МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
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 ___
_____ / Коробейников А.П./
Подпись

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

Нижний Новгород
2023

`

Содержание

`

Введение

В задачах и программных системах довольно часто требуется работать с множествами, однако не во всех язык программирования предусмотрен класс Множество. В стандартной библиотеке языка C++ есть класс Set и Unordered Set. Класс Set создан для работы с уникальными упорядоченными элементами, порядок элементов является избыточным условием для множества, поэтому он нам не подходит. В класс Unordered Set порядок не учитывается, но в нём не предусмотрены теоретико-множественные операции, что тоже нам не подходит. Поэтому для работы с множествами нужно разработать новый класс, который будет подходить нам по всем критериям.

`

1 Постановка задачи

Цель – Разработать класс для работы с множествами, на основе битовых полей.

Задачи:

1. Изучить понятие битовое поле и всю теорию по работе с битовыми полями.

2. Придумать каким способом связать битовое поле с множеством.

3. Изучить теоретико-множественные операции.

4. Разработать класс Битовое поле.

5. Написать тесты, проверяющие корректность работы каждого метода класса Битовое поле.

6. Написать образец, демонстрирующий работу класса Битовое поле.

7. Написать класс Множество, на основе класса Битовое поле.

8. Написать тесты, проверяющие корректность работы каждого метода класса Множество.

9. Написать образец, демонстрирующий работу класса Битовое поле.

10. Реализовать алгоритм «Решето Эратосфена», в котором будет применяться класс Множество.

`

2 Руководство пользователя

2.1 Приложение для демонстрации работы битовых полей

1. Запустите приложение с названием sample_tbitfield.exe. В результате появится окно, показанное ниже (**Error! Reference source not found.**).

Рис. 1. Основное окно программы sample_tbitfield.exe

2. В данном окне приведены примеры двух битовых полей a и b, а также операции с ними и множествами, полученными из них, которые демонстрируют работу класса Битовое поле.
3. Дальше идёт проверка ввода битового поля, вам нужно ввести любое битовое поле длины 8. В результате появится окно, показанное на (Рис. 2)

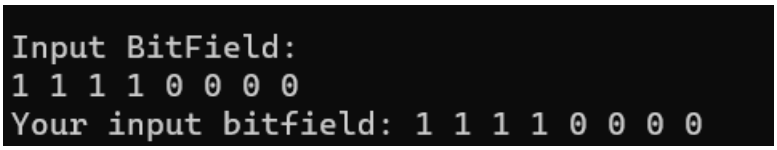


Рис. 2. Окно программы sample_tbitfield.exe, с выводов введённого поля.

4. В этом окне выводится ваше поле, чтобы проверить корректность ввода битового поля.

2.2 Приложение для демонстрации работы множеств

1. Запустите приложение с названием sample_tset.exe. В результате появится окно, показанное ниже (Рис. 3).

`

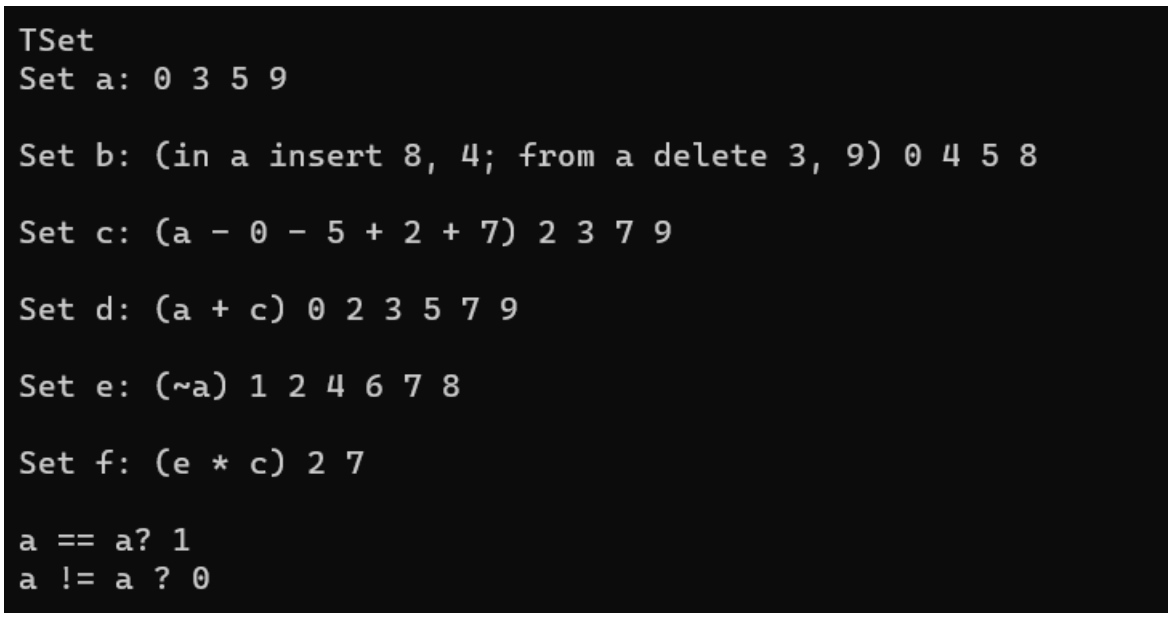


Рис. 3. Основное окно программы sample_tset.exe

2. На данном рисунке приведён пример множества, а затем полученные из него разными способами другие множества, демонстрируя корректность работы методов и операций множества. То какими способами они получены видно на рисунке.
3. В нижней части рисунка показан результат сравнений множества с самим собой.

2.3 «Решето Эратосфено»

1. Запустите приложение с названием sample_primenumbers.exe. В результате появится окно, показанное ниже (Рис. 4).

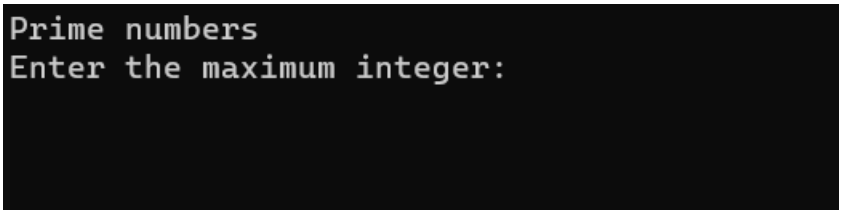


Рис. 4.Основное окно программы sample_primenumbers.exe

2. Далее вам нужно ввести любое натуральное число, в результате чего появиться окно, показанное ниже (в окне приведён пример работы, если ввести число 100) (Рис. 5).

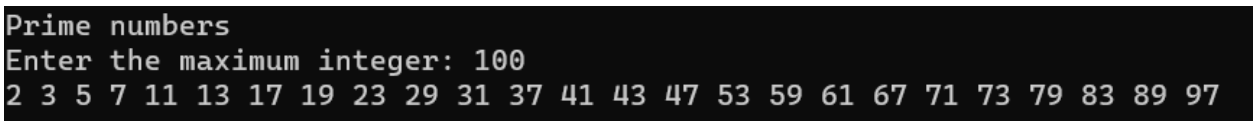


Рис. 5.Окно программы sample_primenumbers.exe если ввести число 100

3. В результате выведутся все просты числа, которые меньше указанного вами

`

`

3 Руководство программиста

3.1 Описание алгоритмов

3.1.1 Битовые поля

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

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

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

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

Первое битовое поле	1	0	0	1	1	0	1	0
|								
Второе битовое поле	1	1	0	0	1	0	1	0
=								
Результат	1	1	0	1	1	0	1	0

`

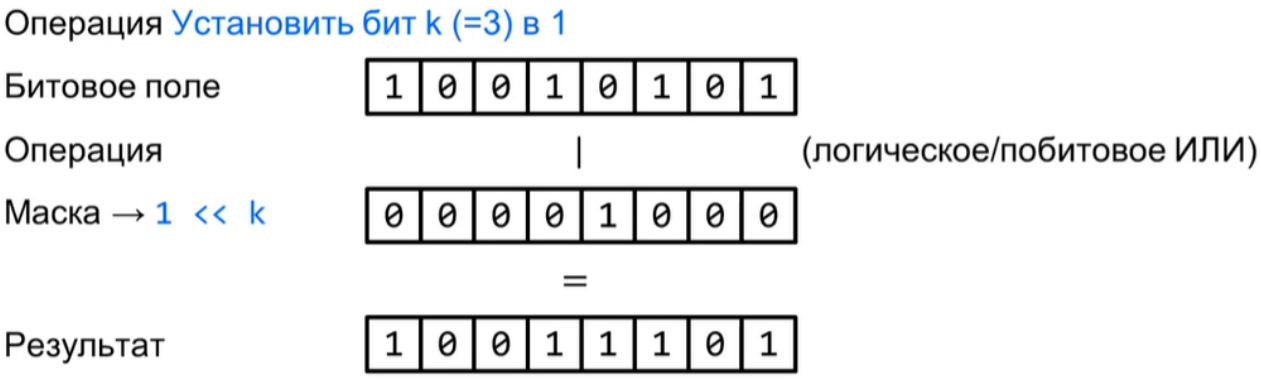
Бинарная операция «Логическое и». Каждый бит одного битового поля производит конъюнкцию с каждым битом другого битового поля:

Первое битовое поле	1	0	0	1	1	0	1	0
&								
Второе битовое поле	1	1	0	0	1	0	1	0
=								
Результат	1	0	0	0	1	0	1	0

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

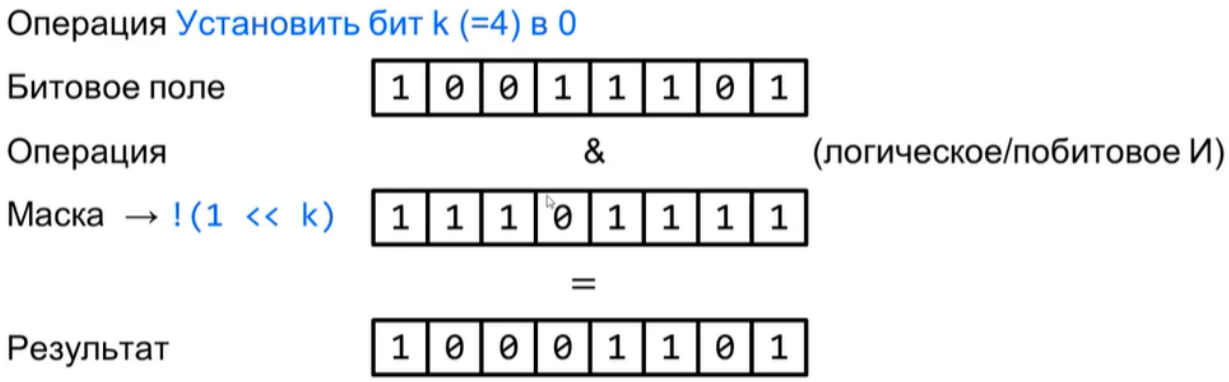
~	1	0	0	1	1	0	1	0
=								
Результат	0	1	1	0	0	1	0	1

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



`

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



3.1.2 Множества

Чтобы определить множество, достаточно указать характеристическое свойство его элементов, т.е. такое свойство, которым обладают все элементы этого множества и только они. (Определение из Большой Российской Энциклопедии).

В нашей лабораторной работе множеством может служить любой счётный набор каких-либо элементов, которые можно проиндексировать (элементов должно быть меньше чем (2^{32} - 1)). И обращаться к элементам множества мы будем именно через индексы, с помощью операций битовых полей.

Некоторые факты из теории множеств:

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

`

Теоретико-множественные операции применимые к множествам:

Операция «Объединение». В результате этой операции получаем множество, в котором есть элементы и первого, и второго множества:

A = {1, 2, 3, 10, 30, 33, 55, 67, 72, 81, 99, 100}

B = {1,3,5, 11, 67, 72, 98}

Результат объединения множеств A\/B:

A\/B = {1, 2, 3, 5, 10, 11, 30, 33, 55, 67, 72, 81, 98, 99, 100}

Операция «Пересечение». В результате этой операции получаем множество, в котором есть только те элементы, которые принадлежат и первому, и второму множеству:

A = {1, 2, 3, 10, 30, 33, 55, 67, 72, 81, 99, 100}

B = {1,3,5, 11, 67, 72, 98}

Результат объединения множеств A/\B:

A/\B = {1, 3, 67, 72}

Операция «Дополнение к множеству». В результате этой операции мы получаем множество, в котором будут только те элементы, которых не было в заданном множестве, но есть в унивёрсе:

U = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

A = {1, 3, 5, 7, 9}

Реузльтат операции «Дополнение к множеству» A:

$\overline{A}$ = {2, 4, 6, 8, 10}

3.1.3 «Решето Эратосфена»

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

1. Выписать подряд все целые числа от двух до n (2, 3, 4, …, n).

2. Пусть переменная p изначально равна двум — первому простому числу.

3. Зачеркнуть в списке числа от 2p до n, считая шагами по p (это будут числа, кратные p: 2p, 3p, 4p, …).

4. Найти первое незачёркнутое число в списке, большее чем p, и присвоить значению переменной p это число.

5. Повторять шаги 3 и 4, пока возможно.

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

`

3.2 Описание программной реализации

3.2.1 Описание класса TBitField

```
class TBitField
{
private:
  int  BitLen;
  TELEM *pMem;
  int  MemLen;
  int bitsInElem = sizeof(TELEM) * 8;
  int shiftSize = ceil( log2(bitsInElem) );


  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.

bitsInElem – количество битов в элементе памяти.

shiftSize – вспомогательное значение для битового целочисленного деления.

Конструкторы:

```
TBitField(int len);
```

Назначение: выделение и инициализация памяти объекта.

Входные параметры: **len** – количество доступных битов.

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

Назначение: выделение памяти и копирование данных.

Входные параметры: **const TBitField &bf** – константная ссылка на битовое поле.

`

Деструктор:

```
~TBitField();
```

Назначение: освобождение выделенной памяти.

Методы:

```
int GetMemIndex(const int n) const;
```

Назначение: получение индекса элемента в памяти.

Входные параметры: **n** – номер бита.

Выходные параметры: индекс элемента в памяти.

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

Назначение: получение маски бита.

Входные параметры: **n** – номер бита.

Выходные параметры: маска бита.

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

Назначение: получение количества доступных битов.

Выходные параметры: **BitLen** – количество доступных битов.

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

Назначение: установить бит в 1.

Входные параметры: **n** – номер бита.

```
void ClrBit(const int n);
```

Назначение: установить бит в 0.

Входные параметры: **n** – номер бита.

```
int GetBit(const int n) const;
```

Назначение: получение значения бита.

Входные параметры: **n** – номер бита.

Выходные параметры: значение бита (0 или 1).

Операции:

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

Назначение: перегрузка операции сравнения на равенство объектов.

Входные параметры: **bf** – битовое поле – объект класса **TBitField.**

Выходные параметры: целое число (0 или 1).

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

Назначение: перегрузка операции сравнения на неравенство объектов.

Входные параметры: **bf** – битовое поле – объект класса **TBitField.**

Выходные параметры: целое число (0 или 1).

`

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

Назначение: присвоение значение полей одного битового поля другому.

Входные параметры: **bf** – битовое поле – объект класса **TBitField.**

Выходные параметры: константная ссылка на объект класса **TBitField.**

```
TBitField operator|(const TBitField &bf);
```

Назначение: сложение двух битовых полей.

Входные параметры: **bf** – битовое поле – объект класса **TBitField.**

Выходные параметры: результирующее битовое поле.

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

Назначение: умножение двух битовых полей.

Входные параметры: **bf** – битовое поле – объект класса **TBitField.**

Выходные параметры: результирующее битовое поле.

```
TBitField  operator~(void);
```

Назначение: инвертирование значений битов битового поля.

Входные параметры: **bf** – битовое поле – объект класса **TBitField.**

Выходные параметры: результирующее битовое поле.

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

Назначение: ввод данных.

Входные параметры: **istr** – поток ввода, **bf** – битовое поле – объект класса **TBitField.**

Выходные параметры: поток ввода.

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

Назначение: вывод данных.

Входные параметры: **ostr** – поток вывода, **bf** – битовое поле – объект класса **TBitField.**

Выходные параметры: поток вывода.

3.2.2 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;
  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:

Max Power – мощность множества.

BitField – характеристический вектор, битовое поле.

Конструкторы:

TSet(int mp);

Назначение: инициализация битового поля.

Входные параметры: **mp** – количество элементов в универсе, мощность множетсва.

TSet(const TSet &s);

Назначение: копирование данных из другого множества.

Входные параметры: s – множество, объект класса TSet.

TSet(const TBitField &bf);

Назначение: преобразование из **TBitField** в **TSet**.

Входные параметры: **bf**– битовое поле, объект класса **TBitField**.

operator TBitField();

Назначение: преобразование из **TSet** в **TBitField**.

Выходные параметры: объект класса **TBitField**.

Методы:

int GetMaxPower(void) const;

Назначение: получение мощности множества.

Выходные параметры: **MaxPower** – мощность множества.

void InsElem(const int Elem);

Назначение: добавление элемента в множество.

Входные параметры **Elem** – добавляемый элемент.

void DelElem(const int Elem);

`

Назначение: удаление элемента из множества.

Входные параметры **Elem** – удаляемый элемент.

```
int IsMember(const int Elem) const;
```

Назначение: проверка на принадлежность элемента множеству.

Входные параметры **Elem** – проверяемый элемент.

Выходные параметры: значение бита (о или 1).

Операции:

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

Назначение: перегрузка операции сравнения, проверка на равенство двух множеств.

Входные параметры: **s** – множество – объект класса **TSet.**

Выходные параметры: целое число (0 или 1).

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

Назначение: перегрузка операции сравнения, проверка на неравенство двух множеств.

Входные параметры: **s** – множество – объект класса **TSet.**

Выходные параметры: целое число (0 или 1).

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

Назначение: присвоение значений полей одного объекта класса другому.

Входные параметры: **s** – множество – объект класса **TSet.**

Выходные параметры: ссылка на объект своего класса **TSet.**

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

Назначение: добавление элемента в множество.

Входные параметры: **Elem** – индекс элемента.

Выходные параметры: результирующее множество, объект класса **TSet.**

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

Назначение: удаление элемента из множества.

Входные параметры: **Elem** – индекс элемента.

Выходные параметры: результирующее множество, объект класса **TSet.**

```
TSet operator+ (const TSet &s);
```

Назначение: объединение двух множеств.

Входные параметры: **S** – объект класса **TSet.**

Выходные параметры: результирующее множество, объект класса **TSet.**

```
TSet operator* (const TSet &s);
```

Назначение: пересечение двух множеств.

Входные параметры: **S** – объект класса **TSet.**

Выходные параметры: результирующее множество, объект класса **TSet.**

`

`TSet operator~ (void);`

Назначение: получение дополнения к множеству.

Выходные параметры: результирующее множество, объект класса TSet.

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

Назначение: ввод данных.

Входные параметры: **`istr`** – поток ввода, **`bf`** – объект класса **`TSet`**.

Выходные параметры: поток ввода.

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

Назначение: вывод данных.

Входные параметры: **`ostr`** – поток вывода, **`bf`** – объект класса **`TSet`**.

Выходные параметры: поток вывода.

`

Заключение

Я изучил всю теорию необходимую для реализации классов Битовое поле и Множество. Разработал класс Битовое поле, который прошёл все тесты, написанные моим преподавателем А. В. Сысоевым. Затем написал образец для класса Битовое поле, который демонстрирует работу всех методов и операций класса. Был разработан класс множество, в который был агрегирован класс Битовое поле. Класс множество также прошёл все тесты. Был написано образец, демонстрирующий работу класса Множество. Также я написал алгоритм «Решето Эратосфена», в котором ключевым элементом является множество, который также показывает корректную работу написанного класса. Цель выполнена, я написал класс необходимый для работы с множествами, который удовлетворяет нашим критериям (неупорядоченность элементов, не нужно хранить информацию об элементах, количество элементов меньше чем (2^{32} - 1)).

`

Литература

1. Лекция «Множества и поля» Сысоев А.В [https://cloud.unn.ru/s/DLRHnt54ircG2WL].
2. Примеры битовых полей в структурах [https://www.bestprog.net/ru/2021/11/28/c-bit-fields-in-structures-ru].
3. Фукнция bit_width [https://learn.microsoft.com/ru-ru/cpp/standard-library/bit-functions?view=msvc-170#bit_width]
4. Функции округления в C++ [https://proginfo.ru/round/]
5. Википедия «Решето Эратосфена» [https://ru.wikipedia.org/wiki/Решето_Эратосфена#Алгоритм]

`

Приложения

Приложение А. Реализация класса TBitField

```
#include "tbitfield.h"
#include  <math.h>

TBitField::TBitField(int len)
{
    if (len <= 0) {
        throw "length < 0";
    }
    BitLen = len;
    MemLen = (BitLen + bitsInElem - 1) >> shiftSize;
    pMem = new TELEM[MemLen];
    memset(pMem, 0, MemLen * sizeof(TELEM));
}

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

TBitField::~TBitField()
{
    delete[]pMem;
}

int TBitField::GetMemIndex(const int n) const   // индекс Мем для бита n
{
    if (n < 0) {
        throw "n < 0";
    }
    return (n >> shiftSize);
}

TELEM TBitField::GetMemMask(const int n) const // битовая маска для бита
{
    if (n < 0) {
        throw "n < 0";
    }
    return (1 << ( (n & (bitsInElem - 1) )));

}

// доступ к битам битового поля

int TBitField::GetLength(void) const // получить длину (к-во битов)
{
  return BitLen;
}

void TBitField::SetBit(const int n) // установить бит
{
    if (n < 0) {
        throw "n < 0";
    }
    if (n >= BitLen) {
        throw "n >= BitLen";
```

```
    `

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

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

int TBitField::GetBit(const int n) const // получить значение бита
{
    if (n < 0) {
        throw "n < 0";
    }
    if (n >= BitLen) {
        throw "n > 0";
    }
    return ((pMem[GetMemIndex(n)] & (GetMemMask(n))) >> (n & (bitsInElem -
1)));
}

// битовые операции

TBitField& TBitField::operator=(const TBitField &bf) // присваивание
{
    if (this == &bf) {
        return(*this);
    }
    if (BitLen != bf.BitLen) {
        BitLen = bf.BitLen;
        MemLen = bf.MemLen;
        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) {
        throw "Non Equal size";
    }
    for (int i = 0; i < MemLen; i++) {
        if (pMem[i] != bf.pMem[i]) {
            return 0;
        }
    }
    return 1;
}

int TBitField::operator!=(const TBitField &bf) const // сравнение
{
    if (BitLen != bf.BitLen) {
        throw "Non Equal size";
    }
```

```
    `

    for (int i = 0; i < MemLen; i++) {
        if (pMem[i] != bf.pMem[i]) {
            return 1;
        }
    }
    return 0;
}

TBitField TBitField::operator|(const TBitField &bf) // операция "или"
{
    TBitField tmp(max(BitLen, bf.BitLen));
    for (int i = 0; i < BitLen; i++) {
        if (GetBit(i)) {
            tmp.SetBit(i);
        }
    }
    for (int 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"
{
    TBitField tmp(max(BitLen, bf.BitLen));
    for (int i = 0; i < min(MemLen, bf.MemLen); i++) {
        tmp.pMem[i] = pMem[i] & bf.pMem[i];
    }
    return (tmp);
}

TBitField TBitField::operator~(void) // отрицание
{
    TBitField tmp(BitLen);

    for (int i = 0; i < BitLen; i++) {
        if (GetBit(i)) {
            tmp.SetBit(i);
        }
    }

    for (int i = 0; i < BitLen; i++) {
        if (tmp.GetBit(i)) {
            tmp.ClrBit(i);
        }
        else {
            tmp.SetBit(i);
        }
    }
    return(tmp);
    throw "Method is not implemented";
}

// ввод/вывод

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

    std::cout << "Input BitField: \n";
    int tmp;
    for (int i = 0; i < bf.BitLen; i++) {
        istr >> tmp;
        if ((tmp != 0) && (tmp != 1)) {
```

```
    `
            throw "The bit cannot take such a value";
        }

        if (tmp == 0) {
            bf.ClrBit(i);
        }
        else {
            bf.SetBit(i);
        }
    }
    return istr;
}

ostream &operator<<(ostream &ostr, const TBitField &bf) // вывод
{
    for (int i = 0; i < bf.BitLen; i++) {
        ostr << bf.GetBit(i) << ' ';
    }
    ostr << endl;
    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.MaxPower;
}

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

TSet::operator TBitField()
{
    TBitField tmp(MaxPower);
    tmp = BitField;
    return tmp;
}

int TSet::GetMaxPower(void) const noexcept // 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
{
    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
{
    if (this == &s) {
        return(*this);
    }
    MaxPower = s.MaxPower;
    BitField = s.BitField;
}

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

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

TSet TSet::operator+(const TSet &s) // объединение
{
    TSet tmp(max(MaxPower, s.MaxPower));
    tmp.BitField = BitField | s.BitField;
    return tmp;
}

TSet TSet::operator+(const int Elem) // объединение с элементом
{
    TSet tmp(MaxPower);
    tmp.BitField = BitField;
    tmp.BitField.SetBit(Elem);
    return tmp;
}

TSet TSet::operator-(const int Elem) // разность с элементом
{
    TSet tmp(MaxPower);
    tmp.BitField = BitField;
    if (!tmp.BitField.GetBit(Elem)) {
        throw "Elem is missing in the set";
    }
    tmp.BitField.ClrBit(Elem);
    return tmp;
}
```

`

```
TSet TSet::operator*(const TSet &s) // пересечение
{
    TSet tmp(max(MaxPower, s.MaxPower));
    tmp.BitField = BitField & s.BitField;
    return tmp;
}

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

// перегрузка ввода/вывода

istream& operator>>(istream& istr, TSet& s) // ввод
{
    for (int i = 0; i < s.MaxPower; i++) {
        s.DelElem(i);
    }
    cout << "Input your set (To finish, enter -1)" << endl;
    int i;

    while (1) {
        cin >> i;
        if (i == -1) {
            return istr;
        }
        if ((i < 0) || (i > s.MaxPower)) {
            throw "OUTOFRANGE";
        }
        s.InsElem(i);
    }
}

ostream& operator<<(ostream &ostr, const TSet &s) // вывод
{
    for (int i = 0; i < s.MaxPower; i++) {
        if (s.BitField.GetBit(i)) {
            ostr << i << ' ';
        }
    }
    ostr << endl;
    return ostr;
}
```