

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(а): студент(ка) группы
3822Б1ФИ1

_____ / Стариков Н.В./
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	7
2.3 Приложение «решето Эратосфена»	9
3 Руководство программиста	10
3.1 Используемые алгоритмы	10
3.1.1 Битовые поля	10
3.1.2 Множества	10
3.1.3 Алгоритм «решето Эратосфена»	10
3.2 Описание классов.....	14
3.2.1 Класс TbitField	14
3.2.2 Класс TSet	17
Заключение	20
Литературы	21
Приложения	22
Приложение А. Реализация класса TBitField	22
Приложение Б. Реализация класса TSet.....	26

Введение

Битовое поле — это структура данных, состоящая из одного или нескольких соседних битов, выделенных для определенных целей, так что любой отдельный бит или группа битов в структуре может быть установлен или проверен. Битовые поля обеспечивают удобный доступ к отдельным битам данных. Они часто используются для управления флагами или настроек. Например, в программировании можно использовать битовые поля для представления различных состояний объекта или опций функции.

Явным образом понятие множества подверглось систематическому изучению во второй половине XIX века в работах немецкого математика Георга Кантора.

Активное применение аппарата теории множеств в современной науке приводит к необходимости создания соответствующих программных решений.

Множества поддерживают базовые операции, такие как объединение, пересечение и разность, что может быть очень удобным при работе с данными. Например, можно объединить два множества, чтобы получить новое множество, содержащее все элементы из обоих исходных множеств.

Битовые поля и множества могут быть использованы для оптимизации некоторых алгоритмов. Например, вы можете использовать битовые поля для представления булевого массива, что позволяет эффективно выполнять операции, такие как поиск, вставка и удаление элементов.

1 Постановка задачи

Цель: изучение битовых полей и множеств и получение навыков практического применения данных структур данных.

Задачи:

1. Изучить принцип работы с битовыми полями и множествами.
2. Разработать программу, которая будет реализовывать операции над битовыми полями и множествами.
3. Протестировать правильность выполнения программы на некоторых примерах.
4. Применить полученные результаты для алгоритма «решето Эратосфена».
5. Сделать выводы о проделанной работе.

2 Руководство пользователя

2.1 Приложение для демонстрации работы битовых полей

1. Запустить sample_tbitfield.exe. В результате появится следующее окно (рис. 1).

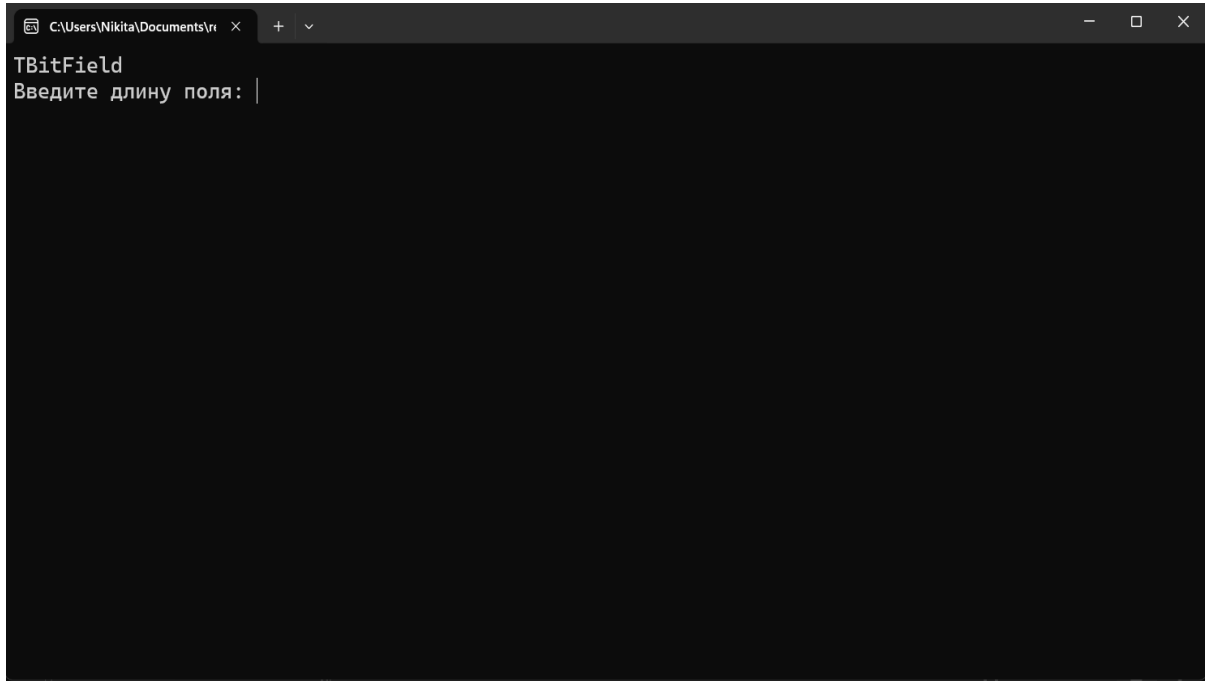


Рис. 1. Основное окно приложения битовых полей

2. Далее необходимо ввести длину битовых полей, например 6 (рис. 2) и (рис. 3).

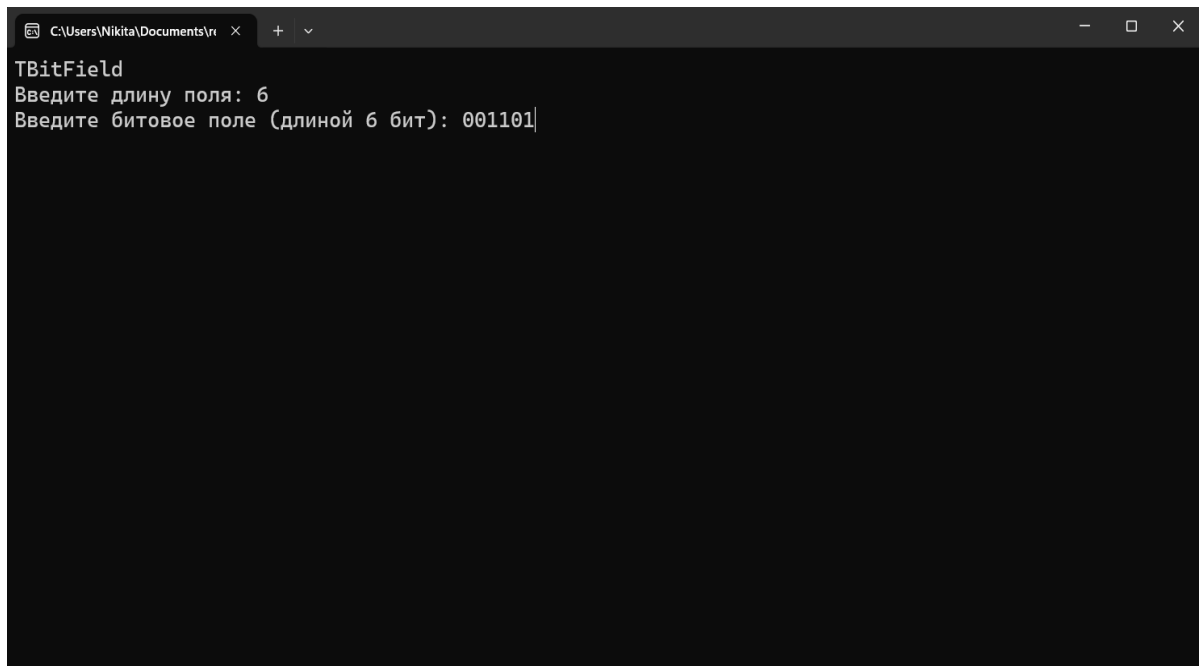


Рис. 2. Ввод длины первого битового поля

```
C:\Users\Nikita\Documents\tr x + v
TBitField
Введите длину поля: 6
Введите битовое поле (длиной 6 бит): 001101
Битовое поле 1: 001101
Введите битовое поле (длиной 6 бит): 111110
```

Рис. 3. Ввод длины второго битового поля

3. Затем получим результат работы программы с введенными битовыми полями (рис. 4).

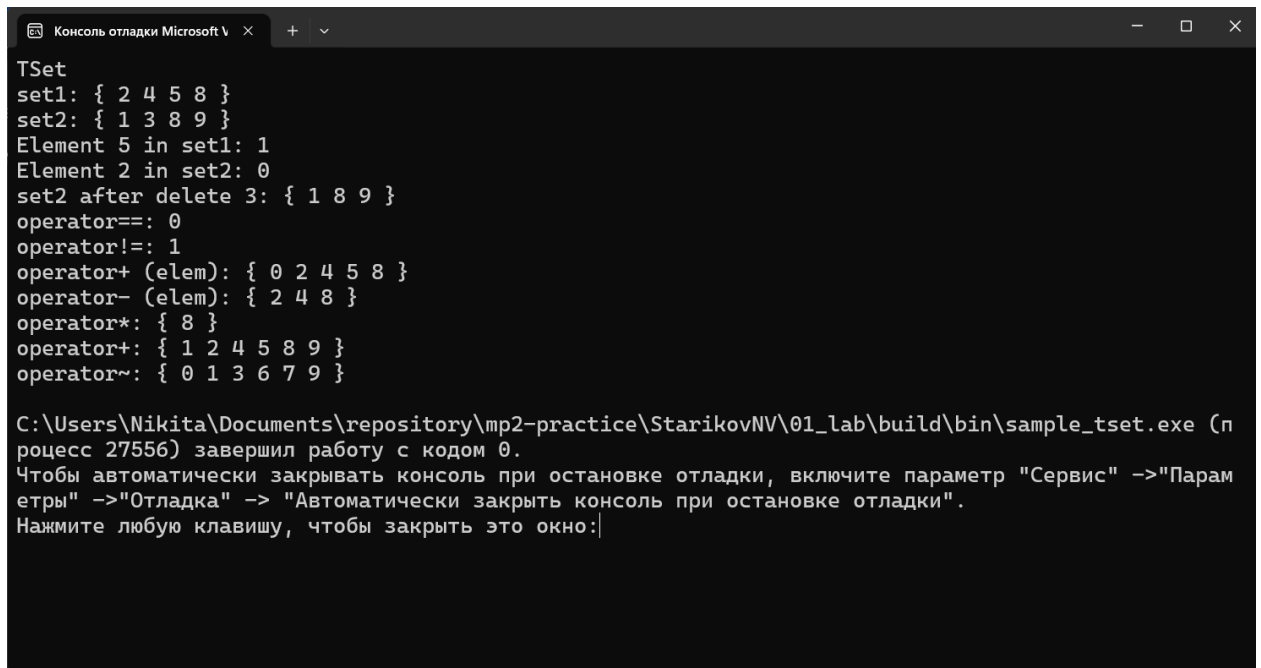
```
Консоль отладки Microsoft V x + v
TBitField
Введите длину поля: 6
Введите битовое поле (длиной 6 бит): 001101
Битовое поле 1: 001101
Введите битовое поле (длиной 6 бит): 111110
Битовое поле 2: 111110
Длина второго битового поля: 6 бит
Установлен бит во втором поле под индексом 4
Битовое поле 2: 111110
Значение бита во втором поле под индексом 5: 0
Битовое поле 2: 111110
Очищен бит во втором поле под индексом 2
Битовое поле 2: 110110
Битовые поля не равны
Битовое поле успешно скопировано
Битовое поле 1: 001101
Результат 1 и 2 операции 'или': 111111
Результат 1 и 2 операции 'и': 000100
Результат операции отрицания первого поля: 110010

C:\Users\Nikita\Documents\repository\mp2-practice\StarikovNV\01_lab\build\bin\sample_tbitfield.e
xe (процесс 8304) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Парам
етры" ->"Отладка" -> "Автоматически закрыть консоль при остановке отладки".
```

Рис. 4. Результат работы программы

2.2 Приложение для демонстрации работы множеств

1. Запустить `sample_tset.exe`. В результате появится следующее окно (рис. 5).



```
TSet
set1: { 2 4 5 8 }
set2: { 1 3 8 9 }
Element 5 in set1: 1
Element 2 in set2: 0
set2 after delete 3: { 1 8 9 }
operator==: 0
operator!=: 1
operator+ (elem): { 0 2 4 5 8 }
operator- (elem): { 2 4 8 }
operator*: { 8 }
operator+: { 1 2 4 5 8 9 }
operator~: { 0 1 3 6 7 9 }

C:\Users\Nikita\Documents\repository\mp2-practice\StarikovNV\01_lab\build\bin\sample_tset.exe (п
роцесс 27556) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Парам
етры" ->"Отладка" -> "Автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно:|
```

Рис. 5. Основное окно работы множеств

2. Создаются два множества максимальной мощности, в которые добавлены элементы, прописанные в коде (рис. 6).

```
set1: { 2 4 5 8 }
set2: { 1 3 8 9 }
```

Рис. 6. Множества

3. Производится проверка наличия элементов в множествах (рис. 7).

```
Element 5 in set1: 1
Element 2 in set2: 0
```

Рис. 7. Проверка элементов

4. Производится удаление элемента из множества `set2` (рис. 8).

```
set2 after delete 3: { 1 8 9 }
```

Рис. 8. Удаление элемента

5. И в конце производятся основные операции с множествами (рис. 9).

```
operator==: 0
operator!=: 1
operator+ (elem): { 0 2 4 5 8 }
operator- (elem): { 2 4 8 }
operator*: { 8 }
operator+: { 1 2 4 5 8 9 }
operator~: { 0 1 3 6 7 9 }
```

Рис. 9. Основные операции

2.3 Приложение «Решето Эратосфена»

1. Запустите sample_primenumbers.exe. В результате появится следующее окно (рис. 10).

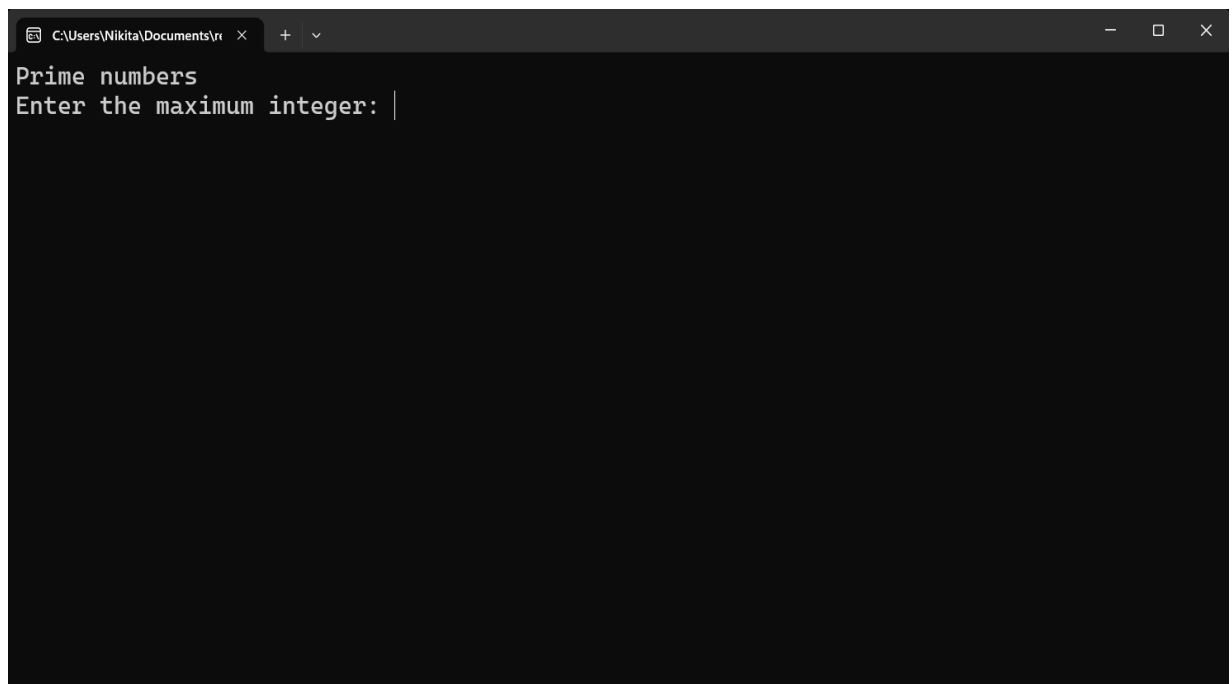
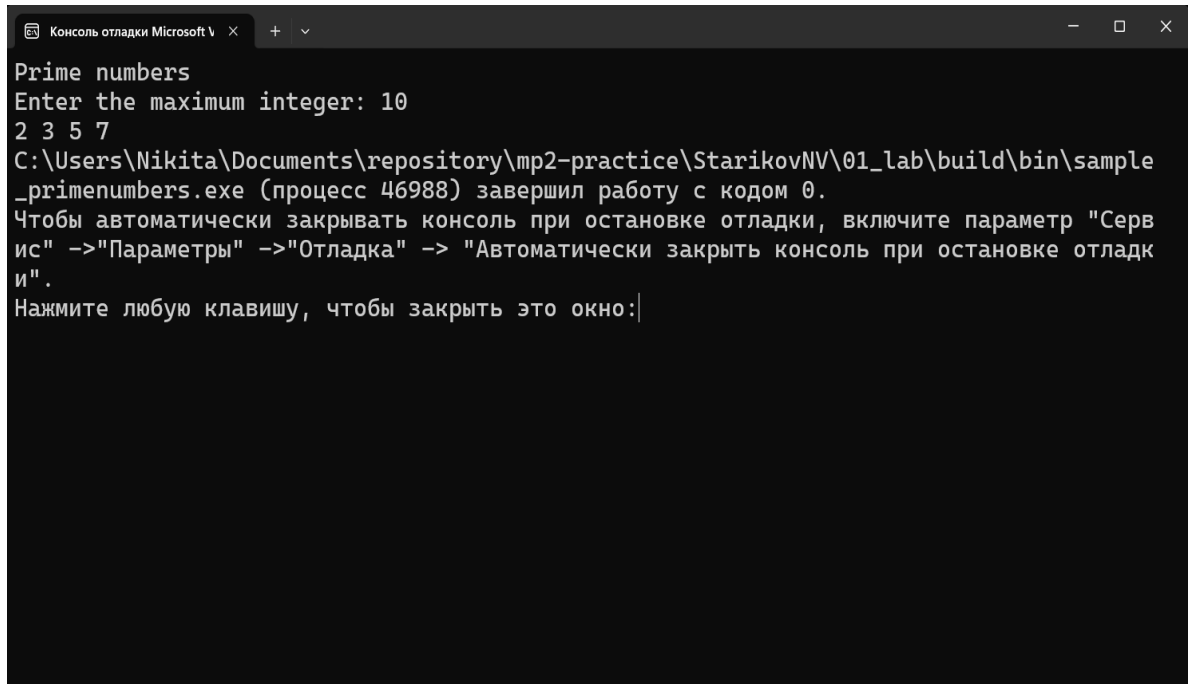


Рис. 10. Основное окно приложения

2. Затем вводим целое положительное число для того, чтобы получить все простые числа до введенного (рис. 11).



```
Консоль отладки Microsoft V x + v
Prime numbers
Enter the maximum integer: 10
2 3 5 7
C:\Users\Nikita\Documents\repository\mp2-practice\StarikovNV\01_lab\build\bin\sample
_primenumbers.exe (процесс 46988) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Серв
ис" ->"Параметры" ->"Отладка" -> "Автоматически закрыть консоль при остановке отладк
и".
Нажмите любую клавишу, чтобы закрыть это окно:|
```

Рис. 11. Результат работы приложение “Решето Эратосфена”

3 Руководство программиста

3.1 Используемые алгоритмы

3.1.1 Битовые поля

Каждое битовое поле задаётся длиной (универс битов), количеством единиц памяти (кол-во характеристических массивов) и памятью для их хранения. Элемент битового поля может находиться в двух состояниях: 1 и 0. 1 – элемент содержится в множестве, а 0 – элемент не содержится в множестве.

С битовыми полями можно реализовать несколько операций:

I. Операция побитовый «И»

После применения операции к битовым полям получаем одно битовое поле, в котором, если хотя бы в одном поле есть 0, то после применения операции в результате будет 0, иначе 1.

A	0	1	0
B	0	1	1
A&B	0	1	0

Входные данные: битовые поля.

Выходные данные: битовое поле, после применении операции.

II. Операция побитовое «ИЛИ»

После применения операции к битовым полям получаем одно битовое поле, в котором, если хотя бы в одном поле есть 1, то после применения операции в результате будет 1, иначе 0.

A	0	1	0
B	0	1	1
A B	0	1	1

Входные данные: битовые поля.

Выходные данные: битовое поле, после применении операции.

III. Операция дополнения

Операция возвращает экземпляр класса, каждый бит которого равен 0, если он есть в исходном классе, и 1 в противном случае. После применения операции к полю получаем инвертированное поле.

A	0	1	0
~A	1	0	1

Входные данные: битовое поле.

Выходные данные: битовое поле, после применения операции.

IV. Операция установки i-го бита в 1

Для реализации этой операции нам понадобится битовое поле, в котором все биты равны 0, и один бит равен 1. Этот бит должен стоять на разряде, который в исходном битовом поле мы должны установить в 1. В нашем случае это бит под номером 2. Вспомогательное поле будем называть битовой маской. Если к первоначальному битовому полю и битовой маске применить операцию побитового «ИЛИ», то в результате получим битовое поле, в котором бит на 2-ом разряде установлен в 1, а все оставшиеся биты равны битам в исходном битовом поле.

A	0	1	0
C	0	0	1
A C	0	1	1

A – первоначальное битовое поле.

C – битовая маска.

A|C – результат применения операции побитовое «ИЛИ» (итоговое битовое поле).

Входные данные: битовые поля.

Выходные данные: битовое поле с установленным в 1 битом.

V. Операция сброса i-го бита в 0

Битовую маску для этой операции получим инвертированием маски, которая использовалась при установке 2-го бита в 1. Такая маска имеет значение 2-го разряда 0, а значения всех остальных разрядов равны 1. Если к первоначальному битовому полю и инвертированной битовой маске применить операцию побитового «И», то в результате получим битовое поле, в котором бит на 2-ом разряде равен 0, а все оставшиеся биты равны битам исходного битового поля.

A	0	1	1
~C	1	1	0
A&~C	0	1	0

A – первоначальное битовое поле.

~C – инвертированная битовая маска.

A&~C – результат применения операции побитовый «И» (итоговое битовое поле).

Входные данные: битовые поля.

Выходные данные: битовое поле со сброшенным в 0 битом.

VI. Операция 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.

A	0	1	0
B	0	1	1
$A=B$	0		
$A \neq B$	1		

Входные данные: битовые поля.

Выходные данные: 0 если битовые поля не равны (равны) и равны (не равны).

3.1.2 Множества

Множество — это абстрактная структура данных, представляющая набор уникальных элементов. Множества на основе битовых полей используют битовые поля для хранения информации о наличии или отсутствии элементов в множестве. Каждый элемент в множестве представлен соответствующим битом.

С множествами можно провести следующие операции:

Множество может быть представлено с использованием характеристического массива, длина которого совпадает с длиной универса. А характеристический массив может реализовываться на базе битовых полей.

$A = \{2\ 4\ 5\ 8\}$ – в универсальном множестве из 10 элементов.

Это множество характеризуется битовым полем:

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

$B = \{1\ 3\ 8\ 9\}$ – в универсальном множестве из 10 элементов.

Это множество характеризуется битовым полем:

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

I. Операция пересечения двух множеств

В результате применения операции побитового «И» получаем множество, являющееся результатом пересечения двух других.

Это множество характеризуется битовым полем:

A	0	0	1	0	1	1	0	0	1	0
B	0	1	0	1	0	0	0	0	1	1

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

Результат применения операции: $A \cap B = \{8\}$

Входные данные: множества.

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

II. Операция объединения двух множеств

В результате применения операции побитового «ИЛИ» получаем множество, являющееся результатом объединения двух других.

Это множество характеризуется битовым полем:

A	0	0	1	0	1	1	0	0	1	0
B	0	1	0	1	0	0	0	0	1	1
$A \cup B$	0	1	1	1	1	1	0	0	1	1

Результат применения операции: $A \cup B = \{1\ 2\ 3\ 4\ 5\ 8\ 9\}$.

Входные данные: множества.

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

III. Операция дополнения

В результате применения к исходному множеству операции побитового отрицания получаем множество, являющееся дополнением к исходному.

Это множество характеризуется битовым полем:

A	0	0	1	0	1	1	0	0	1	0
$\sim A$	1	1	0	1	0	0	1	1	0	1

Результат применения операции: $\sim A = \{0\ 1\ 3\ 6\ 7\ 9\}$.

Входные данные: множество.

Выходные данные: множество, равное дополнению исходного множества.

IV. Операция сравнения

Операция равенства выведет 1, если два множества, которые характеризуются битовыми полями, 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.

A	0	0	1	0	1	1	0	0	1	0
B	0	1	0	1	0	0	0	0	1	1
$A == B$	0									
$A != B$	1									

Результат применения операции: $A == B = 0$; $A != B = 1$.

Входные данные: битовые поля.

Выходные данные: 0, если множества не равны; 1, если множества равны.

3.1.3 Алгоритм «решето Эратосфена»

Решето Эратосфена — алгоритм нахождения всех простых чисел до некоторого целого числа n , который приписывают древнегреческому математику Эратосфену Киренскому. Название алгоритма говорит о принципе его работы, то есть решето подразумевает фильтрацию, в данном случае фильтрацию всех чисел за исключением простых. По мере прохождения списка нужные числа остаются, а ненужные (они называются составными) исключаются.

Алгоритм выполнения состоит в следующем:

- 1) У пользователя запрашивается целое положительное число (обозначим за n).
- 2) Заполнение множества.
- 3) Проверка до квадратного корня n и удаление кратных членов.
 - (a) Начинаем с 2, проходим по всем числам, кратным 2 и меньшим n и удаляем их.
 - (b) Переходим к следующему не удалённому числу, то есть 3, оно будет вторым простым числом.
 - (c) Повторяем шаги (a) и (b).
- 4) Оставшиеся элементы будут простыми числами.

3.2 Описание классов

3.2.1 Класс TbitField

Объявление класса:

```
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; // индекс в pMem для бита n
    TELEM GetMemMask (const int n) const; // битовая маска для бита n
public:
    TBitField(int len);
    TBitField(const TBitField &bf);
    ~TBitField();

    // доступ к битам
    int GetLength(void) const; // получить длину (к-во битов)
    void SetBit(const int n); // установить бит
    void ClrBit(const int n); // очистить бит
    int GetBit(const int n) const; // получить значение бита
```

```

// битовые операции
int operator==(const TBitField &bf) const; // сравнение
int operator!=(const TBitField &bf) const; // сравнение
const 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);
friend ostream &operator<<(ostream &ostr, const TBitField &bf);
};

```

Поля:

BitLen – длина битового поля.

pMem – память для представления битового поля.

MemLen – количество элементов в **pMem** для представления битового поля.

Конструкторы:

```
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.

Деструктор:

```
~TBitField();
```

Назначение: деструктор. Очистка выделенной памяти.

Входные и выходные параметры: отсутствуют.

Методы:

```
int GetMemIndex(const int n) const;
```

Назначение: получение индекса элемента, где хранится бит.

Входные параметры: **n** – номер бита.

Выходные параметры: индекс элемента, где хранится бит с номером **n**.

```
TELEM GetMemMask(const n) const;
```

Назначение: получение битовой маски.

Входные параметры: **n** – номер бита.

Выходные параметры: элемент под номером **n**.

```
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, равным 1.

Входные параметры:

n – номер бита, который нужно установить.

Выходные параметры: отсутствуют.

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

Назначение: очистить бит (установить бит равным 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).

Операторы:

```
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. Присвоить экземпляру ***this** экземпляр **bf**.

Входные параметры:

bf – битовое поле, которое мы присваиваем.

Выходные параметры: ссылка на экземпляр класса `TBitField`, `*this`.

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

Назначение: оператор побитового «ИЛИ».

Входные параметры:

`bf` – битовое поле.

Выходные параметры: Экземпляр класса, который равен `{ *this | bf }`.

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

Назначение: оператор побитового «И».

Входные параметры:

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

Выходные параметры: Экземпляр класса, который равен `{ *this & bf }`.

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

Назначение: оператор инверсии.

Входные параметры: отсутствуют.

Выходные параметры: экземпляр класса, каждый элемент которого равен `{~*this}`.

```
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 Класс TSet

Объявление класса:

```
class TSet
{
private:
    int MaxPower;          // максимальная мощность множества
    TBitField BitField;    // битовое поле для хранения характеристического вектора
public:
    TSet(int mp);
    TSet(const TSet &s);    // конструктор копирования
    TSet(const TBitField &bf); // конструктор преобразования типа
    operator TBitField();   // преобразование типа к битовому полю
};
```

```

// доступ к битам
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
// теоретико-множественные операции
int operator== (const TSet &s) const; // сравнение
int operator!= (const TSet &s) const; // сравнение
const TSet& operator=(const TSet &s); // присваивание
TSet operator+ (const int Elem); // объединение с элементом
// элемент должен быть из того же универса
TSet operator- (const int Elem); // разность с элементом
// элемент должен быть из того же универса
TSet operator+ (const TSet &s); // объединение
TSet operator* (const TSet &s); // пересечение
TSet operator~ (void); // дополнение

friend istream &operator>>(istream &istr, TSet &bf);
friend ostream &operator<<(ostream &ostr, const TSet &bf);
};

```

Поля:

MaxPower — максимальный элемент множества.

BitField — экземпляр битового поля, на котором реализуется множество.

Конструкторы:

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.

Входные параметры:

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

Выходные параметры: отсутствуют.

TSet(const TBitField &bf);

Назначение: преобразование типа, преобразование из **TBitField** в **TSet**.

Входные параметры: **bf** — битовое поле.

Выходные параметры: отсутствуют.

operator TBitField();

Назначение: оператор преобразования из типа **TSet** в тип **TBitField**.

Входные параметры: отсутствуют.

Выходные параметры: объект класса **TBitField**.

Методы:

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.

Операторы:

```
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.

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

Назначение: оператор присваивания. Присвоить экземпляру ***this** экземпляр **s**.

Входные параметры: **s** – множество, которое мы присваиваем.

Выходные параметры: ссылка на экземпляр класса **TSet**, ***this**.

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

Назначение: оператор объединения множеств.

Входные параметры: **Elem** – число.

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

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

Назначение: оператор пересечения множеств.

Входные параметры: **Elem** – число.

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

```
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**.

Заключение

В результате данной лабораторной работы были изучены теоретические основы битовых полей и множеств, а также принципы их использования в программировании. Были реализованы классы TSet и TBitField, а также написаны приложения и тесты для проверки работоспособности реализации. Проведенный анализ результатов показал, что использование битовых полей и множеств может быть очень полезным в решении определенных задач. Они позволяют эффективно выполнять операции объединения, пересечения и разности между наборами элементов. В целом, лабораторная работа помогла понять основные принципы работы с битовыми полями и множествами, их преимущества и ограничения.

Литературы

1. Битовые поля [<https://metanit.com/c/tutorial/6.7.php>].
2. Bit_field [https://en.wikipedia.org/wiki/Bit_field].
3. Битовые множества (bitset) [<https://studfile.net/preview/1872018>].

Приложения

Приложение А. Реализация класса TBitField

```
TBitField::TBitField(int len)
{
    if (len > 0)
    {
        BitLen = len;
        MemLen = (len + sizeof(TELEM) * 8 - 1) / (sizeof(TELEM) * 8);
        pMem = new TELEM[MemLen];
        for (int i = 0; i < MemLen; i++)
            pMem[i] = 0;
    }
    else if (len == 0)
    {
        BitLen = 0;
        MemLen = 0;
        pMem = nullptr;
    }
    else
        throw "Size error";
}

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

TBitField::~TBitField()
{
    if (MemLen > 0)
        delete[] pMem;
}

int TBitField::GetMemIndex(const int n) const // индекс Мем для бита n
{
    return n / (sizeof(TELEM) * 8); // возвращает индекс ячейки для бита с номером n
}

TELEM TBitField::GetMemMask(const int n) const // битовая маска для бита n
{
    return 1 << (n % (sizeof(TELEM) * 8));
}

// 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 >= 0 && n < BitLen)
```

```

        pMem[GetMemIndex(n)] |= GetMemMask(n);
    else
        throw "Error";
}

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

int TBitField::GetBit(const int n) const // получить значение бита
{
    if (n >= 0 && n < BitLen)
        return (pMem[GetMemIndex(n)] & GetMemMask(n)) != 0;
    else
        throw "Error";
}

// битовые операции

const TBitField& TBitField::operator=(const TBitField& bf) // присваивание
{
    if (this != &bf)
    {
        if (MemLen != bf.MemLen)
        {
            delete[] pMem;
            MemLen = bf.MemLen;
            pMem = new TELEM[MemLen];
        }
        BitLen = bf.BitLen;
        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.BitLen)
        return 0;
    for (int i = 0; i < MemLen; i++)
    {
        if (pMem[i] != bf.pMem[i])
            return 0;
    }
    return 1;
}

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

TBitField TBitField::operator|(const TBitField& bf) // операция "или"
{
    int len = BitLen;
    if (bf.BitLen > len)
        len = bf.BitLen;
    TBitField result(len);
    for (int i = 0; i < MemLen; i++)
        result.pMem[i] = pMem[i] | bf.pMem[i];
    return result;
}

TBitField TBitField::operator&(const TBitField& bf) // операция "и"
{

```



```

        int len = BitLen;
        if (bf.BitLen > len)
            len = bf.BitLen;
        TBitField result(len);
        for (int i = 0; i < MemLen; i++)
            result.pMem[i] = pMem[i] & bf.pMem[i];
        return result;
    }

TBitField TBitField::operator~(void) // отрицание
{
    TBitField result(BitLen);
    for (int i = 0; i < BitLen; i++)
        if (!GetBit(i)) result.SetBit(i);
    return result;
}

// ввод/вывод

istream& operator>>(istream& istr, TBitField& bf) // ввод
{
    string str;
    istr >> str;
    for (int i = 0; i < bf.BitLen; i++)
    {
        int bit = str[i] - '0';
        if (bit)
            bf.SetBit(i);
        else
            bf.ClrBit(i);
    }
    return istr;
}

ostream& operator<<(ostream& ostr, const TBitField& bf) // вывод
{
    //for (int i = bf.BitLen - 1; i >= 0; i--)
    for (int i = 0; i < bf.BitLen; i++)
        ostr << bf.GetBit(i);
    return ostr;
}

```

Приложение Б. Реализация класса TSet

```
TSet::TSet(int mp) : BitField(mp)
{
    if (mp >= 0)
        MaxPower = mp;
    else
        throw "Size Error";
}

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

TSet::TSet(const TBitField& bf) : BitField(bf) // конструктор преобразования типа
{
    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 >= MaxPower || Elem < 0)
        throw "Error";
    return BitField.GetBit(Elem);
}

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

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

// теоретико-множественные операции

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

int TSet::operator==(const TSet& s) const // сравнение
{
    return BitField == s.BitField;;
}
```

```

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

TSet TSet::operator+(const TSet& s) // объединение
{
    size_t newMaxPower;
    if (MaxPower > s.MaxPower)
        newMaxPower = MaxPower;
    else
        newMaxPower = s.MaxPower;
    TSet result(newMaxPower);
    result.BitField = BitField | s.BitField;
    return result;
}

TSet TSet::operator+(const int Elem) // объединение с элементом
{
    if (Elem >= MaxPower || Elem < 0)
        throw "Error";
    TSet temp(*this);
    temp.BitField.SetBit(Elem);
    return temp;
}

TSet TSet::operator-(const int Elem) // разность с элементом
{
    if (Elem >= MaxPower || Elem < 0)
        throw "Error";
    TSet tmp(*this);
    tmp.BitField.ClrBit(Elem);
    return tmp;
}

TSet TSet::operator*(const TSet& s) // пересечение
{
    size_t newMaxPower;
    if (MaxPower > s.MaxPower)
        newMaxPower = MaxPower;
    else
        newMaxPower = s.MaxPower;
    TSet result(newMaxPower);
    result.BitField = BitField & s.BitField;
    return result;
}

TSet TSet::operator~(void) // дополнение
{
    TSet result(MaxPower);
    result.BitField = ~BitField;
    return result;
}

// перегрузка ввода/вывода

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

```

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