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 \_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Фролова Е.А./

Подпись

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

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

Подпись

Нижний Новгород  
2023

**Содержание**

[Введение 3](#_Toc149689481)

[1 Постановка задачи 4](#_Toc149689482)

[2 Руководство пользователя 5](#_Toc149689483)

[2.1 Приложение для демонстрации работы битовых полей 5](#_Toc149689484)

[2.2 Приложение для демонстрации работы множеств 7](#_Toc149689485)

[2.3 «Решето Эратосфено» 11](#_Toc149689486)

[3 Руководство программиста 13](#_Toc149689487)

[3.1 Описание алгоритмов 13](#_Toc149689488)

[3.1.1 Битовые поля 13](#_Toc149689489)

[3.1.2 Множества 16](#_Toc149689490)

[3.1.3 «Решето Эратосфена» 18](#_Toc149689491)

[3.2 Описание программной реализации 18](#_Toc149689492)

[3.2.1 Описание класса TBitField 18](#_Toc149689493)

[3.2.2 Описание класса TSet 21](#_Toc149689494)

[Заключение 24](#_Toc149689495)

[Литература 25](#_Toc149689496)

[Приложения 26](#_Toc149689497)

[Приложение А. Реализация класса TBitField 26](#_Toc149689498)

[Приложение Б. Реализация класса TSet 29](#_Toc149689499)

[Приложение С. Реализация Sample\_primenumbers 31](#_Toc149689500)

# Введение

Влияние теории множеств на развитие современной математики очень велико. Прежде всего, теория множеств явилась фундаментом ряда новых математических дисциплин (теории функций действительного переменного, общей топологии, общей алгебры, функционального анализа и др.). Постепенно теоретико-множественные методы находят всё большее применение и в классических частях математики. Например, в области математического анализа они широко применяются в качественной теории дифференциальных уравнений, вариационном исчислении, теории вероятностей и др.

Активное применение аппарата теории множеств в современной науке приводит к необходимости создания соответствующих программных решений. Вместе с тем лишь в отдельных языках программирования предусмотрены встроенные средства для работы с множествами (примером может служить язык Pascal в реализации фирмы Borland).

# Постановка задачи

Цель – изучение одного из возможных подходов к хранению и обработке множеств.

Задачи: создание программных средств, поддерживающих эффективное хранение множеств и реализующих следующие операции.

* Включение элемента в множество;
* исключение элемента из множества;
* проверка наличия элемента в множестве;
* сложение множеств;
* пересечение множеств;
* разность множеств;
* копирование множества;
* вычисление мощности множества. Программные средства должны содержать:
* класс Множество;
* тестовое приложение, демонстрирующее использование основных операций с множествами.

# Руководство пользователя

## Приложение для демонстрации работы битовых полей

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

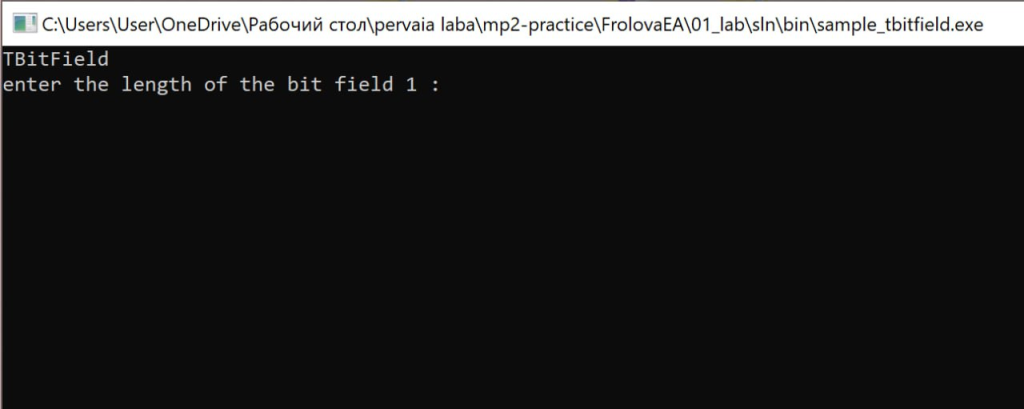


Рис. 1. Основное окно программы.

1. Введем длину битового поля. Теперь требуется ввести его значения (Рис. 1).

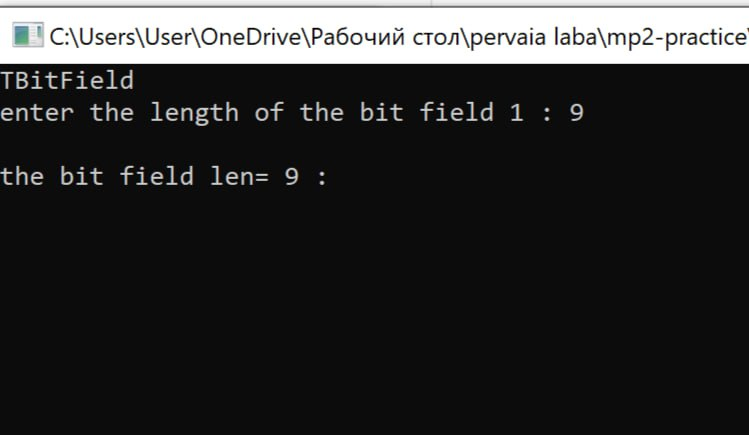


Рис. 2. Ввод первого битового поля.

1. Введем первое битовое поле (Рис. 2).

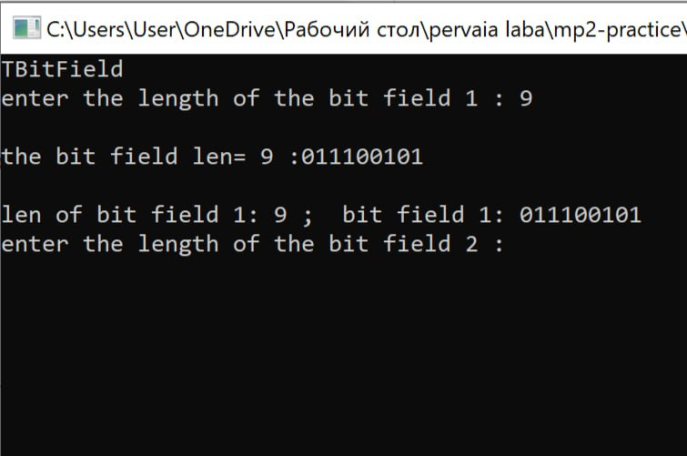


Рис. 3. Результат ввода первого битового поля.

1. Проведём такую же операцию для второго битового поля (Рис. 3).

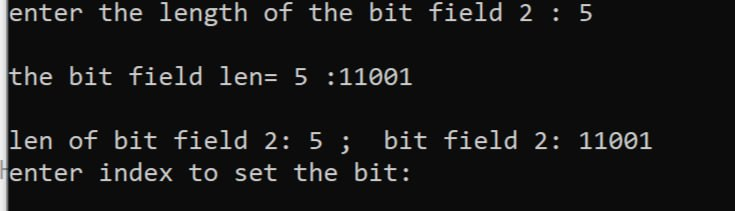


Рис. 4. Результат второго введенного битового поля.

1. Введем индекс по которому хотим поставить бит (Рис. 4).

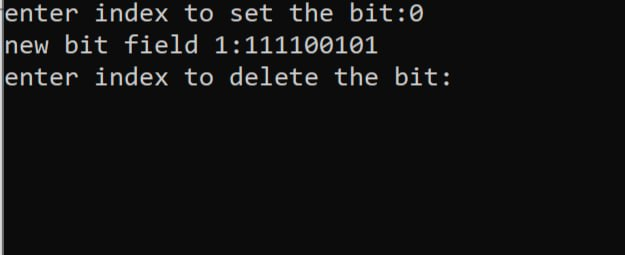


Рис. 5. Результат добавления бита в первое битовое поле.

1. Введем индекс по которому хотим удалить бит (Рис. 5).

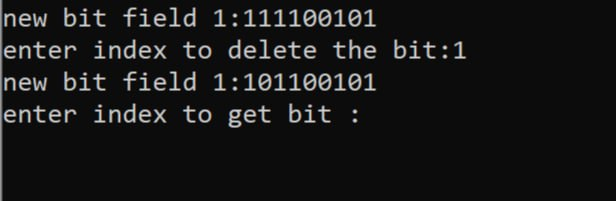


Рис. 6. Результат удаления бита.

1. Введём индекс по которому хотим получить значение бита (Рис. 6).

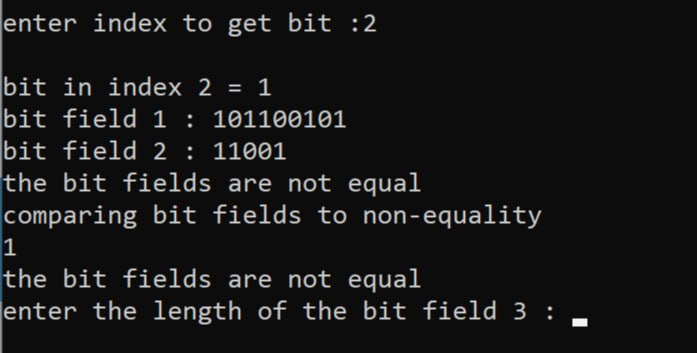


Рис. 7. Значение бита по адресу.

1. Помимо значения бита по адресу мы получили результат сравнения на равенство и неравенство двух полученных битовых полей. Теперь введём третье битовое поле как делали раньше, указав его длину а потом введя элементы (Рис. 7).

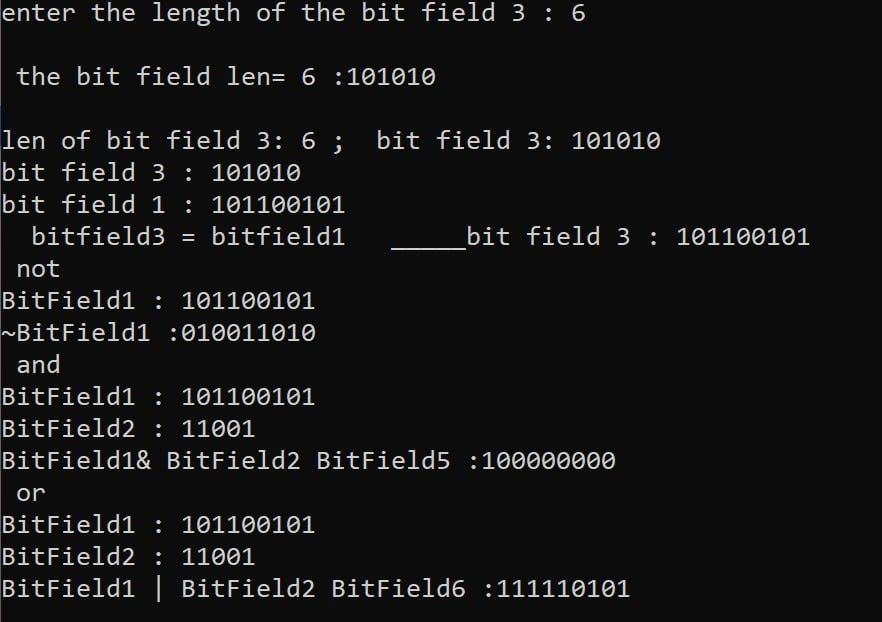


Рис. 8. Основное окно программы.

1. Программа вывела нам ранее введенное битовое поле.
2. Следующим этапом программа присвоила третьему битовому полю значение первого битового поля и вывела его.
3. Так же мы получили результат операции отрицания по отношению к первому битовому полю, результат логической операции «и» для первого и второго битовых полей и результат операции логического «или» для первого и второго битовых полей.
4. Исполнение программы окончено.

## Приложение для демонстрации работы множеств

1. Запустите приложение с названием sample\_tset.exe. В результате появится окно, показанное ниже (Рис. 9).



Рис. 9. Основное окно программы.

1. Требуется ввести максимальную мощность множества (вводим).
2. Далее программа требует ввести количество элементов которые мы хотим поместить в первое множество. Вводим их количества и сами эти элементы (Рис. 10).

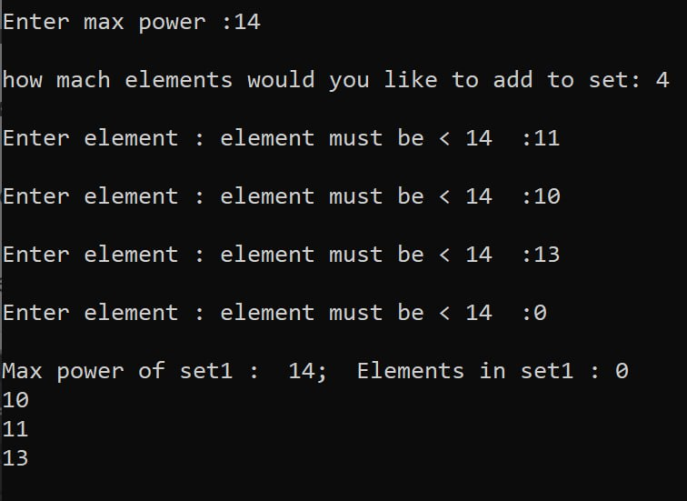


Рис. 10. Основное окно программы.

1. Повторим операцию для второго множества (Рис. 11).

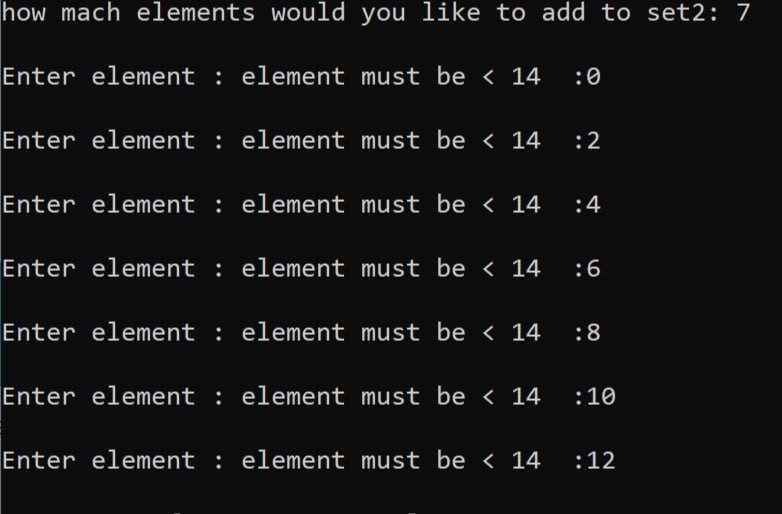


Рис. 11. Основное окно программы.

1. Получаем значение двух множеств (Рис. 12).

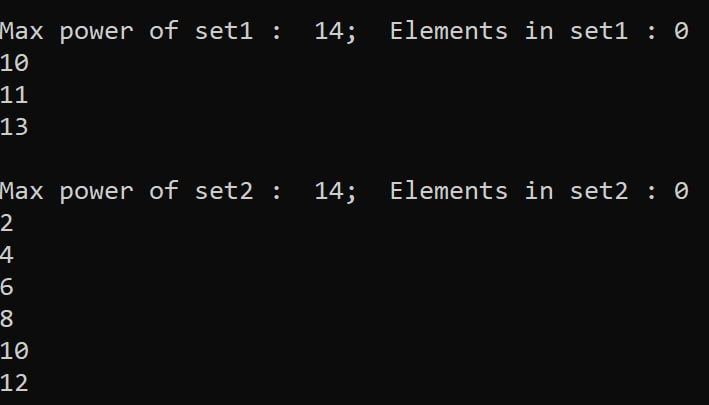


Рис. 12. Значения множеств.

1. Проверяем является ли введённый нам элемент элементом созданных нами множеств. Получаем результат (Рис. 13).

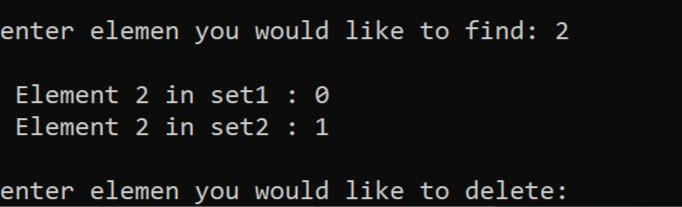


Рис. 13. В первом множестве нет этого элемента, а во втором есть.

1. Далее введем элемент который хотим удалить и выводим результат (Рис. 14).

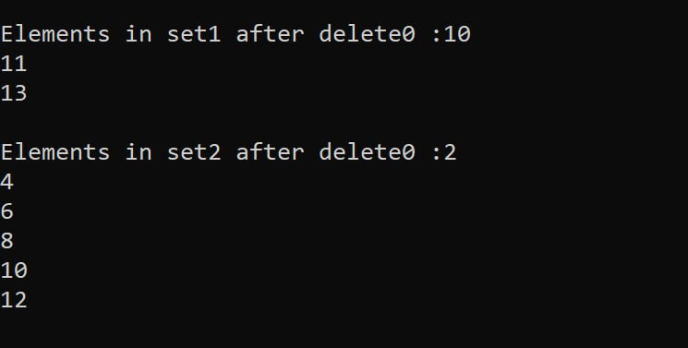


Рис. 14. Основное окно программы.

1. Создаём третье множество. Для него выполняется операция присваивания (присваивание третьему множеству значения первого).
2. Далее программа выводит нам результаты проверки множеств на равенство и неравенство (Рис. 15).

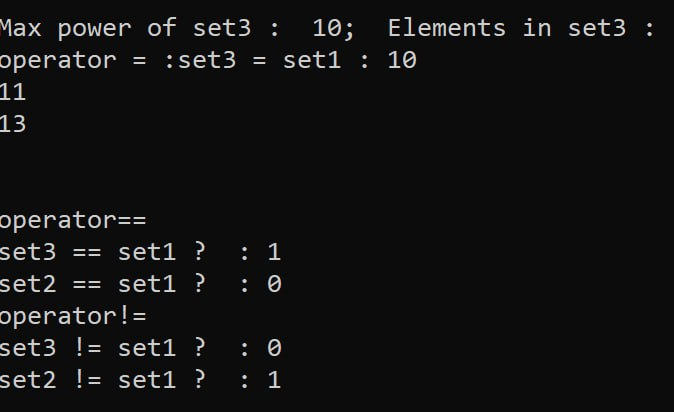


Рис. 15. Основное окно программы.

1. Проводим операцию объединения первого и второго множества. Получаем результат (Рис. 16).

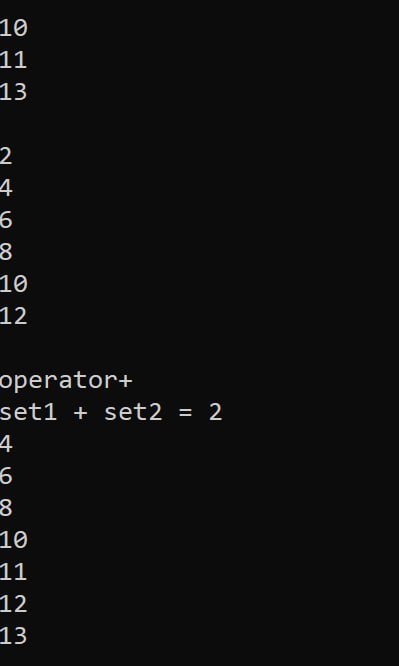


Рис. 16. Основное окно программы.

1. Проверим работу оператора добавления элемента в множествои оператора удаления элемента из множества (Рис. 17).

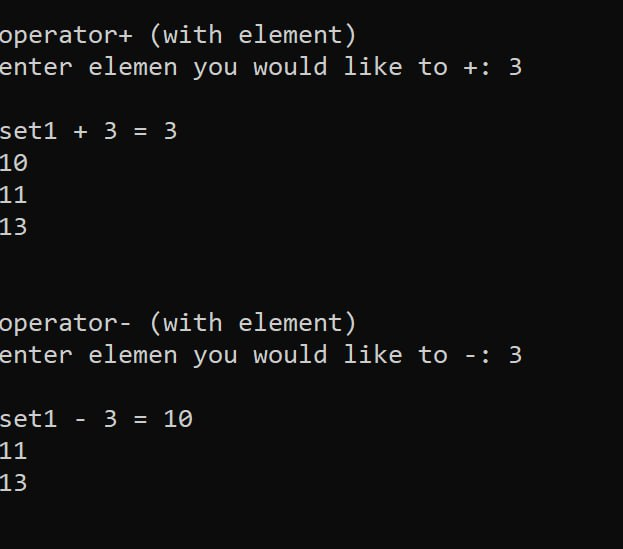


Рис. 17. Основное окно программы.

1. Так же программа выведет нам результат операции пересечения множеств (первого и второго) (Рис. 18).
2. И получим результат операции дополнения множества (второго) (Рис. 19).

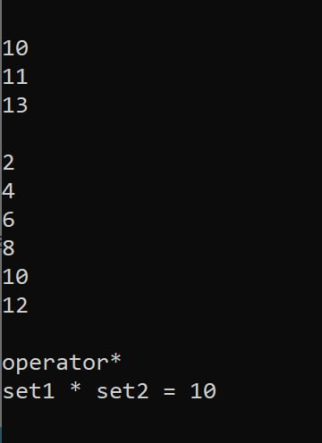


Рис. 18. Основное окно программы.

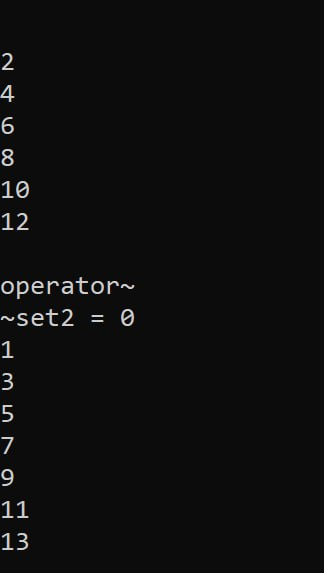


Рис. 19. Основное окно программы.

1. Исполнение программы окончено.

## «Решето Эратосфено»

1. Запустите приложение с названием sample\_primenumbers.exe. В результате появится окно, показанное ниже (Рис. 20).

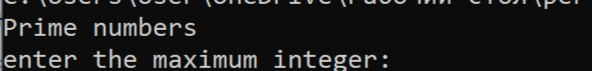


Рис. 20. Основное окно программы.

1. Программа просит нас ввести максимальное число (введем его).
2. В результате нам выводятся все простые числа входящие в это множество (Рис. 21).

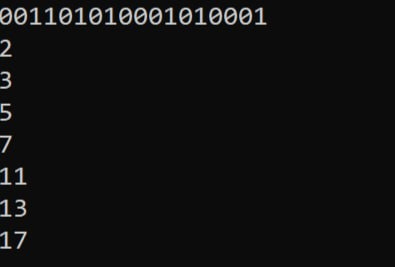


Рис. 21. Основное окно программы.

1. Работа программы окончена.

# Руководство программиста

## Описание алгоритмов

### Битовые поля

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 представляется в качестве последовательности из нулей и единиц, в контексте нашей задач ноль будет обозначать, что элемент отсутствует в множестве, единица – что элемент принадлежит множеству. Поставим каждому множеству в соответствие характеристический вектор (битовое поле). Таким образом все операции над множествами могут быть заменены в таком случае на операции над характеристическими векторами. В дальнейшем в работе мы будем решать задачу хранения и обработки именно характеристических векторов.

Для работы с Битовым полем предлагается реализовать следующие операции:

* установить бит (в единицу);
* очистить бит (в ноль);
* получить значение бита;
* сравнить два битовых поля;
* выполнить операцию “логическое или” для двух битовых полей;
* выполнить операцию “логическое и” для двух битовых полей;
* выполнить операцию “логическое отрицание” для битового поля.

Принцип работы:

* логическое или для двух битовых полей.

Операция требуется для объединения двух множеств.

Пример реализации:

Пусть мощность двух множеств равна 10.

Каждому из множеств соответствуют характеристические векторы.

Множество 1: 0110010001

Множество 2: 1110010110

Результатом объединения этих множеств будет новый характеристический вектор: 1110010111.

* Логическое и для двух битовых полей.

Операция требуется для пересечения двух множеств.

Пример реализации:

Пусть мощность двух множеств равна 10.

Каждому из множеств соответствуют характеристические векторы.

Множество 1: 0110010001

Множество 2: 1110010110

Результатом пересечения этих множеств будет новый характеристический вектор: 0110010000.

* Логическое отрицание для битового поля.

Операция требуется для нахождения дополнения к множеству.

Пример реализации:

Пусть мощность множества равна 10.

Множеству соответствуют характеристические векторы.

Множество 1: 0110010001

Результатом дополнения к множеству будет новый характеристический вектор: 1001101110.

* Сравнение двух битовых полей.

Операция требуется для проверки на равенство двух множеств.

Пример реализации:

Пусть мощность двух множеств равна 10.

Каждому из множеств соответствуют характеристические векторы.

Множество 1: 0110010001

Множество 2: 1110010110

Результатом объединения этих множеств будет число (1 или 0) 1 в случае равенства и 0 в случае неравенства: 0.

* Получить значение бита.

Операция требуется для проверки наличия элемента множества (в качестве результата будет выведено число 1 (что говорит о наличии элемента в множестве) или 0 (что говорит об отсутствии элемента в множестве)).

Пример реализации:

Пусть мощность множества равна 10.

Множеств соответствует характеристический вектор.

Множество 1: 0110010001

Допустим мы хотим узнать входит ли 5 в множество.

В качестве результата нам выйдет число 1 (обход по характеристическому вектору идёт от нуля) что говорит о том, что такой элемент имеется в множестве.

* Реализация битовой маски.

Битовая маска является инструментом для реализации трех необходимых для нас операций. Таких как установка бита (включение элемента в множество), очистка биты (исключение элемента из множества) и получения значения бита.

Для создания битовой маски всё битовое поле изначально инициализируется нулями.

И вычисляется смещение бита.

То есть если мы хотим создать битовую маску по индексу пять (пусть длина битового поля 10) битовая маска будет выглядеть так: 0000010000.

* Установить бит (в единицу).

Операция требуется для добавления элемента в множество.

Алгоритм:

Для установки бита в маске:

1. Вычисляем индекс слова с помощью деления позиции бита на размер слова и взятия целой части результата.
2. Вычисляем смещение бита внутри слова с помощью остатка от деления позиции бита на размер слова.
3. Устанавливаем бит в соответствующем слове с помощью побитовой операции OR с маской, содержащей только один установленный бит на позиции смещения.

Пример реализации:

Пусть битовое поле имеет вид: 1100001100. Мы хотим установить бит по индексу 4 (считаем от нуля).

Тогда битовая маска будет иметь вид 00001000000.

И с помощью уже ранее реализованной операции или мы получим новое битовое поле: 1100101100.

* Очистить бит.

Операция требуется для удаления элемента из множества.

Алгоритм:

1. Вычисляем индекс слова, в котором находится данный бит.
2. Вычисляем смещение бита внутри слова.
3. Сбрасываем бит в соответствующем слове с помощью побитовой операции AND с инвертированной маской, содержащей только один установленный бит на позиции смещения.

Пример реализации:

Пусть битовое поле имеет вид: 1111101100. Мы хотим удалить бит по индексу 2 (считаем от нуля).

Тогда битовая маска будет иметь вид: 00100000000.

Инвертируем маску: 1101111111

И с помощью уже ранее реализованной операции и мы получим новое битовое поле: 1101101100.

### Множества

Основные термины:

*Множеством* называется совокупность некоторых объектов, объединенных в одно целое по какому ‒ либо признаку.

Объекты, из которых состоит множество, называются его *элементами.*

Множество, не содержащее ни одного элемента, называется *пустым*.

Множеством *A*называется *подмножеством* *B*, если каждый элемент множества *A* является элементом множества *B.*

Два множества *A* и *B* называются *равными*, если они состоят из одних и тех же элементов (*A* =*B).*

*Дополнением* множества A (или отрицанием множества A) называется множество всех элементов, не принадлежащих множеству A.

Для работы с Множеством предлагается реализовать следующие операции:

* включение элемента в множество;
* исключение элемента из множества;
* проверка наличия элемента в множестве;
* сравнение множеств;
* сложение множеств;
* пересечение множеств;
* разность множеств;

Данные операции реализуются для характеристических векторов множеств их реализацию мы описали в описании алгоритмов битовых полей.

* Включение элемента в множество.

Входные данные: элемент который надо включить в множество.

Выходные данные: множество со включенным элементом.

Пример реализации:

Пусть множество имеет вид: {1,6,7,8,9}.

Элемент, который мы хотим включить: 5.

Множество со включенным элементом: {1,5,6,7,8,9}.

* Исключение элемента из множества.

Входные данные: элемент который надо исключить в множество.

Выходные данные: множество без элемента.

Пример реализации:

Пусть множество имеет вид: {1,5,6,7,8,9}.

Элемент, который мы хотим исключить: 5.

Множество со включенным элементом: {1,6,7,8,9}.

* Проверка наличия элемента в множестве.

Входные данные: элемент который надо найти.

Выходные данные: 0,1 – (0- если не нашли, 1 – если нашли).

Пример реализации:

Пусть множество имеет вид: {1,5,6,7,8,9}.

Элемент, который мы хотим найти: 5.

Результат: 1.

* Сравнение множеств.

Входные данные: два множества.

Выходные данные: число-результат сравнения множеств.

Множество 1: {1,3,5,7}

Множество 2: {2,3,8,9}

Результатом сравнения этих множеств будет число (1 или 0) 1 в случае равенства и 0 в случае неравенства: 0.

* Сложение множеств.

Входные данные: два множества.

Выходные данные: множество – результат сложения двух множеств.

Множество 1: {1,3,5,7}

Множество 2: {2,3,8,9}

Результатом сложения этих множеств будет множество, включающее элементы двух множеств: {1,2,3,5,7,8,9}.

* Пересечение множеств.

Входные данные: два множества.

Выходные данные: множество – результат пересечения двух множеств.

Множество 1: {1,3,5,7}

Множество 2: {2,3,8,7}

Результатом пересечения этих множеств будет множество, включающее элементы двух множеств: {1,7}.

* Разность множеств.

Входные данные: два множества.

Выходные данные: множество- результат разности множеств.

Множество 1: {1,3,5,7}

Множество 2: {2,3,8,9}

Результатом будет множество {1,5,7}

### «Решето Эратосфена»

Задача: найти все простые числа, не превышающие заданного числа.

Входные данные: число n, задающее диапазон натуральных чисел (от 2 до n).

Выходные данные: множество простых чисел в заданном диапазоне.

Алгоритм:

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

## Описание программной реализации

### Описание класса TBitField

class TBitField

{

private:

int BitLen;

TELEM \*pMem;

int MemLen;

int GetMemIndex(const int n) const;

TELEM GetMemMask (const int n) const;

public:

TBitField(int len);

TBitField(const TBitField &bf);

~TBitField();

int GetLength(void) const;

void SetBit(const int n);

void ClrBit(const int n);

int GetBit(const int n) const;

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

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

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 битовой маски для бита.

Входные параметры: n – номер бита.

Выходные параметры: битовая маска для бита n.

int GetLength(void) const

Назначение: получение получить длину битового поля (к-во битов) .

Входные параметры: нет.

Выходные параметры: максимальное количество битов в битовом поле.

void SetBit(const int n);

Назначение: установить бит по индексу.

Входные параметры: n – номер бита.

Выходные параметры: нет.

void TBitField::ClrBit(const int n);

Назначение: очистить бит по индексу.

Входные параметры: n – номер бита.

Выходные параметры: нет.

int GetBit(const int n) const;

Назначение: получить значение бита по индексу.

Входные параметры: n – номер бита.

Выходные параметры: значение бита.

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

Назначение: проверка элементов типа TBitField на равенство.

Входные параметры: ссылка на элемент типа TBitField.

Выходные параметры: результат сравнения двух элемент типа TBitField.

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

Назначение: проверка элементов типа TBitField на неравенство.

Входные параметры: ссылка на элемент типа TBitField.

Выходные параметры: результат сравнения двух элемент типа TBitField.

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

Назначение: присваивание одним элементом типа TBitField значениий полей другого элемента типа TBitField.

Входные параметры: константная ссылка на элемент типа TBitField.

Выходные параметры: элемент типа TBitField - результат присваивания первому элементу типа TBitField значения второго элемента TBitField.

TBitField operator|(const TBitField &bf);

Назначение: выполнение логической операции or между двумя элементами типа TBitField .

Входные параметры: константная ссылка на элемент типа TBitField.

Выходные параметры: элемент типа TBitField - результат логической операции or между двумя элементами типа TBitField.

TBitField operator&(const TBitField &bf);

Назначение: выполнение логической операции and между двумя элементами типа TBitField .

Входные параметры: константная ссылка на элемент типа TBitField.

Выходные параметры: элемент типа TBitField - результат логической операции and между двумя элементами типа TBitField.

TBitField operator~(void);

Назначение: выполнение операции отрицания элемента типа TBitField .

Входные параметры: нет.

Выходные параметры:

элемент типа TBitField - результат выполнение операции отрицания элемента типа TBitField.

### 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;

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– максимальная мощность множества.

BitField– битовое поле для хранения характеристического вектора.

Методы:

int GetMaxPower(void) const;

Назначение: получение максимальной мощности множества.

Входные параметры: нет.

Выходные параметры: мощность множества.

void InsElem(const int Elem);

Назначение: включить элемента в множество.

Входные параметры: элемент.

Выходные параметры: нет.

void DelElem(const int Elem);

Назначение: удаление элемента из множества.

Входные параметры: элемент.

Выходные параметры: нет.

int IsMember(const int Elem) const;

Назначение: проверка наличие элемента в множестве.

Входные параметры: элемент.

Выходные параметры: значение бита по индексу.

Перегруженные операции:

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

Назначение: проверка элементов типа TSet на равенство.

Входные параметры: ссылка на элемент типа TSet.

Выходные параметры: элемент типа TSet - результат сравнения двух элемент типа TSet.

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

Назначение: проверка элементов типа TSet на неравенство.

Входные параметры: ссылка на элемент типа TSet.

Выходные параметры: элемент типа TSet - результат сравнения двух элемент типа TSet.

TSet& operator=(const TSet &s);

Назначение: присваивание одним элементом типа TSet значениий полей другого элемента типа TSet.

Входные параметры: константная ссылка на элемент типа TSet.

Выходные параметры: элемент типа TSet - результат присваивания первому элементу типа TSetзначения второго элемента TSet.

TSet operator+ (const int Elem);

Назначение: объединение множества с элементом.

Входные параметры: элемент.

Выходные параметры: элемент типа TSet - результат объединения множества с элементом..

TSet operator- (const int Elem);

Назначение: исключение элемента из множества.

Входные параметры: элемент.

Выходные параметры: элемент типа TSet - результат исключения элемента из множества..

TSet operator+ (const TSet &s);

Назначение: объединение двух множеств.

Входные параметры: множество.

Выходные параметры: элемент типа TSet - результат объединения множеств.

TSet operator\* (const TSet &s);

Назначение: операция пересечение множеств.

Входные параметры: множество.

Выходные параметры: элемент типа TSet - результат пересечения множеств.

TSet operator~ (void);

Назначение: операция дополнения к множеству.

Входные параметры: нет.

Выходные параметры: элемент типа TSet – результат операции дополнения к множеству.

# Заключение

В данной лабораторной работе мы научились создавать программные средства, поддерживающих эффективное хранение множеств и выполнение основных операций над множествами:

• включение элемента в множество;

• исключение элемента из множества;

• проверка наличия элемента в множестве;

• сложение множеств;

• пересечение множеств;

• разность множеств;

• копирование множества; 20

• вычисление мощности множества. Программные средства должны содержать:

• класс Множество;

• тестовое приложение, демонстрирующее использование основных операций с множествами.

# Литература

1. Барышева И.В., Мееров И.Б., Сысоев А.В., Шестакова Н.В. Под редакцией Гергеля В.П Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017. – 105с.

# Приложения

## Приложение А. Реализация класса TBitField

TBitField::TBitField(int len)

{

if (len <= 0)

throw "the length is specified incorrectly";

if (len > 0)

{

BitLen = len;

MemLen = ((BitLen - 1) / UnsignedInt) + 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" n < 0 !!!";

return (n ) / UnsignedInt;

}

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

{

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

throw "wrong number";

return 1 << (n % (UnsignedInt));

}

// 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 "wrong number";

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

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

}

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

{

int index = GetMemIndex(n);

int BitIndex = n % UnsignedInt;

unsigned int element = pMem[index];

unsigned int BitMask = ~(GetMemMask(BitIndex));

pMem[index] = element & BitMask;

}

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

{

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

throw "wrong number";

else

{

int index = GetMemIndex(n);// индекс элемента массива

int BitIndex = n % UnsignedInt;//индекс бита в элементе массива

unsigned int element = pMem[index];

unsigned int bitMask = GetMemMask(BitIndex);

return(element & bitMask) != 0;

}

}

// битовые операции

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

{

if (this != &bf)

{

if (BitLen != bf.BitLen)

{

delete[] pMem;

pMem = new TELEM[MemLen];

}

MemLen = bf.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.BitLen)

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

{

if (BitLen != bf.BitLen)

return 1;

else

{

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

{

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

return 1;

}

}

return 0;

}

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

{

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

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

TBitField tmp(maxLen);

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

{

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

if (GetBit(i) == 1)

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);

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

{

if (GetBit(i) == 1)

tmp.SetBit(i);

}

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

{

if (bf.GetBit(i) == 1)

tmp.SetBit(i);

}

return tmp;

}

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

{

TBitField tmp(BitLen);

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

{

if (GetBit(i) == 0)

tmp.SetBit(i);

}

return tmp;

}

// ввод/вывод

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

{

string str;

istr >> str;

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

{

int bit = str[i] - '0';

if (bit == 1)

bf.SetBit(i);

if (bit == 0)

bf.ClrBit(i);

}

return istr;

}

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

{

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

{

ostr << bf.GetBit(i);

}

return ostr;

}

## Приложение Б. Реализация класса TSet

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

{

if (mp >= 0)

MaxPower = mp;

else

throw"Error";

}

// 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;

}

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 "Error";

return BitField.GetBit(Elem);

}

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

{

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

throw "Error";

return BitField.SetBit(Elem);

}

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

{

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

throw "Error";

return BitField.ClrBit(Elem);

}

// теоретико-множественные операции

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

{

if (this != &s)

{

MaxPower = s.MaxPower;

BitField = s.BitField;

}

return \*this;

}

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

{

if (MaxPower != s.MaxPower)

return 0;

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

{

if (BitField.GetBit(i) != s.BitField.GetBit(i))

return 0;

}

return 1;

}

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

{

if (MaxPower != s.MaxPower)

return 1;

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

{

if (BitField.GetBit(i) != s.BitField.GetBit(i))

return 1;

}

return 0;

}

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

{

int MaxP = max(MaxPower, s.MaxPower);

TSet tmp(MaxP);

tmp.BitField = BitField | s.BitField;

return tmp;

}

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

{

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

throw "Error";

TSet tmp(MaxPower);

tmp.BitField = this->BitField;

tmp.BitField.SetBit(Elem);

return tmp;

}

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

{

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

throw "Error";

TSet tmp(MaxPower);

tmp.BitField = this->BitField;

tmp.BitField.ClrBit(Elem);

return tmp;

}

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

{

int MaxP = max(MaxPower, s.MaxPower);

TSet tmp(MaxP);

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

{

istr >> s.BitField;

return istr;

}

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

{

ostr << s.BitField;

return ostr;

}

## Приложение С. Реализация Sample\_primenumbers

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

int n;

cout << "enter the maximum integer: ";

cin >> n;

TSet s(n + 1);

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

s.InsElem(m);

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

if (s.IsMember(m))

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

s.DelElem(k);

cout << s << endl;

return 0;