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

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

Подпись

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

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

Подпись

Нижний Новгород  
2023

# Оглавление

[Оглавление 2](#_Toc24853)

[Введение 1](#_Toc16928)

[1 Постановка задачи 3](#_Toc17611)

[2 Руководство пользователя 4](#_Toc17556)

[2.1 Приложение для демонстрации работы битовых полей 4](#_Toc588)

[2.2 Приложение для демонстрации работы множеств 5](#_Toc21665)

[2.3 Приложение «решето Эратосфена» 6](#_Toc13081)

[3 Руководство программиста 8](#_Toc28537)

[3.1 Использованнные алгоритмы 8](#_Toc27251)

[3.2 Описание классов 8](#_Toc740)

[Заключение 14](#_Toc4194)

[Реализация множеств при помощи битовых полей оказалась эффективным подходом, позволяющим компактно хранить и оперировать наборами элементов. Этот метод позволяет сократить затраты на память, обеспечивает быстрые операции с множествами и может быть применён в решении различных задач, связанных с манипуляцией данными и их анализом. 14](#_Toc19111)

[Литература 15](#_Toc32267)

[1. https://ru.wikipedia.org/ 15](#_Toc24061)

[2. https://prog-cpp.ru/ 15](#_Toc11706)

[3. https://en.cppreference.com/w/ 15](#_Toc5624)

[4. https://github.com/ 15](#_Toc9295)

[Приложения 16](#_Toc31684)

[Приложение А. Реализация класса TBitField 16](#_Toc9961)

[Приложение Б. Реализация класса TSet 18](#_Toc9757)

# Введе**н**ие

Битовое поле — это структура данных, позволяющая компактно хранить и работать с набором битов, устанавливать их значения и проверять их. Каждое поле представляет собой группу битов в памяти компьютера, каждому из которых присваивается определённая позиция (0 или 1).

Использование битовых полей даёт большую гибкость, обеспечивает оптимизацию памяти и удобство работы с различными флагами и настройками в программе. Также они позволяют управлять информацией и хранить её в компактном формате. А простота этой структуры обеспечивает ее кросс-платформенность.

Примеры использования битовых полей:

1. Флаги состояний

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

1. Настройки конфигурации

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

1. Сериализация данных

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

1. *Реализация множеств*

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

# Постановка задачи

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

Задачи:

1. Разработка представления битовых полей

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

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

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

1. Определение операций над множествами

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

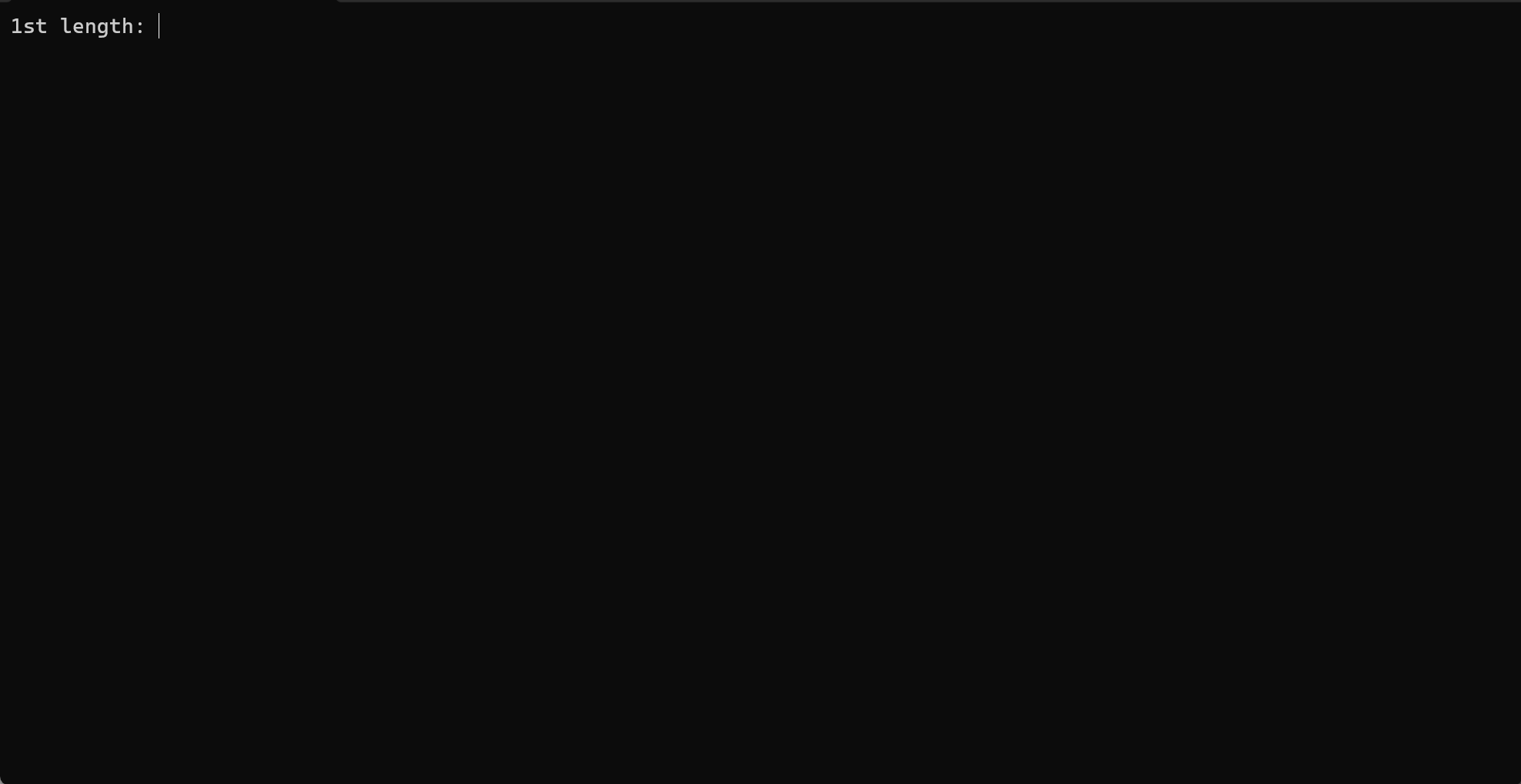
1. Тестирование и отладка.

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

# Руководство пользователя

## Приложение для демонстрации работы битовых полей

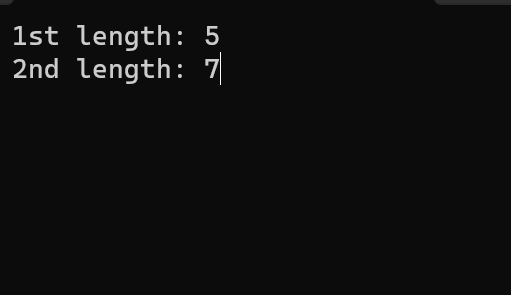
1. Запустить sample\_tbitfield.exe. В результате появится следующее окно (рис. 1):



1. Основное окно приложения

На этом шаге потребуется ввести два числа соответственно: длина первого битового поля, длина второго битового поля

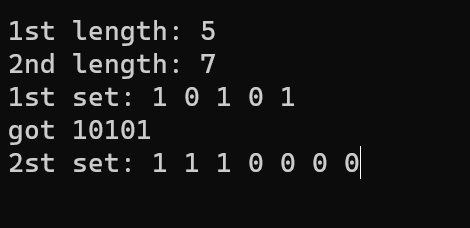
1. После ввода длин полей, следует их заполнить (рис. 2):



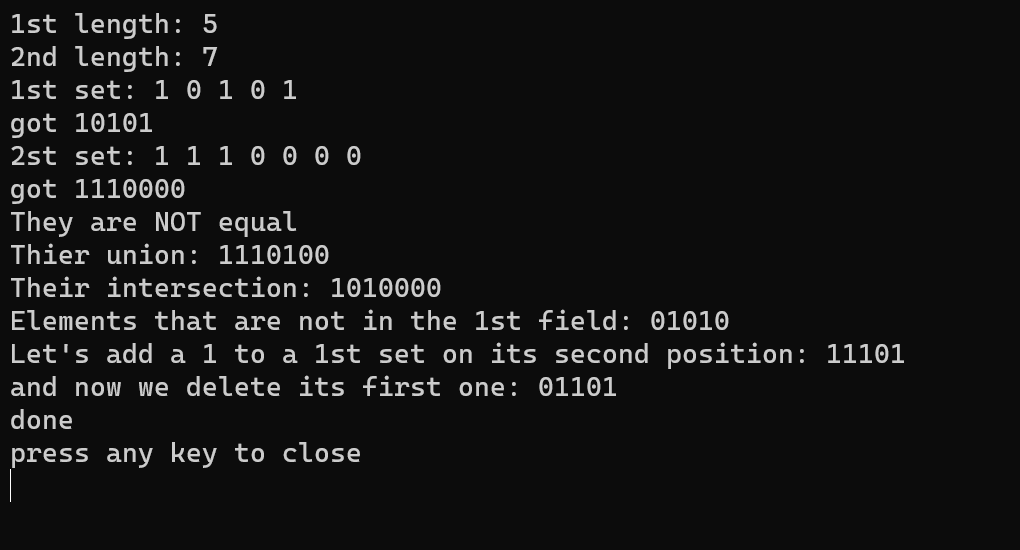
1. Ввод длин

Вводить значения (0 или 1 для каждого элемента в поле) надо по порядку через пробел или Enter, сначала для первого поля, а затем для второго.

1. Для проверки правильности ввода программа выводит полученные ею поля (Рис. 3):



1. Результат ввода полей
2. После ввода второго поля программа покажет примеры результатов операций над полями. После чего попросит ввести любое значение для выхода (Рис. 4).



1. Итог

## Приложение для демонстрации работы множеств

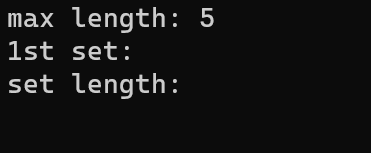
1. Запустить sample\_tset.exe. В результате появится следующее окно (Рис. 5):



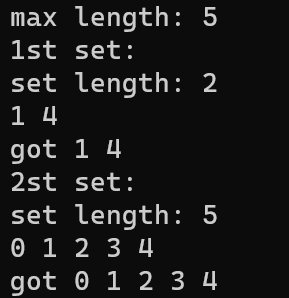
1. Основное окно приложения

Требуется ввести максимально возможный размер множества (то есть его универс)

1. После ввода длины универса (Рис. 6):



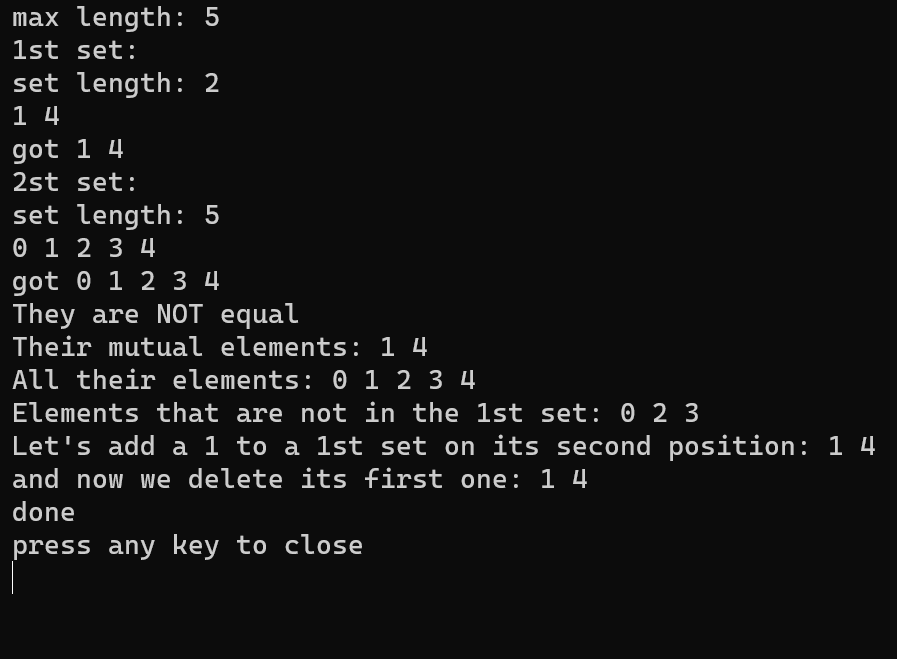
1. Ввод длины
2. Далее потребуется ввести множества соответственно. (Рис. 7):



1. результат ввода множеств

Сначала вводится длина первого множества, а затем номера включённых для него элементов в любом порядке через пробел или Enter. После чего программа выводит полученное значение для проверки корректности ввода. Затем аналогично ввод происходит и для второго множества

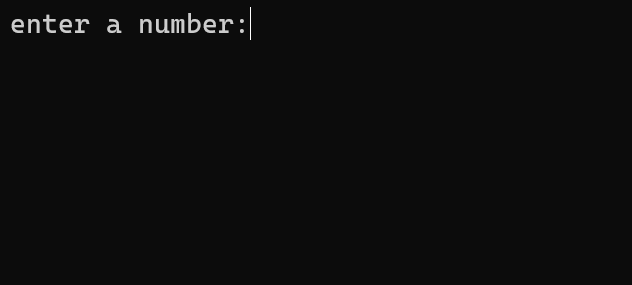
1. Программа покажет примеры результатов операций над множествами. После чего попросит ввести любое значение для выхода (Рис. 8).



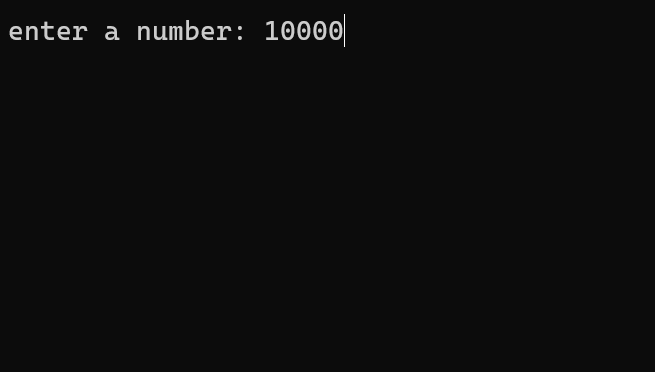
1. Итог

## Приложение «решето Эратосфена»

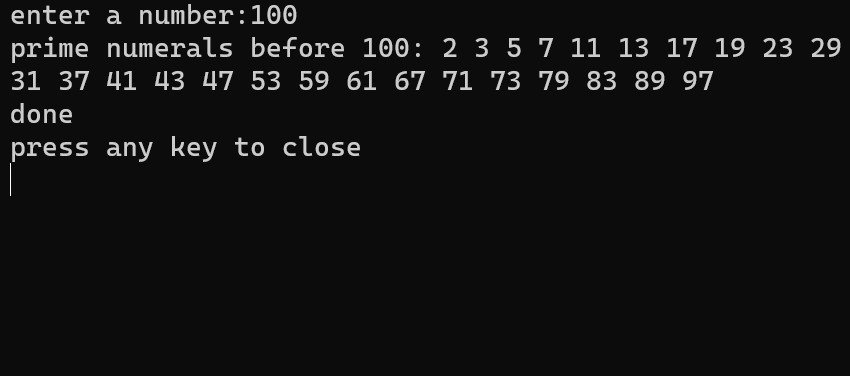
1. Запустить sample\_primenumbers.exe. В результате появится следующее окно (Рис. 9):



1. Основное окно приложения
2. Далее необходимо ввести число, в рамках которого будет искать все простые числа (Рис. 10):



1. Ввод числа
2. Программа выдаст все простые числа, меньше введённого (Рис. 11):



1. Результат

# Руководство программиста

## Использованнные алгоритмы

### Битовые поля

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

Операции над полями реализованы в качестве методов класса полей.

1. Объединение

если хотя бы в одном поле на одном месте стоит 1, то после применения операции в результате будет стоять 1

|  |  |  |
| --- | --- | --- |
| **А** | **В** | **А|В** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

1. Пересечение

если в обоих полях на одном месте стоит 1, то после применения операции в результате будет стоять 1

|  |  |  |
| --- | --- | --- |
| **А** | **В** | **А&В** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

1. Дополнение

Инверсия каждого элемента поля

|  |  |
| --- | --- |
| **А** | **~А** |
| 0 | 1 |
| 1 | 0 |

1. Включение/удаление бита в поле

Включение/удаление бита в поле по его номеру

|  |  |  |  |
| --- | --- | --- | --- |
| **А** | **№** | **+** | **-** |
| 010 | 3 | 011 | 010 |
| 010 | 2 | 010 | 000 |

1. Сравнение

Поэлементное сравнение двух полей

|  |  |  |  |
| --- | --- | --- | --- |
| **А** | **В** | **==** | **!=** |
| 010 | 0011 | 0 | 1 |
| 010 | 010 | 1 | 0 |

### Множества

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

1. Объединение

если элемент есть хотя бы в одном множестве, то он будет включён и в результат

|  |  |  |
| --- | --- | --- |
| **А** | **В** | **А|В** |
| 1, 2, 3 | 2, 3, 4 | 1, 2, 3, 4 |

1. Пересечение

если элемент есть хотя бы в одном множестве, то он будет включён и в результат

|  |  |  |
| --- | --- | --- |
| **А** | **В** | **А&В** |
| 1, 2, 3 | 2, 3, 4 | 2, 3 |

1. Дополнение

Множество, обратное данному

|  |  |
| --- | --- |
| **А** | **~А** |
| 0, 3, 4 | 1, 2 |

1. Включение/исключение элемента из множества

|  |  |  |  |
| --- | --- | --- | --- |
| **А** | **Elem** | **+** | **-** |
| 0, 3, 4 | 3 | 0, 3, 4 | 0, 4 |
| 0, 3, 4 | 2 | 0, 2, 3, 4 | 0, 3, 4 |

1. Сравнение

Поэлементное сравнение двух множеств

|  |  |  |  |
| --- | --- | --- | --- |
| **А** | **В** | **==** | **!=** |
| 0, 3, 4 | 2, 3 | 0 | 1 |
| 0, 3, 4 | 0, 3, 4 | 1 | 0 |

Операции над полями реализованы в виде методов класса множество

### Алгоритм «решето Эратосфена»

1. Получить от пользователя число n, до которого требуется искать простые числа
2. Создать вспомогательное битовое поле длины n+1, заполненное единицами
3. Проверка чисел до квадратного корня n-го и удаление кратных членов
4. Оставшиеся числа будут простыми

## Описание классов

### Класс TBitField

Объявление класса:

class TBitField

{

private:

int BitLen;

TELEM \*pMem;

int MemLen;

int GetMemIndex(const int n) const;

TELEM GetMemMask (const int n) const;

public:

TBitField(int len);

TBitField(const TBitField &bf);

~TBitField();

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);

};

*Поля:*

BitLen – длина битового поля.

pMem – массив вспомогательных единиц

MemLen – количество вспомогательных единц

*Конструкторы:*

TBitField(int len) - конструктор с параметром

TBitField(const TBitField &bf) - конструктор копирования

*Деструктор*:

~TBitField();

*Методы:*

int GetMemIndex(const int n) const;

Назначение: получение индекса элемента, где хранится бит.

Входные данные: n – номер бита.

Выходные данные: индекс элемента, где хранится бит с номером n.

TELEM GetMemMask (const int n) const;

Назначение: получение маски элемента, где хранится бит.

Входные данные: n – номер бита.

Выходные данные: число, являющееся маской бита с номером n

int GetLength(void) const;

Назначение: получение длины поля вне метода.

Входные данные: нет

Выходные данные: длина поля

void SetBit(const int n);

Назначение: установка бита с индексом n единицей

Входные данные: n - номер

Выходные данные: нет

void ClrBit(const int n);

Назначение: установка бита с индексом n нулем

Входные данные: n - номер

Выходные данные: нет

int GetBit(const int n) const;

Назначение: получение значения бита под индексом n

Входные данные: n - номер

Выходные данные: значение бита под индексом n

*Операторы:*

int operator==(const TBitField &bf) const

Назначение: оператор сравнения.

Сравнить на равенство 2 битовых поля.

Входные данные: &bf – битовое поле, с которым идёт сравнение.

Выходные данные: 1 если равны, 0 если нет.

int operator!=(const TBitField &bf) const

Назначение: оператор сравнения на неравенство.

Сравнить на неравенство 2 битовых поля.

Входные данные: &bf – битовое поле, с которым идёт сравнение.

Выходные данные: 0 если равны, 1 если нет.

сonst TBitField& operator=(const TBitField &bf)

Назначение: оператор присваивания. Присвоить полю \*this поле &bf.

Входные данные: &bf – битовое поле, которое присваевается.

Выходные параметры: ссылка на поле \*this равное bf.

TBitField operator|(const TBitField &bf)

Назначение: оператор **или**.

Входные данные: &bf – битовое поле, с которым проводится операция.

Выходные данные: поле (\*this | bf).

TBitField operator&(const TBitField &bf)

Назначение: оператор **и**.

Входные данные: &bf – битовое поле, с которым проводится операция.

Выходные данные: поле (\*this & bf).

TBitField operator~(void);

Назначение: оператор **не**.

Входные данные: нет .

Выходные данные: поле **~**(\*this).

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

Назначение: оператор **ввода** из терминала.

Входные данные: &istr - поток ввода, &bf – битовое поле.

Выходные данные: нет.

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

Назначение: оператор **вывода** в терминал.

Входные данные: &ostr - поток вывода, &bf – битовое поле.

Выходные данные: переданное поле выводится в терминал

### Класс 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);

};

*Поля*:

int MaxPower;  - мощность универса для этого множества

TBitField BitField; - поле, характеризующее множество

*Конструкторы*:

TSet(int mp);

TSet(const TSet &s);

TSet(const TBitField &bf);

*Методы*:

int GetMaxPower() const;

Назначение: получение максимальной мощности множества, то есть размера универса

Входные данные: нет

Выходные данные: максимальная мощность множества.

void InsElem(const int Elem);

Назначение: включение элемента в множество.

Входные данные: Elem – номер включаемого элемента.

Выходные данные: нет

void DelElem(const int Elem);

Назначение: удаление элемента из множества.

Входные данные: Elem – номер удаляемого элемента.

Выходные данные: нет

int IsMember(const int Elem) const;

Назначение: проверка элемента на наличие в множестве.

Входные данные: Elem – номер проверяемого элемента.

Выходные данные:

1 – элемент присутствует в множестве.

0 – элемента в множестве нет.

*Операторы:*

int operator==(const TSet &s) const

Назначение: оператор **сравнения**.

Сравнить на равенство 2 множеств.

Входные данные: &s– множество, с которым идёт сравнение.

Выходные данные: 1 если равны, 0 если нет.

int operator!=(const TSet &s) const

Назначение: оператор **сравнения на неравенство**.

Сравнить на неравенство 2 множеств.

Входные данные: &s– множество, с которым идёт сравнение.

Выходные данные: 0 если равны, 1 если нет.

сonst TSet& operator=(const TSet &s)

Назначение: оператор **присваивания**

Входные данные: &s– множество, которое присваивается.

Выходные параметры: ссылка на множество \*this, равное s.

TSet operator\*(const TSet &s)

Назначение: оператор **пересечения множеств**.

Входные данные: &s– множество, с которым проводится операция.

Выходные данные: множество (\*this) \* (s).

TSet operator+(const TSet &s)

Назначение: оператор **объединения множеств**.

Входные данные: &s – множество, с которым проводится операция.

Выходные данные: множество (\*this + s).

TSet operator-(const TSet &s)

Назначение: оператор **вычитания множества**.

Входные данные: &s – множество, с которым проводится операция.

Выходные данные: множество (\*this - s).

TSet operator-(const int Elem)

Назначение: оператор **исключения элемента**.

Входные данные: &s – номер элемента, с которым проводится операция.

Выходные данные: множество (\*this без Elem).

TSet operator+(const int Elem)

Назначение: оператор **включения элемента**.

Входные данные: &s – номер элемента, с которым проводится операция.

Выходные данные: множество (\*this c Elem).

TSet operator~(void);

Назначение: оператор **обратного множества**.

Входные данные: нет.

Выходные данные: множество **~**(\*this).

friend istream &operator>>(istream &istr, const TSet &s)

Назначение: оператор **ввода** из терминала.

Входные данные: &istr - поток ввода, &s – множество.

Выходные данные: нет.

friend ostream &operator<<(ostream &ostr, const TSet &s)

Назначение: оператор **вывода** в терминал.

Входные данные: &ostr - поток вывода, &s – множество.

Выходные данные: переданное множество выводится в терминал

# Заключение

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

# Литература

1. Битовые поля [<https://learn.microsoft.com/ru-ru/cpp/cpp/cpp-bit-fields?view=msvc-170>]
2. Справочник по С++ [<https://prog-cpp.ru/cpp/>]
3. Справочник по С++ [<https://en.cppreference.com/w/>]

# Приложения

## Приложение А. Реализация класса TBitField

TBitField::TBitField(int len)

{

    if (len < 0)

        throw "negative length caught";

    BitLen = len;

    MemLen = BitLen / bitsInElem + 1;

    pMem = new TELEM[MemLen];

    memset(pMem, 0, Amount);

}

TBitField::TBitField(const TBitField &bf)

{

    BitLen = bf.BitLen;

    MemLen = bf.MemLen;

    pMem = new TELEM[MemLen];

    memcpy(pMem, bf.pMem, Amount);

}

TBitField::~TBitField()

{

    delete[] pMem;

}

int TBitField::GetMemIndex(const int n) const

{

    if (n < BitLen)

        return n >> shiftSize;

    else

        throw "no such bit";

}

TELEM TBitField::GetMemMask(const int n) const

{

   return 1 << (n & (bitsInElem - 1));

}

int TBitField::GetLength(void) const

{

    return BitLen;

}

void TBitField::SetBit(const int n)

{

     if ((n >= BitLen) || (n < 0))

        throw "out of range";

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

}

void TBitField::ClrBit(const int n)

{

    if ((n >= BitLen) || (n < 0))

        throw "out of range";

    pMem[GetMemIndex(n)] = pMem[GetMemIndex(n)] & (~GetMemMask(n));

}

int TBitField::GetBit(const int n) const

{

    if (n >= BitLen || n < 0)

        throw "out of range";

    if ((pMem[GetMemIndex(n)] & GetMemMask(n)) == 0)

        return 0 ;

    else

        return 1;

}

const TBitField& TBitField::operator=(const TBitField &bf)

{

    if (\*this == bf)

        return \*this;

    if (BitLen != bf.BitLen)

    {

        delete[] pMem;

        BitLen = bf.BitLen;

        MemLen = bf.MemLen;

        pMem = new TELEM[MemLen];

    }

    memcpy(pMem, bf.pMem, Amount);

    return \*this;

}

int TBitField::operator==(const TBitField &bf) const

{

    if (BitLen != bf.BitLen)

        return 0;

    for (int i = 0; i < MemLen; i++)

        if (pMem[i] != bf.pMem[i])

            return 0;

     return 1;

}

int TBitField::operator!=(const TBitField &bf) const

{

  return (!(\*this == bf));

}

TBitField TBitField::operator|(const TBitField &bf)

{

    int maxlen = max(BitLen, bf.BitLen);

    TBitField Ftmp(maxlen);

    for (int i = 0; i < Ftmp.MemLen; i++)

        Ftmp.pMem[i] = pMem[i] | bf.pMem[i];

    return Ftmp;

}

TBitField TBitField::operator&(const TBitField &bf)

{

    int maxlen = max(BitLen, bf.BitLen);

    TBitField Ftmp(maxlen);

    for (int i = 0; i < Ftmp.MemLen; i++)

        Ftmp.pMem[i] = pMem[i] & bf.pMem[i];

    return Ftmp;

}

TBitField TBitField::operator~(void)

{

    TBitField Ftmp(BitLen);

    for (int i = 0; i < BitLen; i++)

        if (GetBit(i))

            Ftmp.ClrBit(i);

        else Ftmp.SetBit(i);

    return Ftmp;

}

istream &operator>>(istream &istr, TBitField &bf)

{

    int bt;

        for (int i = 0; i < bf.BitLen; i++) {

            istr >> bt;

            if (!bt) {

                bf.ClrBit(i);

            }

            else if (bt == 1) {

                bf.SetBit(i);

            }

            else{

                cout << "not 0 or 1, setting to 1";

                bf.SetBit(i);

                throw "not 0 or 1";

            }

        }

        return istr;

}

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

{

    for (int i = 0; i < bf.BitLen; i++){

        if (bf.GetBit(i)) ostr << "1";

        else ostr << "0";

    }

    return ostr;

}

## Приложение Б. Реализация класса TSet

TSet::TSet(int mp) : MaxPower(mp), BitField(mp) {}

TSet::TSet(const TSet &s) :

BitField(s.BitField), MaxPower(s.GetMaxPower()) {}

TSet::TSet(const TBitField &bf) :

BitField(bf), MaxPower(bf.GetLength()){}

int TSet::GetMaxPower(void) const

{

    return MaxPower;

}

int TSet::IsMember(const int Elem) const

{

    return BitField.GetBit(Elem);

}

void TSet::InsElem(const int Elem)

{

    BitField.SetBit(Elem);

}

void TSet::DelElem(const int Elem)

{

    BitField.ClrBit(Elem);

}

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

{

    if (\*this == s)

        return \*this;

    BitField = s.BitField;

    MaxPower = s.MaxPower;

    return \*this;

}

int TSet::operator==(const TSet &s) const

{

    if (MaxPower != s.MaxPower)

        return 0;

    return (BitField == s.BitField);

}

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

    return !(\*this == s);

}

TSet TSet::operator+(const TSet &s)

{

    TSet TStmp(max(MaxPower, s.GetMaxPower()));

    TStmp.BitField = BitField | s.BitField;

    return TStmp;

}

TSet TSet::operator+(const int Elem)

{

    TSet TStmp(\*this);

    TStmp.InsElem(Elem);

    return TStmp;

}

TSet TSet::operator-(const int Elem)

{

    TSet TStmp(\*this);

    TStmp.DelElem(Elem);

    return TStmp;

}

TSet TSet::operator\*(const TSet &s)

{

    TSet TStmp(max(MaxPower, s.GetMaxPower()));

    TStmp.BitField = BitField & s.BitField;

    return TStmp;

}

TSet TSet::operator~(void)

{

    TSet TStmp(MaxPower);

    TStmp.BitField = ~BitField;

    return TStmp;

}

istream &operator>>(istream &istr, TSet &s)

{

   int elem;

   int n;

   cout << "set length: ";

   istr >> n;

   for (int i = 0; i < n; i++) {

       istr >> elem;

       s.InsElem(elem);

   }

    return istr;

}

ostream& operator<<(ostream &ostr, const TSet &s)

{

    for (int i = 0; i < s.MaxPower; ++i)

        if (s.IsMember(i))

            ostr << i << " ";

    return ostr;

}