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

Мазур Даниил Андреевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

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

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc531274519)

[2. Постановка задачи 4](#_Toc531274520)

[3. Руководство пользователя 5](#_Toc531274521)

[4. Руководство программиста 6](#_Toc531274522)

[4.1. Описание структуры программы 6](#_Toc531274523)

[4.2. Описание структур данных 7](#_Toc531274524)

[4.3. Описание алгоритмов 9](#_Toc531274525)

[5. Заключение 11](#_Toc531274526)

[6. Литература 12](#_Toc531274527)

# Введение

Одной из главных задач любого разработчика при создании программного обеспечения – реализовать наиболее качественную оптимизацию программы.

Битовое поле в программировании — некоторое количество бит, расположенных последовательно в памяти, значение которых процессор не способен прочитать из-за особенностей аппаратной реализации.

Битовые поля применяются для максимально полной упаковки информации, если не важна скорость доступа к этой информации. Например, для увеличения пропускной способности канала при передаче информации по сети или для уменьшения размера информации при хранении. Также использование битовых полей оправдано, если процессор поддерживает специализированные инструкции для работы с битовыми полями, а компилятор использует эти инструкции при генерировании машинного кода.

**Цель данной 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 некоторых 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".

После запуска программы пользователю предлагается ввести границу целых значений, среди которых будет осуществляться поиск всех простых чисел. 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 алгоритм работы программы различается для классов битового поля и множества.

1. При использовании Битовых полей.

Пользователю предлагается самостоятельно ввести битовую строку. Для этого необходимо задать размер битовой строки . Затем ввести последовательность битов длинной +1 с учетом нулевого элемента. Программа выведет на экран битовую строку с числами, которые отражены в ней, и завершит работу.

1. 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 число, большее k, либо же -1. Программа выведет на экран полученное множество с его битовой строкой и завершит работу.

# Руководство программиста

## Описание структуры программы

В решении содержатся следующие модули:

* Модуль tbitfield.

Он включает в себя заголовочный файл tbitfield.h, в котором определен интерфейс класса битового поля TBitField и файл содержащий его реализацию tbitfield.cpp. Всего в tbitfield.cpp находится 17 функций. В них реализованы конструкторы, деструктор для класса TBitField, перегружены битовые операторы, операторы ввода/вывода битовых полей и прописаны методы доступа к отдельным битам.

* Модуль tset.

Он включает в себя заголовочный файл tset.h, в котором определен интерфейс класса битового поля TSet и файл содержащий его реализацию tset.cpp. Всего в tset.cpp находится 18 функций. В них реализованы конструкторы, деструктор для класса TSet, перегружены теоретико-множественные операторы, операторы ввода/вывода множеств и прописаны методы доступа к отдельным битам.

* Тесты для классов TBitField и TSet.

Для каждого класса реализован набор тесов в файлах test\_tbitfield.cpp (29 тестов) и test\_tset.cpp(25 тестов). Каждый тест представлен отдельной функцией. Тесты проверяют работу каждого метода классов TBitField и TSet.

* Пример 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

#### *Класс TBitField*

Класс TBitField является шаблонным классом. В нем определены три поля и два метода реализации со спецификатором доступа private:

*int bitLen* - длина битового поля

*int \*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

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

Далее в публичной зоне (public):

1. Прописаны 2 конструктора и деструктор:

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

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

*~TBitField()* – деструктор.

1. Реализованы 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.

1. Перегружены 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.

Также в классе определены 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*

Класс TSet является шаблонным классом. В нем определены два поля со спецификатором доступа private:

*int maxPower* - максимальная мощность множества

*TBitField bitField* - битовое поле для хранения характеристического вектора

Далее в публичной зоне (public):

1. Прописаны 3 конструктора:

*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.

1. Реализован метод преобразования типа битовому полю:

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

1. Реализованы 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

1. Перегружены теоретико-множественные операторы:

*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.

Также в классе определены 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.

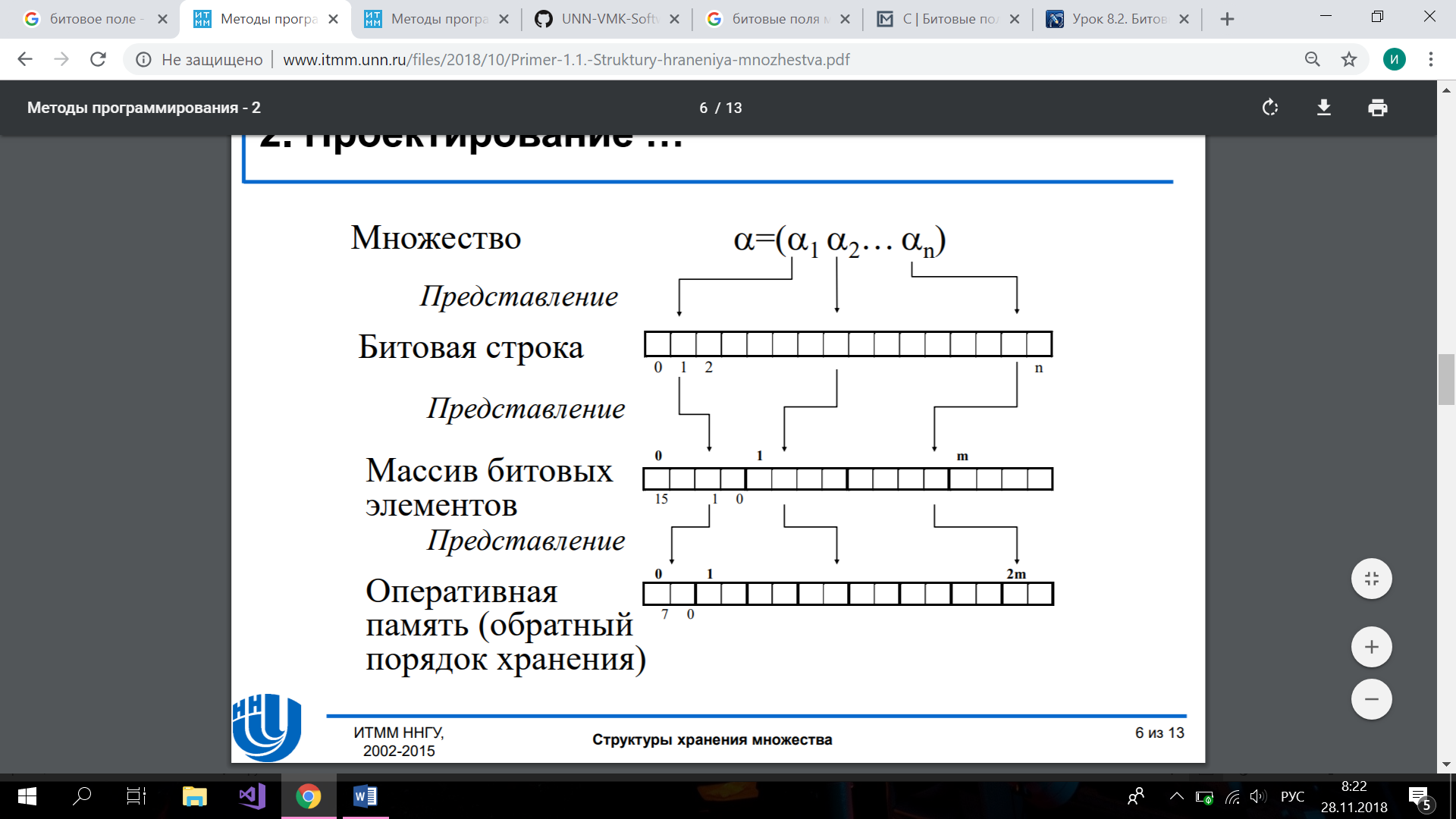
## Описание алгоритмов

В данном разделе будут описаны алгоритмы, применяемые для реализации методов класса TBuildField.

Формирование битового поля.

На входе программа считывает некоторое множество 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 справа налево, однако биты элемента массива нумеруются справа налево (рис 1.)



Множество

Битовая строка

Битовое поле

Рисунок 1. Представление множества в качестве битового поля.

Для оптимизации стандартных методов: получить бит, очистить бит, положить бит и т.д., в классе TBuildField дополнительно были прописаны два метода реализации битового поля:

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 (в моей работе я использую массив целых чисел).

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

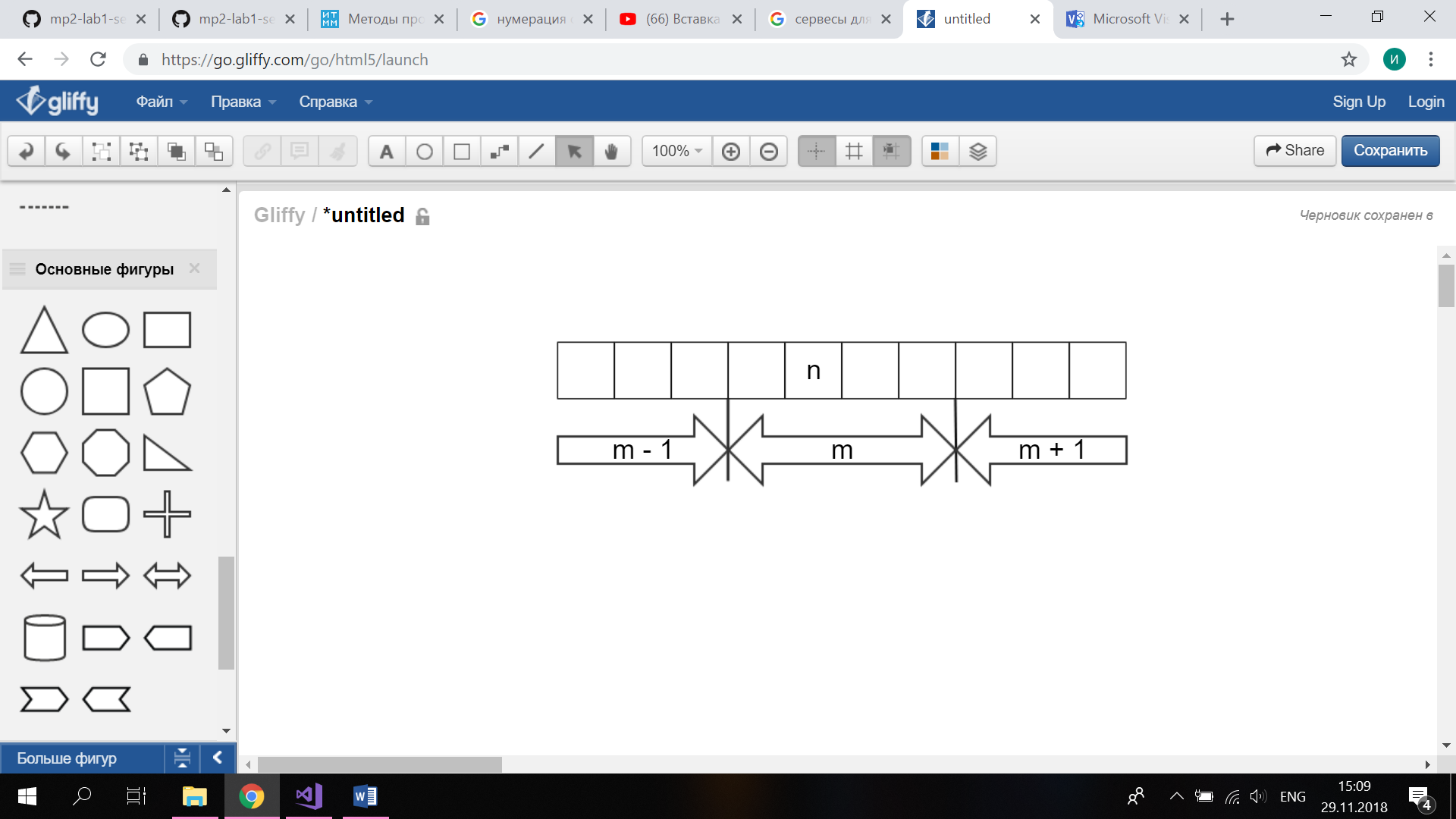


Рисунок 2. GetMemIndex. Возвращает индекс массива m, принимает номер бита n.

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 двичную запись числа. Например, GetMemMask для рисунка 3 вернет «4».

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

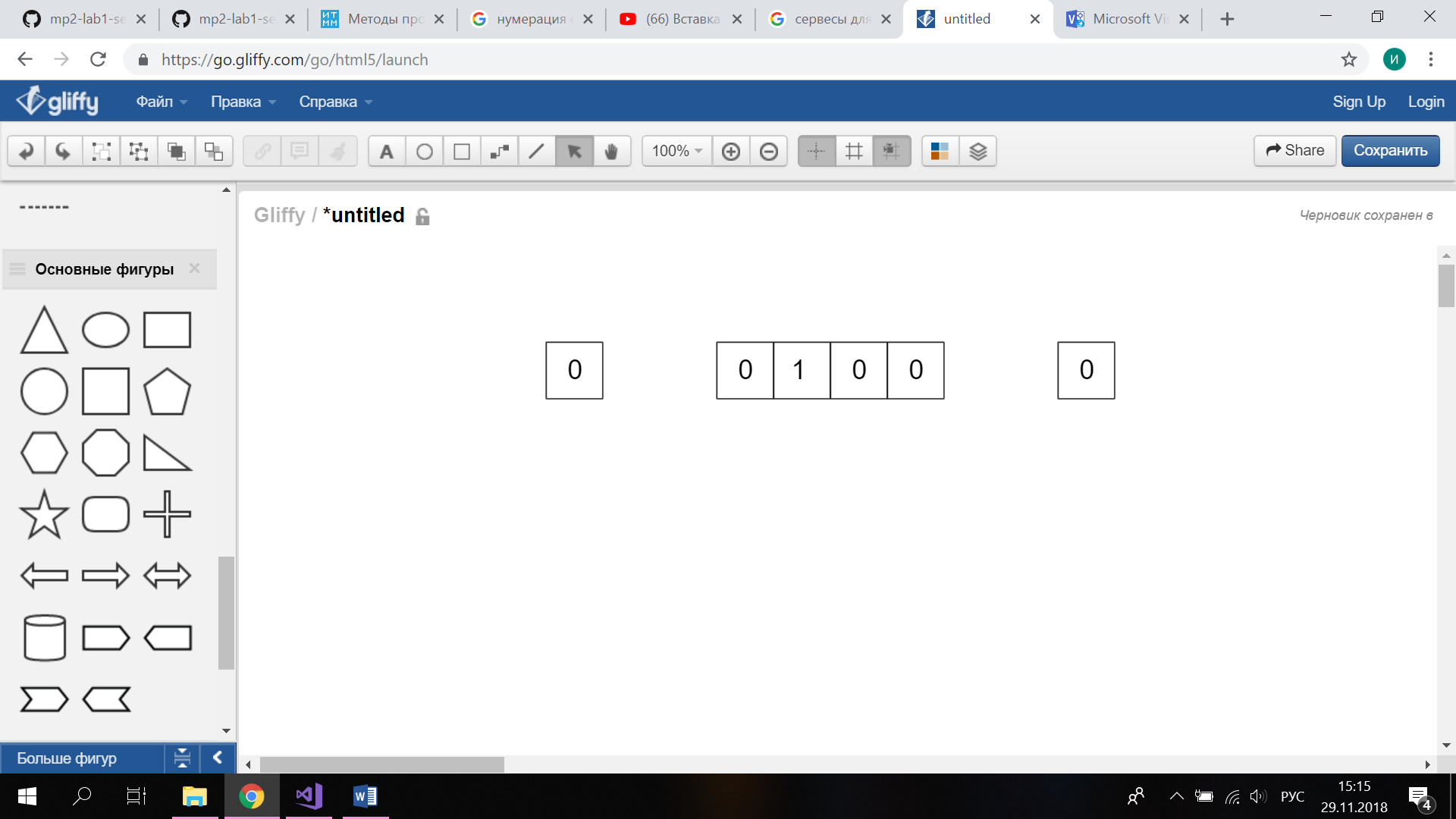


Рисунок 3. GetMemMask. Битовая маска для бита n из рисунка 2.

Благодаря этим методам реализация следующих методов получилась достаточно компактной:

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:

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

# Заключение

В этой работе мы впервые работали с битами напрямую. В ходе выполнения лабораторной был произведен анализ задачи: установлено понятие множества, операции над элементами множества и теоретико-множественные операции.

Осуществлено проектирование работы:

* Понятие характеристического вектора
* Представление вектора в виде битовой строки
* Формирование битовой строки в виде массива
* Битовой формат элемента массива
* Выделение базового класса для реализации битовых строк

Также получены навыки составления тестов для программы.

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

* Книги

1. A.O. Грудзинский. Методы программирования, Издательство Нижегородского госуниверситета, 2006.
2. Васильев А.Н. Самоучитель С++ с примерами и задачами. -СПб.: Наука и Техника, 2016. -480с.

* Ссылки в Internet

1. Гергель В.П. Методические материалы по курсу «Методы программирования 2»: [http://www.itmm.unn.ru/files/2018/10/Primer-1.1.-Struktury-hraneniya-mnozhestva.pdf], 2015.