

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 _____
_____/ Рысев М.Д./
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 Эратосфена»	8
3 Руководство программиста	9
3.1 Описание алгоритмов	9
3.1.1 Битовые поля	9
3.1.2 Множества	100
3.1.3 «Решето Эратосфена»	111
3.2 Описание программной реализации	133
3.2.1 Описание класса TBitField	133
3.2.2 Описание класса TSet	188
Заключение	244
Литература	255
Приложения	26
Приложение А. Реализация класса TBitField	26
Приложение Б. Реализация класса TSet	28

Введение

В программировании и анализе данных часто приходится 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 и 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.

1 Постановка задачи

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

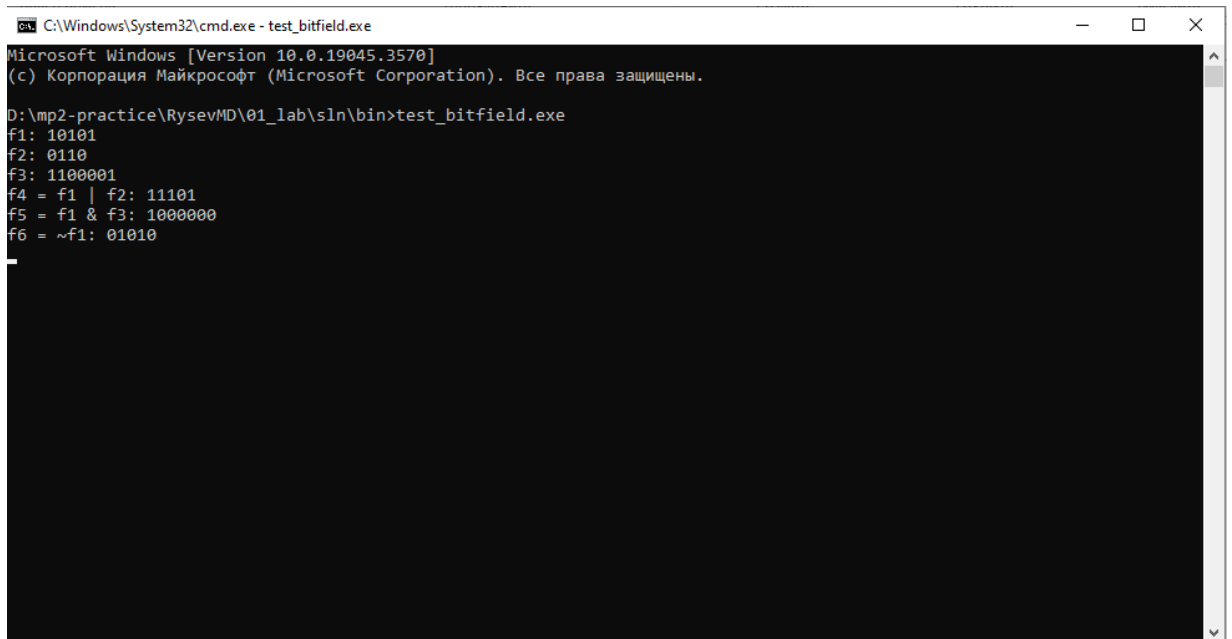
Задачи:

1. Ознакомиться с теорией 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.

2 Руководство пользователя

2.1 Приложение для демонстрации работы битовых полей

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

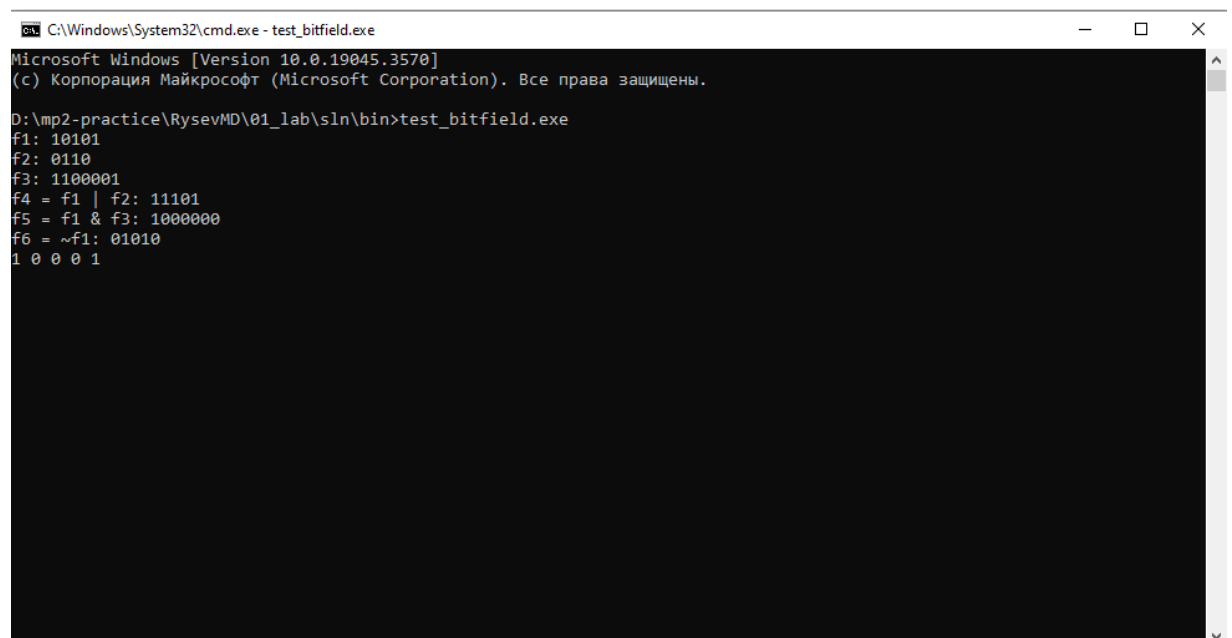


```
C:\Windows\System32\cmd.exe - test_bitfield.exe
Microsoft Windows [Version 10.0.19045.3570]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

D:\mp2-practice\RysevMD\01_lab\sln\bin>test_bitfield.exe
f1: 10101
f2: 0110
f3: 1100001
f4 = f1 | f2: 11101
f5 = f1 & f3: 1000000
f6 = ~f1: 01010
```

Рис. 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).

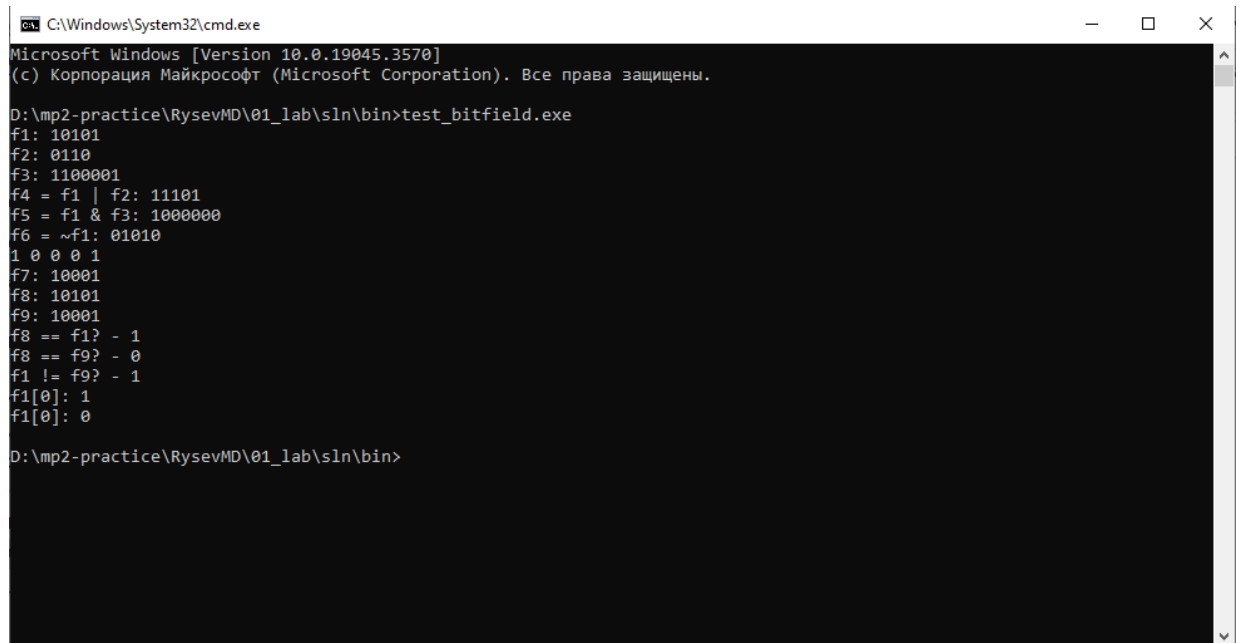


```
C:\Windows\System32\cmd.exe - test_bitfield.exe
Microsoft Windows [Version 10.0.19045.3570]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

D:\mp2-practice\RysevMD\01_lab\sln\bin>test_bitfield.exe
f1: 10101
f2: 0110
f3: 1100001
f4 = f1 | f2: 11101
f5 = f1 & f3: 1000000
f6 = ~f1: 01010
1 0 0 0 1
```

Рис. 2. Ввод нового битового поля

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



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.3570]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

D:\mp2-practice\RysevMD\01_lab\sln\bin>test_bitfield.exe
f1: 10101
f2: 0110
f3: 1100001
f4 = f1 | f2: 11101
f5 = f1 & f3: 1000000
f6 = ~f1: 01010
1 0 0 0 1
f7: 10001
f8: 10101
f9: 10001
f8 == f1? - 1
f8 == f9? - 0
f1 != f9? - 1
f1[0]: 1
f1[0]: 0

D:\mp2-practice\RysevMD\01_lab\sln\bin>
```

Рис. 3. Результат работы программы test_bitfield

2.2 Приложение для демонстрации работы множеств

1. Запустите приложение с названием test_set.exe. В результате появится окно, показанное ниже (рис. 14).

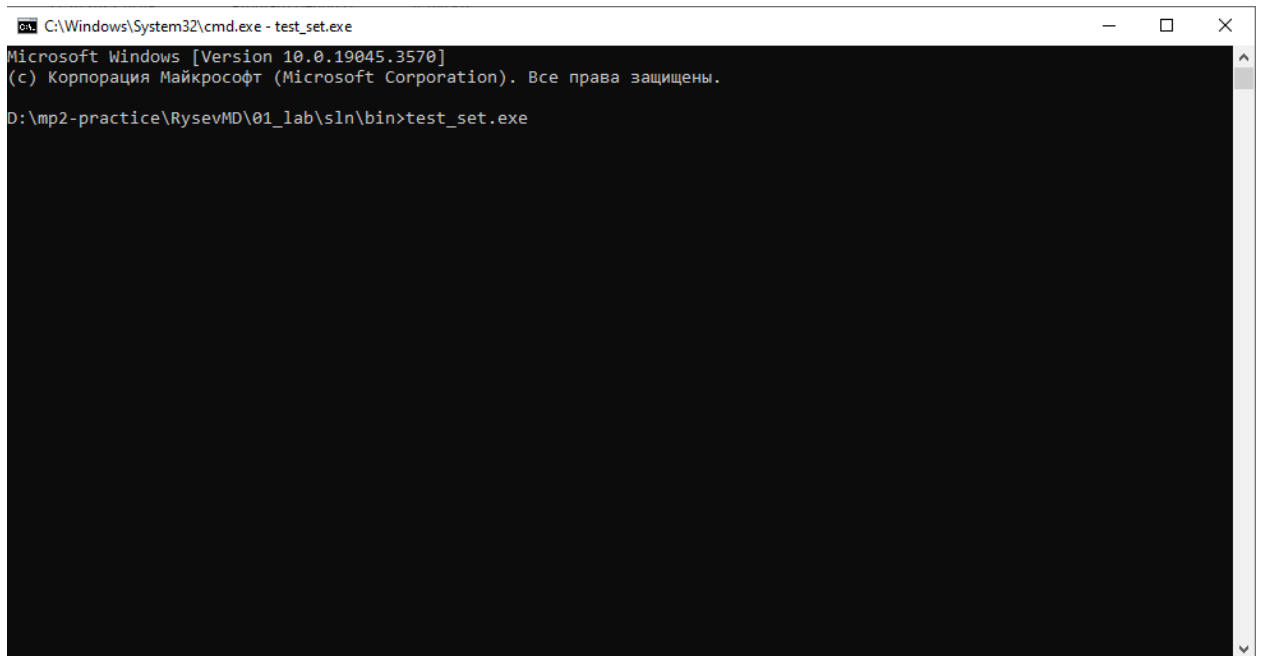


Рис. 4. Основное окно программы test_set

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).

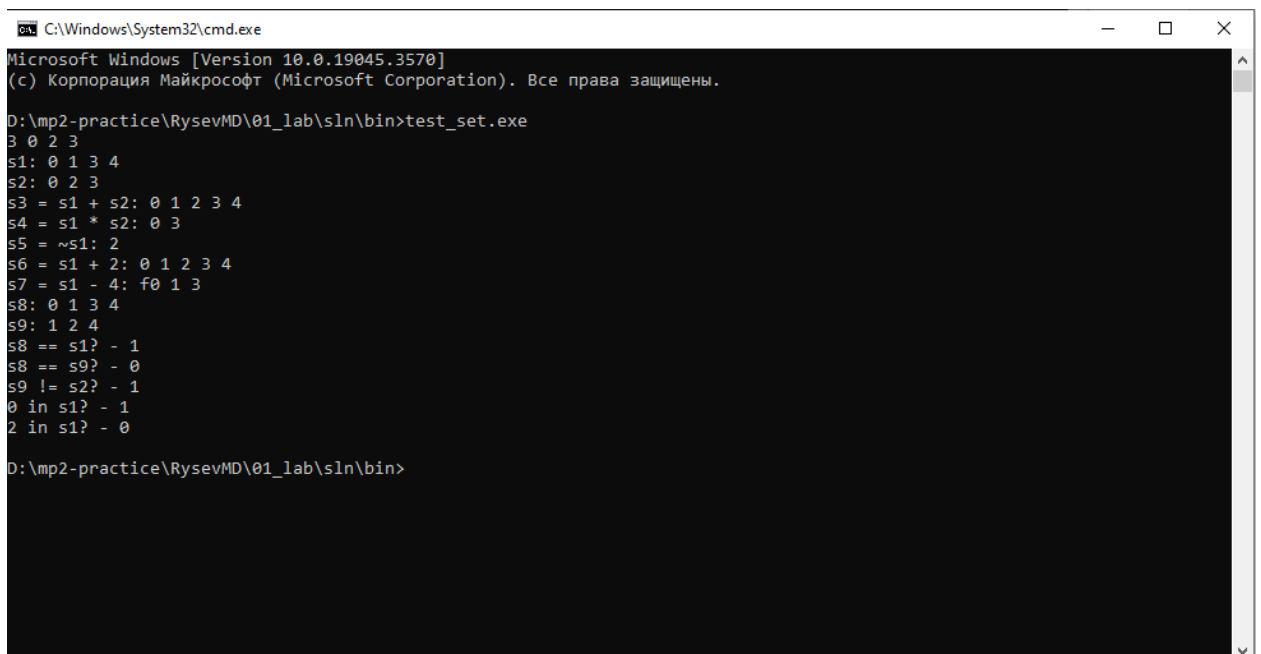


Рис. 5. Результат работы программы test_set

2.3 «Решето Эратосфена»

1. Запустите приложение с названием sample_prime_numbers.exe. В результате появится окно, показанное ниже (рис. 16).

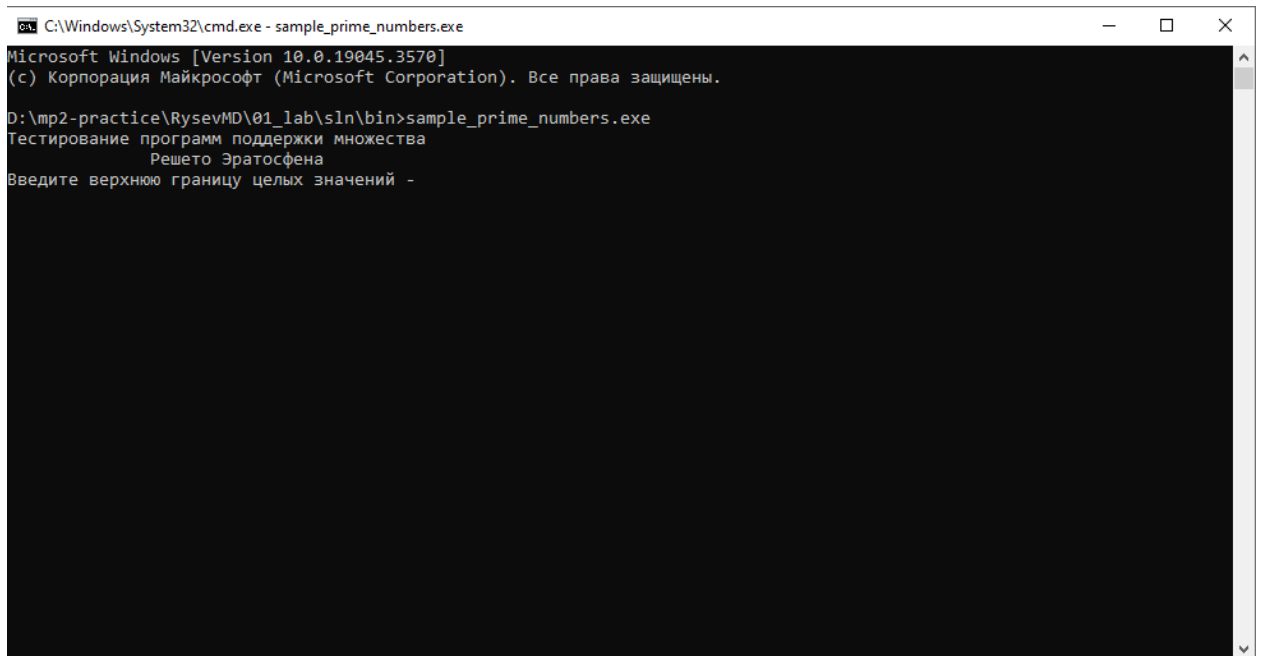


Рис. 6. Основное окно программы sample_prime_numbers

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).

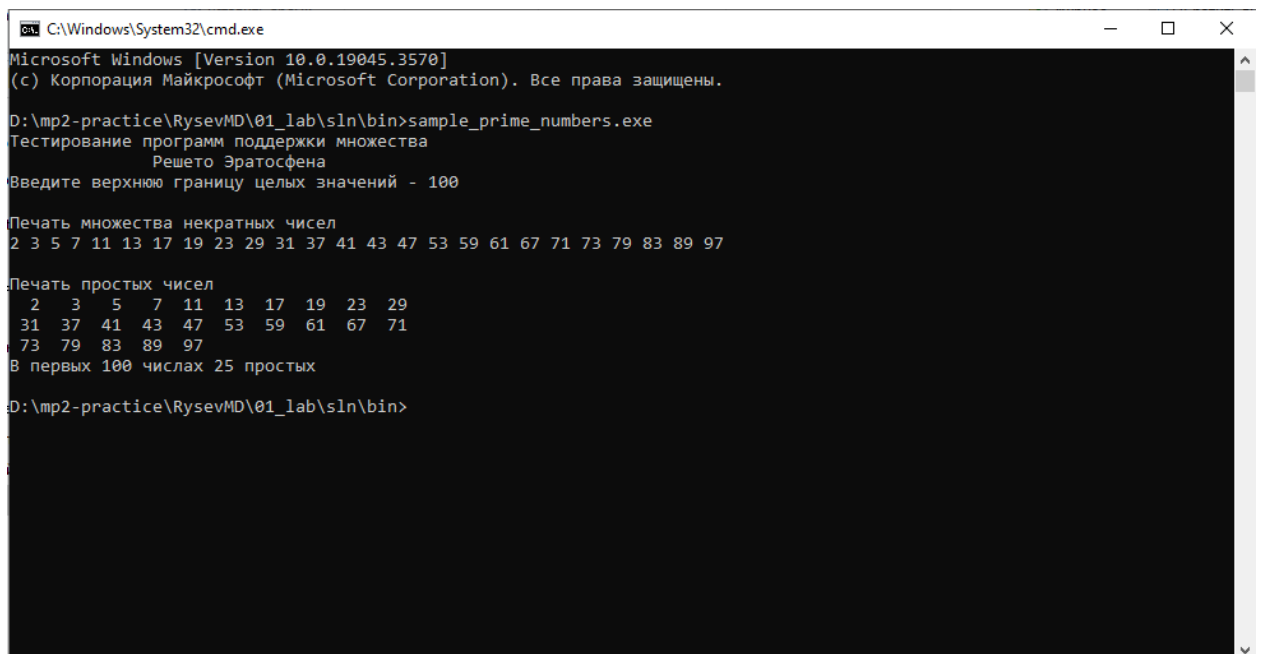


Рис. 7. Результат работы алгоритма “Решето Эратосфена”

3 Руководство программиста

3.1 Описание алгоритмов

3.1.1 Битовые поля

1. Описание класса TbitField указано в пункте 3.2.1.
2. Программа содержит только функцию main(), в которой описана демонстрация работы битового поля. Сначала происходит инициализация и заполнение трёх битовых полей и вывод их содержимого в консоль (рис. 8).

```
//создание битового поля, все ячейки изначально 0
TBitField f1(5);

//приведение поля к виду 10101
f1.SetBit(0);
f1.SetBit(2);
f1.SetBit(4);
cout << "f1: " << f1 << endl;

TBitField f2(4), f3(7);

f2.SetBit(1);
f2.SetBit(2);

f3.SetBit(0);
f3.SetBit(1);
f3.SetBit(6);

cout << "f2: " << f2 << endl;
cout << "f3: " << f3 << endl;
```

Рис. 8. Объявление и вывод полей

3. Затем выводятся результаты побитовых операций “и”, “или”, “отрицание”. Также здесь проверяется корректность работы конструктора копирования (рис. 9).

```
// или, и, отрицание, копирование
TBitField f4 = f1 | f2, f5 = f1 & f3, f6 = ~f1;
cout << "f4 = f1 | f2: " << f4 << endl << "f5 = f1 & f3: " << f5 << endl << "f6 = ~f1: " << f6 << endl;
```

Рис. 9. Побитовые операции, конструктор копирования

4. Далее идёт ввод нового битового поля, демонстрация операций сравнения и функций GetBit(), ClrBit() (рис. 10).

```

TBitField f7(5);
cin >> f7;
cout << "f7: " << f7 << endl;

//сравнения
TBitField f8 = f1, f9 = f7;
cout << "f8: " << f8 << endl;
cout << "f9: " << f9 << endl;
cout << "f8 == f1? - " << (f8 == f1) << endl << "f8 == f9? - " << (f8 == f9) << endl << "f1 != f9? - " << (f1 != f9) << endl;

//Get- Clear- Bit;
cout << "f1[0]: " << f1.GetBit(0) << endl;
f1.ClrBit(0);
cout << "f1[0]: " << f1.GetBit(0) << endl;

```

Рис. 10. Сравнение, GetBit(), ClrBit()

3.1.2 Множества

1. Описание класса TSet указано в пункте 3.2.2.
2. Программа содержит только функцию main(), в которой описана демонстрация работы множества. Сперва происходит создание двух полей, первое из которых заполняется с помощью метода InsElem(), а второе ручным вводом в консоль. Далее программа выводит значения этих множеств и результаты теоретико-множественных операций над ними (рис. 11). Также здесь демонстрируется работа конструктора копирования.

```

//объединение, пересечение, дополнение множеств
TSet s3 = s1 + s2, s4 = s1 * s2, s5 = ~s1;
cout << "s3 = s1 + s2: " << s3 << endl << "s4 = s1 * s2: " << s4 << endl << "s5 = ~s1: " << s5 << endl;

```

Рис. 11. Теоретико-множественные операции, конструктор копирования

3. Затем демонстрируется возможность разности и объединения множества с элементом (рис. 12). Элемент с которым производится операция должен принадлежать универсу.

```

//объединение, разность с элементом
TSet s6 = s1 + 2, s7 = s1 - 4;
cout << "s6 = s1 + 2: " << s6 << endl << "s7 = s1 - 4: f" << s7 << endl;

```

Рис. 12. Разность и объединение с элементом

Рис. 13.

4. Далее идёт демонстрация логических операций над множествами и проверка наличия элемента в множестве (рис. 13).

```
//сравнение множеств, проверка наличия элемента
TSet s8 = s1, s9(5);
s9.InsElem(2);
s9.InsElem(4);
s9.InsElem(1);
cout << "s8: " << s8 << endl << "s9: " << s9 << endl;
cout << "s8 == s1? - " << (s8 == s1) << endl << "s8 == s9? - " << (s8 == s9) << endl << "s9 != s2? - " << (s9 != s2) << endl;
cout << "0 in s1? - " << s1.IsMember(0) << endl << "2 in s1? - " << s1.IsMember(2) << endl;
```

Рис. 14. Логические операции, наличие элемента

3.1.3 «Решето Эратосфена»

1. Изначально программа будет настроена на работу с битовым полем. Чтобы переключить программу на работу с множеством достаточно просто раскомментировать объявление константы USE_SET (рис. 15).

```
//#define USE_SET // Использовать класс TSet,
// закомментировать, чтобы использовать битовое поле
#ifndef USE_SET // Использовать класс TBitField
```

Рис. 15. Константа USE_SET

2. Сперва программа запрашивает число, до которого будет производиться поиск простых чисел, затем создаётся экземпляр класса TSet или TBitField (в зависимости от объявления USE_SET) и происходит его заполнение (рис. 16).

```
int n, m, k, count;

setlocale(LC_ALL, "Russian");
cout << "Тестирование программ поддержки множества" << endl;
cout << "                Решето Эратосфена" << endl;
cout << "Введите верхнюю границу целых значений - ";
cin >> n;
TSet s(n + 1);
// заполнение множества
for (m = 2; m <= n; m++)
    s.InsElem(m);
```

Рис. 16. Создание и заполнение

3. Далее происходит проверка элементов от 2 до квадратного корня из n (числа введённых элементов). Если текущий элемент находится в множестве, то происходит удаление кратных ему элементов (рис. 17).

```

// проверка до sqrt(n) и удаление кратных
for (m = 2; m * m <= n; m++)
    // если m в s, удаление кратных
    if (s.IsMember(m))
        for (k = 2 * m; k <= n; k += m)
            if (s.IsMember(k))
                s.Delete(k);

```

Рис. 17. Удаление кратных

4. Затем производится вывод множества простых чисел и подсчёт количества элементов в нём (рис. 18).

```

// оставшиеся в s элементы – простые числа
cout << endl << "Печать множества некрatных чисел" << endl << s << endl;
cout << endl << "Печать простых чисел" << endl;
count = 0;
k = 1;
for (m = 2; m <= n; m++)
    if (s.IsMember(m))
    {
        count++;
        cout << setw(3) << m << " ";
        if (k++ % 10 == 0)
            cout << endl;
    }
cout << endl;
cout << "В первых " << n << " числах " << count << " простых" << endl;

```

Рис. 18. Вывод результатов

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();
    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.

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.

Поля:

MaxPower - количество элементов в множестве - мощность универса.

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 универса, которому принадлежит множество.

Входные параметры:

Отсутствуют.

Выходные параметры:

Размер универса.

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>

TBitField::TBitField(int len)
{
    if (len < 0) throw "length_cant_be_less_than_zero";
    BitLen = len;
    MemLen = ((BitLen - 1) >> 5) + 1;
    pMem = new TELEM[MemLen];
    memset(pMem, 0, MemLen * sizeof(TELEM));
}

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

TBitField::~TBitField()
{
    delete[] pMem;
}
```

Рис. 19. Реализация TBitField №1.

```
int TBitField::GetMemIndex(const int n) const // индекс Мем для бита n
{
    if (n >= BitLen || n < 0) throw "out_of_range";
    return n >> 5;
}

TELEM TBitField::GetMemMask(const int n) const // битовая маска для бита n
{
    if (n >= BitLen || n < 0) throw "out_of_range";
    return 1 << (BitLen - (n & 31) - 1);
}
```

Рис. 20. Реализация TBitField №2.

```
void TBitField::SetBit(const int n) // установить бит
{
    if (n >= BitLen || n < 0) throw "out_of_range";
    pMem[GetMemIndex(n)] |= GetMemMask(n);
}

void TBitField::ClrBit(const int n) // очистить бит
{
    if (n >= BitLen || n < 0) throw "out_of_range";
    pMem[GetMemIndex(n)] &= ~GetMemMask(n);
}

int TBitField::GetBit(const int n) const // получить значение бита
{
    if (n >= BitLen || n < 0) throw "out_of_range";
    if ((pMem[GetMemIndex(n)] & GetMemMask(n)) == 0) return 0;
    return 1;
}
```

Рис. 21. Реализация TBitField №3.

```

// битовые операции

const TBitField& TBitField::operator=(const TBitField& bf) // присваивание
{
    if (bf == (*this)) return (*this);
    BitLen = bf.BitLen;
    MemLen = bf.MemLen;
    pMem = new TELEM[MemLen];
    memcpy(pMem, bf.pMem, MemLen * sizeof(TELEM));
    return (*this);
}

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

```

Рис. 22. Реализация TBitField №4.

```

TBitField TBitField::operator|(const TBitField& bf) // операция "или"
{
    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) // операция "и"
{
    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;
}

```

Рис. 23. Реализация TBitField №5.

```

TBitField TBitField::operator~(void) // отрицание
{
    TBitField tmp(BitLen);
    for (int i = 0; i < MemLen; i++) {
        tmp.pMem[i] = ~pMem[i];
    }
    return tmp;
}

// ввод/вывод

istream& operator>>(istream& istr, TBitField& bf) // ввод
{
    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) // вывод
{
    for (int i = 0; i < bf.BitLen; i++) cout << bf.GetBit(i);
    return ostr;
}

```

Рис. 24. Реализация TBitField №6.

Приложение Б. Реализация класса TSet

```

#include "tset.h"

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

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

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

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

```

Рис. 25. Реализация TSet №1.

```

int TSet::GetMaxPower(void) const // получить макс. к-во эл-тов
{
    return MaxPower;
}

int TSet::IsMember(const int Elem) const // элемент множества?
{
    if (Elem >= MaxPower) throw "Element not in universe";
    if (BitField.GetBit(Elem)) return 1;
    return 0;
}

void TSet::InsElem(const int Elem) // включение элемента множества
{
    if (Elem >= MaxPower) throw "Element not in universe";
    BitField.SetBit(Elem);
}

void TSet::DelElem(const int Elem) // исключение элемента множества
{
    if (Elem >= MaxPower) throw "Element not in universe";
    BitField.ClrBit(Elem);
}

```

Рис. 26. Реализация TSet №2.

```

// теоретико-множественные операции

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

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

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

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

```

Рис. 27. Реализация TSet №3.

```

TSet TSet::operator+(const int Elem) // объединение с элементом
{
    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) // разность с элементом
{
    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) // пересечение
{
    TSet tmp(max(MaxPower, s.GetMaxPower()));
    tmp.BitField = (BitField & s.BitField);
    return tmp;
}

```

Рис. 28. Реализация TSet №4.

```

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

// перегрузка ввода/вывода

istream& operator>>(istream& istr, TSet& s) // ввод
{
    for (int i = 0; i < s.MaxPower; i++) {
        int val;
        cin >> val;
        s.BitField.SetBit(val);
    }
    return istr;
}

ostream& operator<<(ostream& ostr, const TSet& s) // вывод
{
    for (int i = 0; i < s.MaxPower; i++) {
        if (s.BitField.GetBit(i)) cout << i << " ";
    }
    return ostr;
}

```

Рис. 29. Реализация TSet №5.