Новосибирский государственный технический

университет

Кафедра ТПИ

Объектно-ориентированное программирование

Лабораторная работа №2

**Классы**

Выполнили: Преподаватели:

Яшков Иван Дворецкая Виктория Константиновна

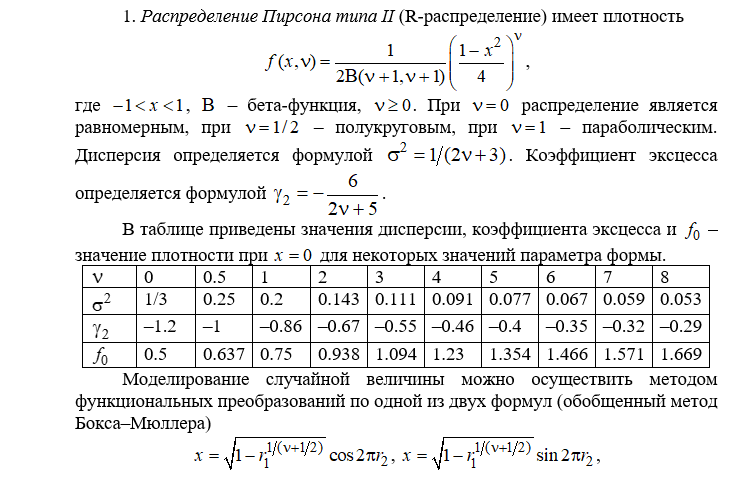
Гавриленко Андрей Лисицин Даниил Валерьевич

Группа ПМИ-13

Бригада 1

Новосибирск, 2023

**Задание (вариант 1):**

**Описание разработанного класса:**

**Атрибуты:**

Атрибутами класса являются параметры сдвига (mu), масштаба (la) и формы (nu), а также массив числовых характеристик распределения (математическое ожидание, дисперсия коэффициенты асимметрии и эксцесса). Уровень доступа этих атрибутов private, обращение к атрибутам будет производиться с помощью специальных функций.

**Функции:**

1. **void setMU(double mu0)** и **double getMU()** нужны для изменения параметра mu в классе и обращения к параметру mu соответственно.

Функция **void setMU()** проверяет, чтобы пользователь не ввел число, которое меньше -10000 или больше 10000, иначе генерирует исключительную ситуацию;

Функция **double getMU()** возвращает значение атрибута mu.

2. void **setLA(double la0)** и **double getLA ()** нужны для изменения параметра la в классе и обращения к параметру la соответственно.

Функция **void setLA()** проверяет, чтобы пользователь не ввел число, которое меньше или равно 0, иначе генерирует исключительную ситуацию;

Функция **double getLA()** возвращает значение атрибута la.

3. void **setNU(double nu0)** и **double getNU()** нужны для изменения параметра nu в классе и обращения к параметру nu соответственно.

Функция **void setNU()** проверяет, чтобы пользователь не ввел число, которое меньше 0, иначе генерирует исключительную ситуацию;

Функция **double getNU()** возвращает значение атрибута nu.

4. **void set\_moments()** и **double\* get\_moments()** нужны для генерации числовых характеристик распределения в классе и обращения к числовым характеристикам соответственно.

Функция **void set\_moments()** имеет уровень доступа private и вызывается автоматически при изменении одного из параметров распределения (mu, la, nu), то есть при вызове других set-функций и конструктора

Функция **double\* get\_moments()** возвращает массив числовых характеристик распределения.

5. **double get\_f(double x)** возвращает значение функции плотности в точке x.

6. **double get\_model()** моделирует случайную величину на основе параметров класса.

7. **void load(string name)** и **void save(string name)** считывают значения файла и сохраняют значения в файл соответственно.

Функция **void load(string name)** получает на вход название файла, из которого следует считать значения. Если файла с таким названием не существует - генерируется исключительная ситуация. Далее файл проверяется на содержание трех элементов - mu, la, nu. Если параметров не хватает, вызывается исключительная ситуация. Все параметры после третьего не считываются. Затем проверяется, являются ли все три параметра числами. Если хотя бы один из параметров не число, вызывается исключительная ситуация. Если все параметры прошли эту проверку, последовательно вызываются функции setMU(), setLA(), setNU(), которые дополнительно проверяют правильность введенных параметров и заносят их в соответствующие атрибуты.

Функция **void save(string name)** получает на вход название файла, в который следует записать значения mu, la, nu. Если файла с таким названием не существует - генерируется исключительная ситуация. Далее в файл последовательно записываются параметры через пробел, при этом если в файле уже была какая-либо информация, она стирается.

8. **generate\_distribution** генерирует выборку из 10000 элементов заданного распределения и ставит им в соответствие значения плотностей. Для генерации выборки используется get\_model, для вычисления функции плотности get\_f. Полученные значения записываются в два файла, один - со значениями x, другой - с значениями функции плотности в соответствующей точке.

Все функции кроме **set\_moments** имеют уровень доступа Public. **set\_moments** имеет уровень доступа Private.

**Конструкторы:**

1. Конструктор, ожидающий передачи параметров сдвига, масштаба и формы. **PearsonDistributionType2(double mu0 = 0, double la0 = 1, double nu0 = 0)** Имеет значения по умолчанию, в случае, если пользователь не передал в конструктор ни одного параметра, генерирует равномерное распределение без сдвига со стандартным масштабом.

2. Конструктор, считывающий атрибуты (параметры сдвига, масштаба и формы) из файла. **PearsonDistributionType2(std::string name)** Получает на вход название файла и далее вызывает функцию **void load(string name)**.

**Описание разработанной программы:**

Программа содержит 3 исходных файла: main.cpp, default.cpp, test.cpp, и 1 заголовочный: Header.h.

**Header.h -** заголовочный файл, содержит объявление необходимых библиотек, описание класса PearsonDistributionType2 и функции test, а также константу PI.

**default.cpp -** файл, содержащий описание всех функций класса PearsonDistributionType2.

**main.cpp** - файл, содержащий функцию main, в которой создается объект класса и вызываются функции test и generate\_distribution.

**test.cpp -** файл, содержащий функцию test, тестирующую класс.

Функция test содержит следующие проверки:

1. Проверка первого конструктора класса: проверить параметры, вывести значения атрибутов. Данная проверка также включает в себя проверку функций getMU, getLA, getNU, get\_moments, так как они вызываются при выводе значений на экран.

2. Проверка сеттеров: попытаться ввести допустимые и недопустимые значения для каждого параметра, вывести полученные сообщения на экран.

3. Проверка функций get\_f и get\_model: ввести некоторые значения, вывести результаты на экран.

4. Проверка функций load и save: попытаться ввести названия существующего и несуществующего файлов, для функции load попробовать ввести разное количество параметров (0, 1, 2, 3) и после посмотреть значения атрибутов.

5. Проверка второго конструктора класса: аналогично проверке функции load.

**Текст программы:**

**Header.h:**

#pragma once

#include <iostream>

#include <fstream>

#include <string>

#include <cmath>

#include <time.h>

const double PI = std::acos(-1);

class PearsonDistributionType2 {

double mu;

double la;

double nu;

double\* moments = new double[4];

public:

PearsonDistributionType2(double mu0 = 0, double la0 = 1, double nu0 = 0);

PearsonDistributionType2(std::string name);

void setMU(double mu0);

double getMU();

void setLA(double la0);

double getLA();

void setNU(double nu0);

double getNU();

double\* get\_moments();

double get\_f(double x);

double get\_model();

void load(std::string name);

void save(std::string name);

void generate\_distribution();

private:

void set\_moments();

};

void test();

**main.cpp:**

#include "Header.h"

int main() {

srand(time(NULL));

PearsonDistributionType2 p(4.3, 6.1, 10);

test();

p.generate\_distribution();

return 0;

}

**default.cpp:**

#include "Header.h"

PearsonDistributionType2::PearsonDistributionType2(double mu0, double la0, double nu0) {

setMU(mu0);

setLA(la0);

setNU(nu0);

};

PearsonDistributionType2::PearsonDistributionType2(std::string name) {

load(name);

}

void PearsonDistributionType2::setMU(double mu0) {

mu = mu0 >= -10000 && mu0 <= 10000 ? mu0 : throw "ERROR: Bad mu. mu should be: -10000 <= mu <= 10000.";

set\_moments();

}

double PearsonDistributionType2::getMU() {

return mu;

}

void PearsonDistributionType2::setLA(double la0) {

la = la0 > 0 && la <= 10000 ? la0 : throw "ERROR: Bad la. la should be: 0 < la <= 10000.";

set\_moments();

}

double PearsonDistributionType2::getLA() {

return la;

}

void PearsonDistributionType2::setNU(double nu0) {

nu = nu0 >= 0 && nu <= 10000 ? nu0 : throw "ERROR: Bad nu. nu should be: 0 <= nu <= 10000.";

set\_moments();

}

double PearsonDistributionType2::getNU() {

return nu;

}

double PearsonDistributionType2::get\_f(double x) {

if (x > -1. \* la + mu && x < 1. \* la + mu) {

return pow((1 - pow((x - mu) / la, 2)) / 4., nu) / (2 \* std::tgamma(nu + 1) \* std::tgamma(nu + 1) / std::tgamma(2 \* nu + 2)) / la;

}

return 0;

}

void PearsonDistributionType2::set\_moments() {

moments[0] = mu;

moments[1] = pow(la, 2) / (2 \* nu + 3);

moments[2] = 1 / pow(moments[1], 3. / 2.) \* 1 / 3 \* (moments[0] - mu);

moments[3] = -6. / (2 \* nu + 5);

}

double\* PearsonDistributionType2::get\_moments() {

return moments;

}

double PearsonDistributionType2::get\_model() {

double r1, r2;

do r1 = (double)rand() / RAND\_MAX; while (r1 == 0. || r1 == 1.);

do r2 = (double)rand() / RAND\_MAX; while (r2 == 0. || r2 == 1.);

return sqrt(1 - pow(r1, 1 / (nu + 1. / 2.))) \* cos(2 \* PI \* r2) \* la + mu;

}

void PearsonDistributionType2::load(std::string name) {

std::ifstream file;

int counter = 0;

double temp;

std::string\* args = new std::string[3];

file.open(name);

if (file.fail()) {

throw "ERROR: Bad file name";

}

else {

while (!file.eof() && counter < 3) {

file >> args[counter];

counter++;

}

if (counter < 3) {

throw "ERROR: File should consist 3 parameters: mu, la, nu.";

}

for (int i = 0; i < 3; i++) {

try {

temp = stod(args[i]);

}

catch (std::invalid\_argument) {

throw "ERROR: One of the parameters is not a number. Check your file and try again.";

}

}

setMU(stod(args[0]));

setLA(stod(args[1]));

setNU(stod(args[2]));

}

file.close();

}

void PearsonDistributionType2::save(std::string name) {

std::ifstream check;

std::ofstream file;

check.open(name);

if (check.fail()) {

throw "ERROR: Bad file name";

}

else {

check.close();

file.open(name, std::ios::out);

file << mu << " " << la << " " << nu;

}

file.close();

}

void PearsonDistributionType2::generate\_distribution() {

std::ofstream file1, file2;

std::string s\_number;

double number;

file1.open("x.txt", std::ios::out);

file2.open("fx.txt", std::ios::out);

for (int i = 0; i < 10000; i++) {

number = get\_model();

s\_number = std::to\_string(number);

s\_number[s\_number.find(".")] = ',';

file1 << s\_number << "\n";

s\_number = std::to\_string(get\_f(number));

s\_number[s\_number.find(".")] = ',';

file2 << s\_number << "\n";

}

file1.close();

}

**test.cpp:**

#include "Header.h"

void test() {

double\* m;

double number;

std::string input[5] = {"f.txt", "file1", "file1.txt", "file2.txt", "file3.txt"};

std::string output[3] = { "o.txt", "out1", "out1.txt" };

PearsonDistributionType2 p1;

//test 1 - first constructor

std::cout << "\t\t----------test 1----------" << std::endl;

if (p1.getMU() == 0 && p1.getLA() == 1 && p1.getNU() == 0) {

std::cout << ":D First constructor is correct:" << std::endl;

}

else {

std::cout << ":C First constructor is incorrect:" << std::endl;

}

m = p1.get\_moments();

std::cout << "\tmu la nu: "<< p1.getMU() << " " << p1.getLA() << " " << p1.getNU() << "; " << std::endl;

std::cout << "\tmoments: " << m[0] << " " << m[1] << " " << m[2] << " " << m[3] << std::endl << std::endl;

//test 2 - set

std::cout << "\t\t----------test 2----------" << std::endl;

p1.setMU(10);

p1.setLA(5);

p1.setNU(1);

if (p1.getMU() == 10 && p1.getLA() == 5 && p1.getNU() == 1) {

std::cout << ":D If you try to enter correct parameters in set functions all will be correct:" << std::endl;

std::cout << "\tmu la nu: " << p1.getMU() << " " << p1.getLA() << " " << p1.getNU() << "; " << std::endl;

std::cout << "\tmoments: " << m[0] << " " << m[1] << " " << m[2] << " " << m[3] << std::endl;

}

else {

std::cout << ":C When you try to enter correct parameters in set functions something is not correct" << std::endl;

std::cout << "\tmu la nu: " << p1.getMU() << " " << p1.getLA() << " " << p1.getNU() << "; " << std::endl;

std::cout << "\tmoments: " << m[0] << " " << m[1] << " " << m[2] << " " << m[3] << std::endl;

}

try {

p1.setMU(-100000);

}

catch (const char\* error) {

std::cout << ":D If you try to enter incorrect parameter to mu you will get that message:\n\t";

std::cout << error << std::endl;

std::cout << "\tmu la nu: " << p1.getMU() << " " << p1.getLA() << " " << p1.getNU() << "; " << std::endl;

std::cout << "\tmoments: " << m[0] << " " << m[1] << " " << m[2] << " " << m[3] << std::endl;

}

try {

p1.setLA(-10);

}

catch (const char\* error) {

std::cout << ":D If you try to enter incorrect parameter to la you will get that message:\n\t";

std::cout << error << std::endl;

std::cout << "\tmu la nu: " << p1.getMU() << " " << p1.getLA() << " " << p1.getNU() << "; " << std::endl;

std::cout << "\tmoments: " << m[0] << " " << m[1] << " " << m[2] << " " << m[3] << std::endl;

}

try {

p1.setNU(-1);

}

catch (const char\* error) {

std::cout << ":D If you try to enter incorrect parameter to nu you will get that message:\n\t";

std::cout << error << std::endl;

std::cout << "\tmu la nu: " << p1.getMU() << " " << p1.getLA() << " " << p1.getNU() << "; " << std::endl;

std::cout << "\tmoments: " << m[0] << " " << m[1] << " " << m[2] << " " << m[3] << std::endl << std::endl;

}

//test 3 - get\_f, get\_model

std::cout << "\t\t----------test 3----------" << std::endl;

number = 10.;

std::cout << "get\_f(" << number << ") = " << p1.get\_f(number) << std::endl;

number = 14.;

std::cout << "get\_f(" << number << ") = " << p1.get\_f(number) << std::endl;

number = -2.;

std::cout << "get\_f(" << number << ") = " << p1.get\_f(number) << std::endl;

for (int i = 0; i < 5; i++) {

number = p1.get\_model();

if (number >= p1.getMU() - p1.getLA() && number <= p1.getMU() + p1.getLA()) {

std::cout << ":D get\_model(" << p1.getMU() << ", " << p1.getLA() << ", " << p1.getNU() << ") = " << number << " correct: " << p1.getMU() - p1.getLA() << " <= " << number << " <= " << p1.getMU() + p1.getLA() << std::endl;

}

else {

std::cout << ":C get\_model(" << p1.getMU() << ", " << p1.getLA() << ", " << p1.getNU() << ") = " << number << " incorrect" << std::endl;

}

}

std::cout << std::endl;

//test 4 - load, save

//file1.txt - ok, file2.txt - not enough params, file3.txt - consist not only numbers

std::cout << "\t\t----------test 4----------" << std::endl;

for (int i = 0; i < 5; i++) {

std::cout << "Name of file: " << input[i] << std::endl;

try {

p1.load(input[i]);

std::cout << "\tFile was successfully read." << std::endl;

std::cout << "\tmu la nu: " << p1.getMU() << " " << p1.getLA() << " " << p1.getNU() << "; " << std::endl;

std::cout << "\tmoments: " << m[0] << " " << m[1] << " " << m[2] << " " << m[3] << std::endl;

}

catch (const char\* error) {

std::cout << "\t" << error << std::endl;

}

}

std::cout << std::endl;

for (int i = 0; i < 3; i++) {

std::cout << "Name of file: " << output[i] << std::endl;

try {

p1.save(output[i]);

std::cout << "\tInformation was successfully entered in the file." << std::endl;

}

catch (const char\* error) {

std::cout << "\t" << error << std::endl;

}

}

std::cout << std::endl;

//test 5 Second constructor

std::cout << "\t\t----------test 5----------" << std::endl;

for (int i = 0; i < 5; i++) {

std::cout << "Name of file: " << input[i] << std::endl;

try {

PearsonDistributionType2 p2(input[i]);

std::cout << "\tFile was successfully read." << std::endl;

std::cout << "\tmu la nu: " << p2.getMU() << " " << p2.getLA() << " " << p2.getNU() << "; " << std::endl;

std::cout << "\tmoments: " << m[0] << " " << m[1] << " " << m[2] << " " << m[3] << std::endl;

}

catch (const char\* error) {

std::cout << "\t" << error << std::endl;

}

}

std::cout << std::endl;

}

**Вызов функции test:**

Функция test состоит из 5 блоков, каждый из которых проверяет тот или иной функционал класса.

**1. Проверка первого конструктора класса:**

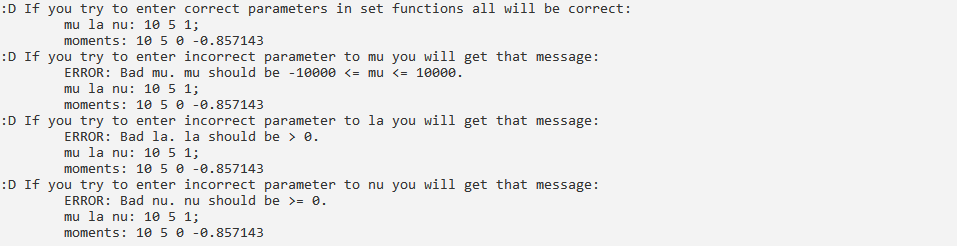
Сообщение в консоли:



Блок проверяет значения mu, la, nu, если они равны значениям по умолчанию, то все правильно, выводим атрибуты класса.

**2. Проверка сеттеров:**

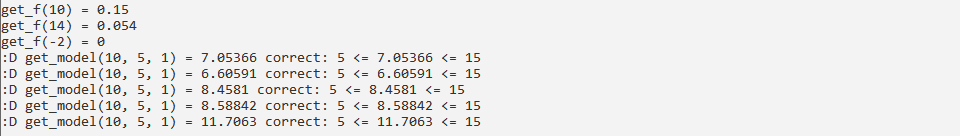
Сообщение в консоли:



Блок сначала проверяет возможность введения допустимых значений, затем недопустимых, в случае ошибок блок выводит их текст, если ошибок нет блок выводит значения атрибутов.

**3. Проверка get\_f и get\_model:**

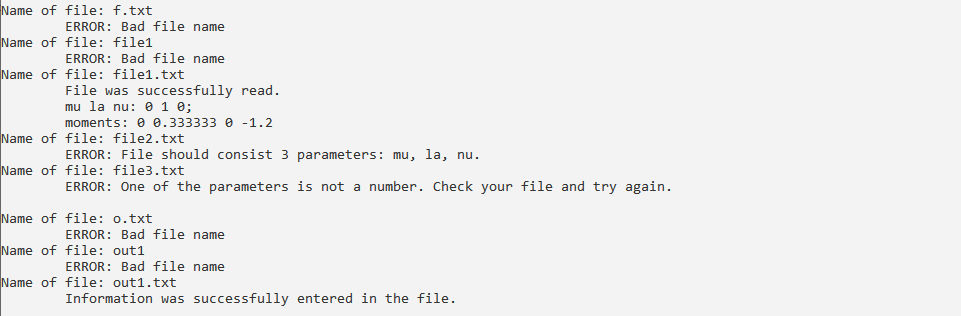
Сообщение в консоли:



Блок выводит значения функции плотности распределения в некоторых точках, затем выводит значения моделирования, и если эти значения лежат в области определения распределения, то считается, что тест пройден.

**4. Проверка load и save:**

Сообщение в консоли:



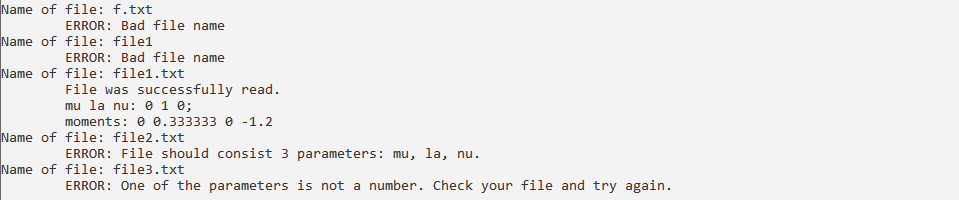
Существуют четыре файла: file1.txt, file2.txt, file3.txt, out1.txt.

Программа сначала пытается считать значения из несуществующих файлов (f.txt, file1), затем пытается прочитать первые три файла из списка. Файл file1.txt содержит правильно записанную информацию о распределении - три параметра mu, la, nu через пробелы. Он прочитывается правильно и блок выводит значения атрибутов. Файл file2.txt содержит недостаточно параметров: 0, 1 или 2 из трех необходимых. При попытке записи в атрибуты вызывается ошибка, сообщающая о недостаточной наполненности файла. Файл file3.txt содержит значение, не являющееся числом (abc). При попытке записи в атрибуты вызывается ошибка, сообщающая о недопустимом значении в файле.

После программа пытается записать значения в несуществующие файлы (o.txt, out), затем пытается записать в последний файл из списка. Параметры mu, la, nu успешно записываются в файл out1.txt.

**5 Проверка второго конструктора класса:**

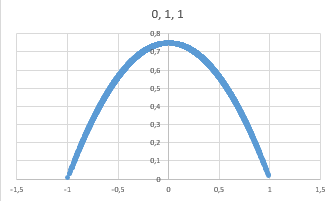
Сообщение в консоли:



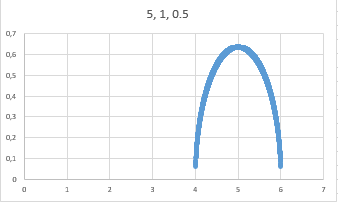
Действия в блоке аналогичны действиям, производимым в предыдущем блоке для проверки функции load.

**Графики:**

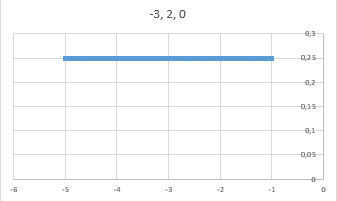
1. Параболическое распределение. mu = 0, la = 1, nu = 1:



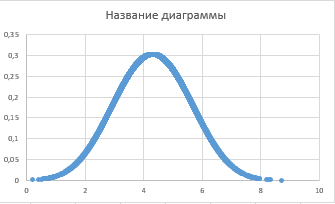
2. Полукруговое распределение. mu = 5, la = 1, nu = 0.5:



3. Равномерное распределение. mu = -3, la = 2, nu = 0:



4. mu = 4.3, la = 6.1, nu = 10:



**Контрольные вопросы проработаны.**