МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН ВОСТОЧНО-КАЗАХСТАНСКИЙ ГУМАНИТАРНЫЙ КОЛЛЕДЖ
ИМЕНИ АБАЯ

Среда разработки программы - Lazarus

Методическое пособие

Специальность: 0111000 «Основное среднее образование»
Квалификация: 0111093 «Учитель информатики»

Научные руководители:
Касенова А.А., Сайлаубекқызы А.

г. Усть-Каменогорск, 2021г.

Среда разработки программы - Lazarus

Методическое пособие

г. Усть-Каменогорск, 2021г.

УДК 004.4(075.32)
ББК К

Программа рассмотрена и рекомендована к утверждению на заседании методического совета Восточно-Казахстанского гуманитарного колледжа имени Абая

«_» _____2021г., протокол №__

Председатель методического совета_____Ж.Слямбекова

Заместитель директора по учебно-производственной работе _____ А.Аубакирова

Методическое пособие на тему «Средой разработка программ - Lazarus»

Научные руководители: А.А.Касенова, Сайлаубекқызы А. – Усть-Каменогорск 2021г.

В данном методическом пособии предусмотрены комплекс данных о Лазарусе. Можно узнать о интерфейсе данной программы, какие компоненты содержаться и как можно с ними работать. Предлагаемое учебное пособие предназначено для студентов педагогический колледжей и вузов, обучающихся по специальности «Информатика».

Содержание

Введение

Язык Pascal, изобретенный в начале 70 - х годов 20 - го века Н. Виртом и названный в честь французского математика и философа Блеза Паскаля, является одним из наиболее распространенных языков программирования для обучения. Что вполне естественно, так как является структурированным, логичным, легко читаемым и понимаемым. Программа на языке Pascal состоит из двух частей: описание действий, которые должны быть выполнены и описание данных, над которыми они выполняются. В тексте программы описание данных предшествует описанию действий. В этом выражается общее правило языка - каждый встречающийся в программе объект должен быть предварительно описан.

Для того чтобы писать и выполнять программы, необходимы компилятор и среда разработки. Существует довольно много компиляторов для языка Pascal. Основным компилятором является *Borland Pascal 7. 0.* Он применяется в основном для консольных приложений. Его логичным продолжением является визуальная среда разработки Borland Delphi. Данный инструмент предназначен для визуального проектирования и создания различных оконных приложений. Методы, подходы, принципы, применяемые в Delphi, сокращают в разы время разработки и поднимают на новый уровень качество разработки. Для того чтобы создать простое окно не надо писать строчки кода, нужно просто нажать на кнопку создания окна. То же самое можно сказать и про множество компонентов, используемых Delphi.

В последние 15 лет велась активная разработка альтернативы компилятору Borland Pascal. Она получила название Free Pascal. Free Pascal Compiler (FPC) это свободно распространяемый компилятор языка Pascal с открытыми исходными кодами, распространяется на условиях GNU General Public License (GNU GPL). Он совместим с Borland Pascal 7. 0 и Object Pascal Delphi, но при этом обладает рядом дополнительных возможностей, например, поддерживает перегрузку операторов. Free Pascal Compiler имеет свою собственную интегрированную среду разработки. Применяется также аббревиатура IDE (Integrated Development Environment). Среда имеет текстовый интерфейс очень похожий на интерфейс Turbo Pascal 7. 0.

Однако со временем текстовые интерфейсы были практически полностью вытеснены так называемыми графическими интерфейсами, работать в которых значительно удобнее. В 1999 г. Клифф Байзмент, Шейн Миллер и Майкл А. Гесса написали графическую среду для бесплатного компилятора FPC. Проект получал название Lazarus. На сегодняшний день следует признать, что идея оказалась весьма плодотворной потому, что среда существует и развивается и поныне. Она нашла свое место в учебном процессе и ее освоение позволит ученику осваивать язык Pascal, решать с его помощью различные задачи, что и определяет актуальность курсовой работы.

Сложилось определенное противоречие: между необходимостью применения алгоритмической среды Lazarus и отсутствием разработанной

технологии, позволяющей осуществлять применение алгоритмической среды в рабочей деятельности.

Стремление найти пути разрешения данного противоречия определило проблему, заключающуюся в теоретическом обосновании и практической реализации алгоритмической среды Lazarus.

Тема 1. Среда визуального программирования Lazarus

Lazarus - это среда визуаль программирования. Технология визуального программирования поз г строить интерфейс будущей программы из специальных компоне реализующих нужные свойства. Количество таких компонентов достаточно велико. Каждый из них содержит готовый программный код и все необходимые для работы данные, что избавляет программиста от создания того, что уже создано ранее. В основе языка Lazarus лежит язык программирования Free Pascal, который ведет свое начало от классического языка Pascal разработанный в конце 60-х годов XX века Николасом Виртом. Он разрабатывал этот язык как учебный язык для своих студентов. С тех пор Pascal, сохранив простоту и структуру языка, разработанного Н. Виртом, превратился в мощное средство программирования. С помощью современного языка Pascal можно производить простые расчеты, разрабатывать программы для проведения сложных инженерных и экономических вычислений.

1.1 Среда Lazarus

На рисунке 1 показано окно, которое появится после запуска Lazarus. В верхней части этого окна размещается главное меню и панель инструментов. Слева расположено окно инспектора объектов, а справа окно редактора исходного кода. Если свернуть или сдвинуть окно редактора, то станет доступным окно формы, представленное на рисунке 2

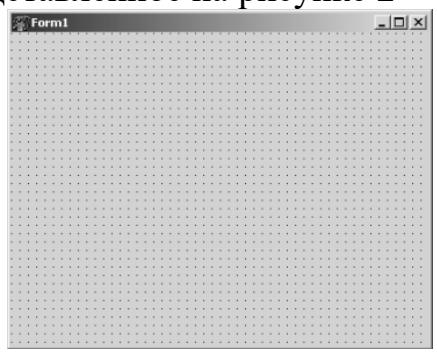

Рисунок 1. Окно формы

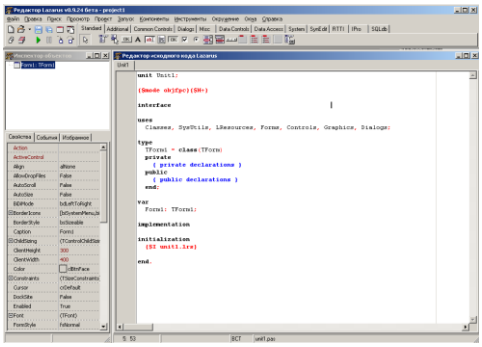

Рисунок 2. Среда визуального программирования Lazarus

Работу над программой в среде визуального программирования условно можно разбить на две части. Первая это создание внешнего вида (интерфейса) будущей программы, вторая- написание программного кода. Файлы, из которых в результате получается программа.

Окно, инспектора объектов и окно формы нужны для создания интерфейса программы, а редактор исходного кода- для работы с ее текстом. Файлы, из которых в результате получается программа, которую называют проектом.

1.2 Главное меню Lazarus

Все команды, необходимые для работы в среде визуального программирования Lazarus, содержатся в главном меню. Доступ к командам главного меню осуществляется одинарным щелчком левой кнопкой мыши. Работа с файлами в среде Lazarus осуществляется при помощи пункта меню Файл. Команды этого пункта меню можно разбить на группы:

– создание новых файлов - Создать модуль, Создать форму, Создать...;

– загрузка ранее созданных файлов - Открыть, Открыть недавнее, Вернуть;

– сохранение файлов - Сохранить, Сохранить как…, Сохранить все;

– закрытие файлов - Закрыть, Закрыть все файлы редактора;

– вывод на печать - Печать…;

– перезагрузка среды - Перезапуск;

– выход из среды – Выход.

Команды, предназначенные для редактирования текста программного кода, собраны в меню Правка. В основном это команды характерные для большинства текстовых редакторов:

– команды отмены или возврата последней операции - Отменить, Вернуть;

– команды переноса, копирования и вставки выделенного фрагмента текста в буфер- Вырезать, Копировать, Вставить;

– команды, работающие с выделенным блоком текста – Сдвинуть блок вправо, Сдвинуть блок влево;

– команды смены регистра -Верхний регистр выделения, Нижний регистр выделения;

– команды выделения фрагмента текста собраны в пункте меню Выделить.

Команды меню Поиск можно разделить на группы. Первая группа - это непосредственно команды поиска и замены, вторая - это команды перехода, а третья - работа с закладкой. В четвертой группе объединены команды поиска, замены и перехода в выделенном фрагменте. Большинство из этих команд используются в текстовых редакторах, смысл остальных понятен из названия.

Пункт меню Просмотр применяют для настройки внешнего вида среды программирования. Первая группа команд открывает или активизирует следующие окна:

– Инспектор объектов - окно, с помощью которого можно описать внешний вид и поведение выделенного объекта;

– Редактор исходного кода - окно, в котором можно создавать и редактировать текст программы;

– Обозреватель кода - содержит общую информацию о проекте;

– Редактор LazDoc - редактор шаблонов;

– Браузер кода - окно проводника проекта.

Следующая группа команд пункта меню Просмотр тоже открывает диалоговые окна. Эти окна носят информационный характер. Так команды Модуль... и Форма... выводя список модулей и форм данного проекта. Назначение команд Показать зависимости модулей и

Показать сведения о модуле говорят сами за себя. Последняя команда этой группы Переключить модуль/форму активизирует либо окно редактора, либо форму. В последней группе команд следует отметить команды Показать палитру компонентов и Показать кнопки быстрого доступа. Устанавливая метки рядом с этими командами, пользователь выводит на экран или наоборот убирает панели инструментов. Командой Окна отладки пользуются во время отладки программного кода. Здесь можно вызывать Окно наблюдений, Окно отладчика, просматривать точки останова и значения переменных в процессе выполнения программы. Команды пункта меню Проект предназначены для выполнения различных операций с проектом:

– команды создания проекта - Создать проект и Создать проект из файла;

– команды вызова ранее созданного проекта - Открыть проект, Открыть недавний проект, Закрыть проект;

– команды сохранения проекта - Сохранить проект, Сохранить проект как..., Опубликовать проект;

– команды управления проектом - Инспектор проекта, Настройка проекта..., Параметры компилятора и т.д.

Команды, позволяющие запускать проект на выполнение и выполнять его отладку, содержатся в пункте главного меню Запуск: Собрать - сборка программы из откомпилированных файлов;

– Собрать все - скомпоновать все файлы проекта;

– Быстрая компиляция - компиляция файлов программы;

– Запуск - запуск проекта с помощью отладчика (компиляция, компоновка и выполнение);

– Пауза - приостановка выполнения программы до нажатия любой клавиши;

– Шаг с входом - режим пошагового отладочного выполнения программы с входом в вызываемые процедуры и функции;

– Шаг в обход - режим пошагового отладочного выполнения программы без входа в вызываемые процедуры и функции;

– Запуск до курсора - отладка и выполнение программы в этом режиме осуществляются до оператора, стоящего в строке помеченной курсором;

– Останов - прерывание выполнения программы;

– Сброс отладчика - сброс всех ранее задействованных отладочных средств и прекращение выполнения программы;

– Настройка сборки + запуска... - настройка параметров компоновки и запуска;

– Вычислить \ Изменить - возможность просмотреть значение переменной и/или найти значение выражения в процессе выполнения программы, при необходимости можно изменить значения любой переменной;

– Добавить наблюдения - в открывающемся окне можно указать переменные и/или выражения, за изменением значений которых следует понаблюдать при отладке программы;

– Добавить точку останова - установка в текущей строке контрольной точки; после запуска программного кода отладчик прекратит его выполнение перед оператором, помеченным точкой останова; в программе можно указать произвольное количество таких точек. Точку

останова также можно добавить щелчком мыши по номеру строки программного кода.

1.3 Окно формы

Окно формы рисунок 3.- это проект интерфейса будущего программного продукта.

Вначале это окно содержит только стандартные элементы – строку заголовка и кнопки развертывания, свертывания и закрытия. Рабочая область окна заполнена точками координатной сетки.

Задача программиста – используя панель компонентов, заполнить форму различными интерфейсными элементами, создавая тем самым внешний вид своей программы.

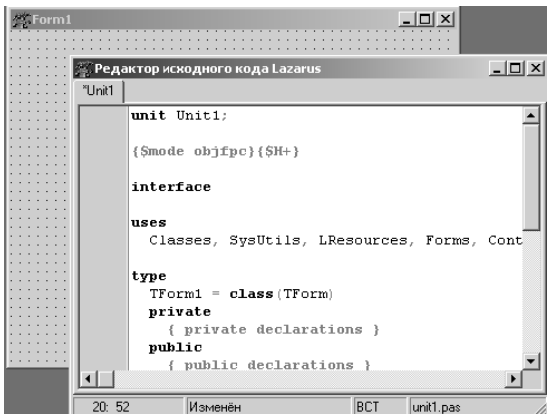

Рисунок 3. Окно формы и окно редактора

1.4 Окно редактора Lazarus

Окно редактора рисунок 3 тесно связано с окном формы и появляется вместе с ним при создании нового проекта. Окно редактора по умолчанию находится на первом плане и закрывает окно формы. Переключаться между этими окнами можно командой Просмотр - Переключить модуль/форму, клавишей F12 или просто щелчком мыши. Окно редактора предназначено для создания и редактирования текста программы, который создается по определенным правилам и описывает некий алгоритм. Если окно формы определяет внешний вид будущей программы, то программный код, записанный в окне редактора, отвечает за ее поведение. Вначале окно редактора содержит текст, обеспечивающий работу пустой формы. Этот программный код появляется в окне редактора автоматически, а программист в ходе работы над проектом вносит в него дополнения, соответствующие функциям программы.

При загрузке Lazarus автоматически загружается последний проект, с которым работал пользователь. Происходит это благодаря установке Открывать последний проект при Запуске, которая находится на вкладке Сервис-Параметры – Параметры IDE.... Если убрать метку рядом с командой Открывать последний проект при запуске, то при загрузке Lazarus будет создавать новый проект.

Настроить окно редактора можно с помощью пункта Общие меню Сервис- Параметры IDE.... редактора рисунок 4. Для того чтобы установить (отменить) ту или иную настройку, достаточно установить (убрать) маркер рядом с ее названием и подтвердить изменения нажатием кнопки Ok.

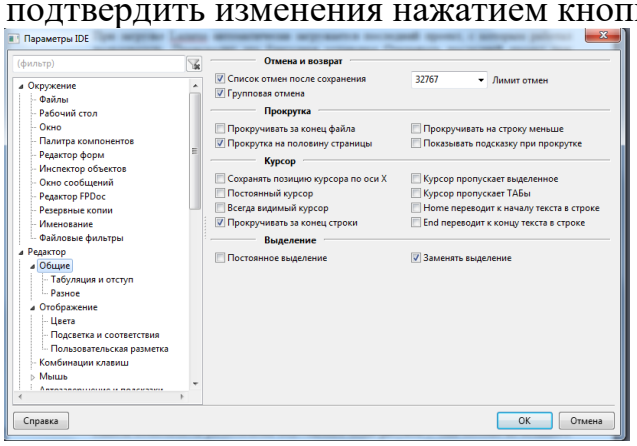

Рисунок 4. Окно настройки файлов в среде Windows

1.5 Панель компонентов

Панель компонентов расположена под главным меню рисунок 5. Она состоит из большого числа групп, в которых располагаются соответствующие компоненты.

Рисунок 5. Панель компонентов

Компонент – это некий функциональный элемент интерфейса, обладающий определенными свойствами. Размещая компоненты на форме, программист создает внешний вид своей будущей программы - окна, кнопки, переключатели, поля ввода и т.п.

Для внедрения нового компонента на форму нужно сделать два щелчка мышкой:

• в панели компонентов, для выбора компонента;

• в рабочем пространстве формы, для указания положения левого верхнего угла компонента.

Компоненты объединяются в группы по функциональному признаку. После создания проекта по умолчанию открывается список группы Standard, содержащий основные элементы диалоговых окон. Просмотреть другие группы можно, раскрывая их щелчком по соответствующей вкладке.

1.6 Инспектор объектов

Окно инспектора объектов располагается слева от окна редактирования. Как правило, оно содержит информацию о выделенном объекте. На рисунке 6 представлен инспектор объектов с информацией о вновь созданной форме. Окно инспектора объектов имеет три

вкладки: Свойства, События, Избранное. Эти вкладки используются для редактирования свойств объекта и описания событий, на которые будет реагировать данный объект. Совокупность свойств отображает внешнюю сторону объекта, совокупность событий – его поведение.

Вкладки инспектора объектов представляют собой таблицу. В левой колонке расположены названия свойств или событий, в правой - конкретные значения свойств или имена подпрограмм, обрабатывающих события. Чтобы выбрать свойство или событие, необходимо щелкнуть левой кнопкой мыши по соответствующей строке.

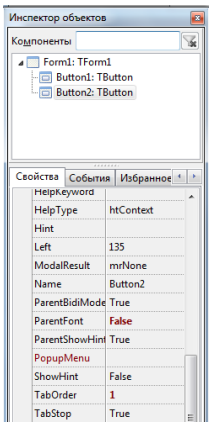

Рисунок 6. Окно инспектора объектов

Тема 2. Интегрированная среда разработки приложений

Процесс создания программы в Lazarus состоит из двух этапов: формирование внешнего вида программы, ее интерфейса и написание программного кода на языке программирования Free Pascal, заставляющего работать элементы интерфейса.

Начнем знакомство с визуальным программированием с создания простой программы про кнопку, которая хочет, чтобы по ней щелкнули. После чего появляется сообщение о работе программы.

Начнем с создания нового проекта. Для этого выполним команду главного меню Проект- Создать проект.... В появившемся диалоговом окне рисунок 8 выберем из списка слово Приложение и нажмем кнопку Создать. Результатом этих действий будет появление

окна формы и окна редактора программного кода.

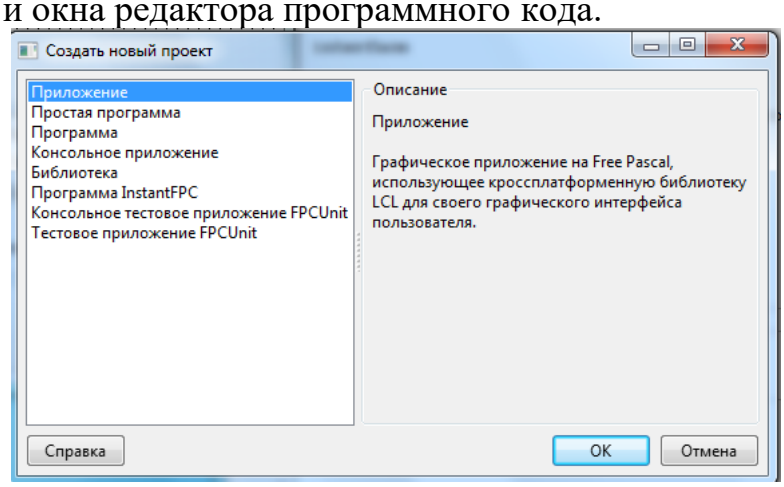

Рисунок 8. Создание нового проекта в среде Windows.

Сохраним созданный проект, воспользовавшись командой Проект-Сохранить проект как…. Откроется окно сохранения программного кода Сохранить Unit1. Создадим в нем новую папку Пример 1 (можно воспользоваться кнопкой Создание новой папки (Создание каталога)), откроем ее и щелкнем по кнопке Сохранить. Тем самым мы сохраним файл Unit1.pas, содержащий текст программы. Сразу же откроется окно Сохранить проект, в

котором также необходимо щелкнуть по кнопке Сохранить. Теперь мы сохранили файл Project1, содержащий общие сведения о проекте. На самом деле все наши манипуляции привели к сохранению более чем двух файлов. Теперь в каталоге Пример 1 хранится файл с текстом программы Unit1.pas, файл Unit1.lfm19, со сведениями о форме Form1, а также файлы Project1.lpn и Project1.lpi, содержащие настройки системы программирования и параметры конфигурации проекта рисунок 9.

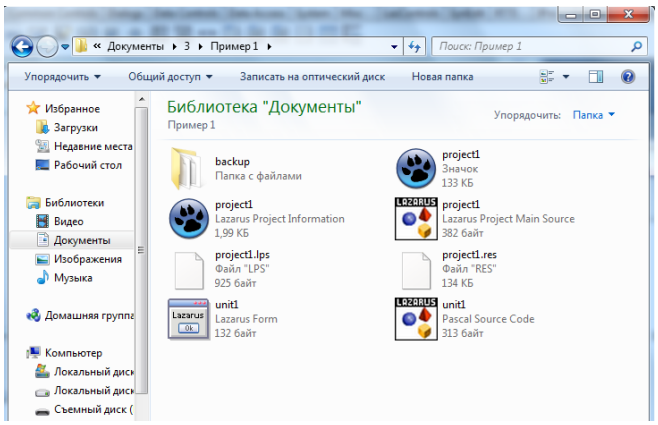

Рисунок 9. Файлы проекта в среде Windows

Теперь можно приступить к визуальному программированию. У нас есть один объект - форма Form1. Для изменения её свойств надо перейти в окно инспектора объектов и Найти нужное свойство и изменить его значение.

Для помещения на форму объекты, следует выбрать требуемый объект на Панели компонентов и поместить как графический объект на форму. Помещенные на форму объекты присоединяются к Главной форме и отображаются в окне Компоненты рисунок 10.

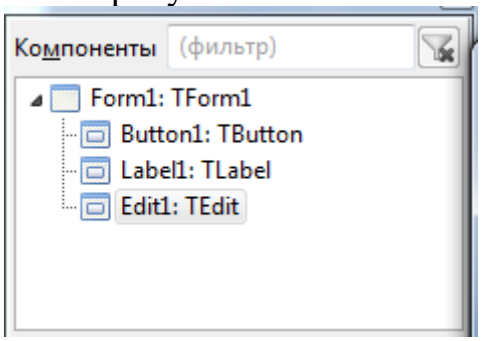

Рисунок 10. Окно компоненты проекта

Для выполнения компиляции проекта, следует его предварительно собрать Запуск- Собрать, затем выполнить компиляцию проекта Запуск-Компилировать.

Для того чтобы посмотреть, как работает наша программа, ее необходимо запустить на выполнение. Сделать это можно командой Запуск — Запустить, функциональной клавишей F9 или кнопкой Запуск в панели инструментов.

Практическое задание 1.

Цель- научиться добавлять объекты и подключать процедуру событий к объектам.

Создайте новый проект- форму, добавив на нее две кнопки рисунок 11.:

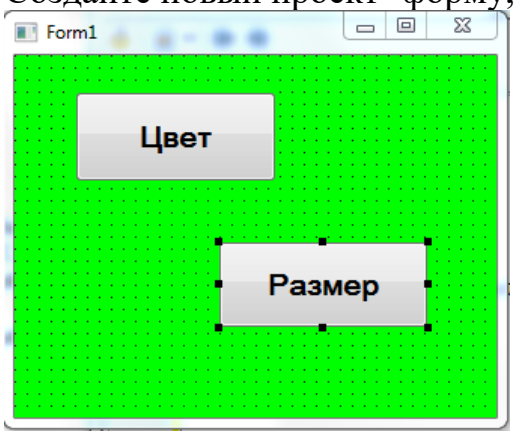

Рисунок 11. Интерфейс программы

Измените свойства формы и свойства кнопок:

Свойство Form1	**Значение**
Color	clLime
Width	140
Height	58
Свойство Button1	**Значение**
Caption	Цвет
Font	Arial- 14- полужирный курсив
Height	60
Width	134
Свойство Button2	**Значение**
Caption	Размер
Font	Arial- 14- полужирный курсив
Height	58
Width	140

Сохраните проект в папке Первый проект.

Выделите кнопку **Button1**, в окне **Инспектора объектов** Выберите вкладку **События,** выберите событие OnClik и двойным щелчком по пустому полю подключите событие по кнопке- щелчок мышью, у вас в окне редактора должен появиться шаблон процедуры события на кнопке **Button1,** рисунок 12.:

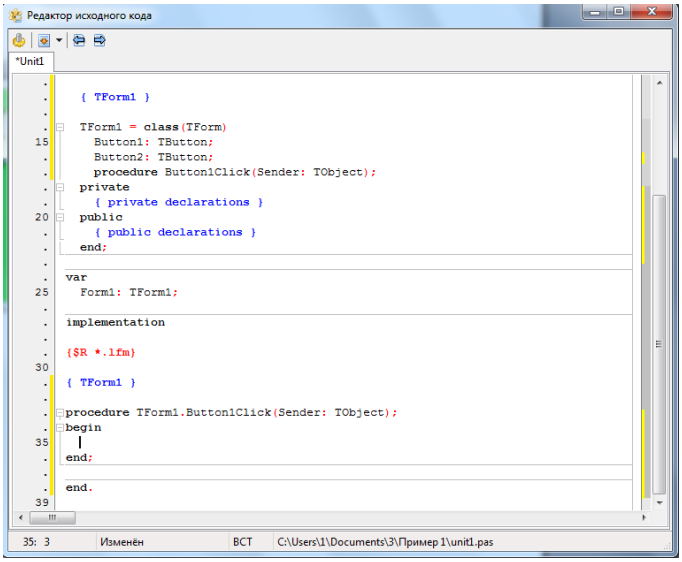

Рисунок 12. Окно редактора кода программы

В окне редактора кода в процедуре напишите код программы

procedure TForm1.Button1Click(Sender: TObject);
begin
Form1.Color:=clRed;
end;

Подключите процедуру щелчок к кнопке **Button2** и в процедуре напишите код программы:

procedure TForm1.Button2Click(Sender: TObject);
begin
Form1.Width:=189;
end;

Выполните сборку проекта, компиляцию и запуск. Нажмите на кнопку цвет, затем на кнопку размер. Что изменилось в программе ?

Запишите в тетради алгоритм создания проекта.

Тема 3. Окна, формы и объекты, процедуры и функции.

3.1 Основные понятия объектно-ориентированного языка

Объектно-ориентированное программирование позволяет программировать в терминах классов:

– определять классы;

– конструировать новые и производные (дочерние) классы на основе существующих классов;

– создавать объекты, принадлежащие классу (экземпляры класса).

Класс описывает свойства (атрибуты) объекта и его методы (включая обработчики событий). При создании объекта он наследует структуру (переменные) и поведение (методы) своего класса. В свою очередь, класс, называемый потомком, производным или дочерним классом (подклассом), также может быть создан на основе другого родительского класса (предка) и при этом наследует его структуру и поведение. Любой компонент (элемент управления) или объект в всегда является экземпляром класса. Программно объект представляет собой переменную объектного типа. Для каждого компонента Lazarus существует свой класс, наследуемый от TComponent. Предком всех объектов, включая компоненты, является класс TObject. Наследование позволяет определять новые классы в терминах существующих классов.

Инкапсуляция - это создание защищенных объектов, доступ к свойствам и методам которых разрешен только через определенные разработчиком «точки входа». Иначе говоря, инкапсуляция - это предоставление разработчику конкретного набора свойств и методов для управления поведением и свойствами объекта, определяемыми внутри класса.

Полиморфизм - это возможность различных объектов реагировать по-разному на одни и те же события. Синтаксис языка поддерживает общепринятую для объектно-ориентированного программирования нотацию: **имя_объекта.свойство** для ссылки на свойство объекта или **имя_объекта.метод** для вызова метода объекта.

Каждый объект обладает набором свойств. Свойства могут быть как наследуемые от родительского класса, так и добавленные индивидуально для создаваемого объекта. Список всех свойств объекта и их значений отображается в диалоговом окне Инспектор объектов. Ссылка на свойство в программном модуле записывается как Имя_объекта.Свойство. Метод - это процедура или функция, ассоциируемая с некоторым объектом. Ссылка на метод в программном модуле записывается как Имя_0бъекта. Метод. Lazarus-приложение выполняется в среде Windows, и как любое Windows- приложение, получает сообщения о возникающих для него событиях. Управление приложением фактически сводится к обработке получаемых сообщений. Методы, в которых содержится код обработки события, называются обработчиками событий (События). Lazarus автоматически генерирует

процедуры обработки событий - обработчики событий для любого компонента. При этом имя обработчика событий формируется из имени компонента и названия события (например, EditlClick). Имя обработчика события автоматически квалифицируется именем класса формы. Например: TForml.ButtonlClick(Sender: TObject). Для каждого компонента предусмотрено одно стандартное событие. Например, для командной кнопки, флажка, списка, поля ввода - это событие Click, а для формы - событие FormCreate. Для того чтобы автоматически добавить в модуль объявление и описание разработчика стандартного события, достаточно выполнить на компоненте формы или самой форме двойной щелчок мышью.

3.2 Структура программы.

Любой проект в Lazarus – это совокупность файлов, из которых создается единый выполняемый файл. В простейшем случае список файлов проекта имеет вид:

- файл описания проекта (.lpi);
- файл проекта (.lpr);
- файл ресурсов (.lrs);
- модуль формы (.lfm);
- программный модуль (.pas);

После компиляции программы из всех файлов проекта создается единый выполняемый файл, имя этого файла совпадает с именем проекта. Программный модуль, или просто модуль, – это отдельно компилируемая программная единица, которая представляет собой набор типов данных, констант, переменных, процедур и функций. Любой модуль имеет следующую структуру:

```
unit имя_модуля; //Заголовок модуля.
interface
//Раздел описаний.
implementation
//Раздел реализаций.
end. //Конец модуля.
```

Заголовок модуля – это зарезервированное слово unit, за которым следует имя модуля и точка с запятой. В разделе описаний, который открывается служебным словом interface, описывают программные элементы – типы, классы, процедуры и функции:

```
interface
uses список_модулей;
type список_типов;
const список_констант;
var список_переменных;
procedure имя_процедуры;
...
```

function имя_функции;

...

Раздел implementation содержит программный код, реализующий механизм работы описанных программных элементов (тексты процедур обработки событий, процедуры и функции, созданные программистом). Процедуры и функции в Lazarus также построены по модульному принципу.

Тело программы начинается со слова begin, затем следуют операторы языка Pascal, реализующие алгоритм решаемой задачи. Операторы в языке Pascal отделяются друг от друга точкой с запятой и могут располагаться в одну строчку или начинаться с новой строки (в этом случае их также необходимо разделять точкой с запятой). Назначение символа « ; » - отделение операторов друг от друга. Тело программы заканчивается служебным словом end. Несмотря на то что операторы могут располагаться в строке как угодно, рекомендуется размещать их по одному в строке, а в случае сложных операторов отводить для каждого несколько строк. Рассмотрим более подробно структуру программы:

program имя_программы;
uses modul1, modul2, ..., moduln;
const описания_констант;
type описания_типов;
var описания_переменных;
begin
оператор_1;
оператор_2;
end.

3.3 Элементы языка

Программа на языке Free Pascal может содержать следующие символы:

•латинские буквы A, B, C…, x, y, z;
•цифры 0, 1, 2…, 9;
•специальные символы + ,–, /, =, <, >, [,], ., (,), ;, :, {, }, $, #, _, @, ‘, ^.

Из символов алфавита формируют ключевые слова и идентификаторы. Ключевые слова – это зарезервированные слова, которые имеют специальное значение для компилятора. Примером ключевых слов являются операторы языка, типы данных и т.п. Ключевые слова используются только так, как они определены при описании языка. Идентификатор – это имя программного объекта, представляющее собой совокупность букв, цифр и символа подчеркивания. Первый символ идентификатора – буква или знак подчеркивания, но не цифра. Идентификатор не может содержать пробел. Прописные и строчные буквы в именах не различаются, например ABC, abc,

Abc – одно и то же имя. Каждое имя (идентификатор) должно быть уникальным и не совпадать с ключевыми словами.

В тексте программы можно использовать комментарии. Комментарий это текст, который компилятор игнорирует. Комментарии используют для пояснений программного кода или для временного отключения команд программного кода при отладке. Комментарии бывают однострочные и многострочные. Однострочный комментарий начинается с двух символов «косая черта» // и заканчивается символом перехода на новую строку. Многострочный комментарий заключен в фигурные скобки {} или располагается между парами символов (* и *). Понятно, что фигурные скобки {} или символы (* и *). К программным объектам относятся константы, переменные, метки, процедуры, функции, модули и программы.

Например:

```
{Комментарий может
выглядеть так!}
(*Или так.*)
//А если вы используете такой способ,
//то каждая строка должна начинаться
//с двух символов ≪косая черта≫.
```

Преобразование строк в другие типы

Таблица 1.

Обозначение	**Тип аргументов**	**Тип результата**	**Действие**
StrToDateTame(S)	строка	Дата и время	преобразует символы из строки s в дату и время
StrToFloat(S)	строка	вещественное	преобразует символы из строки s в вещественное число
StrToInt(S)	строка	целое	преобразует символы из строки s в вещественное число
Val(S,X,Kod)	строка		преобразует строку символов S во внутреннее представление числовой переменной X, если преобразование прошло успешно, Kod=0.

Обратное преобразование

Таблица 2.

Обозначение	**Тип аргументов**	**Тип результата**	**Действие**
DataTimeToStr(V)	Дата и время	строка	преобразует дату и время в строку.
FloatToStr(V)	вещественное	строка	Преобразует вещественное число в строку
IntToStr(V)	целое	строка	преобразует целочисленное число в строку
FloatToStrF(V, F,P,D)	вещественное	строка	преобразует вещественное число V в строку символов с учетом формата F и параметров P, D.

Тема 4. Создание интерфейса проекта

4.1 Массивы

Массив — это структура данных, представляющая собой набор переменных одинакового типа, имеющих общее имя. Массивы удобно использовать для хранения однородной по своей природе информации, например, таблиц и списков.

Массив, как и любая переменная программы, перед использованием должен быть объявлен в разделе объявления переменных. В общем виде инструкция объявления массива выглядит следующим образом:

Имя: **array** [нижний_индекс. .верхний_индекс] of тип

где:

имя — имя массива;

array — зарезервированное слово языка Delphi, обозначающее, что объявляемое имя является именем массива;

нижний_индекс и верхний_и«декс — целые константы, определяющие диапазон изменения индекса элементов массива и, неявно, количество элементов (размер) массива;

тип — тип элементов массива.

Примеры объявления массивов:

temper:array[1..31] of real;

коef:array[0. .2] of integer;

name:array[1..30] of string[25];

При объявлении массива удобно использовать именованные константы. Именованная константа объявляется в разделе объявления констант, который обычно располагают перед разделом объявления переменных. Начинается раздел объявления констант словом const. В инструкции объявления именованной константы указывают имя константы и ее значение, которое отделяется от имени символом "равно". Например, чтобы объявить именованную константу нв, значение которой равно 10, в раздел const надо записать инструкцию: нв=10. После объявления именованной константы ее можно использовать в программе как обычную числовую или символьную константу. Ниже в качестве примера приведено объявление массива названий команд-участниц чемпионата по футболу, в котором используются именованные константы.

```
const
NT = 18; // число команд
SN = 25; // предельная длина названия команды
var
team: array[1..NT] of string[SN];
```

4.2 Использование компонента StringGrid

Для ввода массива удобно использовать компонент StringGrid. Значок компонента StringGrid находится на вкладке Additional рисунок 15.

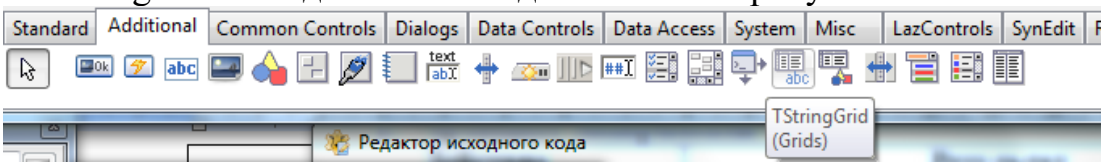

Рисунок 15. Компонент StringGrid

Компонент StringGrid представляет собой таблицу, ячейки которой содержат строки символов. В таблице перечислены некоторые свойства компонента таблица 4. StringGrid.

Таблица 4. Свойство компонента

Свойство	**Определяет**
ColCount	Количество колонок таблицы
RowCount	Количество строк таблицы
Cells	Соответствующий таблице двумерный массив. Ячейка таблицы, находящаяся на пересечении столбца номер col и строки номер row определяется элементом cells [col, row]
FixedCols	Количество зафиксированных слева колонок таблицы. Зафиксированные колонки выделяются цветом и при горизонтальной прокрутке таблицы остаются на месте
FixedRows	Количество зафиксированных сверху строк таблицы. Зафиксированные строки выделяются цветом и при вертикальной прокрутке таблицы остаются на месте
Options . goEditing	Признак допустимости редактирования содержимого ячеек таблицы. True — редактирование разрешено, False — запрещено
Options . goTab	Разрешает (True) или запрещает (False) использование клавиши <TaЬ> для перемещения курсора в следующую ячейку таблицы

4.3 Использование компонента Memo.

Компонент Memo позволяет вводить текст, состоящий из достаточно большого количества строк, поэтому его удобно использовать для ввода символьного массива. Компонент Memo добавляется в форму обычным образом. Значок компонента находится на вкладке Standard.

В таблице 5 перечислены некоторые свойства компонента Memo.

Таблица 5. Свойства компонента

Свойство	**Определяет**
Name	Имя компонента. Используется в программе для доступа к свойствам компонента
Text	Текст, находящийся в поле Memo. Рассматривается как единое целое
Lines	Текст, находящийся в поле Memo. Рассматривается как совокупность строк. Доступ к строке осуществляется по номеру
Lines .Count	Количество строк текста в поле Memo

Практическое задание 2.

Цель- научиться использовать компоненты StringGrid и поле Memo, для работы с массивом.

Задача 1.

Рассмотрим программу, которая вычисляет среднее арифметическое значение элементов массива. Диалоговое окно программы приведено на рисунке 16 Компонент StringGrid используется для ввода массива, компоненты Label1 и Label2- для вывода пояснительного текста и результата расчета, Button1- для запуска процесса расчета рисунок 16.

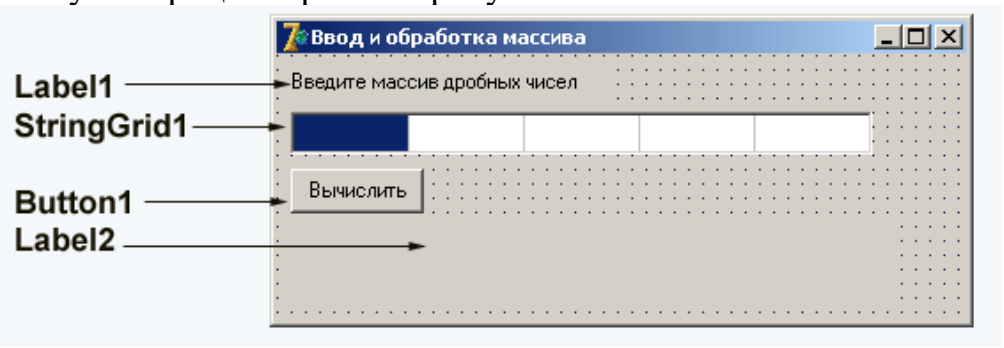

Рисунок 16. Окно интерфейса программы

Добавляется компонент stringGrid в форму точно так же, как и другие компоненты. После добавления компонента к форме нужно выполнить его настройку в соответствии с таблицей 6. Значения свойств Height и Width следует при помощи мыши установить такими, чтобы размер компонента был равен размеру строки.

Таблица 6. Значение свойств

Свойство	**Значение**
ColCount	5
FixedCols	0
RowCount	1
DefaultRowHeight	24
Height	24
DefaultColWidth	64
Width	328
Options . goEditing	True
Options . AlwaysShowEditing	True
Options .goTabs	True

На кнопку посадите процедуру:

```
procedure TForml.ButtonlClick(Sender: Tobject);
 var
a : array[1..5] of integer; // массив
summ: integer; // сумма элементов
sr: real; // среднее арифметическое
i: integer; // индекс
begin
// ввод массива
// считаем, что если ячейка пустая, то соответствующий
// ей элемент массива равен нулю
for i:= 1 to 5 do
if Length(StringGridl.Cells[i-1, 0]) <>0
then a[i] := StrToInt(StringGridl.Cells[i-1,0])
else a[i] := 0;
// обработка массива
summ := 0;
for i :=1 to 5 do
summ := summ + a[i]; sr := summ / 5;
Label2.Caption :='Сумма элементов: ' + IntToStr(summ)+ #13+ 'Среднее арифметическое: ' + FloatToStr(sr);
end;
```

Для того, чтобы курсор автоматически переходил в следующую ячейку таблицы, например, в результате нажатия клавиши <Enter>. Добавим процедуру обработки события onKeyPress. На эту же процедуру можно возложить задачу фильтрации вводимых в ячейку таблицы данных. В нашем случае надо разрешить ввод в ячейку только цифру.

Текст процедуры обработки события OnKeyPress приведен ниже. Следует обратить внимание на свойство Col, которое во время работы программы содержит номер колонки таблицы, в которой находится курсор. Это свойство

можно также использовать для перемещения курсора в нужную ячейку таблицы. Однако нужно учитывать, что колонки таблицы, впрочем, как и строки, нумеруются с нуля.

```
Procedure Tform1.StringGridlKeyPress(Sender: Tobject; var Key: Char);
begin
case Key of
#8,'0'..'9' : ; // цифры и клавиша <Backspace>
#13: // клавиша <Enter>
if StringGridl.Col < StringGridl.ColCount-1
then StringGridl.Col := StringGridl.Col+1;
else key := Chr(0); // остальные символы запрещены
end;
end;
```

Запустите программу, посмотрите как изменилась работа программы.

Задача 2.

Рассмотрим задачу, в которой компонент Memo используется для вывода символьного массива. Создайте интерфейс окна программы, как показано на рисунке 17.

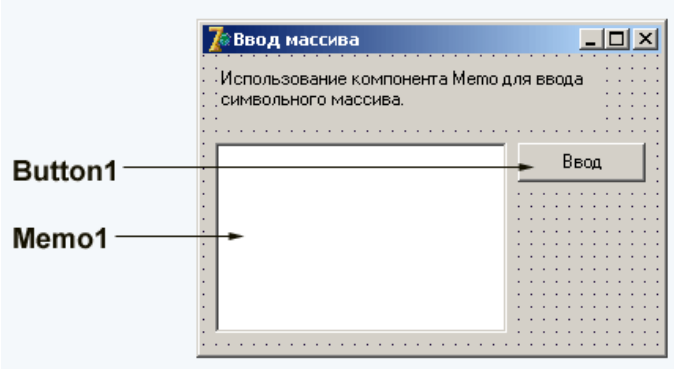

Рисунок 17. Диалоговое окно приложения Ввод массива

На кнопку посадите процедуру:

```
procedure Tforml .ButtonlClick(Sender: Tobject);
const
SIZE=5; // размер массива
var
a:array[l..SIZE]of string[30]; //массив
n: integer; // количество строк, введенных в поле Memo
i:integer; // индекс элемента массива
st:string;
begin
n:=Memo1.Lines.Count;
if n = 0 then begin
ShowMessage('Исходные данные не введены!');
```

```
Exit; // выход из процедуры обработки события
end;
// в поле Memo есть текст
if n > SIZE then begin
ShowMessage('Количество строк превышает размер массива.');
n:=SIZE; // будем вводить только первые SIZE строк
end;
for i:=1 to n do
a[i]:=Form1.Memo1.Lines[i-1]; // строки Memo пронумерованы с нуля
// вывод массива в окно сообщения
if n > 0 then begin
st:='Введенный массив:'+#13;
for i: =1 to n do
st:=st+IntToStr(i)+' '+ a[i]+f13; ShowMessage(st);
end;
end;
```

Задачи для самостоятельного решения.

1. Записать положительные элементы массива X подряд в массив Y. Вычислить сумму элементов массива X и произведение элементов массива Y.
2. Из массива Y удалить элементы, расположенные между максимальным и минимальным элементами.
3. Сформировать массив B, записав в него элементы массива A с нечетными индексами. Вычислить среднее арифметическое элементов массива B и удалить из него максимальный, минимальный и пятый элементы.
4. Дан массив целых чисел X. Переписать пять первых положительных элементов массива и последних два простых элемента в массив Y. Найти максимальный отрицательный элемент массива X.

Тема 5. Графические методы и процедуры.

При разработке оконного приложения Lazarus есть форма, на которой можно рисовать. В распоряжении программиста находится полотно (холст) — свойство Canvas, карандаш — свойство Pen, и кисть — свойство Brush.

Свойством Canvas обладают следующие компоненты:

– форма (класс Tform);
– таблица (класс TsringGrid);
– растровая картинка (класс Timage);
– принтер (класс Tprinter).

При рисовании компонента, обладающего свойством Canvas, сам компонент рассматривается как прямоугольная сетка, состоящая из отдельных точек, называемых пикселями.

Положение пикселя характеризуется его вертикальной (X) и горизонтальной (Y) координатами. Левый верхний пиксель имеет координаты (0,0). Вертикальная координата возрастает сверху вниз, горизонтальная — слева направо. Общее количество пикселей по вертикали определяется свойством Height, а по горизонтали – свойством

Weight. Каждый пиксель может иметь свой цвет. Для доступа к любой точке полотна используется свойство Pixels[X,Y]:Tcolor. Это свойство определяет цвет пикселя с координатами X(integer), Y(integer). Изменить цвета любого пикселя полотна можно с помощью следующего оператора присваивания

Компонент.Canvas. Pixels[X,Y]:=Color;

где Color- переменная или константа типа Tcolor. Определены следующие константы цветов таблица:

Таблица 6. Значение свойств Color

Константа	**Цвет**	**Константа**	**Цвет**
clBlack	Черный	clSilve	Серебристый
clMaroon	Каштановый	clRed	Красный
clGreen	Зеленый	clLime	Салатовый
clOlive	Оливковый	clBlue	Синий
clNavy	Темно-синий	clFuchsia	Ярко-розовый
clPurple	Розовый	clAqua	Бирюзовый
clTeal	Лазурный	clGray	Серый

Цвет любого пикселя можно получить с помощью следующего оператора присваивания:

Color:=Компонент.Canvas. Pixels[X,Y];

где Color- переменная типа Tcolor.

Класс цвета точки Tcolor определяется как длинное целое longint. Переменные этого типа занимают в памяти четыре байта. Четыре байта

переменных этого типа содержат информацию о долях синего (B), зеленого (G) и красного® цветов и устроены следующим образом: $00BBGGRR.

Для рисования используются методы класса Tcanvas, позволяющие изобразить фигуру (линию, прямоугольник и т.д.) или вывести текст в графическом режиме, и три класса, определяющие инструменты вывода фигур и текстов:

– TFont (шрифты);
– TPen (карандаш, перо);
– TBrush (кисть).

Класс TFONT

– Можно выделить следующие свойства объекта Canvas.Tfont:
– Name (тип string) — имя используемого шрифта.
– Size (тип integer) — размер шрифта в пунктах (points).

Пункт – это единица измерения шрифта, равная 0,353 мм, или 1/72 дюйма.

– Style — стиль начертания символов, который может быть

обычным, полужирным (fsBold), курсивным (fsItalic), подчеркнутым (fsUnderline) и перечеркнутым (fsStrikeOut). В программе можно комбинировать необходимые стили, например, чтобы установить стиль «полужирный курсив», необходимо написать следующий оператор:

Объект.Canvas. Font. Style:=[fsItalic,fsBold]

– Color (тип Tcolor) — цвет символов.
– Charset (тип 0..255) — набор символов шрифта. Каждый вид шрифта, определяемый его именем, поддерживает один или более наборов символов. В таблице приведены некоторые значения Charset.

Таблица 7. Значения свойства Charset

Константа	**Значение**	**Описание**
ANSI_CHARSET	0	Символы ANSI
DEFAULT_CHARSET	1	Задается по умолчанию. Шрифт выбирается только по его имени Name и размеру Size. Если описанный шрифт недоступен в системе, будет заменен другим
SYMBOL_CHARSET	2	Стандартный набор символов
MAC_CHARSET	77	Символы Macintosh
RUSSIAN_CHARSET	204	Символы кириллицы

Класс TPEN

Карандаш (перо) используется как инструмент для рисования точек, линий, контуров геометрических фигур. Основные свойства объекта Canvas.Tpen:

– Color (тип Tcolor) – определяет цвет линии;
– Width (тип Integer) – задает толщину линии в пикселях;
– Style – дает возможность выбрать вид линии. Это свойство может принимать значение, указанное в таблице.

Таблица 8. Виды линий

Значение	**Описание**
psSolid	Сплошная линия
psDash	Штриховая линия
psDot	Пунктирная линия
psDashDot	Штрих-пунктирная линия
psDashDodDot	Линия, чередующая шрих и два пунктира
psClear	Нет линии

Mode – определяет, каким образом взаимодействуют цвета пера и полотна. Выбор значения этого свойства позволяет получать различные эффекты, возможные значения Mode приведены в таблице. По умолчанию вся линия вычерчивается цветом, определяемым значением Pen.Color, но можно определять инверсный цвет

линии по отношению к цвету фона. В этом случае независимо от цвета фона, даже если цвет линии и фона одинаков, линия будет видна.

Таблица 9. Возможные значения свойства Mode

Режим	**Операция**	**Цвет пикселя**
pmBlack	Black	Всегда черный
pmWhite	White	Всегда белый
pmNop	-	Неизменный
pmNot	Not Screen	Инверсный цвет по отношению к цвету фона
pmCopy	Pen	Цвет, указанный в свойствах Color пера Pen (это значение принято по умолчанию)
pmNotCopy	Not Pen	Инверсия цвета пера

Класс TBRUSH

Кисть (Canvas.Brush) используется методами, обеспечивающими вычерчивание замкнутых фигур для заливки. Кисть обладает двумя основными свойствами:

– Color (тип Tcolor) – цвет закрашивания замкнутой области;
– Style – стиль заполнения области.

Класс TCANVAS

Это класс является основным инструментом для рисования графики. Рассмотрим наиболее часто используемые методы этого класса.

Procedure MoveTo(X, Y : Integer);

Метод MoveTo изменяет текущую позицию пера на позицию, заданную точкой (X, Y). Текущая позиция хранится в переменной PenPos типа Tpoint. Определение типа Tpoint следующее:

type Tpoint =record
X: Longint;
Y: Longint;
end;

Текущую позицию пера можно считывать с помощью свойства PenPos следующим образом:

X:=PenPos.X;
Y:=PenPos.Y;

Procedure LineTo(X, Y :Integer);

Метод LineTo соединяет прямой линией текущую позицию пера и точку с координатами (X, Y). При этом текущая позиция пера перемещается в точку с координатами (X, Y).

Рассмотрим работу процедуры на примере. Расположим на форме кнопку и рассмотрим процедуру обработки события Tform1.Button1Click, которая рисует прямые линии:

```
Procedure Tform1.Button1Click(Sender: Tobject)
begin
Form1.Canvas.LineTo(30,50);
end;
```

В результате щелчка по кнопке на форме возникнет прямая линия, соединяющая точку с координатами (0,0) и точку с координатами (30,50). Теперь перепишем процедуру обработки события следующим образом:

```
Procedure Tform1.Button1Click(Sender: Tobject)
begin
Form1.Canvas.LineTo(Canvas.PenPos.x+30,
Canvas.PenPos.y+50);
end;
```

При первом щелчке по кнопке на экране прорисуется аналогичная линия. Но при повторном щелчке первая процедура продолжает рисовать линию, соединяющую точки (0,0) и (30,50). Вторая процедура рисует линию, которая соединяет текущую точку с точкой, получившейся из текущей добавлением к координате X числа 30, а к координате Y – числа 50. Т.е. при повторном щелчке по кнопке процедура соединяет прямой линией точки (30,50) и (60,100). При третьем щелчке по кнопке будут соединяться прямой линией точки (60,100) и (90,150) и т.д.

Procedure PolyLine(const Points array of Tpoint);

Метод PolyLine рисует ломаную линию, координаты вершин которой определяются массивом Points. Рассмотрим работу процедуры на примере. Расположим на форме

кнопки Рисовать и Выход и запишем следующие операторы процедур обработки события:

```
procedure Tform1.Button1Click(Sender: Tobject);
var temp :array [1..25] of Tpoint;
i : byte;
j: integer;
begin
j:=1;
for i:=1 to 25 do
begin
//вычисление координат вершин ломаной линии
temp[i].x:=25+(i-1)*10;
temp[i].y:=150-j*(i-1)*5;
j:=-j;
end;
Form1.Canvas.Polyline (temp);
end;
procedure Tform1.Button2Click(Sender: Tobject);
begin
Form1.Close;
end;
```

После запуска программы и щелчка по кнопке «Рисовать» окно формы будет выглядеть, как на рисунке.

Procedure Ellipse (X1, Y1, X2, Y2 :Integer);

Метод Ellipse вычерчивает на холсте эллипс или окружность. X1, Y1, X2, Y2 — это координаты прямоугольника, внутри которого вычерчивается эллипс. Если прямоугольник является квадратом, то вычерчивается окружность.

Рисунок 18. Пример использования процедуры PolyLine

Метод

Procedure Arc (X1,Y1,X2,Y2,X3,Y3,X4,Y4:Integer);

вычерчивает дугу эллипса. X1, Y1, X2, Y2 – это координаты, определяющие эллипс, частью которого является дуга. X3, Y3 —

координаты, определяющие начальную точку дуги, X4, Y4- координаты, определяющие конечную точку дуги. Дуга рисуется против часовой стрелки.

Метод

Procedure Rectangle (X1, Y1, X2, Y2 :Integer);

рисует прямоугольник. X1, Y1, X2, Y2- координаты верхнего левого и нижнего правого углов прямоугольника.

Метод

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

вычерчивает прямоугольник со скругленными углами. X1, Y1, X2,Y2- координаты верхнего левого и нижнего правого углов прямоугольника, а X3, Y3 – размер эллипса, одна четверть которого используется для вычерчивания скругленного угла.

Метод

Procedure PolyGon(const Points array of Tpoint);

рисует замкнутую фигуру (многоугольник) по множеству угловых точек, заданному массивом Points. При этом первая точка соединяется прямой линией с последней. Этим метод PolyGon отличается отметода Poliline, который не замыкает конечные точки. Рисованиеосуществляется текущим пером Pen, а внутренняя область фигуры

закрашивается текущей кистью Brush.

Метод

Procedure Pie (X1,Y1,X2,Y2,X3,Y3,X4,Y4 :Integer);

рисует замкнутую фигуру — сектор окружности или эллипса с помощью текущих параметров пера Pen, внутренняя область закрашивается текущей кистью Brush. Точки (X1, Y1) и (X2, Y2) задают прямоугольник, описывающий эллипс. Начальная точка дуги определяется пересечением эллипса с прямой, проходящей через его его

центр и точку (X3, Y3). Конечная точка дуги определяется пересечением эллипса с прямой, проходящей через его центр и точку (X4,Y4). Дуга рисуется против часовой стрелки от начальной до конечной точки. Рисуются прямые, ограничивающие сегмент и проходящие через центр эллипса и точки (X3, Y3) и (X4, Y4).

Функция

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

пишет строку текста Text, начиная с позиции с координатами (X,Y). Текущая позиция PenPos пера Pen перемещается на конец выведенного текста. Надпись выводится в соответствии с текущими установками шрифта Font, фон надписи определяется установками текущей кисти. Для выравнивания позиции текста на канве можно использовать методы, позволяющие определить высоту и длину текста в пикселях — TextExtent, TextHeight и TextWidth.

Рассмотрим эти функции.

Функция

function TextExtent (const Text : String): Tsize;

возвращает структуру типа Tsize, содержащую длину и высоту в пикселях текста Text, который предполагается написать на канве текущим шрифтом.

Type Tsize = record

cx:Longint; cy:Longint;

end;

Функция

function TextHeight(const Text:String):Integer;

возвращает высоту в пикселях текста Text, который предполагается написать на канве текущим шрифтом.

Литература

1. Фаронов В.В. Delphi. Программирование на языке высокого уровня: Учебник для вузов. - Спб.:Питер, 2005.-640 с.:ил.
2. Чеснокова О.В. Delphi 2007. Алгоритмы и программы. Учимся программировать на Delphi 2007/Чеснокова О.В. Под общ. ред. Алексеева Е.Р.- М.:НТ Пресс, 2008. - 368 с.:ил.
3. Free Pascal.ru - Информационный портал для разработчиков на Free Pascal & Lazarus & MSE. URL http://www.freepascal.ru.
4. Lazarus – News. URL: http://www.lazarus.freepascal.org.
5. Lazarus – Википедия. URL: http://ru.wikipedia.org/wiki/Lazarus.