Lab 2

Task for today: implement fractions

Why should we implement fractions when we have floating point arithmetic and `float` and `double` types?

Operating on floating point types may introduce rounding errors. Many values can't be represented accurately by `float` or `double` types. For example what is the result of this simple program:

```
#include <iostream>
#include <limits>

int main() {
    double a = 0.1;
    double b = a + a;
    if(b == 0.2) {
        std::cout << "b equals to 0.2" << std::endl;
    } else {
        std::cout << "b does not equal to 0.2, but is: " << b << "?!?!?!" << std::endl;
        std::cout.precision(std::numeric_limits<double>::max_digits10);
        std::cout << "b printed with maximum precision is: " << b << std::endl;
    }

    double c = a + a + a;

    if(c == 0.3) {
        std::cout << "c equals to 0.3" << std::endl;
    } else {
        std::cout << "c does not equal to 0.3, but is: " << c << "?!?!?!" << std::endl;
        std::cout.precision(std::numeric_limits<double>::max_digits10);
        std::cout << "c printed with maximum precision is: " << c << std::endl;
    }

    return 0;
}
```

The computer that allowed people to land on the moon didn't have floating point operations. Back then (early sixties!) people were able to calculate some really complicated math on a computer without floating point unit.

How to approach the problem?

1. Represent a fraction as struct with two integer numbers: nominator and denominator
2. After every operation that changes nominator and/or denominator shorten/simplify the fraction, for example the fraction `4/6` should be simplified to `2/3`.
3. Make sure that the denominator is not equal to 0
4. Comment out all contents of the `main` function
5. Implement methods and functions in the order that they are used in the `main` function, uncomment gradually usage of your implemented methods and functions int the `main` function, compile and run to test results.

Shortening, simplification of fractions

1. Calculate greatest common divisor of numerator and denominator
2. Implement a function based on Euclidean algorithm - you will need it in more than one place
3. Make sure that denominator is positive

Sum and subtraction of fractions

1. Use function which calculates greatest common divisor to calculate smallest common denominator for operations `add` and `subtract`

For example instead of such operation:

```
1/6 + 1/9 = 9/54 + 6/54 = 15/54 = 5/18
```

it is better to calculate it like below to avoid overflow in more situations than in the above approach:

```
1/6 + 1/9 = 6/36 + 4/36 = 10/36 = 5/18
```

Reading values of a fraction

1. Read two numbers, first nominator, then denominator. If you don't remember how to read input from the user, check how we did it in the first Lab.
2. If the user entered 0 for denominator, inform her/him of wrong input and read denominator again.

Printing

1. Print fraction in the form: `nominator/denominator`, for example: `1/2`
2. If nominator is equal to 0 then just print `0`, if denominator is equal to 1, then just print the value of the nominator.