Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение

высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт Информационных технологий, Математики и Механики

Отчёт по лабораторной работе

Структура хранения множества

Выполнил:

студент ИТММ гр. 381808-02

Жафяров О.И.

Нижний Новгород

2019 г.

Содержание

Введение........................................................................................................................................3

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

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

Руководство программиста......................................................................................................7

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

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

Описание структуры программы ..............................................................9 Заключение...................................................................................................................................13 Литература....................................................................................................................................14 Приложения.................................................................................................................................15

Приложение 1............................................................................................................15

Приложение 2............................................................................................................16

Исходный код программы........................................................................................17

**Введение**

Теория множеств – учение об общих свойствах множеств. Прежде всего, теория множеств явилась фундаментом ряда новых математических дисциплин.

Постепенно теоретико-множественные методы находят всё большее применение и в классических частях математики (в области математического анализа).

Активное применение аппарата теории множеств в современной науке приводит к необходимости создания соответствующих программных решений.

Программная реализация множества может выполняться различными способами и обычно тесно связана с использованием битовых операций в выбранном языке программирования.

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.

**Постановка задачи**

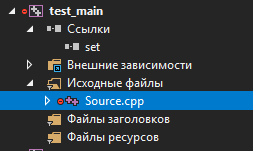
Выполнение работы предполагает решение следующих задач:

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.

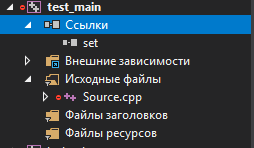
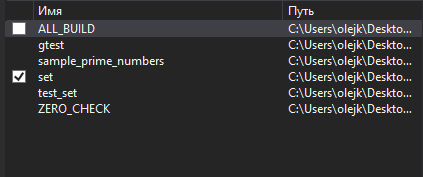
**Руководство пользователя**

Выполнив необходимые задачи, мы получаем две статические библиотеки tbitfield и tset. В данном разделе вы сможете подключить одну из библиотек, чтобы пользовать всеми описанными там методами.

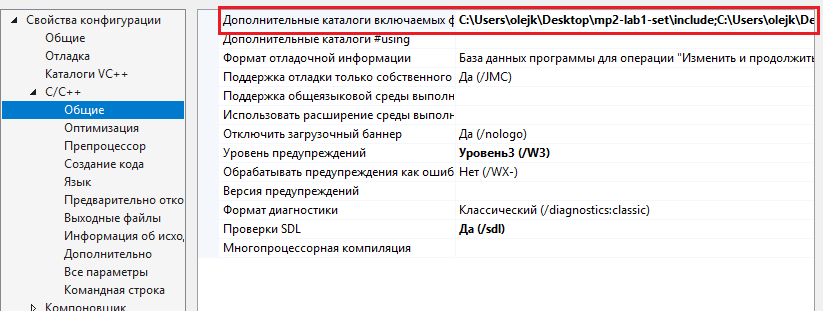
В первую очередь необходимо создать новый проект (в моем случае test\_main). В него добавить Source.cpp.



Теперь необходимо добавить путь до статической библиотеки (указать ее в ссылке)

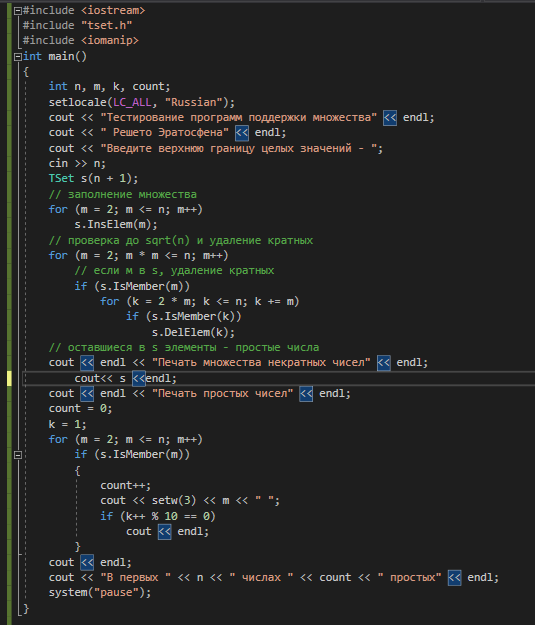
 

Еще нужно указать в настройках проекта, где искать заголовочный файл



Теперь можно пользовать классом TSet и его методами (также и с TBitField)

Результат смотреть в Приложение 2



**Руководство программиста**

**Описание структуры программы**

docs — инструкции по выполнению лабораторной работы, полезные документы.

* gtest — библиотека Google Test.
* include — директория для размещения заголовочных файлов.
* samples — директория для размещения демо-приложений.
* sln — директория с файлами решений и проектов для VS 2008 и VS 2010, вложенные директории vc9 и vc10 соответственно.
* src — директория с исходными кодами (cpp-файлы).
* test — директория с модульными тестами и основным приложением, инициализирующим запуск тестов.
* README.md — информация о проекте, которую вы сейчас читаете.
* Служебные файлы
  + .gitignore — перечень расширений файлов, игнорируемых Git при добавлении файлов в репозиторий.
  + CMakeLists.txt — корневой файл для сборки проекта с помощью CMake. Может быть использован для генерации проекта в среде разработки, отличной от Microsoft Visual Studio.
  + .travis.yml — конфигурационный файл для системы автоматического тестирования Travis-CI. Тесты, входящие в состав шаблонного проекта, регулярно запускаются на удаленной инфраструктуре.

В решении содержатся следующие модули:

Модуль tbitfield, содержащий реализацию класса битового поля (файлы ./include/tbitfield.h, ./src/tbitfield.cpp). Предполагается, что в ходе выполнения работы реализуются методы класса в файле ./src/tbitfield.cpp, при этом заголовочный файл ./include/tbitfield.h с объявлениями должен оставаться неизменным.

* Модуль tset, содержащий реализацию класса множества (файлы ./include/tset.h, ./src/tset.cpp). При выполнении работы так же, как и в случае класса битового поля, разрабатывается только реализация методов класса.
* Тесты для классов битовое поле и множество (файлы ./test/test\_tbitfield.cpp, ./test/test\_tset.cpp).
* Пример использования класса битового поля и множества для поиска простых чисел с использованием алгоритма, называемого "Решетом Эратосфена" (файл ./samples/sample\_prime\_numbers.cpp).

**Описание структур данных**

**Класс Битовое поле BitField.cpp (структура хранения битового поля)**

Битовое поле - это некоторое количество бит, расположенных последовательно в памяти. Реализацией представления битов занимается массив TELEM типа typedef unsigned int.

Помимо того, в поле класса входит:

Bitlen (Длина битового поля - максимальное количество битов)

MemLen (Количество элементов Mem для представления битового поля)

TELEM \*pMem (память для представления битового поля)

**Класс Множество Set.cpp (структура хранения множества)**

Данный класс отвечает за создание и хранение множеств с использованием класса битовых полей. В поле класса входит:

MaxPower (максимальная мощность множества)

TBitField BitField (битовое поле для хранения характеристического вектора)

**Описание алгоритмов**

**Класс TBitField**

1. TBitField(int len)

Создается битовое поле длины len (Bitlen = len). Если введенный len будет меньше нуля, то появится ошибка.

Данная строка в коде MemLen((BitLen-1) / (8 \* sizeof(int))) + 1) отвечает за выделение памяти. После происходит зануление всех элементов массива pMem.

1. TBitField(const TBitField &bf)

Создает новое битовое поле pMem = new TELEM[MemLen];

Далее копирует данные из существующего поля в новый

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

1. ~TBitField()

Освобождение памяти и дальнейшее зануление указателя.

1. int GetLength(void) const

Возвращаем длину битового поля

1. void SetBit(const int n)

Если n не проходит условие (n < 0) || (n >= BitLen), то появляется ошибка.

Далее устанавливает значение бита n (единица) pMem[GetMemIndex(n)] |= GetMemMask(n)

1. void ClrBit(const int n)

Если n не проходит условие (n < 0) || (n >= BitLen), то появляется ошибка.

Далее очищает значение бита n (ноль)

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

1. int GetBit(const int n) const

Если n не проходит условие (n < 0) || (n >= BitLen), то появляется ошибка.

Возвращает значение бита n pMem[GetMemIndex(n)] & GetMemMask(n)

1. int operator==(const TBitField &bf) const

Если два аргумента поля равны, то возвращается 1. Иначе 0.

Однако если размеры полей равны, то сравниваем уже pMem пока не найдем разные.

1. int operator!=(const TBitField &bf) const

Если два аргумента поля равны, то возвращается 0. Иначе 1.

Однако если размеры полей равны, то сравниваем уже pMem пока не найдем одинаковые.

1. TBitField& operator=(const TBitField &bf)

Если адреса полей не равны, то копируем данные поля bf

1. TBitField operator|(const TBitField &bf)

Возвращение результата операции “Или” между элементами массивов pMem bf.pMem

1. TBitField operator&(const TBitField &bf)

Возвращение результата операции “И” между элементами массивов pMem bf.pMem

1. TBitField operator~(void)

Возвращение результата операции “Не” при помощи элементов массива pMem

1. friend istream &operator>>(istream &istr, TBitField &bf)

Ввод битового поля

1. friend ostream &operator<<(ostream &ostr, const TBitField &bf)

Последовательный вывод элементов битового поля

**Класс TSet**

1. TSet(const TSet &s)

Создается новое множество, в которое копируются данные из другого множества s.

1. TSet(const TBitField &bf)

Создается новое множество, в которое копируются данные из битового поля bf.

1. operator TBitField()

Возвращает множество, переделанное в битовое поле.

1. int GetMaxPower(void) const

Возвращает мощность множества.

1. void InsElem(const int Elem)

Во множество добавляется, введенный элемент Elem.

1. void DelElem(const int Elem)

Удаляет из множества, указанный элемент Elem.

1. int IsMember(const int Elem) const

Проверяет, есть ли введенный элемент Elem в множестве.

1. int operator== (const TSet &s) const

Сравнивает битовые поля двух множеств.

1. int operator!= (const TSet &s) const

Сравнивает битовые поля двух множеств.

1. TSet& operator=(const TSet &s)

Данные множества s присваиваются множеству this.

1. TSet operator+ (const int Elem)

Возвращает множество с добавленным в него элементов Elem.

1. TSet operator- (const int Elem)

Возвращает множество с удаленным из него элементов Elem.

1. TSet operator+ (const TSet &s)

Создание массива tmp для хранения результата поэлементной побитовой дизъюнкции.

1. TSet operator\* (const TSet &s)

Создание массива tmp для хранения результата поэлементной побитовой конъюкции.

1. TSet operator~ (void)

Дополнение множества (Основана на перегрузке “Не” TBitField)

1. friend istream &operator>>(istream &istr, TSet &bf)

Ввод множества.

1. friend ostream &operator<<(ostream &ostr, const TSet &bf)

Вывод максимального элемента множества, а после и всех элементов этого множества.

**Эксперименты**

В качестве эксперимента были использованы готовые тесты с целью проверки классов на работоспособность. За данную процедуру отвечает проект "test\_set", который вмещает в себя 3 файла:

test\_main.cpp основной файл в проекте, который отвечает за запуск тестов для двух классов.

int main(int argc, char \*\*argv) {

::testing::InitGoogleTest(&argc, argv); //Запуск тестов

return RUN\_ALL\_TESTS(); //Возвращает результаты тестов

}

Файл test\_tbitfield.cpp содержит в себе список тестов, необходимых для проверки класса tbitfield.cpp с целью обнаружения ошибок в самом классе (см. Приложение 1)

Файл test\_tset.cpp содержит в себе список тестов, необходимых для проверки класса tset.cpp с целью обнаружения ошибок в самом классе (см. Приложение 1)

Также проект “set” был использован в качестве статической библиотеки, чтобы пользователь лично мог убедиться в корректной работе классов (см. В руководстве пользователя)

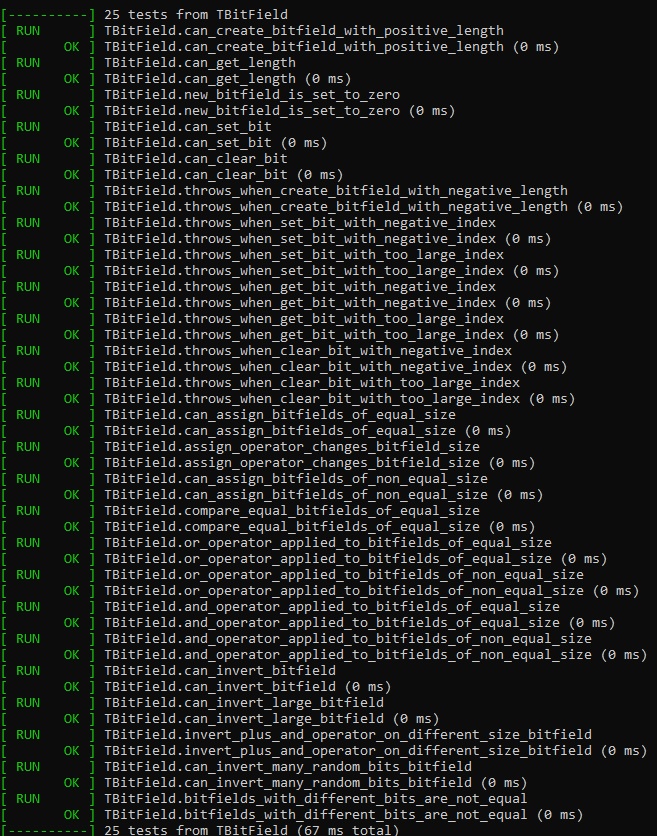
**Заключение**

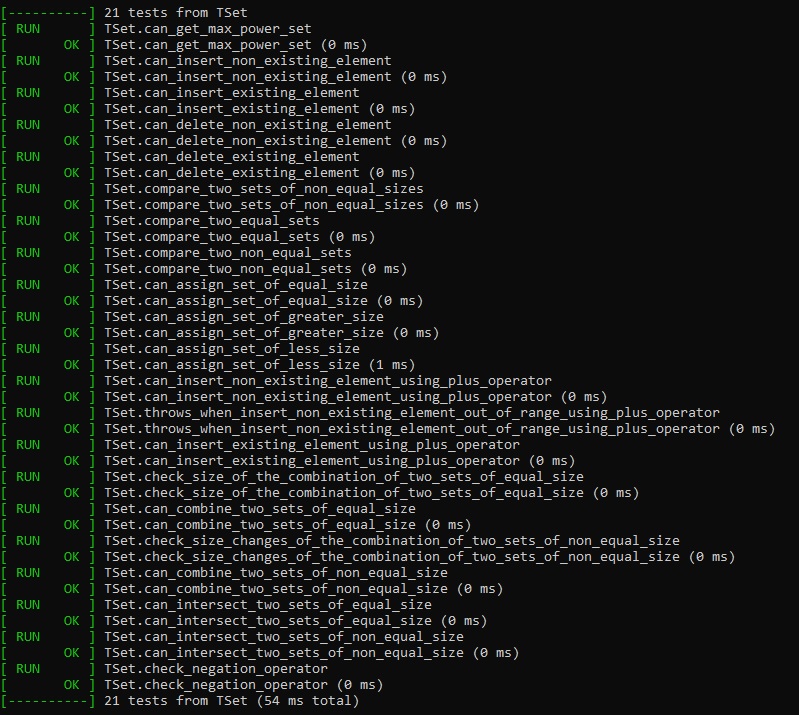
В лабораторной работе были написаны два класса (TbitField, TSet), на основе которых была создана статическая библиотека, успешно прошедшая эксперимент. Каждый класс успешно прошел проверку через Google Test.

В данной работе я научился реализовать Битовое поле и на его основе реализовывать множество. К тому же, научился пользоваться Git-ом и Google Test.

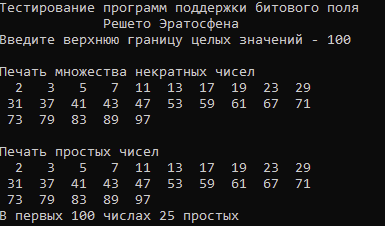
Литература

Приложения

Приложение 1



Приложение 2



Исходный фрагмент программы

Класс TBitField

TBitField::TBitField(int len): BitLen(len), MemLen((BitLen / (8 \* sizeof(TELEM))) + 1)

{

if (len < 0)

{

throw("Error(len<0)");

}

pMem = new TELEM[MemLen];

for (int i = 0; i < MemLen; i++)

{

pMem[i] = 0;

}

}

TBitField::TBitField(const TBitField &bf): BitLen(bf.BitLen), MemLen(bf.MemLen) // конструктор копирования

{

pMem = new TELEM[MemLen];

for (int i = 0; i < MemLen; i++)

{

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

}

}

TBitField::~TBitField()

{

delete[] pMem;

pMem = nullptr;

BitLen = 0;

MemLen = 0;

}

int TBitField::GetMemIndex(const int n) const // индекс Мем для бита n

{

if ((n < 0) || (n >= BitLen))

{

throw "Error(Incorrect bit)";

}

return (n / (8 \* sizeof(TELEM)));

}

TELEM TBitField::GetMemMask(const int n) const // битовая маска для бита n

{

if ((n < 0) || (n >= BitLen))

{

throw("Error(Incorrect bit)");

}

return 1 << (n % (sizeof(TELEM) \* 8));

}

// доступ к битам битового поля

int TBitField::GetLength(void) const // получить длину (к-во битов)

{

return BitLen;

}

void TBitField::SetBit(const int n) // установить бит

{

if ((n < 0) || (n >= BitLen))

{

throw "Error(Incorrect bit)";

}

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

}

void TBitField::ClrBit(const int n) // очистить бит

{

if ((n < 0) || (n >= BitLen))

{

throw("Error(Incorrect bit)");

}

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

}

int TBitField::GetBit(const int n) const // получить значение бита

{

if ((n < 0) || (n >= BitLen))

{

throw "Error(Incorrect bit)";

}

return (pMem[GetMemIndex(n)] & GetMemMask(n));

}

// битовые операции

TBitField& TBitField::operator=(const TBitField &bf) // присваивание

{

if (this != &bf)

{

delete[] pMem;

BitLen = bf.BitLen;

MemLen = bf.MemLen;

pMem = new TELEM[MemLen];

for (int i = 0; i < MemLen; i++)

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

}

return \*this;

}

int TBitField::operator==(const TBitField &bf) const // сравнение

{

if (BitLen != bf.BitLen)

{

return false;

}

else

{

for (int i = 0; i < MemLen; i++)

{

if (pMem[i] != bf.pMem[i])

{

return false;

}

}

}

return true;

}

int TBitField::operator!=(const TBitField &bf) const // сравнение

{

if (BitLen != bf.BitLen)

{

return true;

}

else

{

for (int i = 0; i < MemLen; i++)

{

if (pMem[i] == bf.pMem[i])

{

return false;

}

}

}

return true;

}

TBitField TBitField::operator|(const TBitField &bf) // операция "или"

{

TBitField tmp(\*this);

int min = bf.MemLen;

if (BitLen < bf.BitLen)

{

tmp = bf;

min = MemLen;

}

for (int i = 0; i < min; i++)

{

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

}

return tmp;

}

TBitField TBitField::operator&(const TBitField &bf) // операция "и"

{

TBitField tmp(BitLen);

int min = bf.MemLen;

if (BitLen < bf.BitLen)

{

TBitField h(bf.BitLen);

tmp = h;

min = MemLen;

}

for (int i = 0; i < min; i++)

{

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

}

return tmp;

}

TBitField TBitField::operator~(void) // отрицание

{

TBitField tmp = \*this;

for (int i = 0; i < tmp.BitLen; i++)

{

if (tmp.GetBit(i))

{

tmp.ClrBit(i);

}

else

{

tmp.SetBit(i);

}

}

return tmp;

}

// ввод/вывод

istream &operator>>(istream &istr, TBitField &bf) // ввод

{

int i = 0;

char ch;

// поиск {

do {

istr >> ch;

} while (ch != ' ');

// ввод элементов и включение в множество

while (1) {

istr >> ch;

if (ch == '0')

bf.ClrBit(i++);

else if (ch == '1')

bf.SetBit(i++); else break;

}

return istr;

}

ostream &operator<<(ostream &ostr, const TBitField &bf) // вывод

{

for (int i = 0; i < bf.BitLen; i++)

{

if (bf.GetBit(i))

{

ostr << 1;

}

else

{

ostr << 0;

}

}

return ostr;

}

**Класс TSet**

Set::TSet(const TSet& s) : BitField(s.BitField)

{

MaxPower = s.MaxPower;

}

// конструктор преобразования типа

TSet::TSet(const TBitField& bf) : BitField(bf)

{

MaxPower = bf.GetLength();

}

TSet::operator TBitField()

{

TBitField tmp(this->BitField);

return tmp;

}

int TSet::GetMaxPower(void) const // получить макс. к-во эл-тов

{

return MaxPower;

}

int TSet::IsMember(const int Elem) const // элемент множества?

{

return BitField.GetBit(Elem);

}

void TSet::InsElem(const int Elem) // включение элемента множества

{

BitField.SetBit(Elem);

}

void TSet::DelElem(const int Elem) // исключение элемента множества

{

BitField.ClrBit(Elem);

}

// теоретико-множественные операции

TSet& TSet::operator=(const TSet& s) // присваивание

{

BitField = s.BitField;

MaxPower = s.MaxPower;

return \*this;

}

int TSet::operator==(const TSet& s) const // сравнение

{

return BitField == s.BitField;

}

int TSet::operator!=(const TSet& s) const // сравнение

{

return BitField != s.BitField;

}

TSet TSet::operator+(const TSet& s) // объединение

{

TSet tmp(BitField | s.BitField);

return tmp;

}

TSet TSet::operator+(const int Elem) // объединение с элементом

{

TBitField tmp(MaxPower);

tmp = BitField;

tmp.SetBit(Elem);

TSet temp(tmp);

return temp;

}

TSet TSet::operator-(const int Elem) // разность с элементом

{

TBitField tmp(MaxPower);

tmp = BitField;

tmp.ClrBit(Elem);

TSet temp(tmp);

return temp;

}

TSet TSet::operator\*(const TSet& s) // пересечение

{

TSet tmp(BitField & s.BitField);

return tmp;

}

TSet TSet::operator~(void) // дополнение

{

TSet tmp(~BitField);

return tmp;

}

// перегрузка ввода/вывода

istream& operator>>(istream& istr, TSet& s) // ввод

{

int i = 0;

istr >> i;

while ((i >= 0) && (i < s.MaxPower))

{

s.InsElem(i);

istr >> i;

}

return istr;

}

ostream& operator<<(ostream& ostr, const TSet& s) // вывод

{

for (int i = 0; i < s.GetMaxPower(); i++)

if (s.IsMember(i)) ostr << i;

return ostr;

}