Informatik - AS19

Exercise 10: Struct and Operators

Handout: 18. Nov. 2019 06:00

Due: 25. Nov. 2019 18:00

Task 1: Finite Rings

Open Task

Task

Complete type `int_7` for computing with integers modulo 7:

```
struct int_7 {
    unsigned int value;
};
```

Mathematically, this corresponds to the finite ring $\mathbb{Z}_7 = \mathbb{Z}/7\mathbb{Z}$ of residue classes modulo 7. Addition and subtraction operations work according to the following table which also defines subtraction: $x - y$ is the unique number $z \in 0, \ldots, 6$ such that $x = y + z$.

```
+ | 0 1 2 3 4 5 6
--+--------------
0 | 0 1 2 3 4 5 6
1 | 1 2 3 4 5 6 0
2 | 2 3 4 5 6 0 1
3 | 3 4 5 6 0 1 2
4 | 4 5 6 0 1 2 3
5 | 5 6 0 1 2 3 4
6 | 6 0 1 2 3 4 5
```

Multiplication in $\mathbb{Z}_7$ works according to the following table:

```
* | 0 1 2 3 4 5 6
--+--------------
0 | 0 0 0 0 0 0 0
1 | 0 1 2 3 4 5 6
2 | 0 2 4 6 1 3 5
```

```
3 | 0 3 6 2 5 1 4
4 | 0 4 1 5 2 6 3
5 | 0 5 3 1 6 4 2
6 | 0 6 5 4 3 2 1
```

The task consists in five parts:

1. Specify the invariant (allowed values) for member variable "value", as a comment in file `finite_ring.h`. Do **not** change anything but comments in that file.
2. Complete operation `from_int`, which turns an unsigned integer into its residue class (in file `finite_ring.cpp`).
3. Complete the addition operation (in file `finite_ring.cpp`).
4. Complete the subtraction operation (in file `finite_ring.cpp`).
5. Complete the multiplication operation (in file `finite_ring.cpp`).

Task 2a: Complex Numbers

Open Task

Task

Create your own data type for complex numbers that can be used as a drop-in replacement for double. The definition of the data type must be in file `complex.h`, the forward declarations of operators in file `complex.h`as well, while the implementation of operators must be in file `complex.cpp`.

1. Define a struct named `Complex` that represents a complex number in the cartesian form. As data type use floating point numbers with `double` precision.
2. Implement input and output (>>, <<) operators for reading and writing of complex numbers.

   A complex number $(x \pm yi)$ is represented in the format `[x,y]`, where `x` and `y` follows the format of floating point numbers with `double` precision.

   E.g., $[-2, 5]$ represents $(-2 + 5i)$. **Important: No spaces.**

   You can assume that all input data is correct, and thus the input operation does not require error handling. (Advanced) If you want you can enable error handling on stream level by setting the *failbit* on the input stream in case the input is not a valid complex number.

3. Implement arithmetic operators for addition, subtraction, multiplication, and division ($+$, $-$, $*$, $/$) of complex numbers. For that task, you may assume that division will never be called with a complex whose half square modulus is lower than the smallest positive number representable as floating-point. In particular, you do not need to treat the case of division by zero.
4. Implement the negation operator ($-$) of complex numbers.
5. Implement comparison operators for equality and inequality ($==$, $!=$) of complex numbers.

Task 2b: Calculator For complex numbers

Open Task

Task:

Extend the calculator presented in the lecture with the ability to process complex numbers by using your own data type `Complex` as a drop-in replacement for `double`.

In the programming environment, you find the C++ source of the calculator for double values as file `calculator_double.txt`. Copy the contents of this file to `main.cpp`, add your datatype `Complex` from sub task a) in files `complex.{h|cpp}`, and modify `main.cpp` to support complex numbers. Remember, your datatype `Complex` is supposed to be a drop-in replacement for double.

The method in which we read doubles has been simplified in order to make the code easier to adapt to new datatypes.