

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 группы 3822Б1ФИ1  
\_\_\_\_\_/ Петров Н.Д. /  
Подпись

**Проверил:** к.т.н., доцент каф. ВВиСП  
\_\_\_\_\_/ Кустикова В.Д. /  
Подпись

Нижний Новгород  
2023

# Оглавление

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 Приложение «решето Эратосфена».....	7
3 Руководство программиста .....	8
3.1 Используемые алгоритмы.....	8
3.1.1 Битовые поля .....	8
3.1.2 Множества .....	9
3.1.3 Алгоритм «решето Эратосфена» .....	10
3.2 Описание классов.....	11
3.2.1 Класс TBitField .....	11
3.2.2 Класс TSet .....	13
Заключение .....	15
Литература .....	16
Приложения .....	17
Приложение А. Реализация класса TBitField .....	17
Приложение Б. Реализация класса TSet.....	19
Приложение В. Реализация класса sample_tbitfield .....	21

## Введение

Битовые поля (или битовые множества) являются структурами данных, представляющими множества элементов, где каждый элемент может быть представлен битом (0 или 1) в некотором целочисленном значении. Это эффективный способ работы с наборами элементов, особенно когда множества малы, и каждый элемент может быть представлен уникальным битом.

Пример использования битовых полей включает следующие области:

1. Фильтрация данных: битовые поля можно использовать для фильтрации или классификации данных. Например, в базах данных можно использовать битовые поля для обозначения свойств элементов.
2. Управление доступом: битовые поля часто применяются в системах управления доступом для определения прав доступа пользователей к ресурсам.
3. Оптимизация поиска: битовые индексы используются в различных поисковых алгоритмах и базах данных для ускорения операций поиска.
4. Сжатие данных: битовые поля можно использовать для сжатия информации, когда множество элементов ограничено и их присутствие или отсутствие можно представить битами.
5. Алгоритмы обработки изображений: в обработке изображений битовые поля могут использоваться для представления масок, сегментации и других свойств пикселей.

# 1 Постановка задачи

Цель – реализация классов TBitField и TSet, предназначенных для работы с битовыми полями и множествами на их основе.

Задачи:

1. Исследовать тематическую литературу.
2. Реализовать класс TBitField.
3. Реализовать класс TSet.
4. Провести тестирование разработанных классов для проверки их корректной работы.

## 2 Руководство пользователя

### 2.1 Приложение для демонстрации работы битовых полей

1. Запустить sample\_tbitfield.exe. В результате появится следующее окно (Рис. 1).

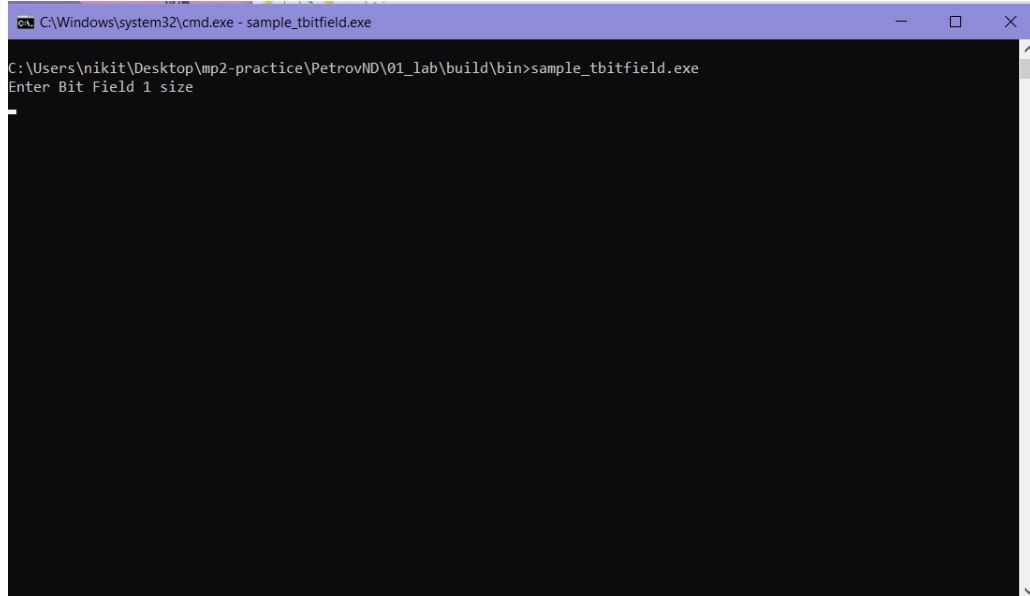


Рис. 1. Основное окно приложения sample\_tbitfield

2. Необходимо ввести размеры первого, второго и третьего битовых полей (1 и 2 будут сгенерированы автоматически, вам будет предложено заполнить третье поле самостоятельно). Затем будет продемонстрирована работа битовых полей (Рис. 2).

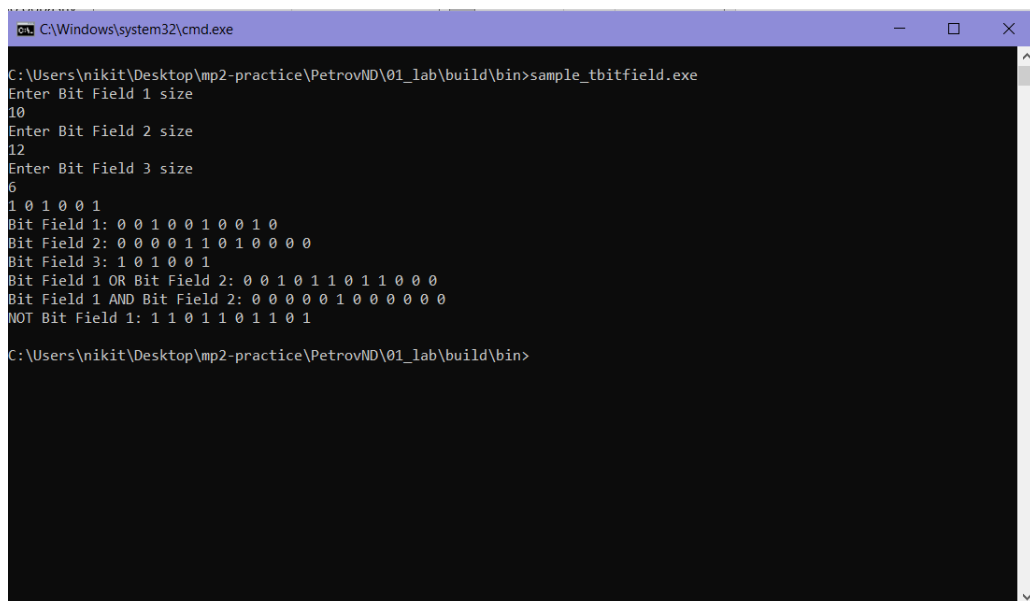


Рис. 2. Основное окно приложения sample\_tbitfield

## 2.2 Приложение для демонстрации работы множеств

1. Запустить sample\_tset.exe. В результате появится следующее окно (Рис. 3).

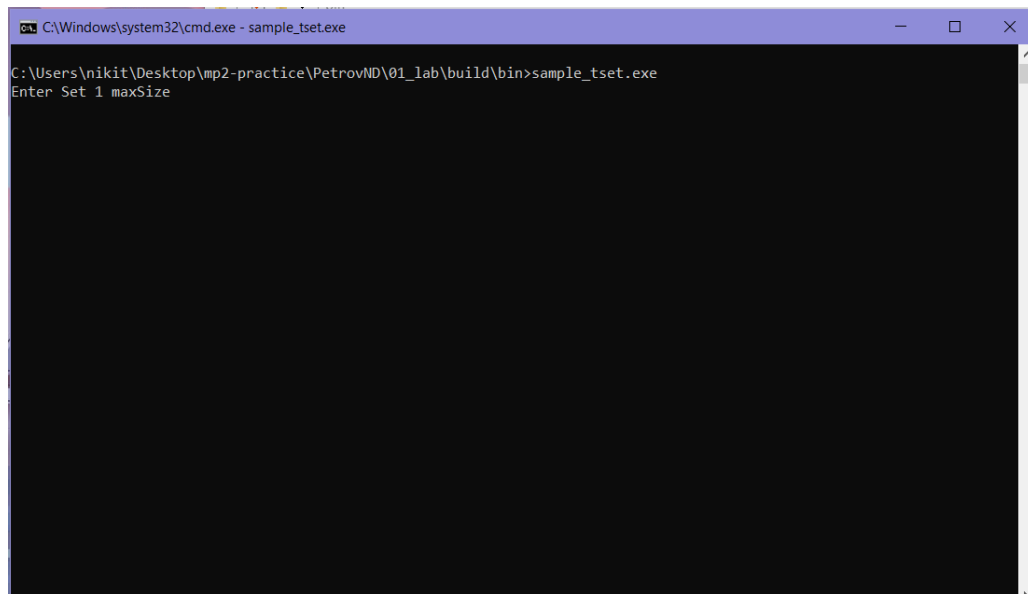


Рис. 3. Основное окно приложения sample\_tset

2. Необходимо ввести максимальную мощность первого, второго и третьего множества (1 и 2 будут заполнены автоматически, вам будет предложено заполнить третье множество самостоятельно) Затем будет продемонстрирована работа множеств (Рис. 4).

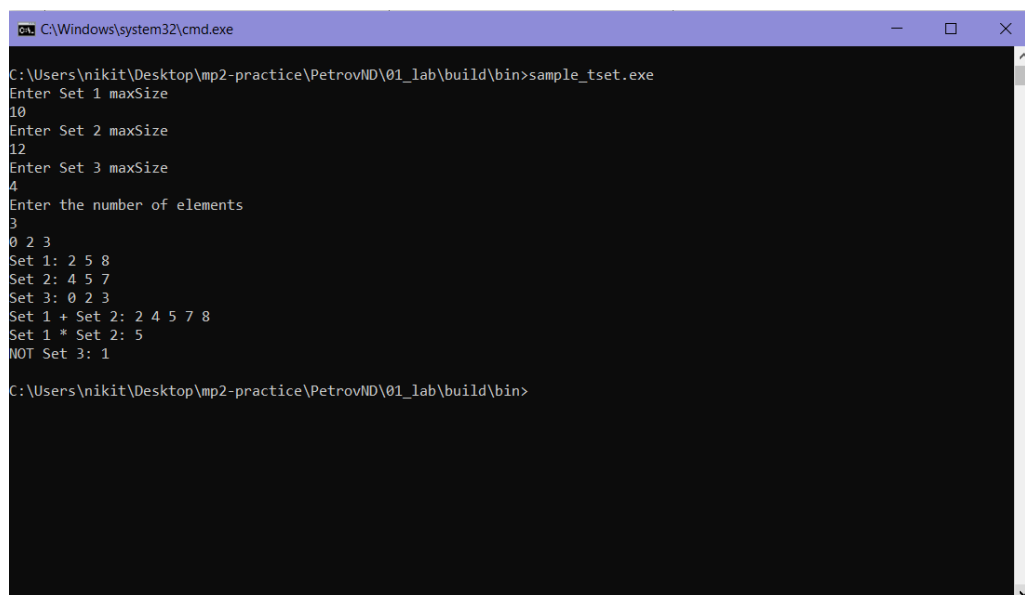
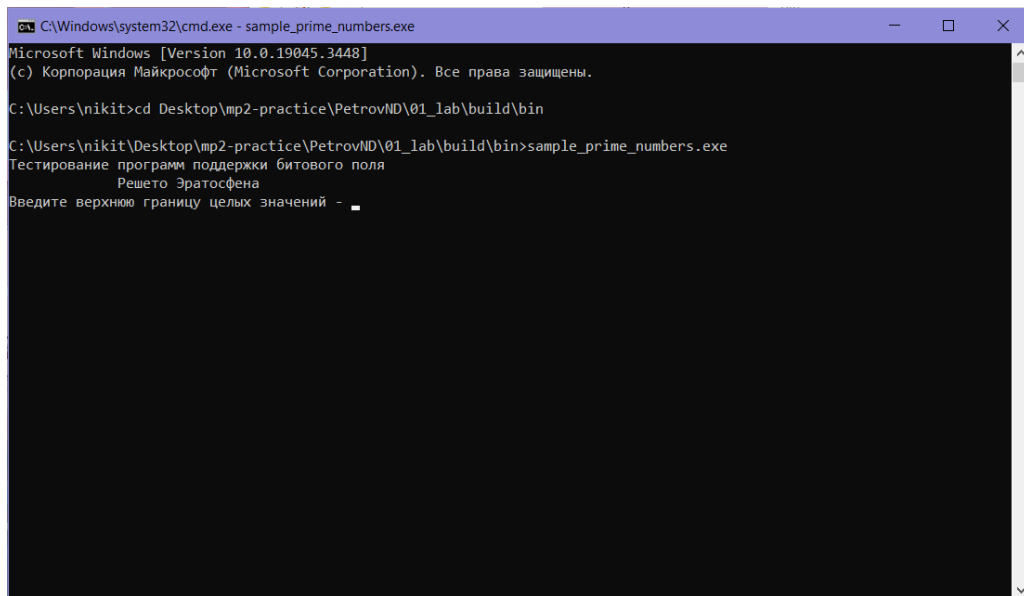


Рис. 4. Основное окно приложения sample\_tset

## 2.3 Приложение «решето Эратосфена»

1. Запустить sample\_prime\_numbers.exe. В результате появится следующее окно (Рис. 5).

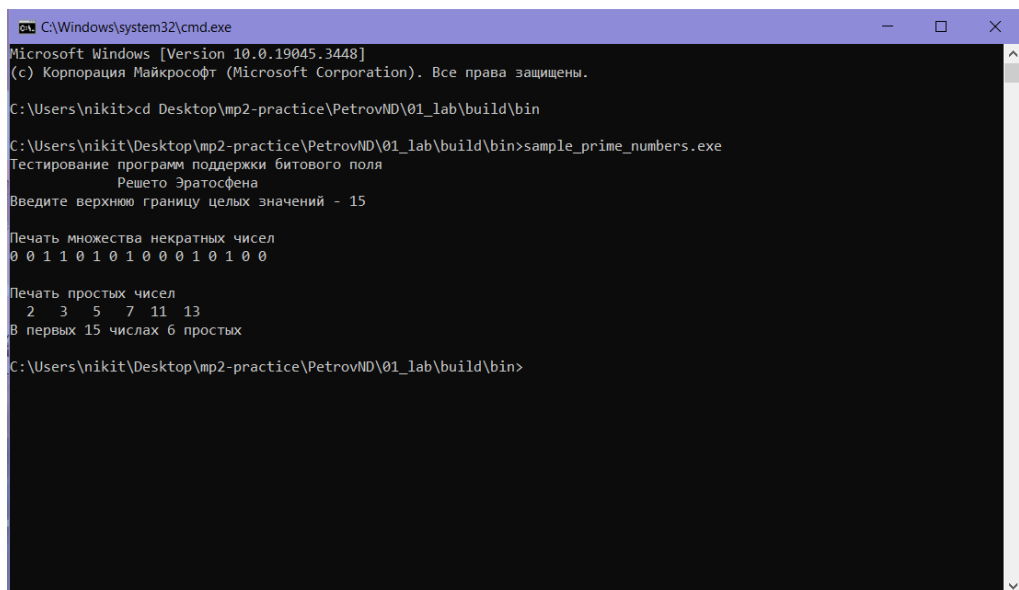


```
C:\Windows\system32\cmd.exe - sample_prime_numbers.exe
Microsoft Windows [Version 10.0.19045.3448]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\nikit>cd Desktop\mp2-practice\PetrovND\01_lab\build\bin
C:\Users\nikit\Desktop\mp2-practice\PetrovND\01_lab\build\bin>sample_prime_numbers.exe
Тестирование программ поддержки битового поля
Решето Эратосфена
Введите верхнюю границу целых значений - _
```

Рис. 5. Основное окно приложения sample\_prime\_numbers

2. Введите верхнюю границу целых значений для получения результата (Рис. 6).



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.19045.3448]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\nikit>cd Desktop\mp2-practice\PetrovND\01_lab\build\bin
C:\Users\nikit\Desktop\mp2-practice\PetrovND\01_lab\build\bin>sample_prime_numbers.exe
Тестирование программ поддержки битового поля
Решето Эратосфена
Введите верхнюю границу целых значений - 15

Печать множества некрatных чисел
0 0 1 1 0 1 0 1 0 0 0 1 0 1 0 0

Печать простых чисел
2 3 5 7 11 13
В первых 15 числах 6 простых

C:\Users\nikit\Desktop\mp2-practice\PetrovND\01_lab\build\bin>
```

Рис. 6. Основное окно приложения sample\_prime\_numbers

## 3 Руководство программиста

### 3.1 Используемые алгоритмы

#### 3.1.1 Битовые поля

Битовое поле представляет собой структуру данных, в которой каждый бит имеет значение либо 0 (сброшен), либо 1 (установлен). Эти биты хранятся в массиве (или нескольких массивах), где каждый элемент массива называется "словом" и содержит несколько битов.

Пример битового поля длины 10 (Рис. 7 ниже):

0	0	1	1	0	1	1	0	0	0
---	---	---	---	---	---	---	---	---	---

Рис. 7. Представление битового поля

Операции над битовыми полями:

1. Установка бита. Выполняется путем изменения соответствующего бита на 1 (Рис. 8).

0	0	1	1	0	1	1	1	0	0
---	---	---	---	---	---	---	---	---	---

Рис. 8. Установка бита под номером 2

2. Сброс бита. Выполняется путем изменения соответствующего бита на 0 (Рис. 9).

0	0	1	1	0	1	0	0	0	0
---	---	---	---	---	---	---	---	---	---

Рис. 9. Сброс бита под номером 3

3. Операции ИЛИ (Рис. 10), И (Рис. 11) и отрицания (Рис. 12). Позволяют объединять, пересекать и инвертировать битовые поля.

А	0	0	1	1	0	1	0	0	0	0
Б	0	1	1	1	0	1	0	1	1	0
А   Б	0	1	1	1	0	1	0	1	1	0

Рис. 10. А или Б

А	0	0	1	1	0	1	0	0	0	0
Б	0	1	1	1	0	1	0	1	1	0
А & Б	0	0	1	1	0	1	0	0	0	0



Рис. 11. А и Б

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

Рис. 12. Отрицание А

- Получение значения бита. Предполагается чтение значения бита без его изменения.
- Сравнение битовых полей. Выполняется определение равенства и неравенства двух битовых полей.

### 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.

Пример множества А, мощность которого равна 10 элементов (Рис. 13):

$A = \{4, 6, 7\}$										
A	0	0	1	1	0	1	0	0	0	0

Рис. 13. Множество А и его битовое поле

Операции над множествами на основе битовых полей:

- Вставка элемента. Устанавливает соответствующий бит для элемента (Рис. 14).

$A = \{1, 4, 6, 7\}$										
A	0	0	1	1	0	1	0	0	1	0

Рис. 14. Вставка элемента 1 в множество А

- Удаление элемента. Сбрасывает соответствующий бит для элемента (Рис. 15).

$A = \{1, 4, 6\}$										
A	0	0	0	1	0	1	0	0	1	0

Рис. 15. Удаление Элемента 7 из множества А

- Операции множественного объединения (Рис. 16), пересечения (Рис. 17) и отрицания (Рис. 18). Выполняются путём объединения, пересечения и отрицания битовых полей.

$A = \{1, 4, 6\}$										
A	0	0	0	1	0	1	0	0	1	0
$B = \{0, 1, 3, 6, 9\}$										
B	1	0	0	1	0	0	1	0	1	1
$A \cup B = \{0, 1, 3, 4, 6, 9\}$										
$A \cup B$	1	0	0	1	0	1	1	0	1	1

Рис. 16. Объединение множеств A и B

$A = \{1, 4, 6\}$										
A	0	0	0	1	0	1	0	0	1	0
$B = \{0, 1, 3, 6, 9\}$										
B	1	0	0	1	0	0	1	0	1	1
$A \cap B = \{1, 6\}$										
$A \cap B$	0	0	0	1	0	0	0	0	1	0

Рис. 17. Пересечение множеств A и B

$B = \{0, 1, 3, 6, 9\}$										
B	1	0	0	1	0	0	1	0	1	1
$\neg B = \{2, 4, 5, 7, 8\}$										
$\sim B$	0	1	1	0	1	1	0	1	0	0

Рис. 18. Отрицание(дополнение) множества B

4. Проверка наличия элемента. Проверка значения соответствующего бита для элемента.

### 3.1.3 Алгоритм «решето Эратосфена»

Алгоритм решето Эратосфена — это алгоритм нахождения всех простых чисел до некоторого целого числа N.

Он заключается в фильтрации чисел, в результате которой нужные числа (простые) остаются, а ненужные (составные) исключаются.

1. Для нахождения всех простых чисел до заданного числа N нужно выполнить следующие шаги:

2. Заполнить массив из N элементов целыми числами подряд от 2 до N.
3. Присвоить переменной p значение 2 (первого простого числа).
4. Удалить из массива числа от p2 до N с шагом p (это будут числа кратные p: p2, p2+p, p2+2p и т. д.).
5. Найти первое неудаленное число в массиве, большее p, и присвоить значению переменной p это число.
6. Повторять два предыдущих шага пока это возможно.
7. Все оставшиеся в массиве числа являются простыми числами от 2 до N

## 3.2 Описание классов

### 3.2.1 Класс TBitField

Объявление класса:

```
class TBitField
{
private:
    int BitLen;
    TELEM *pMem;
    int MemLen;
    const int bitsInElem = sizeof(TELEM) * 8;

    int GetMemIndex(const int n) const;
    TELEM GetMemMask (const int n) const;
public:
    TBitField(int len);
    TBitField(const TBitField &bf);
    ~TBitField();

    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;
    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);
};
```

Поля:

**BitLen** хранит длину битового поля, то есть максимальное количество битов, которые могут быть представлены в данном объекте.

**pMem** является указателем на массив, который используется для фактического хранения битовых данных. Каждый элемент массива **pMem** содержит набор битов.

**MemLen** хранит количество элементов массива **pMem**, необходимых для хранения битовых данных.

Методы:

**int GetMemIndex(const int n) const;**

Назначение: Метод используется для получения индекса элемента, в котором хранится бит с указанным номером.

Входные данные: n - номер бита, для которого нужно найти индекс элемента.

Выходные данные: Индекс элемента массива **pMem**, в котором хранится бит с номером n. Этот индекс будет использоваться для доступа к соответствующему элементу памяти, где хранятся биты.

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

Назначение: Метод используется для получения битовой маски, которая соответствует биту с указанным номером.

Входные данные: n - номер бита, для которого нужно получить битовую маску.

Выходные данные: Битовая маска, в которой только один бит установлен в позиции, соответствующей номеру n. Остальные биты маски равны 0.

**int GetLength() const;**

Назначение: метод используется для получения длины битового поля.

Выходные данные: длина битового поля, то есть количество битов, которое может храниться в объекте TBitField.

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

Назначение: Метод используется для установки бита с указанным номером.

Входные данные: n – номер бита, который нужно установить (установить в 1).

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

Назначение: Метод используется для очистки бита с указанным номером (установка в 0).

Входные данные: n - номер бита, который нужно очистить.

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

Назначение: Метод используется для получения значения бита с указанным номером.

Входные данные: n - номер бита, для которого нужно получить значение (0 или 1).

Выходные данные: Значение бита с номером *n*.

### 3.2.2 Класс TSet

Объявление класса:

```
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;
    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);
};
```

Поля:

**MaxPower** хранит максимальную мощность множества (максимальное количество элементов в множестве).

**BitField** представляет собой битовое поле, где каждый бит соответствует элементу множества. Размер битового поля равен **MaxPower**.

Методы:

**int GetMaxPower() const;**

Назначение: получение максимальной мощности множества, то есть максимального номера элемента, который может быть включен в множество.

Выходные данные: максимальная мощность множества.

**void InsElem(const int Elem);**

Назначение: включение элемента в множество.

Входные данные: **Elem** – номер элемента, который нужно включить в множество.

```
void DelElem(const int Elem);
```

Назначение: удаление элемента из множества.

Входные данные: **Elem** – номер элемента, который нужно удалить из множества.

```
int IsMember(const int Elem) const;
```

Назначение: проверка наличия элемента в множестве.

Входные данные: **Elem** – номер элемента, который нужно проверить.

Выходные данные: 1, если элемент присутствует в множестве, и 0 в противном случае.

## Заключение

В результате проделанной работы были созданы два класса, **TBitField** и **TSet**, предоставляющие возможность работать с битовыми полями и множествами. Эти классы могут быть использованы в различных задачах, требующих манипуляции битами и множествами элементов. Весь функционал классов был реализован и протестирован, что позволяет уверенно использовать их в практических задачах.

## Литература

1. C | Битовые поля (metanit.com) [<https://metanit.com/c/tutorial/6.7.php>]
2. Битовые поля в C++ | Microsoft Learn [<https://learn.microsoft.com/ru-ru/cpp/cpp/cpp-bit-fields?view=msvc-170>]
3. C/C++/Хабр | Habr [<https://habr.com/ru/articles/142662/>]



# Приложения

## Приложение А. Реализация класса TBitField

```
TBitField::TBitField(int len) {
    if (len < 0) {
        throw invalid_argument("Length cannot be negative");
    }
    BitLen = len;
    MemLen = (len + bitsInElem - 1) / bitsInElem;
    pMem = new TELEM[MemLen];
    memset(pMem, 0, MemLen * sizeof(TELEM));
}

TBitField::TBitField(const TBitField& bf) {
    BitLen = bf.BitLen;
    MemLen = bf.MemLen;
    pMem = new TELEM[MemLen];
    memcpy(pMem, bf.pMem, MemLen * sizeof(TELEM));
}

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

int TBitField::GetMemIndex(const int n) const {
    if (n < 0 || n >= BitLen) {
        throw out_of_range("Bit index is out of range");
    }
    return n / bitsInElem;
}

TELEM TBitField::GetMemMask(const int n) const {
    if (n < 0 || n >= BitLen) {
        throw out_of_range("Bit index is out of range");
    }
    int bitIndex = n % bitsInElem;
    TELEM mask = 1;
    mask <=<= bitIndex;
    return mask;
}

int TBitField::GetLength(void) const {
    return BitLen;
}

void TBitField::SetBit(const int n) {
    if (n < 0 || n >= BitLen) {
        throw out_of_range("Bit index is out of range");
    }
    int index = GetMemIndex(n);
    TELEM mask = GetMemMask(n);
    pMem[index] |= mask;
}

void TBitField::ClrBit(const int n) {
    if (n < 0 || n >= BitLen) {
        throw out_of_range("Bit index is out of range");
    }
    int index = GetMemIndex(n);
    TELEM mask = GetMemMask(n);
```

```

        pMem[index] &= ~mask;
    }

    int TBitField::GetBit(const int n) const {
        if (n < 0 || n >= BitLen) {
            throw out_of_range("Bit index is out of range");
        }
        int index = GetMemIndex(n);
        TELEM mask = GetMemMask(n);
        return (pMem[index] & mask) ? 1 : 0;
    }

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

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

    TBitField TBitField::operator|(const TBitField& bf) {
        int len = (BitLen > bf.BitLen) ? BitLen : 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) {
        int len = (BitLen > bf.BitLen) ? BitLen : 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) {

```

```

    TBitField result(BitLen);
    for (int i = 0; i < BitLen; i++) {
        if (!GetBit(i))
            result.SetBit(i);
    }
    return result;
}

istream& operator>>(istream& istr, TBitField& bf) {
    int inputBit;
    for (int i = 0; i < bf.BitLen; i++) {
        istr >> inputBit;
        if (inputBit == 0) {
            bf.ClrBit(i);
        }
        else if (inputBit == 1) {
            bf.SetBit(i);
        }
        else {
            throw invalid_argument("Invalid input. Use 0 or 1.");
        }
    }
    return istr;
}

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

```

## Приложение Б. Реализация класса TSet

```

TSet::TSet(int mp) : MaxPower(mp), BitField(mp) { }

TSet::TSet(const TSet& s) : MaxPower(s.MaxPower), BitField(s.BitField) { }

TSet::TSet(const TBitField& bf) : MaxPower(bf.GetLength()), BitField(bf) { }

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

int TSet::GetMaxPower(void) const {
    return MaxPower;
}

void TSet::InsElem(const int Elem) {
    if (Elem < 0 || Elem >= MaxPower) throw out_of_range("Element is out of range");
    BitField.SetBit(Elem);
}

void TSet::DelElem(const int Elem) {
    if (Elem < 0 || Elem >= MaxPower) throw out_of_range("Element is out of range");

    BitField.ClrBit(Elem);
}

int TSet::IsMember(const int Elem) const {

```

```

        if (Elem < 0 || Elem >= MaxPower) throw out_of_range("Element is out of
range");
        return BitField.GetBit(Elem);
    }

    int TSet::operator==(const TSet& s) const {
        if (MaxPower != s.MaxPower) {
            return 0;
        }
        return BitField == s.BitField;
    }

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

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

    TSet TSet::operator+(const int Elem) {
        if (Elem >= MaxPower || Elem < 0) throw invalid_argument("Out of range");
        TSet result(*this);
        result.InsElem(Elem);
        return result;
    }

    TSet TSet::operator-(const int Elem) {
        if (Elem >= MaxPower || Elem < 0) throw invalid_argument("Out of range");
        TSet result(*this);
        result.DelElem(Elem);
        return result;
    }

    TSet TSet::operator+(const TSet& s) {
        int len = (MaxPower > s.MaxPower) ? MaxPower : s.MaxPower;
        TSet result(len);
        result.BitField = BitField | s.BitField;
        return result;
    }

    TSet TSet::operator*(const TSet& s) {
        int len = (MaxPower > s.MaxPower) ? MaxPower : s.MaxPower;
        TSet result(len);
        result.BitField = BitField & s.BitField;
        return result;
    }

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

    istream& operator>>(istream& istr, TSet& ts) {
        int inputElem, n;
        cout << "Enter the number of elements" << endl;
        cin >> n;

```

```

    for (int i = 0; i < n; i++) {
        istr >> inputElem;
        if (inputElem >= 0 && inputElem < ts.MaxPower) {
            ts.InsElem(inputElem);
        }
        else {
            cerr << "Invalid input. Element out of range." << endl;
        }
    }
    return istr;
}

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

```

## Приложение В. Реализация класса sample\_tbitfield

```

int main() {
    try {
        int a, b, c;
        cout << "Enter Bit Field 1 size" << endl;
        cin >> a;
        cout << "Enter Bit Field 2 size" << endl;
        cin >> b;
        cout << "Enter Bit Field 3 size" << endl;
        cin >> c;

        TBitField bf1(a);
        TBitField bf2(b);
        TBitField bf3(c);

        cin >> bf3;

        bf1.SetBit(2);
        bf1.SetBit(5);
        bf1.SetBit(8);

        bf2.SetBit(4);
        bf2.SetBit(5);
        bf2.SetBit(7);

        cout << "Bit Field 1: " << bf1 << endl;
        cout << "Bit Field 2: " << bf2 << endl;
        cout << "Bit Field 3: " << bf3 << endl;

        cout << "Bit Field 1 OR Bit Field 2: " << (bf1 | bf2) << endl;
        cout << "Bit Field 1 AND Bit Field 2: " << (bf1 & bf2) << endl;
        cout << "NOT Bit Field 1: " << ~bf1 << endl;

    }
    catch (const std::exception& e) {
        std::cerr << "Error: " << e.what() << std::endl;
    }
    return 0;
}

```