МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Белорусский национальный технический университет

Кафедра «Гидропневмоавтоматика и гидропневмопривод»

С. В. Ермилов
М. И. Жилевич
Л. Г. Филипова

ИНФОРМАТИКА

Учебно-методическое пособие
к выполнению лабораторных работ
для студентов специальности 6-05-0715-04
«Гидропневмосистемы мобильных
и технологических машин и оборудования»

Рекомендовано учебно-методическим объединением по образованию в области транспорта и транспортной деятельности

Минск
БНТУ
2024

УДК 004(076.5)(075.8)
ББК 32.81я7
Е73

Р е ц е н з е н т ы:
кафедра «Тракторы и автомобили» УО «БГАТУ»
(зав. кафедрой, канд. техн. наук, доцент *Г. И. Гедроить*);
начальник сектора мобильной гидравлики
ИП «Линтера ТехСервис», канд. техн. наук *Е. М. Заболоцкий*

Ермилов, С. В.

Е73 Информатика : учебно-методическое пособие к выполнению лабораторных работ для студентов специальности 6-05-0715-04 «Гидропневмосистемы мобильных и технологических машин и оборудования» / С. В. Ермилов, М. И. Жилевич, Л. Г. Филипова. – Минск : БНТУ, 2024. – 46 с.

ISBN 978-985-583-971-3.

Данное пособие предназначено для помощи в выполнении лабораторных работ по дисциплине «Информатика» и содержит необходимые теоретические сведения и практические рекомендации, позволяющие программировать на ЭВМ различные вычислительные процессы в среде программирования PascalABC.NET.

Пособие может быть полезно студентам для выполнения расчетов в ходе курсового и дипломного проектирования.

УДК 621.113:681.3(075.8)
ББК 39.33

ISBN 978-985-583-971-3

© Ермилов С. В., Жилевич М. И., Филипова Л. Г., 2024
© Белорусский национальный технический университет, 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Одним из существенных путей повышения эффективности проектирования и эксплуатации гидравлического и пневматического оборудования в настоящее время является использование персональных компьютеров (ПК). Причем ПК применяются не только на стадии научных исследований, но и на этапах проектно-конструкторских работ, для оформления технической документации, сбора и анализа статистической информации при испытаниях, эксплуатации и диагностировании гидро- и пневмоприводов.

Целью курса «Информатика» является обучение студентов методике постановки, подготовки, алгоритмизации и решения задач по расчету и проектированию гидро- и пневмоприводов на современных ПК.

Основное назначение данного пособия – получение студентами практических навыков работы в среде программирования PascalABC.NET, являющейся визуальной средой разработки приложений с поддержкой объектно-ориентированных технологий. Основными преимуществами PascalABC.NET являются низкие системные требования, по сравнению, например, с Visual Studio, что позволяет использовать его на ПК практически любой конфигурации, и платформа Microsoft .NET Framework, дающая возможность быстрого перехода на язык C#.

Дисциплина «Информатика» относится к естественнонаучному модулю, на ней базируется изучение специальных дисциплин, таких как «Математическое моделирование производственных процессов», «Автоматизированное проектирование гидропневмоприводов», «Теория и проектирование гидропневмосистем», «Техническая диагностика гидропневмосистем» и др. Расчеты на ПК выполняются в ходе курсового и дипломного проектирования.

Среда PascalABC распространяется бесплатно. Последнюю версию можно скачать на сайте http://pascalabc.net/ в разделе «Скачать».

1. ИНТЕРФЕЙС СРЕДЫ РАЗРАБОТКИ PascalABC.NET

PascalABC.NET – это система программирования и язык Pascal нового поколения для платформы Microsoft .NET. Система PascalABC.NET содержит все основные элементы современных языков программирования: модули, классы, перегрузку операций, интерфейсы и т. д.

После загрузки PascalABC на дисплее появляется основной экран среды (рис. 1), состоящий из следующих компонентов: главное окно программы, основное меню, область программного кода, окно вывода.

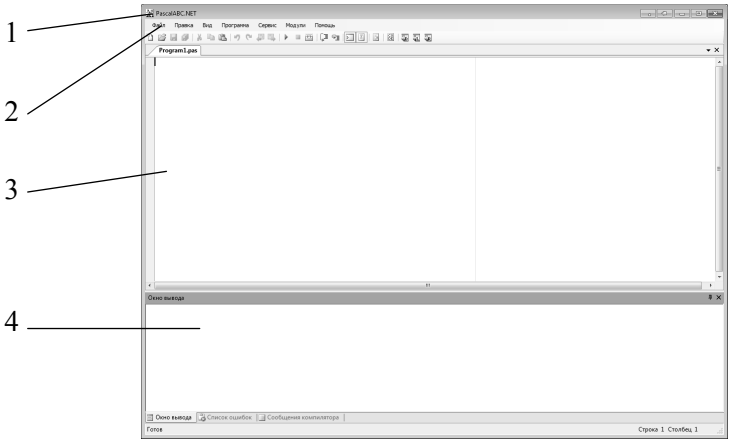


Рис. 1. Основной экран среды PascalABC:
1 – главное окно программы; *2* – основное меню;
3 – область программного кода; *4* – окно вывода

Главное окно программы предназначено для управления процессом разработки программы и всегда присутствует на экране.

Основное меню содержит компоненты для управления программой. Пиктограммы упрощают доступ к наиболее часто применяемым командам основного меню.

Область программного кода позволяет создавать простейшие алгоритмы с использованием встроенного редактора текстов, компилировать их, выполнять, проводить отладку.

Окно вывода предназначено для печати результатов расчета, списка ошибок, сообщений компилятора и пр.

Для работы с формами в среде PascalABC необходимо создать новый проект, выполнив: Файл → Новый проект. В открывшемся окне (рис. 2) необходимо выбрать тип проекта «Приложение Windows Forms», а также указать его имя и адрес расположения в соответствующих строках.

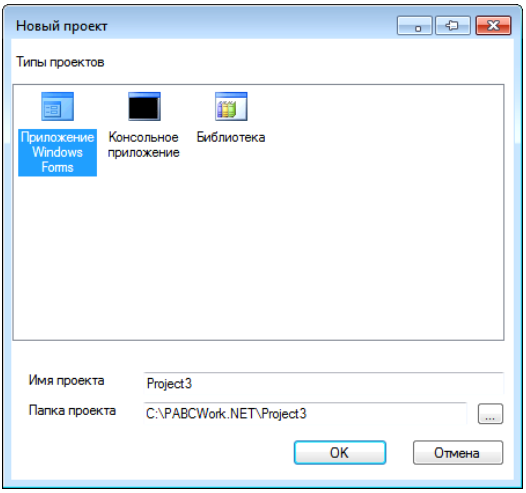


Рис. 2. Диалоговое меню выбора типа проекта

Стандартный интерфейс среды PascalABC после создания нового проекта представлен на рис. 3. Расположение, количество и размер окон при необходимости может быть изменено пользователем.

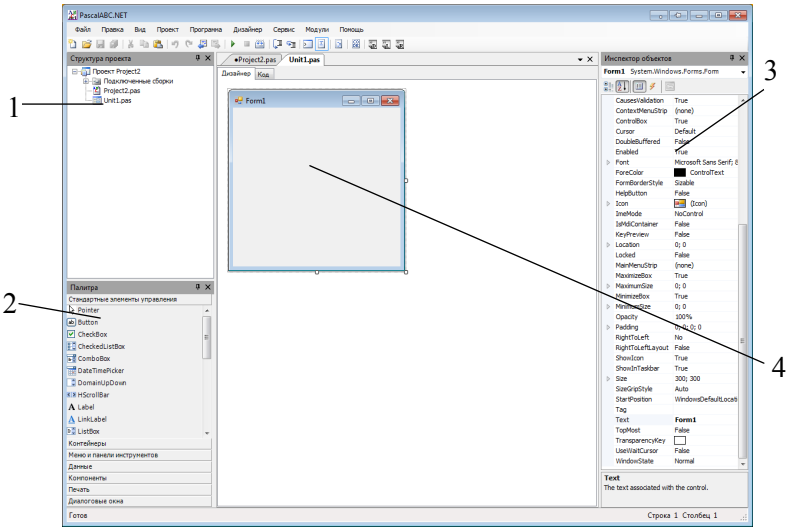


Рис. 3. Стандартный интерфейс среды PascalABC:
1 – окно структуры проекта; *2* – меню компонентов (палитра);
3 – окно инспектора объектов; *4* – окно формы

Окно структуры проекта отображает все файлы, входящие в текущий проект и расположенные в рабочей папке.

Меню компонентов (**палитра**) содержит элементы управления, которые используются для создания интерфейса пользователя.

Окно инспектора объектов (рис. 4) по умолчанию отображает свойства объекта (вкладка Properties). Изменение свойств выбранного элемента позволяет настроить его цвет, размеры, координаты расположения и т. д. Некоторые свойства объединены в группы, что упрощает доступ к ним. Пример – группа свойств *Font* (Шрифт) включает такие элементы, как *Name* (Имя шрифта), *Size* (Размер шрифта) и др. Также в данном окне при переключении во вкладку *Events* можно задать реакцию выбранного компонента на то или иное событие (например, нажатие клавиши мыши, изменение размеров графического окна и др.).

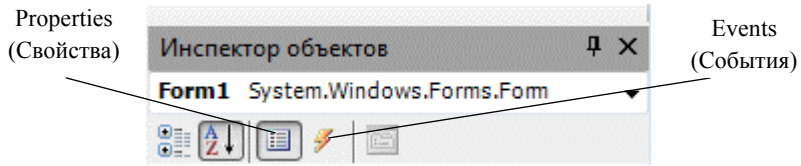


Рис. 4. Вкладки Инспектора объектов

Окно формы является основой для создания будущего интерфейса приложения. В данной области в ходе написания программы размещаются требуемые компоненты. При запуске и работе программы помещенные на форму объекты будут иметь тот же вид, что и на этапе проектирования.

Нажатие на вкладку «Код» открывает **окно текста** программы для написания и редактирования программного кода проекта. При создании проекта окно текста программы заполняется средой программирования без участия пользователя минимальным необходимым набором операторов для нормального функционирования пустой формы в качестве Windows-окна. Размещение на окне формы каких-либо компонентов в автоматическом режиме дополняет текст программы необходимыми библиотеками стандартных программ и типами данных в разделах *uses* и *type* соответственно.

2. ОБЩАЯ СТРУКТУРА ПРОЕКТА В СРЕДЕ PascalABC.NET ПРИ РАБОТЕ С ФОРМАМИ

Приложения в среде разработки PascalABC включают файлы проекта (.pabcproj), исходного текста (.pas) и содержимого формы (.inc).

Файл проекта содержит информацию о подключенных в разрабатываемом проекте модулях. Файл проекта создается и изменяется средой автоматически и не предназначен для редактирования пользователем. Предназначен для открытия проекта в среде программирования.

Файл исходного текста является программным модулем, в котором размещается текст программы, написанной на языке Pascal.

Автоматически сгенерированный модуль имеет следующую структуру:

```
unit Unit1;              // заголовок модуля
interface                // раздел объявлений
        код раздела;
implementation           // раздел реализации
        код раздела;
end.
```

Заголовок модуля является обязательной строкой, открывающей программный код модуля. **Имя модуля** (*Unit1*) обязательно должно совпадать с **именем файла** (*Unit1.pas*).

В **разделе объявлений** описываются типы, переменные, заголовки процедур и функций, которые будут доступны другим модулям посредством раздела подключаемых модулей *Uses*.

В **разделе реализации** располагаются программные коды процедур и функций, описанных в разделе объявлений. Также в данном разделе описываются типы переменных, процедуры и функции, которые будут работать только в пределах данного модуля.

При компиляции среда PascalABC создает файл с расширением .pcu, содержащий в себе результат перевода в машинные коды содержимого файлов с расширениями .pas и .inc. Компоновщик преобразует файлы с расширением .pcu в единый загружаемый файл с расширением .exe.

Программа для работы с формами Windows представляет собой набор алгоритмов, которые выполняются при срабатывании определенного события (например, щелчок мышью по кнопке – событие *OnClick*). Каждое обрабатываемое событие реализуется с помощью страницы *Events* инспектора объектов, автоматически создающей в тексте программы процедуру (*procedure*), между ключевыми словами *begin* и *end* которой программист записывает на языке Pascal требуемый алгоритм.

Процедура является самостоятельным блоком программного кода, который может быть вызван для выполнения в любом месте программы и состоит из имени и тела процедуры. Процедура, автоматически создаваемая средой, имеет вид:

```
//заголовок процедуры
procedure имя процедуры (формальные параметры процедуры);
// тело процедуры
begin
        операторы;
end;
```

Имя и формальные параметры автоматически сгенерированных процедур задаются средой программирования и не редактируются пользователем.

В теле процедуры записываются операторы, которые выполняются при вызове процедуры.

Между заголовком процедуры и ее телом могут располагаться разделы:

```
uses ...;       // раздел подключаемых библиотечных модулей
label ...;      // раздел меток
const ...;      // раздел констант
type ...;       // раздел определения типов данных
var ...;        // раздел описания переменных
```

Любой из разделов может отсутствовать. Разделителем между разделами и операторами является знак точка с запятой.

Раздел подключаемых модулей состоит из слова ***uses*** и списка подключаемых стандартных и пользовательских библиотечных модулей.

Например, *uses GraphABC*; (подключение графического пакета).

Раздел ***label*** используется для описания так называемых меток. Перед любым оператором можно поставить метку, что позволяет выполнить переход на этот оператор из любого места блока с помощью оператора безусловного перехода ***goto***. Метка состоит из имени и следующего за ним двоеточия. Именем может служить идентификатор или число. Максимальная длина имени метки – 127 символов. Если именем метки является идентификатор, то перед употреблением ее необходимо описать. Имена нечисловых меток записывают через запятую после слова ***label.*** За последним именем ставится точка с запятой. Все метки, описанные в разделе описания, должны быть использованы в программе.

Например:

```
 label m1;
begin
        m1: оператор;        // «помеченный» оператор
...
        goto m1;             // переход на метку
end;
```

В разделе описания констант производится присвоение идентификаторам констант постоянных значений. Раздел начинается словом ***const***, после которого следует ряд выражений, присваивающих идентификаторам постоянные числовые или строковые значения.

Например: *const Tel*1 = 'Иванов'; *g* = 9.81; .

Раздел описания пользовательских типов данных начинается словом ***type***, за которым следуют одно или несколько определений типов, разделенных точкой с запятой. Чаще всего применяется для описания структурированных и процедурных типов данных.

Например: *type mas = array* [1..10] *of real*; .

Раздел описания переменных начинается словом ***var***. Затем через запятую перечисляются имена переменных, а через двоеточие следует их тип, причем в конце описания каждого типа ставится точка с запятой.

Например: *var A*1*, b, C: integer; TEV, S: real; L: boolean;*.

3. ТИПЫ ДАННЫХ В PascalABC.NET

Типы данных в Pascal определяют возможные значения переменных, констант, выражений и функций. Они бывают встроенными и пользовательскими. Встроенные типы изначально присутствуют в языке программирования, а пользовательские создаются программистом.

По способу представления и обработки типы данных бывают: простые; структурированные; указатели; процедуры.

Наиболее известные **простые** типы это: целое число (*Integer*), дробное число (*Real*), символьный (*Char*), логическое значение (*Boolean*).

В случае **структурированных** типов под одним ключевым словом группируется несколько совместных значений, таких, например, как координаты точки или имя и фамилия человека. В таком виде набор данных разом легче передавать. В то же время использовать или изменять данные внутри структуры приходится по одному. К структурированным типам относят массивы, записи, списки и т. д.

Указатель содержит адрес байта памяти, в котором находится значение данных определенного типа. Этот тип называют также ссылочным. Применение указателей является гибким средством управления динамической памятью и предоставляет возможность обработки массивов данных большой размерности.

Процедурный тип предназначен для хранения ссылок на процедуры или функции. Основное назначение процедурных переменных – хранение и косвенный вызов действий (функций) в ходе выполнения программы и передача их в качестве параметров.

4. РЕДАКТИРОВАНИЕ ФОРМЫ

Новая форма имеет одинаковые имя (*Name*) и заголовок (*Text*) – Form1. Заголовок может быть изменен в окне инспектора объектов. Для этого необходимо щелкнуть кнопкой мыши по форме и во вкладке Properties инспектора объектов отредактировать свойство *Text* в правой ячейке.

Размер созданной формы может быть отрегулирован с помощью мыши, если «захватить» курсором ее правую или нижнюю кромку.

Создание пользовательского интерфейса, а также перемещение и изменение уже имеющихся элементов на форме осуществляется методом Drag'n'Drop (в переводе с английского означает буквально «тащи-и-бросай»).

Например, для размещения на форме элемента управления «Кнопка» (*Button*) необходимо зажать левую кнопку мыши на соответствующем элементе в меню компонентов «Палитра» и перетащить его в нужное место, после чего отпустить кнопку мыши. На форме появится выбранный элемент управления с маркерами изменения размера (рис. 5).

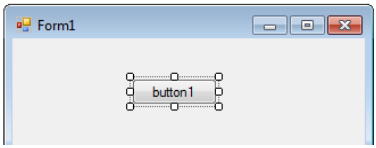


Рис. 5. Размещение кнопки на форме

Элемент управления, помещенный из панели «**Палитра**» на форму, превращается в объект с набором свойств по умолчанию, таких как имя, заголовок, координаты размещения на форме, и т. д.

Таким образом, элемент управления *Button* становится объектом *Button1* (кнопкой) и будет иметь свойства, предписанные ему по умолчанию.

Значения свойств могут быть отредактированы после создания объекта на форме, для чего необходимо его выделить, щелкнув по нему мышью. Тогда во вкладке Properties панели «Инспектор объектов» отображаются свойства выбранного объекта, которые могут быть настроены.

На форму можно поместить несколько экземпляров элементов управления одного класса. Например, несколько кнопок, каждая из которых будет иметь свои индивидуальные свойства.

Для удаления элемента управления необходимо его выделить и нажать клавишу *Delete*.

После создания проекта программный код для формы и элементов управления, размещенных на ней, генерируется автоматически.

5. РЕДАКТОР КОДА. ПРОГРАММНЫЙ КОД ПРОЕКТА

Просмотр, написание, редактирование текста программы осуществляется во вкладке «Код».

В окне редактора кода находится текст программы, сгенерированный конструктором форм Windows. Он изменяется автоматически, если добавлять на форму элементы управления.

Код, который пишется программистом вручную, чаще всего находится в обработчиках событий элементов управления формы.

Чтобы написать программный код для обработчика события, которое идет по умолчанию, нужно выбрать этот элемент на форме и выполнить по нему двойной щелчок. Откроется окно редактора кода, где сгенерируется заготовка процедуры обработчика события (пустая процедура, содержащая заголовок и тело).

Например, двойное нажатие на кнопку *Button* создает событие *Click* (используется преимущественно для вычислений и вывода результата), а на форму – *Load* (позволяет заполнить элементы формы начальными значениями после запуска программы).

Чтобы реализовать простейший линейный алгоритм, необходимы: раздел объявления переменных, функции ввода и вывода данных, оператор присваивания.

Ввод, вывод данных и информации, вмещающихся в одну строку, позволяет выполнить однострочное окно редактирования (компонент *TextBox*). Отредактировав свойство *Text*, можно изменить отображаемую в окне информацию (строка из символов, имеющая тип *String*).

Компонент *Label* позволяет добавлять на форму пояснительные надписи. Имеет аналогичное компоненту *TextBox* свойство *Text* для редактирования.

Если необходимо вывести результаты работы программы в виде подробного отчета, состоящего из нескольких строк текста, используется текстовое окно (например, компонент *ListBox*).

Информация, отображаемая построчно в окне *ListBox*, находится в свойстве *ListBox1.Items.* Метод ***ListBox1.Items.Add*** позволяет до-

бавить новую строку (переменную типа *String*). Очистка окна в ходе работы программы осуществляется методом ***ListBox1.Items.Clear***. Редактирование списка после запуска программы невозможно.

Оператор присваивания тождественен знаку равно в математической записи выражения. Оператор присваивания имеет вид «:=» (двоеточие и равно). При выполнении оператора присваивания вычисляется выражение, стоящее в правой части, и его значение присваивается переменной в левой части. Тип результата выражения должен соответствовать типу переменной в левой части.

Например, *m* := 16; или *m* := 2 * sin(π/4).

Выражение задает порядок выполнения действий над элементами данных и состоит из операндов (констант, переменных, обращений к функциям), круглых скобок и знаков операций.

Арифметические операции выполняют арифметические действия в выражениях над операндами целого и вещественного типов. К основным арифметическим операциям можно отнести следующие: «+» – сложение; «–» – вычитание; «*» – умножение; «/» – деление; *div* – целочисленное деление; *mod* – остаток от целочисленного деления; унарный минус – отрицание знака.

Выполнение каждой операции происходит с учетом их приоритета: сначала операции типа умножения (*, /, *div*, *mod*), а затем операции типа сложения (+, –). Операции одного и того же старшинства выполняются слева направо в порядке их появления в выражении. Выражения в круглых скобках вычисляются в первую очередь. В выражения могут входить **стандартные (встроенные) функции**, вычисляемые до выполнения арифметических операций.

Переменные, входящие в выражение, являются данными числового типа (*Real* или *Integer*). Преобразовать запись строкового типа, находящуюся, например, в переменной *Edit*1.*Text*, в вещественный позволяет стандартная функция ***StrToFloat*(*s*: *String*): *Real***.

Например, *b* := *StrToFloat(Edit*1.*Text);*.

Для преобразования текстовой информации в данные целого типа используется функция ***StrToInt*(*s*: *String*): *Integer***.

Обратными функциями, преобразующими данные типа *Real* и *Integer* в строковый тип (*String)*, соответственно являются ***FloatToStr*(*a*: *Real*): *String*** и ***IntToStr*(*a*: *Integer*): *String***.

Например, команда *ListBox1.Items.Add(FloatToStr(4,75));*.

6. ОПЕРАТОРЫ ЯЗЫКА PASCAL

Оператор является неделимым элементом программы, который дает возможность выполнять определенные алгоритмические действия. Отличием оператора, по отношению к другим элементам, является то, что под ним всегда подразумевается какое-то действие. В языке Pascal операторы состоят из служебных слов. Операторы, используемые в программе, отделяются между собой и от других элементов программы символом «;» (точка с запятой). Все операторы языка Pascal можно условно разбить на две группы: простые и структурированные.

Простые операторы – это операторы, не содержащие в себе других операторов. К ним относятся: оператор присваивания (**:=**); обращение к подпрограмме; оператор безусловного перехода (***goto***).

Структурированные операторы – это операторы, которые содержат в себе другие операторы. К ним относятся: составной оператор; операторы условий (***if***, ***case***); операторы цикла (***for***, ***while***, ***repeat***).

7. ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ АЛГОРИТМОВ

Линейным называется алгоритм, в котором результат получается путем однократного выполнения заданной последовательности действий при любых значениях исходных данных. Операторы будут задействованы последовательно, один за другим, в соответствии с их расположением в тексте программы.

Например, составить программу вычисления выражения для заданных значений *x* и *z*:

$$q = \frac{\left(e^{4z} + 2x\right)^3}{\sqrt{\left|z\,x^2 - 8\right|}} \ln\left(x \cdot arctg\left(z\right)\right).$$

Алгоритм решения задачи:

1. Загрузить PascalABC и создать новый проект.

2. Разместить на форме следующие элементы: *Label* – четыре, *TextBox* – два, *ListBox* – один, *Button* – один. На рис. 6 показан примерный вид итоговой формы.

3. С помощью Инспектора объектов задать следующие значения свойства *Text* (без кавычек): кнопка *Button*1 – «Выполнить»; метки: *label*1 – «Значение переменной *x*»; *label*2 – «Значение переменной *z*»; *label*3 – «Результат выполнения программы:»; *label*4 – «Исходные данные:».

4. Создать событие для формы *Load* (два раза щелкнуть левой кнопкой мыши по форме). Между операторных скобок *begin* и *end* необходимо написать текст, заполняющий элементы *textBox1* и *textBox2* и очищающий *listBox1*. Пример вводимого текста выделен жирным шрифтом в процедуре ***Form1.Form1_Load***.

```
procedure Form1.Form1_Load(sender: Object; e: EventArgs);
begin
  listBox1.Items.Clear;  // Очистка окна редактора listBox1
  textBox1.Text := '4,5';  // Начальное значение x
  textBox2.Text := '16,8';  // Начальное значение z
end;
```

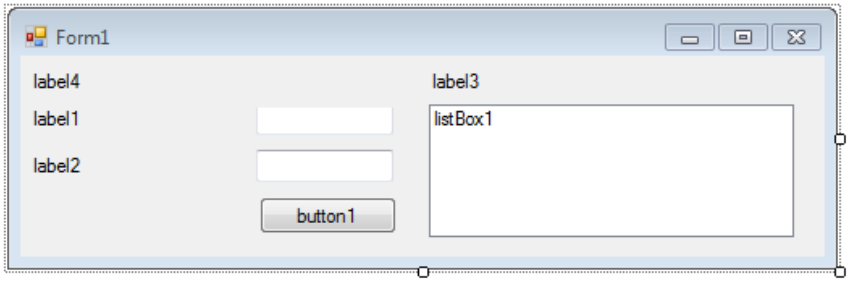


Рис. 6. Пример дизайна формы.

5. Создать событие *Click* для кнопки (два раза щелкнуть левой кнопкой мыши по кнопке *button*1) и записать последовательность операторов, позволяющих вычислить заданное выражение. Пример вводимого текста выделен жирным шрифтом в процедуре ***Form1.button1_Click***.

```
procedure Form1.button1_Click(sender: Object; e: EventArgs);
var a, b, c, d, f, x, z, q: real;
begin
  listBox1.Items.Clear;
  x := StrToFloat(textBox1.Text);
  z := StrToFloat(textBox2.Text);
  a := exp(4 * z)+ 2 * x;
  b := power(a, 3);
  c := abs(z * sqr(x) - 8);
  d := sqrt(c);
  f := ln(x * arctan(z));
  q := c / d * f;
  listBox1.Items.Add('Исходные данные:');
  listBox1.Items.Add('Переменная x = ' + FloatToStr(x));
  listBox1.Items.Add('Переменная z = ' + FloatToStr(z));
  listBox1.Items.Add('Результат расчета:');
  listBox1.Items.Add('q = ' + FloatToStr(q));
end;
```

8. ПРОГРАММИРОВАНИЕ РАЗВЕТВЛЯЮЩИХСЯ АЛГОРИТМОВ

Алгоритм называется разветвленным, если он содержит несколько ветвей, отличающихся друг от друга содержанием вычислений. Выход вычислительного процесса на ту или иную ветвь алгоритма определяется исходными данными задачи. На языке Pascal разветвляющийся алгоритм можно реализовать при помощи переменных типа *Boolean* (принимают значения *true* (правда) и *false* (ложь)) и операторов условного (*if*) и безусловного (*goto*) перехода и множественного ветвления (*case*).

Существуют две формы записи условного оператора *if*:

1) ***if*** условие ***then*** op1
   else op2;
2) ***if*** условие ***then*** op3;

где условие – логическое выражение;

op1, op2, op3 – операторы языка Pascal, причем на их месте может стоять и составной оператор, т. е. последовательность операторов, заключенных в операторные скобки *begin ... end*.

Суть действия оператора заключается в следующем.

Для первого вида: если *условие* выполняется (значение логического выражения – истина, или *true*), то выполняется *оператор 1*, а *оператор 2* пропускается; иначе (*условие* не выполняется (значение логического выражения – ложь, или *false*)) *оператор 1* не выполняется, а выполняется *оператор 2*, следующий за *else*. Иными словами, **если (*if*)** соблюдается некоторое *условие,* **то** (***then***) выполняй *оператор 1*, **иначе** (***else***) выполняй *оператор 2*.

Например,

if $X > Y$ *then* $X := 5.1$ *else* $Y := 10.0$;

Пусть из предварительных расчетов известно, что $X = 10$ и $Y = 8$. Логическое выражение $X > Y$ – истинно, поэтому X присваивается новое значение 5.1, а Y останется без изменения.

Для случая, когда $X = 10$ и $Y = 20$, условие $X > Y$ не выполняется, т. е. значение логического выражения – ложь; поэтому X остается без изменений и выполняется оператор присвоения $Y := 10.0$.

Для второго вида записи оператор ***if*** работает следующим образом: если *условие* соблюдается, то выполняется *оператор 3*, следую-

щий за ***then***, если значение выражения – ложь, тогда выполняется оператор, следующий сразу же за оператором ***if***. Иными словами, **если (*if*)** *условие* соблюдается, **то** (***then***) выполняй *оператор 3*.

Например,

if $X > Y$ ***then*** $X := 10.0$; $Y := 20.0$;

Пусть $X = 3$ и $Y = 1$, тогда логическое выражение $X > Y$ принимает значение истина (условие выполняется). Поэтому далее выполняется оператор присвоения $X := 10.0$, а затем и оператор присвоения $Y := 20.0$, следующий за оператором условного перехода.

Для случая, когда $X = 3$ и $Y = 5$, условие $X > Y$ не выполняется. Поэтому X остается без изменений ($X = 3$), а выполняется оператор, следующий за ***if***, т. е. $Y := 20$.

Оператор ***if*** может быть **вложенным**, т. е. на месте *op*1, *op*2 или *op*3 может стоять другой оператор ***if***.

Например,

```
if условие1 then op1
else
        if условие2 then op2
        else op3;
```

Оператор выбора *case* позволяет сделать выбор из произвольного числа имеющихся вариантов. Он состоит из выражения, называемого *селектором*, и списка параметров, каждому из которых предшествует список *констант выбора* (список может состоять из одной константы). Как и в операторе ***if***, здесь может присутствовать слово ***else***, имеющее тот же смысл.

Общая форма записи:

```
case выражение-селектор of
  список констант выбора 1: оператор 1;
  список констант выбора 2: оператор 2;
  .....
  список констант выбора N: оператор N
else оператор
end;
```

Оператор *case* работает следующим образом. Сначала вычисляется значение выражения-селектора, затем обеспечивается реализация того

оператора, константа выбора которого равна текущему значению селектора. Если ни одна из констант не равна текущему значению селектора, выполняется оператор, стоящий за словом *else*. Если слово *else* отсутствует, активизируется оператор, находящийся за словом *end*, т. е. первый оператор после *case*. Селектор должен относиться к целочисленному, булевскому, строковому типу. Вещественные и строковые типы в качестве селектора запрещены. Список констант выбора состоит из произвольного количества значений и диапазонов, отделенных друг от друга запятыми. Границы диапазона записываются двумя константами через разграничитель «..» (две точки). Тип констант в любом случае должен совпадать с типом селектора. Ниже приведены примеры типичных форм записи оператора *case*.

Селектор интервального типа:

```
case i of
  1..10: writeln ('число', i:4, 'в диапазоне 1-10');
  11..20: writeln ('число', i:4, 'в диапазоне 11-20');
  21..30: writeln ('число', i:4, 'в диапазоне 21-30');
else
  writeln ('число', i:4, 'вне пределов контроля');
end;
```

Селектор целочисленного типа:

```
case i of
  1: z := i + 10;
  2: z := i + 100;
end;
```

Селектор перечисляемого пользовательского типа:

```
var
  season : (winter, spring, summer, autumn);
begin
.............
case season of
  winter: writeln ('winter');
  spring: writeln ('spring');
  summer: writeln ('summer');
  autumn: writeln ('autumn');
end;
```

Аналогичным способом можно организовать ввод и вывод данных перечисляемого типа пользователя и обойти соответствующие ограничения языка Pascal.

9. КОМПОНЕНТЫ, ПРЕДНАЗНАЧЕННЫЕ ДЛЯ РЕАЛИЗАЦИИ РАЗВЕТВЛЕННЫХ АЛГОРИТМОВ ПРИ РАБОТЕ С ФОРМАМИ

При работе с формами в PascalABC для организации оператора *if* часто используются компоненты в виде кнопок-переключателей (*RadioButton* и *CheckBox*). Состояние такой кнопки («выбран» / «не выбран») визуально отражается на форме. В программе состояние кнопки связано со значением логической переменной, проверяемой с помощью *if*. Свойство ***Checked*** определяет, активна кнопка или нет, и, соответственно, принимает значение *true* либо *false*.

Компонент ***RadioButton*** позволяет пользователю выбирать один вариант из нескольких взаимоисключающих вариантов, а ***CheckBox*** является независимым переключателем, позволяющим пользователю указать свое решение (да / нет).

На форме (рис. 7) представлены рассматриваемые кнопки-переключатели. У всех элементов отредактировано свойство *Text*.

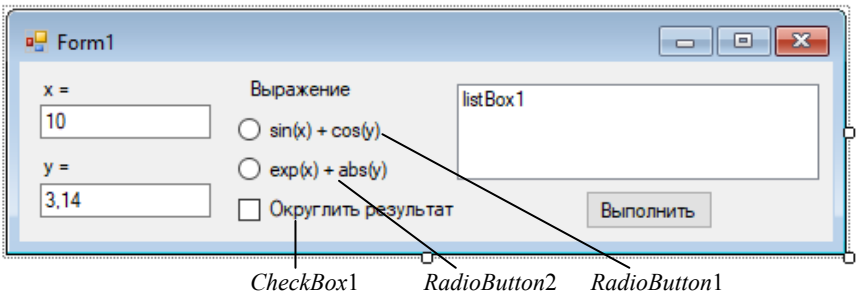


Рис. 7. Пример формы

Пример текста процедуры *Form1.button1_Click*:

```
procedure Form1.button1_Click(sender: Object; e: EventArgs);
var
  x, y, r: real;
```

```
begin
// Ввод исходных данных
  x := StrToFloat(textBox1.Text);
  y := StrToFloat(textBox2.Text);
  r := 0;
// Вывод введенных исходных данных
  listBox1.Items.Clear;
  listBox1.Items.Add('Исходные данные:');
  listBox1.Items.Add('Переменная x = ' + FloatToStr(x));
  listBox1.Items.Add('Переменная y = ' + FloatToStr(y));
// Проверка выбранной кнопки RadioButton и функции
  if radioButton1.Checked = True then
    r := sin(x) + cos(y)
  else
    if radioButton2.Checked = True then
      r := exp(x) + abs(y)
    else
      MessageBox.Show('Ничего не выбрано');
// Проверка состояния CheckBox
  if checkBox1.Checked = True then
    r := round(r);
// Вывод результата
  listBox1.Items.Add('Результат расчета:');
  listBox1.Items.Add('r = ' + FloatToStr(r));
end;
```

Для создания программ множественного ветвления (с оператором *case*) используют списки *ListBox* и *ComboBox* (рис. 8), расположенные в разделе «Стандартные элементы управления» меню «Палитра». Работа с обоими списками и их свойства схожи. Ключевое отличие – в ***ListBox*** можно выбрать несколько строк, изменив свойство *SelectionMode* в меню «Инспектор объектов» со значения *One* (одна строка) на *MultiSimple* или *MultiExtended*. Свойство *SelectedIndex* позволяет передать номер выбранной строки в качестве селектора, причем нумерация строк начинается с нуля. Заполнение списка осуществляется изменением поля *Items* в меню «Инспектор объектов» или в коде программы, например, в событии *Form1_Load* при помощи команды *Items.Add*.

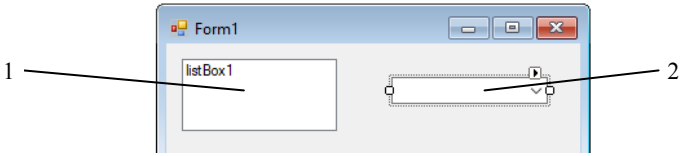


Рис. 8. Элементы *ListBox* (1) и *ComboBox* (2) на форме

Пример использования компонента *ListBox*;

```
case ListBox.ItemIndex of
  1: оператор1; // выбрана строка № 1 => выполнится оператор 1
  2: оператор2; // выбрана строка № 2 => выполнится оператор 2
…
else                    // действие, если ни одна строка не выбрана
  MessageBox.Show('Выберите строку'); // вывод сообщения
end;
```

10. ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ АЛГОРИТМОВ

Алгоритм называется циклическим, если он предусматривает многократное выполнение одних и тех же действий при различных значениях промежуточных данных. Число повторений может быть задано в явной или неявной форме. Если число повторений тела цикла заранее известно, то чаще всего применяется оператор цикла с параметром.

Общий вид оператора цикла с параметром:

for *i* := *s*1 ***to*** *s*2 ***do***
  begin
    тело цикла;
  end;

или

for *i* := *s*1 ***downto*** *s*2 ***do*** тело цикла;

где *for ... do* – заголовок цикла;

тело цикла – многократно повторяющаяся часть программы; телом цикла может быть простой или составной оператор;

i – параметр цикла – переменная, которая изменяется в ходе выполнения оператора цикла при каждом новом выполнении тела цикла;

*s*1 и *s*2 – выражения, задающие соответственно начальное и конечное значения параметра цикла, т. е. параметр цикла изменяется от *s*1 до *s*2 с фиксированным шагом (+1 если ***to***, –1 если ***downto***).

Параметр цикла *for* нельзя изменять в теле цикла, но можно использовать в формулах для вычислений.

Алгоритм работы оператора по первому варианту. Параметру *i* присваивается значение *s*1. Далее параметр *i* сравнивается с конечным значением *s*2. Если *i* <= *s*2, то выполняется тело цикла. Параметр *i* автоматически увеличивается на «единицу», полученная величина снова сравнивается с *s*2. Цикл повторяется до тех пор, пока параметр меньше конечного значения *s*2. Модификация оператора цикла со словом ***downto*** аналогична, но имеет отличия: на каждой

итерации цикла параметр *i* уменьшается на «единицу»; значение *s*1 должно быть больше *s*2.

Пример использования оператора *for*

```
for k := 0 to 9 do
begin
  listBox1.Items.Add(FloatToStr(k));
end;
```

Если шаг изменения параметра отличен от единицы или число повторений тела цикла заранее не известно, то рекомендуется применять операторы **цикла с предусловием** или **с постусловием.**

Структура оператора цикла с предусловием:

while логическое выражение ***do***
begin
  тело цикла;
end;

где ***while*** ... ***do*** – заголовок оператора;
 логическое выражение – выражение булевского типа;
 тело цикла – простой или составной оператор.

Позволяет организовать повторение операторов, помещенных между ***begin*** и ***end***, до тех пор, пока результат логического выражения – «истина». Если результат – «ложь» при первой проверке на входе в цикл, то его тело не выполнится ни разу.

Структура оператора цикла с постусловием:

repeat
  тело цикла;
until логическое выражение;

где ***repeat*** ... ***until*** – заголовок оператора;
 логическое выражение – выражение булевского типа;
 тело цикла – некоторые операторы.

Позволяет организовать повторение операторов, помещенных между ключевыми словами ***repeat*** и ***until***, до тех пор, пока результат логического выражения не станет «истина», после чего управление

передается следующему за циклом оператору. Операторы в теле цикла с постусловием всегда выполняются хотя бы один раз.

Преждевременный выход из тела цикла программист может организовать при помощи команды ***break***. При этом происходит автоматический выход из цикла и управление передается следующей за циклом команде.

В операторах цикла с предусловием и постусловием параметр цикла и его начальное значение необходимо задавать до цикла. В теле цикла должно быть выражение, изменяющее параметр цикла на некоторую величину.

Примеры использования операторов ***while*** и ***repeat***:

Цикл с предусловием:

```
k := 0
while k <= 10 do
begin
 k := k + 2;
 listBox1.Items.Add(FloatToStr(k));
end;
```

Цикл с постусловием:

```
k := 0
repeat
 k := k + 2;
 listBox1.Items.Add(FloatToStr(k));
until k > 10;
```

В приведенных примерах переменная *k* является параметром цикла. Операция присвоения *k* := 0; задает начальное значение параметра, а выражение *k* := *k* + 2; в теле цикла изменяет параметр с шагом «два». Условие выхода из цикла в обоих случаях – параметр *k* становится больше 10. На каждой итерации при выполнении тела цикла печатается значение параметра *k*.

11. ВВОД И ВЫВОД ДАННЫХ ЧЕРЕЗ ВНЕШНИЙ ТЕКСТОВЫЙ ФАЙЛ

Файл – это любой набор элементов одного и того же типа. Файлы могут быть последовательного и прямого доступа. В файлах последовательного доступа каждый элемент может быть прочитан только после перебора всех предыдущих. В файлах прямого доступа можно обратиться к заданному элементу непосредственно, указав его номер в файле.

По отношению к программе файлы могут быть:

1) внутренние – те, которые существуют только во время работы программы и исчезают после выполнения программы (например, стандартные файлы ввода-вывода);

2) внешние – те, что хранятся на внешних носителях (например, на жестком диске) и могут существовать отдельно от программы. Они должны быть описаны в разделе описаний программы.

Текстовые внешние файлы описываются в разделе описания переменных (***var***) следующей конструкцией:

имя файловой переменной: *TextFile*;

где *TextFile* – тип текстового файла.

Для работы с внешними файлами предусмотрен ряд стандартных процедур.

AssignFile – процедура присоединения внешнего файла. Она связывает файловую переменную с именем внешнего файла и его адресом. Эта команда должна быть записана в программе в разделе операторов до работы с внешним файлом:

AssignFile(имя, 'адрес');

где имя – имя файловой переменной;

адрес – место, где хранится файл (указывается в кавычках).

Например,

AssignFile(*f*1, '*d*:*users**aa**lab00.pas*');

AssignFile(*f*1, '*lab*10.*dat*');

Reset(имя) – открыть файл для чтения (подготовить файл для считывания из него данных). Записывается перед оператором чтения данных. Например, Reset(F1).

Rewrite(имя) – открыть файл для записи (подготовить файл к приему некоторой информации, которая получается в результате выполнения программы и должна сохраниться на длительный срок). Например, *Rewrite*(F2). Обращение к процедуре осуществляется перед первой командой записи в файл.

Read(имя, список переменных) – считывание списка переменных из внешнего файла. Например, *Read*(*f*1, *a*, *b*, *c*) – считать переменные *a*, *b*, *c* из внешнего файла *f*1.

Запись списка переменных во внешний файл может быть выполнена командами *Print* или *Write*, после которых в скобках необходимо указать имя файловой переменной и список выводимых переменных. Различия этих команд: *Print* автоматически добавляет пробел после каждой выводимой переменной, а *Write* позволяет использовать форматирование выводимых данных. Если формат не указан, вывод осуществляется в стандартной форме и данные необходимо разделять пробелом в кавычках (' ').

Пример использования, *Print*(*f*2, *x*1, *x*2, *x*3) – записать значения переменных *x*1, *x*2, *x*3 во внешний файл *f*2. Во втором случае запись будет иметь вид: *Write* (*f*2, *x*1, ', ', *x*2, ', ', *x*3).

Под форматом подразумевается заранее предопределенная структура выводимых данных (например, количество символов в выводимом числе).

для целых чисел	для вещественных
Write(*a*:*m*);	*Write*(*a*:*m*:*n*);

где *a* – выводимая переменная;

m – количество позиций, отводимых под число;

n – количество позиций, отводимых под дробную часть числа, причем под точку, отделяющую дробную часть числа, отводится отдельная позиция.

Если количество выводимых символов меньше *m*, то запись числа дополняется пробелами слева.

Например, если *a* = 1, то по команде *Write*(*f*2, '*a* =', *a*:3); выведется запись *a* =..1 (здесь точка означает пробел).

Если необходимо ввести или вывести данные в несколько строк, то к соответствующим командам *Read* и *Print/Write* необходимо до-

бавить окончание *ln*, автоматически перемещающее курсор на новую строчку после выполнения команды.

CloseFile(имя) – закрыть файл. Обычно ставится в конце программы или после последней команды чтения / записи в файл.

Чтение из файла – ввод данных из внешнего файла в оперативную память ПК для выполнения программы. Такой файл называется входным. Для чтения данных из внешнего файла необходимо:

1) описать файловую переменную (*var f*1: TextFile;);
2) присоединить файл (*AssignFile*(*f*1, '*data.pas*'););
3) открыть файл для чтения (*Reset*(*f*1););
4) считать данные из файла (*Read*(*f*1, *A*, *B*););
5) закрыть файл (*CloseFile*(*f*1);).

Запись в файл – вывод данных из оперативной памяти ПК на внешний носитель в ходе выполнения программы. Такой файл называется выходным. Для записи данных во внешний файл необходимо:

1) описать файловую переменную (*var f*2: *TextFile*;);
2) присоединить файл (*AssignFile*(*f*2, '*rezult.pas*'););
3) открыть файл для записи (*Rewrite*(*f*2););
4) записать данные в файл (*Writeln*(*f*2, *t*:4:2, *x*:5:2););
5) закрыть файл (*CloseFile*(*f*2);).

12. РАБОТА С МАССИВАМИ

Массивом называется упорядоченная совокупность конечного числа данных одного типа. Массив может быть одномерным, двумерным и многомерным.

При обозначении массивов все его элементы имеют одинаковое имя, но каждый элемент массива имеет свой индекс. Индекс записывается в квадратных скобках после имени и определяет место элемента массива в упорядоченной последовательности данных (чисел). В качестве индекса используют порядковые типы.

Тип массива описывается в разделе описания пользовательских типов (*type*) или переменных (*var*) программы конструкцией:

имя массива: ***array*** [диапазоны индексов] ***of*** тип элементов;

Примеры описания массивов:

```
type
  mas1 = array [1..16] of real;   // описание типа одномерного массива
var a: mas1;                      // a – массив типа mas1
    b: array [1..4, 1..8] of integer;   // b – двумерный массив
    c: array [1..4] of char;            // c – одномерный массив
```

Эта запись означает следующее. В программе создается некоторый новый тип данных *mas*1, представляющий собой одномерный массив, состоящий из 16 элементов вещественного типа. Все переменные, которые будут относиться к типу *mas*1, представляют собой массив указанной структуры.

Далее следует раздел описания переменных, первая строка которого указывает, что переменная *a* относится к типу *mas*1. Переменная *b* описывает двумерный массив 4×8 с элементами целого типа, а переменная *c* – одномерный массив с элементами символьного типа.

Элементы массивов могут использоваться в выражениях так же, как и обычные переменные, например: *f* := 7 * *a*[4] + *a*[*n* + 1];.

Ввод-вывод массивов осуществляется с помощью операторов цикла. Для *n*-мерного массива используются *n* раз вложенные циклы. Причем сначала выполняется внутренний цикл, а затем – все последующие внешние циклы. Например, ввод элементов одномерного массива *b* из файла можно осуществить с помощью оператора:

```
for i := 1 to n do
  readln(f1, b[i]);
```

где *i* – параметр цикла и индекс элементов;

n – размерность массива.

Одномерные массивы выводятся в строку или в столбец, двумерные и многомерные – в табличном виде.

Часть программы, описывающая вывод элементов двумерного массива *a* во внешний файл в виде прямоугольной матрицы:

```
for i := 1 to n do                // n – количество строк в массиве a
begin                             // начало составного оператора
  for j := 1 to m do              // m – количество столбцов в массиве a
    write (f1, a[i, j]:10:6);     // вывод элементов в строку
  writeln(f1);                    // переход на новую строку
end;                              // конец составного оператора
```

При работе с массивами ввод и вывод информации на экран удобно организовывать в виде таблиц. Компонент ***DataGridView*** из раздела «Данные» меню «Палитра» позволяет отображать информацию в виде двухмерной таблицы. Каждая ячейка такой таблицы представляет собой окно однострочного текстового редактора, поэтому вводимые/выводимые данные необходимо конвертировать в соответствующий тип. Доступ к информации осуществляется с помощью свойства *Value*. На пример: ***dataGridView***1[номер столбца, номер строки]***.Value***. Свойства *ColumnCount* и *RowCount* соответственно устанавливают количество столбцов и строк в таблице. В ходе выполнения лабораторных работ свойствам *AllowUserToAddRows*, *ColumnHeaderVisible* и *RowHeaderVisible* рекомендуется задать значение *False* в инспекторе объектов или в событии *Form_Load*.

Нумерация строк и столбцов начинается с нуля, т. е. максимальное значение индекса строки/столбца **всегда** на единицу меньше, чем их количество. Тогда вывод одномерного массива *a* на 5 элементов в строку будет реализован:

```
datagridview1.RowCount += 1;        // фиксируем число строк
datagridview1.ColumnCount := 5;     // и столбцов
for i := 1 to 5 do
  dataGridView1[0, i – 1](FloatToStr(a[i]));
```

13. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ПОДПРОГРАММ

Подпрограммой называют часть программы, которая выполняет однотипные действия, причем эти действия могут выполняться над различным набором данных. Подпрограмму оформляют в виде специальной структурной единицы.

Использование подпрограмм позволяет:

1) уменьшить объем используемой памяти ЭВМ;

2) сделать программу более компактной;

3) осуществить структурный подход к программированию;

4) решать сложные задачи путем подключения к разработке программы нескольких программистов;

5) экономить время на разработке уже созданных наиболее распространенных алгоритмов обработки данных.

В Pascal используют два вида подпрограмм: **процедуры** и **функции**. Процедура или функция представляет собой последовательность операторов, которая имеет имя, список параметров и может быть вызвана из различных частей программы. Функции, в отличие от процедур, в результате своего выполнения возвращают значение, которое может быть использовано в выражении.

Описание процедуры имеет вид:

procedure имя (список формальных параметров);
раздел описаний
begin
  операторы;
end;

Описание функции имеет вид:

function имя (список формальных параметров): тип возвращаемого значения;
раздел описаний
begin
  операторы;
  имя := выражение;
end;

Здесь *имя* – название подпрограммы, *раздел описаний* аналогичен разделу описаний головной программы, *формальные параметры* – список ключевых переменных с указанием их типа.

Операторы подпрограммы, окаймленные операторными скобками ***begin***/***end***, называются телом этой подпрограммы.

Формальные параметры, передаваемые в процедуру, делят на входные и выходные. При описании выходных параметров перед ними ставится ключевое слово ***var***. Например, в подпрограмме решения квадратных уравнений, где входными данными будут коэффициенты *a*, *b*, *c*, а выходными – корни *x*1 и *x*2, заголовок процедуры будет:

procedure *kvur*(*a*, *b*, *c*: *real*; ***var*** *x*1, *x*2: *real*);

Если в качестве параметров процедуры используются данные сложных типов (например, массивы), необходимо в головной программе описать имя типа этих данных (с помощью раздела ***type***), а затем имя типа указывается в списке формальных параметров.

Например, если в качестве входного параметра используется массив *a* из пяти вещественных элементов, его описывают в головной программе:

*type mas*1 = *array* [1..5] *of real*;

а в процедуре указывают, что переменная *a* относится к типу *mas*1

procedure calc(*a*: *mas*1; *var s*: *real*);

В функции все формальные параметры – входные. Результат выполнения функции – имя функции, т. е. результат присваивается идентификатору, обозначающему название подпрограммы-функции. Результат выполнения функции – одно значение, совпадающее с именем этой функции, поэтому в заголовке указывается тип имени, т. е. тип результата. В разделе операторов функции должен обязательно присутствовать оператор присваивания какого-либо значения переменной, обозначающей имя функции.

Пример заголовка подпрограммы-функции:

function *f*(*x*, *y*: *real*; *n*: *integer*): *real*;

Подпрограмма может быть записана в отдельный файл. Чтобы включить ее в текст головной программы, используют директивы компилятора:

{***$include*** маршрут доступа и имя файла}.

Если файл с подпрограммой расположен в той же папке, что и головная программа, то достаточно указать только имя файла. В конце любой подпрограммы всегда ставится точка с запятой.

В ходе выполнения программы подпрограмма может быть неоднократно вызвана.

Вызов процедуры осуществляется указанием ее имени и перечислением фактических параметров:

имя процедуры (список фактических параметров)

Вызов подпрограммы-функции осуществляется непосредственно в арифметическом выражении указанием ее имени и перечислением фактических параметров:

переменная := имя функции (фактические параметры);

где переменная – некоторая переменная, которой присваивается результат вычисления.

Фактические параметры – это те данные, с которыми работает программист в конкретной программе.

Список фактических параметров должен соответствовать списку формальных параметров по их количеству, типу и последовательности перечисления.

Подпрограммы, описанные в виде процедурных параметров, можно использовать в качестве фактических для обращения к другим процедурам и функциям.

Процедурные параметры – параметры, которые входят в список фактических или формальных параметров и обозначают имя процедуры или функции. Переменные процедурного типа должны быть специально описаны:

type
имя типа 1 = ***procedure***(формальные параметры);
имя типа 2 = ***function***(формальные параметры): тип результата;
var переменная 1: имя типа 1; переменная 2: имя типа 2;

Пример программы с применением функций.
Вычислить значение выражения:

$$y = \frac{\sin(a)+\cos(a)}{\ln(a)} + \frac{\sin(a+b)+\cos(a+b)}{\ln(a+b)} + \frac{\sin(a\cdot b)+\cos(a\cdot b)}{\ln(a\cdot b)}.$$

Однотипное преобразование данных $f(x) = \frac{\sin(x)+\cos(x)}{\ln(x)}$ можно оформить в виде подпрограммы-функции.
На рис. 9 приведен пример формы с элементами

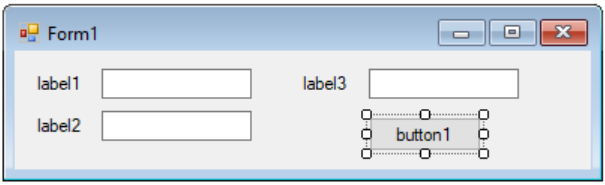


Рис. 9. Пример формы

Текст программы после слова *implementation*:

```
function F(x: real): real;
begin
  F := (sin(x) + cos(x)) / ln(x);
end;

procedure Form1.Form1_Load(sender: Object; e: EventArgs);
begin
  label1.Text := 'a = ';          // задаем подписи меток и кнопки
  label2.Text := 'b = ';
  label3.Text := 'y = ';
  button1.Text := 'Run';
end;
```

```
procedure Form1.button1_Click(sender: Object; e: EventArgs);
var a, b, y: real;
begin
  a := StrToFloat(textBox1.Text); // задаем значение переменной a
  b := StrToFloat(textBox2.Text); // задаем значение переменной b
  y := F(a) + F(a + b) + F(a * b);  // вычисления
  textBox3.Text := FloatToStr(y); // печать результата в textBox3
end;
```

14. ТИПОВЫЕ АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ

Для составления программ самого разнообразного назначения используют ряд типовых алгоритмов. К числу простейших из них можно отнести определение суммы и произведения элементов, нахождение значения минимального и максимального элемента, определение количества и номеров элементов, удовлетворяющих заданному критерию, алгоритмы замены и сортировки элементов.

На рис. 10, *а* представлен фрагмент алгоритма для нахождения минимального элемента. На первом шаге задается предполагаемое значение минимального элемента. Если элеметы заранее известны, можно предположить, что минимальным является первый. Если элементы не известны, в качестве первоначального предполагаемого минимального элемента задают бесконечно большое число. Далее в цикле по очереди все элементы сравнивают с предполагаемым минимальным. Если проверяемый элемент меньше, чем минимальный на текущем шаге, минимальным становится проверяемый. Кроме того, запоминается его номер, как номер минимального. Если проверяемый элемент больше, чем минимальный на текущем шаге, никакие действия не выполняются, текущий элемент игнорируется, осуществляется переход к проверке следующего. Следует отметить, что, если элементы заранее не известны (нет массива данных), необходимо организовать чтение данных внутри цикла (рис. 10, *б*). Поиск максимального элемента выполняется по алгоритму, аналогичному рис. 10, *а*.

На рис. 10, *б* представлен фрагмент алгоритма суммирования положительных элементов. Первоначально сумме присваивается нулевое значение. Затем в цикле вводится значение очередного

элемента и сравнивается с нулем. Если элемент положительный, его значение добавляется к текущей сумме, если нет – элемент игнорируется. Произведение рассчитывают по аналогичному алгоритму, но первоначальному значению присваивают значение единицы.

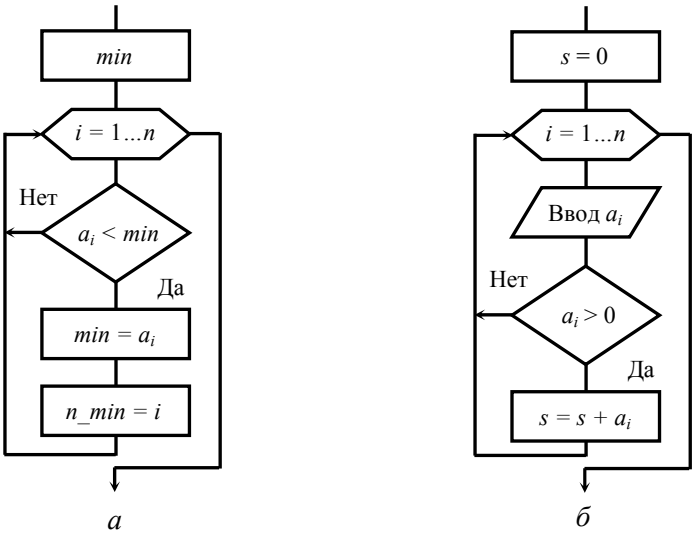


Рис. 10. Фрагменты типовых алгоритмов:
а – определение значения и номера минимального элемента;
б – расчет суммы положительных элементов

Если необходимо поменять местами значения двух переменных (элементов массива), то рекомендуется воспользоваться дополнительной переменной (буфером), куда следует записать значение одной из меняемых переменных (элементов массива).

Объединение алгоритмов поиска экстремального элемента массива и замены элементов и введение дополнительного вложенного цикла с параметром позволяет реализовать алгоритм сортировки массива. Значение параметра вложенного цикла задается значением параметра внешнего цикла.

Общий алгоритм сортировки выбором:

1) вводятся элементы исходного массива;

2) во вложенном цикле определяется максимальный (минимальный) элемент массива и его номер (индекс);

3) после выхода из вложенного цикла осуществляется замена элементов (например, сначала максимального с последним, затем, на очередном шаге внешнего цикла – максимального из оставшихся с предпоследним и т. д.);

4) далее необходимо повторить пункты 2–4 путем организации дополнительного (внешнего) цикла, причем во вложенном цикле начальное (или конечное) значение параметра цикла будет изменяться в соответствии со значением параметра внешнего цикла;

5) в конце программы следует вывести преобразованный массив.

Приложение 1

РЕКОМЕНДАЦИИ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

В ходе выполнения каждой из лабораторных работ студент должен оформить *отчет*, содержащий следующие разделы:

– цель работы;

– задание и исходные данные;

– схема алгоритма решения задачи (головной программы и подпрограмм);

– таблица идентификаторов;

– распечатка головной программы, подпрограмм и результатов расчета;

– выводы.

Примерный перечень лабораторных работ

Лабораторная работа № 1. Основы работы в среде PascalABC.NET. Программирование линейных алгоритмов.

Цель работы: изучить интерфейс среды PascalABC.NET. Научиться составлять простейшие программы в среде PascalABC.NET. Изучить свойства компонентов *listBox*, *textBox*, *Button*. Написать и отладить программу линейного алгоритма.

Задание.

Разработать программу вычисления значения выражения:

$$t = \frac{0,5\sin(\pi + x)}{2 - \sqrt[3]{y}}\left(1 + \frac{\ln(z^2)}{2 + |(x - y)z|}\right).$$

Уточнить количество, наименование и типы исходных данных. Нарисовать схему алгоритма, разбив выражение на части. Ввод исходных данных и необходимых комментариев осуществить при помощи компонентов соответственно *textBox* и *label*. Результаты расчета вывести на компонент *listBox.*

Основные встроенные функции: sin(*X*) – вычисление синуса *x*; cos(*X*) – вычисление косинуса *x*; ln(*X*) – вычисление натурального

логарифма x; sqrt(X) – вычисление $\sqrt{x}$; arctan(X) – вычисление арктангенса x; exp(X) – вычисление e^x; sqr(X) – вычисление x^2; abs(X) – вычисление модуля x; trunc(X) – вырабатывает целый результат путем отбрасывания дробной части аргумента; round(X) – вырабатывает целый результат путем округления до ближайшего целого; odd(x) – принимает значение *True*, если x – нечетное, иначе – *False*.

В Pascal определены только три тригонометрические функции (sin, cos, arctan), для вычисления остальных необходимо использовать известные из тригонометрии соотношения:

$\operatorname{tg}(x) = \sin(x) / \cos(x)$;

$\operatorname{ctg}(x) = \cos(x) / \sin(x)$;

$\arcsin(x) = \operatorname{arctg}\left(x / \sqrt{1 - x^2}\right)$;

$\arccos(x) = \pi / 2 - \arcsin(x)$;

$\operatorname{arcctg}(x) = \pi / 2 - \operatorname{arctg}(x)$.

Возведение в степень – $x^a = \exp(a * \ln(x))$ или $x^a = \text{power}(x, a)$.

Лабораторная работа № 2. Программирование разветвляющихся алгоритмов

Цель работы: научиться пользоваться простейшими компонентами организации разветвляющихся алгоритмов (*CheckBox*, *RadioButton*, *ComboBox*). Написать и отладить программу разветвляющегося алгоритма.

Задание.

Разработать программу вычисления значения выражения:

$$a = \begin{cases} (x+y)^2 - \sqrt{|x/y|}, & \text{если } x + y > 0; \\ (x-y)^2 - \sqrt{|x \times y|}, & \text{если } x - y < 0; \\ (x+y)^2 + 1, & \text{иначе} \end{cases}$$

Отредактировать форму и написать текст программы в соответствии с заданием. Ввод и вывод исходных данных и результата расчета выполнить при помощи компонентов *textBox*. Предусмотреть возможность округления и вывода дробной части результата расчета при помощи компонента *CheckBox*.

Лабораторная работа № 3. Программирование циклических алгоритмов.

Цель работы: изучение структуры и порядка действия операторов цикла. Составить и отладить программу циклического алгоритма.

Задание.

Разработать программу, вычисляющую функцию $f(x) = \cos(x)$ и ее разложение в ряд в виде суммы $s = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}$ для аргумента x, изменяющегося в пределах от x_0 до x_k с шагом h ($h = \frac{x_k - x_0}{n}$, где n – количество разбиений отрезка) и точностью e. С целью упрощения алгоритма выделить рекуррентную формулу. Отредактировать форму и написать текст программы в соответствии с заданием.

Лабораторная работа № 4. Ввод-вывод данных через внешний файл.

Цель работы:

1. Ознакомление со стандартными процедурами для работы с внешними файлами.
2. Изучение порядка чтения и записи данных через внешний файл.
3. Приобретение навыков оформления отчетного файла по результатам выполнения расчетов.

Задание.

Модифицировать выполненные предыдущие лабораторные работы, сохранив номер варианта, изменив организацию ввода-вывода данных. В одной из лабораторных работ (по указанию преподавателя) ввод данных осуществить из внешнего файла. Вывод результатов работы программы во всех лабораторных работах осуществить во внешний файл результатов, параллельно организовав печать основных данных, полученных в ходе выполнения программы, на дисплей. Внешний файл результатов оформить в виде технического отчета. Он должен содержать номер и название лабораторной работы, исходные данные и результаты расчета, снабженные соответствующими комментариями, а также инициалы исполнителей. Текст и числовые значения должны быть отформатированы.

Лабораторная работа № 5. Ввод-вывод массивов. Преобразование одномерных массивов.

Цель работы: изучить данные, относящиеся к типу масив, и свойства компонента *DataGridView*. Написать программу с использованием массивов.

Задание.

Задан двумерный массив A размером $n \times m$. Получить одномерный массив B, присвоив его элементам значения элементов первого столбца исходного массива A, если первый элемент k-й строки массива A больше последнего, и значения последнего столбца в противном случае. Вывести максимальный элемент массива B.

Ввод и вывод массивов необходимо реализовать при помощи компонента *DataGridView*, значения переменных k, m, n и максимальный элемент массива B – при помощи компонентов *textBox*. Вычисления выполнять после нажатия кнопки *Button*.

Лабораторная работа № 6. Упорядочивание одномерных массивов.

Цель работы: изучить типовые алгоритмы сортировки массивов. Написать программу сортировки одномерного массива.

Задание.

Разработать программу для упорядочивания элементов одномерного массива из n элементов.

Ввод и вывод исходного и преобразованного массивов необходимо реализовать при помощи компонентов *DataGridView*. Значения исходного массива присвоить из внешнего файла. Предусмотреть форматный вывод значений элементов исходного и преобразованного массивов во внешний файл с указанием имени массива и соответствующих индексов.

Лабораторная работа № 7. Программирование с использованием подпрограмм типа функция.

Цель работы: изучить возможности среды PascalABC.NET для написания подпрограмм типа функция. Составить и отладить программу, использующую пользовательскую функцию.

Задание.

Разработать программу, которая вычисляет сумму:

$$z = f\left(\sin(\alpha), \ln|\beta|\right) + f\left(\sqrt[3]{\alpha\beta}, \alpha+\beta^2\right)$$

заданной функции $f(x, y)$ двух переменных:

$$f(u,t) = \begin{cases} u + t, & \text{если } u > 0; \\ u - t, & \text{если } u \le 0. \end{cases}$$

Вычисление $f(x, y)$ оформить в виде пользовательской функции. Ввод и вывод исходных данных и результата расчета выполнить при помощи компонентов *textBox*. Предусмотреть форматный вывод исходных и полученных значений во внешний файл, снабдив их соответствующими комментариями. Вычисления выполнять после нажатия кнопки *Button.*

Лабораторная работа № 8. Программирование с использованием процедур.

Цель работы: изучить возможности среды PascalABC.NET для написания подпрограмм типа процедура. Составить и отладить программу, использующую пользовательскую процедуру.

Задание.

Разработать подпрограмму, позволяющую определить сумму всех отрицательных элементов двумерного массива размерностью $k \times l$. Подпрограмму оформить в виде процедуры. Разработать программу решения задачи с использованием разработанной процедуры.

Ввод массива необходимо реализовать при помощи компонента *DataGridView*. Ввод и вывод исходных данных и результата расчета выполнить при помощи компонентов *textBox*. Предусмотреть форматный вывод исходных и полученных значений во внешний файл, снабдив их соответствующими комментариями. Вычисления выполнять после нажатия кнопки *Button*.

Приложение 2

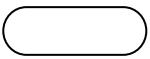
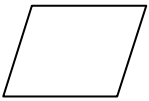
ОСНОВНЫЕ ЭЛЕМЕНТЫ БЛОК-СХЕМ АЛГОРИТМОВ

Блок-схема – совокупность элементов, соответствующих этапам работы алгоритма, и соединяющих их линий. Направление связей между символами изображают сплошной линией со стрелкой. Если алгоритм располагается сверху вниз или слева направо, то стрелку можно не указывать. Дополнительные комментарии к символам изображаются при помощи пунктирных линий.

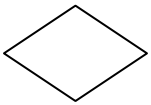
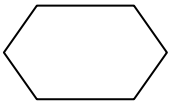
Обозначения основных элементов блок-схем в соответствии с ГОСТ 19.701-90 (*ISO* 5807:1985) представлены в табл. 1.

Таблица 1

Основные элементы блок-схем

Элемент	Описание
	Терминатор начала и конца работы головной программы/подпрограммы. Тип возвращаемого значения (для функции) и аргументов (для функции и процедуры) обычно указывается в комментариях к блоку терминатора
	Операции ввода и вывода данных без указания источника. В комментариях могут быть указаны подробности ввода-вывода данных
	Выполнение над данными операций, не требующих вызова внешних функций. Внутри элемента могут размещаться одна или несколько операций присваивания
	Элемент печати. Применяется для обозначения вывода результатов на печать

Окончание табл. 1

Элемент	Описание
	Ветвление алгоритма (операторы условного перехода, выбора, циклы с пред- и постусловием). Имеет один вход и несколько выходов. Если описывается оператор выбора, то внутри указывается ключ выбора, а на выходящих линиях, количество которых может быть больше двух, – его значения. Во всех остальных случаях внутри указывается логическое выражение, а на выходящих линиях – результат его выполнения (да/нет)
	Подготовка данных. Применяется для организации циклического процесса с параметром. Внутри через запятую записывают параметр, начальное и конечное значения
	Блок подпрограммы (предопределенного процесса). Применяется в целях указания обращения к отдельным подпрограммам (процедурам и функциям). Внутри указывается имя используемой подпрограммы
	Комментарий. Комментарий может быть соединен как с одним блоком, так и группой. Группа блоков выделяется на схеме пунктирной линией

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Жилевич, М. И. Информатика : учебно-методическое пособие к выполнению лабораторных работ для специальности 1-36 01 07 «Гидропневмосистемы мобильных и технологических машин» / М. И. Жилевич, Л. Г. Филипова ; Белорусский национальный технический университет, Кафедра «Гидропневмоавтоматика и гидропневмопривод». – Минск : БНТУ, 2003. – 77 с.

2. Жилевич, М. И. Информатика : учебно-методическое пособие к лабораторным работам для студентов специальности 1-36 01 07 «Гидропневмосистемы мобильных и технологических машин» / М. И. Жилевич, Л. Г. Филипова ; Белорусский национальный технический университет, Кафедра «Гидропневмоавтоматика и гидропневмопривод». – Минск : БНТУ, 2009. – 39 с.

3. Синицын, А. К. Основы алгоритмизации и программирования в среде DELPHI. Базовые типы и простейшие алгоритмы : лаб. практикум по курсу «Основы алгоритмизации и программирования» для студ. 1–2-го курсов всех спец. БГУИР / А. К. Синицын, А. А. Навроцкий. – Минск : БГУИР, 2005. – 80 с.

4. Современное программирование на языке Pascal [Электронный ресурс]. – Режим доступа: https://pascalabc.net. – Дата доступа: 07.02.2023.

5. Блог о компьютерах и программировании для начинающих [Электронный ресурс]. – Режим доступа: https://gospodaretsva.com. – Дата доступа: 07.02.2023.

6. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения : ГОСТ 19.701-90. – Взамен: ГОСТ 19.002-80, ГОСТ 19.003-80 ; введ. РБ 01.01.1992. – М. : Стандартинформ, 2010. – 24 с.

Учебное издание

ЕРМИЛОВ Сергей Владимирович
ЖИЛЕВИЧ Михаил Иванович
ФИЛИПОВА Людмила Геннадьевна

ИНФОРМАТИКА

Учебно-методическое пособие
к выполнению лабораторных работ
для студентов специальности 6-05-0715-04
«Гидропневмосистемы мобильных
и технологических машин и оборудования»

Редактор *Н. Ю. Казакова*
Компьютерная верстка *Н. А. Школьниковой*

Подписано в печать 15.03.2024. Формат 60×84 $^{1}/_{16}$. Бумага офсетная. Ризография.
Усл. печ. л. 2,73. Уч.-изд. л. 1,68. Тираж 50. Заказ 838.

Издатель и полиграфическое исполнение: Белорусский национальный технический университет.
Свидетельство о государственной регистрации издателя, изготовителя, распространителя
печатных изданий № 1/173 от 12.02.2014. Пр. Независимости, 65. 220013, г. Минск.