

Московский авиационный институт  
(Национальный исследовательский университет)  
Факультет "Информационные технологии и прикладная математика"

**Лабораторная работа №1 по курсу  
“Объектно-ориентированное программирование”**

*Студент:* Живалев Е.А.

*Группа:* М8О-206Б

*Преподаватель:* Журавлев А.А.

*Вариант:* 6

*Оценка:* \_\_\_\_\_

*Дата:* \_\_\_\_\_

Москва  
2019

# 1 Исходный код

Ссылка на github : <https://github.com/QElderDelta/OOP>

## modulo.hpp

```
1 #ifndef _MODULO_H_
2 #define _MODULO_H_
3
4
5 #include <iostream>
6
7
8 class Modulo {
9     public:
10         Modulo() : number(0), mod(0) {}
11         Modulo(int number, int mod) : number(number < 0 ? mod + (
number % mod) : number), mod(mod) {}
12         Modulo Add(const Modulo& addend) const;
13         Modulo Multiply(const Modulo& multiplier) const;
14         Modulo Subtract(const Modulo& subtractend) const;
15         Modulo Divide(const Modulo& divisor) const;
16         void Read(std::istream& is);
17         void Print(std::ostream& os) const;
18         void SetNumber(int number);
19         void SetMod(int mod);
20         int GetNumber() const;
21         int GetMod() const;
22         bool IsEqual(const Modulo& to_compare) const;
23         bool IsGreater(const Modulo& to_compare) const;
24         bool IsLess(const Modulo& to_compare) const;
25     private:
26         int number;
27         int mod;
28 };
29
30 #endif
```

## modulo.cpp

```
1 #include <iostream>
2 #include <cassert>
3
4 #include "modulo.hpp"
5
6 int ExtendedEuclid(int a, int b, int& x, int& y) {
7     if(a == 0) {
8         x = 0;
9         y = 1;
10        return b;
11    }
12    int x1, y1;
13    int gcd = ExtendedEuclid(b % a, a, x1, y1);
14    x = y1 - (b / a) * x1;
15    y = x1;
16    return gcd;
17 }
18
19 Modulo Modulo::Add(const Modulo& addend) const {
```

```

20     assert(mod == addend.mod);
21     Modulo result;
22     result.number = (number % mod + addend.number % mod + mod) %
mod;
23     result.mod = mod;
24     return result;
25 }
26
27 Modulo Modulo::Multiply(const Modulo& multiplier) const {
28     assert(mod == multiplier.mod);
29     Modulo result;
30     result.number = ((number % mod) * (multiplier.number % mod) +
mod) % mod;
31     result.mod = mod;
32     return result;
33 }
34
35 Modulo Modulo::Subtract(const Modulo& subtrahend) const {
36     assert(mod == subtrahend.mod);
37     Modulo result;
38     result.number = (number % mod - subtrahend.number % mod + mod)
% mod;
39     result.mod = mod;
40     return result;
41 }
42
43 Modulo Modulo::Divide(const Modulo& divisor) const {
44     assert(mod == divisor.mod);
45     int x, y;
46     if(ExtendedEuclid(divisor.number, mod, x, y) != 1) {
47         std::cerr << "Divisor and modulo aren't coprime, therefore
division can't be made" << std::endl;
48         return {number, 0};
49     }
50     Modulo result;
51     int ModInverse = (x % mod + mod) % mod;
52     result.number = (number * ModInverse) % mod;
53     result.mod = mod;
54     return result;
55 }
56
57 void Modulo::Read(std::istream& is) {
58     is >> number >> mod;
59     if(number % mod >= 0) {
60
61         number %= mod;
62     } else {
63         number = mod + (number % mod);
64     }
65 }
66
67 void Modulo::Print(std::ostream& os) const {
68     os << number << " mod " << mod << std::endl;
69 }
70
71 void Modulo::SetNumber(int number) {
72     this->number = number;
73 }
74

```

```

75 void Modulo::SetMod(int mod) {
76     this->mod = mod;
77 }
78
79 int Modulo::GetNumber() const {
80     return number;
81 }
82
83 int Modulo::GetMod() const {
84     return mod;
85 }
86
87 bool Modulo::IsEqual(const Modulo& to_compare) const {
88     assert(mod == to_compare.mod);
89     return number == to_compare.number;
90 }
91
92 bool Modulo::IsGreater(const Modulo& to_compare) const {
93     assert(mod == to_compare.mod);
94     return number > to_compare.number;
95 }
96
97 bool Modulo::IsLess(const Modulo& to_compare) const {
98     assert(mod == to_compare.mod);
99     return number < to_compare.number;
100 }

```

## main.cpp

```

1  #include <iostream>
2
3  #include "modulo.hpp"
4
5  int main() {
6      Modulo a;
7      Modulo b;
8      Modulo c;
9
10     a.Read(std::cin);
11     b.Read(std::cin);
12
13     std::cout << "Addition:" << std::endl;
14     c = a.Add(b);
15     c.Print(std::cout);
16
17     std::cout << "Subtraction:" << std::endl;
18     c = a.Subtract(b);
19     c.Print(std::cout);
20
21     std::cout << "Multiplication:" << std::endl;
22     c = a.Multiply(b);
23     c.Print(std::cout);
24
25     std::cout << "Division:" << std::endl;
26     c = a.Divide(b);
27     if(c.GetMod()) {
28         c.Print(std::cout);
29     }
30

```

```

31     if(a.IsEqual(b)) {
32         std::cout << "Numbers are equal" << std::endl;
33     }
34
35     if(a.IsGreater(b)) {
36         std::cout << "First number is greater" << std::endl;
37     }
38
39     if(a.IsLess(b)) {
40         std::cout << "First number is less" << std::endl;
41     }
42
43     return 0;
44 }

```

## CMakeLists.txt

```

1  cmake_minimum_required(VERSION 3.1)
2
3  project(lab1)
4
5  add_executable(lab1
6      main.cpp
7      Modulo.cpp
8  )
9
10 set_property(TARGET lab1 PROPERTY CXX_STANDARD 17)
11
12 set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wextra -Werror")

```

## 2 Тестирование

**test\_01.txt:**

Входные данные:

3 5

4 5

Ожидаемый результат:

Addition:

2 mod 5

Пояснение:  $3 + 4 = 7$ ,  $7 \equiv 2 \pmod{5}$

Subtraction:

4 mod 5

Пояснение:  $3 - 4 = -1$ ,  $-1 \equiv 4 \pmod{5}$

Multiplication:

2 mod 5

Пояснение:  $3 \times 4 = 12$ ,  $12 \equiv 2 \pmod{5}$

Division:

2 mod 5

Пояснение: Необходимо найти такое  $c$ , что  $(b \times c) \pmod{5} = a \pmod{5}$ .

Легко проверяется, что  $c = 2$ , так как  $4 \times 2 = 8$ ,  $8 \equiv 3 \pmod{5}$

First number is less

Результат:

Addition:

2 mod 5

Subtraction:

4 mod 5

Multiplication:

2 mod 5

Division:

2 mod 5

First number is less

**test\_02.txt** - проверка работы с отрицательными числами:

Входные данные:

-8 5

7 5

Ожидаемый результат:

Addition:

4 mod 5

Пояснение:  $-8 + 7 = -1$ ,  $-1 \equiv 4 \pmod{5}$

Subtraction:

0 mod 5

Пояснение:  $-8 - 7 = -15$ ,  $-15 \equiv 0 \pmod{5}$

Multiplication:

4 mod 5

Пояснение:  $-8 \times 7 = -56$ ,  $-56 \equiv 4 \pmod{5}$

Division:

2 mod 5

Пояснение: Необходимо найти такое  $c$ , что  $(b \times c) \bmod 5 = a \bmod 5$ .

Легко проверяется, что  $c = 1$ , так как  $7 \times 1 = 7$ ,  $-8 \equiv 2 \bmod 5$ ,  $7 \equiv 2 \bmod 5$

Numbers are equal

Пояснение:  $-8 \equiv 2 \bmod 5$ ,  $7 \equiv 2 \bmod 5$

Результат:

Addition:

4 mod 5

Subtraction:

0 mod 5

Multiplication:

4 mod 5

Division:

1 mod 5

Numbers are equal

**test\_03.txt** - проверка деления:

Входные данные:

11 10

4 10

Ожидаемый результат:

Addition:

5 mod 10

Пояснение:  $11 \equiv 1 \bmod 10$ ,  $1 + 4 = 5$ ,  $5 \equiv 5 \bmod 10$

Subtraction:

7 mod 10

Пояснение:  $11 \equiv 1 \bmod 10$ ,  $1 - 4 = -3$ ,  $-3 \equiv 7 \bmod 10$

Multiplication:

4 mod 10

Пояснение:  $11 \equiv 1 \bmod 10$ ,  $1 \times 4 = 4$ ,  $4 \equiv 4 \bmod 10$

Division:

Divisor and aren't coprime, therefore division can't be made

Пояснение: Так обязательным условием существования обратного числа по данному модулю является взаимная простота этого числа и модуля, а 4 и 10 таковыми не являются, то деление произвести нельзя.

First number is less

Пояснение:  $11 \equiv 1 \bmod 10$ , 1,  $1 < 4$

Результат:

Addition:

5 mod 10

Subtraction:

7 mod 10

Multiplication:

4 mod 10

Division:

Divisor and aren't coprime, therefore division can't be made

First number is less

**test\_04.txt** - проверка невозможности работы с разными модулями:

Входные данные:

1 2

3 4

Ожидаемый результат:

Падение программы в результате невыполнения одного из assert'ов

Результат:

Addition: lab1: /home/qelderdelta/Study/OOP/lab1/Modulo.cpp:20: Modulo  
Modulo::Add(const Modulo&) const: Assertion 'mod == addend.mod' failed. Ава-  
рийный останов (стек памяти сброшен на диск)



### 3 Объяснение результатов работы программы

При выполнении лабораторной работы были использованы следующие свойства модулярной арифметики:

$$(a + b) \bmod m = ((a \bmod m) + (b \bmod m) + m) \bmod m$$

$$(a * b) \bmod m = ((a \bmod m) - (b \bmod m) + m) \bmod m$$

$$(a * b) \bmod m = ((a \bmod m) * (b \bmod m) + m) \bmod m$$

$$(a/b) \bmod m = (a * b^{-1}) \bmod m$$

В каждом случае прибавлялось  $m$  к получившемуся результату для того, чтобы избежать отрицательных чисел. Особо интересным является деление, так как его не всегда можно произвести. Делитель должен иметь обратное число, необходимым условием чего является взаимная простота его и модуля. Для нахождения обратного числа использовался расширенный алгоритм Евклида, который помимо НОДа двух чисел находит такие  $x$  и  $y$ , что:

$$a \times x + b \times y = \gcd(a, b)$$

И, если НОД равен 1, то обратное число равно:

$$\text{modInverse} = (x \bmod m + m) \bmod m$$