

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
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

\_\_\_\_\_/Табунов И. Д./  
Подпись

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

\_\_\_\_\_/Кустикова В.Д./  
Подпись

Нижний Новгород  
2023

# Содержание

|   |    |
|---|----|
| Введение.....   | 3  |
| 1 Постановка задачи.....                                  | 4  |
| 2 Руководство пользователя.....                           | 5  |
| 2.1 Приложение для демонстрации работы битовых полей..... | 5  |
| 2.2 Приложение для демонстрации работы множеств.....      | 6  |
| 2.3 «Решето Эратосфена».....                              | 7  |
| 3 Руководство программиста .....                          | 9  |
| 3.1 Описание алгоритмов .....                             | 9  |
| 3.1.1 Битовые поля .....                                  | 9  |
| 3.1.2 Множества .....                                     | 11 |
| 3.1.3 «Решето Эратосфена» .....                           | 12 |
| 3.2 Описание программной реализации .....                 | 13 |
| 3.2.1 Описание класса TBitField .....                     | 13 |
| 3.2.2 Описание класса Tset .....                          | 17 |
| Заключение .....  | 21 |
| Литература .....  | 22 |
| Приложения .....  | 23 |
| Приложение А. Реализация класса TBitField .....           | 23 |
| Приложение Б. Реализация класса TSet.....                 | 26 |

## **Введение**

В современном программировании и анализе данных часто возникает необходимость работы с большими объемами информации и эффективным представлением данных. Битовые поля и множества являются одним из инструментов, которые позволяют компактно хранить и манипулировать множествами элементов. Битовые поля и множества позволяют представить множество элементов в виде битовых векторов, где каждый бит соответствует наличию или отсутствию элемента в множестве. Такое представление позволяет существенно сократить объем памяти, занимаемый множеством, и ускорить операции над ними.

# 1 Постановка задачи

**Цель работы:** изучение и практическое применение концепции битовых полей и множеств.

**Задачи:**

1. Изучить теоретические основы битовых полей и множеств.
2. Разработать программу, реализующую операции над битовыми полями и множествами.
3. Провести эксперименты с различными наборами данных.
4. Проанализировать полученные результаты и сделать выводы о преимуществах и ограничениях использования битовых полей и множеств.

## 2 Руководство пользователя

### 2.1 Приложение для демонстрации работы битовых полей

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

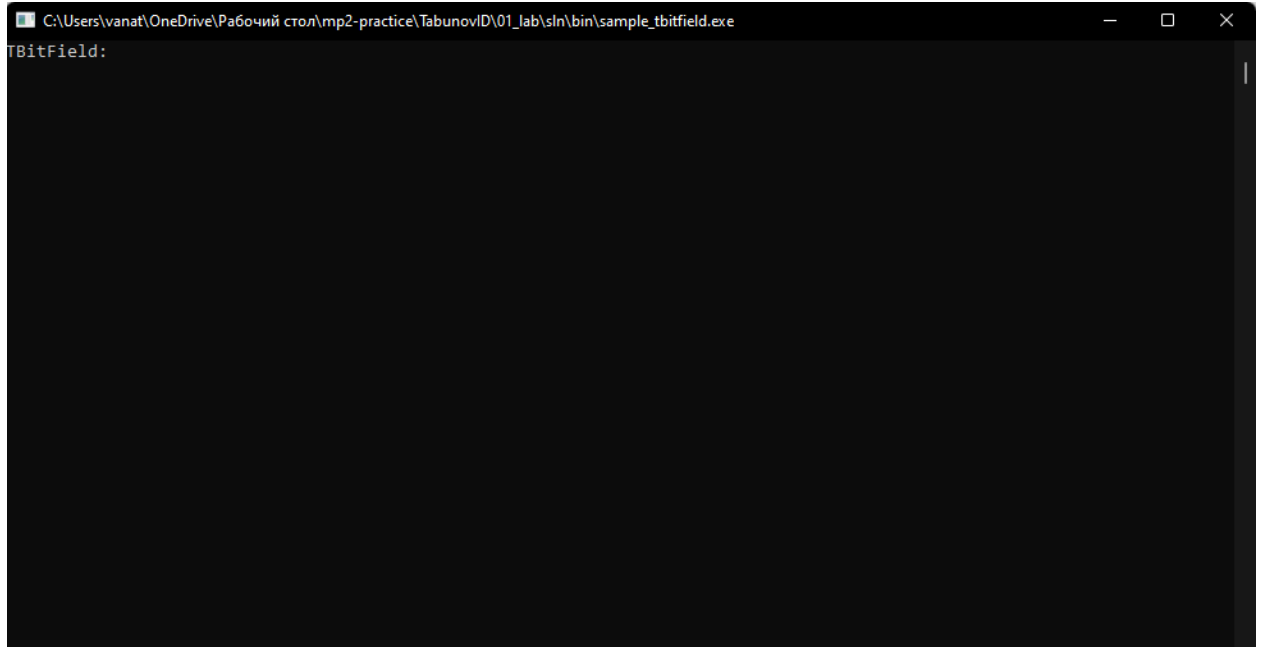


Рис. 1. Основное окно программы

2. Затем вам будет предложено ввести 2 битовых поля длины 8 (Рис. 2).

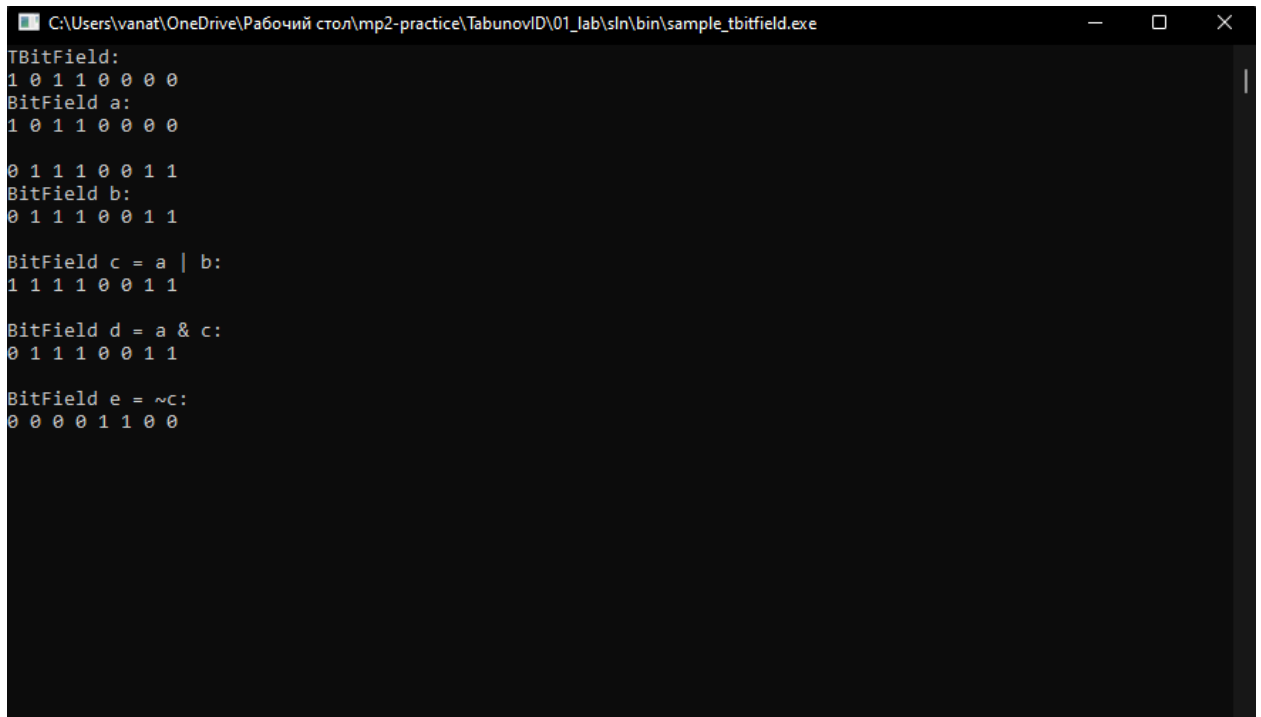
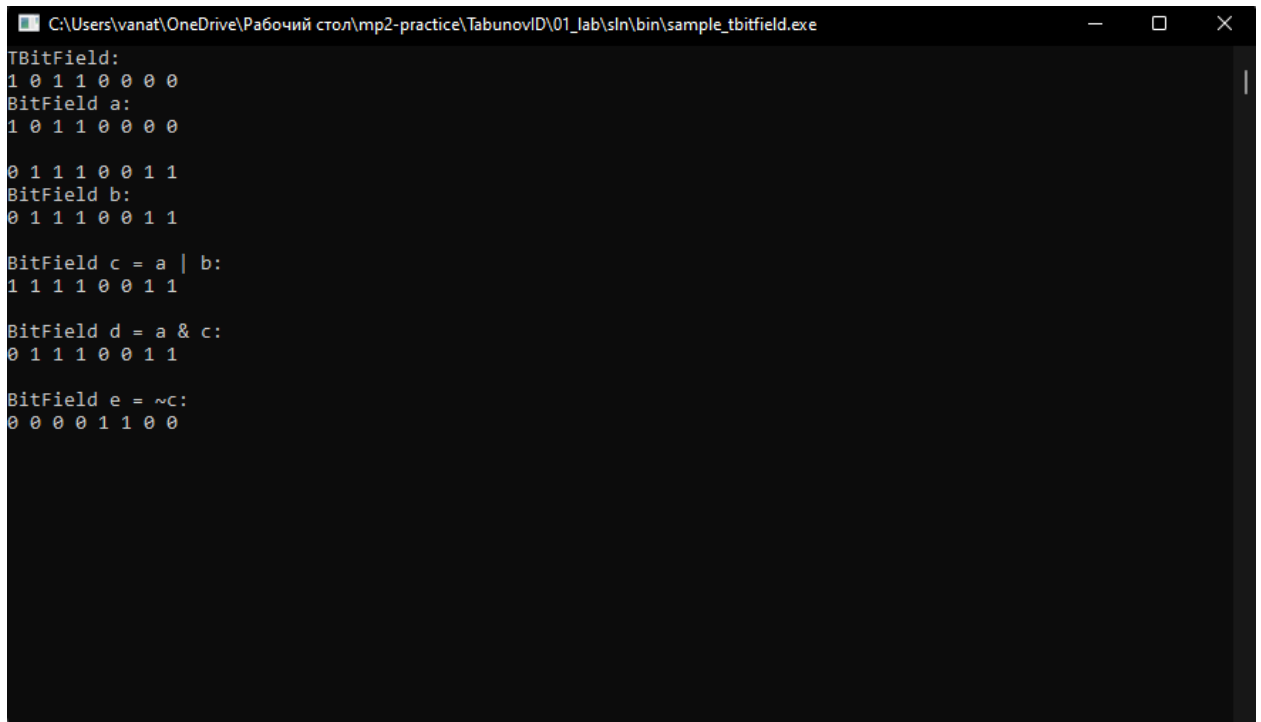


Рис. 2. Ввод битовых полей

3. После ввода множества нулей и /или единиц, будет выведены результаты соответствующих операций и функций (Рис. 3).



```
C:\Users\vanat\OneDrive\Рабочий стол\mp2-practice\Tabunov\ID\01_lab\sln\bin\sample_tbitfield.exe
TBitField:
1 0 1 1 0 0 0 0
BitField a:
1 0 1 1 0 0 0 0

0 1 1 1 0 0 1 1
BitField b:
0 1 1 1 0 0 1 1

BitField c = a | b:
1 1 1 1 0 0 1 1

BitField d = a & c:
0 1 1 1 0 0 1 1

BitField e = ~c:
0 0 0 0 1 1 0 0
```

Рис. 3. Результат тестирования функций класса TBitField

## 2.2 Приложение для демонстрации работы множеств

1. Запустите приложение с названием sample\_tset.exe. В результате появится окно, показанное ниже (Рис. 4).

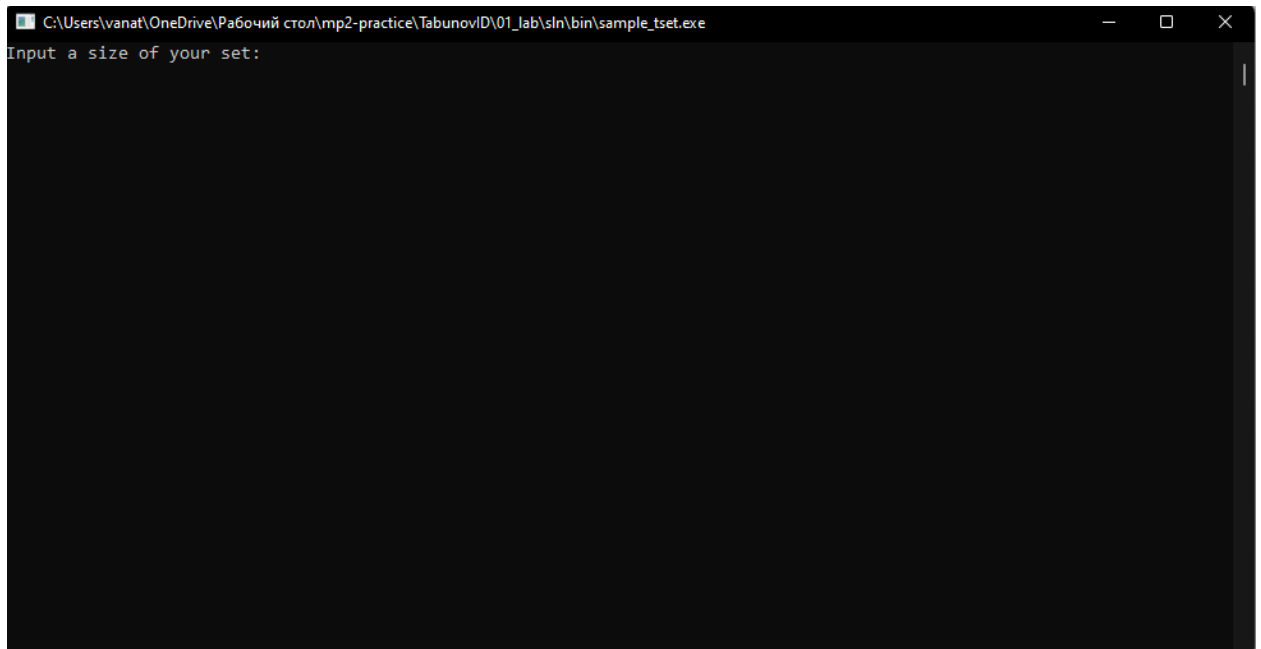
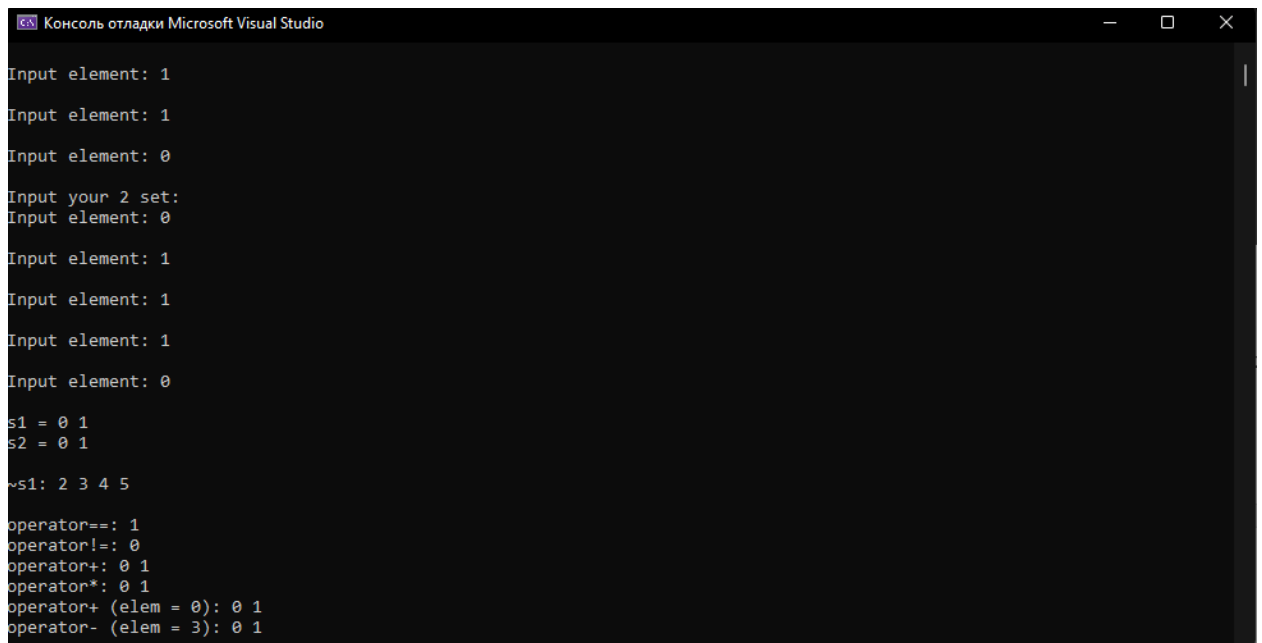


Рис. 4. Основное окно программы

2. Затем вам будет предложено вести свое множество. Необходимо ввести сначала количество чисел в множестве, программа будет ждать N чисел. После ввода

множества нулей и /или единиц, будет выведены результаты соответствующих операций и функций (Рис. 5).



```
Консоль отладки Microsoft Visual Studio

Input element: 1
Input element: 1
Input element: 0
Input your 2 set:
Input element: 0
Input element: 1
Input element: 1
Input element: 1
Input element: 0

s1 = 0 1
s2 = 0 1

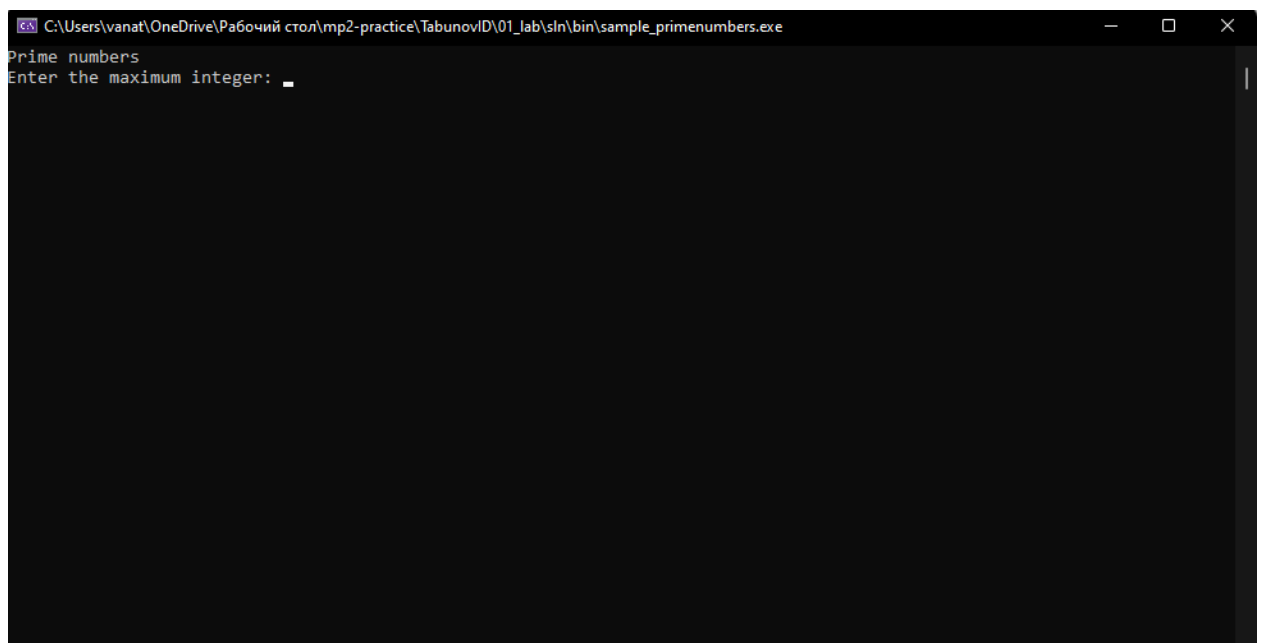
~s1: 2 3 4 5

operator==: 1
operator!=: 0
operator+: 0 1
operator*: 0 1
operator+ (elem = 0): 0 1
operator- (elem = 3): 0 1
```

Рис. 5. Результат тестирования класса **TSet**

## 2.3 «Решето Эратосфена»

1. Запустите приложение с названием sample\_tset.exe. В результате появится окно, показанное ниже (Рис. 6).

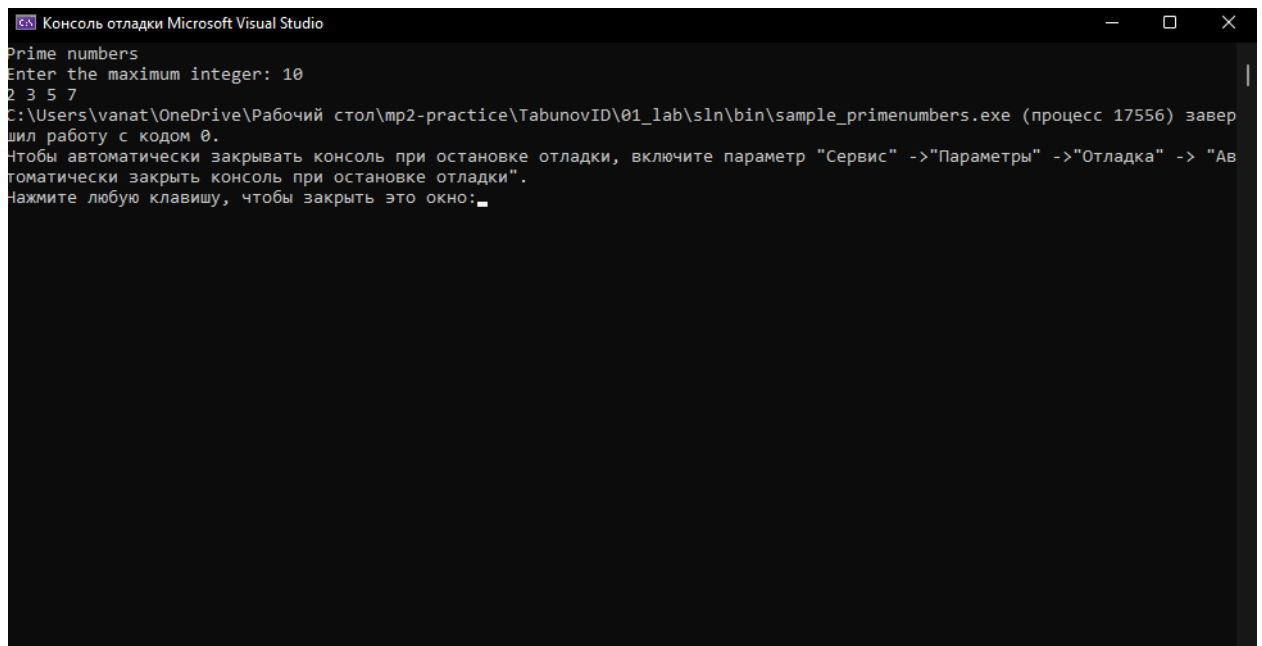


```
C:\Users\vanat\OneDrive\Рабочий стол\mp2-practice\TabunovID\01_lab\sln\bin\sample_primenumbers.exe

Prime numbers
Enter the maximum integer: _
```

Рис. 6. Основное окно программы

2. Затем вам будет нужно ввести целое положительное число. После чего программа выведет простые числа на отрезке до введенного числа и их количество (Рис. 7).



```
Консоль отладки Microsoft Visual Studio
Prime numbers
Enter the maximum integer: 10
2 3 5 7
C:\Users\vanat\OneDrive\Рабочий стол\mp2-practice\TabunovID\01_lab\sln\bin\sample_primenumbers.exe (процесс 17556) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно: 
```

Рис. 7. Результат работы алгоритма «Решето Эратосфена»



## 3 Руководство программиста

### 3.1 Описание алгоритмов

#### 3.1.1 Битовые поля

Битовые поля представляют собой набор чисел, каждый бит которых интерпретируется элементом, равным индексом бита. Битовые поля обеспечивают удобный доступ к отдельным битам данных. Они позволяют формировать объекты с длиной, не кратной 8 битам, что в свою очередь позволяет экономить память, более плотно размещая данные.

Битовое поле хранится в виде класса массива беззнаковых целых чисел, каждое из которых имеет размер 32 бита, максимальный элемент(количество битов), количество беззнаковых целых чисел, которые образуют битовое поле.

Пример битового поля длины 6:

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|

Битовое поле поддерживает операции объединения, пересечения, дополнение (отрицание), сравнения, ввода и вывода.

#### Операция объединения:

Входные данные: битовое поле.

Выходные данные: битовое поле, каждый бит которого равен 1, если он есть хотя бы в 1 битовом поле, которые объединяем, и 0 в противном случае.

Пример:

|     |   |   |   |   |   |   |
|-----|---|---|---|---|---|---|
| A   | 0 | 1 | 1 | 0 | 1 | 1 |
| B   | 1 | 0 | 0 | 0 | 1 | 1 |
| A B | 1 | 1 | 1 | 0 | 1 | 1 |

#### Операция пересечения:

Входные данные: битовое поле.

Выходные данные: битовое поле, каждый бит которого равен 1, если он есть в обоих в 1 битовом поле, которые объединяем, и 0 в противном случае.

Пример:

|     |   |   |   |   |   |   |
|-----|---|---|---|---|---|---|
| A   | 0 | 1 | 1 | 0 | 1 | 1 |
| B   | 1 | 0 | 0 | 0 | 1 | 1 |
| A&B | 0 | 0 | 0 | 0 | 1 | 1 |

### Операция дополнения (отрицания):

Входные данные: отсутствуют.

Выходные данные: битовое поле каждый бит которого равен 0, если он есть исходном классе, и 1 в противном случае.

Пример:

|    |   |   |   |   |   |   |
|----|---|---|---|---|---|---|
| A  | 0 | 1 | 1 | 0 | 1 | 1 |
| ~A | 1 | 0 | 0 | 1 | 0 | 0 |

### Добавление и удаление бита:

Входные данные: индекс (номер) бита.

Выходные данные: битовое поле, с добавленным (удаленным) битом.

Пример:

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|

Результат добавления 1 бита:

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| B | 1 | 1 | 1 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|

Результат удаления 2 бита:

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| C | 0 | 0 | 1 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|

Замечание: Введенный бит должен быть больше 0 и меньше длины битового поля.

### Операции сравнения

Операция равенства выведет 1, если два битовых поля равны, или каждые их биты совпадают, 0 в противном случае. Операция, обратная операции равенства, выведет 0, если хотя бы два бита совпадают, 1 в противном случае.

### 3.1.2 Множества

Множества представляют собой набор целых положительных чисел. В данной лабораторной работе множество реализовано при помощи битового поля, соответственно каждый бит которых интерпретируется элементом, равным индексом бита. 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. Создание множества через битовые поля может сильно сократить использование памяти.

Множество можно представить в виде битового поля.

Пример множества максимальной длины 5 с его битовым полем:

$$A = \{1, 2, 4, 5\}$$

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|

Множество поддерживает операции объединения, пересечения, дополнение (отрицание), сравнения, ввода и вывода.

#### Операция объединения с множеством:

Входные данные: множество.

Выходные данные: множество, равное объединению множеств, содержащее все уникальные элементы из двух множеств.

Пример:

$$A = \{1, 2, 3\}$$

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|

$$B = \{1, 3, 5\}$$

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| B | 0 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|

Результат объединения множеств A|B:

$$A+B = \{1, 2, 3, 5\}$$

|     |   |   |   |   |   |   |
|-----|---|---|---|---|---|---|
| A+B | 0 | 1 | 1 | 1 | 0 | 1 |
|-----|---|---|---|---|---|---|

#### Операция пересечения с множеством:

Операция пересечения множеств, содержащее все уникальные элементы, содержащиеся в обоих множествах.

Входные данные: множество.

Выходные данные: множество, равное пересечению множеств, содержащее все уникальные элементы из двух множеств.

Пример:

$$A = \{1, 2, 3\}$$

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|

$$B = \{1, 3, 5\}$$

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| B | 0 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|

Результат пересечения множеств  $A \cap B$ :

$$A \cap B = \{1, 2, 3, 5\}$$

|            |   |   |   |   |   |   |
|------------|---|---|---|---|---|---|
| $A \cap B$ | 0 | 1 | 1 | 1 | 0 | 1 |
|------------|---|---|---|---|---|---|

**Операция дополнения (отрицания):**

Входные данные: отсутствуют.

Выходные данные: множество, равное дополнению исходного множества.

Пример:

$$A = \{1, 2, 3\}$$

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|

Результат объединения множеств  $\sim A$ :

$$\sim A = \{0, 4, 5\}$$

|          |   |   |   |   |   |   |
|----------|---|---|---|---|---|---|
| $\sim A$ | 1 | 0 | 0 | 0 | 1 | 1 |
|----------|---|---|---|---|---|---|

### 3.1.3 «Решето Эратосфена»

Входные данные: целое положительное число (Далее N).

Выходные данные: множество простых чисел.

Алгоритм:

1. Создаем множество целых чисел от 1 до N.
2. Перебираем все числа от 2 до N включительно.
3. Если число есть в множестве, то убираем из множества все кратные ему числа.
4. Если числа не закончились, то переходим к 3 шагу, иначе заканчиваем.

После чего выводятся все числа от 1 до N с идентификатором (0, если оно не простое, 1 если простое), затем выводятся все простые числа и их количество.

## 3.2 Описание программной реализации

### 3.2.1 Описание класса TBitField

```
class TBitField
{
private:
    int BitLen;
    TELEM *pMem;
    int MemLen;
    int GetMemIndex(const int n) const;
    TELEM GetMemMask (const int n) const;
public:
    TBitField(int len);
    TBitField(const TBitField &bf);
    ~TBitField();

    // доступ к битам
    int GetLength(void) const;
    void SetBit(const int n);
    void ClrBit(const int n);
    int GetBit(const int n) const;

    int operator==(const TBitField &bf) const;
    int operator!=(const TBitField &bf) const;
    const TBitField& operator=(const TBitField &bf);
    TBitField operator|(const TBitField &bf);
    TBitField operator&(const TBitField &bf);
    TBitField operator~(void);

    friend istream &operator>>(istream &istr, TBitField &bf);
    friend ostream &operator<<(ostream &ostr, 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.

**TELEM GetMemMask (const int n) const;**

Назначение: Получение битовой маски

Входные параметры:

**n** – номер бита.

Выходные параметры:

Битовая маска типа.

**TBitField(int len) ;**

Назначение: конструктор с параметром, выделение памяти

Входные параметры:

**len** – длина битового поля.

Выходные параметры:

Отсутствуют.

**TBitField(const TBitField &bf) ;**

Назначение: конструктор копирования. Создание экземпляра класса на основе другого экземпляра.

Входные параметры:

**bf** – ссылка, адрес экземпляра класса, на основе которого будет создан другой.

Выходные параметры:

Отсутствуют.

**~TBitField() ;**

Назначение: деструктор. Отчистка выделенной памяти.

Входные и выходные параметры отсутствуют.

**int GetLength(void) const;**

Назначение: получение длины битового поля.

Входные параметры отсутствуют.

Выходные параметры: длина битового поля.

**void SetBit(const int n)**

Назначение: установить бит = 1.

Входные параметры:

**n** - номер бита, который нужно установить.

Выходные параметры отсутствуют.

**void ClrBit(const int n);**

Назначение: отчистить бит (установить бит = 0).

Входные параметры:

**n** - номер бита, который нужно отчистить .

Выходные параметры отсутствуют.

**int GetBit(const int n) const;**

Назначение: вывести бит (узнать бит).

Входные параметры:

**n** - номер бита, который нужно вывести (узнать).

Выходные параметры: бит (1 или 0, в зависимости есть установлен он, или нет).

**int operator==(const TBitField &bf) const;**

Назначение: оператор сравнения. Сравнить на равенство 2 битовых поля.

Входные параметры:

**bf** – битовое поле, с которым мы сравниваем.

Выходные параметры: 1 или 0, в зависимости равны они, или нет соответственно.

**int operator!=(const TBitField &bf) const;**

Назначение: оператор сравнения. Сравнить на равенство 2 битовых поля.

Входные параметры:

**bf** – битовое поле, с которым мы сравниваем.

**Выходные** параметры: 1 или 0, в зависимости равны они, или нет соответственно.

```
const TBitField& operator=(const TBitField &bf);
```

Назначение: оператор присваивания.

Входные параметры:

**bf** – битовое поле, которое мы присваиваем

Выходные параметры: ссылка на присвоенный экземпляр класса.

```
TBitField operator|(const TBitField &bf);
```

Назначение: оператор побитового «ИЛИ».

Входные параметры:

**bf** – битовое поле.

Выходные параметры: экземпляр класса.

```
TBitField operator&(const TBitField &bf);
```

Назначение: оператор побитового «И».

Входные параметры:

**bf** – битовое поле, с которым мы сравниваем.

Выходные параметры: экземпляр класса.

```
TBitField operator~(void);
```

Назначение: оператор инверсии.

Входные параметры отсутствуют.

Выходные параметры: Экземпляр класса.

```
friend istream &operator>>(istream &istr, TBitField &bf);
```

Назначение: оператор ввода из консоли.

Входные параметры:

**istr** – буфер консоли.

**bf** – класс, который нужно ввести из консоли.

Выходные параметры:

Ссылка на буфер (поток) **istr**.

```
friend ostream &operator<<(ostream &ostr, const TBitField  
&bf);
```

Назначение: оператор вывода из консоли.

Входные параметры:



**istr** – буфер консоли.

**bf** – класс, который нужно вывести в консоль.

Выходные параметры:

Ссылка на буфер (поток) **istr**.

### 3.2.2 Описание класса TSet

```
class TSet
{
private:
    int MaxPower;
    TBitField BitField;
public:
    TSet(int mp);
    TSet(const TSet &s);
    TSet(const TBitField &bf);
    operator TBitField();
    int GetMaxPower(void) const;
    void InsElem(const int Elem);
    void DelElem(const int Elem);
    int IsMember(const int Elem) const;
    int operator== (const TSet &s) const;
    int operator!= (const TSet &s) const;
    const TSet& operator=(const TSet &s);
    TSet operator+ (const int Elem);
    TSet operator- (const int Elem);
    TSet operator+ (const TSet &s);
    TSet operator* (const TSet &s);
    TSet operator~ (void);
    friend istream &operator>>(istream &istr, TSet &bf);
    friend ostream &operator<<(ostream &ostr, const TSet &bf);
};
```

Назначение: представление множества чисел.

Поля:

**MaxPower** – максимальный элемент множества.

**BitField** – экземпляр битового поля, на котором реализуется множество.

Методы:

**TSet(int mp);**

Назначение: конструктор с параметром, выделение памяти.

Входные параметры:

**mp** – максимальный элемент множества.

Выходные параметры:

Отсутствуют.

**TSet(const TSet &s) ;**

Назначение: конструктор копирования. Создание экземпляра класса на основе другого экземпляра .

Входные параметры:

**s** – ссылка, адрес экземпляра класса, на основе которого будет создан другой.

Выходные параметры:

Отсутствуют.

**~TSet() ;**

Назначение: деструктор. Отчистка выделенной памяти.

Входные и выходные параметры отсутствуют.

**int GetMaxPower(void) const;**

Назначение: получение максимального элемента множества.

Входные параметры отсутствуют.

Выходные параметры: максимальный элемент множества.

**void InsElem(const int Elem)**

Назначение: добавить элемент в множество.

Входные параметры:

**Elem** - добавляемый элемент.

Выходные параметры отсутствуют.

**void DelElem(const int Elem)**

Назначение: удалить элемент из множества.

Входные параметры:

**Elem** - удаляемый элемент.

Выходные параметры отсутствуют.

**int IsMember(const int Elem) const;**

Назначение: узнать, есть ли элемент в множестве.

Входные параметры:

**Elem** - элемент, который нужно проверить на наличие.

Выходные параметры: 1 или 0, в зависимости есть элемент в множестве, или нет.

```
int operator==(const TSet &s) const;
```

Назначение: оператор сравнения. Сравнить на равенство 2 множества.

Входные параметры:

**s** – битовое поле, с которым мы сравниваем.

Выходные параметры: 1 или 0, в зависимости равны они, или нет соответственно.

```
int operator!=(const TSet &s) const;
```

Назначение: оператор сравнения. Сравнить на равенство 2 множества.

Входные параметры:

**s** – битовое поле, с которым мы сравниваем.

Выходные параметры: 0 или 1, в зависимости равны они, или нет соответственно.

```
const TSet& operator=(const TSet &s) ;
```

Назначение: оператор присваивания.

Входные параметры:

**s** – множество , которое мы присваиваем.

Выходные параметры: ссылка на экземпляр класса.

```
TSet operator+(const TSet &bf) ;
```

Назначение: оператор объединения множеств.

Входные параметры:

**s** - множество.

Выходные параметры: экземпляр класса.

```
TSet operator*(const TSet &bf) ;
```

Назначение: оператор пересечения множеств .

Входные параметры:

**s** - множество.

Выходные параметры: экземпляр класса.

}

```
TBitField operator~(void) ;
```

Назначение: оператор дополнение до Универса.

Входные параметры отсутствуют .

Выходные параметры: Экземпляр класса, каждый элемент которого равен `{~*this}`, т.е. если `i` элемент исходного экземпляра будет равен будет находится в множестве, то на выходе его не будет, и наоборот.

```
friend istream &operator>>(istream &istr, TSet &s);
```

Назначение: оператор ввода из консоли.

Входные параметры:

**istr** – буфер консоли.

**s** – класс, который нужно ввести из консоли.

Выходные параметры:

Ссылка на буфер (поток) **istr**.

```
friend ostream &operator<<(ostream &ostr, const TSet &s);
```

Назначение: оператор вывода из консоли.

Входные параметры:

**istr** – буфер консоли.

**s** – класс, который нужно вывести в консоль.

Выходные параметры: Ссылка на буфер (поток) **istr**

```
TSet operator+(const int Elem);
```

Назначение: оператор объединения множества и элемента. Данный оператор аналогичен метод добавления элемента в множество.

Входные параметры:

**Elem** - число

Выходные параметры: исходный экземпляр класса, содержащий **Elem**.

```
TSet operator+(const int Elem);
```

Назначение: оператор объединения множества и элемента. Данный оператор аналогичен методу удаления элемента из множества.

Входные параметры:

**Elem** - число

Выходные параметры: исходный экземпляр класса, не содержащий **Elem**.

## Заключение

В ходе выполнения работы "Битовые поля и множества" были изучены и практически применены концепции битовых полей и множеств.

Были достигнуты следующие результаты:

1. Изучены теоретические основы битовых полей и множеств.
2. Разработана программа, реализующая операции над битовыми полями и множествами. В ходе экспериментов была оценена эффективность работы этих операций и сравнена с другими подходами. Результаты показали, что использование битовых полей и множеств позволяет существенно сократить объем памяти и ускорить операции над множествами.
3. Проанализированы полученные результаты и сделаны выводы о преимуществах и ограничениях использования битовых полей и множеств. Оказалось, что эти структуры данных особенно полезны при работе с большими объемами данных, где компактность представления и эффективность операций являются ключевыми факторами.

## **Литература**

1. Битовое поле [[https://ru.wikipedia.org/wiki/Битовое\\_поле](https://ru.wikipedia.org/wiki/Битовое_поле)].

# Приложения

## Приложение А. Реализация класса TBitField

```
#include "tbitfield.h"
#define size 32 //sizeof(ui)
TBitField::TBitField(int len)
{
    if (len <= 0)
        throw "len_is_LEq_zero!";
    BitLen = len;
    MemLen = (BitLen - 1) / (size) + 1;
    pMem = new TELEM[MemLen];

    // зануление битов (может из-за поведения заполнить "мусором")
    for (int i = 0; i < MemLen; ++i)
    {
        pMem[i] = 0;
    }
}

TBitField::TBitField(const TBitField& bf) // конструктор копирования
{
    BitLen = bf.BitLen;
    MemLen = bf.MemLen;
    pMem = new TELEM[MemLen];
    for (int i = 0; i < MemLen; ++i)
    {
        pMem[i] = bf.pMem[i];
    }
}

TBitField::~TBitField()
{
    delete[] pMem;
}

int TBitField::GetMemIndex(const int n) const // индекс Мем для бита n
{
    if (n < 0)
        throw "n_is_below_zero";
    return n / (size);
}

TELEM TBitField::GetMemMask(const int n) const // битовая маска для бита n
{
    if (n > BitLen)
        throw "overload_n";
    if (n < 0)
        throw "n_is_below_zero";
    TELEM Mask = 1u << (size - n % size - 1);
    return Mask;
}

// доступ к битам битового поля

int TBitField::GetLength(void) const // получить длину (к-во битов)
{
    return BitLen;
}

void TBitField::SetBit(const int n) // установить бит
```

```

{
    if (n > BitLen)
        throw "overload_n";
    if (n < 0)
        throw "n_is_below_zero";

    pMem[GetMemIndex(n)] |= GetMemMask(n);
}

void TBitField::ClrBit(const int n) // очистить бит
{
    if (n > BitLen)
        throw "overload_n";
    if (n < 0)
        throw "n_is_below_zero";
    pMem[GetMemIndex(n)] &= ~(GetMemMask(n));
}

int TBitField::GetBit(const int n) const // получить значение бита
{
    if (n > BitLen)
        throw "overload_n";
    if (n < 0)
        throw "n_is_below_zero";
    return (pMem[GetMemIndex(n)] & (GetMemMask(n))) >> (size - n % size -
1);
}

// битовые операции

const TBitField& TBitField::operator=(const TBitField& bf) // присваивание
{
    if (*this == bf) return *this;
    if (BitLen != bf.BitLen)
    {
        delete[] pMem;
        pMem = new TELEM[MemLen];
    }
    BitLen = bf.BitLen;
    MemLen = bf.MemLen;

    for (int i = 0; i < MemLen; ++i)
    {
        pMem[i] = bf.pMem[i];
    }
    return *this;
}

int TBitField::operator==(const TBitField& bf) const // сравнение
{
    if (BitLen != bf.GetLength())
        return 0;
    else
    {
        for (int i = 0; i < BitLen; ++i)
        {
            if (GetBit(i) != bf.GetBit(i))
                return 0;
        }
    }
    return 1;
}

```



```

int TBitField::operator!=(const TBitField& bf) const // сравнение
{
    return !(*this == bf);
}

TBitField TBitField::operator|(const TBitField& bf) // операция "или"
{
    const int maxlen = max(GetLength(), bf.GetLength());
    const int minlen = min(GetLength(), bf.GetLength());
    TBitField A(maxlen);
    if (GetLength() > bf.GetLength())
        A = *this;
    else
        A = bf;
    int i = 0;
    for (; i < minlen; ++i)
    {
        if (GetBit(i) || bf.GetBit(i))
            A.SetBit(i);
    }
    return A;
}

TBitField TBitField::operator&(const TBitField& bf) // операция "и"
{
    const int maxlen = max(GetLength(), bf.GetLength());
    const int minlen = min(GetLength(), bf.GetLength());
    TBitField A(maxlen);
    int i = 0;
    for (; i < minlen; ++i)
    {
        if (GetBit(i) && bf.GetBit(i))
            A.SetBit(i);
    }
    return A;
}

TBitField TBitField::operator~(void) // отрицание
{
    TBitField A(GetLength());
    for (int i = 0; i <= GetMemIndex(GetLength()); ++i)
    {
        A.pMem[i] = ~pMem[i];
    }
    return A;
}

// ввод/вывод

istream& operator>>(istream& istr, TBitField& bf) // ввод
{
    for (int i = 0; i < bf.GetLength(); ++i)
    {
        int val;
        istr >> val;
        if (val > bf.GetLength() || val < 0)
            throw "Wrong element ";
        bf.SetBit(val);
    }
    return itr;
}

```

```
ostream& operator<<(ostream& ostr, const TBitField& bf) // вывод
{
    for (int i = 0; i < bf.GetLength(); ++i)
    {
        ostr << bf.GetBit(i) << " ";
    }
    ostr << "\n";
    return ostr;
}
```

## Приложение Б. Реализация класса TSet

```
#include "tset.h"

TSet::TSet(int mp) : BitField(mp)
{
    MaxPower = mp;
}

// конструктор копирования
TSet::TSet(const TSet& s) : BitField(s.BitField)
{
    MaxPower = s.GetMaxPower();
}

// конструктор преобразования типа
TSet::TSet(const TBitField& bf) : BitField(bf)
{
    MaxPower = bf.GetLength();
}

TSet::operator TBitField()
{
    return BitField;
}

int TSet::GetMaxPower(void) const // получить макс. к-во эл-тов
{
    return MaxPower;
}

int TSet::IsMember(const int Elem) const // элемент множества?
{
    if (Elem < 0 && Elem >= MaxPower)
        throw "Out of Range";
    return BitField.GetBit(Elem);
}

void TSet::InsElem(const int Elem) // включение элемента множества
{
    if (Elem < 0 && Elem >= MaxPower)
        throw "Out of Range";
    return BitField.SetBit(Elem);
}

void TSet::DelElem(const int Elem) // исключение элемента множества
{
    if (Elem < 0 && Elem >= MaxPower)
        throw "Out of Range";
    return BitField.ClrBit(Elem);
}
```

```

// теоретико-множественные операции

const TSet& TSet::operator=(const TSet& s) // присваивание
{
    if (*this == s) return *this;
    MaxPower = s.GetMaxPower();
    BitField = TBitField(MaxPower);
    BitField = BitField | s.BitField; // bitfield = s.bitfield
    return *this;
}

int TSet::operator==(const TSet& s) const // сравнение
{
    if (MaxPower != s.GetMaxPower())
        return 0;

    for (int i = 0; i < MaxPower; ++i)
    {
        if (BitField.GetBit(i) != s.BitField.GetBit(i))
            return 0;
    }
    return 1;
}

int TSet::operator!=(const TSet& s) const // сравнение
{
    return !(*this == s);
}

TSet TSet::operator+(const TSet& s) // объединение
{
    TSet A(max(MaxPower, s.GetMaxPower()));
    A.BitField = BitField | s.BitField;
    return A;
}

TSet TSet::operator+(const int Elem) // объединение с элементом
{
    if (Elem < 0 && Elem > MaxPower)
        throw "Out of Range";
    TSet A(*this);
    A.BitField.SetBit(Elem);
    return A;
}

TSet TSet::operator-(const int Elem) // разность с элементом
{
    if (Elem < 0 && Elem > MaxPower)
        throw "Out of Range";
    TSet A(*this);
    A.BitField.ClrBit(Elem);
    return A;
}

TSet TSet::operator*(const TSet& s) // пересечение
{
    TSet A(max(MaxPower, s.GetMaxPower()));
    A.BitField = BitField & s.BitField;
    return A;
}

TSet TSet::operator~(void) // дополнение

```

```

{
    TSet A(MaxPower);
    A.BitField = ~BitField;
    return A;
}

// перегрузка ввода/вывода

istream& operator>>(istream& istr, TSet& s) // ввод
{
    const int x = s.MaxPower;
    for (int i = 0; i <= x; ++i)
    {
        int val; istr >> val;
        if (val > s.MaxPower)
            throw "Wrong element ";
        s.InsElem(val);
    }
    return istr;
}

ostream& operator<<(ostream& ostr, const TSet& s) // вывод
{
    const int x = s.MaxPower;
    for (int i = 0; i <= x; ++i)
    {
        ostr << s.IsMember(i) << " ";
    }
    return ostr;
}

```