CPSC 1160 SECTION 001 INSTRUCTOR: GLADYS MONAGAN

ASSIGNMENT #1: C++ BASICS/FUNCTIONS

DUE DATE WITH BRIGHTSPACE: WEDNESDAY, JANUARY 11, 2023 AT 11:50 PM

THE LEARNING OUTCOMES

- write a complete C++ program: compile, link, and run the program
- 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 with the lightweight unit testing framework doctest

READINGS

Read in Brightspace **Content | Useful Information | About the Course**
Read in Brightspace **Content | Useful Information | Technically | doctest**

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[1]

Whether we like it or not, there is inflation.
Write a program **calculator** to calculate the rate of inflation for the past year.
calculator asks for a price both one year ago and today.
calculator calculates the inflation rate as the *difference in price* divided by the *year-ago price* and it outputs the rate of inflation as a percentage.
Your program **calculator** should let the user repeat this calculation. If the user enters a non-viable number, the program should output an error message and still continue asking the user whether to calculate the inflation again or not.
Here is an example which you do not need to reproduce exactly.

```
Enter price from a year ago: $ 77
Enter current price of the same item: $ 154
the inflation rate is  100.00%
Want to calculate inflation rate again? (y/n): y

Enter price from a year ago: $ 2.45
Enter current price of the same item: $ 2.95
the inflation rate is  20.41%
Want to calculate inflation rate again? (y/n): n
```

[1] based on Walter Savitch's book *Absolute C++* Programming Project #2 of chapter 3

IMPLEMENTATION OF THE PROGRAM THAT INTERACTS WITH A USER

Use the files provided filling in the needed code.

Write a C++ program **calculator**

- to ask the user to input two decimal number corresponding to the price a year ago and the price today. Assume that the input values given by the user are 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
  the inflation rate should be calculated in the function called **inflation** using the function prototype given in the file **inflation.cpp**
  document the precondition and the postcondition of the function **inflation**
- 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.

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.

IMPLEMENTATION OF A PROGRAM UNIT TESTS

Write a C++ program **unittest_inflation.cpp** that unit tests the function **inflation** found in the file **inflation.cpp** using doctest

You need to download the file **doctest.h** as we did in the lab and/or as documented in
Content | Useful Information | Technically | doctest
where we also show a TEST that compares doubles.

Do not pass as arguments to the inflation values that are not-viable: in other words, the input values to the function inflation that is being TESTed are all possible values that will compute an inflation. The testing of errors with bad input is being done in **calculator** not in the unit tests.

TO SUBMIT WITH BRIGHTSPACE AS A SINGLE ZIP FILE:

1. The file **inflation.cpp** that contains a function to calculate the **inflation**.
   Document the function.
2. A file **calculator.cpp** that implements the program that interacts with the user.
   Keep the #include inflation.cpp as given in the file.
   Document the program.
3. Some "screen captures" or "print screens" that show that your program **calculator** works for at least the following prices
   price a year ago: 2.45 price now: 2.95
   price a year ago: 1.82 price now: 1.55
   price a year ago: 77 price now: 154
   plus at least two cases of erroneous input.
   Put the screen captures in a pdf (you can put them in a Word document and then save the Word document as a pdf).
4. The file **unittest_inflation.cpp** that implements the unit test cases using doctest.
   Keep the #include inflation.cpp as given in the file.
   The documentation in this file will come from the names of the TEST cases.
5. The file doctest.h
6. Possibly a README.txt

Do not submit any executables nor project files.

PARTIAL MARKING SCHEME OUT OF 100

- [/10] Function **inflation** that calculates the inflation in a file called **inflation.cpp**
  - [/5] Comments including precondition and postcondition
  - [/5] Correct computation
- [/40] The tests in the file **unittest_inflation.cpp** that uses doctest
  - test cases to test the function that computes the inflation indicating <u>why</u> those test cases were chosen
    - e.g. special cases
    - e.g. border cases
    - e.g. typical cases for the equivalence classes
  - compare the double values properly (or more like approximately)
- [/35] The program **calculator** that reads the values, calls the function to calculate the inflation, does the output with at most 2 decimal places after the decimal point, properly and asks the user to continue.
  The input values to the function need to be validated.
  Screenshots showing that the program works.
- [/15] Programming Style
  - [/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 format and interface required
- [-20] Has extra #include files
- [-20] The file doctest.h is missing
- ….