Computing and Statistical Data Analysis
Problem sheet 2

Remember to write your name, College and degree programme (e.g., PhD, MSci or MSc) on your paper.

Please turn in a copy of the source code and relevant sample output. Avoid unnecessarily complicated code. Highest marks go to a simple, elegant, robust solution. If you choose to do something complicated (e.g., fancy error checking), then the complexity should buy meaningful additional functionality or robustness.

1: Starting from the algorithm you created to compute factorials in Problem Sheet 1, write a function called `factorial` that takes a single `int` argument and returns the factorial as a `double`. Note that the error checking, e.g., for non-negative integer input, must now be put inside the body of the function.

Put the function in a separate file `factorial.cc` and put the prototype in a header file `factorial.h` with the appropriate "include guards".

Write a short `main` program to test your function (i.e., similar to the program from Problem Sheet 1).

Write a short shell script called, say, `build.sh` to compile and link the program.

Show sample output for a few values of the input value, e.g., $n = 10, 40, 80$.

Overload the `factorial` function so that it can also take an argument of type `double`, which also returns `double`. Illustrate the use of the function in the main program. (Hint: remember the "once and only once" principle.) The function definitions and prototypes can go into the same files as for the original.

2: Write a program that computes a table of values of n, $\sqrt{n}$, $\ln n$ and $n!$ for $n = 1, 2, \ldots n_{\max}$, where the value $n_{\max}$ is set by the user through a `cin` statement. Using the techniques shown in the lecture, display the result on the monitor formatted such that the value of n appears as an integer in a column with 5 spaces, the values of $\sqrt{n}$ and $\ln n$ appears as decimal values with 12 total spaces and five places to the right of the decimal point, and the value of $n!$ is given in scientific notation with five digits to the right of the decimal place.

Then using the methods described in the lectures, also write the table of numbers to an output file. Create the file for $n_{\max} = 20$.

3: Write a function `swap(x,y)` of return type `void` which takes two `int` arguments passed by reference, such that the values of `x` and `y` are swapped after calling the function. Write a short test program which calls `swap` to test and illustrate its use.

4 Look at the `TwoVector` class on the course website. The state of the vector is given by the two variables `m_x` and `m_y`, which give the x and y components of the vector. For this exercise you will modify and extend this class. You should write a small test program (similar to the program `TestTwoVector.cc` on the website) to show that the new class works correctly, by creating `TwoVector` objects, calling the functions, and printing the results to the monitor with `cout`. You should turn in the source code and sample output that show together the solutions for (a), (b) and (c), i.e., all three parts can be answered using the same program.

4(a) Rewrite the `TwoVector` class so that the data members represent polar coordinates, r and θ. That is, get rid of `m_x` and `m_y` are replace them by variables `m_r` and `m_theta`. Rewrite the member functions so that that class behaves the same way as before (the names, return types and signatures of the member functions should be exactly the same as before).

The arguments should be interpreted the same way in both the original and new versions of the class. So, e.g., for the two-argument constructor the arguments should still be interpreted as x and y; these are then used to set `m_r` and `m_theta`.

4(b) In your modified `TwoVector` class, write a public member function

```
void TwoVector::reflect(TwoVector& u){  your code here }
```

such that when a `TwoVector v` calls the function,

```
v.reflect(u);
```

the effect is to reflect `v` about the line defined by the argument `u`. Show in your test program that the function works as expected.

4(c) Overload the operators `+=` and `-=` so that they work with objects of the `TwoVector` class. Show in your test program that they work as expected. The overloaded operators `+=` and `-=` should update the state of the calling object, and also return (by reference, i.e., return type `TwoVector&`) the obdated object (i.e., `return *this;`).

G. Cowan
10 October, 2012