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»

Выполнила: студентка группы 3822Б1ФИ2
_____ / Резанцева А.А.
Подпись

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

Нижний Новгород
2023

Содержание

Введение

Нужны ли в программировании на C++ множества? Да, они нужны для решения многих задач. Например, множество фильмов в «фильмотеке» конкретного пользователя, купленных в онлайн-кинотеатре или начертание символов в текстовых редакторах: Ж, К, Ч, З, ВИ, НИ, … Таким образом возникает необходимость информацию об объекте, в формате состояний, представляющих из себя 0 и 1. Проект «Множества» использует интерфейс битовых полей для работы с теоретико-множественными операциями. Данный вариант решения задачи дает нам возможность использовать только часть предоставляемой типом данных памяти. Обращение к биту с определенным индексом позволяет нам узнать его состояние. Например, относится ли элемент к данному множеству. Любое множество описывается характеристическим вектором, а наиболее эффективный способ его представить/хранить – битовое поле (битовая строка).

1 Постановка задачи

Цель – реализовать класс битовое поле TBitField и класс множество TSet.

Задачи при реализации класса TBitField:

1. Описать и реализовать конструктор, конструктор копирования, деструктор.
2. Описать и реализовать операции доступа к битам: установить бит в 1, установить бит в 0, получить значение бита, получить количество доступных битов.
3. Описать и реализовать вспомогательные методы: получение индекса элемента, получение маски бита.
4. Перегрузить битовые операции: присваивание (=), сравнение (==, !=), побитовое ИЛИ (|), побитовое И (&), побитовое отрицание (~).
5. Перегрузить операции ввода и вывода.

Задачи при реализации класса TSet:

1. Описать и реализовать конструктор, конструктор копирования, конструктор преобразования типа, оператор преобразования типа к битовому полю.
2. Описать и реализовать операции доступа к битам: включить элемент в множество, удалить элемент из множества, проверить наличие элемента в множестве, получить максимальной мощности множества.
3. Перегрузить теоретико-множественные операции: присваивание (=), сравнение (==, !=), объединение (+), пересечение (*), объединение с элементом из множества (+), разность с элементом из множества (-), дополнение (~).
4. Перегрузить операции ввода и вывода.

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).

```
C:\Users\ITMM-230042\Desktop\mp2-practice\RezantsevaAA\01_lab\sln\bin>sample_tbitfield.exe
TBitField
bf1 = 0 0 1 0 1 1

bf2 = 1 1 0 1 0 1


bf1.SetBit(0): 1 0 1 0 1 1

bf1.ClrBit(0): 0 0 1 0 1 1

~bf1: 1 1 0 1 0 0


bf1.GetBit(0): 0
bf1.GetBit(5): 1

operator==: 0
operator!=: 1
operator|: 1 1 1 1 1 1

operator&: 0 0 0 0 0 1


C:\Users\ITMM-230042\Desktop\mp2-practice\RezantsevaAA\01_lab\sln\bin>
```

Рис. 1. Основное окно программы

2. На первом шаге создаются два битовых поля (Рис. 2).

```
bf1 = 0 0 1 0 1 1

bf2 = 1 1 0 1 0 1
```

Рис. 2. Создание битовых полей

3. На следующем шаге нулевой бит битового поля bf1 устанавливается в 1 (Рис. 3).

```
bf1.SetBit(0): 1 0 1 0 1 1
```

Рис. 3. Установка нулевого бита в 1

4. Далее очищаем нулевой бит битового поля bf1, то есть устанавливаем в 0 (Рис. 4).

```
bf1.ClrBit(0): 0 0 1 0 1 1
```

Рис. 4. Установка нулевого бита в 0

5. На четвертом шаге выполняем операцию отрицания битового боля bf1 (Рис. 5).

```
~bf1: 1 1 0 1 0 0
```

Рис. 5. Отрицание битового поля

6. Далее получаем значение нулевого и пятого бита битового поля bf1 (Рис. 6).

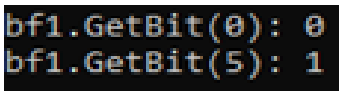

Рис. 6. Получение значение битов на позиции 0 и 5

7. На шестом шаге сравниваем битовые поля bf1 и bf2 (Рис. 7).

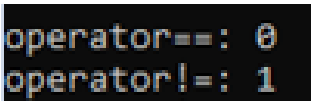

Рис. 7. Операции равенства и неравенства битовых полей

8. Далее выполняются операция «или» и операция «и» для битовых полей bf1 и bf2 (Рис. 8).

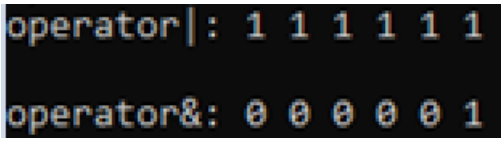

Рис. 8. Операторы «или», «и»

2.2 Приложение для демонстрации работы множеств

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

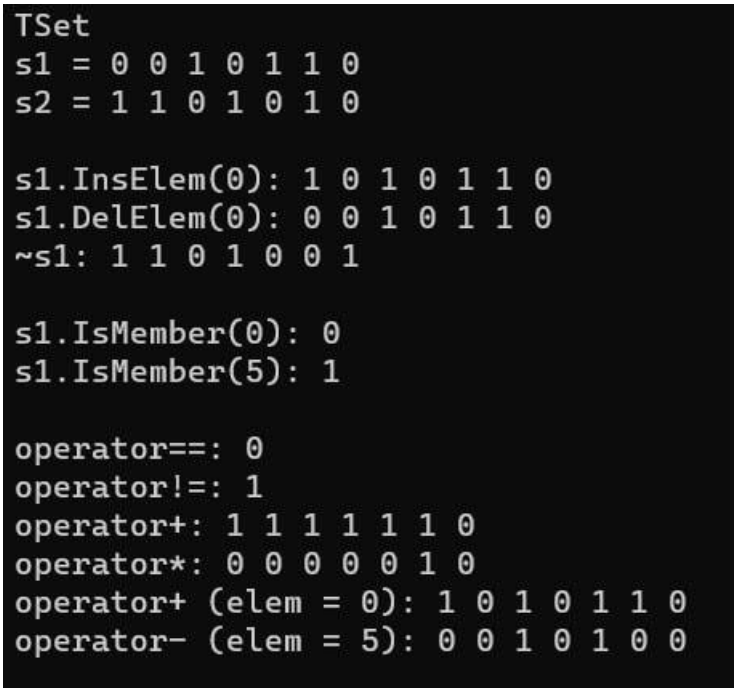

Рис. 9. Основное окно программы

2. На первом шаге создаются множества s1 = {2, 4, 5} и s2 = {0, 1, 3, 5} (Рис. 10).

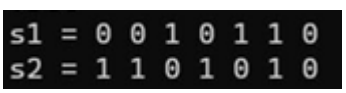

Рис. 10. Множества s1 и s2

3. На втором шаге включаем элемент 0 в множество s1 (Рис. 11).

```
s1.InsElem(0): 1 0 1 0 1 1 0
```

Рис. 11. Включение элемента в множество

4. Далее удаляем элемент 0 из множества s1 (Рис. 12).

```
s1.DelElem(0): 0 0 1 0 1 1 0
```

Рис. 12. Удаление элемента из множества

5. На следующем шаге выводим дополнение множества s1 (Рис. 13).

```
~s1: 1 1 0 1 0 0 1
```

Рис. 13. Дополнение множества

6. Далее проверяем наличие элементов 0 и 5 в множестве s1, где 0 – не является элемнтом множества, 1 – является (Рис. 14).

```
s1.IsMember(0): 0
s1.IsMember(5): 1
```

Рис. 14. Наличие элементов в множестве

7. На шестом шаге сравниваем множества s1 и s2 (Рис. 15).

```
operator==: 0
operator!=: 1
```

Рис. 15. Операции равенства и неравенства множеств

8. Далее идут опирации объединения и пересечения множеств s1 и s2 (Рис. 16).

```
operator+: 1 1 1 1 1 1 0
operator*: 0 0 0 0 0 1 0
```

Рис. 16. Операции объединения и пересечения множеств

9. На последнем этапе выполняются операция объединения элемента 0 с множеством s1 и операция удаления элемента 5 из множества s1 (Рис. 17).

```
operator+ (elem = 0): 1 0 1 0 1 1 0
operator- (elem = 5): 0 0 1 0 1 0 0
```

Рис. 17. Операции объединение элемента с множествои и удаления элемента из множества.

2.3 «Решето Эратосфена»

1. Запустите приложение с названием sample_primenumbers.exe. В результате появится окно, показанное ниже (Рис. 18).

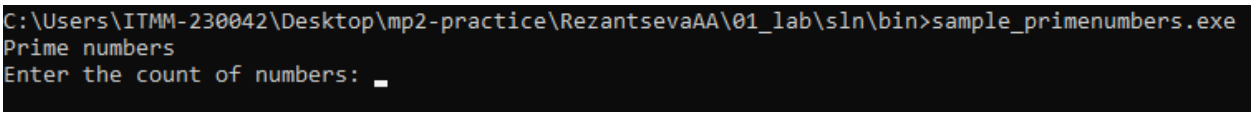

Рис. 18. Запрос программы

2. Введите число, до которого будут выведены все простые числа на экран. Для примера возьмем число 56. Результат появился в окне, показанном ниже (**Ошибка! Источник ссылки не найден.**).

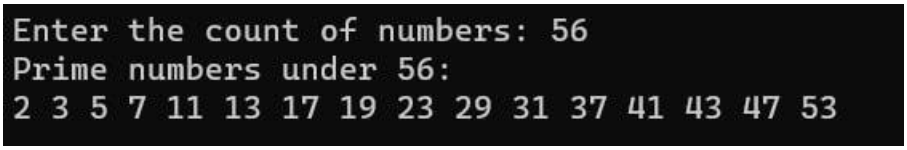

Рис. 19. Простые числа от 2 до 56

3 Руководство программиста

3.1 Описание алгоритмов

3.1.1 Битовые поля

Битовые поля – это характеристические векторы, где индексы каждого элемента – это элементы множества. Битовое поле задается длиной (универс битов), количеством единиц памяти (количество характеристических векторов) и памятью для их хранения. Элемент битового поля может находиться в двух состояниях: 0 – элемент не содержится в множестве, 1 – элемент содержится в множестве.

3.1.2 Множества

Множества — это класс TSet, реализованный на классе битового поля TBitField. Он использует класс TBitField для создания множеств и выполнения теоретико-множественных операций. Максимальная мощность множества будет длиной битового поля.

3.1.3 «Решето Эратосфена»

Решето Эратосфена – это алгоритм, позволяющий найти все простые числа до заданного числа n. Алгоритм:

1. Выписать подряд все числа от 2 до n
2. Пусть переменная p изначально равна двум – первому простому числу
3. Зачеркнуть в списке числа от 2p до n, считая шагами по p (то есть удаляем числа, кратные p).
4. Найти первое не зачеркнутое число в списке, большее чем p, и присвоить значению переменной p это число
5. Повторять шаги 3 и 4, пока возможно

3.2 Описание программной реализации

3.2.1 Описание класса TBitField

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

  int   GetMemIndex(const int n) const;
  TELEM GetMemMask (const int n) const;
public:
  TBitField(int len);
  TBitField(const TBitField &bf);
```

```
 ~TBitField();

  int GetLength(void) const;
  void SetBit(const int n);
  void ClrBit(const int n);
  int GetBit(const int n) const;

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

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

Были реализованы классы: TBitField и TSet:

1. Выполнены задачи при реализации класса TBitField:
   - Описать и реализовать конструктор, конструктор копирования, деструктор.
   - Описать и реализовать операции доступа к битам: установить бит в 1, установить бит в 0, получить значение бита, получить количество доступных битов.
   - Описать и реализовать вспомогательные методы: получение индекса элемента, получение маски бита.
   - Перегрузить битовые операции: присваивание (=), сравнение (==, !=), побитовое ИЛИ (|), побитовое И (&), побитовое отрицание(~).
   - Перегрузить операции ввода и вывода.
2. Выполнены задачи при реализации класса TSet:
   - Описать и реализовать конструктор, конструктор копирования, конструктор преобразования типа, оператор преобразования типа к битовому полю.
   - Описать и реализовать операции доступа к битам: включить элемент в множество, удалить элемент из множества, проверить наличие элемента в множестве, получить максимальной мощности множества.
   - Перегрузить теоретико-множественные операции: присваивание (=), сравнение (==, !=), объединение (+), пересечение (*), объединение с элементом из множества (+), разность с элементом из множества (-), дополнение (~).
   - Перегрузить операции ввода и вывода.

Литература

1. Лекция «Множества и поля» Сысоев А.В [https://cloud.unn.ru/s/DLRHnt54ircG2WL].

Приложения

Приложение А. Реализация класса TBitField

```
#include "tbitfield.h"

TBitField::TBitField(int len)
{
    if (len <= 0)
        throw "len_is_equal_zero!";
    BitLen = len;
    MemLen = (BitLen - 1) / (32) + 1;
    pMem = new TELEM[MemLen];

    for (int i = 0; i < MemLen; ++i)
    {
        pMem[i] = 0;
    }
}

TBitField::TBitField(const TBitField &bf) // конструктор копирования
{
    BitLen = bf.BitLen;
    MemLen = bf.MemLen;
    pMem = new TELEM[MemLen];
    for (int i = 0; i < MemLen; ++i)
    {
        pMem[i] = bf.pMem[i];
    }
}

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

int TBitField::GetMemIndex(const int n) const // индекс Мем для бита n
{
    if (n < 0)
        throw "n_is_below_zero";
    return n / (32);
}

TELEM TBitField::GetMemMask(const int n) const // битовая маска для бита n
{
    if (n > BitLen)
        throw "overload_n";
    if (n < 0)
        throw "n_is_below_zero";
    TELEM Mask = GetBit(n) << (32 - n % 32 * 8 - 1);
    return Mask;
}

// доступ к битам битового поля

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

void TBitField::SetBit(const int n) // установить бит
{
```

```
    if (n > BitLen)
        throw "overload_n";
    if (n < 0)
        throw "n_is_below_zero";

    pMem[GetMemIndex(n)] |= 1u << (32 - n % 32 - 1);
}

void TBitField::ClrBit(const int n) // очистить бит
{
    if (n > BitLen)
        throw "overload_n";
    if (n < 0)
        throw "n_is_below_zero";
    pMem[GetMemIndex(n)] &= ~(1u << (32 - n % 32 - 1));
}

int TBitField::GetBit(const int n) const // получить значение бита
{
    if (n > BitLen)
        throw "overload_n";
    if (n < 0)
        throw "n_is_below_zero";
    return (pMem[GetMemIndex(n)] & (1u << (32 - n % 32 - 1))) >> (32 - n % 32
- 1);
}

// битовые операции

const TBitField& TBitField::operator=(const TBitField &bf) // присваивание
{
    BitLen = bf.BitLen;
    MemLen = bf.MemLen;
    delete[] pMem;
    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.GetLength())
        return 0;
    else
    {
        for (int i = 0; i < BitLen; ++i)
        {
            if (GetBit(i) != bf.GetBit(i))
                return 0;
        }
    }
    return 1;
}

int TBitField::operator!=(const TBitField &bf) const // сравнение
{
    return !(*this == bf);
}

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

```
{
    int maxlen = max(GetLength(), bf.GetLength());
    int minlen = min(GetLength(), bf.GetLength());
    TBitField tmp(1);
    if (GetLength() > bf.GetLength())
        tmp = *this;
    else
        tmp = bf;
    int i = 0;
    for (; i < minlen; ++i)
    {
        if (GetBit(i) || bf.GetBit(i))
            tmp.SetBit(i);
    }
    return tmp;
}

TBitField TBitField::operator&(const TBitField& bf) // операция "и"
{
    int maxlen = max(GetLength(), bf.GetLength());
    int minlen = min(GetLength(), bf.GetLength());
    TBitField tmp(maxlen);
    int i = 0;
    for (; i < minlen; ++i)
    {
        if (GetBit(i) && bf.GetBit(i))
            tmp.SetBit(i);
    }
    return tmp;
}

TBitField TBitField::operator~(void) // отрицание
{
    TBitField tmp(GetLength());
    for (int i = 0; i <= GetMemIndex(GetLength()); ++i)
    {
       tmp.pMem[i] = ~pMem[i];
    }
    return tmp;
}

// ввод/вывод

istream &operator>>(istream &istr, TBitField &bf) // ввод
{
    for (int i = 0; i < bf.GetLength(); ++i)
    {
        int val;
        istr >> val;
        if (val) bf.SetBit(i);
    }
    return istr;
}

ostream &operator<<(ostream &ostr, const TBitField &bf) // вывод
{
    for (int i = 0; i < bf.GetLength(); ++i)
    {
        ostr << bf.GetBit(i) << " ";
    }
    ostr << "\n";
    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.GetMaxPower();
}

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

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

int TSet::GetMaxPower(void) const // получить макс. к-во эл-тов
{
    return MaxPower;
}

int TSet::IsMember(const int Elem) const // элемент множества?
{
    if ((Elem < 0) && (Elem >= MaxPower))
        throw "Out of Range";
    return BitField.GetBit(Elem);
}

void TSet::InsElem(const int Elem) // включение элемента множества
{
    if ((Elem < 0) && (Elem >= MaxPower))
        throw "Out of Range";
    return BitField.SetBit(Elem);
}

void TSet::DelElem(const int Elem) // исключение элемента множества
{
    if ((Elem < 0) && (Elem >= MaxPower))
        throw "Out of Range";
    return BitField.ClrBit(Elem);
}

// теоретико-множественные операции

const TSet& TSet::operator=(const TSet &s) // присваивание
{
    MaxPower = s.GetMaxPower();
    BitField = TBitField(MaxPower);
    BitField = BitField | s.BitField;
    return *this;
}

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

```
{
    if(MaxPower != s.GetMaxPower())
        return 0;
    int minlen = min(MaxPower, s.GetMaxPower());
    for (int i = 0; i < minlen; ++i)
    {
        if (BitField.GetBit(i) != s.BitField.GetBit(i))
            return 0;
    }
    return 1;
}

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

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

TSet TSet::operator+(const int Elem) // объединение с элементом
{
    if ((Elem<0) && (Elem>MaxPower))
        throw "Out of Range";
    TSet tmp(*this);
    tmp.BitField.SetBit(Elem);
    return tmp;
}

TSet TSet::operator-(const int Elem) // разность с элементом
{
    if ((Elem<0) && (Elem>MaxPower))
        throw "Out of Range";
    TSet tmp(*this);
    tmp.BitField.ClrBit(Elem);
    return tmp;
}

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

TSet TSet::operator~(void) // дополнение
{
    TSet tmp(MaxPower);
    tmp.BitField = ~BitField;
    return tmp;
}

// перегрузка ввода/вывода

istream &operator>>(istream &istr, TSet &s) // ввод
{
    const int x = s.MaxPower;
    for (int i = 0; i <= x; ++i)
    {
```

```
        int val; istr >> val;
        s.InsElem(val);
    }
    return istr;
}

ostream& operator<<(ostream &ostr, const TSet &s) // вывод
{
    const int x = s.MaxPower;
    for (int i = 0; i <= x; ++i)
    {
        ostr << s.IsMember(i) << " ";
    }
    return ostr;
}
```