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**

оставь надежды всяк сие читающий (карин т.а.)

**Отчет по учебной практике**

**«Множества на основе битовых полей»**

**Выполнил:**

студент группы 381706-­1

Карин Тимофей Андреевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Проверил:**

ассистент каф. МОСТ ИИТММ

Лебедев Илья Геннадьевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc1375215)

[3. Руководство пользователя 5](#_Toc1375216)

[4. Руководство программиста 6](#_Toc1375217)

[4.1. Описание структуры программы 6](#_Toc1375218)

[4.2 Описание структур данных 6](#_Toc1375219)

[4.3 Описание алгоритмов 8](#_Toc1375220)

[5. Заключение 11](#_Toc1375221)

[6. Список литературы 12](#_Toc1375222)

1. Введение

Множеством в математике называется любая совокупность объектов, выбранная из универсального множества. Универсальным при этом называется множество, содержащее стразу все рассматриваемые элементы. Элементами могут быть данные различных типов, число элементов не ограничено, одинаковых элементов нет.

Мы будем рассматривать множества на примере битовых полей. Что же такое битовое поле? Битовое поле ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) bit field) в [программировании](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) — некоторое количество [бит](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82), расположенных последовательно в [памяти](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C), значение которых [процессор](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) не способен прочитать из-за особенностей аппаратной реализации. Для чего они могут понадобиться? Битовые поля применяются для максимально полной упаковки [информации](https://ru.wikipedia.org/wiki/%D0%94%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5), если не важна скорость доступа к этой информации. Например, для увеличения [пропускной способности](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%BF%D1%83%D1%81%D0%BA%D0%BD%D0%B0%D1%8F_%D1%81%D0%BF%D0%BE%D1%81%D0%BE%D0%B1%D0%BD%D0%BE%D1%81%D1%82%D1%8C) [канала](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D0%BD%D0%B0%D0%BB_%D0%BF%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) при передаче информации по [сети](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C) или для [уменьшения размера](https://ru.wikipedia.org/wiki/%D0%A1%D0%B6%D0%B0%D1%82%D0%B8%D0%B5_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) информации при хранении.

В нашем случае битовое поле будет состоять из набора целых чисел типа integer (int), каждое из которых состоит из 32 бит. То есть в одном числе типа int может поместиться 32 бита.

Как битовые поля связаны с множествами? Каждым битовым полем можно однозначно обозначить конкретное множество. Если порядковый номер бита принадлежит множеству, то этому биту присваивается значение «1», во всех остальных битах (номера которых не принадлежат множеству) устанавливается значение «0».

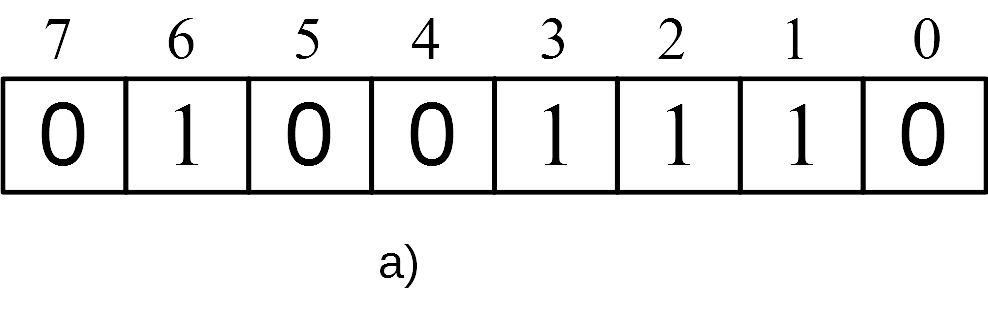
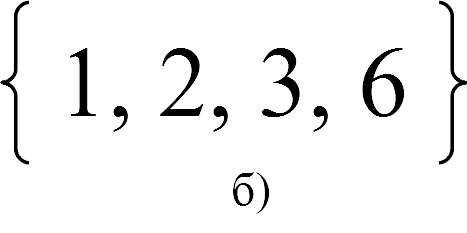
Например: битовое поле на рисунке 1а) задаёт множество, представленное на рисунке 1б.

Рисунок 1. Соответствие битового поля и множества.

Этот пример наглядно показывает, как битовое поле способно значительно сэкономить память. Если бы мы представляли каждый элемент представленного множества отдельным числом типа int, то мы бы потратили 16 байт памяти. Получается, использовав битовые поля, мы потратили в 16 раз меньше памяти.

2. Постановка задачи

Цель данной работы — разработка структуры данных для хранения множеств с использованием битовых полей, а также освоение таких инструментов разработки программного обеспечения, как система контроля версий Git и фрэймворк для разработки автоматических тестов Google Test.

Выполнение работы предполагает решение следующих задач:

* Реализация класса битового поля TBitField согласно заданному интерфейсу.
* Реализация класса множества TSet согласно заданному интерфейсу.
* Обеспечение работоспособности тестов и примера использования.
* Реализация нескольких простых тестов на базе Google Test.
* Публикация исходных кодов в личном репозитории на GitHub.

3. Руководство пользователя

Данная лабораторная работа направлена на создание библиотеки для работы с битовыми полями и множествами и не предназначена для работы пользователей. Однако, пользователь может выступить в роли тестировщика. Проект *check\_cin\_cout* предназначен для тестирования функций ввода/вывода битовых полей и множеств.

Опишем, как именно устроен данный тест. Сначала пользователю необходимо ввести длину создаваемого битового поля. Затем, это поле необходимо ввести. Основные принципы ввода битового поля:

* Вводить необходимо строку из символов «0» и «1»;
* Длина строки не может быть больше размера битового поля;
* Нумерация битов начинается с начала строки. (То есть первый символ обозначает первый бит (точнее нулевой));
* Любой символ, кроме «0», воспринимается программной как символ «1»;
* Если строка короче размера битового поля, заполняются соответствующие первые элементы, а оставшиеся биты заполняются нулями.

Если всё сделать правильно, программа должна вывести результат того, что ввёл пользователь. Вывод производится в строки по 32 символа, каждые 8 символов разделены пробелом (исключительно для удобства чтения). Нумерация опять же начинается с начала.

Затем пользователю необходимо определить, соответствует ли выведенная запись тому, что он вводил. Если ответ положительный, то ему необходимо ввести символ «1» и приступить к следующему тесту. Если данные не совпадают или ничего не вывелось, то символы «2» и «3» соответственно. Естественно, в этом случае тест считается не пройденным, и программа завершает работу.

Если тест всё же пройден, то программа предложит протестировать ввод/вывод множеств. Он выполняется аналогично. Опишем лишь правила ввода множества:

* Ввод производится поэлементно;
* Выход из режима ввода осуществляется введением некорректного элемента (например, отрицательного или выходящего за границы размера множества).
* Порядок элементов не важен, элементы могут повторяться.

4. Руководство программиста

4.1. Описание структуры программы

Программа состоит из нескольких модулей:

* *set –* содержит исходный код битовых полей и множеств. Состоит из заголовочных файлов *tbitfield.h* и *tset.h* и соответствующих им файлов исходного кода. Эти файлы содержат соответственно описание и реализацию классов *TBitField* (битовое поле) и *TSet* (множество).
* *test\_set –* содержит тесты. Состоит из файлов test\_tbitfield и test\_tset, в которых находятся тесты (на базе фреймворка Google test) для битовых полей и множеств соответственно.
* *check\_cin\_cout –* предназначен для тестирования операторов ввода/вывода*.* Сами тесты подробно описаны в главе 3.
* *sample\_prime\_numbers –* сожержит файл sample\_prime\_numbers.cpp, предназначенный для поиска простых чисел по алгоритму «Решето Эратосфена». (Подробно этот модуль описывать не буду, так как к его написанию не имею никакого отношения. Скажу лишь, что он работает).

4.2 Описание структур данных

Рассмотрим класс *TBitField:*

Со спецификатором доступа private:

* *int bitLen* – длина битового поля - макс. к-во битов
* *TELEM \*pMem* – память для представления битового поля
* *int memLen* – количество элементов Мем для представления битового поля
* *int GetMemIndex(const int n) const* – индекс в pМем для бита n
* *TELEM GetMemMask (const int n) const* – битовая маска для бита n

Со спецификатором доступа public:

* *TBitField(int len)* – конструктор;
* *TBitField(const TBitField &bf)* – конструктор копирования;
* *~TBitField()* – деструктор;
* *int GetLength(void) const* – получить длину (количествово битов);
* *void SetBit(const int n)* – установить бит (присвоить биту n значение «1»);
* *void ClrBit(const int n)* – очистить бит (присвоить биту n значение «0»);
* *int GetBit(const int n) const* – получить значение бита;
* *int operator==(const TBitField &bf) const* – проверка на равенство (возвращает «1», если все биты равны. Во всех остальных случаях возвращает «0»);
* *int operator!=(const TBitField &bf) const* – проверка на неравенство (возвращает «0», если все биты равны. Во всех остальных случаях возвращает «1»);
* *TBitField& operator=(const TBitField &bf)* – оператор присваивания;
* *TBitField operator|(const TBitField &bf)* – операция "или";
* *TBitField operator&(const TBitField &bf)* – операция "и";
* *TBitField operator~(void)* – отрицание (инвертирование);
* *friend istream &operator>>(istream &istr, TBitField &bf)* – оператор ввода;
* *friend ostream &operator<<(ostream &ostr, const TBitField &bf)* – оператор вывода;

Рассмотрим класс *TSet*:

Со спецификатором доступа private:

* *int maxPower* – максимальная мощность множества;
* *TBitField BitField* – битовое поле для хранения характеристического вектора;

Со спецификатором доступа public:

* *TSet(int mp)* – конструктор;
* *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* – проверка на равенство;
* *int operator!= (const TSet &s) const* – проверка на неравенство;
* *TSet& operator=(const TSet &s)* – оператор присваивания;
* *TSet operator+ (const int Elem)* – объединение с элементом (элемент должен быть из того же универса);
* *TSet operator- (const int Elem)* – разность с элементом (элемент должен быть из того же универса);
* *TSet operator+ (const TSet &s)* – объединение;
* *TSet operator\* (const TSet &s)* – пересечение;
* *TSet operator~ (void)* – дополнение;
* *friend istream &operator>>(istream &istr, TSet &bf)* – оператор ввода;
* *friend ostream &operator<<(ostream &ostr, const TSet &bf)* – оператор вывода;

4.3 Описание алгоритмов

Битовая маска

Битовая маска представляем из себя элемент битового поля, состоящий из нулей и одной единицы, расположенной в том разряде, для бита которого мы и получаем битовую маску. То есть получается, что нужно единицу сдвинуть на количество разрядов, равное номеру бита, для которого получаем маску. Количество разрядов, на которое необходимо сдвинуть единицу определяется как результат логического «И» между номером бита и элементом битового поля, состоящим только из единиц (то есть *n & 31*).

Логическое «ИЛИ»

Создаём временное битовое поле, длина которого равна длине наименьшего из двух данных полей. Заполняем созданное поле элементами первого битового поля насколько это возможно. Затем, поочерёдно проводим операцию логического «или» между элементами созданного поля и второго поля. Результат записываем в созданное поле. В результате, в нём будет храниться результат логического «ИЛИ» для битовых полей.

Логическое «И»

Создаём временное битовое поле с длиной, равной длине того битового поля, которое короче. Затем в созданное битовое поле записываем значения первого битового поля. Затем с помощью операции логического «И» для целых чисел записываем в созданное битовое поле результат.

Установить бит

Если бит, пришедший в качестве параметра выходит за границы поля, то выбрасываем исключение. Если нет, то получаем битовую маску для этого бита и проводим операцию логического «ИЛИ» между битовой маской и элементов битового поля *Mem,* содержащим этот бит.

Очистить бит

Находим битовую маску для данного бита и инвертируем её. Проводим операцию логического «И» между инвертированной маской и элементом битового поля *Mem*, в котором содержится данный бит.

Получить значение бита

Если бит не принадлежит битовому полю, то выбрасываем исключение. Если принадлежит, то находим результат логического «И» между битовой маской данного бита и элемента *Mem,* в котором он находится. Если бит установлен, то в результате получим битовую маску, если нет, то получим ноль. В любом случае, сдвинув результат на количество разрядов, равное номеру бита (а точнее *n & 31*), получим верный ответ.

С множествами всё проще:

Включению элемента в множество соответствует функция установки бита в битовое поле;

Исключению элемента из множества соответствует функция очистки бита в битовом поле;

Объединению множеств соответствует функция логического «ИЛИ» между битовыми полями;

Пересечению множеств соответствует функция логического «И» между битовыми полями этим множеств.

5. Заключение

В ходе проделанной работы были получены следующие результаты:

* Выполнены все поставленные задачи;
* Изучены битовые поля, способ их реализации, методы работы с ними;
* Получены навыки работы с битовыми операциями;
* Изучены методы хранения и использования множеств;
* Получены первоначальные навыки работы с системой контроля версий GitHub и фреймворком для написания тестов Google Test.

6. Список литературы

1. Ахо Альфред В, Хопкрофт Джон Э и Ульман Джеффри Д Структуры данных и алгоритмы [Книга]. - [б.м.] : Вильямс, 2003.
2. Лафоре Роберт Структуры данных и алгоритмы в Java [Книга]. - СПб : Питер, 2013. - 2 : стр. 704.
3. Павловская Т. А. C/C++ Программирование на языке высокого уровня [Книга]. - СПб : Питер, 2003.
4. Страуструп Бьерн Язык программирования C++ Бином, 2004.