Dr. Ole Klein
M.Sc. Robin Görmer, M.Sc. Philipp Thiele

Lima, Oct. 23, 2019

Spring School on the Introduction on Numerical Modelling of Differential Equations – Programming Exercise 1

Exercise 1.1 [Working with the C++ compiler]

Take some time to get familiar with the compiler output.
The output for errors and warnings is structured as follows:

[file]:[line]:[position]: [type]: [description of the error or warning]

The output of all warnings can be activated bei adding -Wall (Warnings all) to the compiler call:

```
>> g++ -Wall wrong.cc -o wrong
```

Fix all errors and warnings in the file "wrong.cc". The file should compile without errors or warnings

```
#include <iostream>

namespace local
{
  int answer() { int answer = 42; return; }
}

main()
{
  int dummy = 0;

  double a; a = 1.0;
  double const b; b = 2.0;
  double c = a * b;

  std::cout << "The answer is: " << answer() << "\n";
  std::cout << "a = " << a << \n;
  std::cout << "b = " < b << "\n"
  std::cout << "a * b =  << c << "\n";

} // main
```

Exercise 1.2 [Factorial]

Write a program `factorial.cc`, that computes computes for an input `int n` the factorial

$$n! = n \cdot (n-1) \cdot \ldots \cdot 1$$

The following commands will compile and run the file.

```
>> g++ factorial.cc -o factorial
>> ./factorial
```

Test your program for different values of `int n`. What do you notice? change your program for, such that it uses only `unsigned int` and test your program again.

Exercise 1.3 [Euler's Constant]

Euler's constant is defined by the infinite sum

$$e = \sum_{k=0}^{\infty} \frac{1}{k!} = 2.718281828459045\ldots.$$

This formula allows us to compute the value of e to arbitrary precision. Using $e_n := \sum_{k=0}^{n} \frac{1}{k!}$ we can compute the precision with $e_n - e_{n-1}$.

Write a program/function, that computes the approximation of e for a given integer n and print that number on the screen. Make sure to efficiently compute the values of $k!$ by using the previous value.

(a) Compute e_n for $n = 5, 10, 20$, by summing upwards ($k = 0, 1, 2, \ldots$) resp. downwards ($k = n, n-1, n-2, \ldots$). Compare the results and precision. Make sure to use the right data types

(b) Change your program/function, such that the computations can also be done with `float` and compare the results. Furthermore, compute the error to `std::exp(1.0)` from the standard library `<cmath>`.

(c) Optional: Write a program/function, that approximates e to a given precision `tol`.

Bonus Problem 1.4 [Binomial coefficient]

Write a program `binomial.cc`, that computes for specific inputs $k, n \in \mathbb{N}$ the binomial coefficient

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

Use the data types `unsigned int` and treat possible exceptional cases ($k > n$?) with `if` conditions. Test your program intensively.

The allowed range for $n, k \in \mathbb{N}$ can be extended by **not** using all factorials:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{k!}$$

Furthermore, the efficiency can be enhanced by exchanging k with $min(k, n-k)$, due to the symmetry of the binomial coefficient.
Use the above aspects to improve your program and compare your results with the first version.

Optional: Compute the binomial coefficient for $n, k \in \mathbb{N}$ using the product

$$\binom{n}{k} = \prod_{j=1}^{k} \frac{n+1-j}{j}.$$

This product can be implemented using a loop that switches between multiplication and division, **without** a division remainder. Why?

Bonus Problem 1.5 [Greatest common divisor]

To compute the greatest common divisor (gcd) of two numbers, the euclidian algorithm can be used:

> Start by dividing the bigger by the smaller number. After that divide the smaller number by the remainder until the remainder is zero. The last divisor is the gcd. If the remainder is 1 both numbers don't have a common divisor.
> The following example illustrates this algorithm with the numbers 13575 and 345:

- $13575 = 39 \cdot 345$ remainder 120
- $345 = 2 \cdot 120$ remainder 105
- $120 = 1 \cdot 105$ remainder 15
- $105 = 7 \cdot 15$ remainder 0

The greatest common divisor is 15.

Implement this algorithm for two positive integers in C++ and test it on some examples.

(a) Implement the function
- `unsigned int gcD(unsigned int, unsigned int)`. Use the `%`-Operator for divisions with remainder.

(b) Write a `main`-method, that tests your function on the given example. Output the solution to the terminal.

Bonus Problem 1.6 [Bisection algorithm]

The bisection algorithm is used to find the root of a continuous function $f\colon \mathbb{R} \to \mathbb{R}$ in the interval $[a, b] \subset \mathbb{R}$. It works as follows:

If $f(a)$ and $f(b)$ have different signs there exists a root in $[a, b]$. If you seperate the interval into two equal parts $[a, m]$ and $[m, b]$ by using $m = \frac{a+b}{2}$, the root has to be in one of the two intervals. If $f(m) = 0$ we have found the root. Otherwise we have to check for the signs again. If $f(a) \cdot f(m) < 0$ we set $b = m$ and start from the beginning. Otherwise if $f(b) \cdot f(m) < 0$ we set $a = m$ and start from the beginning.

Write a program that consist of the following parts
- a function `f`, for computing the value $f(x) = e^{\sin(x)} - 0.5$
- a function `bisec`, for Finding the root of $f(x)$ on an interval given by a and b using the described algorithm
- a `main`-method.

The (iterative) implementation should terminate if
- a maximal number of iterations `max_iter` has been reached
- for the given tolerance `tol`, $f(m) <$`tol` holds

Print the values for a and b in every iteration to the terminal. Test your program with the accuracy `tol`$= 10^{-6}$ for the interval $I := [-1, 1]$ and verify your results.

Afterwards change your functon `f` and determine the roots of the following functions on the same interval
- $f(x) = \frac{1}{\sqrt{x^2+1}} - \cos(x + 1)$,
- $f(x) = \sin(\cos(x + 1))$,
- $f(x) = \arctan(x^2 + 2x) - 0.2$,
- $f(x) = x^2 + 1$.