Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе №1**

**«Структура хранения множества»**

Выполнила:

студентка ПМИ – 381903-3

Страшенко Мария

Проверил:

ассистент кафедры МОСТ

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)

[Приложение 18](#_Toc23876649)

# 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 Кантора2 . Влияние теории множеств на развитие современной математики очень велико. Прежде всего, теория множеств явилась фундаментом ряда новых математических дисциплин (теории функций действительного переменного, общей топологии, общей алгебры, функционального анализа и др.). Постепенно теоретико-множественные методы находят всё большее применение и в классических частях математики. Например, в области математического анализа они широко применяются в качественной теории дифференциальных уравнений, вариационном исчислении, теории вероятностей и др. Активное применение аппарата теории множеств в современной науке приводит к необходимости создания соответствующих программных решений. Вместе с тем лишь в отдельных языках программирования предусмотрены встроенные средства для работы с множествами (примером может служить язык Pascal в реализации фирмы Borland). 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.

# 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)

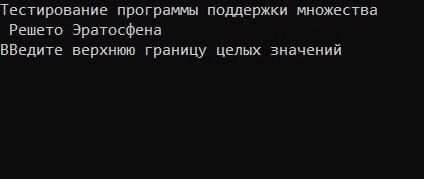
****

Рис1. Старт работы

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

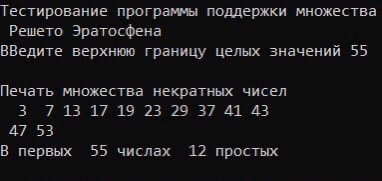
****

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

|  |
| --- |
| **TBitField** |
| * int TBitLen * PP\* pMem (insigned int) * int MemLen |
| “**constructor**”   * TBitField(int len * TBitField(const BitField& bf)   **“destructor”**   * ~TBitField()   “**interface**”   * int GetLength(void) const * void SetBit(const int n) * void ClrBit(const int n) * int GetBit(const int n) const * int operator== (const TBitField& bf) * TBitField& operator=(const TBitField& bf) * TBitField& operator|(const TBitField& bf) * TBitField& operator&(const TBitField& bf) * TBitField& operator~(void) * friend istream& operator>>(istream& istr, TBitField& bf) * friend ostream& operator<<(ostream& ostr, const TBitField&bf) |

В следующей таблице указано применение данных и методов класса.

|  |  |  |
| --- | --- | --- |
| **№** | **Данные** | **Хранит** |
| 1 | BitLen | Длина битового поля |
| 2 | PP\* pMem | Динамическая память битового поля |
| 3 | MemLen | Кол-во элементов битового поля |
|  | | |
| **№** | **Методы** | **Применение** |
| 1 | GetLength | Возвращает длину строки |
| 2 | SetBit | Устанавливает бит из n позиции |
| 3 | ClrBit | Чистит бит |
| 4 | GetBit | Получает значение бита |
| 5 | operator== | Операция сравнения |
| 6 | operator| | Операция пересечения множества (или) |
| 7 | operator& | Операция объединения множества (и) |
| 8 | operator~ | Операция логического отрицания(не) |
| 9 | operator>> | Операция ввода |
| 10 | Operator<< | Операция вывода |
| 11 | operator= | Операция присваивания |

1. **Класс «Set»**

Описание: модуль с классом, реализующий обработку Множеств

|  |
| --- |
| **Set** |
| * int MaxPower * TBitField BF |
| “**constructor**”   * Set(int mp) * Set(const Set& s)   Set(const BitField& bf)  “**interface**”   * int GetMaxPower(void) const; * void InsElem(const int n) * void DelElem(const int n) * int IsMember(const int n) const * int operator== (const Set& s) * Set& operator=(const Set& s) * Set operator+ (const Set& s) * Set operator\* (const Set& s) * Set operator~ (void * friend istream& operator>>(istream& istr, Set& bf) * friend ostream& operator<<(ostream& ostr, const Set&) |

В следующей таблице указано применение данных и методов класса.

|  |  |  |
| --- | --- | --- |
| **№** | **Данные** | **Хранит** |
| 1 | MaxPower | Максимальная мощность множества |
| 2 | BitField BF | Битовое поле для хранения характеристического вектора |
|  | | |
| **№** | **Методы** | **Применение** |
| 1 | GetMaxPower | Получение максимально мощности множества |
| 2 | InsElem | Включить элемент в множество |
| 3 | DelElem | Удаление элемента в множестве |
| 4 | IsMember | Проверка наличия элементов в множестве |
| 5 | operator== | Оператор сравнения |
| 6 | operator= | Оператор присваивания |
| 7 | operator+ | Оператор объединения |
| 8 | operator\* | Оператор пересечения |
| 9 | operator~ | Оператор дополнения |
| 10 | operator>> | Оператор ввода |
| 11 | Operator<< | Оператор вывода |

## 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.

# Приложение

TBitField.h

#pragma once

#ifndef \_\_BITFIELD\_H

#define \_\_BITFIELD\_H

using namespace std;

#include <iostream>

#include <stdlib.h>

typedef unsigned int PP; //

class TBitField

{

private:

int TBitLen; // длина битового поля

PP\* pMem; // динамическая память битового поля

int MemLen; // кол во элементов битового поля

int GetMemIndex(const int n) const; //индекс в pMem для бита n

PP GetMemMask(const int n) const;//битовая маска для бита n

public:

TBitField(int len);

TBitField(const TBitField& bf);

~TBitField();

//доступ к битам

int GetLength(void) const; //gлучить длину колличества битов

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

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

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

// битовые операции

int operator== (const TBitField& bf);

TBitField& operator=(const TBitField& bf);

TBitField& operator|(const TBitField& bf);

TBitField& operator&(const TTBitField& bf);

TBitField& operator~(void); //отрицание

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

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

};

#endif

TBitField.cpp

#include "TBitField.h"

#include <stdio.h>

#include <cmath>

using namespace std;

BitField::TBitField(int len) : TBitLen(len) {

double tmp;

tmp = len / 32.0;

MemLen = ceil(tmp);

pMem = new PP[MemLen];

if (pMem != NULL)

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

pMem[i] = 0;

}

TBitField::TBitField(const TBitField& bf) {

TBitLen = bf.TBitLen;

MemLen = bf.MemLen;

pMem = new PP[MemLen];

if (pMem != NULL)

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

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

}

TBitField :: ~TBitField() {

pMem = NULL;

delete pMem;

}

int TBitField::GetMemIndex(const int n) const { // поменять на 32 бит вместо 16 бит

return n >> 5;

}

PP TBitField::GetMemMask(const int n) const {

return 1 << (n & 15);

}

int TBitField::GetLength(void) const {

return BitLen;

}

void TBitField::SetBit(const int n) {

if ((n > -1) && (n < BitLen))

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

}

void TBitField::ClrBit(const int n) {

if ((n > -1) && (n < BitLen))

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

}

int TBitField::GetBit(const int n) const {

if ((n > -1) && (n < BitLen))

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

return 0;

}

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

TBitLen = bf.TBitLen;

if (MemLen != bf.MemLen) {

MemLen = bf.MemLen;

if (pMem != NULL) delete pMem;

pMem = new PP[MemLen];

}

if (pMem != NULL)

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

return \*this;

}

int TBitField :: operator==(const TBitField& bf) {

int res = 1;

if (TBitLen != bf.TBitLen)

res = 0;

else

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

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

res = 0;

break;

}

return res;

}

TBitField& TBitField :: operator| (const TBitField& bf) {

int i, len = TBitLen;

if (bf.TBitLen > len)

len = bf.TBitLen;

TBitField temp(len);

for (int 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) {

int i, len = TBitLen;

if (bf.TBitLen > len)

len = bf.TBitLen;

TBitField temp(len);

for (int 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) {

int i, len = BitLen;

TBitField temp(len);

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

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

return temp;

}

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

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) {

int len = bf.GetLength();

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

if (bf.GetBit(i))

ostr << '1';

else

ostr << '0';

return ostr;

}

TSet.h

#pragma once

#ifndef \_\_SET\_H

#define \_\_SET\_H

#include "TBitField.h"

using namespace std;

class Set

{

private:

int MaxPower; // максимальная мощность хранения

TBitField BF; // битовое поле для хранения харктеристического вектора

public:

Set(int mp);

Set(const Set& s);// копирование

Set(const TBitField& bf);//преобразование типа

int GetMaxPower(void) const; // получение максимальной мощности

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 элемента в множестве

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

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

Set operator+ (const Set& s);/// включить элемент в множество

Set operator\* (const Set& s);//пересечение

Set operator~ (void);//дополнение

friend istream& operator>>(istream& istr, Set& bf);

friend ostream& operator<<(ostream& ostr, const Set&);

};

#endif

TSet.cpp

#include "TSet.h"

TSet::TSet(int mp) :

MaxPower(mp), BF(mp) {

}

TSet::TSet(const Set& s) :

MaxPower(s.MaxPower), BF(s.BF) {

}

TSet::TSet(const BitField& bf) :

MaxPower(bf.GetLength()), BF(bf) {

}

//TSet :: operator TBitField() {

// TBitField temp(this->BF);

// return temp;

//}

int TSet::GetMaxPower(void) const {

return MaxPower;

}

int TSet::IsMember(const int Elem) const {

return BF.GetBit(Elem);

}

void TSet::InsElem(const int Elem) {

BF.SetBit(Elem);

}

void TSet::DelElem(const int Elem) {

BF.ClrBit(Elem);

}

TSet& TSet ::operator= (const Set& s) {

BF = s.BF;

MaxPower = s.GetMaxPower();

return \*this;

}

int TSet :: operator== (const Set& s) {

return BF == s.BF;

}

TSet TSet :: operator+ (const Set& s) {

Set temp(BF | s.BF);

return temp;

}

TSet TSet :: operator\* (const Set& s) {

Set temp(BF & s.BF);

return temp;

}

TSet TSet :: operator~ (void) {

Set temp(~BF);

return temp;

}

istream& operator>>(istream& istr, Set& s) {

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 Set& s) {

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;

}

Main.cpp

#include "TSet.h"

#include <iomanip>

#include <conio.h>

int main(int argc, char\* argv[])

{ int n, m, k, count;

setlocale(LC\_ALL, "Russian");

cout << "Тестирование программы поддержки множества" << endl;

cout << " Решето Эратосфена" << endl;

cout << "ВВедите верхнюю границу целых значений ";

cin >> n;

Set s(n + 1);

// заполнения множества

for (m = 2; m <= n; m++)

s.InsElem(m);

//проверка до sqrt(n) и удаление кратных

for (m = 2; m \* m <= n; m++)

// если m в s, удаление кратных

if (s.IsMember(m))

for (k = 2 \* m; k <= n; k += m)

if (s.IsMember(k))

s.DelElem(k);

//оставщиеся в s элементы - простые числа

cout << endl << "Печать множества простых чисел " << endl;

count = 0; k = 1;

for (m = 2; m <= n; m++)

if (s.IsMember(m))

{

count++;

cout << setw(5) << m << "";

if (k++ % 10 == 0) cout << endl;

}

cout << endl;

cout << "В первых " << n << " числах " << count << " простых " << endl;

\_getch();

return 0;

}