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:** студент группы 3822Б1ФИ2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Табунов И.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 6](#_Toc147915970)

[2.3 «Решето Эратосфено» 7](#_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» 12](#_Toc147915976)

[3.2 Описание программной реализации 12](#_Toc147915977)

[3.2.1 Описание класса TBitField 12](#_Toc147915978)

[3.2.2 Описание класса TSet 15](#_Toc147915979)

[Заключение 19](#_Toc147915980)

[Литература 20](#_Toc147915981)

[Приложения 21](#_Toc147915982)

[Приложение А. Реализация класса TBitField 21](#_Toc147915983)

[Приложение Б. Реализация класса TSet 24](#_Toc147915984)  
Приложение В. Sample\_primenumbers………………………………..………………27

# Введение

Проект «Множества» использует интерфейс битовых полей для работы с теоретико-множественными операциями. Данный вариант решения задачи дает нам возможность использовать только часть предоставляемой типом данных памяти. Обращение к биту с определенным индексом позволяет нам узнать его состояние. Например, относится ли элемент к данному множеству. Любое множество описывается характеристическим вектором, а наиболее эффективный способ его представить/хранить – битовое поле (битовая строка).

# Постановка задачи

**Цель**:

изучение и практическое применение концепции битовых полей и множеств.

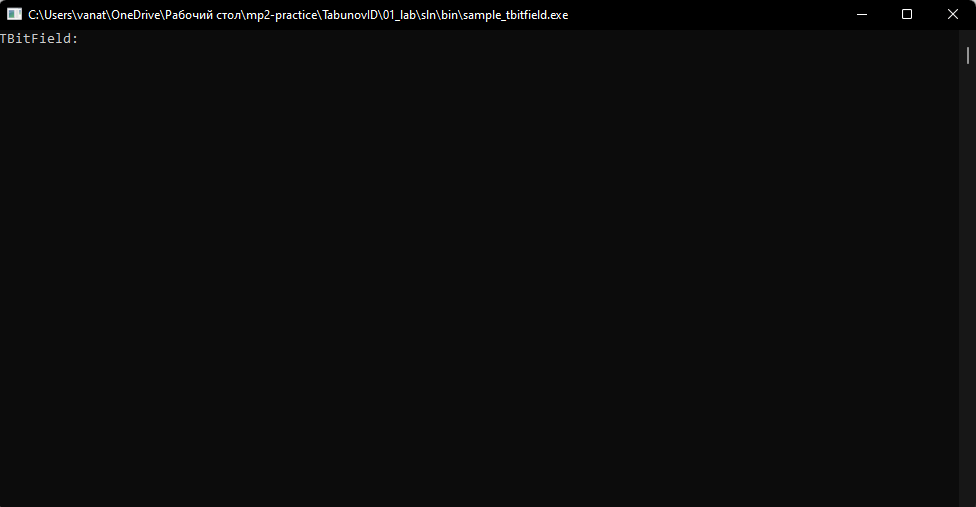
**Задачи**:

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

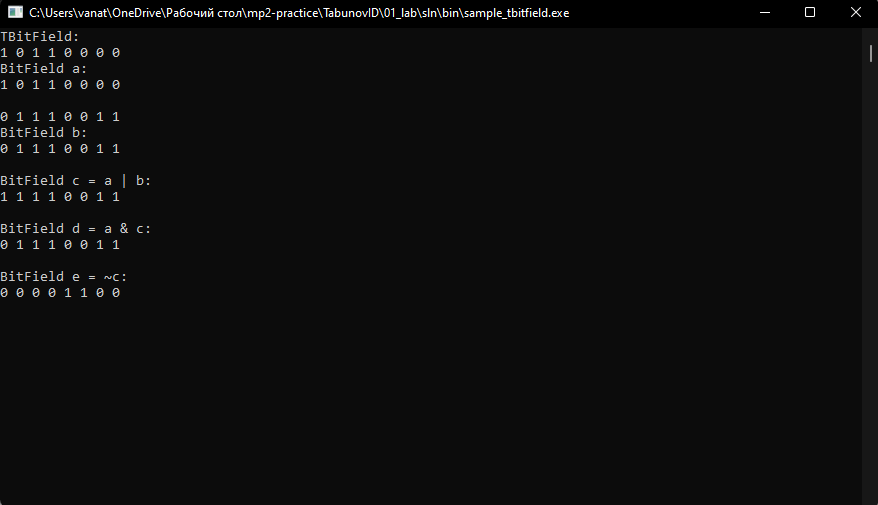
# Руководство пользователя

## Приложение для демонстрации работы битовых полей

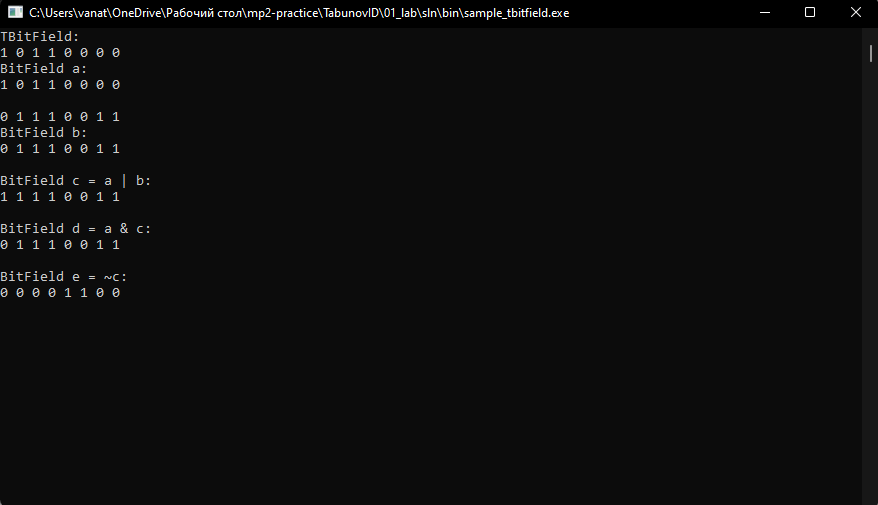
1. Запустите приложение с названием sample\_tbitfield.exe. В результате появится окно, показанное ниже (рис. 1).



1. Основное окно программы
2. Затем вам будет предложено ввести 2 битовых поля длины 8 (рис. 2).



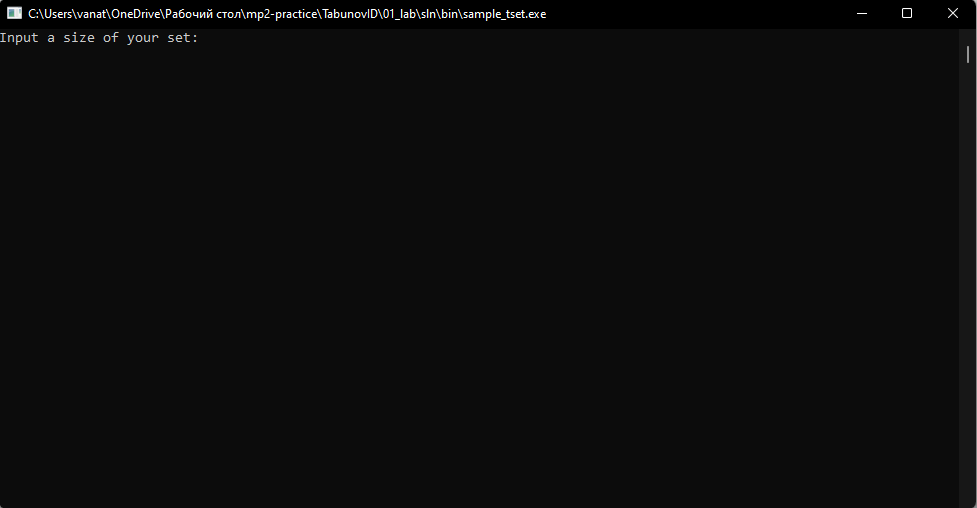
1. Создание битовых полей
2. После ввода множества нулей и /или единиц, будет выведены результаты соответствующих операций и функций (рис. 3).



1. Результат тестирования класса

## Приложение для демонстрации работы множеств

1. Запустите приложение с названием sample\_tset.exe. В результате появится окно, показанное ниже (рис. 4).



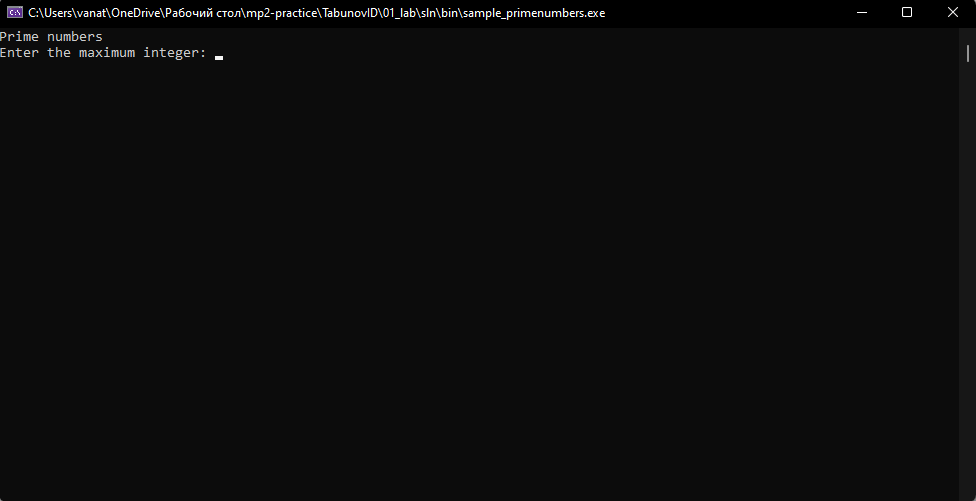
1. Основное окно программы
2. Затем вам будет предложено вести свое множество. Необходимо ввести сначала количество чисел в множестве , программа будет ждать N чисел. После ввода множества нулей и /или единиц, будет выведены результаты соответствующих операций и функций (рис. 5).



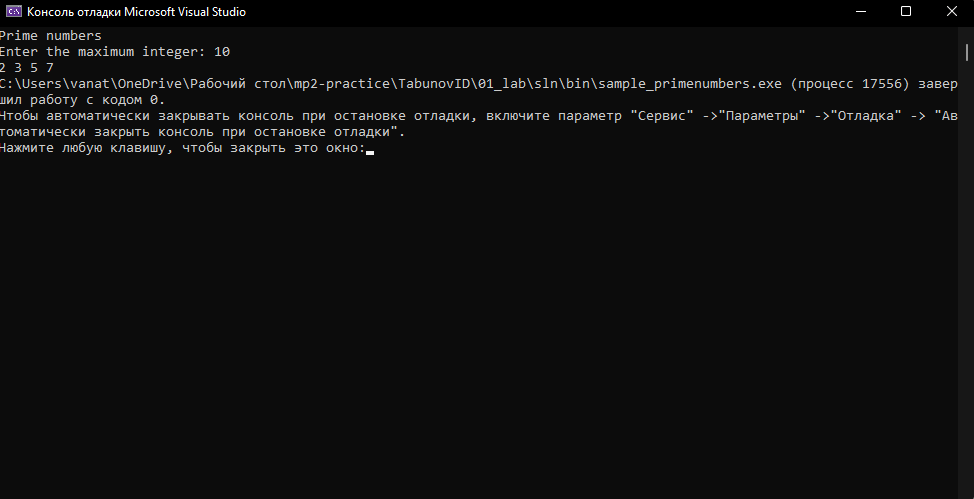
1. Результат работы класса

## «Решето Эратосфена»

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



1. Запрос программы
2. Затем вам будет нужно ввести целое положительное число. После чего программа выведет простые числа на отрезке до введенного числа и их количество (Рис. 7).



1. Простые числа от 2 до 10

# Руководство программиста

## Описание алгоритмов

### Битовые поля

Битовые поля представляют собой набор чисел, каждый бит которых интерпретируется элементом, равным индексом бита. 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.

Битовое поле поддерживает операции объединения, пересечения, дополнение (отрицание), сравнения, ввода и вывода.

**Операция объединения:**

Операция возвращает экземпляр класса, каждый бит которого равен 1, если он есть хотя бы в 1 классе, которые объединяем, и 0 в противном случае.

Пример:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| B | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| A|B | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |

**Операция пересечения:**

Операция возвращает экземпляр класса, каждый бит которого равен 1, если он есть в каждом классе, и 0 в противном случае.

Пример:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| B | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| A&B | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

**Операция дополнения (отрицания):**

Операция возвращает экземпляр класса, каждый бит которого равен 0, если он есть исходном классе, и 1 в противном случае.

Пример:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| ~A | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |

**Операция ввода битового поля:**

Операция позволяет ввести битовое поле из консоли. Для этого необходимо в консоль ввести 1 или 0 столько раз, какова длинна вводимого битового поля.

**Операция вывода битового поля:**

Операция вывода позволяет вывести битовое поле в консоль. Также в битовом поле можно установить или очистить бит.

Пример: установить бит k (=3) в 1

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

|

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

=

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

Пример: очистить бит (установить 0) k (=5)

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

&

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

=

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

**Операции сравнения**

Операция равенства выведет 1, если два битовых поля равны, или каждые их биты совпадают, 0 в противном случае.

**Операция сравнения на неравенство**

Операция сравнения на неравенство выведет 0, если хотя бы два бита совпадают, 1 в противном случае.

### Множества

Множества представляют собой набор целых положительных чисел. В данной лабораторной работе множество реализовано при помощи битового поля, соответственно каждый бит которых интерпретируется элементом, равным индексом бита. 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. Создание множества через битовые поля может сильно сократить использование памяти.

Множество поддерживает операции объединения, пересечения, дополнение (отрицание), сравнения, ввода и вывода.

**Операция объединения с множеством:**

Операция возвращает экземпляр класса, содержащий все уникальные элементы из двух классов.

Пример: A = {2, 4, 5}, B = {0, 1, 3, 5}

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

|

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

=

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

A + B = C = {0, 1, 2, 3, 4, 5}

**Операция пересечения с множеством:**

Операция возвращает экземпляр класса, содержащий все уникальные элементы, которые находятся в каждом классе.

Пример: A = {2, 4, 5}, B = {0, 1, 3, 5}

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

&

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

=

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

A & B = C = {5}

**Операция дополнения (отрицания):**

Операция возвращает экземпляр, класса содержащий элементы, которых нет в исходном классе, не большие максимальному элементу.

Пример: A = {2, 4, 5}

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

~

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

~A = {0, 1, 3}

**Операция ввода множества:**

Операция позволяет ввести множество из консоли. Для этого необходимо в консоль ввести сначала количество элементов в множестве, а дальше сами элементы, причём элементы не могут превышать максимального элемента множества.

**Операция вывода множества:**

Операция вывода позволяет вывести множество в консоль. Также в множестве можно добавить или убрать элемент.

### «Решето Эратосфена»

Задача: дано целое положительное число. Требуется найти все простые числа в отрезке от 2 до этого числа.

Входные данные: целое положительное число (далее n)

Выходные данные: множество простых чисел

Суть алгоритма: идти по натуральным числам и на каждом найденном в множестве числе исключать все элементы, кратные тому, на котором сейчас остановились.

Алгоритм:

1. Выписать подряд все числа от 2 до n
2. Пусть переменная p изначально равна двум – первому простому числу
3. Зачеркнуть в списке числа от 2p до n, считая шагами по p (то есть удаляем числа, кратные p).
4. Найти первое не зачеркнутое число в списке, большее чем p, и присвоить значению переменной p это число
5. Повторять шаги 3 и 4, пока возможно

## Описание программной реализации

### Описание класса TBitField

class TBitField

{

private:

int BitLen;

TELEM \*pMem;

int MemLen;

const tint bitsInElem = 32;

cons tint shiftSize = 5;

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.

bitsInElem – количество битов в элементе памяти.

shiftSize – вспомогательное значение для битового целочисленного деления.

Методы:

TBitField(int len);

Назначение: выделение и инициализация памяти объекта.

Входные параметры: len – количество доступных битов.

TBitField(const TBitField &bf);

Назначение: выделение памяти и копирование данных.

Входные параметры: const TBitField &bf – константная ссылка на битовое поле.

~TBitField();

Назначение: освобождение выделенной памяти.

int GetMemIndex(const int n) const;

Назначение: получение индекса элемента в памяти.

Входные параметры: n – номер бита.

Выходные параметры: индекс элемента в памяти.

TELEM GetMemMask (const int n) const;

Назначение: получение маски бита.

Входные параметры: n – номер бита.

Выходные параметры: маска бита.

int GetLength(void) const;

Назначение: получение количества доступных битов.

Выходные параметры: BitLen – количество доступных битов.

void SetBit(const int n);

Назначение: установить бит в 1.

Входные параметры: n – номер бита.

void ClrBit(const int n);

Назначение: установить бит в 0.

Входные параметры: n – номер бита.

int GetBit(const int n) const;

Назначение: получение значения бита.

Входные параметры: n – номер бита.

Выходные параметры: значение бита (0 или 1).

Операции:

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

Назначение: перегрузка операции сравнения на равенство объектов.

Входные параметры: bf – битовое поле – объект класса TBitField.

Выходные параметры: целое число (0 или 1).

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

Назначение: перегрузка операции сравнения на неравенство объектов.

Входные параметры: bf – битовое поле – объект класса TBitField.

Выходные параметры: целое число (0 или 1).

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

Назначение: присвоение значение полей одного битового поля другому.

Входные параметры: bf – битовое поле – объект класса TBitField.

Выходные параметры: константная ссылка на объект класса TBitField.

TBitField operator|(const TBitField &bf);

Назначение: сложение двух битовых полей.

Входные параметры: bf – битовое поле – объект класса TBitField.

Выходные параметры: результирующее битовое поле.

TBitField operator&(const TBitField &bf);

Назначение: умножение двух битовых полей.

Входные параметры: bf – битовое поле – объект класса TBitField.

Выходные параметры: результирующее битовое поле.

TBitField operator~(void);

Назначение: инвертирование значений битов битового поля.

Входные параметры: bf – битовое поле – объект класса TBitField.

Выходные параметры: результирующее битовое поле.

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

Назначение: ввод данных.

Входные параметры: istr – поток ввода, bf – битовое поле – объект класса TBitField.

Выходные параметры: поток ввода.

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

Назначение: вывод данных.

Входные параметры: ostr – поток вывода, bf – битовое поле – объект класса TBitField.

Выходные параметры: 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:

Max Power – мощность множества.

BitField – характеристический вектор, битовое поле.

Методы:

TSet(int mp);

Назначение: инициализация битового поля.

Входные параметры: mp – количество элементов в универсе, мощность множетсва.

TSet(const TSet &s);

Назначение: копирование данных из другого множества.

Входные параметры: s – множество, объект класса TSet.

TSet(const TBitField &bf);

Назначение: преобразование из TBitField в TSet.

Входные параметры: bf– битовое поле, объект класса TBitField.

operator TBitField();

Назначение: преобразование из TSet в TBitField.

Выходные параметры: объект класса TBitField.

int GetMaxPower(void) const;

Назначение: получение мощности множества.

Выходные параметры: MaxPower – мощность множества.

**void InsElem(const int Elem);**

Назначение: добавление элемента в множество.

Входные параметры Elem – добавляемый элемент.

void DelElem(const int Elem);

Назначение: удаление элемента из множества.

Входные параметры Elem – удаляемый элемент.

int IsMember(const int Elem) const;

Назначение: проверка на принадлежность элемента множеству.

Входные параметры Elem – проверяемый элемент.

Выходные параметры: значение бита (о или 1).

Операции:

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

Назначение: перегрузка операции сравнения, проверка на равенство двух множеств.

Входные параметры: s – множество – объект класса TSet.

Выходные параметры: целое число (0 или 1).

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

Назначение: перегрузка операции сравнения, проверка на неравенство двух множеств.

Входные параметры: s – множество – объект класса TSet.

Выходные параметры: целое число (0 или 1).

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

Назначение: присвоение значений полей одного объекта класса другому.

Входные параметры: s – множество – объект класса TSet.

Выходные параметры: ссылка на объект своего класса TSet.

TSet operator+ (const int Elem);

Назначение: добавление элемента в множество.

Входные параметры: Elem – индекс элемента.

Выходные параметры: результирующее множество, объект класса TSet.

TSet operator- (const int Elem);

Назначение: удаление элемента из множества.

Входные параметры: Elem – индекс элемента.

Выходные параметры: результирующее множество, объект класса TSet.

TSet operator+ (const TSet &s);

Назначение: объединение двух множеств.

Входные параметры: S – объект класса TSet.

Выходные параметры: результирующее множество, объект класса TSet.

TSet operator\* (const TSet &s);

Назначение: пересечение двух множеств.

Входные параметры: S – объект класса TSet.

Выходные параметры: результирующее множество, объект класса TSet.

TSet operator~ (void);

Назначение: получение дополнения к множеству.

Выходные параметры: результирующее множество, объект класса TSet.

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

Назначение: ввод данных.

Входные параметры: istr – поток ввода, bf – объект класса TSet.

Выходные параметры: поток ввода.

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

Назначение: вывод данных.

Входные параметры: ostr – поток вывода, bf – объект класса TSet.

Выходные параметры: поток вывода.

# Заключение

В ходе выполнения данной работы, были разработаны и реализованы библиотеки для представления битового поля TBitField и множества TSet. Также были написаны программы, для демонстрации работы всех методов и операций классов. Был написан алгоритм «Решето Эратосфена», который также показывает корректную работу написанного класса TSet.

# Литература

1. Битовое поле [https://ru.wikipedia.org/wiki/Битовое\_поле].

# Приложения

## Приложение А. Реализация класса TBitField

#include "tbitfield.h"

TBitField::TBitField(int len)

{

if (len <= 0)

throw "len\_is\_equal\_zero!";

BitLen = len;

MemLen = ((len + bitsInElem - 1) >> shiftSize);

pMem = new TELEM[MemLen];

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

{

pMem[i] = 0;

}

}

TBitField::TBitField(const TBitField &bf) // конструктор копирования

{

BitLen = bf.BitLen;

MemLen = bf.MemLen;

pMem = new TELEM[MemLen];

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

{

pMem[i] = bf.pMem[i];

}

}

TBitField::~TBitField()

{

delete[] pMem;

}

int TBitField::GetMemIndex(const int n) const // индекс Мем для бита n

{

if (n < 0)

throw "n\_is\_below\_zero";

return n >> shiftSize; //сдвиг n вправо на shiftSize бит

}

TELEM TBitField::GetMemMask(const int n) const // битовая маска для бита n

{

if (n > BitLen)

throw "overload\_n";

if (n < 0)

throw "n\_is\_below\_zero";

return 1 << (n & (bitsInElem - 1)); //сдвиг 1 влево на (n & (bitsInElem - 1)) бит

}

// доступ к битам битового поля

int TBitField::GetLength(void) const // получить длину (к-во битов)

{

return BitLen;

}

void TBitField::SetBit(const int n) // установить бит

{

if (n > BitLen)

throw "overload\_n";

if (n < 0)

throw "n\_is\_below\_zero";

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

}

void TBitField::ClrBit(const int n) // очистить бит

{

if (n > BitLen)

throw "overload\_n";

if (n < 0)

throw "n\_is\_below\_zero";

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

}

int TBitField::GetBit(const int n) const // получить значение бита

{

if (n > BitLen)

throw "overload\_n";

if (n < 0)

throw "n\_is\_below\_zero";

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

}

// битовые операции

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

{

if (this == &bf)

{

return \*this;

}

if (MemLen != bf.MemLen)

{

if (MemLen > 0)

{

delete[] pMem;

MemLen = bf.MemLen;

pMem = new TELEM[MemLen];

}

}

BitLen = bf.BitLen;

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

{

pMem[i] = bf.pMem[i];

}

return \*this;

}

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

{

if (BitLen != bf.GetLength())

return 0;

else

{

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

{

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

return 0;

}

}

return 1;

}

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

{

return !(\*this == bf);

}

TBitField TBitField::operator|(const TBitField &bf) // операция "или"

{

int maxlen = max(GetLength(), bf.GetLength());

int minlen = min(GetLength(), bf.GetLength());

TBitField tmp(1);

if (GetLength() > bf.GetLength())

tmp = \*this;

else

tmp = bf;

int i = 0;

for (; i < minlen; ++i)

{

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

tmp.SetBit(i);

}

return tmp;

}

TBitField TBitField::operator&(const TBitField& bf) // операция "и"

{

int maxlen = max(GetLength(), bf.GetLength());

int minlen = min(GetLength(), bf.GetLength());

TBitField tmp(maxlen);

int i = 0;

for (; i < minlen; ++i)

{

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

tmp.SetBit(i);

}

return tmp;

}

TBitField TBitField::operator~(void) // отрицание

{

TBitField tmp(GetLength());

for (int i = 0; i <= GetMemIndex(GetLength()); ++i)

{

tmp.pMem[i] = ~pMem[i];

}

return tmp;

}

// ввод/вывод

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

{

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

{

int val;

istr >> val;

if (val) bf.SetBit(i);

}

return istr;

}

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

{

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

{

ostr << bf.GetBit(i) << " ";

}

ostr << "\n";

return ostr;

}

## 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.GetMaxPower();

}

// конструктор преобразования типа

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

{

MaxPower = bf.GetLength();

}

TSet::operator TBitField()

{

return BitField;

}

int TSet::GetMaxPower(void) const // получить макс. к-во эл-тов

{

return MaxPower;

}

int TSet::IsMember(const int Elem) const // элемент множества?

{

if ((Elem < 0) && (Elem >= MaxPower))

throw "Out of Range";

return BitField.GetBit(Elem);

}

void TSet::InsElem(const int Elem) // включение элемента множества

{

if ((Elem < 0) && (Elem >= MaxPower))

throw "Out of Range";

return BitField.SetBit(Elem);

}

void TSet::DelElem(const int Elem) // исключение элемента множества

{

if ((Elem < 0) && (Elem >= MaxPower))

throw "Out of Range";

return BitField.ClrBit(Elem);

}

// теоретико-множественные операции

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

{

if (this == &s)

{

return \*this;

}

MaxPower = s.MaxPower;

BitField = s.BitField;

return \*this;

}

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

{

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;

}

TSet TSet::operator+(const int Elem) // объединение с элементом

{

if ((Elem<0) && (Elem>MaxPower))

throw "Out of Range";

TSet tmp(\*this);

tmp.BitField.SetBit(Elem);

return tmp;

}

TSet TSet::operator-(const int Elem) // разность с элементом

{

if ((Elem<0) && (Elem>MaxPower))

throw "Out of Range";

TSet tmp(\*this);

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;

}

TSet TSet::operator~(void) // дополнение

{

TSet tmp(MaxPower);

tmp.BitField = ~BitField;

return tmp;

}

// перегрузка ввода/вывода

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

{

cout << "Input max power of your set" << endl;

int power;

cin >> power;

cout << "Input your set" << endl;

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

{

if ((i < 0) || (i > s.MaxPower)) {

throw "Out of Range";

}

s.InsElem(i);

}

}

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

{

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

if (s.BitField.GetBit(i)) {

ostr << i << ' ';

}

}

ostr << endl;

return ostr;

}

## Приложение В. Sample\_primenumbers

#include <iostream>

#include "tset.h"

int main()

{

std::cout << "Prime numbers" << std::endl;

int n;

std::cout << "Enter the count of numbers: ";

std::cin >> n;

TSet s(n + 1);

for (int m = 2; m <= n; m++)

s.InsElem(m);

for (int m = 2; m \* m <= n; m++)

if (s.IsMember(m))

for (int k = 2 \* m; k <= n; k += m)

if (s.IsMember(k))

s.DelElem(k);

std::cout << "Prime numbers under " << n << ": " << endl;

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

if (s.IsMember(i))

std::cout << i << ' ';

std::cout << endl;

return 0;

}