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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Шпынов Н.А. /

Подпись

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

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

Подпись

Нижний Новгород  
2023

# Введе**н**ие

## Актуальность

Битовые поля - это структуры данных, которые позволяют компактно хранить и работать с набором булевских значений. Каждое поле представляет собой группу битов в памяти компьютера, каждому из которых присваивается определённая позиция.

Использование битовых полей даёт большую гибкость, обеспечивает оптимизацию памяти и удобство работы с различными флагами и настройками в программе. Также они позволяют управлять информацией и хранить её в компактном формате, что является ценным ресурсом во многих областях разработки программного обеспечения.

## Значимость

Битовые поля являются важным и значимым инструментом в программировании

1. Оптимизация использования памяти

 Использование битовых полей позволяет сократить расход памяти. С их помощью можно упаковывать несколько различных флагов или настроек в одно целочисленное поле, вместо того, чтобы использовать отдельные переменные или массивы.

1. Удобство чтения и записи значений

Битовые поля позволяют мы легко читать и записывать значения в отдельные биты поля. Например, если объекту присвоить своё битовое поле, мы можем использовать каждый бит для отображения различных его свойств или состояний. Это делает код более читаемым и удобным для работы.

1. Кросс-платформенность

Битовые поля позволяют достичь определённой степени аппаратной независимости. Программа, использующая битовые поля, может работать на разных архитектурах и платформах без изменений в логике.

## Применимость

Использование битовых полей даёт множество преимуществ в реализации многих структур и программ. Примеры использования битовых полей:

1. Флаги состояний

битовые поля можно использовать для представления и управления флагами состояния объектов или системы. Каждому состоянию "включено/выключено" или "открыто/закрыто" ставится в соответствие свой бит, значение которого и будет указывать на текущее положение.

1. Настройки конфигурации

Битовые поля позволяют упаковать различные настройки или флаги конфигурации в одно поле, что делает управление, использование и применение настроек какого то объекта более удобными.

1. Сериализация данных

 Битовые поля могут быть использованы для сериализации или десериализации данных (*перевод структуры данных в битовую последовательность и наоборот*), когда требуется компактно передать или сохранить данные.

1. Реализация множеств

Битовые поля являются одним из простейших способов реализации алгебры логики в программе. Каждому элементу в универсе ставится в соответствие свой бит, обозначающий его наличие или отсутствие в данном множестве

# Постановка задачи

Цель – реализовать представление множеств на языке C++ при помощи битовых полей

Задачи:

1. Разработка представления битовых полей

Для существования системы множеств на основе битовых полей, следует создать реализацию их самих.

1. Определить способ хранения битовых полей
2. Создать структуру данных с битовыми полями
3. Реализовать необходимые для работы поля и методы
4. Разработка структуры данных для представления множества

Определить, каким образом будут храниться элементы множества с использованием битовых полей. Потребуется создать специальную структуру данных, в которой каждый бит будет соответствовать наличию или отсутствию определённого элемента.

1. Определение операций над множествами

Разработать алгоритмы и функции для выполнения операций над множествами, таких как объединение, пересечение, разность, проверка на принадлежность элемента, добавление/удаление элементов из множества и другие.

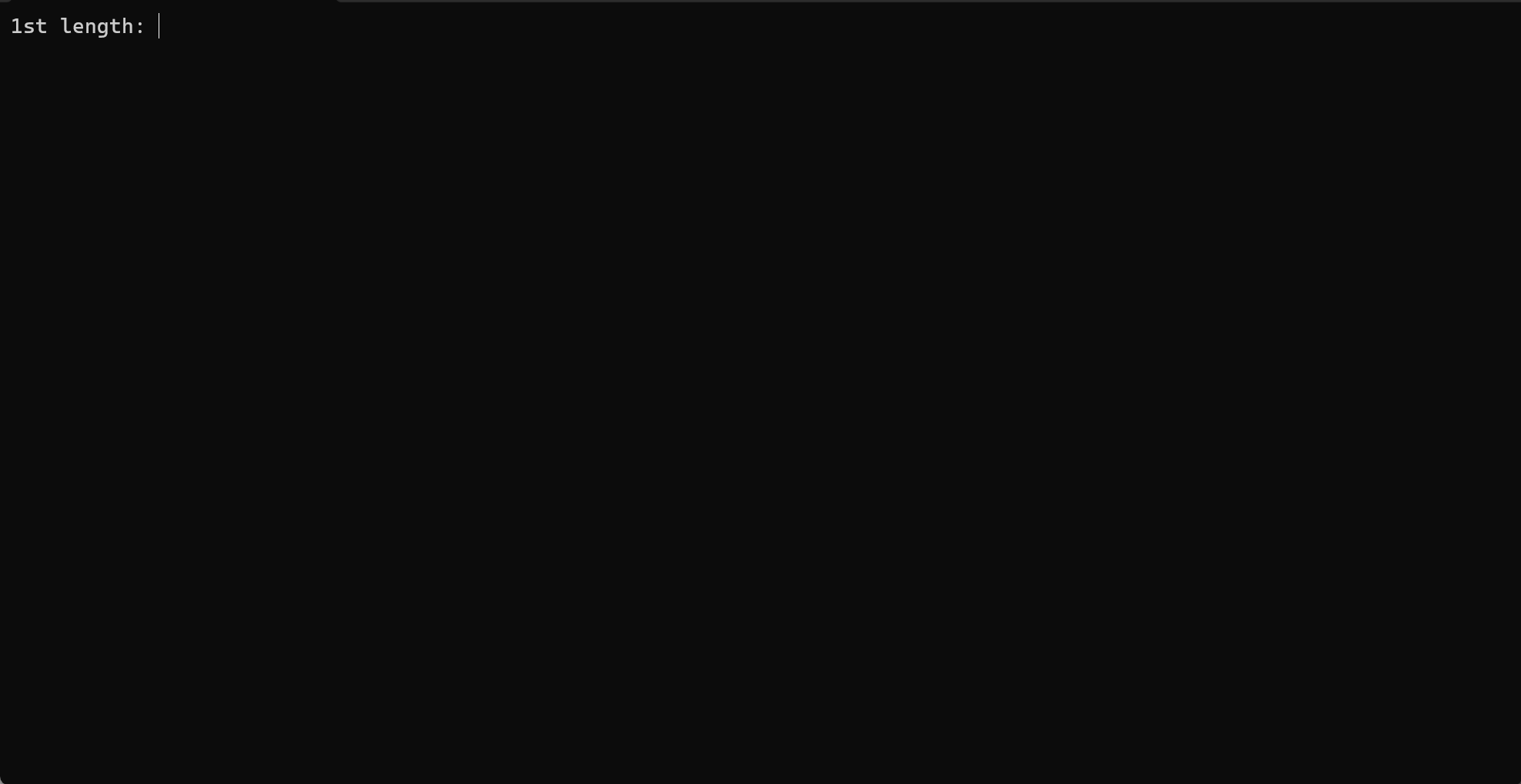
1. Тестирование и отладка.

Как и при любой разработке программного обеспечения, в конце необходимо провести тестирование и отладку кода. Следует обнаружить и исправить возможные ошибки и проблемы, а также убедиться в правильности функционирования.

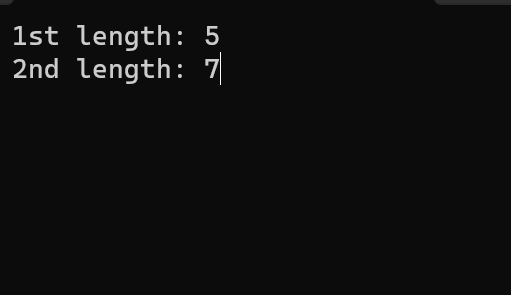
# Руководство пользователя

## Приложение для демонстрации работы битовых полей

1. Запустить sample\_tbitfield.exe. В результате появится следующее окно:



1. Рис. 1. Основное окно приложения
2. Затем программа попросит ввести длины (количество элементов) двух полей:



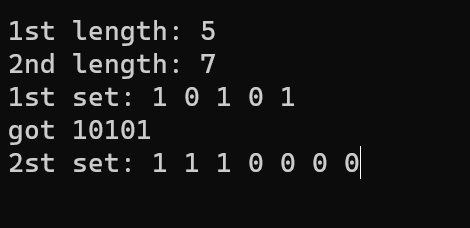
1. Рис. 2. ввод длин
2. Далее потребуется ввести поля соответственно следующим образом. Сначала вводятся все элементы первого поля по очереди через пробел, 0 или 1. Затем точно также и у второго поля:
3. 

Рис. 3. результат ввода полей

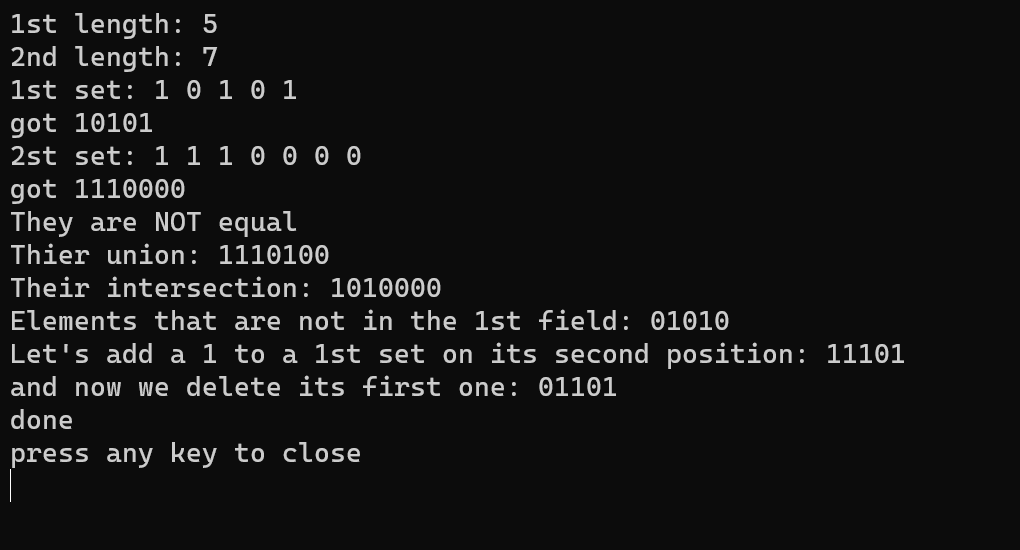
1. Программа покажет примеры результатов операций над полями. После чего попросит ввести любое значение для выхода.
2. 

Рис. 4. итог

## Приложение для демонстрации работы множеств

1. Запустить sample\_tset.exe. В результате появится следующее окно:



Рис. 1. Основное окно приложения

1. Программа запросит длину универса (максимальное количество элементов):

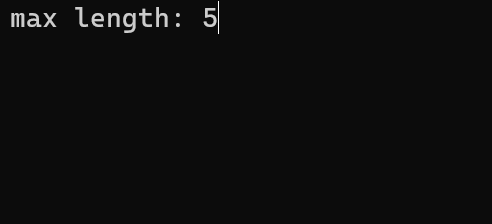


Рис. 2. ввод длины

1. Далее потребуется ввести множества соответственно следующим образом. Сначала вводятся все элементы первого множества через пробел, вводятся элементы в любом порядке по их номеру до того, пока множество не станет максимальным или не будет введено орицательное число, обозначающее, что ввод множества закончен. Затем точно также и у второго множества:

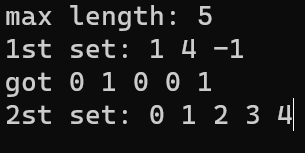


Рис. 3. результат ввода множеств

1. Программа покажет примеры результатов операций над множествами. После чего попросит ввести любое значение для выхода.

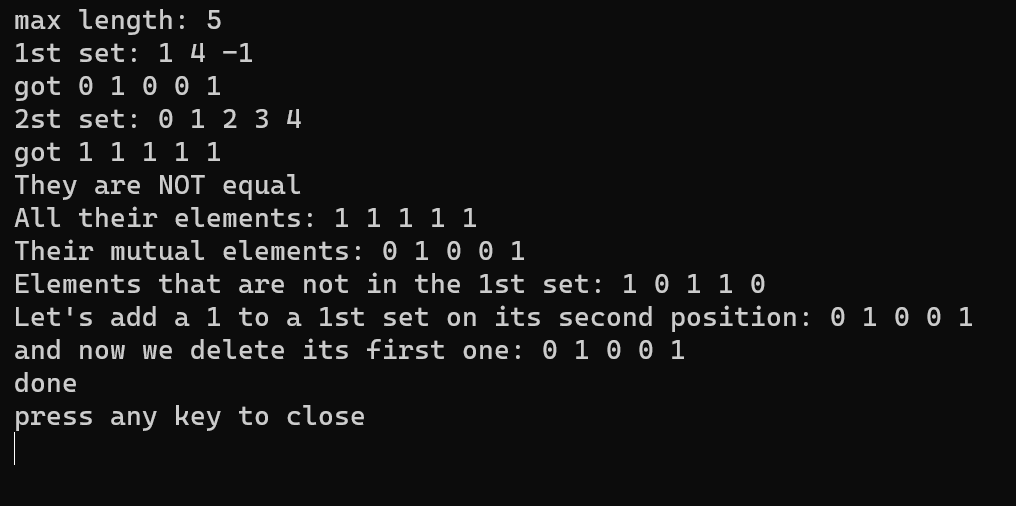


Рис. 4. итог

## Приложение «решето Эратосфена»

1. Запустить sample\_primenumbers.exe. В результате появится следующее окно:

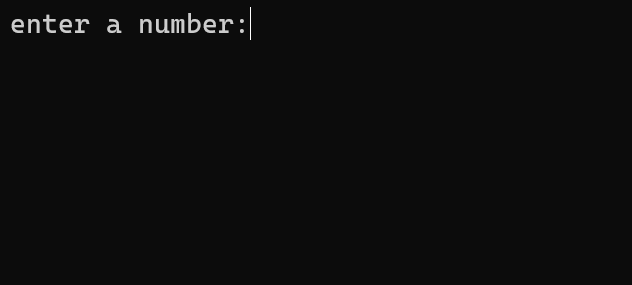


Рис. 1. Основное окно приложения

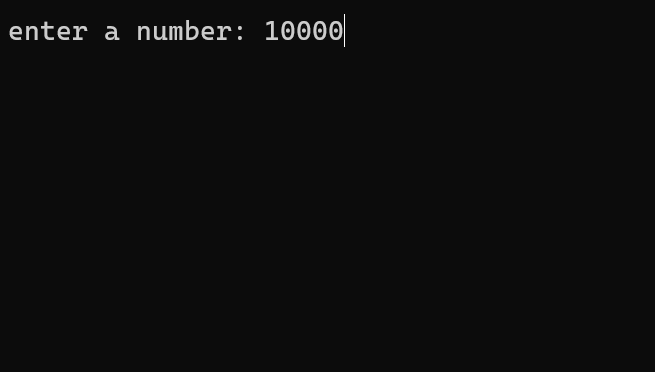
1. Программа запросит число, в рамках которого будет искать все простые числа, нужно его ввести:
2. 

Рис. 2. ввод числа

1. Программа выдаст все простые числа, меньше введённого

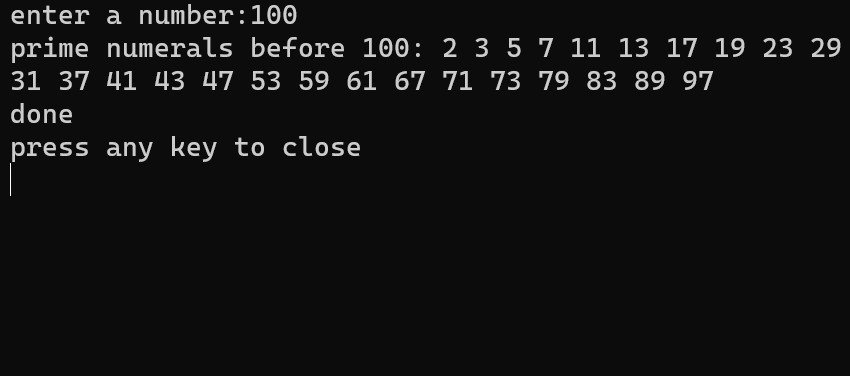


Рис. 3. результат

# Руководство программиста

## Использованнные алгоритмы

### Битовые поля

Битовое поле - это класс, полями которого являются его длина в битах, его длина во вспомогательных единицах, массив этих единиц.

Операции над полями реализованы в качестве методов класса полей

### Множества

Множество - это класс, полями которого является длина его универса и битовое поле, хранящее информацию о включении элементов в это множество массив этих единиц.

Операции над полями реализованы в виде методов класса множество

### Алгоритм «решето Эратосфена»

1. Получить число n, до которого требуется искать простые числа
2. Создать вспомогательное битовое поле длины n+1, заполненное единицами
3. Пройти по полю до максимального числа, квадрат которого меньше n
4. Для каждого из полученных чисел пройди по массиву от удвоенного номера этого числа до конца с шагом в это число и удалить все полученные элементы
5. Вывести результат

## Описание классов

### Класс TbitField

Объявление класса:

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();

// 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 – массив вспомогательных единиц

MemLen – количество вспомогательных единий

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 int n) const;**

Назначение: получение маски элемента, где хранится бит.

Входные данные:

n – номер бита.

Выходные данные:

Число, являющееся маской бита с номером n

**TELEM GetMemMask (const int n) const;**

Назначение: получение маски элемента, где хранится бит.

Входные данные:

n – номер бита.

Выходные данные:

Число, являющееся маской бита с номером n

### Класс TSet

# Заключение

# Реализация множеств при помощи битовых полей оказалась эффективным подходом, позволяющим компактно хранить и оперировать наборами элементов. Этот метод позволяет сократить затраты на память, обеспечивает быстрые операции с множествами и может быть применён в решении различных задач, связанных с манипуляцией данными и их анализом.

# Литература

1. <https://ru.wikipedia.org/>
2. <https://prog-cpp.ru/>
3. <https://en.cppreference.com/w/>
4. <https://github.com/>

# Приложения

## Приложение А. Реализация класса TBitField

## Приложение Б. Реализация класса TSet