General Instructions

1. Download `Practical01.zip` from the course website.

   Extract the file under the `[CppCourse]\[Practicals]` folder. Make sure that the file structure looks like:

```
[CppCourse]
   -> [boostRoot]
   ...
   -> [Practicals]
      -> [Practical01]
         -> Practical01Exercises.hpp
         -> ...
         -> [Src]
            -> Practical01.cpp
```

2. Open the text file `[CppCourse]\CMakeLists.txt`, uncomment the following line by removing the `#`:

```
#add_subdirectory(Practicals/Practical01)
```

   and save the file. This registers the project with `cmake`.

3. Run `cmake` in order ot generate the project following the instructions in `CppInstructionsVScode2025.pdf` section 5.2 paragraph *"Configure CMake"*.

4. The declaration of the functions to be implemented are in `Practical01Exercises.hpp`. Create a `cpp` file for each function and add them to the project.

5. Implement the functions into the newly added `.cpp` file(s) under the `[Src]` folder. Do not modify any of the other files.

6. Compile and run your code (the `main` is provided with the project) following the instructions in `CppInstructionsVScode2025.pdf` section 5.3 *"Building and running projects"*. If the minimum requirements are met, an output text file `Practical01_output.txt` will be created.

7. You are expected to hand in `Practical01_output.txt` and all the `*.cpp` files you created and put any of your own code into. These files are to be submitted via Moodle.

Exercise 1

```cpp
double Norm2(const std::vector<double> & dVec);
```

This function takes a vector $x = (x_1, \ldots, x_n)$ and returns

$$\sqrt{\sum_{i=1}^{n} x_i^2}.$$

Exercise 2

```
double NormInf(const std::vector<double> & dVec);
```

This function takes a vector $x = (x_1, \ldots, x_n)$ and returns

$$\max_{1 \le i \le n} |x_i|.$$

Exercise 3

```
double MonteCarlo1(double dR,
                   double dSigma,
                   double dS0,
                   double dK ,
                   double dT,
                   unsigned long int iN);
```

`MonteCarlo1()` takes the risk-free rate, volatility, initial stock price, strike price, time to maturity and the sample size and returns a Monte Carlo estimate of the corresponding European Call option price assuming that the underlying S_t is a geometric Brownian motion.

$$\begin{aligned} PV &= \exp(-rT)\mathbb{E}[\max(S_T - K, 0)] \\ &\approx \exp(-rT)\frac{1}{N}\sum_{i=1}^{N} \max\left(S_0 \exp\left(\left(r - \tfrac{1}{2}\sigma^2\right) T + \sigma\sqrt{T} Z_i\right) - K, 0\right) \end{aligned}$$

where the Z_i's are independent random variables from standard normal distribution. Note: the header file `Utils/UtilityFunctions.hpp` inside the `utils` namespace contains two functions that one might find useful.

```
    void NormalDist(std::vector<double> &vArg);
```

```
    double NormalDist();
```

The first function takes a vector by reference and fills it up with standard normals. The second function takes no argument but returns one single standard normal variable.

Exercise 4

```cpp
MCResult MonteCarlo2(double dR,
                     double dSigma,
                     double dS0,
                     double dT,
                     unsigned long int iN,
                     Payoff call);
```

`MonteCarlo2()` takes the same arguments as `MonteCarlo1` except for the strike, instead it takes a `Payoff` type, where

```cpp
typedef std::function<double(double)> Payoff;
```

that is, `Payoff` wraps functions that take a `double` and return a `double`. The function `MonteCarlo2()` is to calculate the following two outputs:

1. a Monte-Carlo estimate of the price of the European option that pays $f(S_T)$, where f is a generic payoff function, and the underlying S_t follows geometric Brownian motion, see formula 1,
2. an estimate of the standard deviation of the first output, see formula 2.

These two outputs are to be returned boundeld in an instance of the struct `MCResult`, which has two entries of type `double`:

- `mc_estimate` is the MC estimate of the option value,
- `mc_stdev` the estimated standard deviation of the MC estimate.

Given two `double`'s `x` and `y`, an instance of `MCResult` can be created using the following syntax

```cpp
  MCResult{x, y};
```

Given the payoff function f, the MC estimate is to implement the formula:

$$\texttt{mc\_estimate} = \exp(-rT)\frac{1}{N}\sum_{i=1}^{N} f\left(S_T^{(i)}\right) \tag{1}$$

with

$$S_T^{(i)} = S_0 \exp\left(\left(r - \tfrac{1}{2}\sigma^2\right)T + \sigma\sqrt{T}Z_i\right),$$

where the Z_i's are independent standard normal random variables.
The standard deviation of the MC estimate is defined as follows:

$$\texttt{mc\_stdev} = \exp(-rT)\frac{1}{\sqrt{N}}\sqrt{\frac{1}{N}\sum_{i=1}^{N} f\left(S_T^{(i)}\right)^2 - \texttt{mc\_estimate}^2} \tag{2}$$

Exercise 5

```
double callAt1(double dS);
```

`callAt1()` is a particular function of type `Payoff` implementing a European payoff with strike 1.