Technische Universität München

Informatics I for MSE

MSE IN8011
Prof. Bjoern Menze, Giles Tetteh
Philipp Kaeß

WS 19/20
Homework 1

Due Date: 10.11.19, 23:59

Release Date: 01.11.19

Homework 1 : N-th Partial Sum

Topics: Datatypes, `int`-/`float` Division, Loops, `double` Datatype, Value/Iteration Borders

Introduction

The n-th partial sum of the harmonic series is defined as:

$$H(n) = \sum_{k=1}^{n} \frac{1}{k} = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}$$

For the example of $n = 4$ this yields:

$$H(4) = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} = \frac{25}{12} \approx 2.083$$

Even though one could think that this series will stagnate at some point and for $n \to \infty$ the value of $H(n)$ will converge to some value, this is actually not the case. This function diverges, which means that if one only chooses n high enough, the value of $H(n)$ can outrun any positive barrier. The proof for this can be done analytically, but today we want to show this (at least for moderately high values) numerically.
For this homework, you will write two functions: One function for calculating $H(n)$ for some given n, and one function for calculating the minimum value n, that is needed to reach a given $H(n)$.

Specifications

When the function `calc_h(int n)` is called by a main function, the following should happen:

- The n-th partial harmonic sum is calculated according to the above definitions and returned at the end of the function
- The resulting value has to be the correct mathematical approximation for any possible (32-bit)-int value as input n ($n \leq 2\,147\,483\,647 = 2^{31} - 1$)

When the function `calc_n(double h)` is called by a main function, the following should happen:

- The smallest n, for which $H(n) \geq h$ holds, is calculated and returned
- The resulting value has to be correct for any $h \in [0,20]$ (including)

Tool Box

The explanations below **may** help you to solve this problem.

LOOPS

You will need at least one type of loop in this exercise. You have the choice between a for-loop and a while-loop. Both types are possible, and their usage is a matter of personal preference. You can find explanations and examples of both types in the first exercise sheet.

DOUBLE

As you can imagine, for high values of n the summands ($\frac{1}{n}$) will become very small. For the above defined range of values, it will **not** be sufficient to use `float` numbers. They only have four bytes and for a sufficiently (still in `int`-range) high n the summands will become zero, as the value is smaller than a `float` could represent.

Therefore, use the 8-byte datatype `double` for your summands. For the integers, the datatype `int` will be sufficient.

INT-FLOAT DIVISION

The division in C is a little bit trickier than in real life, due to the different datatypes. An `int`-division (both values `int`) will yield a result of type `int`, and therefore isn't always mathematically correct. E.g. the division (`5/2`) will yield the result `2`, even though the correct result would be `2.5`. The digits after the point are truncated (**not** rounded). This holds not only for the division of direct `int` values, but also for **variables** of type `int`.

As the type `float` is "stronger" than `int`, as soon as at least one part of the division is a `float`, it will be a `float`-division and the result will be the mathematically correct result as `float`. You can achieve this by converting one of the division parts into a `float`. This is done by a so-called type cast, where you put the desired datatype in brackets in front of the variable (`5/(float) 2`). Alternatively, when using direct digits, you can use a .0 to trigger a `float`-division (`5.0/2`).

The exact same thing also holds for `double`-values, as this is basically just a larger `float`.

Tasks

We provide you the template file ***partial_sum.c*** on Moodle. Complete the functions `calc_h()` and `calc_n()`, so that the program fulfills the specifications above. Please note:

- You are only allowed to write code in the spot indicated in the functions. Do not change anything else in the functions. Write your name at the designated spot in the comment block at the beginning of your file.
- You are provided with an additional C-File ***main_partial.c***. This file contains a main-function, which you can use to test your function. Therefore, store the two C-Files in the same directory on your computer, and compile and run **main_partial.c**.

- The main-function will then test both of your functions with five testcases each, and compare the values your functions return with the mathematically correct solutions. If any wrong value occurs, the respective values will be printed. Please note that these testcases are only for your help, and do **not** represent the testcases we will use to correct and grade your homework

- When you are finished, put the ***partial_sum.c*** file **directly** in a zip-File named "HW1_Mustermann_Max" (with your name of course) and upload this file to Moodle. Do NOT upload your main-File and do NOT put any folders into the zip-File.

Threshold

To pass this homework, your program has to fulfill the following conditions:

- Your C-File must be compilable and runnable without any errors and with the flags -Wall and -Werror enabled
- Your functions have to pass at least 80% of our testcases in the above defined range of values with the accuracy of `double` representations
- Your code mustn't get stuck in an infinite loop (computation time has to be less than 20 seconds per test case) or crash for an input value in the range specified above