Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение

высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Факультет Информационных технологий, математики и механики

Отчёт по лабораторной работе

**Множества на битовых полях**

Выполнил:

студент ИТММ гр. 381808-2

Некравцев А. А.

Проверил:

Нижний Новгород

2019 г.

**Содержание**

Введение ….............................................................................................................3

Постановка задачи ….............................................................................................4

Руководство пользователя …................................................................................5

Руководство программиста …..............................................................................6

Описание структуры программы ….........................................................6

Описание структур данных …..................................................................7

Описание алгоритмов …...........................................................................8

Заключение …......................................................................................................10

Приложение ….....................................................................................................11

**Введение**

В лабораторной работе представлена программная реализация множества на основе битового поля.

Множество - это структура данных, позволяющая хранить ограниченное число значений определённого типа без определённого порядка.

Битовое поле - некоторое количество бит, расположенных последовательно в памяти, значение которых процессор не способен прочитать из-за особенностей аппаратной реализации.

Преимущества именно такой структуры для хранения данных:

1. Ограничено место для хранения информации и можно сохранить несколько логических переменных в одном байте;
2. Интерфейсы устройств передают информацию, закодировав биты в один байт;
3. Необходим удобный доступ к отдельным битам в байте;
4. Используются объекты с длинной не кратной байту.

Так как работаем мы с каждым битом по отдельности, ключевыми операциями являются логическое И(&), ИЛИ(|), отрицание(~) и логические сдвиги <<, >> для оптимизации.

**Постановка задачи**

**Цель 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.

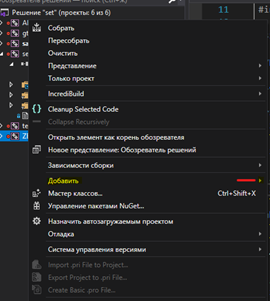
Выполнение работы предполагает решение следующих задач:

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

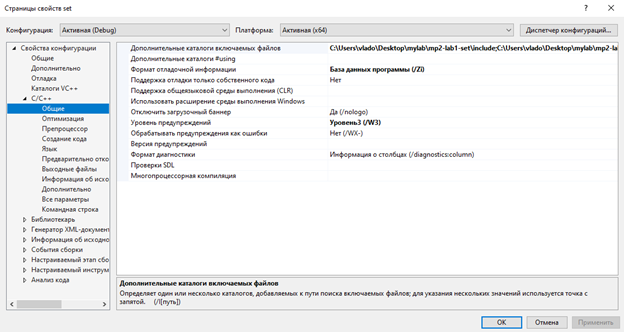
**Руководство пользователя**

В ходе лабораторной работы была получена статическая библиотека(staticlib.lib), для того чтобы ее использовать необходимо:

1. Добавить проект статической библиотеки set.vcxproj
2. В свойство проекта указать ссылку на статическую библиотеку



1. Сделать включение заголовочного файла TSet.h, указав в свойствах своего проекта где необходимо искать заголовочный файл



**Руководство программиста**

**Описание структуры программы**

В разделе *set* лежат заголовочные файлы *tbitfield.h* и *tset.h*, в них определены классы битового поля *TBitField* и множества *TSet*. Их реализация содержится в *tbitfield.cpp* и *tset.cpp*.

Раздел *test\_set* содержит набор тестов, реализованный в файлах *test\_tbitfield.cpp* и *test\_tset.cpp* с помощью фреймворка Google Test.

Раздел *sample\_prime\_numbers* содержит пример использования множества и битового поля для поиска простых чисел с помощью алгоритма “Решето Эратосфена”.

**Описание структур данных**

**TBitField:**

*int bitLen* – длина битового поля;

*int memLen* – количество битов для представления битового поля;

*int \*pMem* – память для представления битового поля;

*int GetMemIndex(const int n) const* – метод определяющий индекс бита n в массиве *pMem*;

*int GetMemMask(const int n) const* – метод возвращающий маску для бита n.

*TBitField(int len)* – конструктор инициализации. Принимает длину битового поля.

*TBitField(const TBitField &bf)* – конструктор копирования.

*~TBitField()* – деструктор.

*int GetLength() const* – возвращает длину битового.

*void SetBit(const int n)* – устанавливает n-й бит битового поля.

*void ClrBit(const int n)* – очищает n-й бит битового поля.

*int GetBit(const int n) const* – возвращает значение бита n.

*int operator==(const TBitField &bf) const* – принимает ссылку на объект класса TBitField, проверяет на равенство два битовых поля. Если они идентичны, то возвращает 1, иначе 0.

*int operator!=(const TBitField &bf) const* – принимает ссылку на объект класса *TBitField*, проверяет на неравенство два битовых поля. Если они различны, то возвращает 1, иначе 0.

*TBitField& operator=(const TBitField &bf)* – принимает ссылку на объект класса *TBitField*, присваивает полученное битовое поле к тому, которым был вызван оператор.

*TBitField operator|(const TBitField &bf)* – принимает ссылку на объект класса *TBitField*, возвращает новый объект класса *TBitField* полученный путем использования операции побитового «ИЛИ» между исходным и пришедшим битовыми полями.

*TBitField operator&(const TBitField &bf) –* принимает ссылку на объект класса *TBitField*, возвращает новый объект класса *TBitField* полученный путем использования операции побитового «И» между исходным и пришедшим битовыми полями.

*TBitField operator~()* – возвращает объект битового поля, полученный путем применения операции побитового отрицания для исходного битового поля.

*istream &operator>>(istream &istr, TBitField &bf)* – ввод битового поля с консоли.

*ostream &operator<<(ostream &ostr, const TBitField &bf)* – вывод битового поля на консоль.

**TSet:**

*int maxPower* – максимальная мощность множества.

*TBitField bitField* – битовое поле для хранения характеристического вектора.

*TSet(int mp)* – конструктор-инициализатор. Принимает максимальную мощность множества.

*TSet(const TSet &s)* – конструктор копирования.

*TSet(const TBitField &bf)* – конструктор преобразования типа. Принимает ссылку на объект с типом TBitField.

*operator TBitField()* – возвращает битовое поле характеристического вектора.

*int GetMaxPower(void) const* – возвращает максимальную мощность множества.

*void InsElem(const int Elem)* – добавляет элемент *Elem* в множество.

*void DelElem(const int Elem)* – удаляет элемент *Elem* из множества.

*int IsMember(const int Elem) const* – проверяет наличие элемента Elem в множестве. Если элемент есть, то возвращает 1, иначе 0.

**Описание алгоритма**

Необходимо представить в памяти множество натуральных чисел. Для этого будем использовать битовую строку, в которой каждый бит будет принимать либо значение «1», если элемент присутствует во входном множестве, либо «0» в противном случае.

Битовую строку будем хранить в виде массива. Заметим, что в битовой строке индексация бит происходит слева направо, в массиве аналогично, однако биты элемента массива нумеруются справа налево.

Для удобства работы с методами получения, очищения, добавления бита и подобным в классе *TBitField* используются вспомогательные методы:

1. *GetMemIndex* – для n-го бита определяется индекс элемента m в массиве pMem, в котором хранится битовое поле. Заметим, что m является целой частью от деления номера бита n на количество битов выделяемых на один элемент массива.
2. *GetMemMask* – определяет битовую маску для n-го бита. Ставим значение 1 в позицию k, определяемую по формуле 2. Битовую маску рассматриваем как двоичную запись числа.

Рассмотрим непосредственно методы работы с битами.

Чтобы установить бит на позицию можем использовать побитовое «ИЛИ» между битами элемента массива *pMem*, в котором находится бит n, и между его битовой маской.

|  |
| --- |
|  |

Чтобы очистить бит n используем побитовое «И» между битами элемента массива *pMem*, в котором находится бит n, и между дополнением его битовой маски.

|  |
| --- |
|  |

Чтобы получить значение бита n используем побитовое «И» между битами элемента массива *pMem*, в котором находится бит n, и между его битовой маской.

|  |
| --- |
|  |

**Заключение**

Реализован класс битового поля TBitField согласно заданному интерфейсу.

Реализован классмножества TSet согласно заданному интерфейсу.

Обеспечена работоспособность тестов.

Реализовано несколько простых тестов на базе Google Test.

Код опубликован в личном репозитории на GitHub.

**Приложение**

Фрагменты исходного кода представлены в личном репозитории на Github:

<https://github.com/Alexey42/mp2-lab1-set/tree/master/src>