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 ф-та ИТММ гр. 381908-1

Круглов А.М

Проверил:

ассистент каф. МО ЭВМ, ВМК

Лебедев И.Г.

Нижний Новгород

2010 г.

Содержание

[Введение 3](#_Toc270962758)

[Постановка задачи 4](#_Toc270962759)

[Руководство пользователя 5](#_Toc270962760)

[Руководство программиста 6](#_Toc270962761)

[Описание структур данных 6](#_Toc270962762)

[Описание алгоритмов 6](#_Toc270962763)

[Описание структуры программы 6](#_Toc270962764)

[Заключение 7](#_Toc270962765)

[Литература 8](#_Toc270962766)

[Приложения 9](#_Toc270962767)

[Приложение 1 9](#_Toc270962768)

[Приложение 2 9](#_Toc270962769)

# Введение

Мно́жество — одно из ключевых понятий [математики](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0); 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.

Данные типа множество позволяют хранить ограниченное число значений определённого типа без определённого порядка. Повторение значений, как правило, недопустимо. За исключением того, что множество в программировании конечно, оно в общем соответствует концепции математического множества. Для этого типа в языках программирования обычно предусмотрены стандартные операции над множествами.

В данной лабораторной работе представлен вариант реализации множества на языке С++.

# Постановка задачи

На основе битовых полей реализовать структуру данных “Множество”, а так же реализовать стандартные операции над ним. После реализации необходимо протестировать код с помощью библиотеки модульного тестирования Google Test.

Реализация включает в себя следующие пункты:

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
6. Знакомство с Cmake

Затем необходимо протестировать скорость и эффективность работы методов.

# Руководство пользователя

Пользователю необходимо запустить приложение sample\_prime\_numbers.exe. После чего откроется консольное окно, где пользователю предложат ввести верхнюю для поиска простых чисел. После чего программа выведет все простые числа до введенного пользователем числа, а так же выведет их количество.

# Руководство программиста

## Описание структуры программы

Программа состоит из двух h-файлов(tset.h и tbitfield.h). Эти файлы подключают 2 класса - TSet и TBitfield соответственно. Решение test-set - тестирует реализацию данных классов с помощью библиотеки GoogleTest. Решение sample\_prime\_numbers приводит реализацию решета Эратосфена с помощью классов TBitfield и TSet.

## Описание структур данных

**TBitfield:**

Имеет конструкторы:

* Конструктор с параметром
* Конструктор копирования

Имеет деструктор

Содержит поля:

* BitLen для хранения длины битового поля,
* TELEM \*pMem массив для хранения битов
* MemLen для хранения количества элементов битового поля.

Реализованы методы:

* GetMemIndex – получение индекса ячейки памяти,
* GetMemMask – маска для бита,
* GetLength – получение длины,
* SetBit – установка бита,
* ClrBit – очистка бита,
* GetBit – получение бита.

Реализованы операции:

* operator== - оператор сравнения,
* operator!= - оператор сравнения,
* operator= - оператор присваивания,
* operator| - операция «ИЛИ»,
* operator& - оператор «И»,
* operator~ - оператор отрицания,
* &operator>> -оператор ввода,
* &operator<< - оператор вывода.

**TSet**:

Имеет конструкторы:

* Конструктор с параметром
* Конструктор копирования
* Конструктор преобразования типа

Имеет деструктор

Содержит поля:

* MaxPower для хранения максимальной мощности множества,
* TBitField BitField – битовое поле для хранения характеристического вектора.

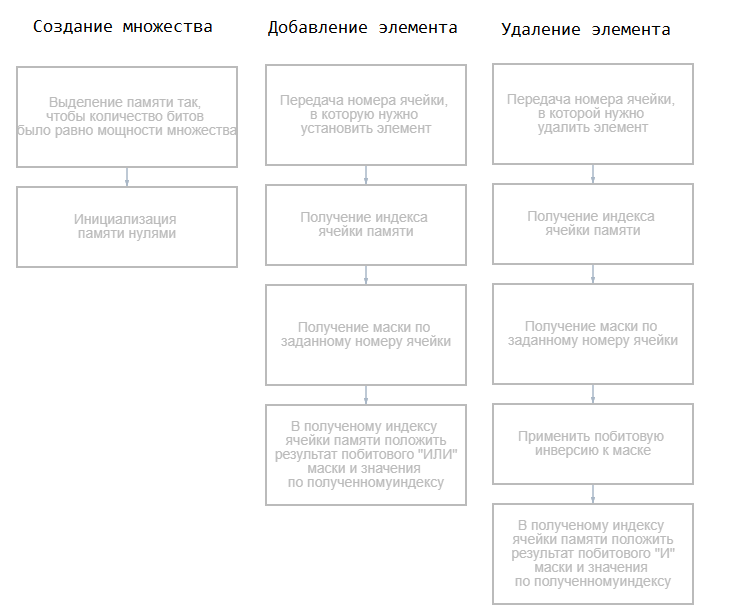
Реализует методы:

* GetMaxPower – получение максимальной мощности множества,
* InsElem – включение элемента в множество,
* DelElem – удаление элемента из множества,

Реализует операции:

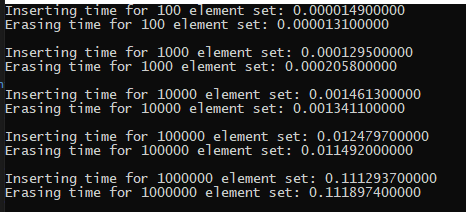
* operator== - оператор сравнения,
* operator!= - оператор сравнения,
* operator= - оператор присваивания,
* operator+ - операция объединение с элементом,
* operator- - операция разность с элементом,
* operator+ - операция объединение множеств,
* operator\* - операция пересечение множеств,
* operator~ - операция дополнения,
* &operator>> -оператор ввода,
* &operator<< - оператор вывода.

## Описание алгоритмов

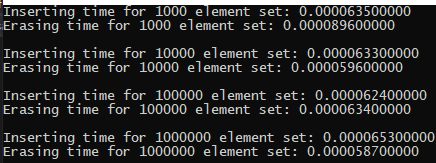


# Эксперименты

Для оценки эффективности времени работы методов был написан небольшой код(он указан в приложении). В этом коде в множество мощности n последовательно добавляются n элементов, а затем все эти элементы удаляются.



Из этого эксперимента видно, что что время вставки всех элементов множества линейно зависит от размера множества.

Теперь проведём эксперимент, в котором будем добавлять одинаковое количество элементов в множества разного размера. (В примере добавлялось 500 элементов)  


Из результатов видно, что вставка одного и того же количества элементов не зависит от размера множества. Следовательно, асимптотика вставки и удаления элемента из множества работает за О(1).

# Заключение

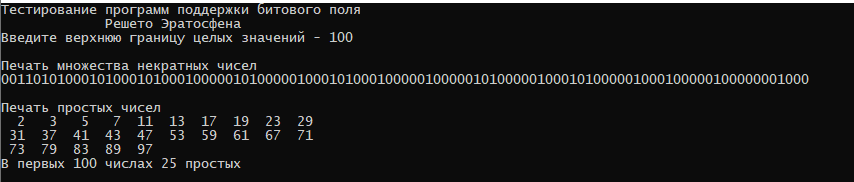
При выполнении данной лабораторной работы нами были успешно разработаны структуры данных битовое поле и множество. Были получены навыки работы с GoogleTest, GitHub. Так же получены навыки в написании cmake-листов. Тестирование методов классов было проведено успешно, так же была выявлена асимптотика методов классов.

# Литература

1. Множества // <http://mathprofi.ru/> URL: <http://mathprofi.ru/mnozhestva.html> (дата обращения: 12.18.20).
2. Представление и обработка множеств // wm-help.net URL: <https://wm-help.net/lib/b/book/27531012/146> (дата обращения: 12.18.20).
3. Битовая маска // <https://zetlab.com/> URL: <https://wm-help.net/lib/b/book/27531012/146> (дата обращения: 12.18.20).

# Приложения

## Приложение 1



## Приложение 2

**tset.h**:

// ННГУ, ВМК, Курс "Методы программирования-2", С++, ООП

//

// tset.h - Copyright (c) Гергель В.П. 07.05.2001

// Переработано для Microsoft Visual Studio 2008 Сысоевым А.В. (19.04.2015)

//

// Множество

#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**:

// ННГУ, ВМК, Курс "Методы программирования-2", С++, ООП

//

// tset.cpp - Copyright (c) Гергель В.П. 04.10.2001

// Переработано для Microsoft Visual Studio 2008 Сысоевым А.В. (19.04.2015)

//

// Множество - реализация через битовые поля

#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) : BitField(bf), MaxPower(bf.GetLength())

{

}

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;

MaxPower = s.MaxPower;

BitField = s.BitField;

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 res(BitField | s.BitField);

return res;

}

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

{

TSet res(\*this);

res.InsElem(Elem);

return res;

}

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

{

TSet res(\*this);

res.DelElem(Elem);

return res;

}

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

{

TSet res(BitField & s.BitField);

return res;

}

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

{

TSet res(~BitField);

return res;

}

// перегрузка ввода/вывода

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

{

char c;

int tmp;

do {

istr >> c;

} while (c != '{');

do{

istr >> tmp;

s.InsElem(tmp);

do {

istr >> c;

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

} while (c != '}');

return istr;

}

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

{

int n = s.GetMaxPower();

char c = ' ';

ostr << '{';

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

if (s.IsMember(i)) {

ostr << c << ' ' << i;

c = ',';

}

}

ostr << '}';

return ostr;

}

**tbitfield.h**:

#include "tbitfield.h"

#include <cmath>

TBitField::TBitField(int len) {

if (len > -1) {

BitLen = len;

MemLen = (len + 8 \* sizeof(TELEM) - 1) >> (int)(log2(8 \* sizeof(TELEM)) + 0.1);

pMem = new TELEM[MemLen];

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

pMem[i] = NULL;

}

else

throw exception();

}

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

return n >> (int)(log2(8 \* sizeof(TELEM)) + 0.1);

}

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

return 1 << (n%(sizeof(TELEM)\*8)); // 32 bits

}

// 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)

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

else

throw exception();

}

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

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

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

else

throw exception();

}

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

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

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

else

throw exception();

}

// битовые операции

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

if (this == &bf)

return \*this;

BitLen = bf.BitLen;

MemLen = bf.MemLen;

if (pMem != NULL)

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

if (BitLen != bf.BitLen)

return 0;

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

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

return 0;

}

}

return 1;

}

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

{

return !(\*this == bf);

}

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

int len = max(BitLen, bf.BitLen);

TBitField result(len);

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

result.pMem[i] = pMem[i];

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

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

return result;

}

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

int len = max(BitLen, bf.BitLen);

TBitField result(len);

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

result.pMem[i] = pMem[i];

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

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

return result;

}

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

TBitField res(\*this);

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

if (res.GetBit(i))

res.ClrBit(i);

else

res.SetBit(i);

}

return res;

}

// ввод/вывод

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

int i = 0;

char c;

do {

istr >> c;

} while (c == ' ');

while (1) {

if (c == '0')

bf.ClrBit(i++);

else if (c == '1')

bf.SetBit(i++);

else

break;

istr >> c;

}

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;

}

**tbitfiedl.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)

};

#endif

**tbifield.cpp**:

#include "tbitfield.h"

#include <cmath>

TBitField::TBitField(int len) {

if (len > -1) {

BitLen = len;

MemLen = (len + 8 \* sizeof(TELEM) - 1) >> (int)(log2(8 \* sizeof(TELEM)) + 0.1);

pMem = new TELEM[MemLen];

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

pMem[i] = NULL;

}

else

throw exception();

}

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

return n >> (int)(log2(8 \* sizeof(TELEM)) + 0.1);

}

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

return 1 << (n%(sizeof(TELEM)\*8)); // 32 bits

}

// 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)

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

else

throw exception();

}

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

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

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

else

throw exception();

}

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

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

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

else

throw exception();

}

// битовые операции

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

if (this == &bf)

return \*this;

BitLen = bf.BitLen;

MemLen = bf.MemLen;

if (pMem != NULL)

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

if (BitLen != bf.BitLen)

return 0;

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

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

return 0;

}

}

return 1;

}

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

{

return !(\*this == bf);

}

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

int len = max(BitLen, bf.BitLen);

TBitField result(len);

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

result.pMem[i] = pMem[i];

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

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

return result;

}

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

int len = max(BitLen, bf.BitLen);

TBitField result(len);

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

result.pMem[i] = pMem[i];

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

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

return result;

}

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

TBitField res(\*this);

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

if (res.GetBit(i))

res.ClrBit(i);

else

res.SetBit(i);

}

return res;

}

// ввод/вывод

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

int i = 0;

char c;

do {

istr >> c;

} while (c == ' ');

while (1) {

if (c == '0')

bf.ClrBit(i++);

else if (c == '1')

bf.SetBit(i++);

else

break;

istr >> c;

}

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;

}

**test1.cpp:**

#include "tbitfield.h"

#include<tset.h>

void check1(int n) {

TSet s(n);

auto ts = std::chrono::system\_clock::now();

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

s.InsElem(i);

auto te = std::chrono::system\_clock::now();

std::chrono::duration<double> elapsed = te - ts;

cout << fixed << setprecision(12) << "Inserting time for "<< n << " element set: " << elapsed.count() <<endl;

}

void check2(int n) {

TSet s(n);

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

s.InsElem(i);

auto ts = std::chrono::system\_clock::now();

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

s.DelElem(i);

auto te = std::chrono::system\_clock::now();

std::chrono::duration<double> elapsed = te - ts;

cout << fixed << setprecision(12) << "Erasing time for " << n << " element set: " << elapsed.count() << endl;

}

int main()

{

check1(100);

check2(100);

cout << endl;

check1(1000);

check2(1000);

cout << endl;

check1(10000);

check2(10000);

cout << endl;

check1(100000);

check2(100000);

cout << endl;

check1(1000000);

check2(1000000);

}

**test2.cpp:**

#include "tbitfield.h"

#include<tset.h>

void check1(int n) {

TSet s(n);

auto ts = std::chrono::system\_clock::now();

for (int i = 0; i < 500; i++)

s.InsElem(i);

auto te = std::chrono::system\_clock::now();

std::chrono::duration<double> elapsed = te - ts;

cout << fixed << setprecision(12) << "Inserting time for "<< n << " element set: " << elapsed.count() <<endl;

}

void check2(int n) {

TSet s(n);

for (int i = 0; i < 500; i++)

s.InsElem(i);

auto ts = std::chrono::system\_clock::now();

for (int i = 0; i < 500; i++)

s.DelElem(i);

auto te = std::chrono::system\_clock::now();

std::chrono::duration<double> elapsed = te - ts;

cout << fixed << setprecision(12) << "Erasing time for " << n << " element set: " << elapsed.count() << endl;

}

int main()

{

check1(1000);

check2(1000);

cout << endl;

check1(10000);

check2(10000);

cout << endl;

check1(100000);

check2(100000);

cout << endl;

check1(1000000);

check2(1000000);

}