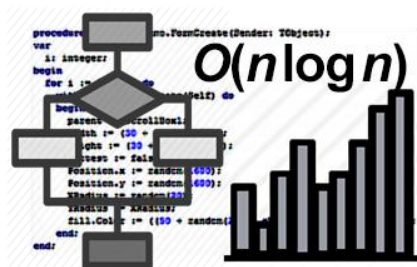


МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ  
РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

## БАЗОВЫЕ АЛГОРИТМЫ И ИХ ПРОГРАММИРОВАНИЕ В СИСТЕМЕ DELPHI

Методические указания к лабораторным работам  
по курсу  
«Программирование и основы алгоритмизации»



Рязань 2018

УДК 621.38

Базовые алгоритмы и их программирование в системе Delphi: методические указания к лабораторным работам по курсу «Программирование и основы алгоритмизации» / Рязан. гос. радиотехн. ун-т; сост.: М.Д. Ершов, А.А. Селяев, В.В. Стротов. Рязань: РГРТУ, 2018. 76 с.

Рассмотрены правила работы с графикой, датой и временем в системе Delphi. Рассматриваются простейшие алгоритмы, такие как сортировка и поиск. Уделено внимание алгоритмам обработки одномерных сигналов. В каждой лабораторной работе приводятся примеры выполнения индивидуального задания.

Предназначены для бакалавров дневной формы обучения направлений подготовки 27.03.04 «Управление в технических системах» и 01.03.02 «Прикладная математика и информатика».

Табл. 9. Ил. 31.

*Delphi, программа, графика, время, алгоритмы, сортировка, поиск, цифровая обработка сигналов, среда*

Печатаются по решению редакционно-издательского совета Рязанского государственного радиотехнического университета.

Рецензент: кафедра автоматики и информационных технологий в управлении Рязанского государственного радиотехнического университета (зав. кафедрой доц. П.В. Бабаян)

Базовые алгоритмы и их программирование в системе Delphi

Составители: Е р ш о в Максим Дмитриевич  
С е л я е в Александр Анатольевич  
С т р о т о в Валерий Викторович

Редактор М.Е. Цветкова  
Корректор С.В. Макушина

Подписано в печать 28.02.18. Формат бумаги 60х84 1/16.

Бумага писчая. Печать трафаретная. Усл. печ. л. 4,75.

Тираж 30 экз. Заказ

Рязанский государственный радиотехнический университет.

390005, Рязань, ул. Гагарина, 59/1.

Редакционно-издательский центр РГРТУ.

## ВВЕДЕНИЕ

В различных сферах деятельности важно уметь сформулировать задачу и составить алгоритм ее решения, чтобы получить требуемый результат. Так, для решения математических, физических, инженерных, повседневных задач, как правило, необходимо выполнить определенную последовательность действий.

Зачастую для детального понимания хода решения задачи необходимо визуализировать процесс решения. С этой целью разработаны лабораторные работы, посвященные изучению графических возможностей среды **Delphi**. В результате выполнения данных работ студенты должны научиться рисовать простейшие и сложные фигуры, строить графики, отображать текстовую информацию, создавать анимацию.

Важное место занимает анализ временной трудоемкости алгоритма. Хотя реальное время решения задачи или выполнения какого-то шага алгоритма зависит от конкретного вычислительного устройства, но все же можно сравнить временную трудоемкость различных решений, реализованных одnogруппниками. С целью обучения работе со временем в среде **Delphi** создана лабораторная работа, посвященная изучению типа «дата-время» и компонента «таймер».

Для получения опыта реализации алгоритмов разработаны лабораторные работы, посвященные алгоритмам сортировки и поиска, которые часто встречаются при решении реальных задач. Так, студенты должны реализовать тот или иной алгоритм сортировки массива. Также должны быть реализованы алгоритмы поиска значений в массиве, положения экстремумов и нулей функций.

Последняя лабораторная работа направлена на изучение основ цифровой обработки сигналов и получение опыта практической реализации простых алгоритмов анализа и обработки одномерных сигналов. Данная работа напрямую связана с формированием навыков алгоритмизации и с закреплением ранее изученного материала. Кроме того, студенты получают знания, которые помогут при изучении других дисциплин в рамках направления «Управление в технических системах».

Лабораторные работы рассчитаны на выполнение с использованием интегрированной среды разработки **Delphi**.

Настоящие указания подготовлены для студентов, обучающихся по направлениям «Управление в технических системах» и «Прикладная математика и информатика», и посвящены получению навыков алгоритмизации и изучению принципов программирования в среде **Delphi**.

## ЛАБОРАТОРНЫЕ РАБОТЫ № 1, 2 РАБОТА С ГРАФИКОЙ

### 1. Цель работы

Целью работы является изучение графических возможностей **Delphi**.

### 2. Графические возможности Delphi

Богатство изобразительных возможностей **Windows** связано с так называемым дескриптором контекста графического устройства (**Device Context**) и тремя входящими в него инструментами – шрифтом, пером и кистью. В **Delphi** созданы специализированные классы-надстройки, существенно упрощающие использование графических инструментов **Windows**: для контекста – класс **TCanvas**, для шрифта – **TFont**, для пера – **TPen**, для кисти – **TBrush**.

Связанные с этими классами объекты автоматически создаются для всех видимых элементов и становятся доступны программе через их свойства **Canvas**, **Font**, **Pen** и **Brush**.

### 3. Класс TCanvas – канва (холст)

Наиболее богатые графические возможности реализованы в классе **TCanvas**. Этот класс называют канвой. Канва не является компонентом, но она входит в качестве свойства в те из них, которые умеют рисовать себя и отображать какую-либо информацию. На канве компонента можно рисовать геометрические фигуры, тексты, составлять из отдельных точек различные узоры и отображать растровые изображения. Свойство **Canvas** имеют многие компоненты и форма в том числе. Для рисования канва использует свойства: шрифт **Font**, перо **Pen** и кисть **Brush** (табл. 1), а также многочисленные методы классов **TCanvas**, **TFont**, **TPen**, **TBrush**.

Методы вывода графических примитивов рассматривают свойство **Canvas** как некоторый абстрактный холст, на котором они могут рисовать (*canvas* переводится как «поверхность», «холст для рисования»). Холст состоит из отдельных точек – пикселей. Положение пикселя характеризуется его горизонтальной (**X**) и вертикальной (**Y**) координатами. Левый верхний пиксель имеет координаты **(0, 0)**. Координаты возрастают сверху вниз и слева направо. Значения координат правой нижней точки холста зависят от размера холста.

Таблица 1  
Свойства канвы

Имя свойства	Тип значения	Описание
<b>Brush</b>	<b>TBrush</b>	Объект-кисть
<b>Pen</b>	<b>TPen</b>	Объект-перо
<b>Font</b>	<b>TFont</b>	Объект-шрифт
<b>Handle</b>	<b>Integer</b>	Дескриптор канвы. Используется при непосредственном обращении к API-функциям <b>Windows</b>
<b>ClipRect</b>	<b>TRect</b>	Определяет текущие размеры области, нуждающейся в прорисовке
<b>PenPos</b>	<b>TPoint</b>	Определяет текущее положение пера в пикселях относительно левого верхнего угла канвы
<b>Pixels</b> [X,Y:Integer]	<b>TColor</b>	Массив пикселей канвы. С помощью этого свойства все пиксели канвы представляются в виде двухмерного массива точек. Изменяя цвет пикселей, можно прорисовывать изображение по отдельным точкам
<b>CopyMode</b>	<b>TCopyMode</b>	Используется при копировании части одной канвы (источника) в другую (приемник) методом <b>CopyRect</b>

Свойство **CopyMode** используется при копировании части одной канвы (источника) в другую (приемник) методом **CopyRect** и может иметь одно из следующих значений (табл. 2).

Таблица 2  
Значения свойства CopyMode

Значение	Описание
<b>cmBlackness</b>	Заполняет область рисования черным цветом
<b>cmDestInvert</b>	Заполняет область рисования инверсным цветом фона
<b>cmMergeCopy</b>	Объединяет изображение на канве и копируемое изображение операцией AND

<b>cmMergePaint</b>	Объединяет изображение на канве и копируемое изображение операцией OR
<b>cmNotSrcCopy</b>	Копирует на канву инверсное изображение источника
<b>cmNotSrcErase</b>	Объединяет изображение на канве и копируемое изображение операцией OR и инвертирует полученное
<b>cmPatCopy</b>	Копирует образец источника
<b>cmPatInvert</b>	Комбинирует образец источника с изображением на канве с помощью операции XOR
<b>cmPatPaint</b>	Комбинирует изображение источника с его образцом с помощью операции OR, затем полученное объединяется с изображением на канве также с помощью OR
<b>cmSrcAnd</b>	Объединяет изображение источника и канвы с помощью операции AND
<b>cmSrcCopy</b>	Копирует изображение источника на канву
<b>cmSrcErase</b>	Инвертирует изображение на канве и объединяет результат с изображением источника операцией AND
<b>cmSrcInvert</b>	Объединяет изображение на канве и источник операцией XOR
<b>cmSrcPaint</b>	Объединяет изображение на канве и источник операцией OR
<b>cmWhitnness</b>	Заполняет область рисования белым цветом

### Методы класса TCanvas

**procedure Draw(X, Y: Integer; Graphic: TGraphic);**

Осуществляет прорисовку графического объекта **Graphic** так, чтобы левый верхний угол объекта расположился в точке (X, Y).

**procedure BrushCopy(const Dest: TRect; Bitmap: TBitmap; const Source: TRect; Color: TColor);**

Копирует часть изображения **Source** на участок канвы **Dest**. Значение **Color** задает цвет в участке канвы **Dest**, который должен заме-

няться на цвет кисти канвы. Метод введен для совместимости с ранними версиями **Delphi**.

```
procedure DrawFocusRect(const Rect: TRect);
```

Прорисовывает прямоугольник с помощью операции XOR, поэтому повторная прорисовка уничтожает ранее вычерченный прямоугольник. Используется в основном для прорисовки нестандартных интерфейсных элементов при получении ими фокуса ввода.

```
procedure CopyRect(Dest: TRect; Canvas: TCanvas; Source: TRect);
```

Копирует изображение **Source** канвы **Canvas** в участок **Dest** текущей канвы. При этом разнообразные специальные эффекты достигаются с помощью свойства **CopyMode**.

```
procedure FrameRect(const Rect: TRect);
```

Очерчивает границы прямоугольника **Rect** текущей кистью толщиной в 1 пиксель без заполнения внутренней части прямоугольника.

```
procedure FillRect(const Rect: TRect);
```

Заполняет текущей кистью прямоугольную область **Rect**, включая ее левую и верхнюю границы, но не затрагивая правую и нижнюю границы.

```
procedure FloodFill(X, Y: Integer; Color: TColor; FillStyle: TFillStyle);
```

Производит заливку канвы текущей кистью. Заливка начинается с точки **(X, Y)** и распространяется во все стороны от нее. Если **FillStyle=fsSurface**, заливка распространяется на все соседние точки с цветом **Color**. Если **FillStyle=fsBorder**, наоборот, заливка прекращается на точках с этим цветом.

```
procedure Ellipse(X1, Y1, X2, Y2: Integer);
```

Чертит эллипс в охватывающем прямоугольнике с координатами **(X1, Y1) - (X2, Y2)**. Заполняет внутреннее пространство эллипса текущей кистью. Пример рисования эллипса приведён на рис. 1, а.

```
procedure Pie(x1, y1, x2, y2, x3, y3, x4, y4:  
Longint) ;
```

Рисует сектор эллипса в охватывающем прямоугольнике с координатами (**X1**, **Y1**) - (**X2**, **Y2**). Начало дуги лежит на пересечении эллипса и луча, проведенного из его центра в точку (**X3**, **Y3**), а конец – на пересечении эллипса с лучом, проведенным из центра в точку (**X4**, **Y4**). Дуга чертится против часовой стрелки. Начало и конец дуги соединяются прямыми с ее центром.

```
procedure Arc(x1,y1,x2,y2,x3,y3,x4,y4: Integer) ;
```

Чертит дугу эллипса в охватывающем прямоугольнике (**X1**,**Y1**)-(**X2**,**Y2**). Начало дуги лежит на пересечении эллипса и луча, проведенного из его центра в точку (**X3**, **Y3**), а конец – на пересечении с лучом из центра в точку (**X4**, **Y4**). Дуга чертится против часовой стрелки.

```
procedure Chord(x1, y1, x2, y2, x3, y3, x4, y4:  
Integer) ;
```

Чертит сегмент эллипса в охватывающем прямоугольнике с координатами левого верхнего угла (**X1**, **Y1**) и координатами правого нижнего угла (**X2**, **Y2**). Начало дуги сегмента лежит на пересечении эллипса и луча, проведенного из его центра в точку (**X3**, **Y3**), а конец – на пересечении с лучом из центра в точку (**X4**, **Y4**). Дуга сегмента чертится против часовой стрелки, а начальная и конечная точки дуги соединяются прямой. Пример рисования дуги приведён на рис. 1, б.



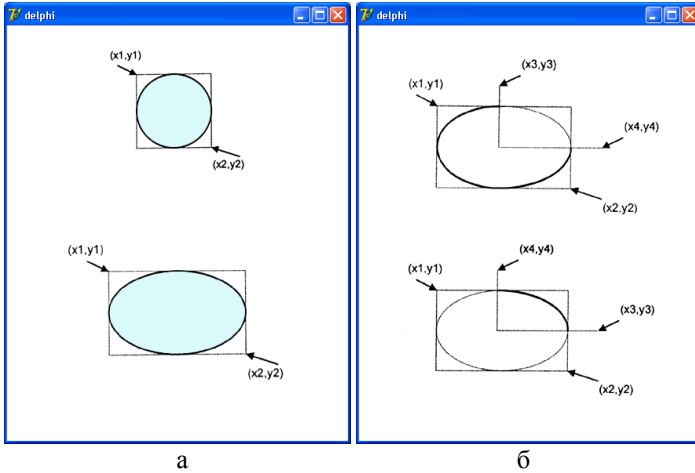


Рис. 1. Пример рисования эллипса (а) и дуги (б)

**procedure MoveTo(X, Y: Integer);**

Перемещает перо в положение (X, Y) без вычерчивания линий.

**procedure LineTo(X, Y: Integer);**

Чертит линию от текущего положения пера до точки (X, Y).

**procedure Polygon(Points: Array of TPoint);**

Вычерчивает пером многоугольник по точкам, заданным в массиве **Points**. Конечная точка соединяется с начальной, и многоугольник заполняется кистью. Без заполнения многоугольника используют процедуру

**procedure Polyline(Points: Array of TPoint);**

Вычерчивает ломаную прямую по точкам, заданным в массиве **Points**.

**procedure Rectangle(X1, Y1, X2, Y2: Integer);**

Вычерчивает и заполняет прямоугольник **(X1, Y1) - (X2, Y2)**. Для вычерчивания без заполнения используются процедуры **FrameRect** или **Polyline**.

```
procedure Refresh;
```

Устанавливает в канве шрифт и кисть по умолчанию.

```
procedure RoundRect (X1, Y1, X2, Y2, X3, Y3: Integer);
```

Вычерчивает и заполняет прямоугольник с координатами **(X1, Y1) - (X2, Y2)** со скругленными углами.

Прямоугольник **(X1, Y1) - (X3, Y3)** определяет дугу эллипса для округления углов.

```
procedure StretchDraw(const Rect: TRect; Graphic: TGraphic);
```

Вычерчивает и при необходимости масштабирует графический объект **Graphic** так, чтобы он полностью занял прямоугольник **Rect**.

```
procedure TextOut(X, Y: Integer; const Text: String);
```

Выводит текстовую строку **Text** так, чтобы левый верхний угол прямоугольника, охватывающего текст, располагался в точке **(X, Y)**.

```
procedure TextRect(Rect: TRect; X, Y: Integer; const Text: String);
```

Выводит текстовую строку **Text** так, чтобы левый верхний угол прямоугольника, охватывающего текст, располагался в точке **(X, Y)**. Если при этом какая-либо часть надписи выходит за границы прямоугольника **Rect**, то она отсекается и не будет видна.

```
function TextExtent(const Text: String): TSize;
```

Возвращает ширину и высоту прямоугольника, охватывающего текстовую строку **Text**.

```
function TextHeight(const Text: String): Integer;
```

Возвращает высоту прямоугольника, охватывающего текстовую строку **Text**.

```
function TextWidth(const Text: String): Integer;
```

Возвращает ширину прямоугольника, охватывающего текстовую строку **Text**.

#### 4. Класс **TFont** – шрифт

С помощью класса **TFont** создается объект-шрифт для любого графического устройства (экрана, принтера, плоттера и т.п.). Его основные свойства приведены в табл. 3.

Таблица 3  
Свойства шрифта

Имя свойства	Тип значения	Описание
<b>Color</b>	<b>TColor</b>	Цвет шрифта
<b>Charset</b>	<b>TFontCharSet</b>	Набор символов. Для русскоязычных программ это свойство обычно имеет значение <b>DEFAULTCHARSET</b> или <b>russian charset</b> . Используйте значение <b>OEMCHARSET</b> для отображения текста <b>MS-DOS</b> (альтернативная кодировка)
<b>FontAdapter</b>	<b>IChangeNotifier</b>	Поставляет информацию о шрифте в компоненты <b>ActiveX</b>
<b>Handle</b>	<b>Integer</b>	Дескриптор шрифта. Используется при непосредственном обращении к API-функциям <b>Windows</b>
<b>Height</b>	<b>Integer</b>	Высота шрифта в пикселях экрана

<b>Name</b>	<b>TFontName</b>	Имя шрифта. По умолчанию имеет значение MS Sans Serif
<b>Pitch</b>	<b>TFontPitch</b>	Определяет способ расположения букв в тексте: значение <b>fpFixed</b> задает моноширинный текст, при котором каждая буква имеет одинаковую ширину; значение <b>fpVariabel</b> определяет пропорциональный текст, при котором ширина буквы зависит от ее начертания; <b>fpDefault</b> определяет ширину, принятую для текущего шрифта
<b>Style</b>	<b>TFontStyles</b>	Стиль шрифта. Может принимать значение как комбинация следующих признаков: <b>fsBold</b> (жирный), <b>fsItalic</b> (курсив), <b>fsUnderline</b> (подчеркнутый), <b>fsStrikeOut</b> (перечеркнутый)
<b>Size</b>	<b>Integer</b>	Высота шрифта в пунктах (1/72 дюйма). Изменение этого свойства автоматически изменяет свойство <b>Height</b> и наоборот

### 5. Класс **TPen** – перо

С помощью класса **TPen** создается объект-перо, служащий для вычерчивания линий. Его основные свойства приведены в табл. 4.

Таблица 4  
Свойства пера

Имя свойства	Тип значения	Описание
<b>Color</b>	<b>TColor</b>	Цвет вычерчиваемых пером линий
<b>Handle</b>	<b>Integer</b>	Дескриптор пера. Используется при непосредственном обращении к API-функциям <b>Windows</b>
<b>Mode</b>	<b>TPenMode</b>	Определяет способ взаимодействия линий с фоном
<b>Width</b>	<b>Integer</b>	Толщина линий в пикселях экрана

Style	TPenStyle	Определяет стиль линий (рис. 2). Учитывается только для толщины линий 1 пиксель. Для толстых линий стиль всегда <b>psSolid</b> (сплошная)
-------	-----------	---

Линии с разным значением свойства **Style** показаны на рис. 2.

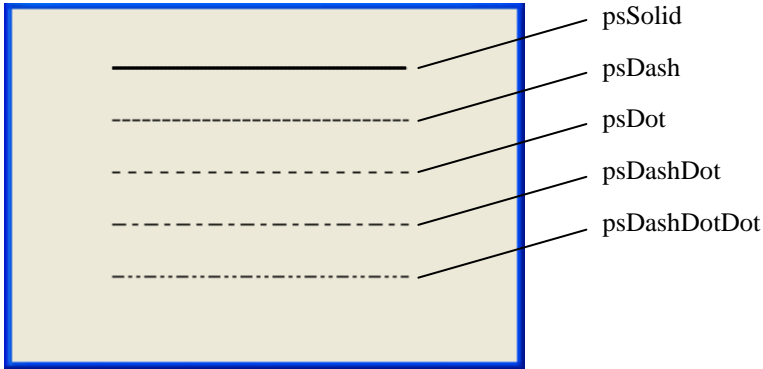


Рис. 2. Линии с разным значением свойства Style

Свойство **Mode** может принимать одно из следующих значений (табл. 5).

Таблица 5  
Значения свойства Mode

Значение	Описание
<b>pmBlack</b>	Линии всегда черные. Свойства <b>Color</b> и <b>Style</b> игнорируются
<b>pmWhite</b>	Линии всегда белые. Свойства <b>Color</b> и <b>Style</b> игнорируются
<b>pmNop</b>	Цвет фона не меняется (линии не видны)
<b>pmNot</b>	Инверсия цвета фона. Свойства <b>Color</b> и <b>Style</b> игнорируются
<b>pmCopy</b>	Цвет линий определяется свойством <b>Color</b> пера
<b>pmNotCopy</b>	Инверсия цвета пера. Свойство <b>Style</b> игнорируется

<b>pmMergePenNot</b>	Комбинация цвета пера и инверсионного цвета фона
<b>pmMaskPenNot</b>	Комбинация общих цветов для пера и инверсионного цвета фона. Свойство <b>Style</b> игнорируется
<b>pmMergeNotPen</b>	Комбинация инверсионного цвета пера и фона
<b>pmMaskNotPen</b>	Комбинация общих цветов для инверсионного цвета пера и фона. Свойство <b>Style</b> игнорируется
<b>pmMerge</b>	Комбинация цветов пера и фона
<b>pmNotMerge</b>	Инверсия цветов пера и фона. Свойство <b>Style</b> игнорируется
<b>pmMask</b>	Общие цвета пера и фона
<b>pmNotMask</b>	Инверсия общих цветов пера и фона
<b>pmNotXor</b>	Инверсия объединения цветов пера и фона операций XOR
<b>pmXor</b>	Объединение цветов пера и фона операций XOR

## 6. Класс TBrush – кисть

Объекты класса **TBrush** служат для заполнения внутреннего пространства замкнутых фигур. Некоторые свойства этого класса приведены в табл. 6.

Таблица 6  
Свойства кисти

Имя свойства	Тип значения	Описание
<b>Bitmap</b>	<b>TBitmap</b>	Содержит растровое изображение, которое будет использоваться кистью для заполнения. Если это свойство определено, свойства <b>Color</b> и <b>Style</b> игнорируются
<b>Color</b>	<b>TColor</b>	Цвет кисти
<b>Handle</b>	<b>Integer</b>	Дескриптор кисти. Используется при обращении к API-функциям <b>Windows</b>
<b>Style</b>	<b>TBrushStyle</b>	Определяет стиль кисти (рис. 3)

Свойство **Color** может принимать любые значения цвета из пространства RGB, задаваемого шестнадцатеричным числом в следующем виде:

**Color1:=\$00BBGGRR;**

где **BB** – интенсивность синей компоненты цвета, **GG** – интенсивность зелёной компоненты цвета, а **RR** – интенсивность красной компоненты цвета. То есть для того, чтобы получить, например, чистый красный цвет, необходимо задать следующее значение:

**MyColor:=\$000000FF; //задан красный цвет**

Также возможно задавать значение цвета, используя константы, приведенные в табл. 7.

Таблица 7  
Константы, содержащие значение цвета

Значение константы	Цвет
<b>clBlack</b>	Черный
<b>clSilver</b>	Серебристый
<b>clMaroon</b>	Каштановый
<b>clRed</b>	Красный
<b>clGreen</b>	Зеленый
<b>clLime</b>	Салатный
<b>clOlive</b>	Оливковый
<b>clBlue</b>	Синий
<b>clNavy</b>	Темно-синий
<b>clFuchsia</b>	Ярко-розовый
<b>clPurple</b>	Розовый
<b>clAqua</b>	Бирюзовый
<b>clTeal</b>	Зелено-голубой
<b>clWhite</b>	Белый
<b>clYellow</b>	Желтый
<b>clGray</b>	Серый

То есть для того, чтобы получить чистый красный цвет, необходимо задать следующее значение:

**MyColor:=clRed; //задан красный цвет**

Примеры стилей заполнения приведены на рис. 3.

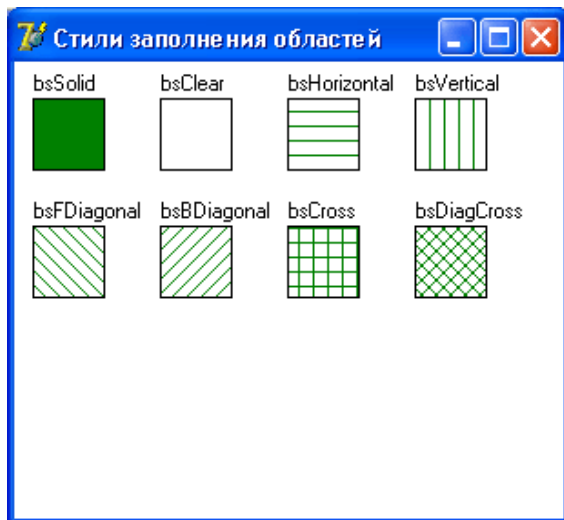


Рис. 3. Заполнение квадратов кистью с разными значениями свойства Style

### 7. Порядок выполнения работы № 1

1. Загрузить среду **Delphi**.
2. Создать новый проект.
3. Создать новую форму (форма отображения графики).
4. Поместить на форму компонент «кнопка».
5. Создать обработчик нажатия на кнопку, в котором организовать отображение схемы задачи в момент времени  $t=0$ . На схеме вывести обозначения необходимых параметров (время, координаты, скорость и т.п.), а также необходимые пояснения.

**Задание к лабораторной работе уточняется у преподавателя.**

### 8. Порядок выполнения работы № 2

1. Загрузить среду **Delphi**.
2. Загрузить проект, созданный на предыдущей работе.
3. Поместить на форму дополнительный компонент «кнопка».
4. Создать обработчик нажатия на кнопку, в котором организовать отображение схемы задачи в моменты времени  $t = \overline{0, n}$ . На схеме



вывести обозначения для параметров (физических величин), которые изменяются со временем.

## 9. Пример выполнения лабораторной работы

### Задание

Составить программу решения квадратных уравнений вида  $X^2 + BX + C = 0$  и  $AX^2 + BX + C = 0$ . По введённым коэффициентам  $A$ ,  $B$  и  $C$  рассчитать корни  $X_1$  и  $X_2$ . Учесть, что во втором случае  $A \neq 0$ .

### Задание к данной работе

Нарисовать схему задачи: отобразить оси координат, параболу, обозначить точки пересечения параболы с осями координат. Заштриховать область между параболой и осью в промежутке между корнями.

### Выполнение задания

По порядку выполняются следующие действия.

1. Загружается **Delphi**.
2. Создаём новый проект.
3. На форму (**Form1**) помещается компонент «кнопка».
4. Там же создаётся обработчик нажатия кнопки «Нарисовать», в котором:
  - а) на канве формы рисуется поле для рисования (белый прямоугольник) размерами 400x300 точек, расположенный в левом верхнем углу формы;
  - б) строится система координат, расставляются подписи по обеим осям;
  - в) строится парабола;
  - г) отмечаются корни.
5. Выбирается точка левее и ниже первого корня, из которой начинается выполнение штриховки. Точно так же выбирается точка правее и ниже второго корня. Выполняется нужная штриховка.

Теперь при щелчке по кнопке рисуются оси координат, парабола, отмечаются корни и отображается штриховка (рис. 4).

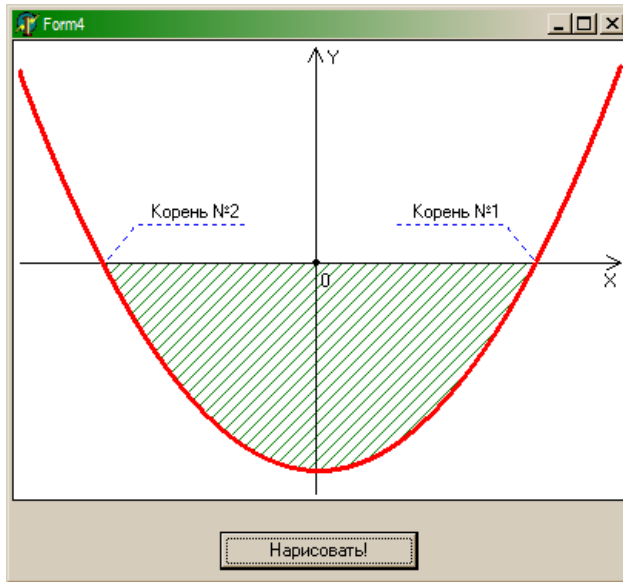


Рис. 4. Пример работы программы с построенной параболой

Код модуля **unit1** представлен ниже:

```
unit Unit1;

interface

uses Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
```

```

implementation
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
var
  i   : integer; //Счётчик
  x,y : real;     //Координаты точки
begin
  //Рисование границ рисунка и поле для рисования
  Form1.Canvas.Brush.Color:=clWhite;
  Form1.Canvas.Pen.Color:=clBlack;
  Form1.Canvas.Pen.Width:=1;
  Form1.Canvas.Pen.Style:=psSolid;
  Form1.Canvas.Rectangle(0,0,400,300);
  //Рисуем и подписываем оси
  //Ось OX
  Form1.Canvas.MoveTo(5,145);
  Form1.Canvas.LineTo(395,145);
  Form1.Canvas.LineTo(385,140);
  Form1.Canvas.MoveTo(395,145);
  Form1.Canvas.LineTo(385,150);
  Form1.Canvas.TextOut(385,150,'X');
  //Ось OY
  Form1.Canvas.MoveTo(197,295);
  Form1.Canvas.LineTo(197,5);
  Form1.Canvas.LineTo(192,15);
  Form1.Canvas.MoveTo(197,5);
  Form1.Canvas.LineTo(202,15);
  Form1.Canvas.TextOut(205,5,'Y');
  //Точка (0,0)
  Form1.Canvas.TextOut(200,150,'0');
  Form1.Canvas.Brush.Color:=clBlack;
  Form1.Canvas.Ellipse(195,143,200,147);
  //Рисуем параболу
  Form1.Canvas.MoveTo(5,20);
  Form1.Canvas.Pen.Color:=clRed;
  Form1.Canvas.Pen.Width:=3;
  for i:=5 to 395 do
  begin
    x:=(i-199)/195;
    y:=280-130*2*sqr(x);
    Form1.Canvas.LineTo(i,round(y));
  end;
  //Обозначаем корни
  Form1.Canvas.Pen.Style:=psDot;
  Form1.Canvas.Brush.Color:=clWhite;
  Form1.Canvas.Pen.Color:=clBlue;
  Form1.Canvas.Pen.Width:=1;

```

```
Form1.Canvas.MoveTo(250,120);
Form1.Canvas.LineTo(320,120);
Form1.Canvas.LineTo(340,145);
Form1.Canvas.TextOut(260,105,'Корень №1');
Form1.Canvas.MoveTo(150,120);
Form1.Canvas.LineTo(80,120);
Form1.Canvas.LineTo(60,145);
Form1.Canvas.TextOut(90,105,'Корень №2');

//Выполняем штриховку
Form1.Canvas.Brush.Color:=clGreen;
Form1.Canvas.Brush.Style:=bsBDiagonal;
Form1.Canvas.FloodFill(220,165,clWhite,fsSurface);
Form1.Canvas.FloodFill(190,165,clWhite,fsSurface);
end;
end.
```

## ЛАБОРАТОРНАЯ РАБОТА № 3 РАБОТА С ДАТОЙ И ВРЕМЕНЕМ

### 1. Цель работы

Целью работы является изучение типа «дата-время» в **Delphi**, компонента «таймер», получение навыков работы с датой и временем и их отображения.

### 2. Тип дата-время

Тип дата-время определяется стандартным идентификатором **TDateTime** и предназначен для одновременного хранения и даты, и времени. Во внутреннем представлении он занимает 8 байт и является вещественным числом с фиксированной запятой. В целой части числа хранится дата, а в дробной – время.

Дата определяется как количество суток, прошедших с 30 декабря 1899 года, а время – как часть суток, прошедших с 0 часов. Например, значение 36444,837 соответствует дате 11.10.1999 и времени 20:05. Количество суток может быть и отрицательным, однако значения меньше -693594 (соответствует дате 00.00.0000 от Рождества Христова) игнорируются функциями преобразования даты к строковому типу.

Над данными типа **TDateTime** определены те же операции, что и над вещественными числами, а в выражениях этого типа могут участвовать константы и переменные целого и вещественного типов.

Для работы с датой и временем используются подпрограммы, перечисленные в табл. 1.

Таблица 1  
Функции для работы с датой и временем

Подпрограмма	Назначение
<b>function Now:</b> <b>TDateTime;</b>	Возвращает текущую дату и время
<b>function Date:</b> <b>TDateTime;</b>	Возвращает текущую дату
<b>function Time:</b> <b>TDateTime;</b>	Возвращает текущее время
<b>function DateTimeToStr</b> <b>(D:TDateTime): String;</b>	Преобразует дату и время в строку символов
<b>function DateToStr</b> <b>(D:TDateTime): String;</b>	Преобразует дату в строку символов

Окончание таблицы 1

<b>function TimeToStr (T:TDateTime): String;</b>	Преобразует время в строку символов
<b>function FormatDateTime (Format: String; Value: TDateTime): String;</b>	Преобразует дату и время из параметра <b>value</b> в строку символов в соответствии со спецификаторами параметра <b>Format</b> (см. табл. 2)

В табл. 2 приведены различные спецификаторы формата даты/времени для использования при вызове функции **FormatDateTime**.

Таблица 2

Спецификаторы формата даты/времени

Спецификатор	Назначение
<b>c</b>	Отображает сначала дату в формате дд.мм.гг, затем пробел и время в формате чч:мм: 07.05.17 09:03
<b>d</b>	Отображает день без ведущего нуля: 7
<b>dd</b>	Отображает день с ведущим нулем: 07
<b>dddd</b>	Отображает день недели: <i>воскресенье</i> (для нерусифицированной версии Windows – <i>sunday</i> )
<b>dddddd</b>	Отображает дату в формате дд.мм.гг: 07.05.17
<b>ddddddd</b>	Отображает дату в формате д Месяц год: 7 Май 2017 (для нерусифицированной версии Windows – 7 May 2017)
<b>m</b>	Отображает число месяца без ведущего нуля: 5
<b>mm</b>	Отображает число месяца с ведущим нулем: 05
<b>mmm</b>	Отображает сокращенное название месяца: <i>анр.</i>
<b>mmmm</b>	Отображает полное название месяца: <i>Май</i>
<b>y</b> или <b>yy</b>	Отображает две последние цифры года: 17
<b>yyy</b> или <b>yyyy</b>	Отображает все цифры года: 2017
<b>h</b>	Отображает час без ведущего нуля: 9
<b>hh</b>	Отображает час с ведущим нулем: 09
<b>n</b>	Отображает минуты без ведущего нуля: 3
<b>nn</b>	Отображает минуты с ведущим нулем: 03
<b>s</b>	Отображает секунды без ведущего нуля: 0
<b>ss</b>	Отображает секунды с ведущим нулем: 00
<b>t</b>	Отображает время в формате чч:мм: 09:03
<b>tt</b>	Отображает время в формате чч:мм:сс: 09:03:00

Окончание таблицы 2

<b>am/pm</b>	Отображает время в 12-часовом формате ( <b>am</b> – до полудня, <b>pm</b> – после полудня). Для спецификаторов <b>hh:mm am/pm</b> получим: <i>09:03 am</i>
<b>ampm</b>	Отображает время в 12-часовом формате, но без указания до/после полудня. Для спецификаторов <b>hh:mm ampm</b> получим: <i>09:03</i>
<b>a/p</b>	Отображает время в 12-часовом формате ( <b>a</b> – до полудня, <b>p</b> – после полудня). Для спецификаторов <b>hh:mm a/p</b> получим: <i>09:03 a</i>
<b>/</b>	Отображает используемый в Windows разделитель даты. Для спецификаторов <b>d/m/y</b> получим: <i>7/5/17</i>
<b>:</b>	Отображает используемый в Windows разделитель времени. Для спецификаторов <b>h:n:s</b> получим: <i>9:3:0</i>

Любые другие символы, указанные в строке **Format**, а также заключенные в апострофы или кавычки, помещаются в выходную строку без преобразования, поэтому спецификаторы **h час n мин** дадут строку *9 час 3 мин*, а **h час "n" мин** – *9 час n мин*.

Поскольку тип **TDateTime** совместим с форматом вещественных чисел, можно без труда определить дату, отстоящую от заданной на сколько-то дней вперед или назад. Для этого достаточно прибавить к заданной дате или отнять от нее нужное целое число. Например, оператор

```
Label1.Caption := DateToStr (Date () + 21) ;
```

поместит в метку **Label1** дату, соответствующую текущей дате плюс 3 недели (21 день). Чуть сложнее с исчислением времени. Например, чтобы добавить к текущему времени полтора часа, следует использовать выражение

```
Time () + StrToTime ('1:30')
```

или

```
Time () + 1.5/24
```

### 3. Компонент **TTimer**

Компонент **TTimer** (таймер) служит для отсчета интервалов реального времени. Его свойство **interval** определяет интервал времени в миллисекундах, который должен пройти от включения таймера до наступления события **onTimer**. Таймер включается при установке значения **True** в его свойство **Enabled**. Включенный таймер все время будет создавать события **onTimer** до тех пор, пока его свойство **Enabled** не примет значения **False**. При добавлении компонента таймер на форму необходимо учитывать, что по умолчанию свойство **Enabled=True**.

Чтобы каждые **interval** миллисекунд выполнялись требуемые действия, необходимо создать процедуру-обработчик события **onTimer** (вкладка **Events**).

Следует отметить, что в силу специфики реализации системного таймера персонального IBM-совместимого компьютера под управлением ОС Windows версии, более ранней, чем NT, минимальный реально достижимый интервал отсчета времени не может быть меньше 55 миллисекунд. Более того, любой интервал времени, отсчитываемый с помощью таймера, всегда кратен 55 миллисекундам. В Windows NT компонент **TTimer** уже обеспечивает точность 10 миллисекунд.

### 4. Порядок выполнения работы

Создать в среде **Delphi** новое приложение, обеспечивающее выполнение следующих действий.

1. Вывод текущего времени в числовом формате. При этом вывод значения часов, минут и секунд должен производиться в разные компоненты.
2. Вывод текущей даты в числовом виде, вывод дня недели в строковом виде. При этом вывод значения года, месяца, числа и дня недели должен производиться в разные компоненты.
3. Вывод текущего времени, имитирующий циферблат стрелочных часов.
4. Должна присутствовать функция будильника, срабатывающего в заданное пользователем время (часы и минуты).

Задание требуемого времени может осуществляться, например, с помощью компонентов **TEdit**.

При срабатывании будильника приложение выполняет произвольное действие (например, показывает какой-либо компонент).



Необходимо наличие возможности остановки/выключения будильника.

## 5. Пример выполнения лабораторной работы

Размещаем на форме необходимые компоненты. Для вывода времени в числовом формате на форму помещаются три метки (рис. 1). Для каждой из них изменяются свойства **color**, **font.name**, **font.size** и **caption**.



Рис. 1. Пример размещения компонентов для отображения времени

Для вывода на экран даты на форму помещаются ещё четыре метки (рис. 2). Для каждой из них изменяются также свойства **color**, **font.name**, **font.size** и **caption**.

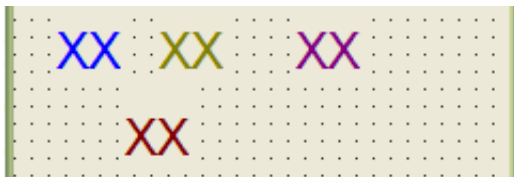


Рис. 2. Пример размещения компонентов для отображения даты

Для обеспечения работы будильника на форму помещаются два компонента для выбора времени активации будильника (**Edit** или **SpinEdit**), несколько меток и переключатель (**Checkbox**), показывающий, включен будильник или нет (рис. 3).

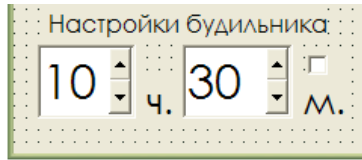



Рис. 3. Пример размещения компонентов для настройки будильника

Для индикации работы будильника и возможности его дезактивации поместим на форму кнопку и придадим её свойству **visible** значение **false**. Соответственно при включении будильника для свойства **visible** необходимо установить значение **true**.

Для обеспечения обновления показаний времени на форме помещаем на неё компонент **Timer** , расположенный на вкладке **System** палитры компонентов. В итоге главная форма приложения примет следующий вид (рис. 4).

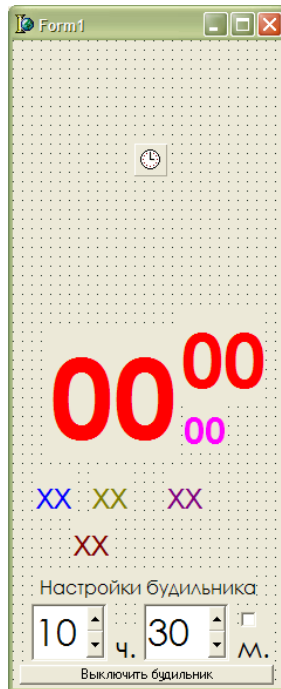


Рис. 4. Пример размещения компонентов на форме приложения

После окончания проектирования формы необходимо с помощью инспектора объектов создать обработчик события **Timer.OnTimer** и установить интервал его вызова (свойство **interval**) равным 1000 мс.

В процедуре-обработчике **TForm1.Timer1Timer** необходимо прописать следующие действия.

1. Получение текущей даты и времени.
2. Преобразование их в строковый тип для отображения и в числовой тип для обеспечения работы будильника.
3. Отображение текущей даты и времени.
4. Считывание параметров будильника (время активации – часы и минуты, флаг активации).
5. Проверка условия активации.

При этом окно работающего приложения будет выглядеть следующим образом (рис. 5).



а – будильник не активен

б – будильник активен

Рис. 5. Варианты окна работающего приложения

Для кнопки «Выключить будильник» необходимо создать процедуру-обработчик события **OnClick**, в которой обеспечивается отключение будильника.

Для рисования циферблата часов будем использовать канву формы. Соответственно в каждый момент времени необходимо в процедуре **TForm1.Timer1Timer** прописать следующие действия.

1. Рисование циферблата часов.
2. Рисование стрелок часов.

Для того чтобы нарисовать точки, расположенные на окружности радиусом  $R$ , применим следующие формулы (см. рис. 6 для пояснений):

$$\begin{cases} x = x_c + R \cdot \cos(\alpha), \\ y = y_c + R \cdot \sin(\alpha). \end{cases} \quad (1)$$

При этом легко понять, что метки, цифры и концы стрелок будут лежать на концентрических окружностях разного радиуса.

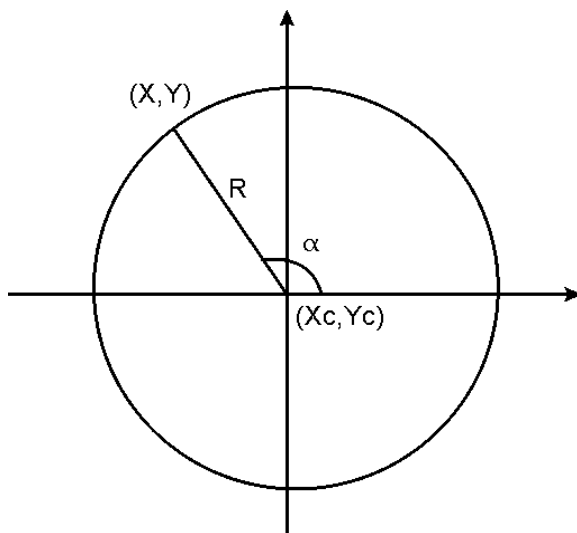


Рис. 6. Положение точки на окружности в зависимости от угла

Работающее приложение будет иметь следующий вид (рис. 7).

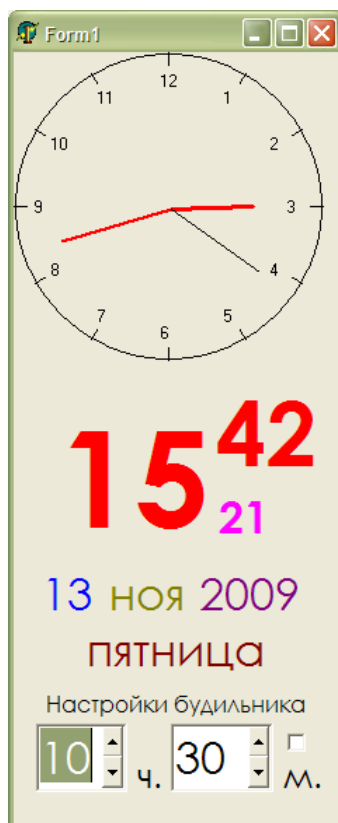


Рис. 7. Вид работающего приложения

## ЛАБОРАТОРНАЯ РАБОТА № 4 ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ СОРТИРОВКИ В СРЕДЕ DELPHI

### 1. Цель работы

Получение опыта практической реализации алгоритмов сортировки одномерных массивов.

### 2. Методы сортировки

Сортировка и поиск являются наиболее общими составными частями систем программирования.

**Сортировка** – это распределение элементов некоторого множества по группам в соответствии с определенными правилами.

**Поиск** – это выполнение совокупности операций с целью определить наличие или отсутствие заданных элементов конкретного типа в некоторой совокупности элементов этого типа.

Среди задач сортировки важное место принадлежит задаче упорядочения – расположению однородных объектов в определенном порядке. Это частный случай задач сортировки. Объектом для упорядочения могут быть числа или символы. Более того, можно решать задачу упорядочения векторов или матриц, если характеризовать их некоторым числом – нормой. Метод упорядочения существенно зависит от того, как представлены упорядочиваемые объекты: как элементы **массива** в оперативной памяти либо как элементы **файла** во внешней памяти. Сортировка данных в массиве называется **внутренней**, а сортировка данных в файле – **внешней**.

Пусть файл включает некоторую последовательность из  $n$  элементов  $\mathbf{r}(1), \mathbf{r}(2), \dots, \mathbf{r}(n)$ . Каждый элемент называется **записью**. С каждой записью  $\mathbf{r}(i)$  связан некоторый ключ  $\mathbf{k}(i)$ . Ключом обычно является некоторое поле внутри записи. Говорят, что **файл отсортирован по ключу**, если для  $i < j$  следует, что  $\mathbf{k}(i)$  предшествует  $\mathbf{k}(j)$  по какому-то признаку. Так, каждая строка телефонного справочника является записью, содержащей фамилию, адрес и номер телефона абонента. Ключом, по которому отсортирован этот файл, является поле фамилии.

Вполне возможно, что две записи в некотором файле имеют одинаковый ключ. Метод сортировки называется устойчивым, если для всех записей  $i$  и  $j$  таких, что  $\mathbf{k}(i) = \mathbf{k}(j)$ , выполняется условие, что в отсортированном файле  $\mathbf{r}(i)$  предшествует  $\mathbf{r}(j)$ , если  $\mathbf{r}(i)$  предшествует  $\mathbf{r}(j)$  в первоначальном файле.

Сортировка выполняется или над самими записями, или над указателями некоторой вспомогательной таблицы. Если объем данных, хранящийся в записях файла, слишком большой, перемещение самих записей может быть нецелесообразным, так как потребует больших временных затрат. В этом случае может быть использована вспомогательная таблица указателей (адресов), которые и будут перемещаться.

На рис. 1 представлен первоначальный файл, на рис. 2 – файл, отсортированный с помощью перемещения самих записей, на рис 3 – файл, отсортированный с использованием вспомогательных указателей.

	Ключ	Другие поля
Запись 1	4	DDD
Запись 2	2	BBB
Запись 3	1	AAA
Запись 4	5	EEE
Запись 5	3	CCC

Файл

Рис. 1. Первоначальный файл

1	AAA
2	BBB
3	CCC
4	DDD
5	EEE

Файл

Рис. 2. Отсортированный файл

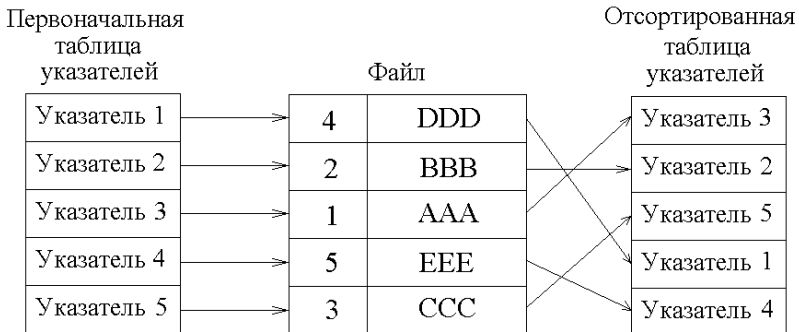


Рис. 3. Отсортированный файл (с помощью указателей)

Существует несколько методов сортировки, классификацию которых представим на рис. 4.

Простые методы

Модифицированные  
методы

Улучшенные методы

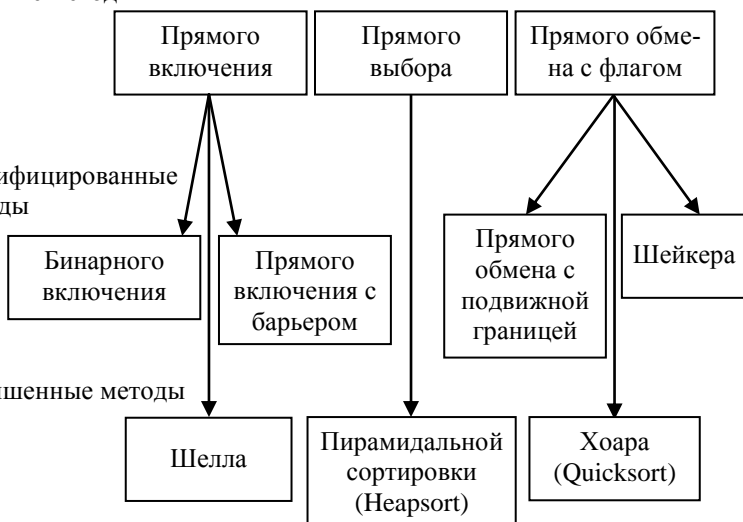


Рис. 4. Классификация методов сортировки

### 2.1. Метод прямого обмена с флагом (метод «пузырька»)

Основная идея метода – сравнивать очередные два соседних элемента файла и производить их взаимный обмен (поменять местами), если они располагаются не в том порядке. Для файла из  $N$  элементов процедура последовательного просмотра файла повторяется  $N-1$  раз.

**Пример.** Упорядочить по возрастанию следующий файл из элементов:

- $X(1) = 9$
- $X(2) = 10$
- $X(3) = 20$
- $X(4) = 3$
- $X(5) = 1$
- $X(6) = 7$
- $X(7) = 2$

Начинаем сравнивать каждую пару элементов файла от конца. На первой итерации делаются следующие шаги-сравнения:



• шаг 1.1:	7 и 2:	обмен:	9, 10, 20, 3, 1, 2, 7
• шаг 1.2:	1 и 2:	нет обмена:	9, 10, 20, 3, 1, 2, 7
• шаг 1.3:	3 и 1:	обмен:	9, 10, 20, 1, 3, 2, 7
• шаг 1.4:	20 и 1:	обмен:	9, 10, 1, 20, 3, 2, 7
• шаг 1.5:	10 и 1:	обмен:	9, 1, 10, 20, 3, 2, 7
• шаг 1.6:	9 и 1:	обмен:	1, 9, 10, 20, 3, 2, 7

Таким образом, самый маленький элемент исходного файла  $X(5)=1$  занял свое место. Если представить файл расположенным вертикально, так, что первый элемент располагается вверху, то перемещаемый минимальный элемент можно именовать всплывающим «пузырьком» – отсюда и название метода. Повторим процедуру просмотра, начав вновь с последнего элемента:

• шаг 2.1:	2 и 7:	нет обмена:	1, 9, 10, 20, 3, 2, 7
• шаг 2.2:	3 и 2:	обмен:	1, 9, 10, 20, 2, 3, 7
• шаг 2.3:	20 и 2:	обмен:	1, 9, 10, 2, 20, 3, 7
• шаг 2.4:	10 и 2:	обмен:	1, 9, 2, 10, 20, 3, 7
• шаг 2.5:	9 и 2:	обмен:	1, 2, 9, 10, 20, 3, 7
• шаг 2.6:	1 и 2:	нет обмена:	1, 2, 9, 10, 20, 3, 7

Теперь свое место занял следующий по величине элемент – 2. Еще раз повторим ту же процедуру:

• шаг 3.1:	3 и 7:	нет обмена:	1, 2, 9, 10, 20, 3, 7
• шаг 3.2:	20 и 3:	обмен:	1, 2, 9, 10, 3, 20, 7
• шаг 3.3:	10 и 3:	обмен:	1, 2, 9, 3, 10, 20, 7
• шаг 3.4:	9 и 3:	обмен:	1, 2, 3, 9, 10, 20, 7
• шаг 3.5:	2 и 3:	нет обмена:	1, 2, 3, 9, 10, 20, 7
• шаг 3.6:	1 и 2:	нет обмена:	1, 2, 3, 9, 10, 20, 7

Выполним ту же процедуру еще раз:

• шаг 4.1:	20 и 7:	обмен:	1, 2, 3, 9, 10, 7, 20
• шаг 4.2:	10 и 7:	обмен:	1, 2, 3, 9, 7, 10, 20
• шаг 4.3:	9 и 7:	обмен:	1, 2, 3, 7, 9, 10, 20
• шаг 4.4:	3 и 7:	нет обмена:	1, 2, 3, 7, 9, 10, 20
• шаг 4.5:	2 и 3:	нет обмена:	1, 2, 3, 7, 9, 10, 20
• шаг 4.6:	1 и 2:	нет обмена:	1, 2, 3, 7, 9, 10, 20

Выполним пятую итерацию:

• шаг 5.1:	10 и 20:	нет обмена:	1, 2, 3, 7, 9, 10, 20
• шаг 5.2:	9 и 10:	нет обмена:	1, 2, 3, 7, 9, 10, 20

- шаг 5.3: 7 и 9: нет обмена: 1, 2, 3, 7, 9, 10, 20
- шаг 5.4: 3 и 7: нет обмена: 1, 2, 3, 7, 9, 10, 20
- шаг 5.5: 2 и 3: нет обмена: 1, 2, 3, 7, 9, 10, 20
- шаг 5.6: 1 и 2: нет обмена: 1, 2, 3, 7, 9, 10, 20

Выполним последнюю шестую итерацию:

- шаг 6.1: 10 и 20: нет обмена: 1, 2, 3, 7, 9, 10, 20
- шаг 6.2: 9 и 10: нет обмена: 1, 2, 3, 7, 9, 10, 20
- шаг 6.3: 7 и 9: нет обмена: 1, 2, 3, 7, 9, 10, 20
- шаг 6.4: 3 и 7: нет обмена: 1, 2, 3, 7, 9, 10, 20
- шаг 6.5: 2 и 3: нет обмена: 1, 2, 3, 7, 9, 10, 20
- шаг 6.6: 1 и 2: нет обмена: 1, 2, 3, 7, 9, 10, 20

Так как после каждой итерации один из элементов занимает свое место, то, очевидно, что на следующей итерации его не следует больше проверять. Следовательно, с каждой итерацией процесс просмотра будет ускоряться. На второй итерации достаточно выполнить пять шагов, на третьей – четыре, а на последней всего один. Кроме того, хотя для полной сортировки файла из  $N$  элементов требуется  $N-1$  итерация (для нашего примера – шесть итераций), часто это происходит быстрее. В нашем примере файл оказался отсортированным уже после четвертой итерации, и последующие итерации оказались лишними. Если на какой-то итерации не было ни одной перестановки, то, очевидно, что файл оказался уже отсортированным, и последующие итерации проводить не следует. Добавив к алгоритму проверку этого условия (флага), в нашем примере можно не проводить последнюю шестую итерацию. Блок-схема этого алгоритма приведена на рис. 5.

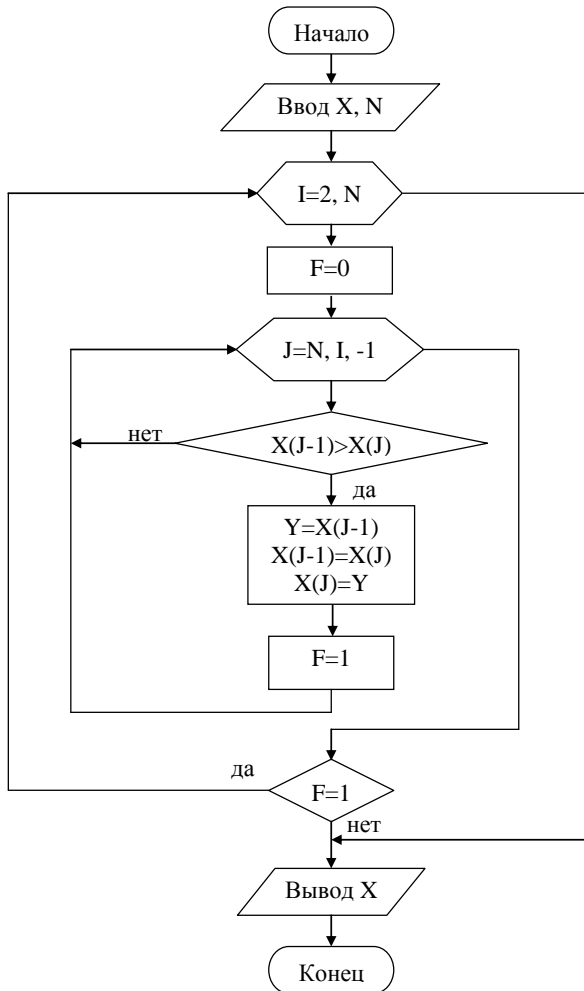


Рис. 5. Блок-схема модифицированной сортировки «пузырьком»

## 2.2. Метод прямого обмена с подвижной границей

Алгоритм сортировки методом прямого обмена с флагом можно улучшить. Пусть при очередном проходе произошел обмен элементов  $X(J-1)$  и  $X(J)$ . Обмена же для всех элементов с индексами  $J-2, J-3, \dots$  в

этом проходе не было. Но это означает, что эти элементы уже упорядочены. Элементы  $X(J-1)$  и  $X(J)$  теперь будут тоже упорядочены. Тогда после этого прохода будут упорядочены элементы  $X(1)$ ,  $X(2)$ , ...,  $X(J)$ , поэтому следующий проход можно закончить на  $(J+1)$ -м элементе, а не на  $J$ -м, как это было бы во внешнем цикле метода прямого обмена с флагом. Такова идея метода обмена с подвижной левой границей. Его блок-схема приведена на рис. 6.

Текущее значение подвижной левой границы  $L$  устанавливается после каждого цикла прохода и зависит от номера  $K$  элемента при последнем обмене в очередном проходе. Обратим внимание, что флаг теперь не нужен. Мы перед каждым очередным проходом устанавливаем значение  $K$  равным  $N$ . Тогда, если файл уже упорядочен, это значение во внутреннем цикле не меняется. Поэтому перед последним условным блоком значение  $L$  станет равным  $N+1$ , и условие продолжения внешнего цикла окажется ложным – сортировка будет закончена.

Достоинством всех вариантов метода обмена является простота алгоритма, недостаток – метод является самым медленным из всех простых методов сортировки.

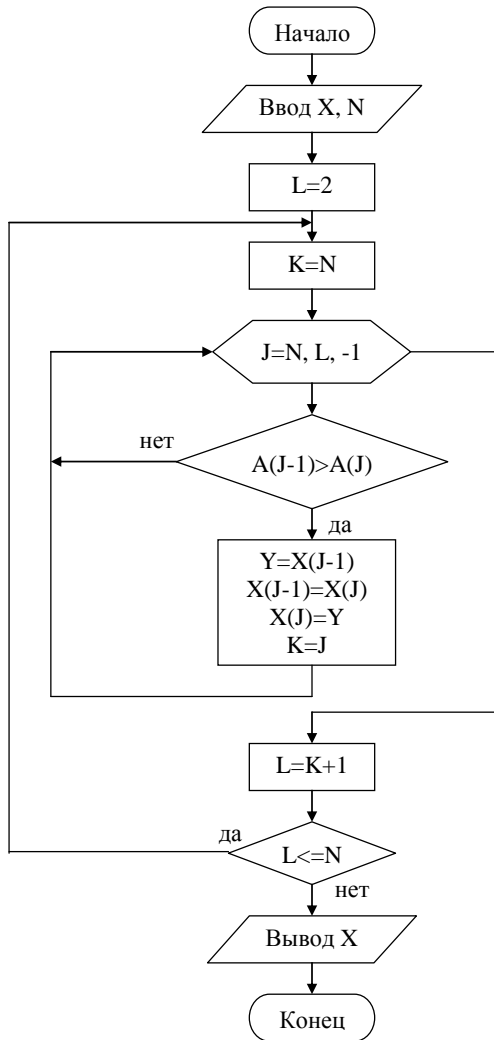


Рис. 6. Блок-схема алгоритма сортировки с подвижной границей

### 2.3. Метод шейкера

В методе прямого обмена все проходы выполняются в одном направлении – справа налево. При этом минимальный элемент сдвигает-

ся на свое место за один проход, но все другие элементы смещаются вправо только на одну позицию. Например, если в файле все элементы, кроме максимального, занимают свои позиции, а он занимает не требуемое  $N$ -е место, а  $J$ -е место ( $J < N$ ), то для окончания сортировки понадобится еще  $N - J$  проходов, за которые будет установлен на свое место всего один элемент.

В методе шейкера устраняется эта несимметричность в смещении минимального и максимального элементов. Достигается это чередованием проходов слева направо и справа налево. Так движется шейкер – стакан для встряхивания коктейля, отсюда и название метода.

## 2.4. Метод прямого включения

Основная идея метода – в уже упорядоченной последовательности элементов  $X(1), X(2), \dots, X(I)$  находить такое значение  $K$  индекса элемента, чтобы после присваивания  $X(K) = X(I+1)$  и включения элемента  $X(I+1)$  в эту последовательность получать новую упорядоченную последовательность из  $(I+1)$ -го элемента. Но прежде надо запомнить значение элемента  $X(I+1)$  в некоторой вспомогательной переменной  $A$ . Затем сместить элементы  $X(K), X(K+1), \dots, X(I)$  на одну позицию в сторону правой границы массива, выполнив присваивания  $X(I+1) = X(I)$ ,  $X(I) = X(I-1), \dots, X(K+1) = X(K)$ , и поместить включаемый элемент на  $K$ -ю позицию:  $X(K) = A$ . Блок-схема алгоритма этого метода представлена на рис. 7.

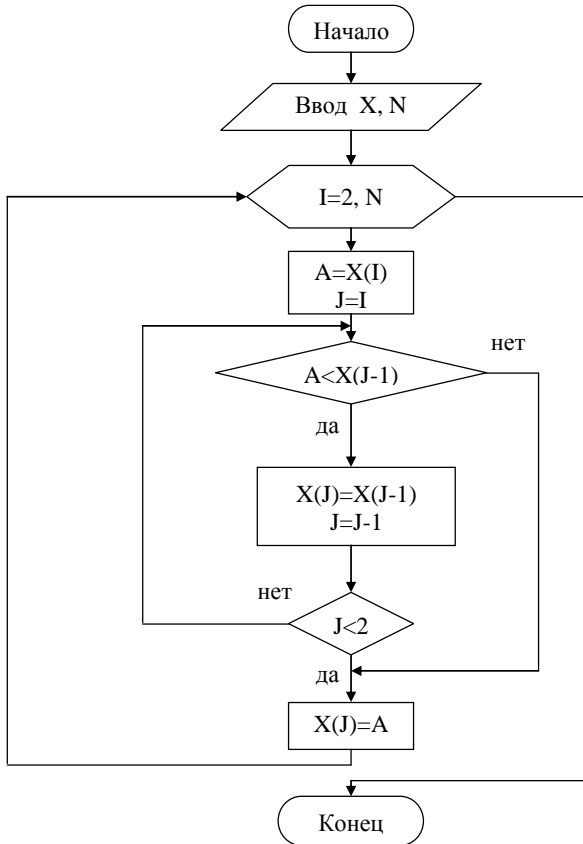


Рис. 7. Блок-схема алгоритма сортировки по методу прямого включения

## 2.5. Метод прямого выбора

Пусть имеется массив из  $N$  элементов. На первом этапе проанализируем все элементы этого массива и найдем минимальный среди них  $X(J) = \min_{I=1, N} X(I)$ . Теперь он должен занять первое место в новом упорядоченном списке элементов. Для этого поменяем местами найденный элемент  $X(J)$  и  $X(1)$ . На втором этапе снова повторим процедуру поиска минимального элемента среди элементов  $X(2), \dots, X(N)$ . По

окончании второго этапа переставляем местами найденный минимальный элемент  $X(J)$  и элемент  $X(I)$ . Эта процедура повторяется  $N-1$  раз.

Блок-схема алгоритма на основе этого метода сортировки представлена на рис. 8.

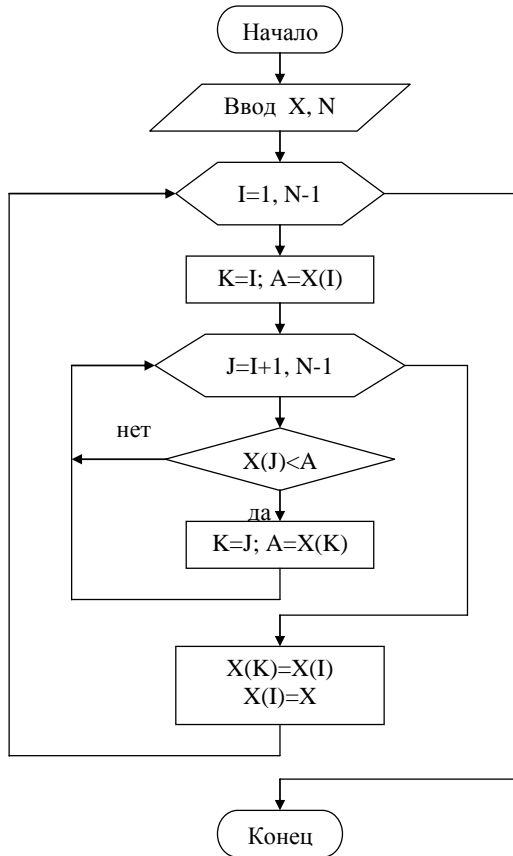


Рис. 8. Блок-схема алгоритма сортировки по методу прямого выбора

Идея метода прямого выбора противоположна идее метода прямого включения. Если в методе прямого включения анализируется уже упорядоченная последовательность, чтобы добавить в нее элемент из еще не упорядоченной последовательности, то в методе прямого вы-



бора, наоборот, анализируется еще не упорядоченная последовательность, чтобы выбрать в ней минимальный элемент и добавить его к уже упорядоченной последовательности. Достоинство метода прямого выбора – простота алгоритма, недостаток – невысокое быстродействие.

### 3. Порядок выполнения работы

1. Создать в среде **Delphi** новый проект.
2. Поместить на форму компоненты для ввода следующих величин:
  - а) количество элементов в массиве;
  - б) задержка между действиями (для отображения).
3. Поместить на форму активные элементы (кнопки и т.п.) для выполнения следующих действий:
  - а) формирование *неупорядоченной* последовательности случайных чисел;
  - б) сортировка сформированной последовательности.
4. Создать процедуру формирования *неупорядоченной* последовательности случайных целых чисел.
5. Создать механизм отображения последовательности ограниченной длины (до 50 элементов-чисел) с возможностью подсветки одного или нескольких элементов (в зависимости от метода сортировки). Для этого могут быть использованы возможности компонентов и их канвы.
6. Создать процедуру сортировки последовательности чисел с визуализацией процесса сортировки. **Метод сортировки выбирается согласно номеру бригады и уточняется у преподавателя.**
7. После выполнения сортировки необходимо вывести на экран следующие данные: количество совершённых действий, таких как сравнения, перестановки, разбиения и т.д. (в зависимости от метода сортировки), а также затраченное на сортировку время.

### 4. Пример выполнения лабораторной работы

В качестве примера приведём порядок выполнения задания для случая так называемой «глупой» сортировки. «Глупая» сортировка имеет нечто общее с сортировкой пузырьком: идёт поиск от начала массива, текущий элемент сравнивается со следующим, если следующий меньше, то производится обмен и возврат в начало цикла.

Для выполнения задания создаём пустой VCL проект и на главной форме размещаем необходимые элементы согласно пп. 2, 3 порядка выполнения. Пример формы приведён на рис. 9.

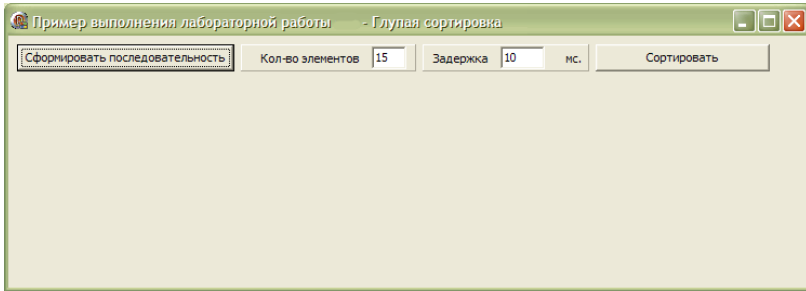


Рис. 9. Пример размещения элементов на главной форме программы

После размещения элементов необходимо создать процедуру формирования последовательности случайных целых чисел. Для этого создадим глобальный массив для хранения элементов последовательности. Размер массива ограничим (согласно заданию) 50 элементами. Далее создадим обработчик нажатия на кнопку «Сформировать последовательность», в котором будем заполнять часть данного массива случайными целочисленными значениями. Количество формируемых значений будем определять из значения, хранящегося в соответствующем поле ввода.

Теперь необходимо создать механизм отображения последовательностей целых чисел. На рис. 10 приведён пример отображения последовательности из 15 элементов. Значение каждого элемента соответствует высоте стилизованного дерева в пикселях, сверху над каждым из них для наглядности приведено значение высоты.

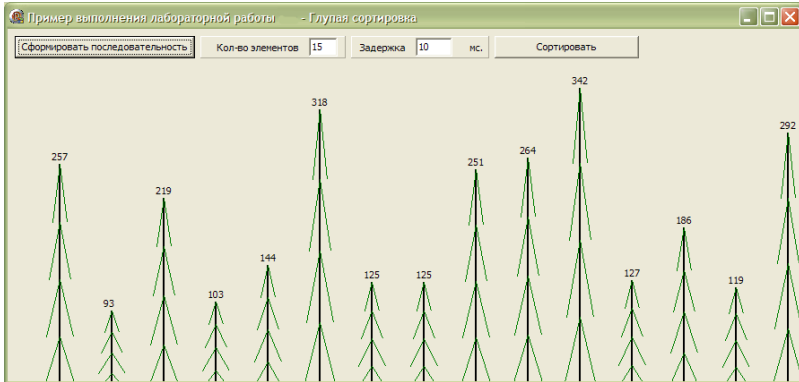


Рис. 10. Пример отображения произвольной последовательности целых чисел

Замечание: геометрические размеры изображения можно использовать для более красивого отображения элементов. Так, в приведённом примере высота максимального дерева на 50 пикселей меньше высоты формы, а интервал между деревьями определяется из ширины формы и количества элементов в последовательности.

Следующим шагом будет создание процедуры «глупой» сортировки последовательности чисел. Для этого создадим процедуру-обработчик нажатия на кнопку «Сортировать» и поместим в нее следующий код:

```
procedure TForm7.Button2Click(Sender: TObject);
var
  i,tmp : integer;
begin
  //Если последовательность не сформирована,
  //то сформировать её
  if (n=0) then //n-кол-во элементов в посл-ти
    // (глобальная переменная)
    Form7.Button1Click(sender);

  i:= 2; //номер текущего элемента

  while (i<=n) do //пока не дойдём до конца
    //последовательности
  begin //(A - глобальный массив элементов)
    if A[i]<A[i-1] then
      //если последующий элемент больше предыдущего
```

```

begin
    tmp:=A[i]; //меняем их местами, используя
    A[i]:=A[i-1]; //временную переменную
    A[i-1]:=tmp;
    i:=2;      //и сбрасываем текущий номер
end
else          //иначе
    i:=i+1;    //переходим к следующему элементу
end;
end;

```

После выполнения сортировки необходимо убедиться, что элементы расположены в правильном порядке. Для этого отображаем последовательность на форме. На рис. 11 показан результат выполнения «глупой» сортировки для последовательности, приведённой на рис. 10.

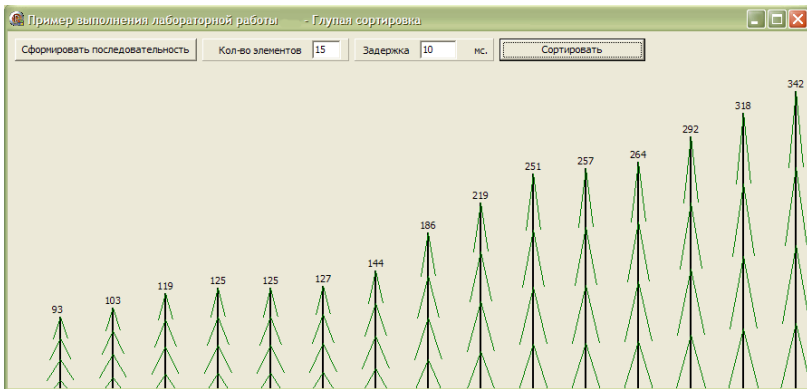


Рис. 11. Пример отображения сортированной по возрастанию последовательности целых чисел

Для иллюстрации процесса сортировки необходимо выводить текущее состояние массива на каждом шаге, выделяя текущий ( $i$ -й) элемент цветом. Пример выделения цветом 3-го элемента приведён на рис. 12.

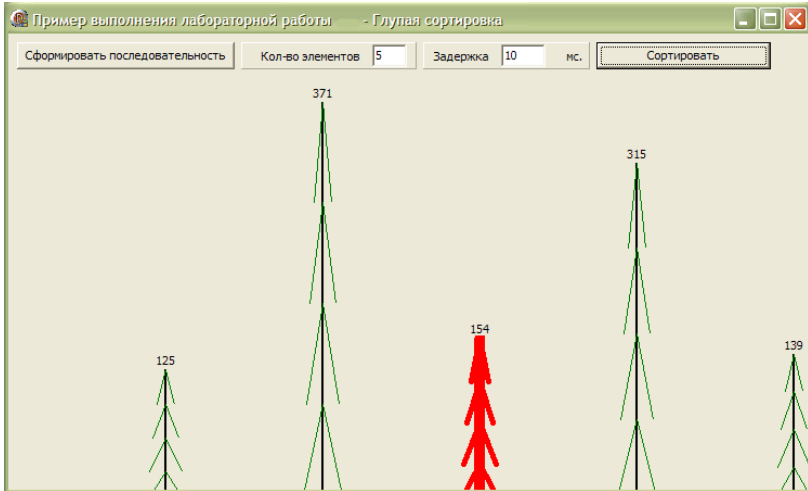


Рис. 12. Пример выделения элемента в процессе сортировки

Для того чтобы после выполнения сортировки вывести на экран данные о том, сколько каких действий было выполнено, добавим в код сортировки счётчики, которые будут увеличиваться на единицу каждый раз, когда будет выполняться сравнение или будет производиться перестановка. Для определения времени выполнения используем функцию `now()`. Результат выведем на экран, например, в виде сообщения (рис. 13).

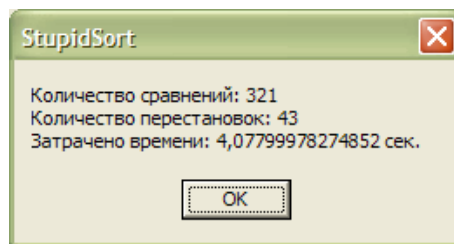


Рис. 13. Пример информационного сообщения о результатах сортировки

## ЛАБОРАТОРНАЯ РАБОТА № 5 ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ПОИСКА В СРЕДЕ DELPHI

### 1. Цель работы

Получение опыта практической реализации алгоритмов поиска значений в массиве, положения экстремумов и нулей функций.

### 2. Методы поиска

#### 2.1. Бинарный поиск

Для поиска нужного элемента в упорядоченном массиве можно использовать алгоритм бинарного поиска. Его суть заключается в том, что вначале с заданным значением  $A$  сравнивается средний элемент массива  $X(N/2)$ . Если он равен значению  $A$ , то алгоритм заканчивает свою работу. В противном случае, если средний элемент больше, чем значение, которое ищется, т.е.  $X(N/2) > A$ , то поиск повторяется в первой половине массива, если же  $X(N/2) < A$ , – во второй половине.

Таким образом, на каждом этапе число оставшихся элементов, среди которых будет организован поиск, делится пополам. Для больших массивов этот метод существенно превосходит по быстродействию последовательный поиск, для которого число оставшихся элементов, среди которых будет проводиться поиск, уменьшается только на единицу.

Обязательным условием для работы метода бинарного поиска является предварительное упорядочение массива. Это, в свою очередь, требует определенного времени. Если в одной программе задачу поиска приходится решать многократно, то имеет смысл предварительно отсортировать массив, а затем искать нужный элемент с помощью бинарного поиска. Характерным примером такого случая является поиск нужного телефона по телефонному справочнику.

#### 2.2. Метод бисекции

Задача заключается в нахождении корней нелинейного уравнения

$$f(x) = 0. \quad (1)$$

Для начала итераций необходимо знать отрезок  $[x_L, x_R]$  значений  $x$ , на концах которого функция принимает значения противоположных знаков. Противоположность знаков значений функции на концах отрезка можно определить множеством способов. Один из этих способов – умножение значений функции на концах отрезка и определение знака произведения путём сравнения результата умножения с нулём:

$$f(x_L) \cdot f(x_R) < 0, \quad (2.1)$$

в действительных вычислениях такой способ проверки противоположности знаков при крутых функциях приводит к преждевременному переполнению.

Для устранения переполнения и уменьшения затрат времени, то есть для увеличения быстродействия, на некоторых программно-компьютерных комплексах противоположность знаков значений функции на концах отрезка нужно определять по формуле:

$$\text{sign}(f(x_L)) \neq \text{sign}(f(x_R)), \quad (2.2)$$

так как одна операция сравнения двух знаков двух чисел требует меньшего времени, чем две операции: умножение двух чисел (особенно с плавающей запятой и двойной длиной) и сравнение результата с нулём. При данном сравнении значения функции  $f(x)$  в точках  $x_L$  и  $x_R$  можно не вычислять, достаточно вычислить только знаки функции  $f(x)$  в этих точках, что требует меньшего машинного времени.

Из непрерывности функции  $f(x)$  и условия (2.2) следует, что на отрезке  $[x_L, x_R]$  существует хотя бы один корень уравнения (если функция  $f(x)$  не монотонна, то она имеет несколько корней и метод приводит к нахождению одного из них).

Найдём значение  $x$  в середине отрезка:

$$x_M = (x_L + x_R) / 2, \quad (3)$$

в действительных вычислениях для уменьшения числа операций вначале (вне цикла) вычисляют длину отрезка по формуле:

$$x_D = (x_R - x_L), \quad (4)$$

а в цикле вычисляют длину очередных новых отрезков по формуле:  $x_D = x_D / 2$  и новую середину по формуле:

$$x_M = x_L + x_D. \quad (5)$$

Вычислим значение функции  $f(x_M)$  в середине отрезка  $x_M$ .

- Если  $f(x_M) = 0$  или, в действительных вычислениях,  $|f(x_M)| \leq \varepsilon_{f(x)}$ , где  $\varepsilon_{f(x)}$  – заданная точность по оси  $y$ , то корень найден.

- Иначе  $f(x_M) \neq 0$  или, в действительных вычислениях,  $|f(x_M)| > \varepsilon_{f(x)}$ , то разобьём отрезок  $[x_L, x_R]$  на два равных отрезка:  $[x_L, x_M]$  и  $[x_M, x_R]$ .

Теперь найдём новый отрезок, на котором функция меняет знак.

- Если значения функции на концах отрезка имеют противоположные знаки на левом отрезке,  $f(x_L) \cdot f(x_M) < 0$  или  $\text{sign}(f(x_L)) \neq \text{sign}(f(x_M))$ , то соответственно корень находится внутри левого отрезка  $[x_L, x_M]$ . Тогда возьмём левый отрезок присвоением  $x_R = x_M$  и повторим описанную процедуру до достижения требуемой точности  $\varepsilon_{f(x)}$  по оси  $y$ .

- Иначе значения функции на концах отрезка имеют противоположные знаки на правом отрезке,  $f(x_M) \cdot f(x_R) < 0$  или  $\text{sign}(f(x_M)) \neq \text{sign}(f(x_R))$ , то соответственно корень находится внутри правого отрезка  $[x_M, x_R]$ . Тогда возьмём правый отрезок присвоением  $x_L = x_M$  и повторим описанную процедуру до достижения требуемой точности  $\varepsilon_{f(x)}$  по оси  $y$ .

За количество итераций  $N$  деление пополам осуществляется  $N$  раз, поэтому длина конечного отрезка в  $2N$  раз меньше длины исходного отрезка.

### 2.3. Метод Ньютона

Метод Ньютона (также известный как метод касательных) – это итерационный численный метод нахождения корня (нуля) заданной функции. Поиск решения осуществляется путём построения последовательных приближений и основан на принципах простой итерации. Метод обладает квадратичной сходимостью. Также метод Ньютона может быть использован для решения задач оптимизации, в которых требуется определить нуль первой производной либо градиента в случае многомерного пространства.

#### *Обоснование*

Чтобы численно решить уравнение  $f(x) = 0$  методом простой итерации, его необходимо привести к следующей форме:  $x = \varphi(x)$ , где  $\varphi$  – сжимающее отображение.

Для наилучшей сходимости метода в точке очередного приближения  $x^*$  должно выполняться условие  $\varphi'(x^*) = 0$ . Решение данного уравнения ищут в виде  $\varphi(x) = x + \alpha(x) \cdot f(x)$ , тогда:



$$\varphi'(x^*) = 1 + \alpha'(x^*) \cdot f(x^*) + \alpha(x^*) \cdot f'(x^*) = 0. \quad (6)$$

В предположении, что точка приближения «достаточно близка» к корню  $x^*$  и что заданная функция непрерывна ( $f(x^*) \approx f(x^*) = 0$ ), окончательная формула для  $\alpha(x)$  такова:

$$\alpha(x) = -1 / f'(x). \quad (7)$$

С учётом этого функция  $\varphi(x)$  определяется выражением:

$$\varphi(x) = x - f(x) / f'(x). \quad (8)$$

Эта функция в окрестности корня осуществляет сжимающее отображение, и алгоритм нахождения численного решения уравнения  $f(x) = 0$  сводится к итерационной процедуре вычисления:

$$x_{n+1} = x_n - f(x_n) / f'(x_n). \quad (9)$$

По теореме Банаха последовательность приближений стремится к корню уравнения  $f(x) = 0$ .

Иллюстрация метода Ньютона приведена на рис. 1. Синим изображена функция  $f(x)$ , нуль которой необходимо найти, красным – касательная в точке очередного приближения  $x_n$ . Здесь мы можем увидеть, что последующее приближение  $x_{n+1}$  лучше предыдущего  $x_n$ .

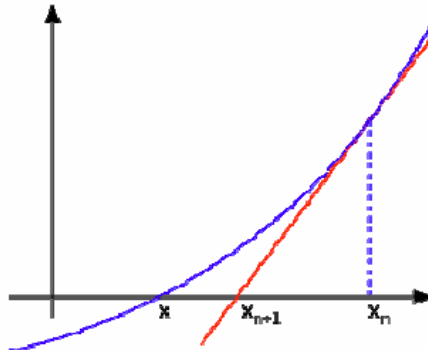


Рис. 1. Иллюстрация метода Ньютона

### Геометрическая интерпретация

Основная идея метода заключается в следующем: задаётся начальное приближение вблизи предположительного корня, после чего строится касательная к исследуемой функции в точке приближения, для которой находится пересечение с осью абсцисс. Эта точка и берётся в качестве следующего приближения. И так далее, пока не будет достигнута необходимая точность.

Пусть  $f(x) : [a, b] \rightarrow R$  – определённая на отрезке  $[a, b]$  и дифференцируемая на нём вещественнозначная функция. Тогда формула итеративного исчисления приближений может быть выведена следующим образом:

$$f'(x_n) = \operatorname{tg} \alpha = \frac{\Delta y}{\Delta x} = \frac{f(x_n) - 0}{x_n - x_{n+1}} = \frac{0 - f(x_n)}{x_{n+1} - x_n}, \quad (10)$$

где  $\alpha$  – угол наклона касательной в точке  $x_n$ .

Следовательно, искомое выражение для  $x_{n+1}$  имеет вид  $x_{n+1} = x_n - f(x_n) / f'(x_n)$ .

Итерационный процесс начинается с некоего начального приближения  $x_0$  (чем ближе к корню, тем лучше, но если предположения о его нахождении отсутствуют, методом проб и ошибок можно сузить область возможных значений, применив теорему о промежуточных значениях).

#### Алгоритм

1. Задаётся начальное приближение  $x_0$ .
2. Пока не выполнено условие остановки, в качестве которого можно взять  $|x_{n+1} - x_n| < \varepsilon$  или  $|f(x_{n+1})| < \varepsilon$  (то есть погрешность в нужных пределах), вычисляют новое приближение:  $x_{n+1} = x_n - f(x_n) / f'(x_n)$ .

## 2.4. Метод золотого сечения

Метод золотого сечения – метод поиска значений действительнозначной функции на заданном отрезке. В основе метода лежит принцип деления в пропорциях золотого сечения. Наиболее широко известен как метод поиска экстремума в решении задач оптимизации.

#### Описание метода

Пусть задана функция  $f(x) : [a, b] \rightarrow R$ ,  $f(x)$  принадлежит  $C([a, b])$ . Тогда для того, чтобы найти определённое значение этой функции на заданном отрезке, отвечающее критерию поиска (пусть это будет минимум), рассматриваемый отрезок делится в пропорции золотого сечения в обоих направлениях (как показано на рис. 2), то есть выбираются две точки  $x_1$  и  $x_2$  такие, что:

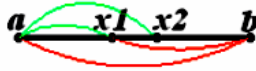


Рис. 2. Иллюстрация выбора промежуточных точек метода золотого сечения

$$\frac{b-a}{b-x_1} = \frac{b-a}{x_2-a} = \varphi = \frac{1+\sqrt{5}}{2} = 1,618..., \quad (11)$$

где  $\varphi$  – пропорция золотого сечения.

Таким образом:

$$\begin{aligned} x_1 &= b - \frac{(b-a)}{\varphi}; \\ x_2 &= a + \frac{(b-a)}{\varphi}. \end{aligned} \quad (12)$$

То есть точка  $x_1$  делит отрезок  $[a, x_2]$  в отношении золотого сечения. Аналогично  $x_2$  делит отрезок  $[x_1, b]$  в той же пропорции. Это свойство и используется для построения итеративного процесса.

#### *Алгоритм*

1. На первой итерации заданный отрезок делится двумя симметричными относительно его центра точками, и рассчитываются значения функции в этих точках.

2. Затем тот из концов отрезка, к которому среди двух вновь поставленных точек ближе оказалась та, значение в которой максимально (для случая поиска минимума), отбрасывают.

3. На следующей итерации в силу показанного выше свойства золотого сечения уже надо искать всего одну новую точку.

4. Процедура продолжается до тех пор, пока не будет достигнута заданная точность.

#### *Формализация*

Шаг 1. Задаются начальные границы отрезка  $a, b$  и точность  $\varepsilon$ .

Шаг 2. Рассчитываются начальные точки деления по формуле (12) и значения целевой функции в этих точках:  $y_1 = f(x_1)$ ,  $y_2 = f(x_2)$ .

- Если  $y_1 \geq y_2$  (для поиска максимума изменить неравенство на  $y_1 \leq y_2$ ), то  $a = x_1$ .
- Иначе  $b = x_2$ .

Шаг 3.

- Если  $|b-a| < \varepsilon$ , то  $x = (a+b)/2$  – решение найдено.
- Иначе возврат к шагу 2.

### 3. Варианты заданий

1. Найти элемент с заданным значением в упорядоченной последовательности целых чисел методом бинарного (двоичного) поиска.
2. Найти положение нуля функции  $x^3 + x^2 + x - 500$  на интервале  $x \in [1, 10]$  с точностью  $\varepsilon = 10^{-5}$  методом бисекции.
3. Найти положение нуля функции  $x^3 + x^2 + x - 500$  на интервале  $x \in [1, 10]$  с точностью  $\varepsilon = 10^{-5}$  методом Ньютона (касательных).
4. Найти положение максимума функции  $-x^3 + x^2 + x - 1$  на интервале  $x \in [0, 2]$  с точностью  $\varepsilon = 10^{-5}$  методом золотого сечения.

### 4. Порядок выполнения работы

#### 4.1. Для варианта № 1

1. Создать в среде **Delphi** новый проект.
2. Поместить на форму компоненты для ввода следующих величин:
  - а) количество элементов в массиве;
  - б) значение, которое необходимо найти;
  - в) задержка между действиями (для отображения).
3. Поместить на форму активные элементы (кнопки и т.п.) для выполнения следующих действий:
  - а) формирование *упорядоченной* последовательности;
  - б) нахождение введённого значения выбранным способом.
4. Создать процедуру формирования *упорядоченной* последовательности целых чисел.
5. Создать механизм отображения последовательности ограниченной длины (до 50 элементов) с возможностью подсветки одного или нескольких элементов. Для этого могут быть использованы возможности компонентов и их канвы.
6. Создать процедуру поиска заданного значения в последовательности целых чисел с визуализацией процесса поиска.
7. После выполнения поиска необходимо вывести на экран следующие данные: количество совершённых действий, таких как сравнения и т.д., а также затраченное на поиск время.

#### 4.2. Для вариантов № 2 и № 3

1. Создать в среде **Delphi** новый проект.
2. Поместить на форму компоненты для ввода следующих величин:
  - а) координаты начальной точки (если надо);
  - б) задержка между действиями (для отображения).
3. Поместить на форму активные элементы (кнопки и т.п.) для запуска процедуры поиска нулей функции заданным способом.
4. Создать процедуру поиска нулей заданной функции с введенными параметрами заданным способом.
5. Создать механизм визуализации процесса нахождения нулей: вывести оси координат, график заданной функции на определенном отрезке, отобразить на нем рассматриваемые точки, отрезки и т.д. (в зависимости от метода).
6. После нахождения нулей необходимо вывести на экран следующие данные: количество совершенных действий, а также затраченное на поиск нуля время.

#### 4.3. Для варианта № 4

1. Создать в среде **Delphi** новый проект.
2. Поместить на форму компоненты для задания задержки между действиями (для отображения).
3. Поместить на форму активные элементы (кнопки и т.п.) для запуска процедуры поиска экстремумов функции заданным способом.
4. Создать процедуру поиска экстремумов заданной функции с введенными параметрами заданным способом.
5. Создать механизм визуализации хода нахождения экстремумов: вывести оси координат, график заданной функции на определенном отрезке, отобразить на нем рассматриваемые точки, отрезки и т.д.
6. После нахождения экстремума необходимо вывести на экран следующие данные: количество совершенных действий, а также затраченное на поиск время.

## 5. Пример выполнения лабораторной работы

В качестве примера приведём порядок выполнения задания для случая линейного поиска значений экстремумов в несортированной последовательности целых чисел.

Для выполнения задания создаём пустой VCL проект и на главной форме размещаем необходимые элементы интерфейса. Пример формы приведён на рис. 3.

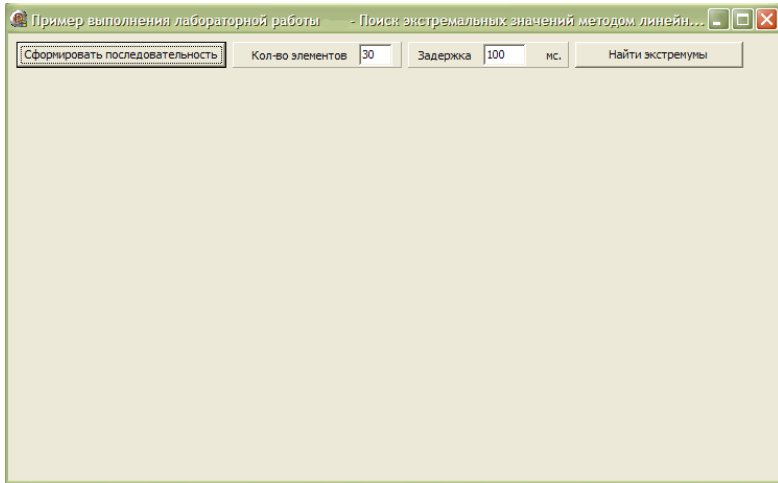


Рис. 3. Пример размещения элементов на главной форме

После размещения элементов необходимо создать процедуру формирования последовательности целых чисел. Для этого создадим глобальный массив для хранения элементов последовательности. Размер массива ограничим 50 элементами. Далее создадим обработчик нажатия на кнопку «Сформировать последовательность», в котором будем заполнять часть данного массива случайными целочисленными значениями. Количество формируемых значений будем определять из значения, хранящегося в соответствующем поле ввода.

Теперь необходимо создать механизм отображения последовательностей целочисленных значений. На рис. 4 приведён пример отображения последовательности из 30 элементов. Значение каждого элемента соответствует высоте дерева в пикселях, сверху над каждым из них для наглядности приведено значение высоты.

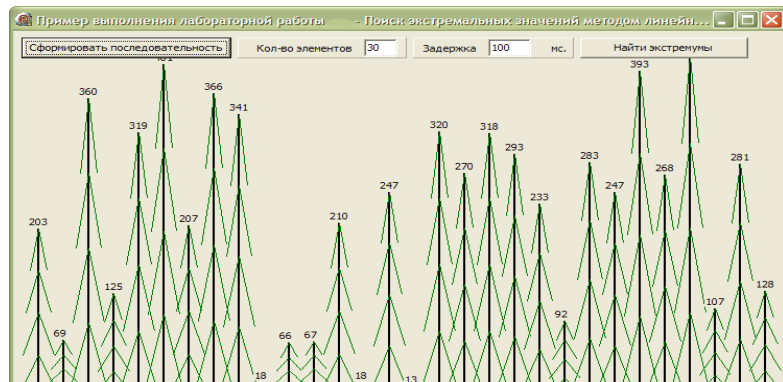


Рис. 4. Пример отображения произвольной последовательности чисел

Следующим шагом будет создание процедуры линейного поиска минимума и максимума в последовательности. Для этого создадим обработчик нажатия на кнопку «Найти экстремумы» и поместим в него следующий код:

```
procedure TForm7.Button2Click(Sender: TObject);
var //переменные
    i,minA,maxA,maxi,mini: integer;
begin
    //сформировать последовательность, если необходимо
    if (n=0) then //n-кол-во элементов в посл-ти
        Form7.Button1Click(sender);
    minA:= maxint; //минимальный элемент
    maxA:= -maxint; //максимальный элемент
    maxi:= -1; //положение максимального элемента
    mini:= -1; //положение минимального элемента
    for i:=1 to n do //цикл по всем элементам
    begin
        if maxA<A[i] then //если элемент больше максимума
        begin
            maxA:= A[i]; //он становится максимумом
            maxi:= i;
        end;
        if minA>A[i] then //если элемент меньше минимума
        begin
            minA:= A[i]; //он становится минимумом
            mini:= i;
        end;
    end;
end;
```

После поиска можно визуально оценить положение минимума и максимума. Для этого их необходимо выделить цветом или начертанием. Отобразим последовательность на форме. Это требуется также для иллюстрации процесса поиска. Необходимо выводить текущее состояние массива на *каждом* шаге, выделяя текущий ( $i$ -й) элемент, текущий максимальный и текущий минимальный элементы. Пример отображения результата поиска экстремумов приведён на рис. 5.

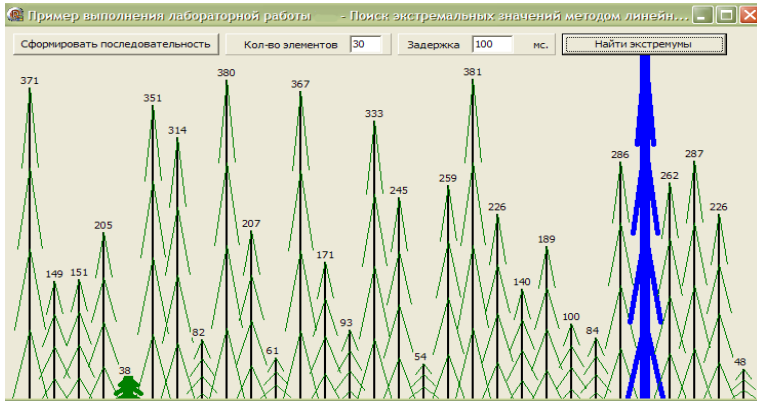


Рис. 5. Пример выделения элементов в процессе поиска экстремумов

Для того чтобы после нахождения экстремумов вывести на экран данные о них и о том, сколько каких действий было выполнено, добавим в код сортировки счётчики, которые будут увеличиваться на единицу каждый раз, когда будет выполняться сравнение или будет производиться перестановка. Для определения времени выполнения используем функцию **now()**. Результат выведем на экран, например, в виде сообщения (рис. 6).

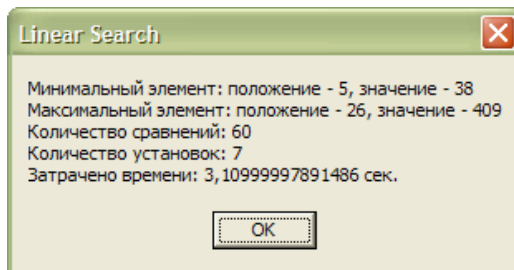


Рис. 6. Пример информационного сообщения о результатах поиска



## ЛАБОРАТОРНАЯ РАБОТА № 6 ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ОБРАБОТКИ ОДНОМЕРНЫХ СИГНАЛОВ В СРЕДЕ DELPHI

### 1. Цель работы

Изучение основ цифровой обработки сигналов и получение опыта практической реализации простых алгоритмов анализа и обработки одномерных сигналов на языке Delphi.

### 2. Цифровая обработка сигналов

**Цифровая обработка сигналов** (ЦОС) – это выполнение различных операций преобразования сигналов, представленных в цифровой форме. Иначе говоря, такие сигналы представляют собой последовательность чисел – значений непрерывной переменной, например, в пространственной, временной или частотной области.

В настоящее время ЦОС применяется во многих областях науки и техники, например при обработке звуковой и речевой информации; обработке данных измерения различных величин датчиками; обработке статистических данных; обработке изображений и видео; в управлении сложными системами; в системах кодирования и передачи информации; в медицине и биоинженерии и в других областях. ЦОС применима как при обработке потоковых данных, так и при обработке сохраненных данных.

Можно выделить следующие основные направления ЦОС.

- Цифровая фильтрация сигналов.
- Спектральный анализ сигналов.
- Цифровая частотная селекция сигналов.
- Многоскоростная обработка сигналов.
- Адаптивная и оптимальная обработка сигналов.
- Обработка многомерных сигналов.

Алгоритмы ЦОС реализуются и применяются на базе компьютеров общего назначения; цифровых сигнальных процессоров; программируемых логических интегральных схем (ПЛИС), в частности на базе программируемых пользователем вентильных матриц (FPGA); интегральных схем специального назначения (ASIC) и т.д.

Обработка сигналов в системах контроля, управления и диагностики является сложной комплексной задачей, требующей для своего решения привлечения разнообразных методов математической статистики.

Анализ сигналов зачастую затруднен наличием ошибок, имеющих случайный характер, которые не позволяют достоверно оценить характеристики полученных зависимостей или описать их функционально. Необходимо применять специальные методы для ослабления случайной составляющей (шума) и выделения полезного сигнала. Оценка полезного сигнала может осуществляться как параметрическими, так и непараметрическими методами в зависимости от априорной информации о полезной и шумовой составляющей.

Для использования параметрических методов обработки необходима априорная информация о модели полезного сигнала. Использование непараметрических методов обработки требует значительно меньше априорной информации, но погрешность оценки сигнала имеет ярко выраженную зависимость от параметров обработки, значения которых зависят от характеристик выделяемого полезного сигнала и закона распределения шума, объема исходных данных сигнала.

Рассмотрим некоторые простые алгоритмы обработки и анализа одномерных сигналов. Данные алгоритмы будут использоваться при выполнении лабораторной работы.

## 2.1. Характеристики случайных величин

Сигналы на входе системы могут быть случайные или детерминированные. Во втором случае мы обычно знаем функцию, с помощью которой можно вычислить значение сигнала в любой момент времени. Для случайных сигналов мы можем только оценить вероятность того, что он примет то или иное значение. Закон распределения случайной величины нам не всегда известен, но мы можем оценить его характеристики с некоторой погрешностью. Одним из самых популярных законов распределения является нормальный закон.

Рассмотрим наиболее простые характеристики закона распределения: среднее (математическое ожидание), дисперсия, среднеквадратическое отклонение. В формулах используется обозначение  $y(x)$  – конечная последовательность с длиной  $N$ . Запись  $y_i$  эквивалентна  $y(x_i)$ .

**Математическое ожидание** представляет собой статистическое усреднение случайной величины  $y(x)$ . Функция математического ожидания является теоретической оценкой среднего взвешенного значения случайного процесса, например, во времени.

$$\mu = \frac{1}{N} \sum_{i=0}^{N-1} y_i \quad (1.1)$$

**Дисперсия** – характеристика разброса значений случайной величины относительно ее среднего значения. Размерность дисперсии равна размерности случайной величины в квадрате.

Оценка дисперсии бывает **смещенной** и вычисляется по формуле

$$\sigma^2 = \frac{1}{N} \sum_{i=0}^{N-1} (y_i - \mu)^2 = -\mu^2 + \frac{1}{N} \sum_{i=0}^{N-1} y_i^2. \quad (1.2)$$

И **несмещенной**:

$$\sigma^2 = \frac{1}{N-1} \sum_{i=0}^{N-1} (y_i - \mu)^2 = -\frac{N}{N-1} \cdot \mu^2 + \frac{1}{N-1} \sum_{i=0}^{N-1} y_i^2. \quad (1.3)$$

**Среднеквадратическое отклонение** так же, как и дисперсия, является показателем разброса значений случайной величины относительно ее среднего значения. Измеряется в тех же единицах, что и сама случайная величина, и равна квадратному корню из дисперсии.

$$\sigma = \sqrt{\sigma^2} \quad (1.4)$$

Соответственно среднеквадратическое отклонение может вычисляться как на основе смещенной оценки дисперсии, так и на основе несмещенной оценки дисперсии.

## 2.2. Гистограмма

**Гистограмма** – способ представления данных в виде столбчатой диаграммы, которая отображает распределение случайных величин. Иногда гистограмму называют частотным распределением, так как она показывает частоту появления тех или иных значений величины. Высота столбца указывает на частоту появления значений в выбранном диапазоне, а количество столбцов соответствует числу выбранных диапазонов.

Для построения гистограммы наблюдаемый диапазон изменения случайной величины разбивается на несколько интервалов и подсчитывается доля от всех значений, попавших в каждый интервал. Величина каждой доли, отнесенная к величине интервала, принимается в качестве оценки значения плотности распределения на соответствующем интервале.

**Алгоритм построения гистограммы** следующий.

- Производится поиск минимального  $y_{\min}$  и максимального  $y_{\max}$  значений в последовательности  $y(x)$ .

- Определяется ширина диапазона значений – разница значений  $y_{\max}$  и  $y_{\min}$ .

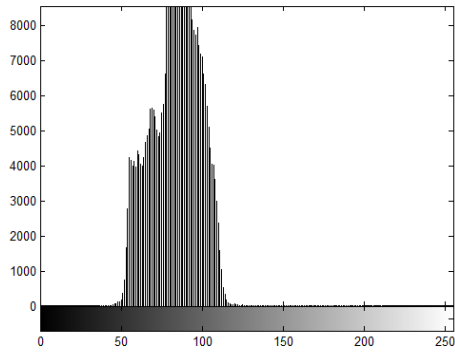
- Выбирается число интервалов, в пределах которых необходимо сгруппировать результаты измерений. Интервал зачастую выбирается равным 1: если последовательность содержит целые числа, то для каждого значения из последовательности будет создан свой разряд гистограммы.

- Подсчитывается число попаданий значений результатов измерений в каждый из интервалов.

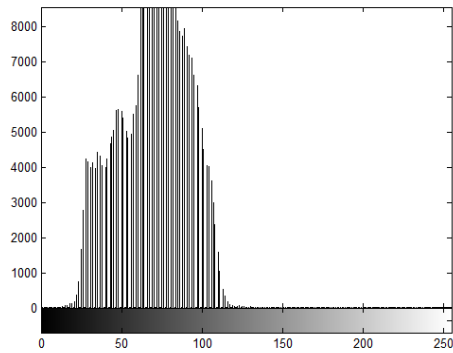
- На оси абсцисс отмечаются интервалы, а на оси ординат – число попаданий значений из последовательности в каждый интервал.

Распространенной операцией при обработке сигналов является **коррекция гистограммы**. Данная операция может применяться, например, как один из этапов алгоритмов распознавания речи. Также коррекция гистограммы используется при обработке изображений, в данном случае она позволяет усилить контрастность изображения и повысить детализацию.

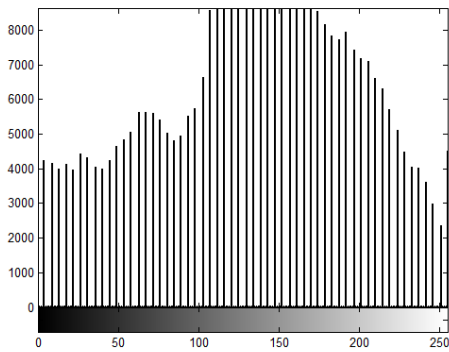
В качестве методов коррекции гистограммы можно выделить линейное растяжение гистограммы и эквализацию гистограммы. На рис. 1, а приведена гистограмма амплитуды некоторого сигнала, на рис. 1, б – результат линейного растяжения, на рис. 1, в – результат эквализации.



а



б



в

Рис. 1. Гистограмма амплитуды сигнала

Рассмотрим простейший из алгоритмов коррекции гистограммы – линейное растяжение. На практике же лучший результат может быть получен при применении различных алгоритмов эквализации.

**Алгоритм линейного растяжения** состоит из следующих шагов.

- Задается интервал  $[v_{\min}; v_{\max}]$ , которому должны принадлежать значения функции – элементы последовательности  $y(x)$ .
- Производится поиск минимального  $y_{\min}$  и максимального  $y_{\max}$  значений в последовательности  $y(x)$ .
- Рассчитываются коэффициенты по следующим формулам:

$$a = \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}}, \quad (2.1)$$

$$b = -a \cdot y_{\min} + v_{\min}. \quad (2.2)$$

- Новые значения элементов последовательности находятся на основании соотношения

$$\tilde{y}(x) = y(x) \cdot a + b. \quad (2.3)$$

### 2.3. Линейная свертка

Пусть даны две конечные последовательности  $y_1$  и  $y_2$  с длинами соответственно  $N_1$  и  $N_2$  отсчетов. **Линейной сверткой** этих последовательностей называют последовательность  $C$  с длиной  $N_c = N_1 + N_2 - 1$ , элементы которой определяются соотношением

$$C(n) = y_1(n) \otimes y_2(n) = \sum_{i=0}^n y_1(i) \cdot y_2(n-i); \quad n = \overline{0, N_c - 1}. \quad (3.1)$$

Зачастую одна из последовательностей представляет собой обрабатываемые данные (сигнал на входе системы), вторая – оператор (импульсный отклик) системы, а функция  $C(n)$  – выходной сигнал системы. Также операцию свертки можно интерпретировать как схожесть одной функции с отражённой и сдвинутой копией другой.

Необходимо учесть, что индексы  $i$  и  $n-i$  в формуле (3.1) будут принимать значения, выходящие за допустимые интервалы  $[0; N_1-1]$  и  $[0; N_2-1]$ . В таких случаях значение соответствующей функции ( $y_1$  или  $y_2$ ) принимается равным 0.

## 2.4. Корреляция

**Корреляция** или корреляционная зависимость – статистическая взаимосвязь двух или более случайных величин. При этом изменения значений одной или нескольких из этих величин сопутствуют систематическому изменению значений другой или других величин.

Важно отметить следующее: если величины коррелируют, то они не обязательно связаны друг с другом, они могут иметь стороннюю общую причину. С другой стороны, отсутствие корреляции между величинами ещё не значит, что между ними нет никакой связи, так, зависимость может иметь сложный нелинейный характер, который корреляция не выявляет.

Следует выделить две формы корреляции: автокорреляция (автокорреляционная функция) и взаимная корреляция (кросскорреляционная функция).

**Автокорреляция** характеризует степень связи между сигналом и его сдвинутой во времени копией. На основе автокорреляции можно судить о периодичности функции, ее частотных характеристиках, так как, если функция строго периодическая, то автокорреляционная функция тоже будет строго периодической. Для расчета можно использовать соотношение

$$R_{auto}(n) = \sum_{i=0}^{N-n-1} y(i) \cdot y(n+i); \quad n = \overline{0, N-1}, \quad (4.1)$$

где  $N$  – количество элементов в последовательности  $y$ . Размер результирующей последовательности  $R_{auto}$  также равен  $N$ .

**Взаимная корреляция** – оценка степени корреляции двух последовательностей. Взаимная корреляция связана со свёрткой и часто используется для поиска заранее известной последовательности в более длинной последовательности. Для расчета можно использовать соотношение

$$R_{cross}(n) = \sum_{i=0}^{N_2-1} y_1(n+i) \cdot y_2(i); \quad n = \overline{0, N_{cross}-1}. \quad (4.2)$$

Размер результирующей последовательности  $R_{cross}$  равен  $N_{cross} = N_1 - N_2 + 1$ . Соответственно длина последовательности  $y_1$  должна быть больше или равна длине последовательности  $y_2$  ( $N_1 \geq N_2$ ).

## 2.5. Интерполяция

**Интерполяция** – способ нахождения промежуточных значений величины по имеющемуся дискретному набору известных значений.

При решении технических и научных задач зачастую приходится иметь дело с последовательностью значений, полученных путем случайной выборки или в ходе эксперимента. Такие последовательности содержат значения функции для ограниченного числа значений аргумента. Часто требуется интерполировать (т.е. оценить) значение этой функции для некоторого промежуточного значения аргумента.

Самым простым методом интерполяции является интерполяция **методом ближайшего соседа** – метод интерполяции, при котором в качестве промежуточного значения выбирается ближайшее известное значение функции. При использовании данного метода также необходимо выбрать способ решения задачи поиска ближайшего соседа. Ведь поиск элементов, близких к заданному, можно осуществлять согласно различным функциям близости. Самой распространенной функцией близости является евклидово расстояние – расстояние между двумя точками евклидова пространства, вычисляемое по теореме Пифагора.

Рассмотрим другой метод интерполяции – **линейную интерполяцию**. Она представляет собой метод построения кривой с использованием линейного полинома ( $k \cdot x + b$ ) для вычисления новых точек в пределах диапазона, заданного двумя точками из дискретной последовательности значений некоторой величины.

Например, пусть даны две точки  $(x_1, y_1)$  и  $(x_2, y_2)$ , тогда приближенное значение  $y$  для аргумента  $x$ , находящегося между  $x_1$  и  $x_2$ , можно найти по формуле

$$y = y_1 + (x - x_1) \frac{y_2 - y_1}{x_2 - x_1}. \quad (5.1)$$

## 2.6. Аппроксимация

Другой задачей, которая тесно связана с интерполяцией, является **аппроксимация** сложной функции некоторой более простой функцией. Предположим, нам известна некоторая функция, но она слишком сложна для выполнения вычислений. Взяв известные точки, соответствующие этой функции, можно построить более простую функцию. Конечно, надо понимать, что в результате использования простой функции вместо исходной результаты расчетов будут содержать



ошибки. Такую ошибку называют **остаточным членом** – разность между исходной функцией и функцией, ее аппроксимирующей. Таким образом, оценка остаточного члена является оценкой точности аппроксимации.

Необходимость в аппроксимации также часто возникает при обработке экспериментальных данных, представленных как дискретный набор точек. В таком случае аппроксимацию называют точечной или дискретной. Наиболее часто встречающимся видом точечной аппроксимации является интерполяция. Если же аппроксимация проводится на непрерывном множестве точек, то она называется непрерывной или интегральной.

Возьмем случай, когда данные, представленные в виде последовательности чисел, служат опорными точками для выявления закона изменения некоторой величины. Рассмотрим аппроксимацию линейной функцией и аппроксимацию синусоидой.

Уравнения линии и синусоиды приведены ниже.

$$y(x) = k \cdot x + b \quad (6.1)$$

$$y(x) = a + b \cdot \sin(c \cdot x + d) \quad (6.2)$$

Аппроксимация заключается в определении коэффициентов ( $k$ ,  $b$  – для линии или  $a$ ,  $b$ ,  $c$ ,  $d$  – для синусоиды) уравнения таких, чтобы все точки из исходной последовательности лежали наиболее близко к аппроксимирующей функции.

Распространенным методом определения коэффициентов уравнения является **метод наименьших квадратов** (МНК), суть которого заключается в минимизации суммы квадратов отклонений значения функции от аппроксимирующей точки.

Решение поставленной задачи сводится к нахождению минимумов следующих функций (для линии и синусоиды соответственно).

$$F(k, b) = \sum_{i=0}^{N-1} (y_i - (k \cdot x_i + b))^2 \quad (6.3)$$

$$F(a, b, c, d) = \sum_{i=0}^{N-1} (y_i - (a + b \cdot \sin(c \cdot x_i + d)))^2 \quad (6.4)$$

Для определения минимумов надо сначала найти частные производные функций по соответствующим коэффициентам, далее производные приравниваются к нулю. Полученные уравнения можно решить относительно искомых коэффициентов численными методами, например методом бисекции или методом Ньютона.

Также из полученной системы уравнений можно вывести формулы для определения значений коэффициентов. Например, для линии получим следующие формулы. Для упрощения записи обозначение

$\sum_{i=0}^{N-1}$  было заменено на  $\text{sum}()$ .

$$\begin{cases} k = \frac{N \cdot \text{sum}(x_i \cdot y_i) - \text{sum}(x_i) \cdot \text{sum}(y_i)}{N \cdot \text{sum}(x_i^2) - \text{sum}^2(x_i)}, \\ b = \frac{\text{sum}(y_i) - k \cdot \text{sum}(x_i)}{N}. \end{cases} \quad (6.5)$$

## 2.7. Сглаживание данных

**Сглаживание** – это задача фильтрации сигнала, целью которой является устранение скачкообразных выбросов. Считается, что наличие таких выбросов является следствием наличия шума. Сглаживающий фильтр по-другому называется усредняющим фильтром.

Сглаживание данных является специальной операцией усреднения с помощью интерполяционных многочленов, обеспечивающей получение уточнённого значения  $\tilde{y}_i$  по заданному значению  $y_i$  и последовательности близлежащих значений  $(\dots, y_{i-1}, y_i, y_{i+1}, \dots)$ , известных со случайной погрешностью.

Наилучшее сглаживание получается для средних точек последовательности. Сглаживание крайних точек производится по специальным формулам. Рассмотрим наиболее простые алгоритмы сглаживания: сглаживание по трем точкам и сглаживание по пяти точкам. Подобные фильтры называются **оконными**. Размер окна (иначе говоря, **апертуры фильтра**) для рассматриваемых случаев равен соответственно 3 и 5.

**Сглаживание по трем точкам** выполняется согласно формулам:

$$\begin{aligned}
\tilde{y}_0 &= \frac{5 \cdot y_0 + 2 \cdot y_1 - y_2}{6}, \\
\tilde{y}_i &= \frac{y_{i-1} + y_i + y_{i+1}}{3}, \\
\tilde{y}_{N-1} &= \frac{5 \cdot y_{N-1} + 2 \cdot y_{N-2} - y_{N-3}}{6}.
\end{aligned} \tag{7.1}$$

**Сглаживание по пяти точкам** выполняется согласно формулам:

$$\begin{aligned}
\tilde{y}_0 &= \frac{3 \cdot y_0 + 2 \cdot y_1 + y_2 - y_4}{5}, \\
\tilde{y}_1 &= \frac{4 \cdot y_0 + 3 \cdot y_1 + 2 \cdot y_2 + y_3}{10}, \\
\tilde{y}_i &= \frac{y_{i-2} + y_{i-1} + y_i + y_{i+1} + y_{i+2}}{5}, \\
\tilde{y}_{N-2} &= \frac{4 \cdot y_{N-1} + 3 \cdot y_{N-2} + 2 \cdot y_{N-3} + y_{N-4}}{10}, \\
\tilde{y}_{N-1} &= \frac{3 \cdot y_{N-1} + 2 \cdot y_{N-2} + y_{N-3} - y_{N-5}}{5}.
\end{aligned} \tag{7.2}$$

Отметим, что не всегда крайним отсчетам уделяется особое внимание. Зачастую они либо не обрабатываются (особенно при большой длине последовательности и малом размере апертуры фильтра), либо для расчета берутся только попадающие в окно фильтра отсчеты (например, отсчеты с номерами 0, 1 и 2 для 0-го отсчета при сглаживании по 5 точкам).

Общая формула для усредняющего фильтра с размером апертуры  $S$  определяется следующим образом.  $S$  – нечетное целое число,  $s2$  – результат целочисленного деления  $S$  на 2 ( $S \text{ div } 2$ ).

$$\tilde{y}_i = \frac{1}{S} \sum_{d=-s2}^{s2} y_{i+d} \tag{7.3}$$

## 2.8. Медианный фильтр

Еще одним из видов цифровых фильтров, широко используемых в цифровой обработке сигналов, является **медианный фильтр**. Такие фильтры могут сохранять без искажений резкие границы объектов,

эффективно подавляя некоррелированные или слабо коррелированные помехи и малоразмерные детали. Медианная фильтрация применяется для устранения аномальных значений в массивах данных, уменьшения выбросов и импульсных помех.

Характерной особенностью медианного фильтра является его нелинейность. Во многих случаях применение медианного фильтра оказывается более эффективным по сравнению с линейными фильтрами, поскольку процедуры линейной обработки являются оптимальными при равномерном или гауссовом распределении помех, что в реальных сигналах может быть далеко не так. Особенно эффективным медианный фильтр оказывается при подавлении импульсных шумов при обработке изображений, акустических сигналов, передаче кодовых сигналов и т.п.

**Медианой** числовой последовательности  $y$ , имеющей нечетную длину  $N$ , является средний по положению элемент ряда, получаемого при упорядочивании (по возрастанию или убыванию) исходной последовательности  $y$ . При четной длине  $N$  медиану принято вычислять как среднее арифметическое двух средних элементов упорядоченного ряда.

Рассмотрим медианную фильтрацию на примере. Пусть задана последовательность из 10 чисел:

23 74 35 52 27 90 31 62 82 65

Зададим размер апертуры фильтра равным 3. Для граничных отсчетов (с индексами 0 и 9 в данном случае) фильтруемый ряд будем дополнять копией этих отсчетов (подчеркнуты). Запишем этапы медианной фильтрации заданной последовательности:

Значение	Фильтруемый ряд	Упорядочивание	Медиана
0: 23	<u>23</u> 23 74	23 23 74	23
1: 74	23 74 35	23 35 74	35
2: 35	74 35 52	35 52 74	52
3: 52	35 52 27	27 35 52	35
4: 27	52 27 90	27 52 90	52
5: 90	27 90 31	27 31 90	31
6: 31	90 31 62	31 62 90	62
7: 62	31 62 82	31 62 82	62
8: 82	62 82 65	62 65 82	65
9: 65	82 65 <u>65</u>	65 65 82	65

Результат фильтрации:

23 35 52 35 52 31 62 62 65 65

### 3. Варианты заданий

*Примечание:* для всех вариантов будем считать, что  $\mathbf{x}$  представляет собой момент времени (в секундах), а массив  $\mathbf{Y}$  содержит амплитуду некоторого сигнала. Таким образом, функция  $Y(x)$  – зависимость амплитуды сигнала от времени.

**Вариант № 1.** Построение и коррекция гистограммы.

- Массив  $\mathbf{Y}$  заполнить в соответствии с формулой:

$$Y_x = \text{round}\left(40 \cdot \left(2 + \sin \frac{x}{5}\right)\right), \quad x \in [0; 199], \text{ шаг изменения } x: 1.$$

- Добавить 2 поля ввода для значений  $v_{\min}$  и  $v_{\max}$ . По умолчанию выставить значения в полях 1 и 200 соответственно. В дальнейшем учесть, что минимальное значение не может быть меньше 1, а максимальное – больше 200.
- Построить гистограмму амплитуды сигнала. Отобразить гистограмму на форме под графиком  $Y(x)$ . При отображении значения гистограммы рекомендуется умножать на 10.
- Процедура обработки сигнала должна выполнить линейное растяжение гистограммы согласно заданным  $v_{\min}$  и  $v_{\max}$  с округлением результата (round). Справа от графика  $Y(x)$  должен быть отображен результат обработки  $\tilde{Y}(x)$ , под которым – скорректированная гистограмма.
- Графики  $Y(x)$  и  $\tilde{Y}(x)$ : отсчеты должны быть соединены линиями. Гистограмма: отсчеты в виде столбцов.

**Вариант № 2.** Линейная свертка.

- Массив  $\mathbf{Y1}$  заполнить в соответствии с формулой:

$$Y_x = 100 + 10 \cdot \left(\frac{-x - 100}{100}\right)^2 \cdot \cos \frac{x + 100}{3 \cdot 100}, \quad x \in [0; 199], \text{ шаг изменения}$$

$x: 1.$

- К каждому значению из массива  $\mathbf{Y1}$  прибавить случайное число из диапазона  $[-7; 7]$ .
- Массив  $\mathbf{Y2}$  с длиной 10 заполнить значениями  $1/10$ . Отображать на форме содержимое массива  $\mathbf{Y2}$  не обязательно.
- Процедура обработки сигнала должна выполнить линейную свертку последовательностей  $\mathbf{Y1}$  и  $\mathbf{Y2}$ . Под графиком  $Y_1(x)$  должен быть отображен результат обработки  $C$ .
- Графики  $Y(x)$  и  $C$ : отсчеты должны быть соединены линиями.

**Вариант № 3. Автокорреляция.**

- Массив **Y** заполнить в соответствии с формулой:

$$Y_x = 30 \cdot \left( 3 + \cos \frac{x}{5} \right), \quad x \in [0; 199], \text{ шаг изменения } x: 1.$$

- К каждому значению из массива **Y** прибавить случайное число из диапазона  $[-30; 30]$ .
- Процедура обработки сигнала должна выполнить расчет автокорреляции для последовательности **Y**. Под графиком  $Y(x)$  должен быть отображен результат обработки  $R_{auto}$ . При отображении результата значения  $R_{auto}$  рекомендуется делить на 10000.
- Модифицировать процедуру обработки так, чтобы автокорреляцию можно было рассчитывать по формуле

$$R_{auto}(n) = \sum_{i=0}^{N-n-1} (y_i - \mu) \cdot (y_{n+i} - \mu), \text{ где } \mu - \text{математическое ожидание.}$$

- Графики  $Y(x)$  и  $R_{auto}$ : отсчеты должны быть соединены линиями.

**Вариант № 4. Взаимная корреляция.**

- Массив **Y1** заполнить в соответствии с формулой:

$$Y_x = 7 \cdot \left( 11 + \frac{\sin \frac{x-k}{4}}{\sin \frac{x-k}{40}} \right), \quad k = 350, \quad x \in [0; 199], \text{ шаг изменения } x: 1.$$

- К каждому значению из массива **Y1** прибавить случайное число из диапазона  $[-20; 20]$ .
- Массив **Y2** заполнить в соответствии с вышеприведенной формулой при  $k = 265$ ,  $x \in [0; 29]$ , шаг изменения  $x$ : 1. График  $Y_2(x)$  отобразить справа от графика  $Y_1(x)$ .
- Процедура обработки сигнала должна выполнить расчет взаимной корреляции последовательностей **Y1** и **Y2**. Под графиками  $Y_1(x)$  и  $Y_2(x)$  должен быть отображен результат обработки  $R_{cross}$ . При отображении результата значения  $R_{cross}$  рекомендуется делить на 5000.
- Графики  $Y_1(x)$ ,  $Y_2(x)$  и  $R_{cross}$ : отсчеты должны быть соединены линиями.

**Вариант № 5. Сглаживание.**

- Массив **Y** заполнить в соответствии с формулой:

$$Y_x = 60 - 30 \cdot \left( 1 - \left( \frac{x+1}{30} \right)^{\frac{120}{x+1}} \right), \quad x \in [0; 199], \text{ шаг изменения } x: 1.$$

- К каждому значению из массива **Y** прибавить случайное число из диапазона  $[-7; 7]$ .
- Добавить на форму компонент для выбора алгоритма сглаживания: по 3 точкам или по 5 точкам.
- Процедура обработки сигнала должна выполнить линейное сглаживание функции  $Y(x)$  по 3 или 5 точкам. Под графиком  $Y(x)$  должен быть отображен результат обработки  $\tilde{Y}(x)$ .
- Графики  $Y(x)$  и  $\tilde{Y}(x)$ : отсчеты должны быть соединены линиями.

**Вариант № 6. Медианный фильтр.**

- Массив **Y** заполнить в соответствии с формулой:

$$Y_x = 7 \cdot \left( 11 + \frac{\sin \frac{x+1}{4}}{\sin \frac{x+1}{40}} \right), \quad x \in [0; 199], \text{ шаг изменения } x: 1.$$

- К каждому значению из массива **Y** прибавить случайное число из диапазона  $[-5; 5]$ .
- Добавить поле ввода для задания размера (целое число не меньше 3) апертуры фильтра.
- Процедура обработки сигнала должна выполнить медианную фильтрацию функции  $Y(x)$  с учетом введенного размера апертуры фильтра. Под графиком  $Y(x)$  должен быть отображен результат обработки  $\tilde{Y}(x)$ .
- Графики  $Y(x)$  и  $\tilde{Y}(x)$ : отсчеты должны быть соединены линиями.

**Вариант № 7. Линейная интерполяция.**

- Массив **X** заполнить значениями из интервала  $[0; 200]$  с шагом 50 (длина массива равна 5).

- Массив **Y** заполнить в соответствии с формулой:  $Y_x = \frac{(x-100)^2}{80}$

(длина массива равна 5).

- Добавить поле ввода для значения «шаг». Величина шага задается целым числом из интервала  $[1; 50]$ .

- Процедура обработки сигнала должна выполнить линейную интерполяцию  $Y(x)$  для значений  $x$ , принадлежащих интервалам:  $[0; 50)$ ,  $[50; 100)$ ,  $[100; 150)$ ,  $[150; 200]$ . Перебор значений  $x$  выполняется с шагом из поля «шаг». Под графиком  $Y(x)$  должен быть отображен результат обработки  $\tilde{Y}(x)$ .

- Графики  $Y(x)$  и  $\tilde{Y}(x)$ : отсчеты должны быть выведены на плоскость в виде закрашенных кружков.

**Вариант № 8. Аппроксимация.**

- Массив **X** заполнить значениями из интервала  $[0; 200]$  с шагом 2 (длина массива равна 101).

- Массив **Y** заполнить в соответствии с формулой:  $Y_x = 0.5 \cdot x + 50$  (длина массива равна 101).

- Добавить поле ввода для задания диапазона генерации случайного числа. Значение  $r$  задается целым числом из интервала  $[1; 50]$ .

- К каждому значению из массива **Y** прибавить случайное число из диапазона  $[-r; r]$ .

- Процедура обработки сигнала должна выполнить аппроксимацию  $Y(x)$  линейной функцией. График полученной функции  $\tilde{Y}(x) = k \cdot x + b$  вывести поверх графика  $Y(x)$ .

- Обеспечить вывод на экран найденных коэффициентов  $k$  и  $b$ .

- График  $Y(x)$ : отсчеты должны быть выведены на плоскость в виде закрашенных кружков.

- График  $\tilde{Y}(x)$ : отсчеты должны быть соединены линиями. Построение функции осуществляется для  $x \in [0; 200]$  с шагом 1.



#### 4. Порядок выполнения работы

1. Создать в среде **Delphi** новый проект.
2. Поместить на форму компоненты для ввода величин, если это предусмотрено в варианте задания.
3. Поместить на форму активные элементы:
  - а) кнопка «Последовательность»;
  - б) кнопка «Обработка».
4. Создать процедуру генерации последовательности чисел согласно варианту задания.
5. Создать процедуру построения графика на основе последовательности чисел согласно варианту задания. Должны быть нанесены оси координат, на каждой оси отмечаются минимальное и максимальное значения (для  $x$  и  $y$  соответственно).
6. Создать процедуру расчета **математического ожидания, дисперсии, среднеквадратического отклонения** для последовательности. Результат необходимо вывести в специальные компоненты на форме или в виде сообщения.
7. Создать процедуру обработки последовательности согласно варианту задания.
8. Создать обработчик нажатия на кнопку «Последовательность», в котором будут вызываться процедуры из пп. 4, 5, 6.
9. Создать обработчик нажатия на кнопку «Обработка», в котором будет вызываться процедура из п. 7.

#### 5. Пример выполнения лабораторной работы

1. Для выполнения лабораторной работы необходимо создать новый VCL проект в среде Delphi.

Размер главной формы рекомендуется установить не менее 500 на 500 пикселей, чтобы обеспечить отображение всех требуемых графиков.

2. Согласно пп. 2 и 3 порядка выполнения и варианту задания на форму помещаются необходимые компоненты управления и ввода величин. Кнопку «Обработка» по умолчанию сделать недоступной.

Рекомендуется располагать компоненты в верхней части формы, а нижнюю часть оставить для отображения графиков.

3. Создается процедура генерации последовательности согласно заданному закону.

Для вариантов **1 – 6** создается массив **Y**, массив **X** создавать не требуется, так как значению  $x$  будет соответствовать индекс массива.

Пример генерации последовательности (варианты **1 – 6**) для  $Y_x = x^2 \cdot \sin x$ ,  $x \in [0; 199]$ , шаг изменения  $x$ : 1.

```
...
var
  FormLR6: TFormLR6;
  arrY: array [0..199] of real;
...
procedure Generate();
var
  x : integer;
begin
  Randomize;
  //Заполнение массива Y
  for x := 0 to 199 do
    begin
      //Вычисление y(x) согласно заданному закону
      arrY[x] := power(x,2)*sin(x);
      //Добавление случайного числа (если задано)
      // Например, из диапазона [-49;49]
      arrY[x] := arrY[x] + Random(99)-49;
    end;
  end;
```

Для вариантов **7 и 8** создается массив для **Y** и для **X**, так как шаг изменения  $x$  отличен от 1.

Пример генерации последовательности (варианты **7 и 8**) для  $Y_x = x^2 \cdot \sin x$ ,  $x \in [0; 250]$ , шаг изменения  $x$ : 5.

```
...
var
  FormLR6: TFormLR6;
  //Длина массивов равна 51 (индексы с 0 по 50)
  arrX, arrY: array [0..50] of real;
...
procedure Generate();
var
  i : integer;
```

```

    x : real;
begin
    Randomize;
    //Начальное значение для x - ноль
    x:=0;
    //Заполнение массивов
    for i := 0 to 50 do
    begin
        //Заполнение X
        arrX[i]:=x;

        //Вычисление y(x) согласно заданному закону
        arrY[i]:=power(x,2)*sin(x);

        //Добавление случайного числа (если задано)
        // Например, из диапазона [-49;49]
        arrY[i]:=arrY[i] + Random(99)-49;

        //Увеличение значения x
        // Например, шаг изменения x равен 5
        x:=x+5;
    end;
end;

```

4. Далее создаются следующие процедуры.

- Для очистки области вывода графиков.
- Для отображения последовательности чисел.
- Для вычисления математического ожидания, дисперсии, средне-квадратического отклонения для последовательности чисел.
- Для обработки последовательности согласно варианту задания.

При отображении графиков при выполнении всех вариантов заданий можно учесть, что минимальное значение функции не может быть меньше 0, а максимальное не может быть больше 200.

5. Создается обработчик нажатия на кнопку «Последовательность».

При нажатии на кнопку «Последовательность» должны быть совершены следующие действия.

- Очистка области вывода графиков и других элементов, предназначенных для вывода результатов.
- Генерация последовательности.
- Графическое отображение последовательности.
- Вычисление математического ожидания, дисперсии, средне-квадратического отклонения.

• Отображение вычисленных значений на форме. Отображение осуществляется любым доступным способом.

```

Procedure      TFormLR6.ButtonSequenceClick      (Sender:
TObject);
var
    mato,disp,sko : real;
begin
    //Очистка формы
    ClearGraphic();

    //Генерация последовательности
    Generate();

    //Отображение последовательности
    // Параметры: смещение графика по x и y
    // относительно левого верхнего угла формы
    DrawSeq(50,100);

    //Вычисление характеристик
    CalcCommonVals(mato,disp,sko);
    //Отображение характеристик на форме
    ...

    //Делаем кнопку «Обработка» доступной
    ButtonProcess.Enabled:=True;
end;

```

6. Создается обработчик нажатия на кнопку «Обработка».

При нажатии на кнопку «Обработка» должна быть вызвана процедура обработки сформированной последовательности чисел. На данном этапе работа программы может быть разделена на части, например:

- считывание необходимых параметров из полей ввода;
- обработка сигнала;
- дополнительные действия (построение гистограммы, отображение коэффициентов и т.п.);
- графическое отображение результата обработки.

## Содержание

ВВЕДЕНИЕ .....	1
ЛАБОРАТОРНЫЕ РАБОТЫ № 1, 2. РАБОТА С ГРАФИКОЙ .....	2
1. Цель работы.....	2
2. Графические возможности Delphi .....	2
3. Класс TCanvas – канва (холст).....	2
4. Класс TFont – шрифт .....	9
5. Класс TPen – перо .....	10
6. Класс TBrush – кисть .....	12
7. Порядок выполнения работы № 1 .....	14
8. Порядок выполнения работы № 2 .....	14
9. Пример выполнения лабораторной работы .....	15
ЛАБОРАТОРНАЯ РАБОТА № 3. РАБОТА С ДАТОЙ И ВРЕМЕНЕМ .....	19
1. Цель работы.....	19
2. Тип дата-время .....	19
3. Компонент TTimer .....	22
4. Порядок выполнения работы .....	22
5. Пример выполнения лабораторной работы.....	23
ЛАБОРАТОРНАЯ РАБОТА № 4. ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ СОРТИРОВКИ В СРЕДЕ DELPHI .....	28
1. Цель работы.....	28
2. Методы сортировки .....	28
2.1. Метод прямого обмена с флагом (метод «пузырька»).....	30
2.2. Метод прямого обмена с подвижной границей .....	33
2.3. Метод шейкера .....	35
2.4. Метод прямого включения .....	36
2.5. Метод прямого выбора .....	37
3. Порядок выполнения работы .....	39
4. Пример выполнения лабораторной работы .....	39
ЛАБОРАТОРНАЯ РАБОТА № 5. ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ПОИСКА В СРЕДЕ DELPHI .....	44
1. Цель работы.....	44
2. Методы поиска .....	44
2.1. Бинарный поиск .....	44
2.2. Метод бисекции.....	44
2.3. Метод Ньютона .....	46
2.4. Метод золотого сечения .....	48
3. Варианты заданий .....	50
4. Порядок выполнения работы .....	50

4.1. Для варианта № 1 .....	50
4.2. Для вариантов № 2 и № 3 .....	51
4.3. Для варианта № 4 .....	51
5. Пример выполнения лабораторной работы .....	52
<b>ЛАБОРАТОРНАЯ РАБОТА № 6. ПРОГРАММИРОВАНИЕ</b>	
<b>АЛГОРИТМОВ ОБРАБОТКИ ОДНОМЕРНЫХ СИГНАЛОВ В СРЕДЕ</b>	
<b>DELPNI</b> .....	55
1. Цель работы .....	55
2. Цифровая обработка сигналов .....	55
2.1. Характеристики случайных величин .....	56
2.2. Гистограмма .....	57
2.3. Линейная свертка .....	60
2.4. Корреляция .....	61
2.5. Интерполяция .....	62
2.6. Аппроксимация .....	62
2.7. Сглаживание данных .....	64
2.8. Медианный фильтр .....	65
3. Варианты заданий .....	67
4. Порядок выполнения работы .....	71
5. Пример выполнения лабораторной работы .....	71