

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
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	6
2.3 «Решето Эратосфена»	9
3 Руководство программиста	10
3.1 Описание алгоритмов	10
3.1.1 Битовые поля	10
3.1.2 Множества	10
3.1.3 «Решето Эратосфена»	10
3.2 Описание программной реализации	11
3.2.1 Описание класса TBitField	11
3.2.2 Описание класса TSet	13
Заключение	16
Литература	17
Приложения	18
Приложение А. Реализация класса TBitField	18
Приложение Б. Реализация класса TSet.....	21

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 проект Множества, который использует интерфейс битовых полей для реализации работы с теоретико-множественными операциями. Это самый оптимальный вариант, поскольку он даёт нам возможность использовать не всю предоставляемую типом данных память, а только его часть. Обращение к определённому биту позволяет нам узнать его состояние для выполнения конкретной задачи. Например, чтобы проверить элемент на принадлежность множеству в нашем случае. Битовые поля в этом случае играют важную роль.

1 Постановка задачи

Цель – реализовать классы: TSet и TBitField

Задачи:

1. Класс для работы с множествами должен поддерживать эффективное хранение данных.
2. 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.
3. Добавить вспомогательные операции получения бита, маски бита, длины битового поля.
4. 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.
5. 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, показанное ниже (рис. 1).

```
Установка битового поля bf2...
Битовое поле bf2: 011110001000000000

Очистка 4-ого бита битового поля bf...
Состояние 4-ого бита bf: 0
Битовое поле bf: 0110010000000000000001

Битовые поля bf и bf2 различны
Битовые поля bf и bf2 различны

Выполнение операций над битовыми полями:

operator &: bf и bf2
01100000000000000000

operator | bf2 или bf
01111100100000000000

operator ~ не bf
10011011111111111111

Результаты операций:
Битовое поле res1: 01100000000000000000
Битовое поле res2: 01111100100000000000
Битовое поле res3: 10011011111111111111
```

Рис. 1. Основное окно программы

2. На первом шаге создаются 3 битовых поля(рис.2)

```
Битовое поле bf: 000000000000000000000000
Битовое поле copy_bf: 000000000000000000000000
Битовое поле bf2: 000000000000000000000000
```

Рис.2 Создание битовых полей

3. На следующем шаге выполняется установка битового поля bf и выводится его длина(рис.3)

```
Длина битового поля bf = 21
Установка битового поля bf...
Битовое поле bf: 011011000000000000000001
```

Рис.3 Установка битового поля bf с выводом длины

4. Далее выполняется установка битового поля bf2(рис.4)

```
Установка битового поля bf2...
Битовое поле bf2: 01111000100000000000
```

Рис.4 Установка битового поля bf2

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

```
Очистка 4-ого бита битового поля bf...  
Состояние 4-ого бита bf: 0  
Битовое поле bf: 01100100000000000001
```

Рис.5 Удаление 4-го бита битового поля bf2

6. В первой строке проверяется операция равенства битовых полей bf и bf2, а во второй- операция неравенства(рис.6)

```
Битовые поля bf и bf2 различны  
Битовые поля bf и bf2 различны
```

Рис.6 Сравнение битовых полей

7. На данном этапе выполняются различные операции с битовыми полями(рис.7)

```
operator &: bf и bf2  
01100000000000000000  
  
operator | bf2 или bf  
011111001000000000  
  
operator ~ не bf  
100110111111111111
```

Рис.7 Операции над битовыми полями

8. На завершающем шаге выводятся результаты вычислений(рис.8)

```
Битовое поле res1: 01100000000000000000  
Битовое поле res2: 01111100100000000000  
Битовое поле res3: 10011011111111111111
```

Рис.8 Результаты вычислений

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
2
Continue?(1/0)1
4
Continue?(1/0)1
6
Continue?(1/0)1
8
Continue?(1/0)0

Ввод элементов множества B:
Введите элементы множества B
1
Continue?(1/0)1
3
Continue?(1/0)1
5
Continue?(1/0)1
7
Continue?(1/0)1
9
Continue?(1/0)1
11
Continue?(1/0)1
13
Continue?(1/0)0

Ввод элементов множества C:
Введите элементы множества C
1
Continue?(1/0)1
2
Continue?(1/0)1
3
Continue?(1/0)1
4
Continue?(1/0)1
7
Continue?(1/0)1
8
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= {2,4,6,8,9,}

Разность множества B с элементом 14
B= {1,3,5,7,9,11,13,}

Объединение множества A с множеством B
res1= {1,2,3,4,5,6,7,8,9,11,13,16,18,19,}

Пересечение множества A с множеством C
res2= {2,4,8,}

Дополнение к множеству C
res3= {0,5,6,9,10,11,12,13,14,15,}

Разность множеств A и C
res4= {6,9,}

Множества A,B,C
A= {2,4,6,8,9,}
B= {1,3,5,7,9,11,13,}
C= {0,5,6,9,10,11,12,13,14,15,}

```

Рис.1 Окно основной программы

2. По началу множества пустые, так как в них не содержится элементов.

Далее представлены мощности множеств (рис.2)

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

Рис.2 Мощности множеств A, B, C

3. На третьем этапе представлен процесс заполнения множеств элементами, введёнными с клавиатуры (рис.3)

```
Ввод элементов множества A:  
Введите элементы множества A:  
2  
Continue?(1/0)1  
4  
Continue?(1/0)1  
6  
Continue?(1/0)1  
8  
Continue?(1/0)0  
  
Ввод элементов множества B:  
Введите элементы множества B:  
1  
Continue?(1/0)1  
3  
Continue?(1/0)1  
5  
Continue?(1/0)1  
7  
Continue?(1/0)1  
9  
Continue?(1/0)1  
11  
Continue?(1/0)1  
13  
Continue?(1/0)0  
  
Ввод элементов множества C:  
Введите элементы множества C:  
1  
Continue?(1/0)1  
2  
Continue?(1/0)1  
3  
Continue?(1/0)1  
4  
Continue?(1/0)1  
7  
Continue?(1/0)1  
8  
Continue?(1/0)0
```

Рис.3 Заполнение множеств A, B, C

4. В этом случае сравниваются два множества на равенство и неравенство (рис.4)

```
Сравнение множеств A и B:  
Проверка на равенство:  
Множества A и B разной мощности  
Проверка на неравенство:  
Множества A и B разной мощности
```

Рис.4 Сравнение множеств A и B

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


```

Объединение множества A с элементом 9
A= {2,4,6,8,9,}

Разность множества B с элементом 14
B= {1,3,5,7,9,11,13,}

Объединение множества A с множеством B
res1= {1,2,3,4,5,6,7,8,9,11,13,16,18,19,}

Пересечение множества A с множеством C
res2= {2,4,8,}

Дополнение к множеству C
res3= {0,5,6,9,10,11,12,13,14,15,}

Разность множеств A и C
res4= {6,9,}

```

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

- В завершение были выведены множества A,B,C, которые принимали участие в программе(рис.6)

```

Множества A,B,C
A= {2,4,6,8,9,}
B= {1,3,5,7,9,11,13,}
C= {0,5,6,9,10,11,12,13,14,15,}

```

Рис.6 Вывод множеств A,B,C

2.3 «Решето Эратосфена»

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

```

Prime numbers
Решето Эратосфена
Введите максимально целое число:
_

```

Рис.1 Окно основной программы

- 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)

```

Prime numbers
Решето Эратосфена
Введите максимально целое число:
30
Простые числа{2,3,5,7,11,13,17,19,23,29,}

```

- 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 характеристические массивы, где индексы каждого элемента – это элементы множества. Каждое битовое поле задаётся длиной (универс битов), количеством единиц памяти (кол-во характеристических массивов) и памятью для их хранения. Элемент битового поля может находиться в двух состояниях: 1 и 0. 1- элемент содержится в множестве, а 0 – элемент не содержится в множестве. Данный алгоритм позволяет реализовать интерфейс для работы с множествами.

3.1.2 Множества

Множества по идее наследуются от класса TBitField. Множество – это класс TSet, реализованный на основе класса TBitField. Работа TSet заключается в том, что он использует класс TBitField как инструмент для создания множеств и осуществления теоретико-множественных операций. Максимальная мощность множества – это и есть длина битового поля. Таким образом, главная роль отводится классу TBitField, который и отвечает за техническую часть работы множеств.

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 $2p, 3p, 4p, \dots$
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 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	Назначение	Входные параметры	Выходные параметры
GetMemIndex	Получение индекса элемента памяти	n- Номер бита	Номер элемента памяти
GetMemMask	Получение битовой маски по номеру бита	n- Номер бита	Битовая маска
GetLength	Получение длины битового поля	отсутствуют	Длина битового поля
SetBit	Установить бит в единицу	n- Номер бита	отсутствуют
ClrBit	Установить бит в ноль	n- Номер бита	отсутствуют
GetBit	Получение значения бита	n- Номер бита	Значение бита (0 или 1)

Операции	Назначение	Входные параметры	Выходные параметры
Равенство(==) Operator==	Проверка на равенство двух битовых полей	bf - Битовое поле	Целое число (0 или 1)
Неравенство(!=) Operator!=	Проверка на неравенство двух битовых полей	bf - Битовое поле	Целое число (0 или 1)
Присваивание(=) Operator!=	Присвоение значений полей одного битового поля другому	bf - Битовое поле	Константная ссылка на объект своего класса TBitField
Побитовое ИЛИ() Operator	Сложение двух битовых полей	bf - Битовое поле	Результирующее битовое поле
Побитовое И(&) Operator&	Умножение двух битовых полей	bf - Битовое поле	Результирующее битовое поле
Побитовое отрицание(~) Operator~	Инвертирование значений битов битового поля	bf - Битовое поле	Результирующее битовое поле
Вывод(<<) Operator<<	Вывод битового поля в формате(1010101001 и т.п)	Lnk ostream,const lnk TBitField	Константная ссылка на поток

Конструкторы	Назначение	Входные параметры	Выходные параметры
Инициализатор	Создание битового поля	Len- Длина битового поля	отсутствуют
Копирование	Копирование битовых полей	Const TBitField& bf - Константная ссылка на битовое поле	отсутствуют
Деструктор	Освобождение памяти	отсутствуют	отсутствуют

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();
    // доступ к битам
    int GetMaxPower(void) const;
    void InsElem(const int Elem);
    void DelElem(const int Elem);
    int IsMember(const int Elem) const;
    // теоретико-множественные операции
    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);
    TSet operator- (const TSet& obj);

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

```

Битовые поля:

MaxPower – максимальная мощность множества
TBitField – битовое поле

Методы/ Операции	Назначение	Входные параметры	Выходные параметры
GetMaxPower	Получение мощности множества	Отсутствуют	Мощность множества
InsElem	Добавление элемента в множество	Elem- добавляемый элемент	отсутствуют
DelElem	Исключение элемента из множества	Elem – удаляемый элемент	отсутствуют
IsMember	Проверка на принадлежность множеству	Elem – элемент	Значение бита (0 или 1)
Равенство(==) Operator==	Проверка на равенство двух множеств	s - множество	Целое число (0 или 1)
Неравенство(!=) Operator!=	Проверка на неравенство двух множеств	s - множество	Целое число (0 или 1)
Присваивание(=) Operator=	Присвоение значений полей одного объекта класса другому	s - множество	Ссылка на объект своего класса TSet
Объединение с элементом Operator+	Побитовое сложение соответствующего элемента множества с элементом	Elem- добавляемый элемент	Результирующее множество

Пересечение с элементом operator&	Побитовое умножение соответствующего элемента множества с элементом	Elem– добавляемый элемент	Результирующее множество
Пересечение множеств Operator&	Побитовое умножение элементов двух множеств	s - множество	Результирующее множество
Разность с элементом -	Исключение соответствующего элемента множества	Elem – вычитаемый элемент	Результирующее множество
Разность множеств Operator-	Исключение элементов одного множества элементами другого множества	s - множество	Результирующее множество
Дополнение к множеству operator~	Инвертировать значения битового поля. Это и будет дополнение к множеству.	отсутствуют	Результирующее множество
Вывод Operator<<	Вывод элементов множества в формате({e1,e2,...,en})	Lnk ostream,const lnk TSet	Ссылка на поток вывода
Ввод Operator>>	Заполнение множества	Lnk ostream, lnk TSet	Ссылка на поток ввода

Конструкты/ операторы	Предназначение	Входные параметры	Выходные параметры
Инициализатор	Создание множеств	Мр – мощность множества	Отсутствуют
Копирование	Копирование множеств	s - множество	Отсутствуют
Преобразование типа	Преобразование из TBitField в TSet	Bf – Битовое поле	Отсутствуют
Оператор преобразования в тип TBitField	Преобразование из TSet в TBitField	отсутствуют	Объект класса TBitField

Заключение

По результатам лабораторной работы были реализованы классы TSet и TBitField, а также написаны приложения и тесты для проверки работоспособности реализации. К лабораторной работе был составлен полный отчёт по теме со всеми подробными описаниями.

Литература

1. Битовые поля и операции над ними
https://rep.bntu.by/bitstream/handle/data/32621/Bitovye_polya_i_operacii_nad_nimi.pdf?sequence=1&isAllowed=y
2. Битовые поля. Урок 32
<https://narodstream.ru/c-urok-32-bitovye-polya/>
3. Битовые поля
<https://www.c-cpp.ru/books/bitovye-polya>

Приложения

Приложение А. Реализация класса TSet

```
class TSet : 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);
}
```

```

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;
    istr >> e;
    bf.InsElem(e);
    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;
}

```

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 Mem 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
```

```

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 &b) // присваивание
{
    if (*this == b) {
        return *this;
    }

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

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

```

```

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

TBitField TBitField::operator|(const TBitField &b) // операция "или"
{
    if (BitLen != b.BitLen) {
        TLEN* p = new TLEN[b.MemLen];
        memcpy(p, pMem, MemLen * sizeof(TLEN));
        delete[] pMem;
        BitLen = b.BitLen;
        MemLen = b.MemLen;
        pMem = p;
    }
    TBitField tmp(*this);
    for (size_t i = 0; i < b.MemLen; i++) {
        tmp.pMem[i] = tmp.pMem[i] | b.pMem[i];
    }
    return tmp;
}

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

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

```