

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Белорусский национальный технический университет

Кафедра «Автомобили»

Н.В. Калинин

**РАБОТА С КОМПОНЕНТАМИ
И ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ
И РАЗВЕТВЛЯЮЩИХСЯ АЛГОРИТМОВ
В ИНТЕГРИРОВАННОЙ
СРЕДЕ DELPHI XE**

Учебно-методическое пособие
для студентов специальности 1-37 01 02
«Автомобилестроение (по направлениям)»

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

Минск
БНТУ
2019

УДК 004
ББК 32.97
К17

Р е ц е н з е н т ы:

кандидат технических наук, доцент, заведующий кафедрой
«Тракторы и автомобили» БГАТУ *Г.И. Гедроить*;
кандидат технических наук, доцент кафедры «Тракторы
и автомобили» БГАТУ *А.Ф. Безручко*

Калинин, Н.В.

К17 Работа с компонентами и программирование линейных и разветвляющихся алгоритмов в интегрированной среде Delphi XE : учебно-методическое пособие для студентов специальности 1-37 01 02 «Автомобилестроение (по направлениям)» / Н.В. Калинин. – Минск: БНТУ, 2019. – 97 с.
ISBN 978-985-550-975-3.

Приведены теоретические сведения для изучения основ работы с интегрированной средой Delphi XE, составления блок-схем и создания программ, содержащих линейные и разветвляющиеся алгоритмы. Подробно рассмотрены конструкции с использованием оператора условия if; сложные условия, включающие условия И, ИЛИ, НЕ, И НЕ, ИЛИ НЕ и условие с исключаяющим ИЛИ. Дана методика построения таблиц истинности для сложных составных условий; проанализирован ряд примеров, конечными результатами которых являются работающие программы. Приведено большое количество блок-схем с их подробным описанием.

Пособие предназначено для студентов дневного и заочного отделения специальности 1-37 01 02 «Автомобилестроение»; также может быть полезно для студентов специальностей 1-37 01 03 «Тракторостроение», 1-37 01 04 «Многоцелевые и гусеничные машины» и 1-37 01 05 «Городской электрический транспорт».

**УДК 004
ББК 32.97**

ISBN 978-985-550-975-3

© Калинин Н.В., 2019
© Белорусский национальный
технический университет, 2019

ПРЕДИСЛОВИЕ

Для построения характеристики дизельного или электрического двигателя, тяговой, динамической, топливной и других характеристик автомобиля, определения размеров деталей (например, валов, шестерен) нужно переводить расчёты.

Может возникнуть необходимость использования математических методов: так, исследовать процессы, происходящие при трогании автомобиля, можно с помощью системы дифференциальных уравнений, решая её методом Рунге-Кутты четвёртого порядка.

Использование вычислительной техники для сложных инженерных расчётов позволяет многократно сократить время: за секунды или доли секунды ЭВМ может выполнить такой объём работ, для которого потребовались бы месяцы, если эти расчёты выполнять вручную или при помощи непрограммируемого калькулятора. Использование ЭВМ позволяет сразу получить наглядные графические зависимости. Производить вычисления можно при помощи электронных таблиц (например, Microsoft Excel), математических пакетов с встроенными функциями (например, MathCAD, MathLab), алгоритмических языков программирования и специальных пакетов, предназначенных для каких-либо расчётов (например, Adams, Amesim).

Данное пособие представляет основы работы с интегрированной средой Delphi XE визуального объектно-ориентированного программирования для 32-битных и 64-битных операционных систем Windows, содержащей алгоритмический язык на основе Pascal (Object Pascal). Объектно-ориентированное программирование позволяет решать сложные задачи с помощью объектов, необходимых для программирования в Windows.

Визуализация – это отображение сложных процессов или понятий в виде графических примитивов. *Визуальное программирование* – это программирование, предусматривающее создание приложений с помощью наглядных средств. Например, открывшаяся при запуске программы форма представляет собой заготовку для диалогового окна, а для создания на нём командной кнопки достаточно положить соответствующий компонент; при этом все фрагменты программы, формирующие данный компонент, Delphi создаёт автоматически.

Объектно-ориентированная программа – это совокупность объектов и способов их взаимодействия. *Объект* – это совокупность свойств, методов и событий, на которые он может реагировать. *События* наступают вследствие действий пользователя (например, по щелчку левой кнопки мыши) и не только. *Свойством* называется совокупность данных и методов их чтения и записи. *Методами* называются функции и процедуры, обеспечивающие все необходимые операции с данными.

Для программной реализации задачи необходимо составить алгоритм.

Понятию «алгоритм» дано немало определений. Применительно к ЭВМ *алгоритм* – это набор операций с указанием последовательности их выполнения, в результате осуществление которых может быть решена задача определённого типа при наличии требуемых исходных данных или достигнута поставленная цель.

Алгоритм начинает работать с некоторым набором данных, которые называются исходными, и в результате этой работы выдаются данные, которые называются результатами расчёта.

При описании с помощью блок-схемы алгоритм изображается геометрическими фигурами (блоками), которые связаны по управлению линиями и, если необходимо, стрелками. В блоках записывается последовательность действий.

Линейный алгоритм – это набор команд и указаний, выполняемых друг за другом.

Разветвляющийся алгоритм содержит хотя бы одно условие, в результате которого обеспечивается переход на один из двух возможных вариантов.

Части 1, 2 и 3 рассчитаны на 2 академических часа каждая, часть 4 – на 4 академических часа.

Первая часть даёт представление об основах работы с интегрированной средой Delphi: как создать проект; как правильно сохранить его и его модули; как правильно проект открыть; как выполнять компиляцию и отладку программы; где расположены компоненты и как их класть на форму; как запрограммировать командную кнопку.

Вторая часть показывает, как можно менять свойства компонентов в программе.

В третьей части рассмотрены основные элементы блок-схемы и приведены теоретические сведения, необходимые для составления

программы; пример программной реализации линейного алгоритма в интегрированной среде Delphi XE.

В четвёртой части рассмотрены конструкции с оператором условия, а также сложные условия, включающие условия И, ИЛИ, НЕ и условие с исключаящим ИЛИ; дана методика построения таблиц истинности для сложных составных условий.

Материал рекомендуется разбирать строго по порядку, поскольку для выполнения, например, задач из третьей части нужно знать весь тот материал, который был пройден в первой. Теоретические сведения излагаются по мере того, как в них возникает потребность для выполнения тех или иных действий, и приведены в объёме, необходимом для выполнения заданий без использования других источников.

Автор выражает благодарность рецензентам и особую признательность доценту Ю.Е. Атаманову за ценные замечания.

1. ОСНОВЫ РАБОТЫ С VCL FORMS APPLICATION. РАБОТА С КОМПОНЕНТАМИ

1.1. Создание VCL Forms Application

Приложение VCL Forms Application состоит из весьма большого количества файлов. Основные из них представлены в табл. 1.1.

Таблица 1.1

Основные файлы проекта

Расширение	Назначение файла
dpr, dproj	Файл проекта
pas	Программный код
dfm	Код формы (показывает положение компонентов и значения их свойств)
exe	Готовое приложение

По причине большого количества файлов рекомендуется создавать отдельную папку для каждого приложения.

1.1.1. Запустите Delphi XE.

1.1.2. Выберите в меню File>New>VCL Forms Application – Delphi (рис. 1).

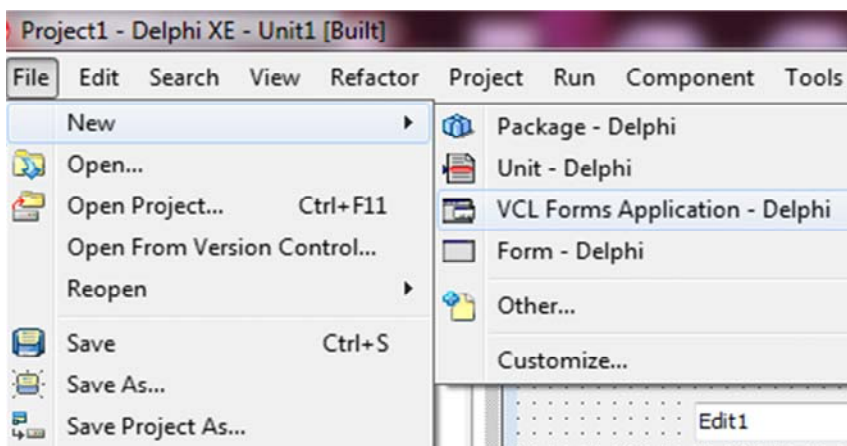


Рис. 1. Создание VCL Forms Application

Откроется *форма* Form1 (может называться по-другому в зависимости от настроек Delphi XE). Форма – это и есть диалоговое окно. На диалоговом окне могут быть таблицы, графики, поясняющие надписи, блоки для ввода текста, кнопки-переключатели и т. д.; для всего этого есть соответствующие *компоненты*: для кнопки – Button, BitBtn; для надписи – Label и т. д. Компоненты расположены на *вкладках* (рис. 2).

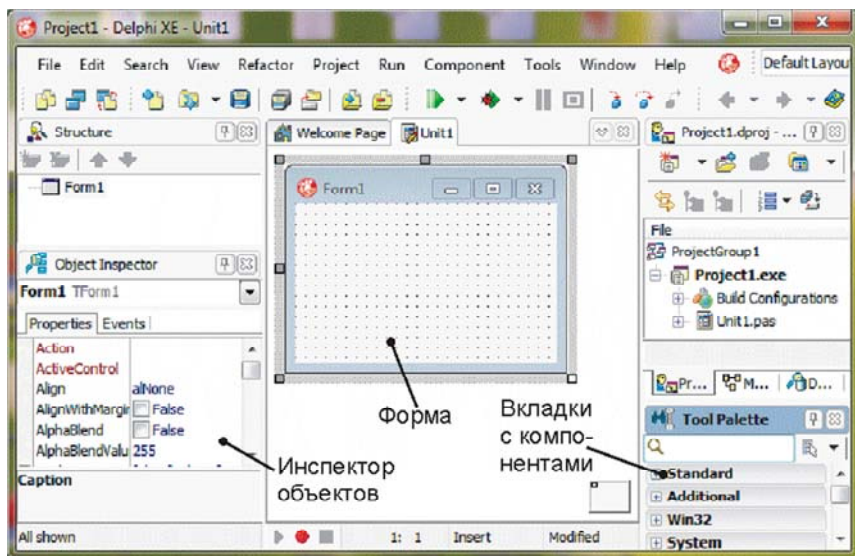


Рис. 2. Форма в режиме редактирования

Вкладка Properties (свойства) инспектора объектов Object Inspector служит для изменения свойств компонентов (цвет, шрифт, число строк и столбцов таблицы и т. д.), а *вкладка Events (события)* – для назначения процедур-обработчиков событий (примеры событий: щелчок по кнопке, ввод символа в таблицу и т. д.). Инспектор объектов вызывается клавишей F11.

Для переключения с формы на соответствующий ей Unit с программным текстом и обратно служит клавиша F12. Также может быть использован специальный значок (рис. 3).

Вид Unit, с которым ещё не работали, представлен на рис. 4.

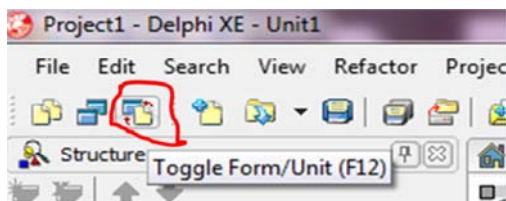


Рис. 3. Переключение с Unit на форму и обратно

```

unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes,
  Graphics, Controls, Forms, Dialogs;
type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
end.
  
```

Рис. 4. Структура Unit, с которым ещё не работали

1.2. Сохранение проекта

Для сохранения проекта первый раз необходимо выбрать в меню File пункт Save Project As. При этом вначале будет предложено сохранить Unit, а затем – проект.

Чтобы не пришлось вносить изменения в файл проекта, переименовывать проект или Unit нужно *только при первом сохранении* или не переименовывать вообще, а *при последующем сохранении* использовать Save, но не Save As и не Save Project As.

1.2.1. Создайте папку Lr1 для сохранения проекта.

1.2.2. Выберите пункт Save Project As в меню File (рис. 5).

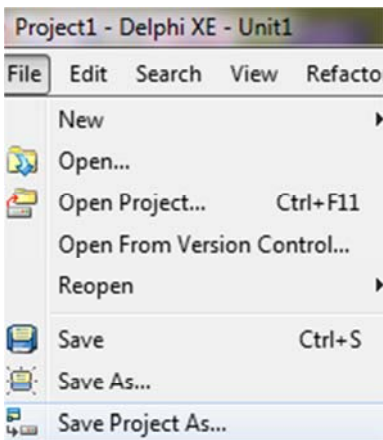


Рис. 5. Сохранение проекта

проекта под другим именем расширение также следует сохранить, записав Lr1.dproj вместо Lr1.

1.2.3. В открывшемся диалоговом окне Save Unit1 As (рис. 6) выберите созданную ранее папку Lr1 и щёлкните по кнопке «Сохранить».

1.2.4. В открывшемся диалоговом окне Save Project1 As выберите папку Lr1 (скорее всего, она там будет по умолчанию), дайте имя проекту Lr1 (рис. 7) и щёлкните по кнопке «Сохранить». На рис. 7 настройки Windows установлены так, что расширения файлов скрыты. Если же расширение не скрыто (то есть записано Project1.dproj, а не Project1), то при сохранении

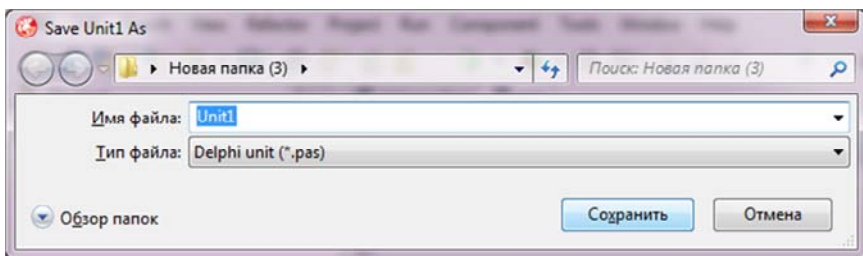


Рис. 6. Диалоговое окно сохранения Unit

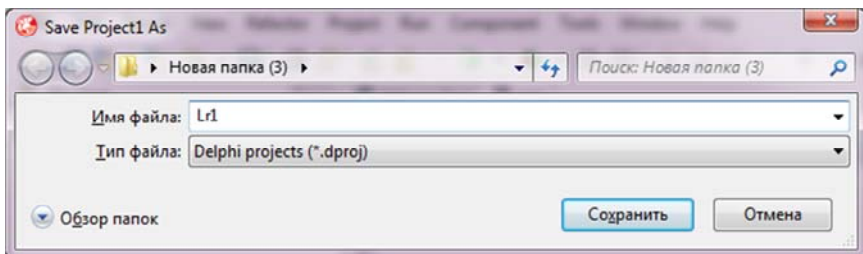


Рис. 7. Диалоговое окно сохранения проекта

1.2.5. Закройте проект.

1.3. Открытие созданного проекта

Рекомендуется открывать файл проекта с расширением .dproj. Можно также открыть файл с расширением .dpr. Если на ЭВМ установлена и более старая версия Delphi, то необходимо указывать, какой программой файл .dpr открывать. Если вместо файла проекта открыть файл Unit.pas или Unit.dfm, то программа в режиме отладки не запустится.

1.3.1. В ранее созданной папке Lr1 выберите файл Lr1.dproj и запустите его.

1.4. Компиляция и отладка программы

На данный момент проект открыт *в режиме создания*. На форму можно укладывать компоненты и записывать программный текст в Unit, но при этом программа не выполняется.

Чтобы проверить, как работает приложение, его необходимо запустить *в режиме отладки*. Для этого выполняется компиляция при помощи Ctrl+F9, после чего при отсутствии видимых компилятором ошибок приложение запускается в режиме отладки F9.

1.4.1. Нажмите Ctrl+F9, а затем F9. На экране вместе с формой в режиме создания появится та же форма, но в режиме отладки (рис.8), а в верхней строке будет написано не **Lr1 – Delphi XE – Unit 1 [Built]**, а **Lr1 – Delphi XE – Unit 1 [Running] [Built]**, форма в режиме отладки будет без нанесённой на неё сетки из точек.

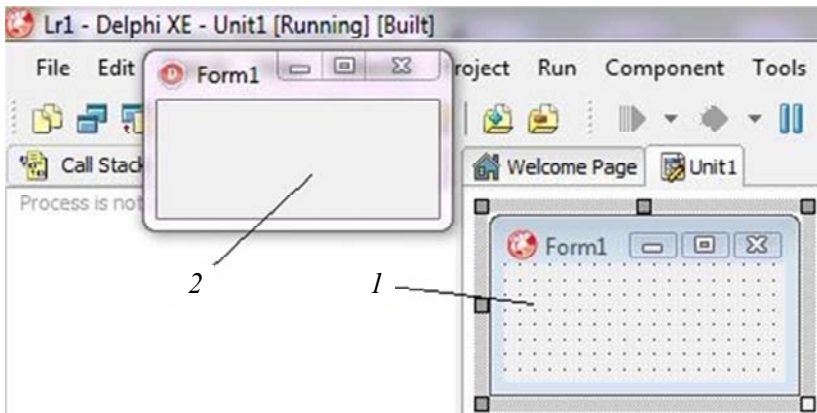


Рис. 8. Запуск приложения в режиме отладки:
1 – форма в режиме создания, 2 – форма в режиме отладки

1.4.2. Закройте режим отладки: для этого щёлкните по × в правом верхнем углу формы, запущенной в режиме отладки (поз. 2 по рис. 8). Надпись вверху снова должна стать Lr1 - Delphi XE - Unit 1 [Built] без слова [Running] – убедитесь в этом.

Уже готовую программу представляет из себя файл .exe. Файл .exe можно копировать и отдавать пользователю; .exe будет работать независимо оттого, установлено Delphi XE на ЭВМ пользователя. После успешной компиляции в папке Debug автоматически создаётся папка Win32 с файлом .exe. В более старых версиях Delphi файл .exe располагался в одной папке с другими файлами проекта.

1.4.3. В папке Lr1 найдите папку Debug и откройте её. В папке Debug откройте папку Win32 и найдите файл Lr1.exe. Расширение .exe может быть скрыто – это зависит от настроек папок.

1.5. Добавление командной кнопки на форму

В качестве кнопки, обеспечивающей закрытие программы используем компонент BitBtn с вкладки Additional (рис. 9).

Чтобы положить компонент на форму, необходимо:

- 1) открыть нужную вкладку на панели Tool Palette (см. рис. 2);
- 2) щёлкнуть по нужному компоненту, чтобы выбрать его; когда компонент будет выбран, он будет выделен цветом (рис. 10);
- 3) щёлкнуть по тому месту формы, на котором должен будет лежать компонент.

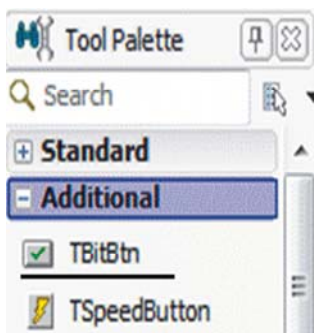


Рис. 9. Расположение компонента BitBtn

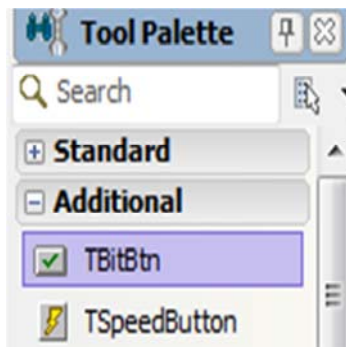


Рис. 10. Вид компонента BitBtn, когда он выбран

В результате форма с компонентом BitBtn должна будет выглядеть, как на рис. 11. При необходимости можно передвинуть компонент BitBtn или изменить его размеры.

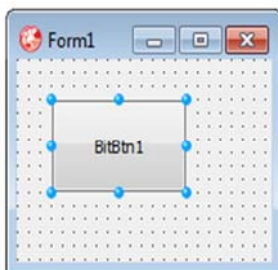


Рис. 11. Форма с компонентом BitBtn

1.5.1. Положите на форму компонент BitBtn.

1.6. Изменение свойств формы и компонентов

Положение компонента на форме, видимость, доступность, размеры, содержание и шрифт отображаемого на нём текста и т. д. определяются значениями соответствующих свойств.

Изменить свойства компонентов и формы можно:

- 1) в **Object Inspector** (за исключением некоторых свойств некоторых компонентов), вкладка **Properties**;
- 2) в тексте программы;
- 3) для некоторых свойств (размер и расположение на форме) – вручную, растягивая и перетаскивая компонент.

Для изменения размеров командной кнопки BitBtn её необходимо выделить, щёлкнув по ней 1 раз. Если на кнопке появились точки по краям, то она выделена (см. рис. 11).

Если случайно будет выполнен двойной щелчок и откроется Unit, то нужно вернуться на форму, нажав клавишу F12, либо значок, показанный на рис. 3.

Чтобы *изменить размер* компонента, необходимо подвести к нему курсор так, чтобы курсор принял вид стрелки, направленной в две стороны (рис. 12). Когда курсор примет такой вид, следует нажать левую кнопку мыши и, не отпуская её, тянуть в направле-

нии, соответствующем изменению размера компонента. В данном случае будем тянуть кнопку вправо и вниз; в результате её размер увеличится (рис. 13). После достижения желаемого размера компонента следует отпустить левую кнопку мыши.

1.6.1. Увеличьте размер кнопки BitBtn1, растянув её.

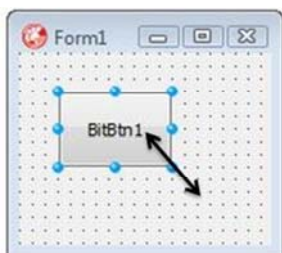


Рис. 12. Изменение размера командной кнопки BitBtn

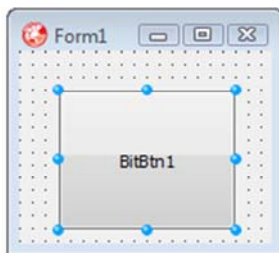


Рис. 13. Вид командной кнопки BitBtn после изменения размера

Аналогично можно изменять размеры многих других компонентов и формы.

Для *перетаскивания* компонента следует его выделить, нажать на компонент левой кнопкой мыши и перемещать курсор, не отпуская левую кнопку мыши. Курсор при этом должен быть в виде обычной стрелки, а не как на рис. 12. После того как компонент переместится туда, куда требовалось, следует кнопку мыши отпустить.

Все те изменения свойств компонента, которые выполнены его растягиванием или перетаскиванием, будут автоматически отображаться в Object Inspector. При *растягивании* изменились свойства Width (ширина) и Height (высота) командной кнопки. На рис. 14 показано значение свойства Width компонента BitBtn до его растягивания, а на рис.15 – после. При *перетаскивании* изменятся свойства Top (расстояние до верхнего края компонента от верхнего края формы) и Left (расстояние от левого края компонента до левого края формы).

При работе с компонентом при помощи Object Inspector необходимо убедиться, что выбран именно тот компонент, свойства которого нужно изменить. На рис. 14, 15 в верхнем окне Object Inspector написано BitBtn1: значит, в нём отображены свойства компонента BitBtn1. Если бы было написано Form1, то в нём бы были отображены свойства формы. Можно щёлкнуть по верхнему окну Object Inspector и в раскрывшемся списке выбрать требуемые объект (рис. 16).

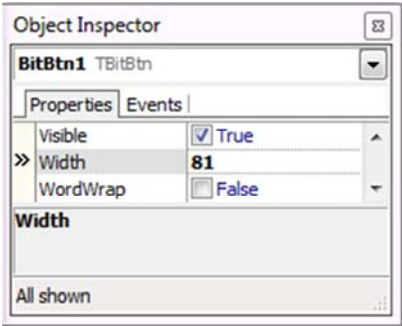


Рис. 14. Исходное значение свойства Width кнопки BitBtn

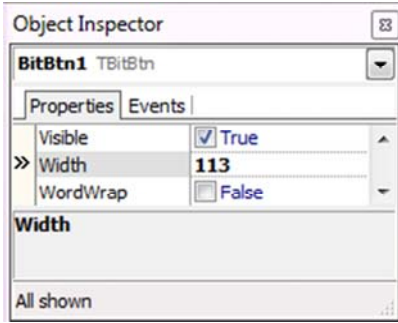


Рис. 15. Значение свойства Width после изменения ширины кнопки

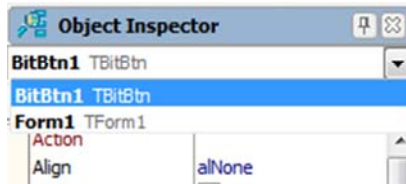


Рис. 16. Выбор требуемого объекта в Object Inspector

Для изменения текста поясняющей надписи служит свойство Caption. В данном случае кнопка будет использована для закрытия программы. Найдём на вкладке Properties инспектора объектов свойство Caption (рис. 17). Напротив свойства Caption будет присвоено записано BitBtn1. Очистим поле, в котором записано BitBtn1, и запишем туда слово Выход (рис. 18). В результате вместо BitBtn1 на кнопке будет написано Выход (рис. 19).

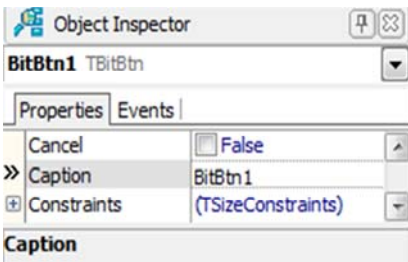


Рис. 17. Исходное значение свойства Caption кнопки BitBtn

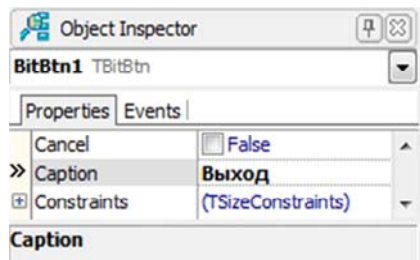


Рис. 18. Значение свойства Caption после его изменения

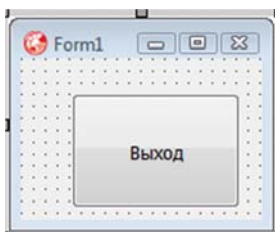


Рис. 19. Изменение поясняющей подписи кнопки

1.7. Программирование кнопки «Выход»

При щелчке по кнопке BitBtn1 должна закрываться программа. Чтобы что-то происходило по щелчку по кнопке, необходимо для кнопки запрограммировать событие **OnClick** (по щелчку). Событию OnClick будет соответствовать процедура BitBtn1Click, которая будет выполняться столько раз, сколько раз будет нажата кнопка.

Чтобы запрограммировать событие OnClick, необходимо:

1) выделить кнопку на форме или выбрать в окне Object Inspector (см. рис. 16);

2) в Object Inspector открыть вкладку Events (события); вкладки Properties (свойства) и Events (события) находятся в Object Inspector сразу под названием выбранного компонента;

3) выбрать событие OnClick и щёлкнуть два раза в строке напротив него (рис. 20).

Вместо п. 1–3 можно щёлкнуть 2 раза по кнопке BitBtnClick.

В обоих случаях в строке напротив названия свойства запишется название процедуры (рис. 21).

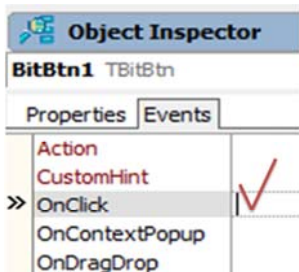


Рис. 20. Строка напротив названия OnClick, по которой необходимо щёлкнуть

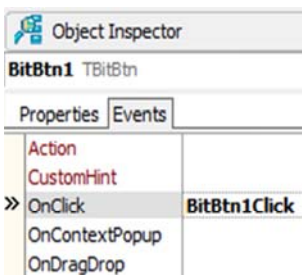
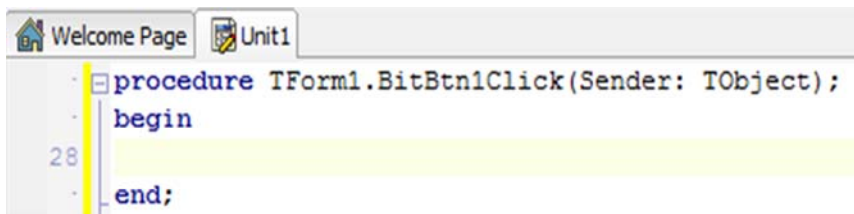


Рис. 21. Вид строки напротив названия свойства OnClick после того, как создана заготовка процедуры

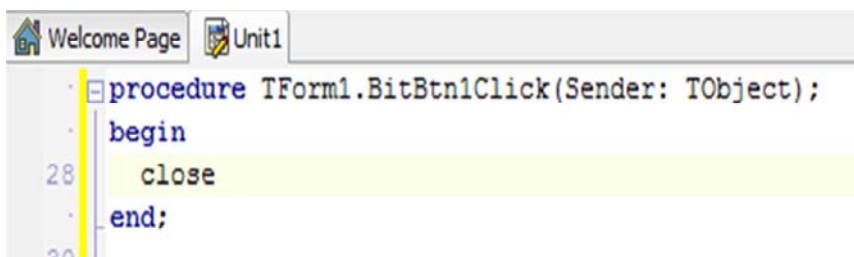
Откроется Unit с уже готовой заготовкой процедуры (рис. 22).

Чтобы форма закрывалась при нажатии на кнопку «Выход», необходимо записать `close` в разделе операторов (то есть между `begin` и `end`) процедуры `BitBtn1Click` (рис. 23).



```
Unit1
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
end;
```

Рис. 22. Заготовка процедуры



```
Unit1
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
  close
end;
```

Рис. 23. Готовая процедура

1.7.1. Щёлкните 2 раза по кнопке «Выход».

1.7.2. Запишите `close` между `begin` и `end`.

При укладывании каждого компонента на форму Delphi будет добавлять в раздел `interface` имя уложенного компонента с указанием типа компонента. При создании процедуры-обработчика Delphi будет добавлять в раздел `interface` имя процедуры. Пока не была уложена кнопка, Unit выглядел так, как на рис. 4. После добавления кнопки `BitBtn1` и создания процедуры `BitBtn1Click` раздел `interface` Unit станет выглядеть аналогично рис. 24: появится имя компонента (`BitBtn1`) с указанием типа (`TBitBtn`) и название процедуры (`BitBtn1Click`).

Далее следует выполнить компиляцию, нажав комбинацию клавиш `Ctrl+F9`.


```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs;
type
  TForm1 = class(TForm)
    BitBtn1: TBitBtn; //имя компонента
    procedure BitBtn1Click(Sender: TObject); //имя процедуры
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

Рис. 24. Вид раздела interface Unit после программирования командной кнопки.

Если компиляция выполнена успешно, то рекомендуется сохранить изменения (File>Save или Ctrl+S). При успешной компиляции должны появиться синие точки слева от текста программы (рис. 25).

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
  close
end;
end.
```

Рис. 25. Образец успешной компиляции

Если при компиляции выдана ошибка, то следует её исправить, после чего повторно выполнить компиляцию при помощи Ctrl+F9. В случае возникновения ошибок в окне под текстом Unit будут перечислены ошибки; также будет сообщение со словами Fatal Error (рис. 26), а синих точек слева от текста программы, которые есть на рис. 25, не будет.

1.7.3. Нажмите Ctrl+F9.

1.7.4. В случае успешной компиляции (нет видимых компилятором ошибок) сохраните изменения в программе, нажав Ctrl+S. В случае обнаружения компилятором ошибки исправьте её и снова выполните п. 1.7.3.

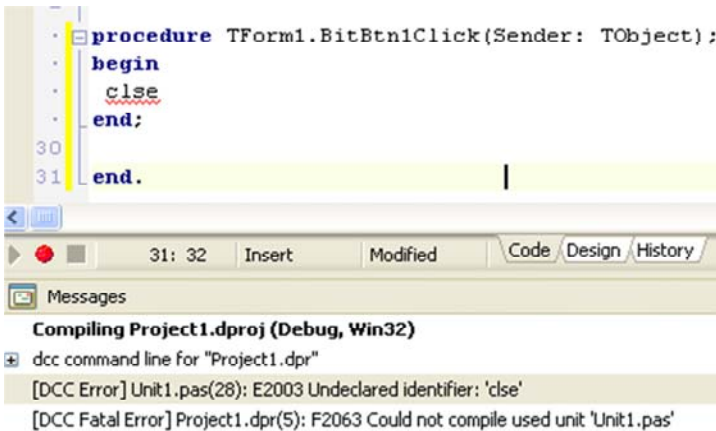


Рис. 26. Сообщения компилятора при ошибке

После успешной компиляции можно программу запускать в режиме отладки, нажав F9 (см. п.1.4 и рис. 8). Появится главная форма в режиме отладки (см. рис. 8), а в панели задач дополнительно появится название формы (рис. 27).



Рис. 27. Добавление названия главной формы в панели задач при запуске программы в режиме отладки

Рекомендуется не работать с программой в режиме создания, пока она запущена в режиме отладки. Если случайно выполнен переход к режиму создания во время работы режима отладки, можно либо щёлкнуть по названию формы в панели задач для перехода программе в режим отладки, либо нажать Ctrl+F9 и в выскочившем сообщении, содержащем слова Debug Session (рис. 28), выбрать кнопку Terminate (прервать сессию отладки).

После проверки работы программы в режиме отладки следует закрыть режим отладки, нажав на кнопку «Выход» (если она запрограммирована) или на крестик вверх формы (см. рис. 8). Программа вернётся в режим создания.

1.7.5. Запустите программу, в режиме отладки нажав F9. Нажмите кнопку «Выход». Программа должна будет вернуться в режим создания.

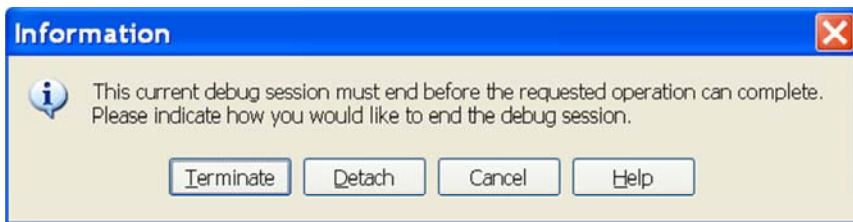


Рис. 28. Сообщение о необходимости прервать сессию отладки

1.7.6. Закройте проект. Если будет выдано сообщение с подтверждением сохранения изменений (рис. 29), выберите Yes (Да).

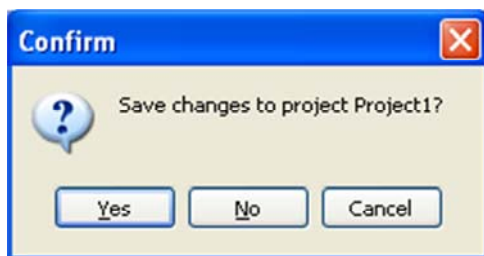


Рис. 29. Запрос о сохранении изменений в проекте.

Контрольные вопросы

1. Основные файлы проекта и их назначение.
2. Создание приложения Vcl Forms Application.
3. Назначение вкладок Object Inspector.
4. Переключение с формы на Unit и обратно.
5. Первое сохранение проекта.
6. Открытие созданного проекта.
7. Работа с проектом в режиме создания.
8. Компиляция программы.
9. Запуск программы в режиме отладки.
10. Чем форма в режиме отладки будет отличаться от формы в режиме создания?
11. Что будет написано в верхней строке Delphi при работе в режиме отладки, а что – при работе в режиме создания?

12. Как закрыть режим отладки, чтобы вернуться к режиму создания?

13. Расположение файла .exe, созданного после первой успешной компиляции

14. Как положить компонент на форму?

15. Перемещение компонента по форме и изменение его размеров при помощи мыши.

16. Какие свойства изменяются в Object Inspector при растягивании или сжатии компонента при помощи мыши?

17. Какие свойства изменяются в Object Inspector при перетаскивании компонента при помощи мыши?

18. Как убедиться, что в Object Inspector отображаются свойства именно того компонента, для которого необходимо поменять значение какого-либо из них, и что делать, если в Object Inspector отображаются свойства не того компонента?

19. Какое свойство кнопки BitBtn отвечает за отображаемый текст на ней?

20. Как запрограммировать кнопку BitBtn, чтобы по щелчку по ней программа завершала работу? Что именно должен записать программист, а что записывается автоматически?

21. Что будет автоматически добавлено в раздел interface Unit, если положить компонент на форму?

22. Что будет автоматически добавлено в раздел interface Unit, если запрограммировать событие OnClick («по щелчку») какого-либо компонента?

23. Как определить, успешно ли выполнена компиляция?

24. При помощи какой клавиши выполняется запуск программы в режиме отладки?

25. Как определить при помощи командной строки, запущен режим отладки или нет?

26. Следует ли работать в режиме создания программы при незакрытом режиме отладки?

27. Что делать, если случайно выполнен переход к режиму создания во время работы режима отладки

28. Если при попытке закрыть проект выдалось сообщение с подтверждением сохранения изменений, то что лучше делать?

2. ИЗМЕНЕНИЕ СВОЙСТВ КОМПОНЕНТА В ПРОГРАММЕ. КОМПОНЕНТ ВЫБОРА RADIOBUTTON

В процессе работы программы может возникнуть необходимость поменять значение какого-либо свойства определённого компонента, чтобы вывести полученные в процессе работы программы результаты расчёта на форму, поменять какие-либо параметры компонента (например, изменить число строк таблицы) и т. д. Кроме того, некоторые свойства (например, число строк и столбцов таблицы **StringGrid**) можно менять только в программе.

Для большого числа свойств это делается следующим образом:

```
<Имя формы>.<Имя компонента>.<Название свойства>:=<Значение свойства>
```

Для изменения надписи кнопки `BitBtn1` (см. рис. 19) с "Выход" на "Выход запрещён" можно записать в текст программы так:

```
Form1.BitBtn1.Caption:='Выход запрещён'
```

`Form1` – это имя (Name) формы, на которой расположен компонент `BitBtn1`, `BitBtn1` – это имя компонента, свойство которого требуется изменить, а `Caption` – это название свойства.

Между названием свойства и его значением расположен *оператор присваивания* ":=".

Оператор присваивания:

```
<Переменная (или свойство)>:=<Требуемое значение>
```

В результате использования оператора присваивания то, что слева от него, станет равным тому, что справа от него. Если справа от него находится арифметическое или какое-либо другое выражение, то вначале оно выполнится, а затем то, что находится слева от оператора присваивания, станет равным результату выражения справа от него.

- Если записано: $a := 2$, то переменная a примет значение 2.
- Если $a = 2$ и $b = 12$, то после выполнения действия $a := b$ будет $a = b = 12$.
- Если $a=2$, то после $a := a + 45$ будет $a = 2 + 45 = 47$.

Свойства компонентов, как и переменные, могут принимать только определённые значения. Значения определяются типом переменных или свойств.

Список возможных значений некоторых свойств можно узнать при работе с Object Inspector. Рассмотрим это на примере свойства Color (цвет).

Задача 2.1.

Изменение цвета панели при помощи двух командных кнопок: с зелёного на обычный и обратно.

2.1.1. Создайте новую папку для проекта (например, с именем Lr2), создайте проект и сохраните в папку (см. гл. 1). Далее положите на форму Panel (вкладка Standard).

2.1.2. Положите две командные кнопки BitBtn (вкладка Additional) на форму, но не на Panel.

2.1.3. В Object Inspector измените названия Caption кнопки (см. рис. 17, 18): первой кнопке дайте название «Зелёная», а второй – «Обычная» (рис. 30).

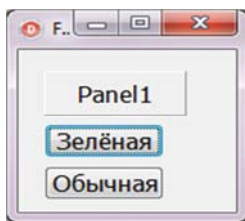


Рис. 30. Вид формы к заданию 2.1.

Далее следует запрограммировать кнопку «Зелёная» (BitBtn1) так, чтобы при щелчке по ней панель окрашивалась в зелёный цвет, а кнопку «Обычная» (BitBtn2) – в тот цвет, который назначен по умолчанию.

Необходимо определить, какое свойство отвечает за изменение цвета панели и какие значения оно может принимать.

За изменение цвета панели отвечает свойство **Color**. Выделим панель, затем найдём свойство Color в Object Inspector. По умолчанию его значение будет равно clBtnFace.

Для изменения свойства оно должно быть выделено, то есть строка с названием свойства должна быть выделена фоном, а внизу Object Inspector должно быть написано название свойства (в данном случае – Color). Если внизу Object Inspector написано название другого свойства, то необходимо выделить свойство Color, щёлкнув мышью его названию в левом столбце (рис. 31).

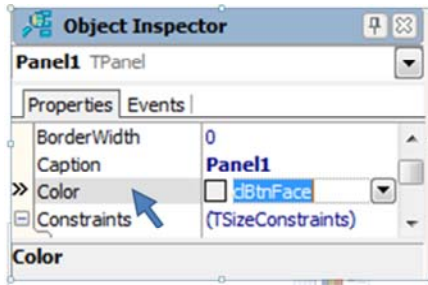


Рис. 31. Выделение свойства Color.

После того как свойство Color выделено, справа от его текущего значения появится стрелка ▼ (рис. 32).

Если нажать на стрелку ▼, то раскроется список возможных значений свойства Color. Найдём значение clGreen, соответствующее зелёному цвету, и запишем в тетрадь или на листок его название.

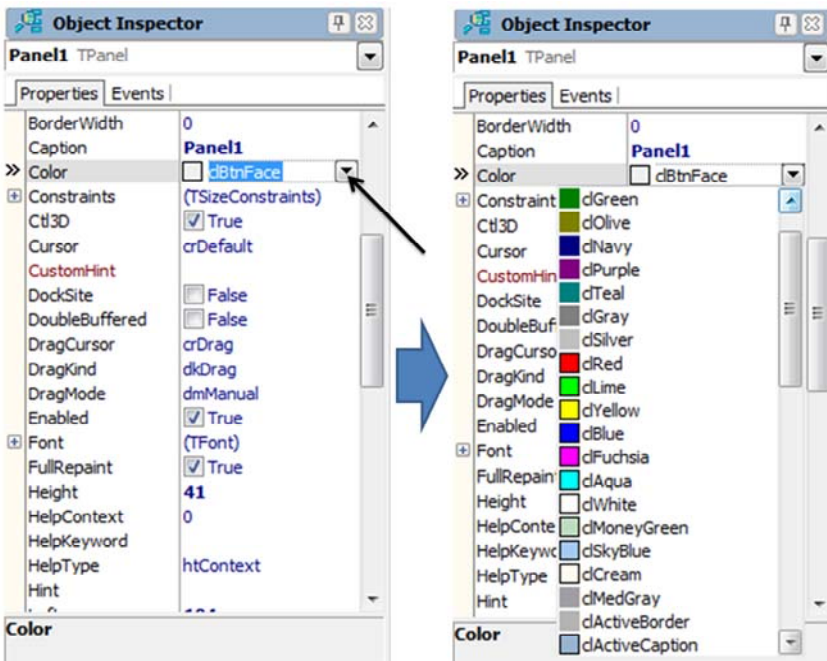


Рис. 32. Значения свойства Color

Если необходимо, чтобы при запуске программы цвет панели был обычным, оставим его значение `clBtnFace`.

2.1.4. Выделите Panel. Найдите свойство Color панели в Object Inspector и выделите свойство Color в Object Inspector. Перепишите в тетрадь или на лист бумаги значение свойства Color по умолчанию.

2.1.5. Раскройте список значений свойства Color. Найдите в данном списке значение свойства Color, соответствующее зелёному цвету, и перепишите его.

Чтобы при щелчке мышью по кнопке «Зелёная» (**BitBtn1**) панель становилась зелёной, необходимо запрограммировать событие **OnClick** («по щелчку») кнопки «Зелёная» (**BitBtn1**). Для этого сделаем двойной щелчок мышью по данной кнопке. Откроется Unit1. Появится заготовка процедуры **BitBtn1Click** (рис. 33).

```
procedure TForm1.BitBtn1Click(Sender: TObject);  
begin  
  
end;
```

Рис. 33. Заготовка процедуры BitBtn1Click

В разделе операторов (то есть между `begin` и `end`) процедуры присвоим значение свойству Color (цвет) формы значение `ClGreen` (зелёный), записав имя формы, имя компонента, имя свойства и значение свойства (рис. 34).

```
procedure TForm1.BitBtn1Click(Sender: TObject);  
begin  
    Form1.Panel1.Color:=clGreen;  
end;
```

Рис. 34. Присвоение значения свойству Color в процедуре BitBtn1Click

Далее выполним компиляцию при помощи комбинации `Ctrl+F9`. В случае успешной компиляции слева от текста программы появятся синие точки (см. рис. 25). В случае обнаружения компилятором ошибок следует их исправить, после чего повторно выполнить компиляцию.

После успешной компиляции вернёмся на форму, нажав `F12`.

Далее запрограммируем кнопку «Обычная» (**BitBtn2**) так, чтобы при щелчке мышью по ней панели возвращался обычный цвет. Для этого создадим процедуру `BitBtn2Click`, щёлкнув 2 раза по кнопке `BitBtn2`, после чего в разделе операторов присвоим свойству **Color** (цвет) панели значение `clBtnFace` (рис. 35), а затем выполним компиляцию.

```
procedure TForm1.BitBtn2Click(Sender: TObject);  
begin  
    Form1.Pane11.Color:=clBtnFace  
end;
```

Рис. 35. Присвоение значения свойству `Color` в процедуре `BitBtn2Click`

Запустим программу в режиме отладки, нажав `F9`. Проверим работу кнопок. Закроем режим отладки и вернёмся к работе в режиме создания программы. Для этого нажмём на `×` вверху формы в режиме отладки (см. рис. 8, поз. 2).

При этом для более быстрого выполнения работы и меньшей вероятности ошибок можно пользоваться всплывающей подсказкой. Когда введём имя формы `Form1` при программировании кнопки `BitBtn1` и после `Form1` поставим точку, то сразу появится всплывающая подсказка с перечислением всего того, что может относиться к форме (рис. 36).

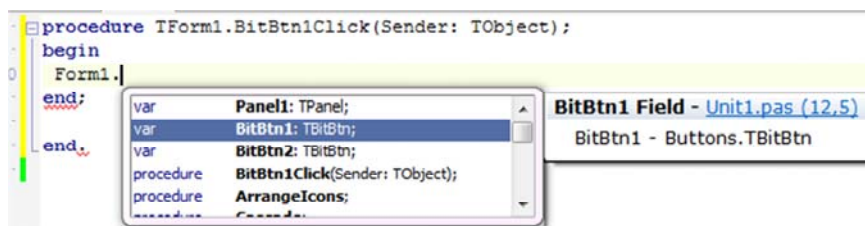


Рис. 36. Всплывающая подсказка.

Можно выбрать в раскрывшемся списке то, что требуется (имя компонента: `Panel`), после чего нажать ввод.

Чтобы выбор происходил быстрее, рекомендуется написать первую букву (буквы) названия компонента, после чего список

сократится и можно будет быстрее выбрать из него требуемое имя компонента (рис. 37).

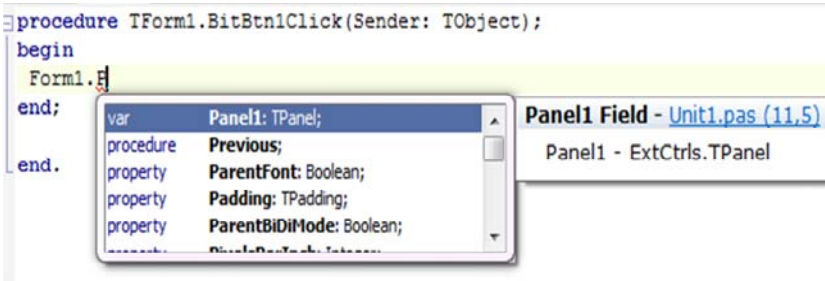


Рис. 37. Изменение содержимого всплывающей подсказки после ввода первой буквы нужного слова.

Можно всплывающую подсказку игнорировать вообще.

Если при вводе имени компонента, свойства и т. д. всплывающая подсказка перестала что-либо предлагать, то это может свидетельствовать об ошибке: такого компонента или свойства нет либо оно неверно записано (рис. 38).

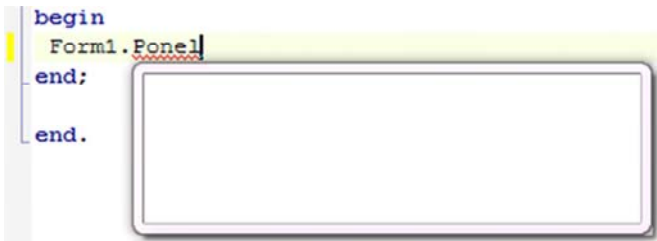


Рис. 38. Вид окна всплывающей подсказки при ошибке

Однако если всплывающая подсказка не появилась вообще, то это не значит, что где-то есть ошибка: возможно, например, что всплывающая подсказка просто отключена.

Если компонент и процедура-обработчик, в которой необходимо поменять его свойство, относятся к одной и той же форме, то имя формы перед именем компонента допускается не записывать. Например, в заголовке процедуры BitBtn1Click присутствует название Form1, поэтому в разделе операторов данной процедуры допускается записывать Panel1.Color вместо Form1.Panel1.Color (рис. 39).

```

procedure TForm1.BitBtn1Click(Sender: TObject);
begin
    Panel1.Color:=clGreen;
end;

```

Рис. 39. Изменение свойства компонента (если компонент и процедура, в которой происходит изменение его свойства, относятся к одной и той же форме)

Если же процедура одного модуля меняет свойства компонента другого модуля (например, Panel находится на Form2, а кнопка для изменения свойства панели – на Form1), то обязательно нужно записывать имя формы перед именем компонента (рис. 40). Также необходимо записывать имя формы, если изменение свойства компонента происходит не в процедуре-обработчике события, а в какой-либо другой процедуре (рис. 41).

```

procedure TForm1.BitBtn1Click(Sender: TObject);
begin
    Form2.Panel1.Color:=ClGreen;
end;

```

Рис. 40. Изменение свойства компонента (если компонент и процедура, в которой происходит изменение его свойства, относятся к разным формам)

2.1.6. Запрограммируйте кнопку «Зелёная» (BitBtn1) согласно рис. 34 или 39. Выполните компиляцию при помощи Ctrl+F9.

2.1.7. Запрограммируйте кнопку «Обычная» (BitBtn2) согласно рис. 35. Выполните компиляцию при помощи Ctrl+F9.

```

procedure hc;
begin
    Form1.Panel1.Color:=clGreen;
end;

```

Рис. 41. Изменение свойства компонента в процедуре, которая не является обработчиком какого-либо свойства компонента.

2.1.8. Запустите программу в режиме отладки, нажав F9. проверьте работу кнопок в режиме отладки, после чего закройте режим отладки.

2.1.9. Сохраните изменения (File>Save или Ctrl+S) и закройте проект.

Переключатели RadioButton

Задача 2.1. выполнена с использованием командных кнопок для изменения цвета. То же самое можно сделать при помощи зависимых кнопок-переключателей RadioButton, компонентов RadioGroup и других.

Задача 2.2.

Изменение цвета панели при помощи зависимых кнопок выбора RadioButton.

2.2.1. Создайте папку Lr2a. Создайте новый проект и сохраните его в эту папку.

2.2.2. На вкладке Standard найдите компонент RadioButton (рис. 42). Положите 4–6 кнопок RadioButton на форму (рис. 43).



Рис. 42. Зависимая кнопка выбора RadioButton



Рис. 43. Кнопки RadioButton на форме

2.2.3. Запустите программу в режиме отладки, нажав F9. При щелчке по кнопке появится точка (рис. 44). Если на кнопке точка, то это значит, что выбрана именно данная кнопка. Щёлкните по каждой из кнопок по очереди. Убедитесь, что включённой на форме без использования компонентов GroupBox Panel может быть только одна кнопка RadioButton. Закройте форму, работающую в режиме отладки, нажав на × в её верхнем правом углу.

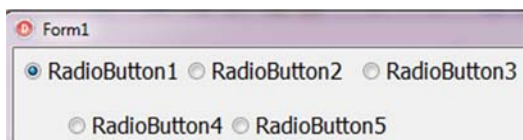


Рис. 44. Кнопки при работе программы в режиме отладки

Свойство, которое отвечает за состояние кнопки, называется **Checked**. Оно может принимать 2 значения: true – кнопка включена, false – кнопка выключена. Чтобы сделать по умолчанию какую-либо из кнопок включённой, можно изменить значение свойства Checked в Object Inspector (рис. 45).

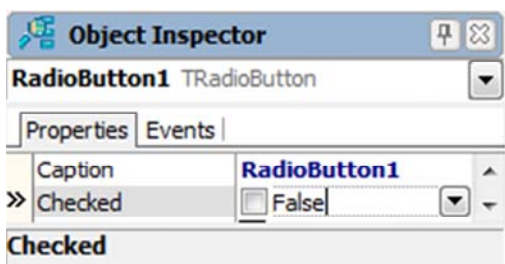


Рис. 45. Свойство Checked в Object Inspector

2.2.4. В режиме создания программы поменяйте в Object Inspector для каждой из кнопок свойство Checked с false на true. Убедитесь, что без использования компонентов GroupBox, Panel свойство Checked может быть равным true только для одной кнопки RadioButton.

Однако может возникнуть необходимость сделать несколько групп переключателей. Например, одна группа переключателей для выбора единиц измерения одной величины, другая – другой, третья – третьей. Чтобы в каждой из групп могла быть включённой одна кнопка, необходимо каждую группу кнопок положить на отдельную Panel или на отдельный GroupBox.

2.2.5. Удалите все кнопки RadioButton с формы.

2.2.6. Положите на форму один компонент Panel и два компонента GroupBox (оба на вкладке Standard). При этом Panel и 2 GroupBox не должны лежать друг на друге.

2.2.7. На Panel положите 3–4 кнопки RadioButton. Также по 3–4 кнопки RadioButton положите на каждый GroupBox (рис. 46).

2.2.8. Запустите программу в режиме отладки, нажав F9. Щёлкните по каждой из кнопок по очереди, чтобы убедиться, что может быть включено по одной кнопке на Panel и каждом GroupBox. Закройте режим отладки и вернитесь к работе с режимом создания.

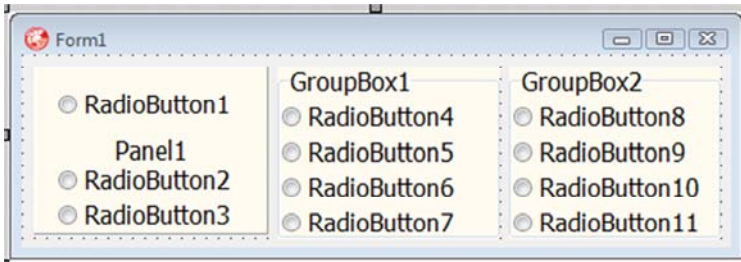


Рис. 46. Кнопки RadioButton на панели и контейнерах GroupBox

2.2.9. В Object Inspector последовательно для каждой кнопки поменяйте значение свойства Checked с false на true. Убедитесь, что свойство Checked может быть равным true одновременно для одной кнопки, лежащей на панели, одной кнопки, лежащей на GroupBox1, и одной кнопки, лежащей на GroupBox2.

Если передвигать Panel или GroupBox, то вместе с ними будут передвигаться и все компоненты, которые на них расположены. Дерево объектов (рис. 47) располагается обычно слева от формы над Object Inspector. Компоненты Radiogroup4, 5, 6, 7 начинаются на ветви, идущей от GroupBox1; значит, они будут дочерними компонентами для GroupBox1, а GroupBox1 будет для них материнским компонентом. Для компонента есть свойства, начинающиеся на слово Parent (рис. 48). Если, например, свойство ParentFont принимает значение true, то это значит, что при изменении шрифта материнского компонента точно также изменится и шрифт дочернего компонента. Если свойство ParentFont принимает значение false, то при изменении шрифта материнского компонента шрифт дочернего компонента не изменится. Для GroupBox и Panel в данном случае в качестве материнского компонента будет форма.

2.2.10. Передвиньте Panel или GroupBox с кнопками.

2.2.11. Удалите Panel. Убедитесь, что все кнопки, расположенные на Panel, тоже удалились.

2.2.12. Для одной из кнопок RadioButton, лежащей на GroupBox1, установите значение false свойства ParentFont. Выделите GroupBox1 и измените размер или тип шрифта для него для при помощи свойства Font. Для этого выделите свойство Font в Object Inspector и щёлкните по кнопке с тремя точками (рис. 49) в правом конце строки: откроется диалоговое окно изменения параметров шрифта.

Убедитесь, что не изменится размер или тип шрифта только для той кнопки GroupBox1, для которой свойство ParentFont приняло значение false. Теперь снова верните значение true свойству ParentFont той кнопки: её размер или тип шрифта станет таким же, как и у остальных кнопок, лежащих на GroupBox1.

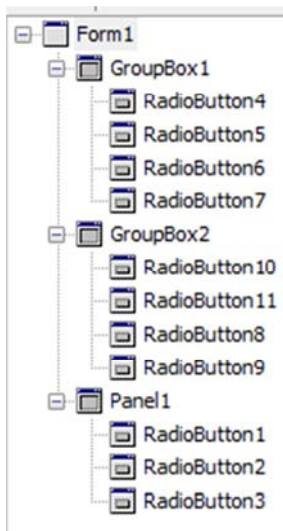


Рис. 47. Дерево объектов

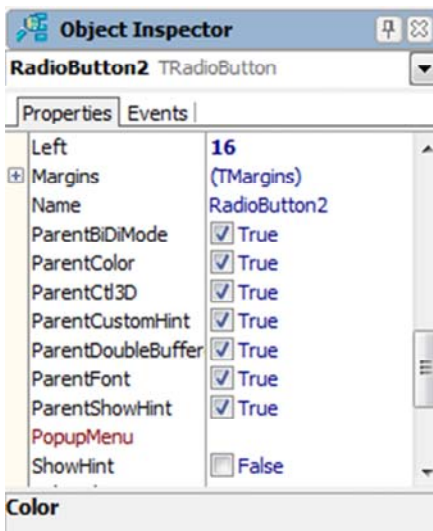


Рис. 48. Свойства Parent



Рис. 49. Свойство Font

2.2.13. Удалите GroupBox1 и GroupBox2. Вместе с ними удалятся и все оставшиеся радиокнопки, форма станет чистой.

2.2.14. Для изменения цвета панели положите на форму Panel и GroupBox. Присвойте свойству Caption компонента GroupBox значение «Цвет панели» при помощи Object Inspector. Положите на GroupBox две кнопки RadioButton, поменяйте Caption RadioButton1 на «Зелёный» и RadioButton2 на «Обычный» (рис. 50).

Состояние кнопок RadioButton должно соответствовать цвету панели. Если панель покрашена в обычный цвет, то должна быть включена кнопка «Обычный», а если – в зелёный цвет, то кнопка

«Зелёный». По умолчанию при запуске программы оставим обычный цвет. Значит, при запуске программы должна быть включённой кнопка «Обычный». Для этого необходимо в Object Inspector свойству Checked кнопки «Обычный» присвоить значение true.

2.2.15. Присвойте значение true свойству Checked кнопки «Обычный» при помощи Object Inspector. Кнопка «Обычный» станет включена (рис. 50).

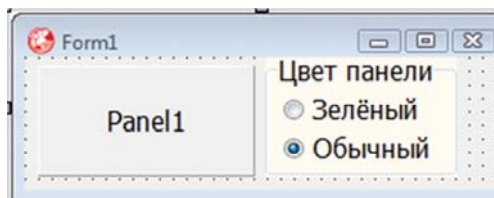


Рис. 50. Форма с Panel, GroupBox и RadioButton

Кнопки RadioButton программируется точно так же, как и BitBtn. Разница в том, что если кнопка RadioButton уже включена (то есть внутри неё есть «точка»), то щелчок по кнопке будет игнорироваться и соответствующая ей процедура OnClick выполняться не будет. Если кнопка выключена, то по щелчку по ней программа выполнится и кнопка включится. В то же время процедура OnClick кнопки BitBtn будет выполняться по каждому щелчку по кнопке BitBtn: в задаче 2.1 если панель уже зелёная, по щелчку по кнопке BitBtn «Зелёная» программа ещё раз попытается покрасить панель в зелёный цвет.

2.2.16. Два раза щёлкните по RadioButton1 «Зелёная» и в разделе операторов (между begin и end) раскрывшейся процедуры RadioButton1Click запишите **Panel1.Color:=clGreen** в соответствии с рис. 39 или 34. Выполните компиляцию при помощи Ctrl+F9.

2.2.17. Два раза щёлкните по RadioButton2 «Обычная» и в разделе операторов раскрывшейся процедуры RadioButton2Click запишите **Panel1.Color:=clBtnFace**. Выполните компиляцию (Ctrl+F9).

2.2.18. Сохраните изменение (Ctrl+S) и запустите программу в режиме отладки (F9). Проверьте работу программы в режиме отладки, после чего закройте режим отладки, нажав на × в верхнем правом углу формы.

Группы свойств

Свойства могут быть собраны в группы.

Например, размер шрифта, цвет шрифта, высота шрифта и некоторые другие параметры шрифта объединены в группу свойств Font (рис. 51), а начертание (жирный, курсив, подчёркивание) – в подгруппу Style. Для работы со свойством Font при помощи Object Inspector можно раскрыть диалоговое окно, щёлкнув по трём точкам в конце строки (см. рис. 49). Но для того, чтобы определить, как обращаться к тому или иному свойству шрифта в программе, предлагается раскрыть группу свойств: слева от названия группы будет «+», если группа закрыта (см. рис. 49). Чтобы группа раскрылась, необходимо щёлкнуть по «+»: группа раскроется и вместо «+» станет «-» (рис. 51). В некоторых версиях вместо «+» может быть другой значок, по которому также надо щёлкнуть для раскрытия группы.

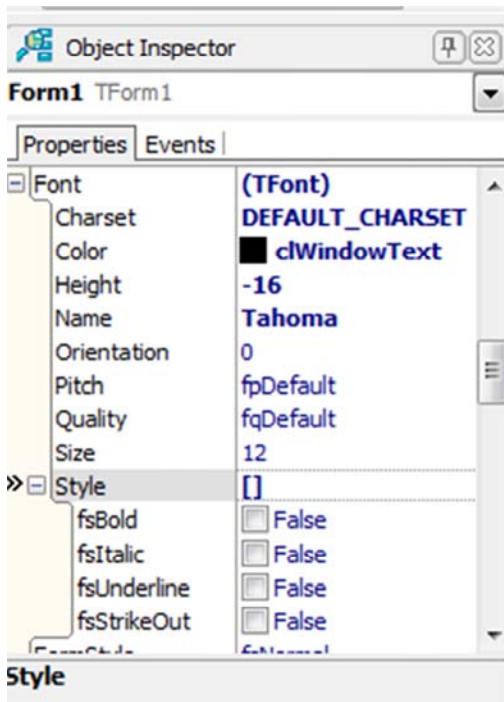


Рис. 51. Группа свойств Font

Если свойство находится в группе, то изменить его можно так:
<Форма>.<Компонент>.<Группа свойств>.<Свойство>:=<...>

А если свойство находится и в подгруппе:

<Форма>.<Компонент>.<Группа>.<Подгруппа>.<Свойство>:=<...>

Цвет шрифта находится в группе свойств Font, поэтому для изменения цвета шрифта панели перед названием свойства Color необходимо записать имя группы свойств:

```
Form1.Pane11.Font.Color:=clGreen
```

Если вместо этого записать Form1.Pane11.Color:=clGreen, то компилятор не выдаст ошибку, а изменится цвет панели вместо цвета шрифта, поскольку Panel содержит и свойство Color, не объединённое в какую-либо группу свойств.

Если записать Form1.Font.Color:=clGreen, то компилятор тоже не выдаст ошибку, но изменится цвет шрифта всей формы, поскольку форма тоже содержит группу свойств Font.

2.2.19. Положите на форму ещё одну панель (Panel2). Чтобы можно было изменять цвет её шрифта, положите один контейнер (GroupBox2), которому поменяйте Caption на «Цвет шрифта панели 2», а на него положите три кнопки RadioButton: «Зелёный», «Красный», «Чёрный» (рис. 52).

2.2.20. Поскольку по умолчанию цвет шрифта чёрный (авто), включите кнопку «Чёрный», установив для неё значение true свойству Checked.

2.2.21. Запрограммируйте все 3 кнопки. Кнопка «Красный» должна менять цвет шрифта на красный:

```
Form1.Pane11.Font.Color:=clRed или
```

```
Pane11.Font.Color:=clRed
```

Кнопка «Зелёный» должна менять цвет шрифта на зелёный, а кнопка «Чёрный» – на чёрный.

Для чёрного шрифта можно ставить значение свойства как то, которое по умолчанию (clWindowText), так и ClBlack. Чем они отличаются? Тем, что если указать цвет ClBlack, то цвет будет чёрным, независимо от настроек Windows, а если указать clWindowText, то при изменении значения настроек Windows цвет может быть и не чёрным, а каким-либо другим.

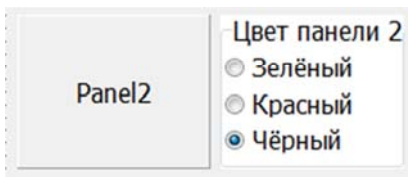


Рис. 52. Panel2

Замечание. Именно для изменения цвета наилучшим образом подходит компонент выбора ColorBox. Изменение цвета при помощи RadioButton рассмотрены для изучения работы со свойствами и компонентом RadioButton.

Если группа (подгруппа) содержит только свойства, принимающие значения true или false, то изменение их значений происходит несколько по-другому: после оператора присваивания в квадратных скобках перечисляются те свойства, значения которых равно true.

Чтобы сделать шрифт формы таким, как в Object Inspector на рис.1, когда все свойства (жирный – fsBold, курсив – fsItalic, подчёркнутый – fsUnderline, зачёркнутый – fsStrikeOut) равны false, то есть шрифт обычный, необходимо записать так же, как и в строке с названием подгруппы Style:

```
Form1.Font.Style:=[]
```

Теперь сделаем шрифт жирным, установив значение true свойству fsBold (рис. 53). В строке со свойством Style станет написано: [fsBold]. Значит, чтобы шрифт был только жирным без подчёркивания, зачёркивания и не курсив, нужно записать:

```
Form1.Font.Style:=[fsBold]
```

Установим свойству Underline значение true (рис. 54), чтобы появилось нижнее подчёркивание. В строке со свойством Size станет написано: [fsBold,fsUnderline]. Значит, чтобы установить в программе жирный шрифт с подчёркиванием, можно записать:

```
Form1.Font.Style:=[fsBold,fsUnderline].
```

Чтобы реализовать при помощи RadioButton все возможности изменения начертания шрифта и их возможные комбинации, потребуется минимум 8 кнопок. Лучше использовать для этого независимые кнопки выбора (CheckBox или CheckListBox) или пользоваться другими средствами Delphi. При использовании малого количества комбинаций (например, только обычный либо жирный шрифт или только обычный, жирный или курсив) использование зависимых кнопок выбора (например, RadioButton) может быть оправдано.

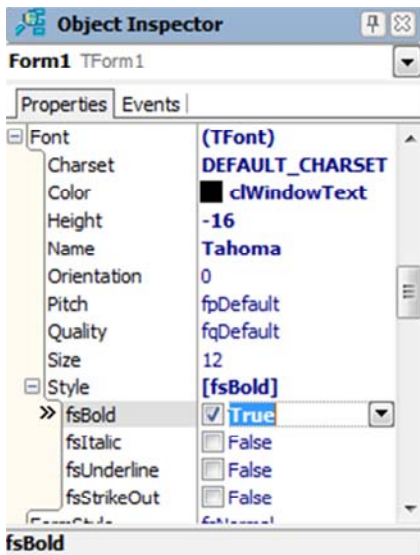


Рис. 53. Жирный шрифт

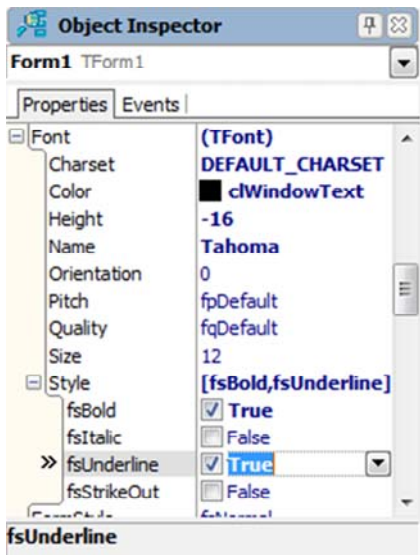


Рис. 54. Жирный шрифт с подчёркиванием

2.2.22. Положите на форму ещё один контейнер (GroupBox3) и положите на него 3 RadioButton. Измените свойство Caption контейнера GroupBox3 на «Начертание шрифта», а для кнопок Caption поменяйте на «Обычный», «Жирный», «Жирный курсив». Поскольку по умолчанию шрифт обычный, включите кнопку «Обычный», указав в Object Inspector свойству Checked значение true. Запрограммируйте кнопки, чтобы они соответствующим образом меняли начертание шрифта формы. Чтобы посмотреть, как правильно присваивать значение подгруппе свойств Style для получения жирного курсива, поставьте значения true свойствам fsBold (жирный) и fsItalic (курсив) подгруппы Style в Object Inspector.

2.2.23. Проверьте работу программы в режиме отладки, после чего закройте режим отладки и закройте проект, сохранив изменения.

Контрольные вопросы

1. Работа с оператором присваивания.
2. Как присвоить какое-либо значение свойству Caption компонента BitBtn в тексте программы?
3. Как пользоваться всплывающей подсказкой?
4. Как будет выглядеть окно всплывающей подсказки, если название свойства вводится с ошибкой?
5. Означает ли отсутствие всплывающей подсказки наличие ошибки при вводе названия свойства?
6. Когда название формы перед названием компонента записывать обязательно, а когда можно не записывать?
7. Что представляет собой компонент RadioButton?
8. Какое свойство кнопки RadioButton отвечает за её состояние (включена или выключена)? Чему равно значение данного свойства, если кнопка включена, а чему, если кнопка выключена?
9. Если положить несколько кнопок RadioButton на форму, то сколько из них может одновременно находиться во включённом положении при отсутствии Panel или GroupBox на форме?
10. Что делать, если с использованием RadioButton требуется создать несколько групп переключателей и в каждой группе должно быть включено по одной кнопке?
11. Как сделать, чтобы при запуске программы была включена одна из кнопок, расположенная на панели?
12. Если при работе программы одна из кнопок RadioButton включена, то можно ли, не внося изменения в программный текст, сделать так, чтобы все кнопки оказались выключены?
13. Если запрограммировано событие «По щелчку» (OnClick) для кнопки RadioButton, то будет ли оно выполняться, если щёлкнуть по кнопке, которая уже включена?
14. Если свойства объединены в группы, то как раскрыть группу свойств в Object Inspector?
15. Как сделать, чтобы для группы свойств Font (шрифт) свойства, входящие в неё, отображались в Object Inspector, и как сделать, чтобы для изменения свойств группы Font открылось диалоговое окно?
16. Как изменить значение свойства, которое входит в группу или подгруппу свойств?

3. ЛИНЕЙНЫЕ АЛГОРИТМЫ

Слова языка

Слова языка отделены друг от друга разделителями (пробелом, символом конца строки, комментарием) и несут определённый смысл в программе. Слово образуется неделимой последовательностью знаков алфавита языка.

В качестве букв используются строчные и заглавные буквы латинского алфавита и знак подчёркивания `_`. Регистр букв не учитывается. В более новых версиях Delphi допускается использование кириллицы. В качестве цифр используются арабские цифры. В языке могут использоваться простые (табл. 3.1) и составные (табл. 3.2) символы, а также пробел (относится к простым символам). В листингах программы пробел может обозначаться как `_`.

Таблица 3.1

Простые символы языка

Символ	Название	Символ	Название
+	Плюс	{ }	Фигурные скобки
-	Минус	.	Точка
*	Звёздочка	,	Запятая
/	Дробная черта	:	Двоеточие
=	Равно	;	Точка с запятой
>	Больше	'	Апостроф
<	Меньше	[]	Квадратные скобки
#	Номер	\$	Знак денежной единицы

К словам языка относятся:

- идентификаторы, которые определяет программист (их также называют «идентификаторы пользователя»);
- стандартные идентификаторы, служащие для обозначения заранее определённых разработчиками типов данных, констант, процедур, функций (функция `sin`, идентификаторы `extended`, `integer` и т. д.);
- зарезервированные слова (`begin`, `end`, `else` и т. д.)

Составные символы языка

Символ	Название
:=	Присваивание
<>	Не равно
..	Диапазон значений
<=	Меньше или равно
>=	Больше или равно
(. .)	Альтернатива квадратных скобок
(* *)	Альтернатива фигурных скобок
//	Комментарии

В процедуре на рис. 55:

- procedure, const, var, begin, string – зарезервированные слова; они при настройках, назначенных Delphi по умолчанию, выделяются жирным шрифтом;
- extended, boolean – стандартные идентификаторы;
- a, b, c, str, ece – идентификаторы пользователя.

```

procedure TForm1.BitBtn1Click(Sender: TObject);
const g=9.81;
Var a,b,c:extended; str:string; ece:boolean;
begin

```

Рис. 55. К идентификаторам

Правила составления идентификаторов пользователя:

- 1) идентификатор может начинаться только с буквы или знака подчёркивания (исключение для обозначения меток);
- 2) идентификатор может состоять только из букв, цифр и знака подчёркивания (точки, кавычки и т. п. недопустимы);
- 3) между двумя идентификаторами должен быть хотя бы один пробел;
- 4) в качестве идентификаторов пользователя недопустимо использовать зарезервированные слова языка и стандартные идентификаторы.

Пример правильно записанных идентификаторов:

Nomer_doma, gi, d154a.

Пример неправильно записанных идентификаторов:

Nomer.doma – ошибка, идентификатор содержит точку;

154a – ошибка, идентификатор начинается с цифры.

Типы данных

Тип – это множество значений, которое могут принимать объекты программы (константы, переменные, функции, выражения), и совокупность операций, допустимых над этими значениями.

Основные типы данных приведены в блок-схеме на рис. 56. В Delphi также есть типы, которые на рис. 56 не приведены.



Рис. 56. Основные типы данных

Целочисленный тип: все целые числа в определённом диапазоне, при выходе за пределы которого необходимо использовать вещественный тип.

Байтовый тип: целые числа в диапазоне от 0 до 255.

Вещественный тип: все действительные числа в определённом диапазоне.

Символьный (литерный) тип: один любой символ.

Булевский (логический) тип: true (истина) или false (ложь), см. свойство Checked компонента RadioButton и свойства подгруппы Style группы Font.

Строковый тип: строка – это последовательность символов. Если для данных символьного типа допустим 1 символ, то в строке их может быть до 255.

Идентификаторы типов

Для определения данных стандартного скалярного типа используются стандартные идентификаторы. Для каждого типа может быть несколько идентификаторов. Например, если тип целочисленный, то для работы с совсем малыми числами можно использовать идентификатор `shortint`; для чисел побольше – идентификатор `smallint`, а для достаточно больших чисел уже нужно использовать, например, идентификатор `integer`. Идентификаторы типов (не все) приведены в табл. 3.3.

Таблица 3.3

Идентификаторы типов

Тип	Идентификаторы типа
Целочисленный	<code>shortint, smallint, integer, int64</code>
Байтовый	<code>byte</code>
Вещественный	<code>extended, double, real, single, comp, currency</code>
Литерный (символьный)	<code>char</code>
Булевский	<code>boolean</code>

Константы и переменные

Переменная – это элемент данных, который может менять своё значение в процессе работы программы. Все те переменные, которые вводит программист, обязательно должны быть описаны им же. Локальные переменные, доступные одной процедуре, могут быть записаны между заголовком процедуры (заголовок начинается со слова `procedure`) и областью операторов, начинающейся со слова `begin`. В процедуре на рис. 55 верхняя строчка – это заголовок процедуры, а всё, что находится между заголовком и `begin` – раздел описания для констант, меток, типов, переменных. После `begin` идёт раздел операторов, в котором описывать переменные нельзя.

При необходимости следует использовать переменные, доступные всем процедурам одного модуля, их можно описывать под `implementation`, а если они должны быть доступны и процедурам другого модуля, то и над `implementation`. В обоих случаях переменные будут глобальными. Рекомендуется без необходимости глобальные переменные не использовать.

Перед описанием переменной используют зарезервированное слово **Var** и обязательно указывают, к какому типу данных она относится:

Var <идентификатор переменной> :<идентификатор типа>;

На рис. 57 приведён пример описания переменных:

a, b, c, str, ece – идентификаторы пользователя, данные переменным (a, b, c – вещественными, str – строковой и ece – булевской);
extended и boolean – стандартные идентификаторы для данных вещественного и булевского типов соответственно;
string – зарезервированное слово для обозначения данных строкового типа.

```
procedure TForm1.BitBtn1Click(Sender: TObject);  
  Var a,b,c:extended; str:string; ece:boolean;  
  begin
```

Рис. 57. Пример описания локальных переменных

Доступ переменных в тексте программы (в разделе операторов) осуществляется по их идентификаторам. Например, чтобы присвоить значение 5,5 переменной *a*, необходимо записать *a*: = 5.5. В тексте программы в качестве дробного разделителя используется точка.

Константами называются элементы данных, значение которых известно заранее и тексте программы не изменяется.

Значение стандартных констант задано в Delphi (например, pi – число пи).

Пользовательские константы определяются в разделе описания (как и переменные). Вначале определяются константы, затем – типы (здесь определение типов данных не рассматривается), далее – переменные.

Определение константы:

Const <идентификатор константы>=<Значение константы>;

Тип констант определяется автоматически.

К примеру, на рис. 57 добавим ещё определение константы ускорения свободного падения (рис. 58).

Ввод и вывод данных

Ввод данных для Vcl Forms Application может быть выполнен при помощи компонентов формы, специальных функций (например, Input для ввода и ShowMessage для вывода) и файлов. Вывод непосредственно на экран аналогично Pascal возможен в консольном приложении, которое здесь не рассматривается.

```

procedure TForm1.BitBtn1Click(Sender: TObject);
const g=9.81;
Var a,b,c:extended; str:string; ece:boolean;
begin

```

Рис. 58. Определение констант и переменных

Для ввода однострочного текста может использоваться свойство Text компонентов Edit (вкладка Standard), LabeledEdit (вкладка Additional). При помощи свойства Text одного компонента можно ввести значение переменной строкового типа. Для этого переменной присваивается значение свойства Text. При необходимости ввода вещественной переменной следует выполнить преобразование **StrToFloat** (строковый в вещественный), а для ввода целочисленной переменной – **StrToInt** (строковый в целочисленный). Пример ввода данных строкового, вещественного и целочисленного типов приведён на рис. 59.

```

procedure TForm1.BitBtn2Click(Sender: TObject);
Var a:string; b:extended; c:integer;
begin
  a:=Edit1.Text;
  b:=StrToFloat(Edit2.Text);
  c:=StrToInt(Edit3.Text);

```

Рис. 59. Ввод данных строкового, вещественного и целочисленного типа при помощи свойства Text компонентов Edit.

Ввод символьных данных при помощи компонентов Edit осуществить сложно и на данном этапе изучения Delphi пока это делать рано.

Для вывода данных в одну строку можно использовать свойство Text компонентов Edit и LabeledEdit. При помощи свойства Caption компонента Panel можно вывести данные в одну строку, а при помощи свойства Caption компонентов Label и LabeledEdit – в несколько строк. Важно помнить, что свойство Caption компонента LabeledEdit находится в группе свойств EditLabel. Для вывода символьной и строковой переменной свойству Caption, Text компонента присваивается значение переменной. Для вывода переменной вещественного типа необходимо выполнить преобразование FloatToStr (вещественный в строковый), для вывода переменной целочисленного типа – IntToStr (допускается FloatToStr) (пример приведён на рис. 60).

```

procedure TForm1.BitBtn3Click(Sender: TObject);
Var a1:string; b1:char; c1:extended; d1:integer;
begin
  { <...> }
  Edit2.Text:=a1;
  Edit3.Text:=b1;
  Edit4.Text:=FloatToStr(c1);
  Edit5.Text:=IntToStr(d1);
end;

```



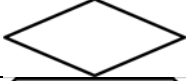

Рис. 60. Пример вывода данных на форму.

Блок-схема алгоритма

Основные элементы блок-схемы для построения алгоритма приведены в табл. 3.4 (ГОСТ 19.701-90 [5] – блоки, ГОСТ 19.003-80 [4] – их размеры).

Таблица 3.4

Основные элементы блок-схемы

Блок	№ п/п	Назначение
	1	Начало или конец программы (подпрограммы)
	2	Ввод или вывод данных, носитель не определён
	3	Процесс
	4	Условие
	5	Начало цикла
	6	Конец цикла
	7	Указание места разрыва блок-схемы; внутри блока помещается цифра с номером разрыва.
---[]	8	Комментарии

Примечание. В блоках 5 и 6 вместо фасок могут быть изображены скругления.

Задание к части 3 – найти сумму трёх чисел.

Чтобы сложить 3 числа, их необходимо ввести. Для ввода каждого числа используем компонент LabeledEdit. Для вывода результата также используем LabeledEdit.

Итого 4 LabeledEdit: для ввода 1-го, 2-го и 3-го чисел, а также для вывода суммы.

Расчёт можно проводить, например, при щелчке по командной кнопке BitBtn. Также для завершения работы программы может использоваться ещё кнопка BitBtn. Итого – 2 кнопки BitBtn (рис. 61).

Для нахождения суммы трёх чисел необходимо:

- 1) ввести исходные данные (значение трёх чисел, сумму которых необходимо найти);
- 2) найти сумму;
- 3) вывести результаты расчёта (сумму чисел) на форму.

Представим это в виде блок-схемы на рис. 62.



Рис. 61. Вид формы

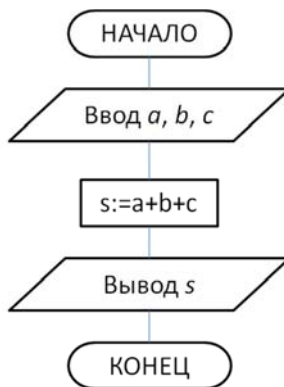


Рис. 62. Блок-схема линейного алгоритма

Приступим к записи программного текста. Запрограммируем процедуру OnClick кнопки «Расчёт». Для этого два раза по ней щёлкнем.

Возможны два режима записи программного текста: режим вставки (Insert) и записи символов поверх (Overwrite).

Режим переключается клавишей Insert и отображается внизу Unit (рис. 63, 64). Если выбран режим Overwrite (запись поверх), то символ ВВОД не вставляется, а при записи текста новые символы затирают старые. Нередко происходит включение режима Overwrite

(рис. 64) случайным нажатием клавиши Insert. Если такой режим не нужен, то следует нажать клавишу Insert и переключиться в режим вставки Insert (рис. 63).

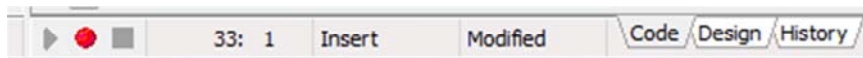


Рис. 63. Режим вставки символов



Рис. 64. Режим записи символов поверх

В разделе описания переменных (то есть между заголовком процедуры и begin) необходимо выполнить описание всех тех переменных, которые будут встречаться в процедуре. Согласно блок-схеме (см. рис. 62) в программе используются переменные a, b, c, s. Используем идентификатор вещественного типа extended для них (рис. 65).

```
procedure TForm1.BitBtn1Click(Sender: TObject);  
  Var a,b,c,s:extended;  
begin
```

Рис. 65. Описание переменных

Теперь приступим к реализации алгоритма (выполняется в разделе операторов, то есть между begin и end процедуры). Алгоритм предписывает выполнение действий строго в том порядке, в каком они заданы. Первым идёт блок ввода исходных данных: переменных a, b, c. Вводим их при помощи свойства Text компонентов LabeledEdit:

```
a:=StrToFloat(LabeledEdit1.Text);  
b:=StrToFloat(LabeledEdit2.Text);  
c:=StrToFloat(LabeledEdit3.Text);
```

Вторым идёт блок нахождения суммы:

```
s:=a+b+c;
```

Третьим идёт вывод результата (суммы s). Это можно сделать при помощи свойства Text компонента LabeledEdit4:

```
LabeledEdit4.Text:=FloatToStr(s);
```

Таким образом, программа примет вид, как на рис. 66.

```
procedure TForm1.BitBtn1Click(Sender: TObject);  
  Var a,b,c,s:extended;  
  begin  
    a:=StrToFloat(LabeledEdit1.Text);  
    b:=StrToFloat(LabeledEdit2.Text);  
    c:=StrToFloat(LabeledEdit3.Text);  
    s:=a+b+c;  
    LabeledEdit4.Text:=FloatToStr(s);  
  end;
```

Рис. 66. Программа кнопки «Расчёт»

3.1. Создайте и сохраните проект.

3.2. Положите на форму компоненты согласно рис. 61: 4 LabeledEdit, 2 командные кнопки BitBtn. Замените Caption кнопок BitBtn на «Расчёт» и «Выход». Для LabeledEdit поменяйте Caption на «Первое число», «Второе число», «Третье число» и «Сумма». Чтобы изменить свойство Caption компонента LabeledEdit, необходимо раскрыть группу свойств EditLabel (рис. 67).

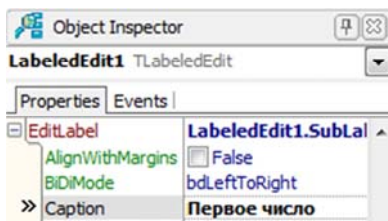


Рис. 67. Свойство Caption компонента LabeledEdit

3.3. Запрограммируйте кнопку BitBtn1 («Расчёт») согласно рис. 66. Выполните компиляцию (Ctrl + F9).

При работе программы в режиме отладки после нажатия кнопки «Расчёт» будет выдано сообщение об ошибке в том случае, если не введено значение хотя бы одного из чисел или будет введено неправильно (например, вместо цифр текст, неверный дробный разделитель и т. д.). При оставлении пустого окна будет выдано сообщение

о том, что выполнение прервано вследствие ошибки EConvertError преобразования строкового выражения в вещественное (рис. 68).

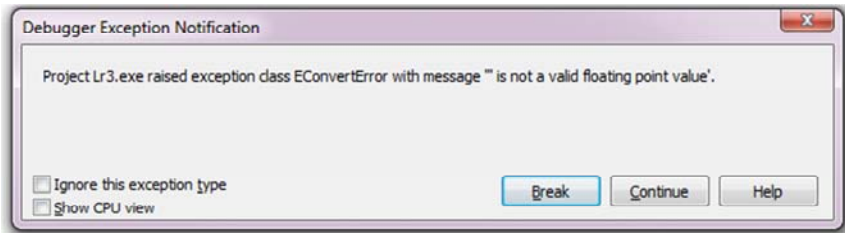


Рис. 68. Сообщение при возникновении ошибки преобразования

Если нажать кнопку Continue, то exe продолжит работу с выдачей анонсированного сообщения (рис. 69) **' is not valid floating point value**. Необходимо закрыть это сообщение кнопкой ОК, после чего ввести правильно данные.

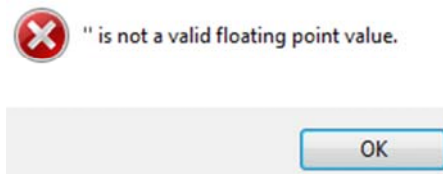


Рис. 69. Сообщение об ошибке при работе exe

Если нажать в сообщении (рис. 68) кнопку Break, то проект остановит работу и в заголовке будет записано: **Lr1 - Delphi XE - Unit 1 [Stopped] [Built]**. Такой режим в данном случае нам не нужен. Поэтому следует нажать Ctrl + F9 и в сообщении о завершении сессии отладки (см. рис. 28) нажать кнопку Terminate («Прервать»), после чего убедиться, что в заголовке проекта исчезло слово Stopped. Если при нажатии Ctrl + F9 такое сообщение не выдалось и надпись Stopped осталась, то необходимо щёлкнуть по Unit, форме или проекту в режиме создания, чтобы они стали активны, после чего снова нажать Ctrl + F9.

В данной работе при возникновении сообщения, аналогичного приведённому на рис. 68, лучше нажимать кнопку Continue, после чего закрывать следующее сообщение (рис. 69) кнопкой ОК.

Если запускать файл exe напрямую из папки Win32, а не проект в режиме отладки, то будет сразу выдаваться сообщение аналогично рис. 69 без выдачи сообщения аналогично рис. 68.

В более старых версиях Delphi при работе в режиме отладки может не быть сообщений аналогично рис. 68, 69, а вместо этого будет выдан файл проекта. В таком случае также необходимо нажать Ctrl + F9, после чего нажать кнопку Terminate в сообщении, аналогичном рис. 28.

При вводе с формы в качестве дробного разделителя по умолчанию обычно устанавливается запятая. При использовании неправильного дробного разделителя также будет возникать ошибка преобразования EConvertError.

3.4. Запрограммируйте кнопку «Выход» аналогично п. 1.7.

3.5. Запустите программу в режиме отладки. Проверьте работу в режиме отладки, введя три числа в блоки редактора «Первое число», «Второе число», «Третье число», после чего нажав кнопку «Расчёт». Закройте режим отладки, нажав кнопку «Выход».

Строковые выражения

Строковое выражение, присваиваемое переменной строкового типа, и символ, присваиваемый переменной символьного типа, обязательно заключаются в апострофы:

```
Label1.Caption:='Ошибка выполнения';
```

```
Key:='@';
```

Во всех остальных случаях апострофы не ставятся:

```
Radiogroup1.ItemIndex:=-1; //целочисленный тип
```

```
CheckBox1.Checked:=false; //булевский тип
```

```
A12:=2.5; //вещественный тип
```

```
Label1.Caption:=st1; {идентификатор переменной строкового или символьного типа}
```

Строковое выражение может состоять из частей, между которыми ставится знак «+» (плюс): <часть 1> + <часть 2> + <часть 3>. Поясняющий текст и переменные записываются в виде отдельных частей и соединяются в одно выражение знаками «+» при выводе в один

блок редактора результатов расчёта вместе с текстовыми пояснениями к ним и/или при выводе в один редактор нескольких значений переменных

Например, в задаче с нахождением суммы чисел может возникнуть следующая ситуация. Пользователь ввёл числа «7», «8» и «9» и нажал кнопку «Расчёт». Была найдена сумма и выдана в качестве результата расчёта. Затем пользователь изменяет число «7» на число «25» и не нажимает на кнопку «Расчёт».

К ПЭВМ подходит другой пользователь, который не знает, для каких именно значений получена сумма – тех, что введены сейчас, или каких-либо других. Он может ещё раз нажать кнопку «Расчёт». А если программа более сложная и расчёт выполняется долго, а также нужно создать много файлов с результатами расчётов, то очень просто перепутать, для каких именно значений они получены. Чтобы избежать подобных ситуаций, рекомендуется вместе с результатами расчёта выводить и значения исходных данных, для которых они получены.

В задаче выведем ответ в виде:

Сумма чисел <значение a >, <знач. b > и <знач. c > равна <знач. s >

Для вывода результата в таком виде строковое выражение составим из 8 блоков (табл. 3.5) с данными одного типа.

Таблица 3.5

Составляющие строкового выражения

№ блока	Должно быть на экране	Должно быть в программе
1	Сумма_чисел_	'Сумма_чисел_'
2	<значение числа a >	FloatToStr(a)
3	,_	','_'
4	<значение числа b >	FloatToStr(b)
5	_и_	'_и_'
6	<значение числа c >	FloatToStr(c)
7	_равна_	'_равна_'
8	<значение суммы s >	FloatToStr(s)

Текстовые пояснения пишутся в апострофах. Пробелы между числом и текстом также надо относить к текстовому пояснению и записывать в апострофах; если в блоках 3, 5 и 7 пробелы не записать, то текст будет написан слитно с числом.

Запишем составные части строкового выражения по порядку и соединим их при помощи знаков «+» (рис. 70).

Если запустить программу в режиме отладки и ввести числа, например, 2, 0,25 и 45, а затем нажать кнопку «Расчёт», то компонент LabeledEdit4 с результатами расчёта станет выглядеть аналогично рис. 71. При желании в блоке 3 (см. табл. 3.5) можно использовать вместо точки точку с запятой, чтобы сложнее было спутать с дробным разделителем.

```
LabeledEdit4.Text:='Сумма чисел '+FloatToStr(a)
+' ,'+FloatToStr(b)+' и '+FloatToStr(c)
+' равна '+FloatToStr(s);
```

Рис. 70. Строковое выражение для вывода результатов расчёта

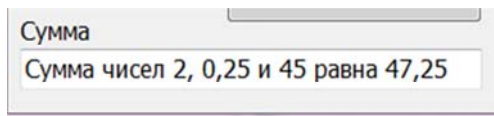


Рис. 71. Вид компонента LabeledEdit с результатами расчёта

Комментарии

Комментарий – это пояснительный текст, который можно записать в любом месте программы, где разрешён пробел. Комментарии компилятором игнорируются. Они служат для программиста. Программист помечает для себя или для другого программиста то, что считает необходимым. Также при отладке бывает целесообразным брать часть программы в комментарии, чтобы выполнять отладку программы по частям и определить, где именно допущена ошибка.

Комментарии заключаются в фигурные скобки. Всё, что находится от открывающейся до закрывающейся скобки, будет игнорироваться компилятором. Вместо фигурных скобок может использоваться их альтернатива – круглые скобки со звёздочками (см. табл. 3.2).

Также можно использовать //. Всё, что находится справа от // в строке, будет восприниматься как комментарий, а всё, что находится слева от них, будет программным текстом.

Комментарии не следует путать с директивами компилятора, которые программой не игнорируются и несут определённый смысл. Отличить их можно так: перед директивой компилятора установлен знак денежной единицы \$. Например {\$R *.dfm} – это директива компилятора, поэтому нельзя начинать с \$ комментарий, взятый в фигурные скобки или в круглые скобки со звёздочкой. // используется только для комментариев, поэтому после // ставить \$ в комментариях можно.

Чтобы сохранить предыдущий вариант вывода исходных данных для отчёта о лабораторной работе и в то же время вывести данные согласно табл. 3.5, возьмём LabeledEdit4.Text:=FloatToStr(s) в комментарии. Также добавим комментарии с целью пояснения сущности программы (рис. 72).

```
procedure TForm1.BitBtn1Click(Sender: TObject);
var a,b,c,s:extended;
  (*a,b,c -- первое, второе и третье число, s -- их сумма*)
begin
  a:=StrToFloat(LabeledEdit1.Text); // ввод первого числа
  b:=StrToFloat(LabeledEdit2.Text); // ввод второго числа
  c:=StrToFloat(LabeledEdit3.Text); // ввод третьего числа
  s:=a+b+c; //нахождение суммы
  { LabeledEdit4.Text:=FloatToStr(s); }
  LabeledEdit4.Text:='Сумма чисел '+FloatToStr(a)
    +', '+FloatToStr(b)+' и '+FloatToStr(c)
    +' равна '+FloatToStr(s); //вывод
end;
```

Рис. 72. Программа с комментариями

3.6. Измените программный текст согласно рис. 72. Выполните компиляцию Ctrl + F9. Запустите программу в режиме отладки, нажав F9, и с помощью программы найдите сумму трёх чисел. Закройте режим отладки. Если программа работает верно, закройте проект, сохранив изменения в нём.

Контрольные вопросы

1. Слова языка: чем образованы и как друг от друга отделяются?
2. Зарезервированные слова и стандартные идентификаторы: назначение, примеры.
3. Правила составления идентификаторов пользователя. Может ли идентификатор пользователя начинаться с цифры, содержать точку? Можно ли в качестве идентификатора использовать какое-либо зарезервированное слово или стандартный идентификатор?
4. Дайте определение понятию «тип» и перечислите основные типы данных.
5. Какие данные относятся к целочисленному типу, а какие – к байтовому?
6. Какие данные относятся к вещественному типу?
7. Что представляют собой данные символьного и строкового типа?
8. Что представляют собой данные булевского типа?
9. Приведите примеры идентификаторов стандартных скалярных типов.
10. Какое зарезервированное слово служит для описания данных строкового типа?
11. Что такое переменная?
12. Где и как описываются локальные переменные?
13. Что называется константами и как их описать?
14. Что такое стандартные константы?
15. Могут ли константы изменять своё значение в процессе работы программы?
16. Могут ли переменные изменять своё значение в процессе работы программы?
17. Ввод данных строкового типа при помощи компонента Edit или LabeledEdit.
18. Ввод данных вещественного типа при помощи компонента Edit или LabeledEdit.
19. Ввод данных целочисленного типа при помощи компонента Edit или LabeledEdit.
20. Вывод данных символьного и строкового типов на форму при помощи компонента Edit или LabeledEdit.
21. Вывод данных целочисленного типа на форму при помощи компонента Edit или LabeledEdit.

22. Вывод данных вещественного типа на форму при помощи компонента Edit или LabeledEdit.

23. Режимы Insert и Overwrite записи программного текста. В каком из режимов символы будут вставляться, а в каком – затирают другие? Какая клавиша осуществляет переключение режимов и где отображается, какой режим выбран?

24. Правила расстановки апострофов при выводе данных на форму или присвоении значения переменным: что нужно заключать в апострофы, а что – нет?

25. Как вывести при помощи компонента Edit или LabeledEdit на форму результаты расчёта с пояснением к ним?

26. Для чего нужны комментарии и как ограничить текст, который в них содержится?

Задачи

1. Найдите значение касательной силы тяги F_k , Н по формуле

$$F_k = \frac{M_e \cdot u_{\text{тр}} \cdot \eta_{\text{тр}}}{r},$$

если радиус колеса r равен 0,3 м; момент двигателя M_e равен 500 Н·м; КПД трансмиссии $\eta_{\text{тр}} = 0,85$; передаточное отношение трансмиссии $u_{\text{тр}} = 30$.

2. Найдите значение силы сопротивления качению для автомобиля массой m_a 2т по формуле

$$F_f = f m_a g,$$

если коэффициент сопротивления качению f равен 0,01. Ускорение свободного падения g примите равным 9,81 м/с² (Н/кг).

3. Найдите значение силы сопротивления воздуха по формуле:

$$F_b = \frac{k_b A_b v_a^2}{3,6^2},$$

если автомобиль движется со скоростью $v_a=120$ км/ч.

Коэффициент сопротивления воздуха k_b предлагается принять равным 0,2 Н·с²/м⁴, площадь лобового сопротивления $A_b = 2,0$ м².

4. РАЗВЕТВЛЯЮЩИЕСЯ АЛГОРИТМЫ. ОПЕРАТОР УСЛОВИЯ IF

Оператор условия if может записываться в виде if (если)... then (то) и if (если)... then (то)... else (иначе); могут использоваться также конструкции с вложенными операторами if.

Правила пунктуации

Точка с запятой *не ставится*:

- после else;
- перед else для оператора if;
- после then и перед then;
- после if;
- после begin.

Точка с запятой *ставится* между операторами и перед else для оператора case.

Точка с запятой *может не ставиться* перед end.

Конструкция if ... then

Конструкция записывается в виде

if <условие> then <действие>

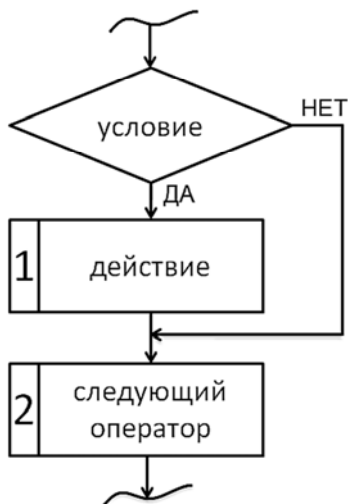


Рис. 73. Схема if... then

Блок-схема конструкции представлена на рис. 73.

Если условие выполняется, то выполняется и действие, а затем программа переходит к следующему оператору. На рис. 73 программа пойдёт по пути «Условие – Блок 1– Блок 2»

Если условие не выполняется, то программа сразу переходит к следующему оператору, не выполняя действие. На рис. 73 программа пойдёт по пути «Условие – Блок 2», минуя блок 1.

Блок условия обозначается ромбом (табл. 3.4, символ 4), из него выходят две стрелки: стрелка «ДА» и стрелка «НЕТ». Если условие выполняется, то

программа идёт по стрелке «ДА» и последовательно выполняет все блоки, которые пойдут вслед за стрелкой «ДА». Для конструкции без else после стрелки «НЕТ» блоки отсутствуют, поскольку в случае невыполнения условия программа сразу переходит по стрелке «НЕТ» к следующему оператору. Там, где стрелки сходятся, оператор условия заканчивается.

Задача 4.1.

Найти значение $y(x)$ при условии:

$$y = \begin{cases} x + 1, & x \leq 0 \\ x^2 + 5, & x > 0 \end{cases}$$

То есть если значение x лежит в промежутке $(-\infty, 0]$, то y нужно находить по 1-му выражению, а если – в промежутке $(0, \infty)$, то по 2-му.

Для решения задачи положим на форму (рис. 74) LabeledEdit для ввода x , LabeledEdit для вывода y и командные кнопки BitBtn («Расчёт» и «Выход»).

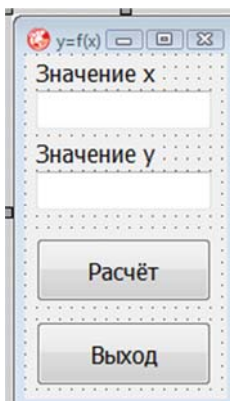


Рис. 74. Форма с компонентами для задачи 4.1

Вначале следует ввести значение x . Затем в зависимости от значения x необходимо найти y по 1-му выражению или по 2-му, после чего вывести y . При использовании конструкции if... then необходимо 2 оператора условия, первый из которых должен выполняться, если $x \leq 0$ (верно 1-ое условие), а второй – если верно второе условие $x > 0$:

если $x \leq 0$, то $y := x + 1$

если $x > 0$, то $y := x^2 + 5$

Составим блок-схему для такого случая (рис. 75).

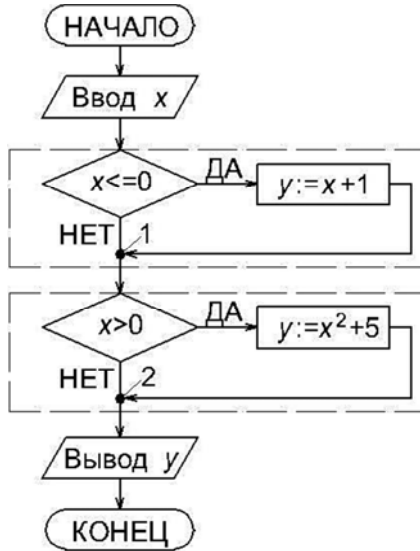


Рис. 75. Блок-схема алгоритма для задачи 4.1 с использованием конструкции if ... then

Программа должна будет работать так.

1) Ввод значения x .

2) Проверка условия $x \leq 0$. Если это условие верно, то выполненные действия $y := x + 1$ и переход к следующему оператору. Если это условие неверно, то сразу переход к следующему оператору.

3) Проверка условия $x > 0$. Если это условие верно, то выполненные действия $y := x^2 + 5$ и переход к следующему оператору. Если условие неверно (x будет находиться в другом интервале значений), то сразу переход к следующему оператору.

4) Вывод значения y .

Всегда будут проверяться оба условия (и по п. 3, и по п. 2). Если условия записаны верно, то истинным будет только одно из них и действие выполнится либо в п. 3, либо в п. 2. Если же будет до-

пущена ошибка при записи условия, то либо выполнится вначале действие в п. 2, а затем – действие по п. 3, либо не выполнится ни одно из действий.

Например, в п. 3 вместо $x > 0$ будет записано $x > -10$. Тогда на интервале $(-10; 0]$ будут верны оба условия (и $x \leq 0$, и $x > -10$), а значит, выполнится вначале действие $y := x + 1$, а затем – действие $y := x^2 + 5$. В результате в качестве результата будет рассчитан y по последнему выражению.

Если же в п. 3 вместо $x > 0$ будет записано $x > 10$, то в интервале $(0; 10]$ оба условия будут неверны, а значит, не выполнится ни действие по п. 2, ни действие по п. 3. В качестве результатов расчёта y будет взято программой любое произвольное число (возможно, 0), поскольку значению y ничего не будет присвоено.

В блок-схеме на рис. 75 каждый оператор условия обведён штриховой линией. Первый оператор условия ($x \leq 0$) будет заканчиваться в точке 1, где сходятся его стрелки «ДА» и «НЕТ». Всё, что выше ромбовидного блока условия, и всё, что ниже точки 1, к 1-му оператору условия относиться не будет.

Второй оператор условия ($x > 0$) будет начинаться от ромбовидного блока с условием $x > 0$ и заканчиваться в точке 2, где будут сходитьсь его стрелки «ДА» и «НЕТ».

Программа кнопки «Расчёт», составленная по блок-схеме на рис. 75, приведена в листинге 4.1.

Оператор if $x \leq 0$ then $y := x + 1$ можно перевести так: если $x \leq 0$, то $y := x + 1$, а оператор if $x > 0$ then $y := x^2 + 5$ – так: если $x > 0$, то $y := x^2 + 5$.

Листинг 4.1. Программа кнопки «Расчёт» для задачи 4.1 с использованием конструкции if... then.

```
procedure TForm1.BitBtn1Click(Sender: TObject);
  Var x, y: extended;
begin
  x:=StrToFloat(LabeledEdit1.Text);
  if x<=0 then y:=x+1;
  if x>0 then y:=sqr(x)+5;
  LabeledEdit2.Text:=FloatToStr(y);
end
```

4.1.2. Запрограммируйте командную кнопку «Расчёт» (см. листинг 4.1) и проверьте работу программы в режиме отладки. Закройте проект с сохранением изменений в нём.

Конструкция if ... then ... else

if <условие> **then** <действие 1> **else** <действие 1'>

Если условие истинно, то *выполняется* действие 1 и *не выполняется* действие 1', после чего программа переходит к следующему оператору.

Если условие ложно, то *не выполняется* действие 1 и *выполняется* действие 1', после чего программа переходит к следующему оператору.

То есть выполняется либо действие 1, либо действие 1'.

Они не могут выполняться оба сразу, и не может быть так, чтобы они оба не выполнились.

Блок-схема конструкции if ... then ... else приведена на рис. 76. Стрелка «ДА» показывает, как будет выполняться программа, если условие будет истинным, а стрелка «НЕТ» показывает, как будет выполняться программа, если условие будет ложным. По стрелке «ДА» идёт действие 1 (после then), а по стрелке «НЕТ» – действие 1' (после else).

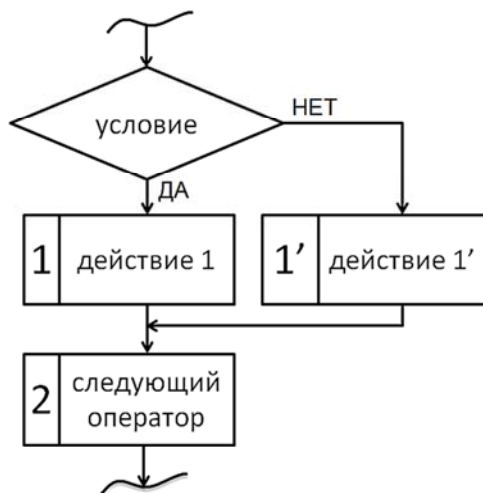


Рис. 76. Схема if ... then ... else

При использовании конструкции с else может потребоваться на одно условие меньше, чем если взять конструкцию без else. Если для решения задачи 4.1 при помощи конструкций if... then потребовалось брать 2 оператора if (1 проверял 1-е условие, другой – 2-е), то для решения при помощи конструкции if ... then ... else достаточно одного оператора if, который проверит только одно условие, второе – методом исключения.

Для задачи 4.1 могут быть только два варианта: либо $x \leq 0$, либо $x > 0$. Значит, достаточно проверить один из них; если он не выполняется, то методом исключения будет второй из них, поскольку третьего варианта нет.

Запишем условие так: если $x \leq 0$, то $y := x + 1$ иначе $y := x^2 + 5$.

```
if x<=0 then y:=x+1 else y:=x+5;
```

Блок-схему представим на рис. 77.

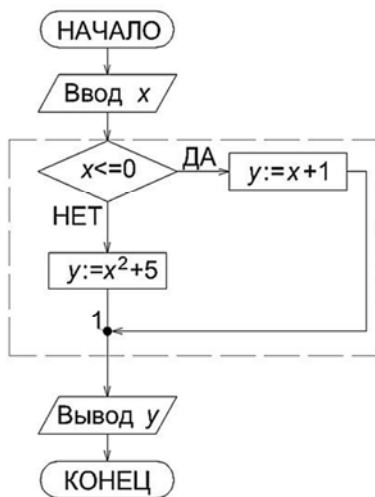


Рис. 77. Блок-схема к задаче 4.1. при использовании конструкции if ... then ... else

Из блока условия $x \leq 0$ выходят 2 стрелки: стрелка «ДА» и стрелка «НЕТ». Если условие выполняется, то программа идёт по стрелке «ДА» до точки 1 выполняя действие $y := x + 1$, и после точки 1, переходит к следующему блоку (выводу y).

Если условие не выполняется, то программа идёт до точки 1 по стрелке «НЕТ», выполняя действие $y := x^2 + 5$, и после точки 1 переходит к следующему блоку (выводу y).

Конструкция оператора условия `if` начинается с ромбовидного блока и заканчивается точкой 1, в которой сходятся стрелки «ДА» и «НЕТ». Можно представить блок-схему и в несколько другом виде (рис. 78). Блок-схемы на рис. 77 и 78 несут абсолютно одинаковый смысл.

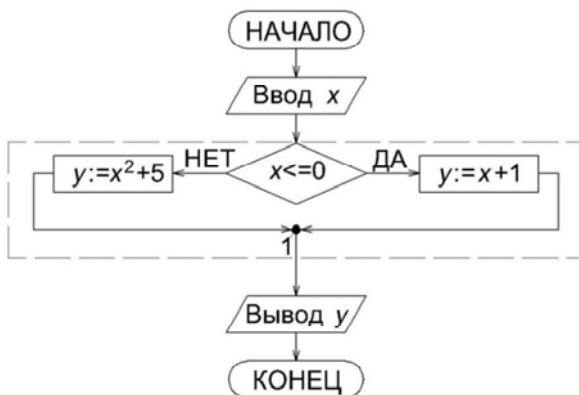


Рис. 78. Вариант блок-схемы при использовании конструкции `if ... then ... else` для задачи 4.1

Если в условии будет допущена ошибка (например, вместо $x \leq 0$ будет записано $x \leq 10$), то в любом случае выполнится одно и только одно из действий: не могут выполняться сразу два действия, и не может быть так, что ни одно из них не выполнится. При ошибке в условии будет неправильно рассчитан результат: на определённом интервале значений x , где должно будет выполняться первое действие, будет вместо него выполняться второе действие или наоборот.

Например, если по ошибке записать `if $x \leq 10$` вместо `$x \leq 0$` , то в диапазоне значений $x \in (0, 10]$ вместо $y := x^2 + 5$ будет выполняться действие $y := x + 1$.

Программный текст кнопки «Расчёт», соответствующий блок-схемам (рис. 77, 78), приведён в листинге 4.2.

Листинг 4.2. Программа кнопки «Расчёт» для задачи 4.1 с использованием конструкции if ... then ... else.

```
procedure TForm1.BitBtn1Click(Sender: TObject);
  Var x,y:extended;
  begin
    x:=StrToFloat(LabeledEdit1.Text);
    if x<=0 then y:=x+1 else y:=sqr(x)+5;
    LabeledEdit2.Text:=FloatToStr(y);
  end
```

4.1.3. Создайте в новой папке проект; положите на него компоненты согласно рис. 74, после чего запрограммируйте кнопку «Расчёт» согласно листингу 4.2. Выполните компиляцию и отладку программы.

Составной оператор

В конструкциях if ... then и if ... then ... else в качестве действия после then и else допускается только один оператор. Однако может быть необходимо большее число операторов.

Например, в задаче 4.1 необходимо не только рассчитать y , но и сообщить пользователю, по какому именно варианту выполнен расчёт. Если при помощи одного однострочного компонента выводить и y , и то, какое выбрано условие, то это можно сделать за 1 действие (хотя и не обязательно): `LabeledEdit4.Text:=FloatToStr(y)+''`, условие 1', если условие 1 истинно, и `LabeledEdit4.Text:=FloatToStr(y)+''`, условие 2', если условие 1 ложно.

Изобразим блок-схему для такого случая на рис. 79. Если программа идёт по стрелке «ДА», то выполняется действие $y = x + 1$ и вывод с сообщением «условие 1», а если по стрелке «НЕТ», то действие $y = x^2 + 5$ и вывод с сообщением «условие 2». Значит, и после then, и после else необходимо по два оператора. Однако допускается только один оператор.

Выход есть: можно несколько (любое количество) простых операторов сгруппировать в составной оператор, который будет восприниматься программой как 1 оператор.

Составной оператор представляет собой группу из произвольного числа операторов, отделённых друг от друга точкой с запятой, и ограниченную операторными скобками begin и end.

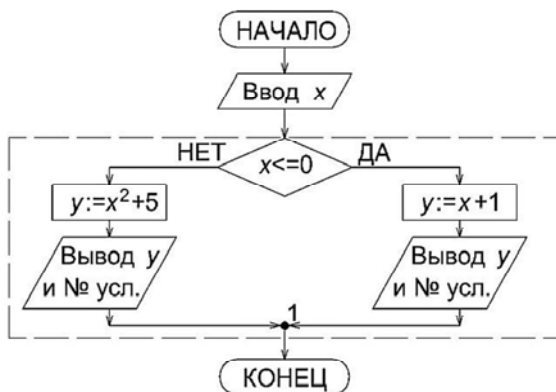


Рис. 79. Блок-схема для задачи 4.1 с использованием конструкции if ... then ... else и составных операторов

Составной оператор для действия после then запишем так:
begin y:=x+1; LabeledEdit1.Text:=FloatToStr(y)+'', условие 1' end.

Составной оператор для действия после else:

begin y:=x+5; LabeledEdit1.Text:=FloatToStr(y)+'', условие 2' end.

Вся программа приведена в листинге 4.3.

Обратите ещё раз внимание, что в операторе if перед else точка с запятой не ставится.

При большом количестве begin и end могут возникнуть сложности при подсчёте количества begin и end, а также ошибки при их расстановке. По этой причине рекомендуется использовать отступы при записи программного текста.

Отступы делаются при помощи пробелов либо перемещением курсора клавишей →. В листинге 4.3 минимальные отступы сделаны для заголовка процедуры и begin, и end, обозначающие начало и конец процедуры. На 2-м уровне (делаются 1–2 пробела от начала строки) идёт описание переменных и операторы. Всё, что находится между началом и концом оператора if, выполняется с ещё бóльшим отступом. Начало и конец составного оператора находится на одном уровне, а то, что между ними, выполнено с бóльшим отступом. Тогда понятно, что end, находящийся на одном уровне с begin составного оператора, закрывает составной оператор, а end, находящийся на одном уровне с begin процедуры, завершает раздел операторов процедуры.

Листинг 4.3. Программа с использованием конструкции if... then... else и составных операторов

```

procedure TForm1.BitBtn1Click(Sender: TObject);
  Var x, y: extended;
begin //начало процедуры
  x:=StrToFloat(LabeledEdit1.Text);
  if x<=0 then
    begin //начало составного оператора после then
      y:=x+1;
      LabeledEdit2.Text:=FloatToStr(y)+' , условие 1'
    end //конец составного оператора после then
  else
    begin //начало составного оператора после else
      y:=sqr(x)+5;
      LabeledEdit2.Text:=FloatToStr(y)+' , условие 2'
    end; //конец составного оператора после else
end; //конец процедуры

```

На рис. 80 приведена конструкция с использованием отступов в общем виде.

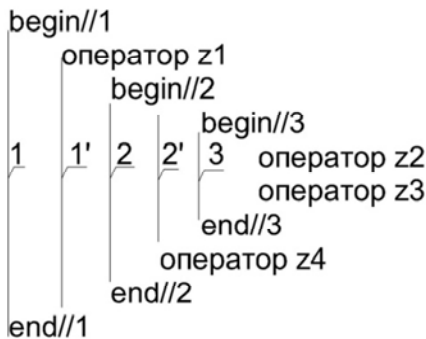


Рис. 80. Программа с использованием отступов

Begin//1 и end//1 – начало и конец раздела операторов процедуры. Они расположены на уровне 1. Операторы, которые расположены на уровне 1' (в данном случае – оператор z1), относятся к процедуре, но не относятся ни к одному из её составных операторов.

Begin//2 и end//2 – начало и конец составного оператора, входящего в процедуру, расположены на уровне 2. В его вложен ещё один оператор, начало которого begin//3 и конец end//3 расположены на уровне 3. Всё, что относится к вложенному составному оператору, ограниченному begin//3 и end//3 (оператор z2 и оператор z3), расположено правее уровня 3.

Оператор z4 относится к составному оператору, ограниченному операторными скобками begin//2 и end//2, но не относится к вложенному составному оператору, ограниченному begin//3 и end//3, поэтому он расположен на уровне 2' между уровнями 2 и 3.

```
begin//1
оператор z1
begin//2
begin//3
оператор z2
оператор z3
end//3
оператор z4
end//2
end//1
```

Точно ту же программу можно записать без отступов (рис. 81). Смысл программы при этом совершенно не изменится. Однако при записи программного текста, аналогичному, рис. 81, менее явно видны начало и конец процедуры, и начало и конец составных операторов и проще совершить ошибку при расстановке begin и end.

Не обязательно записывать каждый оператор на отдельной строке, как это сделано в листинге 4.3. Можно записать текст так же, как и на рис. 82. Каждый из составных операторов поместился на 1 строке, что делает программный текст понятным для чтения.

Рис. 81. Программа без использования отступов

```
procedure TForm1.BitBtn1Click(Sender: TObject);
Var x,y:extended;
begin
  x:=StrToFloat(LabeledEdit1.Text);
  if x<=0 then begin y:=x+1; LabeledEdit2.Text:=FloatToStr(y)+' , условие 1' end
  else begin y:=sqr(x)+5; LabeledEdit2.Text:=FloatToStr(y)+' , условие 2' end;
end;
```

Рис. 82. Программный текст с несколькими операторами в строке

4.1.4. Измените программный текст кнопки «Расчёт» согласно рис. 82 или листингу 4.3. Выполните компиляцию и отладку программы.

Для конструкции с использованием if ... then программа кнопки «Расчёт» приведена в листинге 4.4, блок-схема – на рис. 83.

Листинг 4.4. Программа с использованием конструкции if... then и составных операторов

```

procedure TForm1.BitBtn1Click(Sender: TObject);
  Var x, y: extended;
begin //начало процедуры
  x:=StrToFloat(LabeledEdit1.Text);
  if x<=0 then //начало 1-го оператора условия
    begin //начало составного оператора после then 1-го условия
      y:=x+1;
      LabeledEdit2.Text:=FloatToStr(y)+'', условие 1'
    end; (*конец составного оператора после then 2-го условия и
    конец 1-го оператора условия *)
  if x>0 then //начало 2-го оператора условия
    begin //начало составного оператора после then 2-го условия
      y:=sqr(x)+5;
      LabeledEdit2.Text:=FloatToStr(y)+'', условие 2'
    end; (*конец составного оператора после then 2-го условия и
    конец второго оператора условия*)
end; //конец процедуры
  
```

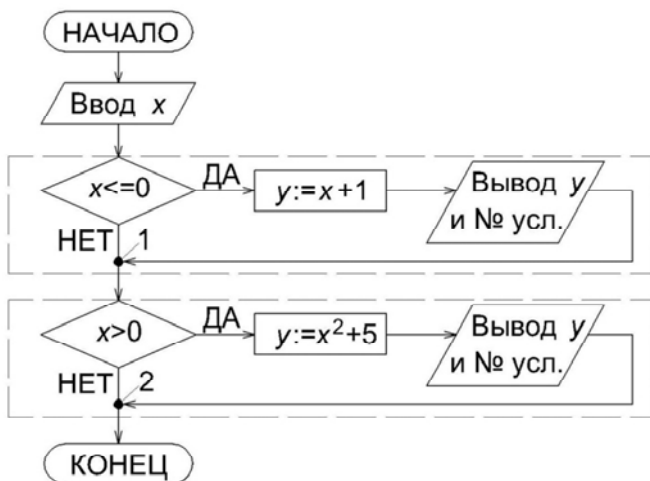


Рис. 83. Блок-схема процедуры кнопки «Расчёт» с использованием конструкции if ... then и составных операторов

Условие И (and)

Может возникнуть необходимость, чтобы какое-либо действие происходило при выполнении сразу двух или более условий; если хотя бы одно из данных условий не выполняется, то действие выполняться не должно. Тогда применяется условие И (and):

if (<условие 1>) and (<условие 2>) then.

Условие 1 и условия 2 берутся в скобки.

В общем случае:

if (<условие 1>) and (<условие 2>)and<...>and(<условие n>) then.

Всё условие И выполняется тогда и только тогда, если выполняются все его составные части. Если хотя бы одно из условий не выполняется, то всё условие также не выполняется.

При создании составных условий рекомендуется составлять таблицу истинности. В таблице указываются все возможные случаи истинности условий и истинность всего условия. Вместо «истина» может быть записано «1», а вместо «ложь» – «0».

Для случая if (<условие 1>) and (<условие 2>) будут возможны такие случаи:

1) выполняются и условие 1, и условие 2 – в этом случае всё условие истинно;

2) выполняется условие 1 и не выполняется условие 2 – в этом случае всё условие ложно, поскольку одна из его составных частей не выполняется, а для истинности условия И (не путать с другими условиями) должны выполняться *все* его составные части;

3) не выполняется условие 1 и выполняется условие 2 – в этом случае всё условие И будет ложным (см. п. 2);

4) не выполняется условие 1 и не выполняется условие 2 – всё условие И будет ложным (см. п. 2).

Таблица истинности условия И (and)

№ случая	Условие 1	Условие 2	Всё условие
1	Истинно (1)	Истинно (1)	Истинно (1)
2	Истинно (1)	Ложно (0)	Ложно (0)
3	Ложно (0)	Истинно (1)	Ложно (0)
4	Ложно (0)	Ложно (0)	Ложно (0)

Конструкции с составными условиями составляются так же, как и с несоставными условиями.

Если всё составное условие истинно, то программа идёт по стрелке «ДА», а если – ложно, то по стрелке «НЕТ». В случае 1 таблицы истинности программа пойдёт по стрелке «ДА», а в случаях 2, 3, 4 – по стрелке «НЕТ».

Задача 4.2.

Найти значение $y(x)$ при условии:

$$y = \begin{cases} x+1, & x \leq 0 & (1) \\ x, & 0 < x < 12 & (2) \\ x^2 + 5, & 12 \leq x < 50 & (3) \\ \ln(x), & x \geq 50 & (4) \end{cases}$$

Чтобы значение x находилось в интервале $(0; 12)$, оно должно быть одновременно больше 0 и меньше 12:

`if (x>0)and(x<12).`

Аналогично, чтобы значение x находилось в интервале $[12; 50)$, оно должно быть больше или равно 12 и меньше 50 в одно и то же время:

`if (x≥12)and(x<50).`

При использовании конструкции `if ... then` потребуются 4 оператора условия:

`if x<0 then y:=x+1;`

`if (x>0) and (x<12) then y:=x;`

`if (x>=12) and (x<50) then y:=sqr(x)+5;`

`if x>=50 then y:=ln(x);`

Форма может выглядеть аналогично рис. 74. Программный текст будет отличаться от листинга 4.1 только тем, что будет 4 оператора `if` вместо двух. Блок-схема приведена на рис. 84.

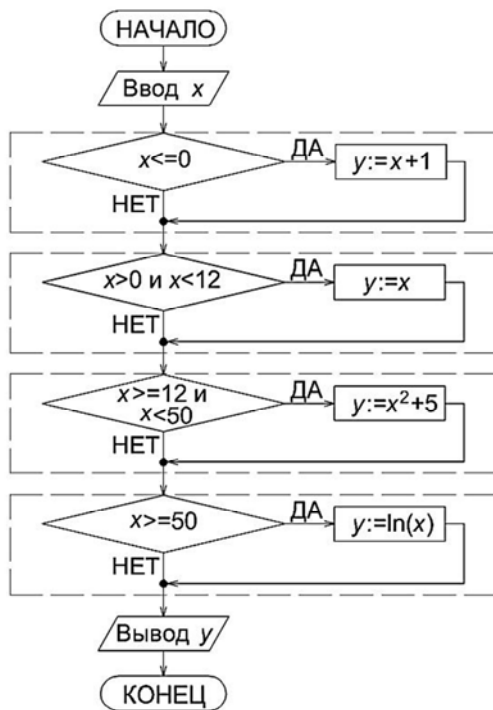


Рис. 84. Блок-схема программы с условием И при использовании конструкции if ... then

4.2.1. Создайте новый проект, сохраните в новую папку. Выполните программную реализацию задачи 4.2: форма приведена на рис. 74, блок-схема – на рис. 84.

Вложенные операторы if

Для конструкции if ... then... else в случае истинности первого условия выполняется действие после then, а если условие ложно – то после else.

В задаче 4.2 двух условий (одно проверяется, а другое – методом исключения после else) недостаточно. В таком случае после else можно записать второй оператор условия; а если и этого окажется недостаточно, то после else второго оператора условия можно записать третий оператор условия и т. д.

Если после else вместо действия будет ещё один оператор if, то это – конструкция с вложенными операторами условия:

```
if <условие 1> then <действие 1>
  else if <условие 2> then <действие 2>
    else <...>
      else if <условие n> then <действие n>
        else <действие n+1>;
```

Как только какое-либо из условий выполняется, то выполняется соответствующее ему действие, после чего программа сразу переходит к оператору, следующему за всей конструкцией, не проверяя при этом оставшиеся условия.

Если условие не выполняется, то программа проверяет следующее за ним и т. д.

Если ни одно из условий не выполняется, то выполняется действие n+1 при его наличии.

Конструкция при использовании составных операторов:

```
if <условие 1> then begin <оператор 1.1>; <оператор 1.2>; <...> end
  else if <условие 2> then begin <2.1>; <2.2>; <...> end
    else <...>
      else if <условие n> then begin <n.1>; <n.2> end
        else begin <(n+1).1>; <(n+1).2>; <...> end;
```

Для задачи 4.2 нужно проверить 4 условия.

При использовании конструкции if ... then ... else проверяется на одно условие меньше – оно проверяется методом исключения: нужно 3 оператора if.

Блок-схема представлена на рис. 85. Такой вариант представления будет достаточно удобным и компактным.

На рис. 86 представлена блок-схема несколько в ином виде: более явно выражены операторы if и стрелки «ДА» и «НЕТ», которые идут от них.

На рис. 87 блок-схема преобразована так, чтобы блоки «НАЧАЛО», «ВВОД X», «ВЫВОД Y» и «КОНЕЦ» находились друг под другом, а не смещались в сторону после конструкции if, как на рис. 86.

Все три блок-схемы несут в себе один и тот же алгоритм, несколько отличается лишь форма представления.

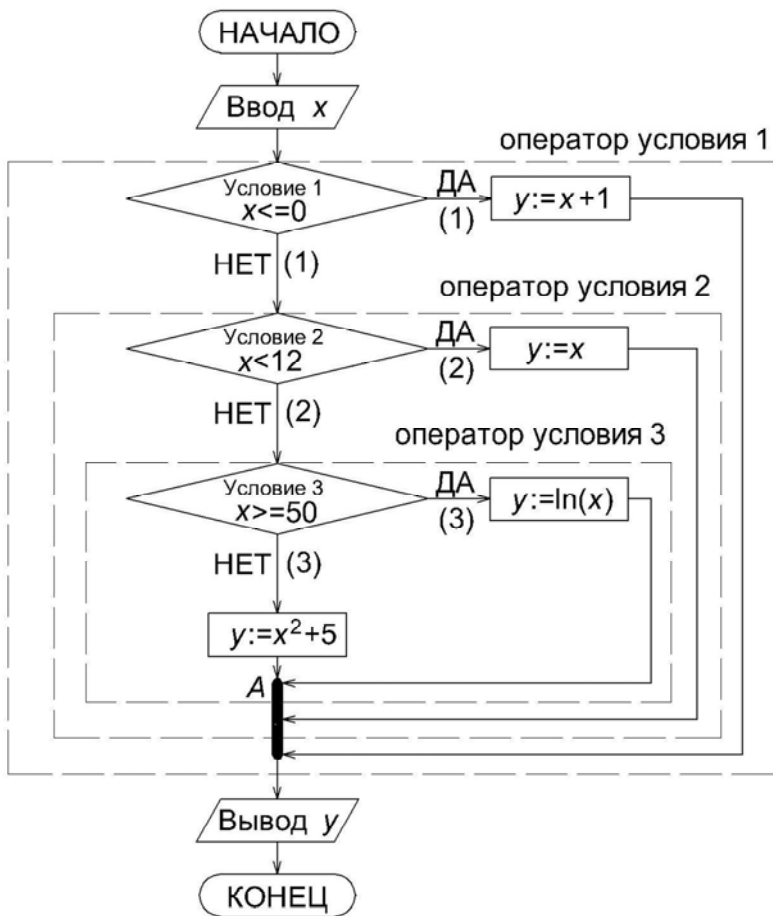


Рис. 85. Блок-схема конструкции с вложенными операторами if к задаче 4.2

Смотрим вначале на блок-схему (рис. 86), в которой наиболее явно выражены операторы условия и стрелки «ДА» и «НЕТ», идущие от них. При работе с блок-схемой всегда **внимательно** следите за тем, куда идут стрелки.

1. Выполняется ввод x .

2. Проверяется условие $x \leq 0$.

2.1. Если условие $x \leq 0$ выполняется (x меньше или равен 0), то программа идёт по стрелке «ДА(1)». Что на стрелке «ДА(1)»

находится? Там находится действие $y:=x+1$. Значит, это действие выполняется. Далее идёт конец оператора: жирная черта A , в которую сходятся все стрелки от него.

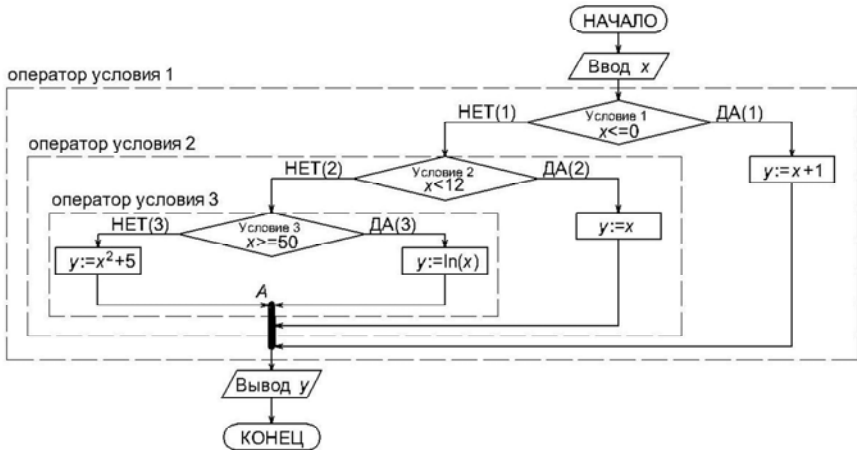


Рис. 86. Вариант представления блок-схемы с вложенными операторами if к задаче 4.2, при котором операторы явно видны

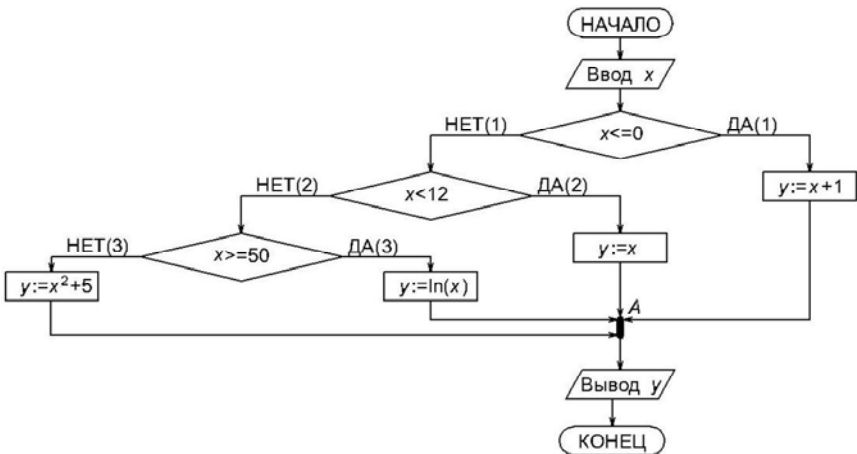


Рис. 87. Вариант представления блок-схемы с вложенными операторами if

2.2. Если условие $x \leq 0$ не выполняется (x оказался больше 0), то программа идёт по стрелке «НЕТ(1)». Идём по стрелке «НЕТ(1)» от её начала. Видим, что первое, что встретим – это второе условие.

Значит, если первое условие ($x \leq 0$) не выполняется, то программа идёт по стрелке «НЕТ(1)» и приходит ко второму условию, а если – выполняется, то идёт по стрелке «ДА(1)» и не приходит к нему.

Согласно блок-схеме конструкции if... then (рис. 84) второе условие должно быть ЕСЛИ ($x > 0$) И ($x < 12$). Здесь его тоже можно записать в таком виде. Однако в случае $x \leq 0$ было бы истинным условие 1 и программа пошла бы по стрелке «ДА(1)», выполнила бы действие $y: = x + 1$ и пошла бы в конец оператора (точка А), не проверяя условие 2 вообще. А если программа пошла по стрелке «НЕТ(1)», то значит $x > 0$, поэтому в условии ЕСЛИ ($x > 0$) И ($x < 12$) можно опустить ($x > 0$), записав просто ЕСЛИ $x < 12$.

В итоге если первое условие $x \leq 0$ ложно, то проверяется второе условие $x < 12$.

2.2.1. Если второе условие истинно, то программа идёт по стрелке ДА(2). Там находится действие $y: = x$, после чего стрелка идёт к жирной линии А, где оператор условия заканчивается. Значит, если второе условие истинно, то выполняется действие $y: = x$, после чего программа идёт к следующему оператору.

2.2.2. Если второе условие ложно, то программа идёт по стрелке НЕТ(2). Первое, что идёт по стрелке 2 за условием 1 – это третье условие. Значит, если второе условие ложно, то программа проверяет третье условие $x \geq 50$.

2.2.2.1. Если третье условие $x \geq 50$ истинно, то программа идёт по стрелке «ДА(3)»: выполняет действие $y: = \ln(x)$ и переходит к следующему оператору.

2.2.2.2. Если третье условие ложно, то программа идёт по стрелке «НЕТ(3)»: выполняет действие $y: = x^2 + 5$ и переходит к следующему оператору.

3. Выполняется вывод y следующий оператор после конструкции с вложенными операторами условия.

Условие попадания x в интервал (12;50] определяется методом исключения после последнего else: если x не попадает в остальные 3 интервала, то x попадает в этот интервал.

Сведём возможные случаи работы программы в табл. 4.1. В листинге 4.5 приведём программную реализацию кнопки «Расчёт» за-

дачи 4.2. согласно блок-схеме (рис. 85, 86, 87) с использованием вложенных операторов условия. Возьмём по 1 значению x из каждого интервала случайным образом.

Таблица 4.1

Работа программы с использованием вложенных операторов if в зависимости от значения введённой переменной

Значение x	Путь
-16	Ввод $x \rightarrow$ Условие 1 ($x \leq 0$) \rightarrow Да(1) $\rightarrow y: = x + 1 \rightarrow$ Вывод y
10	Ввод $x \rightarrow$ Условие 1 ($x \leq 0$) \rightarrow Нет(1) \rightarrow Условие 2 ($x < 12$) \rightarrow Да(2) $\rightarrow y: = x \rightarrow$ Вывод y
154	Ввод $x \rightarrow$ Условие 1 ($x \leq 0$) \rightarrow Нет(1) \rightarrow Условие 2 ($x < 12$) \rightarrow Нет(2) \rightarrow Условие 3 ($x > 50$) \rightarrow Да(3) $\rightarrow y: = \ln(x) \rightarrow$ Вывод y
25	Ввод $x \rightarrow$ Условие 1 ($x \leq 0$) \rightarrow Нет(1) \rightarrow Условие 2 ($x < 12$) \rightarrow Нет(2) \rightarrow Условие 3 ($x > 50$) \rightarrow Нет(3) $\rightarrow y: = x^2 + 5 \rightarrow$ Вывод y

Листинг 4.5. Программа кнопки «Расчёт» с использованием конструкций с вложенными операторами if

```

procedure TForm1.BitBtn1Click(Sender: TObject);
  Var x, y: extended;
  begin
    x:=StrToFloat(LabeledEdit1.Text);
    if x<0 then y:=x+1
      else if x<12 then y:=x
        else if x>50 then y:=ln(x)
          else y:=sqr(x)+5;
    LabeledEdit2.Text:=FloatToStr(y)
  end;

```

4.2.2. Положите на форму (рис. 74) третью кнопку BitBtn и дайте ей название «Расчёт 2», после чего запрограммируйте согласно листингу 4.5.

Для сравнения составим таблицу 4.2, в которой покажем, как будет работать программа при использовании конструкции if... then (блок-схема на рис. 84).

Таблица 4.2

Работа программы в зависимости от значения введённой переменной; использованы независимые операторы if, конструкция if ... then

Значение x	Путь
-16	Ввод $x \rightarrow$ Условие $(x \leq 0) \rightarrow$ Да $\rightarrow y: = x + 1 \rightarrow$ Условие $(x > 0) \text{ И } (x < 12) \rightarrow$ Нет \rightarrow Условие $(x \geq 12) \text{ И } (x < 50)$ \rightarrow Нет \rightarrow Условие $(x \geq 50) \rightarrow$ Нет \rightarrow Вывод y
10	Ввод $x \rightarrow$ Условие $(x \leq 0) \rightarrow$ Нет \rightarrow Условие $(x > 0) \text{ И } (x < 12) \rightarrow$ Да $\rightarrow y: = x \rightarrow$ Условие $(x \geq 12) \text{ И } (x < 50) \rightarrow$ Нет \rightarrow Условие $(x \geq 50) \rightarrow$ Нет \rightarrow Вывод y
154	Ввод $x \rightarrow$ Условие $(x \leq 0) \rightarrow$ Нет \rightarrow Условие $(x > 0) \text{ И } (x < 12) \rightarrow$ Нет \rightarrow Условие $(x \geq 12) \text{ И } (x < 50)$ \rightarrow Нет \rightarrow Условие $(x \geq 50) \rightarrow$ Да $\rightarrow y: = \ln(x) \rightarrow$ Вывод y
25	Ввод $x \rightarrow$ Условие $(x \leq 0) \rightarrow$ Нет \rightarrow Условие $(x > 0) \text{ И } (x < 12) \rightarrow$ Нет \rightarrow Условие $(x \geq 12) \text{ И } (x < 50)$ \rightarrow Да $\rightarrow y: = x^2 + 5 \rightarrow$ Условие $(x \geq 50) \rightarrow$ Нет \rightarrow Вывод y

Как видно, в конструкции с четырьмя независимыми операторами if (табл. 4.2) все условия проверяются в любом случае, в то время как при использовании конструкции с вложенными операторами if (табл. 4.1) условия проверяются до тех пор, пока какое-либо из них не выполнится.

Условие ИЛИ (or)

Если для выполнения действия достаточно истинности только одного условия, то используется конструкция:

```
if (<условие 1>)or(<условие 2>)
```

(для двух условий);

```
if (<условие 1>)or(<условие 2>)or...or(<условие n>)
```

(для n условий).

Действие будет выполняться в том случае, если будет истинным хотя бы одно из условий, и не будет выполняться только в том случае, если *все* условия окажутся ложными. Таблица истинности для условия ИЛИ для двух условий будет выглядеть так:

Таблица истинности для условия ИЛИ

№ случая	Условие 1	Условие 2	Всё условие ИЛИ
1	Истинно (1)	Истинно (1)	Истинно (1)
2	Истинно (1)	Ложно (0)	Истинно (1)
3	Ложно (0)	Истинно (1)	Истинно (1)
4	Ложно (0)	Ложно (0)	Ложно (0)

Задача 4.3.

Для задачи к части 3 (форма приведена на рис. 61, программный текст кнопки «Расчёт» – на рис. 66) необходимо выполнить проверку: не оставил ли пользователь хотя бы один из блоков LabeledEdit для ввода исходных данных пустым, и если да, то выслать сообщение.

Решение. Сразу после begin записываем:

```
if (Lenght(LabeledEdit1.Text)=0) or (Lenght(LabeledEdit2.Text)=0)  
then begin ShowMessage("Не все данные введены!"); exit end;
```

Lenght(LabeledEdit1.Text)=0 – это значит, что в блоке редактора нет ни одного символа.

ShowMessage – функция, позволяющая выслать сообщение в виде диалогового окна с именем проекта и кнопкой ОК, при помощи которой можно его закрыть.

Exit – завершение работы процедуры-обработчика (в данном случае – BitBtn1Click). В том месте, где встречается Exit, процедура выполняется прекращает.

Исключающее ИЛИ (XOR)

if (<условие 1>)xor(<условие 2>)

При использовании двух условий будет истинным тогда, когда одно из условий истинно. Когда истинны или ложны оба условия, всё условие XOR будет ложным.

Если использовать 3, 4, 5 и более условий, то всё условие будет истинным, если истинно нечётное число условий одновременно, и ложным, если истинно чётное число условий или если все условия ложны.

Ниже приведена таблица истинности для условия XOR.

Таблица истинности для исключающего ИЛИ (XOR)

№ случая	Условие 1	Условие 2	Всё условие ИЛИ
1	Истинно (1)	Истинно (1)	Ложно (0)
2	Истинно (1)	Ложно (0)	Истинно (1)
3	Ложно (0)	Истинно (1)	Истинно (1)
4	Ложно (0)	Ложно (0)	Ложно (0)

Условие НЕ

If not <условие> then <действие>

Если условие ложно, то not условие истинно и действие выполняется; если условие истинно, то not условие ложно и действие не выполняется.

If not <условие> then <действие 1> else <действие 1'>

если условие ложно, то выполняется действие 1, если – истинно, то действие 1'.

Таблица истинности условия if not

№ случая	Условие	Условие НЕ
1	Истинно (1)	Ложно (0)
2	Ложно (0)	Истинно (1)

Условие И НЕ

If not((условие 1)and(условие 2))

Когда условие И истинно, тогда ложно условие И НЕ, а когда условие И ложно, тогда истинно условие И НЕ.

Таблица истинности условия И НЕ

№ случая	Условие 1	Условие 2	Условие И	Условие И НЕ
1	Истинно (1)	Истинно (1)	<u>Истинно (1)</u>	Ложно (0)
2	Истинно (1)	Ложно (0)	<u>Ложно (0)</u>	Истинно (1)
3	Ложно (0)	Истинно (1)	<u>Ложно (0)</u>	Истинно (1)
4	Ложно (0)	Ложно (0)	<u>Ложно (0)</u>	Истинно (1)

Если оба условия истинны, то условие И НЕ будет ложным. Если оба условия будут ложным или хотя бы одно условие будет истинным, то всё условие И НЕ будет истинным

Условие ИЛИ НЕ

If not((условие 1)or(условие 2))

Когда условие ИЛИ истинно, тогда ложно условие ИЛИ НЕ, а когда условие ИЛИ ложно, истинно условие ИЛИ НЕ.

Если оба условия ложны, то истинно условие ИЛИ НЕ. Во всех остальных случаях ложно условие ИЛИ НЕ.

Таблица истинности условия ИЛИ НЕ

№ случая	Условие 1	Условие 2	Условие ИЛИ	Условие ИЛИ НЕ
1	Истинно (1)	Истинно (1)	<u>Истинно (1)</u>	Ложно (0)
2	Истинно (1)	Ложно (0)	<u>Истинно (1)</u>	Ложно (0)
3	Ложно (0)	Истинно (1)	<u>Истинно (1)</u>	Ложно (0)
4	Ложно (0)	Ложно (0)	<u>Ложно (0)</u>	Истинно (1)

Флажок CheckBox

Флажок CheckBox (вкладка Standard) является независимой кнопкой выбора. Включение одного флажка не изменяет состояние остальных флажков, в то время как включение RadioButton (зависимой кнопки выбора) меняет состояние остальных кнопок RadioButton, расположенных вместе с ней на одном материнском компоненте (Panel, GroupBox, а при их отсутствии – на форме). Свойство Checked принимает значение true, если флажок включён, и false, если флажок выключен.

1. Создайте проект. Положите на форму 6 флажков CheckBox. Измените свойство Checked для двух из них (например, для 3-го и 6-го) с false на true (рис. 88).

2. Запустите программу. Включите все флажки. Выключите все флажки. Закройте режим отладки.

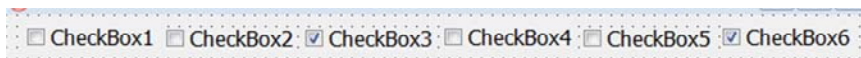


Рис. 88. Форма с флажками CheckBox

Можно сделать так, чтобы флажок мог находиться в трёх положениях: включённом, наполовину включённом и выключенном. Для этого следует поставить свойство AllowGrayed в положение true, а вместо Checked изменять свойство State: cbGrayed – среднее положение, cbChecked – включён, cbUnchecked – выключен. Свойство Checked будет равно true в том случае, если свойство State примет значение cbChecked; в двух других случаях Checked будет равно false.

3. Оставьте на форме 1 флажок. Установите свойство AllowGrayed в положение true.

4. Положите компонент Label (вкладка Standard). Сделайте двойной щелчок по CheckBox и запрограммируйте процедуру TForm1.CheckBox1Click так:

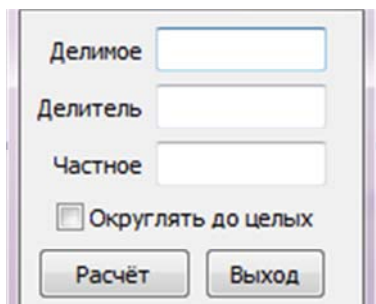
```
if CheckBox1.State=cbChecked then Label1.Caption:='Включено';  
if CheckBox1.State=cbGrayed then Label1.Caption:='Затемнено';  
if CheckBox1.State=cbUnchecked then Label1.Caption:='Выключено';
```

5. Выполните компиляцию программы, после чего запустите её. Щёлкните несколько раз по флажку, чтобы проверить, что надпись изменяется. Закройте режим отладки.

Задача 4.4.

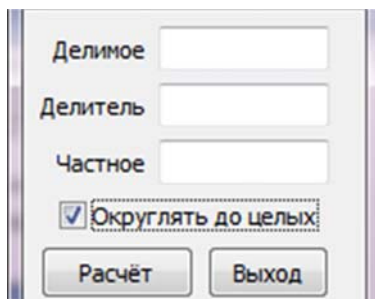
Выполнить деление. При этом обеспечить возможность выводить результат округляя до целого числа и не округляя.

Решение. Положим на форму три LabeledEdit и кнопки «Расчёт» и «Выход» (рис. 89, 90).



The screenshot shows a GUI form with three input fields labeled 'Делимое' (Dividend), 'Делитель' (Divisor), and 'Частное' (Quotient). Below the fields is a checkbox labeled 'Округлять до целых' (Round to integers), which is currently unchecked. At the bottom are two buttons: 'Расчёт' (Calculate) and 'Выход' (Exit).

Рис. 89. Форма к задаче 4.4, флажок выключен



The screenshot shows the same GUI form as in Figure 89, but the checkbox 'Округлять до целых' is now checked. The 'Расчёт' and 'Выход' buttons are also visible.

Рис. 90. Форма к задаче 4.4, флажок включён

Положим флажок «Округлять до целых»; когда он включён (рис. 90), округление должно будет выполняться, а когда выключен (рис. 89), то нет.

Для округления до целых можно использовать функцию Round (<Число>).

Блок-схему алгоритма процедуры OnClick кнопки «Расчёт» приведём на рис. 91. Ввод данных и нахождение результата выполняется одинаково для обоих случаев. Затем если флажок на момент нажатия пользователем кнопки «Расчёт» включён, необходимо также выполнить округление. Когда флажок включён, его свойство Checked будет принимать значение true. Условие «Если флажок включён, то округлить» запишется в виде:

```
if CheckBox1.Checked=true then  
c:=Round(c)
```

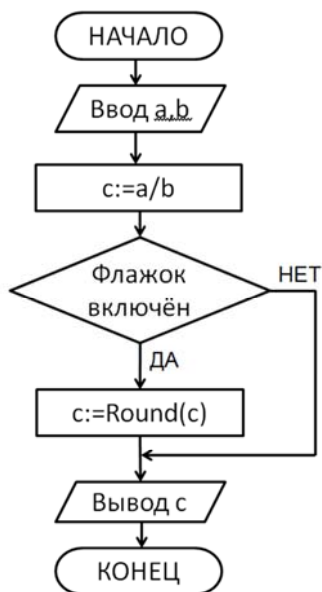


Рис. 91. Блок-схема программы к задаче 4.4

В условии для случая, когда что-то должно быть равно true, можно опускать " =true". В данном случае можно просто записать `if CheckBox1.Checked then c:=Round(c)`

Программный текст процедуры кнопки «Расчёт» приведён ниже.

```
procedure TForm1.BitBtn1Click(Sender: TObject);
  Var a,b,c:extended;
begin
  a:=StrToFloat(LabeledEdit1.Text);
  b:=StrToFloat(LabeledEdit2.Text);
  c:=a/b;
  if CheckBox1.Checked then c:=Round(c);
  LabeledEdit4.Text:=FloatToStr(c);
End;
```

Приоритет операций

Все вычисления выполняются слева направо в соответствии с приоритетом операций. В таблице 4.3 приведены операции в соответствии с приоритетом. Первыми выполняются операции, имеющие приоритет 1, вторые – приоритет 2 и т. д.

Скобки изменяют порядок выполнения операций.

Если аргумент функции – выражение, то вначале определяется значение этого выражения. Например, если записано $y = \sin(x + 8)$, то вначале выполнится выражение $x + 8$, а затем будет вычислен синус.

Таблица 4.3

Приоритет операций

Приоритет	Что выполняется
1	Функции
2	Not (не)
3	* (умножение) / (деление) div (целочисленное деление) mod (остаток при целочисленном делении) and (и)

Приоритет	Что выполняется
4	+ (сложение) - (вычитание) or (или)
5	= (равно) <> (не равно) <= (меньше или равно) >= (больше или равно)

Сложные составные условия

Составное условие может содержать много условий, и в нём могут встречаться одновременно условия И, И НЕ, ИЛИ, ИЛИ НЕ, НЕ и т. д. При составлении сложно составного условия необходимо помнить о приоритете операций (см. табл. 4.3). Например, and имеет приоритет перед or, поэтому условие

If (<условие 1>)or(<условие 2>)and(<условие 3>)or(условие 4) (a)
эквивалентно условию;

If (<условие 1>)or((<условие 2>)and(<условие 3>))or(условие 4) (b)
и не эквивалентно условию

If ((<условие 1>)or(<условие 2>))and((<условие 3>)or(условие 4)) (c).

При использовании and и or на одном уровне для большей ясности можно записывать условие (a) в варианте (b), ставя дополнительные скобки.

При использовании сложных условий с not следует помнить о скобках:

If not((условие 1)and(условие 2)) then
(если условия 1 и 2 не выполняются вместе, то...);

If not(условие 1)and(условие 2) then
(если условие 1 не выполняется и условие 2 выполняется, то...).

Для проверки истинности условий используем флажки CheckBox. Если всё условие будет истинным, программа должна будет выслать соответствующее сообщение. Для этого после then используем ShowMessage. В качестве составляющих частей условий берём состояние флажков. Первому условию будет соответствовать

CheckBox1: если он включён, то условие 1 истинно, а если выключен, – то ложно. Аналогично с остальными условиями: второму условию будет соответствовать CheckBox2, третьему – CheckBox3 и т. д. Итак, включая или выключая поочерёдно флажки, можно будет проверить правильность составления таблицы истинности.

Задача 4.5.

Составить таблицу истинности для условия

If ((<условие 1>)or(<условие 2>))and(<условие 3>)

Смотрим по скобкам: условия 1 и 2 объединены дополнительными скобками, значит – надо определить, когда выполняется условие or, а затем условие and, уже зная, когда выполняется условие or:

If (<условие 1, 2>)and (<условие 3>), (*)

где условие 1,2 – это (<условие 1>)or(<условие 2>). (**)

Составляем таблицу истинности.

Случай 1. Все условия истинны.

1.1. Ставим 1 в столбцы «Условие 1», «Условие 2» и «Условие 3».

1.2. Определяем истинность условия 1,2 (**). Оно будет истинным, если будет истинным хотя бы одно из условий или оба сразу (условие ИЛИ). Условия 1 и 2 истинны, поэтому условие 1,2 истинно. Ставим 1 столбец «Условие 1,2».

1.3. Определяем истинность всего условия (*). Поскольку это условие И, необходимо, чтобы все составные его часть были истинны, то есть чтобы и условие 1,2 (**), и условие 3 были истинны. Они оба истинны (и в столбце 4, и в столбце 5 таблицы цифра 1). Значит, условие (*) будет истинным. Ставим 1 в столбец 6.

Случай 2. Первое условие ложно, второе и третье – истинны.

Здесь и далее те же шаги, что и в случае 1.

2.1. Ставим 1 в столбцы «Условие 2» и «Условие 3» и 0 в столбец "Условие 1».

2.2. Определяем истинность условия 1,2. Условие 2 истинно, поэтому всё условие 1,2 (**) истинно (условие ИЛИ: будет ложно только в том случае, если ложны все составные части). Ставим 1 в столбец 4 «Условие 1,2».

2.3. Условие (*) истинно, если истинны составляющие части (условие 1,2 и условие 3). Смотрим соответствующие столбцы условия: там везде 1. Значит, всё условие (*) в случае 2 истинно. Ставим 1 в столбце 6 для случая 2.

Случай 3. Второе условие ложно, первое и третье истинны.

3.1. Ставим 1 в столбцы «Условие 1» и «Условие 3» и 0 в столбец «Условие 2».

3.2. Определяем истинность условия 1, 2. Поскольку одна из составляющих его частей (условие 2) истинна, оно истинно (см. п. 2.2. и 1.2). Ставим 1 в столбец 4.

3.3. Определяем истинность всего условия (*). Обе части условия (*) истинны (1 в столбце 4 для условия (**)) и 1 в столбце 5 для условия 3), значит, оно истинно (см. п. 2.3 и 1.3). Ставим 1 в столбце 6.

Случай 4. Третье условие ложно, первое и второе истинны.

4.1. Ставим 1 в столбец 2 и 3 («Условие 1» и «Условие 2») и 0 – в столбец 5 («Условие 3»).

4.2. Определяем истинность условия 1,2. Оно истинно (см. п. 2.2. и 1.2). Ставим 1 в столбец 4.

4.3. Определяем истинность всего условия (*). Поскольку это условие И, обе его части должны быть истинны, а значит, в столбцах 4 и 5 должны быть 1. Однако в столбце 5 будет 0, поскольку ложно условие 3. Значит, всё условие И (*) будет ложным, поскольку одна из его частей (условие 3) ложна. Ставим 0 в табл. 6.

Случай 5. Ложны первые 2 условия, условие 3 истинно.

5.1. Ставим 1 в столбец 5 и 0 – в столбцы 2 и 3.

5.2. Определяем истинность условия 1,2 (**). Если все составные части условия ИЛИ не выполняются, то всё условие ИЛИ не выполняется. Условие 1 и 2 не выполняются; значит, условие 1,2 (**) не выполняется. Ставим 0 в столбец 4.

5.3. Определяем истинность всего условия (*). Для условия И необходимо, чтобы все его составные части выполнялись. Значит, должно быть 1 и в столбце 4, и в столбце 5. Но в столбце 4 будет 0. Значит, всё условие (*) не выполняется, в столбец 6 пишем 0.

Случай 6. Условие 1 истинно, условия 2 и 3 ложны.

6.1. Пишем 1 в столбец 2 и 0 – в столбцы 3 и 6.

6.2. Проверяем условие 1,2 (**). Оно истинно, поскольку в столбце 2 единица, что для условия ИЛИ достаточно. Ставим 1 в столбец 4.

6.3. Проверяем всё условие (*), глядя на цифры в столбцах 4 и 5. Оно ложно, поскольку в столбце 5 ноль, а для условия И должны быть обе единицы. Ставим 0 в столбец 6.

**Таблица истинности для составного условия
((<условие 1>)or(<условие 2>))and(<условие 3>)**

№ случая	Усл. 1	Усл. 2	Условие 1,2 (**) (усл 1) or (усл.2)	Усл. 3	Всё условие (*)
1	2	3	4	5	6
1	1	1	1	1	1
2	0	1	1	1	1
3	1	0	1	1	1
4	1	1	1	0	0
5	0	0	0	1	0
6	1	0	1	0	0
7	0	1	1	0	0
8	0	0	0	0	0

Случай 7. Условие 2 истинно, условия 1 и 3 ложны.

7.1. Пишем 1 в столбец 3 и 0 – в столбцы 2 и 5.

7.2. Проверяем условие 1, 2 (**). Оно истинно, поскольку в столбце 3 один, что для условия ИЛИ достаточно. Ставим 1 в столбец 4.

7.3. Проверяем всё условие (*). Там всё аналогично п. 6.3. Ставим 0 в столбец 6.

Случай 8. Все условия ложны.

8.1. Ставим нули в столбцы 2, 3, 5.

8.2. Проверяем условие 1, 2 (**). Если оба условия ложны, то ложно и всё условие И. В столбцах 2 и 3 нули, значит – всё условие 1,2 ложно. Ставим 0 в столбец 4.

8.3. Проверяем всё условие (*), глядя на цифры в столбцах 4 и 5. Там нули, значит условие И (*) тоже ложно. Ставим 0 в столбец 6. Таблица истинности готова.

Кладём три флажка на форму. Для первого Caption меняем на «Условие 1», для 2-го – на «Условие 2», для 3-го – на «Условие 3». Кладём кнопку BitBtn, свойству Caption которой присваиваем условие (рис. 92). И программируем её следующим образом:

```

procedure TForm1.BitBtn1Click(Sender: TObject);
begin
    if ((CheckBox1.Checked) or (CheckBox2.Checked)
        and (CheckBox3.Checked)
            then ShowMessage ('Условие истинно')
end;
```

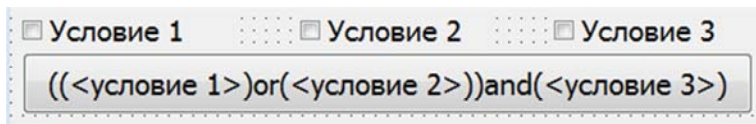


Рис. 92. Форма для проверки истинности условия

Всё условие в программу записывается так же, как и оно записано в задаче. Условием 1 будет состояние первого флажка (включён – истинно, выключен – ложно), условием 2 – состояние 2-го флажка, а условием 3 – 3-го.

Теперь проверяем все 8 возможных случаев. Если в столбце «Условие 1» для i -го случая будет цифра 1 (то есть условие 1 истинно), то флажок «Условие 1» нужно включить, а если 0, то флажок должен быть выключен. Аналогично если условие 2 истинно (в столбце «Условие 2» цифра 1), то флажок «Условие 2» должен быть включён, а если условие 2 ложно (в столбце «Условие 2» цифра 0), то и флажок «Условие 2» должен быть выключен. То же и для условия 3 (табл. 4.4).

Таблица 4.4

Включение и выключение флажков в соответствии с истинностью условий

№ случая	Усл. 1	Флажок 1	Усл. 2	Флажок 2	Усл.3	Флажок 3
1	1	вкл.	1	вкл.	1	вкл.
2	0	выкл.	1	вкл.	1	вкл.
3	1	вкл.	0	выкл.	1	вкл.
4	1	вкл.	1	вкл.	0	выкл.
5	0	выкл.	0	выкл.	1	вкл.
6	1	вкл.	0	выкл.	0	выкл.
7	0	выкл.	1	вкл.	0	выкл.
8	0	выкл.	0	выкл.	0	выкл.

Запустим программу (F9). Для 1-го случая включим все три флажка согласно табл. 4.4. Затем нажмём кнопку BitBtn1. Если появится сообщение «Условие истинно» (рис. 93), то значит, всё условие для случая 1 будет истинным, а если не появится, то лож-

ным. Сообщение появилось. Значит, всё условие будет истинным. Сверяем с таблицей истинности. Да, в таблице истинности всё условие тоже выполняется для случая 1. Значит, для случая 1 определили истинность всего условия верно. Закрываем сообщение кнопкой ОК.

Для 2-го случая согласно табл. 4.4 включаем 2-й и 3-й флажок и выключаем 1-й, после чего жмём кнопку. Появится сообщение «Условие истинно». Смотрим столбец 6 таблицы истинности. Да, там 1. Значит, для случая 2 также определили истинность условия верно. Закрываем сообщение кнопкой ОК.

Аналогично выполняем действия для всех оставшихся случаев. Если правильно определили истинность условия и правильно записали условие в программе, то для случаев 4, 5, 6, 7, 8 сообщение не появится, поскольку для случаев 4, 5, 6, 7, 8 всё условие будет ложным и в столбце 6 таблицы истинности будет 0.



Рис. 93. Сообщение об истинности условия.

При желании можно сделать так, чтобы если условие ложно, то тоже выдавалось бы соответствующее сообщение:

```
if ((CheckBox1.Checked) or (CheckBox2.Checked)
    and (CheckBox3.Checked)
    then ShowMessage ('Условие истинно')
    else ShowMessage ('Условие ложно')
```

Задача 4.6.

Составить таблицу истинности для условия

If (<условие 1>or(<условие 2>)and(<условие 3>))

Две части (условие 2 и условие 3) образуют вместе условие and.

Поэтому условие запишем в виде:

if (<условие 1>)or(<условие 2, 3>), (*)

где условие (2, 3):

if (<условие 2>)and(<условие 3>). (**)

Вначале определяем истинность условия 2, 3 (**): условие И, будет истинным в том случае, если истинны и условие 2, и условие 3. Во всех остальных случаях оно будет ложным.

Затем определяем истинность всего условия (*). Оно будет ложным только в том случае, если будут ложны и условие 1, и условие 2, 3 (**).

Составляем таблицу истинности. Расписываем все возможные комбинации истинности условий. Их будет 8, и они будут такими же, как и для предыдущего условия. Теперь необходимо для каждого из 8 случаев вначале определить истинность условия 2,3 (**), а затем – и всего условия (*). Сделайте это самостоятельно.

4.6.1. Доделайте таблицу истинности для условия задачи 4.6.

4.6.2. Создайте и сохраните проект. Положите 3 флажка CheckBox. Положите кнопку BitBtn (см. рис. 92). Дайте название кнопке (<условие 1>or((<условие 2>)and(<условие 3>)). Запрограммируйте кнопку так:

```
if (CheckBox1.Checked) or
  ((CheckBox2.Checked) and (CheckBox3.Checked))
  then ShowMessage ('Условие истинно')
```

4.6.3. Проверьте правильность составления таблицы истинности, включая поочередно флажки согласно таблице истинности или табл. 4.4.

**Заготовка для таблицы истинности для условия к задаче 4.6:
If (<условие 1>or((<условие 2>)and(<условие 3>))**

№ случая	Усл. 1	Усл. 2	Усл. 3	Условие 2,3 (**) (усл 2) and (усл.3)	Всё условие (*)
1	2	3	5	4	6
1	1	1	1		
2	0	1	1		
3	1	0	1		
4	1	1	0		
5	0	0	1		
6	1	0	0		
7	0	1	0		
8	0	0	0		

Задача 4.7.

Рассмотрим более сложное условие

if <усл. 1>and<усл. 2>and not((<усл. 3>)and(<усл. 4>)).

На одном уровне находятся условия 1 2 и условие 3,4 (**)

not((<усл. 3>)and(<усл. 4>)). (**)

Для определения истинности условия 3,4(**) с not можно вначале определить истинность условия 3,4'(***) без not

((<усл. 3>)and(<усл. 4>)). (***)

Записываем всё условие в виде

if <усл. 1>and<усл. 2>and (<усл. 3, 4>)). (*)

В таблицу истинности записываем все возможные комбинации истинности каждого из четырёх условий (условия 1, условия 2, условия 3 и условия 4). Их будет 16. Значит, надо рассмотреть 16 случаев.

Условие (***) истинно только в том случае, если истинны все условия, входящие в него, поскольку это условие И. Во всех остальных случаях оно ложно.

Условие (**) будет истинно тогда, когда ложно условие (***), и ложно тогда, когда истинно условие (***), поскольку это условие то же, что и (***), но с НЕ перед ним.

Условие (*) будет истинным тогда, когда будут истинны все условия, входящие в состав него (условие 1, условие 2, условие 3, 4).

Случай 1. Все условия истинны.

1.1. Проверяем истинность условия 3,4' (***) . И условие 3, и условие 4 истинны; значит, оно истинно. Ставим 1 в столбец 6.

1.2. Проверяем истинность условия 3,4 (**). Условие 3,4' (***) истинно (цифра 1 в столбце 6 согласно п. 1.1); значит, условие 3, 4 ложно. Пишем 0 в столбец 7.

1.3. Проверяем истинность всего условия (*). Чтобы всё условие было истинно, все его составные части должны быть истинны: в столбцах 2,3 и 7 должны быть единицы. В столбце 7 ноль (см. п. 1.2). Значит, пишем 0 в столбец 8.

Случай 2. Ложно условие 4, а условие 1, условие 2 и условие 3 истинны.

2.1. Условие 4 ложно, а для истинности условия 3,4' (условие И, ***) должны быть истинны и условие 3, и условие 4. Пишем 0 в столбец 6.

2.2. Условие 3,4' (***) ложно; значит, условие 3,4 (**) истинно. Ставим 1 в столбец 7.

2.3. В столбцах 2,3 и 7 стоят единицы; значит, всё условие (*) будет истинным.

Для случаев 3–16 определите истинность условия (*) и его составляющих (***) и (**) самостоятельно по такому же алгоритму.

4.7.1. Перепишите таблицу истинности условия к задаче 4.7 и доделайте её.

4.7.2. Добавьте к форме (см. рис. 92) четвёртый флажок. Добавьте ещё 1 кнопку BitBtn, дайте ей название «Задача 4.7». Запрограммируйте кнопку «Задача 4.7» так:

```
if (CheckBox1.Checked) and (CheckBox2.Checked)
and not ((CheckBox3.Checked) and (CheckBox4.Checked))
then ShowMessage('Условие задачи 4.7 верно');
```

Заготовка таблицы истинности условия к задаче 4.7

<усл. 1>and<усл. 2>and not((<усл. 3>)and(<усл. 4>))

№ случ.	Усл.1	Усл.2	Усл.3	Усл.4	Усл. 3,4' (3)and(4) (***)	Усл.3,4 not((3)and(4)) (**)	Всё усл. (*)
1	2	3	4	5	6	7	8
1	1	1	1	1	1	0	0
2	1	1	1	0	0	1	1
3	1	1	0	1			
4	1	0	1	1			
5	0	1	1	1			
6	1	1	0	0			
7	1	0	0	1			
8	0	0	1	1			
9	1	0	1	0			
10	0	1	0	1			
11	0	1	1	0			
12	1	0	0	0			
13	0	1	0	0			
14	0	0	1	0			
15	0	0	0	1			
16	0	0	0	0			

4.7.3. Проверьте правильность определения истинности всего условия (*), включая поочерёдно флажки согласно столбцам 2, 3, 4, 5 (если в столбце 2 будет единица, флажок 1 включите; если 0, то выключите; аналогично цифра в столбце 3 определяет положение флажка 2, цифра в столбце 4 – флажка 3, цифра в столбце 5 – флажка 4).

Если после расстановки положения флажков таблицы согласно проверяемому случаю и последующего нажатия кнопки «Задача 4.7» сообщение «Условие задачи 4.7 верно» выдалось и в столбце 8 единица либо сообщение не выдалось и в столбце 8 ноль, то истинность условия (*) для проверяемого случая определена верно.

Если же сообщение выдано, но в столбце 8 ноль, или же сообщение не выдано, но в столбце 8 единица, то нужно искать ошибку при определении истинности условия для проверяемого случая.

Сложные конструкции с операторами условия

Могут быть достаточно сложные конструкции, например:

```
if <условие 1> then begin <оператор 1>; <оператор 2>;
  if <условие 1.1> then begin <оператор 1.1>; <оператор 1.2> end
  else begin <оператор 1.1'>; <оператор 1.2'> end;
  <оператор 3>; <оператор 4>;
end else <оператор 1'>;
<оператор 5>.
```

Если в рассматриваемой конструкции условие 1 ложно, то выполняется оператор 1', после чего программа переходит к оператору 5 (рис. 94, табл. 4.5).

Таблица 4.5

Возможные случаи работы программы к рис. 94

№ п/п	Путь
1	Условие 1 → НЕТ(1) → Оператор 1' → Оператор 5
2	Усл. 1 → ДА(1) → Оп-р 1 → Оп-р 2 → Усл. 1.1 → ДА(2) → Оп-р 1.1 → Оп-р 1.2 → Оп-р 5
3	Усл. 1 → ДА(1) → Оп-р 1 → Оп-р 2 → Усл. 1.1 → НЕТ(2) → Оп-р 1.1' → Оп-р 1.2' → Оп-р 5

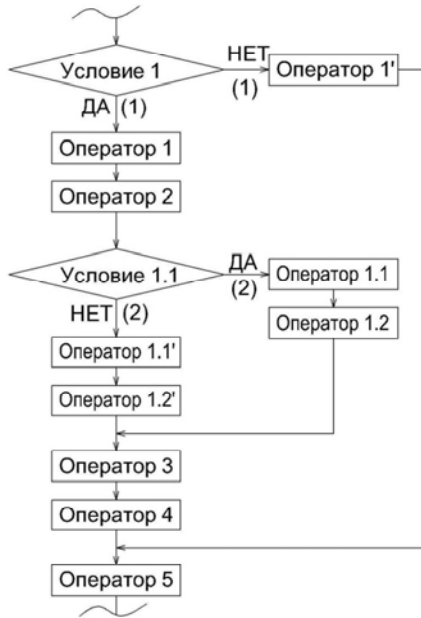


Рис. 94. Блок-схема одной из более сложных конструкций с операторами условия if

Если условие 1 истинно, то выполняются операторы 1 и 2, после чего проверяется условие 1.1. Если условие 1.1 истинно, то выполняются операторы 1.1 и 1.2, а если ложно, то операторы 1.1' и 1.2'. Затем в обоих случаях (и когда условие 1.1 истинно, и когда условие 1.1 ложно) выполняются операторы 3 и 4, после чего программа переходит к оператору 5.

Или такая конструкция (рис. 95, табл. 4.6):

```

if <условие 1> then begin <оператор 1> ; <оператор 2>;
  if <условие 1.1> then begin <оператор 1.1> ; <оператор 1.2>;
    if <условие 1.1.1> then <оператор 1.1.1> ;
    <оператор 1.3>
  end;
end;
<оператор 3>;
  
```

Если условие 1 ложно, то выполняется сразу оператор 3.

Если условие 1 истинно, то в любом случае выполняются операторы 1 и операторы 2. Далее проверяется условие 1.1.



Рис. 95. Блок-схема другой из более сложных конструкций с операторами условия if

Таблица 4.6

Возможные случаи работы программы к рис. 95

№ п/п	Путь
1	Условие 1 → НЕТ(1) → Оператор 3
2	Усл. 1 → ДА(1) → Оп-р 1 → Оп-р 2 → Усл. 1.1 → НЕТ(2) → Оп-р 3
3	Усл. 1 → ДА(1) → Оп-р 1 → Оп-р 2 → Усл. 1.1 → ДА(2) → Оп-р 1.1 → Оп-р 1.2 → Усл. 1.1.1 → ДА(3) → Оп-р 1.1.1 → Оп-р 1.3 → Оп-р 3
4	Усл. 1 → ДА(1) → Оп-р 1 → Оп-р 2 → Усл. 1.1 → ДА(2) → Оп-р 1.1 → Оп-р 1.2 → Усл. 1.1.1 → НЕТ(3) → Оп-р 1.3 → Оп-р 3

Если условие 1.1 ложно, то операторы 1.1 и 1.2 не выполняются и условие 1.1.1 проверяться не будет; программа сразу перейдет от условия 1.1 по стрелке «НЕТ(2)» к оператору 3.

Если условие 1.1 истинно, то выполняются операторы 1.1. и 1.2, после чего проверяется условие 1.1.1: если оно истинно, то выполнится оператор 1.1.1, а оно ложно, то оператор 1.1.1 не выполнится; далее операторы 1.3 и 3 выполняются в обоих случаях независимо от истинности условия 1.1.1.

Задача 4.8.

В задаче 4.4 проверим, не оставил ли пользователь пустыми блоки редактора для ввода исходных данных.

Это можно сделать, как и в задаче 4.3, добавив лишь в самом начале к имеющемуся программному тексту:

```
if      (Lenght(LabeledEdit1.Text)=0)      or
(Lenght(LabeledEdit2.Text)=0) then exit;
```

А можно сделать и по-другому, используя условие if not:

```
procedure TForm1.BitBtn1Click(Sender: TObject);
Var a,b,c:extended;
begin
if      not((Lenght(LabeledEdit1.Text)=0) or
(Lenght(LabeledEdit2.Text)=0)) then
begin
a:=StrToFloat(LabeledEdit1.Text);
b:=StrToFloat(LabeledEdit2.Text);
c:=a/b;
if CheckBox1.Checked then c:=Round(c);
LabeledEdit4.Text:=FloatToStr(c)
end
end;
```

Такой вариант усложняет конструкцию тем, что дополнительно вводит операторные скобки begin end.

Контрольные вопросы

1. Правила расстановки знаков препинания при использовании слов if, then, begin, end, else для оператора if и else для оператора case.
2. Какой блок используется для записи условия при составлении блок-схемы?
3. Как определить по блок-схеме, где заканчивается оператор if?

4. Конструкция `if ... then`: в каком случае действие после `then` будет выполнено, а в каком случае программа сразу перейдёт к следующему оператору без выполнения действия после `then`?

5. Конструкция `if ... then... else`: в каком случае выполнится действие после `then`, а в каком случае – действие после `else`? Может ли оказаться так, что выполнится и действие после `then`, и действие после `else`? Может ли оказаться так, что не выполнится ни действие после `then`, ни действие после `else`, если действия после `then` и `else` не содержат условных операторов?

6. Как по блок-схеме отличить конструкцию `if ... then` от `if ... then ... else`: в каком случае после стрелки «НЕТ» будет что-то выполняться, а в каком случае после стрелки нет программа сразу пойдёт в конец оператора `if`?

7. Что представляет собой составной оператор и когда возникает необходимость в нём?

8. Таблицы истинности условий И, ИЛИ, исключающее ИЛИ.

9. Таблицы истинности условий НЕ, И НЕ, ИЛИ НЕ.

10. Что будет выполнять первым: 1) сложение или умножение; 2) условие И или условие ИЛИ; 3) условие НЕ или условие И; 4) условие НЕ или условие ИЛИ? Как изменить приоритет операций?

11. Приведите пример конструкции с вложенными операторами условия (`else if`).

12. Какое свойство флажка `CheckBox` отвечает за состояние флажка (включён он или выключен), если он может находиться только в двух положениях – «включён и выключен»?

13. Какое свойство отвечает за состояние флажка `CheckBox`, когда он может находиться в трёх положениях, и какие значения оно может принимать?

14. При помощи какого свойства можно изменить режим работы флажка `CheckBox` (например, он может находиться в двух состояниях (включён или выключен), а нужно сделать так, чтобы он мог находиться в трёх состояниях (включён, выключен, затемнён) или наоборот)?

15. Если на форме или на панели расположено сразу несколько флажков `CheckBox`, то может ли быть включено сразу несколько из них? Могут ли быть включены сразу все флажки? Можно ли выключить все флажки?

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Архангельский, А.Я. Приёмы программирования в Delphi / А.Я. Архангельский. – М. : ООО «Бином-Пресс», 2004. – 848 с.
2. Архангельский, А.Я. Программирование в Delphi для Windows. Версии 2006, 2007, TurboDelphi. / А.Я. Архангельский. – М. : ООО «Бином-Пресс», 2007. – 1248 с.
3. Вальвачёв, А.Н. Программировании на языке Паскаль для персональных ЭВМ ЕС / А.Н. Вальвачёв, В.С. Крисевич. – Минск : Выш. школа, 1989. – 223 с.
4. ГОСТ 19.003-80. Схемы алгоритмов и программ. Обозначение условные графические.
5. ГОСТ 19.701-90. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения.
6. Культин, Н.Б. Delphi в примерах и задачах / Н.Б. Культин. – СПб. : БХВ-Петербург, 2004. – 288 с.
7. Культин, Н.Б. Основы программирования в Delphi 7. / Н.Б. Культин. – СПб. : БХВ-Петербург, 2007. – 608 с.
8. Культин, Н.Б. Основы программирования в Embarcadero Delphi / Н.Б. Культин. – Интернет-издание, 2015. – 229 с.
9. Лукин, С.Н. Турбо Паскаль 7.0. Самоучитель для начинающих / С.Н. Лукин. – М. : Диалог-МИФИ, 1999. – 384 с.

ОГЛАВЛЕНИЕ

Предисловие	3
1. Основы работы с VCL Forms Application.	
Работа с компонентами	6
1.1. Создание VCL Forms Application	6
1.2. Сохранение проекта	8
1.3. Открытие созданного проекта.....	10
1.4. Компиляция и отладка программы	10
1.5. Добавление командной кнопки на форму.....	11
1.6. Изменение свойств формы и компонентов	12
1.7. Программирование кнопки «Выход»	15
<i>Контрольные вопросы</i>	19
2. Изменение свойств компонента в программе.	
Компонент выбора Radiobutton	21
<i>Контрольные вопросы</i>	37
3. Линейные алгоритмы	38
<i>Контрольные вопросы</i>	53
4. Разветвляющиеся алгоритмы. Оператор условия if	55
<i>Контрольные вопросы</i>	94
Список рекомендуемой литературы	96

Учебное издание

КАЛИНИН Никита Владимирович

**РАБОТА С КОМПОНЕНТАМИ
И ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ
И РАЗВЕТВЛЯЮЩИХСЯ АЛГОРИТМОВ
В ИНТЕГРИРОВАННОЙ
СРЕДЕ DELPHI XE**

Учебно-методическое пособие
для студентов специальности 1-37 01 02
«Автомобилестроение (по направлениям)»

Редактор *Т. В. Мейкиане*
Компьютерная верстка *Е. А. Беспанской*

Подписано в печать 22.12.2019. Формат 60×84 ¹/₁₆. Бумага офсетная. Ризография.

Усл. печ. л. 5,70. Уч.-изд. л. 4,45. Тираж 100. Заказ 19.

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.