Task for OOP

deadline: Oct. 25, 2011, 11:00 Uhr

2. Design and implement the class HELI which stores as members the position of a helicopter in space, its operativeness, the max. and the min. altitude. Implement the following methods:

   - Standard and parameter constructor.
   - Methods for changing the position of the helicopter: `forward`, `backward` (x–Richtung); `right`, `left` (y–direction); `up`, `down` (z–direction, altitude).
   - Take into account that the helicopter can rise to 5000 m altitude at most and it remains operational. But if it falls below the min. altitude then the helicopter can't move anymore.
   - Change of position is only possible with an operational heli.
   - Write an output operator (`<<`) for your class that prints position and status of your helicopter.

   Your main function should look similar to the following listing:

```
int main(const int argc, const char* argv[])
{
  heli a(0,0,0,5000,0);  // x,y,z-Position, max./min. Altitude

  a.up(4); a.left(100); a.forward(22);
  cout << a << endl;
  a.up(6000);
  cout << a << endl;
  a.down(5001);
  cout << a << endl;
  a.right(17);
  cout << a << endl;

  return 0;
}
```

3. Design and implement a class for rational numbers (I called it `bruch`) containing all four basic arithmetic operations such that the following code can be used without any changes (you have to include the appropriate header file(s)).

```
 1  int main()
 2  {   // (enumerator, denominator) [germ.:(Zaehler, Nenner)] as Parameter list
 3      const bruch a(4,3), b(7,-2);
 4            bruch c(a+b), d(-3,0);
 5
 6      d = a*b;
 7      cout << "a = " << a << endl;
 8      cout << "b = " << b << " has enumerator " << b.getZaehler() << endl;
 9      cout << "c = " << c << endl;
10      cout << "d = " << d << endl;
11      cout << "e = " << d/(b-a)-a*c << endl;
12
13      return 0;
14  }
```

Implement all methods/operators by yourself although you could use also the assignment operator and constructors implicitly generated by the compiler.

- Distinguish between declaration and implementation (header and source file).
- Deactivate first the lines 4–11 in the code above and start with declaration/implementation of parameter constructor and standard constructor (which fraction is equivalent to 0 ?).
- Line 4 requires an additional copy constructor and the addition operator `operator+` .
- Line 6 requires the assignment operator `operator=` and multiplication operator `operator*`.
- Lines 7–10 use the output operator `operator<<` that uses your class as parameter, i.e., it is not a method of your class. Here, you should use the access methods (as `getZaehler()`) appropriately
  Check the correctness of the results after arithmetics!
- More operators are needed in line 11.

After implementation of all basic items above we think about improvements and protection of the class:

- What happens if the denominator is equal to 0?
  That have to be taken into account in the constructors and arithmetic operators.
- The instance `b(7,-2)` would be stored as $\frac{7}{-2}$. Change the parameter constructor (and operators) such that the fraction possesses always a positive denominator.
- Instance d has the value $\frac{-28}{6}$ and this fraction can be reduced to $\frac{-14}{3}$.
  Implement this reduction as a `private` method by first determining the ggT (greatest common divisor) followed by using this number. Reduce the fraction in all methods and constructors wherein it makes sense but do it in a smart way.
- Take care for the division by 0 and handle that case.

$*$ How can you check for leaving the admissible range of the chosen data type in temporary results? Can you catch these cases without using exception handling?