МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

**ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ**

**«Множества на основе битовых полей»**

**Выполнил:** студент группы 381706-1

Суслов Егор Игоревич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

ассистент каф. МОСТ ИИТММ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лебедев И.Г

Нижний Новгород

2019.

**Содержание**

[**1.** **Введение** 3](#_Toc510437279)

[**2.** **Постановка задачи** 4](#_Toc510437280)

[**3.** **Руководство пользователя** 5](#_Toc510437281)

[**4.** **Руководство программиста** 6](#_Toc510437282)

[**4.1.** **Описание структуры программы** 6](#_Toc510437283)

[**4.2.** **Описание структур данных** 6](#_Toc510437284)

[**5.** **Заключение** 8](#_Toc510437285)

[**6.** **Литература** 9](#_Toc510437286)

# **Введение**

Теория множеств – учение об общих свойствах множеств – преимущественно бесконечных. Явным образом понятие множества подверглось систематическому изучению во второй половине XIX века в работах немецкого математика Георга Кантора .

Влияние теории множеств на развитие современной математики очень велико. Прежде всего, теория множеств явилась фундаментом ряда новых математических дисциплин (теории функций действительного переменного, общей топологии, общей алгебры, функционального анализа и др.).

Постепенно теоретико-множественные методы находят всё большее применение и в классических частях математики. Например, в области математического анализа они широко применяются в качественной теории дифференциальных уравнений, вариационном исчислении, теории вероятностей и др.

Активное применение аппарата теории множеств в современной науке приводит к необходимости создания соответствующих программных решений. Вместе с тем лишь в отдельных языках программирования предусмотрены встроенные средства для работы с множествами (примером может служить язык Pascal в реализации фирмы Borland).

Программная реализация множества может выполняться различными способами (в соответствии с требованиями конкретной задачи или с общих позиций) и обычно тесно связана с использованием битовых операций в выбранном языке программирования.

Все операции над множествами могут быть заменены в таком случае на операции над характеристическими векторами. Таким образом, в дальнейшем в работе мы будем решать задачу хранения и обработки именно характеристических векторов.

Поскольку каждый элемент характеристического вектора принимает значения из множества {0, 1}, наиболее эффективной (с точки зрения расхода памяти) является его реализация через битовое поле – непрерывный участок памяти (количество бит в котором достаточно для представления Универса. Реализацию битового поля целесообразно вынести в отдельный класс, скрывающий детали, не существенные для представления и работы с множествами.

Проверка будет осуществляться с помощью «Решето Эратосфена» и Google Tests.

**Решето́ Эратосфе́на** — [алгоритм](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) нахождения всех [простых чисел](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%81%D1%82%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) до некоторого целого числа *n*, который приписывают древнегреческому математику [Эратосфену Киренскому](https://ru.wikipedia.org/wiki/%D0%AD%D1%80%D0%B0%D1%82%D0%BE%D1%81%D1%84%D0%B5%D0%BD). Как и во многих случаях, здесь название алгоритма говорит о принципе его работы, то есть решето подразумевает [фильтрацию](https://ru.wikipedia.org/wiki/%D0%A4%D0%B8%D0%BB%D1%8C%D1%82%D1%80_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), в данном случае фильтрацию всех чисел за исключением простых. По мере прохождения списка нужные числа остаются, а ненужные (они называются [составными](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D0%BD%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE)) исключаются.

Название «решето» метод получил потому, что, согласно легенде, Эратосфен писал числа на дощечке, покрытой воском, и прокалывал дырочки в тех местах, где были написаны [составные числа](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D0%BD%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE). Поэтому дощечка являлась неким подобием решета, через которое «просеивались» все составные числа, а оставались только числа простые. Эратосфен дал таблицу простых чисел до 1000.

# **Постановка задачи**

Реализовать класс для работы с множествами TSet и вспомогательный класс для работы с битовыми полями TBitField

В классе TBitField реализовать следующие операции:

• установить бит (в единицу);

• очистить бит (в ноль);

• получить значение бита;

• сравнить два битовых поля;

• выполнить операцию “логическое или” для двух битовых полей;

• выполнить операцию “логическое и” для двух битовых полей;

• выполнить операцию “логическое отрицание” для битового поля.

В классе TSet реализовать следующие операции:

• включение элемента в множество;

• исключение элемента из множества;

• проверка наличия элемента в множестве;

• сравнение множеств

• сложение множеств;

• пересечение множеств;

• разность множеств;

• копирование множества;

• вычисление максимальной мощности множества.

Все работы должны «заливаться» на GitHub (крупнейший [веб-сервис](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81) для [хостинга](https://ru.wikipedia.org/wiki/%D0%A5%D0%BE%D1%81%D1%82%D0%B8%D0%BD%D0%B3) [IT-проектов](https://ru.wikipedia.org/wiki/IT)и их совместной разработки), проект должен собираться с помощью CMake (кроссплатформенная система автоматизации сборки программного обеспечения из [исходного кода](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4)) , а проверятся с помощью Google Tests (библиотека для модульного тестирования) .

# **Руководство пользователя**

1. После запуска программы пользователю предлагается ввести верхнюю границу целых значений
2. Затем печатаются все простые числа которые не превосходят введенное число
3. Во втором выполняются 46 Google Tests (25 для «битового поля» и 21 для «множества») и выводятся их результаты
4. Выход

# **Руководство программиста**

## **Описание структуры программы**

Проект состоит из следующих модулей:

* **sample\_prime\_numbers**

Модуль cодержит файл код «sample\_prime\_numbers.cpp», в котором реализован алгоритм «Решето Эратосфена» для нахождения простых чисел с помощью структуры «битовое поле» или «множество» (по умолчанию на «битовое поле» , а при раскомментировании 10-ой строки кода: // #define USE\_SET из файла «sample\_prime\_numbers.cpp» на «множество»).

* **set**

Модуль содержит статичесские библиотеки «tbitfield.h» и «tset.h», описывающие структуры битовое поле и множество соответственно, а также файлы кода реализующие заданные в библиотеках методы «tbitfield.cpp» и «tset.cpp» соответственно, в которых реализованы данные структуры.

* **test\_set**

Модуль содержит файлы кода «test\_main.cpp», «test\_tbitfield.cpp» и «test\_set.cpp». Первый содержит main и запускает тесты написанные для множества и битового поля соответственно.

* 1. **Описание структур данных**

#### Структура TBitField

Структура TBitField реализована в виде класса TBitField. Поля со спецификатором доступа «private»:

* **int bitLen//***длина битового поля - макс. к-во битов;*
* **uInt \*pMem//** *память для представления битового поля;*
* **int memLen**// *к-во эл-тов Мем для представления битового поля;*
* **int getMemIndex(const int n) const//** *индекс в pМем для бита n;*
* **uInt get (const int n) const//** *битовая маска для бита n;*

Поля со спецификатором доступа «public»:

* **TBitField(int len)//** *конструктор;*
* **TBitField(const TBitField &bf)//** *конструктор;*
* **~TBitField()//** *деструктор;*
* **int GetLength(void) const//** *получить длину (к-во битов);*
* **void SetBit(const int n)//** *установить бит;*
* **void ClrBit(const int n)//** *очистить бит;*
* **int GetBit(const int n) const//** *получить значение бита;*
* **int operator==(const TBitField &bf) const//** *сравнение;*
* **int operator!=(const TBitField &bf) const//** *сравнение;*
* **TBitField& operator=(const TBitField &bf)//** *присваивание;*
* **TBitField operator|(const TBitField &bf)//** *операция "или";*
* **TBitField operator&(const TBitField &bf)//** *операция "и"* ;
* **TBitField operator~(void)//** *отрицание*;
* **friend istream &operator>>(istream &istr, TBitField &bf)//** *оператор ввода;*
* **friend ostream &operator<<(ostream &ostr, const TBitField &bf)//** оператор вывода

#### Структура TSet

Структура TSet реализована в виде класса TSet. Поля со спецификатором доступа «private»:

* **int maxPower//***максимальная мощность множества;*
* ***TBitField bitField//*** *битовое поле для хранения характеристического вектора;*

Поля со спецификатором доступа «public»:

* **TSet(int mp)//***конструктор;*
* **TSet(const TSet &s)//** *конструктор копирования;*
* **TSet(const TBitField &bf)//** *конструктор преобразования типа;*
* **operator TBitField()//** *преобразование типа к битовому полю;*
* **int GetMaxPower(void) const//** *максимальная мощность множества;*
* **void InsElem(const int Elem)//** *включить элемент в множество;*
* **void DelElem(const int Elem)//** *удалить элемент из множества;*
* **int IsMember(const int Elem) const//** *проверить наличие элемента в множестве;*
* **int operator== (const TSet &s) const**// *сравнение;*
* **int operator!= (const TSet &s) const//** *сравнение;*
* **TSet& operator=(const TSet &s)//** *присваивание;*
* **TSet operator+ (const int Elem)//** *объединение с элементом;*
* **TSet operator- (const int Elem)//** *разность с элементом;*
* **TSet operator+ (const TSet &s)//** *объединение;*
* **TSet operator\* (const TSet &s)//** *пересечение;*
* **TSet operator~ (void)//** *дополнение;*
* **friend istream &operator>>(istream &istr, TSet &bf)//** *оператор ввода;*
* **friend ostream &operator<<(ostream &ostr, const TSet &bf)//** *оператор вывода;*

# **Заключение**

В ходе работы мне удалось изучить:

1. Структуру «битовое поле»
2. Структуру «множество»
3. Использование структуры «битовое поле» для построения структуры «множество»
4. Использование и реализацию Google Tests
5. Использование Сmake для сборки проекта

Написанные классы «TBitField» и «TSet» удачно выполняют все поставленные перед ними задачи. Это подтверждается «Google Tests», также удачно реализуется «Решето Эратосфена» на обоих классах.