

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 \_\_\_\_  
\_\_\_\_\_/Миронов А. И./  
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 «Решето Эратосфено» .....	8
3 Руководство программиста .....	9
3.1 Описание алгоритмов .....	9
3.1.1 Битовые поля .....	9
3.1.2 Множества .....	10
3.1.3 «Решето Эратосфена» .....	11
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:**

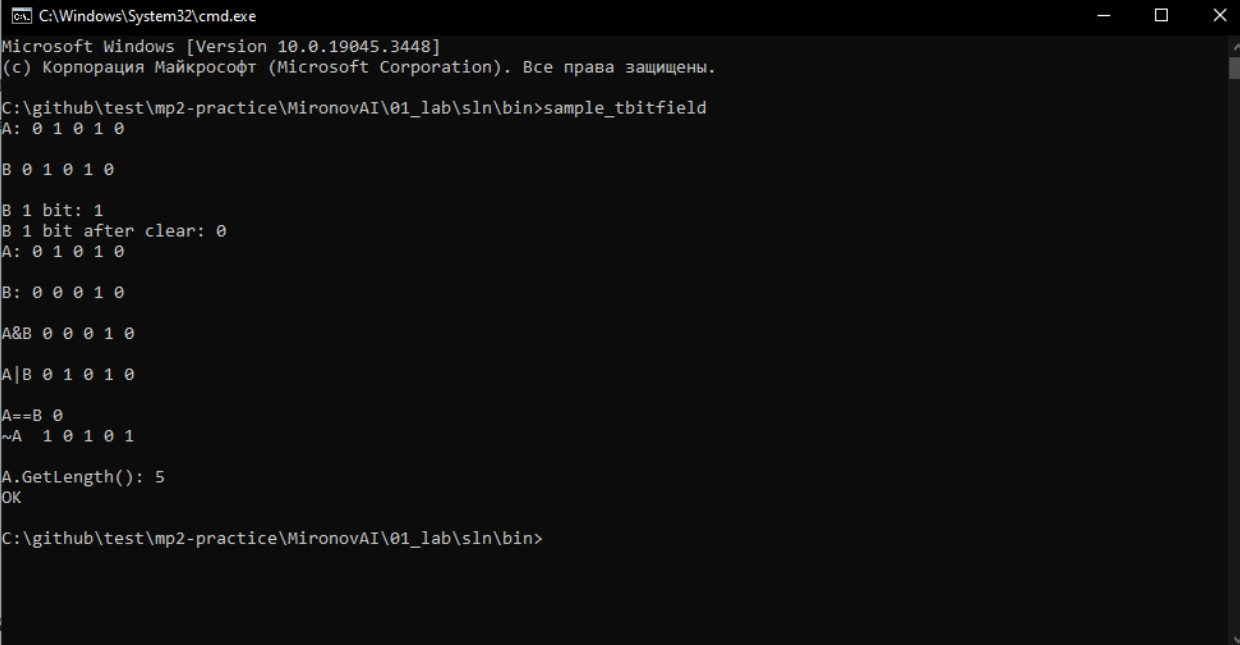
1. Изучить теоретические основы битовых полей и множеств.
2. Разработать программу, реализующую операции над битовыми полями и множествами.
3. Провести эксперименты с различными наборами данных.
4. Проанализировать полученные результаты и сделать выводы о преимуществах и ограничениях использования битовых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

## 2 Руководство пользователя

### 2.1 Приложение для демонстрации работы битовых полей

1. Запустите приложение с названием \*.exe. В результате появится окно, показанное ниже (Ошибка! Источник ссылки не найден.).

Рис. 1. Основное окно программы



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.3448]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\github\test\mp2-practice\MironovAI\01_lab\sln\bin>sample_tbitfield
A: 0 1 0 1 0

B 0 1 0 1 0

B 1 bit: 1
B 1 bit after clear: 0
A: 0 1 0 1 0

B: 0 0 0 1 0

A&B 0 0 0 1 0

A|B 0 1 0 1 0

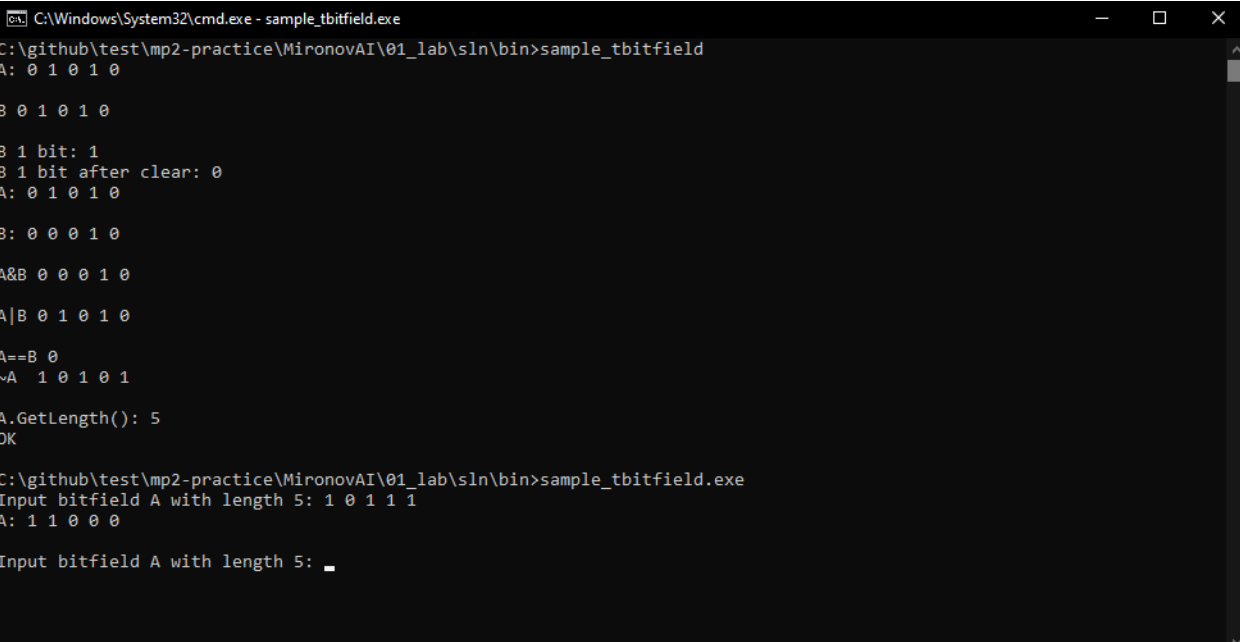
A==B 0
~A 1 0 1 0 1

A.GetLength(): 5
OK

C:\github\test\mp2-practice\MironovAI\01_lab\sln\bin>
```

2. Затем вам будет предложено ввести 2 битовых поля длины 5 (рис 2).

Рис. 2. Ввод битовых полей



```
C:\Windows\System32\cmd.exe - sample_tbitfield.exe
C:\github\test\mp2-practice\MironovAI\01_lab\sln\bin>sample_tbitfield
A: 0 1 0 1 0

B 0 1 0 1 0

B 1 bit: 1
B 1 bit after clear: 0
A: 0 1 0 1 0

B: 0 0 0 1 0

A&B 0 0 0 1 0

A|B 0 1 0 1 0

A==B 0
~A 1 0 1 0 1

A.GetLength(): 5
OK

C:\github\test\mp2-practice\MironovAI\01_lab\sln\bin>sample_tbitfield.exe
Input bitfield A with length 5: 1 0 1 1 1
A: 1 1 0 0 0

Input bitfield A with length 5: _
```

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

Рис. 3. Результат тестирования функций класса TBitField

```
C:\Windows\System32\cmd.exe
A==B 0
~A 1 0 1 0 1

A.GetLength(): 5
OK

C:\github\test\mp2-practice\MironovAI\01_lab\sln\bin>sample_tbitfield.exe
Input bitfield A with length 5: 1 0 1 1 1
A: 1 1 0 0 0

Input bitfield A with length 5: 1 0 1 0 1
B 1 1 0 0 0

B 1 bit: 1
B 1 bit after clear: 0
A: 1 1 0 0 0

B: 1 0 0 0 0

A&B 1 0 0 0 0

A|B 1 1 0 0 0

A==B 0
~A 0 0 1 1 1

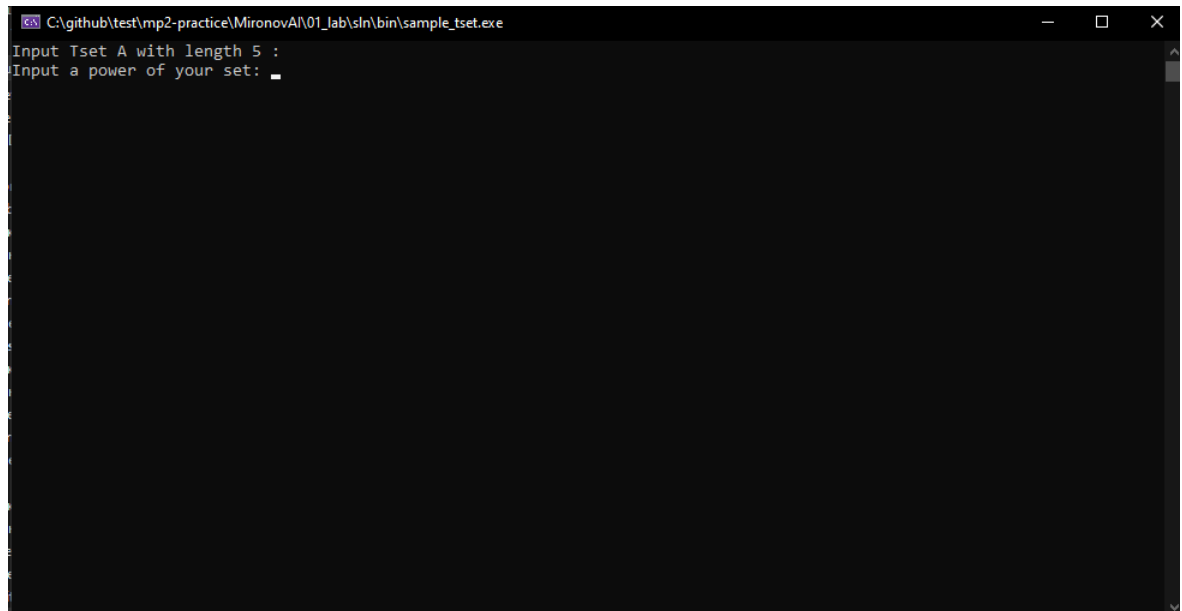
A.GetLength(): 5
OK

C:\github\test\mp2-practice\MironovAI\01_lab\sln\bin>
```

## 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 ввести свое множество. Необходимо ввести сначала количество чисел в множестве, программа будет ждать N чисел. Причём они не должны превышать 5. После ввода множества нулей и /или единиц, будут выведены результаты соответствующих операций и функций (рис 5).

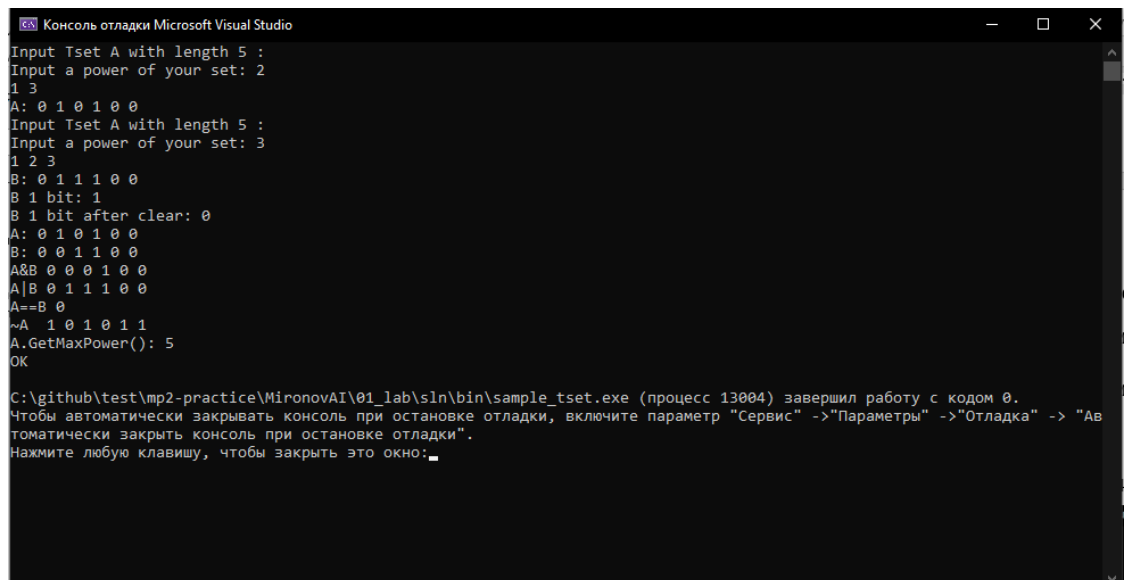


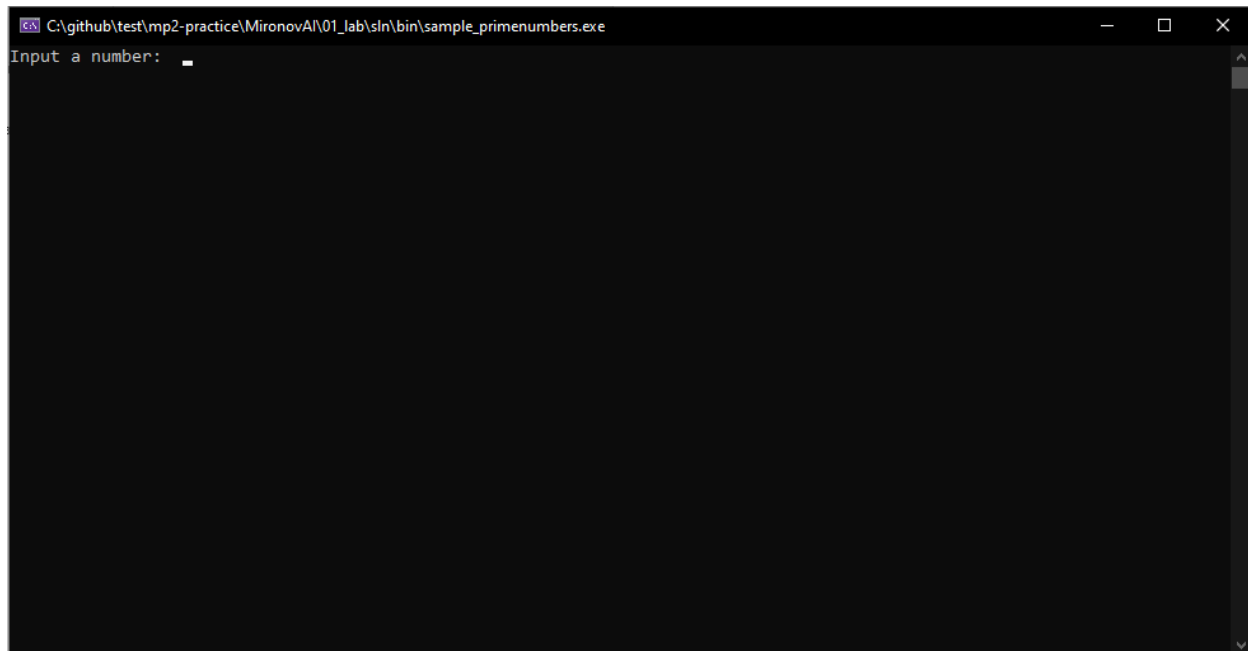
Рис. 5.

Результат тестирования класса **TSet**

## 2.3 «Решето Эратосфена»

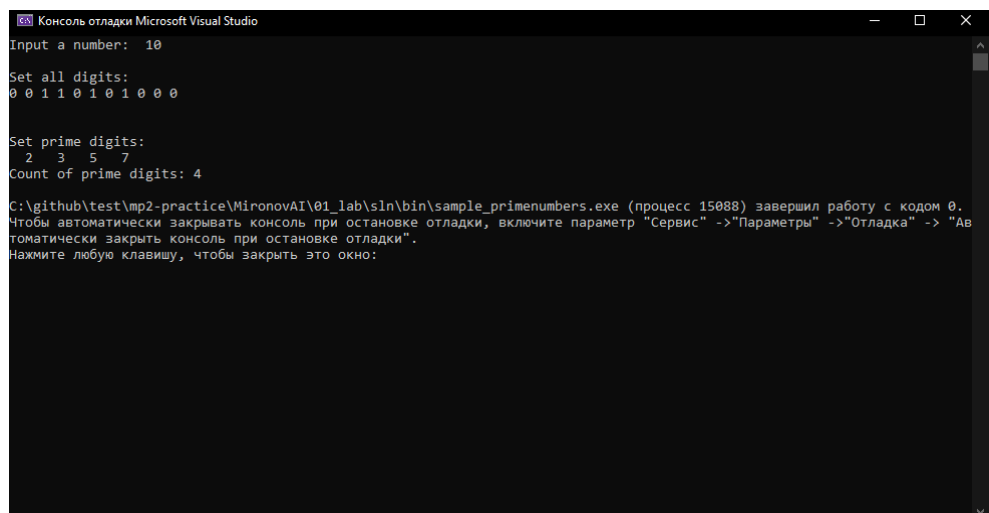
1. Запустите приложение с названием sample\_tset.exe. В результате появится окно, показанное ниже (рис 6).

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).

Рис. 7. Результат работы алгоритма «Решето Эратосфена»





## 3 Руководство программиста

### 3.1 Описание алгоритмов

#### 3.1.1 Битовые поля

1. Начало работы указано в пункте [2.1 этого документа](#)
2. Описание методов и полей класса в пункте [3.2.1 этого документа](#)

Программа алгоритм состоит из единственной функции **void test\_bitfield()**, которая сразу вызывается из основной функции **int main()**. Изначально необходимо алгоритм запрашивает пользователя ввести 2 битовых поля длиной 5 для демонстрации использования экземпляров класса (рис 8)

Рис. 8. Ввод данных алгоритма

```
TBitField a(5), b(5);
cout << "Input bitfield A with length 5: ";
cin >> a;
cout << "A: " << a << endl;
cout << "Input bitfield A with length 5: ";
cin >> b;
cout << "B " << b << endl;
```

3. После чего алгоритм последовательно выведет результат вызова соответствующих операций и методов класса (рис 9).

Рис. 9. Вывод результата работы алгоритма

```
cout << "B 1 bit: " << b.GetBit(1) << endl;

b.ClrBit(1);
cout << "B 1 bit after clear: " << b.GetBit(1) << endl;

cout << "A: " << a << endl;
cout << "B: " << b << endl;
cout << "A&B " << (a & b) << endl;
cout << "A|B " << (a | b) << endl;
cout << "A==B " << (a == b) << endl;
cout << "~A " << (~a) << endl;
cout << "A.GetLength(): " << a.GetLength() << endl;
cout << "OK" << endl;
```

### 3.1.2 Множества

1. Начало работы указано в пункте [2.2 этого документа](#).
2. Описание методов и полей класса в пункте [3.2.2 этого документа](#)

Программа алгоритм состоит из единственной функции `void test_tset()`, которая сразу вызывается из основной функции `int main()`. Изначально необходимо алгоритм запрашивает пользователя ввести 2 множества, элементами которых не могут превышать 5 и должны быть целочисленным целым числом, для демонстрации использования экземпляров класса (рис 10)

Рис. 10. Ввод данных алгоритма

```
TSet a(5), b(5);
cout << "Input Tset A with max power 5 : " << endl;
cin >> a;
cout << "A: " << a << endl;
cout << "Input Tset B with max power 5 : " << endl;
cin >> b;
cout << "B: " << b << endl;
```

3. После чего алгоритм последовательно выведет результат вызова соответствующих операций и методов класса (рис 11).

Рис. 11. Результат работы алгоритма

```
cout << "B 1 bit: " << a.IsMember(1) << endl;
b.Delete(1);
cout << "B 1 bit after clear: " << b.IsMember(1) << endl;

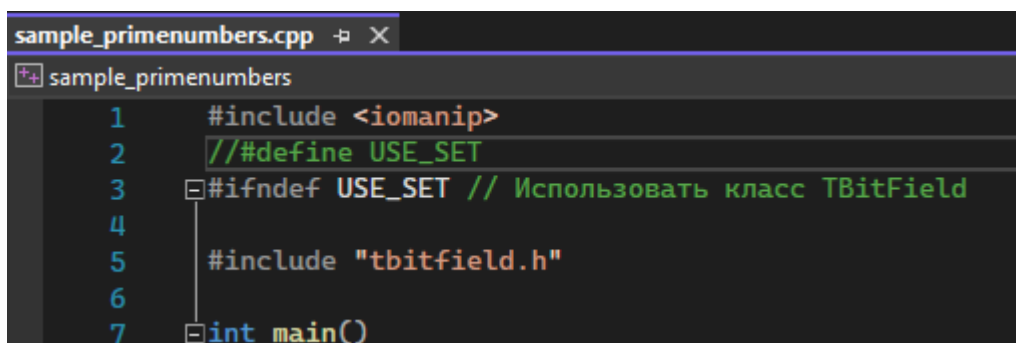
cout << "A: " << a << endl;
cout << "B: " << b << endl;
cout << "A&B " << (a * b) << endl;
cout << "A|B " << (a + b) << endl;
cout << "A==B " << (a == b) << endl;
cout << "~A " << (~a) << endl;

cout << "A.GetMaxPower(): " << a.GetMaxPower() << endl;
cout << "OK" << endl;
```

### 3.1.3 «Решето Эратосфена»

1. Начало работы указано в пункте [2.3 текущего документа](#)
2. Данный алгоритм реализован двумя способами, при помощи классов **TSet** и **TBitField**. Изначально алгоритм будет использовать класс **TBitField**. Для того, чтобы использовать реализацию с использованием **TSet**, необходимо убрать два слеша перед **#define USE\_SET** (рис 12).

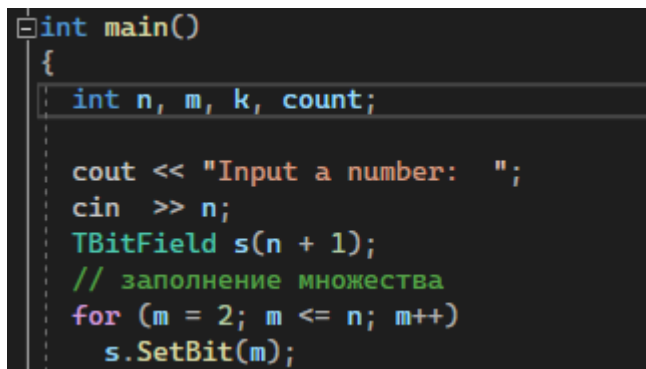
Рис. 12. Использование **TSet**



```
sample_primenumbers.cpp
sample_primenumbers
1  #include <iomanip>
2  // #define USE_SET
3  #ifndef USE_SET // Использовать класс TBitField
4
5  #include "tbitfield.h"
6
7  int main()
```

3. Реализация алгоритма состоит из ввода необходимого числа, создание экземпляра класса **TBitField** или **TSet** соответственно. После чего алгоритм заполняет все элементы классов равными 1 (рис 13).

Рис. 13. Ввод данных и заполнение экземпляра



```
int main()
{
    int n, m, k, count;

    cout << "Input a number: ";
    cin >> n;
    TBitField s(n + 1);
    // заполнение множества
    for (m = 2; m <= n; m++)
        s.SetBit(m);
}
```

4.
  1. После чего алгоритм , начинает перебирать все числа от 2 до N. Если это число есть в нашем множестве, то мы переходим к шагу 2, иначе к шагу 3.
  2. Это число, и дальше все кратные ему числа удаляются из нашего множества.
  3. Выбирается следующее число. Если это число больше N, то алгоритм заканчивается, иначе – переход к шагу 1. (рис 14)

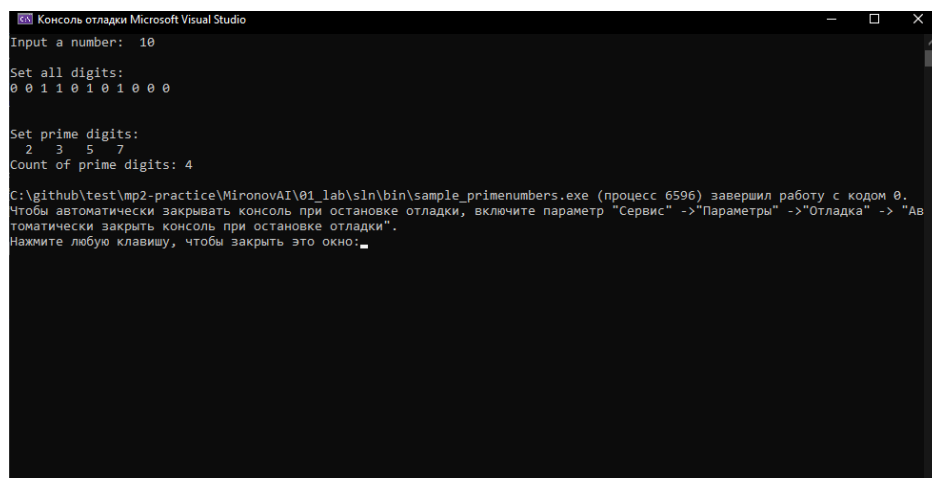
Рис. 14. Алгоритм «Решето Эратосфена»

```
for (m = 2; m * m <= n; m++)
    // если m в s, удаление кратных
    if (s.GetBit(m))
        for (k = 2 * m; k <= n; k += m)
            if (s.GetBit(k))
                s.ClrBit(k);

// оставшиеся в s элементы – простые числа
cout << endl << "Set all digits: " << endl << s << endl;
cout << endl << "Set prime digits: " << endl;
count = 0;
k = 1;
for (m = 2; m <= n; m++)
    if (s.GetBit(m))
    {
        count++;
        cout << setw(3) << m << " ";
        if (k++ % 10 == 0)
            cout << endl;
    }
cout << endl;
cout << "Count of prime digits: " << count << endl;
```

5. После чего выводятся все числа с идентификатором (0, если оно не простое, 1 если простое), затем выводятся все простые числа и количество (рис 15)

Рис. 15. Вывод данных



```
Консоль отладки Microsoft Visual Studio
Input a number: 10

Set all digits:
0 0 1 1 0 1 0 1 0 0 0

Set prime digits:
 2  3  5  7

Count of prime digits: 4

C:\github\test\mp2-practice\MironovAI\01_lab\sln\bin\sample_primenumbers.exe (процесс 6596) 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.
```

## 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 unsigned int

**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. 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}, т.е. если i бит исходного экземпляра будет равен 1 , то на выходе он будет иметь 0.

**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. 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:

&s - множество;

Выходные параметры: Экземпляр класса , который равен { \*this | s }

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

Назначение: оператор пересечения множеств

Входные параметры:

&s - множество;

Выходные параметры: Экземпляр класса , который равен { \*this & s }  
}

**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

```
operator TBitField();
```

Назначение: вывод поля **BitField**

Входные параметры отсутствуют

Выходные данные: поле **BitField**

```
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. Были изучены теоретические основы битовых полей и множеств.
2. Была 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. Были 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. Википедия [[https://ru.wikipedia.org/wiki/Битовое\\_поле](https://ru.wikipedia.org/wiki/Битовое_поле)].

# Приложения

## Приложение А. Реализация класса TBitField

Рис. 16. Реализация класса TBitField 1

```
1  #include "tbitfield.h"
2  #define size 32 //sizeof(ui)
3
4  TBitField::TBitField(int len)
5  {
6      if (len <= 0)
7          throw "len_is_LEq_zero!";
8      BitLen = len;
9      MemLen = (BitLen - 1) / (size) + 1;
10     pMem = new TELEM[MemLen];
11
12     // зануление битов (может из-за поведения заполнить "мусором")
13     for (int i = 0; i < MemLen; ++i)
14     {
15         pMem[i] = 0;
16     }
17 }
18
19 TBitField::TBitField(const TBitField& bf) // конструктор копирования
20 {
21     BitLen = bf.BitLen;
22     MemLen = bf.MemLen;
23     pMem = new TELEM[MemLen];
24     for (int i = 0; i < MemLen; ++i)
25     {
26         pMem[i] = bf.pMem[i];
27     }
28 }
29
30 TBitField::~TBitField()
31 {
32     delete[] pMem;
33 }
34
35 int TBitField::GetMemIndex(const int n) const // индекс Мем для бита n
36 {
37     if (n < 0)
38         throw "n_is_below_zero";
39     return n / (size);
40 }
```

Рис. 17. Реализация класса TBitField 2

```
42 TELEM TBitField::GetMemMask(const int n) const // битовая маска для бита n
43 {
44     if (n > BitLen)
45         throw "overload_n";
46     if (n < 0)
47         throw "n_is_below_zero";
48     TELEM Mask = 1u << (size - n % size - 1);
49     return Mask;
50 }
51
52 // доступ к битам битового поля
53
54 int TBitField::GetLength(void) const // получить длину (к-во битов)
55 {
56     return BitLen;
57 }
58
59 void TBitField::SetBit(const int n) // установить бит
60 {
61     if (n > BitLen)
62         throw "overload_n";
63     if (n < 0)
64         throw "n_is_below_zero";
65
66     pMem[GetMemIndex(n)] |= GetMemMask(n);
67 }
68
69 void TBitField::ClrBit(const int n) // очистить бит
70 {
71     if (n > BitLen)
72         throw "overload_n";
73     if (n < 0)
74         throw "n_is_below_zero";
75     pMem[GetMemIndex(n)] &= ~(GetMemMask(n));
76 }
77
78 int TBitField::GetBit(const int n) const // получить значение бита
79 {
80     if (n > BitLen)
81         throw "overload_n";
82     if (n < 0)
83         throw "n_is_below_zero";
84     return (pMem[GetMemIndex(n)] & GetMemMask(n)) >> (size - n % size - 1);
85 }
```

Рис. 18. Реализация класса TBitField 3

```

88
89 const TBitField& TBitField::operator=(const TBitField& bf) // присваивание
90 {
91     if (*this == bf) return *this;
92     if (BitLen != bf.BitLen)
93     {
94         delete[] pMem;
95         pMem = new TELEM[MemLen];
96     }
97     BitLen = bf.BitLen;
98     MemLen = bf.MemLen;
99
100     for (int i = 0; i < MemLen; ++i)
101     {
102         pMem[i] = bf.pMem[i];
103     }
104     return *this;
105 }
106
107 int TBitField::operator==(const TBitField& bf) const // сравнение
108 {
109     if (BitLen != bf.GetLength())
110         return 0;
111     else
112     {
113         for (int i = 0; i < BitLen; ++i)
114         {
115             if (GetBit(i) != bf.GetBit(i))
116                 return 0;
117         }
118         return 1;
119     }
120 }
121
122 int TBitField::operator!=(const TBitField& bf) const // сравнение
123 {
124     return !(*this == bf);
125 }
126
127

```

Рис. 19. Реализация класса TBitField 4

```

128 TBitField TBitField::operator|(const TBitField& bf) // операция "или"
129 {
130     const int maxlen = max(GetLength(), bf.GetLength());
131     const int minlen = min(GetLength(), bf.GetLength());
132     TBitField A(maxlen);
133     if (GetLength() > bf.GetLength())
134         A = *this;
135     else
136         A = bf;
137     int i = 0;
138     for (; i < minlen; ++i)
139     {
140         if (GetBit(i) || bf.GetBit(i))
141             A.SetBit(i);
142     }
143     return A;
144 }
145
146 TBitField TBitField::operator&(const TBitField& bf) // операция "и"
147 {
148     const int maxlen = max(GetLength(), bf.GetLength());
149     const int minlen = min(GetLength(), bf.GetLength());
150     TBitField A(maxlen);
151     int i = 0;
152     for (; i < minlen; ++i)
153     {
154         if (GetBit(i) && bf.GetBit(i))
155             A.SetBit(i);
156     }
157     return A;
158 }
159
160 TBitField TBitField::operator~(void) // отрицание
161 {
162     TBitField A(GetLength());
163     for (int i = 0; i <= GetMemIndex(GetLength()); ++i)
164     {
165         A.pMem[i] = ~pMem[i];
166     }
167     return A;
168 }
169

```



Рис. 20. Реализация класса TBitField 5

```
170 // ввод/вывод
171
172 istream& operator>>(istream& istr, TBitField& bf) // ввод
173 {
174     for (int i = 0; i < bf.GetLength(); ++i)
175     {
176         int val;
177         istr >> val;
178         if (val > bf.GetLength() || val < 0)
179             throw "Wrong element ";
180         bf.SetBit(val);
181     }
182     return istr;
183 }
184
185 ostream& operator<<(ostream& ostr, const TBitField& bf) // вывод
186 {
187     for (int i = 0; i < bf.GetLength(); ++i)
188     {
189         ostr << bf.GetBit(i) << " ";
190     }
191     ostr << "\n";
192     return ostr;
193 }
```

## Приложение Б. Реализация класса TSet

Рис. 21. Реализация класса Tset 1

```
4  TSet::TSet(int mp) : BitField(mp)
5  {
6      MaxPower = mp;
7  }
8
9  // конструктор копирования
10 TSet::TSet(const TSet& s) : BitField(s.BitField)
11 {
12     MaxPower = s.GetMaxPower();
13 }
14
15 // конструктор преобразования типа
16 TSet::TSet(const TBitField& bf) : BitField(bf)
17 {
18     MaxPower = bf.GetLength();
19 }
20
21 TSet::operator TBitField()
22 {
23     return BitField;
24 }
25
26 int TSet::GetMaxPower(void) const // получить макс. к-во эл-тов
27 {
28     return MaxPower;
29 }
30
31 int TSet::IsMember(const int Elem) const // элемент множества?
32 {
33     if (Elem < 0 && Elem >= MaxPower)
34         throw "Out of Range";
35     return BitField.GetBit(Elem);
36 }
37
38 void TSet::InsElem(const int Elem) // включение элемента множества
39 {
40     if (Elem < 0 && Elem >= MaxPower)
41         throw "Out of Range";
42     return BitField.SetBit(Elem);
43 }
44
```

Рис. 22. Реализация класса TSet 2

```

45 void TSet::DelElem(const int Elem) // исключение элемента множества
46 {
47     if (Elem < 0 && Elem >= MaxPower)
48         throw "Out of Range";
49     return BitField.ClrBit(Elem);
50 }
51
52 // теоретико-множественные операции
53
54 const TSet& TSet::operator=(const TSet& s) // присваивание
55 {
56     if (*this == s) return *this;
57     MaxPower = s.GetMaxPower();
58     BitField = TBitField(MaxPower);
59     BitField = BitField | s.BitField; // bitfield = s.bitfield
60     return *this;
61 }
62
63 int TSet::operator==(const TSet& s) const // сравнение
64 {
65     if (MaxPower != s.GetMaxPower())
66         return 0;
67
68     for (int i = 0; i < MaxPower; ++i)
69     {
70         if (BitField.GetBit(i) != s.BitField.GetBit(i))
71             return 0;
72     }
73     return 1;
74 }
75
76 int TSet::operator!=(const TSet& s) const // сравнение
77 {
78     return !(*this == s);
79 }
80
81 TSet TSet::operator+(const TSet& s) // объединение
82 {
83     TSet A(max(MaxPower, s.GetMaxPower()));
84     A.BitField = BitField | s.BitField;
85     return A;
86 }
87

```

Рис. 23. Реализация класса TSet 3

```

88 TSet TSet::operator+(const int Elem) // объединение с элементом
89 {
90     if (Elem < 0 && Elem > MaxPower)
91         throw "Out of Range";
92     TSet A(*this);
93     A.BitField.SetBit(Elem);
94     return A;
95 }
96
97 TSet TSet::operator-(const int Elem) // разность с элементом
98 {
99     if (Elem < 0 && Elem > MaxPower)
100         throw "Out of Range";
101     TSet A(*this);
102     A.BitField.ClrBit(Elem);
103     return A;
104 }
105
106 TSet TSet::operator*(const TSet& s) // пересечение
107 {
108     TSet A(max(MaxPower, s.GetMaxPower()));
109     A.BitField = BitField & s.BitField;
110     return A;
111 }
112
113 TSet TSet::operator~(void) // дополнение
114 {
115     // BitField = ~BitField;
116     // return *this;
117     TSet A(MaxPower);
118     A.BitField = ~BitField;
119     return A;
120 }

```

Рис. 24. Реализация класса TSet 4

```
124  istream& operator>>(istream& istr, TSet& s) // ввод
125  {
126      const int x = s.MaxPower;
127      int k;
128      cout << "Input a power of your set: ";
129      cin >> k;
130
131      for (int i = 0; i < k; ++i)
132      {
133          int val; istr >> val;
134          if (val > s.MaxPower)
135              throw "Wrong element ";
136          s.InsElem(val);
137      }
138      return istr;
139  }
140
141  ostream& operator<<(ostream& ostr, const TSet& s) // вывод
142  {
143      const int x = s.MaxPower;
144      for (int i = 0; i <= x; ++i)
145      {
146          ostr << s.IsMember(i) << " ";
147      }
148      return ostr;
149  }
```