Lab 4

EC327 Fall 2015

In Lab 4, we define four problems (P1, P2, P3, P4) of varying difficulty. To pass Lab 4, choose one problem and try to solve it. You are not required to solve all four problems!

The problems' solutions focus on the practical utilization of C++ primitive data types, loops (DO-WHILE, FOR, WHILE), functions, and header files.

In your designated lab hours you should demonstrate and explain your solution approach of the chosen problem. You are not required to work outside lab hours! We recommend starting early. For example, you can solve the problem at home, come to lab, demonstrate your solution, and get checked off. However, you must stay in lab for at least one hour; if you finish earlier than that, work on a second problem. Ensure that your code compiles and executes properly in the lab.

For a given problem, you will download the provided testbench and write the corresponding .h and .cpp files, using `#ifndef` / `#define` / `#endif` guards as necessary. Make sure that your code compiles with the provided testbench and runs as expected before getting checked off.

If you need any basic supporting functions (e.g. `abs`, `min`), it is recommended that you simply write your own for practice.

Read the problem statements carefully. Decide which problem to solve, and decide when to start. Do not hesitate to ask any clarifying questions.

Good luck.

1 Vector Length

A straightforward way to compute the square root of a number N is to start with some positive initial guess `x` (feel free to fix this value; do not request user input), then repeat the following until convergence:

```
x = (x + (N / x)) / 2
```

where convergence is defined as the guess being within a relative tolerance:

$$\frac{|x - \frac{N}{x}|}{\min(x, \frac{N}{x}))} \leq tol$$

Note: this is not the condition for your while loop.

Write a `sqrt` function (do NOT `#include <cmath>` and pass the result off as your own), then use it to compute the Cartesian length of a 3-dimensional vector,

$$\ell(< x, y, z >) = \sqrt{x^2 + y^2 + z^2}$$

You may assume `tol` and `N` are positive.

Function prototypes:

```
double sqrt(double N, double tol);
double length(double x, double y, double z, double tol);
```

2 Fraction Reduction

One of the oldest known algorithms is Euclid's greatest-common-divisor algorithm, which computes the largest natural number that evenly divides two other natural numbers. Written recursively, it can be expressed as

```
gcd(a, 0) = a
gcd(a, b) = gcd(b, a % b)
```

Convert this to an iterative process, then use it to reduce a fraction whose numerator and denominator are provided (e.g. 48/64 should reduce to 3/4). Because `reduce_fraction` generates two output values, pass the numerator and denominator by reference, and modify the referenced values. You may assume all values are nonnegative.

Function prototypes:

```
int gcd(int a, int b);
void reduce_fraction(int &a, int &b);
```

3 Newton's method

Newton's method is an iterative method for finding the zeros of a continuous real-valued function.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

In order to compute this, you need to first write a function to compute the derivative — use a 2-point central differencing scheme[1]:

$$f'(x) \approx \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x}$$

Repeat the update process until the estimate is within a specified absolute tolerance,

```
abs(f(x)) <= tol
```

Note: this is not the condition for your while loop.

You may assume `dx` and `tol` are positive. For this problem, download newton.cpp ("Newton's Method skeleton") from Blackboard and modify it to suit your needs.

Function prototypes:

```
double deriv(double (*fn)(double), double x, double dx);
double newton(double (*fn)(double), double dx, double tol);
```

[1] The usual definition of the derivative can be seen as a 2-point forward differencing scheme. By using this central differencing scheme, we can increase the order of precision. In fact, this is order-optimal for differencing schemes using at most three points.

4 Counting Heads

Given N coin flips, the number of ways one can get k heads is given by

$$\binom{N}{k} = \frac{N!}{(N-k)!k!}$$

(pronounced "N choose k"), where $n!$ is the factorial function, defined as $1 \times 2 \times \cdots \times (n-1) \times n$. Write a function to compute this expression, then use it to count the total number of ways one can get at least k out of N heads. You may assume that N and k are always nonnegative, and $k \leq N$.

Warning: naïvely computing the three factorials will not work when dealing with relatively small numbers, such as $N = 40$. To combat this, think about how you might be able to compute the product without the intermediate values. Some terms will cancel.

Function prototypes:

```
int choose(int N, int k);
int head_count(int N, int k);
```