МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Национальный исследовательский**

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

**(ННГУ)**

**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

Голованова Елена Александровна

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

ассистент каф. МОСТ ИИТММ

Лебедев Илья Геннадьевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2018.

Оглавление

[1. Введение 3](#_Toc1236347)

[2. Постановка задачи 4](#_Toc1236348)

[3. Руководство пользователя 5](#_Toc1236349)

[4. Руководство программиста 6](#_Toc1236350)

[a. Описание структуры программы 6](#_Toc1236351)

[b. Описание структур данных 7](#_Toc1236352)

[c. Описание алгоритмов 10](#_Toc1236353)

[5. Заключение 12](#_Toc1236354)

[6. Список литературы 13](#_Toc1236355)

# Введение

Эффективное использование памяти является важной задачей программиста. Он должен уметь компактно и практично использовать память. И чтобы решить эту задачи были придуманы разные структуры данных.

**Структура данных** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *data structure*) — программная единица, позволяющая хранить и обрабатывать множество однотипных и/или логически связанных [данных](https://ru.wikipedia.org/wiki/%D0%94%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5_(%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%82%D0%B5%D1%85%D0%BD%D0%B8%D0%BA%D0%B0)) в [вычислительной технике](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80). Для добавления, поиска, изменения и удаления данных структура данных предоставляет некоторый набор функций, составляющих её интерфейс.

В этой работе мы рассмотрим структуру данных — битовое поле.

**Би́товое поле** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *bit field*) — некоторое количество [бит](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82), расположенных последовательно в [памяти](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C), значение которых [процессор](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) не способен прочитать из-за особенностей аппаратной реализации.

**Битовые поля могут быть полезны по разным причинам, а именно:**

* **Если память ограничена, то в одном байте можно хранить несколько булевых переменных.**
* **Некоторые устройства передают информацию о состоянии, закодированную в байте в одном или нескольких битах**
* **Для некоторых процедур шифрования требуется доступ к отдельным битам внутри файла.**

**Но есть и недостаток в использовании битовых полей** — низкая скорость доступа.

**Цель данной лабораторной работы** — разработка структуры данных для хранения множеств с использованием битовых полей.

# Постановка задачи

Выполнение работы предполагает решение следующих задач:

1. Разработка интерфейса и реализация класса битового поля TBitField.
2. Разработка интерфейса и реализация класса множества TSet.
3. Обеспечение работоспособности тестов и примера использования.
4. Реализация нескольких простых тестов на базе Google Test.

# Руководство пользователя

Пример использования классов битового поля и множества представлен в качестве решения задачи поиска простых чисел с помощью алгоритма "Решето Эратосфена".

При запуске программы с пользователя запрашивается верхняя граница целых значений , среди которых в дальнейшем будут найдены все простые числа. Затем происходит поиск и подсчет простых чисел После на экран выводится множество некратных чисел(битовая строка простых чисел) и простые числа от 0 до .

Далее работа программы различается для классов битового поля и множества.

* При использовании Битовых полей.

Пользователю предлагается самостоятельно ввести битовую строку. Для этого необходимо задать размер битовой строки . Затем ввести последовательность битов длинной +1 (т.к. нужно учесть нулевой элемент). В конце на экран будет выведена полученная битовая строка и множество чисел, которое в ней отражено. На этом работа программы завершается.

* При использовании класса множества.

Пользователю предлагается самостоятельно ввести неотрицательное множество целых чисел. Для этого необходимо задать наибольший элемент множества . Затем ввести набор чисел от 0 до . Для завершения ввода введите -1 или число более . В конце на экран будет выведено полученное множество и его битовая строка. На этом работа программы завершается.

# 4. Руководство программиста

## Описание структуры программы

Для реализации лабораторной работы создается несколько классов, состоящих из файлов:

1. Класс TBitField:

* tbitfield.h – в этом файле определен интерфейс класса битовое поле.
* tbitfield.cpp – реализация таких действий, как конструкторы, деструктор, перегрузка битовых операторов (такие как сравнение, присваивание и т.д.), операторы ввода/вывода, метод доступа к отдельным битам.

2. Класс TSet:

В этом классе происходит наследование некоторых функций битового поля.

* tset.h – в этом файле определен интерфейс класса множества.
* tset.cpp – реализация таких действий, как конструктор по умолчанию, перегрузка теоретико-множественных операторов, операторы ввода/вывода множеств и прописаны методы доступа к отдельным битам.

.

3. Класс TException:

В этом классе описаны исключения.

4.Тесты для классов TBitField и TSet.

Для каждого класса реализован набор тестов в файлах test\_tbitfield.cpp (и test\_tset.cpp. Каждый тест представлен отдельной функцией. Тесты проверяют работу каждого метода классов TBitField и TSet.

5.Файл примера использования:

Для класса битового поля и множества реализован алгоритм поиска простых чисел, называемый "Решетом Эратосфена" (файл sample\_prime\_numbers.cpp).

Модули tbitfield и tset находятся в общей статической библиотеке set. Библиотека подключается к проекту sample\_prime\_numbers, где в исходном файле приводится пример использования класса битового поля TBitField для поиска простых чисел с использованием алгоритма "Решето Эратосфена":

1. Создается объект класса TBitField заданного размера, применяется алгоритм поиска простых чисел и на экран выводится битовая строка простых чисел и сами битовые числа в указанном диапазоне.
2. Создается объект класса TBitField указанного размера, осуществляется ввод битовой строки. Затем полученная строка выводится на экран и выводится множество чисел, отраженных в этой строке.

Аналогично для класса TSet. Библиотека подключается к проекту sample\_prime\_numbers, где в исходном файле приводится пример использования класса множества TSet для поиска простых чисел с использованием алгоритма "Решето Эратосфена":

1. Создается объект класса TSet заданного размера, применяется алгоритм поиска простых чисел и на экран выводится битовая строка простых чисел и сами битовые числа в указанном диапазоне.
2. Создается объект класса TSet указанного размера, осуществляется ввод множества чисел в указанном диапазоне. Затем полученное множество выводится на экран и выводится битовая строка данного множества.

## Описание структур данных

1. Класс TBitField является шаблонным.
   * 1. В зоне private определены три поля и два метода:

* int BitLen – длина битового поля - макс. количество битов;
* TELEM \*pMem – память для представления битового поля;
* int MemLen – количество элементов Мем для представления бит.поля;
* int GetMemIndex(const int n) const – индекс в pМем для бита n;
* TELEM GetMemMask (const int n) const – битовая маска для бита n;

2. В зоне public:

* TBitField(int len) – конструктор;
* TBitField(const TBitField &bf); – конструктор копирования;
* ~TBitField() – Деструктор
* int GetLength(void) const; – получить длину (количество 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 – принимает ссылку на объект класса TStack, выполняет проверку на равенство;
* int operator!=(const TBitField &bf) const – принимает ссылку на объект класса TStack, выполняет проверку на неравенство;
* 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) – перегруженный оператор ввода;
* friend ostream &operator<<(ostream &ostr, const TBitField &bf) – перегруженный оператор вывода.

1. Класс TSet является шаблонным.
   * + 1. В зоне private определены:

* int MaxPower – максимальная мощность множества;
* TBitField BitField – битовое поле для хранения характеристического вектора;

2. В зоне public:

* TSet(int mp) – конструктор с параметром;
* 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 – принимает ссылку на объект класса TSet, выполняет проверку на равенство;
* int operator!= (const TSet &s) const – принимает ссылку на объект класса TSet, выполняет проверку на неравенство;
* 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;
* friend istream &operator>>(istream &istr, TSet &bf) – перегруженный оператор ввода;
* friend istream &operator>>(istream &istr, TSet &bf) – перегруженный оператор вывода.

## Описание алгоритмов

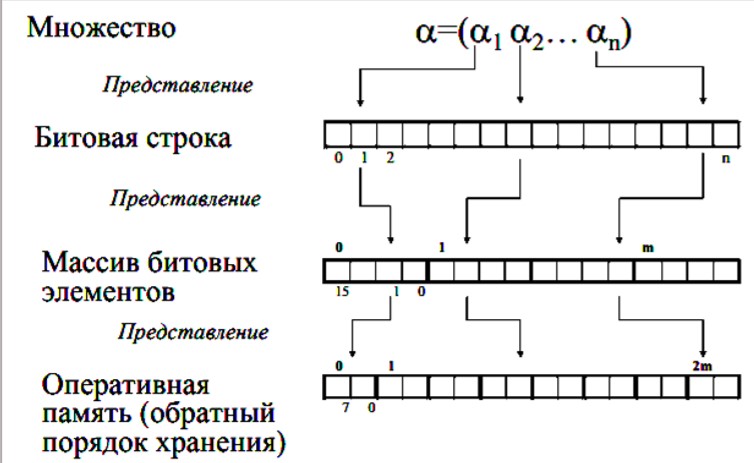
**Класс TBuildField.**

В данном разделе мы рассмотрим некоторые функции и методы класса TBuildField.

* Формирование битового поля:

Нам дано множество натуральных чисел, которое имеет вид битовой строки («1»-элемент есть, и если нет - «0»).

В нашем случае битовая строка хранится в виде массива. Однако, стоит заметить одну особенность – в битовой строке нумерация бит происходит слева направо, в массиве нумерация элементов также происходит справа налево, однако биты элемента массива нумеруются справа налево



*Рис. 1 Представление битового поля*

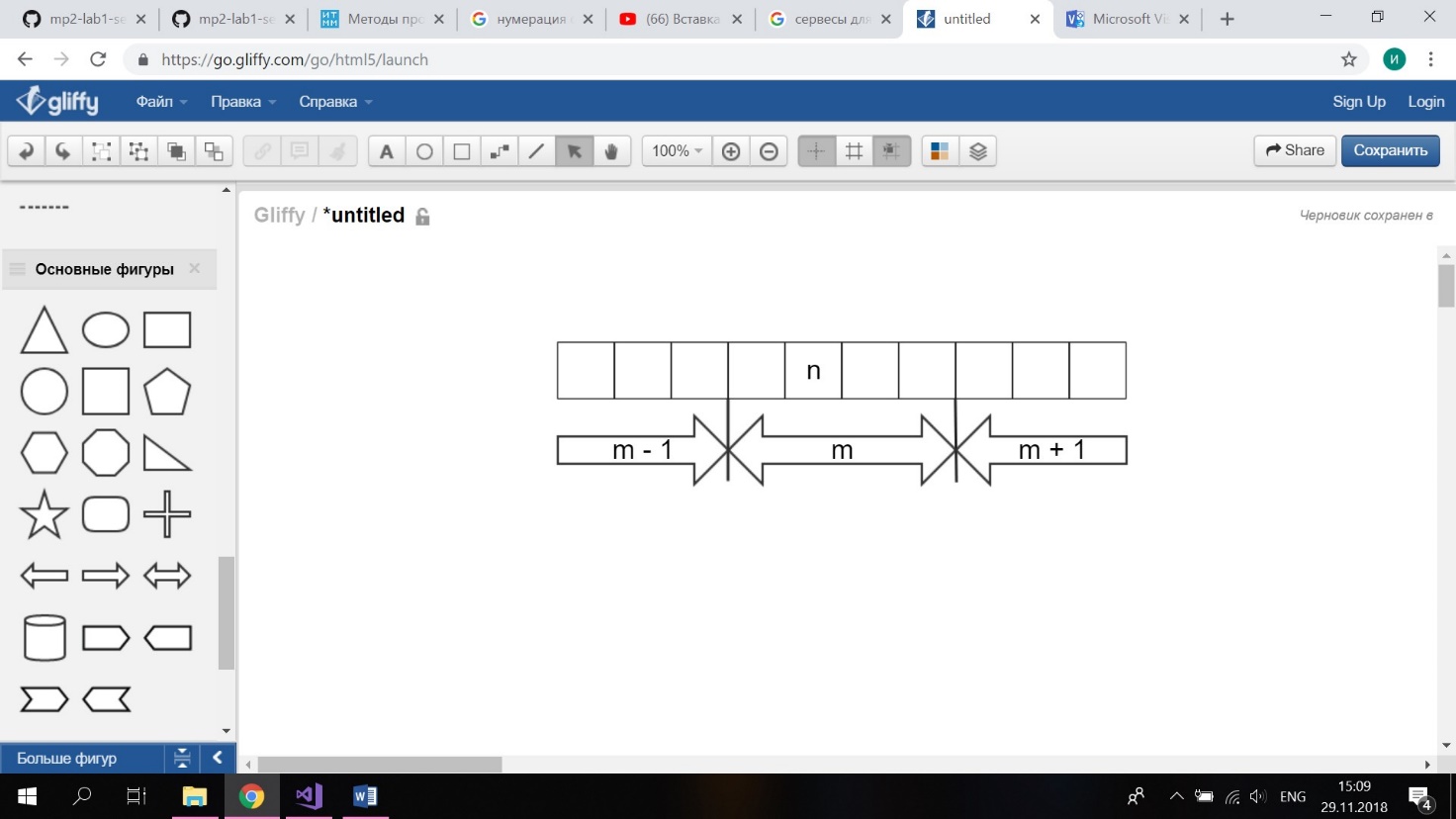
* Методы GetMemIndex и GetMemMask

Для оптимизации основных методов класса TBuildField были прописаны два дополнительных метода:

1. GetMemIndex – возвращает номер m элемента массива pMem, в котором хранится n-й бит (см. рисунок 2).

Индекс элемента m определяется по следующей формуле:

, где TELEM – тип элементов множества.



*Рис.2 Представление массива pMem*

2. GetMemMask – определяет битовую маску (длинной sizeof(int)\*8 – размер элемента массива) для -го бита. Поскольку битовую маску можно рассматривать как двоичное представление числа, то данный метод возвращает целое число. Это число определяется выставлением единичного бита на позиции k, определяемой по следующей формуле:

Далее эти два метода применяются в реализации следующих методов класса TBuildField:

1. SetBit(const int n) – установить бит на позицию n. Чтобы установить бит на позицию n нужно использовать побитовую операцию «или» между битами элемента из массива pMem, в котором хранится бит n, и между его битовой маской.
2. ClrBit(const int n) – очистить n-й бит. Чтобы очистить n-й бит используем побитовую операцию «» между битами элемента массива pMem, в котором находится n-й бит, и между дополнением его битовой маски.
3. GetBit(const int n) – получает значение n-ого бита. Чтобы получить значение n-го бита используем побитовую операцию «И» между битами элемента массива pMem, в котором находится n-й бит, и между его побитовой маской.

# 5. Заключение

В результате выполнения лабораторной работы была разработана структура хранения данных для множеств с использованием битовых полей. В ходе выполнения лабораторной был произведен анализ задачи: установлено понятие множества, операции над элементами множества и теоретико-множественные операции.

Были реализованы класс битового поля (TBitField) и класс множества (TSet), в том числе и методы работы с этими классами.

Были реализованы тесты для проверки работоспособности вышеперечисленных классов на базе GoogleTest.

# 6. Список литературы

1. Википедия. Статья «Битовое поле»: [https://ru.wikipedia.org/wiki/Битовое\_поле].
2. Гергель В.П. Методические материалы по курсу «Методы программирования 2», Нижний Новгород, 2015.