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»**

**Выполнил(а):** студент группы 3822Б1ФИ1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Петров Н.Д. /

Подпись

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

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

Подпись

Нижний Новгород  
2023

Оглавление

[Введение 3](#_Toc149072361)

[1 Постановка задачи 4](#_Toc149072362)

[2 Руководство пользователя 5](#_Toc149072363)

[2.1 Приложение для демонстрации работы битовых полей 5](#_Toc149072364)

[2.2 Приложение для демонстрации работы множеств 6](#_Toc149072365)

[2.3 Приложение «решето Эратосфена» 7](#_Toc149072366)

[3 Руководство программиста 8](#_Toc149072367)

[3.1 Использованные алгоритмы 8](#_Toc149072368)

[3.1.1 Битовые поля 8](#_Toc149072369)

[3.1.2 Множества 8](#_Toc149072370)

[3.1.3 Алгоритм «решето Эратосфена» 8](#_Toc149072371)

[3.2 Описание классов 9](#_Toc149072372)

[3.2.1 Класс TbitField 9](#_Toc149072373)

[3.2.2 Класс TSet 10](#_Toc149072374)

[Заключение 12](#_Toc149072375)

[Литературы 13](#_Toc149072376)

[Приложения 14](#_Toc149072377)

[Приложение А. Реализация класса TBitField 14](#_Toc149072378)

[Приложение Б. Реализация класса TSet 16](#_Toc149072379)

# Введение

Создание классов TBitField и TSet является важным шагом для обеспечения удобства и эффективности работы с битовыми данными и множествами в различных приложениях. Эти классы позволяют улучшить производительность, оптимизировать использование памяти и решать широкий спектр задач, связанных с битовой информацией.

Актуальность: Битовые поля и множества на их основе имеют высокую актуальность в современной информатике. Они используются для оптимизации памяти, ускорения выполнения различных операций, обработки данных и т.д.

Значимость создания классов TBitField и TSet заключается в том, что они предоставляют удобный и эффективный способ работы с битовыми данными и множествами.

Применимость классов TBitField и TSet охватывает множество областей:

1. Вычислительная техника: Битовые поля используются для оптимизации работы с битами в байтах и словах памяти.
2. Теория множеств: Класс TSet позволяет представлять и выполнять операции с множествами на основе битовых полей.
3. Криптография: Битовые поля часто используются для реализации различных криптографических алгоритмов.
4. Оптимизация алгоритмов: Битовые поля могут быть использованы для оптимизации хранения и выполнения операций над битовой информацией в различных алгоритмах и структурах данных.

# Постановка задачи

Цель – реализация классов TBitField и TSet, предназначенных для работы с битовыми полями и множествами на их основе.

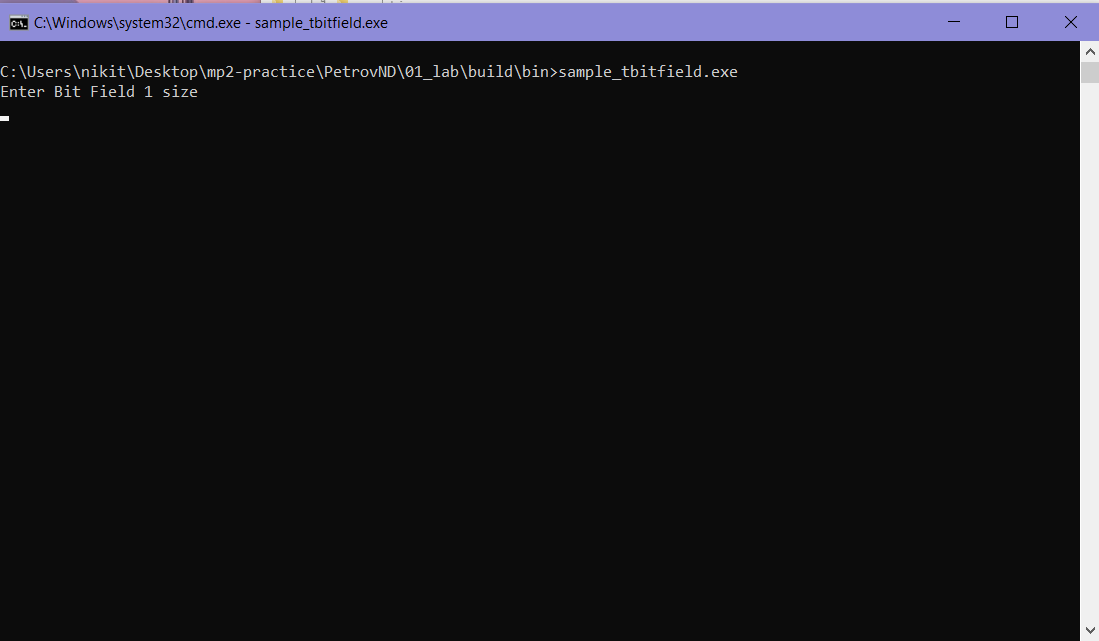
Задачи:

1. Исследовать тематическую литературу.
2. Реализовать класс TBitField.
3. Реализовать класс TSet
4. Провести тестирование разработанных классов для проверки их корректной работы

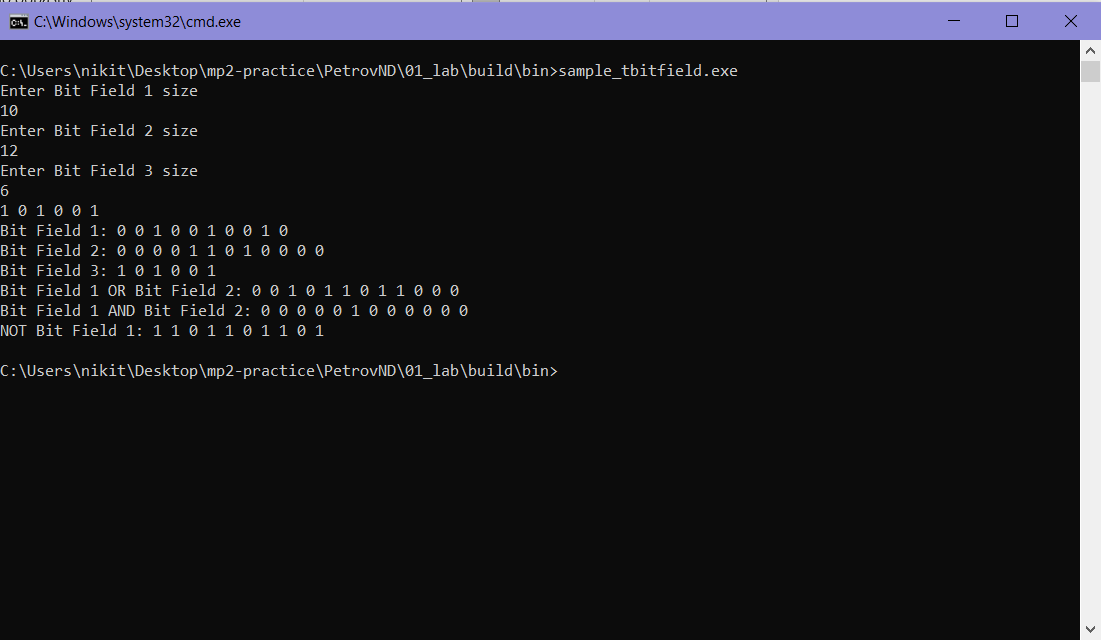
# Руководство пользователя

## Приложение для демонстрации работы битовых полей

1. Запустить sample\_tbitfield.exe. В результате появится следующее окно: Рис. 1



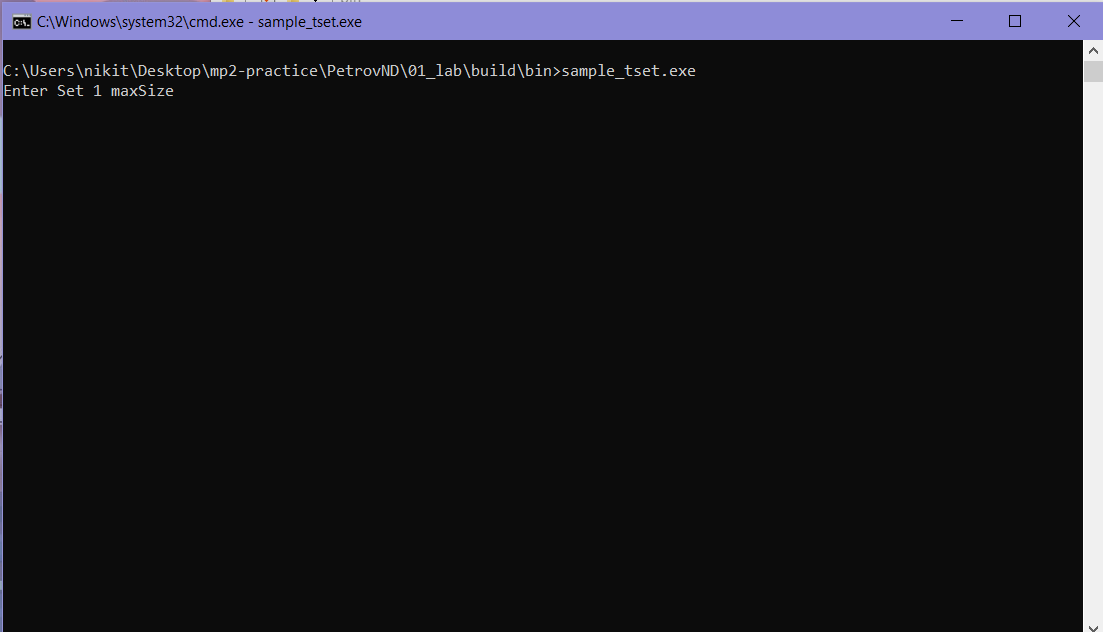
1. Основное окно приложения sample\_tbitfield
2. Необходимо ввести размеры первого, второго и третьего битовых полей (1 и 2 будут сгенерированы автоматически, вам будет предложено заполнить третье поле самостоятельно) Затем будет продемонстрирована работа битовых полей. Рис. 2



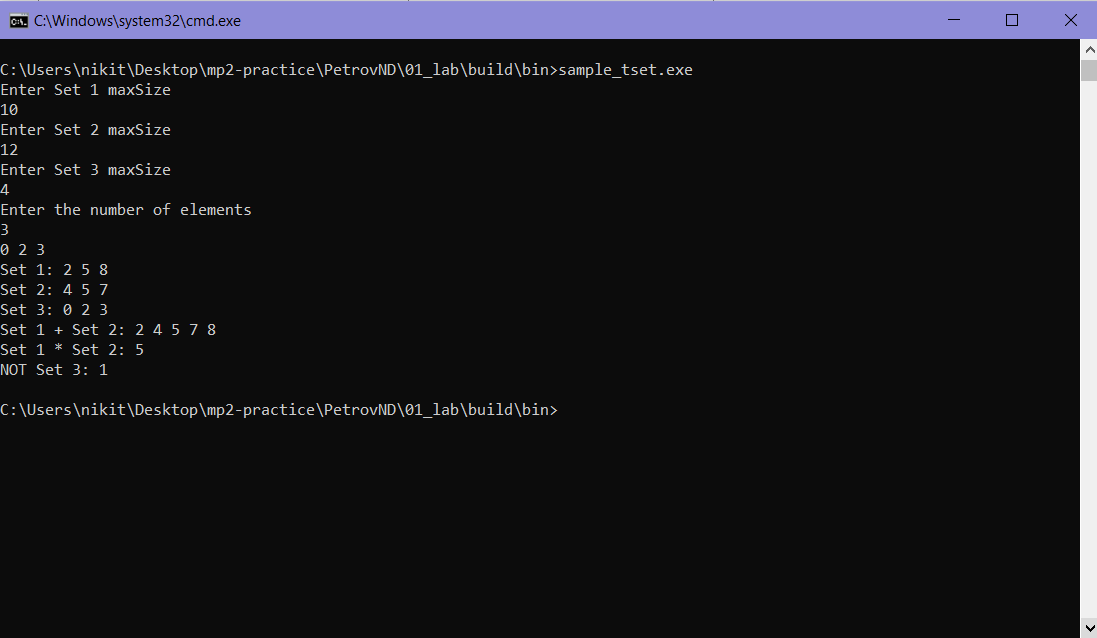
1. Основное окно приложения sample\_tbitfield

## Приложение для демонстрации работы множеств

1. Запустить sample\_tset.exe. В результате появится следующее окно: Рис. 3



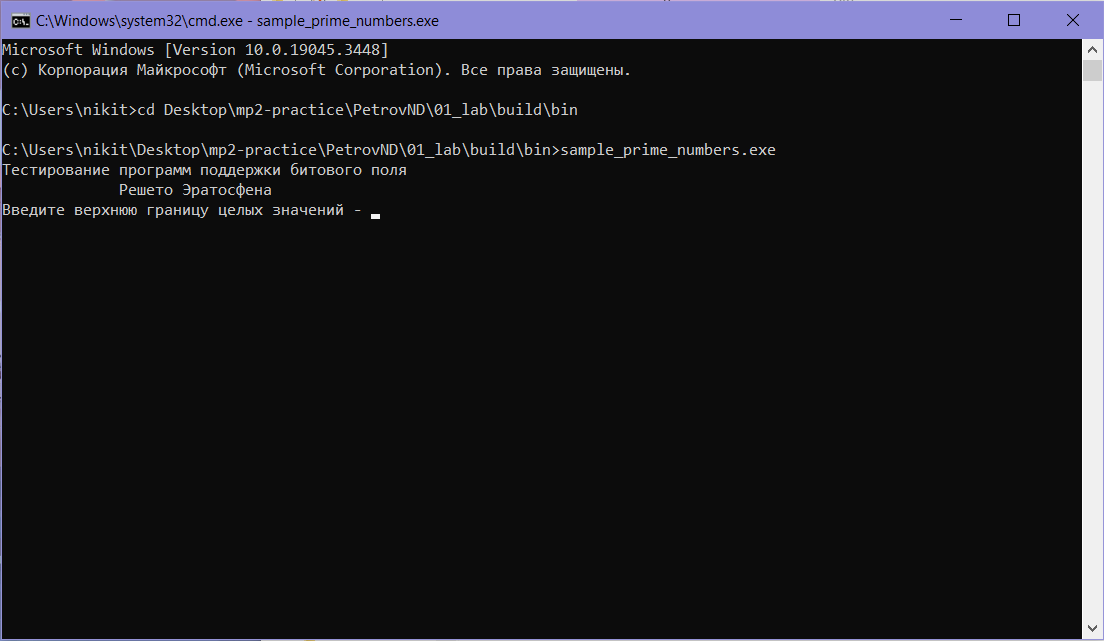
1. Основное окно приложения sample\_tset
2. Необходимо ввести максимальную мощность первого, второго и третьего множества (1 и 2 будут заполнены автоматически, вам будет предложено заполнить третье множество самостоятельно) Затем будет продемонстрирована работа множеств. Рис. 4



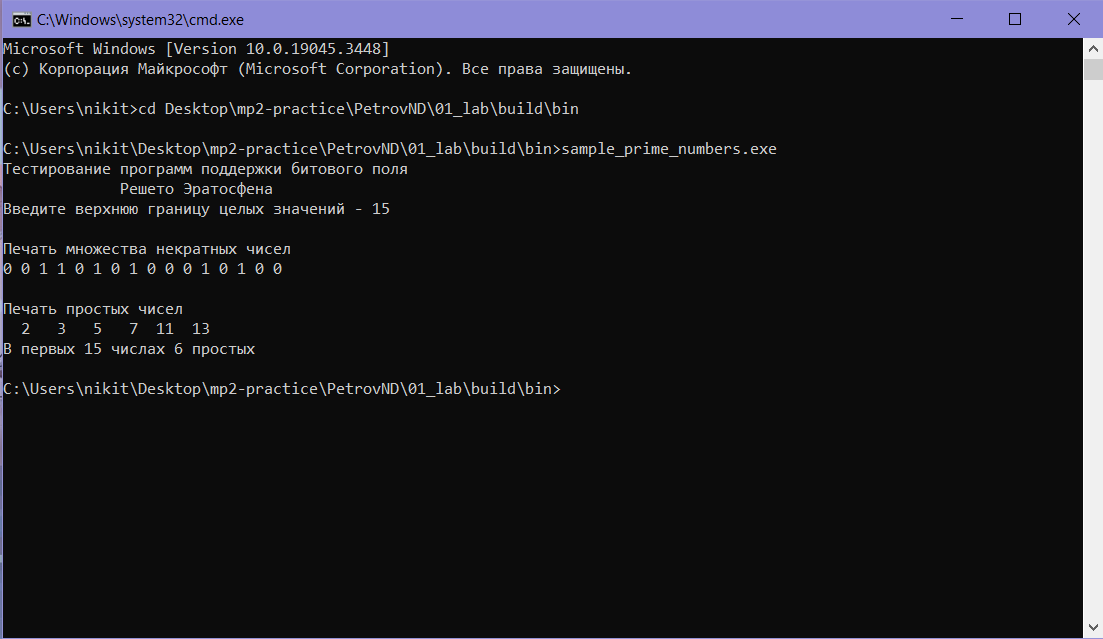
1. Основное окно приложения sample\_tset

## Приложение «решето Эратосфена»

1. Запустить sample\_prime\_numbers.exe. В результате появится следующее окно: Рис. 5



1. Основное окно приложения sample\_prime\_numbers
2. Введите верхнюю границу целых значений для получения результата: Рис. 6



1. Основное окно приложения sample\_prime\_ numbers

# Руководство программиста

## Использованные алгоритмы

### Битовые поля

Определение и хранение: Битовое поле представляет собой структуру данных, в которой каждый бит имеет значение либо 0 (сброшен), либо 1 (установлен). Эти биты хранятся в массиве (или нескольких массивах), где каждый элемент массива называется "словом" и содержит несколько битов. Слова используются для экономии памяти.

Операции над битовыми полями:

1. Установка бита: путем изменения соответствующего бита на 1.
2. Сброс бита: путем изменения соответствующего бита на 0.
3. Получение значения бита: Чтение значения бита без его изменения.
4. Операции ИЛИ, И и отрицания: Позволяют объединять, пересекать и инвертировать битовые поля.
5. Сравнение битовых полей: Определение равенства и неравенства двух битовых полей.

### Множества

Определение и хранение: Множество — это абстрактная структура данных, представляющая набор уникальных элементов. Множества на основе битовых полей используют битовые поля для хранения информации о наличии или отсутствии элементов в множестве. Каждый элемент в множестве представлен соответствующим битом.

Операции над множествами на основе битовых полей:

1. Вставка элемента: Устанавливание соответствующего бита для элемента.
2. Удаление элемента: Сброс соответствующего бита для элемента.
3. Проверка наличия элемента: Проверка значения соответствующего бита.
4. Операции множественного объединения, пересечения и дополнения.

### Алгоритм «решето Эратосфена»

Алгоритм решето Эратосфена — это алгоритм нахождения всех простых чисел до некоторого целого числа N.

Он заключается в фильтрации чисел, в результате которой нужные числа (простые) остаются, а ненужные (составные) исключаются.

1. Для нахождения всех простых чисел до заданного числа N нужно выполнить следующие шаги:
2. Заполнить массив из N элементов целыми числами подряд от 2 до N.
3. Присвоить переменной p значение 2 (первого простого числа).
4. Удалить из массива числа от p2 до N с шагом p (это будут числа кратные p: p2, p2+p, p2+2p и т. д.).
5. Найти первое неудаленное число в массиве, большее p, и присвоить значению переменной p это число.
6. Повторять два предыдущих шага пока это возможно.
7. Все оставшиеся в массиве числа являются простыми числами от 2 до N

## Описание классов

### Класс TbitField

Объявление класса:

class TBitField

{

private:

int BitLen;

TELEM \*pMem;

int MemLen;

const int bitsInElem = sizeof(TELEM) \* 8;

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);

};

Поля:

**BitLen** – член класса, который хранит длину битового поля, то есть максимальное количество битов, которые могут быть представлены в данном объекте.

**pMem** – указатель на массив, который используется для фактического хранения битовых данных. Каждый элемент массива pMem содержит набор битов.

**MemLen** – количество элементов массива pMem, необходимых для хранения битовых данных.

Методы:

int GetMemIndex(const int n) const;

Назначение: Метод используется для получения индекса элемента, в котором хранится бит с указанным номером.

Входные данные: n - номер бита, для которого нужно найти индекс элемента.

Выходные данные: Индекс элемента массива pMem, в котором хранится бит с номером n. Этот индекс будет использоваться для доступа к соответствующему элементу памяти, где хранятся биты.

TELEM GetMemMask(const int n) const;

Назначение: Метод используется для получения битовой маски, которая соответствует биту с указанным номером.

Входные данные: n - номер бита, для которого нужно получить битовую маску.

Выходные данные: Битовая маска, в которой только один бит установлен в позиции, соответствующей номеру n. Остальные биты маски равны 0.

int GetLength(void) const;

Назначение: Метод используется для получения длины битового поля.

Выходные данные: Длина битового поля, то есть количество битов, которое может храниться в объекте TBitField.

void SetBit(const int n);

Назначение: Метод используется для установки бита с указанным номером.

Входные данные: n - номер бита, который нужно установить (установить в 1).

void ClrBit(const int n);

Назначение: Метод используется для очистки бита с указанным номером (установка в 0).

Входные данные: n - номер бита, который нужно очистить.

int GetBit(const int n) const;

Назначение: Метод используется для получения значения бита с указанным номером.

Входные данные: n - номер бита, для которого нужно получить значение (0 или 1).

Выходные данные: Значение бита с номером n.

### Класс TSet

Объявление класса:

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;

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:

**MaxPower** – хранит максимальную мощность множества (максимальное количество элементов в множестве).

**BitField** – представляет собой битовое поле, где каждый бит соответствует элементу множества. Размер битового поля равен MaxPower.

Методы:

int GetMaxPower() const;

Назначение: получение максимальной мощности множества, то есть максимального номера элемента, который может быть включен в множество.

Входные данные: отсутствуют.

Выходные данные: максимальная мощность множества.

void InsElem(const int Elem);

Назначение: включение элемента в множество.

Входные данные: Elem – номер элемента, который нужно включить в множество.

Выходные данные: отсутствуют.

void DelElem(const int Elem);

Назначение: удаление элемента из множества.

Входные данные: Elem – номер элемента, который нужно удалить из множества.

Выходные данные: отсутствуют.

int IsMember(const int Elem) const;

Назначение: проверка наличия элемента в множестве.

Входные данные: Elem – номер элемента, который нужно проверить.

Выходные данные: 1, если элемент присутствует в множестве, и 0 в противном случае.

# Заключение

В результате проделанной работы были созданы два класса, **TBitField** и **TSet**, предоставляющие возможность работать с битовыми полями и множествами. Эти классы могут быть использованы в различных задачах, требующих манипуляции битами и множествами элементов. Весь функционал классов был реализован и протестирован, что позволяет уверенно использовать их в практических задачах.

# Литературы

1. Metanit [[С | Битовые поля (metanit.com)](https://metanit.com/c/tutorial/6.7.php)].
2. MicrosoftLearn [[Битовые поля в C++ | Microsoft Learn](https://learn.microsoft.com/ru-ru/cpp/cpp/cpp-bit-fields?view=msvc-170)].
3. Habr [[Расставим точки над структурами C/C++ / Хабр (habr.com)](https://habr.com/ru/articles/142662/)].

# Приложения

## Приложение А. Реализация класса TBitField

TBitField::TBitField(int len) {

if (len < 0) {

throw invalid\_argument("Length cannot be negative");

}

BitLen = len;

MemLen = (len + bitsInElem - 1) / bitsInElem;

pMem = new TELEM[MemLen];

memset(pMem, 0, MemLen \* sizeof(TELEM));

}

TBitField::TBitField(const TBitField& bf) {

BitLen = bf.BitLen;

MemLen = bf.MemLen;

pMem = new TELEM[MemLen];

memcpy(pMem, bf.pMem, MemLen \* sizeof(TELEM));

}

TBitField::~TBitField() {

delete[] pMem;

}

int TBitField::GetMemIndex(const int n) const {

if (n < 0 || n >= BitLen) {

throw out\_of\_range("Bit index is out of range");

}

return n / bitsInElem;

}

TELEM TBitField::GetMemMask(const int n) const {

if (n < 0 || n >= BitLen) {

throw out\_of\_range("Bit index is out of range");

}

int bitIndex = n % bitsInElem;

TELEM mask = 1;

mask <<= bitIndex;

return mask;

}

int TBitField::GetLength(void) const {

return BitLen;

}

void TBitField::SetBit(const int n) {

if (n < 0 || n >= BitLen) {

throw out\_of\_range("Bit index is out of range");

}

int index = GetMemIndex(n);

TELEM mask = GetMemMask(n);

pMem[index] |= mask;

}

void TBitField::ClrBit(const int n) {

if (n < 0 || n >= BitLen) {

throw out\_of\_range("Bit index is out of range");

}

int index = GetMemIndex(n);

TELEM mask = GetMemMask(n);

pMem[index] &= ~mask;

}

int TBitField::GetBit(const int n) const {

if (n < 0 || n >= BitLen) {

throw out\_of\_range("Bit index is out of range");

}

int index = GetMemIndex(n);

TELEM mask = GetMemMask(n);

return (pMem[index] & mask) ? 1 : 0;

}

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);

}

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];

}

for (int i = 0; i < MemLen; i++) {

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

}

return \*this;

}

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

int len = (BitLen > bf.BitLen) ? BitLen : bf.BitLen;

TBitField result(len);

for (int i = 0; i < MemLen; i++) {

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

}

return result;

}

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

int len = (BitLen > bf.BitLen) ? BitLen : bf.BitLen;

TBitField result(len);

for (int i = 0; i < MemLen; i++) {

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

}

return result;

}

TBitField TBitField::operator~(void) {

TBitField result(BitLen);

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

if (!GetBit(i))

result.SetBit(i);

}

return result;

}

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

int inputBit;

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

istr >> inputBit;

if (inputBit == 0) {

bf.ClrBit(i);

}

else if (inputBit == 1) {

bf.SetBit(i);

}

else {

throw invalid\_argument("Invalid input. Use 0 or 1.");

}

}

return istr;

}

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

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

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

}

return ostr;

}

## Приложение Б. Реализация класса TSet

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

TSet::TSet(const TSet& s) : MaxPower(s.MaxPower), BitField(s.BitField) { }

TSet::TSet(const TBitField& bf) : MaxPower(bf.GetLength()), BitField(bf) { }

TSet::operator TBitField() {

return BitField;

}

int TSet::GetMaxPower(void) const {

return MaxPower;

}

void TSet::InsElem(const int Elem) {

if (Elem < 0 || Elem >= MaxPower) throw out\_of\_range("Element is out of range");

BitField.SetBit(Elem);

}

void TSet::DelElem(const int Elem) {

if (Elem < 0 || Elem >= MaxPower) throw out\_of\_range("Element is out of range");

BitField.ClrBit(Elem);

}

int TSet::IsMember(const int Elem) const {

if (Elem < 0 || Elem >= MaxPower) throw out\_of\_range("Element is out of range");

return BitField.GetBit(Elem);

}

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);

}

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

if (this == &s) {

return \*this;

}

MaxPower = s.MaxPower;

BitField = s.BitField;

return \*this;

}

TSet TSet::operator+(const int Elem) {

if (Elem >= MaxPower || Elem < 0) throw invalid\_argument("Out of range");

TSet result(\*this);

result.InsElem(Elem);

return result;

}

TSet TSet::operator-(const int Elem) {

if (Elem >= MaxPower || Elem < 0) throw invalid\_argument("Out of range");

TSet result(\*this);

result.DelElem(Elem);

return result;

}

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

int len = (MaxPower > s.MaxPower) ? MaxPower : s.MaxPower;

TSet result(len);

result.BitField = BitField | s.BitField;

return result;

}

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

int len = (MaxPower > s.MaxPower) ? MaxPower : s.MaxPower;

TSet result(len);

result.BitField = BitField & s.BitField;

return result;

}

TSet TSet::operator~() {

TSet result(MaxPower);

result.BitField = ~BitField;

return result;

}

istream& operator>>(istream& istr, TSet& ts) {

int inputElem, n;

cout << "Enter the number of elements" << endl;

cin >> n;

for (int i = 0; i < n; i++) {

istr >> inputElem;

if (inputElem >= 0 && inputElem < ts.MaxPower) {

ts.InsElem(inputElem);

}

else {

cerr << "Invalid input. Element out of range." << endl;

}

}

return istr;

}

ostream& operator<<(ostream& ostr, const TSet& ts) {

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

if (ts.IsMember(i)) {

ostr << i << " ";

}

}

return ostr;

}