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 381706-2

Банденков Даниил Викторович

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

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

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc533123557)

[2. Постановка задачи 4](#_Toc533123558)

[3. Руководство пользователя 5](#_Toc533123559)

[4. Руководство программиста 6](#_Toc533123560)

[4.1. Описание структуры программы 6](#_Toc533123561)

[4.2. Описание структур данных 6](#_Toc533123562)

[4.3 Описание алгоритмов 9](#_Toc533123566)

[5. Эксперименты 11](#_Toc533123567)

[6. Заключение 12](#_Toc533123568)

[7. Литература 13](#_Toc533123569)

# 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 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.

Данная работа посвящена изучению одного из возможных подходов к хранению и обработке множеств, а именно битовым полям.

Битовое поле - некоторое количество бит, расположенных последовательно в памяти. Битовые поля применяются для максимально полной упаковки информации, если не важна скорость доступа к этой информации. Можно использовать, например, для увеличения пропускной способности канала при передаче информации по сети или для уменьшения размера информации при хранении

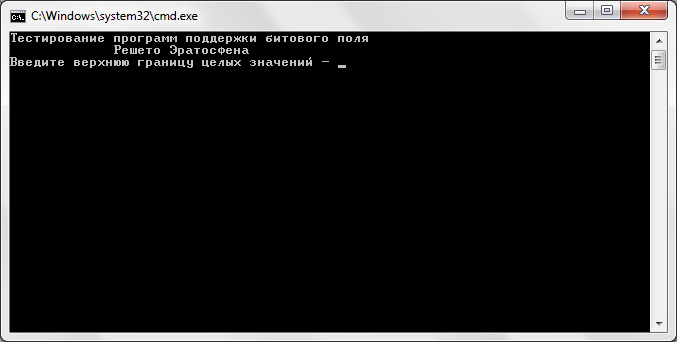
# Постановка задачи

Выполнение работы предполагает решение следующих задач:

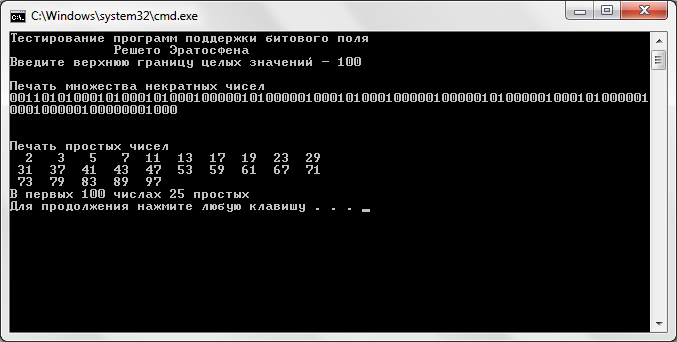
1. Разработка интерфейса и реализация класса битового поля TBitField.
2. Разработка интерфейса и реализация класса множества TSet.
3. Обеспечение работоспособности тестов и примера использования.
4. Реализация нескольких простых тестов на базе Google Test.

# Руководство пользователя

В программе используется алгоритм ≪Решето Эратосфена≫ для нахождения простых чисел. При запуске программы с пользователя запрашивается верхняя граница целых значений *n*, среди которых в дальнейшем будут найдены все простые числа.

Рис.1 Пример начального меню.

Затем происходит поиск и подсчет простых чисел После на экран выводится множество некратных чисел (битовая строка простых чисел) и простые числа от 0 до *n*.

Рис.2 Пример использования программы. Решето Эратосфена.

# Руководство программиста

## Описание структуры программы

В решении содержатся следующие модули:

1. tbitfield.h, tbitfield.cpp – модуль с классом, реализующим операции над *Битовыми полями*.
2. tset.h, tset.cpp – модуль с классом, реализующим обработку *Множеств*.

* sample\_prime\_numbers.cpp,sample\_performance\_check.cpp, sample\_standart\_input\_output.cpp – модуль программы тестирования, с которым работает пользователь (решето Эратосфена), в котором проводятся эксперименты и приводятся работы стандартных потоков ввода и вывода.

1. test\_main.cpp, test\_tbitfield.cpp, test\_tset.cpp – модуль с функциями тестирования для созданных классов. Содержат 25 тестов для класса TBitField и 23 теста для класса TSet.

## Описание структур данных

#### *Класс* TBitField

*Определены три поля и два метода реализации со спецификатором доступа* private:

int bitLen - длина битового поля

TELEM \*pMem - память для представления битового поля

int memLen - количество битов для представления битового поля

int GetMemIndex(const int n) const – метод определяющий индекс бита n в массиве pMem

TELEM GetMemMask(const int n) const – метод возвращающий маску для бита n

*Со спецификатором 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.

*Перегрузка битовых операторов:*

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.

intoperator!=(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 методы ввода\вывода:*

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

*Определены два поля со спецификатором доступа* private:

int maxPower - максимальная мощность множества

TBitField bitField - битовое поле для хранения характеристического вектора.

*Определены два поля со спецификатором доступа* public:

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

TSet(const TSet &s) - конструктор копирования. Принимает ссылку на объект класс TSet.

TSet(const TBitField &bf) - конструктор преобразования типа. Принимает ссылку на объект класс TBitField.

*Метод преобразования типа битовому полю:*

operator TBitField() – возвращает битовое поле характеристического вектора.

*Методы доступа к битам:*

int GetMaxPower(void) 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

*Перегрузка теоретико-множественных операторов:*

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

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

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

TSet operator+(const int Elem) – к исходному множеству добавляет Elem (если его там нет) и возвращает новый объект класса TSet.

TSet operator-(const int Elem) – из исходного множества удаляет Elem (если он там есть) 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 множества.

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

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

*Дружественные методы ввода\вывода:*

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

1. GetMemIndex – по формуле 1 для -го бита определяется индекс элемента в массиве pMem, в котором хранится битовое поле. Т.е. - целая часть от деления номера бита на количество битов в одном элементе массива.

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

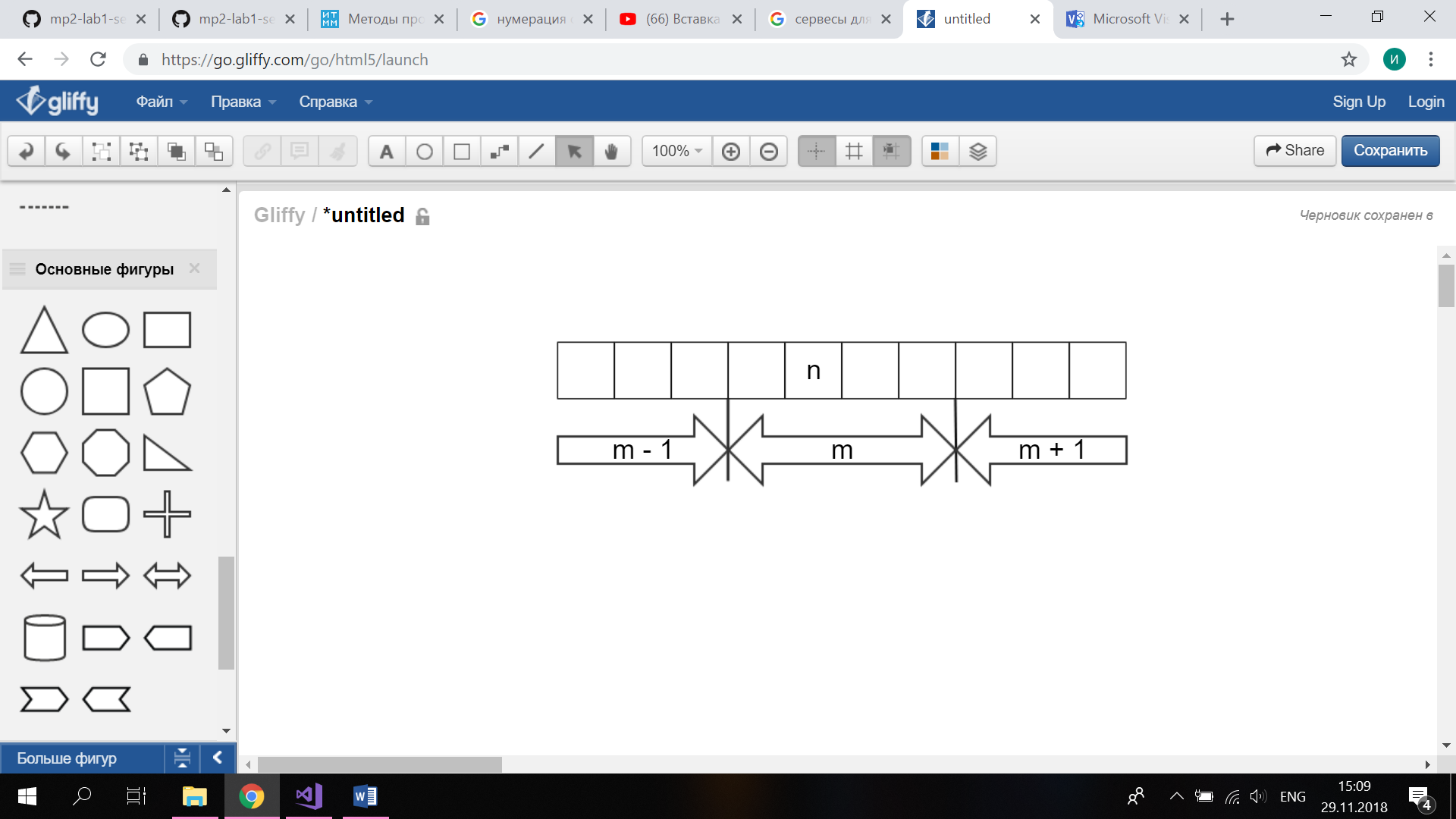


Рисунок 1. GetMemIndex. Возвращает индекс массива m, принимает номер бита n.

1. GetMemMask – определяет битовую маску (длинной sizeof(int)\*8 – размер элемента массива) для -го бита. Выставляет единичный бит на позиции , определяемой по формуле 2. Метод возвращает число, т.к. битовую маску моно рассматривать как двоичную запись числа. Например, GetMemMask для рисунка 3 вернет «4».

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

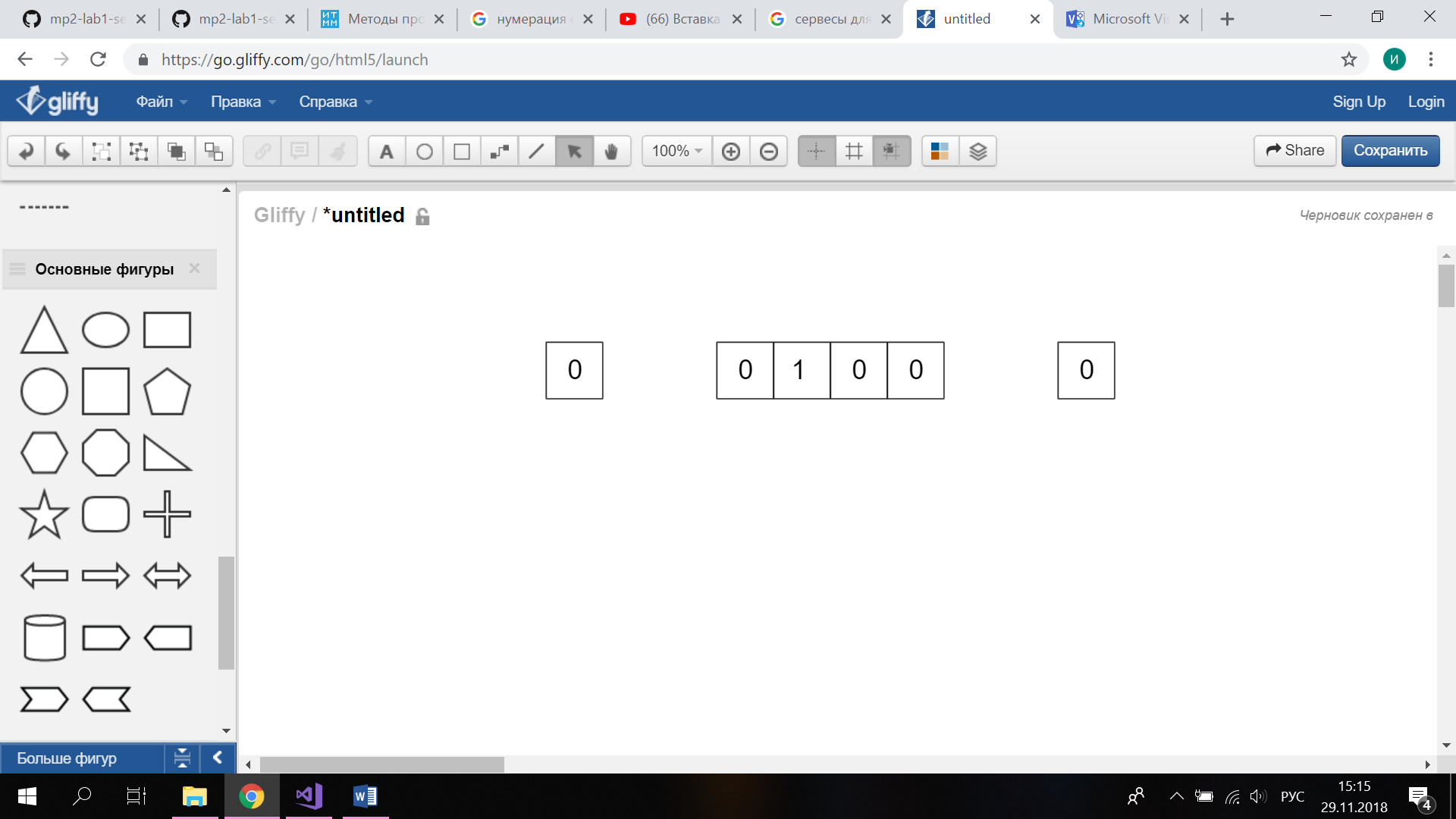


Рисунок 2. GetMemMask. Битовая маска для бита n из рисунка 2.

1. Чтобы установить бит на позицию осталось использовать побитовое «ИЛИ» между битами элемента массива pMem, в котором находится бит n, и между его битовой маской:

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

1. Чтобы очистить бит используем побитовое «И» между битами элемента массива pMem, в котором находится бит , и между дополнением его битовой маски:

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

1. Чтобы получить значение бита n используем побитовое «И» между битами элемента массива pMem, в котором находится бит , и между его битовой маской:

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

# Эксперименты

Для эксперимента был взята логическая операция ИЛИ из TBitField.

Информация о компьютере:

Процессор:

AMD A6 – 3420M APU with Radeon HD Graphics 1.50 GHz

Версия ОС:

Windows 7 (64 бит)

Видеокарта:

Модель: AMD Radeon HD 7450M

Память:

Оперативная память: 6144 ГБ

# Заключение

В ходе выполнения лабораторной был произведен анализ задачи: установлено понятие множества, операции над элементами множества и теоретико-множественные операции.

В результате проделанной работы мне удалось

1. Реализовать класс битового поля TBitField.
2. Реализовать класс множества TSet.
3. Обеспечить работоспособность тестов и примера использования.
4. Реализовать нескольких тестов на базе Google Test.

Таким образом, благодаря этой лабораторной работе мне удалось разработать структуру данных для хранения множеств с использованием битовых полей.

# Литература

1. Битовые поля | программирование на C и C++ - Режим доступа: <http://www.ccpp.ru/books/bitovye-polya> (Дата обращения 18.12.2018)
2. Гергель В.П. Методические материалы по курсу «Методы программирования 2», Нижний Новгород, 2015.
3. <https://ru.wikipedia.org/wiki/Битовое_поле> (Дата обращения 18.12.2018)