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**

**«Разработка структуры хранения матриц специального вида»**

**Выполнил:** студент группы 381903-3

Балашов Максим Алесеевич

**Научный руководитель:**

ассистент каф. МОСТ ИИТММ

Лебедев Илья Геннадьевич

Нижний Новгород

2020

Содержание

[Введение 3](#_Toc23876641)

[1. Постановка задачи 4](#_Toc23876642)

[2. Руководство пользователя 5](#_Toc23876643)

[3. Руководство программиста 6](#_Toc23876644)

[3.1. Описание структуры программы 6](#_Toc23876645)

[3.2. Описание алгоритмов 7](#_Toc23876646)

[Заключение 16](#_Toc23876647)

[Литература 17](#_Toc23876648)

# Введение

**Множество** — одно из ключевых понятий математики; это [**математический объект**](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82), сам являющийся набором, совокупностью, собранием каких-либо объектов, которые называются элементами этого множества и обладают общим для всех их характеристическим свойством. Для того, чтобы понять и разобраться, как устроено множество, была написана программа.

В формальной математике множество — обычно аксиоматическое понятие, и у него нет определения как такового. И его действительно сложно дать — ведь под множеством мы подразумеваем любой набор любых объектов. Этот набор может быть пустым, конечным или бесконечным; он может содержать натуральные, вещественные числа, [матрицы](http://math.siomax.ru/Matrices), целые [последовательности](http://math.siomax.ru/Sequences), мыслимые как самостоятельные объекты, или даже другие множества, такие, как отрезки и лучи вещественной прямой (или даже вся вещественная прямая), а также стулья, столы, вилки, яблоки, торшеры, турнюры и так далее. Притом не налагается абсолютно никаких ограничений на однородность объектов, содержащихся в множестве — одно множество может содержать одновременно как числа, так и, например, другие множества. Объекты в множестве также по умолчанию никак не упорядочены (если не указано обратное, смотри далее про упорядоченные множества). Кстати, объекты, содержащиеся в множестве, обычно называются его элементами.

Основоположник теории множеств, великий немецкий математик Георг Кантор говорил, что множество – это многое, мыслимое как единое*.*

# Постановка задачи

Разработать структуру данных **множество** поддерживающую эффективное хранение множеств и выполняющую основные операций над множествами, освоить инструменты разработки программного обеспечения, такие как Git и Google Test.

Выполнение работы предполагает решение следующих задач:

1. Разработка интерфейса класса **TBitField.**

2. Реализация методов класса **TBitField**.

3. Разработка интерфейса класса **TSet**.

4. Реализация методов класса **TSet**.

5. Реализация нескольких простых тестов на базе **Google Test.**

# Руководство пользователя

На старте программы, нам предлагается ввести верхнюю границу множества( число , которого будет идти поиск простых чисел)

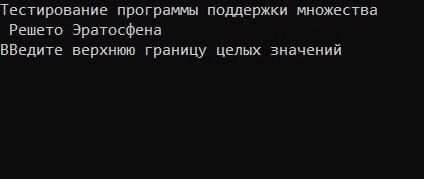
****

Рис1. Старт работы

После корректного ввода границы, программа считает и печатает нам простые числа. Ниже ,подсчитывает сколько простых чисел в указанном диапазоне.

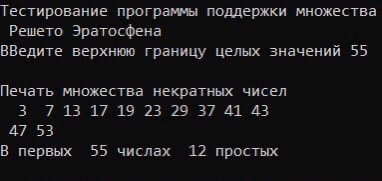
****

Рис2. Ввод и подсчет чисел

## Руководство программиста

## Описание структуры программы

Программа написана на языке программирования C++. Программа состоит из 5 файлов: TBitField.h, TBitField.cpp, TSet.h, TSet.cpp, main.cpp .

В программе реализованы следующие классы:

1. **Класс «TBitField»**

Описание: реализует структуру хранения битовых полей

* int TBitLen // длина битового поля
* PP\* pMem (insigned int)// динамическая память битового поля
* int MemLen // количество элементов битового поля
* TBitField (int len) //конструктор
* TBitField(const BitField& bf)
* ~TBitField() //деструктор
* int GetLength(void) const // возвращает длину строки
* void SetBit(const int n)// устанавливает бит из n позиции
* void ClrBit(const int n)// чистит бит
* int GetBit(const int n) const// получает значение бита
* int operator== (const TBitField& bf)//сравнение
* TBitField& operator=(const TBitField& bf)//присваивание
* TBitField& operator|(const TBitField& bf)// операция пересечения (ИЛИ)
* TBitField& operator&(const TBitField& bf)// операция объединения (И)
* TBitField& operator~(void)//операция логического отрицания (НЕ)
* friend istream& operator>>(istream& istr, TBitField& bf)// ввод
* friend ostream& operator<<(ostream& ostr, const TBitField&bf)//вывод

В следующей таблице указано применение данных и методов класса.

1. **Класс «Set»**

Описание: модуль с классом, реализующий обработку Множеств

* int MaxPower// максимальная мощность множества
* TBitField BF// битовое поле для хранения характеристического вектора
* int GetMaxPower(void) const;// получение максимально мощности множества
* void InsElem(const int n) // включить элемент в множество
* void DelElem(const int n)// удаление элемента в множестве
* int IsMember(const int n) const// проверка наличия элементов в множестве
* int operator== (const Set& s)//сравнение
* Set& operator=(const Set& s)//присваивание
* Set operator+ (const Set& s)//объединение
* Set operator\* (const Set& s)//пересечение
* Set operator~ (void)//дополнение
* friend istream& operator>>(istream& istr, Set& bf)//ввод
* friend ostream& operator<<(ostream& ostr, const Set&)//вывод

В следующей таблице указано применение данных и методов класса.

## Описание алгоритмов

В классе «TBitField» реализованы методы для работы с битовыми полями.

1) Метод «GetLength»

Данный метод возвращает значение длинны битового поля.

Ниже приведен псевдокод алгоритма:

**Функция GetLength (void) const**

{

**вернуть** TBitLen;

}

2) Метод «TSetBit»

Данный метод позволяет установить бит (т.е. 1) в позицию n.

Ниже приведен псевдокод алгоритма:

**Функция TSetBit (const int n)**

{

**если**  ((n > -1) **и** (n < TBitLen)) **тогда**

pMem[GetMemIndex(n)] |= GetMemMask(n);

}

3) Метод «ClrBit »

Данный метод позволяет очистить бит в позиции n.

Ниже приведен псевдокод алгоритма:

**Функция ClrBit (const int n)**

{

**если**  ((n > -1) **и** (n < BitLen)) **тогда**

pMem[GetMemIndex(n)] &=~GetMemMask(n);

}

4) Метод «GetBit»

Данный метод позволяет получить значение бита в позиции n.

Ниже приведен псевдокод алгоритма:

**Функция GetBit (const int n) const**

{

**если** ((n > -1) **и** (n < BitLen)) **тогда**

**вернуть** pMem[GetMemIndex(n)] & GetMemMask(n);

**вернуть** 0;

}

5) Метод «operator=»

Данный метод перегружает операцию присваивания (знак «=»).

Ниже приведен псевдокод алгоритма:

**Функция operator=(const TBitField &bf)**

{

TBitLen = bf.TBitLen;

**если** (MemLen **неравно** bf.MemLen) **тогда**  {

MemLen = bf.MemLen;

**если** (pMem **неравно** NULL) **тогда**

**удалить** pMem;

pMem = **создать память** UI[MemLen];

}

**если** (pMem **неравно** NULL) **тогда**

**цикл от** int i = 0; **до** i < MemLen; **с шагом** i++)

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

**вернуть неявный указатель на объект**;

}

6) Метод «operator==»

Данный метод перегружает операцию сравнения и возвращает 1 – истина или 0 - ложь (знак «==»).

Ниже приведен псевдокод алгоритма:

**Функция operator==(const TBitField &bf)**

{

int res = 1;

**если** (TBitLen неравно bf.TBitLen) **тогда**

res = 0;

**иначе**

**цикл от** int i=0; **до** i<MemLen; **с шагом** i++

**если**  (pMem[i] **неравно** bf.pMem[i]) **тогда**  {

res = 0;

**прекрати**;

}

**вернуть** res;

}

7) Метод «operator|»

Данный метод перегружает операцию «ИЛИ» (знак «|»).

Ниже приведен псевдокод алгоритма:

**Функция operator|(const TBitField &bf)**

{

int i, len = BitLen;

**если** (bf.BitLen > len) **тогда**

len = bf.BitLen;

BitField temp(len);

**цикл от** i = 0; **до** i < MemLen; **с шагом** i++)

temp.pMem[i] = pMem[i];

**цикл от** i = 0; **до** i < bf.MemLen; **с шагом** i++)

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

**вернуть** temp;

}

8) Метод «operator&»

Данный метод перегружает операцию «И» (знак «&»).

Ниже приведен псевдокод алгоритма:

**Функция operator& (const TBitField &bf)**

{

int i, len = TBitLen;

**если** (bf.TBitLen > len) **тогда**

len = bf.TBitLen;

TBitField temp(len);

**цикл от** int i = 0; **до** i < MemLen; **с шагом** i++)

temp.pMem[i] = pMem[i];

**цикл от** i = 0; **до** i = bf.MemLen; **с шагом** i++)

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

**вернуть** temp;

}

9) Метод «operator~»

Данный метод перегружает операцию «отрицание» (знак «~»).

Ниже приведен псевдокод алгоритма:

**Функция operator~ (void)**

{

int i, len = TBitLen;

TBitField temp(len);

**цикл от** int i = 0; **до** i < MemLen; **с шагом** i++)

temp.pMem[i] = ~pMem[i];

**вернуть** temp;

}

В классе «TSet» реализованы методы для работы с элементами множества и множествами.

1) Метод «GetMaxPower»

Данный метод возвращает максимальную мощность множества.

Ниже приведен псевдокод алгоритма:

**Функция GetMaxPower(void) const**

{

**вернуть** MaxPower;

}

2) Метод «IsMember»

Данный метод проверяет наличие элемента в множестве, возвращает работу метода GetBit .

Ниже приведен псевдокод алгоритма:

**Функция IsMember(const int Elem) const**

{

**верунть** BF.GetBit(Elem);

}

3) Метод «InsElem»

Данный метод позволяет включить элемент в множество.

Ниже приведен псевдокод алгоритма:

**Функция InsElem(const int Elem)**

{

BF.TSetBit(Elem);

}

4) Метод «DelElem»

Данный метод позволяет удалить элемент из множества.

Ниже приведен псевдокод алгоритма:

**Функция DelElem(const int Elem)**

{

BF.ClrBit(Elem);

}

5) Метод «operator=»

Данный метод перегружает операцию присваивания (знак «=»).

Ниже приведен псевдокод алгоритма:

**Функция operator=** **(const Set &s)**

{

BF = s.BF;

MaxPower = s.GetMaxPower();

**вернуть** **неявный указатель на объект**;

}

6) Метод «operator==»

Данный метод перегружает операцию сравнения и возвращает 1 – истина или 0 - ложь (знак «==»).

Ниже приведен псевдокод алгоритма:

**Функция operator==(const Set &s)**

{

**вернуть** BF == s.BF;

}

7) Метод «operator+»

Данный метод перегружает операцию «объединение» (знак «+»).

Ниже приведен псевдокод алгоритма:

**Функция operator+(const Set &s)**

{

Set temp(BF | s.BF);

**вернуть** temp;

}

8) Метод «operator\*»

Данный метод перегружает операцию «пересечение» (знак «\*»).

Ниже приведен псевдокод алгоритма:

**Функция operator\* (const Set &s)**

{

Set temp(BF & s.BF);

**вернуть** temp;

}

9) Метод «operator~»

Данный метод перегружает операцию «дополнение» (знак «~»).

Ниже приведен псевдокод алгоритма:

**Функция operator~ (void)**

{

Set temp(~BF);

**вернуть** temp;

}

# 

# Заключение

При выполнении лабораторной работы были реализованы два класса **BitField** и **Set** позволяющие выполнять различные операции над множествами и получены навыки работы с Google Tests и Cmake.

# Литература

1. Лафоре Р. Объектно-ориентированное программирование в C++. Классика Computer Science. 4-е изд. – СПБ.: Питер, 2018. – 928 с.
2. Страуструп Бьерн Язык программирования C++. Специальное издание. Пер. с англ. – М.: Издательство Бином, 2017 г. – 1136 с.
3. Шилдт Г. С++ Базовый курс. 3-е изд. – М.: Издательство Вильямс, 2018. 624 с.