Санкт-Петербургский государственный политехнический университет
Институт компьютерных наук и технологий
Кафедра «Информационные и управляющие системы»

К У Р С О В О Й П Р О Е К Т

МОДЕЛИРОВАНИЕ СИСТЕМЫ, ФОРМАЛИЗОВАННОЙ КАК СИСТЕМА МАССОВОГО ОБСЛУЖИВАНИЯ
по дисциплине «Алгоритмизация и основы программирования»

Выполнил
студент гр.13534/1 Стойкоски Н.С.

Руководитель
доцент, к.т.н. И.А.Веренинов

«_21_» __марта__ 2017г.

Санкт-Петербург
2017

ЗАДАНИЕ
НА ВЫПОЛНЕНИЕ КУРСОВОГО ПРОЕКТА

студенту группы 13534/1 Стойкоски Н.С.
(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) *15 мая 2017 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 ***24*** страниц машинописного текста

5. Перечень графического материала (с указанием обязательных чертежей и плакатов) нет

6. Консультанты Нет

7. Дата получения задания: «_10___» _февраля_____ 2017__ г.

Руководитель _____ *Веренинов И. А.*
(подпись) *(инициалы, фамилия)*

Задание принял к исполнению _____ *Стойкоски Н.С.*
(подпись) *(инициалы, фамилия)*

_____
(дата)

СОДЕРЖАНИЕ

Введение

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

Данная СМО состоит из трёх источников – с простейшим потоками заявок. Параметры источников: $\lambda_1=4$ $\lambda_2=3$, $\lambda_3=3$. Объем буфера – 5. Дисциплина записи в буфер - в порядке поступления. Дисциплина выборки из буфера – бесприоритетная. Поток обслуживания – простейший с параметры: $\lambda_4^* = 1 \le \lambda_4 \le \lambda_4^* = 6$ с $\Delta\lambda_4 = 0.5$. Точность моделирования - δ=20%, P=0.95. Надо найти вероятность отказа $P_{\text{отказа i}} = f_{1i}(\lambda)$ i=1,2,3 Среднее время пребывания в системе $M(T_{\text{преб i}}) = f_{2i}(\lambda_4)$ i=1,2,3 при меняющейся интенсивности потока обслуживания.

1. Инструкция программиста

Данный программный продукт состоит из 6 файлов: CMO.PAS, CMO_FUNC.PAS, CMO_GRAPH.PAS, CMO_INTERFACE.PAS, ABOUT.TXT и USAGE.TXT.

CMO_FUNC.PAS представляет собой одноименный функциональный модуль программы, который содержит описание используемых в нем типов объектов, таких как «буфер» (Buffer), «источник» (Istocnik), «прибор» (Pribor). Методы функционального модуля обеспечивают формирование результатов одной длинной реализации, т.е. зависимость вероятностей отказов и математического ожидания от количества заявок, от всех источников. Эти данные в процессе моделирования сразу же в графическом режиме выводятся на экран, с оцифровкой по осям координат.

CMO_GRAPH.PAS представляет собой графический модуль программы. Он содержит объекты, необходимые для вывода в процессе моделирования зависимостей интересующих параметров от числа заявок – «график» Graphic. При этом графики строятся в режиме online в процессе моделирования. Кроме того, этот модуль обеспечивает вывод результирующих зависимостей необходимых параметров от варьируемой интенсивности λ прибора с помощью объекты «график» Graphic (предок), «гистрограмма» BarChart (потомок) и «таблица» Table.

CMO_INTERFACE.PAS является интерфейсным модулем. Он содержит описание объектов, позволяющих формировать пользовательское меню с выпадающим подменю – MenuList (кнопка меню), MenuBox (кнопка подменю), TextBox (текствое поле) – для ввод данных, Window (окно) и GUI (графический интерфейс пользователя) – обьект который управляет всеми элементами интерфейса. Данный модуль использует файлы ABOUT.TXT и USAGE.TXT для вывода вспомогательный текст в окнах.

В файле CMO_MAIN.PAS представлена головная программа, в которой включается графический режим и запускается GUI из интерфейсного модуля. Головная программа обеспечывает связь между объект графического интерфейса и Modulator – объект, управляющий всей системой моделирование. При этом используются вспомогательные модули CRT, GRAPH, CMO_GRAPH, CMO_INTERFACE, CMO_FUNC. Кроме того, данный модуль обеспечивает фиксацию окончательных результатов моделирования при варьировании основного параметра системы, который изменяется при переходе к другой реализации. Эти результаты записываются в массив записей. В графическом режиме установлен одностраничный режим работы видеоадаптера, в качестве драйвера видеоадаптера выбран EGAVGA.BGI.

Более подробные комментарии приведены в исходном коде программы в приложениях 1, 2, 3, 4, 5.

2. Инструкция оператора

При запуске данной программы создается пользовательское хоризонтальное меню с выпадающим подменю.

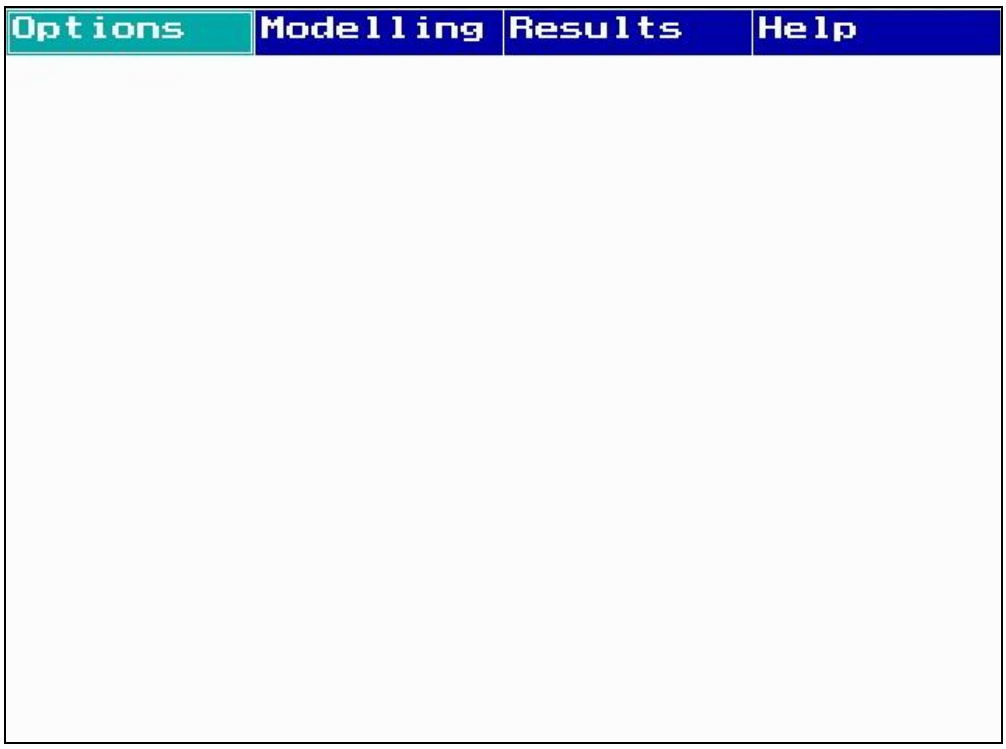


Рис.1 пользовательское меню

Навигация по интерфейсу

перемещение сфокусированного элемента - клавиши со стрелкой влево, вправо, вверх, вниз.
Открыть меню или подменю/ переход на следующее текстовое поле - клавиша Enter
Закрыть окно/ подменю/ и выход из программа - клавиша ESC

Ввод параметры

Интенсивности источников – Options -> Istocniki
Параметры прибора – Options –>Pribor

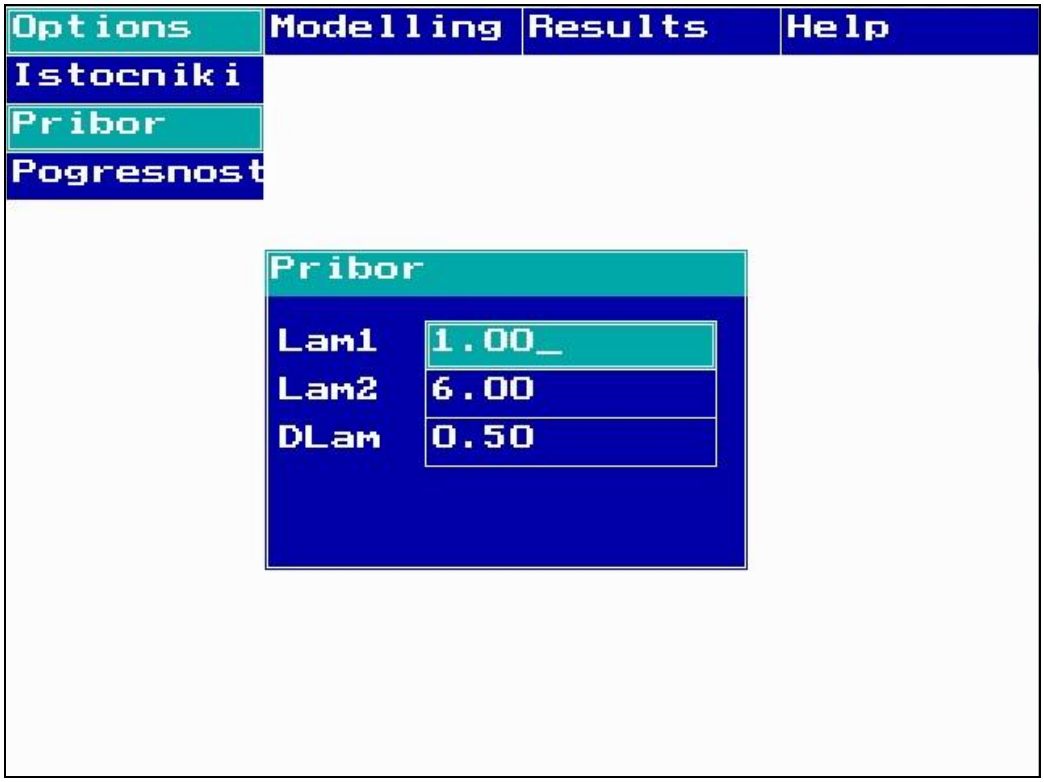


Рис.2 Ввод параметры прибора

Ввод данных в текстовых полях

Ввод значениях – клавиши 0-9 (цифры) и ‘.’ (точка)
Очистить символь перед курсора – клавиша Backspace

Моделирование

Modelling -> Normal Mode (моделирование с задержка для посмотр результатов после каждая реализация)
Modelling -> Fast Mode (моделирование без задержка)
-> Появится окно KMIN, для ввод длина реализации -> начинается моделирование.

Просмотр окончательных результатов

Results -> Graph (окончательные результаты в виде график)
Results -> BarChart (окончательные результаты в вуде гистрограмма)
Results -> Table (окончательныее результаты в виде таблица, в этом режиме можно посмотреть и погрешност моделирование вероятностей отказа

при нажатия клавиша со стрелкой вправо, и веврнутся назад при нажатия клавиша со стрелкой влево).

Дополнительная информация/ помощ

Help -> Usage, Help -> About

Примеры результатов

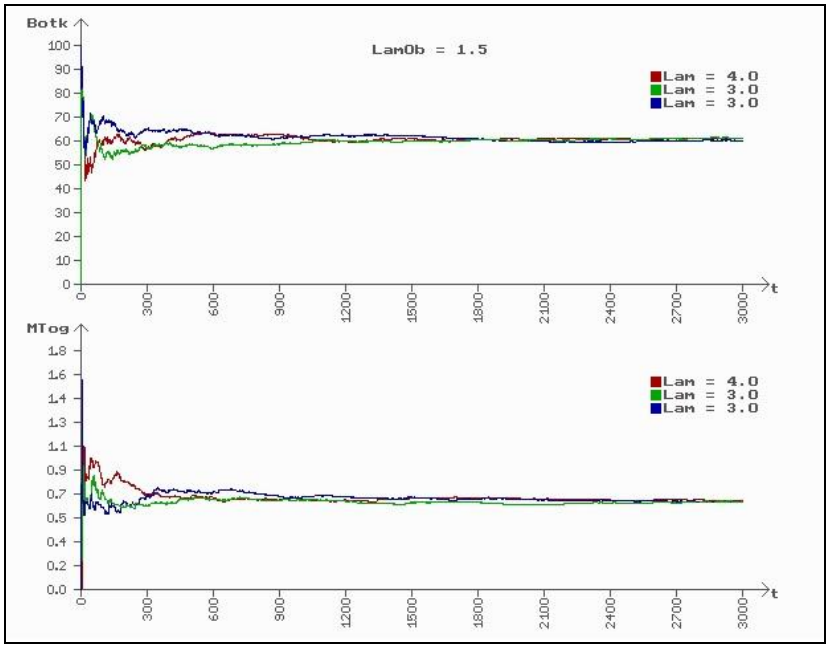


Рис.3 Результаты одной реализаций

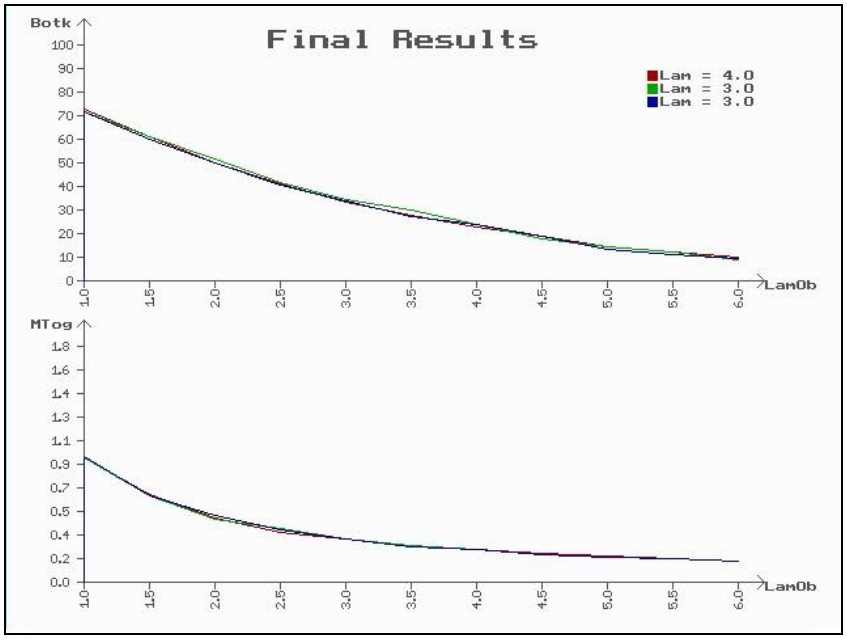


Рис.4 Окончательные результаты в виде график

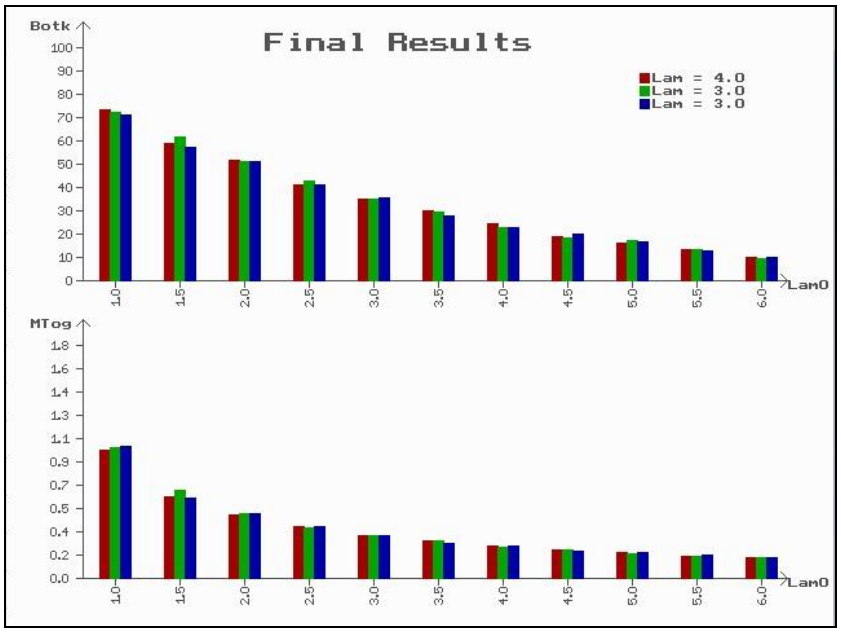
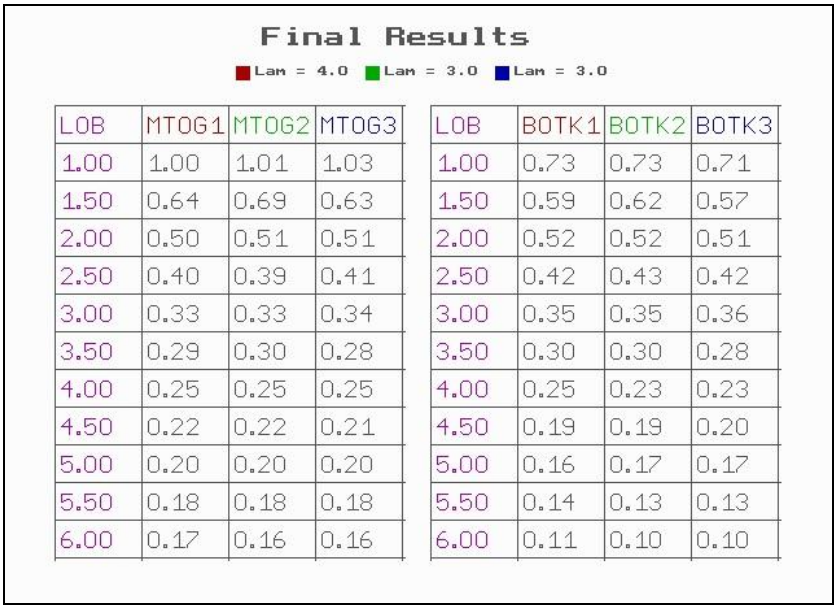


Рис.5 Окончательные результаты в виде гистрограмма



LOB	MTOG1	MTOG2	MTOG3
1.00	1.00	1.01	1.03
1.50	0.64	0.69	0.63
2.00	0.50	0.51	0.51
2.50	0.40	0.39	0.41
3.00	0.33	0.33	0.34
3.50	0.29	0.30	0.28
4.00	0.25	0.25	0.25
4.50	0.22	0.22	0.21
5.00	0.20	0.20	0.20
5.50	0.18	0.18	0.18
6.00	0.17	0.16	0.16

LOB	BOTK1	BOTK2	BOTK3
1.00	0.73	0.73	0.71
1.50	0.59	0.62	0.57
2.00	0.52	0.52	0.51
2.50	0.42	0.43	0.42
3.00	0.35	0.35	0.36
3.50	0.30	0.30	0.28
4.00	0.25	0.23	0.23
4.50	0.19	0.19	0.20
5.00	0.16	0.17	0.17
5.50	0.14	0.13	0.13
6.00	0.11	0.10	0.10

Рис.6 Окончательные результаты в виде таблица

Список использованных источников

Веренинов И.А. Программирование на языке высокого уровня: учеб. пособие / И. А. Веренинов. – СПб.: Изд-во Политехн. ун-та, 2013. – 212с.

Веренинов И.А. Методические указания к курсовому проектированию по курсам "Основы программирования", “Структуры и алгоритмы обработки данных”, “Технология программирования для студентов кафедры ИУС факультета технической кибернетики. – СПб.: 2012. – 26с.

Приложение 1. Исходный код головной программы СМО

```
Program cmo;
Uses crt, graph, cmo_interf, cmo_func, cmo_graph;

type
Modulator = Object {объект, управляющий всей системой модуляций}
  ist: array[1..3] of Istocnik; {массив источниковЃ
  prib: Pribor;  {прибор}
  buf: Buffer;   {буфер}
  graf1, graf2: Graphic; {график для вывод результата одной реализации}

  KMIN, NMIN, PMIN: integer;  {длина реализации, номер источника от которого
  пришла заявка раньше, номер самый медленный источник}
  LamOb: real; {интенсивность прибора для одна реализация}
  color: array[1..3] of integer; {цвета для графики}

  aMTOG, aBOTK: array[1..3, 1..50] of Real; {окончательные значения (средное
время пробивания в систему, вероятность отказа) для каждого источника [1..3]
после каждая реализация}
  rGraf1, rGraf2: Graphic; {графики окончательных результатах}
  tabl1, tabl2: Table; {таблици окончательных результатах}
  bchart1, bchart2: BarChart; {гистрограммы окончательных результатах}

  LIst, LPrib: array[1..3] of real; {Интенсивностей Источников, и параметры
прибора LPrib[1] – начальное, LPrib[2]- конечное, LPrib[3] - шаг}
  Finished: boolean; {модуляция закончена}

  Constructor Init(BSize, kmin1: integer); {инициялизация обьем буфера, и
длина реализации}
  Procedure SetIstocniki(L1,L2,L3: real); {инициализация интенсивностей
источников}
  Procedure SetPribor(L1,L2,DL: real); {инициализация параметры прибора}
  Procedure Reset; {очистить настройки}
  Procedure Modulate(var log: text); {модуляция одна реализация/вутр. цикл}
  Procedure Run(var log: text; delay: boolean); {все реализации/внешный цик}
  Procedure ShowFinalGraphic; {показать график окончательных результатах}
  Procedure ShowFinalBarChart; {показать гистрограмма окончательных
результатах}
  Procedure ShowFinalTable; {показать таблица окончательных результатах}
  Procedure ShowFinalPogresnost; {покказать таблица погрешности окончательных
результатах}
end;
Constructor Modulator.Init(BSize:integer; kmin1: integer);
var i: integer;
begin
  KMIN := kmin1;
  buf.init(BSize);
  Finished := false;
  color[1] := red;
  color[2] := green;
  color[3] := blue;
end;
Procedure Modulator.SetIstocniki(L1,L2,L3: real);
```

```
var i: integer;
begin
  LIst[1] := L1;
  LIst[2] := L2;
  LIst[3] := L3;

  for i:=1 to 3 do
    ist[i].Init(i, LIst[i]);

  PMIN := 1;
  for i:=2 to 3 do
    if ist[i].LAM < ist[PMIN].LAM then
      PMIN := i;
end;
Procedure Modulator.SetPribor(L1,L2,DL: real);
begin
  LPrib[1] := L1;
  LPrib[2] := L2;
  LPrib[3] := DL;
end;
Procedure Modulator.Run(var log: text; delay: boolean);
var CLOB: real; i,j: integer;
begin
  i := 1;
  CLOB := LPrib[1];
  while (CLOB - LPrib[2]) < EPS do
  begin
    LamOb := CLOB;
    Reset;
    Modulate(log);

    if delay then
      readkey;

    for j:=1 to 3 do
    begin
      aMTOG[j][i] := Ist[j].MTOG;
      aBOTK[j][i] := Ist[j].BOTK;
    end;
    CLOB := CLOB + LPrib[3];
    i := i+1;
  end;
  Finished := true;
end;
Procedure Modulator.ShowFinalGraphic;
var i, j: integer; CLOB: real;
begin
  cleardevice;
  rGraf1.init(60,210, red, LPrib[2], 100, LPrib[1], 'LamOb', 'Botk');
  rGraf1.AddTitle('Final Results', 2);
  rGraf1.AddColorGuide(LIst[1], LIst[2], LIst[3]);
  rGraf2.init(60,440, blue, LPrib[2], 0, LPrib[1], 'LamOb', 'MTog');

  i := 1;
  CLOB := LPrib[1];
  while (CLOB - LPrib[2]) < EPS do
```

```
  begin
    for j:=1 to 3 do
    begin
      rGraf1.addPoint(CLOB, aBOTK[j][i], Color[j], j);
      rGraf2.addPoint(CLOB, aMTOG[j][i], Color[j], j);
    end;
    CLOB := CLOB + LPrib[3];
    i := i+1;
  end;
  readkey;
end;
Procedure Modulator.ShowFinalTable;
var i, j: integer; CLOB: real; c: char;
begin
  cleardevice;
  SetColor(DarkGray);
  SetTextStyle(DefaultFont, HorizDir, 2);
  OutTextXY(200, 20, 'Final Results');

  tabl1.Init(40,80,350,270,trunc(((LPrib[2]-LPrib[1])/LPrib[3])+2),4);
  if tabl1.rows <= 12 then
    tabl1.DrawBorders;

  tabl1.setColText(1, 'LOB'  , magenta);
  tabl1.setColText(2, 'MTOG1', red);
  tabl1.setColText(3, 'MTOG2', green);
  tabl1.setColText(4, 'MTOG3', blue);
  tabl1.AddColorGuide(LIst[1], LIst[2], LIst[3]);

  tabl2.Init(330,80,350,270,trunc(((LPrib[2]-LPrib[1])/LPrib[3])+2),4);
  if tabl2.rows <= 12 then
    tabl2.DrawBorders;

  tabl2.setColText(1, 'LOB', magenta);
  tabl2.setColText(2, 'BOTK1', red);
  tabl2.setColText(3, 'BOTK2', green);
  tabl2.setColText(4, 'BOTK3', blue);

  i := 1;
  CLOB := LPrib[1];
  while (CLOB - LPrib[2]) < EPS do
  begin
    tabl1.SetCell(i+1, 1, CLOB, magenta, false);
    tabl2.SetCell(i+1, 1, CLOB, magenta, false);
    for j:=1 to 3 do
    begin
      tabl1.SetCell(i+1, j+1, aMTOG[j][i], darkGray, false);
      tabl2.SetCell(i+1, j+1, aBOTK[j][i], darkGray, false);
    end;
    CLOB := CLOB + LPrib[3];
    i := i+1;
  end;
  c := readkey;
  if keypressed then
    c := readkey;
  if c = #77 then
```

```
    ShowFinalPogresnost;
end;
Procedure Modulator.ShowFinalPogresnost;
var i, j: integer; CLOB, sig: real; c: char;
begin
  SetFillStyle(1, white);
  bar(330, 80, 640, 480);

  tabl2.Init(330,80,350,270,trunc(((LPrib[2]-LPrib[1])/LPrib[3])+2),4);
  if tabl2.rows <= 12 then
    tabl2.DrawBorders;
  tabl2.setColText(1, 'LOB', magenta);
  tabl2.setColText(2, 'Pogr1', red);
  tabl2.setColText(3, 'Pogr2', green);
  tabl2.setColText(4, 'Pogr3', blue);

  i := 1;
  CLOB := LPrib[1];
  while (CLOB - LPrib[2]) < EPS do
  begin
    tabl2.SetCell(i+1, 1, CLOB, magenta, false);
    for j:=1 to 3 do
    begin
      if aBOTK[j][i] = 0 then
        continue;
      sig := sqrt(3.84*(1-aBOTK[j][i])/(KMIN*aBOTK[j][i]));
      tabl2.SetCell(i+1, j+1, sig*100, darkGray, true);
    end;
    CLOB := CLOB + LPrib[3];
    i := i+1;
  end;

  c := readkey;
  if keypressed then
    c := readkey;
  if c = #75 then
    ShowFinalTable;
end;
Procedure Modulator.ShowFinalBarChart;
var i, j: integer; CLOB: real;
begin
  cleardevice;
  bChart1.init(60,210, red, LPrib[2], 100, LPrib[1], 'LamOb', 'Botk');
  bChart1.AddTitle('Final Results', 2);
  bChart1.AddColorGuide(LIst[1], LIst[2], LIst[3]);
  bChart2.init(60,440, blue, LPrib[2], 0, LPrib[1], 'LamOb', 'MTog');

  i := 1;
  CLOB := LPrib[1];
  while (CLOB - LPrib[2]) < EPS do
  begin
    for j:=1 to 3 do
    begin
      bChart1.addPoint(CLOB, aBOTK[j][i], Color[j], j);
      bChart2.addPoint(CLOB, aMTOG[j][i], Color[j], j);
    end;
```

```
    CLOB := CLOB + LPrib[3];
    i := i+1;
  end;
  readkey;
end;
Procedure Modulator.Reset;
var i: integer; ns:string;
begin
  buf.Reset;

  {Generating First Zayavki}
  Randomize;
  for i:=1 to 3 do
    ist[i].Reset;

  prib.Init(LamOb);

  cleardevice;
  str(LamOb:3:1, ns);

  {Empty graphics}
  graf1.init(60, 210, red, KMIN, 100, 0, 't', 'Botk');
  graf1.addTitle('LamOb = ' + ns, 1);
  graf1.addColorGuide(ist[1].Lam, ist[2].Lam, ist[3].Lam);
  graf2.init(60, 440, blue, KMIN, 0, 0, 't', 'MTog');
  graf2.addColorGuide(ist[1].Lam, ist[2].Lam, ist[3].Lam);
end;
Procedure Modulator.Modulate(var log: text);
var i: integer;
begin
  while ist[PMIN].KOL < KMIN do
  begin
    NMIN := 1;
    for i:= 2 to 3 do
      if ist[i].TPOST < ist[NMIN].TPOST then
        NMIN := i;

    if (prib.TOSV <= ist[NMIN].TPOST) AND (not prib.IDLE) then
    begin
      ist[prib.req.id].updateStats(true, prib.TOSV - prib.req.TPOST);
      Prib.idle := true;
    end
    else if (not buf.empty) AND (prib.idle) then
    begin
      prib.obsluz(buf.first);
      buf.pop;
      prib.getNextTOSV;
    end
    else begin
      if not buf.full then
        buf.push(NMIN, ist[NMIN].TPOST)
      else
        ist[NMIN].updateStats(false, 0);
        ist[NMIN].getNextPost;
      end;
      for i:=1 to 3 do
```

```
      begin
        graf1.addPoint(ist[i].KOL, ist[i].BOTK, color[i], i);
        graf2.addPoint(ist[i].KOL, ist[i].MTOG, color[i], i);
      end;
    end;

    writeln(log, 'LamOb: ', LamOb:3:2);
    writeln(log, 'BOTK: ', ist[PMIN].BOTK:3:2);
    writeln(log, 'MTOG: ', ist[PMIN].MTOG:3:2);
end;

Procedure InitModulator(var g: GUI;var modl: Modulator);
begin
  {Buffer Size, Dlina Realizacii}
  Modl.Init(5, g.Windows[3].TextBoxes[1].getInt);
  {Lambda Istocniki}
  With g.Windows[1] do
    Modl.SetIstocniki(TextBoxes[1].getReal, TextBoxes[2].getReal,
                      TextBoxes[3].getReal);
  {Lambda Pribor}
  With g.Windows[2] do
    Modl.SetPribor(TextBoxes[1].getReal, TextBoxes[2].getReal,
                   TextBoxes[3].getReal);
end;

var
modl: Modulator; g: GUI;
log: text;  gd, gm, code: integer;
c: char;
begin

  assign(log, 'log.txt');
  rewrite(log);

  initgraph(gd, gm, '');
  setBkColor(white);
  cleardevice;

  g.init;
  while true do
  begin
    if KeyPressed then
    begin
      c := readKey;
      code := g.HandleInput(c);
      case code of
         0: {Exit Program}
          break;
        21: {Fast Modulation}
        begin
          InitModulator(g, modl);
          Modl.Run(log, false);
          g.Redraw;
        end;
        22: {Normal Modulation}
        begin
```

```
          InitModulator(g, modl);
          Modl.Run(log, true);
          g.Redraw;
        end;
        31: {Graph Final Results}
        begin
          if Modl.Finished then
            Modl.ShowFinalGraphic;
          g.Redraw;
        end;
        32: {Table Final Results}
        begin
          If Modl.Finished then
            Modl.ShowFinalTable;
          g.Redraw;
        end;
        33: {BarChart Final Results}
        begin
          if Modl.Finished then
            Modl.ShowFinalBarChart;
          g.Redraw;
        end;
      end;
    end;
  end;

  close(log);
  closegraph;
end.
```

Приложение 2. Исходный код модуля CMO_FUNC

```
unit cmo_func;

INTERFACE

const
EPS = 0.00001; {для сравнение переменных типа real}

type
Request = Record {заявка}
  TPost: real; {момонт поступления}
  ID: integer; {номер источника}
end;

pQueue = ^queue; {тип указатель на элемент очереди}
Queue = Record {элемент очереди}
  Req: Request; {заявка}
  Next: pQueue; {указатель на следующий элемент}
end;
```

```
Pribor = Object {прибор}
  LAM: real; {интензивность}
  TAYOB, TOSV: real; {длительность обслужывания, время освобождения прибора}
  Idle: boolean; {прибор пустой/ не обрабатывает заявка}
  Req: Request; {заявка внутри прибора}
  Constructor Init(Lambda: real); {инициализация с интензивность}
  Procedure obsluz(nreq: Request); {обслужить заявка}
  Procedure getNextTOSV; {вычислить време освобождение прибора}
end;

Istocnik = Object {источник}
  ID: integer; {номер (1..3)}
  LAM: real; {интензивность}
  TPOST: real; {момент поступления заявка}

  KOL, KOBR, KOTK: integer; {количество заявок, кол. Обработаных заявок, кол.
отказов}
  TOG, MTOG, BOTK: real; {общее время пребывания в систему, среднее время
пребы в систему, вероятность отказа}

  Procedure getNextPost; {вычислить момент поступления заявка}
  Procedure updateStats(OBR: boolean; DT: real); {обновить
статистику(KOL,KOBR,MTOG…), OBR-заявка обработана (или отказ), DT- время
пребывания в систему}
  Procedure Reset; {Очистить все параметры}
  Constructor Init(N:integer; L: real); {инициализация, номер, интензивность}
end;

Buffer = Object {буфер}
  pFirst,  pLast:   pQueue; {указатель на первый и последный элемент}
  First,    Last:  Request; {заявка в первый и последный элемент}
  Size, Max_Size:  integer; {количество заявок в буфере, макс. количество}

  Constructor Init(MSIZE: integer); {инициализация – обьем буфера}
  Procedure Push(id:integer; TPost:real); {добавить заявка в буфере}
  Procedure Pop; {вытащить заявка которая пришла первая}
  Procedure Reset; {очистить буфер}
  Function Full: Boolean; {буфер полный}
  Function Empty: Boolean; {буфер пустой}
end;

IMPLEMENTATION

Procedure Pribor.Obsluz(nReq: Request);
begin
  Req := nReq;
  Idle := false;
end;
Procedure Pribor.GetNextTOSV;
begin
  TAYOB := -1/Lam*ln(random);
  TOSV := req.TPOST + TAYOB;
end;
Constructor Pribor.Init(Lambda: real);
begin
  LAM  := Lambda;
```

```
  TOSV := 0;
  idle := true;
end;
Constructor Istocnik.Init(N:integer; L: real);
begin
  LAM := L; ID := n;
  Reset;
end;
Procedure Istocnik.Reset;
begin
  TPOST := 0;
  KOL := 0; KOBR := 0; KOTK :=0;
  TOG := 0; MTOG := 0; BOTK :=0;
  getNextPost;
end;
Procedure Istocnik.GetNextPost;
begin
  TPOST := TPOST + -1/Lam*ln(random);
end;
Procedure Istocnik.updateStats(OBR: boolean; DT: real);
begin
  KOL := KOL + 1;
  if OBR then
  begin
    KOBR := KOBR + 1;
    TOG  := TOG + DT;
    MTOG := TOG / KOBR;
  end
  else KOTK := KOTK + 1;

  BOTK := KOTK / KOL;
end;
Constructor Buffer.Init(MSIZE: integer);
begin
  Max_Size := MSIZE;
  Reset;
end;
Procedure Buffer.Reset;
var tmp: pQueue;
begin
  while pFirst <> NIL do
  begin
    tmp := pFirst;
    pFirst := pFirst^.Next;
    Dispose(tmp);
  end;
  Size := 0;
  pFirst := NIL;
  pLast := NIL;
end;
Procedure Buffer.Push(id:integer; TPOST:real);
var tmp: pQueue; rTmp: Request;
begin
  rTmp.ID := id;
  rTmp.TPOST := TPOST;
```

```
  new(tmp);
  tmp^.Req  := rTmp;
  tmp^.Next := NIL;

  if pFirst = NIL then
  begin
    pFirst := tmp;
    pLast := tmp;
  end
  else begin
    pLast^.Next := tmp;
    pLast := tmp;
  end;

  Size  := size + 1;
  Last  := pLast^.Req;
  First := pFirst^.Req;
end;
Procedure Buffer.Pop;
var tmp: pQueue;
begin
  if pFirst <> NIL then
  begin
    Tmp := pFirst;
    pFirst := pFirst^.Next;
    Dispose(tmp);
    Size := Size - 1;
    First := pFirst^.Req;
  end;
end;
Function Buffer.Full: Boolean;
begin
  Full := false;
  if Size >= Max_Size then
    Full := true;
end;
Function Buffer.Empty: Boolean;
begin
  Empty := false;
  if Size = 0 then
    Empty := true;
end;

begin
end.
```

Приложение 3. Исходный код модуля CMO_GRAPH

```
unit cmo_graph;

INTERFACE
Uses crt, graph;

const
EPS = 0.00001; {для сравнение переменных типа real}

type
Graphic = Object {график}
  X0, Y0, H, W: integer; {нижняя левая точка, ширина, высота}
  MAX_X, MAX_Y, OffsetX: real; {макс значение на ось X, макс значение на ось
Y, начальное значение ось X (в точка X0)}
  Color: integer; {цвет}
  PX,PY: array[1..3] of integer; {предедушый точки от источники 1..3 }
  RMX, RMY: real; {коеффициент масштаб x,y}
  Name_X, Name_Y: String[5]; {наименование осей}

  Constructor Init(x,y,cl:integer;mx1,my1,osx:real;nx,ny:String);
{инициализация: x,y - нижняя левая точка, cl-цвет, mx1, my1 - макс значение
x, y; osx – начальное значение ось x; nx,ny – наименование осей}
  Procedure DrawOy; {нарисовать ось y}
  Procedure DrawOx;virtual; {нарисовать ось x}
  Procedure AddPoint(mx, my: real; col, id: integer);virtual; {добавить
точка: mx,my: точка без масштаб, col-цвет, id- номер источника}
  Procedure AddTitle(title: String;FSize: integer); {добавить заголовка,
fsize – размер фонта}
  Procedure AddColorGuide(L1, L2, L3: real); {добавить цветной указатель,
принимает значениях интензивностей источников}
end;
BarChart = Object(Graphic) {гистрограмма, наследует график}
  Procedure AddPoint(mx, my: real; col, id: integer);virtual; {добавить
точка: mx,my: точка без масштаб, col-цвет, id- номер источника}
  Procedure DrawOx;virtual; {нарисовать ось x}
end;
Table = Object {таблица}
  X0, Y0, H, W: integer; {левая верхная точка, высота, ширина}
  ROWS, COLS: integer; {количество строк, столбцов}
  Constructor Init(x1,y1,h1,w1,rows1,cols1: integer); {инициализация: левая
верхная точка, высота, ширина, кол. Строк, столбцов}
  Procedure DrawBorders; {нарисовать границы таблицы}
  Procedure SetColText(col:integer; title:String; color:integer); {добавить
текст - title в первая строка, в столбец col, с цвет - color}
  Procedure SetCell(row, col:integer; inf:real; color:integer; prc:boolean);
{добавить значение inf в строка row, столбец col, с цвет color, значение в
проценты?}
  Procedure AddColorGuide(L1,L2,L3: real); {добавить цветной указатель,
принимает значениях интензивностей источников}
end;

IMPLEMENTATION
```

```
{ * * * * * TABLE * * * * * * }

Constructor Table.Init(x1,y1,h1,w1,rows1,cols1: integer);
begin
  X0 := x1; Y0 := y1;
  H  := h1; W  := w1;
  ROWS := rows1;
  COLS := cols1;
end;
Procedure Table.DrawBorders;
var i,x,dx,y,dy: integer;
begin
  SetColor(DarkGray);

  y := y0; dy := H div rows;
  for i:= 1 to rows+1 do
  begin
    Line(x0, y, x0+w, y);
    y := y + dy;
  end;

  x:=x0; dx:= W div cols;
  for i:=1 to cols+1 do
  begin
    Line(x, y0, x, y0+h);
    x := x + dx;
  end;
end;
Procedure Table.SetColText(col:integer; title:String; color: integer);
begin
  SetTextStyle(SmallFont, HorizDir, 7);
  SetColor(color);
  OutTextXY(x0+5+(col-1)*(w div cols), y0+5, title);
end;
Procedure Table.SetCell(row, col:integer; inf: real; color:integer; prc:
boolean);
var x,y: integer; ns: String[4];
begin
  SetTextStyle(SmallFont, HorizDir, 7);
  SetColor(color);
  x := x0+5+(col-1)*(w div cols);
  y := y0+5+(row-1)*(h div rows);
  if prc then
  begin
    str(inf:2:0, ns);
    OutTextXY(x,y, ns + '%');
  end
  else begin
    str(inf:2:2, ns);
    OutTextXY(x,y, ns);
  end;
end;
Procedure Table.AddColorGuide(L1,L2,L3: real);
var ns: String[4];
begin
  SetColor(DarkGray);
```

```
  SetTextStyle(DefaultFont, HorizDir, 1);

  str(L1:2:1, ns);
  setFillStyle(1, red);
  bar(x0+140,y0-30, x0+150, y0-20);
  OutTextXY(x0+155, y0-30, 'Lam = ' + ns);

  str(L2:2:1, ns);
  setFillStyle(1, green);
  bar(x0+240, y0-30, x0+250, y0-20);
  OutTextXY(x0+255, y0-30, 'Lam = ' + ns);

  str(L3:2:1, ns);
  setFillStyle(1, blue);
  bar(x0+340,y0-30, x0+350, y0-20);
  OutTextXY(x0+355, y0-30, 'Lam = ' + ns);
end;

{ * * * * * * * GRAPHIC * * * * * * * }

Constructor Graphic.Init(x,y,cl:integer;mx1,my1,osx:real;nx,ny:String);
var i: integer;
begin
  X0 := x-1; Y0 := y+1;
  W  := 500; H  := 180;
  MAX_X := mx1; MAX_Y := my1;
  OffsetX := osx;

  for i:=1 to 3 do
  begin
    PX[i] := x0;
    PY[i] := y0;
  end;

  RMX := w/(MAX_X - OffsetX);
  if MAX_Y <> 0 then
    RMY := H/MAX_Y;

  Color := cl;
  Name_X := nx; Name_Y := ny;
  DrawOy; DrawOx;
end;
Procedure Graphic.AddColorGuide(L1,L2,L3: real);
var ns: String[4];
begin
  SetTextStyle(DefaultFont, HorizDir, 1);

  str(L1:2:1, ns);
  setFillStyle(1, red);
  bar(490,y0-160, 497, y0-153);
  OutTextXY(500, y0-160, 'Lam = ' + ns);

  str(L2:2:1, ns);
  setFillStyle(1, green);
  bar(490, y0-150, 497, y0-143);
  OutTextXY(500, y0-150, 'Lam = ' + ns);
```

```
  str(L3:2:1, ns);
  setFillStyle(1, blue);
  bar(490,y0-140, 497, y0-133);

  OutTextXY(500, y0-140, 'Lam = ' + ns);
end;
Procedure Graphic.AddTitle(title: String; FSize: integer);
begin
  SetTextStyle(DefaultFont, HorizDir, FSize);
  if FSize = 1 then
    OutTextXY(280, y0-180, title)
  else
    OutTextXY(200, 20, title);
end;
Procedure Graphic.DrawOx;
var ix, dx: real; rxi: integer;ns:String[4];
begin
  setColor(darkGray);
  setTextStyle(DefaultFont, HorizDir, 1);

  line(x0, y0, x0+w+20, y0);
  line(x0+w+20, y0, x0+w+15, y0+5);
  line(x0+w+20, y0, x0+w+15, y0-5);
  outTextXY(x0+w+21, y0, name_x);

  setTextStyle(SmallFont, VertDir, 4);
  DX := (max_x-OffsetX) / 10; ix := OffsetX;
  while (ix - max_x) < EPS{ <= max_x} do
  begin
    rxi := x0 + trunc((ix-OffsetX)*RMX);
    line(rxi, y0, rxi, y0+5);
    if OffsetX > 0 then
      str(ix:2:1, ns)
    else
      str(ix:3:0, ns);
    outTextXY(rxi - 5, y0+5, ns);
    ix := ix + dx;
  end;
end;
Procedure Graphic.DrawOy;
var iy, dy: integer; ns:String[4];
begin
  setColor(darkGray);
  setTextStyle(DefaultFont, HorizDir, 1);

  line(x0, y0, x0, y0-h-20);
  line(x0, y0-h-20, x0-5, y0-h-15);
  line(x0, y0-h-20, x0+5, y0-h-15);
  outTextXY(x0-40, y0-h-20, name_y);

  SetTextStyle(SmallFont, HorizDir, 4);
  dy := H div 10; iy := 0;
  while iy <= h do
  begin
    line(x0, y0-iy, x0-5, y0-iy);
```

```
    if MAX_Y = 0 then
      str(iy/100:3:1, ns)
    else
      str(iy/RMY:3:0, ns);
    outTextXY(x0-25, y0-iy-5, ns);
    iy := iy + dy;
  end;
end;
Procedure Graphic.AddPoint(mx, my:real; col, id: integer);
var X, Y: integer;
begin
  my := my*100;
  X := X0 - trunc(OffsetX*RMX) + trunc(mx*RMX);
  if MAX_Y = 0 then
    Y := Y0 - trunc(my)
  else
    Y := Y0 - trunc(my * RMY);

  if mx - MAX_X < EPS then
  begin
    SetColor(col);
    Line(PX[id], PY[id], X, Y);
    PX[id] := X; PY[id] := Y;
  end;
end;

{ * * * * * * * BAR CHART * * * * * * }

Procedure BarChart.AddPoint(mx, my: real; col, id: integer);
var X, Y: integer;
begin
  my := my*100;
  X := X0 - trunc(OffsetX*RMX) + trunc(mx*RMX);
  if MAX_Y = 0 then
    Y := Y0 - trunc(my)
  else
    Y := Y0 - trunc(my * RMY);

  if mx - MAX_X < EPS then
  begin
    SetColor(col);
    SetFillStyle(1, col);
    Bar(X-12+8*(id-1),Y, X-4+8*(id-1), Y0);
  end;
end;
Procedure BarChart.DrawOx;
begin
  SetColor(darkGray);
  Line(x0,y0,x0+25,y0);
  x0 := x0 + 25;
  inherited DrawOx;
end;
begin
end.
```

Приложение 4. Исходный код модуля CMO_INTERFACE

```
unit cmo_interf;

INTERFACE
uses crt, graph;

const
MBOX_W = 160; {ширина MenuBox}
MBOX_H = 30; {высота MenuBox}
WDOW_W = 300; {ширина окно – Window}
WDOW_h = 200; {высота окно}
TBOX_W = 180; {ширина текстовое поле}
TBOX_H = 30; {высота текстовое поле}
MAX_C  = 10; {макс. символы в текстовое поле}

Type
MenuBox = Object {подменю}
  X, Y: integer; {левая верхная точка}
  H, W: integer; {ширина, высота}
  Text: String[MAX_C+1]; {текст}
  Foc: boolean; {сфокусированно}
  Opened: boolean; {открыто}
  visible: boolean; {видно}
  Procedure Redraw;virtual; {перерисовать}
  Procedure SetFocus; {сфокусировать}
  Procedure ClearFocus; {очистить фокус}
  Procedure SetText(s: String); {добавить текст}
  Procedure Show; {показать на экране}
  Constructor Init(x1,y1: integer); {инициализация, левая верхная точка}
end;
MenuList = Object(MenuBox) {меню}
  N_Items: integer; {количество подменюов}
  Items: array[1..3] of MenuBox; {подменю}
  Procedure Redraw;virtual; {перерисовать}
  Procedure AddItem(s: String); {добавить подменю}
  Procedure Open; {открыть}
  Procedure Close; {закрыть}
  Constructor Init(x1,y1: integer); {инициализация, левая верхная точка}
end;
TextBox = Object(MenuBox) {текстовое поле}
  CPOS: integer; {позиция курсора}
  Constructor Init(x1,y1: integer); {инициализация, левая верхная точка}
  Procedure PutChar(c: char); {добавить символь}
  Procedure ClearChar; {очистить символь перед курсора}
  Procedure Activate; {активировать – можно вводить/очистить текст}
  Procedure Deactivate; {деактивация – невозможно вводит/очистить текст}
  Procedure SetText(s: string); {задать начальное значение/ текст}
  Function getReal: real; {получить записанное значение в виде real}
  Function getInt: integer; {получить записанное значение в виде integer}
end;
Window = Object {окно}
  X,Y: integer; {левая верхная точка}
  H,W: integer; {ширина, высота}
```

```
  N_TextBoxes: integer; {количество текстовых полях}
  ActiveBox: integer; {активное текствое поле}
  Title: String[20]; {заголовка}
  Labels: array[1..3] of String[5]; {текст перед текстовых полях}
  TextBoxes: array[1..3] of TextBox; {текстовые поля}
  Constructor Init(x1,y1: integer); {инициализация, левая верхная точка}
  Procedure AddTextBox(text: string); {добавить текствое поле}
  Procedure SetTitle(s: string); {задать заголовка}
  Procedure AddText(filename: string); {добавить текст с файла}
  Function HandleInput:boolean; {обрабатывать ввод пользователя}
  Procedure Show; {показать на экране}
end;
GUI = Object {пользовательский интерфейс}
  Menus: array[1..4] of MenuList; {меню}
  Windows: array[1..6] of Window; {окна}
  ActiveList: integer; {активное меню (1..4)}
  ActiveBox: integer; {активное подменю(0..3)}
  Constructor Init; {инициализация}
  Function HandleInput(c: char):integer; {обрабатывать ввод пользователя}
  Procedure Redraw; {перерисовать}
end;

IMPLEMENTATION

{* * * * * * MenuBox * * * * * * * * * }

Constructor MenuBox.Init(x1,y1: integer);
begin
  X := x1; Y := y1;
  H := MBOX_H;
  W := MBOX_W;
  Foc := false; Opened := false;
  Visible := false;
end;
Procedure MenuBox.Show;
begin
  Visible := true;
  Redraw;
end;
Procedure MenuBox.SetText(s: String);
begin
  Text := s;
  if Visible then
    Redraw;
end;
Procedure MenuBox.Redraw;
begin
  if FOC then
    SetFillStyle(1, cyan)
  else
    SetFillStyle(1, blue);
  bar(X,Y,X+W, Y+H);

  SetColor(white);
  Rectangle(X,Y,X+W,Y+H);
```

```
  if NOT FOC then
    SetColor(blue);
  Rectangle(X+2,Y+2, X+W-2, Y+H-2);

  SetColor(white);
  SetTextStyle(DefaultFont, HorizDir, 2);
  OutTextXY(X+5,Y+5, Text);
end;
Procedure MenuBox.SetFocus;
begin
  Foc := true;
  Redraw;
end;
Procedure MenuBox.ClearFocus;
begin
  Foc := false;
  Redraw;
end;

{ * * * * * * * MenuList * * * * * * * * * }

Procedure MenuList.Redraw;
var i: integer;
begin
  inherited Redraw;
  If Opened then
    for i:=1 to N_Items do
      Items[i].Redraw;
end;
Procedure MenuList.AddItem(s: string);
begin
  N_Items := N_Items+1;
  Items[N_Items].Init(X, Y+N_Items*H);
  Items[N_Items].SetText(s);
end;
Procedure MenuList.Open;
var i: integer;
begin
  Opened := true;
  for i:=1 to N_Items do
    Items[i].Show;
end;
Procedure MenuList.Close;
begin
  Opened := false;
end;
Constructor MenuList.Init(x1,y1:integer);
begin
  Inherited Init(x1,y1);
  N_Items := 0;
  Opened := false;
end;

{ * * * * * * TextBox * * * * * * * * * * * }

Constructor TextBox.Init(x1,y1: integer);
```

```
begin
  X := x1; H := TBOX_H;
  Y := y1; W := TBOX_W;
  CPOS := 1; Text[0] := '0';
  FOC := false;
end;
Procedure TextBox.Activate;
begin
  Text[CPOS] := '_';
  inherited setFocus;
end;
Procedure TextBox.Deactivate;
begin
  Text[CPOS] := #0;
  inherited ClearFocus;
end;
Function TextBox.getReal: real;
var r: real; err: integer; s: string[10];
begin
  s := Copy(Text, 1, CPOS-1);
  val(s, r, err);
  getReal := r;
end;
Function TextBox.getInt: integer;
begin
  getInt := trunc(getReal);
end;
Procedure TextBox.ClearChar;
begin
  if CPOS > 1 then
  begin
    Text[CPOS-1] := '_';
    Text[CPOS] := #0;
    CPOS := CPOS-1;
    Redraw;
  end;
end;
Procedure TextBox.PutChar(c: char);
begin
  if CPOS < MAX_C then
  begin
    Text[CPOS] := c;
    CPOS := CPOS + 1;
    Text[CPOS] := '_';
    TextBox.Redraw;
  end;
end;
Procedure TextBox.SetText(s: string);
var i: integer;
begin
  for i:=1 to Length(s) do
    Text[i] := s[i];
  CPOS := Length(s)+1;
end;

{* * * * * * * Window * * * * * * * * * * }
```

```
Constructor Window.Init(x1,y1: integer);
begin
  X := x1; W := WDOW_W;
  Y := y1; H := WDOW_H;
  N_TextBoxes := 0;
  ActiveBox := 1;
end;
Procedure Window.SetTitle(s: string);
begin
  Title := s;
end;
Procedure Window.AddTextBox(text: string);
begin
  N_TextBoxes := N_TextBoxes + 1;
  Labels[N_TextBoxes] := text;
  TextBoxes[N_TextBoxes].Init(X+W div 3,Y+15+N_TextBoxes*30);
end;
Procedure Window.Show;
var i: integer;
begin
  SetFillStyle(1, cyan);
  bar(X,Y,X+W,Y+MBOX_H);
  SetFillStyle(1, blue);
  bar(X,Y+MBOX_H,X+W, Y+H);

  SetColor(white);
  Rectangle(X,Y,X+W,Y+H);
  Rectangle(X+2,Y+2, X+W-2, Y+H-2);

  SetTextStyle(DefaultFont, HorizDir, 2);
  OutTextXY(X+5,Y+5, Title);

  if N_TextBoxes >= 1 then
  Begin
    TextBoxes[1].Activate;
    for i:=1 to N_TextBoxes do
    begin
      OutTextXY(X+10, Y+20+i*30, Labels[i]);
      TextBoxes[i].Redraw;
    end;
  End;
end;
Function Window.HandleInput:boolean;
var c: char;
begin
  HandleInput := true;
  while true do
  begin
    if not keypressed then
      continue;

    c := readkey;
   case c of
    #8:  TextBoxes[ActiveBox].clearChar; {Backspace}
    #72: {Up arrow}
```

```
    begin
      if ActiveBox > 1 then
      begin
        TextBoxes[ActiveBox].Deactivate;
        ActiveBox := ActiveBox - 1;
        TextBoxes[ActiveBox].Activate;
      end;
    end;
    #13, #80:  {Enter, Down Arrow}
    begin
      if ActiveBox = N_TextBoxes then
      begin
        if c = #13 then
        begin
          TextBoxes[ActiveBox].Deactivate;
          ActiveBox := 1;
          break;
        end;
      end
      else begin
        TextBoxes[ActiveBox].Deactivate;
        ActiveBox := ActiveBox + 1;
        TextBoxes[ActiveBox].Activate;
      end;
    end;
    '0'..'9', '.': TextBoxes[ActiveBox].putChar(c);
    #27: {ESC}
    begin
      HandleInput := false;
      TextBoxes[ActiveBox].Deactivate;
      ActiveBox := 1;
      break;
    end;
   end;
  end;
end;
Procedure Window.AddText(filename: string);
var f: text; iy: integer; line: string;
begin
  assign(f, filename);
  reset(f);
   {
  SetTextStyle(SmallFont, HorizDir, 7);
  }
  SetTextStyle(DefaultFont, HorizDir, 1);
  iy := Y+30;
  while not eof(f) do
  begin
    readln(f, line);
    OutTextXY(X+10, iy, line);
    iy := iy+15;
  end;
  close(f);
end;
{ * * * * * GUI * * * * * * * * * * }
```

```
Constructor GUI.Init;
begin
  ActiveList := 1;
  ActiveBox  := 0;

  Menus[1].Init(0,0);
  Menus[1].SetText('Options');
  Menus[1].SetFocus;

  Menus[1].AddItem('Istocniki');
  Menus[1].AddItem('Pribor');
  Menus[1].AddItem('Pogresnost');
  Menus[1].Show;

  Menus[2].Init(160, 0);
  Menus[2].SetText('Modelling');
  Menus[2].AddItem('Fast Mode');
  Menus[2].AddItem('Normal Mode');
  Menus[2].Show;

  Menus[3].Init(320, 0);
  Menus[3].SetText('Results');
  Menus[3].AddItem('Graph');
  Menus[3].AddItem('Table');
  Menus[3].AddItem('Bar Chart');
  Menus[3].Show;

  Menus[4].Init(480, 0);
  Menus[4].Settext('Help');
  Menus[4].AddItem('Usage');
  Menus[4].AddItem('About');
  Menus[4].Show;

  Windows[1].Init(160, 150);
  Windows[1].SetTitle('Istocniki');
  Windows[1].AddTextBox('Lam1');
  Windows[1].TextBoxes[1].SetText('4.00');
  Windows[1].AddTextBox('Lam2');
  Windows[1].TextBoxes[2].SetText('3.00');
  Windows[1].AddTextBox('Lam3');
  Windows[1].TextBoxes[3].SetText('3.00');

  Windows[2].Init(160, 150);
  Windows[2].SetTitle('Pribor');
  Windows[2].AddTextBox('Lam1');
  Windows[2].TextBoxes[1].SetText('1.00');
  Windows[2].AddTextBox('Lam2');
  Windows[2].TextBoxes[2].SetText('6.00');
  Windows[2].AddTextBox('DLam');
  Windows[2].TextBoxes[3].SetText('0.50');

  Windows[3].Init(160, 150);
  Windows[3].SetTitle('Dlina Realizacii');
  Windows[3].AddTextBox('KMIN');
  Windows[3].TextBoxes[1].SetText('3000');
```

```
  Windows[4].Init(160,150);
  Windows[4].SetTitle('Pogresnost');
  Windows[4].AddTextBox('DOV VER');
  Windows[4].TextBoxes[1].SetText('0.95');

  Windows[5].Init(150,100);
  Windows[5].H := 320;
  Windows[5].W := 380;
  Windows[5].SetTitle('Usage');

  Windows[6].Init(150,100);
  Windows[6].H := 200;
  Windows[6].W := 350;
  Windows[6].SetTitle('About');
end;
Procedure GUI.Redraw;
var i: integer;
begin
  cleardevice;
  for i:=1 to 4 do
    Menus[i].Show;
end;
Function GUI.HandleInput(c: char): integer;
begin
  HandleInput := 1;
  case c of
    #72:  {up arrow}
    begin
      if ActiveBox > 0 then
      begin
        Menus[ActiveList].Items[ActiveBox].ClearFocus;
        ActiveBox := ActiveBox - 1;
        if ActiveBox > 0 then
          Menus[ActiveList].Items[ActiveBox].SetFocus;
      end
      else
        ActiveBox := 0;
    end;
    #80: {down arrow}
    begin
      if (ActiveBox < Menus[ActiveList].N_Items) And
         (Menus[ActiveList].Opened) then
      begin
        If ActiveBox > 0 then
          Menus[ActiveList].Items[ActiveBox].ClearFocus;
        ActiveBox := ActiveBox + 1;
        Menus[ActiveList].Items[ActiveBox].SetFocus;
      end;
    end;
    #75: {Left arrow}
    begin
      if (ActiveList > 1) And (ActiveBox = 0) then
      begin
        Menus[ActiveList].ClearFocus;
        Menus[ActiveList].Close;
        ActiveList := ActiveList - 1;
```

```
    Menus[ActiveList].SetFocus;
    Redraw;
  end;
end;
#77: {Right Arrow}
begin
  if (ActiveList < 4) And (ActiveBox = 0) then
  begin
    Menus[ActiveList].ClearFocus;
    Menus[ActiveList].Close;
    ActiveList := ActiveList + 1;
    Menus[ActiveList].SetFocus;
    Redraw;
  end;
end;
#13: {ENTER}
begin
  if ActiveBox = 0 then
  begin
    ActiveBox := 1;
    Menus[ActiveList].Open;
    Menus[ActiveList].Items[ActiveBox].SetFocus;
  end
  else if (ActiveList = 1) AND (ActiveBox = 1) then
  begin
    Windows[1].Show;
    Windows[1].HandleInput;
    clrscr;
    Redraw;
  end
  else if (ActiveList = 1) AND (ActiveBox = 2) then
  begin
    Windows[2].Show;
    Windows[2].HandleInput;
    clrscr;
    Redraw;
  end
  else if (ActiveList = 1) AND (ActiveBox = 3) then
  begin
    Windows[4].Show;
    readkey;
    clrscr;
    Redraw;
  end
  else if (ActiveList = 2) AND (ActiveBox > 0) then
  begin
    Windows[3].Show;
    if Windows[3].HandleInput <> false then
      HandleInput := ActiveList*10 + ActiveBox;
    clrscr;
    Redraw;
  end
  else if (ActiveList = 3) AND (ActiveBox > 0) then
  begin
    HandleInput := ActiveList*10 + ActiveBox;
  end
```

```
      else if (ActiveList = 4) AND (ActiveBox = 1) then
      begin
        Windows[5].Show;
        Windows[5].AddText('Usage.txt');
        readkey;
        clrscr;
        Redraw;
      end
      else if (ActiveList = 4) AND (ActiveBox = 2) then
      begin
        Windows[6].Show;
        Windows[6].AddText('About.txt');
        readkey;
        clrscr;
        Redraw;
      end;
    end;
    #27:
    begin
      if Menus[ActiveList].Opened then
      begin
        if ActiveBox > 0 then
          Menus[ActiveList].Items[ActiveBox].ClearFocus;
        Menus[ActiveList].Close;
        ActiveBox := 0;
        Redraw;
      end
      else
        HandleInput := 0;  {ESC}
    end;
  end;
end;

begin
end.
```

Приложение 5. Дополнительные файлы

USAGE.TXT

```
   * Navigating through the interface:

     Left Arrow, Right Arrow
     Up Arrow, Down Arrow

   * Activate Item/Option * * * * * *

     Enter Key

   * Close/Exit Active Item * * * * *
```

```
   ESC Key


 *  Table Results  contain extra
 *  information  ->  Right Arrow
 *  to access Pogresnost Results
```

ABOUT.TXT

```
SMO - An application for modelling
a system for mass service.

Author: Stoykoski Nikola
Group: 13534/1

2017
```