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**

**«Битовые поля на основе множеств»**

**Выполнил:** студент группы 381706-1

Голубева А. С.

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

аспирант каф. МОСТ ИИТММ

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

Нижний Новгород

2018.

Содержание

[**1.Введение** 3](#_Toc532418883)

[**2.Постановка задачи** 4](#_Toc532418884)

[**3.Руководство пользователя** 5](#_Toc532418885)

[**4.Руководство программиста** 6](#_Toc532418886)

[4.1 Описание структуры программы 6](#_Toc532418887)

[4.2 Описание структур данных 6](#_Toc532418888)

[4.3 Описание алгоритмов 8](#_Toc532418889)

[**5.Заключение** 10](#_Toc532418890)

[**6.Литература** 11](#_Toc532418891)

# **1.Введение**

Во многих вопросах приходится рассматривать некоторую совокупность элементов как единое целое. Набор каких-либо объектов называют множеством. Множество – одно из основных понятий современной математики, используемое почти во всех ее разделах. Активное применение аппарата теории множеств в современной науке приводит к необходимости создания соответствующих программных решений. Для представления и хранения множеств используются структуры данных.

**Структура** **данных** — это программная единица, позволяющая хранить и обрабатывать множество однотипных и/или логически связанных данных в вычислительной технике.

С++ имеет структуру, называемую битовыми полями, которая позволяет работать с отдельными битами. *Битовое поле* – это особый компонент структуры, определяющий длину отдельного ее элемента. Битовые поля полезны по нескольким причинам.

* Если ограничено место для хранения информации, можно сохранить несколько логических переменных в одном байте.
* Некоторые интерфейсы устройств передают информацию, закодировав биты в один байт.
* Некоторым процессам кодирования необходимо получить доступ к отдельным битам в байте.

**Битовое поле** – это элемент структуры, определенный как некоторое число битов, обычно меньшее, чем число битов в целом числе (оно по величине не превосходит машинного слова и зависит от реализации компилятора). Они предназначены для экономного размещения в памяти данных небольшого диапазона, обеспечивают удобный доступ к отдельным битам данных.

**Целью данной работы** является разработка структуры данных для хранения множеств с использованием битовых полей.

# **2.Постановка задачи**

Выполнение работы предполагает решение следующих задач:

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

**Битовое поле**

Для работы с Битовым полем предлагается 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.

**Множество**

Для работы с Множеством предлагается 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;
* дополнение множества;
* нахождение мощности множества;

# **3.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 .

Работа программы 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.

Работа программы при использовании класса множества.

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

# **4.Руководство программиста**

## 4.1 Описание структуры программы

Программа состоит из следующих модулей:

* Модуль TBitField - содержит заголовочный файл tbitfield.h, в котором определен интерфейс класса TBitField и файл tbitfield.cpp, содержащий реализацию этого класса.
* Модуль TSet - содержит заголовочный файл tset.h, в котором определен интерфейс класса Tset, и файл tset.cpp, содержащий реализацию этого класса.

Модули TBitField и Tset находятся в статической библиотеке set.

* Модуль test\_set –содержит файлы test\_tbitfield.cpp и test\_tset.cpp,в которых реализованы тесты для классов TBitField и TSet.
* Модуль sample\_prime\_numbers. Содержит файл sample\_prime\_numbers.cpp, в котором реализован пример использования класса битового поля и множества для поиска простых чисел с использованием алгоритма, называемого "Решетом Эратосфена".

## 4.2 Описание структур данных

**Класс TBitField**

Спецификатор доступа private:

*int bitLen* – длина битового поля;

*int memLen* – количество битов для представления битового поля;

*int \*pMem* – память для представления битового поля;

*int getMemIndex(const int n) const* – метод определяющий индекс бита n в массиве *pMem*;

*int getMemMask(const int n) const* – метод возвращающий маску для бита n.

Спецификатор доступа public:

*TBitField(int len)* – конструктор инициализации. Принимает длину битового поля.

*TBitField(const TBitField &bf)* – конструктор копирования. Принимает ссылку на объект TBitField.

*~TBitField()* – деструктор.

Методы доступа к битам:

*int GetLength() const* – возвращает длину битового.

*void SetBit(const int n)* – устанавливает n-й бит битового поля.

*void ClrBit(const int n)* – очищает n-й бит битового поля.

*int GetBit(const int n) const* – возвращает значение бита n.

Битовые операторы:

*int operator==(const TBitField &bf) const* – принимает ссылку на объект класса TBitField, проверяет на равенство два битовых поля. Если они идентичны, то возвращает 1, иначе 0.

*int operator!=(const TBitField &bf) const* – принимает ссылку на объект класса *TBitField*, проверяет на неравенство два битовых поля. Если они различны, то возвращает 1, иначе 0.

*TBitField& operator=(const TBitField &bf)* – принимает ссылку на объект класса *TBitField*, присваивает полученное битовое поле к исходному.

*TBitField operator|(const TBitField &bf)* – принимает ссылку на объект класса *TBitField*, возвращает новый объект класса *TBitField* полученный путем использования операции побитового «ИЛИ» между исходным и пришедшим битовыми полями.

*TBitField operator&(const TBitField &bf) –* принимает ссылку на объект класса *TBitField*, возвращает новый объект класса *TBitField* полученный путем использования операции побитового «И» между исходным и пришедшим битовыми полями.

*TBitField operator~()* – возвращает объект битового поля, полученный путем применения операции побитового отрицания для исходного битового поля.

Дружественные функции:

*friend istream &operator>>(istream &istr, TBitField &bf)* – ввод битового.

*friend ostream &operator<<(ostream &ostr, const TBitField &bf)* – вывод битового поля на экран.

**Класс TSet**

Спецификатор доступа private:

*int maxPower* – максимальная мощность множества.

*TBitField bitField* – битовое поле для хранения характеристического вектора.

Спецификатор доступа public:

*TSet(int mp)* – конструктор инициализации. Принимает максимальную мощность множества.

*TSet(const TSet &s)* – конструктор копирования. Принимает ссылку на объект класса TSet.

*TSet(const TBitField &bf)* – конструктор преобразования типа. Принимает ссылку на объект класса TBitField.

*operator TBitField()* – возвращает битовое поле характеристического вектора.

Методы доступа к битам:

*int GetMaxPower() const* – возвращает максимальную мощность множества.

*void InsElem(const int Elem)* – добавляет элемент *Elem* в множество.

*void DelElem(const int Elem)* – удаляет элемент *Elem* из множества.

*int IsMember(const int Elem) const* – проверяет наличие элемента Elem в множестве. Если элемент есть, то возвращает 1, иначе 0.

Теоретико-множественные операторы:

*int operator==(const TSet &s) const* –принимает ссылку на объект класса Tset, проверяет на совпадение два битовых поля. Если они совпадают, то возвращает 1, иначе 0.

*int operator!=(const TSet &s) const* – принимает ссылку на объект класса Tset , проверяет на несовпадение два битовых поля. Если они не совпадают, то возвращает 1, иначе 0.

*TSet& operator=(const TSet &s)* – принимает ссылку на объект класса Tset , присваивает полученное множество исходному множеству.

*TSet operator+(const int Elem*) – к исходному множеству добавляет *Elem* и возвращает новый объект класса *TSet*.

*TSet operator-(const int Elem)* – из исходного множества удаляет *Elem* и возвращает новый объект класса *TSet*.

*TSet operator+(const TSet &s)* – принимает ссылку на объект класса TSet, возвращает новый объект класса TSet, который является объединением исходного и полученного в качестве аргумента множества.

*TSet operator\*(const TSet &s)* – принимает ссылку на объект класса TSet, возвращает новый объект класса TSet, который является пересечением исходного и полученного в качестве аргумента множества.

*TSet operator~()* – возвращает дополнение к исходному множеству *.*

Дружественные функции:

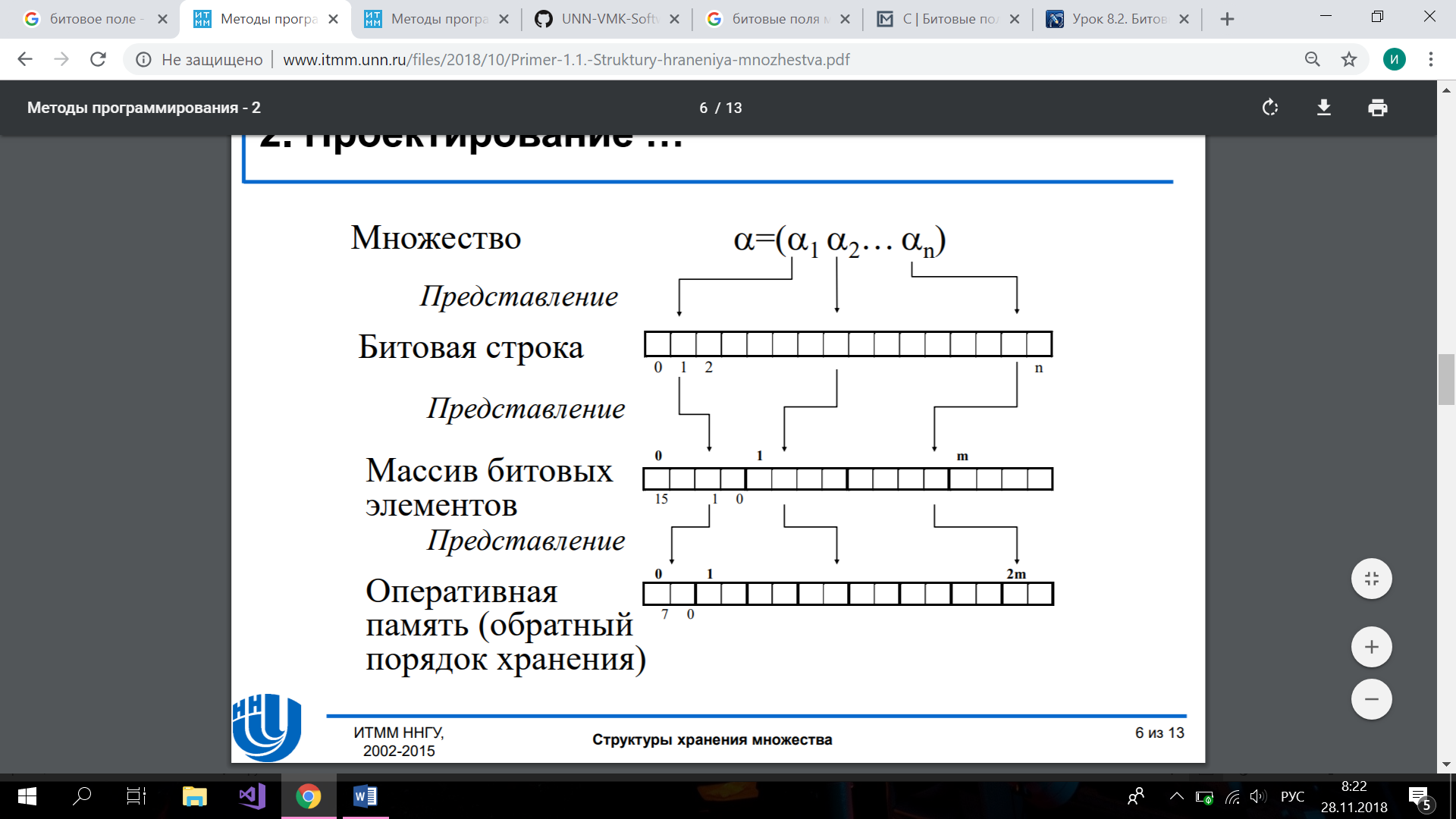
*friend istream &operator>>(istream &istr, TSet &bf)* – ввод множества .

*friend ostream &operator<<(ostream &ostr, const TSet &bf)* – вывод характеристического вектора множества на экран.

## 4.3 Описание алгоритмов

Требуется представить в памяти множество натуральных чисел. Будем использовать для этого битовую строку, в которой каждый бит будет принимать либо значение «1», если элемент присутствует во входном множестве, либо «0» , если его нет.

Битовую строку будем хранить в виде массива. Отметим, что в битовой строке индексация бит происходит слева направо, в массиве аналогично, однако биты элемента массива нумеруются справа налево (Рис. 1).



Множество

Битовая строка

Битовое поле

Рис.1 Представление множества в виде битового поля

Для реализации методов работы с битами в классе TBitField реализованы два вспомогательных метода:

*GetMemIndex* – для -го бита определяется индекс элемента в массиве pMem, в котором хранится битовое поле. является целой частью от деления номера бита на количество битов выделяемых на один элемент массива.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | |  |  | |  |  |  |
|  |  |  |  |

*GetMemMask* – определяет битовую маску для -го бита. Ставим значение 1 в позицию , определяемую по формуле 2. Битовую маску рассматриваем как двоичную запись числа.

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | |  | ( | |  |

Теперь методы для работы с битами реализуем следующим образом:

* Чтобы установить бит на позицию используем побитовое «ИЛИ» между битами элемента массива *pMem*, в котором находится бит , и между его битовой маской.
* Чтобы очистить бит используем побитовое «И» между битами элемента массива *pMem*, в котором находится бит , и между дополнением его битовой маски.
* Чтобы получить значение бита используем побитовое «И» между битами элемента массива *pMem*, в котором находится бит , и между его битовой маской.

# 

# **5.Заключение**

В результате лабораторной работы была разработана структура данных для хранения множеств с использованием битовых полей. Выполнены поставленные задачи: реализованы класс битового поля TBitField и класс множества TSet и операции для работы с ними, обеспечена работоспособность тестов и примеров использования, реализованы несколько тестов на базе GoogleTest.

# **6.Литература**

1. Национальный открытый университет «Интуит». Курс «Структуры и алгоритмы компьютерной обработки данных». Лекция 19.Битовые поля <http://www.intuit.ru/studies/courses/648/504/lecture/11442>(Дата обращения: 09.12.2018)
2. Лабораторный практикум. Составители: Барышева И.В., Мееров И.Б., Сысоев А.В., Шестакова Н.В. Под редакцией Гергеля В.П. Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017. – 105с.

<http://www.unn.ru/books/met_files/Pract_ADS.pdf> (Дата обращения: 09.12.2018)

1. Методы программирования

<http://www.itmm.unn.ru/files/2018/10/Primer-1.1.-Struktury-hraneniya-mnozhestva.pdf> (Дата обращения: 09.12.2018)