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

modulo.hpp

```
#ifndef _MODULO_H_
#define _MODULO_H_


#include <iostream>
#include <cassert>

class Modulo {
    public:
        Modulo() : number(0), mod(0) {}
        Modulo(int number, int mod);
        Modulo& operator+=(const Modulo& rhs);
        Modulo& operator*=(const Modulo& rhs);
        Modulo& operator-=(const Modulo& rhs);
        Modulo& operator/=(const Modulo& rhs);
        friend Modulo operator+(Modulo lhs, const Modulo& rhs);
        friend Modulo operator*(Modulo lhs, const Modulo& rhs);
        friend Modulo operator-(Modulo lhs, const Modulo& rhs);
        friend Modulo operator/(Modulo lhs, const Modulo& rhs);
        friend std::istream& operator>>(std::istream& is, Modulo&
    mod);
        friend std::ostream& operator<<(std::ostream& os, const
    Modulo& mod);
        void SetNumber(int number);
        void SetMod(int mod);
        int GetNumber() const;
        int GetMod() const;
        friend bool operator==(const Modulo& lhs, const Modulo&
    rhs);
        friend bool operator>(const Modulo& lhs, const Modulo& rhs
    );
        friend bool operator<(const Modulo& lhs, const Modulo& rhs
    );
    private:
        int number;
        int mod;
};

Modulo operator"" _mod(const char* str, std::size_t);

#endif
```

modulo.cpp

```
#include <iostream>
#include <cassert>

#include "modulo.hpp"

int ExtendedEuclid(int a, int b, int& x, int& y) {
    if(a == 0) {
        x = 0;
        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 %
        this->mod) + this->mod) % this->mod;
38      return *this;
39  }
40
41  Modulo& Modulo::operator-=(const Modulo& rhs) {
42      assert(this->mod == rhs.mod);
43      this->number = (this->number % this->mod - rhs.number % this->
        mod + this->mod) % this->mod;
44      return *this;
45  }
46
47  Modulo& Modulo::operator/=(const Modulo& rhs) {
48      assert(this->mod == rhs.mod);
49      int x, y;
50      if(ExtendedEuclid(rhs.number, this->mod, x, y) != 1) {
51          throw std::invalid_argument("Divisor and aren't coprime,
        therefore division can't be made");
52      }
53      int ModInverse = (x % this->mod + this->mod) % this->mod;
54      this->number = (this->number * ModInverse) % this->mod;
55      return *this;
56  }
57
58  Modulo operator+(Modulo lhs, const Modulo& rhs) {
59      assert(lhs.mod == rhs.mod);
60      lhs += rhs;
61      return lhs;
62  }
63
64  Modulo operator*(Modulo lhs, const Modulo& rhs) {
65      assert(lhs.mod == rhs.mod);
```

```
    lhs *= rhs;
    return lhs;
}

Modulo operator-(Modulo lhs, const Modulo& rhs) {
    assert(lhs.mod == rhs.mod);
    lhs -= rhs;
    return lhs;
    Modulo result;
}

Modulo operator/(Modulo lhs, const Modulo& rhs) {
    assert(lhs.mod == rhs.mod);
    lhs /= rhs;
    return lhs;
}

std::istream& operator>>(std::istream& is, Modulo& m) {
    is >> m.number >> m.mod;
    assert(m.mod != 0);
    if(m.number % m.mod >= 0) {
        m.number %= m.mod;
    } else {
        m.number = m.mod + (m.number % m.mod);
    }
    return is;
}

std::ostream& operator<<(std::ostream& os, const Modulo& m) {
    os << m.number << " mod " << m.mod;
    return os;
}

void Modulo::SetNumber(int number) {
    this->number = number % this->mod;
}

void Modulo::SetMod(int mod) {
    this->mod = mod;
    this->number %= mod;
}

int Modulo::GetNumber() const {
    return number;
}

int Modulo::GetMod() const {
    return mod;
}

bool operator==(const Modulo& lhs, const Modulo& rhs) {
    assert(lhs.mod == rhs.mod);
    return lhs.number == rhs.number;
}

bool operator>(const Modulo& lhs, const Modulo& rhs) {
    assert(lhs.mod == rhs.mod);
    return lhs.number > rhs.number;
}
```

```
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 Modulo operator"" _mod(const char* str, std::size_t) {
132      std::string number, mod;
133      int i = 0;
134      while(str[i] != '%') {
135          number += str[i];
136        i++;
137      }
138      i++;
139      while(str[i] != '\0') {
140          mod = str[i];
141          i++;
142      }
143      return Modulo(std::stoi(number), std::stoi(mod));
144 }
```

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
```

```
    if(a > b) {
        std::cout << "First number is greater" << std::endl;
    }

    if(a < b) {
        std::cout << "First number is less" << std::endl;
    }

    Modulo d = "5%3"_mod;
    std::cout << d.GetNumber() << std::endl;
    std::cout << d.GetMod() << std::endl;

    return 0;
}
```

test.cpp

```
#include <gtest/gtest.h>
#include "modulo.cpp"

TEST(ModuloTest, ConstructorTest) {
    Modulo a{-8, 5};
    ASSERT_EQ(a.GetMod(), 5);
    ASSERT_EQ(a.GetNumber(), 2);
    Modulo b{2, 5};
    ASSERT_EQ(b.GetMod(), 5);
    ASSERT_EQ(b.GetNumber(), 2);
    Modulo c{8, 5};
    ASSERT_EQ(c.GetMod(), 5);
    ASSERT_EQ(c.GetNumber(), 3);
}

TEST(ModuloTest, AdditionTest) {
    Modulo a{-8, 5};
    Modulo b{7, 5};
    Modulo c = a + b;
    ASSERT_EQ(c.GetNumber(), 4);
    a += b;
    ASSERT_EQ(a.GetNumber(), 4);
    a.SetNumber(3);
    a.SetMod(3);
    b.SetNumber(1);
    b.SetMod(3);
    a += b;
    ASSERT_EQ(a.GetNumber(), 1);
}

TEST(ModuloTest, SubtractionTest) {
    Modulo a{-8, 5};
    Modulo b{7, 5};
    Modulo c = a - b;
    ASSERT_EQ(c.GetNumber(), 0);
    a -= b;
    ASSERT_EQ(a.GetNumber(), 0);
    a.SetNumber(3);
    a.SetMod(3);
    b.SetNumber(1);
    b.SetMod(3);
    a -= b;
```

```
    ASSERT_EQ(a.GetNumber(), 2);
}

TEST(ModuloTest, MultiplicationTest) {
    Modulo a{-8, 5};
    Modulo b{7, 5};
    Modulo c = a * b;
    ASSERT_EQ(c.GetNumber(), 4);
    a *= b;
    ASSERT_EQ(a.GetNumber(), 4);
    a.SetNumber(3);
    a.SetMod(3);
    b.SetNumber(1);
    b.SetMod(3);
    a *= b;
    ASSERT_EQ(a.GetNumber(), 0);
}

TEST(ModuloTest, DivisionTest) {
    Modulo a{-8, 5};
    Modulo b{7, 5};
    Modulo c = a / b;
    ASSERT_EQ(c.GetNumber(), 1);
    a /= b;
    ASSERT_EQ(a.GetNumber(), 1);
    a.SetNumber(3);
    a.SetMod(3);
    b.SetNumber(1);
    b.SetMod(3);
    a /= b;
    ASSERT_EQ(a.GetNumber(), 0);
}

TEST(ModuloTest, ComparisionTest) {
    Modulo a{-8, 5};
    Modulo b{7, 5};
    ASSERT_TRUE(a == b);
    ASSERT_FALSE(a < b);
    ASSERT_FALSE(a > b);
    a.SetNumber(3);
    a.SetMod(3);
    b.SetNumber(1);
    b.SetMod(3);
    ASSERT_FALSE(a == b);
    ASSERT_TRUE(a < b);
    ASSERT_FALSE(a > b);
}

int main(int argc, char** argv) {
    ::testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.1)

project(lab2)

```

```
enable_testing()

find_package(GTest REQUIRED)

add_subdirectory(googletest)

include_directories(googletest/googletest/include)

add_executable(ModuloTest
    test.cpp
)

target_link_libraries(ModuloTest gtest gtest_main)

gtest_add_tests(ModuloTest "" AUTO)

add_executable(lab2
    main.cpp
    modulo.cpp
)

set_property(TARGET lab2 PROPERTY CXX_STANDARD 17)

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: $3 + 4 = 7,\ 7 \equiv 2\ mod\ 5$
Subtraction:
4 mod 5
Пояснение: $3 - 4 = -1,\ -1 \equiv 4\ mod\ 5$
Multiplication:
2 mod 5
Пояснение: $3 \times 4 = 12,\ 12 \equiv 2\ mod\ 5$
Division:
2 mod 5
Пояснение: Необходимо найти такое c, что $(b \times c)\ mod\ 5 = a\ mod\ 5$.
Легко проверяется, что $c = 2$, так как $4 \times 2 = 8, 8 \equiv 3\ mod\ 5$
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: $-8 + 7 = -11,\ -11 \equiv 4\ mod\ 5$
Subtraction:
0 mod 5
Пояснение: $-8 - 7 = -15,\ -15 \equiv 0\ mod\ 5$
Multiplication:
4 mod 5
Пояснение: $-8 \times 7 = -56,\ -56 \equiv 4\ mod\ 5$
Division:

2 mod 5
Пояснение: Необходимо найти такое c, что $(b \times c)\ mod\ 5 = a\ mod\ 5$.
Легко проверяется, что $c = 1$, так как $7 \times 1 = 7, -8 \equiv 2\ mod\ 5, 7 \equiv 2\ mod\ 5$
Numbers are equal
Пояснение: $-8 \equiv 2\ mod\ 5,\ 7 \equiv 2\ mod\ 5$

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:
Входнные данные:
11 10
4 10

Ожидаемый результат:
Addition:
5 mod 10
Пояснение: $11 \equiv 1\ mod\ 10,\ 1 + 4 = 5,\ 5 \equiv 5\ mod\ 10$
Subtraction:
7 mod 10
Пояснение: $11 \equiv 1\ mod\ 10,\ 1 - 4 = -3,\ -3 \equiv 7\ mod\ 10$
Multiplication:
4 mod 10
Пояснение: $11 \equiv 1\ mod\ 10,\ 1 \times 4 = 4,\ 4 \equiv 4\ mod\ 10$
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: $11 \equiv 1\ mod\ 10,\ 1,\ 1 < 4$

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:
Входнные 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. Аварийный останов (стек памяти сброшен на диск)

3 Объяснение результатов работы программы

При выполнении лабораторной работы были использованы следующие свойства модулярной арифметики:

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

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

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

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

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 числе находит такие x и y, что:

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

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

$$modInverse = (x\ mod\ m + m)\ mod\ m$$

4 Выводы

В ходе выполнения лабораторной работы я 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.