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 на основе битовых полей»**

**Выполнил:** студент группы 381706-1

Денисов Владислав Львович

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

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

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc531466307)

[2. Постановка задачи 4](#_Toc531466308)

[3. Руководство пользователя 5](#_Toc531466309)

[4. Руководство программиста 7](#_Toc531466310)

[4.1 Описание структуры программы 7](#_Toc531466311)

[4.2 Описание структур данных 7](#_Toc531466312)

[4.3 Описание алгоритмов 9](#_Toc531466313)

[5. Заключение 12](#_Toc531466314)

[6. Литература 13](#_Toc531466315)

# Введение

Множества. Похожее понятие есть в математике, а точнее в теории множеств. Влияние его настолько велико, что теория множеств является фундаментом ряда математических дисциплин (общая алгебра, функциональный анализ и т.д.). Что же это такое? Множество — неупорядоченный набор элементов, без повторов. И все? Ни тебе произвольного доступа, ничего! Зачем же такое нужно?

Мы сталкиваемся с множествами в повседневной жизни: множества предметов, множество людей и т.п. Множества используются в науке, и с её развитием возникла потребность в соответствующем программном решении.

Вспомним, что любая программа является неким набором данных и команд, обрабатывающих эти данные. Одна из главных задач программиста состоит в выборе наиболее оптимальной модели хранения данных для конкретной задачи и реализации таких алгоритмов, которые будут наилучшим образом использовать доступные ресурсы. Сделать это помогает знание того, какие бывают структуры данных.

**Структура данных** – программная единица, которая определяет метод хранения и обработки различных логически связанных данных в вычислительной технике.

Речь в нашей лабораторной работе идет о множествах, построенных на битовых полях. Рассмотрим их несколько подробнее. Битовое поле – некоторое количество бит, расположенных последовательно. Рассмотрим, чем мы можем руководствоваться, чтобы сделать выбор в пользу именно такой структуры для хранения данных:

1. Ограничено место для хранения информации и можно сохранить несколько логических (истина/ложь) переменных в одном байте;
2. Интерфейсы устройств передают информацию, закодировав биты в один байт;
3. Необходим удобный доступ к отдельным битам в байте;
4. Используются объекты с длинной не кратной байту.

Кроме того, битовые поля часто применяют для представления совокупности данных целого типа. Поскольку любое множество может быть описано характеристическим вектором

,

то целесообразно сформировать битовую строку и хранить ее в битовом поле.

В данной лабораторной работе мы изучим битовые поля и реализуем структуру хранения множеств с использованием битовых полей.

# Постановка задачи

В рамках лабораторной работы ставится задача эффективной реализации структуры данных для хранения множеств с использованием битовых полей.

Нам необходимо реализовать следующие операции над множествами:

* включение элемента в множество;
* исключение элемента из множества;
* проверка наличия элемента в множестве;
* объединение множества с элементом;
* разность множества с элементом;
* сравнение множеств
* сложение множеств;
* пересечение множеств;
* нахождение дополнения множества;
* вычисление мощности множества;
* копирование множеств.

Сделаем также следующие основные допущения:

1. Условимся рассматривать в дальнейшем конечные множества.
2. Элементы множества проиндексированы (каждому элементу соответствует уникальный индекс).
3. Множество индексов элементов составляет непрерывный диапазон целых значений.

Программное решение будет выглядеть следующим образом:

1. Вспомогательный класс битового поля TBitField.
2. Класс множества TSet, реализованный с использованием вспомогательного класса TBitField.
3. Программа, демонстрирующая работу класса «Множество».
4. Набор автоматических тестов с использованием фреймворка 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 .

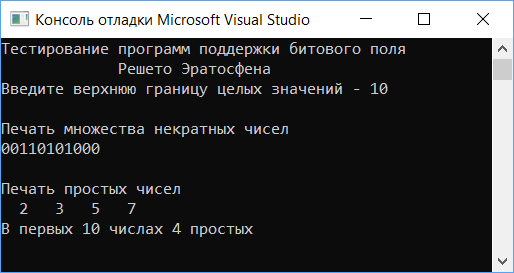


Рис.1 Пример использования программы. Решето Эратосфена.

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 (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. Пример показан на Рис.2.

1. При использовании класса множества.

Пользователю предлагается самостоятельно ввести множество, состоящее из неотрицательных целых чисел. Для этого необходимо задать наибольший элемент множества . Затем ввести набор чисел от 0 до . Для завершения ввода вводим -1 либо число . В конце на экран будет выведено полученное множество и его битовая строка. На этом работа программы завершается. Пример показан на Рис.3.

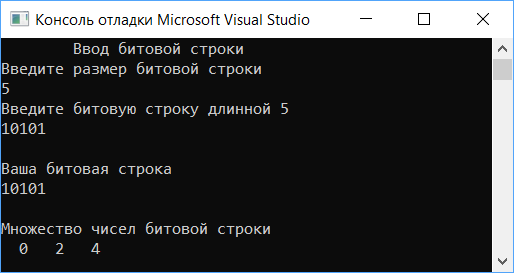


Рис. 2 Тестирование Битового поля (класс TBitField).

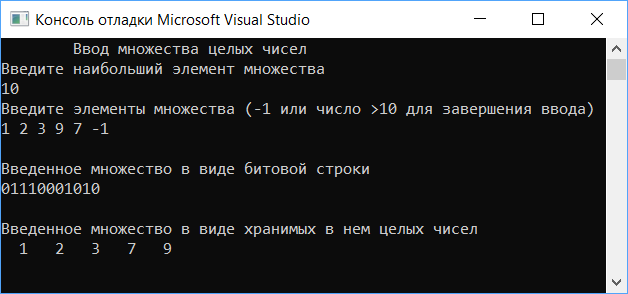


Рис. 3 Тестирование Множества (класс TSet).

# Руководство программиста

## 4.1 Описание структуры программы

Программа состоит из следующих модулей:

* Модуль *set* – статическая библиотека. Содержит заголовочные файлы *tbitfield.h* и *tset.h*, в которых определены соответственно интерфейсы классов битового поля *TBitField* и множеств *TSet*. Файлы, содержащие их реализацию – *tbitfield.cpp* и *tset.cpp*.
* Модуль *test\_set*. Содержит для каждого класса (*TBitField* и *TSet*) набор тестов, реализованный в файлах *test\_tbitfield.cpp* (29 тестов) и *test\_tset.cpp* (25 тестов) с помощью использования фреймворка Google Test
* 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".

## Описание структур данных

#### Класс TBitField – битовое поле

Рассмотрим класс *TBitField* подробно.

**Элементы, объявленные со спецификатором 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.

**Со спецификатором доступа public:**

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

*TBitField(const TBitField &bf)* – конструктор копирования.

*~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 тому, которым был вызван оператор.

*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 поля с консоли.

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

#### Класс TSet – множество

Рассмотрим класс *TSet* подробно.

**Элементы, объявленные со спецификатором private:**

*int maxPower* – максимальная мощность множества.

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

**Со спецификатором доступа public:**

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

*TSet(const TSet &s)* – конструктор копирования.

*TSet(const TBitField &bf)* – конструктор преобразования типа. Принимает ссылку на объект с типом 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. Если элемент есть, то возвращает 1, иначе 0.

**Теоретико-множественные операторы:**

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

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

*TSet& operator=(const TSet &s)* – присваивает полученное множество в то, которое вызвало оператор.

*TSet operator+(const int Elem*) – к исходному множеству добавляет *Elem* (если его там нет, иначе ничего не меняет) и возвращает новый объект класса *TSet*.

*TSet operator-(const int Elem)* – из исходного множества удаляет *Elem* (если он там есть, иначе ничего не меняет) и возвращает новый объект класса *TSet*.

*TSet operator+(const TSet &s)* – объединяет множество, которое вызвало оператор, с пришедшим в качестве аргумента, результат возвращается как новый объект класса *TSet*.

*TSet operator\*(const TSet &s)* – выполняет операцию пересечения двух множеств, результат возвращает как новый объект класса *TSet*.

*TSet operator~(void)* – возвращает дополнение к исходному множеству как новый элемент класса *TSet.*

**Дружественные функции:**

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

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

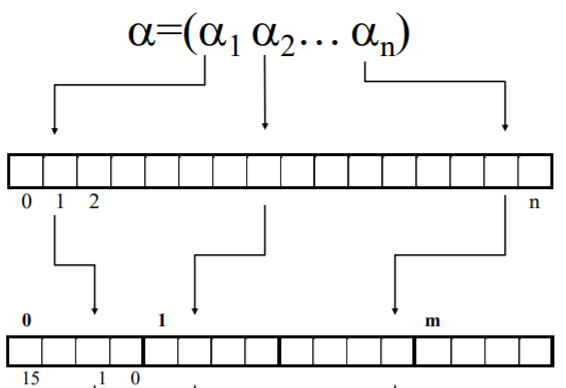
## Описание алгоритмов

Рассмотрим некоторые алгоритмы, работа которых не очевидна на первый взгляд.

**Создание битового поля.**

Необходимо представить в памяти множество натуральных чисел. Для этого будем использовать битовую строку, в которой каждый бит будет принимать либо значение «1», если элемент присутствует во входном множестве, либо «0» в противном случае.

Битовую строку будем хранить в виде массива. Заметим, 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 битового поля.

Для удобства работы с методами получения, очищения, добавления бита и подобным в классе *TBitField* используются вспомогательные методы:

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

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

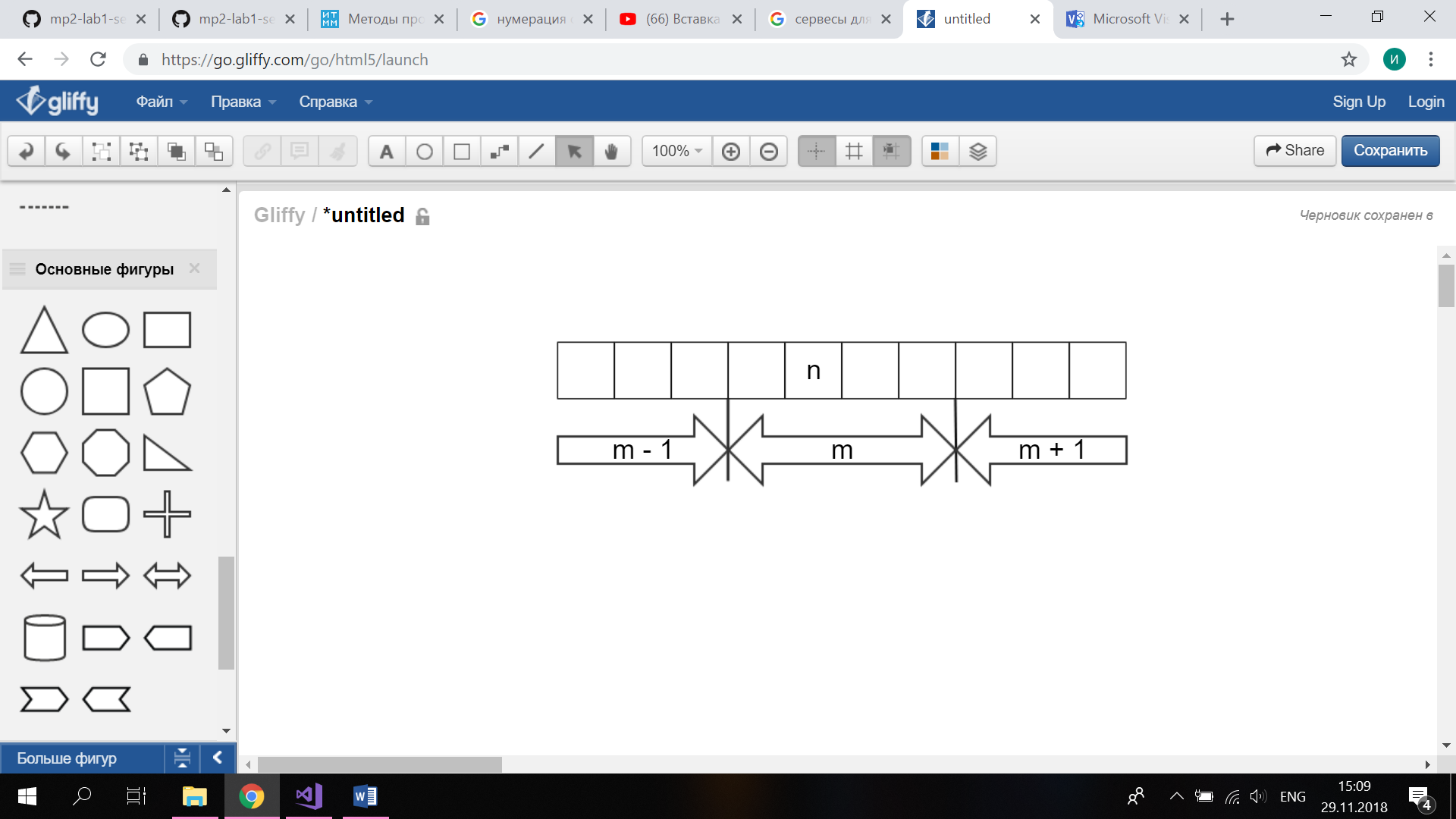


Рис 2. Пример того, где может находится бит n для элемента m.

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

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

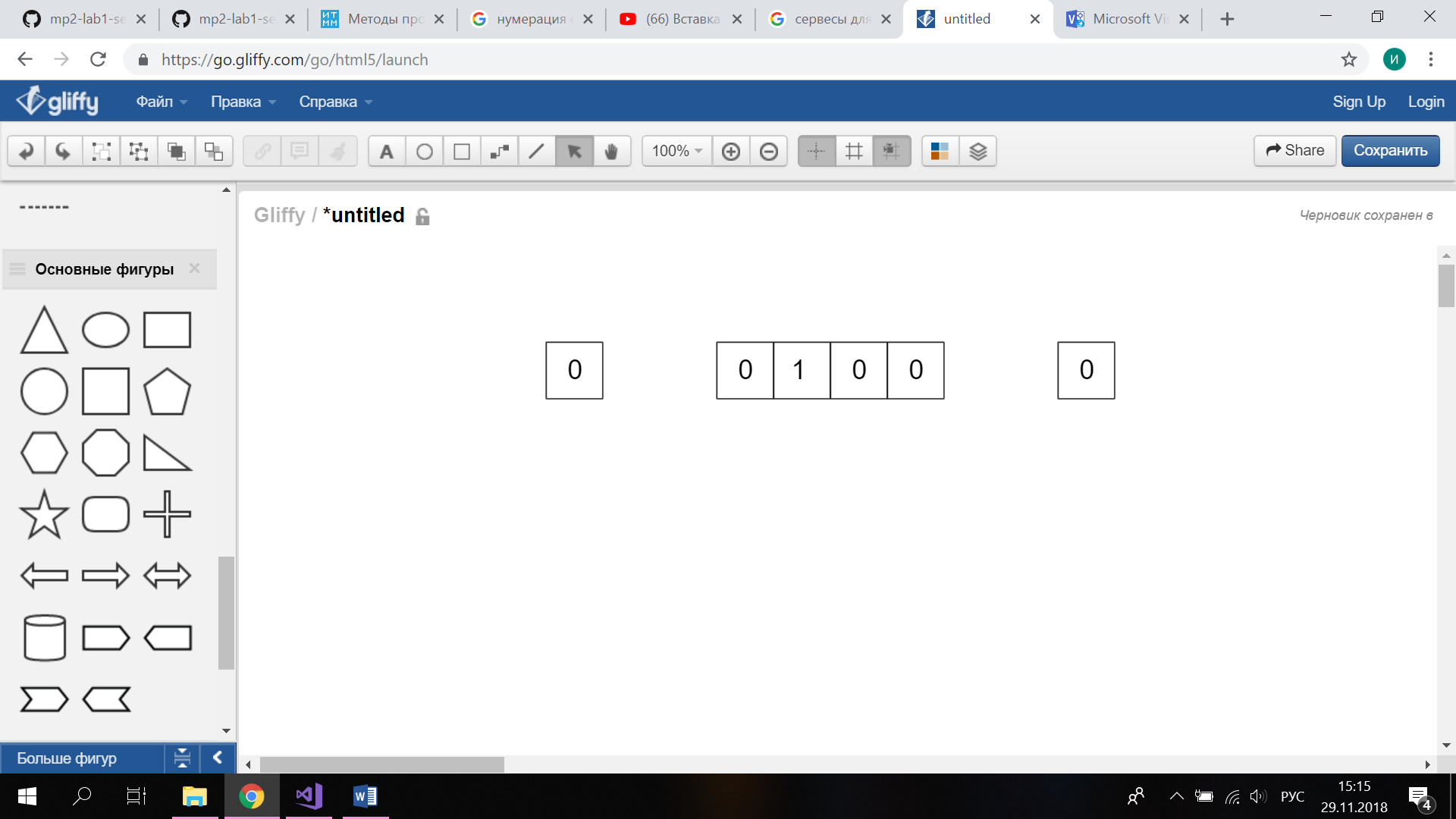


Рис. 3. Битовая маска бита n для Рис.2.

Рассмотрим непосредственно методы работы с битами.

Чтобы установить бит на позицию можем использовать 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:

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

# Заключение

В результате лабораторной работы были изучены битовые поля, способ их хранения в памяти и методы работы с ними.

Было реализовано множество на основе битовых полей и соответствующие операции над его элементами, установленные в разделе «Постановка задачи».

Произведено знакомство с фреймворком Google Test. Обеспечена работоспособность тестов, которые были предложены в качестве некоторой базы для лабораторной работы. Были добавлены несколько своих тестов.

Программное решение было проверено на примере предложенного алгоритма «Решето Эратосфена».

# Литература

1. Структуры данных | Библиотека программиста [Электронный ресурс] URL: <https://proglib.io/p/data-structure-algorithms/> (дата обращения 29.11.2018)
2. Википедия: свободная электронная энциклопедия: на русском языке [Электронный ресурс] // URL: https://ru.wikipedia.org/wiki/Битовая\_маска (дата обращения: 01.12.2018)
3. Методы программирования [Электронный ресурс] // URL: <http://www.itmm.unn.ru/files/2018/10/Primer-1.1.-Struktury-hraneniya-mnozhestva.pdf> (дата обращения: 01.12.2018)