

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»

Выполнила: студентка группы
3822Б1ФИ2

_____/ Ясакова Т.Е./
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.....	5
2.3 «Решето Эратосфено»	6
3 Руководство программиста	Ошибка! Закладка не определена.
3.1 Описание алгоритмов	Ошибка! Закладка не определена.
3.1.1 Битовые поля	Ошибка! Закладка не определена.
3.1.2 Множества	Ошибка! Закладка не определена.
3.1.3 «Решето Эратосфена»	Ошибка! Закладка не определена.
3.2 Описание программной реализации	8
3.2.1 Описание класса TBitField	10
3.2.2 Описание класса TSet	13
Заключение	17
Литература	18
Приложения	19
Приложение А. Реализация класса TBitField	19
Приложение Б. Реализация класса TSet.....	21
Приложение В. Sample_primenumbers	23

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.

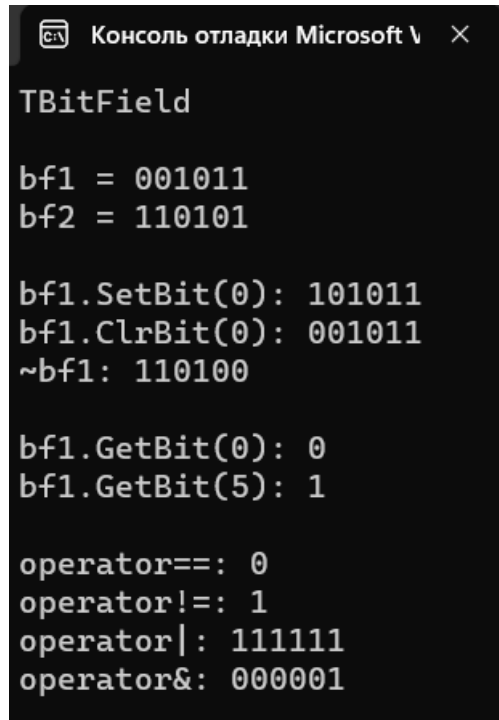
Задачи:

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).



```
Консоль отладки Microsoft V
TBitField

bf1 = 001011
bf2 = 110101

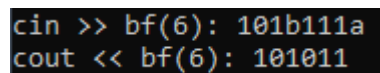
bf1.SetBit(0): 101011
bf1.ClrBit(0): 001011
~bf1: 110100

bf1.GetBit(0): 0
bf1.GetBit(5): 1

operator==: 0
operator!=: 1
operator |: 111111
operator&: 000001
```

Рис. 1. Основное окно программы `sample_tbitfield.exe`

2. В появившемся окне вы можете ознакомиться с примером работы реализованных операций. Введите строку с данными битового поля, она должна содержать “0” и “1”, а также быть указанной длины (в данном случае 6). Другие символы будут считаться за “0”, а из строки большей длины будет учитываться только подстрока указанной длины. Нажмите кнопку ввода, и выведется результат, пример которого указан на рисунке ниже (рис. 2).



```
cin >> bf(6): 101b111a
cout << bf(6): 101011
```

Рис. 2. Пример функций ввода и вывода класса `TBitField`

2.2 Приложение для демонстрации работы множеств

1. Запустите приложение с названием `sample_tset.exe`. В результате появится окно, показанное ниже (рис. 3).

```

TSet

s1 = 001011
s2 = 110101

s1.InsElem(0): 101011
s1.DelElem(0): 001011
~s1: 110100

s1.IsMember(0): 0
s1.IsMember(5): 1

operator==: 0
operator!=: 1
operator+: 111111
operator+ (elem): 101011
operator- (elem): 001010
operator*: 000001

```

Рис. 3. Основное окно программы sample_tset.exe

- В появившемся окне вы можете ознакомиться с примером работы реализованных операций. Введите строку с данными битового поля, она должна содержать “0” и “1”, а так же быть указанной длины (в данном случае 6). Другие символы будут считаться за “0”, а из строки большей длины будет учитываться только подстрока указанной длины. Нажмите кнопку ввода, и выведется результат, пример которого указан на рисунке ниже (рис. 4)

```

cin >> s(6): Count of entered values: 3
2
3
7
cout << s(6): 2 3

```

Рис. 4. Пример функций ввода и вывода класса TSet

2.3 «Решето Эратосфено»

- Запустите приложение с названием sample_primenumbers.exe. В результате появится окно, показанное ниже (рис. 5).

```

Prime numbers

Enter the count of numbers: |

```

Рис. 5. Начало работы программы sample_primenumbers.exe

2. Введите положительное целое число, чтобы вывести все простые числа до этого числа (включительно). Напечатается результат, показанный на рисунке ниже (рис. 6).

```
Prime numbers  
  
Enter the count of numbers: 40  
Prime numbers under 40:  
2 3 5 7 11 13 17 19 23 29 31 37
```

Рис. 6. Результат работы программы sample_primenumbers.exe

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 байту, что в свою очередь позволяет экономить память, более плотно размещая данные.

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

Операция ИЛИ:

Операция возвращает экземпляр битового поля, каждый бит которого равен 1, если он есть хотя бы в 1 классе, которые объединяем, и 0 в противном случае.

Пример:

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

Операция И:

Операция возвращает экземпляр класса, каждый бит которого равен 1, если он есть в каждом классе, и 0 в противном случае.

Пример:

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

Операция дополнения (отрицания):

Операция возвращает экземпляр класса, каждый бит которого равен 0, если он есть исходном классе, и 1 в противном случае.

Пример:

A	0	1	1	0	0	1	0	1
~A	1	0	0	1	1	0	1	0

Операция установки бита в единицу:

Исходное поле:

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

В результате установки бита на позицию 3 получается поле:

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

Операция установки бита в ноль:

Исходное поле:

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

В результате очищения бита на позиции 4 получится поле:

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

3.1.2 Множества

Множества полностью основаны на характеристических массивах - битовых полях. Битовое поле описывает каждый элемент универсума: если бит равен 1, то элемент присутствует в множестве; если бит равен 0, то в множестве его нет.

Каждое множество может иметь свой смысл и применяемость. В данной работе мы рассматриваем множество натуральных чисел, на основе которого мы формируем подмножества.

A - множество натуральных нечетных чисел на промежутке [2,5).

$A = \{3, 4\}$

Битовое поле 011 – характеристический массив множества A.

3.1.3 «Решето Эратосфена»

Задача: дано целое положительное число. Требуется найти все простые числа в отрезке от 2 до этого числа.

Входные данные: целое положительное число (далее n)

Выходные данные: множество простых чисел

Суть алгоритма: идти по натуральным числам и на каждом найденном в множестве числе исключать все элементы, кратные тому, на котором сейчас остановились.

Алгоритм:

1. Выписать подряд все числа от 2 до n
2. Пусть переменная p изначально равна двум – первому простому числу
3. Зачеркнуть в списке числа от 2p до n, считая шагами по p (то есть удаляем числа, кратные p).

4. Найти первое не зачеркнутое число в списке, большее чем *p*, и присвоить значению переменной *p* это число
5. Повторять шаги 3 и 4, пока возможно

3.2 Описание программной реализации

3.2.1 Описание класса TBitField

```
typedef unsigned int TELEM;
class TBitField
{
private:
    int BitLen;
    TELEM *pMem;
    int MemLen;

    const int bitsInElem = 32;
    const int shiftSize = 5;

    // методы реализации
    int GetMemIndex(const int n) const noexcept;
    TELEM GetMemMask (const int n) const noexcept;
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;

    // битовые операции
    bool operator==(const TBitField &bf) const;
    bool 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& in, TBitField& bf);
    friend ostream& operator<<(ostream& out, 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.

bitsInElem – вспомогательное значение, количество битов в элементе памяти.

shiftSize – вспомогательное значение для битового целочисленного деления.

Конструкторы:

TBitFields(int len);

Назначение: выделение и инициализация памяти объекта.

Входные параметры: **len** – количество доступных битов.

TBitFields(const TBitFields &bf);

Назначение: выделение памяти и копирование данных.

Входные параметры: **bf** – объект класса **TBitFields**.

~TBitFields();

Назначение: очистка выделенной памяти.

Методы:

int GetMemIndex(const int n) const noexcept;

Назначение: получение индекса элемента в памяти.

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

Выходные параметры: индекс элемента в памяти.

TELEM GetMemMask (const int n) const noexcept;

Назначение: получение маски бита

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

Выходные параметры: маска бита

int GetLength(void) const;

Назначение: получение количества доступных битов.

Выходные параметры: **BitLen** - количество доступных битов.

void SetBit(const int n);

Назначение: изменить значение бита на 1.

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

void ClrBit(const int n);

Назначение: изменить значение бита на 0.

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

int GetBit(const int n) const;

Назначение: получение значения бита.

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

Выходные параметры: значение бита – 1 или 0.

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

Назначение: перегрузка операции сравнения, сравнение на равенство объектов.

Входные параметры: **bf** – объект класса **TBitField**.

Выходные параметры: **true** или **false**.

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

Назначение: перегрузка операции сравнения, сравнение на неравенство объектов.

Входные параметры: **bf** – объект класса **TBitField**.

Выходные параметры: **true** или **false**.

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

Назначение: присваивание значений объекта **bf**.

Входные параметры: **bf** – объект класса **TBitField**.

Выходные параметры: ссылка на объект класса **TBitField** (себя).

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

Назначение: создание объекта с примененной побитовой операцией ИЛИ.

Входные параметры: **bf** – объект класса **TBitField**.

Выходные параметры: объект класса **TBitField**.

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

Назначение: создание объекта с примененной побитовой операцией И.

Входные параметры: **bf** – объект класса **TBitField**.

Выходные параметры: объект класса **TBitField**.

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

Назначение: создание объекта с примененной побитовой операцией отрицания.

Выходные параметры: объект класса **TBitField**.

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

Назначение: ввод данных.

Входные параметры: **in** – поток ввода, **bf** – объект класса **TBitField**.

Выходные параметры: поток ввода.

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

Назначение: вывод данных.

Входные параметры: `out` – поток вывода, `bf` – объект класса `TBitField`.

Выходные параметры: поток вывода.

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();

    // доступ к битам
    int GetMaxPower(void) const noexcept;
    void InsElem(const int Elem);
    void DelElem(const int Elem);
    bool IsMember(const int Elem) const;

    // теоретико-множественные операции
    bool operator== (const TSet& s) const;
    bool 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& in, TSet& bf);
    friend ostream& operator<<(ostream& out, const TSet& bf);
};
```

Назначение: представление множества.

Поля:

`MaxPower` – мощность множества.

`BitField` – характеристический массив.

Конструкторы:

```
TSet(int mp);
```

Назначение: инициализация битового поля.

Входные параметры: `mp` – количество элементов в универсуме.

TSet(const TSet& s);

Назначение: копирование данных из другого множества.

Входные параметры: **s** – объект класса **TSet**.

TSet(const TBitField& bf);

Назначение: формирование множества на основе битового поля.

Входные параметры: **bf** – объект класса **TBitField**.

operator TBitField();

Назначение: получение поля **BitField**.

Выходные параметры: объект класса **TBitField**.

Методы:

int GetMaxPower(void) const noexcept;

Назначение: получение максимальной мощности множества.

Выходные параметры: **MaxPower** – максимальная мощность множества.

void InsElem(const int Elem);

Назначение: добавить элемент в множество.

Входные параметры: **Elem** – индекс элемента.

void DelElem(const int Elem);

Назначение: удаление элемента из множества.

Входные параметры: **Elem** – индекс элемента.

bool IsMember(const int Elem) const;

Назначение: проверка, состоит ли элемент в множестве.

Входные параметры: **true** или **false**.

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

Назначение: перегрузка операции сравнения, сравнение на равенство объектов.

Входные параметры: **s** – объект класса **TSet**.

Выходные параметры: **true** или **false**.

bool operator!= (const TSet& s) const;

Назначение: перегрузка операции сравнения, сравнение на неравенство объектов.

Входные параметры: **s** – объект класса **TSet**.

Выходные параметры: **true** или **false**.

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

Назначение: присваивание значений объекта **s**.

Входные параметры: **s** – объект класса **TSet**.

Выходные параметры: ссылка на объект класса **TSet** (себя).

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

Назначение: добавление элемента в множество.

Входные параметры: **Elem** – индекс элемента.

Выходные параметры: объект класса **TSet**.

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

Назначение: удаление элемента из множества.

Входные параметры: **Elem** – индекс элемента.

Выходные параметры: объект класса **TSet**.

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

Назначение: объединение двух множеств.

Входные параметры: **s** – объект класса **TSet**.

Выходные параметры: объект класса **TSet**.

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

Назначение: пересечение двух множеств.

Входные параметры: **s** – объект класса **TSet**.

Выходные параметры: объект класса **TSet**.

```
TSet operator~ (void);
```

Назначение: получение дополнения к множеству.

Выходные параметры: объект класса **TSet**.

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

Назначение:

Входные параметры: **s** – объект класса **TSet**, **in** – поток ввода.

Выходные параметры: поток ввода.

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

Назначение:

Входные параметры: **s** – объект класса **TSet**, **out** – поток вывода.

Выходные параметры: поток вывода.

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. Лекция «Множества и битовые поля» Сысоева А.В.
[<https://cloud.unn.ru/s/DLRHnt54ircG2WL>].
2. Битовые поля [https://www.youtube.com/watch?v=_XJAeR7obBk].
3. Решето Эратосфена [<https://yandex.ru/video/preview/2908856360907561981>].

Приложения

Приложение А. Реализация класса TBitField

```
#include "tbitfield.h"

TBitField::TBitField(int len)
{
    if (len > 0) {
        BitLen = len;
        MemLen = ((len + bitsInElem - 1) >> shiftSize);
        pMem = new TELEM[MemLen];
        memset(pMem, 0, MemLen * sizeof(TELEM));
    }
    else if (len == 0) {
        BitLen = 0;
        MemLen = 0;
        pMem = nullptr;
    }
    else {
        throw "error: BitFields size < 0";
    }
}

TBitField::TBitField(const TBitField &bf) // конструктор копирования
{
    BitLen = bf.BitLen;
    MemLen = bf.MemLen;
    if (MemLen) {
        pMem = new TELEM[MemLen];
        memcpy(pMem, bf.pMem, MemLen * sizeof(TELEM));
    }
    else {
        pMem = nullptr;
    }
}

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

int TBitField::GetMemIndex(const int n) const noexcept // индекс Мем для бита
n
{
    return n >> shiftSize;
}

TELEM TBitField::GetMemMask(const int n) const noexcept // битовая маска для
бита n
{
    return 1 << (n & (bitsInElem - 1));
}

// 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 || n < 0) throw "error: index out of range";
    pMem[GetMemIndex(n)] |= GetMemMask(n);
}
```

```

void TBitField::ClrBit(const int n) // очистить бит
{
    if (n >= BitLen || n < 0) throw "error: index out of range";
    pMem[GetMemIndex(n)] &= ~GetMemMask(n);
}
int TBitField::GetBit(const int n) const // получить значение бита
{
    if (n >= BitLen || n < 0) throw "error: index out of range";
    return ((pMem[GetMemIndex(n)] & GetMemMask(n)) >> (n & (bitsInElem -
1)));
}

// битовые операции
const TBitField& TBitField::operator=(const TBitField &bf) // присваивание
{
    if (MemLen != bf.MemLen)
        if (MemLen > 0) {
            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);
}
bool TBitField::operator==(const TBitField &bf) const // сравнение
{
    if (BitLen != bf.BitLen) return false;
    bool flag = true;
    for (int i = 0; i < MemLen; i++)
        if (pMem[i] != bf.pMem[i]) {
            flag = false;
            break;
        }
    return flag;
}
bool TBitField::operator!=(const TBitField &bf) const // сравнение
{
    return !((*this) == bf);
}
TBitField TBitField::operator|(const TBitField &bf) // операция "или"
{
    int len = (BitLen > bf.BitLen) ? BitLen : bf.BitLen;

    TBitField res(len);
    for (int i = 0; i < BitLen; i++) {
        if (GetBit(i)) res.SetBit(i);
    }
    for (int i = 0; i < bf.BitLen; i++) {
        if (bf.GetBit(i)) res.SetBit(i);
    }
    return res;
}
TBitField TBitField::operator&(const TBitField &bf) // операция "и"
{
    int len = (BitLen > bf.BitLen) ? BitLen : bf.BitLen;
    int mlen = (BitLen < bf.BitLen) ? BitLen : bf.BitLen;

    TBitField res(len);
    for (int i = 0; i < BitLen; i++) {
        if (GetBit(i) && bf.GetBit(i)) res.SetBit(i);
    }
}

```

```

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

// ВВОД/ВЫВОД
istream& operator>>(istream& in, TBitField& bf) // ввод
{
    string ans;
    in >> ans;

    int blen = bf.BitLen;
    int len = (ans.size() < blen) ? ans.size() : blen;

    for (int i = 0; i < blen; i++) {
        bf.ClrBit(i);
    }
    for (int i = 0; i < len; i++) {
        if (ans[i] == '1') bf.SetBit(i);
    }

    return in;
}
ostream& operator<<(ostream& out, const TBitField& bf) // вывод
{
    int len = bf.BitLen;
    for (int i = 0; i < len; i++) {
        if (bf.GetBit(i))
            out << '1';
        else
            out << '0';
    }
    return out;
}

```

Приложение Б. Реализация класса TSet

```

#include "tset.h"

TSet::TSet(int mp) : BitField(mp)
{
    if (mp >= 0) {
        MaxPower = mp;
    }
    else {
        throw "error: Set size < 0";
    }
}

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

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

```

```

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

// доступ к битам
int TSet::GetMaxPower(void) const noexcept // получить макс. к-во эл-тов
{
    return MaxPower;
}
bool TSet::IsMember(const int Elem) const // элемент множества?
{
    if (Elem >= MaxPower || Elem < 0) throw "error: index out of range";

    if (BitField.GetBit(Elem))
        return true;
    return false;
}
void TSet::InsElem(const int Elem) // включение элемента множества
{
    if (Elem >= MaxPower || Elem < 0) throw "error: index out of range";
    BitField.SetBit(Elem);
}
void TSet::DelElem(const int Elem) // исключение элемента множества
{
    if (Elem >= MaxPower || Elem < 0) throw "error: index out of range";
    BitField.ClrBit(Elem);
}

// теоретико-множественные операции
const TSet& TSet::operator=(const TSet& s) // присваивание
{
    MaxPower = s.MaxPower;
    BitField = s.BitField;
    return *this;
}
bool TSet::operator==(const TSet& s) const // сравнение
{
    if (MaxPower != s.MaxPower) return false;
    return (BitField == s.BitField);
}
bool TSet::operator!=(const TSet& s) const // сравнение
{
    return !(*this == s);
}
TSet TSet::operator+(const TSet& s) // объединение
{
    int len = (MaxPower > s.MaxPower) ? MaxPower : s.MaxPower;
    TSet res(len);

    res.BitField = BitField | s.BitField;
    return res;
}
TSet TSet::operator+(const int Elem) // объединение с элементом
{
    if (Elem >= MaxPower || Elem < 0) throw "error: index out of range";

    TSet res(*this);
    res.InsElem(Elem);
    return res;
}
TSet TSet::operator-(const int Elem) // разность с элементом

```

```

{
    if (Elem >= MaxPower || Elem < 0) throw "error: index out of range";

    TSet res(*this);
    res.DelElem(Elem);
    return res;
}
TSet TSet::operator*(const TSet& s) // пересечение
{
    int len = (MaxPower > s.MaxPower) ? MaxPower : s.MaxPower;
    TSet res(len);

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

// перегрузка ввода/вывода
istream& operator>>(istream& in, TSet& s) // ввод
{
    in >> s.BitField;
    return in;
}
ostream& operator<<(ostream& out, const TSet& s) // вывод
{
    out << s.BitField;
    return out;
}

```

Приложение B. Sample_primenumbers

```

#include <iostream>
#include "tset.h"

int main()
{
    std::cout << "Prime numbers\n" << std::endl;

    int n;
    cout << "Enter the count of numbers: ";
    cin >> n;

    TSet s(n + 1);
    for (int m = 2; m <= n; m++)
        s.InsElem(m);
    for (int m = 2; m * m <= n; m++)
        if (s.IsMember(m))
            for (int k = 2 * m; k <= n; k += m)
                if (s.IsMember(k))
                    s.DelElem(k);

    cout << "Prime numbers under " << n << ": " << endl;
    for (int i = 0; i <= n; i++)
        if (s.IsMember(i))
            cout << i << ' ';
    cout << endl;
}

```

```
    return 0;  
}
```