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**

**«Множества на основе битовых полей»**

**Выполнил:** студент группы 381706-1

Шашкин Евгений Вадимович

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

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

Нижний Новгород

2018.

Оглавление

[1. Введение. 2](#_Toc532330636)

[2. Постановка задачи. 3](#_Toc532330637)

[3. Руководство пользователя. 4](#_Toc532330638)

[4. Руководство программиста. 5](#_Toc532330639)

[4.1. Описание структуры программы. 5](#_Toc532330640)

[4.2. Описание структур данных. 5](#_Toc532330641)

[4.3. Описание алгоритмов. 7](#_Toc532330642)

[5. Заключение. 9](#_Toc532330643)

[6. Литература. 10](#_Toc532330644)

# Введение.

Компьютерная программа, написанная на одном из огромного множества языков программирования, состоит из команд. Команды же, как правило, оперируют данными. Несмотря на то, что современные компьютеры обладают большим объемом памяти относительно своих предков, по-прежнему актуальна задача экономии памяти. Располагать данные в памяти компьютера более компактно помогают различные структуры данных.

**Структура данных** — программная единица, позволяющая хранить и обрабатывать множество однотипных и/или логически связанных данных в вычислительной технике.

В данной лабораторной работе рассматривается множество, построенное на битовом поле. Битовое поле — это **n**-ое количество бит, расположенных последовательно.

**Множество** — одно из ключевых понятий математики; это математический объект, сам являющийся набором, совокупностью, собранием каких-либо объектов, которые называются **элементами** этого множества и обладают общим для всех их характеристическим свойством. Изучением общих свойств множеств занимаются теория множеств, а также смежные разделы математики и математической логики.

Примерами множества в повседневной жизни может служить множество жителей города или множество слов в книге.

Множества на основе битовых полей выгодно использовать, когда:

* Не важна скорость доступа к данным.
* Необходим удобный доступ к отдельным битам в байте.
* Устройства передают информацию, закодировав биты в байт.
* Если количество памяти ограничено, то в одном байте можно хранить несколько переменных, принимающих значение TRUE и FALSE.

Битовые поля очень часто применяются к целочисленным данным.

# Постановка задачи.

Реализовать библиотеку для хранения и работы с множествами на основе битовых полей. Библиотека должны содержать:

* Класс битового поля TBitField, который содержит инструменты для хранения и работы с битовыми полями.
* Класс множества TSet, который содержит инструменты для хранения и работы с множествами.
* Тесты на базе Google Test, покрывающие классы TBitField и TSet.
* Пример использования библиотеки.

Так же, должны быть реализованы несколько простых тестов на базе Google Test.

# Руководство пользователя.

В качестве примера использования классов битового поля и множества рассмотрим решение задачи поиска простых чисел с помощью алгоритма «Решето Эратосфена».

При запуске программы от пользователя запрашивают ввести верхнюю границу целых значений. Затем происходит поиск и подсчет простых чисел и на экран выводятся:

* Множество некратных чисел (битовая строка простых чисел).
* Простые числа от 0 до значения, введённого пользователем.

Далее программа может работать в двух режимах:

* **Использование класса множеств.**

1. Пользователь вводит наибольший элемент множества.
2. Пользователь вводит набор чисел от 0 до наибольшего элемента.
3. На экран будет выведено полученное множество и его битовая строка.

* **Использование класса битовых полей.**

1. Пользователь вводит размер битовой строки.
2. Пользователь вводит свою битовую строку.
3. На экран будет выведена полученная битовая строка и множество чисел.

# Руководство программиста.

# Описание структуры программы.

Программа состоит из четырёх модулей:

1. Модуль **tset.**

Этот модуль состоит из заголовочного файла **tset.h**, в котором описывается интерфейс класса **TSet**, и файла **tset.cpp**, в котором содержится реализация методов класса **TSet**. В этом модуле реализованы такие методы, как: конструкторы (инициализатор, копирования и преобразования типа), деструктор, доступ к отдельным битам, перегрузки теоретико-множественных операторов (сравнение, объединение, присваивание, пересечение и дополнение), операторы ввода/вывода.

1. Модуль **tbitfield**.

Этот модуль состоит из заголовочного файла **tbitfield.h**, в котором описывается интерфейс класса **TBitField**, и файла **tbitfield.cpp**, в котором содержится реализация методов класса **TBitField**. В этом модуле реализованы такие методы, как: конструкторы (инициализатор и копирования), деструктор, доступ к отдельным битам, перегрузки битовых операторов (==, !=, =, |, &, ~), операторы ввода/вывода, методы получения индекса бита и его маски.

1. Пример использования.

Этот модуль содержит файл **sample\_prime\_numbers.cpp**. В нём описаны примеры использования битового поля и множества с использованием алгоритма поиска простых чисел «Решето Эратосфена».

К проекту **sample\_prime\_numbers** подключается библиотека, содержащая в себе модули **tset** и **tbitfield.**

1. Тесты.

Этот модуль содержит файлы **test\_tbitfield.cpp**, в котором находятся 29 тестов для методов класса **TBitField**, и **test\_tset.cpp**, в котором находятся 25 тестов для методов класса **TSet**.

# Описание структур данных.

**Класс TSet.**

Класс **TSet** содержит два поля со спецификатором доступа **private**:

* **int maxPower** – максимальная мощность множества.
* **TBitField bitField** – битовое поле для хранения характеристического вектора.

Далее идут методы класса со спецификатором доступа **public**:

* **TSet(int mp**) – конструктор-инициализатор, где **mp** – максимальная мощность множества.
* **TSet(const TSet &s)** – конструктор копирования, где **&s** – ссылка на объект класса **TSet**.
* **TSet(const TBitField &bf)** – конструктор преобразования типа, где **&bf** – ссылка на объект класса **TBitField**.
* **operator TBitField()** – преобразование типа к битовому полю.
* **int GetMaxPower(void) const** – возвращает максимальную мощность множества.
* **void InsElem(const int elem) –** включаетэлемент в множество.
* **void DelElem(cons tint elem)** – удаляет элемент из множества.
* **int IsMember(const int elem) const** – проверяет наличие элемента в множестве.
* **int operator== (const TSet &s) const** – оператор сравнения. Если два битовых поля равны – возвращает 1, иначе – 0.
* **int operator!= (const TSet &s) const** – оператор сравнения. Если два битовых поля не равны – возвращает 1, иначе – 0.
* **TSet& operator=(const TSet &s)** – оператор присваивания. Принимает ссылку на объект класса **TSet** и присваивает принятое множество к исходному.
* **TSet operator+ (const int elem)** –объединение с элементом. Добавляет **elem** к исходному множеству.
* **TSet operator- (const int elem)** – разность с элементом. Удаляет **elem** из исходного множества.
* **TSet operator+ (const TSet &s)** – объединение множеств. Принимает ссылку на объект класса **TSet**. Возвращает объект класса **TSet**, полученный в результате объединения полученного множества с исходным.
* **TSet operator\* (const TSet &s)** – пересечение множеств. Принимает ссылку на объект класса **TSet**. Возвращает объект класса **TSet**, полученный в результате пересечения полученного множества с исходным.
* **TSet operator~ (void)** – возвращает дополнение к исходному множеству.
* **friend istream &operator>>(istream &istr, TSet &bf)** – дружественная функция ввода множества с клавиатуры.
* **friend ostream &operator<<(ostream &ostr, const TSet &bf)** – дружественная функция вывода характеристического вектора множества.

**Класс TBitField.**

Класс **TBitField** содержит три поля и два метода со спецификатором доступа **private**:

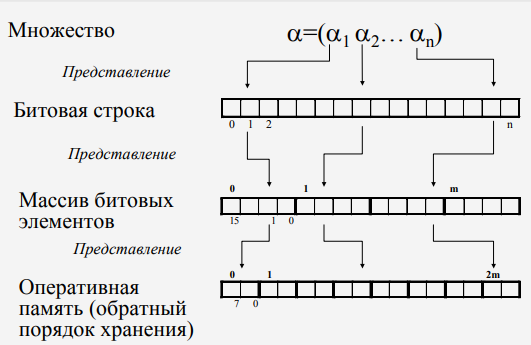
* **int bitLen** –длина битового поля- максимальное количество битов.
* **uInt \*pMem** –память для представления битового поля.
* **int memLen** –количество битов для представления битового поля.
* **int GetMemIndex(const int n) const** –метод, определяющий индекс бита n.
* **uInr GetMemMask (const int n) const** – метод, возвращающий маску для бита n.

Далее идут методы класса со спецификатором доступа **public**:

* **TBitField(int len)** – конструктор-инифиализатор.
* **TBitField(const TBitField &bf)** – конструктор копирования.
* **~TBitField()** – деструктор.
* **int GetLength(void) const** – получить длину (количество битов)
* **void SetBit(const int n)** – установить бит номер n.
* **void ClrBit(const int n**) – очистить бит номер n.
* **int GetBit(const int n) const** – получить значение бита номер n.
* **int operator==(const TBitField &bf) const** – оператор сравнения двух битовых полей. Если два битовых поля равны – возвращает 1, иначе – 0.
* **int operator!=(const TBitField &bf) const** – оператор сравнения двух битовых полей. Если два битовых поля не равны – возвращает 1, иначе – 0.
* **TBitField& operator=(const TBitField &bf)** – оператор присваивания. Принимает ссылку на объект класса **TBitField** и присваивает принятое битовое поле к исходному.
* **TBitField operator|(const TBitField &bf)** – оператор "или". Принимает ссылку на объект класса **TBitField**. Возвращает объект класса **TBitField**, полученный в результате применения к двум битовым полям операции «ИЛИ».
* **TBitField operator&(const TBitField &bf)** – оператор "и". Принимает ссылку на объект класса **TBitField**. Возвращает объект класса **TBitField**, полученный в результате применения к двум битовым полям операции «И».
* **TBitField operator~(void)** – отрицание. Возвращает объект класса **TBitField**, полученный в результате использования побитового отрицания для исходного битового поля.
* **friend istream &operator>>(istream &istr, TBitField &bf)** – дружественная функция ввода битового поля с клавиатуры.
* **friend ostream &operator<<(ostream &ostr, const TBitField &bf)** –дружественная функциявывода битового поля.

# Описание алгоритмов.

Рассмотрим алгоритм формирования битовых полей.



*Рис. 1 – представление битового поля.*

На вход подаётся множество натуральных чисел. Если элемент присутствует в множестве, то бит с этим номером принимает значение 1, если нет – 0.

Битовую строку удобно хранить в массиве. Но в массиве нумерация элементов, как и нумерация бит, происходит справа налево, а биты отдельного элемента массива нумеруются справа налево.

Для упрощения и увеличения скорости работы таких методов класса **TBitField**, как получить бит, очистить бит, положить бит, были реализованы вспомогательные функции:

1. **uInt GetMemMask (const int n) const** – возвращает число, которое в двоичном виде представляет собой битовую маска для бита номер n.

Битовая маска – десятичное число, в двоичном представлении которого, на месте искомого бита стоит 1, а на всех остальных разрядах – 0.

Маска вычисляется по формуле:

1. **Int GetMemIndex(const int n) const** – возвращает индекс элемента для бита номер n.

Этот индекс вычисляется по формуле: (uInt – целочисленный беззнаковый тип данных).

Эти методы упрощают реализацию следующих функций:

1. **void SetBit(const int n)** – устанавливает бит номер n.

Чтобы установить бит в n-ую позицию, нужно выполнить побитовую операцию «ИЛИ» для текущего битового поля и маски для бита номер n.

**pMem[GetMemIndex(n)] |= GetMemMask(n)**.

1. **void ClrBit(const int n)** – удаляет бит номер n.

Чтобы удалить бит из n-ой позиции, нужно выполнить побитовую операцию «И» для битового поля и дополнения для битовой маски бита номер n.

**pMem[GetMemIndex(n)] &= ~GetMemMask(n)**.

1. **int GetBit(const int n) const** – считывает значение бита номер n.

Чтобы считать бит из n-ой позиции, нужно выполнить побитовую операцию «И» для битового поля и битовой маски бита номер n.

**return (pMem[GetMemIndex(n)] & GetMemMask(n))**.

# Заключение.

В данной лабораторной работе я впервые писал собственную структуру данных, а так же тестировал её с помощью Google Test. В ходе работы я познакомился с такими понятиями, как:

* Битовое поле.
* Характеристический вектор.
* Множество на основе битового поля.
* Битовый формат элемента массива.
* Битовая строка в виде массива.

Таким образом, мне удалось реализовать библиотеку для работы с такой структурой данных, как множество на основе битовых полей, а так же мной были получены навыки тестирования с использованием библиотеки Google Test.

# Литература.

1. Ссылка из Википедии про структуры данных:

<https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85>

1. Ссылка из Википедии про множества:

<https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%BE>

1. Ссылка из Википедии про битовые поля:

<https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82%D0%BE%D0%B2%D0%BE%D0%B5_%D0%BF%D0%BE%D0%BB%D0%B5>

1. Гергель В.П. Методические материалы по курсу «Методы программирования 2», 2015.