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

Власов Андрей Сергеевич

\_\_\_\_\_\_\_\_\_\_\_ 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](#_Toc531274524) 6

[4.3. Описание алгоритмов](#_Toc531274525) 8

[5. Заключение 11](#_Toc531274526)

[6. Литература 12](#_Toc531274527)

# Введение

Структура данных (англ. data structure) — 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. Одной из таких структур является битовое поле.

Битовые поля обеспечивают удобный доступ к отдельным битам данных. Они позволяют формировать объекты с длиной, не кратной байту. Что в свою очередь позволяет экономить память, более плотно размещая данные.

Цель данной лабораторной работы: реализация битовых полей и множеств.

# 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.

1. 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.

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 -1 или число более . В конце на экран будет выведено полученное множество и его битовая строка. На этом работа программы завершается.

# Руководство программиста

## Описание структуры программы

Программа состоит из следующих модулей:

* Модуль Set.

Содержит в себе заголовочный файл tbitfield.h, в котором определен интерфейс класса битового поля TBitField и файл tbitfield.cpp , содержащий его реализацию. А также включает в себя заголовочный файл tset.h, в котором определен интерфейс класса битового поля TSet и файл tset.cpp , содержащий его реализацию.

* Модуль Test\_Set.

Содержит файлы test\_tbitfield.cpp и test\_tset.cpp , в которых находятся тесты (на базе фреймворка Google test). Каждый тест представлен отдельной функцией. Тесты проверяют работу каждого метода классов TBitField и TSet.

* Модуль Sample.

Содержит файл sample\_prime\_numbers.cpp - пример использования класса битового поля и множества, в котором реализован алгоритм поиска простых чисел, называемый "Решетом Эратосфена".

## 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.

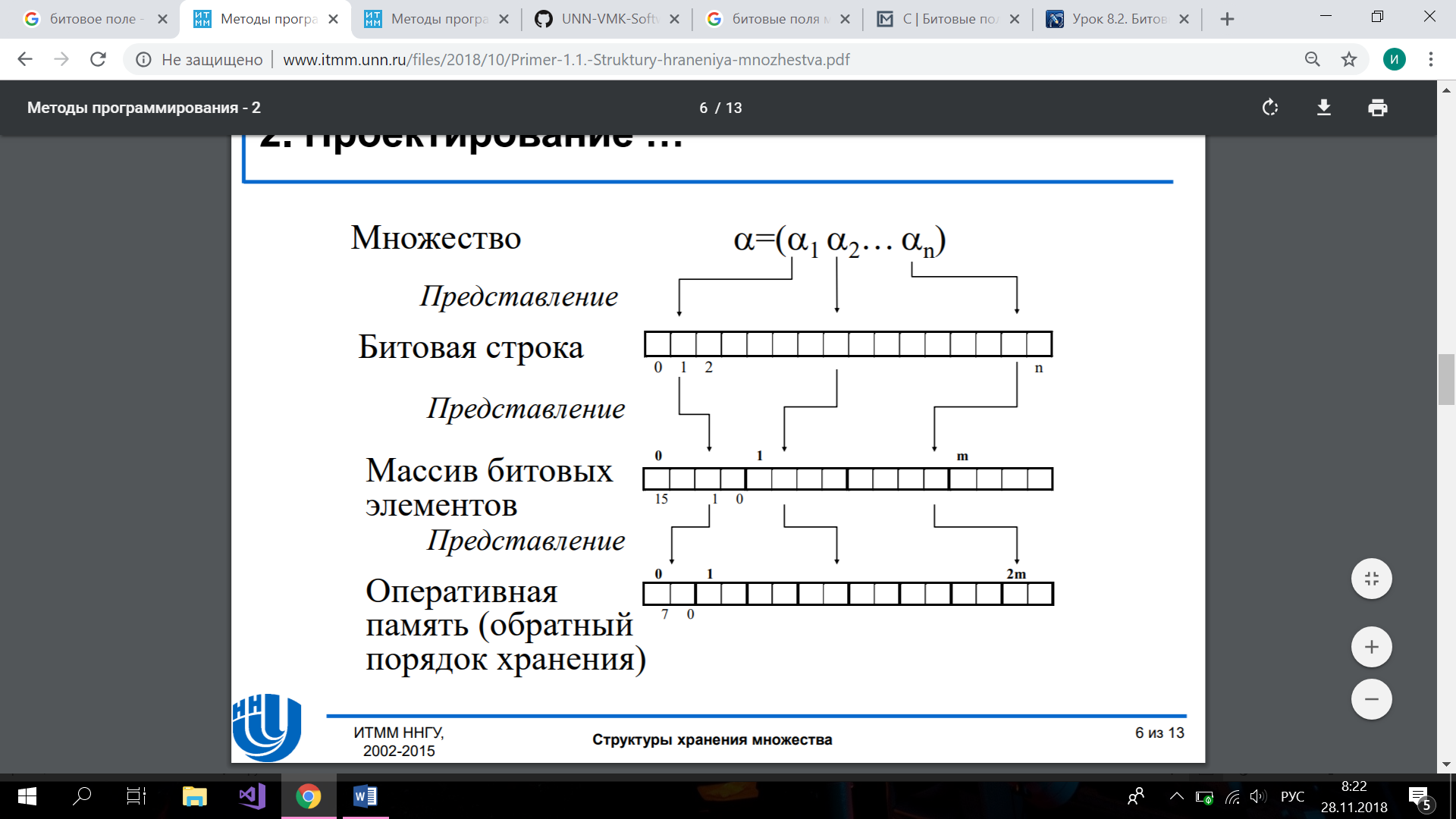
## Описание алгоритмов

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

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 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).

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

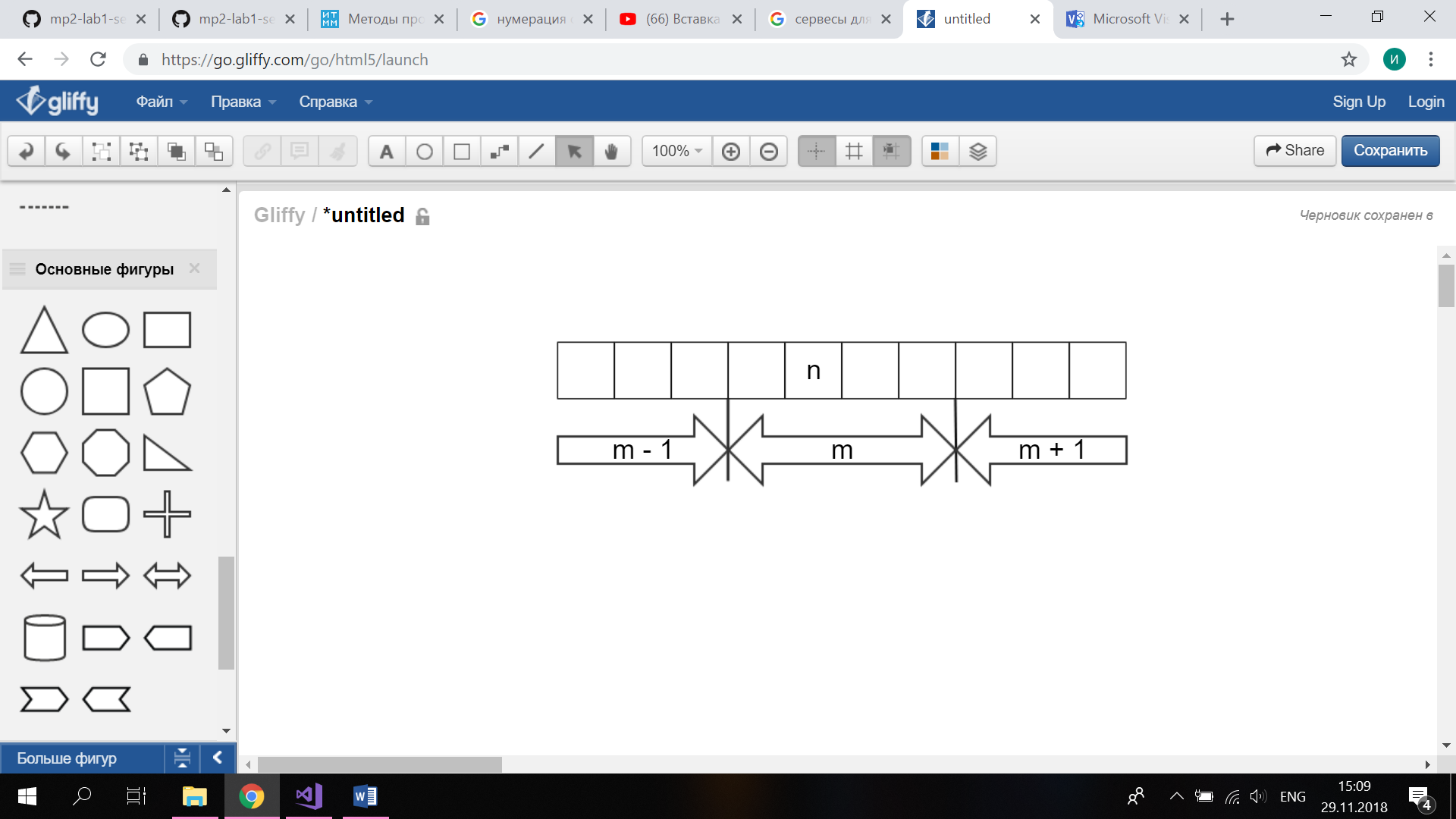


Рисунок 2. GetMemIndex. Возвращает индекс массива m, принимает номер бита n.

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».

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

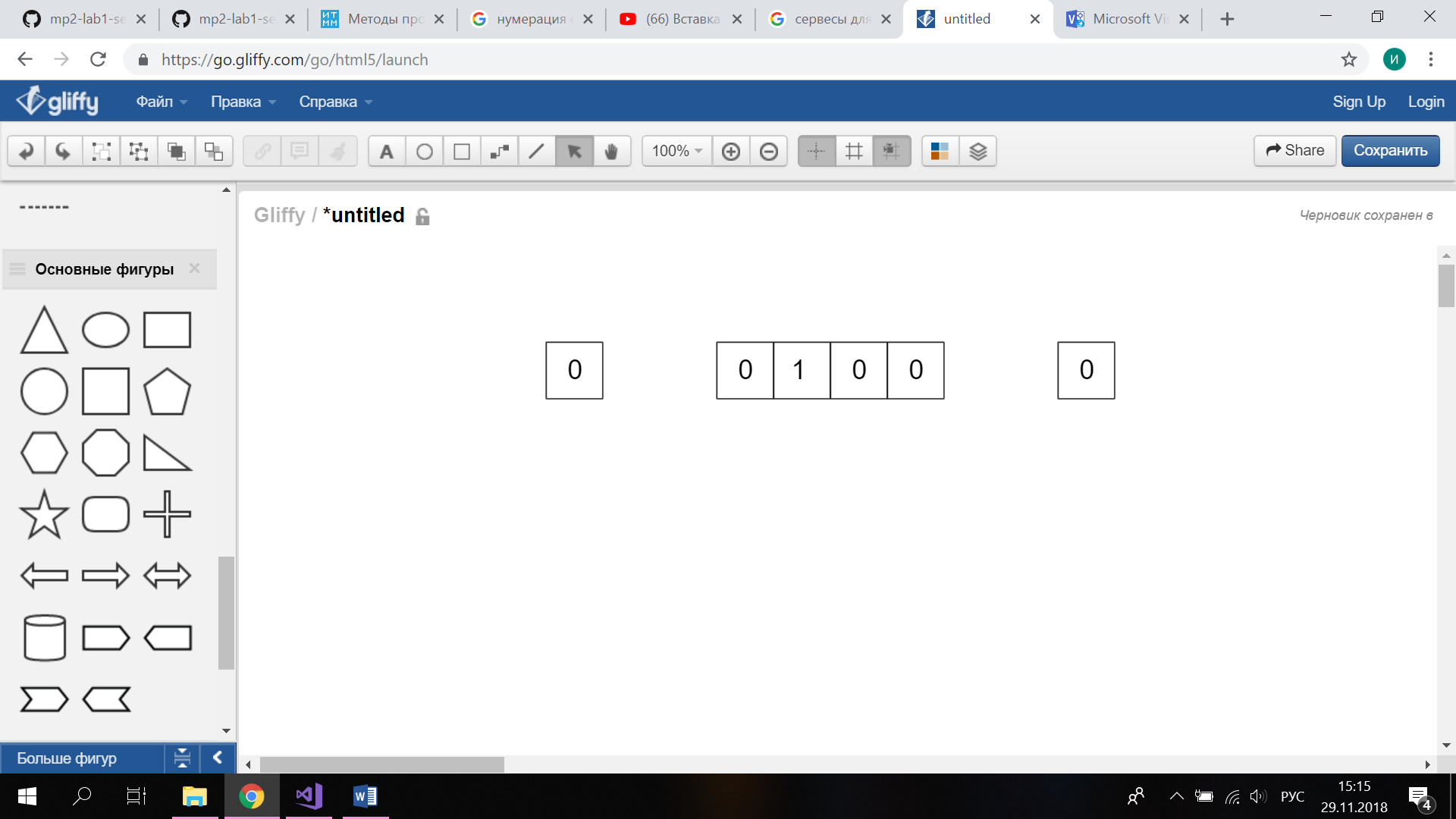


Рисунок 3. GetMemMask. Битовая маска для бита n из рисунка 2.

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 данной лабораторной работе была выполнена реализация такой структуры данных как битовые поля. А именно была разработана библиотека, реализующая класс битового поля и класс множества, показан пример использования, а также написан набор тестов с использованием Google C++ Testing Framework.

# Литература

Internet – ресурсы:

1. <https://ru.wikipedia.org/wiki/Битовое_поле>
2. <http://www.c-cpp.ru/books/bitovye-polya>

Книги:

1. Гергель В.П. Методические материалы по курсу «Методы программирования 2», Нижний Новгород, 2015.
2. Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы.: Пер. с англ. М.: Издат. дом «Вильямс», 2000. С. 58–76