gМИНИСТЕРСТВО 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 3822Б1ФИ2
_____ / Холин К.И./
Подпись

**Проверил:** к.т.н, доцент каф. ВВиСП
_____ / Кустикова В.Д./
Подпись

Нижний Новгород
2023

# Содержание

Введение....3
1 Постановка задачи....4
2 Руководство пользователя....5
2.1 Приложение для демонстрации работы битовых полей....5
2.2 Приложение для демонстрации работы множеств....7
2.3 «Решето Эратосфена»....9
3 Руководство программиста....11
3.1 Описание алгоритмов....11
3.1.1 Битовые поля....11
3.1.2 Множества....14
3.1.3 «Решето Эратосфена»....16
3.2 Описание программной реализации....16
3.2.1 Описание класса TBitField....16
3.2.2 Описание класса TSet....19
Заключение....25
Литература....26
Приложения....27
Приложение А. Реализация класса TSet....27
Приложение Б. Реализация класса TBitField....31

# Введение

В C++ иногда возникают такие ситуации, когда информацию об объекте достаточно хранить в формате состояний (статусов), представляющих из себя 0 и 1. На этом основывается представление множества в виде характеристического массива. Это оптимальный вариант, поскольку формально обеспечивает хранение признака наличия или отсутствия элемента универса во множестве, а не прямое хранение элементов. Обращение к определённому биту позволяет нам узнать его состояние для выполнения конкретной задачи. Битовые поля в этом случае играют важную роль.

# 1 Постановка задачи

Цель – реализовать классы для представления битовых полей TBitField и множеств TSet.

Задачи:

1. Разработать класс TBitField. Написать следующие операции для работы с битовыми полями: установить бит в 1,установить бит в 0,получить значение бита,сравнить два битовых поля,сложить и инвертировать,вывести битовое поле требуемого формата и ввести битовое поле. Добавить вспомогательные операции получения бита,маски бита,длины битового поля.
2. Разработать класс TSet. Написать следующие операции для работы с множествами: вставка элемента,удаление,проверка наличия,сравнение множеств,объединение множеств,пересечение,разность,копирование,вычисление мощности множества,вывод элементов множества требуемого формата и ввод. Добавить вспомогательные операции для получения мощности множества.

# 2 Руководство пользователя

## 2.1 Приложение для демонстрации работы битовых полей

1. Запустите приложение с названием sample_TBitField.exe. В результате появится окно, представленное ниже (рис. 1).

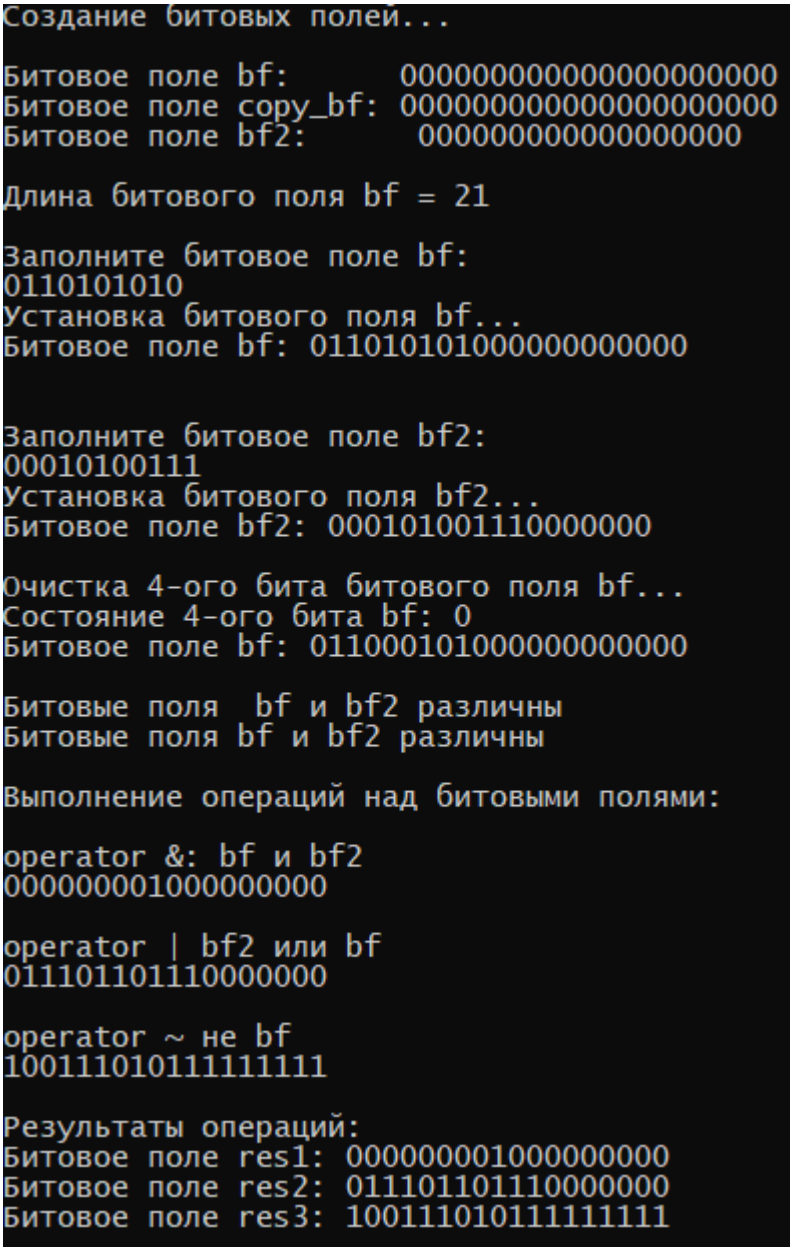


Рис. 1. Основное окно программы

2. Сначала создаются 3 битовых поля: bf,copy_bf и bf2 (рис. 2).

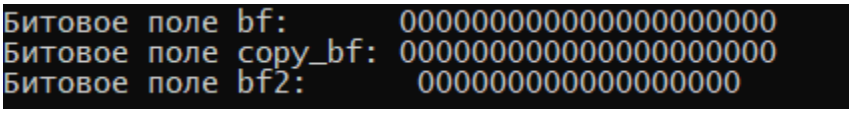


Рис. 2. Создание битовых полей.

3. На следующем шаге заполняется битовое поле bf и выводится на экран(рис. 3).

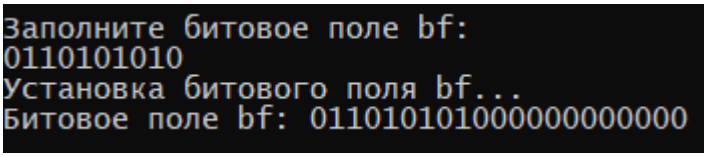


Рис. 3. Установка битового поля bf с выводом длины

4. После заполнения битового поля bf заполняется битовое поле bf2(рис. 4)

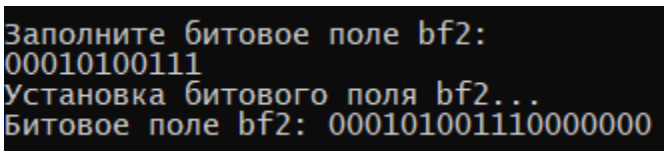


Рис. 4. Установка битового поля bf2

5. На 5 этапе удаляется бит с номером 4 из битового поля bf(рис. 5)

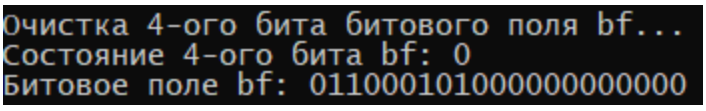


Рис. 5. Удаление 4-го бита битового поля bf2

6. В первой строке проверяется операция проверки на равенство битовых полей bf и bf2,а во второй- операция на неравенство(рис.6)

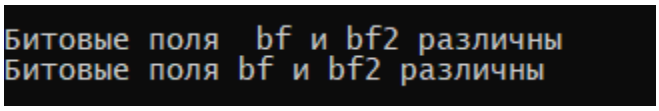


Рис. 6. Сравнение битовых полей

7. Выполняются различные операции с битовыми полями, как показано на (рис. 7)

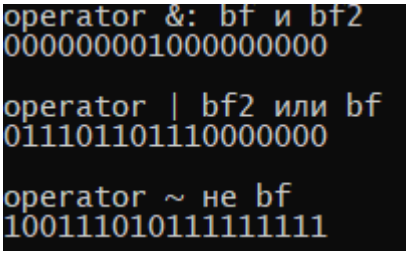


Рис. 7. Операции над битовыми полями

8. И в завершение выводятся все результаты вычислений(рис. 8)

```
Результаты операций:
Битовое поле res1: 0000000010000000000
Битовое поле res2: 0111011011110000000
Битовое поле res3: 1001110101111111111
```

Рис. 8. Результаты вычислений операций над битовыми полями

## 2.2 Приложение для демонстрации работы множеств

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

```
Создание множеств...

Представление мощностей множеств:
Мощность множества A = 11
Мощность множества B = 21
Мощность множества копии B = 21
Мощность множества C = 16

Ввод элементов множества A:
Введите элементы множества A
How many element do you want enter?
8
2 3 4 5 9 10 0 1
Continue?(1/0)0

Ввод элементов множества B:
Введите элементы множества B
How many element do you want enter?
5
5 1 9 0 4
Continue?(1/0)0

Ввод элементов множества C:
Введите элементы множества C
How many element do you want enter?
3
2 4 6
Continue?(1/0)0

Сравнение множеств A и  B:

Проверка на равенство:
Множества A и B разной мощности

Проверка на неравенство:
Множества A и B разной мощности

Выполнение теоретико-множественных операций над множествами:

Объединение множества A с элементом 9
A= {0,1,2,3,4,5,9,10,}

Разность множества B с элементом 14
B= {0,1,4,5,9,}

Объединение множества A с множеством B
res1= {0,1,2,3,4,5,9,10,}

Пересечение множества A с множеством C
res2= {2,4,}

Дополнение к множеству C
res3= {0,1,3,5,7,8,9,10,11,12,13,14,15,}

Разность множеств A и C
res4= {0,1,3,5,9,10,}

Множества A,B,C
A= {0,1,2,3,4,5,9,10,}
B= {0,1,4,5,9,}
C= {2,4,6,}
```

Рис. 1. Окно основной программы

2. По началу множества пустые, так как в них не содержится никаких элементов. Максимальные мощности множеств представлены на (рис. 2)

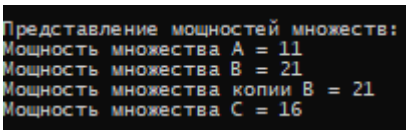


Рис. 2. Мощности множеств A,B,C

3. На третьем этапе идёт заполнение множеств с предварительным вводом количества элементов. (рис. 3)

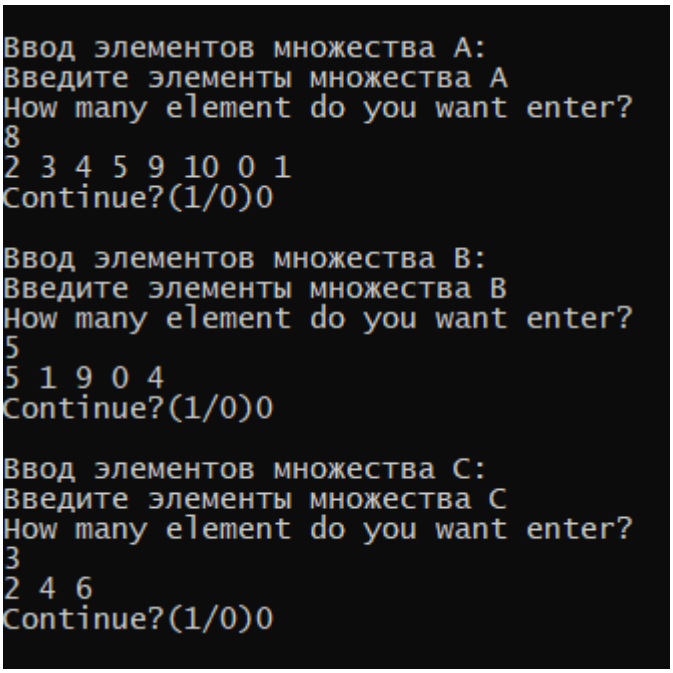


Рис. 3. Заполнение множеств A,B,C

4. В первом случае множества A и B проверяются на равенство,а во втором – на неравенство.(рис. 4)

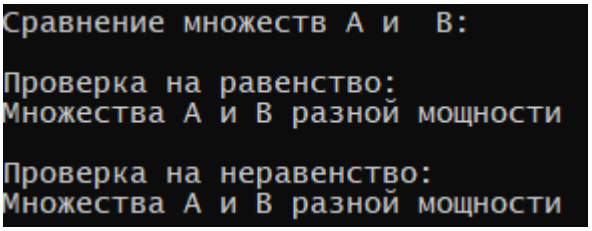


Рис. 4. Сравнение множеств A и B

5. На рисунке ниже приведены основные операции с множествами(рис. 5)

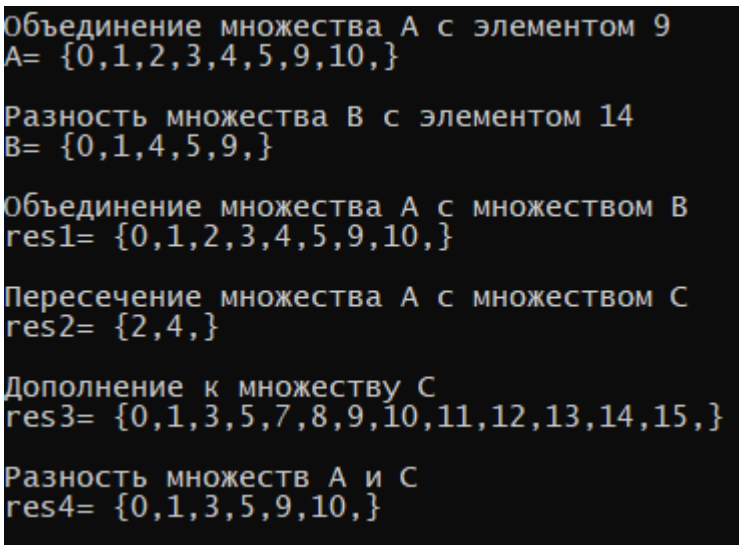


Рис. 5. Основные операции с множествами A,B,C

6. Для удобства сравнения результатов на экран выводятся множества A,B,C (рис. 6)

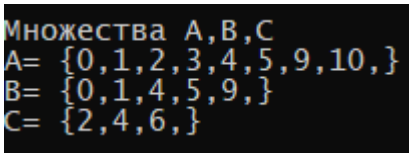


Рис. 6. Вывод множеств A,B,C

## 2.3 «Решето Эратосфена»

1. Откройте приложение sample_primenumbers.exe.В результате появится окно ниже (рис. 1).

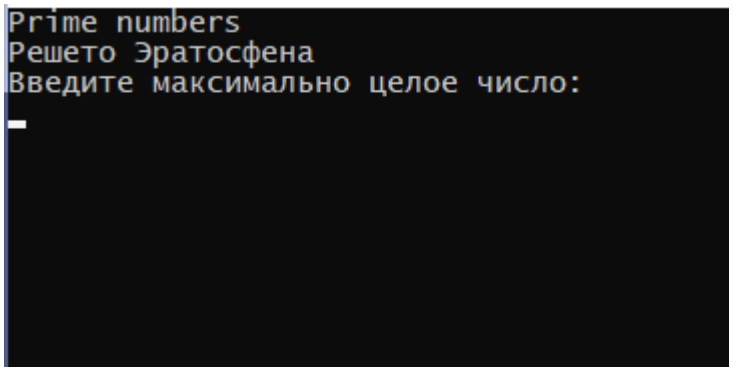


Рис. 1. Окно основной программы

2. Вам будет необходимо ввести число ,до которого будут выведены все простые числа на экран. Для примера введём число 30 и посмотрим на результат(рис. 2)

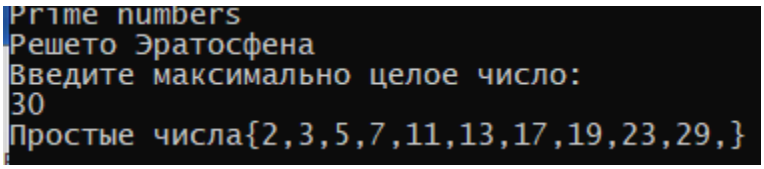


Рис. 2. Все простые числа от 2 до 30

# 3 Руководство программиста

## 3.1 Описание алгоритмов

### 3.1.1 Битовые поля

Битовые поля представляют из себя последовательность нулей и единиц. Элемент битового поля может находиться в двух состояниях: 1 и 0. 1 означает, что элемент находится в состоянии «включен», а 0 – элемент в состоянии «выключен». Приведём пример.

Допустим, имеется битовое поле. Длина битового поля равна 8. В строке index указаны номера элементов от 0 до 7, которые могут находиться в состоянии 0 и 1. В строке bits указаны состояния битов. Например, бит с индексом 1 имеет состояние 0, а бит с индексом 4 – состояние 1.

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| bits | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |

Рассмотрим базовые операции для работы с битовыми полями.

1. Побитовое ИЛИ.

   Операция сложения битовых полей реализуется по следующему принципу. Если один из соответствующих битов битовых полей имеет состояние 1, то результирующее значение равно 1, в противном случае 0.

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Bits1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| Bits2 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |

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

2. Побитовое И.

   Операция умножения битовых полей реализуется по следующему принципу. Если один из соответствующих битов битовых полей имеет состояние 0, то результирующее значение равно 0. Если оба бита равны 1, то результирующее значение 1.

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Bits1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| Bits2 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |

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

3. Отрицание.

   Операция отрицания битовых полей реализуется по следующему принципу. Каждый бит из универса битов меняет своё состояние на противоположное, то есть если бит имел значение 1, то теперь 0. В противном случае состояние бита изменится с 0 на 1.

   Length = 8

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Bits1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |

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

4. Получение битовой маски.

   Битовая маска получается путём сдвига 1 на k бит влево.

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| bits | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |

Бит с номером 3 будет иметь битовую маску, указанную ниже.

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

5. Установить бит в 1.

   Установить бит в 1 означает изменение текущего значения бита на 1. Для этого необходимо получить битовую маску того бита, у которого мы хотим поменять состояние, и выполнить побитовое сложение двух битовых полей.

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| bits | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |

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

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

6. Установить бит в 0.

Установить бит в 0 означает изменение текущего значения бита на 0. Для этого необходимо получить битовую маску того бита, у которого мы хотим поменять состояние, инвертировать её и выполнить побитовое умножение с исходным битовым полем.

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| bits | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |

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

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

7. Получить состояние бита.

Для получения состояния бита необходимо выполнить побитовое умножение битовой маски с битовым полем и полученный результат сдвинуть на количество бит, равное номеру бита, для которого создавалась битовая маска.

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| bits | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |

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

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

8. Получение номера элемента в памяти.

В случае с длинными битовыми полями возникает потребность определить номер элемента в памяти. Для этого необходимо найти целую часть от деления номера

интересующего бита на размер типа данных элемента, который используется для хранения битовых полей.

### 3.1.2 Множества

Множества основываются на битовых полях. Если элемент принадлежит множеству, то в характеристическом массиве бит этого элемента будет находиться в состоянии «включён»,а иначе – «выключен». Приведём в рассмотрение базовые операции над множествами.

Базовые операции для работы с множествами:

1. Включить элемент в множество. Для добавления элемента множества используется операция битовых полей Установить бит в 1.

Пусть дано множество A.

A = 101110

Нужно добавить элемент с номером 5.

A = 101111

В результате элемент с номером 5 будет добавлен.

2. Исключить элемент из множества. Для исключения элемента из множества применяется операция битовых полей Установить бит в 0.

Пусть дано множество A.

A = 101111

Нужно исключить элемент с номером 5.

A = 101111

В результате элемент с номером 5 будет добавлен.

3. Проверка на принадлежность. Чтобы выполнить проверку на принадлежность элемента множеству, нужно получить состояние i-того бита битового поля с помощью операции получения значения бита битового поля. Если бит имеет состояние «включён»,то это означает,что элемент принадлежит множеству,а иначе – не принадлежит.

4. Пусть дано множество A.

A = 101111

Нужно проверить на принадлежность элемента с номером 2 множеству A.

A = 101111

В результате будет известно,что по номеру 2 в памяти бит имеет состояние 1. Поэтому элемент с номером 2 принадлежит множеству.

5. Объединение. Операция объединения основана на операции побитовое ИЛИ двух битовых полей.

Пусть даны множества А и Б.

А = 101111

Б = 111000

В результате операции объединения получим множество С:

С = 111111

6. Пересечение Операция пересечения основана на побитовом И двух битовых полей.

Пусть даны множества А и Б.

А = 101111

Б = 111000

В результате операции пересечения получим множество С:

С = 111111

7. Дополнение базируется на операции побитового инвертирования битового поля.

Пусть дано множество А.

А = 100101

В результате операции инвертирования получим множество С:

С = 011010

8. Разность. Интерпретируется как побитовое И одного битового поля с инвертированным другим битовым полем.

Пусть даны множества А и Б.

А = 101111

Б = 111000

Сначала нужно применить операцию инвертирования к множеству Б:

С = 000111

Далее выполняем побитовое И с множествами А и С:

000111

### 3.1.3 «Решето Эратосфена»

Решето Эратосфена – это алгоритм, позволяющий найти все простые числа до заданного числа n. Суть этого алгоритма заключается в следующем:

1. Выписать подряд все числа от 2 до n
2. Пусть у нас есть переменная p=2 –первое простое число
3. Зачёркиваем все числа,кратные 2p,3p,4p…
4. Находим первое простое число в списке,большее p. Присваиваем его p
5. Повторяем шаги 3 и 4.

Данный алгоритм позволяет легко и быстро найти все простые числа.

## 3.2 Описание программной реализации

### 3.2.1 Описание класса TBitField

```
class TBitField
{
private:
  int  BitLen;
  TELEM *pMem;
  int  MemLen;

  // методы реализации
  int   GetMemIndex(const int n) const;
  TELEM GetMemMask (const int n) const;

  int BitsInMem = 16;
  int shiftsize = 4;

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
  int operator==(const TBitField &bf) const;
  int operator!=(const TBitField &bf) const;
  const TBitField& operator=(const TBitField &bf);
  TBitField  operator|(const TBitField &bf);
  TBitField  operator&(const TBitField &bf);
  TBitField  operator~(void);

  friend istream& operator>>(istream& istr, TBitField& obj);
  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:

```
TELEM GetMemMask (const int n) const;
```
Назначение:

получение индекса элемента памяти

Входные параметры:

n – номер бита

Выходные параметры

номер элемента памяти

```
TELEM GetMemMask (const int n) const;
```
Назначение:

получение битовой маски по номеру бита

входные параметры-

n - номер бита

Выходные параметры:

битовая маска

```
int GetLength(void) const;
```
Назначение:

получение длины битового поля

Входные параметры:

отсутствуют

Выходные параметры:

длина битового поля

```
void SetBit(const int n);
```
Назначение:

установить бит в 1

Входные параметры:

n - номер бита

Выходные параметры:

отсутствуют

int  GetBit(const int n) const;

Назначение:

получение значения бита

Входные параметры:

n - номер бита

Выходные параметры:

значение бита

void ClrBit(const int n);

Назначение:

установить бит в 0

Входные параметры:

n- Номер бита

Выходные параметры:

Отсутствуют

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

Назначение:

вывод битового поля

Входные параметры:

ostream& ostr-ссылка на стандартный поток вывода,

const TBitField& bf  - константная ссылка на битовое поле

Выходные параметры:

битовая строка из 0 и 1

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

Назначение:

ввод битового поля

Входные параметры:

Istream& istr-Ссылка на стандартный поток ввода,

TBitFitField& bf- неконстантная ссылка на битовое поле

Выходные параметры:

битовая строка из 0 и 1

TBitField(int len);

Назначение:

создание битового поля

Входные параметры:

len-длина битового поля

Выходные параметры:

отсутствуют

TBitField(const TBitField &bf);

Назначение:

копирование битовых полей

Входные параметры:

Const TBitField& bf – константная ссылка на битовое поле

Выходные параметры:

отсутствуют

~TBitField();

Назначение:

освобождение памяти

Входные параметры:

отсутствуют

Выходные параметры:

отсутствуют

### 3.2.2 Описание класса TSet

```
class TSet
{
private:
  int MaxPower;       // максимальная мощность множества
  TBitField BitField; // битовое поле для хранения характеристического 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 Elem);       // включить элемент в множество
  void DelElem(const int Elem);       // удалить элемент из множества
  int IsMember(const int Elem) const; // проверить наличие элемента в множестве
```

```
  // теоретико-множественные операции
  int operator== (const TSet &s) const; // сравнение
  int operator!= (const TSet &s) const; // сравнение
  const TSet& operator=(const TSet &s);  // присваивание
  TSet operator+ (const int Elem); // объединение с элементом
                        // элемент должен быть из того же универса
  TSet operator- (const int Elem); // разность с элементом
                        // элемент должен быть из того же универса
  TSet operator+ (const TSet &s);  // объединение
  TSet operator* (const TSet &s);  // пересечение
  TSet operator~ (void);           // дополнение
  TSet operator-(const TSet& obj); //разность

  friend istream &operator>>(istream &istr, TSet &bf);
  friend ostream &operator<<(ostream &ostr, const TSet &bf);
};
```

поля:

**MaxPower – максимальная мощность множества**

**TBitField – битовое поле**

Методы

int GetMaxPower(void) const

Назначение:

получение мощности множества

Входные параметры:

отсутствуют

Выходные параметры:

мощность множества

void InsElem(const int Elem);

Назначение:

добавление элемента в множество

Входные параметры:

Elem- добавляемый элемент

Выходные параметры:

Отсутствуют

void DelElem(const int Elem);

Назначение:

исключение элемента из множества

Входные параметры:

Elem– удаляемый элемент

Выходные параметры:

отсутствуют

int IsMember(const int Elem) const;

Назначение:

проверка на принадлежность

Входные параметры:

Elem – элемент для проверки

Выходные параметры:

значение бита

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

Назначение:

проверка на равенство двух множеств

Входные параметры:

s - множество

Выходные параметры:

целое число

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

Назначение:

проверка на неравенство двух множеств

Входные параметры:

s – множество

Выходные параметры:

целое число

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

Назначение:

присваивание значений полей одного объекта класса другому

Входные параметры:

s – множество

Выходные параметры:

Ссылка на объект своего класса TSet

TSet operator+ (const int Elem);

Назначение:

побитовое ИЛИ множества с элементом

Входные параметры:

Elem- добавляемый элемент

Выходные параметры:

результирующее множество

```
TSet operator- (const int Elem);
```

Назначение:

исключение соответствующего элемента множества

Входные параметры:

Elem – вычитаемый элемент

Выходные параметры:

результирующее множество

TSet operator+ (const TSet &s);

Назначение:

побитовое ИЛИ двух множеств

Входные параметры:

s – множество

Выходные параметры:

результирующее множество

TSet operator* (const TSet &s);

Назначение:

побитовое И двух множеств

Входные параметры:

s – множество

Выходные параметры:

результирующее множество

TSet operator- (const TSet &s);

Назначение:

Вычитание множеств

Входные параметры:

s – множество

Выходные параметры:

результирующее множество

TSet operator~ (void);

Назначение:

ннвертировать значения битов битового поля.

Входные параметры:

отсутствуют

Выходные параметры:

результирующее множество

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

Назначение:

вывод элементов множества в формате({e1,e2,…,en})

Входные параметры:

Ostream& ostr- ссылка на стандартный поток вывода,

Const TSet& s-константная ссылка на объект класса TSet

Выходные параметры:

Строка формата A={e1,e2,…,en}

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

Назначение:

Заполнение множества элементами

Входные

параметры:

Istream& istr – ссылка на поток,TSet& s –ссылка на объект класса TSet

Выходные параметры:

Последовательность введённых элементов множества.

TSet(int mp);

Назначение:

создание множеств

Входные параметры:

mp – мощность множества

Выходные параметры:

отсутствуют

operator TBitField();

Назначение:

преобразование из TSet в TBitField

Входные параметры:

отсутствуют

Выходные параметры:

объект класса TBitField

TSet(const TSet &s);

Назначение:

копирование множеств

Входные параметры:

s – множество

Выходные параметры:

отсутствуют

```
TSet(const TBitField &bf);
```

Назначение:

преобразование из TBitField в TSet

Входные параметры:

bf – Битовое поле

Выходные параметры:

отсутствуют

# Заключение

По результатам лабораторной работы были реализованы классы TSet и TBitField,а также написаны приложения и тесты для проверки работоспособности реализации классов.

# Литература

1. С. 306 Герберт Шилдт «Курс С++» , 845 с.
2. С.64 С.М Наместников «Основы программирования на С++», 136 с.

# Приложения

## Приложение А. Реализация класса TSet

```
TSet::TSet(int mp) : BitField(mp)
{
    MaxPower = mp;
}

// конструктор копирования
TSet::TSet(const TSet &s) : BitField(s.GetMaxPower())
{
    MaxPower = s.GetMaxPower();
    BitField = s.BitField;
}

// конструктор преобразования типа
TSet::TSet(const TBitField &bf): BitField(bf)
{
    MaxPower = bf.GetLength();
    BitField = bf;

}

TSet::operator TBitField()
{
    TBitField obj(BitField);
    return obj;
}

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
{
    if (Elem < 0 || Elem >= MaxPower) {
        throw "element not exist";
    }
    BitField.SetBit(Elem);
}

void TSet::DelElem(const int Elem) // исключение элемента множества
{
    if (Elem < 0 || Elem >= MaxPower) {
        throw "element not exist";
    }
    BitField.ClrBit(Elem);
}


// теоретико-множественные операции

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

```
    MaxPower = s.MaxPower;
    BitField = s.BitField;
    return *this;
}

int TSet::operator==(const TSet &s) const // сравнение
{
    return BitField == s.BitField;
}

int TSet::operator!=(const TSet &s) const // сравнение
{
    return !(BitField == s.BitField);
}

TSet TSet::operator+(const TSet &s) // объединение
{
    if (*this == s) {
        return *this;
    }
    TBitField res(1);
    res = BitField | s.BitField;
    return TSet(res);
}

TSet TSet::operator+(const int Elem) // объединение с элементом
{
    if (Elem < 0 || Elem >= MaxPower) {
        throw "element not exist";
    }
    if (IsMember(Elem)) {
        return TSet(*this);
    }
    TBitField res(BitField);
    res.SetBit(Elem);
    return TSet(res);
}

TSet TSet::operator-(const int Elem) // разность с элементом
{
    if (Elem < 0 || Elem >= MaxPower) {
        throw "element not exist";
    }
    if (!IsMember(Elem)) {
        return TSet(*this);
    }
    TBitField res(BitField);
    res.ClrBit(Elem);
    return TSet(res);
}

TSet TSet::operator-(const TSet& obj) {
    TBitField res(1);
    TBitField inv(obj.BitField);
    res = BitField & (~inv);
    return TSet(res);
}


TSet TSet::operator*(const TSet &s) // пересечение
{
    if (*this == s) {
        return *this;
```

```
    }
    TBitField res(1);
    res = BitField &s.BitField;
    return TSet(res);
}

TSet TSet::operator~(void) // дополнение
{
    TBitField tmp(*this);
    tmp = ~tmp;
    return TSet(tmp);
}

// перегрузка ввода/вывода

istream& operator>>(istream& istr, TSet& bf) {
    unsigned int e = 1;
    size_t count;
    cout << "How many element do you want enter?" << endl;
    cin >> count;
    int i = 0;
    while (i < count) {
        istr >> e;
        bf.InsElem(e);
        i++;
    }
    return istr;
}

ostream& operator<<(ostream &stream, const TSet &obj) // вывод
{
    size_t i, n;
    stream << "{";
    n = obj.MaxPower;
    for (i = 0; i < n; i++) {
        if (obj.IsMember(i)) {


            stream << i << ",";
        }
    }
    stream << "}";
    return stream;
}
```

Пример:

```
int main()
{

 setlocale(LC_ALL, "rus");
 cout << "Создание множеств..." << endl;
 //создание множеств
 TSet A(10 + 1);
 TSet B(20 + 1);
 TSet C(15 + 1);
 TSet copy_B(B);
 TSet res1(1), res2(1), res3(1),res4(1);
 cout << endl;
 cout << "Представление мощностей множеств:" << endl;
 //мощности множеств
 cout << "Мощность множества A = " << A.GetMaxPower() << endl;
 cout << "Мощность множества B = " << B.GetMaxPower() << endl;
```

```
cout << "Мощность множества копии B = " << copy_B.GetMaxPower() << endl;
cout << "Мощность множества C = " << C.GetMaxPower() << endl;
cout << endl;
int choice = 1;

//заполнение множеств
cout << "Ввод элементов множества A:" << endl;
cout << "Введите элементы множества A" << endl;
while (choice == 1) {
	cin >> A;
	cout << "Continue?(1/0)";
	cin >> choice;
}
choice = 1;
cout << endl;
cout << "Ввод элементов множества B:" << endl;
cout << "Введите элементы множества B" << endl;
while (choice == 1) {
	cin >> B;
	cout << "Continue?(1/0)";
	cin >> choice;
}
choice = 1;
cout << endl;
cout << "Ввод элементов множества C:" << endl;
cout << "Введите элементы множества C" << endl;
while (choice == 1) {
	cin >> C;
	cout << "Continue?(1/0)";
	cin >> choice;
}
 //проверка тройного присваивания
/*A = B = copy_B;*/
/*A = copy_B = B;*/

//проверка на равенство множеств
cout << endl;
cout << "Сравнение множеств A и  B: " << endl;
cout << endl;
cout << "Проверка на равенство:" << endl;
if (A == B) {
	cout << "Множества A и B равной мощности" << endl;
}
else {
	cout << "Множества A и B разной мощности" << endl;
}
cout << endl;
//проверка на неравенство множеств
cout << "Проверка на неравенство:" << endl;
if (A != B) {
	cout << "Множества A и B разной мощности" << endl;
}
else {
	cout << "Множества A и B равной мощности" << endl;
}
cout << endl;
//теоретико-множественные операции над множествами
cout << "Выполнение теоретико-множественных операций над множествами: "
<< endl;
cout << endl;
cout << "Объединение множества A с элементом 9" << endl;
A = A + 9;
cout << "A= " << A << endl;
```

```
    cout << endl;
    cout << "Разность множества B с элементом 14" << endl;
    B = B - 14;
    cout << "B= " << B << endl;
    cout << endl;

    cout << "Объединение множества A с множеством B" << endl;
    res1 = A + B;
    cout << "res1= " << res1 << endl;
    cout << endl;

    cout << "Пересечение множества A с множеством C" << endl;
    res2 = A * C;
    cout << "res2= " << res2 << endl;
    cout << endl;

    cout << "Дополнение к множеству C" << endl;
    res3 = ~C;
    cout << "res3= " << res3 << endl;
    cout << endl;

    cout << "Разность множеств A и C" << endl;
    res4 = A - C;
    cout << "res4= " << res4 << endl;
    cout << endl;

    //Множества A,B,C
    cout << "Множества A,B,C" << endl;
    cout << "A= " << A << endl;
    cout << "B= " << B << endl;
    cout << "C= " << C << endl;

       return 0;
}
```

# Приложение Б. Реализация класса TBitField

```
TBitField::TBitField(int len)
{
  if (len < 0) {
    throw "Negative length";
  }
  BitLen = len;
  MemLen = ((len + BitsInMem - 1) >> shiftsize);//количество участков памяти под хранение элементов 1-N
  pMem = new TELEM[MemLen];//создать характеристический массив
  memset(pMem, 0, MemLen * sizeof(TELEM));//заполнить MemLen кусков нулями
}

TBitField::TBitField(const TBitField &obj)
{
  BitLen = obj.BitLen;
  MemLen = obj.MemLen;
  pMem = new TELEM[MemLen];
  memcpy(pMem, obj.pMem, sizeof(TELEM) * MemLen);
}

TBitField::~TBitField()
{
  delete[] pMem;
  MemLen = 0;
```

```
    BitLen = 0;
}

int TBitField::GetMemIndex(const int n) const // индекс Мем для бита n
{
    return n >> shiftsize;
}

TELEM TBitField::GetMemMask(const int n) const // битовая маска для бита n
{
    return 1 << (n & (BitsInMem-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 < 0 || n >= BitLen) {
        throw "Negative length";
    }
    pMem[GetMemIndex(n)] = pMem[GetMemIndex(n)] | GetMemMask(n);
}

void TBitField::ClrBit(const int pos) // очистить бит
{
    if (pos < 0 || pos >= BitLen) {
        throw "Negative length";
    }
    pMem[GetMemIndex(pos)] = pMem[GetMemIndex(pos)] & ~GetMemMask(pos);
}

int TBitField::GetBit(const int n) const // получить значение бита
{
    if (n < 0 || n >= BitLen) {
        throw "Negative length";
    }
    int test = pMem[GetMemIndex(n)] & GetMemMask(n);
    return (pMem[GetMemIndex(n)] & GetMemMask(n) );
}

// битовые операции

const TBitField& TBitField::operator=(const TBitField &bf) // присваивание
{
    if (this == &bf) {
        return *this;
    }

    BitLen = bf.BitLen;
    if (MemLen != bf.MemLen) {
        MemLen = bf.MemLen;
        TELEM* p = new TELEM[MemLen];
        delete[] pMem;
        pMem = p;
    }
    memcpy(pMem, bf.pMem, MemLen * sizeof(TELEM));
    return *this;
}
```

```
int TBitField::operator==(const TBitField &bf) const // сравнение
{
   if (BitLen != bf.BitLen) {
      return false;
   }
   for (size_t i = 0; i < MemLen; i++) {
      if (pMem[i] != bf.pMem[i]) {
         return false;
      }
   }
   return true;
}



int TBitField::operator!=(const TBitField &bf) const // сравнение
{
   if (BitLen != bf.BitLen) {
      return true;
   }
   for (size_t i = 0; i < MemLen; i++) {
      if (pMem[i] == bf.pMem[i]) {
         return false;
      }
   }
   return true;
}

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


      if (BitLen != bf.BitLen) {
         TELEM* p = new TELEM[bf.MemLen];
         memset(p, 0, bf.MemLen * sizeof(TELEM));
         memcpy(p, pMem, MemLen * sizeof(TELEM));
         delete[] pMem;
         BitLen = bf.BitLen;
         MemLen = bf.MemLen;
         pMem = p;
      }
      TBitField tmp(*this);
      for (size_t i = 0; i < bf.MemLen; i++) {
         tmp.pMem[i] = tmp.pMem[i] | bf.pMem[i];
      }
      return tmp;

}

TBitField TBitField::operator&(const TBitField &bf) // операция "и"
{
   if (BitLen != bf.BitLen) {
      TELEM* p = new TELEM[bf.MemLen];
      memset(p, 0, bf.MemLen * sizeof(TELEM));
      memcpy(p, pMem, bf.MemLen * sizeof(TELEM));
      delete[] pMem;
      BitLen = bf.BitLen;
      MemLen = bf.MemLen;
      pMem = p;
   }
   TBitField tmp(*this);
   for (size_t i = 0; i < bf.MemLen; i++) {
      tmp.pMem[i] = tmp.pMem[i] & bf.pMem[i];
   }
```

```
    return tmp;
}

TBitField TBitField::operator~(void) // отрицание
{
    TBitField tbf = (*this);
    for (int i = 0; i < BitLen; i++)
    {
        if (tbf.GetBit(i))
            tbf.ClrBit(i);
        else
            tbf.SetBit(i);
    }
    return tbf;
}

// вывод
ostream &operator<<(ostream &ostr, const TBitField &bf) // вывод
{
    for (int i = 0; i < bf.BitLen; i++) {
        if (bf.GetBit(i)) {
            ostr << "1";
        }
        else { ostr << "0"; }
    }
    return ostr;
}

//Ввод
istream& operator>>(istream& istr, TBitField& obj) {
    string BitField;
    istr >> BitField;

    for (int i = 0; i < BitField.length(); i++) {
        if (BitField[i] == '1') {
            obj.SetBit(i);
        }
        else {
            obj.ClrBit(i);
        }
    }
    return istr;
}
```

ПРИМЕР:

```
int main()
{
 setlocale(LC_ALL, "rus");
 cout << "Создание битовых полей..." << endl;
 TBitField bf(20 + 1);//оригинал
 TBitField copy_bf(bf);//копия
 TBitField bf2(17 + 1);
 TBitField res1(1), res2(2), res3(3);
 cout << endl;
 cout << "Битовое поле bf: " <<      "     "<< bf << endl;
 cout << "Битовое поле copy_bf: " << copy_bf << endl;
 cout << "Битовое поле bf2: " << "      " << bf2<< endl;
 cout << endl;

 cout << "Длина битового поля bf = " << bf.GetLength() << endl;
```

```
cout << endl;

cout << "Заполните битовое поле bf: " << endl;
cin >> bf;
cout << "Установка битового поля bf..." << endl;
cout << "Битовое поле bf: " << bf << endl;
cout << endl;

//проверка бита на принадлежность
bool status_bit = bf.GetBit(16);
cout << endl;

cout << "Заполните битовое поле bf2: " << endl;
cin >> bf2;
cout << "Установка битового поля bf2..." << endl;
cout << "Битовое поле bf2: " << bf2 << endl;
cout << endl;

cout << "Очистка 4-ого бита битового поля bf..." << endl;
bf.ClrBit(4);//установить 4-ый бит в 0.
cout << "Состояние 4-ого бита bf: " << bf.GetBit(4) << endl;
cout << "Битовое поле bf: " << bf << endl;
cout << endl;

if (bf == bf2) {
	cout << "Битовые поля bf и bf2 одинаковы" << endl;
}
else {
	cout << "Битовые поля  bf и bf2 различны" << endl;
}

if (bf != bf2) {
	cout << "Битовые поля bf и bf2 различны" << endl;
}
else {
	cout << "Битовые поля bf и bf2 одинаковы" << endl;
}
cout << endl;

////проверка на операции к себе
//bf = bf | bf;
//copy_bf = copy_bf & copy_bf;
//bf2 = ~bf2;

cout << "Выполнение операций над битовыми полями: " << endl;
cout << endl;
//проверка операций
cout << "operator &: bf и bf2" << endl;
res1 = bf & bf2;
cout << res1 << endl;
cout << endl;
cout << "operator | bf2 или bf" << endl;
res2 = bf2 | bf;
cout << res2 << endl;
cout << endl;
cout << "operator ~ не bf" << endl;
res3 = ~bf;
cout << res3 << endl;

//проверка тройного присваивания
/*bf = bf2 = copy_bf;*/
cout << endl;
cout << "Результаты операций: " << endl;
```

```
 //вывод на экран
 cout << "Битовое поле res1: " << res1 << endl;
 cout << "Битовое поле res2: " << res2 << endl;
 cout << "Битовое поле res3: " << res3 << endl;

    return 0;
}
```