

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

_____ / Лысов И.М. /
Подпись

Проверил: к.т.н., доцент каф. ВВиСП

_____ / Кустикова В.Д. /
Подпись

Нижний Новгород
2023

Оглавление

Введение	3
1. Постановка цели и задач.....	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»	11
3. Руководство программиста	12
3.1 Используемые алгоритмы	12
3.1.1. Битовые поля.....	12
3.1.2. Множества.....	12
3.1.3. Алгоритм «решето Эратосфена».....	12
3.2 Использование классов	13
3.2.1 Класс TbitField	13
3.2.2 Класс Tset	16
Заключение.....	20
Используемая Литература	21
Приложение.....	22

Введение

Битовое поле - это структура данных, состоящая из одного или нескольких смежных битов, которые были выделены для определенных целей, так что любой отдельный бит или группа битов в структуре могут быть установлены или проверены. 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.

Теория множеств – учение об общих свойствах множеств – преимущественно бесконечных. 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.

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.

Важным преимуществом использования битовых операций является тот факт, что позволяют выполнять различные манипуляции с битами, такие как установка, сброс или инвертирование конкретных битов.

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. В целом можно сказать, что битовые поля и множества имеют широкое применение в различных областях, особенно в программировании и компьютерных системах. Они позволяют эффективно использовать память и упрощают работу с большим количеством данных.

1. Постановка цели и задач

Цель: Изучить битовые поля и множества. Получить навык практического применения данных структур данных.

Задачи:

1. Изучить основные принципы работы с битовыми полями и множествами.
2. Разработать программу, которая будет реализовывать операции над битовыми полями и множествами.
3. Протестировать корректность выполнения программы на различных примерах и тестах.
4. Применить полученные знания и реализовать алгоритм «решето Эратосфена» отбора простых элементов из заданного множества элементов.
5. Сделать выводы о проделанной работе.

2. Руководство Пользователя

2.1 Приложение для демонстрации работы битовых полей

1. Запустить sample_bitfield.exe. В результате появится следующее окно (См. Рис.1)

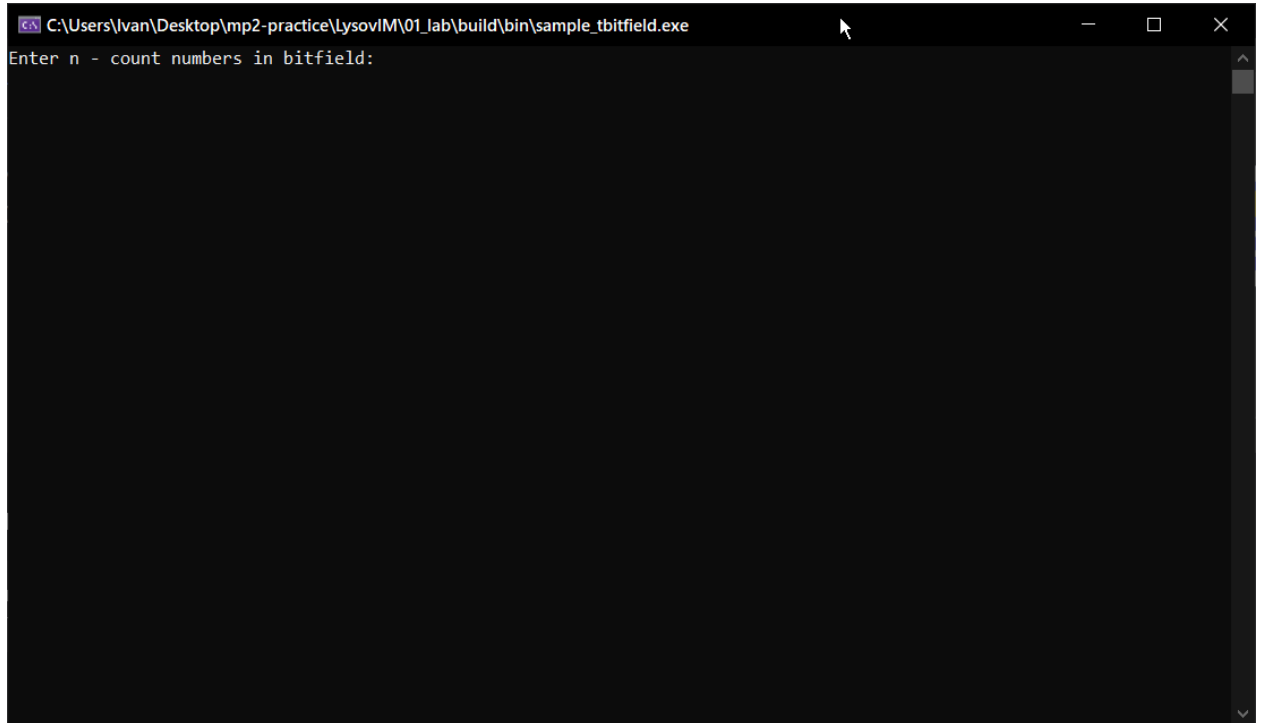
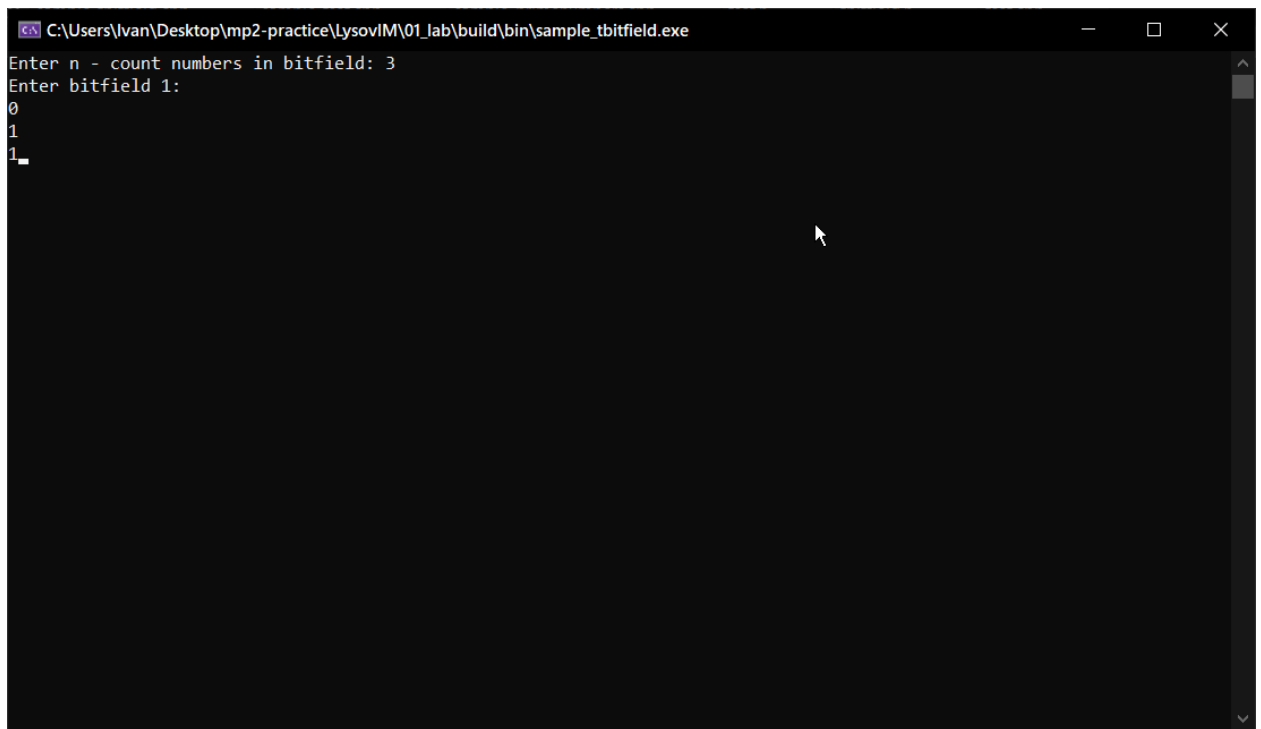


Рис.1 Основное меню приложения для работы с битовыми полями

На этом шаге необходимо ввести количество элементов в битовом поле (n). Далее это число будет использоваться для второго поля как $(n+1)$ элементов в битовом поле.

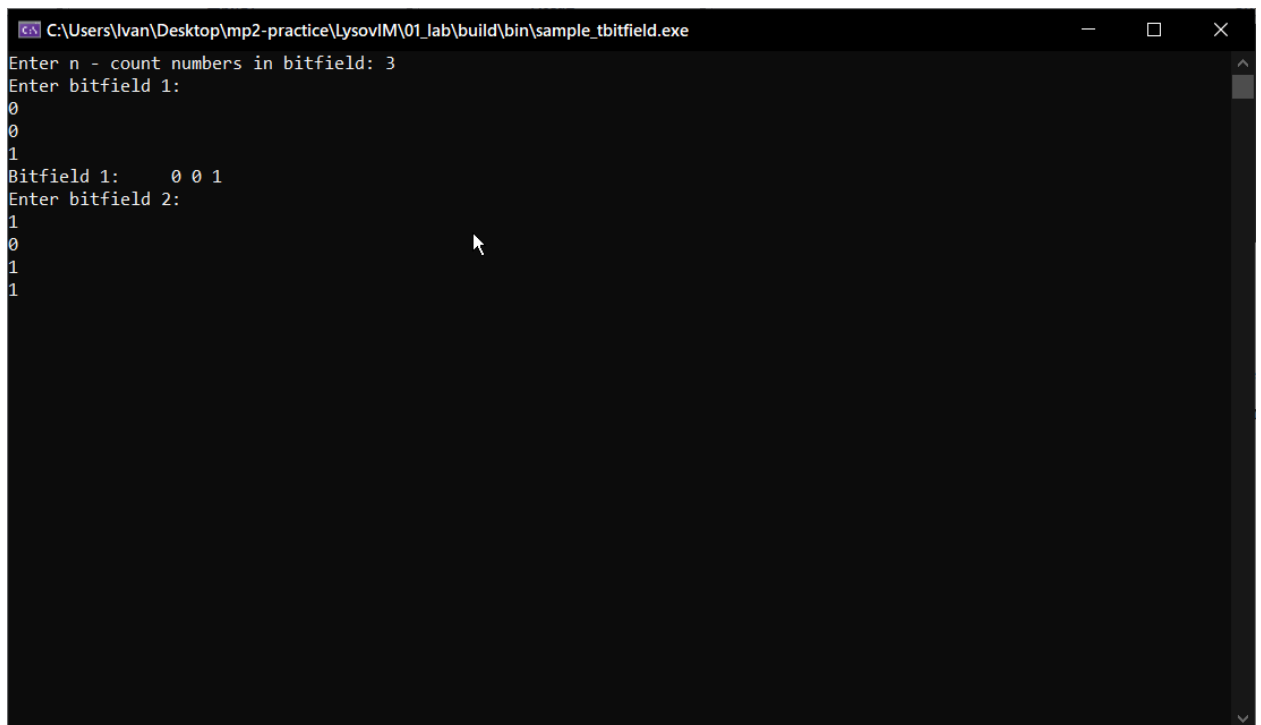
2. После ввода числа элементов в множестве программа требует ввести соответствующие битовые значения для первого поля (См. Рис.2)



```
C:\Users\Ivan\Desktop\mp2-practice\LysovIM\01_lab\build\bin\sample_tbitfield.exe
Enter n - count numbers in bitfield: 3
Enter bitfield 1:
0
1
1_
```

Рис.2 Ввод значений первого битового поля

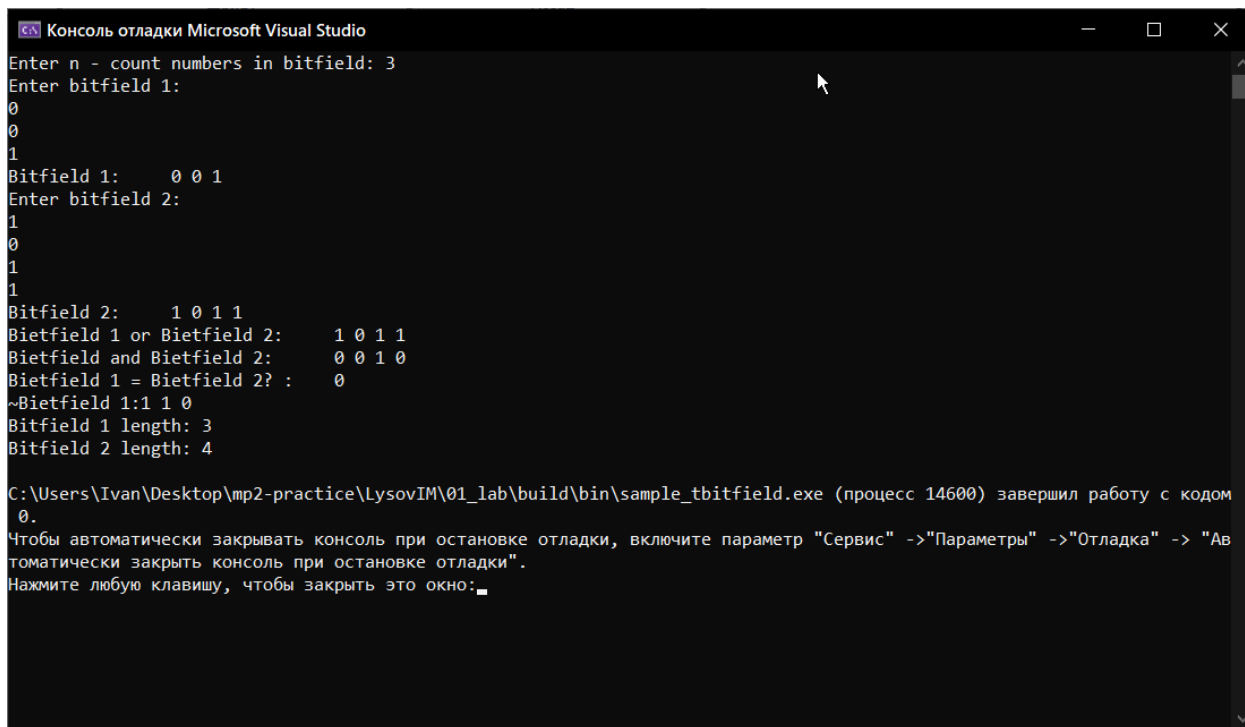
3. После ввода значений первого битового поля, программа выводит значение этого поля. Далее программа просит ввести значения второго битового поля, причем для $(n+1)$ элементов. (См. Рис. 3)



```
C:\Users\Ivan\Desktop\mp2-practice\LysovIM\01_lab\build\bin\sample_tbitfield.exe
Enter n - count numbers in bitfield: 3
Enter bitfield 1:
0
0
1
Bitfield 1: 0 0 1
Enter bitfield 2:
1
0
1
1_
```

Рис. 3 Ввод значений второго битового поля

4. После нажатия клавиши Enter программа выводит результаты операций работы с битовыми полями. Используются следующие операции (“или”, “и”, проверка равны ли два поля, отрицание первого битового поля). (См. Рис.4)



```
Консоль отладки Microsoft Visual Studio
Enter n - count numbers in bitfield: 3
Enter bitfield 1:
0
0
1
Bitfield 1:    0 0 1
Enter bitfield 2:
1
0
1
1
Bitfield 2:    1 0 1 1
Bitfield 1 or Bitfield 2:    1 0 1 1
Bitfield and Bitfield 2:    0 0 1 0
Bitfield 1 = Bitfield 2? :  0
~Bitfield 1:1 1 0
Bitfield 1 length: 3
Bitfield 2 length: 4

C:\Users\Ivan\Desktop\mp2-practice\LysovIM\01_lab\build\bin\sample_tbitfield.exe (процесс 14600) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно: _
```

Рис. 4 Результат работы программы

2.2 Приложение для демонстрации работы множеств

1. Запустить sample_tset.exe. В результате появится следующее окно (См. Рис.5)

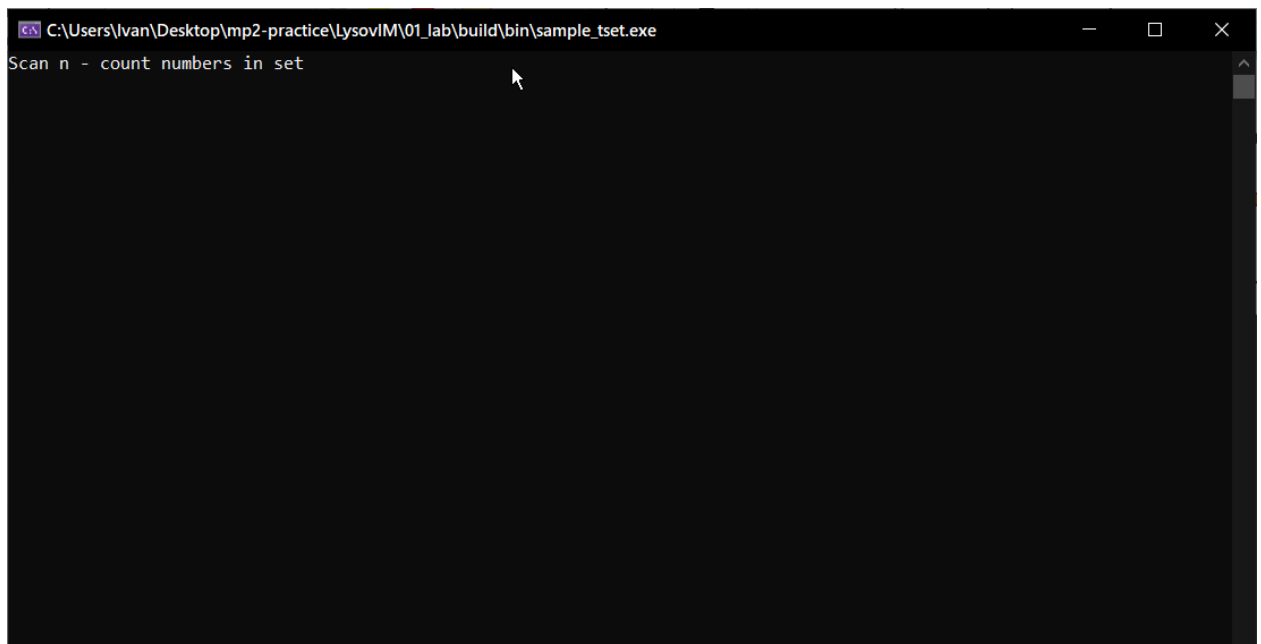


Рис.5 Основное меню приложения для работы с множествами
На этом шаге необходимо ввести количество элементов в битовом поле (n). Далее это число будет использоваться для второго поля как $(n+1)$ элементов в битовом поле.

2. После ввода числа элементов в множестве программа требует ввести соответствующие значения для первого множества (См. Рис.6)

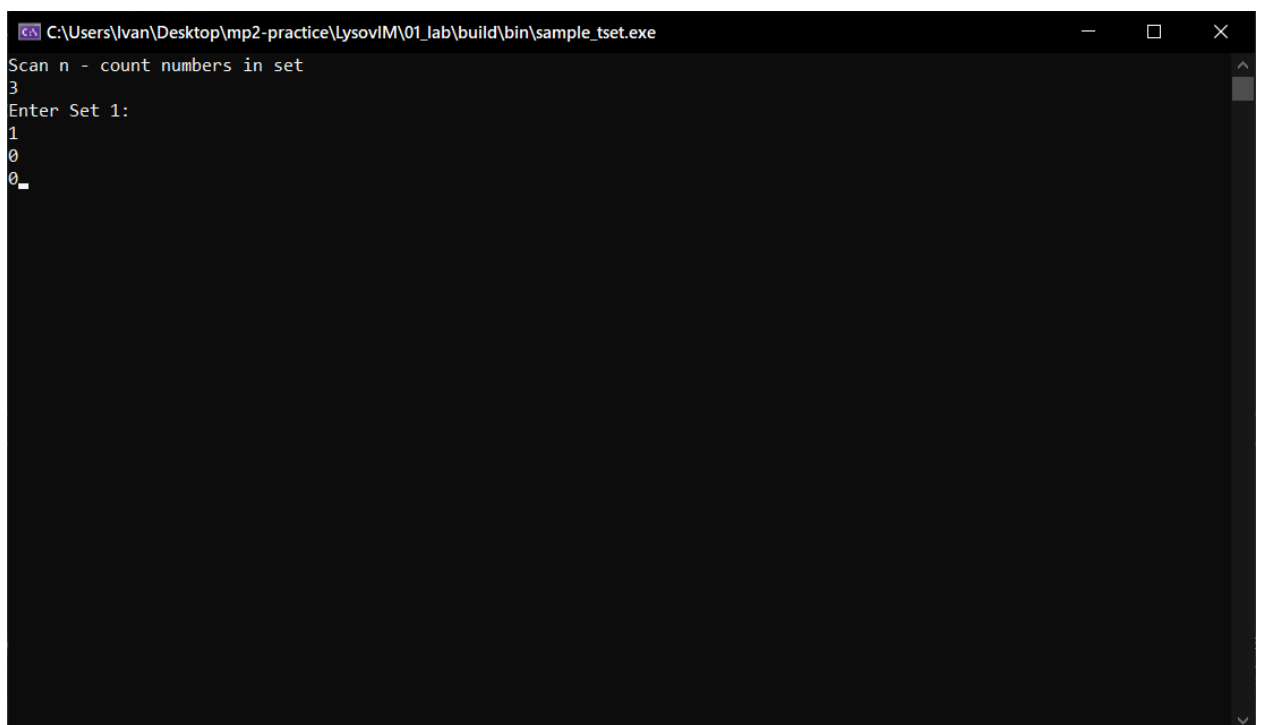


Рис.6 Ввод значений первого множества

3. После ввода значений первого множества, программа выводит значение этого множества, с помощью бит (0 – нет элемента в множестве, 1 – есть элемент в

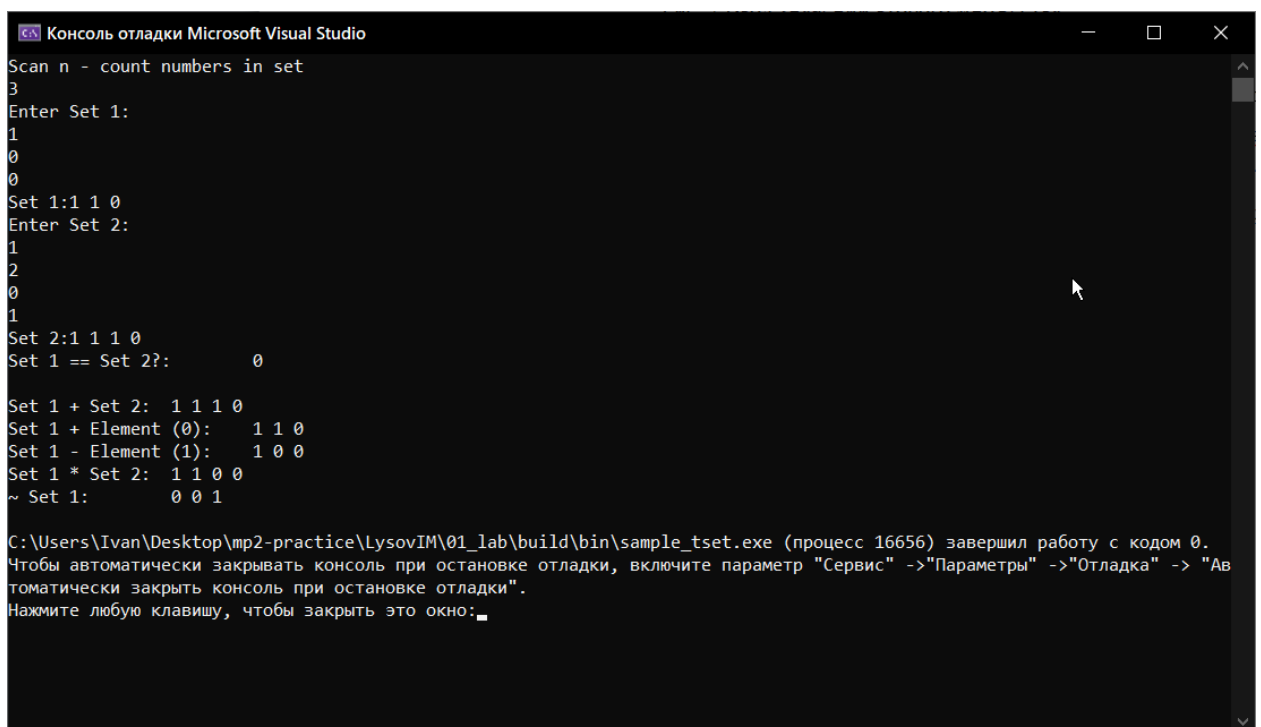
множестве). Далее программа просит ввести значения второго множества, причем для $(n+1)$ элементов. (См. Рис. 7)



```
C:\Users\Ivan\Desktop\mp2-practice\LysovIM\01_lab\build\bin\sample_tset.exe
Scan n - count numbers in set
3
Enter Set 1:
1
0
0
Set 1: 1 1 0
Enter Set 2:
1
2
0
1
```

Рис. 7 Ввод значений второго множества

- После нажатия клавиши Enter программа выводит результаты операций работы с битовыми полями. Используются следующие операции (проверка на равенства двух множеств, Множество 1 || множество 2, множество 1 + Элемент (0), множество 1 - Элемент (1), Множество 1 & множество 2, отрицание первого множества). (См. Рис.8)



```
Консоль отладки Microsoft Visual Studio
Scan n - count numbers in set
3
Enter Set 1:
1
0
0
Set 1: 1 1 0
Enter Set 2:
1
2
0
1
Set 2: 1 1 1 0
Set 1 == Set 2?:      0

Set 1 + Set 2:  1 1 1 0
Set 1 + Element (0):  1 1 0
Set 1 - Element (1):  1 0 0
Set 1 * Set 2:  1 1 0 0
~ Set 1:      0 0 1

C:\Users\Ivan\Desktop\mp2-practice\LysovIM\01_lab\build\bin\sample_tset.exe (процесс 16656) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно: _
```

Рис. 8 Результат работы программы

2.3 Приложение «решето Эратосфена»

1. Запустите sample_primenumbers.exe. В результате появится следующее окно (См. Рис.9):

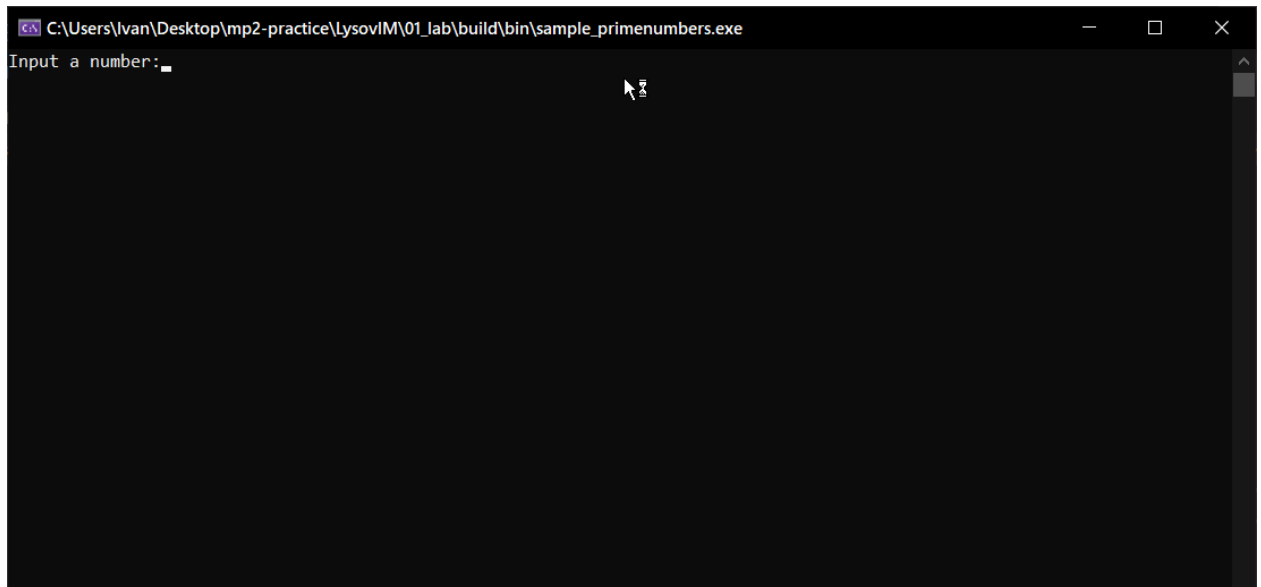


Рис.9 Основное окно приложения

2. Далее необходимо ввести целое положительное число для того, чтобы получить все простые числа до введенного числа (См. Рис.10)

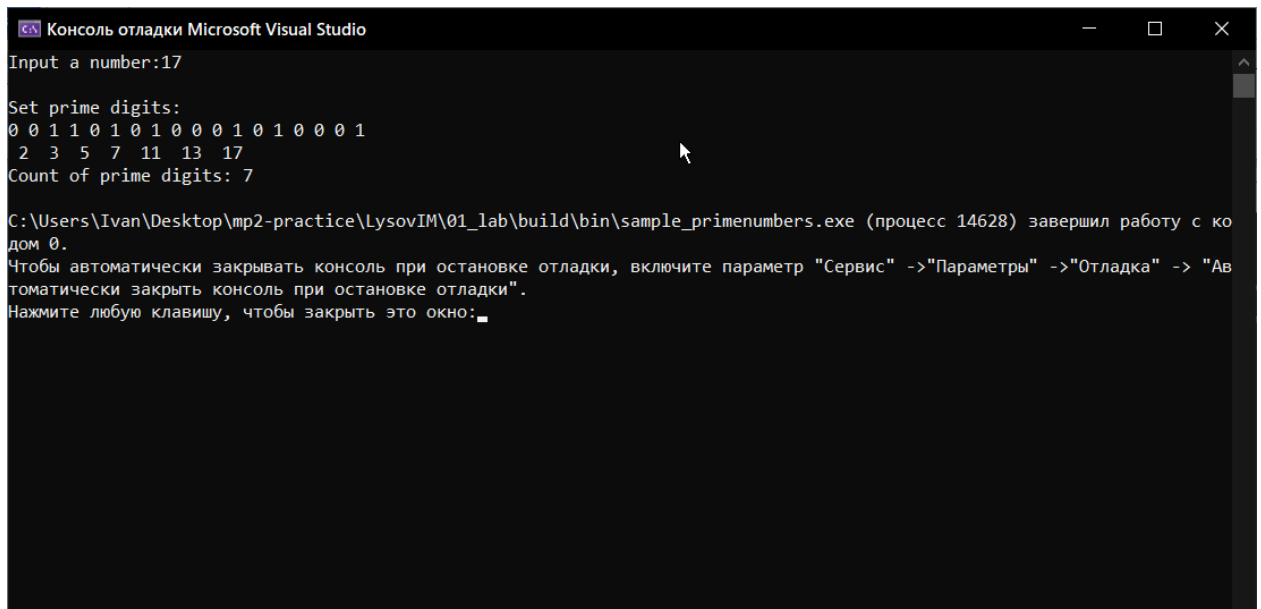


Рис.10 Результат работы алгоритма «решето Эратосфена»

3. Руководство программиста

3.1 Используемые алгоритмы

3.1.1. Битовые поля

Класс битового поля (TbitField) содержит в себе длину битового поля, память и количество элементов, содержащемуся в нем, где каждый элемент может быть равен 0 или 1. С помощью бит реализуются основные операции, где эти биты применяются. Реализация операций представлена в конце работы.

3.1.2. Множества

Класс множества (Tset) основан на классе битовых полей (TbitField). На нем основаны теоретико-множественные операции (объединение, пересечение, отрицание и т.д.), получение максимальной длины, а также добавление и удаление элементов, сравнение на равенство и ввод, вывод. Реализация операций представлена в конце работы.

3.1.3. Алгоритм «решето Эратосфена»

Решето Эратосфена — алгоритм нахождения всех простых чисел до некоторого целого числа n , который приписывают древнегреческому математику Эратосфену Киренскому. Как и во многих случаях, здесь 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
- 2) Заполнение множества
- 3) Проверка до квадратного корня и удаление кратных членов
- 4) Полученные элементы будут простыми числами

3.2 Использование 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 (#02)
    TELEM GetMemMask (const int n) const; // битовая маска для бита n (#03)
public:
    TBitField(int Len); // (#01)
    TBitField(const TBitField &bf); // (#П1)
    ~TBitField(); // (#С)
    // доступ к битам
    int GetLength(void) const; // получить длину (к-во битов) (#0)
    void SetBit(const int n); // установить бит (#04)
    void ClrBit(const int n); // очистить бит (#П2)
    int GetBit(const int n) const; // получить значение бита (#Л1)
    // битовые операции
    int operator==(const TBitField &bf) const; // сравнение (#05)
    int operator!=(const TBitField &bf) const; // сравнение
    TBitField& operator=(const TBitField &bf); // присваивание (#П3)
    TBitField operator|(const TBitField &bf); // операция "или" (#06)
    TBitField operator&(const TBitField &bf); // операция "и" (#Л2)
    TBitField operator~(void); // отрицание (#С)

    friend istream &operator>>(istream &istr, TBitField &bf); // (#07)
    friend ostream &operator<<(ostream &ostr, const TBitField &bf); // (#П4)
};
```

Поля:

BitLen – длина битового поля.

***pMem** – память для представления битового поля

MemLen – количество элементов Меморы для представления битового поля

Конструкторы:

➤ **TBitField(int Len);**

Назначение: Конструктор с параметром и выделение памяти

Входные параметры: Len – длина битового поля

Выходные параметры: Отсутствуют

➤ `TBitField(const TBitField &bf);`

Назначение: Конструктор копирования. Создаёт экземпляр класса на основе имеющегося экземпляра

Входные параметры: &bf – ссылка, адрес экземпляра класса, на основе которого будет создан другой.

Выходные параметры: Отсутствуют

Деструктор:

➤ `~TBitField();`

Назначение: Очистка выделенной памяти

Входные и выходные параметры: Отсутствуют

Методы:

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

Назначение: Получение индекса элемента, где хранится бит.

Входные данные: n – номер бита.

Выходные данные: Индекс элемента, где хранится бит с номером n.

➤ `TELEM GetMemMask(const n) const;`

Назначение: Получение битовой маски

Входные данные: n – номер бита.

Выходные данные: 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 значение бита 1

Входные параметры: n - номер бита, которому нужно установить значение 1

Выходные параметры: Отсутствуют

➤ `void ClrBit(const int n);`

Назначение: Установить значение бита 1

Входные параметры: n - номер бита, который нужно отчистить

Выходные параметры: Отсутствуют

➤ `int GetBit(const int n) const;`

Назначение: Вывести значение бита

Входные параметры: n - номер бита, значение которого нужно вывести

Выходные параметры: Значение бита

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, если равны то 1, 0 если нет.

➤ `int operator!=(const TBitField &bf) const;`

Назначение: Оператор сравнения. Сравнить на неравенство 2 битовых поля

Входные параметры: &bf – битовое поле, с которым мы сравниваем

Выходные параметры: 1 или 0, в зависимости не 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 и `&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 и `&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 хар-го 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 n); // включить элемент в множество
    void DelElem(const int n); // удалить элемент из множества
    int IsMember(const int n) const; // проверить наличие элемента в
    // множестве
    // теоретико-множественные операции
    int operator== (const TSet &s); // сравнение
    TSet& operator=(const TSet &s); // присваивание
    TSet operator+ (const int n); // включение элемента в множество
    TSet operator- (const int n); // удаление элемента из множества
```



```

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 и 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

Деструктор:

~TSet();

Назначение: Очистка выделенной памяти

Входные и выходные параметры: Отсутствуют

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 – элемент, который хотим добавить

Выходные параметры: Отсутствуют

- `void DelElem(const int Elem)`

Назначение: Удалить элемент из множества

Входные параметры: Elem - элемент, который хотим удалить

Выходные параметры: Отсутствуют

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

Назначение: Проверить наличие элемента в множестве

Входные параметры: Elem - элемент, который нужно проверить на наличие

Выходные параметры: 1 или 0, в зависимости от наличия элемента в множестве.

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

- `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: &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 универсального множества

Входные параметры: Отсутствуют

Выходные параметры: Экземпляр класса, каждый элемент которого равен {~*this}

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

Назначение: Оператор ввода из консоли

Входные параметры: &istr – буфер консоли и &s – класс, который нужно ввести из консоли

Выходные параметры: Ссылка на поток &istr

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

Назначение: Оператор вывода из консоли

Входные параметры: &istr – буфер консоли и &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 результате данной лабораторной работы были изучены теоретические основы битовых полей и множеств, а также принципы их использования в программировании. На основе полученных знаний была разработана программа, которая реализует операции над битовыми полями и множествами. Проведенный анализ результатов показал, что использование битовых полей и множеств может быть очень полезным в решении определенных задач. Они позволяют эффективно работать с большим количеством данных, а также выполнять операции объединения, пересечения и разности между наборами элементов.

Используемая Литература

[Битовые поля в C++ | Microsoft Learn](#)

[iditect.com/programming/cpp/cpp-bit-fields.html#:~:text=Let's delve into bit fields,Allocates 4 bits for flag3%7D](#)

[С | Битовые поля \(metanit.com\)](#)

[Bit-field - cppreference.com](#)

[C++ Bit Fields - GeeksforGeeks](#)

Приложение

Реализация класса Tset:

```
#include "tset.h"

TSet::TSet(int mp) :MaxPower(mp), BitField(mp) {}
TSet::TSet(const TSet& s) : BitField(s.BitField), MaxPower(s.GetMaxPower())
{}
TSet::TSet(const TBitField& bf) :MaxPower(bf.GetLength()), BitField(bf) {}
int TSet::GetMaxPower() const
{
    return MaxPower;
}
TSet::operator TBitField()
{
    return BitField;
}

int TSet::IsMember(const int Elem) const
{
    if (Elem >= MaxPower || Elem < 0)
        throw ("Error: Element is out of universe");
    return BitField.GetBit(Elem);
}

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

void TSet::DelElem(const int Elem)
{
    if (Elem >= MaxPower || Elem < 0)
        throw ("Error: Element is out of universe");
    return BitField.ClrBit(Elem);
}

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

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

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

```

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

TSet TSet::operator+(const int Elem)
{
    if (Elem >= MaxPower || Elem < 0)
        throw ("Error: Element is out of universe");
    TSet obj(*this);
    obj.InsElem(Elem);
    return obj;
}

TSet TSet::operator-(const int Elem)
{
    if (Elem >= MaxPower || Elem < 0)
        throw ("Error: Element is out of universe");
    TSet obj(*this);
    obj.DelElem(Elem);
    return obj;
}

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

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

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

ostream& operator<<(ostream& ostr, const TSet& s)
{
    const int Z = s.MaxPower - 1;
    for (int i = 0; i <= Z; ++i)
    {
        ostr << s.IsMember(i) << " ";
    }
    return ostr;
}

```

Реализация класса Tbitfield:

```
#include "tbitfield.h"
const int bitsInElem = 32;
const int shiftSize = 5; //Значение для сдвига

TBitField::TBitField(int Len) //Конструктор по умолчанию
{
    if (Len > 0) {
        BitLen = Len; // длина битового поля - макс. к-во битов
        MemLen = ((Len + bitsInElem - 1) >> shiftSize); // к-во эл-тов Мем
        для представления бит.поля
        pMem = new TELEM[MemLen]; //Массив для предоставления битового поля
        memset(pMem, 0, MemLen * sizeof(TELEM)); //Функция копирования одного
        объекта в другой заданное количество раз
    }
    else if (Len < 0)
    {
        throw ("Field less than 0");
    }
    else if (Len == 0) {
        MemLen = 0; // к-во эл-тов Мем для представления бит.поля
        BitLen = 0; // длина битового поля - макс. к-во битов
        pMem = nullptr; //Значение пустого указателя
    }
}

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()
{
    delete[] pMem;
}

int TBitField::GetMemIndex(const int n) const
{
    return n >> shiftSize;
}

TELEM TBitField::GetMemMask(const int n) const
{
    return 1 << (n & (bitsInElem - 1));
}

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

void TBitField::SetBit(const int n)
```



```

{
    if (n >= BitLen || n < 0)
        throw("The bit is not in the range");
    pMem[GetMemIndex(n)] |= GetMemMask(n);
}

void TBitField::ClrBit(const int n)
{
    if (n >= BitLen || n < 0)
        throw("The bit is not in the range");
    pMem[GetMemIndex(n)] &= ~GetMemMask(n);
}

int TBitField::GetBit(const int n) const {
    if (n >= BitLen || n < 0)
        throw("The bit is not in the range");
    return (pMem[GetMemIndex(n)] & GetMemMask(n)) ? 1 : 0;
}

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;
}

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

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;
}

TBitField TBitField::operator&(const TBitField& bf)
{
    int Len = max(BitLen, bf.BitLen);
    TBitField obj(Len);
    for (int i = 0; i < obj.MemLen; i++) {
        obj.pMem[i] = pMem[i] & bf.pMem[i];
    }
    return obj;
}

```

```

}

TBitField TBitField::operator|(const TBitField& bf)
{
    int Len = max(BitLen, bf.BitLen);
    TBitField obj(Len);
    for (int i = 0; i < obj.MemLen; i++)
        obj.pMem[i] = pMem[i] | bf.pMem[i];
    return obj;
}

TBitField TBitField::operator~()
{
    TBitField obj(BitLen);
    for (int i = 0; i < BitLen; i++)
        if (GetBit(i) == 0)
            obj.SetBit(i);
    return obj;
}

istream& operator>>(istream& istr, TBitField& bf) {
    int ans;
    for (int i = 0; i < bf.BitLen; i++) {
        istr >> ans;
        if (ans == 0) {
            bf.ClrBit(i);
        }
        else if (ans == 1) {
            bf.SetBit(i);
        }
        else {
            throw ("The Element doesn't match the bitfield");
            break;
        }
    }
    return istr;
}

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

```