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 механики**

**ЛАБОРАТОРНАЯ РАБОТА**

на тему:

**«Битовые поля и множества»**

**Выполнил(а):** студент(ка) группы \_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Рысев М.Д./

Подпись

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

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

Подпись

Нижний Новгород  
2023

**Содержание**

[Введение 3](#_Toc147915966)

[1 Постановка задачи 4](#_Toc147915967)

[2 Руководство пользователя 5](#_Toc147915968)

[2.1 Приложение для демонстрации работы битовых полей 5](#_Toc147915969)

[2.2 Приложение для демонстрации работы множеств 7](#_Toc147915970)

[2.3 «Решето Эратосфено» 8](#_Toc147915971)

[3 Руководство программиста 9](#_Toc147915972)

[3.1 Описание алгоритмов 9](#_Toc147915973)

[3.1.1 Битовые поля 9](#_Toc147915974)

[3.1.2 Множества 10](#_Toc147915975)

[3.1.3 «Решето Эратосфена» 11](#_Toc147915976)

[3.2 Описание программной реализации 13](#_Toc147915977)

[3.2.1 Описание класса TBitField 13](#_Toc147915978)

[3.2.2 Описание класса TSet 18](#_Toc147915979)

[Заключение 24](#_Toc147915980)

[Литература 25](#_Toc147915981)

[Приложения 26](#_Toc147915982)

[Приложение А. Реализация класса TBitField 26](#_Toc147915983)

[Приложение Б. Реализация класса TSet 28](#_Toc147915984)

# Введение

В программировании и анализе данных часто приходиться работать с большими объёмами информации, из чего вытекает необходимость поиска более менее оптимального варианта представления этих данных и манипуляции с ними. Битовые поля и множества как раз являются одним из этих вариантов. Они позволяют хранить набор элементов в виде вектора из нулей из единиц, где ноль означает отсутствие элемента в наборе, а единица, наоборот, его наличие. Таким образом, мы получаем вариант хранения данных, который позволяет нам сэкономить объём занимаемой памяти, а также ускорить операции над этими данными.

# Постановка задачи

Цель – Реализация класса “Битовое поле” и класса “Множество”. Практическое применение битовых полей и множеств.

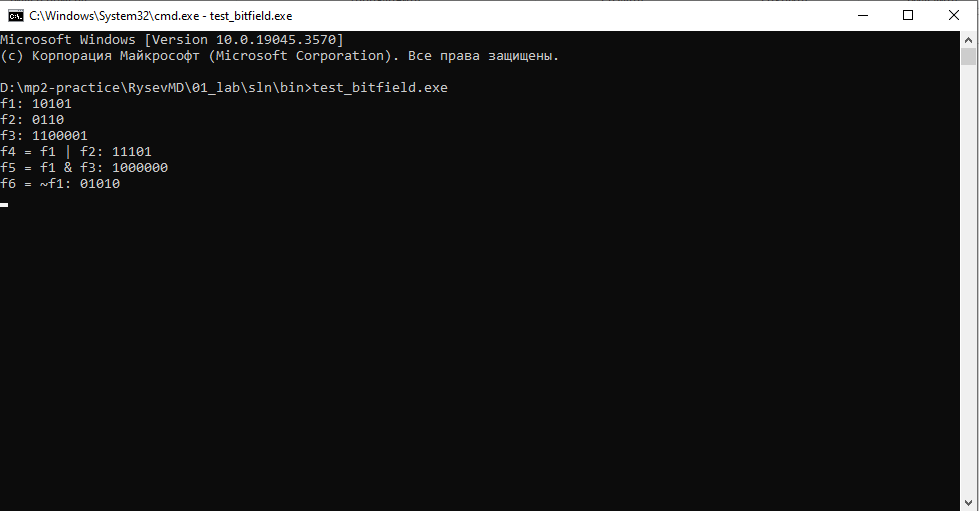
Задачи:

1. Ознакомиться с теорий о множествах и битовых полях.
2. Реализовать классы, предназначенные для представления битовых полей и множеств и операций над ними.
3. Протестировать работу полученных классов.
4. Написать программы, предназначенные для демонстрации работы полей и битовых множеств.

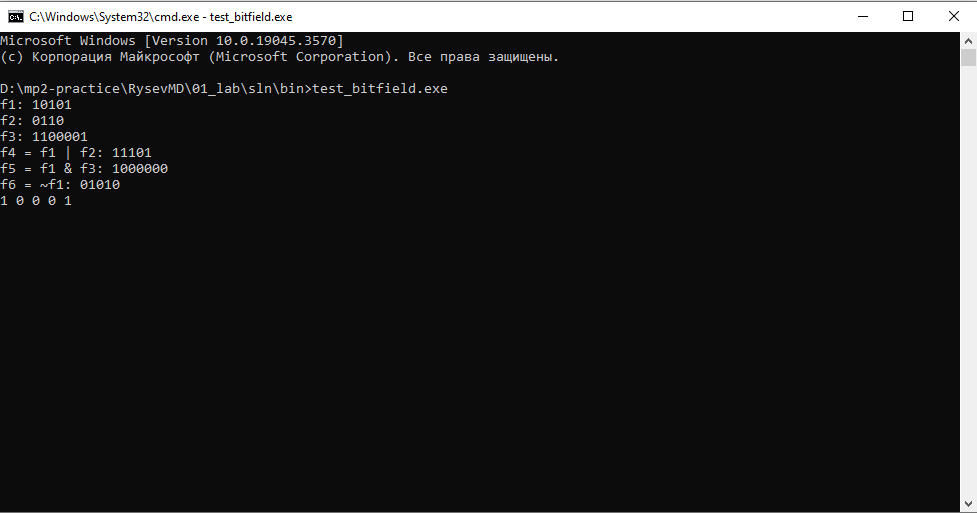
# Руководство пользователя

## Приложение для демонстрации работы битовых полей

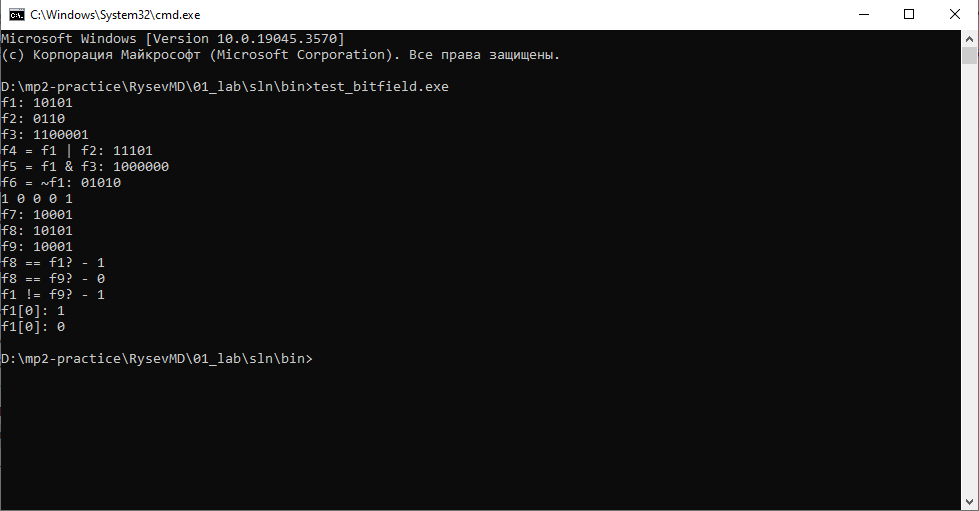
1. Запустите приложение с названием test\_bitfield.exe. В результате появится окно, показанное ниже ().



1. Основное окно программы test\_bitfield
2. В окне уже будут выведены шесть битовых полей и демонстрация операций объединения, пересечения и удаления. Затем необходимо ввести битовое поле длинны пять (рис .2).



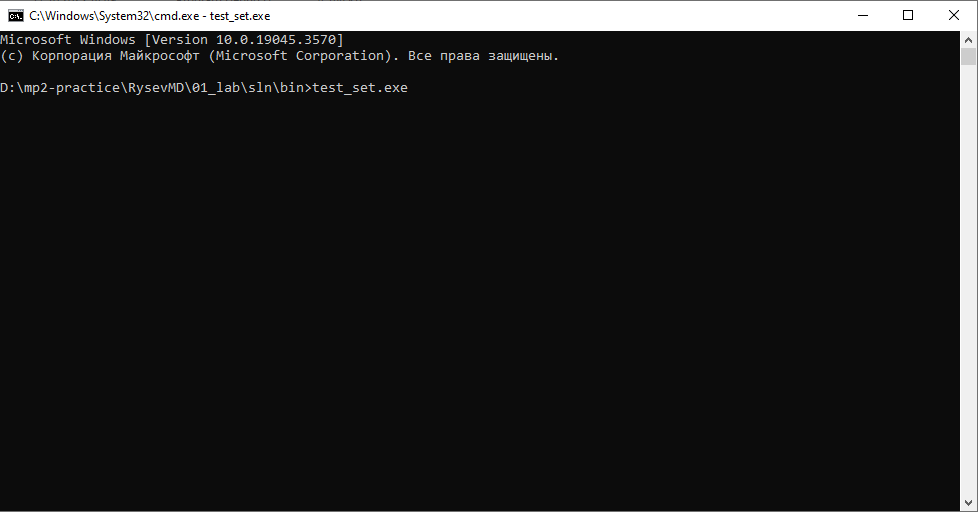
1. Ввод нового битового поля
2. После ввода битового поля будут выведены результаты сравнения битовых полей на равенство и неравенство, а также операции взятия и очистки бита (рис. 3)



1. Результат работы программы test\_bitfield

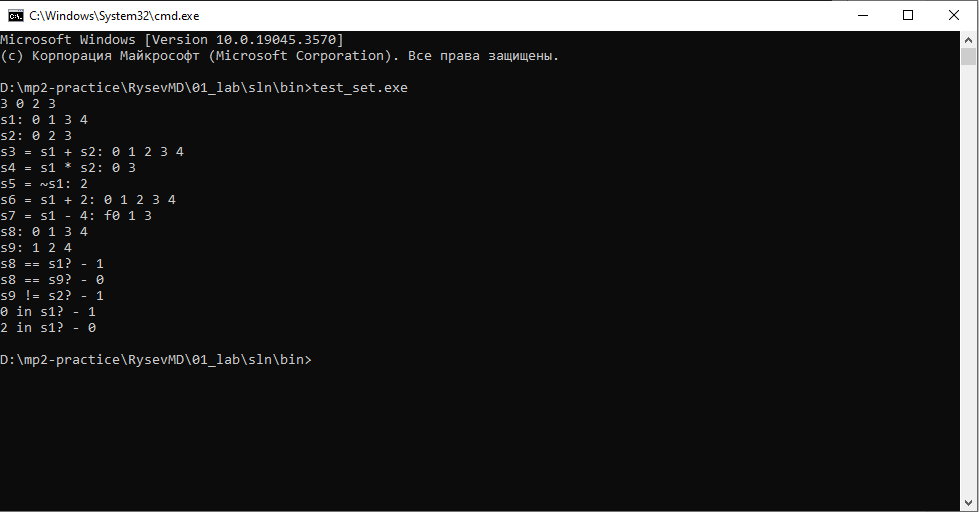
## Приложение для демонстрации работы множеств

1. Запустите приложение с названием test\_set.exe. В результате появится окно, показанное ниже (4).



1. Основное окно программы test\_set

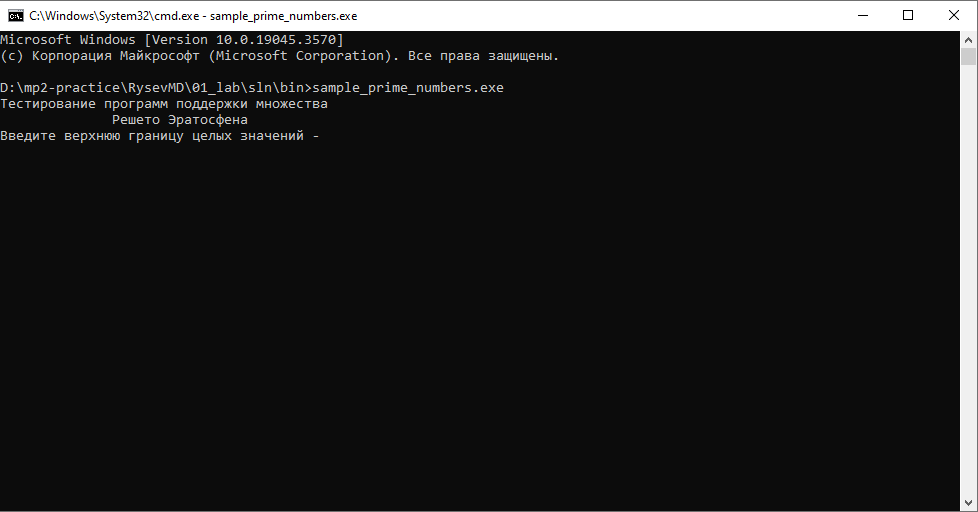
2. Программа будет ожидать ввода четырёх чисел, значения которых не превышают тройки. После ввода чисел будет выведена демонстрация включения и исключения элемента из множества, теоретико-множественных операций, операций сравнения и проверка наличия элемента в множестве (рис. 5).



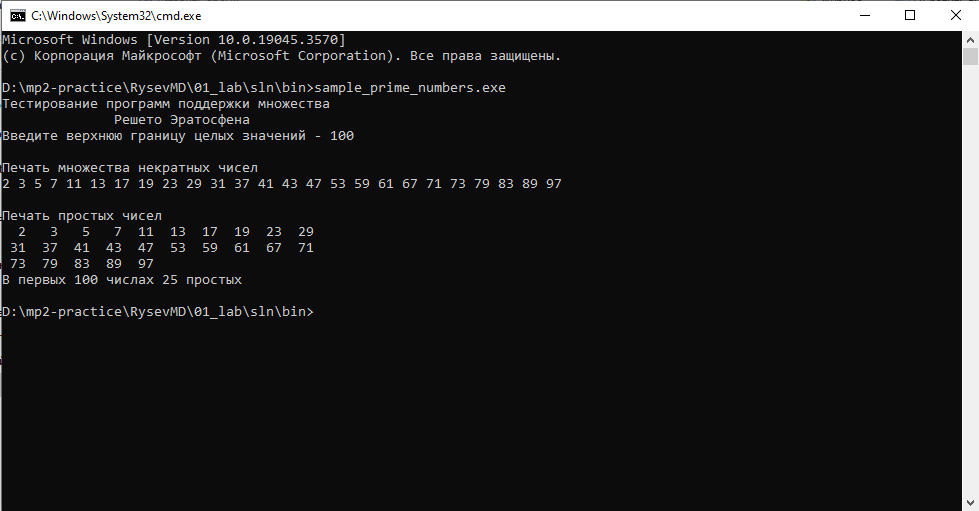
1. Результат работы программы test\_set

## «Решето Эратосфена»

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



1. Основное окно программы sample\_prime\_numbers
2. Введите число, до которого хотите осуществить поиск простых чисел (введённое число должно принадлежать множеству натуральных чисел). Затем в окне будет выведен результат работы программы (рис. 7).



1. Результат работы алгоритма “Решето Эратосфена”

# Руководство программиста

## Описание алгоритмов

### Битовые поля

Битовые поля - удобный способ хранить наборы данных, представленных в форме множеств целых неотрицательных чисел. Каждый бит поля хранит в себе состояние некоторого элемента, а именно - 1 если элемент является частью множества, 0 в противном случае.

Пример битового поля длины 7:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 |

Битовые поля поддерживают следующие операции: объединение, пересечение двух полей, дополнение поля, сравнение.

**Объединение**. Операция возвращает результирующее битовое поле, каждый бит которого равен 1, если хотя бы в одном из исходных полей бит с таким же индексом равен 1, и 0, если бит со схожим индексом равен нулю в обоих исходных полях.

Пример:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Bf1 | 0 | 0 | 1 | 1 | 0 |
| Bf2 | 0 | 1 | 0 | 1 | 0 |
| Bf1 | Bf2 | 0 | 1 | 1 | 1 | 0 |

**Пересечение**. Операция возвращает результирующее битовое поле, каждый бит которого равен 1, если в обоих из исходных полей биты с таким же индексом равны 1, и 0, если хотя бы в одном их полей бит со схожим индексом равен нулю.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Bf1 | 0 | 0 | 1 | 1 | 0 |
| Bf2 | 0 | 1 | 0 | 1 | 0 |
| Bf1 & Bf2 | 0 | 0 | 0 | 1 | 0 |

**Дополнение**. Операция возвращает результирующие битовое поле, каждый бит которого равен 1, если в исходном поле бит с таким же индексом равен 0, и наоборот, если в исходном поле некоторые бит равен 1, то в результирующем поле этот же бит будет равен 0.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Bf | 1 | 1 | 0 | 1 | 0 |
| ~Bf | 0 | 0 | 1 | 0 | 1 |

**Сравнение**. Операция возвращает 1 если поля побитово равны, 0 в противном случае.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Bf1 | 1 | 1 | 0 | 1 | 1 |
| Bf2 | 1 | 1 | 0 | 1 | 1 |
| Bf1 == Bf2 | 1 |

Реализация битового поля указана в приложении А данного файла.

### Множества

В данной лабораторной работе множества представляют собой набор из неотрицательных целых чисел. Реализуется наше множество на базе битового поля. Это значит, что задавая множество, мы ставим ему в соответствие какое-то битовое поле, каждый бит которого хранит информацию о наличии элемента в множестве.

Пример множества: A = {1, 3, 5, 7, 9} при |U| = 10 (универсальное множество состоит из десяти элементов 0, 1, ... , 9). Битовое поле соответствующее множеству A:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер бита | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Значение | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

Множества поддерживают следующие операции: объединение, пересечение множеств, объединение с элементом, разность с элементом, дополнение множества, сравнение множеств.

**Объединение**. Операция возвращает результирующее множество, содержащее как все уникальные элементы из первого исходного множества, так и все уникальные элементы из второго исходного множества.

Пример:

A = {1, 3, 5}

B = {0, 2, 3, 5}

A | B = {0, 1, 2, 3, 5}

**Пересечение**. Операция возвращает результирующее множество, содержащее только те элементы, которые есть одновременно в обоих исходных множествах.

Пример:

A = {1, 3, 5}

B = {0, 2, 3, 5}

A \* B = {3, 5}

**Объединение с элементом**. Операция возвращает множество, в которое добавлен элемент, участвующий в операции. Обязательным условием является наличие элемента в универсальном множестве.

Пример:

A = {1, 3, 5}, |U| = 10

a = 7

A + a = {1, 3, 5, 7}

**Разность с элементом**. Операция возвращает множество, из которого удалён элемент, участвующий в операции. Обязательным условием является наличие элемента в универсальном множестве.

Пример:

A = {1, 3, 5}, |U| = 10

a = 5

A - a = {1, 3}

**Дополнение множества**. Операция возвращает множество, в котором содержаться элементы принадлежащие универсальному множеству, но не принадлежащие исходному множеству.

Пример:

A = {1, 3, 5}, |U| = 7

~A = {0, 2, 4, 6}

**Сравнение множеств**. Операция возвращает 1, если множества поэлементно равны, и 0 в противном случае.

A = {1, 3, 5}

B = {0, 2, 3, 5}

A == B? - 0

Реализация множества описана в приложении Б данного файла.

### «Решето Эратосфена»

Данный алгоритм реализован двумя способами: с использованием битовых полей и с использованием множеств. Что бы переключиться с одного способа на другой достаточно закомментировать (или раскомментировать объявление константы ) USE\_SET.

Алгоритм работает по следующему принципу:

1. Запрашивается число N, до которого будет осуществляться поиск простых чисел.
2. Формируется битовое поле (или множество) от 2 до N, и все элементы битового поля (множества) помечаются как имеющиеся.
3. Запускается перебор чисел от 2 до корня квадратного из N:
   1. Если текущее число x есть в битовом поле (множестве), то из битового поля (множества) удаляются все кратные числу x элементы.
   2. Если текущее число x не содержится в битовом поле (множестве) то осуществляется переход к следующему элементу

После окончания работы алгоритма происходит вывод результата в консоль.

## Описание программной реализации

### 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();

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 маски бита.

Входные параметры:

n – номер бита.

Выходные параметры:

Маска бита.

TBitField(int len);

Назначение:

Конструктор. Создание битового поля.

Входные параметры:

len – число элементов.

Выходные параметры:

Отсутствуют.

TBitField(const TBitField &bf);

Назначение:

Конструктор. Копирования битового поля.

Входные параметры:

&bf – ссылка на битовое поле.

Выходные параметры:

Отсутствуют.

~TBitField();

Назначение:

Деструктор. Очистка выделенной памяти.

Входные параметры:

Отсутствуют.

Выходные параметры:

Отсутствуют.

int GetLength(void) const;

Назначение:

Получение размера битового поля.

Входные параметры:

Отсутствуют.

Выходные параметры:

Длина битового поля.

void SetBit(const int n);

Назначение:

Установка значения бита в единицу.

Входные параметры:

n – номер бита.

Выходные параметры:

Отсутствуют.

void ClrBit(const int n);

Назначение:

Установка значения бита в ноль.

Входные параметры:

n – номер бита.

Выходные параметры:

Отсутствуют.

int GetBit(const int n) const;

Назначение:

Получение значения бита.

Входные параметры:

n – номер бита.

Выходные параметры:

Значение бита.

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

Назначение:

Оператор сравнения на равенство.

Входные параметры:

&bf – ссылка на битовое поле.

Выходные параметры:

Результат сравнения – 0 или 1.

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

Назначение:

Оператор сравнения на неравенство.

Входные параметры:

&bf – ссылка на битовое поле.

Выходные параметры:

Результат сравнения – 0 или 1.

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

Назначение:

Оператор присваивания.

Входные параметры:

&bf – ссылка на битовое поле.

Выходные параметры:

Ссылка на объект TbitField.

TBitField operator|(const TBitField &bf);

Назначение:

Побитовое “ИЛИ”.

Входные параметры:

&bf – ссылка на битовое поле.

Выходные параметры:

Объект класса TbitField.

TBitField operator&(const TBitField &bf);

Назначение:

Побитовое “И”

Входные параметры:

&bf – ссылка на битовое поле.

Выходные параметры:

Объект класса TbitField.

TBitField operator~(void);

Назначение:

Побитовое отрицание

Входные параметры:

Отсутствуют.

Выходные параметры:

Объект класса TbitField.

friend istream &operator>>(istream &istr, TBitField &bf);

Назначение:

Чтение битового поля из консоли.

Входные параметры:

&istr – ссылка на поток ввода.

&bf – ссылка на битовое поле.

Выходные параметры:

Ссылка на поток ввода.

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

Назначение:

Вывод битового поля в консоль.

Входные параметры:

&ostr – ссылка на поток вывода.

&bf – ссылка на битовое поле.

Выходные параметры:

Ссылка на поток вывода.

### 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.

Поля:

MaxPower – количество элементов в множестве – мощность унивёрса.

BitField – битовое поле.

Методы:

TSet(int mp);

Назначение:

Конструктор. Создание множества.

Входные параметры:

mp – количество элементов.

Выходные параметры:

Отсутствуют.

TSet(const TSet &s);

Назначение:

Конструктор. Копирование множества.

Входные параметры:

&s – ссылка на множество.

Выходные параметры:

Отсутствуют.

TSet(const TBitField &bf);

Назначение:

Конструктор. Преобразование битового поля к множеству.

Входные параметры:

&s – ссылка на множество.

Выходные параметры:

Отсутствуют.

operator TBitField();

Назначение:

Преобразование множества к битовому полю.

Входные параметры:

Отсутствуют.

Выходные параметры:

Объект класса TbitField.

int GetMaxPower(void) const;

Назначение:

Получение размера унивёрса, которому принадлежит множество.

Входные параметры:

Отсутствуют.

Выходные параметры:

Размер унивёрса.

void InsElem(const int Elem);

Назначение:

Вставка элемента.

Входные параметры:

Elem – вставляемый элемент.

Выходные параметры:

Отсутствуют.

void DelElem(const int Elem);

Назначение:

Удаление элемента.

Входные параметры:

Elem – удаляемый элемент.

Выходные параметры:

Отсутствуют.

int IsMember(const int Elem) const;

Назначение:

Проверка наличия элемента в множестве.

Входные параметры:

Elem – искомый элемент.

Выходные параметры:

Результат поиска - 0 или 1.

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

Назначение:

Сравнение на равенство.

Входные параметры:

&s – ссылка на множество.

Выходные параметры:

Результат сравнения – 0 или 1.

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

Назначение:

Сравнение на неравенство.

Входные параметры:

&s – ссылка на множество.

Выходные параметры:

Результат сравнения – 0 или 1.

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

Назначение:

Оператор присваивания.

Входные параметры:

&s – ссылка на множество.

Выходные параметры:

Ссылка на объект класса TSet.

TSet operator+ (const int Elem);

Назначение:

Объединение с элементом.

Входные параметры:

Elem – элемент.

Выходные параметры:

Объект класса TSet.

TSet operator- (const int Elem);

Назначение:

Разность с элементом.

Входные параметры:

Elem – элемент.

Выходные параметры:

Объект класса TSet.

TSet operator+ (const TSet &s);

Назначение:

Объединение множеств.

Входные параметры:

&s – ссылка на множество.

Выходные параметры:

Объект класса TSet.

TSet operator\* (const TSet &s);

Назначение:

Пересечение множеств.

Входные параметры:

&s – ссылка на множество.

Выходные параметры:

Объект класса TSet.

TSet operator~ (void);

Назначение:

Дополнение множества.

Входные параметры:

Отсутствует.

Выходные параметры:

Объект класса TSet.

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

Назначение:

Чтение множества из консоли.

Входные параметры:

&istr – ссылка на поток ввода.

&s – ссылка на множество.

Выходные параметры:

Ссылка на поток ввода.

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

Назначение:

Вывод множества в консоль.

Входные параметры:

&ostr – ссылка на поток вывода.

&s – ссылка на множество.

Выходные параметры:

Ссылка на поток вывода.

# Заключение

1. Была изучена теория битовых полях и множествах.
2. Реализованы классы, предназначенные для представления битовых полей (TBitFieald) и множеств (TSet).
3. Проведены тесты на корректность работы работы классов TSet и TBitFieald.
4. Написаны программы, предназначенные для демонстрации работы полей и битовых множеств.

# Литература

1. Майкрософт [<https://learn.microsoft.com/ru-ru/cpp/cpp/cpp-bit-fields?view=msvc-170>].

# Приложения

## Приложение А. Реализация класса TbitField

#include "tbitfield.h"

#include <iostream>

#include <cmath>

TBitField::TBitField(int len)

{

if (len < 0) throw "length\_cant\_be\_less\_then\_zero";

BitLen = len;

MemLen = ((BitLen - 1) >> 5) + 1;

pMem = new TELEM[MemLen];

memset(pMem, 0, MemLen \* sizeof(TELEM));

}

TBitField::TBitField(const TBitField& bf)

{

BitLen = bf.BitLen;

MemLen = bf.MemLen;

pMem = new TELEM[MemLen];

memcpy(pMem, bf.pMem, MemLen \* sizeof(TELEM));

}

TBitField::~TBitField()

{

delete[] pMem;

}

int TBitField::GetMemIndex(const int n) const

{

if (n >= BitLen || n < 0) throw "out\_of\_range";

int s = ceil(log2(sizeof(int) \* 8 - 1));

return n >> s;

}

TELEM TBitField::GetMemMask(const int n) const

{

if (n >= BitLen || n < 0) throw "out\_of\_range";

return 1 << (BitLen - (n & sizeof(TELEM) \* 8 - 1) - 1);

}

int TBitField::GetLength(void) const

{

return BitLen;

}

void TBitField::SetBit(const int n)

{

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

throw "out\_of\_range";

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

}

void TBitField::ClrBit(const int n)

{

if (n >= BitLen || n < 0) throw "out\_of\_range";

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

}

int TBitField::GetBit(const int n) const

{

if (n >= BitLen || n < 0) throw "out\_of\_range";

if ((pMem[GetMemIndex(n)] & GetMemMask(n)) == 0) return 0;

return 1;

}

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

{

if (bf == (\*this)) return (\*this);

if (BitLen != bf.BitLen)

{

pMem = new TELEM[bf.MemLen];

}

BitLen = bf.BitLen;

MemLen = bf.MemLen;

memcpy(pMem, bf.pMem, MemLen \* sizeof(TELEM));

return (\*this);

}

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

{

if (bf.BitLen != BitLen) return 0;

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

if (bf.GetBit(i) != GetBit(i)) return 0;

}

return 1;

}

int TBitField::operator!=(const TBitField& bf) const

{

return !(\*this == bf);

}

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

{

int maxl = max(BitLen, bf.BitLen), minl = min(BitLen, bf.BitLen), i;

TBitField tmp(maxl);

for (i = 0; i < minl; i++) {

if (GetBit(i) | bf.GetBit(i)) tmp.SetBit(i);

}

while (i < BitLen) {

if (GetBit(i)) tmp.SetBit(i);

i += 1;

}

while (i < bf.BitLen) {

if (bf.GetBit(i)) tmp.SetBit(i);

i += 1;

}

return tmp;

}

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

{

int maxl = max(BitLen, bf.BitLen), minl = min(BitLen, bf.BitLen), i;

TBitField tmp(maxl);

for (i = 0; i < minl; i++) {

if (GetBit(i) & bf.GetBit(i)) tmp.SetBit(i);

}

return tmp;

}

TBitField TBitField::operator~(void)

{

TBitField tmp(BitLen);

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

if (!GetBit(i)) tmp.SetBit(i);

}

return tmp;

}

istream& operator>>(istream& istr, TBitField& bf)

{

for (int i = 0; i < bf.BitLen; i++) {

int num;

cin >> num;

if (num) bf.SetBit(i);

}

return istr;

}

ostream& operator<<(ostream& ostr, const TBitField& bf)

{

for (int i = 0; i < bf.BitLen; i++) cout << bf.GetBit(i);

return ostr;

}

## Приложение Б. Реализация класса TSet

#include "tset.h"

TSet::TSet(int mp) : BitField(mp)

{

MaxPower = mp;

}

TSet::TSet(const TSet& s) : BitField(s.BitField)

{

MaxPower = s.MaxPower;

}

TSet::TSet(const TBitField& bf) : BitField(bf)

{

MaxPower = bf.GetLength();

}

TSet::operator TBitField()

{

return BitField;

}

int TSet::GetMaxPower(void) const

{

return MaxPower;

}

int TSet::IsMember(const int Elem) const

{

if (Elem >= MaxPower) throw "Element not in universe";

if (BitField.GetBit(Elem)) return 1;

return 0;

}

void TSet::InsElem(const int Elem)

{

if (Elem >= MaxPower) throw "Element not in universe";

BitField.SetBit(Elem);

}

void TSet::DelElem(const int Elem)

{

if (Elem >= MaxPower) throw "Element not in universe";

BitField.ClrBit(Elem);

}

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

{

if (s == (\*this)) return (\*this);

MaxPower = s.MaxPower;

BitField = s.BitField;

return (\*this);

}

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

{

if (MaxPower != s.MaxPower) return 0;

return (BitField == s.BitField);

}

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

{

return !((\*this) == s);

}

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

{

TSet tmp(max(MaxPower, s.GetMaxPower()));

tmp.BitField = BitField | s.BitField;

return tmp;

}

TSet TSet::operator+(const int Elem)

{

if (Elem >= MaxPower) throw "Element not in universe";

TSet tmp(MaxPower);

tmp.BitField = BitField;

tmp.BitField.SetBit(Elem);

return tmp;

}

TSet TSet::operator-(const int Elem)

{

if (Elem >= MaxPower) throw "Element not in universe";

TSet tmp(MaxPower);

tmp.BitField = BitField;

tmp.BitField.ClrBit(Elem);

return tmp;

}

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

{

TSet tmp(max(MaxPower, s.GetMaxPower()));

tmp.BitField = (BitField & s.BitField);

return tmp;

}

TSet TSet::operator~(void)

{

TSet tmp(MaxPower);

tmp.BitField = ~BitField;

return tmp;

}

istream& operator>>(istream& istr, TSet& s)

{

int n;

cout << "Input count of element(|U| = " << s.GetMaxPower() << "): ";

istr >> n;

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

int val;

cin >> val;

s.BitField.SetBit(val);

}

return istr;

}

ostream& operator<<(ostream& ostr, const TSet& s)

{

for (int i = 0; i < s.MaxPower; i++) {

if (s.BitField.GetBit(i)) cout << i << " ";

}

return ostr;

}