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

Кольтюшкина Янина Вадимовна

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

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

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc532899254)

[2. Постановка задачи 4](#_Toc532899255)

[3. Руководство пользователя 5](#_Toc532899256)

[4. Руководство программиста 7](#_Toc532899257)

[4.1. Описание структуры программы 7](#_Toc532899258)

[4.2. Описание структур данных 7](#_Toc532899259)

[4.3. Описание алгоритмов 8](#_Toc532899260)

[5. Заключение 11](#_Toc532899261)

# Введение

На протяжении своей жизни мы постоянно так или иначе сталкиваемся с какими-то данными. Представленные в самом разном виде (словарь по английскому, выписка о расходах за месяц и многое другое), они по сути все представляют собой просто большой объем информации, записанной в каком-то виде. Примером может послужить массив – набор элементов (может быть упорядоченным или нет). И этот массив данных может быть просто огромным. И вот тут уже перед программистом встает задача как хранить большие объемы данных в ограниченной памяти компьютера. По сути именно для решения этой проблемы и существуют структуры данных.

**Цель данной работы** — разработка структуры данных для хранения множеств с использованием битовых полей, а также освоение таких инструментов разработки программного обеспечения, как система контроля версий Git и фрэймворк для разработки автоматических тестов Google Test.

**Но это еще не все. Так же мы познакомимся в целом с тем, что такое структура данных, как и зачем они применяются и более детально разберем что из себя представляет Битовое 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

- Реализация класса битового поля TBitField согласно заданному интерфейсу.

- Реализация класса множества TSet согласно заданному интерфейсу.

- Обеспечение работоспособности тестов и примера использования.

- Реализация нескольких простых тестов на базе Google Test.

- Публикация исходных кодов в личном репозитории на GitHub.

# Руководство пользователя

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

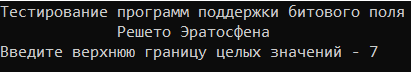


Рис. 1 (верхняя граница целых значений)

Далее на экран выведется множество некратных чисел в двоичной системе и простые числа в десятичной (все в пределах введенной границы). А так же выводится количество простых чисел в диапазоне до заданной границы.

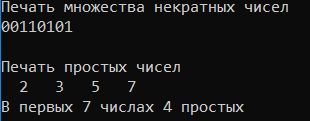


Рис.2 (вывод некратных и простых чисел, количества простых чисел)

После этого этапа начинается разветвление программы для работы с множеством и битовым полем.

Битовое поле:

Пользователь вводит размер битовой строки

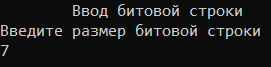


Рис. 3 (ввод размера битовой строки)

И саму битовую строку указанной длины



Рис. 4 (ввод значений битовой строки)

Затем на экран выведется введенная битовая строка и множество чисел этой строки

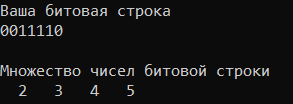


Рис. 5 (вывод строки и множества чисел)

Множество:

Пользователь вводит наибольший элемент множества целых чисел

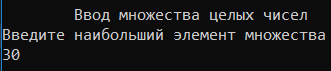


Рис. 6 (ввод наибольшего элемента)

Затем нужно ввести сами элементы множества

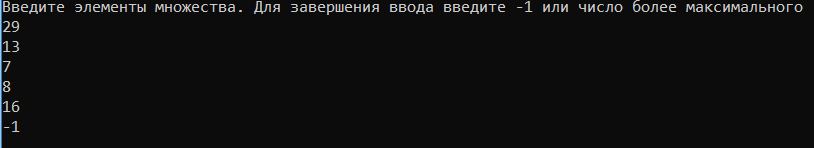


Рис. 7 (ввод элементов множества)

После всего этого на экран выведется множество в преставлении битовой строки и в целых числах.

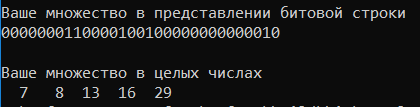


Рис. 8 (вывод множества в двух представлениях)

# 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*.

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

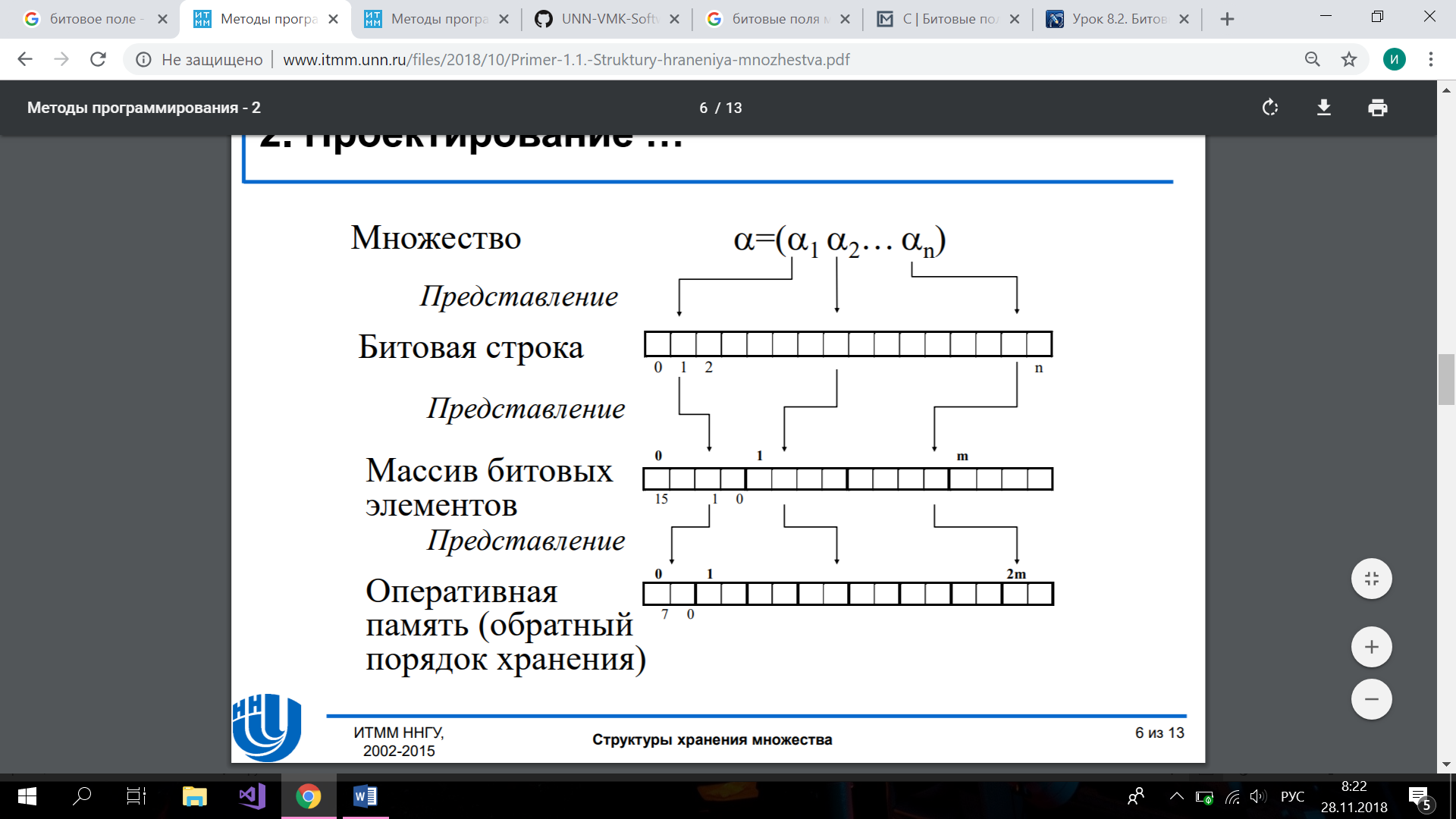
friend istream &operator>>(istream &istr, TSet &bf);- ввод множества с клавиатуры

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 (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).

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

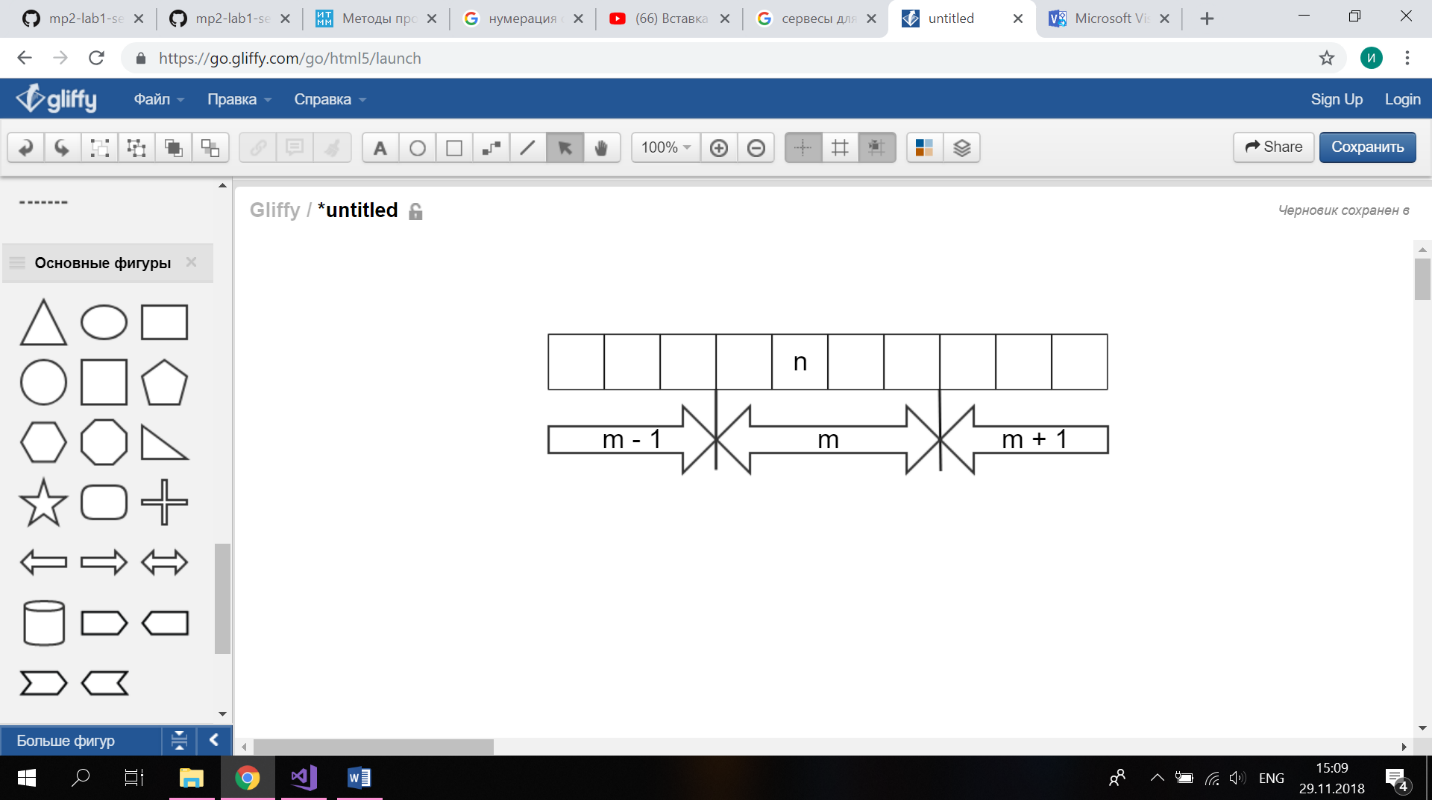


Рисунок 2. GetMemIndex. Возвращает индекс массива m, принимает номер бита n.

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».

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

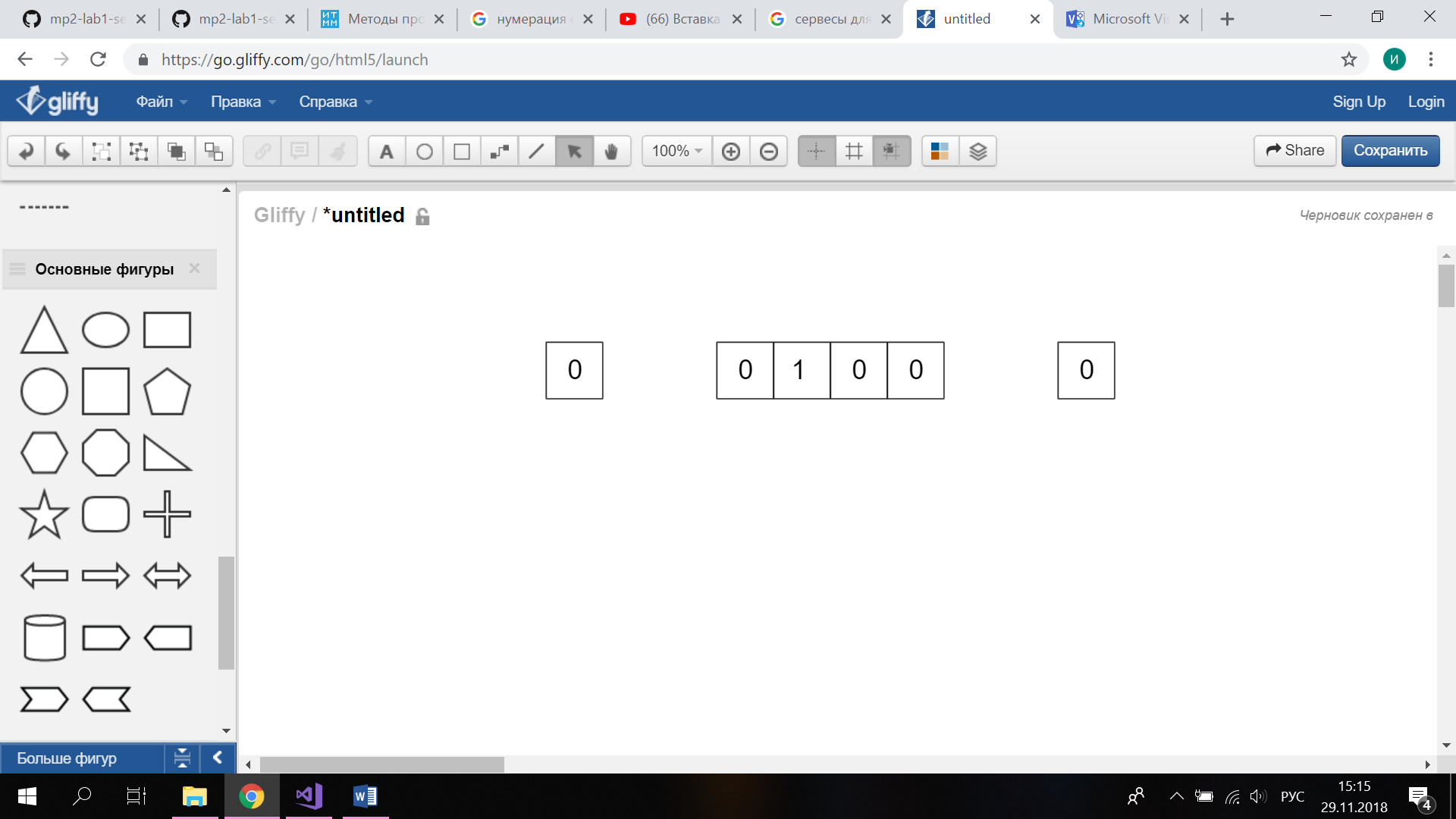


Рисунок 3. GetMemMask. Битовая маска для бита n из рисунка 2.

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 лабораторной работе я смогла справиться со всеми поставленными задачами. В первую очередь, на собственном опыте попробовать реализовать такую структуру данных, как битовое поле. Вместе с ней, согласно заданному интерфейсу, я реализовала классы TBitField и TSet. Написание нескольких своих тестов в дополнение к уже имеющимся, а так же достижение одной из главных целей – того, чтобы все тесты были пройдены, помогло мне разобраться с системой автоматических тестов Google Test. Но и это еще не все. Полезным на практике навыком является и публикация исходного кода в личном репозитории на GitHub, что не раз еще пригодится в будущем

1. **Литература**
2. 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
3. Битовые поля структур и объединений: “https://studfiles.net/preview/6272664/page:6/”.
4. Битовые поля: “https://studfiles.net/preview/5828100/page:19/”.
5. Структуры данных “https://ru.wikipedia.org/wiki”