МИНИСТЕРСТВО НАУКИ И ВЫСШЕГООБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механик**

ЛАБОРАТОРНАЯ РАБОТА

на тему:

**«БИТОВЫЕ ПОЛЯ И МНОЖЕСТВА»**

**Выполнил(а):** студент(ка) группы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Суворов Д.И. /

Подпись

**Проверил:** к.т.н., доцент каф. ВВиСП

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Кустикова В.Д. /

Подпись

Нижний Новгород  
2023

# Введение

Множество – это математическая модель, являющаяся совокупностью объектов, которые называются элементами множества, обладающих определёнными признаками, едиными для всех элементов одного множества. Теория множеств – это математическая дисциплина, которая изучает свойства, отношения и операции над множествами. Область применения теории множеств – огромна. Эта теория используется во многих разделах математики, например, в алгебре, топологии, геометрии, математической логике, теории вероятности и т.д. Она также широко применяется в компьютерных науках, физике и экономике. Так, в компьютерных науках теория множеств используется для описания структур данных, для моделирования и решения различных задач; в экономике - для моделирования социальных и экономических процессов. Таким образом множество – это универсальный математический инструмент для моделирования, описания, изучения различных процессов, а также для решения множества алгоритмических и математических задач.

Одним из самых эффективных способов представления множества в виде структуры данных несомненно является битовое поле. Эта структура данных представляет собой непрерывную последовательность битов, каждый из которых может быть либо 0, либо 1. Так как данные хранятся в битах – битовое поле позволяет очень компактно хранить и быстро производить манипуляцию над данными. Благодаря таким сильным сторонам, битовые поля применяются не только в отношении с множествами, но также применяются во множестве компьютерных областей, которые связаны с обработкой или хранением данных: базы данных, сетевые протоколы, криптография и т.п.

Всё описанное выше доказывает высокую значимость и активную применимость множеств и битовых полей.

# Постановка задачи

Цель – получить теоретические основы и практические навыки применения структуры данных битовое поле.

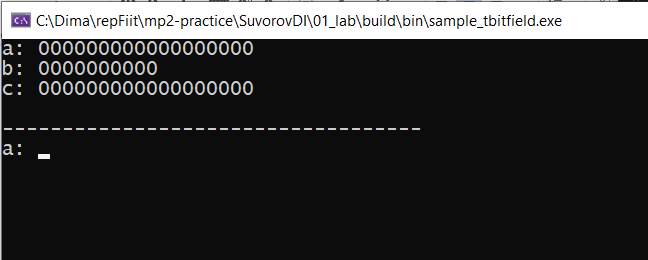
Задачи:

1. Изучить предметную область: узнать, что такое битовое поле;
2. Разработать класс, представляющий битовое поле;
3. Реализовать все необходимые операции с битовыми полями;
4. Разработать класс множеств;
5. Реализовать все необходимые операции с множествами;
6. Протестировать корректность работы разработанного функционала;
7. Применить разработанную программу для решения задачи о простых числах (решето Эратосфена);
8. Сделать вывод о проделанной работе.

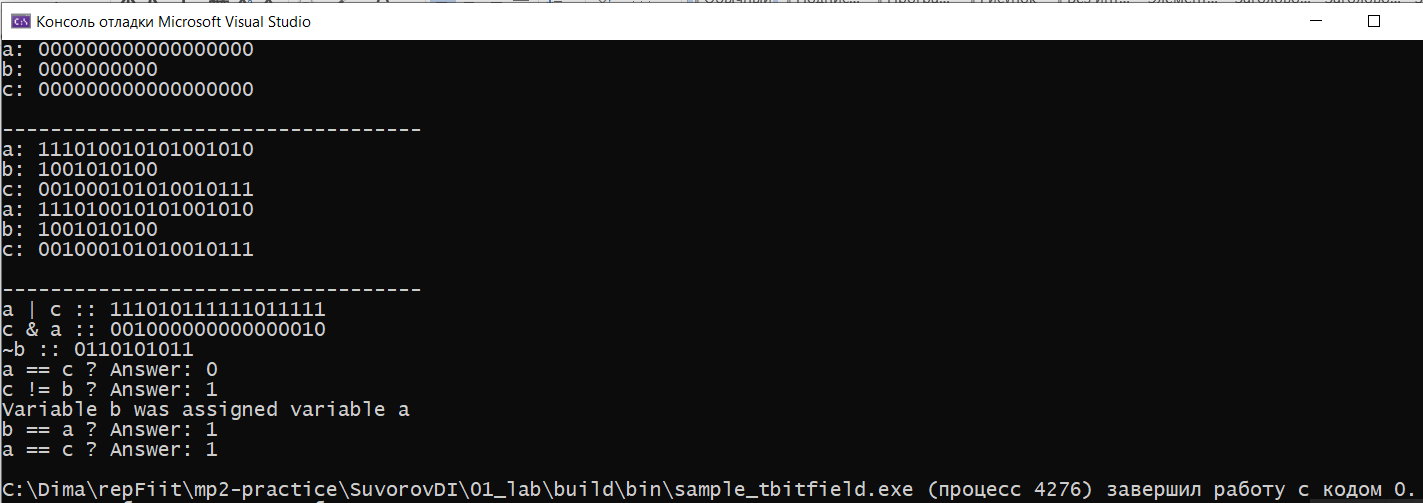
# Руководство пользователя

## Приложение для демонстрации работы битовых полей

1. Запустить sample\_tbitfield.exe. В результате появится следующее окно:



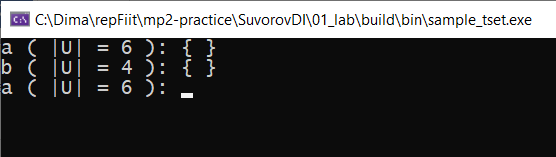
1. Стартовое окно приложения
2. В первых трёх строчках выведены заранее заготовленные обнуленные битовые поля с длинами 18, 10, 18. После разделительной полосы необходимо ввести значение каждого битового поля.
3. После ввода соответственных битовых полей будут выведены результаты применения основных операций между ними. Каждый вывод сопровождается соответствующими пометками, которые описывают, результат какой операции выведен.



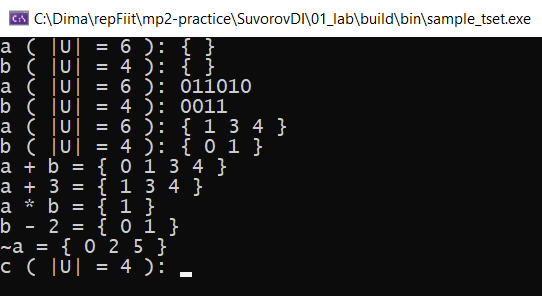
1. Итог работы программы

## Приложение для демонстрации работы множеств

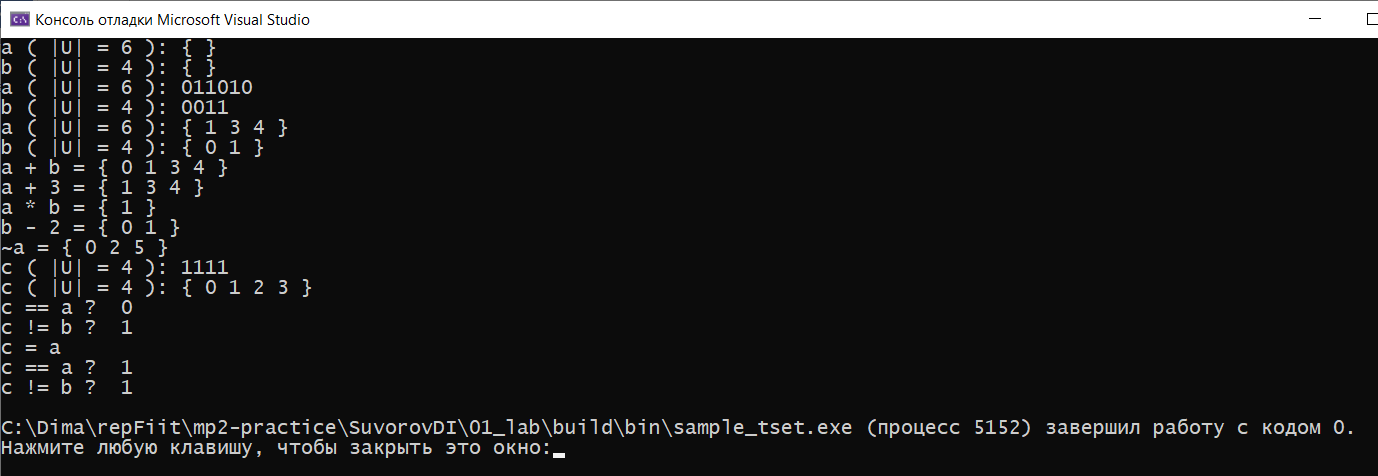
1. Запустить sample\_tset.exe. В результате появится следующее окно:



1. Стартовое окно приложения
2. Заранее в программе созданы два множества a и b с мощностью универсов соответственно 6 и 4. Необходимо задать множества в виде битовой строки. Где каждый бит это 0 или 1. Причём 0, стоящий на i-том месте в битовом поле (места считаются с 0 справа налево) – означает, что элемент i из универса не принадлежит множеству; 1, стоящая на i-том месте в битовом поле – означает, что элемент i из универса принадлежит множеству.
3. Задав множества a и b программа выведет пример использования доступных операций с этими множествами:



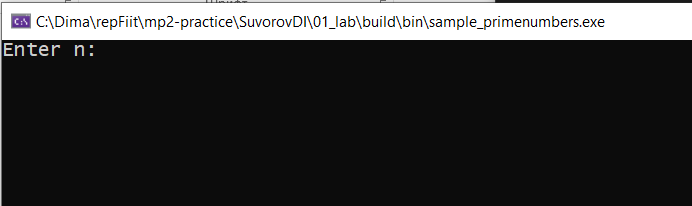
1. Доступные операции над множествами a и b
2. Необходимо задать множество c в универсе из 4 элементов аналогичным способом, каким и множества a и b. После ввода программа выведет результаты применения операций с этим множеством.



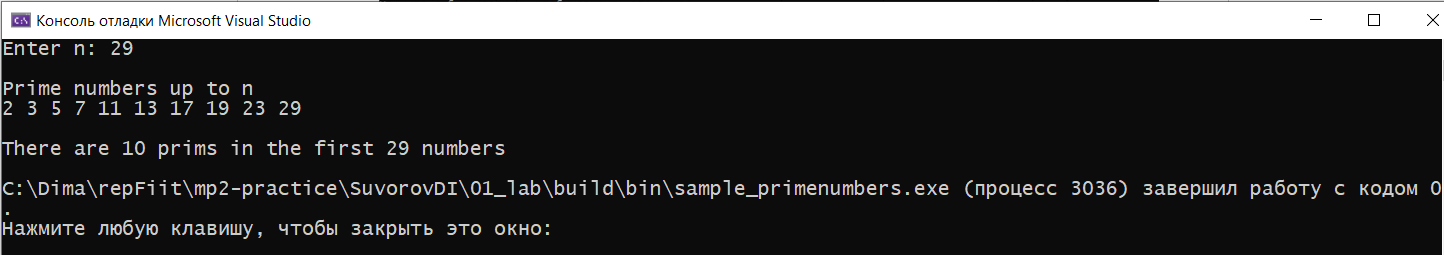
1. Итоговое окно приложения

## Приложение «решето Эратосфена»

1. Запустить sample\_primenumbers.exe. В результате появится следующее окно:



1. Стартовое окно приложения
2. Необходимо ввести натуральное число. После ввода натурального числа нажать Enter и программа выведет все простые числа от 2 до введённого числа включительно.



1. Итоговое окно приложения

# Руководство программиста

## Использованнные алгоритмы

### Битовые поля

Битовые поля представляют собой непрерывное хранение 0 и 1 (битов) в памяти. В языке программирования C++ нет типа данных, который может напрямую хранить значение 0 или 1 в одном бите. Для того, чтобы реализовать эту структуру данных мы будем использовать доступный тип данных unsigned int. Как любой без знаковый целочисленный тип данных он хранится в памяти в виде двоичного представления, то есть из непрерывной последовательности 0 и 1 длинной 32 бита (4 байта). Значит любое битовое поле длины n можно представить как массив из n // 32 + 1 unsigned int значений (где // – целочисленное деление).

Для битовых полей реализованы операции доступа к конкретному биту (установить бить в 1, в 0, получить текущее значение бита); побитовые операции (и, или, не); операции ввода и вывода битового поля; операции копирования и сравнения.

### Множества

Множество основано на битовом поле. Длина битового поля для множества – это максимальная длина для этого множества, то есть мощность универса. Если бит на i-том разряде битового поля равен 1, то элемент i из универса входит в множество, если равен 0, то соответствующий элемент не входит в множество.

Для множеств реализованы операции добавления и удаления элемента из множества; операции проверки наличия элемента в множестве; теоретико-множественные операции между множествами (объединение, пересечение множеств); взятие дополнения к множеству; операции ввода и вывода множества; операции копирования и сравнения.

### Алгоритм «решето Эратосфена»

Решето Эратосфена — алгоритм нахождения всех простых чисел до некоторого целого числа n включительно. Алгоритм выполнения:

1) У пользователя запрашивается целое положительное число N, до которого необходимо найти все простые числа;

2) Создается множество из положительных целых чисел от 2 до N включительно;

3) Далее берутся подряд элементы, которые принадлежат множеству. На каждом шаге берется следующий такой элемент и из множества удаляются все последующие элементы, которые кратны ему;

4) Эти шаги продолжаются до квадратного корня от N;

5) Таким образом в множестве остаются только простые числа.

## Описание классов

### Класс TbitField

Объявление класса:

class TBitField {

private:

elem\_t\* bField;

int fieldSize;

int memSize;

int bitsInElem = sizeof(elem\_t)\*8;

int shiftSize = round(log2(bitsInElem));

int GetMemIndex(const int n) const;

elem\_t 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;

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

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

TBitField& operator=(const TBitField& bf);

TBitField operator|(const TBitField& bf);

TBitField operator&(const TBitField& bf);

TBitField operator~(void);

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

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

};

Поля:

fieldSize – длина битового поля

bField – указатель типа elem\_t на область памяти, где хранится представление битового поля

memSize – длина выделенной для битового поля памяти

bitsInElem – количество битов в типе elem\_t

shiftSize – число на которое надо сдвинуть значение, чтобы взять остаток от деления на bitsInElem.

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.

elem\_t GetMemMask(const int n) const;

Назначение: получение битовой маски

Входные данные: n – номер бита.

Выходные данные: число типа elem\_t у которого все биты, кроме n-ого равны 0, а n-ый равен 1.

int GetLength(void) const;

Назначение: получение длины битового поля

Выходные данные: целое число – количество бит в битовом поле

void SetBit(const int n);

Назначение: установление бита под номером n в 1

Входные данные: n – номер бита.

void ClrBit(const int n);

Назначение: установление бита под номером n в 0

Входные данные: n – номер бита.

int GetBit(const int n) const;

Назначение: получение бита под номером n

Входные данные: n – номер бита.

Выходные данные: целое число – значение бита

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

Назначение: получение результата сравнения двух битовых полей

Входные данные: bf – ссылка на константный объект типа Битовое поле – битовое поле, с которым будет сравниваться текущее.

Выходные данные: логическое значение – true, если поля равны; false – в противном случае

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

Назначение: получение результата сравнения двух битовых полей

Входные данные: bf – ссылка на константный объект типа Битовое поле – битовое поле, с которым будет сравниваться текущее.

Выходные данные: логическое значение – false, если поля равны; true – в противном случае

TBitField& operator=(const TBitField& bf);

Назначение: копирование значения входного битового поля в текущий

Входные данные: bf – ссылка на константный объект типа Битовое поле – битовое поле, значение из которого будут копироваться.

Выходные данные: ссылка на изменённый текущий объект

TBitField operator|(const TBitField& bf);

Назначение: получение результата операции битового ИЛИ между текущим и переданным битовыми полями

Входные данные: bf – ссылка на константный объект типа Битовое поле – битовое поле, с которым у текущего будет производиться операция.

Выходные данные: результирующий объект, полученный в результате операции

TBitField operator&(const TBitField& bf);

Назначение: получение результата операции битового И между текущим и переданным битовыми полями

Входные данные: bf – ссылка на константный объект типа Битовое поле – битовое поле, с которым у текущего будет производиться операция.

Выходные данные: битовое поле, полученное в результате битовой операции

TBitField operator~(void);

Назначение: получение результата операции битового отрицания текущего битового поля

Выходные данные: битовое поле, полученное в результате битовой операции

### Класс TSet

class TSet

{

private:

int MaxPower; // максимальная мощность множества

TBitField BitField; // битовое поле для хранения характеристического вектора

public:

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

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

TSet(const TBitField& bf) : MaxPower(bf.GetLength()), BitField(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 std::istream& operator>>(std::istream& istr, TSet& s);

friend std::ostream& operator<<(std::ostream& ostr, const TSet& s);

};

Поля:

MaxPower – максимальная мощность множества (мощность универса)

BitField – битовое поле, описывающее множество

Методы:

int GetMaxPower(void) const;

Назначение: получение мощности универса для текущего множества

Выходные данные: целое число – мощность универса

void InsElem(const int Elem);

Назначение: добавление элемента в множество

Входные данные: целочисленная константа Elem – элемент, который будет добавлен

void DelElem(const int Elem);

Назначение: удаление элемента из множества

Входные данные: целочисленная константа Elem – элемент, который будет удален

int IsMember(const int Elem) const;

Назначение: проверка наличия переданного элемента в множестве

Входные данные: целочисленная константа Elem – элемент, наличие которого проверяется

Выходные данные: целое число – 1, если входит в множество; 0, если не входит

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

Назначение: получение результата сравнения двух множеств

Входные данные: s – ссылка на константный объект типа множество – множество, с которым будет сравниваться текущее.

Выходные данные: целое значение – 1, если поля равны; 0 – в противном случае

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

Назначение: получение результата сравнения двух множеств

Входные данные: s – ссылка на константный объект типа множество – множество, с которым будет сравниваться текущее.

Выходные данные: целое значение – 0, если поля равны; 1 – в противном случае

TSet& operator=(const TSet& s);

Назначение: копирование значения входного множества в текущее

Входные данные: s – ссылка на константный объект типа множество – множество, значение из которого будут копироваться.

Выходные данные: ссылка на изменённый текущий объект

TSet operator+ (const int Elem);

Назначение: добавление элемента в множество

Входные данные: элемент, который будет добавлен в множество

Выходные данные: результирующее множество

TSet operator- (const int Elem);

Назначение: удаление элемента из множества

Входные данные: элемент, который будет удален из множества

Выходные данные: результирующее множество

TSet operator+ (const TSet& s);

Назначение: объединение двух множеств

Входные данные: множество, с которым будет объединяться текущее

Выходные данные: результирующее множество

TSet operator\* (const TSet& s);

Назначение: пересечение двух множеств

Входные данные: множество, с которым будет пересечено текущее

Выходные данные: результирующее множество

TSet operator~ (void);

Назначение: взятие дополнения к множеству

Выходные данные: результирующее множество

# Заключение

В результате данной лабораторной работы были изучены теоретические основы и практические навыки применения битовых полей и множеств в программировании. На основе полученных знаний в этой предметной области была разработана программа, которая реализует хранение битовых полей и множеств, а также операции над ними. Была произведена проверка корректной работы реализованного функционала с различными наборами данных. В целом, лабораторная работа помогла понять основные принципы работы с битовыми полями и множествами, их преимущества и ограничения.

# Литературы

1. [Битовое поле (C++) — Википедия (wikipedia.org)](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82%D0%BE%D0%B2%D0%BE%D0%B5_%D0%BF%D0%BE%D0%BB%D0%B5_(C%2B%2B)) [https://en.wikipedia.org/wiki/Bit\_field]
2. Битовые поля в C++ [ <https://learn.microsoft.com/ru-ru/cpp/cpp/cpp-bit-fields?view=msvc-170> ]

# Приложения

## Приложение А. Реализация класса TBitField

TBitField::TBitField(int len) {

if (len < 0)

throw std::exception("Negative size does not support");

fieldSize = len;

memSize = len / bitsInElem + 1;

bField = new elem\_t[memSize];

for (int i = 0; i < fieldSize; i++) {

ClrBit(i);

}

}

TBitField::TBitField(const TBitField& bf) {

fieldSize = bf.fieldSize;

memSize = bf.memSize;

bField = new elem\_t[memSize];

for (int i = 0; i < memSize; i++)

bField[i] = bf.bField[i];

}

TBitField::~TBitField() {

delete[] bField;

}

int TBitField::GetMemIndex(const int pos) const {

return pos >> shiftSize;

}

elem\_t TBitField::GetMemMask(const int pos) const {

return 1 << (pos & (bitsInElem - 1));

}

int TBitField::GetLength(void) const {

return fieldSize;

}

void TBitField::SetBit(const int n) {

if (n >= fieldSize || n < 0)

throw std::exception("out of range");

bField[GetMemIndex(n)] |= GetMemMask(n);

}

void TBitField::ClrBit(const int n) {

if (n >= fieldSize || n < 0)

throw std::exception("out of range");

bField[GetMemIndex(n)] &= ~(GetMemMask(n));

}

int TBitField::GetBit(const int n) const {

if (n >= fieldSize || n < 0)

throw std::exception("out of range");

return (bField[GetMemIndex(n)] & GetMemMask(n)) >> (n & (bitsInElem - 1));

}

bool TBitField::operator==(const TBitField& bf) const {

if (fieldSize != bf.fieldSize)

return false;

for (int i = 0; i < memSize - 1; i++)

if (bField[i] != bf.bField[i])

return false;

for (int i = (memSize - 1) \* bitsInElem; i < fieldSize; i++)

if (GetBit(i) != bf.GetBit(i))

return false;

return true;

}

bool TBitField::operator!=(const TBitField& bf) const {

return !(\*this == bf);

}

TBitField& TBitField::operator=(const TBitField& bf) {

if (this == &bf)

return \*this;

fieldSize = bf.fieldSize;

if (memSize != bf.memSize) {

memSize = bf.memSize;

delete[] bField;

bField = new elem\_t[memSize];

}

for (int i = 0; i < memSize; i++) {

bField[i] = bf.bField[i];

}

return \*this;

}

TBitField TBitField::operator|(const TBitField& bf) {

const TBitField\* max\_field = this;

int min\_size = bf.fieldSize;

if (fieldSize < min\_size) {

max\_field = &bf;

min\_size = fieldSize;

}

TBitField res(\*max\_field);

for (int i = 0; i < min\_size; i++)

if ((GetBit(i) || bf.GetBit(i)) == 1)

res.SetBit(i);

return res;

}

TBitField TBitField::operator&(const TBitField& bf) {

int max\_size = fieldSize;

int min\_size = bf.fieldSize;

if (bf.fieldSize > max\_size) {

max\_size = bf.fieldSize;

min\_size = fieldSize;

}

TBitField res(max\_size);

for (int i = 0; i < min\_size; i++)

if ((GetBit(i) && bf.GetBit(i)) == 1)

res.SetBit(i);

return res;

}

TBitField TBitField::operator~(void) {

TBitField res(fieldSize);

for (int i = 0; i < memSize; i++) {

res.bField[i] = ~bField[i];

}

return res;

}

## Приложение Б. Реализация класса TSet

TSet::operator TBitField() {

TBitField tmp(BitField);

return tmp;

}

int TSet::GetMaxPower(void) const {

return MaxPower;

}

void TSet::InsElem(const int Elem) {

if (Elem >= MaxPower)

throw std::exception("Elem is out of univers");

BitField.SetBit(Elem);

}

void TSet::DelElem(const int Elem) {

if (Elem >= MaxPower)

throw std::exception("Elem is out of univers");

BitField.ClrBit(Elem);

}

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

return BitField.GetBit(Elem);

}

int TSet::operator==(const TSet& s) const {

return BitField == s.BitField;

}

int TSet::operator!=(const TSet& s) const {

return !(\*this == s);

}

TSet& TSet::operator=(const TSet& s) {

MaxPower = s.MaxPower;

BitField = s.BitField;

return \*this;

}

TSet TSet::operator+(const int Elem) {

TSet res(\*this);

res.InsElem(Elem);

return res;

}

TSet TSet::operator- (const int Elem) {

TSet res(\*this);

res.DelElem(Elem);

return res;

}

TSet TSet::operator+(const TSet& s) {

return TSet(BitField | s.BitField);

}

TSet TSet::operator\*(const TSet& s) {

return TSet(BitField & s.BitField);

}

TSet TSet::operator~(void) {

return TSet(~BitField);

}