Chapter 2 exercises

1. The arguments of the function

   ```
   int main(int argc,char* argv[])
   ```

   for a console application give the number of command line arguments with which the console application was invoked (`argc`) and an array of pointers to C–style strings containing those command line arguments (`argv`). Thus, if the console application is called `my_program`, invoking

   ```
   my_program test 123 3.14
   ```

   will result in

   $$\begin{aligned} \texttt{argc} &\Longleftarrow 4 \\ \texttt{argv[0]} &\Longleftarrow \texttt{"my\_program"} \\ \texttt{argv[1]} &\Longleftarrow \texttt{"test"} \\ \texttt{argv[2]} &\Longleftarrow \texttt{"123"} \\ \texttt{argv[3]} &\Longleftarrow \texttt{"3.14"} \end{aligned}$$

   (a) Write a console application which calculates the factorial of an integer passed as a command line argument.

   (b) Write a console application which calculates the square root of a decimal (floating point) number passed as a command line argument.

2. Using the "shapes" class hierarchy from Exercise 7 in Chapter 1, populate a `std::list` of shapes with different instances of shapes. Calculate the sum of the areas of all shapes.

3. Add another type of shape to the class hierarchy of shapes used in Exercise 2. Does the code you created for Exercise 2 still work when you include this new shape in your `std::list`?

4. The Black/Scholes call option pricing formula is

   $$C = SN(d_1) - Ke^{-rT}N(d_2)$$

   with

   $$d_{1,2} = \frac{\ln \frac{S}{K} + (r \pm \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}$$

   where S is the initial stock price, K is the strike, T is the time to maturity, r is the riskfree interest rate, σ is the volatility and N is the cumulative distribution function of the standard normal distribution.

(a) Implement this in C++.

(b) Using the `Rootsearch` template from Section 2.5.1, implement a function which calculates the implied volatility of an option, i.e. the σ such that given all other parameters, the Black/Scholes price of the option matches a given market price.

5. The library code accompanying the textbook includes the `CSV2Array()` template function, which uses the Boost library to parse a CSV file into a Blitz `Array`. This is a convenient way of reading numerical data, but also passing a large number of named parameters to a program. The C++ source file `CSV2ArrayExample.cpp` on the website illustrates this. Modify your code from Exercise 4 to accept the required inputs via a CSV file of named parameters.