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

```
#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) {}
        Modulo Add(const Modulo& addend) const;
        Modulo Multiply(const Modulo& multiplier) const;
        Modulo Subtract(const Modulo& subtracthend) const;
        Modulo Divide(const Modulo& divisor) const;
        void Read(std::istream& is);
        void Print(std::ostream& os) const;
        void SetNumber(int number);
        void SetMod(int mod);
        int GetNumber() const;
        int GetMod() const;
        bool IsEqual(const Modulo& to_compare) const;
        bool IsGreater(const Modulo& to_compare) const;
        bool IsLess(const Modulo& to_compare) const;
    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;
        return b;
    }
    int x1, y1;
    int gcd = ExtendedEuclid(b % a, a, x1, y1);
    x = y1 - (b / a) * x1;
    y = x1;
    return gcd;
}

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 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          number %= mod;
61      } else {
62          number = mod + (number % mod);
63      }
64  }
65
66  void Modulo::Print(std::ostream& os) const {
67      os << number << " mod " << mod << std::endl;
68  }
69
70  void Modulo::SetNumber(int number) {
71      this->number = number;
72  }
73
74  void Modulo::SetMod(int mod) {
```

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

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:
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:

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