Homework 0

Due by: 19/6/2024 23:59

In this homework you will practice C++ features we will use in class and future homework. Please review the "C++ essential" presentation for background.

Guidelines:

1. See the general guidelines from HW-1.
2. Make sure your code #include the header files (.h files) provided with this homework.
3. In total, you should upload (together) three files to gradescope:
    a. solution.pdf file (with answers to all the "additional questions" (see ahead))
    b. lambda.cpp file
    c. cdf.h file
4. Make sure you use the exact filenames above.
5. Make sure you use the exact function and class names mentioned below. Otherwise, you will face compilation errors during automatic testing.
6. This homework assignment is only partly auto-graded. You will see the total points for the programming exercises immediately after auto grading. However, your final grade will be available only after your answers to the *additional questions* part will be manually graded. This will happen only after the due date.

Programming Exercises

Lambda functions (30 points)

Review the header file *lambda.h* before you work on this exercise.

1. Exercise 1: [20 points] Write the *init_globals* function to:
    a. Define a lambda function which gets a vector of integers by value and returns a new vector that contains only the even numbers from the input vector.
    b. Assign that lambda function to the extern global variable *by_value*.
    c. Define a modified version of this lambda function from (a) which gets a vector of integers by reference and returns a new vector with only the even numbers from the input vector.
    d. Assign that lambda function to the extern global variable *by_ref*.
    e. Write a lambda function that compares two integers by their absolute values. This lambda function gets a global integer counter and increment it per each compare operation.
    f. Assign that lambda function to the extern global variable *cmp_lambda.*

    Please note: the global variables are already defined as extern variables in the *lambda.h* file (the test framework takes care to create them). Don't try to redefine them in your submitted code (as this will fail your submission).

2. Exercise 2: [10 points] Write the *ascending_sort* function to sort a vector of integers in ascending order of their absolute values, using the C++ STL sort.
    g. This function should use the lambda function assigned to *cmp_lambda* in *init_globals()*. (hence, you may consider calling the *init_globals()* before the *ascending_sort* function)
    h. The *ascending_sort()* should return the sorted vector should print the total amount of comparison operations applied during the sort.

Please note: don't use other functions than C++ STL sort, as you may get a different number of comparisons than expected by the test framework).

Upload a "*lambda.cpp*" file with only the above two functions [*init_globals()* and *ascending_sort()]* to gradescope. Avoid uploading additional code like your *main()* or global variable definitions. Our testing system will link to these two functions to test them automatically.

Functors (30 points)

Review the header file *functor.h* before you work on this exercise.

1. Exercise 3: [15 points] Define a sub-class named *Accumulator* which inherit from the *IAccumulator base class. This function should be used as a* functor (meaning, you should overwrite its *operator()*) that gets a float number as an input argument, adds it to a total sum, and returns the current total sum. Initialize total sum to 0.
2. Exercise 4: [15 points] Write the *cdf* function that gets a PDF vector (you can always assume a valid PDF vector) and the functor as parameters and returns a CDF vector from the input PDF vector.

Upload a "***cdf.h***" file with the *cdf* function and the functor class to gradescope. Avoid uploading additional code (like your *main()*)

Additional questions

The following are several text questions only. Include your answers to all the below questions in a *solution.pdf* file.

smart pointers (30 points)

1. Question 1: [10 points] Describe at least one pro and one con of smart pointers, compared with raw pointers.
2. Question 2: [20 points] Create a simple program that shows a use-case whose performance (execution time) with raw pointers is better than with smart pointers.
   a. You are only allowed to use the same exact program and only replace raw pointers with smart pointers.
   b. Use time measurements in C++ or external tools like gprof (you can easily install in your Linux machine). See the "c++ essential" presentation for time measurements.
   c. Make sure your program is small and focuses only on the specific use case you want to demonstrate. Measure execution times of relevant parts of the program to demonstrate your use-case.
   d. Share your source code (its important parts) and the time measurements you printed to stdout in the *solution.pdf* file. Add explanations, as needed (but first, try to make everything simple and straightforward).

Functors, function pointers and lambda functions (10 points)

Compare functors, function pointers and lambda functions.

3. Question 3: [5 points] Describe at least one case where using functors may be preferable (have an advantage) over 1) lambda functions and 2) over function pointers.

4. Question 4: [5 points] Describe at least one case where using functors is less preferable (has no clear advantage or clear disadvantage) over lambda functions and function pointers.

Good luck!