CPSC 1160 SECTION 001 INSTRUCTOR: GLADYS MONAGAN

ASSIGNMENT #1: C++ BASICS/FUNCTIONS

DUE DATE WITH BRIGHTSPACE: WEDNESDAY, SEPTEMBER 14, 2022 AT 11:50 PM

THE LEARNING OUTCOMES

- write a complete C++ program: compile, link, and run the program in C++17
- write a function using functional abstraction and structured programming
- test user input and report errors
- refer to documentation on how to format console output
- unit test a with the lightweight unit testing framework doctest

READINGS

Read in Brightspace **Useful Information | About the Course**

Read chapters 1-6 of the textbook.
Consult §4.10 for formatting the output (we will not cover this in class as such)
For now, you can skip §3.9, §4.4, §4.5, §4.7, §5.12, §6.14.
Peruse §5.13 but note that these are big "no-no"s.

PROBLEM AT HAND

Write a program to gauge the rate of inflation for the past year.
The program asks for a price (such as the price of a phone plan or of a latte) both one year ago and today. It estimates the inflation rate as the difference in price divided by the year-ago price.
Your program should let the user repeat this calculation as often as the user wishes.
Define a function to compute the rate of inflation.

The inflation rate should of type double and printed out to standard output as a percentage
e.g. the output is 5.32 for 5.32%

from Walter Savitch's book *Absolute C++* Programming Project #2 of chapter 3

IMPLEMENTATION OF THE PROGRAM THAT INTERACTS WITH A USER

Write a C++ program

- to ask the user to input two decimal number corresponding to the price a year ago and the price today. Assume that the input is numeric.
- to verify that the given numbers are viable dollar amounts: should negative decimals be allowed? And what about 0?
- to calculate the inflation rate
- to output to standard output what the price was a year ago, what it is now and what the inflation rate is as a percentage:
  output the inflation rate with at most two significant digits after the decimal point
- to ask the user whether to continue calculating the inflation or not

Deal with input errors gracefully, giving the user some feedback about the error. Do not just "exit" the program if there is an error.

Program procedurally. Your program should be well structured:

- Do not use **break** unless you are using a **switch** statement
- Do not use **continue**
- Do not use **while (true)** but you can and should use **while** loops or **do while** loops or **for** loops as appropriate
- Do not use **exit**
- You should have enough comments in the source code so that they suffice as internal documentation. You do not need to submit any external documentation.
- Document the function(s) properly with precondition, postcondition, etc.

Use the "incorrect" file include presented in the lab.

IMPLEMENTATION OF A PROGRAM UNIT TESTS

Write a C++ program that unit tests using doctest the function that calculates the inflation. It should be the same function in the same file that was used in the program that interacts with the user.
Use the "incorrect" file include presented in the lab.

TO SUBMIT WITH BRIGHTSPACE AS A SINGLE ZIP FILE:

1. A file called **inflation.cpp** that contains a function to calculate the inflation.
   Document the function.

2. A file that implements the program that interacts with the user and has #include inflation.cpp
   Document the program.

3. Some "screen captures" or "print screens" that show your program of 2 works e.g. how it is rejecting poor input. Put the screen captures in a MS Word document or a pdf.

4. A file that contains the unit tests and has #include inflation.cpp.
   The documentation in this file will come from the names of the TEST_CASEs

PARTIAL MARKING SCHEME OUT OF 100

- [/10] Function that calculates the inflation in a file called inflation.cpp
  - [/5] Comments including precondition and postcondition
  - [/5] Correct computation
- [/35] Program that uses doctest
  - test cases to test the function that computes the inflation indicating why those test cases were chosen
    - e.g. special cases
    - e.g. border cases
    - e.g. typical cases large inflation
    - e.g. typical cases for deflation (negative inflation)
- [/35] Program reads the values, calls the function to calculate the inflation, does the output properly and asks the user to continue. The input values to the function need to be validated. Screenshots show that the program works.
- [/20] Programming Style
  - [/5] Modularity and Structured Programming
  - [/5] Internal Documentation
  - [/4] Descriptive variable names
  - [/2] Consistent, conventional variable , constants, and function naming
  - [/2] Proper indentation and formatting of code
  - [/2] Named constants e.g. for the number of decimal places required

Penalties

- [-40] Does not follow the file formats required
- [-20] Has extra #include files
- ….