

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

**Лабораторная работа №2 по курсу  
“Объектно-ориентированное программирование”**

*Студент:* Живалев Е.А.

*Группа:* М8О-206Б

*Преподаватель:* Журавлев А.А.

*Вариант:* 5

*Оценка:* \_\_\_\_\_

*Дата:* \_\_\_\_\_

Москва  
2019

# 1 Исходный код

Ссылка на github : [https://github.com/QElderDelta/oop\\_exercise\\_02](https://github.com/QElderDelta/oop_exercise_02)

## modulo.hpp

```
1 #ifndef _MODULO_H_
2 #define _MODULO_H_
3
4
5 #include <iostream>
6 #include <cassert>
7
8 class Modulo {
9     public:
10         Modulo() : number(0), mod(0) {}
11         Modulo(int number, int mod);
12         Modulo& operator+=(const Modulo& rhs);
13         Modulo& operator*=(const Modulo& rhs);
14         Modulo& operator-=(const Modulo& rhs);
15         Modulo& operator/=(const Modulo& rhs);
16         friend std::istream& operator>>(std::istream& is, Modulo&
mod);
17         friend std::ostream& operator<<(std::ostream& os, const
Modulo& mod);
18         void SetNumber(int number);
19         void SetMod(int mod);
20         int GetNumber() const;
21         int GetMod() const;
22         friend bool operator==(const Modulo& lhs, const Modulo&
rhs);
23         friend bool operator>(const Modulo& lhs, const Modulo& rhs
);
24         friend bool operator<(const Modulo& lhs, const Modulo& rhs
);
25     private:
26         int number;
27         int mod;
28 };
29
30 Modulo operator"" _mod(const char* str, std::size_t);
31
32 #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(int number, int mod) {
20     assert(mod != 0);
21     this->mod = mod;
22     if(number > 0) {
23         this->number = number % mod;
24     } else {
25         this->number = (number % mod) + mod;
26     }
27 }
28
29 Modulo& Modulo::operator+=(const Modulo& rhs) {
30     assert(this->mod == rhs.mod);
31     number = (number % mod + rhs.number % mod + mod) % mod;
32     return *this;
33 }
34
35 Modulo& Modulo::operator*=(const Modulo& rhs) {
36     assert(this->mod == rhs.mod);
37     this->number = ((this->number % this->mod) * (rhs.number %
38     this->mod) + this->mod) % this->mod;
39     return *this;
40 }
41
42 Modulo& Modulo::operator-=(const Modulo& rhs) {
43     assert(this->mod == rhs.mod);
44     this->number = (this->number % this->mod - rhs.number % this->
45     mod + this->mod) % this->mod;
46     return *this;
47 }
48
49 Modulo& Modulo::operator/=(const Modulo& rhs) {
50     assert(this->mod == rhs.mod);
51     int x, y;
52     if(ExtendedEuclid(rhs.number, this->mod, x, y) != 1) {
53         throw std::invalid_argument("Divisor and aren't coprime,
54         therefore division can't be made");
55     }
56     int ModInverse = (x % this->mod + this->mod) % this->mod;
57     this->number = (this->number * ModInverse) % this->mod;
58     return *this;
59 }
60
61 Modulo operator+(Modulo lhs, const Modulo& rhs) {
62     assert(lhs.mod == rhs.mod);
63     lhs += rhs;
64     return lhs;
65 }
66
67 Modulo operator*(Modulo lhs, const Modulo& rhs) {
68     assert(lhs.mod == rhs.mod);
69     lhs *= rhs;
70     return lhs;
71 }

```

```

70 Modulo operator-(Modulo lhs, const Modulo& rhs) {
71     assert(lhs.mod == rhs.mod);
72     lhs -= rhs;
73     return lhs;
74     Modulo result;
75 }
76
77 Modulo operator/(Modulo lhs, const Modulo& rhs) {
78     assert(lhs.mod == rhs.mod);
79     lhs /= rhs;
80     return lhs;
81 }
82
83 std::istream& operator>>(std::istream& is, Modulo& m) {
84     is >> m.number >> m.mod;
85     assert(m.mod != 0);
86     if(m.number % m.mod >= 0) {
87         m.number %= m.mod;
88     } else {
89         m.number = m.mod + (m.number % m.mod);
90     }
91     return is;
92 }
93
94 std::ostream& operator<<(std::ostream& os, const Modulo& m) {
95     os << m.number << " mod " << m.mod;
96     return os;
97 }
98
99 void Modulo::SetNumber(int number) {
100     if(number >= 0) {
101         this->number = number % this->mod;
102     } else {
103         this->number = (number % this->mod) + this->mod;
104     }
105 }
106
107 void Modulo::SetMod(int mod) {
108     assert(mod != 0);
109     this->mod = mod;
110     this->number %= mod;
111 }
112
113 int Modulo::GetNumber() const {
114     return number;
115 }
116
117 int Modulo::GetMod() const {
118     return mod;
119 }
120
121 bool operator==(const Modulo& lhs, const Modulo& rhs) {
122     assert(lhs.mod == rhs.mod);
123     return lhs.number == rhs.number;
124 }
125
126 bool operator>(const Modulo& lhs, const Modulo& rhs) {
127     assert(lhs.mod == rhs.mod);
128     return lhs.number > rhs.number;

```

```

129 }
130
131 bool operator<(const Modulo& lhs, const Modulo& rhs) {
132     assert(lhs.mod == rhs.mod);
133     return lhs.number < rhs.number;
134 }
135
136 Modulo operator"" _mod(const char* str, std::size_t) {
137     std::string number, mod;
138     int i = 0;
139     while(str[i] != '%') {
140         number += str[i];
141         i++;
142     }
143     i++;
144     while(str[i] != '\\0') {
145         mod = str[i];
146         i++;
147     }
148     return Modulo(std::stoi(number), std::stoi(mod));
149 }

```

## main.cpp

```

1 #include <iostream>
2
3 #include "modulo.hpp"
4
5
6 int main() {
7     Modulo a;
8     Modulo b;
9     Modulo c;
10
11     std::cin >> a >> b;
12
13     std::cout << "Addition:" << std::endl;
14     c = a + b;
15     std::cout << c << std::endl;
16
17     std::cout << "Subtraction:" << std::endl;
18     c = a - b;
19     std::cout << c << std::endl;
20
21     std::cout << "Multiplication:" << std::endl;
22     c = a * b;
23     std::cout << c << std::endl;
24
25     std::cout << "Division:" << std::endl;
26     try {
27         c = a / b;
28     } catch(std::exception& e) {
29         std::cerr << e.what() << std::endl;
30     }
31     std::cout << c << std::endl;
32
33     if(a == b) {
34         std::cout << "Numbers are equal" << std::endl;
35     }

```

```

36
37     if(a > b) {
38         std::cout << "First number is greater" << std::endl;
39     }
40
41     if(a < b) {
42         std::cout << "First number is less" << std::endl;
43     }
44
45     Modulo d = "5%3"_mod;
46     std::cout << d.GetNumber() << std::endl;
47     std::cout << d.GetMod() << std::endl;
48
49     return 0;
50 }

```

## test.cpp

```

1  #include <gtest/gtest.h>
2  #include "modulo.cpp"
3
4  TEST(ModuloTest, ConstructorTest) {
5      Modulo a{-8, 5};
6      ASSERT_EQ(a.GetMod(), 5);
7      ASSERT_EQ(a.GetNumber(), 2);
8      Modulo b{2, 5};
9      ASSERT_EQ(b.GetMod(), 5);
10     ASSERT_EQ(b.GetNumber(), 2);
11     Modulo c{8, 5};
12     ASSERT_EQ(c.GetMod(), 5);
13     ASSERT_EQ(c.GetNumber(), 3);
14 }
15
16 TEST(ModuloTest, AdditionTest) {
17     Modulo a{-8, 5};
18     Modulo b{7, 5};
19     Modulo c = a + b;
20     ASSERT_EQ(c.GetNumber(), 4);
21     a += b;
22     ASSERT_EQ(a.GetNumber(), 4);
23     a.SetMod(3);
24     a.SetNumber(3);
25     b.SetMod(3);
26     b.SetNumber(1);
27     a += b;
28     ASSERT_EQ(a.GetNumber(), 1);
29 }
30
31 TEST(ModuloTest, SubtractionTest) {
32     Modulo a{-8, 5};
33     Modulo b{7, 5};
34     Modulo c = a - b;
35     ASSERT_EQ(c.GetNumber(), 0);
36     a -= b;
37     ASSERT_EQ(a.GetNumber(), 0);
38     a.SetMod(3);
39     a.SetNumber(3);
40     b.SetMod(3);
41     a.SetNumber(1);

```

```

42     a -= b;
43     ASSERT_EQ(a.GetNumber(), 2);
44 }
45
46 TEST(ModuloTest, MultiplicationTest) {
47     Modulo a{-8, 5};
48     Modulo b{7, 5};
49     Modulo c = a * b;
50     ASSERT_EQ(c.GetNumber(), 4);
51     a *= b;
52     ASSERT_EQ(a.GetNumber(), 4);
53     a.SetMod(3);
54     a.SetNumber(3);
55     b.SetMod(3);
56     b.SetNumber(1);
57     a *= b;
58     ASSERT_EQ(a.GetNumber(), 0);
59 }
60
61 TEST(ModuloTest, DivisionTest) {
62     Modulo a{-8, 5};
63     Modulo b{7, 5};
64     Modulo c = a / b;
65     ASSERT_EQ(c.GetNumber(), 1);
66     a /= b;
67     ASSERT_EQ(a.GetNumber(), 1);
68     a.SetMod(3);
69     a.SetNumber(3);
70     b.SetMod(3);
71     b.SetNumber(1);
72     a /= b;
73     ASSERT_EQ(a.GetNumber(), 0);
74 }
75
76 TEST(ModuloTest, ComparisionTest) {
77     Modulo a{-8, 5};
78     Modulo b{7, 5};
79     ASSERT_TRUE(a == b);
80     ASSERT_FALSE(a < b);
81     ASSERT_FALSE(a > b);
82     a.SetNumber(3);
83     a.SetMod(3);
84     b.SetNumber(1);
85     b.SetMod(3);
86     ASSERT_FALSE(a == b);
87     ASSERT_TRUE(a < b);
88     ASSERT_FALSE(a > b);
89 }
90
91 TEST(ModuloTest, LiteralTest) {
92     Modulo a = "-8%5"_mod;
93     ASSERT_EQ(a.GetNumber(), 2);
94     ASSERT_EQ(a.GetMod(), 5);
95     a = "7%3"_mod;
96     ASSERT_EQ(a.GetNumber(), 1);
97     ASSERT_EQ(a.GetMod(), 3);
98 }
99
100 int main(int argc, char** argv) {

```

```
101     ::testing::InitGoogleTest(&argc, argv);
102     return RUN_ALL_TESTS();
103 }
```

## CMakeLists.txt

```
1 cmake_minimum_required(VERSION 3.1)
2
3 project(lab2)
4
5 enable_testing()
6
7 find_package(GTest REQUIRED)
8
9 add_subdirectory(googletest)
10
11 include_directories(googletest/googletest/include)
12
13 add_executable(ModuloTest
14     test.cpp
15 )
16
17 target_link_libraries(ModuloTest gtest gtest_main)
18
19 gtest_add_tests(ModuloTest "" AUTO)
20
21 add_executable(lab2
22     main.cpp
23     modulo.cpp
24 )
25
26 set_property(TARGET lab2 PROPERTY CXX_STANDARD 17)
27
28 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:

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

## 4 Выводы

В ходе выполнения лабораторной работы я еще раз убедился в том, что написание функций для операций - зло, ведь есть механизм переопределения операторов, который позволяет делать это намного удобнее и элегантнее. Нельзя не отметить и то, насколько пользовательские литералы могут повысить читаемость кода. Конкретно в моём случае это не так ощутимо, но при работе с единицами измерения, например, метрами, километрами и т.д. код становится значительно элегантнее. Также я познакомился с такой замечательной утилитой как `gtest`, которая позволяет сильно упростить написание юнит-тестов.