Important message on plagiarism

The single most important point for you to realize before the beginning of your studies at ShanghaiTech is the meaning of “plagiarism”:

Plagiarism is the practice of taking someone else's work or ideas and passing them off as one's own. It is the misrepresentation of the work of another as your own. It is academic theft; a serious infraction of a University honor code, and the latter is your responsibility to uphold. Instances of plagiarism or any other cheating will be reported to the university leadership, and will have serious consequences. Avoiding any form of plagiarism is in your own interest. If you plagiarize and it is unveiled at a later stage only, it will not only reflect badly on the university, but also on your image/career opportunities.

Plagiarism is academic misconduct, and we take it very serious at ShanghaiTech. In the past we have had lots of problems related to plagiarism especially with newly arriving students, so it is important to get this right upfront:

You may…

• … discuss with your peers about course material.
• … discuss generally about the programming language, some features, or abstract lines of code. As long as it is not directly related to any homework, but formulated in a general, abstract way, such discussion is acceptable.
• … share test cases with each other.
• … help each other with setting up the development environment etc.

You may not …

• … read, possess, copy or submit the solution code of anyone else (including people outside this course or university)!
• … receive direct help from someone else (i.e. a direct communication of some lines of code, no matter if it is visual, verbal, or written)!
• … give direct help to someone else. Helping one of your peers by letting him read your code or communicating even just part of the solution in written or in verbal form will have equal consequences.
• … gain access to another one’s account, no matter if with or without permission.
• … give your account access to another student. It is your responsibility to keep your account safe, always log out, and choose a safe password. Do not just share access to your computer with other students without prior lock--out and disabling of automatic login functionality. Do not just leave your computer on without a lock even if it is just for the sake of a 5--minute break.
• … work in teams. You may meet to discuss generally about the material, but any work on the homework is to be done individually and in privacy. Remember, you may not allow anyone to even just read your source code.

With the Internet, "paste", and "share" are easy operations. Don’t think that it is easy to hide and that we will not find you, we have just as easy to use, fully automatic and intelligent tools that will identify any potential cases of plagiarism. And do not think that being the original author will make any difference. Sharing an original solution with others is just as unethical as using someone else’s work.

CS100 Homework 8 (Fall, 2019)

This is the specification for homework 8. It contains 3 connected problems and is on the recently taught material (mainly rvalue references and functional programming).

Percentage of this homework over the whole score: 11%

Submission deadline: 2020-1-4 23:59

// Congratulations, this is your last homework for CS100! (and maybe also the first homework in the year 2020!)

Note for Online Judge

The usage of PIEGON Online Judge is the same as before. The online judge will open for all three problems at the same time, and all problems will be judged together. When submitting, you should push all (*.hpp) files into your repository. Do not put the /src/ folder or the whole HW8 folder into the repository.

The online judge will be open no later than 2019-12-28 23:59.

Problem 1: Implementation of Poly.hpp

You are to implement a class named `poly` that simply represents a uni-variate, scalar valued polynomial. A polynomial can be represented by a vector of coefficients, in ascending order of powers. For example: $3x^3 - x + 6$ can be represented by $\{6, -1, 0, 3\}$, since the coefficients of x^0, x^1, x^2, x^3 are $6, -1, 0, 3$, respectively. The interface of the `class poly` is given to you, where you need to implement basic functions and operators. We have provided comments for them, and they are also pretty self explaining.

There are some further functions in the Poly class that deserve attention.

- The function `eval` evaluates the value of this polynomial as a function at "param".
- The function `der` constructs the first-order derivative of the polynomial.
- The most important one is

  ```
  std::pair<T,T> operator()( T param ) const
  ```

  which returns the value of the polynomial at param and also the first order derivative at param (just an evaluation of the derivative polynomial). This function is going to be useful for Problem 2.

Note: there is an intricate link to functional programming: a polynomial can be interpreted as a simple function that takes in a single parameter and returns a scalar. More interestingly, the sum, product, etc. or even composition of two scalar polynomimals remains a scalar polynomial, so many of the operations on these polynomials can be understood as functional forms. However, not all functional forms can be easily realized, as the division of two polynomials does not return a polynomial.

Problem 2: Implementation of functional_forms.hpp

Here we consider polynomials as functional forms, and use lambda functions that either one or two uni-variate functions and construct a new uni-variate function from it. Six basic operations need to be defined:

- sum
- subtraction
- multiplication
- division
- function composition
- power function

These functions should be easily understood: if $f(x) = x^2 + 1$, and $g(x) = x - 3$, then $(f + g)(x) = x^2 + x - 2$, and $(f(g))(x) = x^2 - 6x + 10$. *(verify it!)*

There is one speciality about the involved functions. For all of them, we assume that they return an std::pair<double,double>, where "first" represents the actual value of that the function at at certain point, and "second" the first order derivative.

Example: if $f(x) = x^2 + 1$, and $g(x) = x - 3$, then calling `derSum(f, g)(0.5)` will return an `std::pair` of $(-1.25, 2)$. *(verify it!)*

Notes:

- For all above functional forms the derivative is clearly defined:

$$\text{if } g(x) = f(x) + h(x), \text{ then } \frac{dg}{dx}(x) = \frac{df}{dx}(x) + \frac{dh}{dx}(x)$$

$$\text{if } g(x) = f(x) - h(x), \text{ then } \frac{dg}{dx}(x) = \frac{df}{dx}(x) - \frac{dh}{dx}(x)$$

$$\text{if } g(x) = f(x) \cdot h(x), \text{ then } \frac{dg}{dx}(x) = \frac{df}{dx}(x)h(x) + f(x)\frac{dh}{dx}(x)$$

$$\text{if } g(x) = \frac{f(x)}{h(x)}, \text{ then } \frac{dg}{dx}(x) = \frac{\frac{df}{dx}(x)h(x) - f(x)\frac{dh}{dx}(x)}{h(x) \cdot h(x)}$$

$$\text{if } g(x) = h(f(x)), \text{ then } \frac{dg}{dx}(x) = \frac{dh}{dx}(f(x))\frac{df}{dx}(x)$$

$$\text{if } g(x) = (f(x))^p, \text{ then } \frac{dg}{dx}(x) = p(f(x))^{(p-1)}\frac{df}{dx}(x)$$

- The polynomials implement the `operator()` and it returns `std::pair<double,double>` (if double is the template parameter). They are used as an input to the above functional forms.

The advantage of using the above functional forms is that we have automatic differentiation! There is an example in the file "test_lambdas.cpp" which uses the below complicated function:

$$g(x) = \frac{(p_2(p_1(x)))^3 - p_3(x)}{p_4(x) + p_5(x) \cdot p_6(x)}$$

It wouldn't be possible to combine polynomials into new polynomials as the expression contains a division. Furthermore, deriving the deriviative by hand would be tedious, and derivate calculation with finite differences is inaccurate. However, calculating either the value or the derivative at any given point would become simple by using above lambda functions.

Hint: (This is not part of your implementation) If you want to verify the correctness of your differentiation, you can easily do so by using finite difference approximation:

$$\frac{dg}{dx} = \lim_{\Delta x \to 0} \frac{g(x + \Delta x) - g(x)}{\Delta x}$$

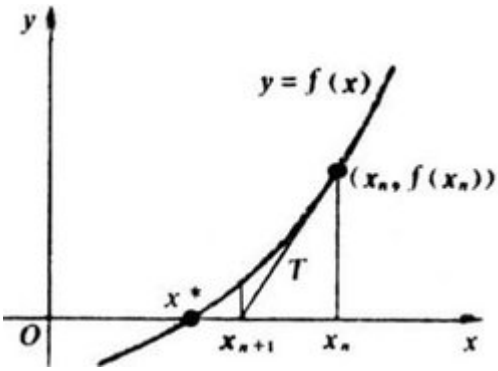
Problem 3: Implementation of newton.hpp

Here you need to implement Newton's method (for simple univariate functions). It's a method for finding nearest root from a starting point by iteratively using linear approximation. You can follow the link below, or search for "Newton's method" on any search engine.

Its basic idea:

Starting at $x_0 = startingPoint$, and repeatedly find the next x by $x_{k+1} = x_k - (f(x_k)/f'(x_k))$ until $f(x_n)$ is close enough to 0.

An illustration:



We can easily apply it to the functionals from problem 2, as they return the value and the derviative. The first parameter of this function (f) is exactly a lambda function generated from problem 2. You can refer to the usage in the provided example "test_newton.cpp" for details.