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

Множества на основе битовых полей

Выполнил:

студент института ИТММ гр.381908-1

Труханов А. В.

Проверил:

ассистент каф. МОСТ, ИИТММ

Лебедев И.Г

Нижний Новгород

2021 г.

Содержание

[Введение 3](#_Toc72877455)

[Постановка задачи 4](#_Toc72877456)

[Руководство пользователя 5](#_Toc72877457)

[Руководство программиста 6](#_Toc72877458)

[Описание структуры программы 6](#_Toc72877459)

[Описание структур данных 6](#_Toc72877460)

[Описание алгоритмов 6](#_Toc72877461)

[Эксперименты 8](#_Toc72877462)

[Заключение 10](#_Toc72877463)

[Литература 11](#_Toc72877464)

[Приложения 12](#_Toc72877465)

[Приложение 1 12](#_Toc72877466)

# Введение

Теория множеств – учение об общих свойствах множеств – преимущественно бесконечных. Явным образом понятие множества подверглось систематическому изучению во второй половине XIX века в работах немецкого математика Георга Кантора.

Программная реализация множества может выполняться различными способами (в соответствии с требованиями конкретной задачи или с общих позиций) и обычно тесно связана с использованием битовых операций в выбранном языке программирования. Данная работа посвящена изучению одного из возможных подходов к хранению и обработке множеств.

Понятие множества, или совокупности, принадлежит к числу простейших математических понятий; оно не определяется, но может быть пояснено при помощи примеров. Так, можно говорить о множестве всех книг, составляющих данную библиотеку, множестве всех точек данной линии, множестве всех решений данного уравнения.

# 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

Пользователю нужно запустить файл sample\_prime\_numbers.exe.

Откроется консольное приложение для тестирования программ поддержки битового поля на основе решета Эратосфена.

Программа запросит пользователя ввести верхнюю границу целых значений. (Рис. 1)

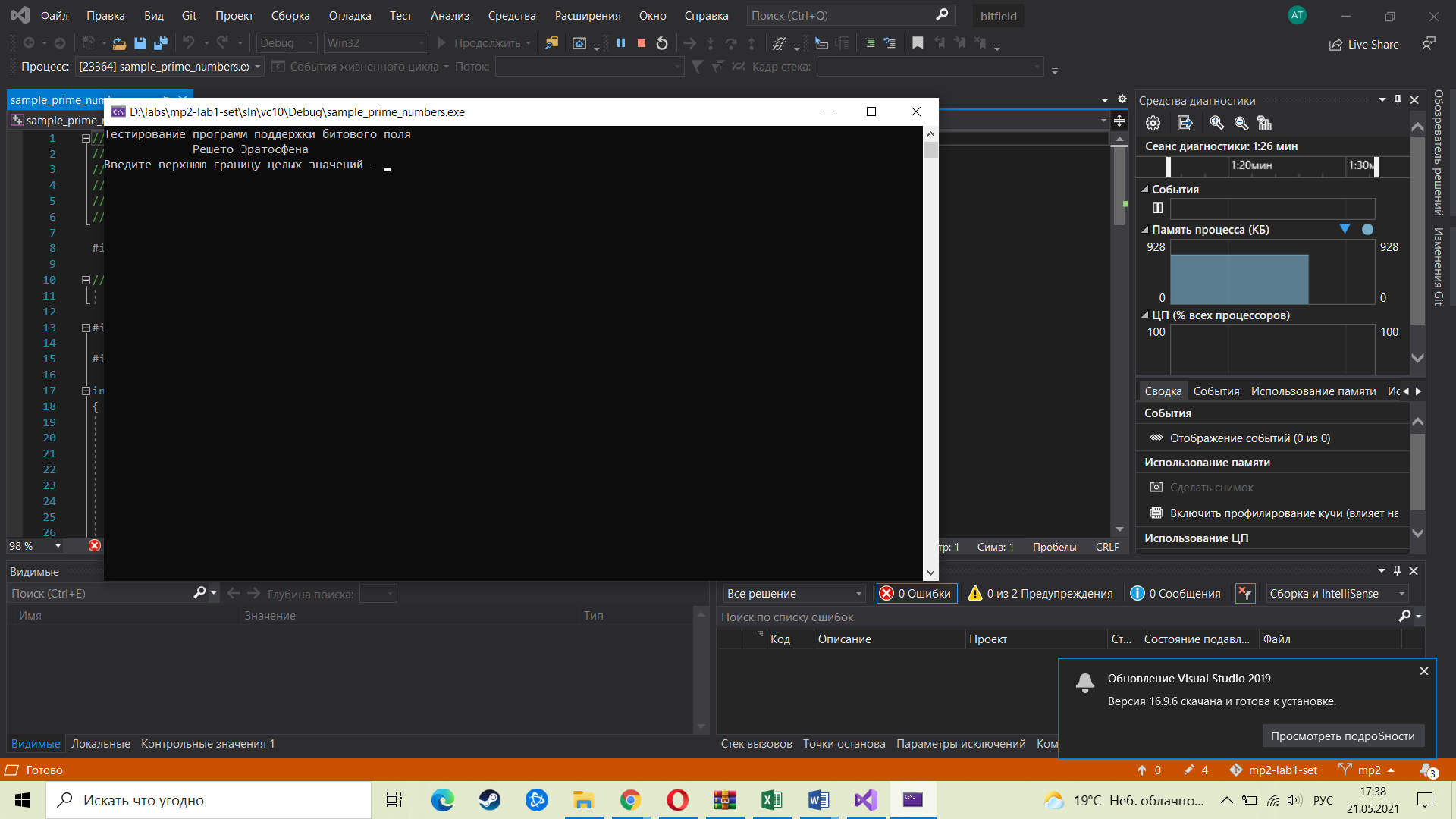
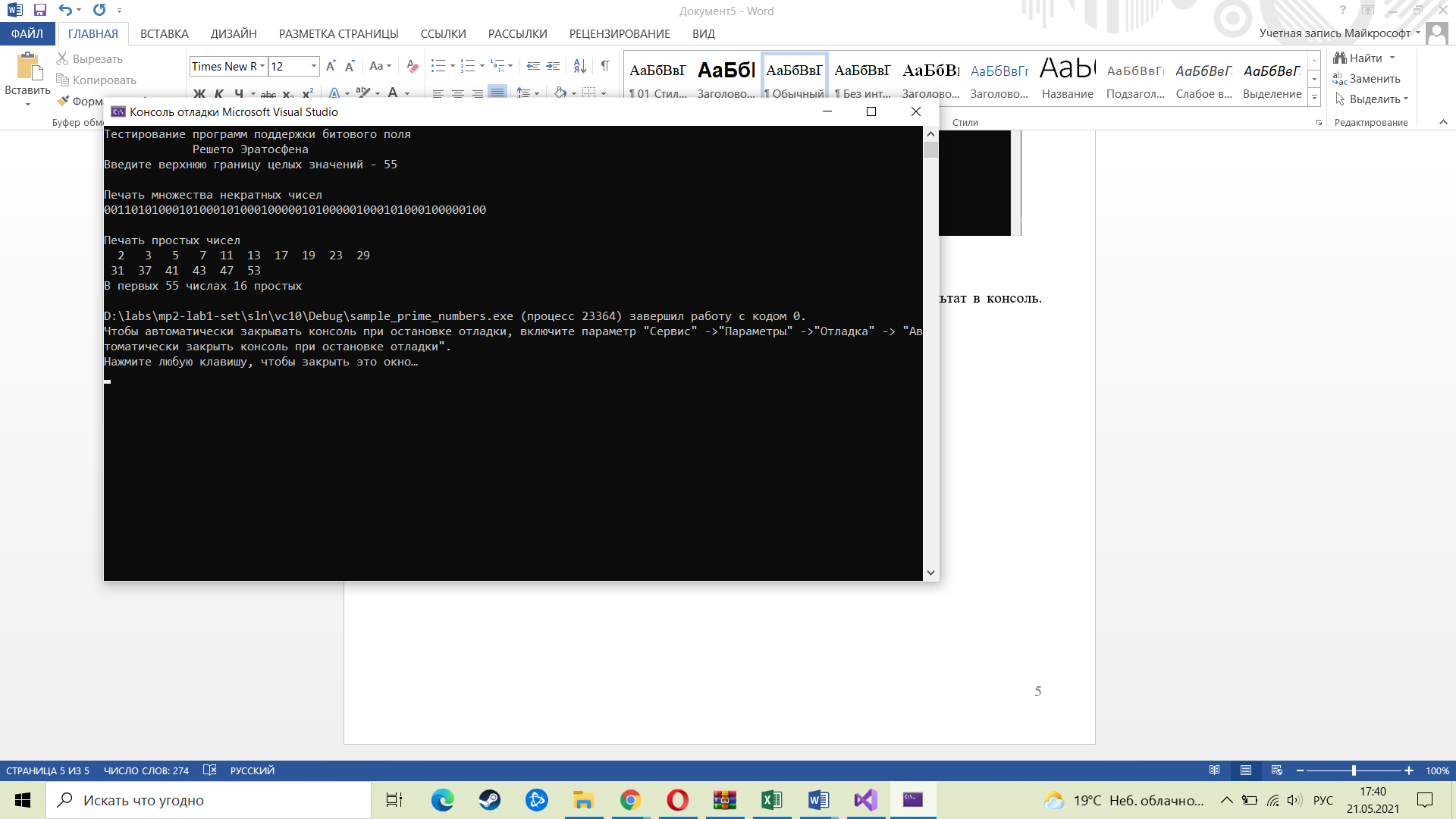


Рис.1

После ввода числа, программа выполнит действия и выведет результат в консоль. (Рис. 2)



Для повторного выполнения потребуется перезапустить программу.

# 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 .

## Описание структур данных

Класс “TSet” модуль с классом, реализующий обработку множеств:

* int MaxPower – максимальная мощность множества
* TBitField BF - Битовое поле для хранения характеристического вектора
* int GetMaxPower(void) const – получение максимальной мощности множества
* void InsElem(const int n) – включить элемент в множество
* void DelElem(const int n) – удалить элемент из множества
* int IsMember(const int n) const – проверить наличие элементов в множестве
* operator== - оператор сравнения
* operator= - оператор присваивания
* operator+ - оператор объединения
* operator\* - оператор пересечения
* operator~ - оператор дополнения
* operator>> - оператор ввода
* operator<< - оператор вывода

Класс “TBitField” реализует структуру хранения битовых полей:

* int BitLen – длина битового поля
* pp\* pMem(insigned int) – динамическая память битового поля
* int MemLen – количество элементов битового поля
* int GetLength(void) const – получить длину строки
* int SetBit(const int n) – установить бит из n позиций
* int ClrBit(const int n) – очистить бит
* int GetBit(const int n) const – получить значение бита
* operator== - оператор сравнения
* operator| - оператор пересечения
* operator& - оператор объединения
* operator~ - оператор логического отрицания
* operator>> - оператор ввода
* operator<< - оператор вывода
* operator= - оператор присваивания

## Описание алгоритмов

* результаты операций – новые экземпляры классов Битовое поле и Множество;
* длина нового битового поля (мощность нового множества) – максимум из длин (мощностей) аргументов;
* основная часть метода «и» – цикл, копирующий элементы поля pMem (динамического массива) в новое битовое поле (каждый элемент в данном случае 24 имеет тип int, то есть содержит 32 бита) и цикл, выполняющий логическое «и» с каждым элементом второго аргумента операции;
* реализации метода «пересечение» сводится к вызову метода «и» для соответствующих битовых полей.

*Создание множества:*

* Инициализируем битовое поле размером, равным мощности множества,
* Выделяем память,
* Заполняем элементы нулями.

*Добавление элемента в множество:*

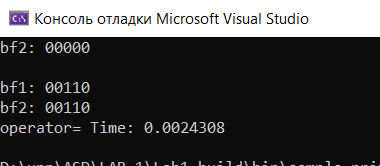
* Инициализируем битовое поле,
* Передаем элемент в класс битового поля,
* На основе элемента получаем индекс и маску,
* Используя побитовое «ИЛИ», присваиваем по полученному индексу, полученную маску.

*Удаление элемента из множества:*

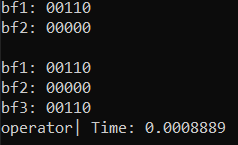
* Передаем элемент в класс битового поля,
* На основе элемента получаем индекс и маску,
* Используя побитовое «И», присваиваем по полученному индексу, полученную маску, предварительно применив к маске побитовую инверсию.

# Эксперименты

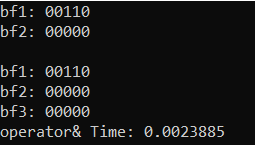
Результат выполнения операции присваивание:



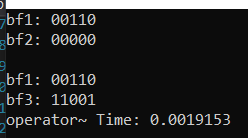
Результат операции “ИЛИ”:



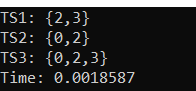
Результат операции “И”:



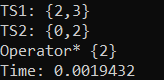
Результат операции отрицание:



Результат объединения множеств:



Результат операции пересечения множеств:



Результат операции дополнение:



# Заключение

В ходе выполнения лабораторной работы была разобрана реализация битовых полей и множеств на языке программирования С++. Также получены навыки работы с Google Tests и CMake.

# Литература

1. Лабораторный практикум. Барышева И.В., Мееров И.Б., Сысоев А.В., Шестакова Н.В. Под редакцией Гергеля В.П. Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017. – 105с.

# Приложения

## Приложение 1

tbitfield.h

#ifndef \_\_BITFIELD\_H\_\_

#define \_\_BITFIELD\_H\_\_

#include <iostream>

using namespace std;

typedef unsigned int TELEM;

class TBitField

{

private:

int BitLen; // длина битового поля - макс. к-во битов

TELEM \*pMem; // память для представления битового поля

int MemLen; // к-во эл-тов Мем для представления бит.поля

// методы реализации

int GetMemIndex(const int n) const; // индекс в pМем для бита n (#О2)

TELEM GetMemMask (const int n) const; // битовая маска для бита n (#О3)

public:

TBitField(int len); // (#О1)

TBitField(const TBitField &bf); // (#П1)

~TBitField(); // (#С)

// доступ к битам

int GetLength(void) const; // получить длину (к-во битов) (#О)

void SetBit(const int n); // установить бит (#О4)

void ClrBit(const int n); // очистить бит (#П2)

int GetBit(const int n) const; // получить значение бита (#Л1)

// битовые операции

int operator==(const TBitField &bf) const; // сравнение (#О5)

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

TBitField& operator=(const TBitField &bf); // присваивание (#П3)

TBitField operator|(const TBitField &bf); // операция "или" (#О6)

TBitField operator&(const TBitField &bf); // операция "и" (#Л2)

TBitField operator~(void); // отрицание (#С)

friend istream &operator>>(istream &istr, TBitField &bf); // (#О7)

friend ostream &operator<<(ostream &ostr, const TBitField &bf); // (#П4)

};

tbitfield.cpp

#include "tbitfield.h"

TBitField::TBitField(int len) : BitLen(len)

{

if (len < 0)

throw ("Length should be positive");

BitLen = len;

if (len % (sizeof(TELEM) \* 8) == 0)

MemLen = len / (sizeof(TELEM) \* 8);

else

MemLen = (len / (sizeof(TELEM) \* 8)) + 1;

pMem = new TELEM[MemLen];

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

pMem[i] = 0;

}

TBitField::TBitField(const TBitField& bf)// конструктор копирования

{

BitLen = bf.BitLen;

MemLen = bf.MemLen;

pMem = new TELEM[MemLen];

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

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

}

TBitField::~TBitField()

{

if (pMem != NULL)

{

delete[] pMem;

pMem = NULL;

}

}

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

{

if ((n <= -1) || (n > BitLen))

throw "Error";

return n >> 5;

}

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

{

if ((n <= -1) || (n > BitLen))

throw "Error";

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 <= -1) || (n > BitLen))

throw "Error";

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

}

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

{

if ((n <= -1) || (n > BitLen))

throw "Error";

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

}

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

{

if ((n <= -1) || (n > BitLen))

throw "Error";

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

}

// битовые операции

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

{

if (this == &bf)

return \*this;

else {

BitLen = bf.BitLen;

if (MemLen != bf.MemLen) {

MemLen = bf.MemLen;

delete[] pMem;

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

{

int res = 1;

if (BitLen != bf.BitLen)

res = 0;

else

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

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

res = 0;

else

res = 1;

return res;

}

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

{

int res = 0;

if (BitLen != bf.BitLen)

res = 1;

else

{

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

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

res = 1;

else res = 0;

}

return res;

}

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

{

int i, len = BitLen;

if (bf.BitLen > len)

len = bf.BitLen;

TBitField temp(len);

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

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

for (i = 0; i < bf.MemLen; i++)

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

return temp;

}

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

{

int i, len = BitLen;

if (bf.BitLen > len)

len = bf.BitLen;

TBitField temp(len);

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

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

for (i = 0; i < bf.MemLen; i++)

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

return temp;

}

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

{

TBitField temp(\*this);

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

if (temp.GetBit(i))

temp.ClrBit(i);

else

temp.SetBit(i);

}

return temp;

}

// ввод/вывод

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

{

int i = 0;

char ch;

do {

istr >> ch;

} while (ch != ' ');

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

{

int len = bf.GetLength();

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

if (bf.GetBit(i))

ostr << '1';

else ostr << '0';

return ostr;

}

tset.h

#ifndef \_\_SET\_H\_\_

#define \_\_SET\_H\_\_

#include "tbitfield.h"

class TSet

{

private:

int MaxPower; // максимальная мощность множества

TBitField BitField; // битовое поле для хранения характеристического вектора

public:

TSet(int mp);

TSet(const TSet &s); // конструктор копирования

TSet(const TBitField &bf); // конструктор преобразования типа

operator TBitField(); // преобразование типа к битовому полю

// доступ к битам

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

// теоретико-множественные операции

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

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

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

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);

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

};

#endif

tset.cpp

#include "tset.h"

TSet::TSet(int mp) : MaxPower(mp), BitField(mp) {}

// конструктор копирования

TSet::TSet(const TSet &s) : MaxPower(s.MaxPower), BitField(s.BitField) {}

// конструктор преобразования типа

TSet::TSet(const TBitField &bf) : MaxPower(bf.GetLength()), BitField(bf) {}

TSet::operator TBitField()

{

TBitField temp(this->BitField);

return temp;

}

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

{

if (this == &s)

return \*this;

BitField = s.BitField;

MaxPower = s.GetMaxPower();

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 temp(BitField | s.BitField);

return temp;

}

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

{

BitField.SetBit(Elem);

return \*this;

}

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

{

BitField.ClrBit(Elem);

return \*this;

}

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

{

TSet temp(BitField & s.BitField);

return temp;

}

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

{

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

if (BitField.GetBit(i) != 0)

BitField.ClrBit(i);

else

BitField.SetBit(i);

return \*this;

}

// перегрузка ввода/вывода

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

{

int temp;

char ch;

do {

istr >> ch;

} while (ch != '{');

do {

istr >> temp;

s.InsElem(temp);

do {

istr >> ch;

} while ((ch != ',') && (ch != '}'));

} while (ch != '}');

return istr;

}

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

{

int i, n;

char ch = ' ';

ostr << "{";

n = s.GetMaxPower();

for (i = 0; i < n; i++)

{

if (s.IsMember(i))

{

ostr << ch << ' ' << i;

ch = ',';

}

}

ostr << "}";

return ostr;

}