МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Национальный исследовательский**

**Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**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

Колесова Кристина Юрьевна

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

ассистент каф. МОСТ ИИТММ

Лебедев Илья Геннадьевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2018.

Оглавление

[Введение 3](#_Toc532406345)

[Постановка задачи 4](#_Toc532406346)

[Руководство пользователя 5](#_Toc532406347)

[Руководство программиста 6](#_Toc532406348)

[Описание структуры программы 6](#_Toc532406349)

[Описание структур данных 7](#_Toc532406350)

[Описание алгоритмов 9](#_Toc532406351)

[Заключение 12](#_Toc532406352)

# Введение

Во время разработки любой программы могут встречаться различного рода ограничения. Одно из частых ограничений – ограничение памяти. В таких случаях перед программистом стоит задача наиболее компактно и практично расположить все данные в памяти. Удачно решить такую проблему позволяет знание различных структур данных.

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.

Одной из структур данных, которые обеспечивают максимальную экономию памяти являются битовые поля.

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. Например, для увеличения пропускной способности канала при передаче информации по сети или для уменьшения размера информации при хранении.

Данная структура данных позволяет компактно реализовать такой тип данных, как множество. Множество - тип и структура данных в информатике, которая является реализацией математического объекта множество. Данные типа множество позволяют хранить ограниченное число значений определённого типа без определённого порядка. Повторение значений, как правило, недопустимо.

Цель данной работы — разработка структуры данных для хранения множеств с использованием битовых полей

# Постановка задачи

Выполнение работы предполагает решение следующих задач:

1. Реализация класса битового поля TBitField.
2. 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 у пользователя запрашивается верхняя граница целых значений maxElem, среди которых в дальнейшем будут найдены все простые числа. Затем происходит поиск и подсчет простых чисел Затем на экран выводится множество некратных чисел (битовая строка простых чисел) и простые числа от 0 до maxElem.

Далее работа программы различается для классов битового поля и множества.

* Использование Битовых полей.

Пользователю предлагается самостоятельно ввести битовую строку. Для этого необходимо задать размер битовой строки k. Затем ввести последовательность битов длинной k+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.

* Использование класса множества.

Пользователю предлагается самостоятельно ввести неотрицательное множество целых чисел. Для этого необходимо задать наибольший элемент множества k > 0. Затем ввести набор чисел от 0 до k. Для завершения ввода нужно ввести -1 или число больше k.

В конце на экран будет выведено полученное множество и его битовая строка. На этом работа программы завершается.

# Руководство программиста

## Описание структуры программы

В решении содержатся следующие модули:

* Модуль tbitfield

Этот модуль содержит реализацию класса битового поля. Он включает в себя следующие файлы:

* tbitfield.h – в этом файле определен интерфейс класса битового поля TBitField
* tbitfield.cpp – в этом файле содержится реализация класса TBitField. Реализованы конструкторы, деструктор, перегружены битовые операторы, операторы вводы/вывода битовых полей и методы доступа к отдельным битам
* Модуль tset

Этот модуль содержит реализацию класса множества. Он включает в себя следующие файлы:

* tset.h – в этом файле определен интерфейс класса битового поля TSet
* tset.cpp – в этом файле содержится реализация класса TSet. Реализованы конструкторы, деструктор для класса, перегружены теоретико-множественные операторы, операторы ввода/вывода множеств и прописаны методы доступа к отдельным битам.
* Тесты для классов TBitField и TSet

Этот модуль содержит тесты, проверяющие работу каждого метода классов TBitField и TSet. Он включает в себя следующие файлы:

* test\_tbitfield.cpp – содержит тесты для класса TBitField
* test\_tset.cpp - содержит тесты для класса TSet
* Пример использования

В качестве примера использования класса битового поля и множества реализован алгоритм для поиска простых чисел, называемого "Решетом Эратосфена". Модуль включает в себя файл sample\_prime\_numbers.cpp.

## Описание структур данных

* Класс TBitField

1. В зоне private определены три поля и два метода:

* *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

1. В зоне public прописаны два конструктора, деструктор, методы доступа к битам и перегружены битовые операторы:

* *TBitField(int len)* – конструктор инициализации. Принимает длину битового поля.
* *TBitField(const TBitField &bf)* – конструктор копирования. Принимает ссылку на объект TBitField.
* *~TBitField()* – деструктор
* *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
* *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

1. Определены две дружественные функции воода/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

1. В зоне private определены два поля:

* *int maxPower* - максимальная мощность множества
* *TBitField bitField* - битовое поле для хранения характеристического вектора

1. В зоне public прописаны три конструктора, метод преобразования типа, методы доступа к битам и перегружены теоретико-множественные операторы:

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

1. Определены две дружественные функции воода/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 некоторые алгоритмы, используемые для реализации методов класса TBuildField.

* Формирование битового поля

На вход мы получаем множество натуральных чисел, которое необходимо представить в виде битовой строки. Каждый бит принимает значение «1», если элемент присутствует в множестве, или «0» если элемент в множестве отсутствует.

Удобнее всего будет хранить битовую строку в массиве. Но стоит учитывать одну особенность – нумерация бит в битовой строке происходит слева направо, также и нумерация элементов в массиве ведется слева направо, а вот биты элемента массива нумеруются справа налево (рис. 1)

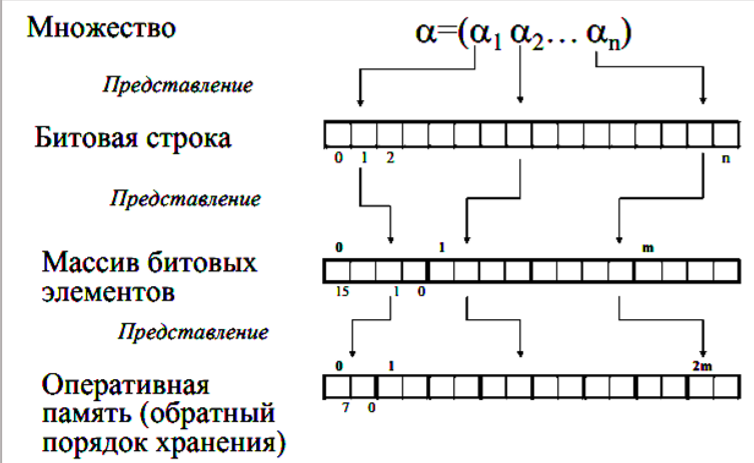


Рисунок . Представление битового поля.

* Методы GetMemIndex и GetMemMask

Для оптимизации основных методов класса TBuildField были прописаны два дополнительных метода:

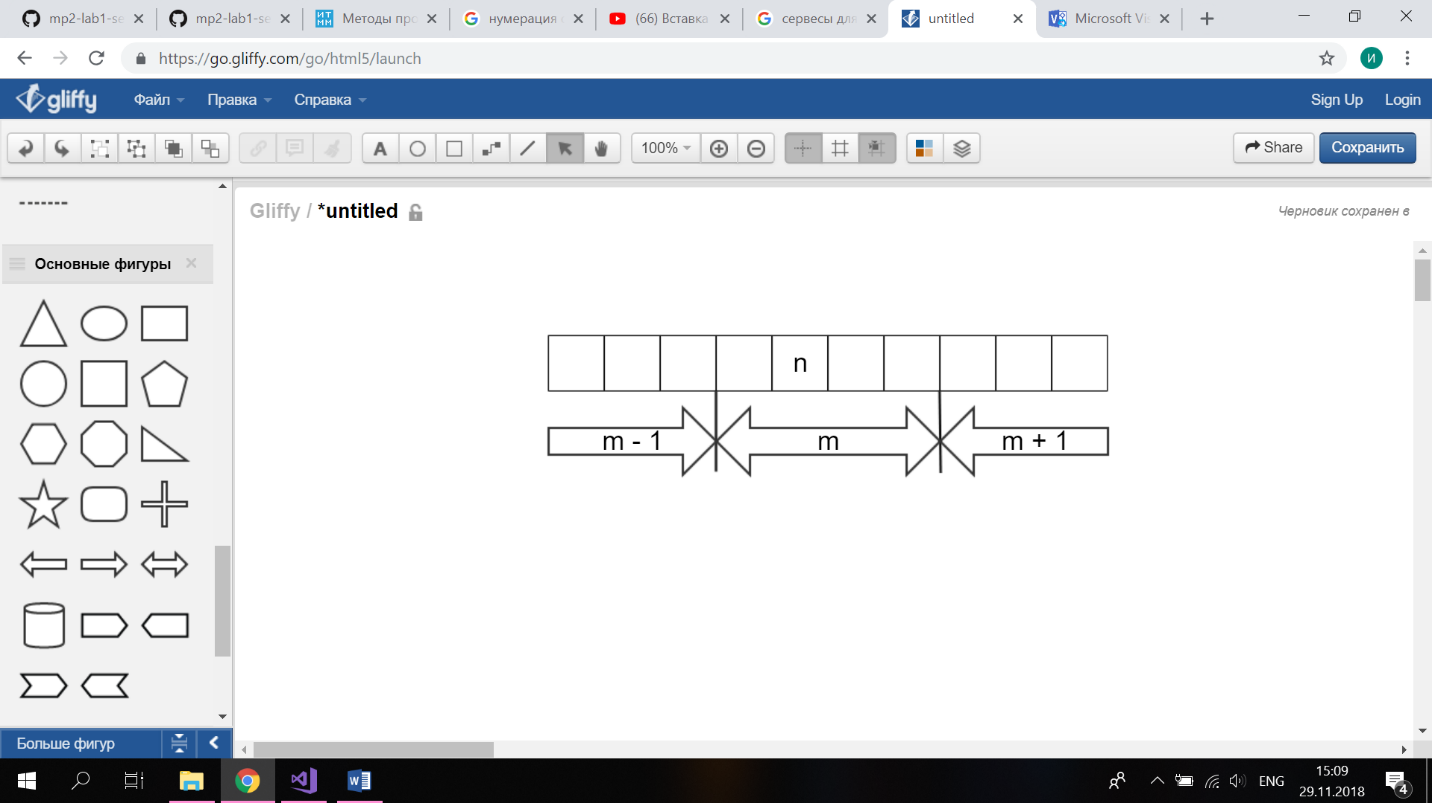
1. *GetMemIndex* – возвращает номер m элемента массива pMem, в котором хранится n-й бит (см. рисунок 2).

Рисунок . Представление массива pMem

Индекс элемента m определяется по следующей формуле:

, где uint – тип элементов множества.

1. *GetMemMask* – определяет битовую маску для -го бита.

Поскольку битовую маску можно рассматривать как двоичное представление числа, то данный метод возвращает целое число. Это число определяется выставлением единичного бита на позиции k, определяемой по следующей формуле:

Далее эти два метода применяются в реализации следующих методов класса TBuildField:

1. *SetBit(const int n)* – установить бит на позицию n. Чтобы установить бит на позицию n нужно использовать побитовую операцию «ИЛИ» между битами элемента из массива pMem, в котором хранится бит n, и между его битовой маской: .
2. *ClrBit(const int n)* – очистить n-й бит. Чтобы очистить n-й бит используем побитовую операцию «И» между битами элемента массива pMem, в котором находится n-й бит, и между дополнением его битовой маски: .
3. *GetBit(const int n)* – получает значение n-ого бита. Чтобы получить значение n-го бита используем побитовую операцию «И» между битами элемента массива pMem, в котором находится n-й бит, и между его побитовой маской:

# Заключение

В результате выполнения лабораторной работы была разработана структура хранения данных для множеств с использованием битовых полей.

Были реализованы класс битового поля (TBitField) и класс множества (TSet), в том числе и методы работы с этими классами.

Были реализованы тесты для проверки работоспособности вышеперечисленных классов на базе GoogleTest.

# Список литературы

1. Википедия. Статья «Битовое поле»: [https://ru.wikipedia.org/wiki/Битовое\_поле].
2. Васильев А.Н. Самоучитель С++ с примерами и задачами. -СПб.: Наука и Техника, 2016. -480с.
3. Гергель В.П. Методические материалы по курсу «Методы программирования 2», Нижний Новгород, 2015.