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»**

**Выполнила:** студентка группы 3822Б1ФИ2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Ясакова Т.Е./

Подпись

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

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

Подпись

Нижний Новгород  
2023

**Содержание**

[Введение 3](#_Toc153825531)

[1 Постановка задачи 4](#_Toc153825532)

[2 Руководство пользователя 5](#_Toc153825533)

[2.1 Приложение для демонстрации работы битовых полей 5](#_Toc153825534)

[2.2 Приложение для демонстрации работы множеств 5](#_Toc153825535)

[2.3 «Решето Эратосфено» 6](#_Toc153825536)

[3 Руководство программиста 8](#_Toc153825537)

[3.1 Описание алгоритмов 8](#_Toc153825538)

[3.1.1 Битовые поля 8](#_Toc153825539)

[3.1.2 Множества 9](#_Toc153825540)

[3.1.3 «Решето Эратосфена» 11](#_Toc153825541)

[3.2 Описание программной реализации 11](#_Toc153825542)

[3.2.1 Описание класса TBitField 11](#_Toc153825543)

[3.2.2 Описание класса TSet 14](#_Toc153825544)

[Заключение 18](#_Toc153825545)

[Литература 19](#_Toc153825546)

[Приложения 20](#_Toc153825547)

[Приложение А. Реализация класса TBitField 20](#_Toc153825548)

[Приложение Б. Реализация класса TSet 22](#_Toc153825549)

[Приложение В. Sample\_primenumbers 24](#_Toc153825550)

# Введение

В современном программировании и анализе данных часто возникает необходимость работы с большими объемами информации и эффективным представлением данных. Битовые поля и множества являются одним из инструментов, которые позволяют компактно хранить и манипулировать множествами элементов. Битовые поля и множества позволяют представить множество элементов в виде битовых векторов, где каждый бит соответствует наличию или отсутствию элемента в множестве. Такое представление позволяет существенно сократить объем памяти, занимаемый множеством, и ускорить операции над ними.

# Постановка задачи

**Цель работы:** изучение и практическое применение концепции битовых полей и множеств.

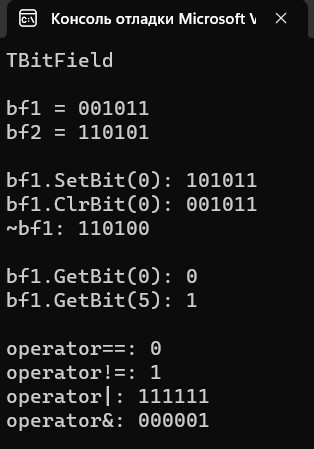
**Задачи:**

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 ().



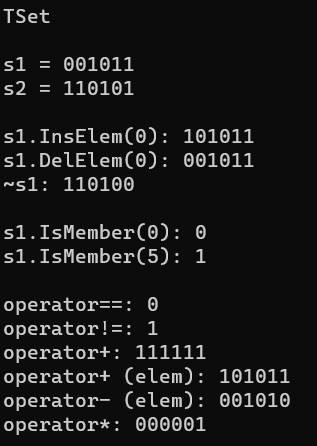
1. Основное окно программы sample\_tbitfield.exe
2. В появившемся окне вы можете ознакомиться с примером работы реализованных операций. Введите строку с данными битового поля, она должна содержать “0” и “1”, а также быть указанной длины (в данном случае 6). Другие символы будут считаться за “0”, а из строки большей длинны будет учитываться только подстрока указанной длины. Нажмите кнопку ввода, и выведется результат, пример которого указан на рисунке ниже ().



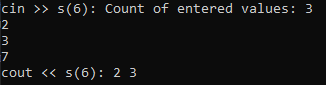
1. Пример функций ввода и вывода класса TBitField

## Приложение для демонстрации работы множеств

1. Запустите приложение с названием sample\_tset.exe. В результате появится окно, показанное ниже (рис. 3).



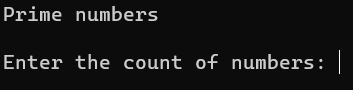
1. Основное окно программы sample\_tset.exe
2. В появившемся окне вы можете ознакомиться с примером работы реализованных операций. Введите строку с данными битового поля, она должна содержать “0” и “1”, а так же быть указанной длины (в данном случае 6). Другие символы будут считаться за “0”, а из строки большей длинны будет учитываться только подстрока указанной длины. Нажмите кнопку ввода, и выведется результат, пример которого указан на рисунке ниже ().



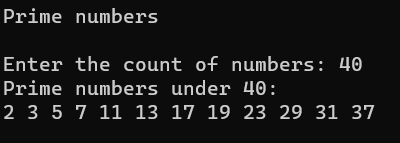
1. Пример функций ввода и вывода класса TSet

## «Решето Эратосфено»

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



1. Начало работы программы sample\_primenumbers.exe
2. Введите положительное целое число, чтобы вывести все простые числа до этого числа (включительно). Напечатается результат, показанный на рисунке ниже (рис. 6).



1. Результат работы программы sample\_primenumbers.exe

# Руководство программиста

## Описание алгоритмов

### Битовые поля

Битовые поля представляют собой набор нулей и единиц, где индексы каждого элемента – это элементы множества. Элемент битового поля может находиться в двух состояниях: 0 – элемент не содержится в множестве, 1 – элемент содержится в множестве. Битовые поля позволяют экономить память в случае, когда полю нужно информации меньше чем один байт. В памяти они представляются через обычные типы данных, например, через unsigned char:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Индекс Элемента | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Значение бита | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |

С битовыми полями можно производить операции, которые присущи множествам, именно поэтому множество можно представить как битовое поле.

Бинарная операция «Логическое или». Операция предполагает выполнение дизъюнкции над соответствующими битами двух полей.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Первое битовое поле | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |

|

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Второе битовое поле | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |

=

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

Бинарная операция «Логическое и». Каждый бит одного битового поля производит конъюнкцию с каждым битом другого битового поля:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Первое битовое поле | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |

&

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Второе битовое поле | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |

=

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

Унарная операция «Отрицание». Каждый бит битового поля меняется на противоположный:

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

=

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

Можно добавить единицы, что в множестве означает добавить элемент. Делают это через побитовые операции и «Логическое или»:

**Операция установить бит номер i в единицу:**

Рассмотрим битовое поле, у которого i – ый бит равен 0. Чтобы установить его в единицу, сначала создаём маску с помощью следующей битовой операции: (1 << i). Затем производим операцию «Логическое или» между битовым полем и маской. В результате получаем исходное битовое поле, у которого i –ый бит равен 1.

Пример (i =3):

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Битовое поле | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |

|

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

=

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

Можно обратить единицу в ноль, что в множестве означает удалить элемент. Делают это через побитовые операции и «Логическое и»:

**Операция установить бит номер i в ноль:**

Рассмотрим битовое поле, у которого i – ый бит равен 1. Чтобы установить его в 0, сначала создаём маску с помощью следующих битовых операций: ~(1 << i). Затем производим операцию «Логическое и» между битовым полем и маской. В результате получаем исходное битовое поле, у которого i –ый бит равен 0.

Пример (i =3):

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Битовое поле | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |

&

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

=

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

### Множества

Множеством может служить любой счётный набор каких-либо элементов, которые можно проиндексировать (элементов должно быть меньше чем (232 - 1)). Множество можно хранить в виде характеристического массива, длина которого совпадает с размером универса, а каждый элемент принимает значение 0 или 1, 0 соответствует отсутствию элемента во множестве, 1 – наличию. Характеристический массив можно хранить в виде битового поля.

Некоторые факты из теории множеств:

* То, что данный объект Х есть элемент множества М, записывают так: X М
* Если характеристическим свойством не обладает вообще ни один объект, говорят, что это свойство определяет пустое множество.
* Если каждый элемент множества А является в то же время элементом множества В, то множество А называется подмножеством множества В (А В)
* Одна из определяющих характеристик множества - его мощность
* Для множеств с конечным числом элементов мощность определяется как количество элементов множества
* Для конечных множеств принята форма записи А = {a1, a2, …, an}
* Множество всех возможных элементов называется универс и обычно обозначается U.

Теоретико-множественные операции применимые к множествам:

Операция «Объединение двух множеств». В результате этой операции получаем множество, в котором есть элементы и первого, и второго множества. Реализуется как операция побитового ИЛИ над соответствующими битовыми полями, которые содержат характеристические вектора.

U = {1, 2, 3, 4, 5} Его характеристический вектор {1, 1, 1, 1, 1}

A = {1, 2, 3} Его характеристический вектор {1, 1, 1, 0, 0}

B = {1, 3, 5} Его характеристический вектор {1, 0, 1, 0, 1}

Результат объединения множеств A˅B:

A˅ B = {1, 2, 3, 5} Его характеристический вектор {1, 1, 1, 0, 1}

Операция «Пересечение двух множестве». В результате этой операции получаем множество, в котором есть только те элементы, которые принадлежат и первому, и второму множеству: Реализуется как операция побитового И над соответствующими битовыми полями, которые содержат характеристические вектора.

U = {1, 2, 3, 4, 5} Его характеристический вектор {1, 1, 1, 1, 1}

A = {1, 2, 3} Его характеристический вектор {1, 1, 1, 0, 0}

B = {1, 3, 5} Его характеристический вектор {1, 0, 1, 0, 1}

Результат объединения множеств A/\B:

A ˄ B = {1, 3} Его характеристический вектор {1, 0, 1, 0, 0}

Операция «Дополнение к множеству». В результате этой операции мы получаем множество, в котором будут только те элементы, которых не было в заданном множестве, но есть в унивёрсе. Реализуется как операция отрицания для соответствующего битового поля, которое содержит характеристический массив.

U = {1, 2, 3, 4, 5} Его характеристический вектор {1, 1, 1, 1, 1}

A = {1, 3, 5} Его характеристический вектор {1, 0, 1, 0, 1}

Результат операции «Дополнение к множеству» A:

Ā = {2, 4} Его характеристический вектор {0, 1, 0, 1, 0}

### «Решето Эратосфена»

Входные данные: целое положительное число.

Выходные данные: все простые числа, не превышающие заданного вами целого положительного числа.

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

Для нахождения всех простых чисел не больше заданного числа n, следуя методу Эратосфена, нужно выполнить следующие шаги:

1. Выписать подряд все целые числа от двух до n (2, 3, 4, …, n).
2. Пусть переменная p изначально равна двум — первому простому числу.
3. Зачеркнуть в списке числа от 2p до n, считая шагами по p (это будут числа, кратные p: 2p, 3p, 4p, …).
4. Найти первое незачёркнутое число в списке, большее чем p, и присвоить значению переменной p это число.
5. Повторять шаги 3 и 4, пока возможно.

Теперь все незачёркнутые числа в списке — это все простые числа от 2 до n.

## Описание программной реализации

### Описание класса TBitField

typedef unsigned int TELEM;

class TBitField

{

private:

int BitLen;

TELEM \*pMem;

int MemLen;

const int bitsInElem = 32;

const int shiftSize = 5;

// методы реализации

int GetMemIndex(const int n) const noexcept;

TELEM GetMemMask (const int n) const noexcept;

public:

TBitField(int len);

TBitField(const TBitField &bf);

~TBitField();

// доступ к битам

int GetLength(void) const;

void SetBit(const int n);

void ClrBit(const int n);

int GetBit(const int n) const;

// битовые операции

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

bool 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& in, TBitField& bf);

friend ostream& operator<<(ostream& out, 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 – вспомогательное значение, количество битов в элементе памяти.

shiftSize – вспомогательное значение для битового целочисленного деления.

**Конструкторы**:

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: bf – объект класса TBitField.

~TBitField();

Назначение: очистка выделенной памяти.

**Методы**:

int GetMemIndex(const int n) const noexcept;

Назначение: получение индекса элемента в памяти.

Входные параметры: n – номер бита.

Выходные параметры: индекс элемента в памяти.

TELEM GetMemMask (const int n) const noexcept;

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: изменить значение бита на 1.

Входные параметры: n – номер бита.

void ClrBit(const int n);

Назначение: изменить значение бита на 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 – 1 или 0.

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

Назначение: перегрузка операции сравнивания, сравнение на равенство объектов.

Входные параметры: bf – объект класса TBitField.

Выходные параметры: true или false.

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

Назначение: перегрузка операции сравнивания, сравнение на неравенство объектов.

Входные параметры: bf – объект класса TBitField.

Выходные параметры: true или false.

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

Назначение: присваивание значений объекта bf.

Входные параметры: bf – объект класса TBitField.

Выходные параметры: ссылка на объект класса TBitField (себя).

TBitField operator|(const TBitField &bf);

Назначение: создание объекта с примененной побитовой операцией ИЛИ.

Входные параметры: bf – объект класса TBitField.

Выходные параметры: объект класса TBitField.

TBitField operator&(const TBitField &bf);

Назначение: создание объекта с примененной побитовой операцией И.

Входные параметры: bf – объект класса TBitField.

Выходные параметры: объект класса TBitField.

TBitField operator~(void);

Назначение: создание объекта с примененной побитовой операцией отрицания.

Выходные параметры: объект класса TBitField.

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

Назначение: ввод данных.

Входные параметры: in – поток ввода, bf – объект класса TBitField.

Выходные параметры: поток ввода.

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

Назначение: вывод данных.

Входные параметры: out – поток вывода, bf – 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();

// доступ к битам

int GetMaxPower(void) const noexcept;

void InsElem(const int Elem);

void DelElem(const int Elem);

bool IsMember(const int Elem) const;

// теоретико-множественные операции

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

bool 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& in, TSet& bf);

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

};

Назначение: представление множества.

**Поля**:

MaxPower – мощность множества.

BitField – характеристический массив.

**Конструкторы**:

TSet(int mp);

Назначение: инициализация битового поля.

Входные параметры: mp – количество элементов в универсуме.

TSet(const TSet& s);

Назначение: копирование данных из другого множества.

Входные параметры: s – объект класса TSet.

TSet(const TBitField& bf);

Назначение: формирование множества на основе битового поля.

Входные параметры: bf – объект класса TBitField.

operator TBitField();

Назначение: получение поля BitField.

Выходные параметры: объект класса TBitField.

**Методы**:

int GetMaxPower(void) const noexcept;

Назначение: получение максимальной мощности множества.

Выходные параметры: MaxPower – максимальная мощность множества.

void InsElem(const int Elem);

Назначение: добавить элемент в множество.

Входные параметры: Elem – индекс элемента.

void DelElem(const int Elem);

Назначение: удаление элемента из множества.

Входные параметры: Elem – индекс элемента.

bool IsMember(const int Elem) const;

Назначение: проверка, состоит ли элемент в множестве.

Входные параметры: true или false.

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

Назначение: перегрузка операции сравнивания, сравнение на равенство объектов.

Входные параметры: s – объект класса TSet.

Выходные параметры: true или false.

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

Назначение: перегрузка операции сравнивания, сравнение на неравенство объектов.

Входные параметры: s – объект класса TSet.

Выходные параметры: true или false.

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

Назначение: присваивание значений объекта s.

Входные параметры: s – объект класса TSet.

Выходные параметры: ссылка на объект класса TSet (себя).

TSet operator+ (const int Elem);

Назначение: добавление элемента в множество.

Входные параметры: Elem – индекс элемента.

Выходные параметры: объект класса TSet.

TSet operator- (const int Elem);

Назначение: удаление элемента из множества.

Входные параметры: Elem – индекс элемента.

Выходные параметры: объект класса TSet.

TSet operator+ (const TSet& s);

Назначение: объединение двух множеств.

Входные параметры: s – объект класса TSet.

Выходные параметры: объект класса TSet.

TSet operator\* (const TSet& s);

Назначение: пересечение двух множеств.

Входные параметры: s – объект класса TSet.

Выходные параметры: объект класса TSet.

TSet operator~ (void);

Назначение: получение дополнения к множеству.

Выходные параметры: объект класса TSet.

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

Назначение:

Входные параметры: s – объект класса TSet, in – поток ввода.

Выходные параметры: поток ввода.

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

Назначение:

Входные параметры: s – объект класса TSet, out – поток вывода.

Выходные параметры: поток вывода.

# Заключение

В ходе выполнения работы "Битовые поля и множества" были изучены и практически применены концепции битовых полей и множеств.

Были достигнуты следующие результаты:

1. Изучены теоретические основы битовых полей и множеств.
2. Разработана программа, реализующая операции над битовыми полями и множествами. В ходе экспериментов была оценена эффективность работы этих операций и сравнена с другими подходами. Результаты показали, что использование битовых полей и множеств позволяет существенно сократить объем памяти и ускорить операции над множествами.
3. Проанализированы полученные результаты и сделаны выводы о преимуществах и ограничениях использования битовых полей и множеств. Оказалось, что эти структуры данных особенно полезны при работе с большими объемами данных, где компактность представления и эффективность операций являются ключевыми факторами.

# Литература

1. Лекция «Множества и битовые поля» Сысоева А.В. [https://cloud.unn.ru/s/DLRHnt54ircG2WL].
2. Битовые поля [https://www.youtube.com/watch?v=\_XJAeR7obBk].
3. Решето Эратосфена [https://yandex.ru/video/preview/2908856360907561981].

# Приложения

## Приложение А. Реализация класса TBitField

#include "tbitfield.h"

TBitField::TBitField(int len)

{

if (len > 0) {

BitLen = len;

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

pMem = new TELEM[MemLen];

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

}

else if(len == 0) {

BitLen = 0;

MemLen = 0;

pMem = nullptr;

}

else {

throw "error: BitFiels size < 0";

}

}

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

{

BitLen = bf.BitLen;

MemLen = bf.MemLen;

if (MemLen) {

pMem = new TELEM[MemLen];

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

}

else {

pMem = nullptr;

}

}

TBitField::~TBitField()

{

if (MemLen > 0)

delete[] pMem;

}

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

{

return n >> shiftSize;

}

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

{

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

}

// 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 || n < 0) throw "error: index out of range";

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

}

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

{

if (n >= BitLen || n < 0) throw "error: index out of range";

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

}

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

{

if (n >= BitLen || n < 0) throw "error: index out of range";

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

}

// битовые операции

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

{

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);

}

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

{

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

bool flag = true;

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

if (pMem[i] != bf.pMem[i]) {

flag = false;

break;

}

return flag;

}

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

{

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

}

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

{

int len = (BitLen > bf.BitLen) ? BitLen : bf.BitLen;

TBitField res(len);

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

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

}

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

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

}

return res;

}

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

{

int len = (BitLen > bf.BitLen) ? BitLen : bf.BitLen;

int mlen = (BitLen < bf.BitLen) ? BitLen : bf.BitLen;

TBitField res(len);

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

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

}

return res;

}

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

{

TBitField res(BitLen);

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

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

return res;

}

// ввод/вывод

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

{

string ans;

in >> ans;

int blen = bf.BitLen;

int len = (ans.size() < blen) ? ans.size() : blen;

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

bf.ClrBit(i);

}

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

if (ans[i] == '1') bf.SetBit(i);

}

return in;

}

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

{

int len = bf.BitLen;

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

if (bf.GetBit(i))

out << '1';

else

out << '0';

}

return out;

}

## Приложение Б. Реализация класса TSet

#include "tset.h"

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

{

if (mp >= 0) {

MaxPower = mp;

}

else {

throw "error: Set size < 0";

}

}

// 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;

}

// доступ к битам

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

{

return MaxPower;

}

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

{

if (Elem >= MaxPower || Elem < 0) throw "error: index out of range";

if (BitField.GetBit(Elem))

return true;

return false;

}

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

{

if (Elem >= MaxPower || Elem < 0) throw "error: index out of range";

BitField.SetBit(Elem);

}

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

{

if (Elem >= MaxPower || Elem < 0) throw "error: index out of range";

BitField.ClrBit(Elem);

}

// теоретико-множественные операции

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

{

MaxPower = s.MaxPower;

BitField = s.BitField;

return \*this;

}

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

{

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

return (BitField == s.BitField);

}

bool TSet::operator!=(const TSet& s) const // сравнение

{

return !(\*this == s);

}

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

{

int len = (MaxPower > s.MaxPower) ? MaxPower : s.MaxPower;

TSet res(len);

res.BitField = BitField | s.BitField;

return res;

}

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

{

if (Elem >= MaxPower || Elem < 0) throw "error: index out of range";

TSet res(\*this);

res.InsElem(Elem);

return res;

}

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

{

if (Elem >= MaxPower || Elem < 0) throw "error: index out of range";

TSet res(\*this);

res.DelElem(Elem);

return res;

}

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

{

int len = (MaxPower > s.MaxPower) ? MaxPower : s.MaxPower;

TSet res(len);

res.BitField = BitField & s.BitField;

return res;

}

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

{

TSet res(MaxPower);

res.BitField = ~BitField;

return res;

}

// перегрузка ввода/вывода

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

{

in >> s.BitField;

return in;

}

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

{

out << s.BitField;

return out;

}

## Приложение В. Sample\_primenumbers

#include <iostream>

#include "tset.h"

int main()

{

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

int n;

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

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);

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

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

if (s.IsMember(i))

cout << i << ' ';

cout << endl;

return 0;

}