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:** студент группы 3822Б1ФИ2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Табунов И. Д./

Подпись

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

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

Подпись

Нижний Новгород  
2023

**Содержание**

[Введение 3](file:///C:\github\test\mp2-practice\report_template.docx#_Toc147915966)

[1 Постановка задачи 4](file:///C:\github\test\mp2-practice\report_template.docx#_Toc147915967)

[2 Руководство пользователя 5](file:///C:\github\test\mp2-practice\report_template.docx#_Toc147915968)

[2.1 Приложение для демонстрации работы битовых полей 5](file:///C:\github\test\mp2-practice\report_template.docx#_Toc147915969)

[2.2 Приложение для демонстрации работы множеств 5](file:///C:\github\test\mp2-practice\report_template.docx#_Toc147915970)

[2.3 «Решето Эратосфено» 5](file:///C:\github\test\mp2-practice\report_template.docx#_Toc147915971)

[3 Руководство программиста 6](file:///C:\github\test\mp2-practice\report_template.docx#_Toc147915972)

[3.1 Описание алгоритмов 6](file:///C:\github\test\mp2-practice\report_template.docx#_Toc147915973)

[3.1.1 Битовые поля 6](file:///C:\github\test\mp2-practice\report_template.docx#_Toc147915974)

[3.1.2 Множества 6](file:///C:\github\test\mp2-practice\report_template.docx#_Toc147915975)

[3.1.3 «Решето Эратосфена» 6](file:///C:\github\test\mp2-practice\report_template.docx#_Toc147915976)

[3.2 Описание программной реализации 6](file:///C:\github\test\mp2-practice\report_template.docx#_Toc147915977)

[3.2.1 Описание класса TBitField 6](file:///C:\github\test\mp2-practice\report_template.docx#_Toc147915978)

[3.2.2 Описание класса TSet 7](file:///C:\github\test\mp2-practice\report_template.docx#_Toc147915979)

[Заключение 8](file:///C:\github\test\mp2-practice\report_template.docx#_Toc147915980)

[Литература 9](file:///C:\github\test\mp2-practice\report_template.docx#_Toc147915981)

[Приложения 10](file:///C:\github\test\mp2-practice\report_template.docx#_Toc147915982)

[Приложение А. Реализация класса TBitField 10](file:///C:\github\test\mp2-practice\report_template.docx#_Toc147915983)

[Приложение Б. Реализация класса TSet 10](file:///C:\github\test\mp2-practice\report_template.docx#_Toc147915984)

# Введение

В современном мире информационных технологий большую роль играют операции с большими объемами данных. Одной из важных операций является эффективная работа с битовыми множествами. Битовое множество - это структура данных, которая позволяет компактно хранить и оперировать набором битов. Такая структура данных может быть использована в различных областях: от алгоритмов сжатия данных до обработки больших объемов информации.

Актуальность и применяемость данной лабораторной работы состоит в том, что она позволяет познакомиться с основными принципами работы с битовыми множествами и реализовать их на языке программирования C++. Использование битовых множеств позволяет существенно ускорить операции с большими объемами данных и снизить требования по памяти.

Таким образом, данная лабораторная работа является актуальной и полезной для студентов и специалистов в области информационных технологий, которые имеют необходимость эффективно работать с битами и битовыми множествами.

# Постановка задачи

**Цель работы:** изучение и практическое применение концепции битовых полей и множеств.

**Задачи:**

1. Изучить теоретические основы битовых полей и множеств.

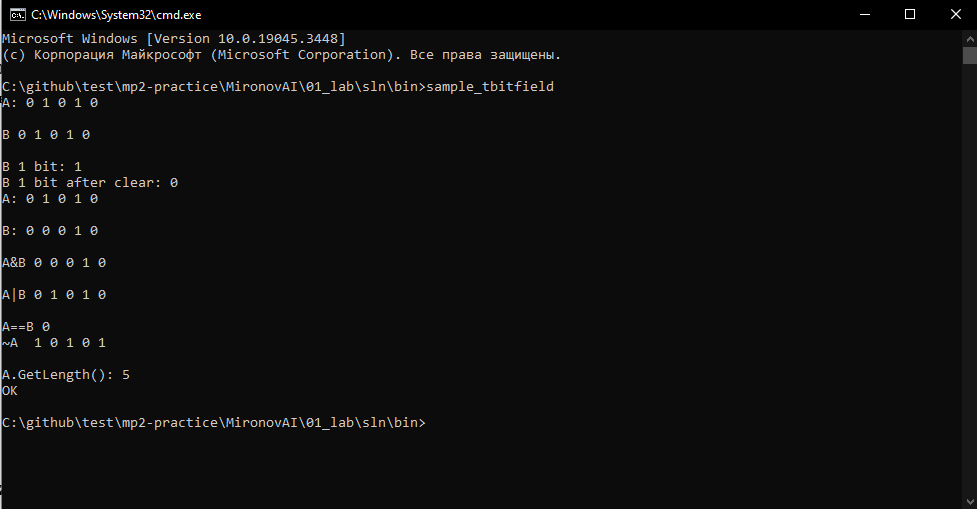
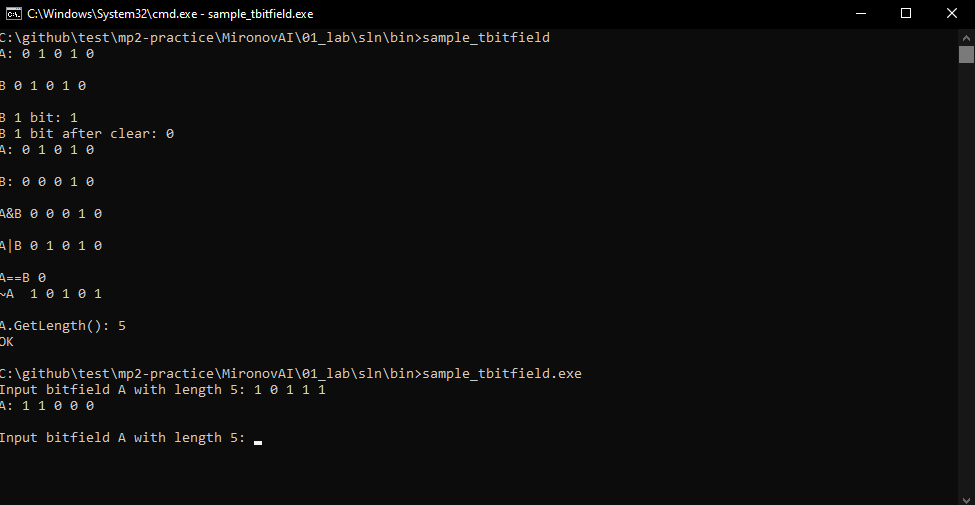
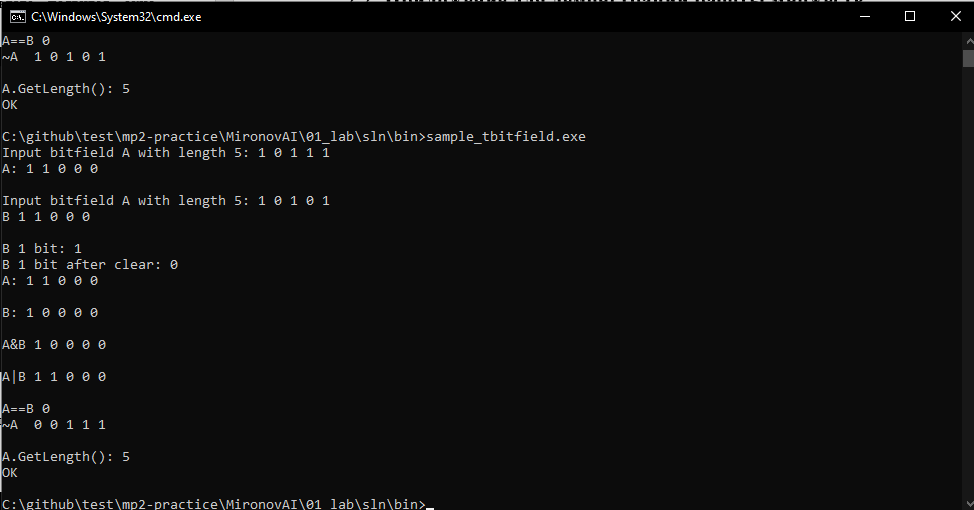
2. Разработать программу, реализующую операции над битовыми полями и множествами.

3. Провести эксперименты с различными наборами данных.

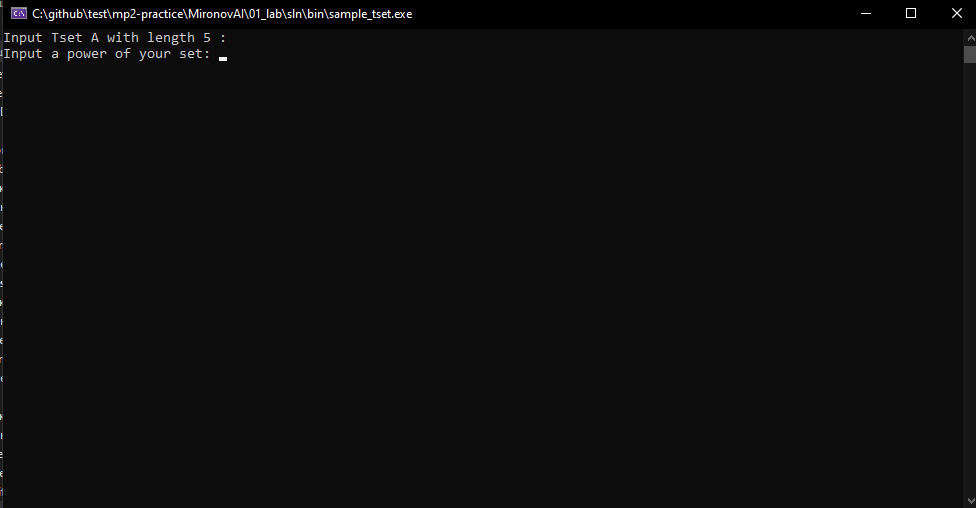
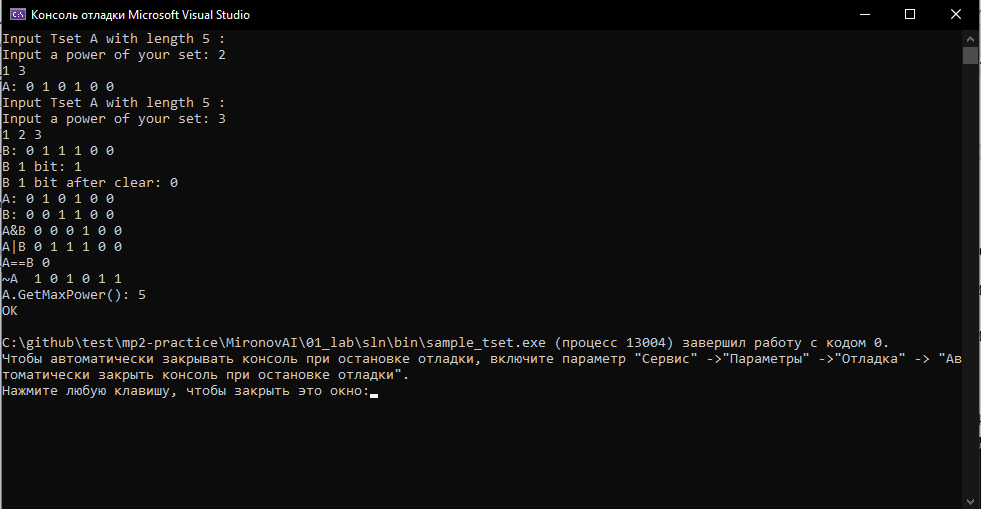
4. Проанализировать полученные результаты и сделать выводы о преимуществах и ограничениях использования битовых полей и множеств.

# Руководство пользователя

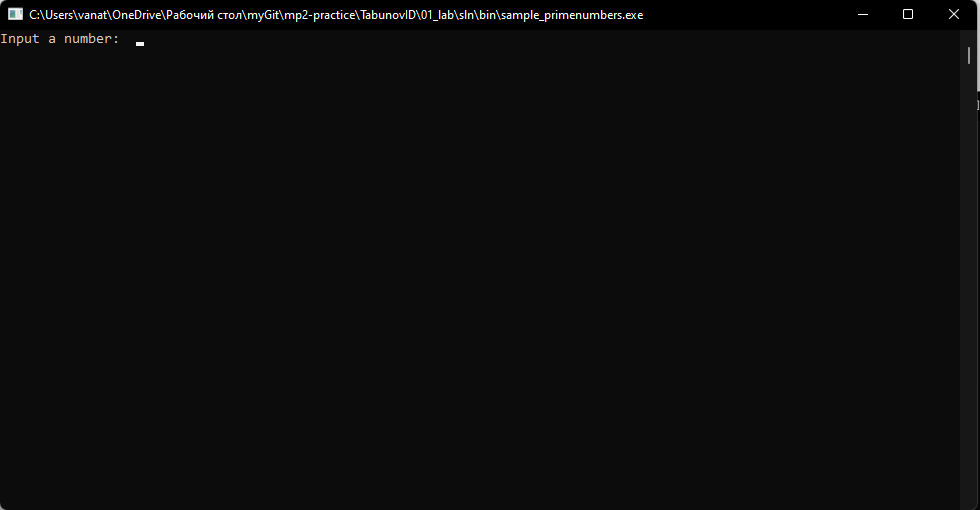
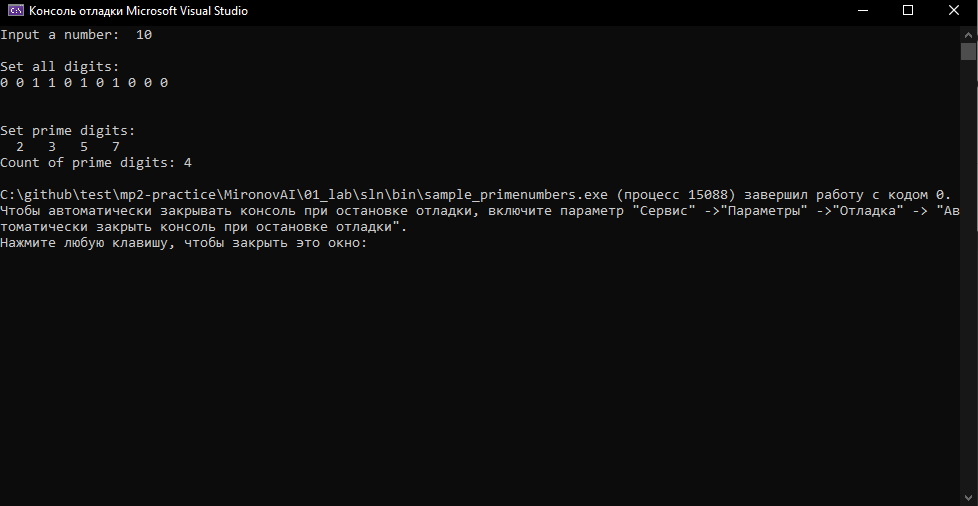
## Приложение для демонстрации работы битовых полей

1. Запустите приложение с названием \*.exe. В результате появится окно, показанное ниже (рис. 1).
2. Основное окно программы
3. Затем вам будет предложено ввести 2 битовых поля длины 5 (рис 2).
4. Ввод битовых полей
5. После ввода множества нулей и /или единиц, будет выведены результаты соответствующих операций и функций (рис 3).
6. Результат тестирования функций класса TBitField

## Приложение для демонстрации работы множеств

1. Запустите приложение с названием sample\_tset.exe. В результате появится окно, показанное ниже (рис 4).
2. Основное окно программы
3. Затем вам будет предложено вести свое множество. Необходимо ввести сначала количество чисел в множестве , программа будет ждать N чисел. Причём они не должны превышать 5.После ввода множества нулей и /или единиц, будет выведены результаты соответствующих операций и функций (рис 5).
4. Результат тестирования класса TSet

## «Решето Эратосфена»

1. Запустите приложение с названием sample\_tset.exe. В результате появится окно, показанное ниже (рис 6).
2. Основное окно программы
3. Затем вам будет нужно ввести целое положительное число. После чего программа выведет простые числа на отрезке до введенного числа и их количество (рис 7).
4. Результат работы алгоритма «Решето Эратосфена»

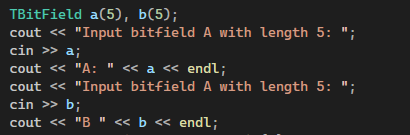
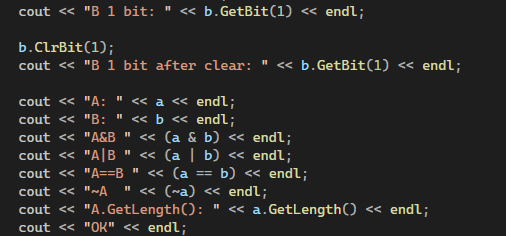
# Руководство программиста

## Описание алгоритмов

### Битовые поля

1. Начало работы указано в пункте [2.1 этого документа](file:///C:\github\test\mp2-practice\report_template.docx#_Приложение_для_демонстрации)
2. Описание методов и полей класса в пункте [3.2.1 этого документа](file:///C:\github\test\mp2-practice\report_template.docx#_Описание_класса_TBitField)

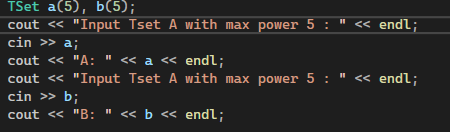
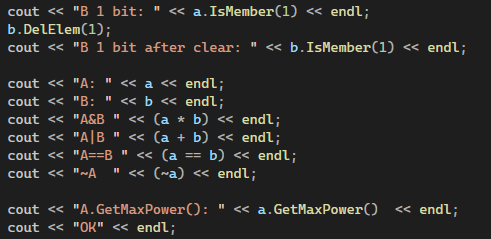
Программа алгоритм состоит из единственной функции void test\_bitfield(), которая сразу вызывается из основной функции int main(). Изначально необходимо алгоритм запрашивает пользователя ввести 2 битовых поля длинной 5 для демонстрации использования экземпляров класса (рис 8)

1. Ввод данных алгоритма 
2. После чего алгоритм последовательно выведет результат вызова соответствующих операций и методов класса (рис 9).
3. Вывод результата работы алгоритма

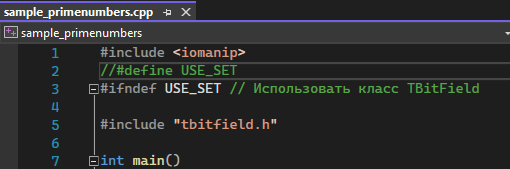
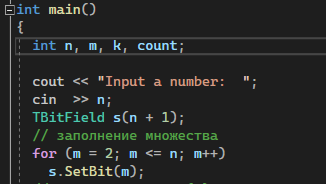
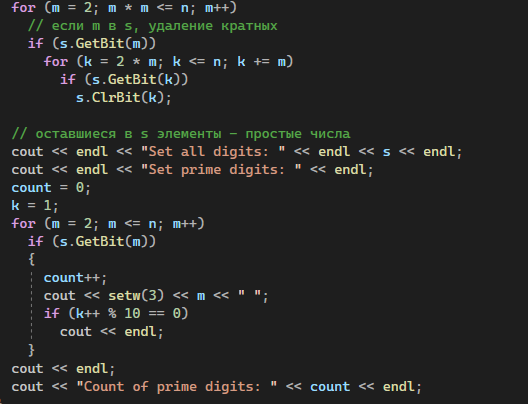
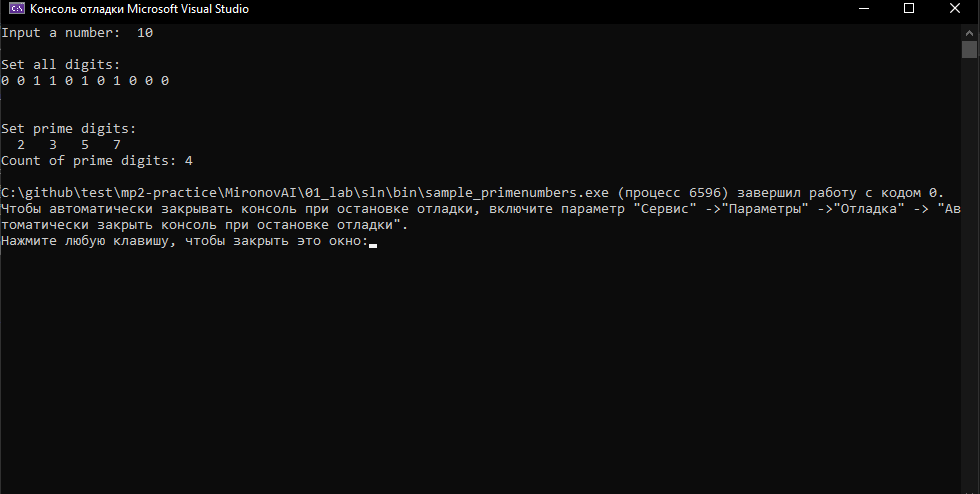
### Множества

1. Начало работы указано в пункте [2.2 этого документа](file:///C:\github\test\mp2-practice\report_template.docx#_Приложение_для_демонстрации_1).
2. Описание методов и полей класса в пункте [3.2.2 этого документа](file:///C:\github\test\mp2-practice\report_template.docx#_Описание_класса_TSet)

Программа алгоритм состоит из единственной функции void test\_tset(), которая сразу вызывается из основной функции int main(). Изначально необходимо алгоритм запрашивает пользователя ввести 2 множества, элементами которых не могут превышать 5 и должны быть целочисленным целым числом, для демонстрации использования экземпляров класса (рис 10)

1. Ввод данных алгоритма
2. После чего алгоритм последовательно выведет результат вызова соответствующих операций и методов класса (рис 11).
3. Результат работы алгоритма 

### «Решето Эратосфена»

1. Начало работы указано в пункте [2.3 текущего документа](file:///C:\github\test\mp2-practice\Решето_Эратосфена#_)
2. Данный алгоритм реализован двумя способами, при помощи классов TSet и TBitField. Изначально алгоритм будет использовать класс TBitField. Для того, чтобы использовать реализацию с использованием TSet, необходимо убрать два слеша перед #define USE\_SET (рис 12).
3. Использование TSet
4. Реализация алгоритма состоит из ввода необходимого числа, создание экземпляра класса TBitField или TSet соответственно. После чего алгоритм заполняет все элементы классов равными 1 (рис 13).
5. Ввод данных и заполнение экземляра
   1. После чего алгоритм , начинает перебирать все числа от 2 до N. Если это число есть в нашем множестве, то мы переходим к шагу 2, иначе к шагу 3.
   2. Это число, и дальше все кратные ему числа удаляются из нашего множества.
   3. Выбирается следующее число. Если это число больше N, то алгоритм заканчивается, иначе – переход к шагу 1. (рис 14)
6. Алгоритм «Решето Эратосфена»
7. После чего выводятся все числа с идентификатором (0, если оно не простое, 1 если простое), затем выводятся все простые числа и количество (рис 15)
8. Вывод данных

## Описание программной реализации

### 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();

// доступ к битам

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.

Методы:

**int GetMemIndex(const int n) const;**

Назначение: получение индекса элемента в памяти…

Входные параметры:

n – номер бита.

Выходные параметры:

Номер элемента в памяти.

TELEM GetMemMask (const int n) const;

Назначение: Получение битовой маски

Входные параметры:

n – номер бита

Выходные параметры:

Битовая маска типа unsigned int

TBitField(int len);

Назначение: конструктор с параметром, выделение памяти

Входные параметры:

len – длина битового поля

Выходные параметры:

Отсутствуют

TBitField(const TBitField &bf);

Назначение: конструктор копирования. Создание экземпляра класса на основе другого экземпляра

Входные параметры:

&bf – ссылка, адрес экземпляра класса, на основе которого будет создан другой.

Выходные параметры:

Отсутствуют

~TBitField();

Назначение: деструктор. Отчистка выделенной памяти

Входные и выходные параметры отсутствуют

int GetLength(void) const;

Назначение: получение длинны битового поля

Входные параметры отсутствуют

Выходные параметры: длинна битового поля

void SetBit(const int n)

Назначение: установить бит = 1

Входные параметры:

n - номер бита, который нужно установить

Выходные параметры отсутствуют

void ClrBit(const int n);

Назначение: отчистить бит (установить бит = 0)

Входные параметры:

n - номер бита, который нужно отчистить

Выходные параметры отсутствуют

int GetBit(const int n) const;

Назначение: вывести бит (узнать бит)

Входные параметры:

n - номер бита, который нужно вывести (узнать)

Выходные параметры: бит (1 или 0, в зависимости есть установлен он, или нет)

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

Назначение: оператор сравнения. Сравнить на равенство 2 битовых поля

Входные параметры:

&bf – битовое поле, с которым мы сравниваем

Выходные параметры: 1 или 0, в зависимости равны они, или нет соответственно

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

Назначение: оператор сравнения. Сравнить на равенство 2 битовых поля

Входные параметры:

&bf – битовое поле, с которым мы сравниваем

Выходные параметры: 1 или 0, в зависимости равны они, или нет соответственно

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

Назначение: оператор присваивания. Присвоить экземпляру \*this экземпляр &bf

Входные параметры:

&bf – битовое поле, которое мы присваиваем

Выходные параметры: ссылка на экземпляр класса TBitField, \*this

TBitField operator|(const TBitField &bf);

Назначение: оператор побитового «ИЛИ»

Входные параметры:

&bf – битовое поле

Выходные параметры: Экземпляр класса , который равен { \*this | bf }

TBitField operator&(const TBitField &bf);

Назначение: оператор побитового «И»

Входные параметры:

&bf – битовое поле, с которым мы сравниваем

Выходные параметры: Экземпляр класса , который равен { \*this & bf }

TBitField operator~(void);

Назначение: оператор инверсии

Входные параметры отсутствуют

Выходные параметры: Экземпляр класса, каждый элемент которого равен {~\*this}, т.е. если i бит исходного экземпляра будет равен 1 , то на выходе он будет иметь 0.

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

Назначение: оператор ввода из консоли

Входные параметры:

&istr – буфер консоли

&bf – класс, который нужно ввести из консоли

Выходные параметры:

Ссылка на буфер (поток) &istr

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

Назначение: оператор вывода из консоли

Входные параметры:

&istr – буфер консоли

&bf – класс, который нужно вывести в консоль

Выходные параметры: Ссылка на буфер (поток) &istr

### 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 чисел.

Поля:

MaxPower – максимальный элемент множества.

BitField – экземпляр битового поля, на котором реализуется множество.

.

Методы:

TSet(int mp);

Назначение: конструктор с параметром, выделение памяти

Входные параметры:

mp – максимальный элемент множества.

Выходные параметры:

Отсутствуют

TSet(const TSet &s);

Назначение: конструктор копирования. Создание экземпляра класса на основе другого экземпляра

Входные параметры:

&s – ссылка, адрес экземпляра класса, на основе которого будет создан другой.

Выходные параметры:

Отсутствуют

~TSet();

Назначение: деструктор. Отчистка выделенной памяти

Входные и выходные параметры отсутствуют

int GetMaxPower(void) const;

Назначение: получение максимального элемента множества

Входные параметры отсутствуют

Выходные параметры: максимальный элемент множества

void InsElem(const int Elem)

Назначение: добавить элемент в множество

Входные параметры:

Elem - добавляемый элемент

Выходные параметры отсутствуют

void DelElem(const int Elem)

Назначение: удалить элемент из множества

Входные параметры:

Elem - удаляемый элемент

Выходные параметры отсутствуют

int IsMember(const int Elem) const;

Назначение: узнать, есть ли элемент в множестве

Входные параметры:

Elem - элемент, который нужно проверить на наличие

Выходные параметры: 1 или 0, в зависимости есть элемент в множестве, или нет

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

Назначение: оператор сравнения. Сравнить на равенство 2 множества

Входные параметры:

&s – битовое поле, с которым мы сравниваем

Выходные параметры: 1 или 0, в зависимости равны они, или нет соответственно

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

Назначение: оператор сравнения. Сравнить на равенство 2 множества

Входные параметры:

&s – битовое поле, с которым мы сравниваем

Выходные параметры: 0 или 1, в зависимости равны они, или нет соответственно

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

Назначение: оператор присваивания. Присвоить экземпляру \*this экземпляр &s

Входные параметры:

&s – множество , которое мы присваиваем

Выходные параметры: ссылка на экземпляр класса TSet, \*this

TSet operator+(const TSet &bf);

Назначение: оператор объединения множеств

Входные параметры:

&s - множество;

Выходные параметры: Экземпляр класса , который равен { \*this | s }

TSet operator\*(const TSet &bf);

Назначение: оператор пересечения множеств

Входные параметры:

&s - множество;

Выходные параметры: Экземпляр класса , который равен { \*this & s }

}

TBitField operator~(void);

Назначение: оператор дополнение до Универса

Входные параметры отсутствуют

Выходные параметры: Экземпляр класса, каждый элемент которого равен {~\*this}, т.е. если i элемент исходного экземпляра будет равен будет находится в множестве, то на выходе его не будет, и наоборот

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

Назначение: оператор ввода из консоли

Входные параметры:

&istr – буфер консоли

&s – класс, который нужно ввести из консоли

Выходные параметры:

Ссылка на буфер (поток) &istr

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

Назначение: оператор вывода из консоли

Входные параметры:

&istr – буфер консоли

&s – класс, который нужно вывести в консоль

Выходные параметры: Ссылка на буфер (поток) &istr

operator TBitField();

Назначение: вывод поля BitField

Входные параметры отсутствуют

Выходные данные: поле BitField

TSet operator+(const int Elem);

Назначение: оператор объединения множества и элемента. Данный оператор аналогичен метод добавления элемента в множество

Входные параметры:

Elem - число

Выходные параметры: исходный экземпляр класса, содержащий Elem

TSet operator+(const int Elem);

Назначение: оператор объединения множества и элемента. Данный оператор аналогичен методу удаления элемента из множества.

Входные параметры:

Elem - число

Выходные параметры: исходный экземпляр класса, не содержащий Elem

# Заключение

В ходе выполнения работы "Битовые поля и множества" были изучены и практически применены концепции битовых полей и множеств.

Были достигнуты следующие результаты:

1. Были изучены теоретические основы битовых полей и множеств.

2. Была разработана программа, реализующая операции над битовыми полями и множествами. В ходе экспериментов была оценена эффективность работы этих операций и сравнена с другими подходами. Результаты показали, что использование битовых полей и множеств позволяет существенно сократить объем памяти и ускорить операции над множествами.

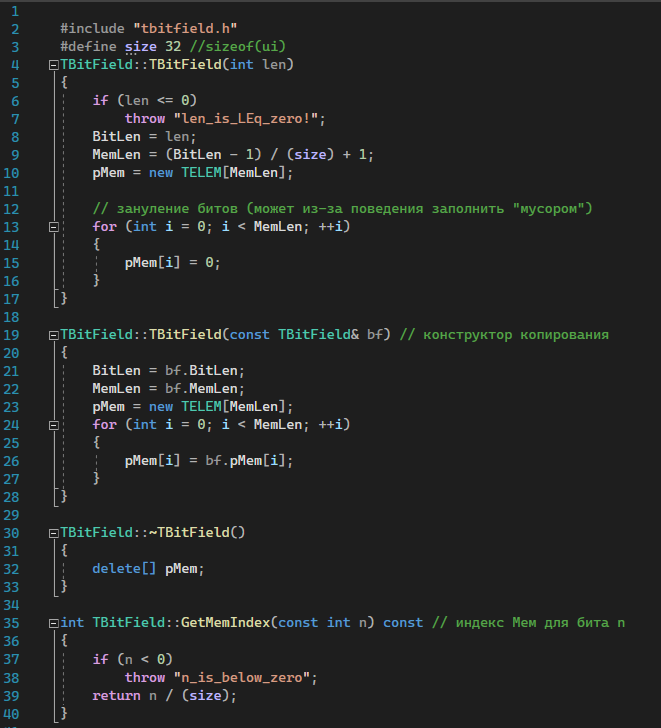
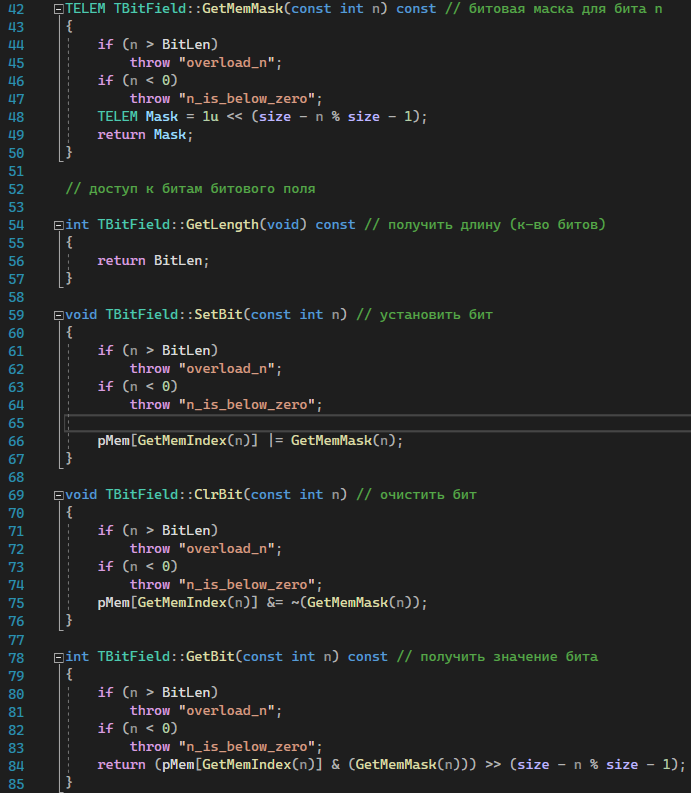
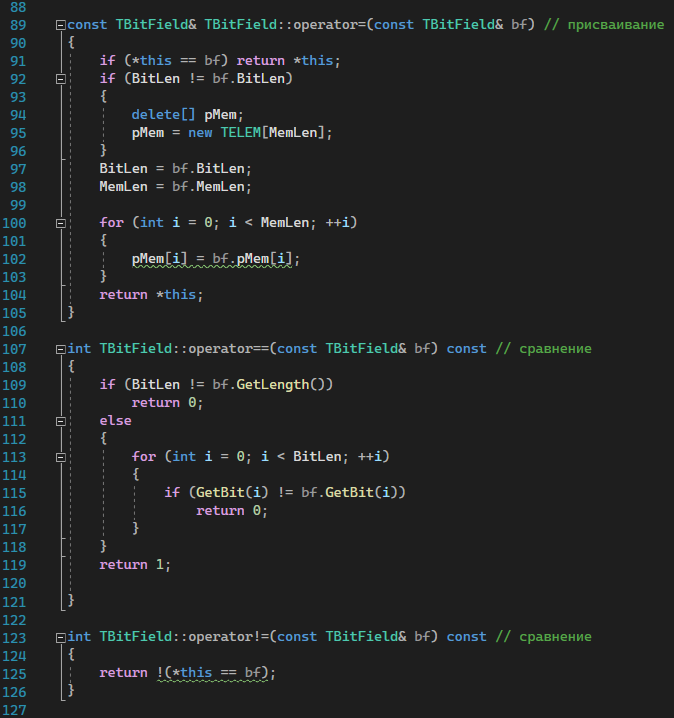
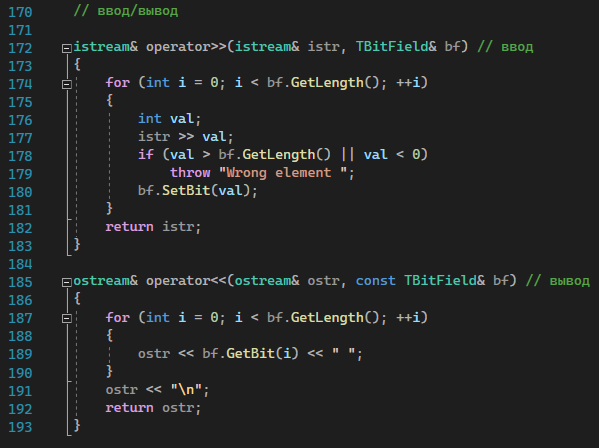
3. Были проанализированы полученные результаты и сделаны выводы о преимуществах и ограничениях использования битовых полей и множеств. Оказалось, что эти структуры данных особенно полезны при работе с большими объемами данных, где компактность представления и эффективность операций являются ключевыми факторами.

# Литература

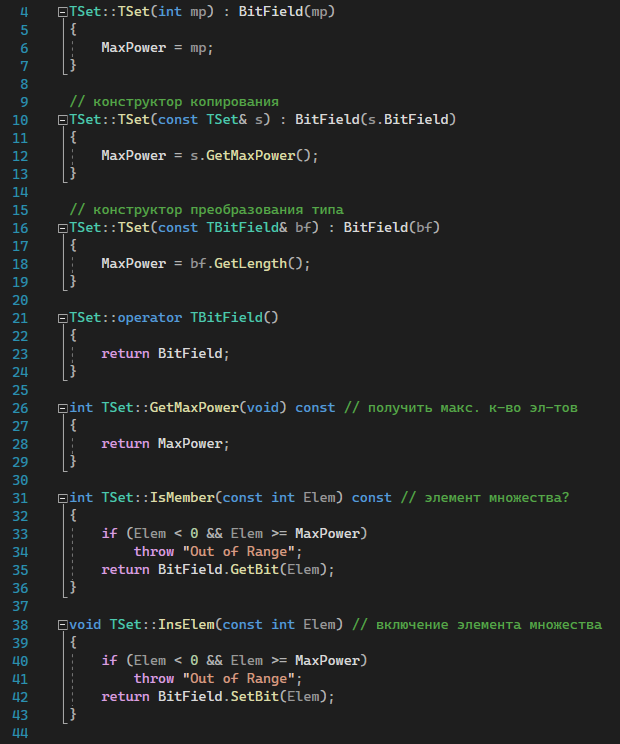
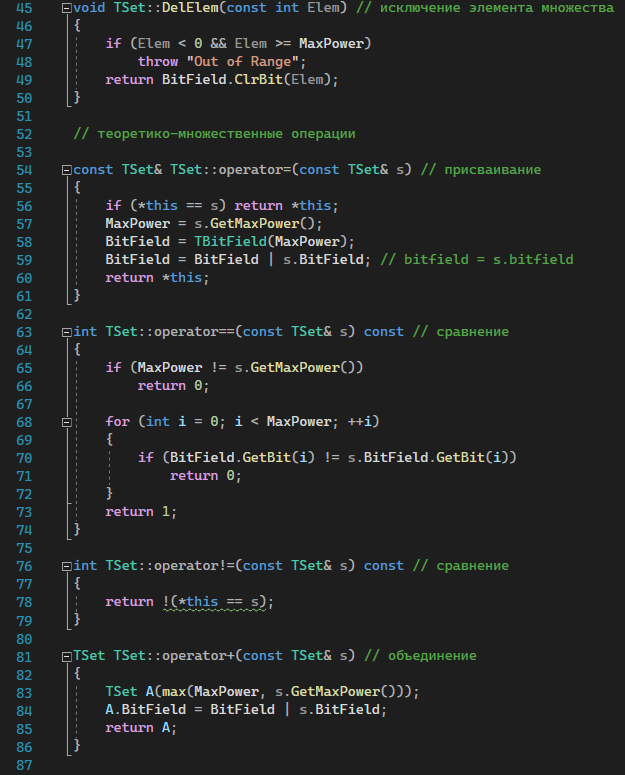
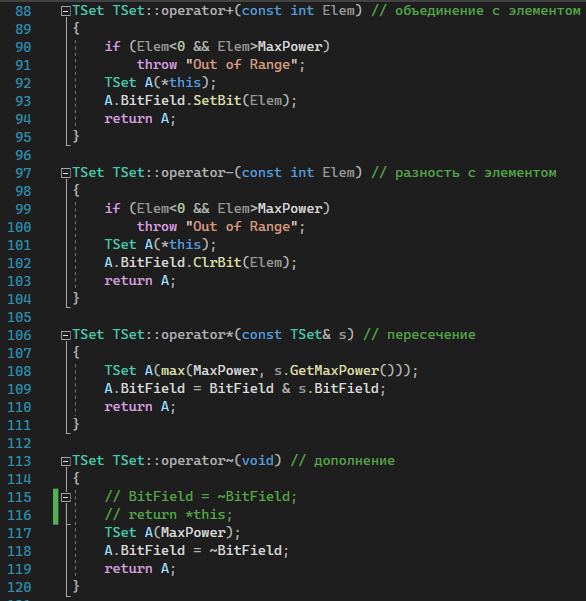
1. Википедия [https://ru.wikipedia.org/wiki/Битовое\_поле].

# Приложения

## Приложение А. Реализация класса TBitField

1. Реализация класса TBitField 1
2. Реализация класса TBitField 2
3. Реализация класса TBitField 3
4. Реализация класса TBitField 4
5. Реализация класса TBitField 5

## Приложение Б. Реализация класса TSet

1. Реализация класса Tset 1
2. Реализация класса TSet 2
3. Реализация класса TSet 3
4. Реализация класса TSet 4