ГОУ ВПО «ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

КАФЕДРА ПРОГРАММНОЙ ИНЖЕНЕРИИ

Отчет

по производственной практике

Руководитель:

асс. каф. ПИ

Филипишин Д.А.

\_\_\_\_.\_\_\_\_.2021г.

Выполнил:

ст. гр. ПИ-18б

Моргунов А. Г.

\_\_\_\_.\_\_\_\_.2021г.

Донецк – 2020

СОДЕРЖАНИЕ

[Введение 3](#_Toc76375463)

[1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И ПОСТАНОВКА ЗАДАЧИ 4](#_Toc76375464)

[1.1 Постановка задачи 4](#_Toc76375465)

[1.2 Распределение задач для совместной разработки 4](#_Toc76375466)

[1.3 Анализ предметной области 4](#_Toc76375467)

[2 ОБОСНОВАНИЕ ВЫБОРА ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ 6](#_Toc76375468)

[3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОЙ СИСТЕМЫ 7](#_Toc76375469)

[4 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ СИСТЕМЫ 8](#_Toc76375470)

[Заключение 25](#_Toc76375471)

# Введение

Точный результат арифметических действий не всегда можно получить из-за ограниченного числа разрядов. Также, мы ограничены размером чисел, с которыми можем работать. Если же нам необходимо выполнить арифметические действия над очень большими числами, то в таких случаях мы сами должны позаботиться о представлении чисел в вычислительной машине и о точном выполнении арифметических операций над ними.

«Длинными» называются числа, для представления которых в базовых типах данных не хватает количества двоичных разрядов. «Длинная» арифметика – это реализация арифметических операций над «длинными» числами.

Для большинства программ, предоставляемых процессором, вполне хватает базовых типов. Тем не менее, встречаются задачи, для которых не хватает значений, предусмотренных базовыми типами. Число из 10000 цифр не поместится ни в один регистр памяти. Поэтому компьютерное представление таких чисел и операции над ними приходится реализовывать самостоятельно. Создавая класс, организующий считывание, вывод, да и сами алгоритмы работы с длинными числами, можно добиться как ускорения времени, затрачиваемого над операциями, так и в разы сократить используемую память.

Возникает вопрос в актуальности использования длинной арифметики. Длинные числа повсеместно используются в бухгалтерии для точного подсчета денежных и других средств. Очень активную роль в последнее время большие числа стали играть в криптографии.

# 1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И ПОСТАНОВКА ЗАДАЧИ

1.1 Постановка задачи

Итоговым продуктом должен бвть класс, который может работать с десятичными числами произвольной длинны. Реализуемые операции:

* сложение;
* вычитание;
* умножение;
* деление;
* остаток от деления;
* префиксный декремент;
* постфиксный декремент;
* префиксный инкремент;
* постфиксный инкремент;
* сравнение (<, <=, >, >=, ==, !=);
* операции ввода/вывода( <<, >>).

Также необходимо полностью протестировать весь функционал класса. Для тестирования можно использовать google тесты.

1.2 Распределение задач для совместной разработки

Разработка тестов для проверки функционала класса. Обеспечить полное прокрытие кода тестами, для исключения всех возможных неточностей в работе класса.

1.3 Анализ предметной области

Работа с «длинными» числами может активно использоваться в криптографии. Без чисел произвольного размера максимально можно использовать число с 20 знаками, что ограничивает размер ключей. Для улучшения безопасности данных можно использовать «длинные» числа, которые позволяют создавать ключи потенциально неограниченной длины, одноко чем больше длина ключа, тем дольше будет генерация ключа.

# 2 ОБОСНОВАНИЕ ВЫБОРА ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ

Для разработки класса был выбран язык С++. Он обладает множеством преимуществ, которые выгодно выделяют его на фоне остальных языков программирования.

Во-первых, он очень быстрый. Программа, которая грамотно написана на С++ в подавляющем большинстве случаев окажется быстрее программы с тем же функционалом, написанной на другом языке программирования.

Во-вторых, он обладает завидной универсальностью. Поскольку компилятор С++ есть на подавляющем большинстве операционных системы, появляется возможность написать одну программу, которая сможет скомпилироваться на любой платформе.

В-третьих, до сих пор идет активная поддержка и модификация С++. Разрабатывают и выпускают новые стандарты, что позволяет языку не отставать от своих конкурентов, и даже опережать их как в функциональности, так и в удобстве написания кода.

В-четвертых, он до сих пор остается одним из самых востребованных языков программирования, потому что имеет широкие возможности, позволяющие оптимизировать программируемые системы до такого состояния, до которого невозможно добраться, используя более высокоуровневые языки программирования.

Также были использованы google тесты, которые позволяют сравнить результат выполнения функции с ожидаемым результатом. Это является очень полезной функцией, которая позволяет сразу выявлять ошибки и оперативно с ними справляться.

# 3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОЙ СИСТЕМЫ

При проектировании программы были определены функции, для которых и были написаны тесты. При написании тестов был сделан упор на как можно большее покрытие кода, что позволит сразу отлавливать возникающие ошибки.

# 4 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ СИСТЕМЫ

В результате разработки были реализованы все функции, которые требовались от класса (п. 1.1). Для каждой реализованной функции были написаны google тесты. Код тестирования:

#include "../../BigInteger.cpp"

#include <gtest/gtest.h>

#include <sstream>

TEST(Summ, summ\_positive\_and\_negative) {

ASSERT\_EQ(BigInteger(0) + BigInteger(0), BigInteger(0));

ASSERT\_EQ(BigInteger(0) + BigInteger(12), BigInteger(12));

ASSERT\_EQ(BigInteger(0) + BigInteger(-12), BigInteger(-12));

ASSERT\_EQ(BigInteger(35) + BigInteger(127), BigInteger(35 + 127));

ASSERT\_EQ(BigInteger(127) + BigInteger(35), BigInteger(35 + 127));

ASSERT\_EQ(BigInteger(-35) + BigInteger(127), BigInteger(-35 + 127));

ASSERT\_EQ(BigInteger(35) + BigInteger(-127), BigInteger(35 + (-127)));

ASSERT\_EQ(BigInteger(-127) + BigInteger(35), BigInteger(35 + (-127)));

ASSERT\_EQ(BigInteger(127) + BigInteger(-35), BigInteger(-35 + 127));

ASSERT\_EQ(BigInteger(-127) + BigInteger(-35), BigInteger(-35 + (-127)));

ASSERT\_EQ(BigInteger(-35) + BigInteger(-127), BigInteger(-35 + (-127)));

ASSERT\_EQ(BigInteger(-35) + BigInteger(35), BigInteger(-35 + 35));

ASSERT\_EQ(BigInteger(35) + BigInteger(-35), BigInteger(-35 + 35));

ASSERT\_EQ(BigInteger(-35) + BigInteger(-35), BigInteger(-35 + (-35)));

ASSERT\_EQ(BigInteger(999) + BigInteger(99), BigInteger(999 + 99));

ASSERT\_EQ(BigInteger(99) + BigInteger(99), BigInteger(99 + 99));

ASSERT\_EQ(BigInteger(9) + BigInteger(999), BigInteger(9 + 999));

ASSERT\_EQ(BigInteger(9) + BigInteger(9), BigInteger(9 + 9));

ASSERT\_EQ(BigInteger(10'00000) + BigInteger(99'99999), BigInteger(10'00000 + 99'99999));

BigInteger x = 0;

ASSERT\_EQ(x += x, BigInteger(0));

x = 1;

ASSERT\_EQ(x += x, BigInteger(2));

x = 12;

ASSERT\_EQ(x += x, BigInteger(24));

x = -37;

ASSERT\_EQ(x += x, BigInteger(-74));

x = 99;

ASSERT\_EQ(x += x, BigInteger(198));

x = 9;

ASSERT\_EQ(x += x, BigInteger(18));

x = 123'456'789;

ASSERT\_EQ(x += x, BigInteger(123'456'789) + BigInteger(123'456'789));

}

TEST(difference, difference\_positive\_and\_negative) {

ASSERT\_EQ(BigInteger(0) - BigInteger(0), BigInteger(0));

ASSERT\_EQ(BigInteger(0) - BigInteger(123), BigInteger(-123));

ASSERT\_EQ(BigInteger(0) - BigInteger(-123), BigInteger(123));

ASSERT\_EQ(BigInteger(35) - BigInteger(127), BigInteger(35 - 127));

ASSERT\_EQ(BigInteger(127) - BigInteger(35), BigInteger(127 - 35));

ASSERT\_EQ(BigInteger(-35) - BigInteger(127), BigInteger(-35 - 127));

ASSERT\_EQ(BigInteger(35) - BigInteger(-127), BigInteger(35 - (-127)));

ASSERT\_EQ(BigInteger(-127) - BigInteger(35), BigInteger(-127 - 35));

ASSERT\_EQ(BigInteger(127) - BigInteger(-35), BigInteger(127 - (-35)));

ASSERT\_EQ(BigInteger(-127) - BigInteger(-35), BigInteger((-127) - (-35)));

ASSERT\_EQ(BigInteger(-35) - BigInteger(-127), BigInteger(-35 - (-127)));

ASSERT\_EQ(BigInteger(-35) - BigInteger(35), BigInteger(-35 - 35));

ASSERT\_EQ(BigInteger(35) - BigInteger(-35), BigInteger(35 - (-35)));

ASSERT\_EQ(BigInteger(-35) - BigInteger(-35), BigInteger(-35 - (-35)));

ASSERT\_EQ(BigInteger(35) - BigInteger(35), BigInteger(35 - 35));

ASSERT\_EQ(BigInteger(10'000) - BigInteger(999), BigInteger(10'000 - 999));

ASSERT\_EQ(BigInteger(10'000) - BigInteger(99'999), BigInteger(10'000 - 99'999));

ASSERT\_EQ(BigInteger(222) - BigInteger(333), BigInteger(222 - 333));

ASSERT\_EQ(BigInteger(123'876) - BigInteger(99'999), BigInteger(123'876 - 99'999));

}

TEST(less, different\_size\_or\_sign\_TRUE) {

BigInteger x = 1;

BigInteger y = 10;

ASSERT\_TRUE(x < y);

x = -1;

y = 10;

ASSERT\_TRUE(x < y);

x = 0;

y = 10;

ASSERT\_TRUE(x < y);

x = -10;

y = -1;

ASSERT\_TRUE(x < y);

x = 1;

y = 10;

ASSERT\_TRUE(x < y);

}

TEST(less, different\_size\_or\_sign\_FALSE) {

BigInteger y = 1;

BigInteger x = 10;

ASSERT\_FALSE(x < y);

y = -1;

x = 10;

ASSERT\_FALSE(x < y);

y = 0;

x = 10;

ASSERT\_FALSE(x < y);

y = -10;

x = -1;

ASSERT\_FALSE(x < y);

y = 1;

x = 10;

ASSERT\_FALSE(x < y);

}

TEST(less, equals\_size\_TRUE) {

BigInteger x = 0;

BigInteger y = 1;

ASSERT\_TRUE(x < y);

x = 1;

y = 2;

ASSERT\_TRUE(x < y);

x = -1;

y = 2;

ASSERT\_TRUE(x < y);

x = -10;

y = 20;

ASSERT\_TRUE(x < y);

x = -1;

y = 0;

ASSERT\_TRUE(x < y);

x = 100;

y = 101;

ASSERT\_TRUE(x < y);

x = 102;

y = 301;

ASSERT\_TRUE(x < y);

x = -1000;

y = 2222;

ASSERT\_TRUE(x < y);

x = -100;

y = 123;

ASSERT\_TRUE(x < y);

x = -101;

y = -100;

ASSERT\_TRUE(x < y);

x = 101;

y = 999;

ASSERT\_TRUE(x < y);

}

TEST(less, equals\_size\_FALSE) {

BigInteger y = 0;

BigInteger x = 0;

ASSERT\_FALSE(x < y);

y = 1;

x = 2;

ASSERT\_FALSE(x < y);

y = -1;

x = 2;

ASSERT\_FALSE(x < y);

y = -10;

x = 20;

ASSERT\_FALSE(x < y);

y = -1;

x = 0;

ASSERT\_FALSE(x < y);

y = 100;

x = 101;

ASSERT\_FALSE(x < y);

y = -1000;

x = 2222;

ASSERT\_FALSE(x < y);

y = -100;

x = 123;

ASSERT\_FALSE(x < y);

y = -101;

x = -100;

ASSERT\_FALSE(x < y);

y = 101;

x = 999;

ASSERT\_FALSE(x < y);

}

TEST(great, TRUE) {

BigInteger x = 1;

BigInteger y = 2;

ASSERT\_GT(y, x);

x = 100;

y = 200;

ASSERT\_TRUE(y > x);

x = -100;

y = 0;

ASSERT\_TRUE(y > x);

x = -100;

y = 1110;

ASSERT\_TRUE(y > x);

x = -100;

y = 110;

ASSERT\_TRUE(y > x);

}

TEST(great, FALSE) {

BigInteger y = 1;

BigInteger x = 2;

ASSERT\_GT(x, y);

y = 100;

x = 200;

ASSERT\_FALSE(y > x);

y = -100;

x = 0;

ASSERT\_FALSE(y > x);

y = -100;

x = 1110;

ASSERT\_FALSE(y > x);

y = -100;

x = 110;

ASSERT\_FALSE(y > x);

}

TEST(equals, TRUE) {

BigInteger x = 0;

BigInteger y = 0;

ASSERT\_EQ(x, y);

x = 1;

y = 1;

ASSERT\_TRUE(x == y);

x = -1;

y = -1;

ASSERT\_TRUE(x == y);

x = 1;

ASSERT\_TRUE(1 == x);

ASSERT\_TRUE(x == 1);

}

TEST(equals, FALSE) {

BigInteger x = 0;

BigInteger y = 1;

ASSERT\_NE(x, y);

x = -1;

y = 1;

ASSERT\_FALSE(x == y);

x = 0;

y = -1;

ASSERT\_FALSE(x == y);

x = -1;

ASSERT\_FALSE(1 == x);

ASSERT\_FALSE(x == 1);

}

TEST(not\_equals, TRUE) {

BigInteger x = -1;

BigInteger y = 1;

ASSERT\_NE(x, y);

x = -1;

y = 5;

ASSERT\_TRUE(x != y);

x = 0;

y = -2;

ASSERT\_TRUE(x != y);

x = -3;

ASSERT\_TRUE(1 != x);

ASSERT\_TRUE(x != 1);

}

TEST(not\_equals, FALSE) {

BigInteger x = 0;

BigInteger y = 0;

ASSERT\_EQ(x, y);

x = 1;

y = 1;

ASSERT\_FALSE(x != y);

x = -1;

y = -1;

ASSERT\_FALSE(x != y);

x = 1;

ASSERT\_FALSE(1 != x);

ASSERT\_FALSE(x != 1);

}

TEST(great\_or\_equals, TRUE) {

BigInteger x = 1;

BigInteger y = 1;

ASSERT\_TRUE(x >= y);

x = 1;

y = -1;

ASSERT\_TRUE(x >= y);

x = 0;

y = 0;

ASSERT\_TRUE(x >= y);

x = 2;

y = 1;

ASSERT\_TRUE(x >= y);

x = -1;

y = -2;

ASSERT\_TRUE(x >= y);

x = -1;

y = -1;

ASSERT\_TRUE(x >= y);

x = 1;

y = 1;

ASSERT\_TRUE(x >= y);

}

TEST(great\_or\_equals, FALSE) {

BigInteger x = 0;

BigInteger y = 1;

ASSERT\_FALSE(x >= y);

x = 1;

y = 2;

ASSERT\_FALSE(x >= y);

x = -1;

y = 2;

ASSERT\_FALSE(x >= y);

x = -10;

y = 20;

ASSERT\_FALSE(x >= y);

x = -1;

y = 0;

ASSERT\_FALSE(x >= y);

x = 100;

y = 101;

ASSERT\_FALSE(x >= y);

x = -1000;

y = 2222;

ASSERT\_FALSE(x >= y);

x = -100;

y = 123;

ASSERT\_FALSE(x >= y);

x = -101;

y = -100;

ASSERT\_FALSE(x >= y);

x = 101;

y = 999;

ASSERT\_FALSE(x >= y);

}

TEST(less\_equals, TRUE) {

BigInteger x = 0;

BigInteger y = 0;

ASSERT\_LE(x, y);

x = 1;

y = 1;

ASSERT\_TRUE(x <= y);

x = -1;

y = -1;

ASSERT\_TRUE(x <= y);

x = 1;

ASSERT\_TRUE(1 <= x);

ASSERT\_TRUE(x <= 1);

x = -1;

y = 10;

ASSERT\_TRUE(x <= y);

x = 0;

y = 10;

ASSERT\_TRUE(x <= y);

x = -10;

y = -1;

ASSERT\_TRUE(x <= y);

x = 1;

y = 10;

ASSERT\_TRUE(x <= y);

x = 1;

y = 2;

ASSERT\_TRUE(x <= y);

x = -1;

y = 2;

ASSERT\_TRUE(x <= y);

x = -10;

y = 20;

ASSERT\_TRUE(x <= y);

x = -1;

y = 0;

ASSERT\_TRUE(x <= y);

x = 100;

y = 101;

ASSERT\_TRUE(x <= y);

x = -1000;

y = 2222;

ASSERT\_TRUE(x <= y);

x = -100;

y = 123;

ASSERT\_TRUE(x <= y);

x = -101;

y = -100;

ASSERT\_TRUE(x <= y);

x = 101;

y = 999;

ASSERT\_TRUE(x <= y);

}

TEST(less\_equals, FALSE) {

BigInteger x = 1;

BigInteger y = 2;

ASSERT\_FALSE(y <= x);

x = 100;

y = 200;

ASSERT\_FALSE(y <= x);

x = -100;

y = 0;

ASSERT\_FALSE(y <= x);

x = -100;

y = 1110;

ASSERT\_FALSE(y <= x);

x = -100;

y = 110;

ASSERT\_FALSE(y <= x);

}

TEST(operator\_bool, cast\_TRUE) {

BigInteger true\_value = 1;

ASSERT\_TRUE(true\_value);

true\_value = -1;

ASSERT\_TRUE(true\_value);

true\_value = -324;

ASSERT\_TRUE(true\_value);

true\_value = 127;

ASSERT\_TRUE(true\_value);

}

TEST(multiplication, multiply\_positive\_and\_negative) {

ASSERT\_EQ(BigInteger(0) \* BigInteger(27), BigInteger(0 \* 27));

ASSERT\_EQ(BigInteger(27) \* BigInteger(0), BigInteger(27 \* 0));

ASSERT\_EQ(BigInteger(7) \* BigInteger(9), BigInteger(7 \* 9));

ASSERT\_EQ(BigInteger(6) \* BigInteger(6), BigInteger(6 \* 6));

ASSERT\_EQ(BigInteger(0) \* BigInteger(0), BigInteger(0 \* 0));

ASSERT\_EQ(BigInteger(27) \* BigInteger(135), BigInteger(27 \* 135));

ASSERT\_EQ(BigInteger(999) \* BigInteger(9), BigInteger(999 \* 9));

ASSERT\_EQ(BigInteger(99) \* BigInteger(999), BigInteger(99 \* 999));

ASSERT\_EQ(BigInteger(1350) \* BigInteger(2700), BigInteger(1350 \* 2700));

ASSERT\_EQ(BigInteger(1300) \* BigInteger(2700), BigInteger(1300 \* 2700));

ASSERT\_EQ(BigInteger(2700) \* BigInteger(135), BigInteger(2700 \* 135));

ASSERT\_EQ(BigInteger(135) \* BigInteger(2700), BigInteger(2700 \* 135));

ASSERT\_EQ(BigInteger(207) \* BigInteger(1355), BigInteger(207 \* 1355));

ASSERT\_EQ(BigInteger(1095) \* BigInteger(270), BigInteger(1095 \* 270));

ASSERT\_EQ(BigInteger(-135) \* BigInteger(2), BigInteger((-135) \* 2));

ASSERT\_EQ(BigInteger(-135) \* BigInteger(-27), BigInteger(135 \* 27));

ASSERT\_EQ(BigInteger(200700) \* BigInteger(10305), BigInteger(200700 \* 10305));

ASSERT\_EQ(BigInteger(-200700) \* BigInteger(-10305), BigInteger(-200700 \* (-10305)));

ASSERT\_EQ(BigInteger(20000) \* BigInteger(130), BigInteger(20000 \* 130));

ASSERT\_EQ(BigInteger(1000) \* BigInteger(99990), BigInteger(1000 \* 99990));

}

TEST(division, division\_positive\_and\_negative){

ASSERT\_EQ(BigInteger(0) / BigInteger(27), BigInteger(0 / 27));

ASSERT\_EQ(BigInteger(27) / BigInteger(135), BigInteger( 27 / 135));

ASSERT\_EQ(BigInteger(1) / BigInteger(135), BigInteger(1 / 135));

ASSERT\_EQ(BigInteger(99) / BigInteger(900), BigInteger(99 / 900));

ASSERT\_EQ(BigInteger(13500) / BigInteger(27000), BigInteger(13500 / 27000));

ASSERT\_EQ(BigInteger(135) / BigInteger(2700), BigInteger(135 / 2700));

ASSERT\_EQ(BigInteger(2700) / BigInteger(2700), BigInteger(2700 / 2700));

ASSERT\_EQ(BigInteger(-999) / BigInteger(999), BigInteger(-999 / 999));

ASSERT\_EQ(BigInteger(-27) / BigInteger(-27), BigInteger(27 / 27));

ASSERT\_EQ(BigInteger(27) / BigInteger(1), BigInteger(27));

ASSERT\_EQ(BigInteger(999) / BigInteger(9), BigInteger(999 / 9));

ASSERT\_EQ(BigInteger(123456) / BigInteger(111), BigInteger(123456 / 111));

ASSERT\_EQ(BigInteger(2700) / BigInteger(135), BigInteger(2700 / 135));

ASSERT\_EQ(BigInteger(207) / BigInteger(13), BigInteger(207 / 13));

ASSERT\_EQ(BigInteger(2700) / BigInteger(1350), BigInteger( 2700 / 1350));

ASSERT\_EQ(BigInteger(27000) / BigInteger(300), BigInteger(27000 / 300));

ASSERT\_EQ(BigInteger(1095) / BigInteger(1000), BigInteger(1095 / 1000));

ASSERT\_EQ(BigInteger(3672) / BigInteger(36), BigInteger(3672 / 36));

ASSERT\_EQ(BigInteger(24060) / BigInteger(12), BigInteger(24060 / 12));

ASSERT\_EQ(BigInteger(-135) / BigInteger(2), BigInteger((-135) / 2));

ASSERT\_EQ(BigInteger(99) / BigInteger(-9), BigInteger( -99 / 9));

ASSERT\_EQ(BigInteger(-135) / BigInteger(-27), BigInteger(135 / 27));

ASSERT\_EQ(BigInteger(-13) / BigInteger(-27), BigInteger(13 / 27));

ASSERT\_EQ(BigInteger(333) / BigInteger(331), BigInteger(333 / 331));

ASSERT\_EQ(BigInteger(255) / BigInteger(27), BigInteger(255 / 27));

ASSERT\_EQ(BigInteger(111) / BigInteger(50), BigInteger(111 / 50));

ASSERT\_EQ(BigInteger(-13) / BigInteger(-2), BigInteger(-13 / (-2)));

ASSERT\_EQ(BigInteger(999'999'999) / BigInteger(111'111'112), BigInteger(999'999'999 / 111'111'112));

ASSERT\_EQ(BigInteger(381) / BigInteger(75), BigInteger(381 / 75));

ASSERT\_EQ(BigInteger(-93) / BigInteger(-18), BigInteger(-93 / (-18)));

ASSERT\_EQ(BigInteger(23) / BigInteger(63), BigInteger(23 / 63));

ASSERT\_EQ(BigInteger(201) / BigInteger(-105), BigInteger(201 / (-105)));

ASSERT\_EQ(BigInteger(225) / BigInteger(25), BigInteger(225 / 25));

}

TEST(division, division\_by\_zero) {

ASSERT\_THROW(BigInteger(5) / BigInteger(0), std::runtime\_error);

ASSERT\_THROW(BigInteger(-5) / BigInteger(0), std::runtime\_error);

ASSERT\_THROW(BigInteger(0) / BigInteger(0), std::runtime\_error);

}

TEST(operator\_bool, cast\_FALSE) {

BigInteger false\_value = 0;

ASSERT\_FALSE(false\_value);

}

TEST(division\_on\_module, positive\_and\_negative\_numbers) {

for (short int i = 1; i < 25; ++i) {

ASSERT\_EQ(BigInteger(i \* i) % BigInteger(25 - i), BigInteger((i \* i) % (25 - i)));

}

for (long i = 100; i > 34; --i) {

ASSERT\_EQ(BigInteger(i \* i) % BigInteger(34 - i), BigInteger((i \* i) % (34 - i)));

}

}

TEST(division\_on\_module, value\_is\_zero) {

ASSERT\_THROW(BigInteger(5) % BigInteger(0), std::runtime\_error);

ASSERT\_THROW(BigInteger(-5) % BigInteger(0), std::runtime\_error);

ASSERT\_THROW(BigInteger(0) % BigInteger(0), std::runtime\_error);

}

TEST(negative\_number, positive\_and\_negative\_number) {

ASSERT\_EQ(-BigInteger(1), BigInteger(-1));

ASSERT\_EQ(-BigInteger(-1), BigInteger(1));

ASSERT\_EQ(-BigInteger(0), BigInteger(0));

}

TEST(enter\_from\_istream, random\_value) {

std::stringstream stream;

BigInteger value;

std::string string\_BigInteger;

for (size\_t i = 0; i < 100; ++i) {

string\_BigInteger.clear();

if (i % 2 == 0) {

string\_BigInteger += '-';

}

string\_BigInteger += std::to\_string(rand());

stream.clear();

stream << string\_BigInteger;

stream >> value;

ASSERT\_EQ(value.toString(), string\_BigInteger);

}

stream.ignore(32645, '\n');

string\_BigInteger = "1\n";

stream.clear();

stream << string\_BigInteger;

stream >> value;

ASSERT\_EQ(value.toString(), "1");

stream.ignore(32645, '\n');

string\_BigInteger = "123asd";

stream.clear();

stream << string\_BigInteger;

stream >> value;

ASSERT\_EQ(value.toString(), "123");

stream.ignore(32645, '\n');

string\_BigInteger = "-123asd";

stream.clear();

stream << string\_BigInteger;

stream >> value;

ASSERT\_EQ(value.toString(), "-123");

stream.ignore(32645, '\n');

string\_BigInteger = "-0000123asd";

stream.clear();

stream << string\_BigInteger;

stream >> value;

ASSERT\_EQ(value.toString(), "-123");

stream.ignore(32645, '\n');

string\_BigInteger = "-0000123asd";

stream.clear();

stream << string\_BigInteger;

stream >> value;

ASSERT\_EQ(value.toString(), "-123");

stream.ignore(32645, '\n');

string\_BigInteger = "-0000asd";

stream.clear();

stream << string\_BigInteger;

stream >> value;

ASSERT\_EQ(value.toString(), "0");

stream.ignore(32645, '\n');

string\_BigInteger = "-1asd";

stream.clear();

stream << string\_BigInteger;

stream >> value;

ASSERT\_EQ(value.toString(), "-1");

stream.ignore(32645, '\n');

string\_BigInteger = "-0000asd";

stream.clear();

stream << string\_BigInteger;

stream >> value;

ASSERT\_EQ(value, 0);

stream.ignore(32645, '\n');

string\_BigInteger = "+0000asd";

stream.clear();

stream << string\_BigInteger;

stream >> value;

ASSERT\_EQ(value, 0);

}

TEST(enter\_from\_istream, random\_value\_error) {

std::stringstream stream;

BigInteger value;

std::string buffer;

stream.clear();

stream << "-\n";

stream >> value;

stream >> buffer;

ASSERT\_EQ(buffer, "");

stream.clear();

stream << "+\n";

stream >> value;

stream >> buffer;

ASSERT\_EQ(buffer, "");

stream.clear();

stream << "-a";

stream >> value;

stream >> buffer;

ASSERT\_EQ(buffer, "");

stream.clear();

stream << "+b\n";

stream >> value;

stream >> buffer;

ASSERT\_EQ(buffer, "");

stream.clear();

stream << "b\n";

stream >> value;

stream >> buffer;

ASSERT\_EQ(buffer, "");

stream.clear();

stream << "abc\n";

stream >> value;

stream >> buffer;

ASSERT\_EQ(buffer, "");

}

TEST(out\_stream, random\_value) {

std::stringstream stream;

BigInteger value1, value2;

value1 = -123;

stream.clear();

stream << value1;

stream >> value2;

ASSERT\_EQ(value1, value2);

value1 = -0;

stream.clear();

stream << value1;

stream >> value2;

ASSERT\_EQ(value1, value2);

value1 = -1;

stream.clear();

stream << value1;

stream >> value2;

ASSERT\_EQ(value1, value2);

value1 = 1;

stream.clear();

stream << value1;

stream >> value2;

ASSERT\_EQ(value1, value2);

value1 = 10'000;

stream.clear();

stream << value1;

stream >> value2;

ASSERT\_EQ(value1, value2);

}

TEST(inc, random\_value) {

int integer = -100;

BigInteger bigInteger = integer;

for (; bigInteger < 100; ++bigInteger, ++integer) {

ASSERT\_EQ(bigInteger, integer);

}

bigInteger = -101;

for (BigInteger old\_value = bigInteger++; old\_value < 100; old\_value = bigInteger++) {

ASSERT\_EQ(old\_value + 1, bigInteger);

}

}

TEST(dic, random\_value) {

int integer = 100;

BigInteger bigInteger = integer;

for (; bigInteger > -100; --bigInteger, --integer) {

ASSERT\_EQ(bigInteger, integer);

}

bigInteger = 101;

for (BigInteger old\_value = bigInteger--; old\_value > -100; old\_value = bigInteger--) {

ASSERT\_EQ(old\_value - 1, bigInteger);

}

}

int main(int argc, char\*\* argv) {

testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}

# Заключение

Результат работы – класс, который позволяет работать с числами произволдьной длины реализованный при помощи языка программирования C++ и протестированная с помощью google тестов.

К преимуществам класса относятся: быстрая работа, удобство при использовании класса за счет перегрузки базовых операторов, оттестированный функционал, что обеспечивает правильность работы всех реализованных функций.

При дальнейшей работе над классом можно реализовать более сложные математические операции, а также добавить возможность работы нецелыми числами.