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**

**«Структура хранения множества»**

**Выполнил:** студент группы 381706-1

Полетуева Анастасия Николаевна

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

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

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc1327240)

[2. Постановка задачи 4](#_Toc1327241)

[3. Руководство пользователя 5](#_Toc1327242)

[4. Руководство программиста 6](#_Toc1327243)

[4.1. Описание структуры программы 6](#_Toc1327244)

[4.2. Описание структур данных 6](#_Toc1327245)

[4.3. Описание алгоритмов 8](#_Toc1327246)

[5. Заключение 9](#_Toc1327247)

[6. Литература 10](#_Toc1327248)

# Введение

Множество — это структура данных, содержащая конечный набор элементов некоторого типа. Каждый элемент содержится только в одном экземпляре, т.е. разные элементы множества не равны между собой. Элементы множества никак не упорядочены. Множество — это потенциально неограниченная структура, оно может содержать любое конечное число элементов.

Чтобы определить множество, достаточно указать характеристическое свойство

элементов, т. е. такое свойство, которым обладают все элементы этого множества и только

они.

Поскольку построение математической структуры основано на понятии множества, само множество не может быть определено, как структура данных. В дальнейших рассуждениях мы будем опираться на следующее описание множества, вытекающее из сделанных выше допущений.

Каждому множеству A поставим в соответствие характеристический вектор ,

Все операции над множествами могут быть заменены в таком случае на операции над характеристическими векторами или по-другому – над битовой строкой, которая хранится в битовом поле.

Структура данных — 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.

Битовое поле представляет целое или беззнаковое целое значение, занимающее в памяти фиксированное число битов. Число связанных бит –ширина поля.

Такая структура хранения довольно удобна и вот лишь несколько примеров, почему:

- при ограниченной памяти вы можете хранить несколько булевых (истина, ложь) переменных в одном байте;

- интерфейсы некоторых устройств передают информацию, которая кодируется в биты внутри одного байта;

- некоторые программы кодирования нуждаются в доступе к битам внутри байта.

# Постановка задачи

В рамках лабораторной работы ставится задача разработки класса «Битовое поле» (класс TBitField). Для работы с этой структурой данных будут реализованы методы:

* 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.

Для работы с «Множеств» (TSet) предлагается реализовать следующие операции:

* включение элемента в множество;
* исключение элемента из множества;
* проверка наличия элемента в множестве;
* сравнение множеств
* сложение множеств;
* пересечение множеств;
* разность множества с элементом;
* копирование множества;

вычисление максимальной мощности множества.

Выполнение работы предполагает решение следующих задач:

1. Разработка и реализация классов – TBitField, TSet.
2. Реализация набора тестов, написанных с использованием GoogleTesting Framework.

# Руководство пользователя

При запуске программы пользователю предлагается ввести верхнюю границу целых значений для решета Эратосфена.

После этого на экран выведется множество некратных чисел в двоичной системе, а также простые числа, но уже в десятичной (и то и то в пределах введенной границы). А также будет выведено количество простых чисел в заданном диапазоне от 0 до заданной границы.

Далее идет разветвление программы для работы с множеством и битовым полем.

**Множество:**

Пользователю необходимо ввести наибольший элемент множества целых чисел. Далее нужно ввести сами элементы множества. После чего на экран выведется множество в преставлении битовой строки и в целых числах.

**Битовое поле:**

Пользователя необходимо ввести размер битовой строки, а затем и саму битовую строку указанной длины. Затем на экран выведется введенная битовая строка и множество чисел этой строки.

# Руководство программиста

## Описание структуры программы

Программа состоит из следующих модулей:

* Модуль set – статическая библиотека. Включает в себя заголовочные файлы tbitfield.h и tset.h с описанием классов битового поля TBitField и множеств TSet. А так же файлы с реализацией методов этих классов tbitfield.cpp и tset.cpp. Проект, использующий фреймворк Google Test, для проверки правильности работы классов «TBitField» и «TSet»;
* Модуль test\_set – набор тестов для обоих классов. Включает в себя файлы test\_tbitfield.cpp и test\_tset.cpp
* Модуль sample\_prime\_numbers – пример использования классов с использованием алгоритма «Решето Эратосфена». Включает в себя файл sample\_prime\_numbers.cpp.

## Описание структур данных

**Класс TBitField:**

Поля:

* BitLen - длина битового поля - макс. к-во битов
* \*pMem - память для представления битового поля
* MemLen - количество элементов Мем для представления битового поля
* Конструкторы и деструктор:
* TBitField(int len);
* TBitField(const TBitField &bf);
* ~TBitField();

Методы:

* 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

Перегрузки:

* 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); - ввод битового поля с клавиатуры
* friend ostream &operator<<(ostream &ostr, const TBitField &bf); - вывод битового поля

**Класс Tset:**

Поля:

* MaxPower - максимальная мощность множества
* BitField - битовое поле для хранения характеристического вектора
* Конструкторы:
* TSet(int mp);
* TSet(const TSet &s);
* TSet(const TBitField &bf);

Методы:

* 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

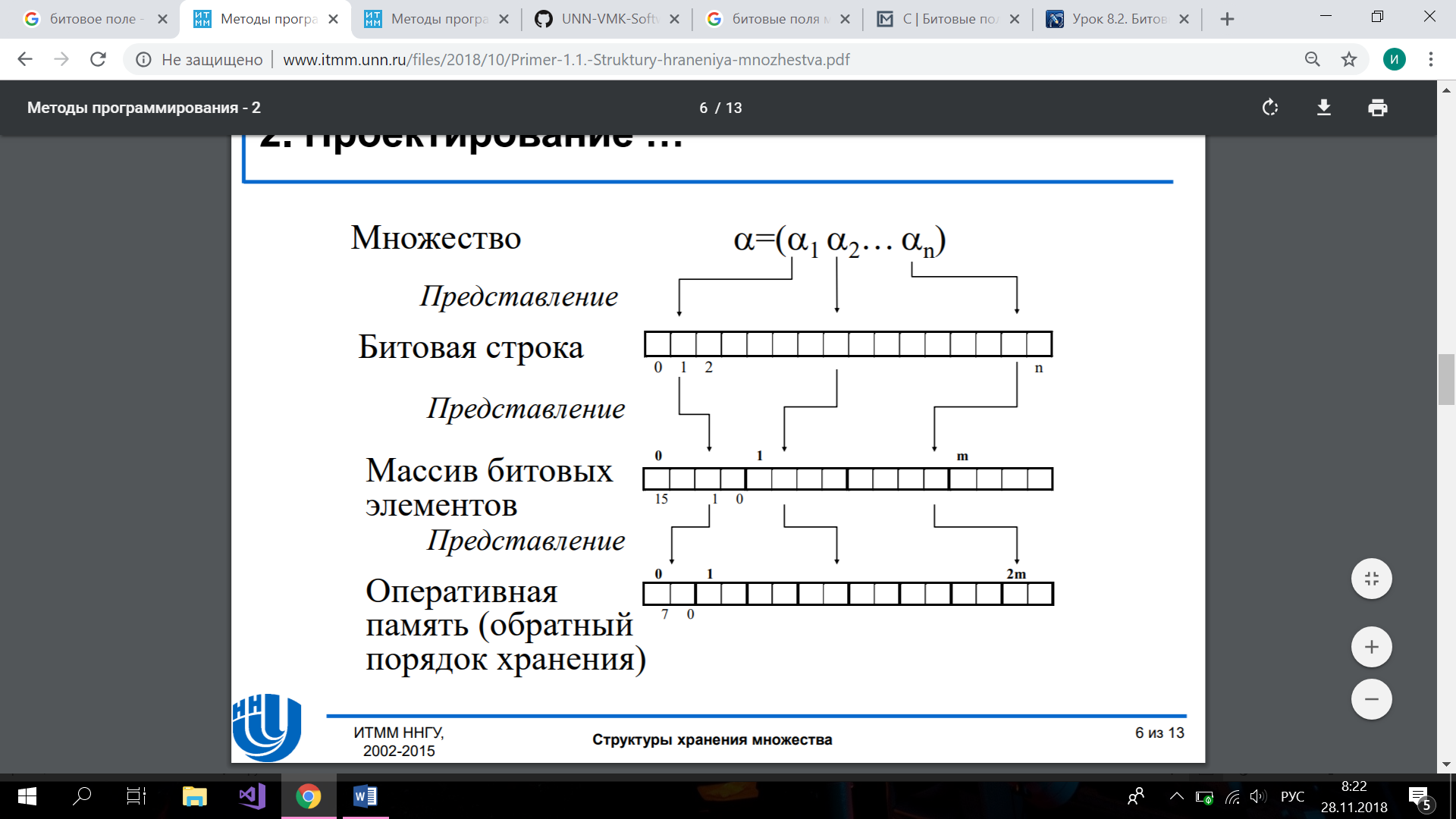
Перегрузки:

* 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);- ввод множества с клавиатуры
* friend ostream &operator<<(ostream &ostr, const TSet &bf); - вывод характеристического вектора множества

## Описание алгоритмов

Формирование битового поля:

На входе мы имеем множество натуральных чисел. Его можно представить виде битовой строки, где каждый бит принимает значение «1», если элемент присутствует во входном множестве, и «0» в противном случае. Битовую строку удобнее всего хранить в массиве. Однако, стоит заметить одну особенность – в битовой строке нумерация бит происходит слева направо, в массиве нумерация элементов также происходит слева направо, однако биты элемента массива нумеруются справа налево.



Множество

Битовая строка

Битовое поле

Рисунок 1. Представление множества в качестве битового поля.

Для написания методов: установить, очистить бит и получить его значение, нужно было реализовать следующие методы:

* GetMemIndex: по формуле n << 5 для n-го бита устанавливается индекс в массиве pMem, в котором хранится битовое поле.
* GetMemMask: по формуле 1 << (n & 31) для n-го бита устанавливается битовая маска, то есть запись единичного бита на место, посчитанное данной формулой.

Тогда для:

* Установка бита: по формуле pMem[GetMemIndex(n)] |= GetMemMask(n) или по-другому побитовое ИЛИ между битами массива pMem и его битовой маски.
* Очистить бит: по формуле pMem[GetMemIndex(n)] &= ~GetMemMask(n) или по-другому побитовое И между битами массива pMem и инверсией его битовой маски.
* Получить значение бита: по формуле pMem[GetMemIndex(n)] & GetMemMask(n) или по-другому побитовое И между битами массива pMem и его битовой маски. Возвращает либо 0, либо 1.

# Заключение

В ходе выполнения данной лабораторной была реализована структура хранения множества с помощью битовых полей. Созданы классы «TBitField» и «TSet», реализованы методы, описанные в «Описание структур данных». С помощью этих классов была реализована небольшая программа – «Решето Эратосфена», которая помогает находить простые числа. Разработаны тесты на основе Google C++ Testing Framework.

# Литература

1. Гергель В.П. Методические материалы по курсу «Методы программирования» (часть 1), 2015, -96с.