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»**

Выполнил:

студент ИИТММ группы 381908-4

Яшин Егор Олегович

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

**Множество** — одно из ключевых понятий математики; 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. Для того, чтобы понять и разобраться, как устроено множество, была написана программа.

# 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.

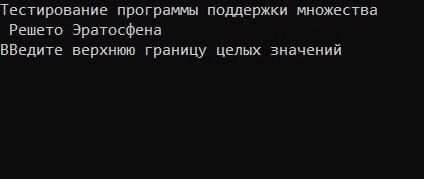
****

Рис1. Старт работы

После корректного ввода границы, программа считает и печатает нам простые числа,

подсчитывает сколько простых чисел в указанном диапазоне.

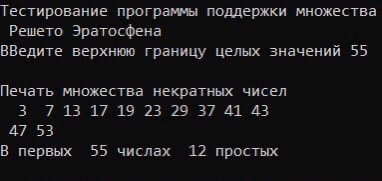
****

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 .

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.

# 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;

}