МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

Высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий.**

**Отчет по лабораторной работе №1**

**«Структура хранения множества»**

**Выполнил:** студент группы 381903-3

Зинкин Кирилл Сергеевич

**Проверил:**

ассистент каф. МОСТ ИИТММ 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

**Множество** — тип и **структура** данных в информатике, которая является реализацией математического объекта множество. Данные типа множество позволяют хранить ограниченное число значений определенного типа без определенного типа.

**Множество** — это математический объект сам являющийся наборов, совокупностью, собранием каких-либо объектов, которые называются элементами этого множества и обладают общим для всех их характеристическим свойством.

**Структура данных** — это программная единица, позволяющая хранить и обрабатывать множество однотипных и/или логически связанных данных в вычислительной технике. Для добавления, поиска, изменения и удаления структура данных предоставляет собой некоторый набор функций, составляющих ее интерфейс.

# 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 подсчитывается количество простых чисел в указанном диапазоне.

****

Рис2. Ввод и подсчет чисел

## Руководство программиста

## Описание структуры программы

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

Битовое Поле:

Iny BitLen

PP\* pMem (insigned int)

int MemLen

Конструктор:

TBitField(int len

TBitField(const BitField& bf)

Деструктор:

~TBitField()

Интерфейс:

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 | CleanBit | 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»**

Класс, реализующий обработку Множеств

Конструктор:

Set(int mp)

Set(const Set& s)

Set(const BitField& bf)

Интерфейс:

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 «CleanBit »

Данный метод позволяет очистить бит в позиции n.

Ниже приведен псевдокод алгоритма:

**Функция CleanBit (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 «=»).

6) Метод «operator==»

Данный метод перегружает операцию сравнения и возвращает 1 – истина или 0 - ложь (знак «==»).

Ниже приведен псевдокод алгоритма:

7) Метод «operator|»

Данный метод перегружает операцию «ИЛИ» (знак «|»).

8) Метод «operator&»

Данный метод перегружает операцию «И» (знак «&»).

9) Метод «operator~»

Данный метод перегружает операцию «отрицание» (знак «~»).

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.CleanBit(Elem);

}

5) Метод «operator=»

6) Метод «operator==»

Данный метод перегружает операцию сравнения и возвращает 1 – истина или 0 - ложь (знак «==»).

8) Метод «operator\*»

Данный метод перегружает операцию «пересечение» (знак «\*»).

9) Метод «operator~»

Данный метод перегружает операцию «дополнение» (знак «~»).

# 

# 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, был подключен тревис.

# 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

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

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

#include <iostream>

#include <fstream>

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)

void file();

};

#endif

TBitField.cpp

#include "tbitfield.h"

TBitField::TBitField(**int** len)

{

**if** (len < 0)

**throw** - 1;

BitLen = len;

MemLen = BitLen / (8 \* **sizeof**(TELEM)) + 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 >= 0 && n < BitLen)

{

**return** n >> 5;

}

**else**

**throw** - 1;

}

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

{

**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** - 8;

**int** i = GetMemIndex(n);

pMem[i] |= GetMemMask(n);

}

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

{

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

**throw** - 1;

**int** i = GetMemIndex(n);

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

}

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

{

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

**throw** 8;

**int** i = GetMemIndex(n);

**if** ((pMem[i] & GetMemMask(n)) != 0)

{

**return** 1;

}

**return** 0;

}

// битовые операции

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

{

**if** (**this** == &bf)

**return** \***this**;

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

{

**if** (**this** == &bf)

**return** 1;

**if** (BitLen != bf.BitLen)

**return** 0;

**for** (**int** i = 0; i < BitLen; i++)

**if** (GetBit(i) != bf.GetBit(i))

**return** 0;

**return** 1;

}

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

{

**if** (\***this** == bf)

**return** 0;

**return** 1;

}

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

{

**if** (MemLen > bf.MemLen)

{

TBitField result(\***this**);

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

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

**return** result;

}

**else**

{

TBitField result(bf);

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

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

**return** result;

}

}

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

{

**if** (BitLen > bf.BitLen)

{

TBitField result(BitLen);

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

**if** (GetBit(i) & bf.GetBit(i))

result.SetBit(i);

**return** result;

}

**else**

{

TBitField result(bf.BitLen);

**for** (**int** i = 0; i < BitLen; i++)

**if** (GetBit(i) & bf.GetBit(i))

result.SetBit(i);

**return** result;

}

}

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

{

TBitField bf(BitLen);

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

{

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

}

**return** bf;

}

// ввод/вывод

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

{

**char** c;

**int** k = 0;

**do**

{

istr >> c;

**if** (c == '1')

{

bf.SetBit(k);

}

**if** (c == '0')

{

bf.ClrBit(k);

}

k++;

} **while** ((c == '1') || (c == '0'));

**return** istr;

}

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

{

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

{

ostr << bf.GetBit(i);

}

**return** ostr;

}

**void** TBitField::file()

{

ofstream out;

out.open("Bit.txt", ios::app);

**if** (out.is\_open())

{

out << \***this** << endl;

}

cout << endl << "File written" << endl;

}

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"

#include "tbitfield.h"

TSet::TSet(**int** mp) : BitField(mp)

{

MaxPower = mp;

}

// конструктор копирования

TSet::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()

{

**return** BitField;

}

**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

{

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 temp(BitField | s.BitField);

**return** temp;

}

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

{

TSet res(BitField);

**if** (Elem > MaxPower)

**throw** - 1;

res.InsElem(Elem);

**return** res;

}

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

{

TSet res(BitField);

res.DelElem(Elem);

**return** res;

}

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

{

TSet temp(BitField & s.BitField);

**return** temp;

}

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

{

TSet res(~BitField);

**return** res;

}

// перегрузка ввода/вывода

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

{

**int** i;

cin >> i;

**for** (; ((i > -1) && (i < s.MaxPower)); cin >> i)

s.BitField.SetBit(i);

**return** istr;

}

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

{

cout << "MaxPower is " << s.MaxPower << endl;

cout << s.BitField;

**return** ostr;

}

}

sample\_prime\_numbers.cpp

#include <iomanip>

// #define USE\_SET // Использовать класс TSet,

// закоментировать, чтобы использовать битовое поле

#ifndef USE\_SET // Использовать класс TBitField

#include "tbitfield.h"

**int** main()

{

**int** n, m, k, count;

setlocale(LC\_ALL, "Russian");

cout << "Тестирование программ поддержки битового поля" << endl;

cout << " Решето Эратосфена" << endl;

cout << "Введите верхнюю границу целых значений - ";

cin >> n;

TBitField s(n + 1);

// заполнение множества

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

s.SetBit(m);

s.file();

// проверка до sqrt(n) и удаление кратных

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

// если m в s, удаление кратных

**if** (s.GetBit(m))

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

**if** (s.GetBit(k))

s.ClrBit(k);

// оставшиеся в s элементы - простые числа

cout << endl << "Печать множества некратных чисел" << endl << s << endl;

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

count = 0;

k = 1;

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

**if** (s.GetBit(m))

{

count++;

cout << setw(3) << m << " ";

**if** (k++ % 10 == 0)

cout << endl;

}

cout << endl;

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

}

#else

#include "tset.h"

**int** main()

{

**int** n, m, k, count;

setlocale(LC\_ALL, "Russian");

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

cout << " Решето Эратосфена" << endl;

cout << "Введите верхнюю границу целых значений - ";

cin >> n;

TSet s(n + 1);

// заполнение 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 м в s, удаление кратных

**if** (s.IsMember(m))

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

**if** (s.IsMember(k))

s.DelElem(k);

// оставшиеся в s элементы - простые числа

cout << endl << "Печать множества некратных чисел" << endl << s << endl;

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

count = 0;

k = 1;

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

**if** (s.IsMember(m))

{

count++;

cout << setw(3) << m << " ";

**if** (k++ % 10 == 0)

cout << endl;

}

cout << endl;

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

}

#endif