МИНИСТЕРСТВО НАУКИ И ВЫСШЕГООБРАЗОВАНИЯ

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**

**ЛАБОРАТОРНАЯ РАБОТА**

на тему:

**«Битовые поля и множества»**

**Выполнил:** студент группы 3822Б1ФИ2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Ляпин Н.А./

Подпись

**Проверил:** к.т.н, доцент каф. ВВиСП

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Кустикова В.Д./

Подпись

Нижний Новгород  
2023

**Содержание**

[Введение 3](#_Toc147915966)

[1 Постановка задачи 4](#_Toc147915967)

[2 Руководство пользователя 5](#_Toc147915968)

[2.1 Приложение для демонстрации работы битовых полей 5](#_Toc147915969)

[2.2 Приложение для демонстрации работы множеств 5](#_Toc147915970)

[2.3 «Решето Эратосфено» 5](#_Toc147915971)

[3 Руководство программиста 6](#_Toc147915972)

[3.1 Описание алгоритмов 6](#_Toc147915973)

[3.1.1 Битовые поля 6](#_Toc147915974)

[3.1.2 Множества 6](#_Toc147915975)

[3.1.3 «Решето Эратосфена» 6](#_Toc147915976)

[3.2 Описание программной реализации 6](#_Toc147915977)

[3.2.1 Описание класса TBitField 6](#_Toc147915978)

[3.2.2 Описание класса TSet 7](#_Toc147915979)

[Заключение 8](#_Toc147915980)

[Литература 9](#_Toc147915981)

[Приложения 10](#_Toc147915982)

[Приложение А. Реализация класса TBitField 10](#_Toc147915983)

[Приложение Б. Реализация класса TSet 10](#_Toc147915984)

# Введение

Одним из способов хранить большие данные является битовое поле.

Одно из основных преимуществ использования битовых полей заключается в экономии памяти. Вместо того чтобы хранить несколько переменных, каждая из которых использует все биты своего типа данных, мы можем объединить их в одну переменную, эффективно используя каждый бит для определенного состояния или флага. Таким образом, мы можем существенно сократить объем памяти, занимаемый такими переменными.

Кроме экономии памяти, битовые поля также могут улучшить производительность программы, особенно в ситуациях, где требуется быстрое и эффективное чтение и запись флагов или состояний. Битовые операции, такие как логическое И (&), логическое ИЛИ (|) и сдвиги, позволяют нам манипулировать отдельными битами в битовом поле, что может быть очень полезным в определенных сценариях программирования.

Множества, в свою очередь, являются наборами элементов, где каждый элемент может быть включен или исключен из множества. Множества можно представить с помощью битовых полей, где каждый бит соответствует наличию или отсутствию элемента в множестве. Такое представление позволяет быстро проверять принадлежность элемента к множеству, выполнять операции объединения, пересечения и разности множеств, используя битовые операции.

# Постановка задачи

Цель: Целью данной лабораторной работы является разработка своих Битового поля и Множества битого поля с помощью языка программирования C++. Основными целями, которые мы реализуем, являются операции с битовыми полями, включающие установку, сброс и проверку наличия определенного бита. В то же время, класс Множества будет предоставлять операции с множествами, такие как объединение, пересечение и разность.

Для достижения поставленных целей мы разработаем класс, который будет предоставлять функциональность для работы с отдельными битами внутри переменной. Мы реализуем основные операции с битовыми полями, чтобы можно было устанавливать, сбрасывать и проверять наличие битов.

Задачи:

- Спроектировать класс TBitField, который будет иметь возможность работы с битовым полем.

- Реализовать класс TBitField, реализовать его поля, конструкторы, методы и перегрузки операторов.

- Спроектировать класс TSet, который будет предоставлять функциональность для работы с множествами

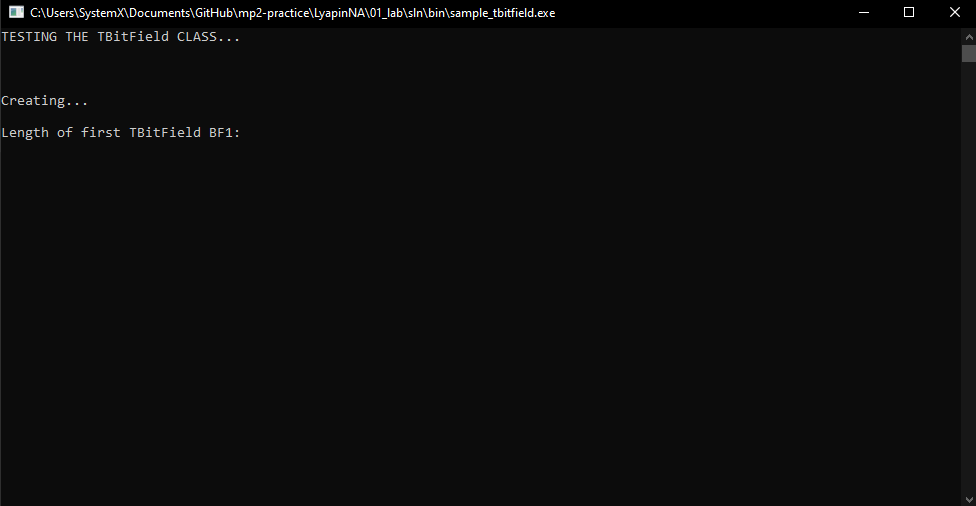
- Реализовать класс TSet, его поля, конструкторы, методы и перегрузки операторов.

- Проверка работоспособности этих классов с помощью тестов.

# Руководство пользователя

## Приложение для демонстрации работы битовых полей

1. Запустите приложение с названием sample\_tbitfield.exe. В результате появится окно, показанное ниже ().



1. Основное окно программы.
2. На первом этапе мы проверяем правильность создания битовых полей. Далее нам необходимо ввести длины для двух экземпляров битовых полей BF1 и BF2. Для примера, я введу значения 6 и 10 (рис. 2). Использование отрицательных значений гарантирует нам исключение.

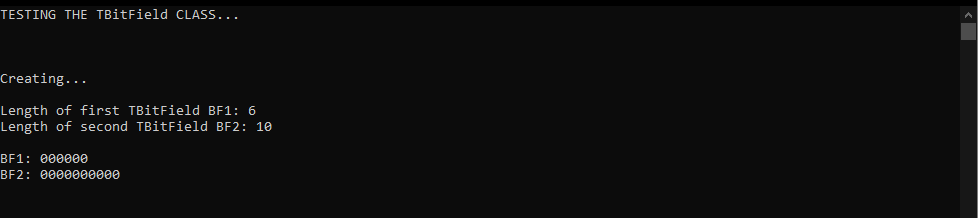


Рис. 2 Созданные в окне программы двух битовых полей BF1 и BF2.

1. На втором этапе мы проверяем работу методов, которые обеспечивают работу с битами поля. Среди этих методов: получение значение бита, установка и очищение бита, получение длины поля (рис. 3). В этом примере мы в первом битовом поле устанавливаем 4ый бит, а во втором, после предварительной установки всех битов поля, очищаем 2ой бит. Также здесь наглядно показана работа метода получения длины поля и перегрузка вывода для них.

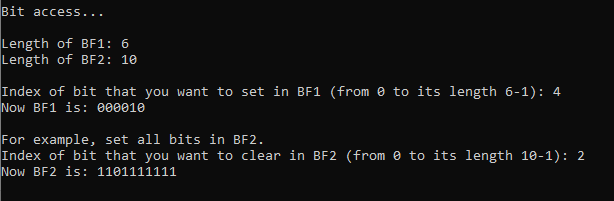


Рис. 3. Тестирование методов доступа к битам поля.

1. На заключительном этапе мы проверяем работу операторов битовых полей, таких как: операторы сравнения, отрицания, «или», «и» и присваивания. (рис. 4).\

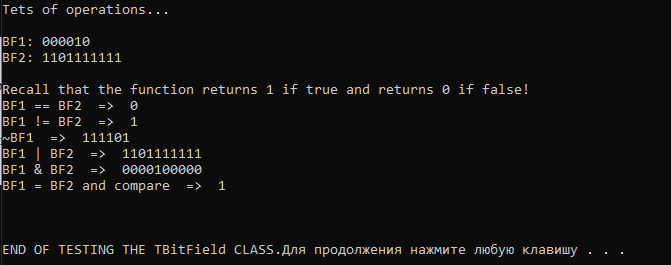


Рис. 4. Тестирование работы операторов битового поля.

## Приложение для демонстрации работы множеств

1. Запустите приложение с названием sample\_tset.exe. В результате появится окно, показанное ниже ().

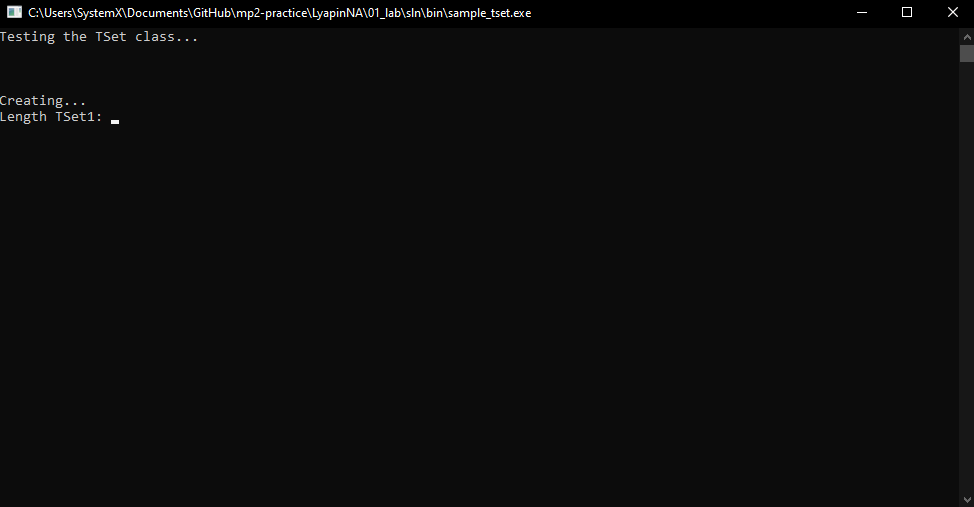


Рис. 1. Стартовое окно приложения

1. На первом этапе мы проверяем создание экземпляров множества. Создадим два экземпляра размера 6 и 10 соответственно. (рис. 2).

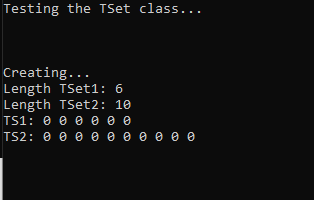


Рис. 2. Создавшиеся экземпляры TS1 и TS2.

1. На втором этапе мы проверяем работу методов класса, которые позволяют нам получить доступ к битовому полю. На картинке (рис. 3) мы видим включение элемента под индексом 4 в экземпляре TS1, удаление элемента под индексом 2 в экземпляре TS2 (после предварительного включения всех элементов), получение значение множества под индексом из TS1 под индексом 2 и получение мощности множества.

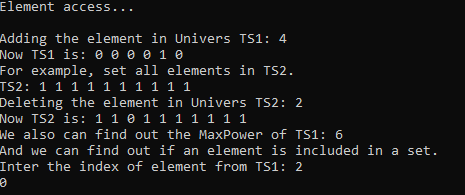


Рис. 3. Демонстрация работы методов множества.

1. На заключительном этапе мы проверяем работу перегрузок операторов класса

(рис. 4). Мы проверяем работу операторов сравнения, работу дополнения, сложения и умножения двух экземпляров классов, прибавления и вычитания элемента множества.

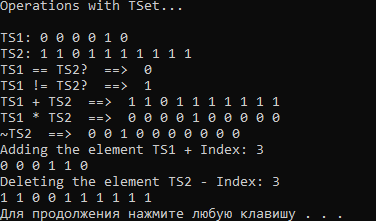


Рис. 4. Демонстрация работы операторов множества.

## «Решето Эратосфено»

# Руководство программиста

## Описание алгоритмов

### Битовые поля

Битовые поля реализуются на языке программирования C++ с помощью класса с полями, конструкторами, методами и перегрузками операторов, которые необходимы для правильной работы этой структуры данных.

В классе мы храним поля: размер битового поля, размер массива целочисленных элементов, благодаря которым реализуется битовое поле, а также размер этого массива элементов.

Пример битового поля: 0010111 – битовое поле, размера 7.

У битового поля реализованы операции сравнения, пересечения, обьединения, дополнения.

Операция сравнения выполняет поэлементное сравнение каждого бита в одной поле с каждым битом в другом поле, причем эти поля должны быть одного размера, иначе поля предварительно не могут быть равными.

Результатом операции пересечения является битовое поле, которое образовано из двух исходных. Алгоритм создания следующий: сравниваются каждые два бита из двух полей с одним индексом, на это же место в результирующем поле устанавливается 1, если хотя бы один их предыдущих элементов поля был равен 1, и 0, если оба элемента на этом индексе из двух полей равны 0.

Результатом операции обьединения является битовое поле, которое также, как и в случае операции пересечения, образовано из двух исходных. Алгоритм также схож с алгоритмом пересечения, но при сравнении элементов полей для того чтобы в результирующем поле элемент был равен 1 необходимо, чтобы на этом же индексе оба элемента в этих двух полях были равны 1, в остальных случаях в результирующем поле на этом месте стоит 0.

Дополнение

### Множества

### «Решето Эратосфена»

## Описание программной реализации

### Описание класса TBitField

class TBitField

{

private:

int BitLen; // длина битового поля - макс. к-во битов

TELEM \*pMem; // память для представления битового поля

int MemLen; // к-во эл-тов Мем для представления бит.поля

// методы реализации

int GetMemIndex(const int n) const;

TELEM GetMemMask (const int n) const;

public:

TBitField(int len);

TBitField(const TBitField &bf);

…

};

Назначение: представление битового поля.

Поля:

BitLen – длина битового поля – максимальное количество битов.

pMem – память для представления битового поля.

MemLen – количество элементов для представления битового поля.

Методы:

int GetMemIndex(const int n) const;

Назначение: получение индекса элемента в памяти…

Входные параметры:

n – номер бита.

Выходные параметры:

Номер элемента в памяти.

### Описание класса TSet

# Заключение

Что сделано?

# Литература

1. Название страницы [ссылка].
2. Фамилия И.О. Название. – Издательство, год. – количество страниц с
3. .

# Приложения

## Приложение А. Реализация класса TBitField

## Приложение Б. Реализация класса TSet