МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ
И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Северо-Кавказский филиал
ордена Трудового Красного Знамени федерального государственного
бюджетного образовательного учреждения высшего образования
«Московский технический университет связи и информатики»

Кафедра **«Информатика и вычислительная техника »**

Автоматизация управления информационными системами

Лабораторные работы по Delphi и Lazarus

Ростов-на-Дону
2021

Содержание

1. Компоненты среды Delphi и Lazarus. Репозиторий объектов и эксперты.

Обращение к справочной системе Help

Для вызова справочной системы необходимо выбрать соответствующую команду в выпадающем меню Help или отметить элемент интерфейса в исходном тексте и нажать клавишу F1. При нажатии кнопки “Разделы” в окне Help появляется диалоговое окно справочной системы Windows 95, которое позволяет вам просмотреть содержание всех файлов Help группы, отыскать ключевое слово по индексу или начать процесс поиска.

Меню и команды Delphi

Чтобы выдать команду в среде Delphi, можно воспользоваться тремя основными способами:

- С помощью меню.
- С помощью полоски SpeedBar (инструментальной линейки).
- С помощью SpeedMenu (одного из локальных меню, которое активизируется при нажатии правой кнопки мыши).

Меню File

Команды выпадающего меню File можно использовать для работы, как с проектами, так и с файлами исходного кода.

К командам, работающим с проектами, относятся New, New Application, Open, Reopen, Save Project As, Save All, Close All, Add to Project и Remove from Project. С файлами исходного кода работают команды New, New Form, New Data Module, Open, Reopen, Save As, Save, Close и Print. Основной командой является File/New, которую можно использовать для вызова экспертов, для начала работы с новым приложением, для наследования формы из уже существующей и т.д. Чтобы открыть проект или файл исходного кода, с которыми вы работали последний раз, используйте команду File/Reopen.

Меню Edit

Стандартные возможности меню Edit применимы как к тексту, так и к компонентам формы. Можно копировать и вставлять тот или иной текст в редакторе, копировать и вставлять компоненты в одной форме или из одной формы в другую. Также можно копировать и вставлять компоненты в другое групповое окно той же формы, например, в панель или блок группы; копировать компоненты из формы в редактор, и наоборот. Delphi помещает компоненты в буфер обмена, преобразуя их в текстовое описание. Можно соответствующим образом отредактировать этот текст, а затем вставить его обратно в форму в виде нового компонента. Можно выбрать несколько компонентов и скопировать их как в другую форму, так и в текстовый редактор. Это может пригодиться, когда вам придется работать с рядом схожих компонентов. Вы сможете скопировать один компонент в редактор, размножить его нужное число раз, а затем вставить назад в форму целую группу.

Меню Search

Если вы выберете команду Incremental Search, то вместо того чтобы показать диалоговое окно, где вводится образец для поиска, Delphi переходит в редактор. Когда вы введете первую букву, редактор перейдет к первому слову, которое начинается с этой буквы. Продолжайте набор букв и, курсор будет последовательно переходить к словам, в начале которых будут стоять введенные символы. Эта команда очень эффективна и чрезвычайно быстра. Команда Browse Symbol вызывает Object Browser – инструмент, который можно использовать для просмотра многих деталей при исследовании откомпилированной программы.

Меню View

Большинство команд меню View применяются для отображения какого-либо окна среды Delphi, например Project Manager, Breakpoints List или Components List. Эти окна не связаны друг с другом. Команда Toggle Form/Unit используется для

перехода от формы, над которой вы работаете к ее исходному коду, и обратно. Команда New edit window создает дубликат окна редактирования и его содержимого. В Delphi это единственный способ просмотреть два файла рядом друг с другом, поскольку редактор для показа нескольких загруженных файлов использует ярлычки. После дублирования окна редактирования могут содержать разные файлы. Последние две команды меню View можно использовать для удаления с экрана полоски SpeedBar и палитры Components, хотя при этом среда Delphi становится менее удобнее для пользователя. Команда Build All заставляет Delphi откомпилировать каждый исходный файл проекта, даже если после последней трансляции он не был изменен. Для проверки написанного кода без создания программы можно использовать команду Syntax Check. Команда Information дает некоторые подробности о последней выполненной вами трансляции. Команда Options применяется для установки опций проекта: опций компилятора и редактора связей, опций объекта приложения и т.д.

Меню Run

Меню Run можно было бы назвать Debug (отладка). Большинство команд в нем относится к отладке, включая саму команду Run. Программа, запускаемая внутри среды Delphi, выполняется в ее интегрированном отладчике (если не отключена соответствующая опция). Для быстрого запуска приложения используется клавиша F9. Остальные команды применяются в процессе отладки для пошагового выполнения программы, установки точек прерывания, просмотра значений переменных и объектов, и т.п.

Меню Component

Команды меню Component можно использовать для написания компонентов, добавления их в библиотеку, а также для конфигурирования библиотеки или палитры компонентов.

Меню Tools

Меню Tools содержит список нескольких внешних программ и инструментальных средств. Команда Tools позволяет сконфигурировать это выпадающее меню и добавить в него новые внешние средства. Меню Tools также включает команду для настройки репозитория и команду Options, которая конфигурирует всю среду разработки Delphi.

Полоска кнопок быстрого доступа SpeedBar

Наиболее часто используемые команды Delphi имеются в инструментальной линейке SpeedBar. Изменить размеры SpeedBar можно буксируя толстую линию между ней и палитрой Ccomponents. Другие разрешенные в SpeedBar операции добавляют, удаляют или заменяют пиктограммы с помощью команды Configure собственного локального меню SpeedBar. Эта операция вызывает инструмент SpeedBar Editor. Чтобы добавить пиктограмму в SpeedBar, необходимо найти ее в нужной категории и отбуксировать в полоску. Подобным образом можно отбуксировать пиктограмму за пределы SpeedBar или просто передвинуть ее в другое место.

Локальные меню. SpeedMenu

Хотя меню Delphi содержит большое количество элементов, не все команды доступны через выпадающие меню. Иногда для некоторых окон или областей окна приходится использовать SpeedMenu (локальное меню). Чтобы активизировать SpeedMenu, нужно нажать над необходимым элементом интерфейса пользователя правую кнопку мыши или клавиши Alt и F10.

Работа с формами

Проектирование форм – ядро визуальной разработки в среде Delphi. Каждый помещаемый в форму компонент или любое задаваемое свойство сохраняется в файле, описывающем форму (DFM-файл), а также оказывает некоторое влияние на исходный текст, связанный с формой (PAS-файл).

Можно начать новый пустой проект, создав пустую форму или начать с существующей формы (используя различные доступные шаблоны) или добавить в проект новые формы. Проект (приложение) может иметь любое число форм.

При работе с формой можно обрабатывать ее свойства, свойства одного из ее компонентов или нескольких компонентов одновременно. Чтобы выбрать форму или компонент, можно просто щелкнуть по нему мышью или воспользоваться Object Selector (комбинированный список в Object Inspector), где всегда отображены имя и тип выбранного элемента. Для выбора нескольких компонентов можно или нажать клавишу Shift и щелкать по компонентам левой кнопкой мыши, или отбуксировать в форме рамку выбора.

SpeedMenu формы содержит ряд полезных команд. Для изменения относительного расположения компонентов одного вида можно использовать команды Bring to Front и Send To Back. Командой Revert To Inherited можно воспользоваться, чтобы в унаследованной форме установить те значения свойств выбранного компонента, которые были
у них в родительской форме. При выборе сразу нескольких компонентов вы можете выровнять их или изменить их размеры.

Спомощью SpeedMenu можно также открыть два диалоговых окна, в которых устанавливается порядок обхода визуальных управляющих элементов и порядок создания невизуальных управляющих элементов. Команда Add To Repository добавляет текущую форму в список форм, доступных для использования в других проектах.

Для установки положения компонента кроме применения мыши имеются еще два способа: ♦ Установка значений для свойств Top и Left.

♦ Использование клавиш курсора при нажатой клавише Ctrl.

Метод Ctrl+клавиша курсора особенно удобен при тонкой подстройке положения элемента. Точно также, нажимая клавиши курсора при нажатой клавише Shift, можно подстроить размер компонента.

Палитра компонентов

Чтобы добавить в текущую форму новый компонент, можно щелкнуть на одной из страниц палитры Components, а затем, чтобы разместить новый элемент, щелкнуть в форме. Причем в форме можно или буксировать мышь с нажатой левой кнопкой, чтобы установить сразу и размер, и положение компонента, или просто щелкнуть один раз, позволяя Delphi установить размер по умолчанию.

Каждая страница палитры содержит ряд компонентов, которые обозначены пиктограммами и именами, появляющимися ввиле подсказки. Эти имена являются официальными названиями компонентов. В действительности это

названия классов, описывающих компоненты без первой буквы T (например, если класс называется Tbutton, имя будет Button). Если необходимо поместить в форму несколько компонентов обного и того же вида, то при выборе компонента щелчком в палитре удерживайте нажатой клавишу Shift Затем при каждом щелчке в форме Delphi будет вставлять новый компонент выбранного вида.Чтобы остановить эту операцию, просто щелкните по стандартному селектору (пиктограмма стрелки) слева от палитры Components.

Object Inspector

Object Inspector используется при проектировании формы для установки свойств компонента (или самой формы). В его окне в двух колонках изменяемого размера перечислены свойства (или события) выбранного элемента и их значения. Окно Object Selector в верхней части Object Inspector указывает текущий компонент и его тип данных, и этот селектор можно использовать для изменения текущего выбора.

BObject Inspector перечислены не все свойства компонентов, а только те, что могут быть установлены на этапе проектирования. Правая колонка Object Inspector разрешает правильное редактирование в соответствии с типом данных свойства. В зависимости от свойства можно вставить строку или число, выбрать из списка опций (на эту возможность указывает стрелка) или вызвать специальный редактор (об этом говорит овальная кнопка). Для некоторых свойств, таких, как Color, разрешен и ввод значения, и выбор элемента из списка, и вызов специального редактора.

Написание кода

При проектировании формы в Delphi обычно приходится писать кое-какой код для отклика на некоторые из ее событий. Когда вы нажимаете кнопку мыши в форме или на компоненте, Windows посылает вашему приложению сообщение, информируя его об этом событии. Реакция Delphi состоит в получении сообщения о событии и вызове соответствующего метода отклика на событие. Для каждого вида компонентов Delphi определяет ряд различных событий. Вы можете узнать о событиях, доступных для формы или компонента, рассматривая страницу Events окна Object Inspector в тот момент, когда выбран необходимый элемент.

Вставим в форму компонент – кнопку – и заставим его откликаться на событие, связанное с нажатием левой кнопки мыши. При щелчке по кнопке мышью в работающей программе возникает событие OnClick (По щелчку). Пока это событие никак не обрабатывается программой и поэтому нажатие кнопки не приведет ни к каким последствиям. Чтобы заставить программу реагировать на нажатие кнопки, необходимо написать на языке Object Pascal фрагмент программы, который называется обработчиком события. Фрагмент оформляется в виде специальной подпрогаммы – процедуры.

Чтобы заставить Delphi самостоятельно сделать заготовку для процедуры обработчика события OnClick, дважды щелкните мышью по вновь вставленному компонету. В ответ Delphi активизирует окно кода и вы увидите в нем такой текстовый фрагмент:

```
procedure TForm1.Button1Click(Sender: Tobject);
begin

end;
```

Слово **procedure** извещает компилятор о начале подпрограммы –процедуры. За ним следует имя процедуры TForm1.Button1Click. Это имя – составное: оно состоит из имени класса TForm1 и собственно имени процедуры Button1Click.

Каждый компонент принадлежит к строго определенному классу, а все конкретные экземпляры компонентов, вставляемые в форму, получают имя класса с добавленным числовым индексом. По используемому в Delphi соглашению все имена классов начинаются с буквы T. Таким образом, имя TForm1 означает имя класса, созданного по образцу стандартного класса Tform. Если вы посмотрите начало текста в окне кода, то увидите следующие строки:

```
type
  TForm1 = class(TForm)
    Button1: Tbutton;
    procedure TForm1.Button1Click(Sender: Tobject);
 private
    {private declarations}
 public
```

```
    {public declaration}
  end;
var
  Form1: TForm1;
```

Строка TForm1 = **class**(TForm) определяет новый класс TForm1, который порожден от стандартного класса TForm. Стандартный класс TForm описывает пустое Windows-окно, в то время как класс TForm1 описывает окно с уже вставленным в него компонентом *кнопка.*

Каждый раз когда вы работаете над событием, Delphi открывает редактор с исходным файлом, связанным с формой. Редактор позволяет работать над несколькими файлами исходного текста одновременно, используя образ “записной книжки с ярлычками”. Каждая страница записной книжки соответствует отдельному файлу.

Компиляция проекта

Компиляция проекта происходит при запуске его на выполнение. Транслируя проект, Delphi компилирует только те файлы, которые изменялись. Но если вы выберите в меню Project команду Build All, то будет откомпилирован каждый файл, даже если он не изменялся.

Проект содержит список файлов исходного кода, которые являются частями проекта, а также соответствующих им форм (при наличии таковых). Сначала каждый файл исходного кода превращается в откомпилированный модуль Delphi – файл с тем же именем, что и у исходного файла на языке Паскаль, но с расширением DCU.

Во время компиляции и компоновки самого проекта, откомпилированные модули, составляющие проект, сливаются (или связываются) друг с другом и с кодом библиотеки VCL, образуя исполнимый файл.

Интегрированный отладчик

В Delphi имеется встроенный отладчик, обладающий огромным количеством возможностей. При каждом запуске из среды Delphi, программа уже выполняется в отладчике. Для установки точки останова или щелкните в промежутке между левой рамкой окна редактирования и текстом, или выберите в SpeedMenu команду Toggle Breakpoint, или нажмите клавишу F5. Когда вы разместили ряд точек останова, можете использовать команду Breakpoints меню View, чтобы открыть окно Breakpoints List. Один из пунктов в верхней части окна Breakpoints List предполагает добавление условия в точке останова, так чтобы программа выполнялась только при выполнении данного условия. Кнопка Step Over на линейке SpeedBar позволяет просмотреть выполнение операторов один за другим, а кнопка Trace Into позволяет трассировать вызываемые методы (т.е. выполнять шаг за шагом код подпрограмм).

Если программа остановлена в отладчике, вы можете проверить значение любого идентификатора (для переменных, объектов, компонентов, свойств и т.д.), который доступен в этой точке программы. Для этого существуют два способа: использовать диалоговую панель Evaluate/Modify или добавить элемент в окно Watch List. Самый простой способ открыть диалоговую панель Evaluate/Modify – выделить переменную в редакторе исходного текста, а затем выбрать команду Evaluate/Modify из SpeedMenu редактора. Вы можете устанавливать элементы наблюдения, используя команду Add Wath at Cursor в Speed Menu редактора.

Файлы, создаваемые системой

Когда вы сохраняете новый проект, Delphi создает ряд файлов. Здесь приводится списо наиболее важных
файлов.

- Основной файл проекта типа .DPR. Это основной модуль исходного текста проекта. Имеется только один DPR-файл для каждого проекта. Этот файл, кроме всего прочего, перечисляет имена других файлов, составляющих проект.
- Файлы формы типа .DFM. Это двоичные файлы ресурсов, содержащие определение визуальных форм. В проекте может быть много форм и каждая имеет собственный .DFM файл.
- Файл модуля Паскаля типа .PAS. Содержит код Object Pascal для соответствующей формы или для автономного модуля кода.
- Файл опций проекта типа .OPT. Файл, который содержит различные установки Delphi (текстовый файл).
- Откомпилированные файлы модуля типа .DCU. Содержат объектный код существующего .PAS-файла модуля.
- Откомпилированные программные файлы типа .EXE. Это собственно программы Windows.
- Откомпилированные файлы динамических библиотек типа .DLL. Это откомпилированные модули Windows, которые могут использоваться одновременно многими программами Windows.

Страницы репозитория объектов

В Delphi есть несколько команд меню, с помощью которых вы можете создать новую форму, новое приложение, новый модуль данных, новый компонент и т.п. Эти команды находятся в меню File, а также в других выпадающих меню. Но если вместо них выдать команду File/New, Delphi откроет окно Object Repository.

Репозиторий используется для создания новых элементов любого вида: форм, приложений, модулей данных, библиотек, компонентов и т.д. Диалоговое окно Object Repository содержит несколько страниц:

- Страница New позволяет создавать новые элементы многих разных типов.

- Страница текущего проекта (в действительности на ярлычке данной страницы вы увидите имя проекта, например Project1) позволяет унаследовать форму или модуль данных от аналогичного объекта, включенного в ваш текущий проект.
- Страницы Forms, Dialogs , Data Modules позволяют создавать новые формы, диалоговые панели и модули данных, используя эксперты или существующие объекты этих типов.
- Страница Project позволяет скопировать файлы из хранящегося в репозитории проекта.

В нижней части диалогового окна Object Repository находятся три радиокнопки, с помощью которых можно указать: хотите ли вы скопировать существующий элемент, унаследовать его или применить непосредственно, не копируя.

Страница New

Список элементов, которые можно создать с помощью этой страницы:

- Application создает новый пустой проект.
- Data Module создает новый пустой модуль данных.
- DLL создает новую библиотеку DLL.
- Form создает новую пустую форму.
- Text открывает в редакторе новый текстовый файл.
- Unit создает новый пустой модуль, не связанный с формой.

Страница Forms

Ниже приведен список необходимых для работы предопределенных форм:

- About Box – простая панель “О программе”.
- Duil List Box – форма с двумя разными списками; позволяет пользователю выбрать ряд элементов в одном списке и нажатием кнопки переместить их во второй. Кроме компонентов эта форма содержит значительное количество не очень простого кода на языке Паскаль.

Страница Dialogs

Эта страница похожа на предыдущую, но содержит другие элементы.

- Dialog Expert – простой эксперт, который способен сгенерировать различные диалоговые панели с одной или несколькими страницами.
- Dialog with help – два варианта диалоговой панели с кнопкой вызова справочной информации.
- Password dialog – диалоговая панель с простым окном редактирования, которая имеет необходимые для ввода пароля опции; код отсутствует.
- Standart Dialog Box – стандартная диалоговая панель, которая доступна в двух вариантах с различным расположением кнопок.

Страница Data Modules

Модуль данных это особый вид формы, который никогда не появляется на экране во время выполнения и может использоваться для хранения невизуальных компонентов. Чаще всего он применяется для описания доступа к базе данных.

Страница Projects

Эта страница содержит схемы проектов, которые вы можете использовать в качестве стартовой площадки для создания собственного приложения.

- Application Expert – простой эксперт, в котором вы можете выбрать файловую структуру и некоторые другие элементы приложения.
- MDI Application задает ключевые элементы программы с интерфейсом Multiple Document Interface (MDI). В этом приложении определена основная форма для окна MDI-фрейма, содержащая меню, строку состояния и инструментальную линейку. Кроме того, в нем имеется вторая форма, которую на этапе выполнения можно использовать для создания дочерних окон.
- SDI Application определяет основную форму со стандартными атрибутами современного интерфейса пользователя, включая инструментальную линейку и строку состояния, а также типичную панель About.
- Win95 LogoApplication описывает простое приложение, в котором присутствует большинство элементов, необходимых для получения логотипа Windows 95. Данная команда в основном создает SDI-приложение с компонетом RichEdit и вставляет в него код, который делает приложение совместимым с электронной почтой.

Эксперты Delphi

Delphi разрешает не только копировать или использовать существующий код, но и создавать новые формы, приложения или другие файлы с кодом, применяя эксперт. Эксперт позволяет вам ввести ряд опций и с помощью некоторой внутренней схемы создает код, соответствующий вашему заказу.

Application Expert

Его можно активизировать из страницы Project окна Object Repository. Первая страница этого эксперта позволяет добавить в программу некоторые стандартные выпадающие меню: File, Edit, Window и Help. На второй странице эксперта вы зададите расширения тех файлов, которые должна рассматривать программа. Вам придется ввести как описание файла, например Текстовый файл (*.txt) так и его расширение txt Эти величины будут использоваться в качестве значений по умолчанию диалоговыми окнами File Open и File Save, которые Application Expert добавит в программу (если вы выбрали поддержку файлов).

Application Expert выведет на экран прекрасное визуальное средство, которым вы можете воспользоваться для построения инструментальной линейки. В нем вы выбираете одно из выпадающих меню и появляется ряд стандартных кнопок, которые

соответствуют его типичным элементам (но только, если это меню было выбрано на первой странице эксперта).

После завершения работы над инструментальной линейкой вы можете перейти на последнюю страницу эксперта. Здесь устанавливаются многие дополнительные опции, например, можно доказать поддержку интерфейса MDI, добавить строку состояния или разрешить всплывающие подсказки. Вы также можете задать имя нового приложения и указать каталог для его исходных файлов. Каталог для приложений должен уже существовать. Если вы хотите поместить файлы проекта в новый каталог, выберите кнопку Browse и введите новый путь – появится диалоговое окно с вопросом, хотите ли вы создать новый каталог.

Dialog Box Expert

Это простой эксперт, предоставленный вместе с исходным текстом в качестве демонстрационного примера. Вы можете воспользоваться этим экспертом как инструментом для построения диалоговых панелей двух различных видов: простых и многостраничных диалоговых панелей. Если вы выберите простую диалоговую панель, эксперт перейдет к третьей странице, где вы сможете задать компоновку кнопок. Если вы выберите многостраничную панель, появится промежуточная страница, которая позволяет ввести тексты для разлимчных ярлычков.

Задание № 1.1

- Изучить вышеизложенный материал с помощью компьютера.
- Создать с помощью Application Expert приложение, содержащее выпадающие меню File, Edit, Window и Help, инструментальную линейку, строку состояния и всплывающие подсказки. Приложение записать в каталог C:/Program Files/Borland/Delphi3/user. Если такой каталог не существует, создать его. Исследовать в окне редактирования полученный код.
- Построить с помощью Dialog Box Expert простую и многостраничную диалоговые панели.
- Создать MDI Application, SDI Application, Win95 LogoApplication. Исследовать код. Приложение на диске не запоминать.

2. Типы данных и операторы языка Паскаль.

Тип определяет значения, которые может иметь переменная, и операции, которые могут быть выполнены над этой переменной. После слова var сопровождается список имен переменных, сопровождаемых двоеточием и именем типа данных.

Предопределенные типы данных

Существует несколько предопределенных типов данных, которые можно разделить на три группы:
перечислимые типы, вещественные типы и строки.

Перечислимые типы

Три наиболее важных перечислимых типа –Integer (целочисленный), Boolean (логический) и Char (символьный). Однако существует несколько других типов, которые имеют тот же смысл, но иное внутреннее представление и диапазон значений.

Ниже приведен полный список перечислимых типов:

- Integer, Cardinal, ShortInt, SmallInt, LongInt, Byte, Word
- Boolean, ByteBool, WordBool, LongBool
- Char, ANSIChar, WideChar

Задание № 2.1

Сконструировать форму с шестью кнопками, имена которых ShortInt, SmallInt, Integer, Byte, Word, Cardinal; с четырьмя статическими надписями (компонент Label) Type, Size, Max, Min и четырьмя надписями для вывода информации о типе при каждом нажатии одной из кнопок. Для этого записать для каждой кнопки метод отклика на событие OnClick, используя свойство Caption надписей для вывода информации и функции SizeOf – размер внутреннего представления переменной данного типа, High – самое высокое значение в диапазоне перечислимого типа, Low – самое низкое значение, а также функцию IntToStr – преобразование числа в строку. Пример строки кода:

```
SizeLabel.Caption:=IntToStr(SizeOf(Number));
```

Вещественные типы

Вещественные типы представляют разнообразные форматы чисел с плавающей запятой. Меньше всего памяти требуется доля хранения чисел типа Single. Затем идут числа типа real, Double и Extended. Все это – типы чисел с плавающей запятой, имеющие разную точность представления.

Кроме них есть еще тип данных Comp, который описывает очень длинные целые числа, и Currency – тип данных, который имеет четыре десятичных знака после запятой и 64-битовое внутреннее представление. Последний тип данных был добавлен для показа больших денежных сумм без потери младших значащих цифр. Вещественные ипы используются в программах, содержащих математические формулы. Сама Delphi использует вещественные типы в типе данных TdateTime.

Типы данных, специфичные для Windows

В Delphi имеются типы данных, которые определены системой Windows – дескриптор и ссылка на цвет. Их имена, соответственно, Thandle и TcolorRef. Первый тип – это лишь переопределение типа данных Cardinal, а второй переопределение типа LongInt.

Ссылка на цвет – это просто число, описывающее цвет. Вы можете выбрать любой цвет с помощью функции RGB или напрямую значение для красной, зеленой и синей составляющих величины типа TcolorRef.

В Windows дескриптор – это число, которое является ссылкой на внутреннюю структуру данных системы. Например, когда вы работаете с окном, система выдает вам дискриптор окна (HWND). Тем самым она сообщает, что окно, с которым вы имеете дело, является окном под номером, например, 142. С этого момента ваше приложение может использовать данный номер, чтобы попросить систему обработать необходимое окно: переместить его, изменить размеры, уменьшить до пиктограммы и т.п.

Другими словами, дескриптор является внутренним кодом, который вы можете применить для обращения к конкретному элементу, обрабатываемому системой, включая окна, растровые изображения, пиктограммы, блоки памяти, курсоры, шрифты, меню и т.п.

Приведение и преобразование типов

Вы не можете присвоить переменной значение другого типа. Если в этом все же возникла необходимость, имеются две возможности. Первая возможность – приведение типов, которое выглядит как простой вызов функции, но вмето имени функции используется имя типа данных адресата:

```
var
  N:Integer;
  C:Char;
  B:Boolean;
begin
   N:= Integer('X');
   C:= Char(N) ;
   B:= Boolean(0);
```

Строго говоря, операцию приведения можно осуществлять между теми типами данных, которые имеют одинаковый размер. Обычно безопасным является приведение между перечислимыми или между вещественными типами, но вы также можете выполнить приведение между типами указателей (а также объектов).

Вторая возможность – использовать подпрограмму преобразования типов, например – Trunc – преобразует значение вещественного типа в значение целочисленного типа, отсекая дробную часть; IntToStr – преобразует число в строку; StrToInt – преобразует строку в число, вызывая исключение в случае неправильной строки и т.д.

Массивы

Используйте для работы с массивами функции Low и High (особенно в циклах), поскольку они делают код независимым от диапазона массива. Если позже вы измените объявленный диапазон индексов массива, то код, который использует Low и High, останется работоспособным, а код, который жестко привязан к диапазону массива работать не будет. Функции Low и High облегчают поддержку вашего кода и делают его более надежным. Применение этих функций не приводит к лишним затратам на этапе выполнения. Во время компиляции они преобразуются в константные выражения, а не вдействительные обращения к функциям.

Длинные строки

Чтобы устранить ограничения традиционных строк Паскаля, в Delphi введена поддержка длинных строк. В действительности имеется два типа строк:

- Тип ShortString соответствует обычным строкам Паскаля. Каждый элемент короткой строки имеет тип ANSIChar.
- Тип AnsiString соответствует новым длинным строкам переменного размера. Такие строки размещаются динамически и их размер практически не ограничен. В основе таких строк также лежит тип ANSIChar.

В зависимости от значения новой директивы компилятора $H вы получите или короткую, или длинную строку. По умолчанию стоит значение $H+, что соответствует длинным строкам.

Операторы языка Паскаль

Здесь будет дано краткое описание использования операторов Object Pascal в Delphi.

Рассмотрим пример, который демонстрирует различие между фиксированным счетчиком и циклом с псевдослучайным счетчиком. Начните новый пустой проект и поместите в его основную форму список и две кнопки. Теперь в событие OnClick кнопок можно добавить некоторый код. Первая кнопка содержит простой цикл for для отображения списка чисел. До выполнения этого цикла, который добавляет несколько строк в свойство Items списка, вы должны очистить содержимое самого списка.

```
procedure TForm1.Botton1.Click (Sender:TObject);
  var
    I:Integer;

begin
  ListBox1.Items.Clear;
   For I:= 1 to 20 do
      ListBox1.Items.Add(' String ' + IntToStr(I));
end;
```

Код, связанный со второй кнопкой использует цикл while, который основан на счетчике, увеличивающемся случайным образом.

```
procedure TForm1.Botton2.Click(Sender:TObject) ;
  var
     I:Integer;

begin
  ListBox1.Items.Clear;
  Randomize;
  I:=0;
  while    I<1000 do
     begin
        I:=I+Random(100);
        ListBox1.Items.Add(' Random number: ' + IntToStr(I));
     end;
end;
```

При каждом щелчке по второй кнопке числа будут различными, так как они зависят от генератора случайных чисел.

Элементы управления редактированием

Класс TCustomEdit – это абстрактный класс для всех элементов управления редактированием в Delphi. Он включает простой элемент управления редактированием, элементы управления редактированием по маске и все элементы управления мемо.

Некоторые свойства и методы, реализованные классом TCustomEdit

Используйте или	*Чтобы сделать это*
установите это	
Brush	Определить цвет и шаблон, используемые в качестве фона оконного элемента управления
CanFocus	Определить, может ли оконный элемент управления получить фокус
Clear	Очистить содержимое элемента управления редактированием
Enabled	Определить доступность элемента управления
Focused	Определить находится ли оконный элемент управления в фокусе
Font	Определить шрифт, используемый для вывода текста в элементе управления
GetSelTextBuf	Скопировать выбранный текст из элемента управления в буфер
GetTextBuf	Скопировать текст из элемента управления в буфер
GetTextLen	Получить длину текста элемента управления
Hide	Сделать элемент управления невидимым
Hint	Определить текст, который отображается в подсказке для элемента управления
SelectAll	Выбрать весь текст в элементе управления
SelLength	Определить длину выбранного текста в элементе управления
SelStart	Определить исходную позицию выбранного текста
SelText	Получить доступ к выбранному тексту в элементе управления редактирования
SetFocus	Установить фокус на оконный элемент управления
Show	Сделать элемент управления видимым
Text	Обратиться к изменяемому тексту на элементе управления

Класс TEdit инкапсулирует большинство возможностей стандартного элемента управления редактированием известного как “поле” или “ текстовое поле”. Элемент управления редактированием предоставляет одну доступную для редактирования строку текста внутри элемента управления с необязательной рамкой. При желании

текст на элементе управления редактированием может быть предназначен только для чтения, так что пользователь изменять его не сможет.

Класс TEdit предусматривает только основные функциональные возможности элемента управления редактированием. При необходимости ограничить диапазон ввода, воспринимаемый этим элементом управления, используйте вместо него элемент управления редактированием по маске (TMaskEdit). Класс TEdit порожден непосредственно от TCustomEdit.

Задание № 2.2

Сконструировать форму, которая будет содержать следующие управляющие элементы:

- Окно редактирования со связанной с ним меткой Operand 1. В этом окне вводится первый операнд.
- Окно редактирования со связанной с ним меткой Operator. В этом окне вводится операция. В программе предусмотрены операции +, -, / и * .
- Окно редактирования со связанной с ним меткой Operand 2. В этом окне вводится второй операнд.

- Окно редактирования со связанной с ним меткой Result. В этом окне отображается результат запрошенной вами операции.
- Кнопка Close, которая закрывает приложение.

Тип операндов – целый.

Задание № 2.3

Сконструировать форму, которая будет содержать следующие управляющие элементы:

- Элемент управления редактированием по маске (TMaskEdit) со связанной с ним меткой Operand 1.
- Элемент управления редактированием по маске (TMaskEdit) со связанной с ним меткой Operand 2.
- Элемент управления редактированием по маске (TMaskEdit) со связанной с ним меткой Result.
- Кнопка Close, которая закрывает приложение.
- Шесть кнопок операций + , - , * , / , mod , div .
- Кнопка Clear для очистки окон элементов управления редактированием.

Тип операндов – вещественный. Написать код, который будет реализовывать указанные операции по нажатию соответствующей кнопки (связать код с событием OnClick кнопки).

3. Классы и модули.

Классы и сокрытие информации

Класс может содержать сколько угодно данных и любое количество методов. Однако для соблюдения всех правил объектно-ориентированного подхода данные должны быть скрыты, или инкапсулированы, внутри использующего их класса. Использование метода для получения доступа к внутреннему представлению объекта уменьшает риск возникновения ошибочных ситуаций и позволяет автору класса модифицировать внутреннее представление в будущих версиях. В Object Pascal имеются две различные конструкции, подразумевающих инкапсуляцию, защиту и доступ к переменным: классы и модули. С классами связаны некоторые специальные ключевые слова – спецификаторы доступа:

- private – элементы интерфейса класса видны только в пределах модуля, в котором определен класс. Вне этого модуля private-элементы не видны и недоступны. Если в одном модуле создано несколько классов, то все они видят private-разделы друг друга.
- Раздел public не накладывает ограничений на область видимости перечисленных в нем полей, методов и свойств. Их можно вызывать в любом другом модуле программы.
- Раздел published также не ограничивает область видимости, однако в нем перечислены свойства, которые должны быть доступны не только на этапе исполнения, но и на этапе конструирования программы. Раздел published используется при разработке компонентов. Без объявления раздел считается объявленным как published. Такой умалчиваемый раздел располагается в самом начале объявления класса любой формы и продолжается до первого объявленного раздела. В раздел published среда помещает описание вставляемых в форму компонентов. Сюда не нужно помещать собственные элементы или удалять элементы, вставленные средой.
- Раздел protected доступен только методам самого класса, а также любым потомкам, независимо от того, находятся ли они втом же модуле или нет.

Порядок следования разделов может быть любой, любой раздел может быть пустым.

Классы и модули

Приложения Delphi интенсивно используют модули. За каждой формой скрывается соответствующий ей модуль. Однако модули не обязаны иметь соответсвующие формы.

Модуль содержит раздел interface, где объявлено все, что доступно для других модулей, и раздел implementation с реальным кодом. Наконец, модуль может иметь два необязательных раздела : initialization с некоторым кодом запуска, который выполняется при загрузке в память программы, использующей данный модуль, и finalization, который выполняется при завершении программы.

Предложение uses в начале раздела interface к каким другим модулям мы должны получить доступ из раздела interface текущего модуля. Если же на другие модули необходимо сослаться из клда подпрограмм и методов, вы должны добавить новое предложение uses в начале раздела implementation.

В интерфейсе модуля можно объявить несколько различных элементов, в том числе процедуры, функции, глобальные переменные и типы данных. Также можно поместить в модуль класс. Delphi это делает автоматически при создании каждой формы. Чтобы создать новый не относящийся к форме модуль, выберите команду File/New и отметьте на странице New появившегося окна Object Repository элемент Unit.

Модули и область видимости

Область видимости идентификатора (переменной, процедуры, функции или типа данных, определяет ту часть кода, в которой доступен этот идентификатор. Основное правило состоит в том, что идентификатор является значимым только внутри его области видимости.

Если вы объявили идентификатор внутри блока определения процедуры, вы не сможете использовать данную переменную вне этой процедуры. Область видимости идентификатора охватывает всю процедуру, включая вложенные блоки.

Если вы объявили идентификатор в области реализации модуля, вы не можете применить его вне модуля,
номожете использовать в любом блоке и процедуре, определенных внутри модуля. Если вы объявили идентификатор в
интерфейсной части модуля, его область
видимости распространяется на любой другой модуль, где объявлен идентификатор. Любой идентификатор, объявленный в интерфейсе модуля, является глобальным; все другие идентификаторы принято называть локальными.

Модули и программы

Приложение Delphi создается из файлов исходного текста двух разных видов. Это один или несколько модулей и один файл программы.

Модули можно считать вторичными файлами, к которым обращается основная часть приложения – программа. На практике файл программы обычно является автоматически сгенерированным файлом с ограниченной ролью. Он нужен только для запуска программы, которая выполняет основную форму. Код файла программы, или файла проекта Delphi (DPR), можно отредактировать вручную или с помощью Project Manager и некоторых опций проекта.

Информация о типе на этапе выполнения

Правило языка Object Pascal о совместимости типов для классов-потомков позволяет вам использовать класс-потомок там, где ожидается класс предок, обратное невозможно. Теперь предположим, что класс Dog содержит функцию Eat, которая отсутствует в классе Animal.

Если переменная MyAnimal ссылается на объект типа Dog, вызов этой функции должен быть разрешен. Но если вы попытаетесь вызвать эту функцию, а переменная ссылается на объект другого класса, возникнет ошибка. Поскольку компилятор не способен определить, будет ли значение правильным на этапе выполнения, то делая явное приведение типов, мы рискуем вызвать опасную ошибку этапа выполнения программы.

Для решения данной проблемы можно воспользоваться некоторыми подходами, основанными на системе RTTI.
Каждый объект энает свой тип и своего предка и можем получить эту информацию с помощью операции is. Параметрами
операции is являются объект и тип:
if MyAnimal is Dog then …

Выражение is становится истинным, только если в настоящее время объект MyAnimal имеет тип Dog или тип потомка от Dog. Другими словами, это выражение приобретает значение True, если вы можете без риска присвоить объект (MyAnimal) переменной заданного типа данных (Dog). Такое прямое приведение можно выполнить так:

```
if MyAnimal is Dog then
  MyDog:= Dog(MyAnimal) ;
```

То же действие можно выполнить напрямую с помощью другой операции RTTI – as. Мы можем написать так:

```
MyDog:= MyAnimal as Dog ;
Text:= MyDog.Eat ;
```

Если мы хотим вызвать функцию Eat, можно использовать и другую нотацию:

```
(MyAnimal as Dog).Eat ;
```

Результатом выражения будет объект с типом данных класса Dog, поэтому выможете применить к нему любой метод этого класса.

Приведение с операцией as отличается от традиционного приведения тем, что в случае несовместимости типа объекта с типом, к которому вы пытаетесь его привести, порождается исключение EInvalidCast.

Чтобы избежать этого исключения, используйте операцию is и в случае успеха делайте простое приведение:

```
if MyAnimal is Dog then
  (Dog(MyAnimal)).Eat;
```

Задание № 3.1

Открыть модуль, не связанный с формой и поместить в него три класса:

- Класс Animal, который содержит в разделе public объявление конструктора Create и объявление метода-функции: Verse – звук, издаваемый животным. Тип результата возвращаемого функцией – string. Метод Verse объявить виртуальным и абстрактным. В разделе private класса определить переменную Kind : string.
- Класс Dog объявить потомком класса Animal. В разделе public этого класса объявить конструктор и методы Verse и Eat. Метод Eat типа string объявить виртуальным (пища животного).
- Класс Cat объявить потомком класса Animal. Раздел public класса содержит те же определения, что и соответствующий раздел класса Dog.

В реализациях конструктора каждого класса переменной Kind присваивается имя соответствующего животного, напрамир для класса Animal : Kind := ‘An Animal’.

Вреализациях методов Verse возвращается звук, издаваемый животным, например Verse := ‘Mieow’.

Вреализациях методов Eat возвращается название пищи, которой питается соответствующее животное.

- Задать имя модуля и имя проекта, в который этот модуль будет включен.
- Добавить в проект форму, которой присвоить имя Animals, также задать имя модулю, связанному с формой.
- В форме расположить три кнопки опций (компонент RadioButton) с названиями Animal, Dog, Cat ; кнопкой команды (компонент Button) с названием Kind и две крупный надписи (компонент Label) . Нажатию одной из кнопок опций

будет соответствовать выбор животного. При нажатии кнопки команды надписи должны отобразить звук, издаваемый животным и его пищу.

- Определите в классе формы private-переменную MyAnimal. Запишите код для обработчика события OnCreate формы, где создается объект типа Dog на который ссылается переменная MyAnimal.
- В обработчиках события OnClick каждой кнопки опций записать код, который удаляет текущий объект и создает новый.
- В обработчике события OnClick кнопки команды записать код, который будет помещать в надписи звук, издаваемый животным и его пищу. Для работы с методом Eat используйте операцию is для приведения типов.
- Если вы все сделали правильно, при запуске приложения надписи будут отображать пищу и звук для Dog и Cat и приложение завершит работу по ошибке при выборе Animal.
- Уберите ключевое слово abstract в объявлении метода Verse. Запустите приложение снова. Посмотрите что изменилось в работе приложения. Объясните различия.
- Попробуйте использовать метод Eat без приведения типов (без is).
- Разработайте два класса потомка от Dog, которые будут отображать особенности двух пород собак. Разработайте методы для этих классов, позволяющие получить некоторые характеристики породы (рост, длина шерсти, длина ушей и т.д.). Дополните форму компонентами позволяющими увидеть все характеристики разработанных классов.

4. Использование компонентов

Буксировка из одного компонента в другой

В Delphi буксировка обычко выполняется путем нажатия кнопки мыши над одним компонентом и освобождения над другим. Для этой операции можно написать код, который обычно копирует в компонент адресат свойство, значение или что-нибудь еще.

Задание № 4.1

- Создать форму, в которой расположить четыре надписи с названиями каждого цвета и надпись-адресат с некоторым описательным текстом. Цвет фона надписей задается с помощью свойства Color. Цвет текста можно задать используя свойство Font: на этапе проектирования открывается окно. Для того чтобы разрешить буксировку: выбрать значение dmAutomatic для свойства DragMode для надписей источников и предоставить методы отклика для пары событий в надписи-адресате.
- Первое событие – OnDragOver. Оно вызывается, когда во время буксировки вы перемещаете курсор над компонентом. Это событие указывает, что компонент воспринимает буксировку. Обычно событие считается завершенным после определения следующего факта: имеет ли компонент Source (тот, который инициализировал операцию буксировки) необходимый тип. Для события OnDragOver записать следующий код :

```
Accept:= Source is TLabel;
```

Этот код воспринимает операцию буксировки, активизируя соответствующий курсор, если только исходный объект действительно является объектом надписи.

- Второй метод, который вы должны написать, соответствует событию OnDragDrop:

```
Label5.Color:= (Source as TLabel).Color;
```

Обработка исключений

На этапе выполнения Delphi порождает исключения, когда какой-либо процесс идет неправильно. Если код вашей подпрограммы написан соответствующим образом, он может распознать возникшую проблему и попытаться ее решить; в противном случае исключение передается в код, который вызвал вашу подпрограмму и т.д. В конечном счете, если никто не обработал исключение, его обрабатывает Delphi, выводя на экран стандартное сообщение об ошибке и пытаясь продолжить выполнение программы.

Весь механизм строится на четырех ключевых словах:

try – определяет начало защищенного блока кода;

except – определяет конец защищеннго блока кода и вводит операторы обработки исключений в следующем виде: on (тип исключения) do (оператор)

finally – указывает необязательный блок, который используется для освобождения ресурсов, распределенных в блоке try перед обработкой исключения; этот блок завершается ключевым словом end.

raise – оператор, используемый для порождения исключений. Большинство исключений, которые вы встретите при программировании на Delphi, будут порождаться системой, но вы также можете создать их в собственном коде, когда во время выполнения обнаружатся недопустимые или несогласованные данные. Кроме того, ключевое слово raise можно использовать внутри обработчика для повторного порождения исключения, т.е. для передачи его следующему обработчику.

Примеры стандартных классов исключений:

- EAbort – реализует обработку любого исключения без сообщения;
- EConvertError – Ошибка преобразования в функциях StrToInt или StrToFloat;

- EDivByZero – ошибка целочисленного деления на ноль.

Если вы хотите, чтобы при правильной обработке исключений программа продолжала выполняться, отключите опцию отладки Break on Exception в окне Environment Options.

Восприятие ввода для пользователя

Обратим особое внимание на качество, характерное для многих управляющих элементов – фокус. Как определить какое окно или поле имеет фокус ввода? В каждый конкретный момент фокус имеет только одно поле. Вы можете перемещать фокус, используя клавишу Tab или щелкая мышью по другому компоненту. Каждый раз когда компонент получает или теряет фокус, к нему приходит соответствующее событие, которое указывает, что пользователь достиг (OnEnter) или покинул (OnExit) компонент.

При использовании компонента Edit для ввода чисел пользователь вместо цифры может набрать букву. Функции преоразования вернут код ошибки, что поможет определить действительно ли введено число. Когда можно выполнить такую проверку? Возможно, когда изменится значение блока редактирования, когда компонент потеряет фокус или когда пользователь щелкнет по некоторой кнопке в диалоговой панели. Можно просматривать входной поток в блоке редактирования и останавливать любой нечисловой код.

Задание № 4.2

- Создать форму с пятью полями редактирования и пятью соответствующими надписями, которые поясняют, какой вид проверки осуществляет соответствующий компонент Edit. Форма также содержит кнопку для проверки содержимого первого поля редактирования.
- Записать следующий код для события OnClick кнопки:

```
var
  Number,Code:Integer ;
begin
   if Edit1.Text<>' '
    then begin
         val( Edit1.Text,Number,Code);
         if Code <> 0
          then begin
               Edit1.SetFocus;
               MessageDlg(' Not a number in the first edit
              ',mtError,[mbOK],0); end;
        end;
end;
```

- Для события OnExit компонента Edit2 записать следующий код

```
var
  Number,Code:In
  teger;
begin
  if (Sender as
   TEdit).Text <> ' '
   then begin
        val((Sender as TEdit).Text,Number,Code);
        if Code <> 0
         then begin
              (Sender as TEdit).SetFocus;
              MessageDlg('The edit field number ' + IntToStr((Sender as
              TEdit).Tag)
              + 'does not have a valid number',mtError,[mbOK],0) ;
            end;
     end;
end;
```

Этот код можно использовать для любого компонента TEdit. Текст сообщения об ошибке можете написать свой.

- Третий компонент Edit выполняет аналогичную проверку при каждом изменении его содержимого (используя событие OnChange}.

- Компонент Edit имеет событие OnKeyPress, которое соответствует нажатию клавиши пользователем. Записать код
  для этого события компонента Edit4:

```
begin
  if not (key in['0'..'9',#8]}
   then begin
         Key:= #0;
         MessageBeep($FFFFFFFF);
       end;
end;
```

- Для события OnEnter компонента Edit5 записать код, в котором необходимо преобразовать введенные символы в число с помощью функции StrToInt. Использовать исключение для обработки ошибки EConvertError.

5. Использование компонентов

Создание редактора для текста формата RTF

Windows 95 содержит новый управляющий элемент, который способен поддерживать формат Rich Text Format (RTF). Компонент Delphi Rich Edit инкапсулирует поведение этого стандартного управляющего элемента. Поместите в форму три компонента: Panel, которая занимает верхнюю часть формы; Button, расположенная на панели; RichEdit, который занимает всю остальную часть формы. Нажатие на кнопку должно изменять шрифт выделенного фрагмента текста в окне RichEdit. Программа отображает стандартное диалоговое окно Font, использовав в качестве начального значения исходный шрифт компонента RichEdit. Затем выбранный пользователем шрифт скопируется в атрибуты текущего выбранного текста. Хотя свойства DefAttributes и SelAttributes компонента RichEdit не имеют тип TFont, они совместимы с ним, поэтому для копирования значения можно применить метод Assign:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if RichEdit1.SelLength>0 then
  begin
    FontDialog1.Font.Assign(RichEdit1.DefAttributes);
     if FontDialog1.Execute then
       RichEdit1.SelAttributes.Assign(FontDialog1.Font);
  end
  else
    ShowMessage ('No text selected');
end;
```

Задание № 5.1

Создать вышеописанную форму и записать код для события OnClick кнопки. В форму поместить компонент FontDialog.

Выбор

Существуют два стандартных управляющих элементаWindows, которые позволяют пользователю выбирать разные опции. Первый – чекбокс. Он соответствует опции, которую можно выбрать независимо от других. Второй управляющий элемент – кнопка опций, которая соответствует исключительному выбору. Например, если вы видите две кнопки опций с метками A и B, вы можете выбрать или A, или B, но не обе одновременно. Еще одна особенность множественного выбора заключается в том, что вы обязательно должны выбрать одну из опций.

Группирование кнопок опций

Кнопки опций подразумевают исключительный выбор. Однако форма может содержать несколько групп кнопок опций. Сама система Windows не способна определить как связаны друг с другом различные кнопки опций. В Windows и в Delphi эта проблема решается так: связанные кнопки опций помещаются внутри компонента-контейнера. Для совместного хранения кнопок опций – и функционального и визуального – стандартный интерфейс пользователя Windows использует управляющий элемент блока группы. В Delphi этот управляющий элемент реализован в компоненте GroupBox. Аналогичный компонент, который используется для кнопок опций, - RadioGroup, представляющий собой блок группы с нарисованными внутри него несколькими копиями кнопок опций. Компонент RadioGroup способен автоматически выравнивать свои кнопки опций и вы легко можете добавить в него новые элементы на этапе выполнения.

Правила построения блока группы с кнопками опций очень просты: разместите в форме компонент GroupBox, а затем поместите в него кнопки опций. Компонент GroupBox способен содержать другие управляющие элементы и вместе с компонентом Panel является одним из самых распространенных компонентов-контейнеров. Если вы отключите или скроете блок группы, все управляющие элементы внутри него также будут заблокированы или скрыты.

Вы по-прежнему можете обрабатывать отдельные кнопки опций, но можете работать также со всем массивом управляющих элементов, принадлежащих блоку группы. Соответствующее свойство называется Controls. другое свойство ControlCount – хранит число элементов. К этим двум свойствам можно получить доступ только на этапе выполнения.

Задание № 5.2

Выполнить построитель английских предложений по типу The book is on the table. Цель данного прмера состоит
втом, чтобы создать инструмент для построения подобных фраз путем выбора из различных доступных опций. 1) Поместите в форму компонент GroupBox с заголовком First Object и затем кнопки опций с заголовками The book, The pen, The pencil, The chair.

2) Поместите еще один компонент GroupBox с заголовком Position с опциями on, under, near.

3) Поместите компонент RadioGroup с заголовком Second Object с опциями the table, the big box, the carpet, the computer. В этом случае для создания элементов вы должны ввести список значений в свойство Items (тип TStringList).

4) Одну из кнопок опций в каждой группе нужно обязательно пометить, установив свойство Checked в True, соответствующее значение должно иметь и свойство ItemIndex группы опций (которое указывает на текущий выбор).
5) Поместите в верхнюю часть формы компонент Label, где будет отображаться построенная фраза.
6) Выберите в форме все кнопки опций (щелкая по каждой из них при нажатой клавише Shift) и введите имя метода TForm1.ChangeText в окне Object Inspector после события OnClick. Код этого метода представлен ниже.

```
procedure TForm1.ChangeText (Sender: TObject);
var
  Phrase:String;
  I:Integer;
begin
  For i:=0 to GroupBox1.ControlCount-1
   do if (GroupBox1.Controls[i] as TRadioButton).Checked
      then Phrase:= (GroupBox1.Controls[i] as
  TRadioButton).Caption; Phrase:= Phrase + ' is
  ';
  For i:=0 to GroupBox2.ControlCount-1
    do with
         GroupBox2.Controls[i] as TRadioButton do
         if Checked
           then Phrase:= Phrase + Caption;
         Label1.Caption:= Phrase + ' ' +
         RadioGroup1.Items[RadioGroup1.ItemIndex];
end;
```

Список со многими опциями

Когда вам нужно добавить много опций, кнопки опций не подходят. Для решения этой проблемы используется список. Список способен хранить в небольшом месте большое количество опций и может содержать полоску прокрутки, чтобы показывать на экране только ограниченную часть всего выбора. Другое преимущество списка состоит в том, что вы легко можете добавить в него новые элементы или удалить некоторые из текущих. Списки чрезвычайно гибки и мощны.

Еще одна важная особенность: используя компонент ListBox, вы можете осуществлять как однозначный выбор – поведение, аналогичное группе кнопок опций, - так и множественный выбор – подобно группе чекбоксов.

Задание № 5.3

Выполнить построитель английских предложений, используя компонент ListBox.

1) Поместите в центре формы компонент RadioGroup с заголовком Position и опциями on, under, near (одну опцию пометьте).
2) Слева и справа от блока группы поместите два списка и добавьте несколько строк в свойства Items обоих списков. Вы можете скопировать все строки из редактора свойств Items одного списка и вставить в редактор такого же свойства другого списка. Для удобства строки можно отсортировать, установив свойство Sorted списков в True. Не забудьте также поместить над списками пару надписей, раскрывающих их содержание.
3) В верхнею часть формы поместите компонент Label, в котором будет отображаться построенная фраза.
4) В нижней части формы поместите поле редактирования с надписью и кнопку с заголовком Add . В поле редектирования будет вводиться строка текста, которую необходимо добавить в оба списка по нажатию кнопки Add.
5) Чтобы первоначально выбрать по строке из каждого списка, записать метод для события OnCreate формы: **procedure**

```
TForm1.FormCreate(Sender:TObject);
var
  N:Integ
  er;
begin
```

```
  N:=
  ListBox1.Items.IndexOf('book'
  ); ListBox1.ItemIndex:= N;
  N:=
  ListBox2.Items.IndexOf('table
  '); ListBox2.ItemIndex:= N;
end;
```

6) Создать процедуру ChangeText, соединенную с событиями OnClick группы опций и двух списков. Чтобы получить из списка выбранный текст, вы должны только взять номер выбранного элемента (хранится в свойстве этапа выполнения ItemIndex) и затем отыскать строку в соответствующей ячейке массива Item (ListBox1. Items [ListBox1. ItemIndex]).

7) Добавлять строку String1 в список с помощью метода Add класса TStrings – **`ListBox1.Items.Add(String1).`**

6. Создание и обработка меню.

Структура меню

Обычно меню имеет два уровня. Полоска меню, которая находится под заголовком окна, содержит имена выпадающих меню. Каждое выпадающее меню содержит несколько элементов. Однако структура меню очень гибка. Элемент меню можно поместить непосредственно в полоску, а выпадающее меню внутрь другого выпадающего меню. Выпадающее меню внутри другого выпадающего меню (выпадающее меню второго уровня) встречается довольно часто,
и для этого случая Windows предоставляет специальный визуальный значок – маленький треугольник справа от меню. Нередко вместо выпадающего меню второго уровня вы можете просто сгруппировать ряд опций в первоначальном выпадающем меню и поместить два разделителя: один до группы и один после.

Различные роли элементов меню

Существует три основных типа элементов меню:

- Команды – элементы меню, которые используются для выдачи команды и выполнения действия. Визуально они никак не выделяются.
- Установщики состояния – элементы меню, которые используются для переключения опции в положения включено – выключено и изменения состояния какого – либо элемента. Если эти команды имеют два состояния, то в активном положении слева от них обычно стоит галочка. В этом случае выбор команды изменяет состояние на противоположное.
- Элементы вызова диалога – элементы меню, которые вызывают диалоговою панель. Реальное различие между этими и другими элементами меню состоит в следующем: с помощью этих элементов пользователь должен получить возможность исследовать вероятные действия соответствующей диалоговой панели. Такие команды должны иметь визуальный ключ в виде трех точек после текста.

Редактирование меню с помощью Menu Designer

Система Delphi включает специальный редактор для меню Menu Designer. Чтобы вызвать этот инструмент, поместите компонент меню в форму и щелкните по нему. Не волнуйтесь о точном положении данного компонента в форме, поскольку на результат это не влияет : само меню всегда помещается правильно – под заголовком формы. Menu Designer позволяет создавать меню путем простого написания текста команд, перемещать элементы или выпадающие меню с помощью буксировки и легко устанавливать свойства элементов. Для создания выпадающего меню второго уровня нужно выбрать команду Create submenu в SpeedMenu инструмента (локальном меню, которое вызывается правой кнопкой мыши).

Горячие клавиши меню

Общее свойство элементов меню – наличие подчеркнутой буквы. Эту букву можно использовать для выбора меню
с помощью клавиатуры. При одновременном нажатии клавиши Alt и клавиши с буквой на экране появляется соответствующее выпадающее меню. Чтобы определить подчеркнутую клавишу, просто поместите перед ней символ амперсанта (&), например &File. Элементам меню можно назначить горячие клавиши. Для этого нужно указать значение для свойства ShortCut, выбрав одну из стандартных комбинаций.

Задание № 6.1

1) С помощью Menu Designer добавьте в форму полоску меню. Эта полоска имеет три выпадающих меню: меню File с единственной командой Exit; меню Options с командами Font, Color, Left, Center, Hight (между Color и Left установите разделитель, для Left, Center, Hight назначьте горячие клавиши); меню Help с элементом About. Чтобы поместить разделитель, вместо текста команды вставьте просто дефис.
2) В форму поместите компонент RichEdit и две пиктограммы диалогов FontDialog и ColorDialog.
3) Для отклика на команды меню вы должны определить метод для события OnClick каждого элемента меню. Код метода TForm1.Font1Click :

```
procedure TForm1.Font1Click(Sender:TObject);
begin
  FontDialog1.Font:= RichEdit1.Font;
  FontDialog1.Execute;
  RichEdit1.Font:= FontDialog1.Font;
end;
```

Код метода TForm1.Color1Click аналогичен предыдущему.
Код метода TForm1.Left1Click :

```
procedure TForm1.Left1Click(Sender:TObject);
begin
  RichEdit1.Paragraph.Alignment:= taLeftJustify;
  Left1.Checked:= True;
  Center1.Checked:= False;
  Right1Checked:= False;
end;
```

Чтобы поставить галочку в ряде выбираемых опций, установите свойство Checked элемента меню в окне Object Inspector в True.

4) Код методов для элементов Center и Right аналогичен предыдущему.

Изменение элементов меню

Для модификации элемента меню чаще всего используются три свойства. Свойство Checked используется, чтобы добавить или удалить галочку рядом с элементом меню. С помощью свойства Enabled элемент меню можно пригасить, после чего пользователь не сможет его выбрать. Последнее свойство этой группы Caption представляет текст элемента меню. Изменяя текст элемента меню, вы указываете пользователю, что программа перешла в другое состояние.

Задание № 6.2

1) На форме расположить две панели , две кнопки и компонент RichEdit. Первая панель содержит два поля редактирования, а вторая два чекбокса. Также необходимо построить выпадающее меню File, Buttons, View, Pulldowns. Команды меню File – Open , SaveAs. Команды меню Buttons содержит изменяемый текст (с ‘Enable First’ на ‘Disable First’). Команды меню View – Edit Boxes, Check Boxes. Команды Pulldowns – Remove File, Disable Buttons, Disable View. Поместите в форму пиктограммы необходимых диалогов.
2) Код методов, которые загружают и сохраняют файлы :

```
procedure TForm1.Open1Click(Sender:TObject);
begin
  if OpenDialog1.Execute
    then RichEdit1.Lines.LoadFromFile OpenDialog1.FileName);
end;

procedure TForm1.SaveAs1Click(Sender:TObject);
begin
  if SaveDialog1.Execute
    then RichEdit1.Lines.SaveToFile(SaveDialog1.FileName);
end;
```

3) Компоненты внутри панелей в действительности не используются. Однако вам необходимо воспользоваться двумя кнопками, чтобы скрыть или отобразить каждую из двух панелей вместе с управляющими элементами, которые в них содержатся. Те же действия можно выполнить с помощью двух команд меню : View/ Edit Boxes и View/ Check Boxes. Когда вы выбираете одну из этих команд меню или нажимаете одну из кнопок, происходит три разных действия. Во-первых, отображается или скрывается панель. Во-вторых, текст кнопки изменяется с Hide на Show, и наоборот. В-третьих, рядом с соответствующим элементом меню появляется или исчезает галочка. Ниже приведен код одного из двух методов, который связан с событиями щелчка как команды меню, так и кнопки :

```
procedure TForm1.ViewEdit1Click(Sender:TObject);
begin
  Panel1.Visible:= not Panel1.Visible;
  ViewEdit1.Checked:= not ViewEdit1.Checked;
  if Panel1.Visible
    thenButton1.Caption:=‘Hide’;
      else Button1.Caption:= ‘Show’;
end;
```

4) Команды меню Buttons применяют другой подход. Для показа текущего состояния они используют не галочку, а изменение текста. Кроме того, они разрешают или запрещают соответствующую команду View и кнопку.

```
procedure TForm1.ButtonsFirst1Click(Sender:TObject);
begin
  if Buttons1.Enabled
    then begin
          Buttons1.Enabled:= False;
          ViewEdit1.Enabled:= False;
          ButtonsFirst1.Caption:= 'Enable &First';
       end
     else begin
          Buttons1.Enabled:= True;
          ViewEdit1.Enabled:= True;
          ButtonsFirst1.Caption:= 'Disable &First';
         End;
end;
```

5) Команды меню Pulldowns должны скрывать выпадающее меню указанные в элементах и показывать галочку для выбранного элемента. Запишите код для каждого элемента этого меню самостоятельно.

7. Получение ввода от мыши. Рисование в форме.

Когда пользователь нажимает одну из кнопок мыши, указатель которой находится над формой (или над компонентом), Windows посылает приложению несколько сообщений. Для написания кода, откликающегося на эти сообщения, Delphi определяет несколько событий. Основных событий два : OnMouseDown, которое происходит при нажатии одной из кнопок мыши, и OnMouseUp, которое происходит при освобождении кнопки.

Еще одно важное системное сообщение связано с перемещением мыши – сообщение OnMouseMove. Событие OnClick также доступно и в форме. Его основной смысл состоит в том, что левая кнопка мыши нажимается и отпускается над одним и тем же окном или компонентом. Однако в период между этими двумя действиями курсор может переместиться за пределы окна или компонента, причем левая кнопка мыши будет удерживаться нажатой. Если вы в определенной позиции нажмете кнопку мыши, а затем переместите мышь в другое место и отпустите кнопку, то щелчок не произойдет. В этом случае окно получает только сообщение о нажатии, несколько сообщений о перемещении и сообщение об освобождении.

События, связанные с мышью

Метод, соответствующий событию OnMouseDown имеет несколько параметров:

```
procedure
  TForm1.FormMouseDown(Sender:TObject;Button:TMouseButton;Shift:TShift
  State; X,Y:Integer ) ;
```

Кроме обычного параметра Sender здесь присутствуют еще четыре :

1) Button – показывает, какая из трех кнопок мыши была нажата. Возможные значения : mbRight, mbLeft, mbCenter.
2) Shift – показывает, какие влияющие на мышь клавиши были нажаты при возникновении события. Такой клавишей может быть Alt, Ctrl или Shift, нажатая вместе с самой кнопкой мыши. Данный параметр имеет тип множества, т.к. несколько клавиш могут быть нажаты одновременно. Это означает, что при анализе условия вы должны применять не проверку на равенство, а оператор in.
3) X и Y – показывают координаты позиции мыши относительно клиентской области.

Рисование в форме

Canvas (холст) – это область рисунка в форме и многих других графических компонентах.

Чтобы получить доступ

кпикселам формы, используйте свойство формы Canvas и свойство Pixels для Canvas. Свойство Pixels – это двумерный массив, соответствующий цветам отдельных пикселов в Canvas. Canvas.Pixels[10,20] соответствует цвету пиксела, который находится на 10 пикселов правее и на 20 пикселов ниже точки отсчета. Обращайтесь с

массивом пикселов как с любым другим свойством; чтобы изменить цвет пиксела присвойте новое значение. Чтобы определить цвет пиксела – прочитайте значение.

Каждое свойство Canvas имеет воображаемое перо для рисования линий и контуров. Свойство Pen (перо) определяет цвет и размер линий и границ фигур. Свойствами пера являются его цвет, размер (если это сплошная линия) или стиль. Работая с пером, вы можете прочитать (но не изменить) его текущую позицию (свойство PenPos). Позиция пера определяет исходную точку следующей линии, которую программа может нарисовать с помощью метода LineTo. Для изменения позиции вы можете применить метод MoveTo канвы.

Свойство Brush (кисть) определяет цвет очерченой поверхности. Кисть используется для закрашивания замкнутых фигур. Свойствами кисти являются ее цвет, стиль и иногда растровое изображение.

Свойство Font определяет шрифт, который используется методом холста TextOut для написания текста в форме. Шрифт имеет имя, размер, стиль, цвет и т.п.

Задание № 7.1

1) Поместить в форму меню Color с командами PenColor и BrushColor, которые будут соответственно изменять цвет пера и кисти с помощью стандартной диалоговой панели.
2) В форме реализовать рисование окружностей, эллипсов и прямоугольников различных размеров и цветов с помощью мыши, используя свойство Canvas формы. Дальше даются подсказки для реализации этой задачи.
3) Запишите следующий код для события OnMouseDown :

```
if Button = mbLeft
 then begin
      Center.X:= X;
      Center.Y:= Y;
      if ssShift in Shift
       then Circle:= False
         elseCircle := True;
    end;
```

Поле формы Circle типа Boolean определяет вид фигуры, значения координат центра фигуры записываются в поля формы Center типа TPoint.

4) Запишите следующий код для события OnMouseUp :

•

```
Radius.X:=abs(Center.X-X);
Radius.Y:=abs(center.Y-Y);
if Circle
  then Canvas.Ellipse(Center.X-Radius.X,Center.Y-
      Radius.Y,Center.X+Radius.X,Center.Y+ Radius.Y)
   else Canvas.Rectangle(Center.X-Radius.X,Center.Y-
Radius.Y,Center.X+Radius.X, Center.Y+ Radius.Y);
```

5) Запишите следующий код для события OnMouseMove :

```
Caption:= Format('Координаты: x=%d, y=%d ',[X,Y]);
```

6) Запустите приложение. Если все сделано правильно, то вы будете наблюдать изменение координат в заголовке формы при продвижении мыши; сможете рисовать окружности и эллипсы нужного размера (щелкая кнопкой и удерживая ее при перемещении мыши по горизонтали и вертикали); сможете рисовать прямоугольник нужного размера, используя ту же технологию.

Черчение и рисование в системе Windows

- ♦ Черчение – вы обращаетесь к канве и вызываете некоторые ее методы. Поскольку изображение не сохраняется, форма может частично или целиком потерять свое содержимое (при закрытии окна формы другим окном или при уменьшении размера окна формы).
- ♦ Рисование – это технология, которая позволит приложению перерисовывать всю ее поверхность при любых возможных условиях.

  Для вызова перерисовки можно использовать методы Invalidate, UpDate, ReFresh и Repaint.

Задание № 7.2

1) Начиная с оператора if., код для события OnMouseUp перенести в код для события OnPaint.
2) В код для события OnMouseUp вставить в конце вызов метода Invalidate (который вызывает косвенно метод FormPaint, связанный с событием OnPaint).
3) Запустить приложение. При правильном выполнении всех инструкций, в форме будет рисоваться только одна фигура.
4) Выполните первое задание для компонента PainBox.
5) Изучите возможности компонента Shape.

8. Инструментальная линейка и строка состояния

Во многих приложениях Windows имеются инструментальная линейка в верхней части окна и строка состояния в его нижней части. Инструментальная линейка

содержит обычно несколько маленьких кнопок, которые вы можете нажимать щелчком мыши, чтобы задавать команды или переключать опции вкл. и выкл. Иногда инструментальная линейка может содержать комбинированный список, строку редактирования или некоторые другие управляющие элементы.

Строка состояния обычно имеет одну или несколько областей с текстовым описанием текущего состояния программы. У вас может быть область для координат, для показа выбранного шрифта или отображения всплывающих подсказок о том, что делать дальше, сообщений об ошибках и т.д. Фактически приложение определяет, что должно войти в строку состояния.

Построение инструментальной линейки

Для создания инструментальной линейки или строки состояния в Delphi вы можете использовать компонент Panel, Добавив в него несколько кнопок или других панелей, или же можете использовать специальный компонент StatusBar.

Чтобы построить типичную инструментальную линейку, вам нужно поместить панель в верхней части формы и разместить в ней несколько компонентов SpeedButton (быстрая кнопка). Быстрые кнопки могут иметь заголовок и значок, хотя обычно они имет только графический элемент. Быстрые кнопки могут вести себя подобно командным кнопкам, чекбоксам или кнопкам опций (радиокнопкам) и иметь другие растровые изображения для различных ситуаций. Быстрые кнопки являются графическим элементом, они не имеют дескриптора окна (т.о. не использует ресурсы окон), не могут получать фокус, не участвуют в переборе клавишей Tab и быстрее создаются и закрашиваются.

Если вы просто выбираете компонент SpeedButton и помещаете его экземпляр в панель, то в результате получите графическую командную кнопку. Потом можно выбрать растровое изображение или ввести заголовок и записать код для события OnClick. Чтобы добавить группу быстрых кнопок, которые будут работать подобно радиокнопкам, поместите их
в панель и присвойте их свойствам GroupIndex одинаковое значение. Все кнопки, имеющие одинаковое свойство GroupIndex, станут осуществлять взаимоисключающий выбор аналогично кнопкам опций. Одна из этих кнопок должна быть всегда выбранной, поэтому не забывайте присвоить свойству Down значение True для одной из них на этапе проектирования или как только программа начинает работать.

Вкачестве альтернативы у вас могут быть взаимоисключающие кнопки, которые все могут быть ненажатыми. Это означает, что вы можете щелкнуть на выбранной кнопке и отменить ее выбор. Вы можете выбрать этот режим, присвоив свойству AllowAllUp для всех кнопок группы значение True. Чтобы быстрая кнопка работала как чекбокс необходимо выполнить следующее. Чекбокс – это группа только с одним эелементом, в котором все кнопки могут быть невыбранными. На практике вы добиваетесь этого добавив новую быструю кнопку, присвоив ей конкретное значение для свойства GroupIndex (отличное от индексов других групп, и выбрав значение True для свойства AllowAllUp..

Задание № 8.1

Постройте инструментальную линейку, в которой будут располагаться следующие кнопки:

1) командная, при нажатии которой будет издаваться звук;
2) группа из трех кнопок, которые будут осуществлять выравнивание текста в компоненте Label по левому, правому краю и по центру и вести себя как кнопки опций;
3) группа из трех кнопок, которые будут изменять стиль текста компонента Label (полужирный, курсив, подчеркнутый) и могут быть все ненажатыми – в этом случае текст нормальный (вы должны не просто выбрать стиль, когда нажата кнопка , но и отменить ее выбор, восстанавливая нормальный стиль, когда кнопка освобождена. В методах,
   отвечающих за событие щелчка этих кнопок, вы должны написать:

```
procedure TForm1.SpeedButton1Click(Sender:TObject);
begin
  if SpeedButton1.Down
    then Label1.Font.Style:= [fsBold]
     else Label1.Font.Style:= []
end;
```

4) кнопка, которая будет изменять размер шрифта с 24 на 12 и вести себя как чекбокс, т.е. при нажатой кнопке размер шрифта 24, при отжатой – 12.
5) Добавьте в форму меню, с помощью которого можно будет отключать и включать различные кнопки используя свойство Enabled , скрывать и показывать линейку с помощью свойства Visible панели.

Добавление всплывающих подсказок в инструментальную линейку

Всплывающая подсказка – текст, который кратко описывает быструю кнопку под курсором. Этот текст обычно выводится на экран в окне желтого цвета, после того как курсор мыши оставался неподвижным над кнопкой в течение некоторого времени. Для добавления всплывающих посказок в инструментальную линейку приложения просто устанавливают значение свойства ShowHints панели, которая используется как инструментальная линейка, равным True. Вы также можете изменить значение по умолчанию свойств HintColor и HintPause глобального объекта Application и, что необязательно, добавить непосредственно конкретный текст для всплывающей подсказки кнопок инструментальной линейки (свойство Hint). Текст подсказок можно изменять при изменении состояния кнопки написав некоторый код, использующий свойство Hint.

Комбинированный список в инструментальной линейке

Существует несколько распространенных приложений, которые используют комбинированные списки в инструментальных линейакх для того, чтобы показывать списки стилей, шрифтов, размеров шрифта и т.д. Используем в следующем задании такой подход: создадим комбинированный список, чтобы позволить пользователю выбирать шрифт и окно редактирования с прокруткой компонент (SpinEdit) для установки размера шрифта, каждый с собственной меткой.

Два новых компонента и их метки имеют соответствующее сообщение в свойстве Hint. Комбинированный список заполняется, когда начинает работать приложение, копируя названия текущих шрифтов из объекта Screen.

```
procedure TForm1.Form1Create (Sender:TObject);
var
  I:Integer;
begin
 for I:= 1 to Screen.Fonts.Count do
   {скопируйте название шрифта в комбинированный список}
   ComboBox1.Items.Add Screen.Fonts.Strings[I-1]);
   {выберите текущий шрифт}
   ComboBox1.ItemIndex:=ComboBox1.Items.IndexOf(Label1.Font.Name);
end;
```

Когда выбирается новый элемент комбинированного списка – текст текущего элемента комбинированного списка копируется в название шрифта надписи.

```
procedure TForm1.ComboBox1Change(Sender:TObject);
begin
 Label1.Font.Name:= ComboBox1.Items[ComboBox1.ItemIndex];
end;
```

Для управляющего элемента редактирования с прокруткой :

```
procedure TForm1.SpinEdit1Change Sender:TObject);
begin
 if not (SpinEdit1.Text = ' ')
   then Label1.Font.Size:= SpinEdit1.Value;
end;
```

Задание № 8.2

1) Из формы, созданной в предыдущем примере, удалить кнопку изменения размера шрифта
2) Согласно вышеизложенному поместить в форму комбинированный список, чтобы позволить пользователю выбирать шрифт и окно редактирования с прокруткой компонент (SpinEdit) для установки размера шрифта, каждый с собственной меткой.
3) Записать код для добавленных компонентов.

Построение строки состояния

Система Delphi Содержит специальный компонент StatusBar, основанный на специальном управляющем элементе системы Windows 95. Этот коипонент может быть использован почти как панель, когда значением его свойства SimplePanel является True. В этом случае вы можете использовать свойство SimpleText, чтобы вывести некоторый текст. Этот компонент позволяет вам определить несколько подпанелей с помощью редактора свойства Panels. Каждая подпанель имеет свои собственные графические атрибуты, настраиваемые с помощью редактора. Другой характеристикой компонента строки состояния является специальная область, которая добавляется в нижний правый угол линейки и полезна для изменения размера самой формы.

Строка состояния используется для многих целей. Наиболее часто ее используют для вывода на экран информации об элементе меню, который в настоящее время выбран пользователем. При этом нужно сделать два шага. Сначала ввести строку как свойство Hint каждого элемента меню. Затем записать некоторый код для обработки события OnHint приложения. Вам нужно добавить вручную новый метод в форму и затем его присвоить свойству OnHint объекту Application при запуске :

```
procedure TForm1.Form1Create(Sender:TObject);
begin
  Application.OnHint:= ShowHint;
End;
```

Винтерфейсной части кода формы вы можете добавить следующее определение **procedure TForm1.ShowHint(Sender: Object);**

Эта процедура копирует текущее значение свойства Hint приложения, которое временно содержит копию всплывающей подсказки выбранного элемента, в строку состояния :

```
procedure
TForm1.ShowHint(Sender:TObjec
t); begin
  StatusBar1.Panels[0].Text:=Application.Hint;
end;
```

Задание № 8.3

Сконструировать строку состояния, в которую вывести подсказки для всех команд всплывающего меню формы, построенной в предыдущих заданиях.

9. Приложение с несколькими формами. Многостраничные формы.

Обычно приложения имеют основное окно, несколько плавающих инструментальных панелей (или палитр) и диалоговых панелей, которые могут вызываться с помощью команд меню или командных кнопок.

Диалоговые панели в Delphi основаны на формах. С диалоговой панелью пользователь обычно связывает понятие модального окна. Модальное окно – это такое окно, которое получает фокус и должно быть закрыто прежде, чем пользователь может перейти обратно к основному окну. Это верно для панелей сообщения и диалоговых панелей. В системе Delphi можно также иметь немодальные диалоговые панели и модальные формы. Мы должны учитывать два момента:

1) рамки формы и пользовательский интерфейс определяют, выглядит ли эта форма как диалоговая панель
2) использование двух функций – Show и ShowModal для вывода на экран второй формы определяет поведение последней (немодальная или модальная).

Если вы используете функцию Show, вторая форма будет выведена на экран как немодальная. Код просто выводит форму на экран, а не создает ее. Форма создается файлом проекта. Первая созданная форма становится основной формой приложения. Когда модальная форма создается и выполняется с помощью функции ShowModal, она остается активной, до тех пор пока вы не закроете ее. Функция ShowModal не возвращает значение до тех пор, пока форма не закрыта. В это время основная форма остается недоступной. Как только модальная форма будет закрыта, функция ShowModal завершает работу, а код удаляет объект из памяти.

Добавление второй формы в программу

Для добавления второй формы можно нажать кнопку New Form или использовать команду File/New, переместиться на страницу Forms или Dialogs и выбрать один из доступных шаблонов формы. Если у вас в проекте две формы, используйте кнопку Select Form или Select Unit инструментальной линейки, чтобы управлять формами. Вы также можете выбрать, какая форма является основной и какие формы нужно создать автоматически при запуске, используя страницу Forms диалоговой панели Project Options. Чтобы компилировать код первой формы, надо включить модуль, содержащий вторую форму (Unit2), с помощью оператора uses в список модулей в начале кода. Или можно выбрать первую форму и выполнить команду File/Use Unit. Чтобы закрыть вторую форму, вы можете использовать ее системное меню или нажать кнопку Close, помещенную в форму. При этом Delphi не закрывает вторую форму, а лишь скрывает ее, поэтому всегда есть возможность вывести на экран вторую форму.

Создание диалоговой панели

Чтобы построить диалоговую панель вместо формы, выберите значение bsDialog для свойства BorderStyle формы. После этого форму можно вывести на экран как модальное или немодальное окно, используя Show или ShowModal. Модальные диалоговые панели распространены больше, чем немодальные. Модальных форм следует избегать, т.к. пользователь не захочет иметь с ними дело. Вот один из подходов использования диалоговой панели :

1) Устанавливать начальные значения, каждый раз когда вы исполняете диалоговую панель.
2) Показывать диалоговую панель.
3) Если была нажата кнопка OK, скопировать новые значения обратно в форму.

Задание № 9.1

1) Подготовить две формы, в каждую из которых поместить по кнопке. В первой форме кнопка используется, чтобы показать второе окно, во второй форме – чтобы себя закрыть. Чтобы выполнить вторую форму запишите код:

```
procedure TForm1.Button1Click(Sender:TObject);
begin
  Form2.Show;
end;
```

2) Создадим форму, которая имеет две кнопки, используемые для создания модальной и немодальной форм. Как только вы создаете две новые формы ModalForm и ModalessForm, можно написать следующий метод, соответствующий событию OnClick одной из двух кнопок:

```
procedure TForm1.ModalButtonClick(Sender:TObject);
var
  Modal:TModalForm;
begin
   Modal:=TModalForm.Create Application);
   Modal.ShowModal;
   Modal.Free;
end;
```

Код для другой кнопки аналогичен, с той лишь разницей, что используется функция Show. Запишите его самостоятельно.

3) В форму поместить две надписи и кнопку. При нажатии кнопки должна появляться диалоговая панель с двумя растровыми кнопками OK , Cancel и два чекбокса, которые позволяют показывать и скрывать надписи формы. Записать следующий код :

```
procedure TForm1.Button1Click(Sender:TObject ;
var
  old1,old2:Boolean;
begin
  old1:= Form2.CheckBox1.Checked;
  old2:= Form2.CheckBox2.Checked;
  if (Form2.ShowModal = mrOk)
    then begin
          Label1.Visible:= Form2.CheckBox1.Checked;
          Label2.Visible:= Form2.CheckBox2.Checked;
        end
     else begin
           Form2.CheckBox1.Checked:= old1;
           Form2.CheckBox2.Checked:= old2;
         end;
end;
```

Построение блокнотов

Компоненты PageControl, TabControl (TabSet) и TabSheet используются для обработки страниц и ярлычков. TabSheet используется внутри компонента PageControl. Компонент TabControl (TabSet) используется для построения автономных ярлычков (не связанных со страницами). С помощью компонента PageControl можно построить приложение или диалоговую панель, используя понятие многостраничности или блокнота. Для добавления новых страниц (табличных листов) используется локальное меню компонента PageControl. Каждый объект TabSheet имеет собственный заголовок, который выводится на экран как ярлычок. Переключение между страницами на этапе проектирования можно производить с помощью локального меню компонента PageControl или щелчком на ярлычке. Если вы поместите компонет на странице блокнота, то он будет доступен только на этой странице. Чтобы использовать один компонент для всех страниц, его необходимо поместить в форму вне компонента PageControl (или до выравнивания его с областью клиента) и затем переместить его на передний план, вызвав команду Bring to Front из локального меню формы. Допустим мы поместили в форму две кнопки, которые позволяют передвигаться по страницам : Next, Previous – которые представляют альтернативу использования ярлычков. Код, связанный с одной из кнопок :

```
procedure TForn1.BitBtnClick(Sender:TObject);
begin
  PageControl1.SelectNextPage(true);
end;
```

Другая кнопка вызывает ту же процедуру, передавая значение false в качестве параметра для выбора предыдущей страницы. Свойство SelectNextPage рассматривает последнюю страницу как единственную перед первой и перемещение происходит непосредственно между этими страницами. Существует третий метод, доступный для изменения страниц (после ярлычков и кнопок). Компонент список (ListBox) заполняется с помощью метода FormCreate путем копирования заголовка каждой страницы (свойство Page хранит список объектов TabSheet).

```
for I:= 0 to PageControl1.PageCount-1
  do ListBox1.Items. Add( PageControl1.Page[I].Caption;
```

Блокнот с набором ярлычков

Стандартный подход к построению блокнота с ярлычками в Delphi 1 использует два раздельных компонента NoteBook и TabSet. На этапе проектирования можно работать на страницах блокнота, изменяя значение его свойства PageIndex ; вы не можете щелкать на ярлысках, как с компонентами PageControl. Как только вы вводите новое значение для свойства PageIndex или ActivePage в Object Inspector, соответственно изменяется видимая страница блокнота. Можно также выбрать локальное меню блокнота, в котором имеются команды для перемещения к следующей и предыдущей странице.

Альтернативой использования PageControl, TabSet и NoteBook является использование компонента TabbedNoteBook.

Задание № 9.2

1) Поместите в форму компонент NoteBook, а затем компонент TabSet и выровняйте последний с нижней частью формы (alButtom), а компонент NoteBook с клиентской

областью (alClient). Присвойте названия страницам блокнота, выбирая свойство Pages и вводя некоторые значения в соответствующем редакторе. Подготовить ярлычки с помощью ввода некоторых строк для свойства Tabs.

2) Чтобы соединить блокнот с множеством ярлычков нужно записать код для события OnChange компонента TabSet. **procedure TForm1.TabSet1Change(Sender:TObject;NewTab:Integer; var AllowChange:Boolean); begin**

```
  NoteBook1.PageIndex:= NewTab;
end;
```

Вы можете активизировать страницу, используя также их названия (свойство ActivePage блокнота) вместо индекса (свойство PageIndex). Это работает, если только названия страниц совпадают с названиями ярлычков.

```
NoteBook1.ActivePage:=TabSet1.Tabs[NewTab];
```

Задание № 9.3

1) Поместите компонент TabbedNoteBook в форму. Создайте три страницы. Поместите кнопку Close, которая должна присутствовать на всех страницах.
2) На первой странице блокнота поместите список (ListBox) с названиями страниц. Чтобы изменить страницы в блокноте с помощью ListBox написать следующий код :

```
procedure TForm1.ListBox1Click(Sender:TObject);
begin
  TabbedNoteBook1.PageIndex:=ListBox1.ItemIndex;
end;
```

3) На второй странице блокнота поместите компонент FontDialog и кнопку для изменения шрифта. Чтобы изменить шрифт, когда пользователь щелкнет по соответствующей кнопке запишите код:

```
if FontDialog1.Execute
   then TabbedNoteBook1.TabFont:= FontDialog1.Font;
```

Задание № 9.4

1) В форму поместим компонент DriveComboBox и DirectoryListBox. На правой стороне формы поместить компонет Image и под ним TabSet.
2) Когда пользователь выбирает новый каталог, ярлычки должны сразу вывести на экран имена файлов с растровыми изображениями в ярлычки каждый раз, когда изменяется выбор в списке каталога. Для этого поместите компонент FileListBox в форму и поместите его позади других компонентов или присвойте его свойству Visible значение False. Затем измените свойство Mask списка файлов на *.bmp. Для того чтобы связать компонент DirectoryListBox с компонентом FileListBox запишите в окне Object Inspectorдля свойства DirectoryListBox следующее имя компонента:

   FileListBox1. Теперь добавьте метод для обработки события OnChange списка каталога

```
procedure TForm1.DirectoryListBox1Change(Sender:TObject);
begin
  with FileListBox1
    do if Item.Count = 0
        then begin
             TabSet1.Tabs.Clear;
             Image1.Visible:= False;
             TabSet1.Tabs.Add('None');
           end
         else begin
              Image1.Visible:= True;
              TabSet1.Tabs:= FileListBox1.Items;
             end;
end;
```

3) Когда ярлычки выводят на экран имена файлов, для загрузки соответствующего растрового изображения в компонент Image используется следующий код :

```
procedure
TForm1.TabSet1Change(Sender:TObject;NewTab:Integer;var
AllowChange:Boolean); begin
   if TabSet1.Tabs[NewTab] <> 'None' then
TabSet1.Tabs[NewTab]; end;
```