МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

Высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

**Отчет по учебной практике**

**«Структура хранения множества: битовые поля»**

**Выполнил:** студент группы 381706-1

Нечаева Екатерина Владимировна

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

ассистент каф. МОСТ ИИТММ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лебедев И.Г

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc531274519)

[2. Постановка задачи 4](#_Toc531274520)

[3. Руководство пользователя 5](#_Toc531274521)

[4. Руководство программиста 7](#_Toc531274522)

[4.1. Описание структуры программы 7](#_Toc531274523)

[4.2. Описание структур данных 7](#_Toc531274524)

[4.3. Описание алгоритмов 8](#_Toc531274525)

[5. Заключение 10](#_Toc531274526)

[6. Литература 11](#_Toc531274527)

# Введение

Понятие множества, или совокупности, принадлежит к числу простейших математических понятий; оно не определяется, но может быть пояснено при помощи примеров. Так, можно говорить о множестве всех книг, составляющих данную библиотеку, множестве всех точек данной линии, множестве всех решений данного уравнения. Книги данной библиотеки, точки данной линии, решения данного уравнения являются элементами соответствующего множества.

Чтобы определить множество, достаточно указать характеристическое свойство

элементов, т. е. такое свойство, которым обладают все элементы этого множества и только

они.

Поскольку построение математической структуры основано на понятии множества, само множество не может быть определено, как структура данных. В дальнейших рассуждениях мы будем опираться на следующее описание множества, вытекающее из сделанных выше допущений.

Каждому множеству A поставим в соответствие характеристический вектор ,

Все операции над множествами могут быть заменены в таком случае на операции над характеристическими векторами или по-другому – над битовой строкой, которая хранится в битовом поле. Таким образом, в дальнейшем в работе мы будем решать задачу хранения и обработки именно характеристических векторов.

# Постановка задачи

В рамках лабораторной работы ставится задача разработки программы, поддерживающую эффективное хранение множеств, выполнение основных операций над множествами при помощи битового поля.

Для работы с Битовым полем предлагается реализовать следующие операции:

* установить бит (в единицу);
* очистить бит (в ноль);
* получить значение бита;
* сравнить два битовых поля;
* выполнить операцию “логическое или” для двух битовых полей;
* выполнить операцию “логическое и” для двух битовых полей;
* выполнить операцию “логическое отрицание” для битового поля.

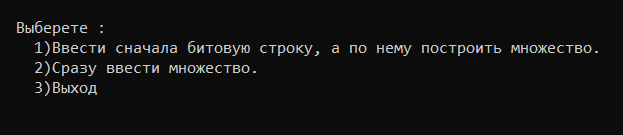
Для работы с Множеством предлагается реализовать следующие операции:

* включение элемента в множество;
* исключение элемента из множества;
* проверка наличия элемента в множестве;
* сравнение множеств
* сложение множеств;
* пересечение множеств;
* разность множества с элементом;
* копирование множества;
* вычисление максимальной мощности множества.

# Руководство пользователя

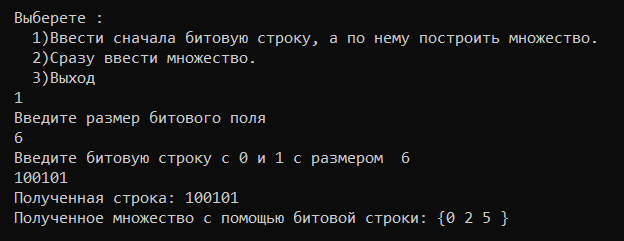
После запуска программы появляется меню, где пользователю предлагается выбор:

* Ввести битовую строку для вывода множества
* Ввести сразу множество



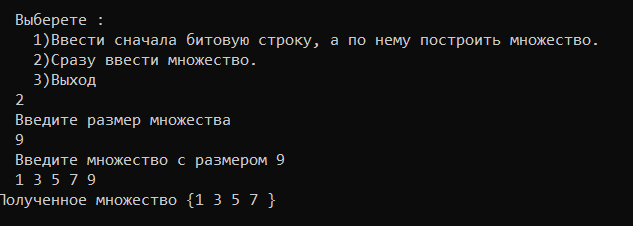
*Рисунок 1. Меню программы.*

Если пользователь выбирает первый пункт, ему предлагается сначала ввести размер битовой строки, а затем ввести ее саму. После этого выводится полученная строка и множество.

**

*Рисунок 2. Пример использования первого пункта меню программы.*

Если же пользователь выбрал второй пункт, ему также предлагается ввести размер множества, а затем ввести числа через пробел, но последним числом, чтобы закончить ввод чисел, нужно записать число равное размеру множества или больше его. Позже полученное множество выводится на экран.

**

*Рисунок 3. Пример использования второго пункта меню программы.*

Так как программа зациклена, пользователь может выбирать пункты 1 и 2 столько, сколько ему нужно, чтобы завершить работу нужно ввести цифру 3.

# Руководство программиста

## Описание структуры программы

Программа состоит из следующих модулей:

* Модуль set. Содержит заголовочные файлы tset.h и tbitfield.h, а также исполняемые tcet.cpp и tbitfield.cpp, в котором реализованы методы классов TSet и TBitField, описанные в заголовочных файлах.
* Модуль test\_set. Содержит тесты для классов, описанных выше.

## Описание структур данных

#### *Класс TBitField*

Поля класса:

* int BitLen; // длина битового поля - макс. к-во битов
* TELEM \*pMem; // память для представления битового поля
* int MemLen; // к-во эл-тов Мем для представления бит.поля

Приватные методы:

* int GetMemIndex(const int n) const; // индекс в pМем для бита n
* TELEM GetMemMask (const int n) const; // битовая маска для бита n

Публичные методы:

* TBitField(int len); // конструктор-инициализатор
* TBitField(const TBitField &bf); // конструктор копирования
* ~TBitField(); // деструктор
* int GetLength(void) const; // получить длину (к-во битов)
* void SetBit(const int n); // установить бит
* void ClrBit(const int n); // очистить бит
* int GetBit(const int n) const; // получить значение бита
* int operator==(const TBitField &bf) const; // сравнение
* int operator!=(const TBitField &bf) const; // сравнение
* TBitField& operator=(const TBitField &bf); // присваивание
* TBitField operator|(const TBitField &bf); // операция "или"
* TBitField operator&(const TBitField &bf); // операция "и"
* TBitField operator~(void); // отрицание
* friend istream &operator>>(istream &istr, TBitField &bf); // перегрузка ввода
* friend ostream &operator<<(ostream &ostr, const TBitField &bf); // перегрузка вывода

#### *Класс TSet*

Поля класса:

* int MaxPower; // максимальная мощность множества
* TBitField BitField; // битовое поле для хранения характеристического вектора

Публичные методы:

* TSet(int mp); // конструктор-инициализатор
* TSet(const TSet &s); // конструктор копирования
* TSet(const TBitField &bf); // конструктор преобразования типа
* operator TBitField(); // преобразование типа к битовому полю
* int GetMaxPower(void) const; // максимальная мощность множества
* void InsElem(const int Elem); // включить элемент в множество
* void DelElem(const int Elem); // удалить элемент из множества
* int IsMember(const int Elem) const; // проверить наличие элемента в множестве
* int operator== (const TSet &s) const; // сравнение
* int operator!= (const TSet &s) const; // сравнение
* TSet& operator=(const TSet &s); // присваивание
* TSet operator+ (const int Elem); // объединение с элементом
* TSet operator- (const int Elem); // разность с элементом
* TSet operator+ (const TSet &s); // объединение
* TSet operator\* (const TSet &s); // пересечение
* TSet operator~ (void); // дополнение
* friend istream &operator>>(istream &istr, TSet &bf); // перегрузка ввода
* friend ostream &operator<<(ostream &ostr, const TSet &bf); // перегрузка вывода

## Описание алгоритмов

В данном разделе будут описаны алгоритмы, применяемые для реализации некоторых методов класса TBuildField.

Битовую строку удобнее всего хранить в массиве. В битовой строке нумерация бит происходит слева направо, также, как и в массиве.

Для написания методов: установить, очистить бит и получить его значение, нужно было реализовать следующие методы:

* GetMemIndex: по формуле *n << 5* для n-го бита устанавливается индекс в массиве pMem, в котором хранится битовое поле.
* GetMemMask: по формуле *1 << (n & 31)* для n-го бита устанавливается битовая маска, то есть запись единичного бита на место, посчитанное данной формулой.

Тогда для:

* Установка бита: по формуле *pMem[GetMemIndex(n)] |= GetMemMask(n)* или по-другому побитовое ИЛИ между битами массива pMem и его битовой маски.
* Очистить бит: по формуле *pMem[GetMemIndex(n)] &= ~GetMemMask(n)* или по-другому побитовое И между битами массива pMem и инверсией его битовой маски.
* Получить значение бита: по формуле *pMem[GetMemIndex(n)] & GetMemMask(n)* или по-другому побитовое И между битами массива pMem и его битовой маски. Возвращает либо 0, либо 1.

# Заключение

В ходе выполнения данной лабораторной была реализована структура хранения множества с помощью битовых полей. Также были реализованы теоретико-множественные операции над множествами, а именно: сравнение, сложение, пересечение множеств, объединение и разность множества с элементом; были приведены формулы для установления бита на позицию и удаления бита в битовой маске.

# Литература

1. Учебно-методическое пособие [Электронный ресурс]

<http://www.unn.ru/books/met_files/Pract_ADS.pdf>

1. Битовые поля [Электронный ресурс]

<https://studfiles.net/preview/5086640/page:2/>

1. Структура хранения множества [Электронный ресурс]

<https://vikidalka.ru/2-117276.html>

1. Битовые операции [Электронный ресурс]

<https://tproger.ru/translations/bitwise-operations/>