Разработка приложений в среде *LAZARUS*

Lazarus – это бесплатная и свободная графическая среда разработки программного обеспечения на языке *Object Pascal* для компилятора *Free Pascal Compiler* (*FPC*), также свободного программного обеспечения. Оба продукта распространяются под лицензией *GPL* (*General Public License* – универсальная общественная лицензия *GNU*, универсальная общедоступная лицензия *GNU* или открытое лицензионное соглашение *GNU*).

Lazarus является кросплатформенной средой разработки, т. е. позволяет создавать программы в графическом окружении, практически неизменном, для широкого класса операционных систем: *Linux*, *FreeBSD*, *Mac OS X*, *Microsoft Windows*, *Android*. Для создания исполняемой программы достаточно снова собрать ее в соответствующей операционной системе.

3.2.1 Общая характеристика среды

Среда разработки базируется на библиотеке визуальных компонентов *Lazarus Component Library*, которая содержит достаточное число компонент, позволяющих создавать формы при помощи визуального проектирования графического интерфейса пользователя (*GUI*, англ. *graphical user interface*). В *Lazarus* также возможно разрабатывать консольные приложения и динамически подгружаемые библиотеки: в *Windows dynamic- link library*, или *DLL*. Динамические библиотеки используются для разработки других программ.

При запуске *Lazarus* после установки на рабочем столе появляется набор несвязанных окошек. Первое, расположенное в самом верху рабочего стола, имеет название *Lazarus Editor – projectl* (название зависит от используемой версии и названия открытого проекта). Это главное окно управления проектом и оно содержит «Главное меню» и «Палитру Компонентов» (*Component Palette*).

Строкой ниже располагается «Главное меню» с обычными пунктами *File*, *Edit*, *Search*, *View* и некоторыми специфичными для *Lazarus*. Ниже слева располагается набор кнопок,

предоставляющих быстрый доступ к некоторым функциям главного меню, и справа – «Палитра Компонентов».

Под окном редактора *Lazarus* слева располагается окно «Инспектора Объектов», а справа от него – «Редактор Исходного кода» (*Lazarus Source Editor*). Может присутствовать и другое окно меньшего размера, озаглавленное *Form1*, расположенное поверх «Редактора Исходного Кода».

Если его в данный момент не видно, то можно переключиться к нему, нажав клавишу *F*12, которая позволяет переключаться между «Редактором Исходного Кода» и «Окном Формы».

Окно формы – это место, где разрабатывают графический интерфейс программы, а в «Редакторе Исходного Кода» отображается разрабатываемый *Pascal*-код приложения.

Когда начинается разработка нового проекта, по умолчанию создается стандартная форма. Если щелкнуть мышкой в любом месте формы, то можно увидеть ее свойства в «Инспекторе Объектов» у левого края экрана.

Другие окна, которые могут появиться в процессе работы: «Инспектор Проектов», содержащий сведения о файлах, включенных в проект, и позволяющий вам добавлять и удалять файлы из проекта; окно «*Messages*» (Сообщения), отображающее сообщения компилятора, ошибки и отчеты по проекту.

Главное текстовое меню содержит следующие пункты: *File*, *Edit*, *Search*, *View*, *Project*, *Run*, *Components*, *Tools*, *Environment*, *Windows*, *Help*. И в локализованном (русифицированном) варианте – Файл, Правка, Поиск, Вид, Проект, Запуск, Пакет, Сервис, Окружение, Окно, Справка.

Любой пункт можно выбрать, если навести на него курсор и нажать левую кнопку мыши или использовать горячие клавиши.

Инспектор объектов (Object Inspector): окно, отображающее возможности текущей формы. Чуть выше располагается окно, содержащее древовидную структуру текущего проекта. Нижняя, главная панель имеет две вкладки: *Properties* (свойства) и *Events* (События).

Редактор исходного кода (Source Editor): основное окно для редактирования исходных текстов. «Функции» и вид «Редактора исходного кода» имеют настройки, вызываемые из основного меню выбором.

После подготовки кода программы выполняю запуск компилятора и сборки проекта.

Собрать (Build): запускается сборка (т. е. компиляция) любых файлов проекта, которые были изменены со времени последней сборки.

Собрать все (Build all): запускается сборка всех файлов проекта, независимо от наличия изменений.

Быстрая компиляция (Quik Compile): быстрая компиляция с поиском ошибок.

Прервать сборку (Abort build): останавливает процесс сборки на ходу.

Запуск (Run): это обычный путь запуска компилятора, и если компиляция прошла успешно, запускается приложение.

Приостановить (Pause): задерживается выполнение работающей программы. Это позволяет проверить промежуточные результаты. Выполнение программы можно продолжить повторным выбором «Запуск».

Показать точку исполнения (*Show execution point*).

Шаг со входом (Step into): применяется совместно с отладчиком, запуская программу на один шаг вплоть до точки, помеченной в исходном тексте.

Шаг в обход (Step over): вызывает пошаговое выполнение вплоть до помеченного оператора, пропускает его и продолжает работу с нормальной скоростью.

Запуск до курсора (Run to cursor): запускает выполнение, пока не встретится оператор, где находится курсор, и произойдет остановка. Выполнение продолжится с нормальной скоростью после выбора «Запуск».

Останов (Stop): прекращается выполнение программы. Продолжить выполнение выбором «Запуск» (*Run*) нельзя, но можно запустить сначала.

Параметры Запуска (Run Parameters): открывается многостраничное окно, позволяющее передать программе опции командной строки и параметры; настроить отображение выполняемой программы; изменить некоторые переменные системного окружения.

Сброс отладчика (Reset debugger): отладчик приводится в исходное состояние, так что точки останова, значения переменных и т. д. будут «забыты».

Собрать файл (Build file): происходит компиляция только файла, открытого в «Редакторе».

Запустить файл (Run file): компиляция, сборка и выполнение только открытого файла.

Параметры сборки и запуска (*Configure Build + Run file*): открывается многостраничное окно с опциями сборки только данного файла, когда выбрано «Собрать проект» (*Build Project*), позволяющее выбрать рабочий каталог, использовать различные макросы и т. д. Затем файл собирается и выполняется.

Оперативная справка открывает окно браузера с картинкой бегущего гепарда и несколькими связями на веб-сайты *Lazarus*, *FreePascal* и *WiKi*.

Параметры справки – открывается меню с опциями выбора инструмента просмотра и баз данных для чтения информации «Справки».

Кнопочная панель – это маленькая панель в левой верхней части основного окна, слева от палитры компонентов, имеет набор кнопок, повторяющих наиболее часто применяемые выборы основного меню: создать модуль, «Открыть» (со стрелкой вниз для отображения списка недавно использованных файлов), «Сохранить», «Сохранить все», «Создать форму», «Переключить Форма/Модуль» (т. е. показать либо форму, либо модуль исходного кода), «Показать модули», «Показать формы», «Запуск», «Пауза», «Шаг со входом», «Шаг в обход».

3.2.2 Палитра компонентов

Библиотека визуальных компонентов (*Visual Component Library*, *VCL*) содержит большое количество классов, предназначенных для быстрой разработки приложений. Библиотека написана на *Object Pascal*.

В *VCL* содержатся невизуальные и визуальные компоненты, а также другие классы, начиная с абстрактного класса *TObject*. При этом все компоненты являются классами, но не все классы являются компонентами.

Все классы *VCL* расположены на определенном уровне иерархии и образуют дерево (иерархию) классов. Знание происхождения объекта оказывает значительную помощь при его изучении, так как потомок наследует все элементы объекта-родителя. Так, если свойство *caption* принадлежит классу *TControl*,

то это свойство будет и у его потомков, например у классов *TButton* и *TCheckBox* и у компонентов – кнопки *Button* и независимого переключателя *CneckBox* соответственно.

Это панель инструментов с вкладками, каждая из которых представляет собой набор иконок, составляющих функциональную группу компонентов. Самая левая иконка на каждой вкладке в виде стрелочки называется Средством выбора.

Если навести курсор мыши на иконку палитры компонентов без нажатия, появится название данной компоненты. Каждое название начинается с *T*, что означает Тип, а точнее Класс компоненты. При выборе компоненты для размещения на форме, *Class* добавится в секцию *type* раздела *interface* модуля (обычно в виде части на *TForml*), и *instance* (образец) этого класса добавится в секцию *var* (обычно как переменная *Forml*). Все *Methods* (методы), разработанные для формы или ее компонент (процедуры или функции), будут помещены в раздел *implementation* модуля.

Вкладки (их имена достаточно понятны и не требуют разъяснений): *Standart*; *Additional*; *Common Controls*; *Dialogs*; *Misc*; *Data Controls*; *Data Access*; *System*; *SynEdit*.

Компоненты палитры *StdCtrls*, *ComCtrls* и *ExtCtrls* содержат определения и описания многих из наиболее часто используемых элементов управления для построения форм и других объектов в приложениях *Lazarus*. Рассмотрим использование палитры компонентов на примере использования подпалитры *Common*.

Многие из используемых компонент на палитре имеют соответствующий базовый (родительский) класс, например такие, как *TCustomButton*, *TCustomMemo* или *TCustomScrollBar*. Некоторые свойства и методы, имеющие отношение к размещенному на форме элементу определяются, соответственно, более полно в классе *TCustomXXX* и наследуются от базового класса.

Если на форме размещен компонент редактора, то не требуется явно добавлять код для его создания. Компонент автоматически создается *IDE* вместе с формой, и уничтожается, когда форма разрушается.

Существует несколько способов определения и изменения свойств компонента. Если компонент разместить на дизайнере формы и посмотреть на свойства в инспекторе объектов, то можно наблюдать за изменением некоторых свойства при перемещении

компонента по форме. Кроме того, с помощью инспектора объектов, можно самостоятельно и независимо выбрать значение, связанное со свойством объекта, и ввести новое значение. Также можно явно изменить свойства объекта, если ввести новое значение свойства в редакторе исходного кода. Новое значение также отобразится в инспекторе объектов.

Таким образом, существует возможность использовать три разных способа определения свойства каждого объекта: с помощью перемещения при помощи мыши на форме; путем установки значения параметров в инспекторе объектов; посредством написания кода в редакторе.

Рассмотрим наиболее распространенные свойства компонентов, определенных в этих объектах. Нестандартные или контрольно-специфические свойства могут быть найдены в справке среды разработки для индивидуального управления ими. Дополнительную помощь можно получить, выбрав свойство или ключевое слово, либо в «Инспекторе Объектов», либо в «Редакторе Исходного Кода», и нажав клавишу *F*1. В этом случае будет осуществлен переход на соответствующую страницу помощи в документации проекта.

3.2.3 Часто используемые свойства компонентов

Действие (Action). Основное действие или событие, связанное с объектом. Если выбрано действие *Exit*, то кнопка может вызвать действие *Close*.

Выравнивание (Align). Определяет способ, в соответствии с которым объект должен быть выровнен по отношению к родительскому объекту. Возможные значения *alTop* (размещены в верхней части и используют всю доступную ширину), *alBottom*, *alLeft* (расположены слева и используют всю доступную высоту), *alRight*, *alNone* (разместить в любом месте на родительском элементе управления) или *alClient* (занимает все свободное пространство рядом и выравнивается по верхней, нижней, левой или правой стороне базового объекта)

Якорь (Anchor). Используется для сохранения положения элемента управления на определенном расстоянии от края родительского элемента управления.

Автовыбор (AutoSelect). Используется для выделения всего текста, когда объект получает фокус или нажата клавиша *Enter*.

BorderSpacing. Пространство между закрепленным элементом управления и его родителем.

Caption. Текст, который отображается на элементе или вблизи элемента контроля. По умолчанию *Caption* устанавливается такой же, как и Имя, а программист изменяет название на осмысленный текст.

CharCase. Указывает, как текст отображается в элементе управления редактирования текста.

Constraints. Устанавливает минимальный и максимальный размеры для элемента управления.

Color. Цвет, который будет использоваться для отрисовки элемента управления или цвета текста, который в нем содержится.

Enabled. Логическое свойство для определения, могут ли элементы быть выбраны, и разрешены ли выполнения действий с ними. Если элемент контроля не включен, он помечен серым цветом на форме.

Font. Шрифт, используемый для написания текста, связанного с элементом управления.

Hint. Короткий кусок информативного всплывающего текста, который появляется, если курсор мыши наведен на элемент управления.

Items. Перечень тех сущностей, которые объект содержит, например группу изображений, серии строк текста, ряд действий в *ActionList*, и т. д.

Lines. Массив строк, содержащих текстовые данные в контрольной группе с более чем одной строкой данных.

Name. Идентификатор, под которым элемент управления определен в программе.

PopUpMenu. Окно, содержащее контекстно-зависимое меню, которое появляется при нажатии правой кнопки мыши на объект, элемент управления. *Position* (*or Top*, *Left*) Определяет, где объект или элемент управления находится в базовых форме или окне.

ReadOnly. Логическое свойство, которое, если имеет значение *True*, означает, что содержимое элемента управления может быть прочитано пользователем или вызывающей программой, но не может быть переписано или изменено. *ShowHint*. Позволяет отображать небольшое окно с контекстной справкой или другим описанием, которое будет отображаться при наведении курсора мыши. Размещено над элементом управления.

Size (or Height and Width). Размер элемента управления.

Style. Набор вариантов, доступных для стиля, который зависит от рода элемента управления.

TabOrder. Целое число, определяющее, где в последовательности вкладок на форме этот элемент управления расположен. x *TabStop*. Логическое свойство, которое при значении *True* размещает элемент в последовательности объектов, до которых пользователь может добраться при последовательном нажатии клавиши табуляции.

Text. Строка текста, которая представляет фактические данные, которые этот объект содержит. В частности, относится к таким типам объектов, как *Text*, *Memo* и *StringList*.

Visible. Если содержит значение *True*, то объект можно увидеть в форме; если – *False*, то объект скрыт.

WordWrap. Логический флаг, определяющий, включен ли перенос слов. События (*Event Actions*)

3.2.4 Основные события компонентов

Многие действия перечислены на вкладке «События» Инспектора объектов (*Object Inspector*). При выборе записи в списке *ComboBox*, появляется выпадающий список, показывающий любые действия, которые уже определены в *IDE*, и позволяет вам выбрать одно из них, чтобы оно было связано с этим событием. Также возможно выбрать многоточие, и тогда будет осуществлен переход в область редактора исходного кода, где можно начать определять собственные инструкции действий для выбранного события.

Для большинства элементов управления достаточно обеспечить написание кода для *OnClick*, хотя для более сложных объектов может быть необходимо также обеспечить действие *OnEntry* (когда курсор мыши входит в управления и передает ему фокус) и *OnExit* (когда курсор мыши покидает элемент управления) или написать обработчик событий для *OnChange* или *OnScrol*.

Всплывающее меню, которое появляется при нажатии правой кнопкой мыши на объекте в дизайнере форм, содержит в качестве своего первого пункта «Создать событие по умолчанию», и выбор этой опции, будет иметь тот же эффект, что и выбор с многоточием в инспекторе объектов по умолчанию для данного события. Как правило, на событие *OnClick* средствами *IDE* будет создано поле со

стандартным программным кодом в редакторе, где можно ввести код для обработчика выбранного события.

По отношению к событийной модели общей стратегией в объектно-ориентированном программировании является предоставление *ActionList* с объектом для ввода, удаления или редактирования некоторого количества предварительно определенных действий, из которых наиболее подходящие могут быть выбраны для использования в каждом конкретном случае и для каждого элемента управления.

OnChange. Действия, которые будут приняты, если любое изменение будет обнаружено.

OnClick. Предлагаемое действие, когда нажата кнопка мыши.

Click. Метод для эмуляции в коде действия однократного нажатия на элементе управления.

OnDragDrop. Предлагаемое решение во время события *OnDragDrop*. *OnEditingDone*. Предлагаемое решение, когда пользователь закончил все изменения или модификации объекта.

OnEntry. Действие в результате события, когда курсор мыши входит в область, занимаемую объектом, и переносит фокус на этот объект.

OnExit. Предлагаемое действие, когда мышь перемещается из области объекта с потерей фокуса на объекте.

OnKeyPress. Действие, по событию соответствующее любому нажатию кнопки.

OnKeyDown. Действия, активирующиеся, когда клавиша нажата, а фокус находится в элементе управления.

OnKeyUp. Действия, выполняющиеся, когда клавиша освобождена, в то время как фокус находится в этом элементе управления.

OnMouseMove. Действия, реализующиеся, когда курсор мыши перемещается над элементом управления, находящемся в фокусе.

OnMouseDown. Действия, выполняющиеся, когда кнопка мыши нажата и в то же время находится в элементе управления, находящемся в фокусе.

OnMouseUp. Действия, выполняющиеся, когда кнопка мыши не нажата, а курсор находится над этим элементом управления. Означает, что кнопка мыши была ранее нажата и была освобождена. Случай, когда курсор входит в область элемента

управления, но кнопка мыши еще не была нажата, определен событиями *OnEntry* или *OnMouseEnter*.

OnResize. Действие, совершаемое при изменении размеров элемента управления.

Конструкторы и деструкторы – это два специальные метода, связанные с каждым элементом управления:

Конструкторы: такие как, например, создание, выделения памяти и системных ресурсов, которые необходимы объекту. Они также вызывают конструктор любого подобъектов, присутствующего в классе.

Деструкторы: удаляют объект и освобождают память и другие ресурсы, ранее выделенные объекту. Если вы вызываете метод *Destroy* для уничтожения объекта, который ранее не был инициализирован, то он сгенерирует ошибку. Всегда используйте метод *Free* для освобождения объектов, потому, что он проверяет, является ли значение объекта равно нулю перед вызовом метода *Destroy*.