`, and control structures.

Both methods work with a sequence of approximations x_0, x_1, $\ldots x_n$ (with $n \geq 0$), with the intention that eventually x_n is sufficiently close to $\sqrt{v}$. The halting-criterion is the desired precision with which the value should be approximated. Traditionally, the precision, which is denoted by ϵ, is a very small, positive value ($0 < \epsilon \ll 1$). In sum, we are looking for an x_n such that:

$$|x_n \cdot x_n - v| \leq \epsilon.$$

This value (x_n) can be computed by two algorithms (see Part 2 and Part 3).

Part 1: Desktop test cases

The algorithms are already given in this assignment. To check your understanding of the algorithms, you should do a desktop test of the following values for v (set ϵ to 0.1):

- $v = 0$,
- $v = 1$,
- some v for which $0 < v < 1$,
- some $v > 1$ for which there is a natural number $n > 0$ such that $n \cdot n = v$, and
- some $10 < v < 100$ that does not have the previous property.

Include the results of these desktop tests as comment in "`main.cpp`".

Part 2: Inclusion

Design and implement the function `void inclusion (double eps, double v)` that approximates $\sqrt{v}$ through a sequence of pairs of values $(a_0, b_0), (a_1, b_1), \ldots, (a_n, b_n)$ having the property:

$$a_i \cdot a_i \leq v \qquad \text{and} \qquad b_i \cdot b_i \geq v$$

The value of a_0 is 0. The value of b_0 is the maximum of v and 1. If a_0 happens to be the square root of v (that is, $a_0 \cdot a_0 = v$), then you are done (and a_0 is the result). If b_0 happens to be the square root of v (that is, $b_0 \cdot b_0 = v$), then you are done (and b_0 is the result). In these cases, print the message: Inclusion square root of v is (with the value of v) followed by the value of the variable carrying the result. Otherwise, you iterate over i. In iteration i, the average of the values a_i and b_i is $x_i = (a_i + b_i)/2$. If x_i satisfies the halting-criterion (that is, $|x_i \cdot x_i - v| \leq \epsilon$), then you are done (and x_i is the result). Otherwise, you continue with:

$$(a_{i+1}, b_{i+1}) = \begin{cases} (x_i, b_i) & \text{if } x_i \cdot x_i < v \\ (a_i, x_i) & \text{otherwise} \end{cases}$$

When you find a solution for x_i, print the message: Inclusion square root of v is x_n for epsilon ϵ (with the values of v, x_n, and ϵ)

Part 3: Newton-Raphson

The classic Newton-Raphson algorithm (published by Isaac Newton at the end of the 17th century) computes the zero-value of a given function f using a sequence of approximations (it computes a value x_n, such that $f(x_n) \approx 0$). It works as follows: if x_i is an approximation of the desired zero-value, then you can compute a better approximation x_{i+1} with:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}, \text{ where } f' \text{ is the derivate function of } f.$$

The algorithm can be used to compute the square root of a value v by choosing $f(x) = x^2 - v$. The derivate function of f is $f'(x) = 2x$.
Design and implement the algorithm by the function `void newton_raphson (double eps, double v)`. The value of x_0 is the maximum of v and 1. The final value x_n satisfies the property $|x_n \cdot x_n - v| \leq \epsilon$ in the same way as in Part 2. Finally, the function prints the message: Newton Raphson square root of v is x_n for epsilon ϵ (with the values of v, x_n, and ϵ).

Part 4: Comparing the algorithms

Compare the effectiveness of the `inclusion` and `newton_raphson` implementations. Do this by counting how many approximations x_0, x_1, $\ldots x_n$ are generated by both algorithms when applied to each desktop test case that you have developed in Part 1. Add your results in comment in "`main.cpp`". The output of the algorithm must be extended as follows: on each line of output, the values of n, a_n, x_n, b_n are shown in the case of `inclusion` and the values of n, x_n in the case of `newton_raphson`, where each value is separated with a 'tab'-character: (`'\t'`).

4 Products

As product-to-deliver you only need to upload to Brightspace "`main.cpp`" that you have created with solutions for each part of the assignment.

Deadline

Lab assignment: Friday, September 20, 2024, 23:59h

Important notes:

1. check that you have actually submitted your solution in Brightspace.
2. the deadline is firm, and it is impossible to upload a solution after expiry. In that case you fail the assignment.
3. you can upload solutions to Brightspace several times, but only the last submission is stored.
4. identify yourself and your lab partner in every uploaded document. The identification consists of your first and last names, student numbers, and number of (sub) assignment. By identifying yourself, you declare that the uploaded work has been created solely by you and your lab partner.
5. your work will be judged and commented upon. We expect that you obtain the feedback, read it, and use it to for the next exercises.
6. it is essential that you only submit your own solution, never copy somebody/something else's solution, and never share your solution—in particular: **AI tools (including but not limited to Github Copilot or ChatGPT) are not permitted**, solutions from previous year cannot be reused, and finally, you and your lab partner take joint responsibility for the assignment you submit.