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: 5

Оценка: _____

Дата: _____

Москва
2019

1 Исходный код

Ссылка на github : https://github.com/QElderDelta/oop_exercise_01

modulo.hpp

```
#ifndef _MODULO_H_
#define _MODULO_H_


#include <iostream>


class Modulo {
    public:
        Modulo() : number(0), mod(0) {}
        Modulo(int number, int mod) : number(number < 0 ? mod + (number % mod) : number % mod), mod(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);
        friend Modulo operator"" _mod(const char* c, std::size_t);
        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;
};

#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::operator+=(const Modulo& rhs) {
20      assert(this->mod == rhs.mod);
21      number = (number % mod + rhs.number % mod + mod) % mod;
22      return *this;
23  }
24
25  Modulo& Modulo::operator*=( const Modulo& rhs) {
26      assert(this->mod == rhs.mod);
27      this->number = ((this->number % this->mod) * (rhs.number %
        this->mod) + this->mod) % this->mod;
28      return *this;
29  }
30
31  Modulo& Modulo::operator-=(const Modulo& rhs) {
32      assert(this->mod == rhs.mod);
33      this->number = (this->number % this->mod - rhs.number % this->
        mod + this->mod) % this->mod;
34      return *this;
35  }
36
37  Modulo& Modulo::operator/=(const Modulo& rhs) {
38      assert(this->mod == rhs.mod);
39      int x, y;
40      if(ExtendedEuclid(rhs.number, this->mod, x, y) != 1) {
41          throw std::invalid_argument("Divisor and aren't coprime,
        therefore division can't be made");
42      }
43      int ModInverse = (x % this->mod + this->mod) % this->mod;
44      this->number = (this->number * ModInverse) % this->mod;
45      return *this;
46  }
47
48  Modulo operator+(Modulo lhs, const Modulo& rhs) {
49      assert(lhs.mod == rhs.mod);
50      lhs += rhs;
51      return lhs;
52  }
53
54  Modulo operator*(Modulo lhs, const Modulo& rhs) {
55      assert(lhs.mod == rhs.mod);
56      lhs *= rhs;
57      return lhs;
58  }
59
60  Modulo operator-(Modulo lhs, const Modulo& rhs) {
61      assert(lhs.mod == rhs.mod);
62      lhs -= rhs;
63      return lhs;
64      Modulo result;
65  }
```

```
66
67 Modulo operator/(Modulo lhs, const Modulo& rhs) {
68     assert(lhs.mod == rhs.mod);
69     lhs /= rhs;
70     return lhs;
71 }
72
73 std::istream& operator>>(std::istream& is, Modulo& m) {
74     is >> m.number >> m.mod;
75     if(m.number % m.mod >= 0) {
76         m.number %= m.mod;
77     } else {
78         m.number = m.mod + (m.number % m.mod);
79     }
80     return is;
81 }
82
83 std::ostream& operator<<(std::ostream& os, const Modulo& m) {
84     os << m.number << " mod " << m.mod;
85     return os;
86 }
87
88 Modulo operator""_mod(const char* str, std::size_t) {
89      std::string number, mod;
90      int i = 0;
91      while(str[i] != '%') {
92          number += str[i];
93        i++;
94      }
95      i++;
96      while(str[i] != '\0') {
97          mod = str[i];
98          i++;
99      }
100     std::cerr << std::stoi(number) % std::stoi(mod) << std::endl;
101     return Modulo(std::stoi(number), std::stoi(mod));
102 }
103
104 void Modulo::SetNumber(int number) {
105     this->number = number;
106 }
107
108 void Modulo::SetMod(int mod) {
109     this->mod = mod;
110 }
111
112 int Modulo::GetNumber() const {
113     return number;
114 }
115
116 int Modulo::GetMod() const {
117     return mod;
118 }
119
120 bool operator==(const Modulo& lhs, const Modulo& rhs) {
121     assert(lhs.mod == rhs.mod);
122     return lhs.number == rhs.number;
123 }
124
```

```cpp
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;
}
```

main.cpp

```cpp
#include <iostream>

#include "modulo.hpp"


int main() {
    Modulo a;
    Modulo b;
    Modulo c;

    std::cin >> a >> b;

    std::cout << "Addition:" << std::endl;
    c = a + b;
    std::cout << c << std::endl;

    std::cout << "Subtraction:" << std::endl;
    c = a - b;
    std::cout << c << std::endl;

    std::cout << "Multiplication:" << std::endl;
    c = a * b;
    std::cout << c << std::endl;

    std::cout << "Division:" << std::endl;
    try {
        c = a / b;
    } catch(std::exception& e) {
        std::cerr << e.what() << std::endl;
    }

    if(a == b) {
        std::cout << "Numbers are equal" << std::endl;
    }

    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;
```

```
}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.1)

project(lab2)

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:
Входнные 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:
Входнные 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 впервые познакомился с таким инструментом как CMake, который, на мой взгляд, является очень удобным. Также я еще раз убедился в том, что написание функций для операций - зло, ведь есть механизм переопределения операторов.