|  |  |  |
| --- | --- | --- |
| Санкт-Петербургский политехнический университет Петра Великого  Институт компьютерных наук и технологий  **Высшая школа программной инженерии** | | |
| **КУРСОВОЙ ПРОЕКТ** | | |
| МОДЕЛИРОВАНИЕ СИСТЕМЫ, ФОРМАЛИЗОВАННОЙ КАК СИСТЕМА МАССОВОГО ОБСЛУЖИВАНИЯ | | |
| по дисциплине «Алгоритмизация и структурное программирование» | | |
| Выполнил | | |
| студент гр. 13534/4 |  | Мащенко Б.Б. |
| Руководитель | | |
| доцент, к.т.н. |  | Веренинов И.А. |
| «25» марта 2019г. | | |
| Санкт-Петербург | | |
| 2019 | | |

ЗАДАНИЕ

НА ВЫПОЛНЕНИЕ КУРСОВОГО ПРОЕКТА

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| студенту группы | | 13534/4 | |  | Мащенко Богдану Борисовичу | | | | | | | | |
|  | | *(номер группы)* | |  | *(фамилия, имя, отчество)* | | | | | | | | |
| ***1. Тема проекта (работы)*** | | | |  | | | | | | | | | |
| *Разработка программной системы моделирования дискретной стохастической системы, формализованной как одноканальная система массового обслуживания с многомерным входным потоком и памятью заявок с применением технологии ООП* | | | | | | | | | | | | | |
| ***2. Срок сдачи студентом законченного проекта работы)*** | | | | | | | | | | | | | *25 марта 2019 года* |
| ***3. Исходные данные к проекту (работе)*** | | | | | | | | |  | | | | |
| *Перечень индивидуальных заданий на проект с указанием конкретного номера задания*  *Методическое пособие по курсовому проектированию* | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
| ***4.Содержание пояснительной записки***(перечень подлежащих разработке вопросов): введение, основная часть (раскрывается структура основной части), заключение, список использованных источников, приложения)*.* | | | | | | | | | | | | | |
| ***Введение с формулировкой конкретного задания*** | | | | | | | | | | | | | |
| ***Инструкция системного программиста с указанием состава, назначения модулей, особенностей размещения драйверов и т.д.*** | | | | | | | | | | | | | |
| ***Инструкция по запуску, особенностей интерфейса, выводу результатов***  ***Примеры результатов в графической и табличной форме*** | | | | | | | | | | | | | |
| ***Полные исходные тексты всех модулей с комментариями полей данных и методов типов объектов*** | | | | | | | | | | | | | |
| Примерный объем пояснительной записки | | | | | | ***65*** | | | | страниц машинописного | | | |
| текста | | | | | | | | | | | | | |
| 5. Перечень графического материала (с указанием обязательных чертежей и | | | | | | | | | | | | | |
| плакатов) | Нет | | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
| 6. Консультанты | | | Нет | | | | | | | | | | |
|  | | | | | | | | | | | | | |
| 7. Дата получения задания: «11» февраля 2019г. | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
| Руководитель | | | | |  | | |  | | | *Веренинов И.А.* | | |
|  | | | | | *(подпись)* | | |  | | | *(инициалы, фамилия)* | | |
| Задание принял к исполнению | | | | |  | |  | | | | *Мащенко Б.Б.* | | |
|  | | | | | *(подпись)* | |  | | | | *(инициалы, фамилия)* | | |
|  | | | | | | | | | | | |  | |
|  | | | | | | | | | | | | *(дата)* | |

Оглавление

[Введение 5](#_Toc3394110)

[Инструкция программиста 6](#_Toc3394111)

[Инструкция оператора 7](#_Toc3394112)

[Главное меню 7](#_Toc3394113)

[Настройка параметров 8](#_Toc3394114)

[Моделирование 9](#_Toc3394115)

[Результаты 12](#_Toc3394116)

[Список использованных источников 15](#_Toc3394117)

[Приложение 1. Исходный код программы SMO.PAS 16](#_Toc3394118)

[Приложение 2. Исходный код модуля BOOS.PAS 17](#_Toc3394119)

[Приложение 3. Исходный код модуля BAS.PAS 23](#_Toc3394120)

[Приложение 4. Исходный код модуля BUFFER.PAS 25](#_Toc3394121)

[Приложение 5. Исходный код модуля DEVICE.PAS 27](#_Toc3394122)

[Приложение 6. Исходный код модуля WRITER.PAS 30](#_Toc3394123)

[Приложение 7. Исходный код модуля STATIST.PAS 33](#_Toc3394124)

[Приложение 8. Исходный код модуля SOURCE1.PAS 38](#_Toc3394125)

[Приложение 9. Исходный код модуля SOURCE2.PAS 40](#_Toc3394126)

[Приложение 10. Исходный код модуля MENU.PAS 42](#_Toc3394127)

[Приложение 11. Исходный код модуля SUPM.PAS 46](#_Toc3394128)

[Приложение 12. Исходный код модуля MENUMOD.PAS 53](#_Toc3394129)

[Приложение 13. Исходный код модуля MENURES.PAS 59](#_Toc3394130)

## Введение

Задача состоит в разработке программной системы моделирования, формализованной как система массового обслуживания (СМО) с двумя входными потоками и памятью заявок с применением технологии ООП.

ДАНО:

* Количество источников - 2
* Потоки заявок - I-простейший , II-равномерный τ1=0.1≤τ≤τ2=0.5
* Объем буфера - 2
* Дисциплина записи в буфер - в порядке поступления с отказами
* Дисциплина выборки из буфера - бесприоритетная
* Поток обслуживания - простейший с λобсл=1 (параметр закона)
* Точность оценки вероятностей - δ=10%, Р=0.9 (оценить Kmin)

НАЙТИ :

* Вероятность отказа Ротказа 2 = f 1 ( λ1 )
* Среднее время ожидания М ( Тожид i ) = f 2 i ( λ1 ) λ1\* =0.5≤ λ1 ≤λ1\*\*

=1.5 с Δλ1 =0.1, i =1,2,

* Среднее число заявок в буфере от первого источника K( λ1 )

## Инструкция программиста

Данный программный продукт состоит из 13 файлов: BAS.PAS, BOOS.PAS, BUFFER.PAS, DEVICE.PAS, WRITER.PAS, STATIST.PAS, SMO.PAS, SOURCE1.PAS, SOURCE2.PAS, MENU.PAS, MENUMOD.PAS, MENURES.PAS, SUPM.PAS.

Модули BAS.PAS, BOOS.PAS, BUFFER.PAS, DEVICE.PAS, WRITER.PAS, STATIST.PAS, SOURCE1.PAS, SOURCE2.PAS обеспечивают работу самой системы массового обслуживания, содержа все необходимые объекты. BOOS.PAS определяет, какое событие произойдет в ближайшее время и отправляет его на обработку. BAS.PAS отвечает за то, куда следует направить сгенерированную заявку. BUFFER.PAS содержит определение и реализацию бесприоритетного буфера заявок на 2 элемента. DEVICE.PAS обеспечивает обработку заявки и расчет времени ее выполнения. WRITER.PAS через объект предоставляет интерфейс для логирования действий системы. STATIST.PAS хранит статистику и выполняет ее простейшую обработку. SOURCE1.PAS и SOURCE2.PAS хранят соответствующие данные об источниках.

MENU.PAS нужен для отрисовки интерфейса и обработки команд пользователя. MENUMOD.PAS, MENURES.PAS, SUPM.PAS представляют собой пользовательские кнопки. SUPM.PAS отвечает за установку начальных параметров источником и начального kmin, MENUMOD.PAS – за начало моделирования системы с возможностью отрисовки online-графика или без него, MENURES.PAS – за отображение статистики моделирования, а также обеспечивает отрисовку графиков.

В файле SMO.PAS представлена головная программа. В ней происходит инициализация основных объектов программы и запускается главное меню.

Более подробные комментарии приведены в исходном коде программы в приложениях 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 , 13.

## Инструкция оператора

### Главное меню

При запуске данной программы открывается главное меню.

Навигация по интерфейсу:

* Перемещение сфокусированного элемента - стрелка вправо, влево для главного меню, вверх и вниз для подменю.
* Открыть выбранный пункт - клавиша Enter.

Закрыть подменю / выйти из программы - клавиша ЕND.



*Рис. 1. Главное меню.*

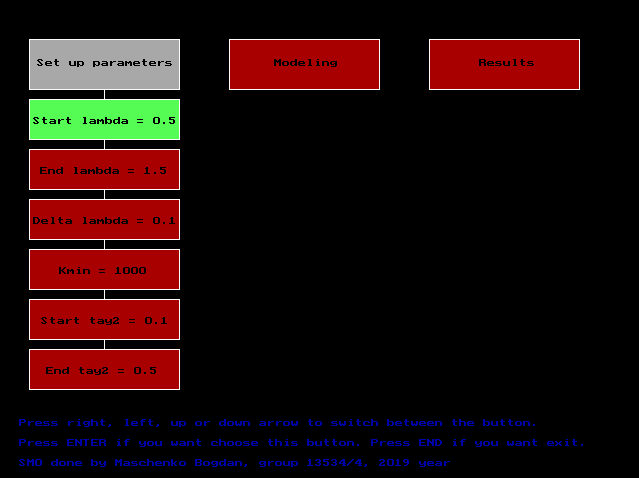
### Настройка параметров

Настройка параметров моделирования производиться в меню «Config».

* Start lambda, End lambda – границы параметров 1-го потока. Параметр Delta lambda -шаг изменения интенсивности первого источника.
* Kmin – длина реализации.
* Start tay2, End tay2 – параметры 2-го источника.

Ввод данных в текстовых полях:

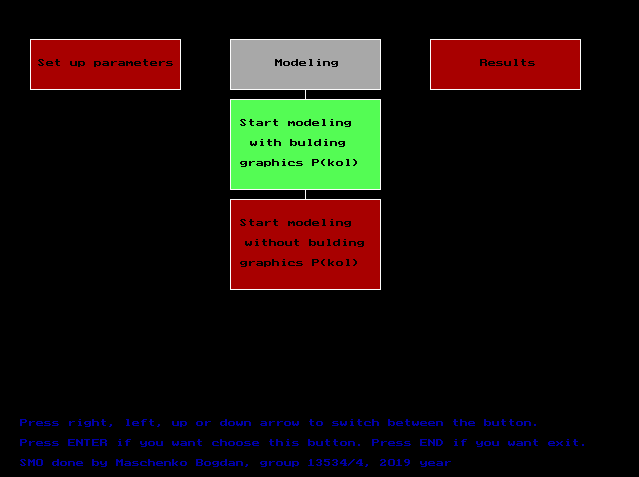
* Ввод значения – клавиши 0-9 (цифры), точка в качестве разделителя целой и дробной части.
* Подтвердить – клавиша Enter.



*Рис. 2. Ввод параметров моделирования (меню Set up parameters).*

### Моделирование

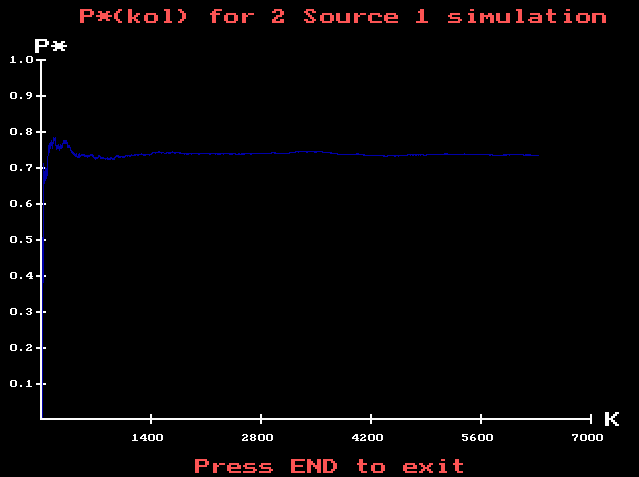
Моделирование происходит в меню Modeling.



*Рис. 3. Меню Modeling.*

Пользователю предоставляется два варианта:

* Моделирование с отрисовкой online-графика вероятности отказа заявок от второго источника Pотказа 1(k), где k - количество заявок для 1 симуляции.

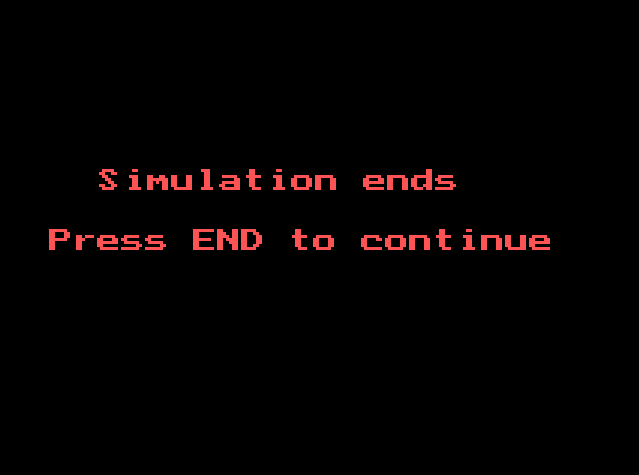


*Рис. 4. Подменю Start modeling with building graphics P(kol).*

* Моделирование без отрисовки online-графика. Simulation started означает, что симуляция началась, Simulation end – закончилась и для дальнейших действий требуется нажать END.



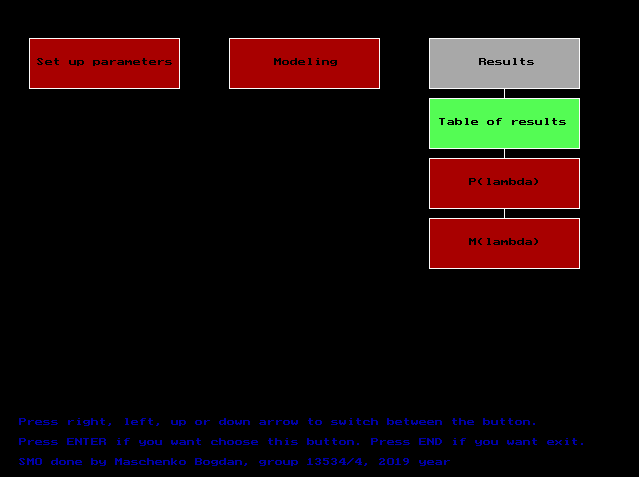
*Рис. 5. Подменю Start modeling without building graphics P(kol). Simulation started.*



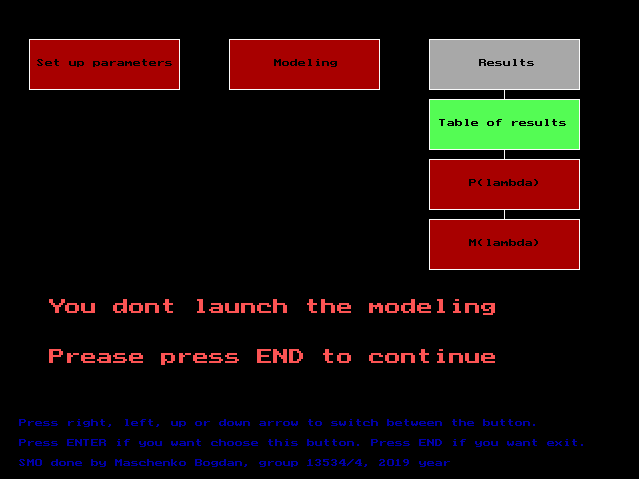
*Рис. 6. Подменю Start modeling without building graphics P(kol). Simulation ends.*

### Результаты

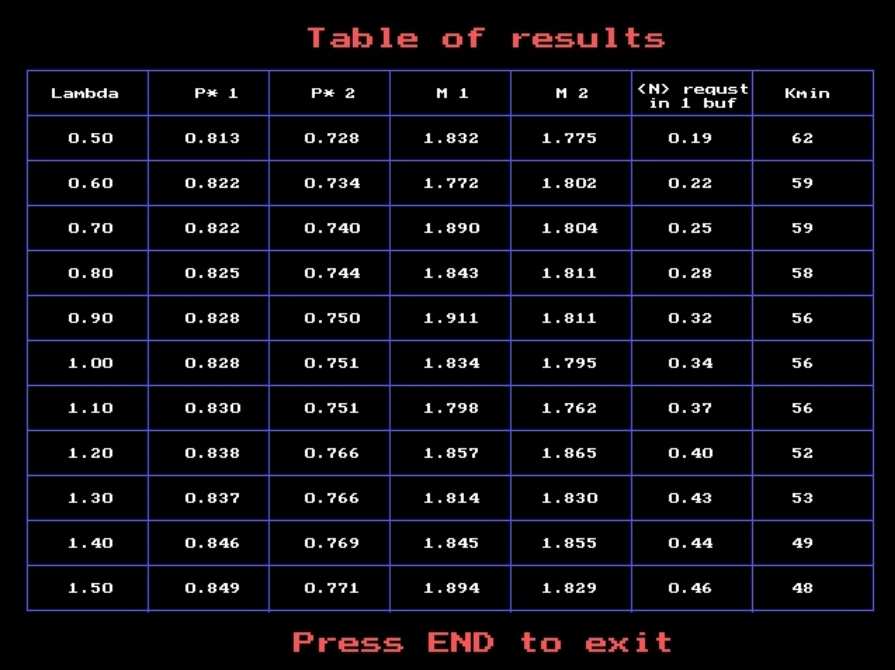
Меню Results содержит три опции: просмотр Table of results, графика зависимости вероятности отказа от интенсивности прибора P(lambda) и графика зависимости математического времени пребывания в системе от интенсивности прибора M(lambda). Подменю Table of results, P(lambda), M(lambda) доступны пользователю только после того, как была промоделирована работа системы, иначе будет выдано текстовое сообщение об ошибке.



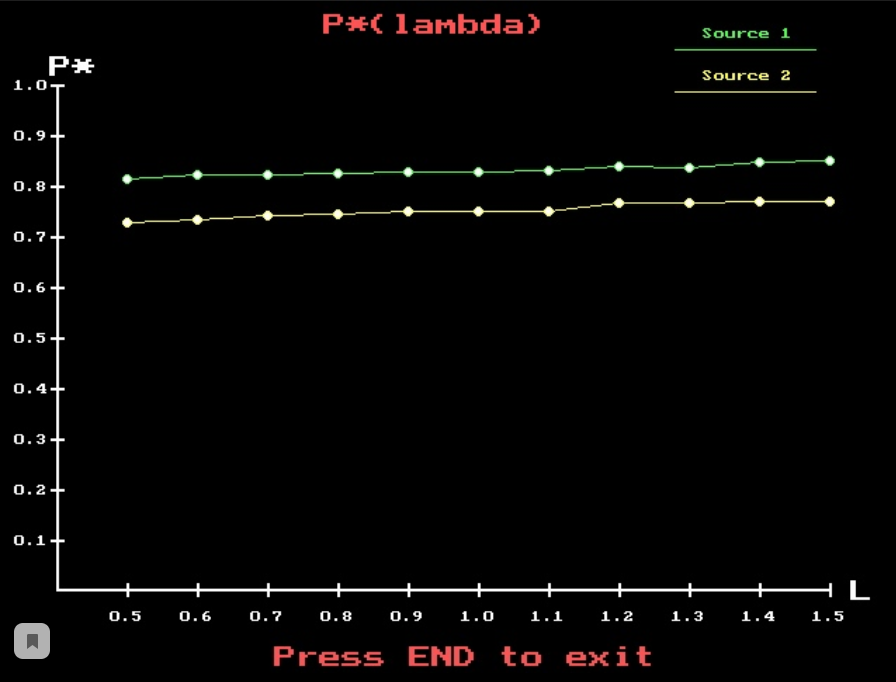
*Рис. 7. Меню Results.*



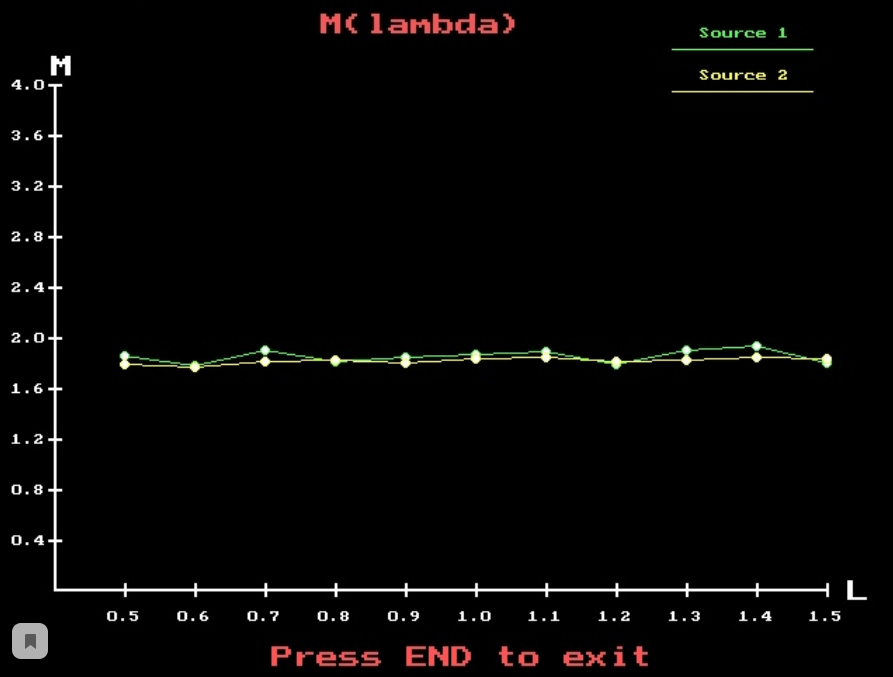
*Рис. 8. Проверка на запуск моделирования.*



*Рис. 9. Подменю Table of results.*



*Рис. 10. Подменю P(lambda).*



*Рис. 11. Подменю M(lambda).*

## Список использованных источников

1. Веренинов И.А. Программирование на языке высокого уровня: учеб. пособие / И. А. Веренинов. – СПб.: Изд-во Политехн. ун-та, 2013. – 212с.
2. Веренинов И.А. Методические указания к курсовому проектированию по курсу «Алгоритмизация и структурное программирование» для студентов высшей школы программной инженерии ИКНТ. – СПб.: 2016. – 24с.

## Приложение 1. Исходный код программы SMO.PAS

{Main file of project}

program SMO;

uses Menu;

var

MainMenu : p\_MainMenu;

BEGIN

randomize;

new(MainMenu, Init);

MainMenu^.launchMainMenu;

END.

## Приложение 2. Исходный код модуля BOOS.PAS

{Implementation of BOOS at the project}

Unit Boos;

Interface

uses Source1, Source2, Writer, Statist, Bas, Device, graph, crt;

type

p\_Boos = ^BoosObj;

BoosObj = object

private

{instance of class Source1}

firstSource : p\_Source1;

{instance of class Source2}

secondSource : p\_Source2;

{instance of class Device}

dev : p\_Device;

{instance of class Bas}

basIn : p\_Bas;

{instance of class Statistic}

stat : p\_Statistic;

{instance of class Writer}

writerIn : p\_Writer;

{current iteration}

iteration : longint;

{time of get the request}

tpost : array[1..2] of real;

{long implementation}

kmin : longint;

{const values}

koefStudenta : real;

accuracyOfEstimationOfProbability : real;

public

{initialization the field}

constructor Init(newFirstSource : p\_Source1; newSecondSource : p\_Source2;

newDev : p\_Device; newBasIn : p\_Bas;

newStat : p\_Statistic; newWriterIn : p\_Writer;

koefStudenta\_, accuracyOfEstimationOfProbability\_ : real);

{reset field of class}

procedure resetField;

{launch event}

procedure launchEvent(indGraph : boolean);

{begin new iteration}

procedure beginNewIteration(indGraph : boolean);

{start simulation}

procedure startSimulation(indGraph : boolean);

{update kmin}

procedure updateKmin(it : longint);

{setter and getter for kmin}

procedure setKmin(kmin\_ : longint);

function getKmin : longint;

function getNewKmin(it : longint) : longint;

end;

Implementation

constructor BoosObj.Init;

var

i : longint;

begin

firstSource := newFirstSource;

secondSource := newSecondSource;

dev := newDev;

basIn := newBasIn;

stat := newStat;

writerIn := newWriterIn;

koefStudenta := koefStudenta\_;

accuracyOfEstimationOfProbability := accuracyOfEstimationOfProbability\_;

iteration := 0;

kmin := 1000;

end;

procedure BoosObj.resetField;

begin

tpost[1] := 0;

tpost[1] := firstSource^.tay(tpost[1]);

tpost[2] := 0;

tpost[2] := secondSource^.tay(tpost[2]);

dev^.clearDevice;

end;

procedure BoosObj.launchEvent;

var

nmin : longint;

numberOfSource : longint;

finishedTime: real;

time: string;

timeR, newObjectTime: real;

sNStr, tStr, iStr, help: string;

prevKol, prevKotk : longint;

begin

if tpost[1] < tpost[2] then

nmin := 1

else

nmin := 2;

str(iteration, iStr);

if (dev^.getWorking = true) and (dev^.getTosv < tpost[nmin]) then

begin

timeR := dev^.getTosv;

str(timeR:4:3, time);

dev^.finishWorking(numberOfSource, finishedTime, newObjectTime);

if newObjectTime <> -1 then

stat^.newAverageExpectionTime(numberOfSource, iteration, timeR - newObjectTime);

stat^.incrementKobr(numberOfSource, iteration);

str(numberOfSource, sNStr);

str(finishedTime:4:3, tStr);

{writerIn^.writeMSG(concat('iteration : ', iStr,' time : ', time,' device finished request ', sNStr, ' from ', tStr));}

exit;

end;

str(nmin, sNStr);

str(tpost[nmin]:4:3, tStr);

if ((nmin = 2) and (indGraph) and (iteration = 1)) then

begin

prevKol := stat^.getKol(2, iteration);

prevKotk := stat^.getKotk(2, iteration);

end;

case basIn^.createNewRequest(nmin, tpost[nmin], iteration) of

1:

begin

{writerIn^.writeMSG(concat('iteration : ', iStr, ' time :', tStr,

' : request from source : ', sNStr, ' was denied'));}

stat^.incrementKotk(nmin, iteration);

end;

2:

begin

{str(stat^.getQuantityRequestInBuffer(nmin, iteration), help);

writerIn^.writeMSG(concat('now quantity for source : ', sNStr, ' is ', help));}

stat^.incrementQuantityRequestInBuffer(nmin, iteration);

{writerIn^.writeMSG(concat('iteration : ', iStr, ' time :',

tStr, ' : request from source : ', sNStr, ' was added'));}

end;

3:

begin

{writerIn^.writeMSG(concat('iteration : ', iStr, ' time : ',

tStr, ' : request from source : ', sNStr,' was passed to device'));}

stat^.newAverageExpectionTime(nmin, iteration, 0);

stat^.incrementRequestGoingInstantlyToDevice(nmin, iteration);

end;

end;

case nmin of

1:

tpost[1] := firstSource^.tay(tpost[1]);

2:

tpost[2] := secondSource^.tay(tpost[2]);

end;

stat^.incrementKol(nmin, iteration);

if ((nmin = 2) and (indGraph) and (iteration = 1)) then

begin

setcolor(blue);

if prevKol > 0 then

line(42 + round((prevKol / (kmin \* 7)) \* 550), 418 - round((prevKotk / prevKol) \* 360),

42 + round((stat^.getKol(2, 1) / (kmin \* 7)) \* 550),

418 - round((stat^.getKotk(2, 1) / stat^.getKol(2, 1)) \* 360))

else

line(42, 418, 42 + round((stat^.getKol(2, 1) / (kmin \* 7)) \* 550),

418 - round((stat^.getKotk(2, 1) / stat^.getKol(2, 1)) \* 360));

end;

end;

procedure BoosObj.beginNewIteration;

var

minKol: longint;

kol: string;

begin

repeat

launchEvent(indGraph);

if (stat^.getKol(1, iteration) < stat^.getKol(2, iteration)) then

minKol := stat^.getKol(1, iteration)

else

minKol := stat^.getKol(2, iteration);

until (minKol >= kmin);

end;

procedure BoosObj.updateKmin;

begin

kmin := round((koefStudenta \* (1 - stat^.getProbabilityOfReject(1, it))) /

(accuracyOfEstimationOfProbability \* accuracyOfEstimationOfProbability

\* stat^.getProbabilityOfReject(1, it)));

end;

procedure BoosObj.startSimulation;

begin

stat^.resetField;

while (firstSource^.getFinallyLambda - firstSource^.getLambda > -0.005) do

begin

inc(iteration);

writerIn^.preIterationMSG(stat, iteration, firstSource^.getLambda, kmin);

resetField;

beginNewIteration(indGraph);

writerIn^.IterationMSG(stat, iteration);

firstSource^.updateLambda;

end;

iteration := 0;

firstSource^.setLambda(0.5);

firstSource^.setFinallyLambda(1.5);

end;

procedure BoosObj.setKmin;

begin

kmin := kmin\_;

end;

function BoosObj.getKmin;

begin

getKmin := kmin;

end;

function BoosObj.getNewKmin;

begin

getNewKmin := round((koefStudenta \* (1 - stat^.getProbabilityOfReject(1, it))) /

(accuracyOfEstimationOfProbability \* accuracyOfEstimationOfProbability

\* stat^.getProbabilityOfReject(1, it)));;

end;

BEGIN

END.

## Приложение 3. Исходный код модуля BAS.PAS

{Implementation the units of analysis of the status}

Unit Bas;

Interface

uses Buffer, Device;

type

p\_Bas = ^BasObj;

BasObj = object

private

{instance of class Device}

dev : p\_Device;

{instance of class Buffer}

buff : p\_Buffer;

public

{initialization the field}

constructor Init(newDevice : p\_Device; newBuffer : p\_Buffer);

{method which create new request with his the index

if return 1 - buffer is busy and request is denied

if return 2 - request is going in buffer

if return 3 - request is going instantly to device }

function createNewRequest(numberOfSource : longint; timeOfSource : real;

iteration : longint) : longint;

end;

Implementation

constructor BasObj.Init;

begin

dev := newDevice;

buff := newBuffer;

end;

function BasObj.createNewRequest;

begin

if dev^.getWorking = false then

begin

dev^.startWorking(numberOfSource, timeOfSource);

createNewRequest := 3;

end

else if buff^.Full = false then

begin

buff^.setRequest(numberOfSource, timeOfSource);

createNewRequest := 2;

end

else

createNewRequest := 1;

end;

BEGIN

END.

## Приложение 4. Исходный код модуля BUFFER.PAS

{Implementation of Buffer with capacity = 2 and without priority}

Unit Buffer;

Interface

type

p\_Buffer = ^BufferObj;

BufferObj = object

private

{capacity of Buffer}

capacity : longint;

{length Of Buffer}

indBuf : longint;

{massive number and time of Source in Buffer according}

masBufN : array[1..2] of longint;

masBufT : array[1..2] of real;

public

{initialization the field}

constructor Init(capacity\_ : longint);

{return quantity elemnts in Buffer}

function getQuantity : longint;

{check for emptiness and fullness of the buffer according}

function Empty : boolean;

function Full : boolean;

{setter and getter of Request for Buffer according}

procedure setRequest(bufN : longint; bufT : real);

procedure getRequest(var bufN : longint; var bufT : real);

{clear all buffer}

procedure clearBuffer;

end;

Implementation

constructor BufferObj.Init;

begin

capacity := capacity\_;

indBuf := 0;

end;

function BufferObj.Empty;

begin

Empty := (indBuf = 0);

end;

function BufferObj.Full;

begin

Full := (indBuf = CAPACITY);

end;

procedure BufferObj.setRequest;

begin

inc(indBuf);

masBufN[indBuf] := bufN;

masBufT[indBuf] := bufT;

end;

procedure BufferObj.getRequest ;

begin

dec(indBuf);

bufN := masBufN[1];

bufT := masBufT[1];

masBufN[1] := masBufN[2];

masBufT[1] := masBufT[2];

end;

procedure BufferObj.clearBuffer;

var

i : longint;

begin

for i := 1 to capacity do

begin

masBufN[i] := 0;

masBufT[i] := 0;

end;

indBuf := 0;

end;

function BufferObj.getQuantity;

begin

getQuantity := indBuf;

end;

BEGIN

END.

## Приложение 5. Исходный код модуля DEVICE.PAS

{Implementation of working our device}

Unit Device;

Interface

uses Buffer;

type

p\_Device = ^DeviceObj;

DeviceObj = object

private

{time of start service in the device}

tho : real;

{common time of service}

tayob : real;

{time when device becomes a free}

tosv : real;

{numver of source in device}

numberOfSource : longint;

{check the device is working or not}

working : boolean;

{const lambda of processing}

lambdaObr : real;

{instance of class Buffer}

buff : p\_Buffer;

public

{initialization the field}

constructor Init(newBuff : p\_Buffer; lambdaObr\_ : real);

{getter for field working}

function getWorking : boolean;

{getter for field tho}

function getTho : real;

{getter and maker for field tayob}

function getTayob : real;

{make tayob}

procedure makeTayob;

{getter for field tosv}

function getTosv : real;

{start working}

procedure startWorking(newNumberOfSource : longint; newTho : real);

{finish working}

procedure finishWorking(var finishNumberOfSource : longint;

var finishTime : real;

var newTimeOfNewRequest : real);

{clear the device}

procedure clearDevice;

end;

Implementation

constructor DeviceObj.Init;

begin

buff := newBuff;

tho := 0;

tayob := 0;

tosv := 0;

numberOfSource := 0;

lambdaObr := lambdaObr\_ {2};

working := false;

end;

function DeviceObj.getWorking;

begin

getWorking := working;

end;

function DeviceObj.getTho;

begin

getTho := tho;

end;

function DeviceObj.getTayob;

begin

getTayob := tayob;

end;

procedure DeviceObj.makeTayob;

begin

tayob := -(ln(random) / lambdaObr );

end;

function DeviceObj.getTosv;

begin

getTosv := tosv;

end;

procedure DeviceObj.startWorking;

begin

numberOfSource := newNumberOfSource;

tho := newTho;

working := true;

makeTayob;

if tosv > newTho then

tosv := tosv + tayob

else

tosv := newTho + tayob;

end;

procedure DeviceObj.finishWorking;

var

startNewNumberOfSource : longint;

startNewTime : real;

begin

finishNumberOfSource := numberOfSource;

finishTime := tho;

working := false;

if buff^.Empty = true then

newTimeOfNewRequest := -1

else begin

buff^.getRequest(startNewNumberOfSource, startNewTime);

newTimeOfNewRequest := startNewTime;

startWorking(startNewNumberOfSource, startNewTime);

end;

end;

procedure DeviceObj.clearDevice;

begin

tho := 0;

numberOfSource := 0;

working := false;

tosv := 0;

buff^.clearBuffer;

end;

BEGIN

END.

## Приложение 6. Исходный код модуля WRITER.PAS

{Implementation for writer}

Unit Writer;

Interface

uses Statist;

type

p\_Writer = ^WriterObj;

WriterObj = object

private

{name for file text}

output: text;

{path to our file}

path : string;

public

{initialization the field}

constructor Init;

{closing file}

destructor Done;

{send message}

procedure writeMSG (msg: string);

{send message about preitertion}

procedure preIterationMSG (stat : p\_Statistic; iteration: longint; lambda: real; kmin : longint);

{send message about iteration}

procedure IterationMSG (stat : p\_Statistic; iteration: longint);

{send message about average quantity of request in Buffer}

procedure averageQuantityOfRequestInBufferMSG(stat : p\_Statistic);

{reset file}

procedure resetFile;

end;

Implementation

constructor WriterObj.Init;

begin

path := 'output.txt';

assign(output, path);

rewrite(output);

end;

destructor WriterObj.Done;

begin

close(output);

end;

procedure WriterObj.writeMSG;

begin

writeln(output, msg);

end;

procedure WriterObj.preIterationMSG;

var

help : string;

begin

writeMSG(' ');

str(iteration, help);

writeMSG(concat('Beginning iteration ', help));

str((round (lambda \* 10)) / 10:0:2, help);

writeMSG(concat('lambda = ', help));

str(kmin, help);

writeMSG(concat('kmin = ', help));

writeMSG(' ');

writeMSG('\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-');

end;

procedure WriterObj.IterationMSG;

var

help : string;

begin

writeMSG('#################################################################################');

writeMSG(' ');

str(stat^.getKol(1, iteration), help);

writeMSG(concat('kol for first source = ', help));

str(stat^.getKobr(1, iteration), help);

writeMSG(concat('kobr for first source = ', help));

str(stat^.getKotk(1, iteration), help);

writeMSG(concat('kotk for first source = ', help));

str(stat^.getKol(2, iteration), help);

writeMSG(concat('kol for second source = ', help));

str(stat^.getKobr(2, iteration), help);

writeMSG(concat('kobr for second source = ', help));

str(stat^.getKotk(2, iteration), help);

writeMSG(concat('kotk for second source = ', help));

stat^.calculateProbabilityOfReject(iteration);

str(stat^.getProbabilityOfReject(1, iteration):0:2, help);

writeMSG(concat('P for first source = ', help));

str(stat^.getProbabilityOfReject(2, iteration):0:2, help);

writeMSG(concat('P for second source = ', help));

str(stat^.getAverageExpectionTime(1, iteration):4:3, help);

writeMSG(concat('M for first source = ', help));

str(stat^.getAverageExpectionTime(2, iteration):4:3, help);

writeMSG(concat('M for second source = ', help));

str(stat^.getCommonAvExpTime(iteration):4:3, help);

writeMSG(concat('Common M = ', help));

str(stat^.getQuantityRequestInBuffer(1, iteration), help);

writeMSG(concat('Quantity of request in buffer for first source = ', help));

str(stat^.getQuantityRequestInBuffer(2, iteration), help);

writeMSG(concat('Quantity of request in buffer for second source = ', help));

writeMSG(' ');

writeMSG('#################################################################################');

end;

procedure WriterObj.averageQuantityOfRequestInBufferMSG;

var

help : string;

begin

writeMSG(' ');

str(stat^.getAverageQuantityRequestInBuffer(1), help);

writeMSG(concat('Average quantity of request in Buffer for first source = ', help));

str(stat^.getAverageQuantityRequestInBuffer(2), help);

writeMSG(concat('Average quantity of request in Buffer for second source = ', help));

end;

procedure WriterObj.resetFile;

begin

rewrite(output);

end;

BEGIN

END.

## Приложение 7. Исходный код модуля STATIST.PAS

{Implementation for class Statistic which write in file our statistics}

Unit Statist;

Interface

const \_NUMBER\_OF\_ITERATION\_ = 11;

type

p\_Statistic = ^StatisticObj;

StatisticObj = object

private

{quantity request of source for everyone \_NUMBER\_OF\_ITERATION\_}

kol : array[1..2,1..\_NUMBER\_OF\_ITERATION\_] of longint;

{quantity requst of source have processed for everyone \_NUMBER\_OF\_ITERATION\_}

kobr : array[1..2,1..\_NUMBER\_OF\_ITERATION\_] of longint;

{quantity request of source have got reject for everyone \_NUMBER\_OF\_ITERATION\_}

kotk : array[1..2,1..\_NUMBER\_OF\_ITERATION\_] of longint;

{probability of reject}

probabilityOfReject : array[1..2,0..\_NUMBER\_OF\_ITERATION\_] of real;

{common probability Of Reject}

commonProbabilityOfReject : array[0..\_NUMBER\_OF\_ITERATION\_] of real;

{counter quantity of request sent in buffer}

quantityOfRequestInBuffer : array[1..2,1..\_NUMBER\_OF\_ITERATION\_] of longint;

{average processing time}

averageQuantityRequestInBuffer : array[1..2] of longint;

{average expection time}

averageExpectionTime : array[1..2,1..\_NUMBER\_OF\_ITERATION\_] of real;

{common average expection time}

commonAverageExpectionTime : array[1..\_NUMBER\_OF\_ITERATION\_] of real;

{request Reached Device}

requestReachedDevice : array[1..2,1..\_NUMBER\_OF\_ITERATION\_] of longint;

{request going in device instantly}

requestGoingInstantlyToDevice : array[1..2,1..\_NUMBER\_OF\_ITERATION\_] of longint;

public

{initialization our the massive for statistics}

constructor Init(probabilityOfReject\_ : real);

{increment quantity request of source}

procedure incrementKol(i, numberOfIteration: longint);

{increment quantity request of source have processed}

procedure incrementKobr(i, numberOfIteration: longint);

{increment quantity request of source have got reject}

procedure incrementKotk(i, numberOfIteration: longint);

{calculate probability of reject}

procedure calculateProbabilityOfReject(numberOfIteration : longint);

{change average processing time after new request in buffer}

procedure calculateAverageQuantityRequestInBuffer(i : longint);

{change average expection time after new request in buffer}

procedure newAverageExpectionTime(i, iteration : longint; newTime : real);

{calculate common average expection time}

procedure calculateAverageExpectionTime(numberOfIteration : longint);

{increment quantityOfRequestInBuffer}

procedure incrementQuantityRequestInBuffer(i, iteration : longint);

{increment quantityOfRequestInBuffer}

procedure incrementRequestGoingInstantlyToDevice(i, iteration : longint);

{reset all field}

procedure resetField;

{getters for all field of class}

function getKol(i, numberOfIteration : longint) : longint;

function getKobr(i, numberOfIteration : longint) : longint;

function getKotk(i, numberOfIteration : longint) : longint;

function getProbabilityOfReject(i, numberOfIteration : longint) : real;

function getCommonProbabilityOfReject(numberOfIteration : longint) : real;

function getAverageQuantityRequestInBuffer(i : longint) : longint;

function getAverageExpectionTime(i, numberOfIteration : longint) : real;

function getCommonAvExpTime(numberOfIteration : longint) : real;

function getQuantityRequestInBuffer(i, numberOfIteration : longint) : longint;

function getRequestGoingInstantlyToDevice(i, numberOfIteration : longint) : longint;

end;

Implementation

constructor StatisticObj.Init;

begin

probabilityOfReject[1,0] := probabilityOfReject\_;

probabilityOfReject[2,0] := probabilityOfReject\_;

commonProbabilityOfReject[0] := probabilityOfReject\_;

end;

procedure StatisticObj.resetField;

var

i, j : longint;

begin

for i := 1 to 2 do

begin

for j := 1 to \_NUMBER\_OF\_ITERATION\_ do

begin

kol[i,j] := 0;

kobr[i,j] := 0;

kotk[i,j] := 0;

averageQuantityRequestInBuffer[i] := 0;

averageExpectionTime[i,j] := 0;

quantityOfRequestInBuffer[i,j] := 0;

probabilityOfReject[i,j] := 0;

commonProbabilityOfReject[j] := 0;

commonAverageExpectionTime[j] := 0;

requestReachedDevice[i,j] := 0;

requestGoingInstantlyToDevice[i,j] := 0;

end;

end;

end;

procedure StatisticObj.incrementKol;

begin

inc(kol[i, numberOfIteration]);

end;

procedure StatisticObj.incrementKobr;

begin

inc(kobr[i, numberOfIteration]);

end;

procedure StatisticObj.incrementKotk;

begin

inc(kotk[i, numberOfIteration]);

end;

procedure StatisticObj.calculateProbabilityOfReject;

var

i : longint;

begin

for i := 1 to 2 do

probabilityOfReject[i, numberOfIteration] :=

kotk[i, numberOfIteration] / kol[i, numberOfIteration];

commonProbabilityOfReject[numberOfIteration] :=

(probabilityOfReject[1, numberOfIteration] + probabilityOfReject[2, numberOfIteration]) / 2;

end;

procedure StatisticObj.calculateAverageQuantityRequestInBuffer;

var

temp : longint;

j : longint;

begin

temp := 0;

for j := 1 to \_NUMBER\_OF\_ITERATION\_ do

temp :=temp + quantityOfRequestInBuffer[i, j];

averageQuantityRequestInBuffer[i] := round(temp / \_NUMBER\_OF\_ITERATION\_);

end;

procedure StatisticObj.newAverageExpectionTime;

begin

averageExpectionTime[i, iteration] := (averageExpectionTime[i, iteration] \*

requestReachedDevice[i, iteration] + newTime) / (requestReachedDevice[i, iteration] + 1);

inc(requestReachedDevice[i, iteration]);

end;

procedure StatisticObj.calculateAverageExpectionTime;

begin

commonAverageExpectionTime[numberOfIteration] := (averageExpectionTime[1, numberOfIteration] +

averageExpectionTime[2, numberOfIteration]) / 2;

end;

procedure StatisticObj.incrementQuantityRequestInBuffer;

begin

inc(quantityOfRequestInBuffer[i, iteration]);

end;

procedure StatisticObj.incrementRequestGoingInstantlyToDevice;

begin

inc(requestGoingInstantlyToDevice[i, iteration]);

end;

function StatisticObj.getKol;

begin

getKol := kol[i, numberOfIteration];

end;

function StatisticObj.getKobr;

begin

getKobr := kobr[i, numberOfIteration];

end;

function StatisticObj.getKotk;

begin

getKotk := kotk[i, numberOfIteration];

end;

function StatisticObj.getProbabilityOfReject;

begin

getProbabilityOfReject := kotk[i, numberOfIteration] / kol[i, numberOfIteration];

end;

function StatisticObj.getAverageExpectionTime;

begin

getAverageExpectionTime := averageExpectionTime[i, numberOfIteration];

end;

function StatisticObj.getQuantityRequestInBuffer;

begin

getQuantityRequestInBuffer := quantityOfRequestInBuffer[i, numberOfIteration];

end;

function StatisticObj.getAverageQuantityRequestInBuffer;

begin

calculateAverageQuantityRequestInBuffer(i);

getAverageQuantityRequestInBuffer := averageQuantityRequestInBuffer[i];

end;

function StatisticObj.getCommonAvExpTime;

begin

calculateAverageExpectionTime(numberOfIteration);

getCommonAvExpTime := commonAverageExpectionTime[numberOfIteration];

end;

function StatisticObj.getCommonProbabilityOfReject(numberOfIteration : longint) : real;

begin

getCommonProbabilityOfReject := commonProbabilityOfReject[numberOfIteration];

end;

function StatisticObj.getRequestGoingInstantlyToDevice;

begin

getRequestGoingInstantlyToDevice := requestGoingInstantlyToDevice[i, numberOfIteration];

end;

BEGIN

END.

## Приложение 8. Исходный код модуля SOURCE1.PAS

{Imlementation of Source1}

Unit Source1;

Interface

type

p\_Source1 = ^Source1Obj;

Source1Obj = object

private

{data for 1 source}

lambda : real;

deltaLambda : real;

finallyLambda : real;

public

{initialization the field}

constructor Init;

{random value of tay}

function tay(tpost : real) : real;

{getter for lambda}

function getLambda : real;

{getter for delta lambda}

function getDeltaLambda : real;

{getter for finally Lambda}

function getFinallyLambda : real;

{change lambda}

procedure updateLambda;

{setters for field}

procedure setLambda(lambda\_ : real);

procedure setDeltaLambda(deltaLambda\_ : real);

procedure setFinallyLambda(finallyLambda\_ : real);

end;

Implementation

constructor Source1Obj.Init;

begin

lambda := 0.5;

deltaLambda := 0.1;

finallyLambda := 1.5;

end;

function Source1Obj.tay;

begin

tay := tpost - (ln(random) / lambda );

end;

function Source1Obj.getLambda;

begin

getLambda := lambda;

end;

function Source1Obj.getDeltaLambda;

begin

getDeltaLambda := deltaLambda;

end;

function Source1Obj.getFinallyLambda;

begin

getFinallyLambda := finallyLambda;

end;

procedure Source1Obj.updateLambda;

begin

lambda := lambda + deltaLambda;

end;

procedure Source1Obj.setLambda;

begin

lambda := lambda\_;

end;

procedure Source1Obj.setDeltaLambda;

begin

deltaLambda := deltaLambda\_;

end;

procedure Source1Obj.setFinallyLambda;

begin

finallyLambda := finallyLambda\_;

end;

BEGIN

END.

## Приложение 9. Исходный код модуля SOURCE2.PAS

{Implementation for Source2}

Unit Source2;

Interface

type

p\_Source2 = ^Source2Obj;

Source2Obj = object

private

{constants for source2}

tay1, tay2 : real;

public

{initialization the field}

constructor Init;

{getters for tay1 and tay2}

function getTay1 : real;

function getTay2 : real;

{random value of tay}

function tay(tpost : real) : real;

{setters for field}

procedure setTay1(tay1\_ : real);

procedure setTay2(tay2\_ : real);

end;

Implementation

constructor Source2Obj.Init;

begin

tay1 := 0.1;

tay2 := 0.5;

end;

function Source2Obj.getTay1;

begin

getTay1 := tay1;

end;

function Source2Obj.getTay2;

begin

getTay2 := tay2;

end;

function Source2Obj.tay;

begin

tay := tpost + tay1 + (tay2 - tay1) \* random;

end;

procedure Source2Obj.setTay1;

begin

tay1 := tay1\_;

end;

procedure Source2Obj.setTay2;

begin

tay2 := tay2\_;

end;

BEGIN

END.

## Приложение 10. Исходный код модуля MENU.PAS

{Draw menu}

Unit Menu;

Interface

{program Graphic;}

uses graph, crt, SUPM, Bas, Buffer, Device, Boos, Statist,

Writer, Source1, Source2, MenuMod, MenuRes;

type

p\_MainMenu = ^MainMenuObj;

MainMenuObj = object

private

{variable for Initgraph}

gh, gm : integer;

{instance of classes}

boosIn : p\_Boos;

buff: p\_Buffer;

writerIn : p\_Writer;

firstSource : p\_Source1;

secondSource : p\_Source2;

dev : p\_Device;

basIn : p\_Bas;

stat : p\_Statistic;

SubMenuSup : p\_SMSUP;

SubMenuMod : p\_MenuMod;

SubMenuRes : p\_MenuRes;

public

{initialization our the massive for menu}

constructor Init;

{write a tips}

procedure drawHelp;

{draw button of setting params}

procedure drawButtonOfSettingParams(color, fraim : longint);

{draw button of modeling}

procedure drawButtonOfModeling(color, fraim : longint);

{draw button of results}

procedure drawButtonOfResults(color, fraim : longint);

{launch main menu}

procedure launchMainMenu;

end;

Implementation

constructor MainMenuObj.Init;

begin

gm := 0;

gh := 0;

new (firstSource, Init);

new (secondSource, Init);

new (stat, Init(0.9));

new (buff, Init(2));

new (dev, Init(buff, 1));

new (basIn, Init(dev, buff));

new (writerIn, Init);

new (boosIn, Init(firstSource, secondSource, dev, basIn, stat, writerIn, 2.71, 0.1));

new(SubMenuSup, Init(boosIn));

new(SubMenuMod, Init(boosIn, stat));

new(SubMenuRes, Init(stat, firstSource, boosIn));

end;

procedure MainMenuObj.drawHelp;

begin

setcolor(blue);

SetTextStyle(DefaultFont, HorizDir, 1);

OutTextXY(20, 420, 'Press right, left, up or down arrow to switch between the button.');

OutTextXY(20, 440, 'Press ENTER if you want choose this button. Press END if you want exit.');

OutTextXY(20, 460, 'SMO done by Maschenko Bogdan, group 13534/4, 2019 year');

end;

procedure MainMenuObj.drawButtonOfSettingParams;

begin

setcolor(color);

setfillstyle(1, color);

Bar(30, 40, 180, 90);

setcolor(fraim);

Rectangle(30, 40, 180, 90);

setcolor(black);

SetTextStyle(DefaultFont, HorizDir, 1);

OutTextXY(38, 60, 'Set up parameters');

end;

procedure MainMenuObj.drawButtonOfModeling;

begin

setcolor(color);

setfillstyle(1, color);

Bar(230, 40, 380, 90);

setcolor(fraim);

Rectangle(230, 40, 380, 90);

setcolor(black);

SetTextStyle(DefaultFont, HorizDir, 1);

OutTextXY(275, 60, 'Modeling');

end;

procedure MainMenuObj.drawButtonOfResults;

begin

setcolor(color);

setfillstyle(1, color);

Bar(430, 40, 580, 90);

setcolor(fraim);

Rectangle(430, 40, 580, 90);

setcolor(black);

SetTextStyle(DefaultFont, HorizDir, 1);

OutTextXY(480, 60, 'Results');

end;

procedure MainMenuObj.launchMainMenu;

var

button : char;

ind : longint;

begin

Initgraph(gm, gh, '');

drawButtonOfSettingParams(7, 15);

drawButtonOfModeling(4, 15);

drawButtonOfResults(4, 15);

drawHelp;

ind := 1;

button := readkey;

repeat

if ((button = #77) and (ind < 3)) then

inc(ind)

else if ((button = #75) and (ind > 1)) then

dec(ind);

case ind of

1:

begin

drawButtonOfSettingParams(7, 15);

drawButtonOfModeling(4, 15);

drawButtonOfResults(4, 15);

drawHelp;

button := readkey;

if button = #13 then

begin

SubMenuSup^.launchSubmenuOfSUP;

button := readkey;

end;

end;

2:

begin

drawButtonOfSettingParams(4, 15);

drawButtonOfModeling(7,15);

drawButtonOfResults(4, 15);

drawHelp;

button := readkey;

if button = #13 then

begin

SubMenuMod^.launchSubmenuMod;

button := readkey;

end;

end;

3:

begin

drawButtonOfSettingParams(4, 15);

drawButtonOfModeling(4, 15);

drawButtonOfResults(7, 15);

drawHelp;

button := readkey;

if button = #13 then

begin

SubMenuRes^.launchSubmenuRes;

button := readkey;

end;

end;

end;

until button = #79;

writerIn^.averageQuantityOfRequestInBufferMSG(stat);

writerIn^.Done;

CloseGraph;

end;

BEGIN

END.

## Приложение 11. Исходный код модуля SUPM.PAS

{Draw submenu of buttonSup Set up parametres}

Unit SupM;

Interface

uses graph, crt, BOOS, SOURCE1, SOURCE2;

type

p\_SMSUP = ^SupObj;

SupObj = object

private

{instance SOURCE1}

firstSource : p\_Source1;

{instance SOURCE2}

secondSource : p\_Source2;

{instance Boos}

boosIn : p\_Boos;

{new kmin}

kmin\_ : longint;

{new lambda, delta lambda, finally lambda}

lambda\_, deltaLambda\_, finallyLambda\_ : real;

{new tay1 and tay2}

tay1\_, tay2\_ : real;

{longint and real --> string}

field : string;

public

{initialization the field}

Constructor Init(boosIn\_ : p\_Boos);

{draw buttonSup for start lambda}

procedure drawButtonForStartLambda(color, fraim : longint);

{draw buttonSup for end lambda}

procedure drawButtonForEndLambda(color, fraim : longint);

{draw buttonSup for delta lambda}

procedure drawButtonForDeltaLambda(color, fraim : longint);

{draw buttonSup for kmin}

procedure drawButtonForKmin(color, fraim : longint);

{draw buttonSup for start tay2}

procedure drawButtonForStartTay(color, fraim : longint);

{draw buttonSup for end tay2}

procedure drawButtonForEndTay(color, fraim : longint);

{launch submenu of sup}

procedure launchSubmenuOfSUP;

end;

Implementation

constructor SupObj.Init;

begin

new(firstSource, Init);

new(secondSource, Init);

boosIn := boosIn\_;

end;

procedure SupObj.drawButtonForStartLambda;

begin

setcolor(color);

setfillstyle(1, color);

Bar(30, 100, 180, 140);

setcolor(fraim);

Rectangle(30, 100, 180, 140);

setcolor(black);

SetTextStyle(DefaultFont, HorizDir, 1);

str(firstSource^.getLambda:2:1, field);

OutTextXY(34, 118, concat('Start lambda = ', field));

setcolor(fraim);

line(105, 100, 105, 91);

end;

procedure SupObj.drawButtonForEndLambda;

begin

setcolor(color);

setfillstyle(1, color);

Bar(30, 150, 180, 190);

setcolor(fraim);

Rectangle(30, 150, 180, 190);

setcolor(black);

SetTextStyle(DefaultFont, HorizDir, 1);

str(firstSource^.getFinallyLambda:2:1, field);

OutTextXY(41, 168, concat('End lambda = ', field));

setcolor(fraim);

line(105, 150, 105, 140);

end;

procedure SupObj.drawButtonForDeltaLambda;

begin

setcolor(color);

setfillstyle(1, color);

Bar(30, 200, 180, 240);

setcolor(fraim);

Rectangle(30, 200, 180, 240);

setcolor(black);

SetTextStyle(DefaultFont, HorizDir, 1);

str(firstSource^.getDeltaLambda:2:1, field);

OutTextXY(34, 218, concat('Delta lambda = ', field));

setcolor(fraim);

line(105, 200, 105, 190);

end;

procedure SupObj.drawButtonForKmin;

begin

setcolor(color);

setfillstyle(1, color);

Bar(30, 250, 180, 290);

setcolor(fraim);

Rectangle(30, 250, 180, 290);

setcolor(black);

SetTextStyle(DefaultFont, HorizDir, 1);

str(boosIn^.getKmin, field);

OutTextXY(60, 268, concat('Kmin = ', field));

setcolor(fraim);

line(105, 250, 105, 240);

end;

procedure SupObj.drawButtonForStartTay;

begin

setcolor(color);

setfillstyle(1, color);

Bar(30, 300, 180, 340);

setcolor(fraim);

Rectangle(30, 300, 180, 340);

setcolor(black);

SetTextStyle(DefaultFont, HorizDir, 1);

str(secondSource^.getTay1:2:1, field);

OutTextXY(42, 318, concat('Start tay2 = ', field));

setcolor(fraim);

line(105, 300, 105, 290);

end;

procedure SupObj.drawButtonForEndTay;

begin

setcolor(color);

setfillstyle(1, color);

Bar(30, 350, 180, 390);

setcolor(fraim);

Rectangle(30, 350, 180, 390);

setcolor(black);

SetTextStyle(DefaultFont, HorizDir, 1);

str(secondSource^.getTay2:2:1, field);

OutTextXY(47, 368, concat('End tay2 = ', field));

setcolor(fraim);

line(105, 350, 105, 340);

end;

procedure SupObj.launchSubmenuOfSUP;

var

buttonSup : char;

indSup : longint;

begin

drawButtonForStartLambda(10, 15);

drawButtonForEndLambda(4, 15);

drawButtonForDeltaLambda(4, 15);

drawButtonForKmin(4, 15);

drawButtonForStartTay(4, 15);

drawButtonForEndTay(4, 15);

indSup := 1;

buttonSup := readkey;

repeat

if ((buttonSup = #80) and (indSup < 6)) then

inc(indSup)

else if ((buttonSup = #72) and (indSup > 1)) then

dec(indSup);

case indSup of

1:

begin

drawButtonForStartLambda(10, 15);

drawButtonForEndLambda(4, 15);

drawButtonForDeltaLambda(4, 15);

drawButtonForKmin(4, 15);

drawButtonForStartTay(4, 15);

drawButtonForEndTay(4, 15);

buttonSup := readkey;

if buttonSup = #13 then

begin

read(lambda\_);

setfillstyle(1, black);

Bar(0, 0, 640, 15);

firstSource^.setLambda(lambda\_);

drawButtonForStartLambda(10, 15);

buttonSup := readkey;

end;

end;

2:

begin

drawButtonForStartLambda(4, 15);

drawButtonForEndLambda(10, 15);

drawButtonForDeltaLambda(4, 15);

drawButtonForKmin(4, 15);

drawButtonForStartTay(4, 15);

drawButtonForEndTay(4, 15);

buttonSup := readkey;

if buttonSup = #13 then

begin

read(finallyLambda\_);

setfillstyle(1, black);

Bar(0, 0, 640, 15);

firstSource^.setFinallyLambda(finallyLambda\_);

drawButtonForEndLambda(10, 15);

buttonSup := readkey;

end;

end;

3:

begin

drawButtonForStartLambda(4, 15);

drawButtonForEndLambda(4, 15);

drawButtonForDeltaLambda(10, 15);

drawButtonForKmin(4,15);

drawButtonForStartTay(4, 15);

drawButtonForEndTay(4, 15);

buttonSup := readkey;

if buttonSup = #13 then

begin

read(deltaLambda\_);

setfillstyle(1, black);

Bar(0, 0, 640, 15);

firstSource^.setDeltaLambda(deltaLambda\_);

drawButtonForDeltaLambda(10, 15);

buttonSup := readkey;

end;

end;

4:

begin

drawButtonForStartLambda(4, 15);

drawButtonForEndLambda(4, 15);

drawButtonForDeltaLambda(4, 15);

drawButtonForKmin(10, 15);

drawButtonForStartTay(4, 15);

drawButtonForEndTay(4, 15);

buttonSup := readkey;

if buttonSup = #13 then

begin

read(kmin\_);

setfillstyle(1, black);

Bar(0, 0, 640, 15);

boosIn^.setKmin(kmin\_);

drawButtonForKmin(10, 15);

buttonSup := readkey;

end;

end;

5:

begin

drawButtonForStartLambda(4, 15);

drawButtonForEndLambda(4, 15);

drawButtonForDeltaLambda(4, 15);

drawButtonForKmin(4, 15);

drawButtonForStartTay(10, 15);

drawButtonForEndTay(4, 15);

buttonSup := readkey;

if buttonSup = #13 then

begin

read(tay1\_);

setfillstyle(1, black);

Bar(0, 0, 640, 15);

secondSource^.setTay1(tay1\_);

drawButtonForStartTay(10, 15);

buttonSup := readkey;

end;

end;

6:

begin

drawButtonForStartLambda(4, 15);

drawButtonForEndLambda(4, 15);

drawButtonForDeltaLambda(4, 15);

drawButtonForKmin(4, 15);

drawButtonForStartTay(4, 15);

drawButtonForEndTay(10, 15);

buttonSup := readkey;

if buttonSup = #13 then

begin

read(tay2\_);

setfillstyle(1, black);

Bar(0, 0, 640, 15);

secondSource^.setTay2(tay2\_);

drawButtonForEndTay(10, 15);

buttonSup := readkey;

end;

end;

end;

if buttonSup = #79 then

begin

drawButtonForStartLambda(0, 0);

drawButtonForEndLambda(0, 0);

drawButtonForDeltaLambda(0, 0);

drawButtonForKmin(0, 0);

drawButtonForStartTay(0, 0);

drawButtonForEndTay(0, 0);

end;

until buttonSup = #79;

end;

BEGIN

END.

## Приложение 12. Исходный код модуля MENUMOD.PAS

{Draw submenu for button modeling}

Unit MenuMod;

Interface

uses graph, crt, Statist, Boos;

type

p\_MenuMod = ^MenuModObj;

MenuModObj = object

private

{instance of class Boos and Statist}

stat : p\_Statistic;

boosIn : p\_Boos;

public

{initialization the field}

constructor Init(boosIn\_ : p\_Boos; stat\_ : p\_Statistic);

{draw osi koordinat for graphics}

procedure drawOsKoord;

{draw button modeling with building graphics P(kol)}

procedure drawButtonModelingWithGraphics(color, fraim : longint);

{draw button modeling without building graphics P(kol)}

procedure drawButtonModelingWithoutGraphics(color, fraim : longint);

{write a tips}

procedure drawHelp;

{draw button of setting params}

procedure drawButtonOfSettingParams(color, fraim : longint);

{draw button of modeling}

procedure drawButtonOfModeling(color, fraim : longint);

{draw button of results}

procedure drawButtonOfResults(color, fraim : longint);

{draw black background}

procedure drawBackground;

{launch submenu of button modeling}

procedure launchSubmenuMod;

end;

Implementation

constructor MenuModObj.Init;

begin

boosIn := boosIn\_;

stat := stat\_;

end;

procedure MenuModObj.drawButtonModelingWithGraphics;

begin

setcolor(color);

setfillstyle(1, color);

Bar(230, 100, 380, 190);

setcolor(fraim);

Rectangle(230, 100, 380, 190);

setcolor(black);

SetTextStyle(DefaultFont, HorizDir, 1);

OutTextXY(240, 120, 'Start modeling');

OutTextXY(250, 140, 'with bulding');

OutTextXY(240, 160, 'graphics P(kol)');

setcolor(fraim);

line(305, 100, 305, 91);

end;

procedure MenuModObj.drawButtonModelingWithoutGraphics;

begin

setcolor(color);

setfillstyle(1, color);

Bar(230, 200, 380, 290);

setcolor(fraim);

Rectangle(230, 200, 380, 290);

setcolor(black);

SetTextStyle(DefaultFont, HorizDir, 1);

OutTextXY(240, 220, 'Start modeling');

OutTextXY(245, 240, 'without bulding');

OutTextXY(240, 260, 'graphics P(kol)');

setcolor(fraim);

line(305, 200, 305, 190);

end;

procedure MenuModObj.drawHelp;

begin

setcolor(blue);

SetTextStyle(DefaultFont, HorizDir, 1);

OutTextXY(20, 420, 'Press right, left, up or down arrow to switch between the button.');

OutTextXY(20, 440, 'Press ENTER if you want choose this button. Press END if you want exit.');

OutTextXY(20, 460, 'SMO done by Maschenko Bogdan, group 13534/4, 2019 year');

end;

procedure MenuModObj.drawButtonOfSettingParams;

begin

setcolor(color);

setfillstyle(1, color);

Bar(30, 40, 180, 90);

setcolor(fraim);

Rectangle(30, 40, 180, 90);

setcolor(black);

SetTextStyle(DefaultFont, HorizDir, 1);

OutTextXY(38, 60, 'Set up parameters');

end;

procedure MenuModObj.drawButtonOfModeling;

begin

setcolor(color);

setfillstyle(1, color);

Bar(230, 40, 380, 90);

setcolor(fraim);

Rectangle(230, 40, 380, 90);

setcolor(black);

SetTextStyle(DefaultFont, HorizDir, 1);

OutTextXY(275, 60, 'Modeling');

end;

procedure MenuModObj.drawButtonOfResults;

begin

setcolor(color);

setfillstyle(1, color);

Bar(430, 40, 580, 90);

setcolor(fraim);

Rectangle(430, 40, 580, 90);

setcolor(black);

SetTextStyle(DefaultFont, HorizDir, 1);

OutTextXY(480, 60, 'Results');

end;

procedure MenuModObj.drawBackground;

begin

setcolor(black);

setfillstyle(1, black);

Bar(0, 0, 640, 480);

end;

procedure MenuModObj.drawOsKoord;

var

button : char;

i : longint;

help : string[10];

begin

drawBackground;

setcolor(12);

SetTextStyle(DefaultFont, HorizDir, 2);

OutTextXY(80, 10, 'P\*(kol) for 2 Source 1 simulation');

OutTextXY(195, 460, 'Press END to exit');

setfillstyle(1, white);

Bar(40, 60, 41, 420);

Bar(40, 419, 590, 420);

setcolor(15);

SetTextStyle(DefaultFont, HorizDir, 2);

OutTextXY(35, 40, 'P\*');

OutTextXY(605, 413, 'K');

SetTextStyle(DefaultFont, HorizDir, 1);

for i := 1 to 10 do

begin

Bar(36, 60 + (i - 1) \* 36, 45, 61 + (i - 1) \* 36);

str((0.1 \* i):2:1, help);

OutTextXY(10, 57 + (10 - i) \* 36, help);

end;

for i := 1 to 5 do

begin

Bar(40 + 110 \* i, 415, 41 + 110 \* i, 424);

str(round((boosIn^.getKmin / 5) \* 7 \* i), help);

OutTextXY(22 + 110 \* i, 435, help);

end;

end;

procedure MenuModObj.launchSubmenuMod;

var

buttonMod : char;

indMod : longint;

begin

drawButtonModelingWithGraphics(10, 15);

drawButtonModelingWithoutGraphics(4, 15);

indMod := 1;

buttonMod := readkey;

repeat

if ((buttonMod = #80) and (indMod < 2)) then

inc(indMod)

else if ((buttonMod = #72) and (indMod > 1)) then

dec(indMod);

case indMod of

1:

begin

drawButtonModelingWithGraphics(10, 15);

drawButtonModelingWithoutGraphics(4, 15);

buttonMod := readkey;

if buttonMod = #13 then

begin

drawOsKoord;

boosIn^.startSimulation(true);

repeat

buttonMod := readkey;

until buttonMod = #79;

drawBackground;

drawButtonOfSettingParams(4, 15);

drawButtonOfModeling(7,15);

drawButtonOfResults(4, 15);

drawHelp;

drawButtonModelingWithGraphics(10, 15);

drawButtonModelingWithoutGraphics(4, 15);

buttonMod := readkey;

end;

end;

2:

begin

drawButtonModelingWithGraphics(4, 15);

drawButtonModelingWithoutGraphics(10, 15);

buttonMod := readkey;

if buttonMod = #13 then

begin

drawBackground;

setcolor(12);

SetTextStyle(DefaultFont, HorizDir, 3);

OutTextXY(100, 200, 'Simulation started');

boosIn^.startSimulation(false);

delay(25000);

drawBackground;

setcolor(12);

SetTextStyle(DefaultFont, HorizDir, 3);

OutTextXY(100, 170, 'Simulation ends');

OutTextXY(50, 230, 'Press END to continue');

repeat

buttonMod := readkey;

until buttonMod = #79;

drawBackground;

drawButtonOfSettingParams(4, 15);

drawButtonOfModeling(7,15);

drawButtonOfResults(4, 15);

drawHelp;

drawButtonModelingWithGraphics(4, 15);

drawButtonModelingWithoutGraphics(10, 15);

buttonMod := readkey;

end;

end;

end;

if buttonMod = #79 then

begin

drawButtonModelingWithGraphics(0, 0);

drawButtonModelingWithoutGraphics(0, 0);

end;

until buttonMod = #79;

end;

BEGIN

END.

## Приложение 13. Исходный код модуля MENURES.PAS

{Draw submenu for button modeling}

Unit MenuRes;

Interface

uses graph, crt, Statist, Source1, Boos;

type

p\_MenuRes = ^MenuResObj;

MenuResObj = object

private

{instance for class Source1 and Statistic}

stat : p\_Statistic;

firstSource : p\_Source1;

booisIn : p\_Boos;

public

{initialization the field}

constructor Init(stat\_ : p\_Statistic; firstSource\_ : p\_Source1; booisIn\_ : p\_Boos);

{draw lines P}

procedure drawLinesP(color, source : longint);

{draw lines M}

procedure drawLinesM(color, source : longint);

{draw black background}

procedure drawBackground;

{draw table of results}

procedure drawTableOfResults;

{draw graphics P\*}

procedure drawGraphicsP;

{draw graphics M}

procedure drawGraphicsM;

{draw button table of result}

procedure drawButtonTableOfResults(color, fraim : longint);

{draw button P(lambda)}

procedure drawButtonP(color, fraim : longint);

{draw button M(lambda)}

procedure drawButtonM(color, fraim : longint);

{draw warning}

procedure drawWarning;

{write a tips}

procedure drawHelp;

{draw button of setting params}

procedure drawButtonOfSettingParams(color, fraim : longint);

{draw button of modeling}

procedure drawButtonOfModeling(color, fraim : longint);

{draw button of results}

procedure drawButtonOfResults(color, fraim : longint);

{launch submenu of result}

procedure launchSubmenuRes;

end;

Implementation

{initialization the field}

constructor MenuResObj.Init;

begin

stat := stat\_;

firstSource := firstSource\_;

booisIn := booisIn\_;

end;

procedure MenuResObj.drawBackground;

begin

setcolor(black);

setfillstyle(1, black);

Bar(0, 0, 640, 480);

end;

procedure MenuResObj.drawWarning;

var

button : char;

begin

setcolor(12);

SetTextStyle(DefaultFont, HorizDir, 2);

OutTextXY(50, 300, 'You dont launch the modeling');

OutTextXY(50, 350, 'Prease press END to continue');

repeat

button := readkey;

until button = #79;

setfillstyle(1, black);

Bar(45, 290, 400, 400);

end;

procedure MenuResObj.drawTableOfResults;

var

button : char;

iteration : longint;

help : string;

begin

if stat^.getKol(1, 1) = 0 then

begin

drawWarning;

exit;

end;

drawBackground;

setcolor(12);

SetTextStyle(DefaultFont, HorizDir, 2);

OutTextXY(220, 20, 'Table of results');

OutTextXY(210, 450, 'Press END to exit');

setcolor(9);

Rectangle(20, 50, 622, 434);

for iteration:= 1 to 11 do

begin

line(20, 50 + iteration \* 32, 622, 50 + iteration \* 32);

end;

for iteration:= 1 to 6 do

begin

line(20 + iteration \* 86, 50, 20 + iteration \* 86, 435);

end;

SetTextStyle(DefaultFont, HorizDir, 1);

setcolor(15);

OutTextXY(38, 63, 'Lambda');

OutTextXY(140, 63, 'P\* 1');

OutTextXY(223, 63, 'P\* 2');

OutTextXY(312, 63, 'M 1');

OutTextXY(397, 63, 'M 2');

OutTextXY(455, 60, '<N> requst');

OutTextXY(462, 70, 'in 1 buf');

OutTextXY(560, 63, 'Kmin');

for iteration := 1 to 11 do

begin

str((firstSource^.getLambda + (iteration - 1) \*

firstSource^.getDeltaLambda):3:2, help);

OutTextXY(50, 63 + iteration \* 32, help);

str((stat^.getProbabilityOfReject(1, iteration)):4:3, help);

OutTextXY(133, 63 + iteration \* 32, help);

str((stat^.getProbabilityOfReject(2, iteration)):4:3, help);

OutTextXY(218, 63 + iteration \* 32, help);

str(stat^.getAverageExpectionTime(1, iteration):4:3, help);

OutTextXY(303, 63 + iteration \* 32, help);

str(stat^.getAverageExpectionTime(2, iteration):4:3, help);

OutTextXY(387, 63 + iteration \* 32, help);

str(((stat^.getQuantityRequestInBuffer(1, iteration)/

(stat^.getKobr(1, iteration) + stat^.getKobr(2, iteration) -

stat^.getRequestGoingInstantlyToDevice(1, iteration) -

stat^.getRequestGoingInstantlyToDevice(2, iteration)))\*2):3:2, help);

OutTextXY(477, 63 + iteration \* 32, help);

str(booisIn^.getNewKmin(iteration), help);

OutTextXY(565, 63 + iteration \* 32, help);

end;

repeat

button := readkey;

until button = #79;

end;

procedure MenuResObj.drawLinesP;

var

i : longint;

begin

setcolor(color);

for i := 1 to 10 do

begin

line(40 + 50 \* i, 420 - round(stat^.getProbabilityOfReject(source, i) \* 360),

40 + 50 \* (i + 1), 420 - round(stat^.getProbabilityOfReject(source, (i + 1)) \* 360));

FillEllipse(40 + 50 \* i, 420 - round(stat^.getProbabilityOfReject(source, i) \* 360), 3, 3);

FillEllipse(40 + 50 \* (i + 1), 420 - round(stat^.getProbabilityOfReject(source, (i + 1)) \* 360), 3, 3);

end;

end;

procedure MenuResObj.drawGraphicsP;

var

button : char;

iteration, i : longint;

help : string;

begin

if stat^.getKol(1, 1) = 0 then

begin

drawWarning;

exit;

end;

drawBackground;

setcolor(12);

SetTextStyle(DefaultFont, HorizDir, 2);

OutTextXY(230, 10, 'P\*(lambda)');

OutTextXY(195, 460, 'Press END to exit');

setfillstyle(1, white);

Bar(40, 60, 41, 420);

Bar(40, 419, 590, 420);

setcolor(15);

SetTextStyle(DefaultFont, HorizDir, 2);

OutTextXY(35, 40, 'P\*');

OutTextXY(605, 413, 'L');

SetTextStyle(DefaultFont, HorizDir, 1);

for i := 1 to 10 do

begin

Bar(36, 60 + (i - 1) \* 36, 45, 61 + (i - 1) \* 36);

str((0.1 \* i):2:1, help);

OutTextXY(10, 57 + (10 - i) \* 36, help);

end;

for iteration := 1 to 11 do

begin

Bar(40 + 50 \* iteration, 415, 41 + 50 \* iteration, 424);

str((firstSource^.getLambda + firstSource^.getDeltaLambda \*

(iteration - 1)):2:1, help);

OutTextXY(28 + 50 \* iteration, 435, help);

end;

SetTextStyle(DefaultFont, HorizDir, 1);

setcolor(10);

line(480, 35, 580, 35);

OutTextXY(500, 20, 'Source 1');

setcolor(14);

line(480, 65, 580, 65);

OutTextXY(500, 50, 'Source 2');

drawLinesP(10, 1);

drawLinesP(14, 2);

repeat

button := readkey;

until button = #79;

end;

procedure MenuResObj.drawLinesM;

var

i : longint;

begin

setcolor(color);

for i := 1 to 10 do

begin

line(40 + 50 \* i, 420 - round((stat^.getAverageExpectionTime(source, i) / 4) \* 360),

40 + 50 \* (i + 1), 420 - round((stat^.getAverageExpectionTime(source, (i + 1)) / 4) \* 360));

FillEllipse(40 + 50 \* i, 420 - round((stat^.getAverageExpectionTime(source, i) / 4) \* 360), 3, 3);

FillEllipse(40 + 50 \* (i + 1), 420 - round((stat^.getAverageExpectionTime(source, (i + 1)) / 4) \* 360), 3, 3);

end;

end;

procedure MenuResObj.drawGraphicsM;

var

button : char;

iteration, i : longint;

help : string;

begin

if stat^.getKol(1, 1) = 0 then

begin

drawWarning;

exit;

end;

drawBackground;

setcolor(12);

SetTextStyle(DefaultFont, HorizDir, 2);

OutTextXY(230, 10, 'M(lambda)');

OutTextXY(195, 460, 'Press END to exit');

setfillstyle(1, white);

Bar(40, 60, 41, 420);

Bar(40, 419, 590, 420);

setcolor(15);

SetTextStyle(DefaultFont, HorizDir, 2);

OutTextXY(38, 40, 'M');

OutTextXY(605, 413, 'L');

SetTextStyle(DefaultFont, HorizDir, 1);

for i := 1 to 10 do

begin

Bar(36, 60 + (i - 1) \* 36, 45, 61 + (i - 1) \* 36);

str((0.4 \* i):2:1, help);

OutTextXY(10, 57 + (10 - i) \* 36, help);

end;

for iteration := 1 to 11 do

begin

Bar(40 + 50 \* iteration, 415, 41 + 50 \* iteration, 424);

str((firstSource^.getLambda + firstSource^.getDeltaLambda \*

(iteration - 1)):2:1, help);

OutTextXY(28 + 50 \* iteration, 435, help);

end;

SetTextStyle(DefaultFont, HorizDir, 1);

setcolor(10);

line(480, 35, 580, 35);

OutTextXY(500, 20, 'Source 1');

setcolor(14);

line(480, 65, 580, 65);

OutTextXY(500, 50, 'Source 2');

drawLinesM(10, 1);

drawLinesM(14, 2);

repeat

button := readkey;

until button = #79;

end;

procedure MenuResObj.drawButtonTableOfResults;

begin

setcolor(color);

setfillstyle(1, color);

Bar(430, 100, 580, 150);

setcolor(fraim);

Rectangle(430, 100, 580, 150);

setcolor(black);

SetTextStyle(DefaultFont, HorizDir, 1);

OutTextXY(440, 120, 'Table of results');

setcolor(fraim);

line(505, 100, 505, 91);

end;

procedure MenuResObj.drawButtonP;

begin

setcolor(color);

setfillstyle(1, color);

Bar(430, 160, 580, 210);

setcolor(fraim);

Rectangle(430, 160, 580, 210);

setcolor(black);

SetTextStyle(DefaultFont, HorizDir, 1);

OutTextXY(470, 180, 'P(lambda)');

setcolor(fraim);

line(505, 160, 505, 150);

end;

procedure MenuResObj.drawButtonM;

begin

setcolor(color);

setfillstyle(1, color);

Bar(430, 220, 580, 270);

setcolor(fraim);

Rectangle(430, 220, 580, 270);

setcolor(black);

SetTextStyle(DefaultFont, HorizDir, 1);

OutTextXY(470, 240, 'M(lambda)');

setcolor(fraim);

line(505, 220, 505, 210);

end;

procedure MenuResObj.drawHelp;

begin

setcolor(blue);

SetTextStyle(DefaultFont, HorizDir, 1);

OutTextXY(20, 420, 'Press right, left, up or down arrow to switch between the button.');

OutTextXY(20, 440, 'Press ENTER if you want choose this button. Press END if you want exit.');

OutTextXY(20, 460, 'SMO done by Maschenko Bogdan, group 13534/4, 2019 year');

end;

procedure MenuResObj.drawButtonOfSettingParams;

begin

setcolor(color);

setfillstyle(1, color);

Bar(30, 40, 180, 90);

setcolor(fraim);

Rectangle(30, 40, 180, 90);

setcolor(black);

SetTextStyle(DefaultFont, HorizDir, 1);

OutTextXY(38, 60, 'Set up parameters');

end;

procedure MenuResObj.drawButtonOfModeling;

begin

setcolor(color);

setfillstyle(1, color);

Bar(230, 40, 380, 90);

setcolor(fraim);

Rectangle(230, 40, 380, 90);

setcolor(black);

SetTextStyle(DefaultFont, HorizDir, 1);

OutTextXY(275, 60, 'Modeling');

end;

procedure MenuResObj.drawButtonOfResults;

begin

setcolor(color);

setfillstyle(1, color);

Bar(430, 40, 580, 90);

setcolor(fraim);

Rectangle(430, 40, 580, 90);

setcolor(black);

SetTextStyle(DefaultFont, HorizDir, 1);

OutTextXY(480, 60, 'Results');

end;

procedure MenuResObj.launchSubmenuRes;

var

buttonRes : char;

indRes : longint;

begin

drawButtonTableOfResults(10, 15);

drawButtonP(4, 15);

drawButtonM(4, 15);

indRes := 1;

buttonRes := readkey;

repeat

if ((buttonRes = #80) and (indRes < 3)) then

inc(indRes)

else if ((buttonRes = #72) and (indRes > 1)) then

dec(indRes);

case indRes of

1:

begin

drawButtonTableOfResults(10, 15);

drawButtonP(4, 15);

drawButtonM(4, 15);

buttonRes := readkey;

if buttonRes = #13 then

begin

drawTableOfResults;

drawBackground;

drawButtonOfSettingParams(4, 15);

drawButtonOfModeling(4,15);

drawButtonOfResults(7, 15);

drawHelp;

drawButtonTableOfResults(10, 15);

drawButtonP(4, 15);

drawButtonM(4, 15);

buttonRes := readkey;

end;

end;

2:

begin

drawButtonTableOfResults(4, 15);

drawButtonP(10, 15);

drawButtonM(4, 15);

buttonRes := readkey;

if buttonRes = #13 then

begin

drawGraphicsP;

drawBackground;

drawButtonOfSettingParams(4, 15);

drawButtonOfModeling(4,15);

drawButtonOfResults(7, 15);

drawHelp;

drawButtonTableOfResults(4, 15);

drawButtonP(10, 15);

drawButtonM(4, 15);

buttonRes := readkey;

end;

end;

3:

begin

drawButtonTableOfResults(4, 15);

drawButtonP(4, 15);

drawButtonM(10, 15);

buttonRes := readkey;

if buttonRes = #13 then

begin

drawGraphicsM;

drawBackground;

drawButtonOfSettingParams(4, 15);

drawButtonOfModeling(4,15);

drawButtonOfResults(7, 15);

drawHelp;

drawButtonTableOfResults(4, 15);

drawButtonP(4, 15);

drawButtonM(10, 15);

buttonRes := readkey;

end;

end;

end;

if buttonRes = #79 then

begin

drawButtonTableOfResults(0, 0);

drawButtonP(0, 0);

drawButtonM(0, 0);

end;

until buttonRes = #79;

end;

BEGIN

END.