Exercise 4
Classes, Numerical Errors
(Hand out: 30.04.2025, hand in: 07.05.2025)

Review

1. What does the line
   `using namespace std ;`
   in a C++ program mean? (1 P)
   - ☐ It loads the input/output library.
   - ☐ It allows the usage of global variables with names that contain spaces.
   - ☐ It sets the namespace to `std`, so that one can write, e.g., `cout` instead of `std::cout`.

2. How must variables be passed to functions that they can be changed by this function? (1 P)
   - ☐ as a pointer: `int *n`
   - ☐ as a variable: `int n`
   - ☐ as a reference: `int &n`

---

1. Task *Boolean satisfiability problem (SAT)* (3 + 1 BP)
Which of the following logic formula can be satisfied (= true)? By which setting can this be fulfilled? (2 P)

a) $(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3} \vee \overline{x_4}) \wedge (x_2 \vee x_3 \vee x_4)$

b) $(x_1 \vee x_2) \wedge (\overline{x_2} \vee x_1) \wedge (\overline{x_1} \vee x_3) \wedge (\overline{x_3} \vee \overline{x_1})$

The symbol's meaning: $\vee \rightarrow$ or; $\wedge \rightarrow$ and; $\overline{x} \rightarrow$ not x
Hint: Program in C++ nested loops that change the setting of the logical variables `bool` $x_1 \ldots x_4$ (truth table).
Alternatively (for experts $\rightarrow$ 1 BP!): Use just *one* loop and the function `ibits()`, to get all possible combinations for $x_1 \ldots x_4$:

```
int ibits (const unsigned int& i, int pos, int len) {
  return (i >> pos) & ~(-1 << len) ;
}
```

Question: Depending on the number n of Boolean variables $x_1 \ldots x_n$, how many combinations must be tested? (1 P)

2. Task *Implement a complex data type* (4 P + 1 BP)
Implement your own struct/class `complex_d` for complex numbers (with `double` type members) as shown in the lecture and test it. This class should be able to perform the following operations: `+,-,*,/` and also taking the absolute of a complex number: `absolute()`. Test your class with some basic calculations in the `main` program.
Bonus: Avoid using the `cmath` library at all by implementing also your own method `sqrt()` for the absolute. *Hint:* Use Heron's method. You might need to declare the corresponding function as `const`:

```
double my_sqrt(double a) const { ... }
```

(1 BP)

3. Task *Absorption* (3 P)

a) Consider the example from the lecture:

```
float y = 100000010., inc = 1. ;
for (float x = 100000001. ; x <= y ; x += inc) { ... }
```

and rewrite it correctly with an integer loop counter. Note that this will not solve the absorption problem. (1 P)

b) Write a C++-program to show that the Kahan summation algorithm gives the more accurate result when adding `float inc = 1.E-7` n times in a *loop* to the initial value `float x = 7.` Also compare both results for the case of double precision, i.e. using `double` instead of `float`. (2 P)