Министерство образования Республики Беларусь

Учреждение высшего профессионального образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики и технологий программирования

Курсовая работа по дисциплине Программирование

Пояснительная записка

К курсовому проекту по теме

Электронный органайзер

Научный руководитель:

\_\_\_\_\_\_\_\_\_\_\_Сухарев А.А.

Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2016 г.

Выполнил:

Студент

ИиТП

Специальность\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Курс\_\_\_\_\_\_\_\_ Группа \_\_\_\_\_\_\_\_\_\_

очная

1

553505

Форма обучения \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Наривончик А.М.

Минск, 2016 год

Оглавление

[1. Задание на курсовой проект 3](#_Toc452474007)

[2. Введение 4](#_Toc452474008)

[2.1 Актуальность 4](#_Toc452474009)

[2.2 Цель работы 4](#_Toc452474010)

[2.3 Задачи 5](#_Toc452474011)

[3. Основная часть 6](#_Toc452474012)

[3.1 Теоретические сведения, относящиеся к ходу выполнения курсовой работы 6](#_Toc452474013)

[3.1.1 Визуальное программирование на Delphi 6](#_Toc452474014)

[3.1.2 Основы ООП в языке Delphi 7](#_Toc452474015)

[3.2 Ход выполнения курсового проекта 9](#_Toc452474016)

[3.2.1 Программная структура приложения . Главная форма. 9](#_Toc452474017)

[3.2.2 Планировщик задач 12](#_Toc452474018)

[3.2.3 Менеджер контактов 13](#_Toc452474019)

[3.2.4 Будильник 14](#_Toc452474020)

[3.3 Формат хранения данных 15](#_Toc452474021)

[3.3.1 Формат хранения будильников. 15](#_Toc452474022)

[3.3.2 Формат хранения событий. 16](#_Toc452474023)

[3.3.3 Формат хранения контактов. 17](#_Toc452474024)

[3.4 Сложности, возникшие при выполнении курсового проекта. 17](#_Toc452474025)

[Заключение 18](#_Toc452474026)

[Литература 18](#_Toc452474027)

# 1. Задание на курсовой проект

1. Изучить основные возможности языка программирования Delphi:

* Научиться работать с VCL Forms Application.
* Изучить базовые возможности ООП в языке Delphi.
* Научиться писать «совершенный код».

2. Реализовать практичный, с дружественным интерфейсом электронный органайзер, содержащий:

* Менеджер контактов (пользовательская телефонная книга)
* Будильник
* Учёт событий, привязанных к определённой дате и времени
* Сервис напоминания о событиях, заданных пользователем
* Календарь
* Калькулятор

# 2. Введение

## 2.1 Актуальность

Прогресс не стоит на месте. Устаревают оборудование и программы, материалы и технологии, даже образ мышления. В наше время никто и не подумает терять кучу времени на написание письма (впрочем это была замечательная традиция). Напечатать, бесспорно, быстрее, чем написать. И давно уже не следует переживать о почтовых марках и о том, что сведения, содержащиеся в письме, будут уже не актуальны к моменту их получения адресатом. Люди понемногу все дальше удаляются от бумажных носителей информации, и это естественно: электронные записи не занимают места, значительно удобнее в правке и работе и их не придется все время носить в сумках и портфелях.

Электронный органайзер – очень удобное изобретение, оно выделяется значительным преимуществом перед любыми бумажными носителями, сейчас это наилучшая замена блокноту. К примеру, важное преимущество – «электронная напоминалка». Можно забыть о тех временах, когда мы выпускали из виду важное событие или пропускали долгожданные свидания, концерты и встречи! С этого момента программа напоминание, настроенная лично Вами, загодя предупредит о важном событии.

Электронный органайзер очень практичен, сильные стороны его с каждым днем привлекают большое количество новых пользователей. Не теряйте драгоценные минуты на записи в бумажных блокнотах, в которых позже нереально что-нибудь найти! Лучше занесите все в [электронный ежедневник](http://www.organizeit.ru/articles/2.php). И тогда можно свободно отдыхать, не волнуясь, что пропустишь какое-либо важное мероприятие.

Ещё одним из удобств электронного органайзера является будильник – ценная и незаменимая вещь для любого современного человека. Также Электронный органайзер может хранить список всех ваших контактов, они всегда будут находиться под рукой.

## 2.2 Цель работы

1. Рассмотреть основные возможности языка Delphi.
2. Научиться создавать программный продукт по разработанному алгоритму;
3. Научиться выполнять отладку и тестирование программного продукта;
4. Научиться применять математические методы для решения задач
5. Уметь осуществлять настройку программных продуктов;
6. Научиться оформлять программную документацию;

## 2.3 Задачи

1. Изучить литературу по основам языка Delphi:

* Научиться работать с VCL Forms Application
* Изучить основы ООП программирования
* Научиться писать качественный гибкий код

1. Найти и изучить специализированные информационные ресурсы по теме курсового проекта.
2. Написать полноценную программу с дружественным интерфейсом.
3. Написать пояснительную записку по итогам выполнения курсового проекта.

# 3. Основная часть

## 3.1 Теоретические сведения, относящиеся к ходу выполнения курсовой работы

### 3.1.1 Визуальное программирование на Delphi

***Интегрированная среда разработки***

Создание прикладных программ, или приложений, в Delphi выполняется в интегрированной среде разработки IDE (Integrated Development Environment). IDE служит для организации взаимодействия с программистом и включает в себя ряд окон, содержащих различные элементы управления. С помощью средств интегрированной среды разработчик может проектировать интерфейсную часть приложения, а также писать программный код, и связывать его с управляющими элементами. При этом вся работа по созданию приложения, включая отладку, происходит в IDE. 

**Главные составные части среды программирования**  
 Интегрированная среда программирования Delphi состоит из следующих основных частей:

1. Главное окно
2. Дизайнер форм (Form Designer)
3. Редактор исходного текста (Editor Window)
4. Палитра компонентов (Component Palette)
5. Инспектор объектов (Object Inspector)

В главном окне отображается главное меню, панель инструментов и палитра компонентов. Главное меню содержит обширный набор команд для доступа к функциям Delphi. Панель инструментов находится под Главным меню в левой части Главного окна и содержит 15 кнопок быстрого доступа для вызова часто используемых команд Главного меню. Палитра компонентов находится справа от Панели инструментов и содержит множество компонентов, размещаемых в создаваемых приложениях. Все компоненты делятся на группы, каждая из которых в Палитре компонентов располагается на отдельной странице (Standard, Additional, Win 32 и др.).

Дизайнер форм в Delphi служит для создания визуального интерфейса программы. Дизайнер Форм первоначально состоит из одного пустого окна, которое заполняется компонентами, выбранными в палитре компонентов.   
Редактор исходного текста (редактор кода) представляет собой текстовый редактор, с помощью которого можно создавать и редактировать программный код.   
 Инспектор объектов состоит из двух страниц, каждую из которых можно использовать для определения поведения данного компонента. Первая страница – это список свойств, вторая – список событий. Если нужно изменить что-нибудь, связанное с определенным компонентом, то обычно делается это в инспекторе объектов. Страница событий связана с Редактором кода. Если дважды щелкнуть мышкой на правую сторону какого-либо пункта, то соответствующий данному событию код автоматически запишется в редактор. Сам редактор немедленно получит фокус, и появится возмо-жность добавить код обработчика данного события.

### 3.1.2 Основы ООП в языке Delphi

**ООП** – это парадигма программирования, разновидность императивного программирования, согласно которой программа строится как набор взаимодействующих объектов, причем язык поддерживает как описание таких объектов, так и описание всех их особенностей (сходства и различия разновидностей, все виды взаимодействия между ними).

Таким образом, согласно определению ООП, программа должна состоять только из объектов, следовательно, объекты должны уметь делать все то, что умеют делать обычные программы. Поскольку программы могут принимать данные, производить вычисления и получать результат, объекты должны уметь делать две вещи: хранить данные (некоторую их часть, непосредственно относящуюся к этому объекту) и обрабатывать эти данные (опять же, каждый объект умеет обрабатывать свои данные, а если требуется участие других объектов, вступать с ними во взаимодействие). Это приводит нас к первому из трех принципов ООП – инкапсуляции.

**Инкапсуляция** – принцип ООП, согласно которому в объектах объединены и данные, и методы их обработки. Данные класса хранятся в полях класса, подпрограммы для работы с полями называются методами класса.

**Наследование** – принцип ООП, позволяющий описывать разновидности одного и того же класса объектов таким образом, что все эти разновидности автоматически обладают общими для них свойствами и умеют выполнять общие для них операции, однако могут дополнить этот набор специфическими для этой разновидности свойствами и операциями. При этом любую разновидность можно рассматривать и как конкретную разновидность, и как объект общего для них класса. Общий класс в случае наследования называется базовым классом, классом-предком, классом-родителем или суперклассом; класс, описывающий разновидность, называется подклассом, классом-потомком или классом-наследником. При помощи наследования можно строить так называемое дерево классов (или иерархию классов), последовательно уточняя описание класса и переходя от общих понятий к частным.

**Полиморфизм** (в смысле ООП) – принцип ООП, согласно которому разновидности объектов могут реализовывать общие для разновидностей операции разными способами. Для такой возможности придуманы виртуальные методы. Они функционируют по следующей схеме. Каждый объект, наряду со значениями своих полей, хранит указатель на специальную таблицу виртуальных методов (virtual method table, VMT). Таблица виртуальных методов индивидуальна и единственна для каждого класса. В ней хранятся адреса всех виртуальных методов класса (как собственных, так и унаследованных). Связь между объектом и VMT класса осуществляется во время начальной инициализации объекта, то есть при вызове конструктора. Виртуальные методы идентифицируются по константе-смещению в VMT. Во время выполнения программы из экземпляра объекта извлекается указатель на VMT и, используя константу-смещение, вычисляется адрес необходимого метода.

Одним из желательных принципов ООП являются принципы сокрытия реализации и сохранения целостности. Поля классов являются по сути обычными переменными, и их можно свободно изменять, однако это противоречит данным принципам. Соответственно, желательно, чтобы язык программирования позволял предпринять меры по ограничению такого свободного использования объектов. Это достигается с помощью специальной концепции, называемой областями видимости. Любой член класса можно снабдить информацией, указывающей, какая у него будет область видимости. Области видимости, предусмотренные в языке Object Pascal, следующие: **strict private, public, protected, private, published, automated.**

Поля класса обычно делаются в области видимости private. Это позволяет полностью отдать управление полями на откуп самому объекту. Остальные члены класса могут быть как общедоступными, так и служебными. Соответственно, они, в зависимости от логики, могут располагаться в любой области видимости.

Перед тем как пользоваться объектами, нужно их создать. Еще одним отличием ООП от обычного процедурного программирования является то, что объекты не существуют с самого начала работы программы, а создаются и уничтожаются по мере надобности. (Если в программе все объекты создаются в начале и уничтожаются в конце, то это опять-таки признак того, что программа, скорее всего, не подходит для применения ООП). За создание объектов отвечает специальный метод, называемый ***конструктором***, а за уничтожение – специальный метод, называемый ***деструктором***.

Конструктор, также как и метод, имеет заголовок и тело. Тело конструктора также пишется после объявления класса. Конструктор выполняет две задачи: сначала он находит свободное место в динамической памяти, занимает это место под поля объекта; затем он выполняет свое тело, которое, как правило, выполняет начальную инициализацию его полей (поля, как и обычные переменные, должны быть инициализированы перед их использованием).

Деструктор также выполняет две функции: сначала, если удаляемый объект отвечал за управление какой-то частью системы, он оповещает другие объекты о том, что в связи с удалением он «прекращает исполнять свои обязанности». Затем деструктор автоматически освобождает память, которую занимал объект, то есть говорит программе, что эта память снова свободна.

Развитие концепций ООП привело к появлению понятия **свойства** (property). Свойство – это элемент класса, работа с которым происходит так же, как с полем объекта. Разница между полем и свойством заключается в следующем: обращение к свойству компилятор транслирует в обращение к полю или в вызов метода, следовательно, при работе со свойствами могут выполняться некоторые действия.

Подчеркнём, что свойства класса призваны облегчить работу с объектом для пользователя. Фактически, свойства «живут» только до компиляции программы, во время которой заменяются методами или полями. В отличие от полей свойства не занимают места в памяти. Это накладывает определённые ограничения на их использование. Свойства нельзя передавать в качестве **var** - параметров в подпрограммы, к ним нельзя применять операцию взятия адреса.

***Событие*** — это свойство процедурного типа, предназначенное для создания пользовательской реакции на то или иное входное воздействие. Каждое событие должно сопровождаться методом – обработчиком события, который начинает выполнять действие или блок действий при вызове события.

## 3.2 Ход выполнения курсового проекта

### 3.2.1 Программная структура приложения . Главная форма.

Начало работы приложения начинается с формы приветствия(Greeting), отображающей текущую дату.

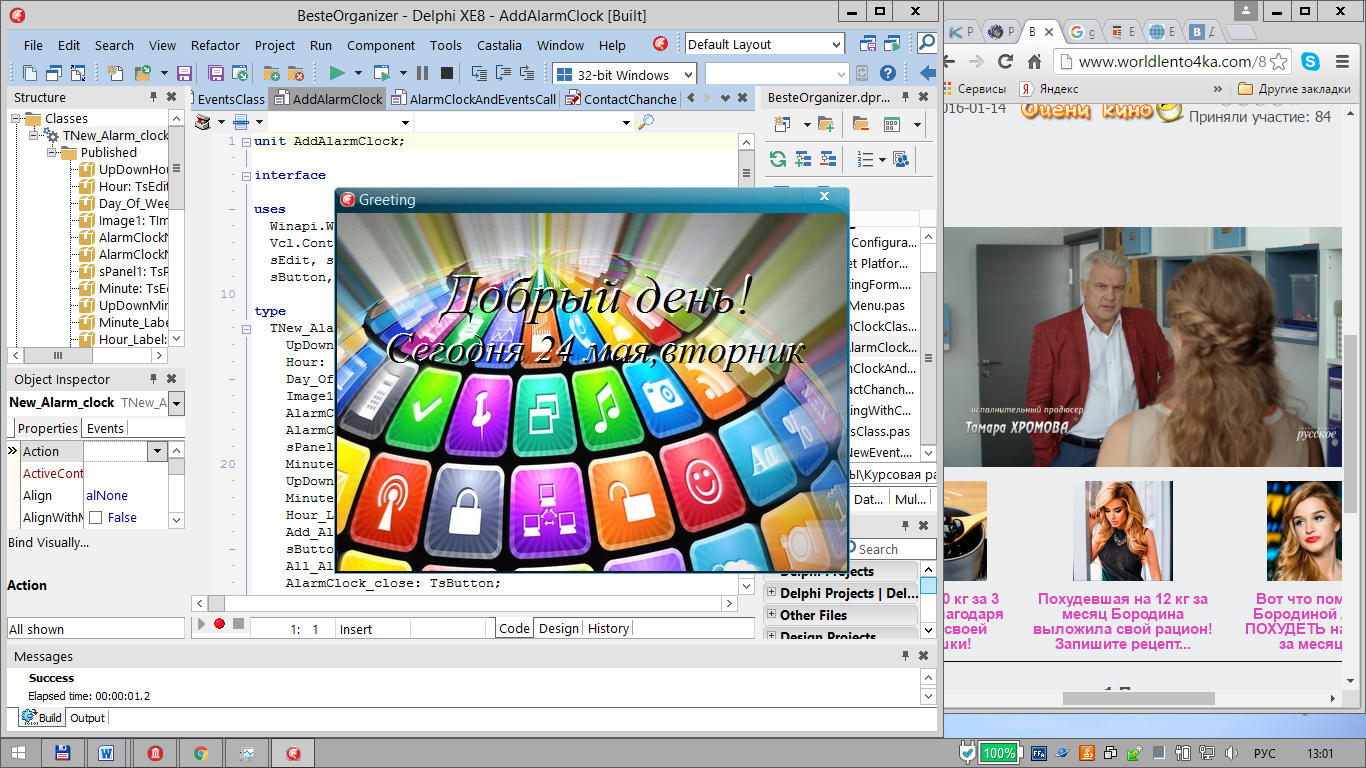


Рис.1. Форма приветствия

В приложении используется главная форма с главным меню(Organizer), обеспечивающим переход ко всем функциям приложения *Органайзер (см. рис 2).*

Осуществляется доступ к форме, используемой планировщиком задач для добавления событий (New\_Event). Просмотр всех событий и событий по конкретной дате осуществляется также на главной форме.

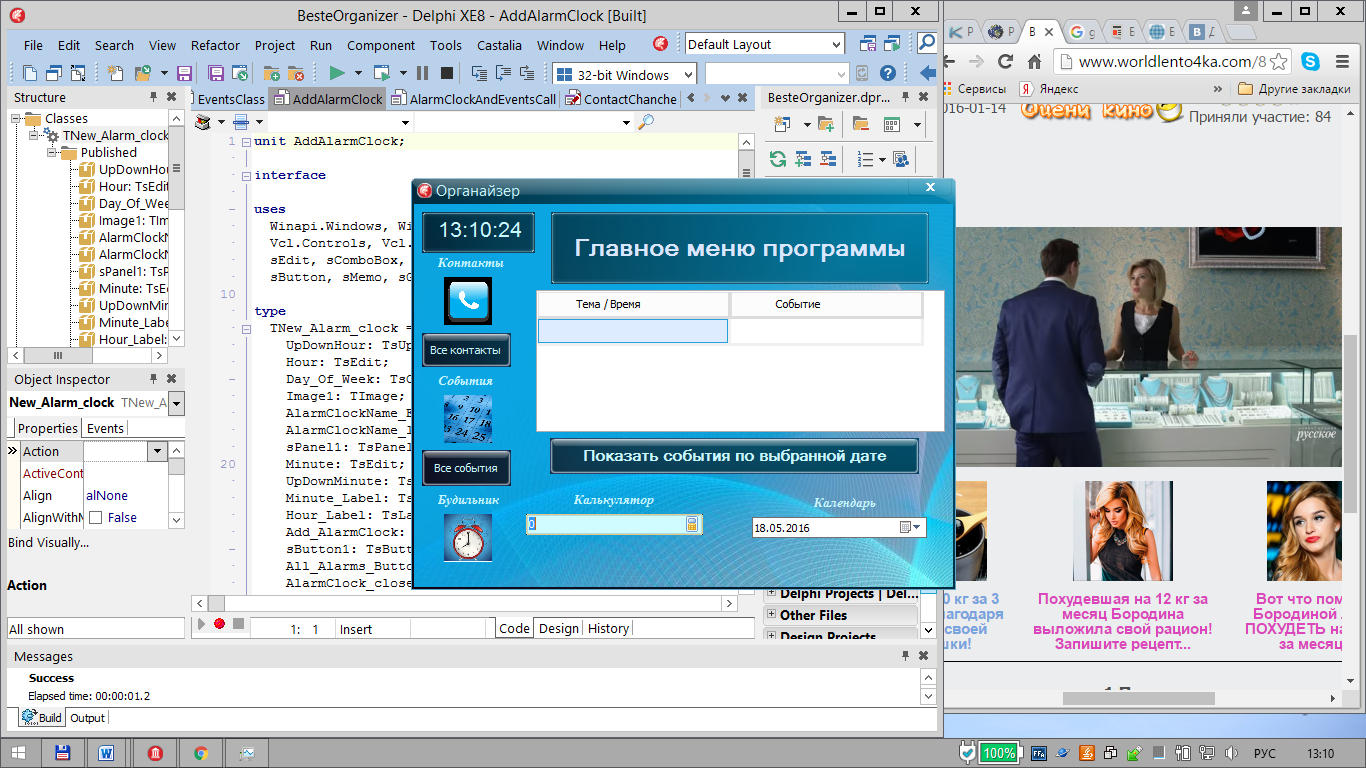


Рис.2. Главное меню программы

С главной формы осуществляется переход на форму для добавления и редактирования контактов (ViewAndEditContacts), также существует возможность просмотреть все контакты пользователя.

С главного меню можно получить доступ к форме конструктора нового будильника (New\_Alarm\_Clock).

#### 3.2.1.1 Список текущих событий

Список текущих событий отображается в компоненте **StringGrid**, один из столбцов которого содержит данные о теме и времени события, а во вто-ром столбце содержится описание события *(см. рис 4).* В программе сущест-вует возможность просмотреть список всех событий, а также можно просмотреть список событий на выбранную дату.

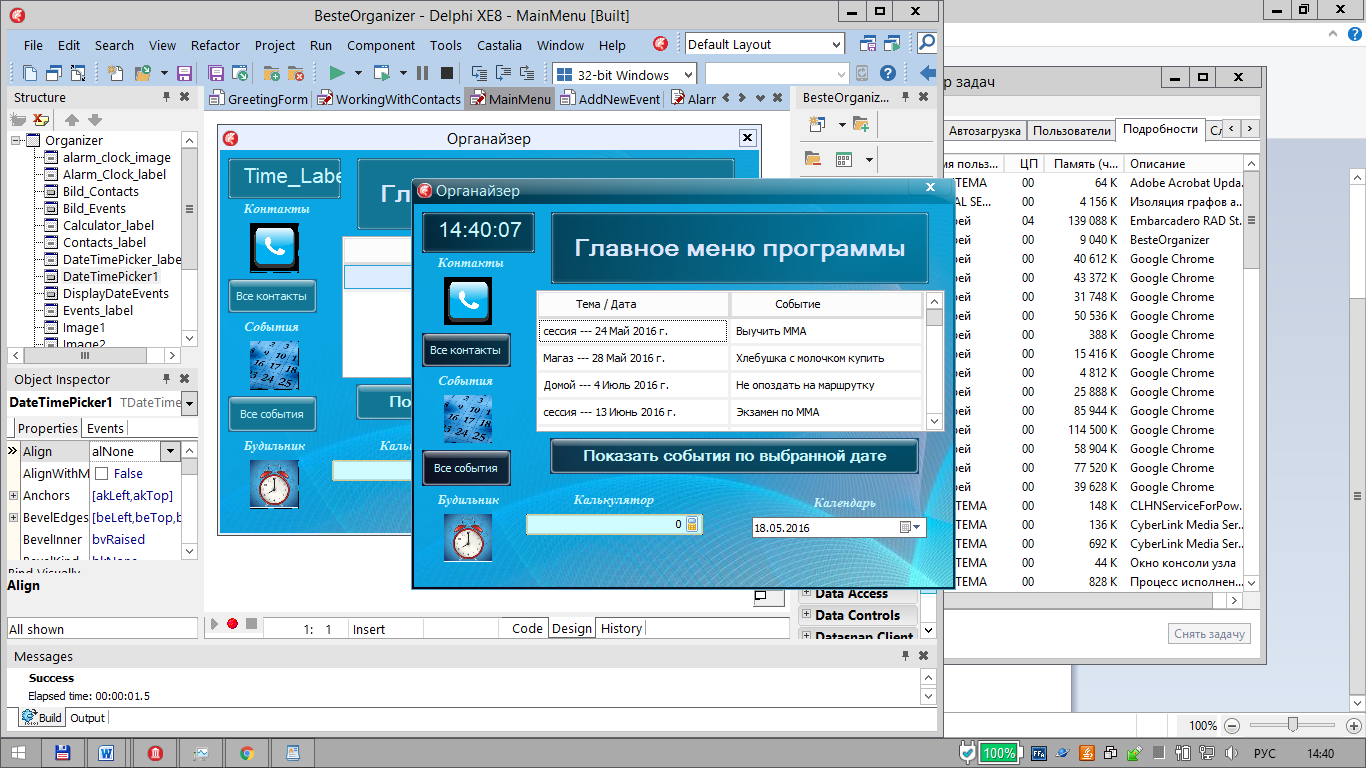


Рис.4. Таблица событий

#### 3.2.1.2 Календарь

В качестве календаря для главной формы приложения был взят компонент Delphi **DateTimePicker** *(см. рис. 5).*

Компонент **DateTimePicker** используется для быстрого выбора даты с помощью мыши. Сразу после размещения в форме календарь готов к работе.

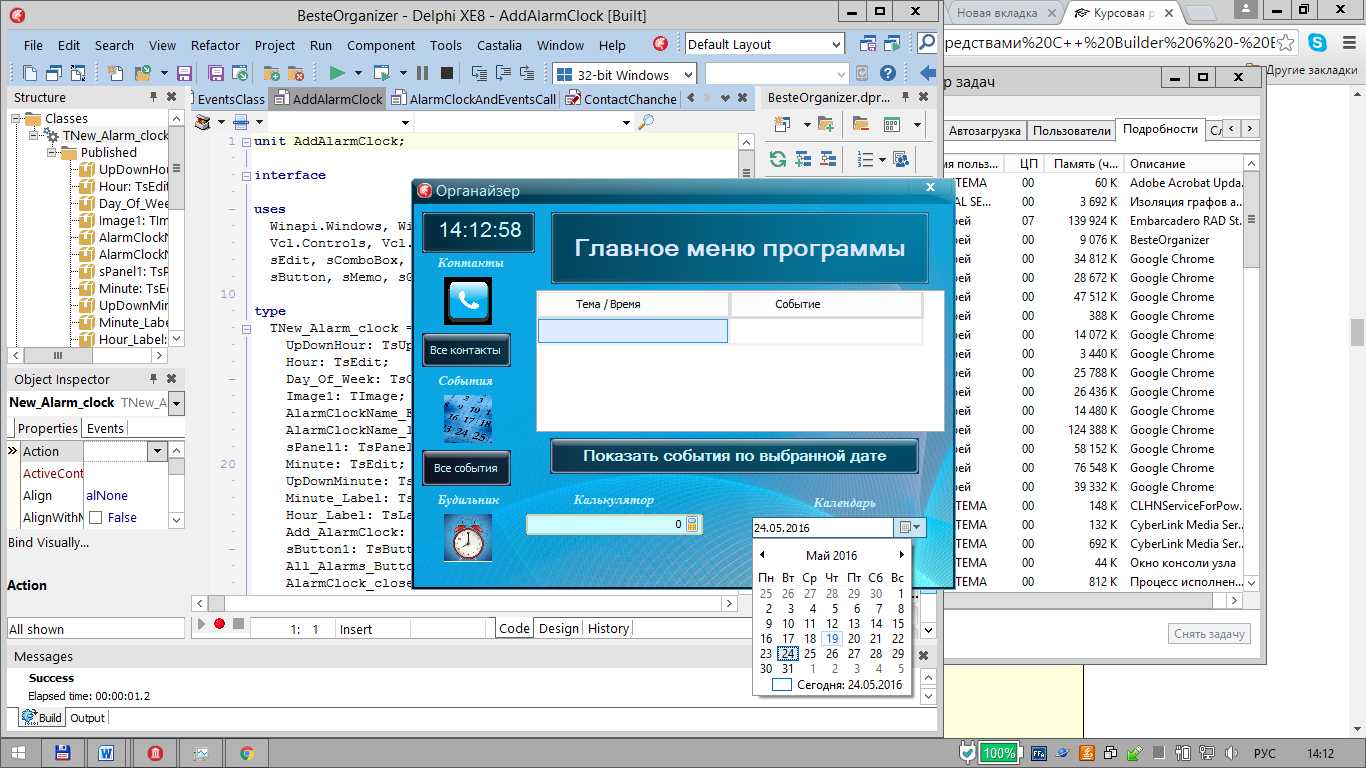


Рис.5. Календарь на форме

Синим прямоугольником выделена текущая дата (она же подробнее указана в нижней части календаря), синим подсвечена дата, выбранная с помощью мыши или дата, устанавливаемая в календаре по умолчанию. С помощью кнопок в верхней части календаря можно перемещаться по месяцам. Чтобы вернуться к месяцу с текущей датой, достаточно щелкнуть на ней в нижней части календаря.

Календарь выполняет одну из основных функций приложения: способствует просмотру событий на конкретную дату.

#### 3.2.1.3 Калькулятор

Калькулятор представляет собой стандартный компонент **TsCalcul-ator** *(см. рис 6).* Это удобная вещь для быстрого вычисления стандартных математических операций.

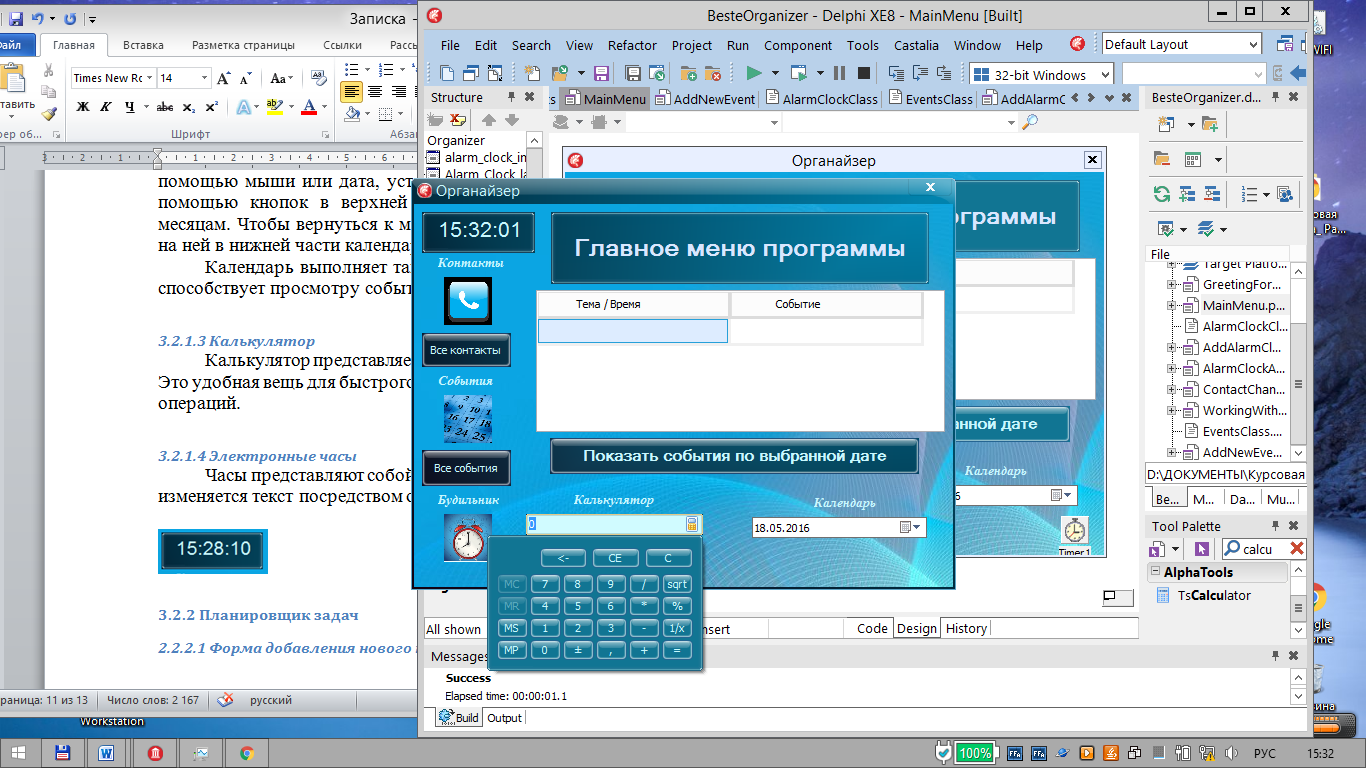


Рис.6. Калькулятор на форме

#### 3.2.1.4 Электронные часы

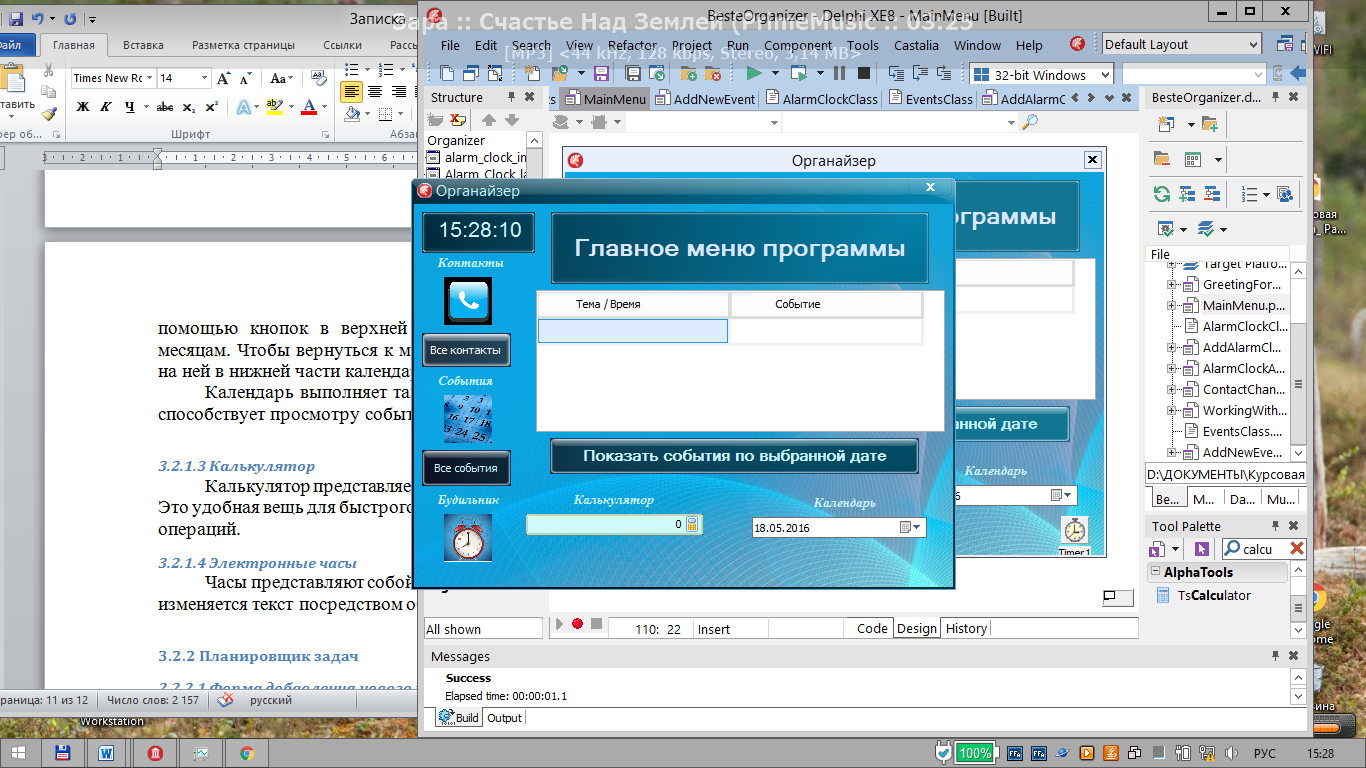
 Часы представляют собой поле **Label**, в котором каждую секунду изменяется текст посредством обработчика события таймера **OnTimer** *(см. рис 7)*.

Рис.7. Электронные часы на форме

### 3.2.2 Планировщик задач

#### 2.2.2.1 Форма добавления нового напоминания

Планировщик задач представляет собой форму, которая имеет функционал, позволяющий добавлять новые события и удалять старые (см. рис 8).

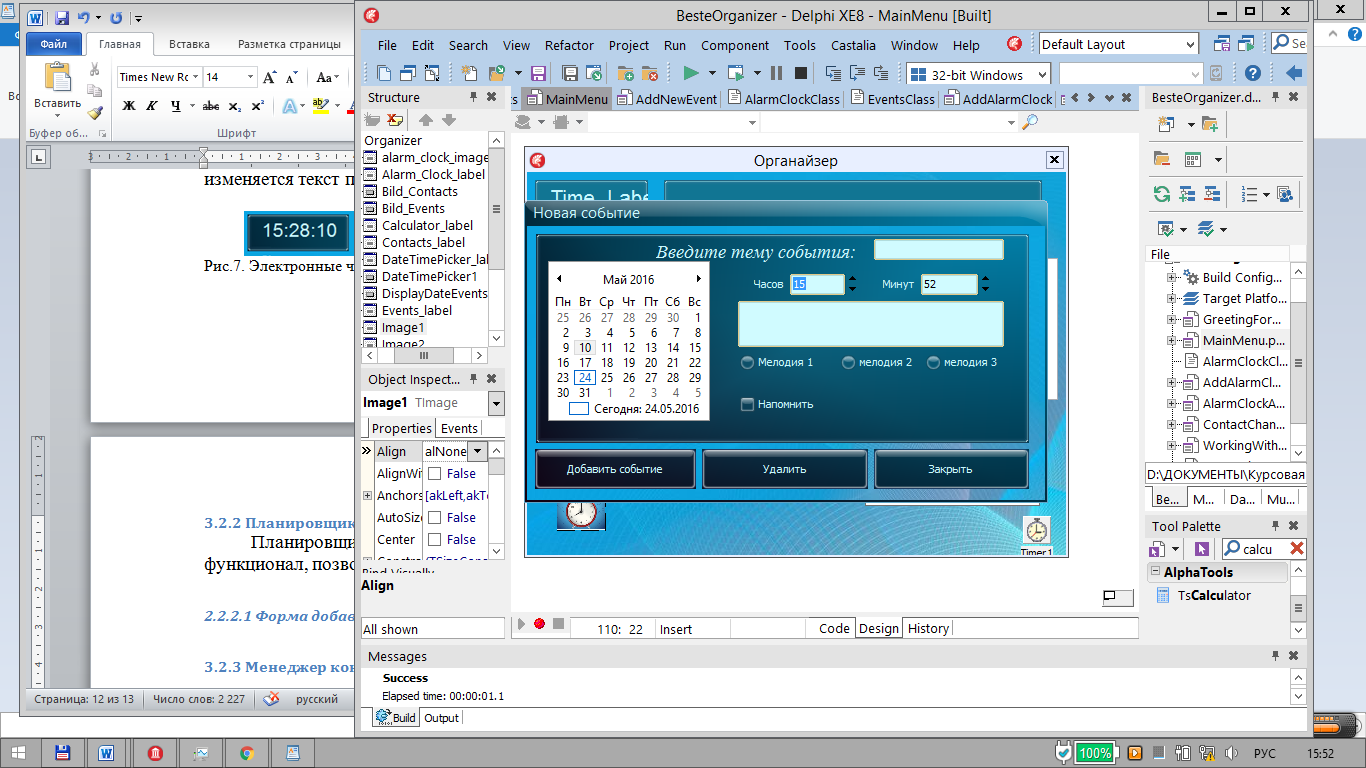


Рис.8. Планировщик задач

На форме имеется поле **Edit**, в котором пользователь вводит тему события, поле **Memo** для ввода более подробной информации о том или ином событии. Также имеются поля **Edit** в которых вводится время напоминания о событии. Регулировать время помогают компоненты **UpDown**, один для регулирования часов, второй – для минут. Дату события выбираем в календаре – компонент **TMonthCalendar**.

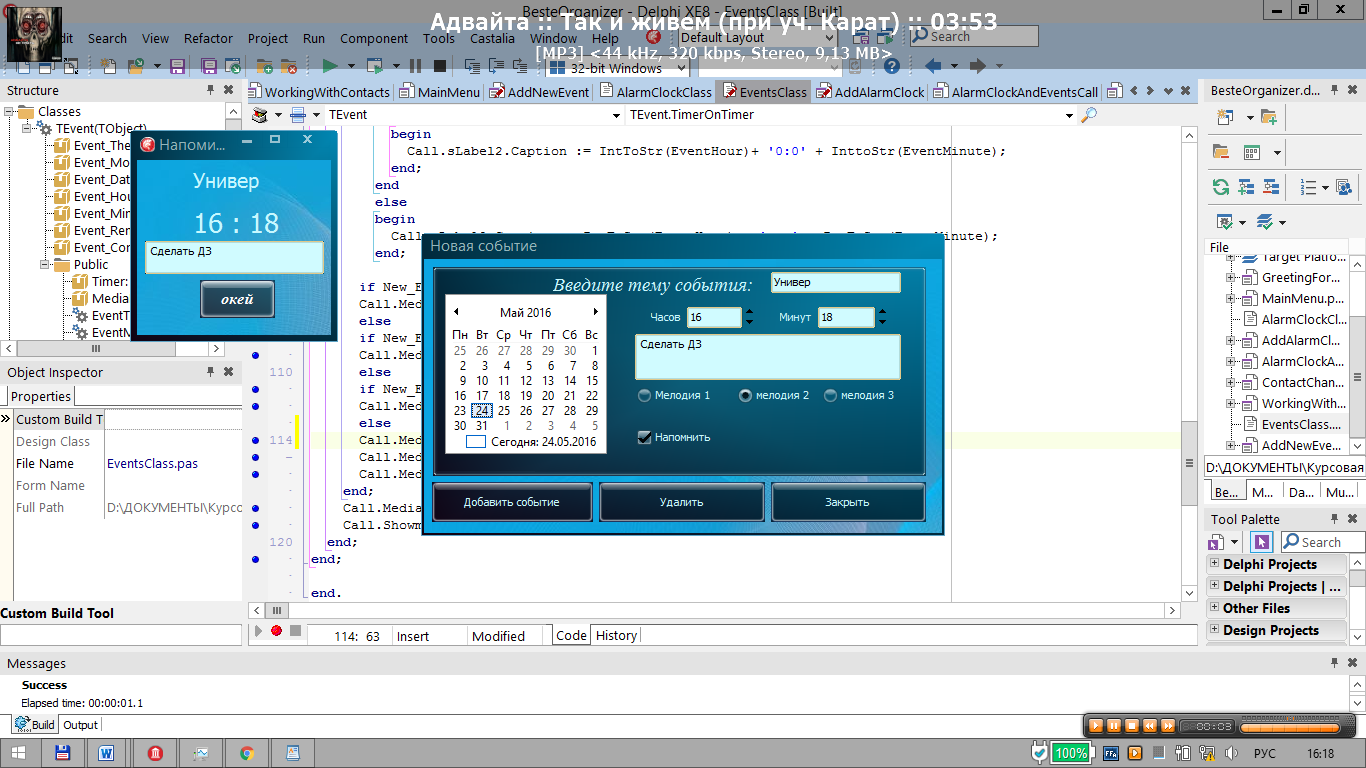
 Ещё одна из возможностей, которая предоставляется пользователю – это выбор мелодии напоминания, имеется 3 варианта выбора посредством компонентов **CheckBox**. Пользователь также вправе отказаться от звукового напоминания, вместо этого он может непосредственно просматривать события в любое комфортное ему время. Если же пользователь согласен с звуковым напоминанием, то в нужное время появится форма (см. рис. 9), на которой отображена тема события, время и подробное описание события. Появление формы сопровождается проигрыванием мелодии, выбранной пользователем при добавлении события. При нажатии на кнопку «окей» форма закрывается и мелодия останавливается, что свидетельствует о том, что пользователь вспомнил о текущем событии.

Рис.9. Окно напоминания

Также пользователь вправе удалять события по своему усмотрению, для этого в планировщике задачи есть кнопка **Button** с событием **OnClick.** Пользователь должен ввести тему события и его дату и кликнуть на кнопку «Удалить».

### 3.2.3 Менеджер контактов

#### 3.2.3.1 Форма работы добавления, удаления контактов

Менеджер контактов представляет собой форму (см. рис. 10), которая имеет функционал, позволяющий добавлять новые контакты, удалять старые и переходить на форму редактирования контактов.

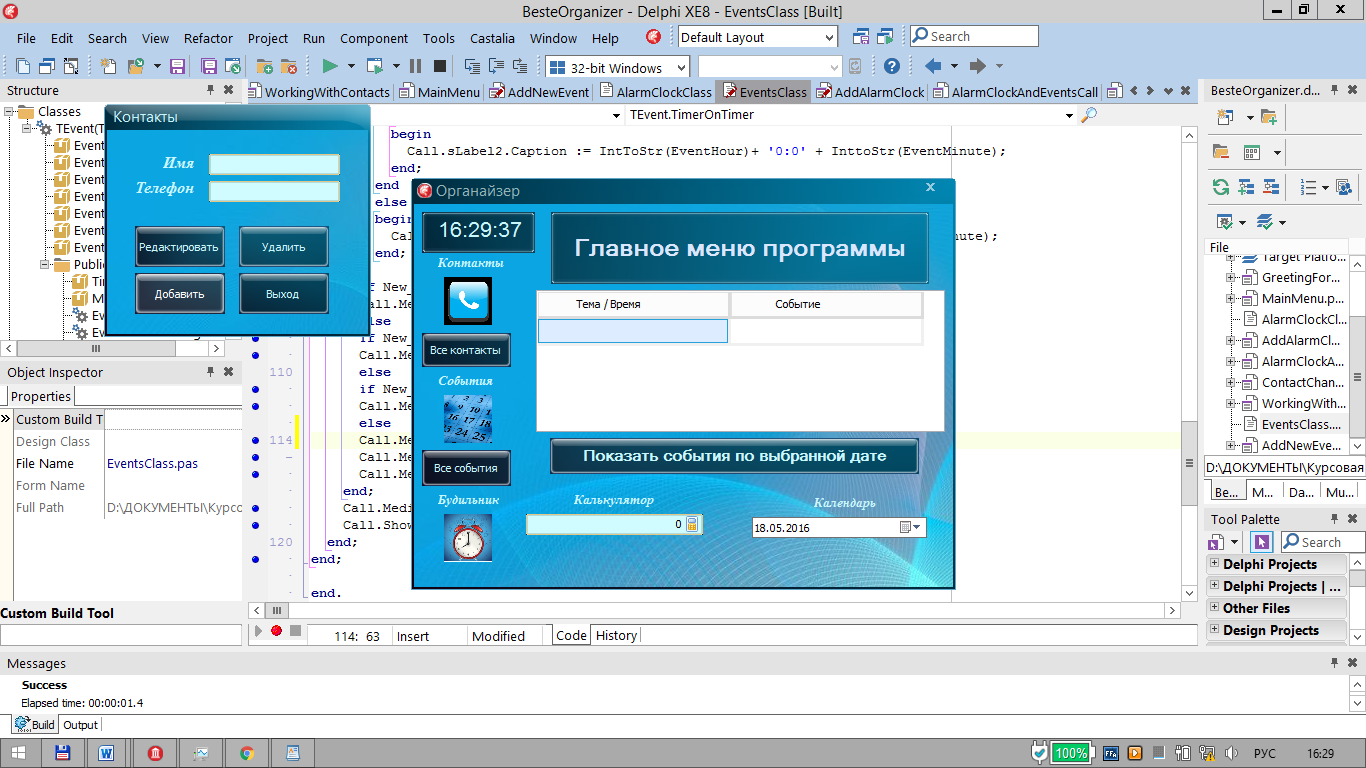


Рис.10. Менеджер контактов

Для добавления нового контакта необходимо ввести имя контакта и непосредственно сам номер в два соответствующих поля **Edit**. Чтобы удалить контакт, нужно ввести его имя и номер в соответствующие поля и нажать кнопку «удалить».

При нажатии на кнопку «редактировать» открывается новое окно редактора контактов.

#### 3.2.3.2. Форма редактирования контактов

Форма редактирования контактов представляет собой вид, предложенный на картинке ниже (см. рис. 11).

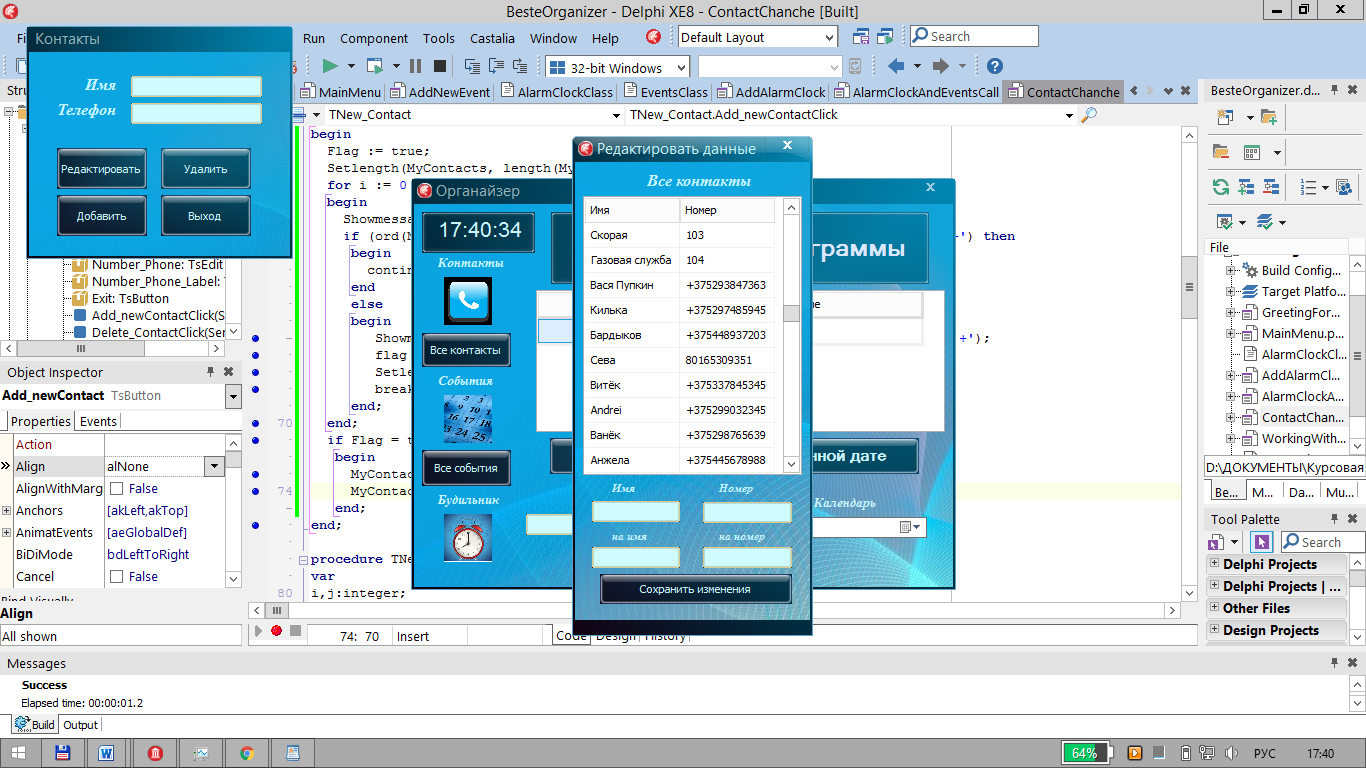


Рис.11. Редактор контактов

Чтобы изменить контакт, нужно ввести старое имя и старый номер, а также новое имя и новый номер и нажать на кнопку «Сохранить изменения». Если совпадение найдено, то произойдёт изменение в списке контактов.

### 3.2.4 Будильник

#### 3.2.4.1 Форма для создания новых будильников и редактирования старых

Чтобы открыть форму для создания и редактирования будильников, необходимо кликнуть на картинку с изображением будильника. Форма редактора будильника имеет следующий вид (см. рис. 12).

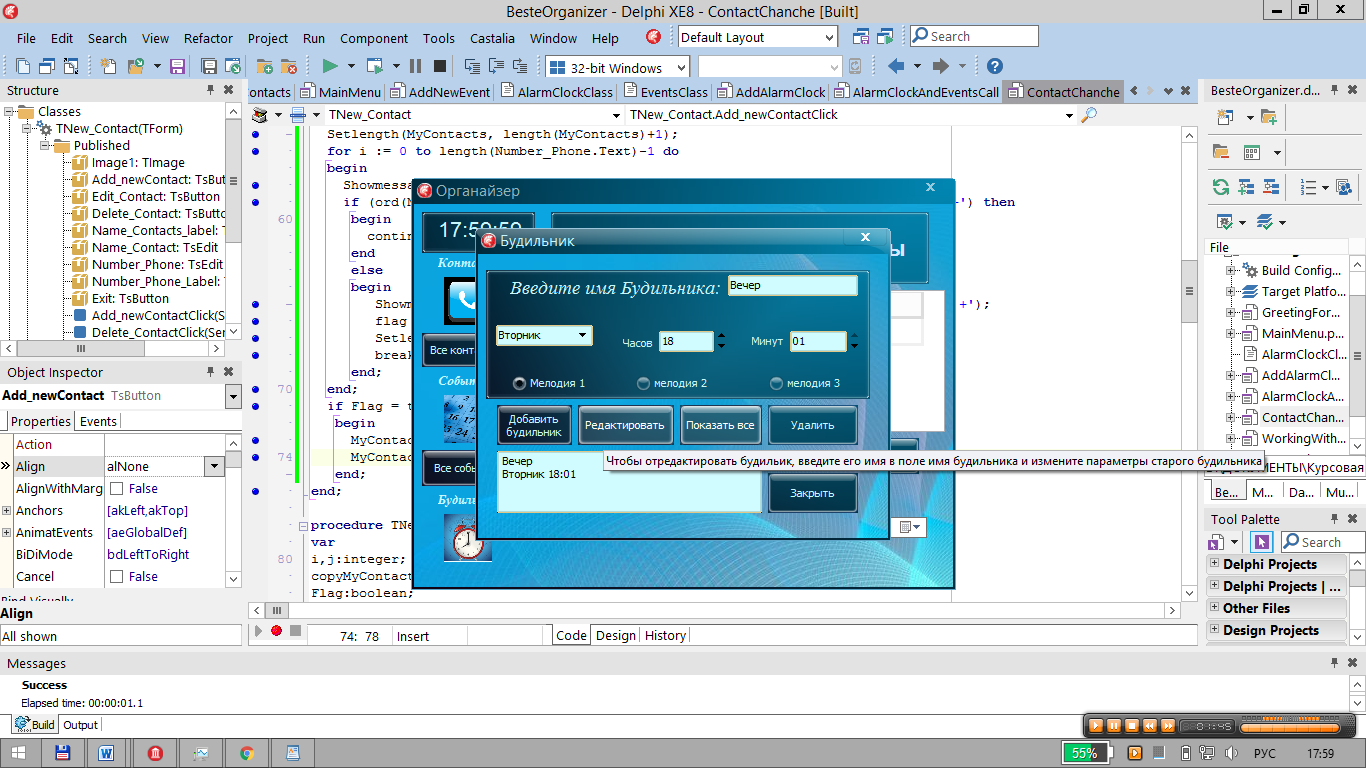


Рис.12. Конструктор будильника

Чтобы добавить будильник, необходимо ввести его имя в поле **Edit**, выбрать день недели из предложенного списка компонента **ComboBox,** выбрать необходимое время будильника. Также есть возможность выбора мелодии будильника.

На форме есть кнопка «показать все», эта кнопка отображает все действующие на данный момент будильники в компоненте **Memo.**

Любой из будильников можно удалить в случае необходимости.

Редактирование будильника осуществляется по его имени, у будильника с искомым именем изменяются его текущие параметры на новые.

#### 3.2.4.2 Форма для работы со сработавшим будильником

На форме сработавшего будильника отображается имя будильника и его время срабатывания, параллельно играет мелодия, выбранная пользова-телем (см. рис. 13).

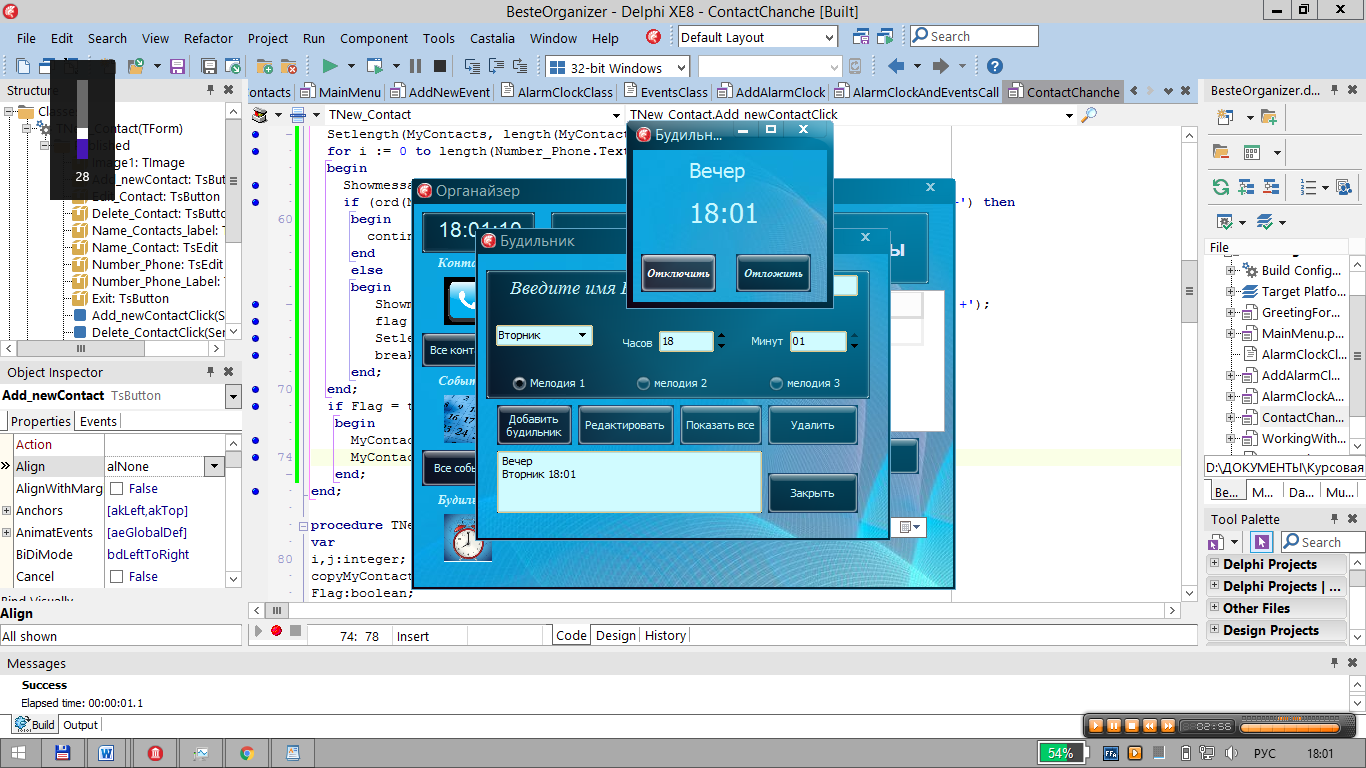


Рис. 13. Сработавший будильник

У сработавшего будильника есть 2 варианта развития его дальнейшей судьбы:

1. Отключение – для будильника, объекта, вызывается его деструктор. О формате хранения данных будет сказано чуть позже.
2. Отложить – будильник сработает очередной раз через 5 минут.

## 3.3 Формат хранения данных

### 3.3.1 Формат хранения будильников.

Каждый будильник представляет собой объект класса типа **TAlarm\_Clock**.

**Type**

TAlarm\_Clock = **class**

**private**

AlarmClockName:string;

AlarmClockDay:string;

AlarmClockHour:integer;

AlarmClockMinute:integer;

**public**

Timer:TTimer;

**property** Alarm\_Clock\_Name:string read AlarmClockName write AlarmClockName;

**property** Alarm\_Clock\_Day:string read AlarmClockDay write AlarmClockDay;

**property** Alarm\_Clock\_Hour:integer read AlarmClockHour write AlarmClockHour;

**property** Alarm\_Clock\_Minute:integer read AlarmClockMinute write AlarmClockMinute;

**constructor** create(newAlarm\_Clock\_Name, newAlarm\_Clock\_Day:string; newAlarm\_Clock\_Hour, newAlarm\_Clock\_Minute:integer);

**Procedure** TimerOnTimer(Sender:TObject);

**end;**

Каждый объект имеет **Private** поля, доступ к которым осуществляется через свойства:

AlarmClockName:string; //Имя будильника

AlarmClockDay:string; //День недели

AlarmClockHour:integer; //Часы

AlarmClockMinute:integer; //Минуты

Инициализация полей осуществляется в конструкторе, формальными параметрами которого являются данные, введённые пользователем на форме конструктора будильника.

**constructor create(newAlarm\_Clock\_Name, newAlarm\_Clock\_Day:string; newAlarm\_Clock\_Hour, newAlarm\_Clock\_Minute:integer);**

Также у каждого будильника есть объект таймер. Обработчиком события которого является метод: **Procedure TimerOnTimer(Sender:TObject);** Обработчик события отвечает за срабатывание будильника. Интервал таймера составляет одну секунду. При совпадении времени, заданного будильнику с текущим временем таймер останавливается и начинает играть мелодия.

### 3.3.2 Формат хранения событий.

Каждое событие представляет собой объект класса типа **TEvent**.

**Type**

TEvent = **class**

**private**

Event\_Theme:string;

Event\_Month:string;

Event\_Date:integer;

Event\_Hour:integer;

Event\_Minute:integer;

Event\_Reminder:boolean;

Event\_Content:String;

**public**

Timer:TTimer;

**constructor** create(newEventTheme, newEventMonth:string; newEventDate, newEventHour, newEventMinute:integer; newEventReminder:boolean; newEventContent:String);

**property** EventTheme: string read Event\_Theme write Event\_Theme;

**property** EventMonth:string read Event\_Month write Event\_Month;

**property** EventDate:integer read Event\_Date write Event\_Date;

**property** EventHour:integer read Event\_Hour write Event\_Hour;

**property** EventMinute:integer read Event\_Minute write Event\_Minute;

**property** EventReminder:boolean read Event\_Reminder write Event\_Reminder;

**property** EventContent:String read Event\_Content write Event\_Content;

**Procedure** TimerOnTimer(Sender:TObject);

**end;**

Каждый объект имеет **Private** поля, доступ к которым осуществляется через свойства:

Event\_Theme:string; // Тема события

Event\_Month:string; // Месяц события

Event\_Date:integer; // Число события

Event\_Hour:integer; // Час события

Event\_Minute:integer; // Минута события

Event\_Reminder:boolean; // Флаг, говорящий о необходимости напомнить

Event\_Content:String; // Полное описание события

Инициализация полей осуществляется в конструкторе, формальными параметрами которого являются данные, введённые пользователем в форме конструктора событий.

**constructor create(newEventTheme, newEventMonth:string; newEventDate, newEventHour, newEventMinute:integer; newEventReminder:boolean; newEventContent:String);**

Также у каждого события есть объект таймер. Обработчиком события которого является метод: **Procedure TimerOnTimer(Sender:TObject);**  Обработчик события отвечает за срабатывание напоминания в случае, если пользователь в этом нуждается.Интервал таймера составляет одну секунду. При совпадении времени, заданного будильнику с текущим временем таймер останавливается и начинает играть мелодия.

Множество событий пользователя во время работы приложения хранятся в динамическом массиве объектов класса типа **TEvent**. При открытии приложения события, которые грядут в будущем считываются из файла “Events”, который находится в папке **Debug**, в массив, а события, которые уже имели место быть игнорируются. При закрытии приложения файл открывается для перезаписи (**rewrite**) и в него сохраняются свежие события.

### 3.3.3 Формат хранения контактов.

Каждый контакт представляет собой запись, состоящую двух полей типа **string:** имя контакта и номер телефона контакта.

**type**

Contact = **record**

Contact\_Name:string;

Contact\_Number:string;

**end;**

Множество контактов пользователя во время работы приложения хранятся в динамическом массиве, а чтобы их не потерять, они записаны в файле “Contacts”, который находится в папке Debug, откуда они и считываются при открытии приложения.

**var**

MyContacts: **array of** Contact;

## 

## 3.4 Сложности, возникшие при выполнении курсового проекта.

В ходе выполнения курсового проекта возникали трудности различного рода. Трудности с разработкой эскиза будущего проекта, с распределением частей проекта по их функционалу в различных модулях, разработкой оптимальных алгоритмов. Также возникали вопросы с наиболее подходящим форматом хранения данных, в результате чего во время работы приложения контакты было решено хранить в массиве записей, а события и будильники в массиве объектов соответствующих классов. А для сохранения данных при выходе из приложения всю информацию было решено хранить в текстовых файлах, которые располагаются в папке с **EXE** файлом проекта. В работе приходилось осуществлять большое количество проверок на вводимые пользователем данные. Для решения всех этих проблем было изучено множество электронных источников, что поспособствовало реализации хорошего проекта.

# Заключение

В результате проделанной курсовой работы был создан электронный органайзер "Beste Organizer". Полученное приложение можно использовать в качестве ежедневника для записей контактов и событий. Электронный ежедневник содержит удобный планировщик задач с напоминалкой. В планировщике задач есть возможность просмотра событий по конкретной дате, что поможет еще лучше спланировать свой рабочий график. Ещё одной полезной вещью полученного оганайзера является будильник – важная и незаменимая вещь для каждого человека. В общем и целом проект получился довольно хорошим и полезным для повседневной жизни.

# 

# Литература

1. Архангельский, А. Delphi 2006. Справочное пособие. Язык Delphi, классы, функции Win32 и .NET / А. Я. Архангельский; – СПб. : Бином-Пресс, 2009. – 1152 с. : ил.

2. Кэнту, М. Delphi 2005. Для профессионалов / Марко Кэнту ; пер. с англ. – СПб. : Питер, 2007. – 912 с. : ил.

3. Calvert, Ch. Object Pascal Style Guide / Charles Calvert [Электронный ресурс]. Режим доступа : <http://edn.embarcadero.com/article/10280>.

4. Культин Н.Б. Основы программирования в Delphi 2006 для Windows. Самоучитель – СПб.: БХВ Петербург, 2006, - 384 стр.

**Интернет-ресурсы**

<http://www.snkey.net/books/delphi/ch1-0.html>

<http://www.delphi-manual.ru/index.php>

<http://www.snkey.net/books/delphi/ch3-2.html>

[http://mydelphiprogramms.jimdo.com](http://mydelphiprogramms.jimdo.com/2012/03/22/%D0%BA%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D1%8B-delphi/)