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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Шкребко М.С. /

Подпись

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

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

Подпись

Нижний Новгород  
2023

**Содержание**

[Введение 3](#_Toc148511688)

[1 Постановка задачи 4](#_Toc148511689)

[2 Руководство пользователя 5](#_Toc148511690)

[2.1 Приложение для демонстрации работы битовых полей 5](#_Toc148511691)

[2.2 Приложение для демонстрации работы множеств 5](#_Toc148511692)

[2.3 Приложение «решето Эратосфена» 7](#_Toc148511693)

[3 Руководство программиста 8](#_Toc148511694)

[3.1 Использованнные алгоритмы 8](#_Toc148511695)

[3.1.1 Битовые поля 8](#_Toc148511696)

[3.1.2 Множества 8](#_Toc148511697)

[3.1.3 Алгоритм «решето Эратосфена» 8](#_Toc148511698)

[3.2 Описание классов 8](#_Toc148511699)

[3.2.1 Класс TbitField 8](#_Toc148511700)

[3.2.2 Класс TSet 12](#_Toc148511701)

[Заключение 16](#_Toc148511702)

[Литературы 17](#_Toc148511703)

[Приложения 18](#_Toc148511704)

[Приложение А. Реализация класса TBitField 18](#_Toc148511705)

[Приложение Б. Реализация класса TSet 20](#_Toc148511706)

# Введение

Множество – это математическая модель, которая состоит из объектов, которые называются элементами множества, обладающих определёнными признаками, которые относятся к каждому элементу. Теория множеств – это математическая дисциплина, которая изучает свойства, отношения и операции над множествами. Теория множеств применяется во многих областях. К примеру, она используется в физике для моделирования и изучения физических систем и процессов, используется для описания и анализа свойств и взаимодействий различных объектов и частиц. В программировании теория множеств является основой для многих алгоритмических и структурных концепций в компьютерных науках. Она используется для моделирования и решения различных задач, таких как поиск, сортировка, анализ данных и многих других областях. Так же используется в математике, логике, социологии, теории вероятности, экономике и во многих других областях. Таким образом множество – это удобный математический инструмент для моделирования различных процессов, а также для решения множества алгоритмических и математических задач.

Эффективный способ хранения множества в виде структуры данных является битовое поле. Битовые поля актуальны в настоящее время, так как они широко используются в различных областях, связанных с обработкой и хранением данных. Они позволяют компактно представлять и манипулировать большими объемами информации, что особенно важно в сферах, где требуется эффективное использование памяти или передача данных. Битовые поля имеют весомую часть в обработке данных и решении различных задач. Они позволяют компактно представлять флаги, состояния или другие булевы значения, что упрощает программирование и повышает производительность. Битовые поля также используются для работы с масками, фильтрами и другими алгоритмами, что способствует оптимизации процессов и улучшению качества решений. Битовые поля применимы во многих областях, включая программирование, базы данных, сетевые протоколы, компьютерные игры, криптографию и другие. Они могут быть использованы для оптимизации работы с памятью, управления ресурсами, фильтрации данных, проверки состояний и многих других задач. Битовые поля также могут быть эффективно использованы для создания компактных структур данных и алгоритмов, что важно в сферах с ограниченными ресурсами или высокими требованиями к производительности.

# Постановка задачи

**Цель** – изучить битовые поля и множества.

**Задачи**:

1. Изучить основные понятия и принципы работы с битовыми полями и множествами.
2. Разработать программу, которая будет реализовывать операции над битовыми полями и множествами.
3. Протестировать корректность выполнения программы на различных примерах.
4. Протестировать программу с помощью алгоритма «решето Эратосфена».
5. Сделать вывод.

# Руководство пользователя

## Приложение для демонстрации работы битовых полей

1. Запустить sample\_bitfield.exe. В результате появится следующее окно ().

Изображение выглядит как снимок экрана, текст, программное обеспечение, компьютер

Автоматически созданное описание

1. Основное окно приложения битовых полей
2. Дальше необходимо ввести битовое поле длины 6 (рис. 2):

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

1. Ввод битового поля
2. Одно поле создаётся само и выводится результат работы программы ():

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

1. Результат работы программы

## Приложение для демонстрации работы множеств

1. Запустить sample\_tset.exe. В результате появится окно, показанное ниже   
   (рис. 4).

Изображение выглядит как снимок экрана, программное обеспечение, Мультимедийное программное обеспечение, текст

Автоматически созданное описание

1. Основное окно работы множеств
2. Дальше вводим длину и сами множества ().

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

1. Ввод множества
2. Вывод работы программы ():

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

1. Результат работы программы

## Приложение «решето Эратосфена»

* 1. Запустите sample\_primenumbers.exe. В результате появится следующее окно ():

Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Автоматически созданное описание

1. Основное окно приложения
2. Далее необходимо ввести целое положительное число для того, чтобы получить все простые числа до введенного ()

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

1. Результат работы приложения решето Эратосфена

# Руководство программиста

## Использованные алгоритмы

### Битовые поля

Битовые поля – это структура данных, которая представляет из себя хранилище 0 и 1 в памяти.

Существуют следующие операции для работы с битовыми полями: доступ к конкретному биту, побитовые операции (побитовые конъюнкция, дизъюнкция между двумя битовыми полями, и побитовое отрицание битового поля). Также два битовых поля можно сравнить.

* **Операция побитовое «ИЛИ»**

После применения операции к битовым полям получаем одно битовое поле, в котором, если хотя бы в одном поле есть 1, то после применения операции в результате будет 1, иначе 0.

|  |  |  |  |
| --- | --- | --- | --- |
| А | 1 | 0 | 0 |
| В | 0 | 1 | 0 |
| AUB | 1 | 1 | 0 |

* **Операция побитовое «И»**

После применения операции к битовым полям получаем одно битовое поле, в котором, если хотя бы в одном поле есть 0, то после применения операции в результате будет 0, иначе 1.

|  |  |  |  |
| --- | --- | --- | --- |
| А | 1 | 0 | 0 |
| В | 1 | 1 | 0 |
| A∩B | 1 | 0 | 0 |

* **Операция побитовое «НЕ»**

После применения операции к полю получаем инвертированное поле.

|  |  |  |  |
| --- | --- | --- | --- |
| А | 1 | 0 | 0 |
| ¬A | 0 | 1 | 1 |

**Установление значения отдельного бита в 1**

Для реализации этой операции нам понадобится битовое поле, в котором все биты равны 0, и один бит равен 1. Этот бит должен стоять на разряде, который в исходном битовом поле необходимо установить в 1, обозначим за i. Такой битовой маски можно добиться сдвигом целочисленной 1 на i бит влево.

Вспомогательное поле будем называть битовой маской. Если к исходному битовому полю и битовой маске применить операцию битового «ИЛИ», то в результате получим битовое поле, в котором бит на i-ом разряде установлен в 1, а все оставшиеся биты равны битам в исходном битовом поле.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Изначальное битовое поле |  | Битовая маска |  | Результат |
| 0 | **|** | 0 | **=** | 0 |
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 1 | 0 | 1 |
| **0** | **1** | **1** |
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 0 | 1 |
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

**Установление значения отдельного бита в 0**

Битовая маска для этой операции имеет значение i-ого разряда 0, а значения всех остальных разрядов равны 1. Такую маску можно получить инвертированием маски, которая использовалась при установлении определенного бита в 1.

При применении к исходному битовому полю и инвертированной маске операции битового «И» в результате получим битовое поле, в котором бит на i-ом разряде равен 0, а все оставшиеся биты равны битам исходного битового поля.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Изначальное битовое поле |  | Инвертированная  битовая маска |  | Результат |
| 0 | **&** | 1 | **=** | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |
| **1** | **0** | **0** |
| 0 | 1 | 0 |
| 1 | 1 | 1 |
| 0 | 1 | 0 |
| 0 | 1 | 0 |
| 0 | 1 | 0 |

Получение текущего значения определённого бита:

Применим к исходному полю и битовой маске, которая использовалась для операции установления бита в 1, операцию битового «И». В результате мы получим битовое поле, в котором бит на i-ом разряде сохранил такое же значение, какое было на i-ом разряде, а все остальные биты равны 0. Применим сдвиг вправо на i бит и получим значение бита на i-ом разряде в исходном битовом поле.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Изначальное битовое поле |  | Битовая маска |  | Промежуточный результат |  | Результат |
| 0 | **&** | 0 | **=** | 0 | **>>** | 0 |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| **1** | **1** | **1** | 0 |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | **1** |

### Множества

Множество – это структура данных, которое основано на битовом поле. Если бит в i-ом разряде битового поля равен 1, то i-ый элемент входит в текущее множество, если равен 0, то не входит во множество.

Для множеств реализованы операции добавления и удаления элемента из множества; операция проверки наличия элемента во множестве; операции: объединения, пересечения множеств; взятие дополнения к множеству.

Числа, которые находятся в множестве, показывают, какие именно элементы на i-ой позиции входят в множество.

Примеры множественных операций:

A = {1, 3, 7, 10, 13} – в универсальном множестве из 14 элементов.

Это множество характеризуется битовым полем:

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

B = {0, 3, 6, 7, 11, 13} – в универсальном множестве из 14 элементов.

Это множество характеризуется битовым полем:

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

A U B = {0, 1, 3, 6, 7, 10, 11, 13}

Множество, являющееся результатом объединения двух других, получается в результате применения к ним операции побитового «ИЛИ».

Это множество характеризуется битовым полем:

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

A ∩ B = {3, 7, 13}

Множество, являющееся результатом пересечения двух других, получается в результате применения к ним операции побитового «И».

Это множество характеризуется битовым полем:

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

¬A = {0, 2, 4, 5, 6, 8, 9, 11, 12}

Множество, являющееся дополнением к исходному, получается в результате применения к исходному операции побитового отрицания.

Это множество характеризуется битовым полем:

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

### Алгоритм «решето Эратосфена»

Решето Эратосфена — алгоритм нахождения всех простых чисел до некоторого целого числа n. Работа заключается в фильтрации всех чисел за исключением простых. По мере прохождения списка нужные числа остаются, а ненужные удаляются.

Входные данные: натуральное число n – число, до которого включительно пользователь хочет вывести простые числа.

Выходные данные: все простые числа в диапазоне от 1 до n включительно.

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 классов

### Класс TBitField

Объявление класса:

class TBitField

{

private:

int BitLen;

TELEM \*pMem;

int MemLen;

int Bits = (sizeof(TELEM) \* 8);

int GetMemIndex(const int n) const;

TELEM GetMemMask (const int n) const;

public:

TBitField(int len);

TBitField(const TBitField &bf);

~TBitField();

// доступ к битам

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

void SetBit(const int n); // установить бит

void ClrBit(const int n); // очистить бит

int GetBit(const int n) const; // получить значение бита

// битовые операции

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

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

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

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

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

TBitField operator~(void); // отрицание

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

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

};

**Поля:**

BitLen – длина битового поля.

pMem – память для представления битового поля.

MemLen – количество элементов для представления битового поля.

**Конструкторы:**

• TBitField(int len)

Входные параметры: len – длина битового поля.

Выходные параметры: отсутствуют.

• TBitField(const TBitField &bf)

Создание экземпляра класса на основе другого экземпляра.

Входные параметры: bf – ссылка, адрес экземпляра класса, на основе которого будет создан другой.

Выходные параметры: отсутствуют.

**Деструктор:**

~TBitField()

Очистка выделенной памяти.

Входные и выходные параметры отсутствуют.

**Методы:**

int GetMemIndex(const int n) const;

Назначение: получение индекса элемента, где хранится бит.

Входные данные:

n – номер бита.

Выходные данные:

Индекс элемента, где хранится бит с номером n.

TELEM GetMemMask(const n) const;

Назначение: получение битовой маски.

Входные данные:

n – номер бита.

Выходные данные:

Элемент под номером n.

• int GetLength(void) const;

Назначение: получение длинны битового поля.

Входные данные: отсутствуют.

Выходные данные: длина битового поля.

• void SetBit(const int n)

Назначение: установить бит равным 1.

Входные данные: n - номер бита, который нужно установить.

Выходные данные: отсутствуют.

•void ClrBit(const int n);

Назначение: отчистить бит.

Входные данные: n - номер бита, который нужно отчистить.

Выходные данные: отсутствуют.

• int GetBit(const int n) const;

Назначение: получить бить.

Входные данные: n - номер бита.

Выходные данные: бит 1 или 0, в зависимости установлен он, или нет.

**Операторы:**

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

Назначение: оператор сравнения.

Входные данные: bf – битовое поле, с которым мы сравниваем.

Выходные данные:1 или 0, в зависимости равны они или нет.

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

Назначение: оператор сравнения.

Входные параметры:

bf – битовое поле, с которым мы сравниваем.

Выходные параметры: 1 или 0, в зависимости равны они, или нет

Соответственно.

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

Назначение: оператор присваивания.

Входные параметры:

bf – битовое поле, которое мы присваиваем.

Выходные параметры: ссылка на экземпляр класса TBitField.

• TBitField operator|(const TBitField &bf);

Назначение: оператор побитового «ИЛИ».

Входные параметры:

bf – битовое поле.

Выходные параметры: экземпляр класса, который равен { \*this | bf }.

• TBitField operator&(const TBitField &bf);

Назначение: оператор побитового «И».

Входные параметры:

bf – битовое поле, с которым мы сравниваем.

Выходные параметры: экземпляр класса, который равен { \*this & bf }.

• TBitField operator~(void);

Назначение: оператор инверсии.

Входные параметры отсутствуют.

Выходные параметры: экземпляр класса, каждый элемент которого

равен{~\*this}.

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

Назначение: оператор ввода из консоли

Входные параметры:

istr – буфер консоли

bf – класс, который нужно ввести из консоли.

Выходные параметры: ссылка на буфер istr.

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

Назначение: оператор вывода из консоли

Входные параметры:

ostr – буфер консоли.

bf – класс, который нужно вывести в консоль.

Выходные параметры: ссылка на буфер istr.

### Класс TSet

Объявление класса:

class TSet

{

private:

int MaxPower; // максимальная мощность множества

TBitField BitField; // битовое поле для хранения хар-го вектора

public:

TSet(int mp);

TSet(const TSet &s); // конструктор копирования

TSet(const TBitField &bf); // конструктор преобразования типа

operator TBitField(); // преобразование типа к битовому полю

// доступ к битам

int GetMaxPower(void) const; // максимальная мощность множества

void InsElem(const int n); // включить элемент в множество

void DelElem(const int n); // удалить элемент из множества

int IsMember(const int n) const; // проверить наличие элемента в

// множестве

// теоретико-множественные операции

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

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

TSet operator+ (const int n); // включение элемента в множество

TSet operator- (const int n); // удаление элемента из множества

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

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

TSet operator~ (void); // дополнение

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

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

**Поля**:

MaxPower – максимальный элемент множества.

BitField – экземпляр битового поля, на котором реализуется множество.

**Конструкторы:**

• TSet(int mp)

Входные параметры: mp – максимальный элемент множества.

Выходные параметры: отсутствуют.

• TSet(const TSet &s)

Входные параметры: s – ссылка, адрес экземпляра класса, на основе которого будет создан другой.

Выходные параметры: отсутствуют.

**Деструктор:**

~TSet().

Очистка выделенной памяти.

Входные и выходные параметры отсутствуют.

**Методы:**

• int GetMaxPower(void) const;

Назначение: получение максимального элемента множества.

Входные параметры: отсутствуют.

Выходные параметры: максимальный элемент множества.

• void InsElem(const int Elem)

Назначение: добавить элемент в множество.

Входные параметры: Elem - добавляемый элемент.

Выходные параметры отсутствуют

• void DelElem(const int Elem)

Назначение: удалить элемент из множества.

Входные параметры: Elem - удаляемый элемент.

Выходные параметры отсутствуют.

• int IsMember(const int Elem) const;

Назначение: узнать, есть ли элемент в множестве.

Входные параметры: Elem - элемент, который нужно проверить на наличие.

Выходные параметры: 1 или 0, в зависимости есть элемент в множестве или нет**Операторы:**

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

Назначение: оператор сравнения.

Входные параметры: s – битовое поле, с которым мы сравниваем поле.

Выходные параметры: 1 или 0, в зависимости равны они, или нет.

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

Назначение: оператор сравнения.

Входные параметры: s – битовое поле, с которым мы сравниваем.

Выходные параметры: 1 или 0, в зависимости равны они или нет.

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

Назначение: оператор присваивания.

Входные параметры: s – множество, которое мы присваиваем.

Выходные параметры: ссылка на экземпляр класса TSet.

• TSet operator+(const TSet &bf);

Назначение: оператор объединения множеств.

Входные параметры: s – множество.

Выходные параметры: экземпляр класса, который равен { \*this | s }.

• TSet operator\*(const TSet &bf);

Назначение: оператор пересечения множеств.

Входные параметры: s – множество.

Выходные параметры: экземпляр класса, который равен { \*this & s }.

• TBitField operator~(void);

Назначение: оператор дополнение до Универса.

Входные параметры: отсутствуют.

Выходные параметры: экземпляр класса, каждый элемент которого равен {~\*this}.

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

Назначение: оператор ввода из консоли.

Входные параметры:

istr – буфер консоли

s – класс, который нужно ввести из консоли

Выходные параметры: ссылка на буфер (поток) istr.

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

Назначение: оператор вывода из консоли.

Входные параметры:

ostr – буфер консоли.

s – класс, который нужно вывести в консоль.

Выходные параметры: Ссылка на буфер (поток) ostr.

• operator TBitField();

Назначение: вывод поля BitField.

Входные параметры отсутствуют.

Выходные данные: поле BitField.

• TSet operator+(const int Elem);

Назначение: оператор объединения множества и элемента. Данный оператор

аналогичен метод добавления элемента в множество.

Входные параметры: Elem – число.

Выходные параметры: исходный экземпляр класса, содержащий Elem.

• TSet operator+ (const int Elem);

Назначение: оператор объединения множества и элемента.

Входные параметры: Elem – число.

Выходные параметры: исходный экземпляр класса, не содержащий Elem.

# Заключение

В результате данной лабораторной работы были изучены основы битовых полей и множеств, а также принципы их использования в программировании. Была разработана программа, которая реализует операции над битовыми полями и множествами. Были проведены эксперименты с различными наборами данных, чтобы проверить работу программы и изучить скорость работы. Проведенный анализ результатов показал, что использование битовых полей и множеств может быть очень полезным в решении определенных задач.

# Литература

1. Битовые поля в C++ [<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>

#define Shiftsize 5

TBitField::TBitField(int len)

{

if (len < 0)

throw "out of range";

BitLen = len;

MemLen = BitLen / Bits + 1;

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 "out of range";

return n >> Shiftsize;

}

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

{

return 1 << (n & (Bits - 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 "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))

return 1;

else return 0;

}

// битовые операции

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

{

if (\*this == bf)

return \*this;

BitLen = bf.BitLen;

if (MemLen != bf.MemLen) {

MemLen = bf.MemLen;

delete[] pMem;

pMem = new TELEM[MemLen];

}

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.BitLen)

return 0;

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

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

return 0;

for (int i = (MemLen - 1) \* Bits; 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 len = max(BitLen, bf.BitLen);

TBitField result(len);

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

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

return result;

}

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

{

int len = max(BitLen, bf.BitLen);

TBitField result(len);

for (int i = 0; i < result.MemLen; i++) {

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

}

return result;

}

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

{

TBitField result(BitLen);

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

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

return result;

}

// ввод/вывод

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

{

int bit;

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

{

istr >> bit;

if (bit == 0)

{

bf.ClrBit(i);

}

else if (bit == 1)

{

bf.SetBit(i);

}

else

{

throw ("Error");

}

}

return istr;

}

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

{

ostr << "{ ";

for (int i = bf.BitLen - 1; i >= 0; i--)

{

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

}

ostr << "}";

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) : MaxPower(bf.GetLength()), BitField(bf)

{}

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?

{

return BitField.GetBit(Elem);

}

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

{

return BitField.SetBit(Elem);

}

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

{

BitField.ClrBit(Elem);

}

// теоретико-множественные операции

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

{

MaxPower = s.MaxPower;

BitField = s.BitField;

return \*this;

}

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

{

if (MaxPower == s.GetMaxPower() && BitField == s.BitField)

{

return 1;

}

else return 0;

}

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

{

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

}

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

{

return TSet(BitField | (s.BitField));

}

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

{

TSet tmp(\*this);

tmp.InsElem(Elem);

return tmp;

}

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

{

TSet tmp(\*this);

tmp.DelElem(Elem);

return tmp;

}

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

{

return TSet(BitField & (s.BitField));

}

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

{

return TSet(~BitField);

}

// перегрузка ввода/вывода

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

{

int InputElem;

while (true)

{

istr >> InputElem;

if (InputElem == -1)

{

break;

}

if (InputElem >= 0 && InputElem < s.MaxPower)

s.InsElem(InputElem);

else

throw "out of range";

}

return istr;

}

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

{

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

{

if (s.IsMember(i))

{

ostr << i << " ";

}

}

return ostr;

}

## Приложение В. Пример использования битовых полей

#include <iostream>

#include "tbitfield.h"

const int n = 6;

void bitfield()

{

TBitField bf1(n);

cout << "Enter bf" << endl;

cin >> bf1;

cout << "1:" << bf1 << endl;

TBitField bf2(n);

bf2.SetBit(0);

bf2.SetBit(1);

bf2.SetBit(2);

bf2.SetBit(5);

cout << "2: " << bf2 << endl;

bf2.ClrBit(1);

cout << "2:" << bf2 << endl;

cout << "1&2:" << (bf1 & bf2) << endl;

cout << "1|2:" << (bf1 | bf2) << endl;

cout << "1==2:" << (bf1 == bf2) << endl;

cout << "~1:" << (~bf1) << endl;

}

int main()

{

try {

bitfield();

}

catch ( const char\* exception) {

cout << exception << endl;

}

}

## Приложение Г. Пример использования множеств

#include <iostream>

#include "tset.h"

void set() {

int n;

cout << "Enter len" << endl;

cin >> n;

TSet s1(n);

TSet s2(n);

cout << "Enter s1" << endl;

cin >> s1;

cout << "1:" << s1 << endl;

cout << "Enter s2" << endl;

cin >> s2;

cout << "2:" << s2 << endl;

cout << "1==2:" << (s1 == s2) << "\n" << endl;

cout << "+:" << (s1 + s2) << endl;

cout << "+(Element):" << (s2 + 2) << endl;

cout << "-(Element):" << (s1 - 4) << endl;

cout << "\*:" << (s1 \* s2) << endl;

cout << "~1:" << ~s1 << endl;

}

int main()

{

try {

set();

}

catch (const char\* exception) {

cout << exception << endl;

}

}

## Приложение Д. Алгоритм решета Эратосфена

#include <iostream>

#include "tbitfield.h"

using namespace std;

int main()

{

int n, m, k, count;

setlocale(LC\_ALL, "Russian");

cout << "Решето Эратосфена" << endl;

cout << "Использование класса \"Битовое поле\"" << endl;

cout << "Введите максимальное целое число: ";

cin >> n;

TBitField s(n + 1);

// заполнение множества

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

s.SetBit(m);

// проверка до sqrt(n) и удаление кратных

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

// если m в s, удаление кратных

if (s.GetBit(m))

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

s.ClrBit(k);

//оставшиеся в s элементы - простые числа

cout << endl << "Печать множества некратных чисел" << endl << s << endl;

cout << endl << "Печать простых чисел" << endl;

count = 0;

k = 1;

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

if (s.GetBit(m)) {

count++;

cout << m << " ";

if (k++ % 10 == 0)

cout << endl;

}

cout << endl;

/\*cout << "В первых " << n << " числах " << count << " простых" << endl;\*/

return 0;

}