Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа**

**по курсу «Объектно-ориентированное программирование»**

**III Семестр**

**Задание 2  
Вариант 5**

**Операторы, литералы**

|  |  |
| --- | --- |
| Студент: | Валов Вадим Вячеславович |
| Группа: | М8О-208Б-18 |
| Преподаватель: | Журалвев А.А. |
|  |  |
| Оценка: |  |
| Дата: |  |

# 1. Код программы на языке С++

**1. Файл modulo.h**

#ifndef \_MODULO\_H\_

#define \_MODULO\_H\_

#include <iostream>

#include <sstream>

**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& addend) **const**;

Modulo **operator**\*(**const** Modulo& multiplier) **const**;

Modulo **operator**-(**const** Modulo& subtracthend) **const**;

Modulo **operator**/(**const** Modulo& divisor) **const**;

**void** **Read**(std::istream& is);

**void** Print(std::ostream& os) **const**;

**void** **SetNumber**(**int** number,**int** h);

**void** **SetMod**(**int** mod, **int** f);

**int** GetNumber() **const**;

**int** GetMod() **const**;

**bool** **operator**==(**const** Modulo& to\_compare);

**bool** **operator**>(**const** Modulo& to\_compare);

**bool** **operator**<(**const** Modulo& to\_compare);

**private:**

**int** number;

**int** mod;

**int** h;

**int** f;

};

Modulo **operator**""\_mod(**const** **char**\* str, **size\_t** size);

#endif

**2. Файл 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::**operator**+(**const** Modulo& addend) **const** {

assert(mod == addend.mod);

Modulo result;

result.number = (number + addend.number)%mod;

result.mod = mod;

**return** result;

}

Modulo Modulo::**operator**\*(**const** Modulo& multiplier) **const** {

assert(mod == multiplier.mod);

Modulo result;

result.number = (number \* multiplier.number)%mod;

result.mod = mod;

**return** result;

}

Modulo Modulo::**operator**-(**const** Modulo& subtrahend) **const** {

assert(mod == subtrahend.mod);

Modulo result;

result.number = abs((number%mod - subtrahend.number%mod + mod) % mod);

result.mod = mod;

**return** result;

}

**int** amodb(**int** a, **int** b){

**return** ( a % b );

}

Modulo Modulo::**operator**/(**const** Modulo& divisor) **const** {

assert(mod == divisor.mod);

**int** x, y, gcd;

gcd = ExtendedEuclid(divisor.number, mod, x, y);

assert(gcd == **1**);

Modulo result;

**int** ModInverse = (x % mod + mod) % mod;

result.number = (number \* ModInverse) % mod;

result.mod = mod;

**return** result;

}

**void** Modulo::Read(std::istream& is) {

is >> number >> mod;

**if**(number % mod >= **0**) {

number %= mod;

} **else** {

number = mod + (number % mod);

}

}

**void** Modulo::Print(std::ostream& os) **const** {

os << number << " mod " << mod << std::endl;

}

**void** Modulo::SetNumber(**int** number, **int** h) {

**this**->h = number;

}

**void** Modulo::SetMod(**int** mod, **int** f) {

**this**->f = mod;

}

**int** Modulo::GetNumber() **const** {

**return** number;

}

**int** Modulo::GetMod() **const** {

**return** mod;

}

**bool** Modulo::**operator**==(**const** Modulo& to\_compare){

**return**(**this**->number == to\_compare.number);

}

**bool** Modulo::**operator**>(**const** Modulo& to\_compare){

**return**(**this**->number > to\_compare.number);

}

**bool** Modulo::**operator**<(**const** Modulo& to\_compare){

**return**(**this**->number < to\_compare.number);

}

Modulo **operator**""\_mod(**const** **char**\* str, **size\_t** size) {//"2,3"\_mod

std::istringstream is(str);

**char** tmp;

**int** h, f;

is >> h >> tmp >> f ;

**return** {h,f};

}

**3. Файл main.cpp**

#include <iostream>

#include "modulo.hpp"

**int** **main**() {

Modulo a;

Modulo b;

Modulo c;

a.Read(std::cin);

b.Read(std::cin);

std::cout << "Addition:" << std::endl;

c = a+b;

c.Print(std::cout);

std::cout << "Subtraction:" << std::endl;

c = a-b;

c.Print(std::cout);

std::cout << "Multiplication:" << std::endl;

c = a\*b;

c.Print(std::cout);

**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;

}

std::cout << "Division:" << std::endl;

c = a/b;

**if**(c.GetMod()) {

c.Print(std::cout);

}

**return** **0**;

}

**2. Ссылка на репозиторий на Github**

https://github.com/vindosVP/oop\_exercise\_02

**3.Набор тестов**

Test01.txt

6 3

2 3

Test02.txt

9 4

3 4

Test03.txt

6 3

2 3

3

**4. Результат выполнения тестов**

Result\_test01.txt

Addition:

2 mod 3

Subtraction:

2 mod 3

Multiplication:

0 mod 3

Division:

0 mod 3

First number is less

Result\_test02.txt

Addition:

0 mod 4

Subtraction:

2 mod 4

Multiplication:

3 mod 4

Division:

3 mod 4

First number is less

Result\_test03.txt

Addition:

2 mod 3

Subtraction:

1 mod 3

Multiplication:

0 mod 3

Division:

0 mod 3

First number is less

**Объяснение работы программы**

Данная программа создает класс Modulo для работы с целыми числами по модулю N. В классе есть быть два поля: число и N.

В программе раелизованы следующие операции:

1.Сравнение чисел по модулю N

2. Сложение и вычитание по модулю N

3. Умножение и деление по модулю N

Вывод: Проделав данную работу я изучил перегрузку операторов и пользовательские литералы. Сделал вывод, что перегрузка операторов необходима для переопределения того, что должен делать данный оператор, если это необходимо. Например, если класс реализован двумя переменными, и их необходимо сравнить как одно целое, то необходимо применить перегрузку операторов. Так же пользовательские литералы необходимы для просчета константных переменных где это возможно, так как расчеты производятся на этапе компиляции.