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 но основе битовых полей»**

**Выполнил:** студент группы 381706-1

Корнев Никита Алексеевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

ассистент каф. МОСТ ИИТММ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лебедев И.Г

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc532749026)

[2. Постановка задачи 4](#_Toc532749027)

[3. Руководство пользователя 5](#_Toc532749028)

[4. Руководство программиста 6](#_Toc532749029)

[4.1. Описание структуры программы 6](#_Toc532749030)

[4.2. Описание структур данных 7](#_Toc532749031)

[4.3 Описание алгоритмов 9](#_Toc532749032)

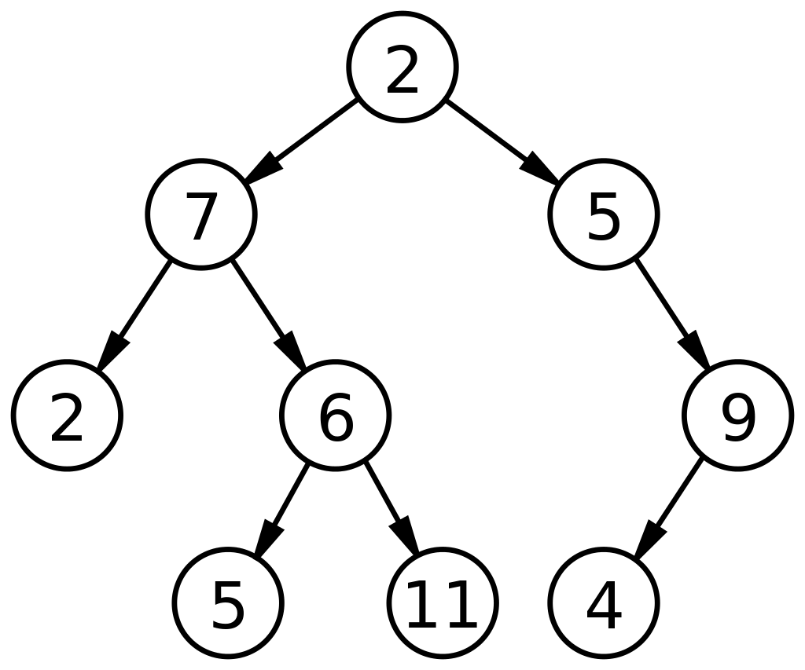
[5. Заключение 10](#_Toc532749033)

[6. Литература 11](#_Toc532749034)

# Введение

Структура данных (англ. data structure) — программная единица, позволяющая хранить и обрабатывать множество однотипных и/или логически связанных данных в вычислительной технике. Для добавления, поиска, изменения и удаления данных структура данных предоставляет некоторый набор функций, составляющих её интерфейс.

Многие классические структуры данных представлены в стандартных библиотеках языков программирования или непосредственно встроены в языки программирования. Например, бинарное дерево, простой пример ветвящейся связной структуры данных.



В данной лабораторной работе нам предстоит самостоятельно реализовать такие структуры данных, как:

* Битовое поле
* Множества

А также освоить базовые навыки работы с фреймворком Google Tests и научиться работать с сервисом контроля версий ПО GitHub.

# Постановка задачи

1. Освоение общих принципов работы с Git и GitHub.
2. Освоение общих принципов работы с Google Tests.
3. Создание полноценных реализаций классов TBitField и TSet, проходящих все автоматические тесты.

# Руководство пользователя

В качестве примера использования структур данных «битовые поля» и «множества» предлагается реализованный в данном проекте алгоритм для поиска простых чисел «решето Эратосфена». После запуска программы необходимо ввести верхний порог чисел, до которого будет осуществляться поиск. После ввода данного числа, программа выведет на экран все простые числа до верхнего порога. Для нахождения чисел программа использует структуру «множества». Если необходимо использовать структуру «битовые поля», а не «множеста», нужно заккоментировать строчку «#define USE\_SET».

# Руководство программиста

## Описание структуры программы

Проект состоит из следующих модулей:

1. sample\_prime\_numbers

Модуль cодержит файл код «sample\_prime\_numbers.cpp», в котором реализован алгоритм «Решето Эратосфена» для нахождения простых чисел с помощью структур «битовое поле» либо «множество».

1. Set

Модуль содержит файлы заголовков «tbitfield.h» и «tset.h», описывающие структуры «битовое поле» и «множество» соответственно, а также файлы кода «tbitfield.cpp» и «tset.cpp», в которых реализованы данные структуры.

1. test\_set

Модуль содержит файлы кода «test\_main.cpp», «test\_tbitfield.cpp» и «test\_set.cpp». В последних двух реализованы тесты для проверки корректности работы методов данных классов.

## 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 бит.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;*

## Описание алгоритмов

Описание некоторых алгоритмов, занимающих более 1 строки кода.

TBitField

* TBitField(int len)

Получает на вход длину битового поля **len** (в случае, если len < 0, выбрасывает исключение). Затем выделяет память под хранение поля данного размера.

* void setBit(const int n)

Получает на вход номер бита **n** (если номер < 0 или больше размера поля, выбрасывает исключение), который необходимо внести в битовое поле. Определяет ячейку, содержащую этот номер, создает битовую маску, содержащую только бит номер **n**, выполняет операцию побитового сложения с исходным полем.

* void clrBit(const int n)

Аналогично предыдущему алгоритму получает на вход номер бита **n**, проверяет его на корректность, создает битовую маску поля с номером **n**, получает ее отрицание, т.е. бит номер **n** становится 0, а все остальные биты – 1, после чего выполняет побитовое сложение маски с исходным полем.

# Заключение

В данном лабораторной работе мне удалось:

* Успешно реализовать классы для структур «множества» и «битовые поля»
* Освоить работу с Google Tests и написать свои собственные тесты для проверки работы написааных мною классов
* Освоить базовые принципы работы с GitHub

# Литература

Интернет-источники:

1. <https://ru.wikipedia.org/wiki/Структура_данных>
2. <https://ru.wikipedia.org/wiki/Множество>
3. <https://ru.wikipedia.org/wiki/Битовое_поле>

Книги:

1. Гергель В.П. Методические материалы по курсу «Методы программирования 2», Нижний Новгород, 2015.
2. A.O. Грудзинский. Методы программирования, Издательство Нижегородского госуниверситета, 2006.