МИНИСТЕРСТВО ПО РАЗВИТИЮ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И КОММУНИКАЦИЙРЕСПУБЛИКИ УЗБЕКИСТАН

ФЕРГАНСКИЙ ФИЛИАЛ ТАШКЕНТСКОГО УНИВЕРСИТЕТА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

КУРСОВАЯ РАБОТА

по дисциплине: “Программирование на языке C++”

Тема:

_____

_____

Выполнил: студент гр. _____ _____

Руководитель: _____

Члены приемной комиссии : _____

_____

_____

Фергана - 2017

Заведующий кафедрой
“Программный инжиниринг”
_____ Тураев Н.М.
“___” _____ 2016

ЗАДАНИЕ

на курсовой проект по дисциплине

“Программирование на языке С++”

Студент _____

Группа_____

Тема курсовой работы

_____

_____

_____

Используемые материалы_____

_____

_____

Части подготовленные на компьютере_____

_____

_____

1)_____
2)_____
3)_____
4)_____

Пояснительная записка_____

_____

Календарный план защиты курсовой

Дата	Вид выполненной работы	Пояснение

Руководитель_____ _____
(Подпись) (дата)

Студент_____ _____

МИНИСТЕРСТВО ПО РАЗВИТИЮ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И КОММУНИКАЦИЙ РЕСПУБЛИКИ УЗБЕКИСТАН ФЕРГАНСКИЙ ФИЛИАЛ ТАШКЕНТСКОГО УНИВЕРСИТЕТА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Студент группы_____
Факультет “_____”
по предмету «Программирование на языке С++»

_____
(ф.и.о.)

Курсовая работа на
тему:_____
_____

Рецензия

1. Цель работы_____
_____
_____
_____
_____

2. Содержание частей работы_____
_____
_____
_____

3. Положительные стороны
работы_____
_____
_____
_____
_____
_____

4. Отрицательные стороны
работы_____
_____
_____
_____

Вывод_____
_____
_____
_____
_____

Рецензент
_____
(подпись, фамилия, имя, отчество)

«___»_____ 2017 г.

Тема: Создание программы - дневника с записями дел на определенную дату

ПЛАН:

ВВЕДЕНИЕ

Модернизация экономики страны во многом зависит от внедрения современных высоких технологий и в нашей стране большое внимание уделяется развитию современных информационно – коммуникационных технологий. Узбекистан идет по пути прогресса и модернизации технологий производства благодаря принятым первым Президентом РУз И.А. Каримовым указам в области ИКТ – “О дальнейшем развитии компьютеризации и внедрении информационно-коммуникационных технологий“ (ведомости ОлийМажлиса Республики Узбекистан, 2002 г., № 4-5, ст. 98; собрание законодательства Республики Узбекистан, 2006 г., № 28-29, ст. 262). В связи со стремлением к освоению новых технологий создана данная курсовая работа.

Целью данной курсовой работы является приобретение знаний и практических навыков самостоятельного программирования задач в среде Windows, а также освоение инструментальных средств отладки и методов программирования. Основная цель данной курсовой работы – обучение. Студент учится формировать постановку задач, составлять алгоритмы их решения и формализовать эти алгоритмы в популярной среде программирования. В ходе написания программы студент осваивает язык C++ и особенности его трансляции в компиляторе Builder.

Основная задача данной курсовой работы – обучение. Студент учится формировать постановку задач, составлять алгоритмы их решения и формализовать эти алгоритмы в популярной среде программирования.Цель данной курсовой работы – является систематизация, углубление и активное применение знаний по системному программированию, закрепление знаний, полученных в лекционном курсе, а также на практических занятиях. Достижение цели обеспечивается решением следующей задачи:

Разработка приложения «Дневник дел» при помощи динамических библиотек C++ Builder.

I. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 История C++

C++ компилируемый язык программирования общего назначения, сочетает свойства как высокоуровневых, так и низкоуровневых языков программирования. В сравнении с его предшественником, языком программирования Си, наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования. Название «язык программирования C++» происходит от языка программирования C, в котором унарный оператор ++ обозначает инкремент переменной.

Язык программирования C++ широко используется для разработки программного обеспечения. А именно, создание разнообразных прикладных программ, разработка операционных систем, драйверов устройств, а также видео игр и многое другое. Существует несколько реализаций языка программирования C++ — как бесплатных, так и коммерческих. Их производят проекты: GNU, Microsoft и Embarcadero (Borland). Проект GNU — проект разработки свободного программного обеспечения (СПО).

Язык программирования C++ был создан в начале 1980-х годов, его создатель сотрудник фирмы BellLaboratories — Бьёрн Страуструп.

Он придумал ряд усовершенствований к языку программирования C, для собственных нужд. Т. е. изначально не планировалось создания языка программирования C++. Ранние версии языка C++, известные под именем «Си с классами», начали появляться с 1980 года. Язык C, будучи базовым языком системы UNIX, на которой работали компьютеры фирмы Bell, является быстрым, многофункциональным и переносимым. Страуструп добавил к нему возможность работы с классами и объектами, тем самым зародил предпосылки нового, основанного на синтаксисе C, языка программирования. Синтаксис C++ был основан на синтаксисе C, так как Бьёрн Страуструп стремился сохранить совместимость с языком C.
В 1983 году произошло переименование языка из «Си с классами»в «язык программирования C++».

В него были добавлены новые возможности: виртуальные функции, перегрузка функций и операторов, ссылки, константы и многое другое. Его первый коммерческий выпуск состоялся в октябре 1985 года.

Бьерн Страуструп высвободил объектно-ориентированный потенциал *C* путем перенесения возможностей классов *Simula 67* в *C*. Первоначально новый язык носил имя "C с классами" и только потом стал называться *C++*. Язык *C++* достиг популярности, будучи разработанным в

BellLabs, позже он был перенесен в другие индустрии и корпорации. Сегодня это один из наиболее популярных языков программирования в мире. *C++* наследует как хорошие, так и плохие стороны *C*.

Бьерн Страуструп: "Я придумал C++, записал его первоначальное определение и выполнил первую реализацию. Я выбрал и сформулировал критерии проектирования *C++*, разработал его основные возможности и отвечал за судьбу предложений по расширению языка в комитете по стандартизации *C++*", - пишет автор самого популярного языка программирования. - "*Язык C++* многим обязан языку *C*, и язык *C* остается подмножеством *языка C++* (но в C++ устранены несколько серьезных брешей системы типов C). Я также сохранил средства *C*, которые являются достаточно низкоуровневыми, чтобы справляться с самыми критическими системными задачами. *Язык C*, в свою очередь многим обязан своему предшественнику, BCPL; кстати, стиль комментариев // был взят в C++ из BCPL. Другим основным источником вдохновения был язык Simula67. Концепция классов (с производными классами и виртуальными функциями) была позаимствована из него. Средства перегрузки операторов и возможность помещения объявлений в любом месте, где может быть записана инструкция, напоминает Algol68. "

Название **C++** выдумал РикМасситти. Название указывает на эволюционную природу перехода к нему от C. "++" - это операция приращения в C. Чуть более короткое имя C+ является синтаксической ошибкой; кроме того, оно уже было использовано как имя совсем другого языка. Знатоки семантики C находят, что *C++* хуже, чем ++C. Названия D язык не получил, поскольку он является расширением C и в нем не делается попыток исцеляться от проблем путем выбрасывания различных особенностей...

Изначально *язык программирования C++* был разработан, чтобы автору и его друзьям не приходилось программировать на ассемблере, C или других современных языках высокого уровня. Основным его предназначением было сделать написание хороших программ более простым и приятным для отдельного программиста. Плана разработки *C++* на бумаге никогда не было; проект, документация и реализация двигались одновременно. Разумеется, внешний интерфейс *C++* был написан на *C++*. Никогда не существовало "Проекта C++" и "Комитета по разработке C++". Поэтому *C++* развивался и продолжает развиваться во всех направлениях, чтобы справляться со сложностями, с которыми сталкиваются пользователи, а также в процессе дискуссий автора с его друзьями и коллегами.

В **языке C++** полностью поддерживаются принципы объектно-ориентированного программирования, включая три кита, на которых оно

стоит: инкапсуляцию, наследование и полиморфизм. *Инкапсуляция в C++* поддерживается посредством создания нестандартных (пользовательских) типов данных, называемых классами. *Язык C++* поддерживает наследование. Это значит, что можно объявить новый тип данных (класс), который является расширением существующего.
Хотя **язык программирования C++** справедливо называют продолжением C и любая работоспособная программа на языке C будет поддерживаться *компилятором C++*, при переходе от C к C++ был сделан весьма существенный скачок. *Язык C++* выигрывал от своего родства с языком C в течение многих лет, поскольку многие программисты обнаружили, что для того, чтобы в полной мере воспользоваться преимуществами *языка C++*, им нужно отказаться от некоторых своих прежних знаний и приобрести новые, а именно: изучить новый способ концептуальности и решения проблем программирования. Перед тем как начинать осваивать *C++*, *Страуструп* и большинство других программистов, использующих *C++* считают изучение языка C необязательным.

Имя языка, получившееся в итоге, происходит от оператора унарного постфиксного инкремента C `++` (увеличение значения переменной на единицу). Имя C+ не было использовано потому, что является синтаксической ошибкой в C и, кроме того, это имя было занято другим языком. Язык также не был назван D, поскольку «*является расширением C и не пытается устранять проблемы путём удаления элементов C*».

Главное нововведение C++ - механизм классов, дающий возможность определять и использовать новые типы данных. Программист описывает внутреннее представление объекта класса и набор функций-методов для доступа к этому представлению. Одной из заветных целей при создании C++ было стремление увеличить процент повторного использования уже написанного кода. Концепция классов предлагала для этого механизм наследования. Наследование позволяет создавать новые (производные) классы с расширенным представлением и модифицированными методами, не затрагивая при этом скомпилированный код исходных (базовых) классов. Вместе с тем наследование обеспечивает один из механизмов реализации полиморфизма - базовой концепции объектно-ориентированного программирования, согласно которой, для выполнения однотипной обработки разных типов данных может использоваться один и тот же код. Собственно, полиморфизм - тоже один из методов обеспечения повторного использования кода.

Введение классов не исчерпывает всех новаций языка C++. В нем реализованы полноценный механизм структурной обработки исключений, отсутствие которого в C значительно затрудняло написание надежных программ, механизм шаблонов - изощренный механизм макрогенерации, глубоко встроенный в язык, открывающий еще один путь к повторной используемости кода, и многое другое.

Таким образом, генеральная линия развития языка была направлена на расширение его возможностей путем введения новых высокоуровневых конструкций при сохранении сколь возможно полной совместимости с ANSI C. Конечно, борьба за повышение уровня языка шла и на втором фронте - те же классы позволяют при грамотном подходе упрятывать низкоуровневые операции, так что программист фактически перестает непосредственно работать с памятью и системно-зависимыми сущностями. Однако язык не содержит механизмов, вынуждающих разработчика правильно структурировать программу, а авторы не выпустили никаких систематических рекомендаций по использованию его довольно изощренных конструкций. Не позаботились они своевременно и о создании стандартной библиотеки классов, реализующей наиболее часто встречающиеся структуры данных.

Все это привело к тому, что многие разработчики вынуждены были сами исследовать лабиринты языковой семантики и самостоятельно отыскивать успешно работающие идиомы. Так, например, на первом этапе развития языка многие создатели библиотек классов стремились построить единую иерархию классов с общим базовым классом Object. Эта идея была заимствована из Smalltalk - одного из наиболее известных объектно-ориентированных языков. Однако она оказалась совершенно нежизнеспособной в C++ - тщательно продуманные иерархии библиотек классов оказывались негибкими, а работа классов - неочевидной. Для того чтобы библиотеками классов можно было пользоваться, их приходилось поставлять в исходных текстах.

C++ Builder (по-русски обычно произносят [*си-плюс-плюс би́лдэр*], [*си би́лдэр*]) — программный продукт, инструмент быстрой разработки приложений (RAD), интегрированная среда программирования (IDE), система, используемая программистами для разработки программного обеспечения на языке программирования C++.

Изначально разрабатывался компанией BorlandSoftware, а затем её подразделением CodeGear, ныне принадлежащим компании EmbarcaderoTechnologies.

C++ Builder объединяет в себе комплекс объектных библиотек (STL, VCL, CLX, MFC и др.), компилятор, отладчик, редактор кодаи многие другие компоненты. Цикл разработки аналогичен Delphi[1]. Большинство компонентов, разработанных в Delphi, можно использовать и в C++ Builder без модификации, но обратное утверждение не верно.

C++ Builder содержит инструменты, которые при помощи drag-and-drop действительно делают разработку визуальной, упрощает программирование благодаря встроенному WYSIWYG — редактору интерфейса и пр.

C++Builder первоначально создавалась только для платформы MicrosoftWindows. Поздние версии, содержащие кроссплатформенную компонентную библиотеку Borland, поддерживают и Windows, и Linux.

В 2003 году Borland выпустила *C++BuilderX* (CBX), написанный при помощи той же инфраструктуры, что и JBuilder, который при этом был мало похож на C++ Builder или Delphi. Этот продукт предназначался для разработки больших программ для крупных предприятий, но коммерческого успеха не достиг. В конце 2004 года Borland объявила, что продолжит развитие классического C++ Builder и объединит его со средой разработки Delphi, прекратив, таким образом, разработку C++ BuilderX. Спустя примерно год после этого объявления, Borland выпустила *BorlandDeveloperStudio 2006*, который включал в себя *Borland C++Builder 2006*, предлагавший улучшенное управление конфигурацией и отладкой. *BorlandDeveloperStudio 2006* — единственный полноценный комплект, содержащий Delphi, C++ Builder и C# Builder. В 2007 году CodeGear выпустила *C++Builder 2007*, в котором реализовала полную поддержку API MicrosoftWindowsVista, увеличила полноту соответствия стандарту ANSI C++, увеличила скорость компиляции и сборки до 500 %, включила поддержку MSBuild, архитектур баз данных DBX4 и «VCL для Web», поддерживающий AJAX. Поддержка API MicrosoftWindowsVista включила в себя приложения, изначально оформленные в стиле Vista, и естественную поддержку VCL для Aero и VistaDesktop. *CodeGear RAD Studio 2007* содержит*C++Builder 2007* и *Delphi*. Также в 2007 году CodeGear «воскресила» марку «Turbo» и выпустила две «Turbo» версии C++Builder:

Turbo C++ Professional и Turbo C++ Explorer (бесплатный), основанных на *Borland C++ Builder 2006.*

В конце 2008 года компания CodeGear выпустила новую версию RAD Studio, в которую вошли Delphi 2009 и C++Builder 2009. В 2009 году в составе RAD Studio вышел C++Builder 2010.

Ранее сообщалось, что следующая версия, CodeGear C++ Builder (кодовое имя «Commodore»), будет обладать поддержкой x86-64 и возможностью создавать машинный x86-64 код. Однако в 2010 году в состав RAD Studio XE включена версия C++ Builder XE без этой функциональности.

В 2012 году Embarcadero выпустила C++ Builder XE3, совместимый с Windows 8. В 2013 году был выпущен C++ Builder XE4.

1.2 C++ Builder и его компоненты

C++ Builder представляет собой SDI-приложение, главное окно которого содержит настраиваемую инструментальную панель (слева) и палитру компонентов (справа). Помимо этого, по умолчанию при запуске C++ Builder появляются окно инспектора объектов (слева) и форма нового приложения (справа). Под окном формы приложения находится окно редактора кода. Формы являются основой приложений C++ Builder. Создание пользовательского интерфейса приложения заключается в добавлении в окно формы элементов объектов C++ Builder, называемых компонентами. Компоненты C++ Builder располагаются на палитре компонентов, выполненной в виде многостраничного блокнота. Важная особенность C++ Builder состоит в том, что он позволяет создавать собственные компоненты и настраивать палитру компонентов, а также создавать различные версии палитры компонентов для разных проектов.

Компоненты разделяются на видимые (визуальные) и невидимые (невизуальные). Визуальные компоненты появляются во время выполнения точно так же, как и во время проектирования. Примерами являются кнопки и редактируемые поля. Невизуальные компоненты появляются во время проектирования как пиктограммы на форме. Они никогда не видны во время выполнения, но обладают определенной функциональностью (например, обеспечивают доступ к данным, вызывают стандартные диалоги Windows 95 и др.)

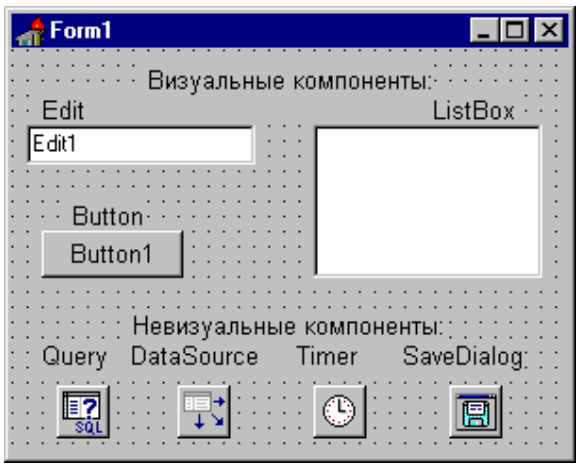
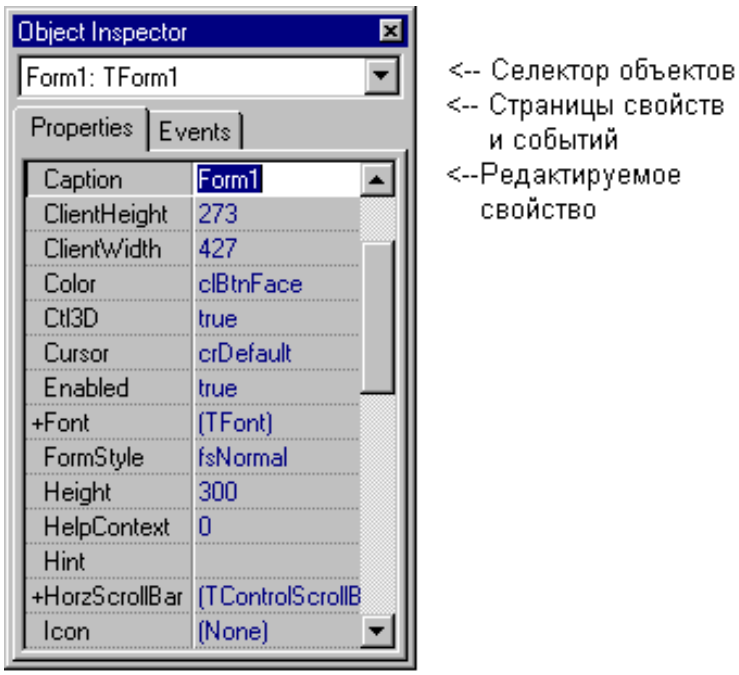


Рис. 2. Пример использования видимых и невидимых компонентов

Для добавления компонента в форму можно выбрать мышью нужный компонент в палитре и щелкнуть левой клавишей мыши в нужном месте проектируемой формы. Компонент появится на форме, и далее его можно перемещать, менять размеры и другие характеристики.

Каждый компонент C++ Builder имеет три разновидности характеристик: свойства, события и методы. Если выбрать компонент из палитры и добавить его к форме, инспектор объектов автоматически покажет свойства и события, которые могут быть использованы с этим компонентом. В верхней части инспектора объектов имеется выпадающий список, позволяющий выбирать нужный объект из имеющихся на форме. Для эффективной разработки пользовательских интерфейсов приложений C++ Builder нередко возникает необходимость в манипулировании компонентами на формах. Большинство операций для манипулирования компонентами находятся в меню Edit: К различным опциям этого меню следует обращаться после того, как на форме вы ран один или несколько компонентов, свойства которых требуется изменить.



Компоненты Standard

На этой вкладке располагаются компоненты (рис. 1), с помощью которых происходит подключение к программе стандартных интерфейсных элементов, имеющихся во всех версиях операционных систем Windows. Рассмотрим эти компоненты по порядку слева направо. □**Frame** (Кадр) — предназначен для создания контейнера (окна) для размещения других компонентов. Данный компонент очень похож на форму **Form**. Для размещения этого компонента на форме необходимо первый раз создать его с помощью команд **File | New | Frame**. Именно такое сообщение появляется при попытке размещения этого компонента стандартным образом (рис. 18.2).

Рис. 1. Палитра компонентов Standard

□ **MainMenu** (Главное меню) — предназначен для создания главного меню программы. С этим и другими компонентами мы познакомимся поближе в процессе создания новых программ. □**PopupMenu** (Всплывающее меню) — предназначен для создания всплывающего меню некоторых компонентов. Обычно с помощью этого ком-понента создается контекстное меню. □**Label** (Этикетка) — создает на форме текстовую метку или надпись.

□ **Edit** (Редактирование) — создает на форме поле для редактирования текстовой строки.
□ **Memo** (Поле) — отображает на форме поле для редактирования текстовых строк. Обычно служит для создания редакторов и полей для вывода блоков данных.
□**Button** (Кнопка) — является самым распространенным компонентом. Служит для создания в приложении различных прямоугольных кнопок с текстовой надписью.
□ **CheckBox** (Ячейка состояния) — позволяет создавать на форме приложения ячейку с двумя состояниями (без галочки и с галочкой) и строкой названия. Щелчок левой кнопкой мыши по этому компоненту во время работы программы вызывает изменение состояния компонента на противоположное. В программе всегда можно узнать состояние этого компонента и тем самым выполнять то или иное действие.
□**RadioButton** (Радиокнопка) — создает круглое поле с двумя состояниями (с точкой и без точки) и текстовой строкой, поясняющей ее назначение в программе. Обычно несколько таких компонентов, расположенных на форме, позволяют переключить только один элемент из группы. Для наглядности сказанного и закрепления материала на практике создайте новое приложение и расположите на форме несколько компонентов RadioButton. После чего запустите приложение на выполнение и пощелкайте левой кнопкой мыши поочередно по каждому из них. Вы увидите, что можно изменить состояние только для одного из этих компонентов, так как остальные компоненты переключают при этом свое состояние автоматически.
□**ListBox** (Окно списка) — создает прямоугольное поле для отображения текстовых строк с возможностью их выбора, добавления или удаления при работе программы.
□ **ComboBox** (Комбинированный список) — позволяет создавать на форме элемент, являющийся комбинацией строки ввода и выпадающего списка для выбора. Фактически объединяет в себе компоненты ListBox и Edit.
□**ScrollBar** (Линейка прокрутки) — создает элемент, похожий на линейку с бегунком и кнопками для прокрутки окна, к которому относится этот элемент. Кроме того, с его помощью можно изменять в пределах некоторого заданного интервала значение величины какого-либо параметра.
□**GroupBox** (Окно группы) — служит для создания области, визуально объединяющей на форме несколько интерфейсных элементов.
□ **RadioGroup** (Группа радиокнопок) — позволяет создавать на форме контейнер в виде прямоугольной рамки для объединения группы взаимоисключающих радиокнопок.

□ **Panel** (Панель) — создает пустую область, на которой можно разместить другие компоненты. Как правило, используется для создания панели инструментов в программе.
□**ActionList** (Список действий) — осуществляет управление взаимодействием между интерфейсными элементами и логикой программы.

На рис. 4 приведено окно формы, на которой расположены все перечисленные компоненты в порядке их описания слева направо и сверху вниз, начиная с **MainMenu**.

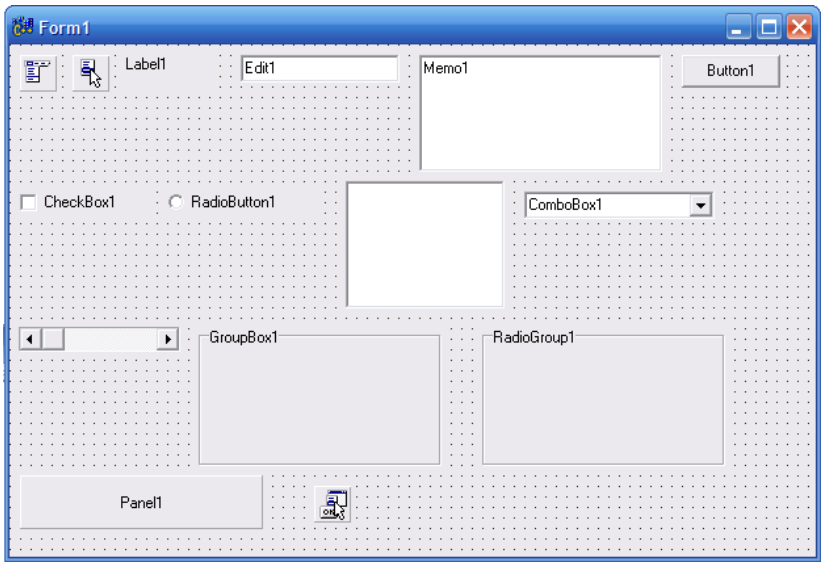
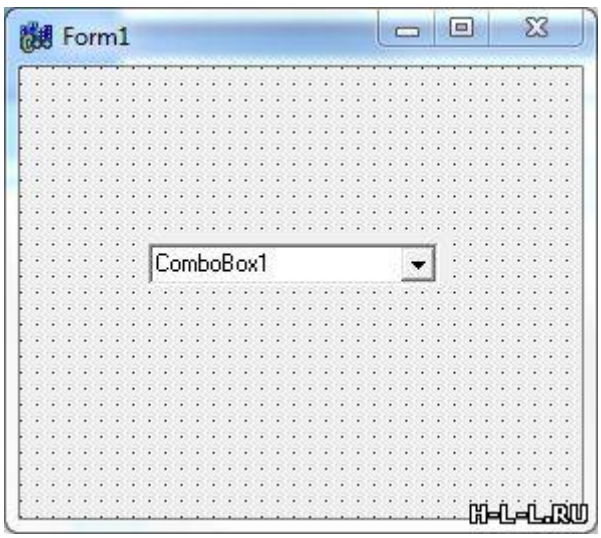
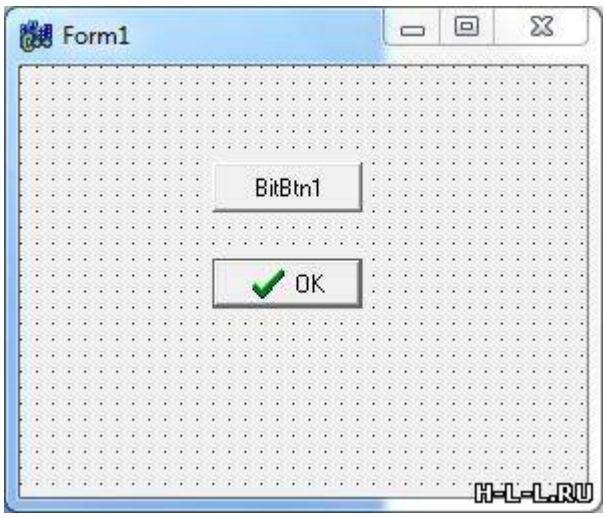


Рис. 4. Окно формы с компонентами

ComboBox - выпадающий список элементов.



BitBtn - кнопка с рисунком.



Изменение размера компонентов

Изменение размера компонента можно проводить как при добавлении его на форму, так и после этого.При добавлении компонента следует выбрать его на палитре компонентов. Далее ужно поместить курсор мыши на форму, нажать левую клавишу и перемещать мышь, в результате чего на форме появится прямоугольник, изображающий границы буущего компонента. Когда прямоугольник приобретет необходимые размеры, нужно отпустить кнопку мыши (рис.3).

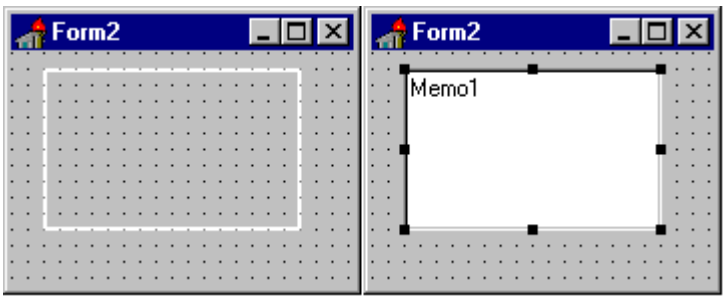


Рис. 3. Изменение размера компонента при его добавлении на форму.

Если перевести курсор мыши на один из появившихся вокруг компонента малеьких черных квадратиков, курсор мыши изменяет форму. Перемещая

этот курсор и вместе с ним границу компонента, можно изменять его размеры.

Для изменения размеров нескольких компонентов следует выбрать их одним из описанных выше способов. Далее нужно выбрать пункт меню Edit/Size. Появится диалоговое окно Size. Выберите опции размера. Для точной установки размера в пикселах можно ввести числа в поля Width и Height. Далее нужно нажать кнопку OK.

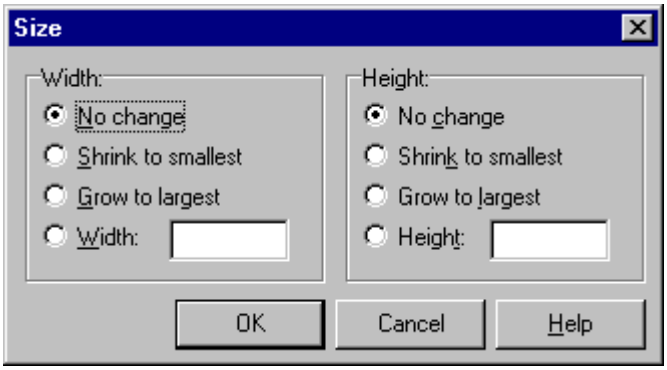
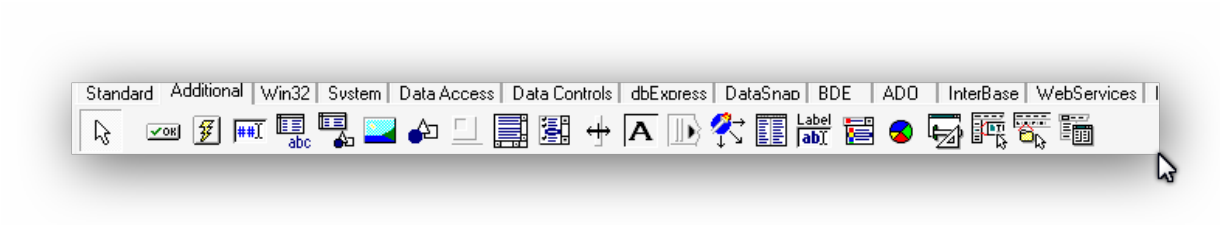


Рис. 4. Установка свойств компонентов с использованием меню EDIT/SIZE

Можно добавить несколько копий компонента одного типа, выбирая компонент из палитры при нажатой клавише Shift. В этом случае вокруг компонента появляется п ямоугольник, окружающий этот компонент. После этого каждый щелчок мышью на фоме приводит к появлению на ней копии компонента. Закончив режим многократного копирования, следует щелкнуть мышью на инструменте выбора курсора (первая кнопка на палитре компонентов с изображением стрелки).

Библиотека Визуальных Компонент позволяет программистам визуально создавать программные приложения, не прибегая более к кодированию классов "вручную", или кодированию в рамках стандартных библиотек MFC (MicrosoftFoundationClass), или OWL (ObjectWindowsLibrary).

C++ программистам теперь не надо создавать или манипулировать объектами интерфейса с пользователем путем написания соответствующего кода. Подавляющее большинство приложений вы будете разрабатывать визуально с помощью Редактора форм C++Builder, добавляя лишь несколько строк к .обработчикам ключевых событии компонент. Используйте объекты всегда, когда это возможно; твердо сопротивляйтесь позыву написать новый код то тех пор, пока все другие возможности не будут исчерпаны.



Вам потребуется оперативное владение устройством Библиотеки Визуальных Компонент. Глубина необходимых программистам знаний о составе и функциональных характеристиках Библиотеки определяется тем, как вы собираетесь ее использовать. С помощью команды главного меню **Help | VCL Reference** вы можете получать сведения из справочной службы в процессе работы с Библиотекой.Программист создает законченное приложение посредством интерактивного взаимодействия с интегрированной визуальной средой C++Builder, используя компоненты VCL для создания интерфейса программы с пользователем и с другими управляющими элементами: обслуживания баз данных, контролируемого ввода параметров и т.д.

II АНАЛИТИЧЕСКАЯ ЧАСТЬ

2.1 Информации о компоненте MonthCalendar

Компонент MonthCalendar

Компонент MonthCalendar похож на компонент DateTimePicker, работающий в режиме ввода дат. Правда, в компоненте MonthCalendar предусмотрены некоторые дополнительные возможности:
- можно допустить множественный выбор дат в некотором диапазоне (свойство MultiSelect);
- можно указывать в календаре номера недель с начала года (свойство WeekNumbers);
- перестраивать календарь, задавая первый день каждой недели (свойство FirstDayOfWeek) и т.п.
Для некоторых офисных приложений все это достаточно удобно.

Компонент CCalendar

Компонент CCalendar представляет собой менее красочный и более обыденно оформленный календарь на один месяц. Вместо свойства Date в нем предусмотрены отдельные свойства Year - год, Month - месяц, Day - день. Все это целые числа, с которыми иногда удобнее иметь дело, чем с типом TDateTime. Перед отображением на экране или в процессе проектирования надо задать значения Month и Year, чтобы компонент отобразил календарь на указанный месяц указанного года. Впрочем, если вам надо иметь календарь на текущий месяц, надо нужно установить в true значение свойства UseCurrentDate (установлено по умолчанию). В этом случае по умолчанию будет показан календарь на текущий месяц с выделенным в нем текущим днем. Свойство StartOfWeek задает день, с которого начинается неделя. По умолчанию задано 0 - воскресенье, как это принято в западных календарях. Но для нас все-таки как-то привычнее начинать неделю с рабочего дня понедельника. Так что желательно задать StartOfWeek = 1.

switch

Это еще один вид оператора условия (выбора). Его лучше использовать если, допустим, нам надо вывести дни недели, т.е. когда у нас не одно, а

множество условий выбора. Поэтому буду называть эту структуру - оператор выбора while. Вот его синтаксис:

```
switch( определенное значение) {
  case значение1: //тело
    break;
  ...
  ...
  casезначениеN: //тело
    break;
  default: // тело }
```

определенное значение

Значение, которое является можно сказать меткой, т.е. именно его будет находить наша программа в теле оператора выбора switch

case

Таких операторов выбора case может быть очень много, так как это просто различные варианты значений. Но работать будет именно тот оператор case, который будет равен нашему "определенному значению"

default

Оператор выбора default будет выполняться, если ни один из операторов выбора case не содержит нужного числа. Т.е. если ничего не совпало с "определенным значением", то будет выполняться оператор по умолчанию - default. Да, и двоеточие в конце операторов case значение: и default: являются обязательными

break

Оператор break является своего рода остановкой выполнения условия, т.е. с помощью него мы сразу же завершаем выполнение нашего оператора выбора и дальше продолжаем выполнение программы. Но вы можете не доумевать; зачем пользоваться оператор break, когда оператор выбора и без него сам завершит свою работу. Это ж даже не цикл, здесь необязательно, как бы принудительно, завершать процесс, но есть одно но. И это Но давайте рассмотрим на примере:

```
intznachenie = 1;
switch( znachenie ) {
  case 1:
    printf("Значение 1");
  case 2:
    printf("Значение 2");
  default:
    printf("Другое значение!");}
```

По логике, да и по выше описанному материалу, на экране должно появиться сообщение: Значение 1. Но так как мы взяли и опустили оператор break, то на экране появится: Значение 1 Значение 2 Другое значение.

Т.е. из-за того, что нет оператора прерывания, оператор (функция) выбора switch, найдя нужное значение в операторах case, далее будет выполняться все подряд. Поэтому не забывайте вставлять в противовес каждому оператору case оператор break. Так же надо отметить, что в качестве определенно значения могут использоваться и символы, это не запрещено, только надо их брать в одинарные кавычки. Например: 'd'

Синтаксис

```
switch ( <переменная> ) {
case значение1:
  Выполнить если <переменная> == значение1
break;
case значение2:
  Выполнить если <переменная> == значение2
break;
...
default:
выполнить, если ни один вариант не подошел
break;
}
```

Переменная в скобках сравнивается со значениями, описанными после ключевого слова case. После двоеточия находится код, который будет выполнен в случае если переменная оказалась равной текущему значению. break необходим для того, чтобы прервать выполнение switch.

Рассмотримпример, гденетbreak:

```
int a=1;
switch(a)
{
case 1:
a++;
case 2:
a++;
case 3:
a++;
}
cout<<"a= "<<a;
```

Данная программа выведет a = 4.

Значения для сравнения, описанные после case могут быть только константами, поэтому следующий вариант использования switch-case — неверен:

```
int a = 10;
int b = 10;
int c = 20;


switch ( a ) {
case b:
  // Code
break;
case c:
  // Code
break;
default:
// Code
break;   }
```

При попытке скомпилировать данную программу, вы получите подобное сообщение:

Сравнение switch-case с if-else

Если у вас возникают проблемы с пониманием того, как работает switch-case, посмотрите на следующую if-else конструкцию, она работает точно так же, как и switch

```
if ( 1 == input )
```

```
{
playgame();
}
else if ( 2 == input )
{
loadgame();
}
else if ( 3 == input )
{
playmultiplayer();
}
else if ( 4 == input )
{
cout<< "Thank you for playing!\n";
}
else
{
cout<< "Error, bad input, quitting\n";
}
```

Если мы можем сделать то же самое с помощью if-else, зачем вообще нужен switch? Главное преимущество этой конструкции в том, что нам понятно, как работает программа: единственная переменная контролирует поведение программы. В случае с if-else, придется внимательно читать каждое условие.

2.2Описание алгоритма работы программы дневник дел на определенную дату.

1) Открытие главного окна программы

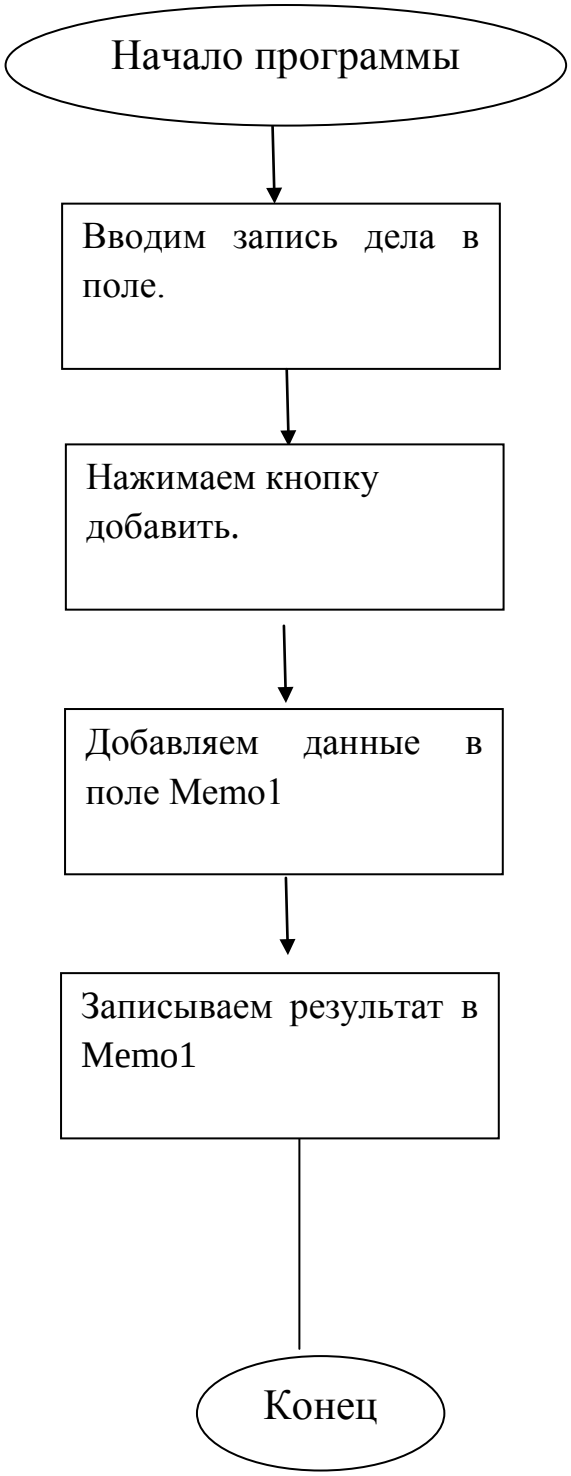
2) Вводим запись дела в поле.

3) Нажимаем кнопку Добавить.

4) Добавляем данные в поле Memo1

5) Записываем результат в Memo1

2.3 БЛОК-СХЕМА

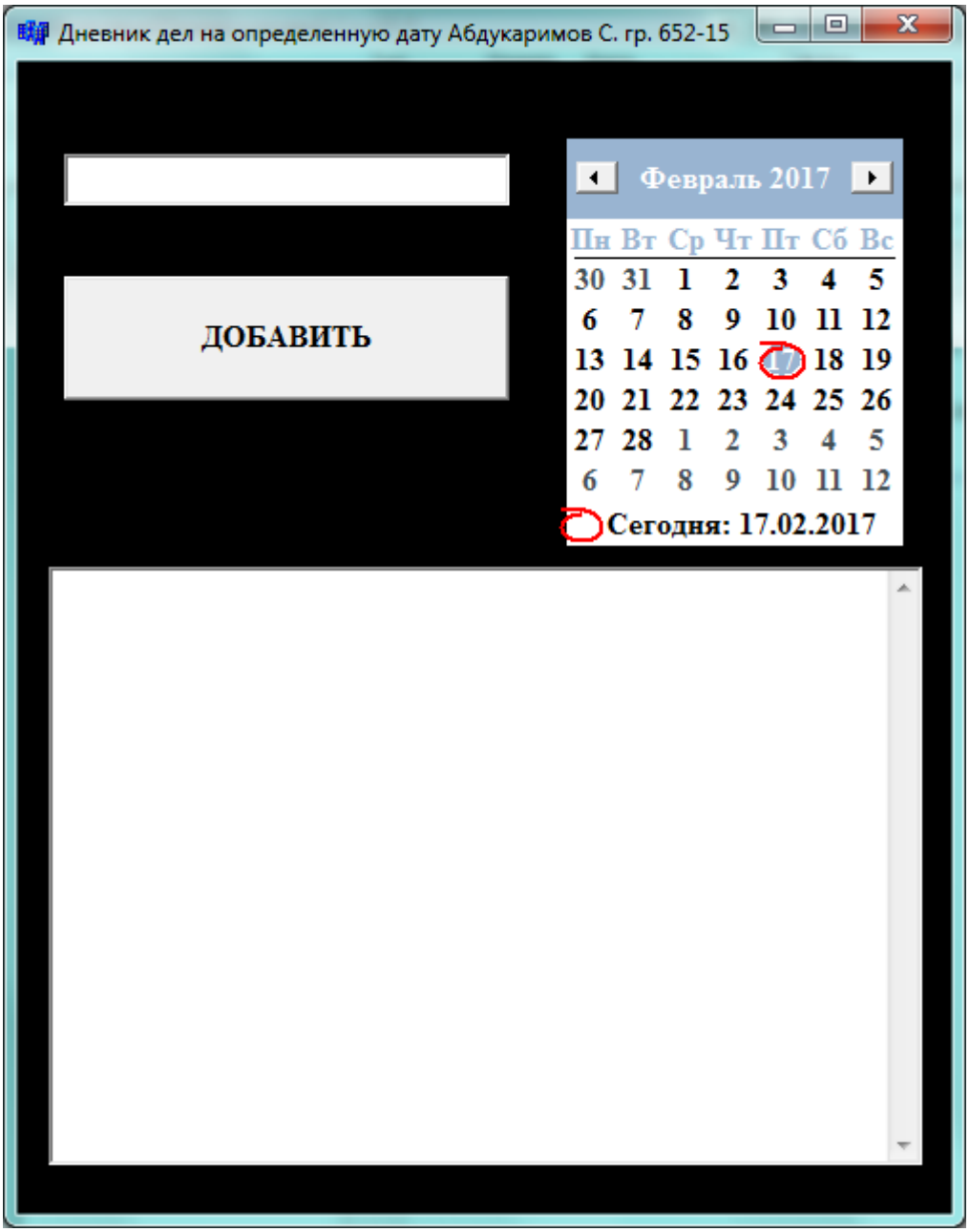


III ПРАКТИЧЕСКАЯ ЧАСТЬ

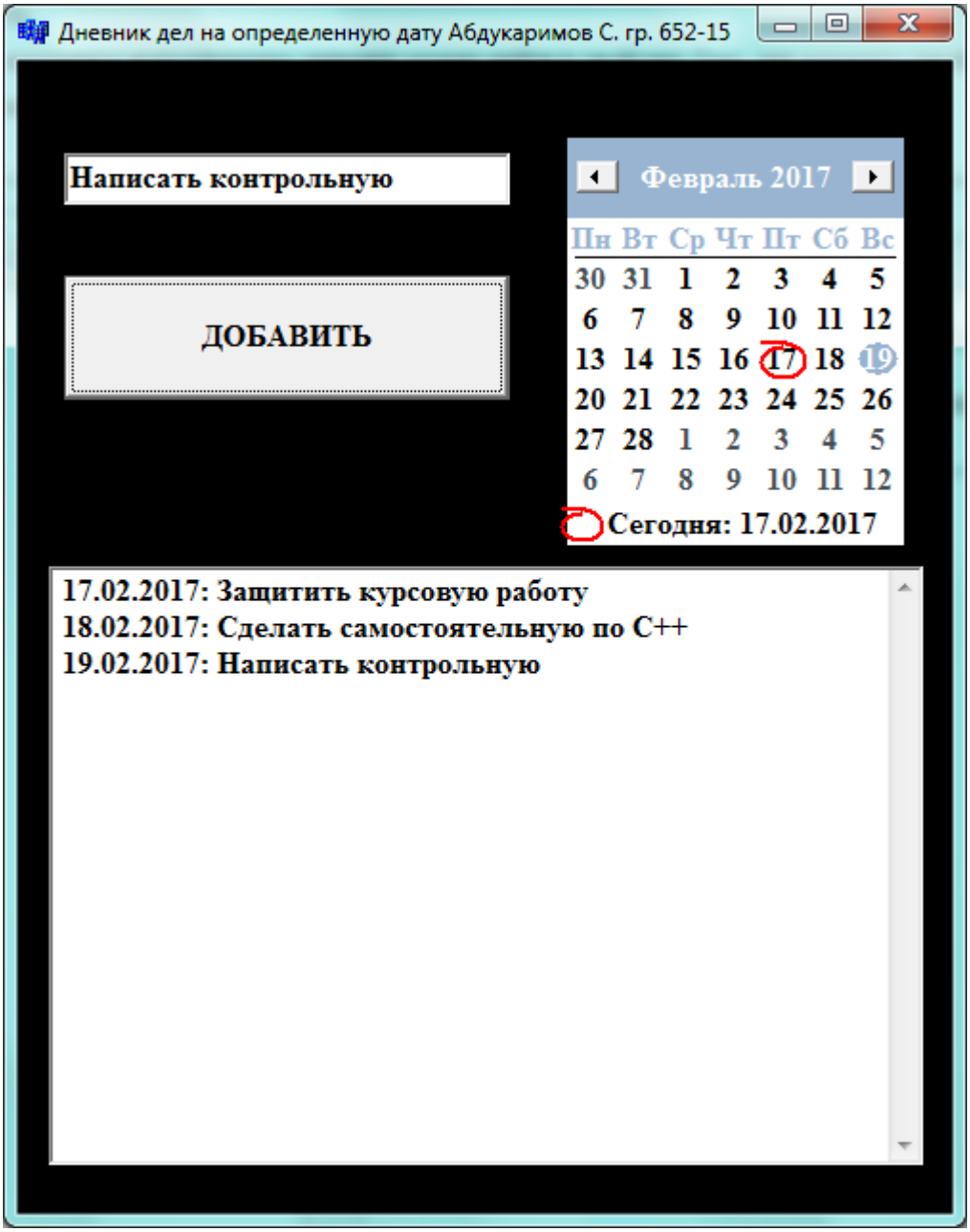
3.1 Описание выполненного задания

В данной курсовой работе ясоставил программу, позволяющую сделать дневник с записями дел на определенную дату. Я использовал компоненты MonthCalendar, Memo.

Вот главное окно программы:



При заполнении и выборе даты, необходимо записать информацию в поле Memo. Вот результат:



3.2 Кодпрограммы

```
//---------------------------------------------------------------------------

#ifndef delaH
#define delaH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
```

```
#include <Forms.hpp>
#include <ComCtrls.hpp>
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published:        // IDE-managed Components
        TEdit *Edit1;
        TButton *Button1;
        TMonthCalendar *MonthCalendar1;
        TMemo *Memo1;
        void __fastcall Button1Click(TObject *Sender);
private:        // User declarations
public:                 // User declarations
        __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif

//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop
//---------------------------------------------------------------------------
USEFORM("dela.cpp", Form1);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
        try
```

```
        {
                Application->Initialize();
                Application->CreateForm(__classid(TForm1), &Form1);
                Application->Run();
        }
        catch (Exception &exception)
        {
                Application->ShowException(&exception);
        }
        catch (...)
        {
                try
                {
                        throw Exception("");
                }
                catch (Exception &exception)
{
                        Application->ShowException(&exception);
                }
        }
        return 0;
}
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "dela.h"
//---------------------------------------------------------------------------
```

```
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
Memo1->Lines->Add(DateToStr(MonthCalendar1->Date)+": "+Edit1->Text);
}
//---------------------------------------------------------------------------
```

ЗАКЛЮЧЕНИЕ

Основная я изучил работу с компонентами C++ Builder и составил программу дневник с записями дел на дату. В этой работе я научился формировать постановку задач, составлять алгоритмы их решения и формализовать эти алгоритмы в популярной среде программирования. В ходе написания программы студент осваивает язык C++ и особенности его трансляции в компиляторе Builder.Также одной из основных задач является обучение правильному оформлению документации программы, что немаловажно для специалиста.

Я узнал и использовал основные компоненты в C++ Builder. Я основном работал с некоторые компоненты например: Memo, Edit, Label, Button, Month Calendar т.д.

Также одной из основных задач является обучение правильному оформлению документации программы, что немаловажно для специалиста.

При выполнении курсового проекта были пройдены все этапы разработки специализированного прикладного программного обеспечения:

- постановка задачи;

- формализация задачи, определение входной и выходной информации;

- сбор необходимых исходных данных, используемых в программе;

- определение необходимых требований к техническим и программным средствам для функционирования приложения;

- составление логической структуры решения задачи и программы;

- подготовка инсталяционного пакета;

- составление инструкции пользователя.

СПИСОК ЛИТЕРАТУРЫ

1. Подбельский В.В. Программирование на языке Си / В.В. Подбельский, С.С.Фомин. - М.: Финансы и статистика, 2008. – 326с.
2. И.А. Каримов *“По пути модернизации страны и устойчивого развития экономики”*.
3. Иванова Г.С. Объектно-ориентированное программирование. Учебник. МГТУ им Баумана. 320 стр, 2003 г.
4. Крупник А.Б. Изучаем C++. Питер. 251 стр, 2003 г.
5. Мейерс С. Наиболее эффективное использование C++. 35 новых рекомендаций. ДМК-Пресс, 304 стр, 2000 г.
6. Архангельский А.Я. Программирование в C++Builder 6. – М.:ЗАО «Издательство БИНОМ», 2004.
7. Страуструп Б. Язык программирования C++. Ч 1: Пер. с англ. – Киев: ДиаСофт, 1993.