**1)**Перечислите состав программного проекта в среде C++ Builder.

Можно выделить следующие основные элементы системы:

1.  Палитра инструментов и главное меню системы, содержащие основные команды по управлению проектом и настройками системы.

2.  Палитра компонент – содержит список компонентов, доступных программисту для добавления в проект.

3.  Инспектор объектов – инструмент, предназначенный для управления свойствами и обработчиками событий компонент проекта на этапе разработки приложения.

4.  Редактор форм – инструмент визуального представления видимых и невидимых компонент и манипулирования ими на этапе проектирования.

5.  Редактор кода – отображает код проекта, позволяет вносить в него изменения, для чего имеются удобные встроенные средства.

6.    
Окно дерева объектов – удобный инструмент отображения компонент проекта в иерархии “владелец-подчиненный».

Помимо этого, достойны упоминания элементы среды, которые не отображаются на экране при начальной загрузке, но могут быть открыты дополнительно. К таким элементам можно отнести хранилище объектов и менеджер проекта.

5.Охарактеризуйте состав области видимости переменных и функций в проекте

Каждая переменная кроме имени, типа данных и значения имеет ещё одну характеристику – область видимости переменной. Область видимости переменной – это участок кода программы, в котором данная переменная доступна для использования.

Рассмотрим пример, написанный ранее.

В этом коде переменные **a**, **b** и **max** доступны для использования только внутри функции **max\_num**. А переменные **x,y** и **m** – внутри функции **main**. Область видимости этих переменных – тело функции, в которой они объявлены. Это означает, к примеру, что мы не можем избавиться от переменной **max** внутри функции **max\_num** и присваивать значение сразу переменной **m**. Т.е. вот такой код работать не будет:

Если вы попытаетесь его скомпилировать, то компилятор будет ругаться на несуществующую переменную m внутри функции **max\_num**.

Решить эту проблему довольно просто. Достаточно объявить переменную **m** вне всех функций, сразу после директив препроцессора.

Переменные, объявленные таким образом, называются глобальными, т.к. доступ к ним возможен из любого места в этом файле (доступны глобально). Те переменные, которые мы использовали ранее, называются локальными, т.к. они доступны только в некоторых местах программы (доступны локально).

К локальным относятся также переменные, объявленные, например, внутри тела цикла. Возможно, вы замечали, что если в программе два цикла **for**, то в каждом из них счётчик может иметь имя **i** и объявляться в заголовке цикла.

Дело как раз в области видимости и времени жизни локальных переменных. Каждый раз при создании цикла создаётся новая переменная с именем **i**. На протяжении всего тела цикла данная переменная доступна для использования, т.е. её область видимости – это блок тело цикла. После того как цикл завершается, эта переменная удаляется из памяти. Её время жизни – время работы цикла. Для переменных, объявленных внутри функции, время жизни переменных – время выполнения данной функции.

В этом случае создаётся одна переменная i, которая в дальнейшем используется во всех циклах. Попробуйте добавить вывод на экран значения переменной **i** между циклами. В первом случае должна появиться ошибка, т.к. переменная **i** существует только внутри циклов, а во втором случае всё будет нормально и значение **i** будет равно **10**.

7. Охарактеризуйте состав и назначение панели инструментов управления проектом.

Инструменты и средства управления на панели инструментов служат для того, чтобы добавлять новые элементы пользовательского интерфейса. Чтобы открыть панель инструментов, щелкните на кнопке Панель инструментов. Панель инструментов обычно располагается вдоль левой стороны экрана. После того, как средства управления внесены в форму, они становятся объектами, или программируемыми элементами пользовательского интерфейса. Средства управления, находящиеся на панели инструментов, можно использовать для добавления в форму рисунков, этикеток, кнопок, списков, полос прокрутки, меню и геометрических фигур. После того, как пользователь запустит программу, эти элементы появятся на экране и будут работать так же, как любой объект в стандартном Windows-приложении.

Панель инструментов содержит также средства управления для создания объектов, выполняющих специальные «заэкранные» операции, такие как, управление информацией в базах данных, контроль временных интервалов и т.д. *Если вы поместите указатель мыши на тот или иной инструмент, то через некоторое время рядом с этим инструментом появится его название-подсказка.*

**Окно Properties (Свойства)**Окно Properties (Свойства) позволяет изменять характеристики (установ­ки) элементов пользовательского интерфейса в форме. Например, вы можете изменить шрифт, кегль или выравнивание в сооб­щении, которое выводится вашей программой. (С помощью Visual Basic вы можете выводить текст любым шрифтом, инсталлированным в вашей системе - так же, как в программах Excel или Word.) Изменять установки свойств в окне Properties можно как при создании пользовательского интерфейса, так и тогда, когда программа уже запущена - через программный код. Окно свойств содержит список всех объектов, использующихся в данном пользовательском интерфейсе (форме). В окне свойств также могут перечисляться изменяемые установки свойств, для каждого объекта. Свойства можно просматривать в алфавитном порядке или по категориям.

В следующем упражнении вы измените свойство Caption (Надпись) у командной кнопки**.**

**Окно Project (проект).**Visual Basic-программа состоит из нескольких файлов, которые собираются вместе (компилируются), когда программа готова. Чтобы помочь вам переключаться между отдельными компонентами программы при работе над проектом, разработчики Visual Basic включили в среду программирования окно проекта (Project). В окне Project перечисляются все файлы, используемые при программировании, и осуществляется доступ к ним при помощи двух кнопок: View Code (Просмотр кода) и View Object (Просмотр объекта). Команды в меню File (Файл) в окне Project (Проект) позволяют вам добавлять, удалять или сохранять ваши файлы. Изменения отображаются в окне Project сразу после их внесения в проект.

***Файл проекта***содержит список всех поддерживаемых файлов в про­граммном проекте и имеет расширение .vbp (Visual Basic Project). B Visual Basic 6, в окно проекта, можно одновременно загрузить несколько файлов проектов. Для переключения между ними достаточно щелкнуть мышью на имени проекта. Под именем проекта в окне Project показаны компоненты каждого проекта и связь между ними. «Дерево» проектов похоже на структуру папок в окне Проводника в Windows. Вы можете разворачивать и

сворачивать эти «ветви», щелкая на значках (+) и (-) рядом с папками.

**Просмотр окна проекта**1. Щелкните на кнопке Project Explorer (Проводник проекта) на панели

инструментов. Окно Project теперь выделено. Если это окно было закрыто, сейчас оно появилось.

2. Дважды щелкните на строке заголовка окна Project (Проект), чтобы отобразить его на экране как плавающее (не прикрепленное). Ваш экран будет выглядеть так, как показано на рис.1.4.

3. Щелкните на значках (+) около папок Forms (Формы) и Modules (Модули), если эти папки еще не развернуты. Файл проекта в этом программном проекте называется StepUp. В проекте StepUp перечислены файлы StepUp.frm и StepUp.bas. StepUp.bas содержит код, разделяемый всеми частями программы. StepUp.frm содержит форму пользовательского интерфейса и программный код, связанный с объектами в форме. Когда программа компилируется в выполняемый файл, эти файлы будут скомбинированы в единственный .ехе-файл.

4. Дважды щелкните на заголовке окна Project, чтобы прикрепить его.

 Панель элементов управления — основной рабочий инструмент при визуальной разработке форм приложения. Панель элементов управления вызывается из меню**View** (Вид) командой **Toolbox** (Панель элементов управления). Для вызова этой панели можно воспользоваться также кнопкой**Toolbox** на стандартной панели инструментов. В составе панели элементов управления содержатся основные элементы управления форм метки, текстовые поля, кнопки, списки и другие элементы для быстрого визуального проектирования макета формы.

**Действие** - это программная процедура, определенная разработчиком, либо действие Visual Basic по умолчанию.

Посередине находится текущая форма - экран вашей будущей программы. Он может быть любого размера по вашему усмотрению. Растягивается форма мышкой за края, или нужный размер указывается в соответствующих полях свойств формы.

Все объекты, задействованные в программе, имеют свои свойства. Свойства объектов могут быть самыми разнообразными - размер окна, цвет его фона, шрифт (его тип, размер, цвет). В свойствах также указывается, доступен объект только для чтения или нет, какая у него рамка и т. п.

Компонентов, вообще говоря, может быть бесчисленное множество. Многие поставляются с Visual Basic (количество и виды зависят от типов пакетов, о которых я рассказал выше), еще больше изготовляется другими компаниями (вы можете найти их в Internet), и, кроме того, вы можете создавать свои собственные компоненты.

9. Окно редактирования кода, его состав, назначение, работа с ним.

Обычно в окне редактора используются три панели (три отдельных окна). На самом деле окон может быть и больше и меньше, и скоро вы узнаете обо всех, но на рис. 13.12 показано только расположение основных окон.

* **Project**(Панель проекта), располагается в верхнем левом углу редактора. В ней отображается иерархическое дерево модулей приложения. Если это окно неактивно, выполните команду **View, Project Explorer**либо нажмите комбинацию клавиш <Ctrl>+<R>.
* **Properties**(Панель свойств), находится под панелью проекта. Она позволяет просматривать и изменять свойства различных входящих в проект объектов, отображаемых на панели проекта. Список свойств может отображаться как в алфавитном порядке, так и по категориям.
* **Code**(Панель редактора кода). Это окно занимает большую часть экрана и является "многодокументным", т. е. можно открыть одновременно несколько окон данного типа для разных модулей. Оно представляет собой высокоинтеллектуальный текстовый процессор, существенно облегчающий написание кода VBA.

Проект приложения состоит из модулей, которые делятся на три категории:

* **Microsoft Access Class Objects**(Модули классов Access) — включает все модули форм и отчетов;
* **Modules**(Модули) — стандартные модули;
* **Class Modules**(Модули классов) — модули пользовательских классов, если они присутствуют в приложении.

Список объектов в окне проекта является иерархическим, кроме модулей самого проекта, он может включать ссылки на объекты из внешних библиотек и список модулей этих библиотек.

В верхней части панели проекта расположены три кнопки:

* **View Code**(Просмотр кода). При нажатии этой кнопки указатель мыши перемещается в окно кода, в котором отображается код выделенного в иерархии проекта объекта.
* **View Object**(Просмотр объекта). Эта кнопка доступна только когда в дереве проекта выделен модуль формы или отчета. Тогда она позволяет быстро переключиться на соответствующий модулю объект, т. е. в окно Конструктора формы или отчета.
* **Toogle Folders**(Переключение видов представления дерева). При нажатой кнопке все элементы в дереве проекта группируются в папках по категориям, при отжатой — они отображаются в алфавитном порядке..

Панель свойств позволяет просматривать и изменять свойства различных объектов, входящих в проект. Для отображения свойств объекта его нужно выделить либо в дереве проекта, либо в окне Конструктора формы (отчета). Во многом эта панель дублирует диалоговое окно **Свойства**(Properties) в формах и отчетах. И в том и в другом случае изменение свойств объекта статично, поэтому они могут быть использованы для задания начальных или постоянных свойств объекта.

Чтобы изменить значение свойства объекта, необходимо:

1. Выделить имя свойства в левой колонке.
2. Изменить значение в правой колонке, либо введя его вручную, либо путем выбора из списка.

Для каждого модуля, выбранного на панели проекта, открывается свое окно редактора кода. В данном случае в окне редактора мы видим модуль "Заставка" (Startup) приложения "Борей" (рис. 13.13).

Код в каждом модуле состоит из разделов, переключение между которыми выполняется путем выбора значения из списка **Object**(Объект), который размещается в левом верхнем углу окна. Список разделов различается для разных типов модулей.

Для модуля формы этот список включает раздел **General**(Общая область), который содержит объявления переменных, констант, специальных типов, внешних процедур. Данный раздел открывается по умолчанию при открытии окна кода. Для стандартного модуля этот раздел единственный — он содержит описание всех процедур. Модуль класса имеет разделы **General и Class.**Модуль формы, кроме раздела **General,**содержит раздел самой формы и разделы для каждого элемента управления. Модуль отчета включает в себя раздел для объекта **Report**(Отчет) и разделы для каждой области отчета. Для полей отчета не определены события, поэтому отсутствуют и соответствующие разделы в модуле кода.

10. Охарактеризуйте назначение и состав Инспектора объектов.

Любой размещаемый на форме компонент характеризуется некоторым набором параметров: положением, размером, цветом и т. д. Часть этих параметров, например, положение и размеры компонента, программист может изменять, манипулируя с компонентом в окне формы. Для изменения других параметров предназначено окно Инспектора объектов. Это окно содержит две страницы - Properties (Свойства) и Events (События). Страница properties служит для установки нужных свойств компонента, страница Events позволяет определить реакцию компонента на то или иное событие. Совокупность cвойств отображает видимую сторону компонента: положение относительно левого верхнего угла рабочей области формы, его размеры и цвет, шрифт и текст надписи на нем и т. п.; совокупность событий - его поведенческую сторону: будет ли компонент реагировать на щелчок мыши или на нажатие клавиш, как он будет вести себя в момент появления на экране или в момент изменения размеров окна и т. п.

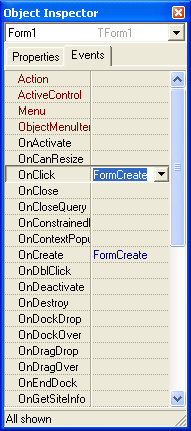
Каждая страница окна Инспектора объектов представляет собой двухколончатую таблицу, левая колонка которой содержит название свойства или события, а правая - конкретное значение свойства или имя подпрограммы обрабатывающей соответствующее событие.

Строки таблицы выбираются щелчком мыши и могут отображать простые или сложные свойства. К простым относятся свойства, определяемые единственным значением - числом, строкой символов, значением True (Истина) или False (Ложь) и т. п. Например, свойство caption (Заголовок) представляется строкой символов, свойства Height (Высота) и width (Ширина) - числами, свойство Enabled (Доступность) - значениями True или False. Сложные свойства определяются совокупностью значений. Слева от имени таких свойств указывается значок “+”, а щелчок мышью по этому символу приводит к раскрытию списка составляющих сложного свойства. Чтобы закрыть раскрытый список, нужно щелкнуть по значку “-” сложного свойства.

В верхней части окна Инспектора объектов располагается раскрывающийся список всех помещенных на форму компонентов. Поскольку форма сама по себе является компонентом, ее имя также присутствует в этом списке. В отличие от предыдущих версий раскрывающийся список содержит не только имена компонентов, но и их классы.

В локальном меню окна, которое появляется после щелчка по нему правой кнопкой, имеется ряд опций, позволяющих настроить окно. В частности, после выбора stay on Top, окно Инспектора объектов будет “всплывать” над всеми другими окнами независимо от его активности. Такое состояние окна удобно при частом его использовании, например, при конструировании сложной формы, содержащей множество компонентов. Если выбрать в локальном меню опцию Arrange и затем by Category, все строки окна Инспектора объектов будут представлять собой раскрывающиеся списки свойств, упорядоченные по категориям (рис.4).

*Поэкспериментируйте самостоятельно с локальным меню Инспектора объектов, чтобы понять назначение тех или иных его опций. Если вы случайно или намеренно сделаете окно невидимым, нажмите F 11 или выберите опцию view | object inspector, чтобы оно вновь появилось на экране.*



11. Свойства объекта и назначение их объектам. Обязательные свойства объекта.

Свойства объектов в JavaScript являются теми же самыми переменными, за тем исключением, что они закреплены за объектом. Как и все переменные JavaScript, имя объекта (которое тоже может быть переменной) и имя свойства чуствительны к регистру. Свойства объекта определяют его индивидуальные характеристики.

Получить значение свойства можно с помощью операторов точки (.) или квадратных скобок ([]). Значение в **левой части** оператора должно быть **ссылкой на объект**, обычно это просто имя переменной, содержащей ссылку на объект, но это может быть любое допустимое в JavaScript выражение, являющееся объектом.

Значение в **правой части** оператора должно быть **именем свойства**. При использовании оператора **точка** (.) справа должен находиться простой **идентификатор**, соответствующий имени свойства. Доступ к любому имени свойства, которое содержит невалидный JavaScript идентификатор (например, имя свойства содержит в себе пробел, тире или начинается с цифры), может быть получен с использованием квадратных скобок ([]).

Значение свойства может быть любым: как значением простого типа, так и другим объектом. Значения можно вкладывать друг в друга, причем вложений можно делать сколько угодно:

Значение объекта может быть обновлено путем присваивания. Если свойство с таким именем уже существует в объекте, его значение заменяется:

Квадратные скобки также позволяют обратиться к свойству, имя которого хранится в переменной:

Если **имя свойства хранится в переменной**, то единственный способ к нему обратиться – это применение оператора [] (**квадратные скобки**).

**Определение методов**

**Метод** – это свойство объекта, значением которого является функция. Методы определяются так же, как и обычные функции, за тем исключением, что они присваиваются свойству объекта. Вызов метода осуществляется точно также, как и вызов обычной функции – с помощью оператора вызова ():

Методы объектов можно добавлять и удалять в любой момент, точно также как вы поступаете с обычными свойствами объектов:

Для доступа к свойствам объекта внутри метода используется **ключевое слово this**. Использование **this** позволяет ссылаться на объект, к которому принадлежит метод:

Здесь вместо **this** внутри функции (метода) **greeting** можно было бы обратиться к объекту, используя переменную **person** (сам объект):

В предыдущем примере мы определили функцию внутри свойства **greeting** во время определения объекта **person**. Однако, мы можем определить сначала функцию, а затем закрепить ее за свойством **person.fun**. В таком случае поведение **this** не изменится:

Эти примеры показывают, что на поведение **this** совсем не влияет то, где была определена функция, а имеет значение только то, что функция была вызвана из свойства объекта.

Через ключевое слово **this** метод может получить доступ не только к любому свойству объекта, но и передать куда-нибудь ссылку на сам объект:

Известно, что каждый объект содержит ссылку на свой объект-прототип. Если метод находится где-то в цепочке прототипов, то **this** ссылается на **объект**, **на котором был вызван метод**, т.е. как будто этот метод является методом самого объекта, а не прототипа.

В этом примере объект **person**, не имеет собственного свойства (метода) **fullName**, а наследует это свойство от своего прототипа **prototypeObject**. Поскольку поиск свойства начался с **person.fullName**, то **this** внутри функции **fullName** будет ссылаться на объект **person**. Таким образом, если **fullName** вызывается как метод объекта **person**, то и **this** относится к **person**. Это очень важная особеность прототипного наследования JavaScript.

**Удаление свойств**

Оператор [delete](http://wm-school.ru/js/js_other%20operators.php" \l "nav_4" \t "_blank) удаляет свойство из объек­та. Он удаляет свойство объекта, если оно есть. Удаление не касается свойств объектов, связанных по цепочке прототипов. Если свойство с таким же именем существует в цепочке прототипов, объект унаследует это свойство из прототипа.

Выражение **delete** возвращает true, если удаление свойства прошло успешно. **delete** вернет false только в том случае, когда свойство не было удалено (например, если свойство относится к защищенному объекту, принадлежащему браузеру). **true** возвращается даже в том случае, если свойство, которое вы пытаетесь удалить, не существует в объекте:

##### Расположение объекта

Положение левого верхнего угла элемента управления задается свойствами Left (слева) и Top (сверху), определяющими координаты этого угла относительно левого верхнего угла контейнера - элемента, в который вложен данный. Контейнером может оказаться само диалоговое окно или вкладка элемента MultiPage или TabStrip.

Если оба параметра равны 0, левый верхний угол элемента совпадает с верхним левым углом контейнера. Для большинства систем координаты Left и Top должны находиться в диапазоне от -32767 до +32767. При отрицательных значениях на экране будет видна только часть элемента управления. Значения свойств Top и Left автоматически изменяются, когда пользователь перемещает элемент управления по экрану.

Размеры объекта на экране задаются свойствами Height (высота) и Width (ширина). Единица измерения - точка. Рекомендуемый диапазон значений этих параметров от 0 до 32767. Реальный диапазон зависит от видеосистемы компьютера. Значения свойств Height и Width автоматически изменяются, когда пользователь изменяет размер элемента управления на экране. При этом предыдущие значения размера запоминаются в свойствах OldHeight и OldWidth.

##### Параметры внешнего вида объекта

Свойство BackColor определяет цвет фона объекта. При задании из списка свойств объекта на этапе проектирования цвет можно выбрать непосредственно из таблицы цветов. При установке из программы в качестве значений используются целые числа. Цвет можно задавать, используя функцию RGB. Она определяет цвет как смешение трех компонентов, ѕ красной, зеленой и синей составляющим, каждая из которых принимает значения от 0 до 255. Например, RGB (255, 255, 0) задает желтый цвет. Чаще всего, цвет задается константами VBA, например, константа vbGreen определяет зеленый цвет.

Свойство BackStyle определяет прозрачность фона. По умолчанию его значение fmBackStyleOpaque = 1. При этом фон непрозрачен, виден его цвет, заданный свойством BackColor. Если значение BackStyle равно fmBackStyleTransparent = 0, фон прозрачен и виден объект, лежащий под данным объектом. На прозрачность рисунков свойство BackStyle не влияет.

Свойство ForeColor задает цвет переднего плана объекта (текста) ; возможные значения этого свойства такие же, как и у BackColor.

Свойство BorderColor задает цвет рамки (границы) объекта; возможные значения такие же, как и у BackColor.

Свойство BorderStyle определяет наличие или отсутствие рамки. Если его значение - fmBorderStyleNone = 0, видимая рамка у элемента управления отсутствует. При значении fmBorderStyleSingle = 1, у объекта есть рамка (в одну линию). Это значение устанавливается по умолчанию для всех объектов с рамками, кроме элементов управления ComboBox, Frame, Label, ListBox и TextBox, у которых по умолчанию рамки нет.

Альтернативный способ задания внешнего вида границы объекта предоставляет свойство SpecialEffect ; вот его значения:

* fmSpecialEffectFlat = 0 - внешний вид объекта плоский, он отделяется от окружения границей и цветом; установлено по умолчанию для элементов Image и Label, но может быть установлено для любого элемента;
* fmSpecialEffectRaised = 1 - верхняя и левая границы объекта подсвечены, а правая и нижняя затенены;
* fmSpecialEffectSunken = 2 - верхняя и левая границы объекта затенены, а правая и нижняя подсвечены; элемент углублен в окружающее окно; установлено по умолчанию для элементов типа Frame, CheckBox и OptionButton ;
* fmSpecialEffectEtched = 3 - углубленная рамка по краям элемента;
* fmSpecialEffectBump = 6 - объект выделен выступом снизу и справа, а сверху и слева плоский.

Отметим, что значения 1, 3 и 6 не используются для элементов типа CheckBox и OptionButton.

Для изображения границ объекта используются свойства SpecialEffect либо BorderStyle, но не оба вместе. При установке одного из них система автоматически обнуляет другое.

Свойство ControlTipText определяет краткий текст, который появится на экране рядом с объектом, когда указатель мыши будет находиться на данном объекте.

Булево свойство Visible определяет, виден ли объект на экране; по умолчанию True - объект виден.

##### Шрифт

Свойство Font (шрифт) определяет характеристики символов, используемых в тексте диалогового окна или элемента управления. Значение этого свойства имеет тип Object. Основное свойство шрифта - имя Name. Если оно не указано, используется *системный шрифт*.

Визуальные характеристики шрифта устанавливают булевы свойства: Bold - полужирное начертание, Italic - наклонное, Underline - подчеркивание. Размер шрифта в точках задается свойством Size. Свойство Weight задает затемненность символов при печати.

##### Свойства поведения объекта

Булево свойство AutoSize обеспечивает автоматическое приспособление размеров элемента управления к размерам его внутреннего содержимого (названия, текста, элемента списка). Значение по умолчанию False - размеры элемента считаются постоянными; если размеры содержимого выходят за рамки объекта, оно обрезается.

Булево свойство Enabled определяет, может ли элемент управления оказаться в фокусе в результате действий пользователя. True (по умолчанию) делает элемент доступным. False отключает элемент, т. е. не позволяет пользователю получить доступ к нему с помощью мыши и клавиатуры (но элемент управления и в этом случае доступен для процедур программы). Отключенный элемент "сереет", т. е. выглядит на экране более тускло, чем включенный.

Если отключена рамка Frame, одновременно недоступны все находящиеся в ней элементы управления.

Булево свойство TabStop определяет, может ли пользователь установить фокус на данный элемент управления, используя стандартные клавиши Tab или Shift+Tab; по умолчанию равно True, т. е. элемент доступен.

Когда в результате нажатия клавиши Tab или комбинации Shift+Tab фокус должен переместиться на элемент MultiPage или TabStrip, он устанавливается на первую не отключенную закладку соответствующего элемента, т. е. на страницу со значением True свойства Enabled. Если таковой нет, соответствующий элемент отключен и не может попасть в фокус.

Если для командной кнопки значение Enabled - True, а TabStop - False, она не попадает в фокус при обходе элементов с помощью клавиши Tab, но окажется в фокусе, если ее щелкнуть (при условии, что ее свойство TakeFocusOnClick установлено в True ).

Булево свойство Locked определяет, может ли пользователь редактировать объект (изменять его значение). По умолчанию оно равно False, т. е. содержимое объекта доступно для редактирования.

Комбинации свойств Enabled и Locked задают различные варианты поведения элемента управления:

* - оба свойства истинны: элемент может попасть в фокус и пользователь может копировать, но не редактировать данные элемента; изображение элемента в диалоговом окне нормальное (не серое) ;
* - Enabled истинно, Locked ложно: элемент управления может попасть в фокус, а его содержимое можно копировать и изменять;
* - Enabled ложно, Locked истинно: элемент не может попасть в фокус и выглядит тусклым; его данные нельзя ни изменить, ни скопировать; то же происходит, когда оба эти свойства ложны.

#### Другие свойства

Свойство TabIndex определяет порядковый номер элемента управления в обходе всех элементов с помощью клавиши Tab. Нумерация элементов начинается с 0.

Свойство ControlSource связывает данные основного документа приложения с состоянием элемента управления (значением свойства Value ). Значение этого свойства - строка, задающая ячейку рабочего листа, связанную с данным элементом. Например, чтобы задать в качестве значения этого свойства для флажка CheckBox1 ячейку С5, используется присвоение вида:

CheckBox1. ControlSource = "C5"

При этом изменения состояния элемента управления будут автоматически заноситься в эту ячейку, и наоборот, изменения в данных будут отражаться на состоянии элемента управления. Если текущее значение свойства Value для элемента равно Null, ячейка, связанная с ним не содержит никакого значения (пустая). По умолчанию значение ControlSource - пустая строка.

Свойство Tag задает текстовый комментарий к объекту. Оно никак не связано с его другими свойствами и может быть использовано для хранения вспомогательной информации об элементе управления. Иногда свойство Tag служит альтернативой свойству Name для идентификации элемента в программе.

Свойство Font определяет шрифт, используемый в изображении данного объекта. Его значение - объект класса Font, идентифицируемый по имени (свойству Name ). По умолчанию шрифт элемента управления тот, что установлен для его контейнера.

#### CheckBox - флажок (кнопка выбора)

Флажок, или кнопка независимого выбора, позволяет пользователю указать, выбирает или не выбирает он опцию (обычно ее название является также названием флажка, т. е. значением свойства Caption ). В зависимости от значения свойства TripleState у флажка может быть два или три состояния. По умолчанию значение этого свойства False, что соответствует двум значениям состояния флажка: True - флажок выбран, включен и False - флажок не выбран, выключен. Если для свойства TripleState установить True, у флажка появится также нейтральное состояние Null. Состояния флажка передаются в программу через свойство Value (Значение). Вот, например, возможные состояния флажка "С оптимизацией":

Верхний флажок находится в невыбранном состоянии ( False ), средний - в выбранном состоянии (значение True ) и нижний - в нейтральном состоянии (значение Null ). Кроме того, прямоугольная рамка вокруг названия нижнего флажка показывает, что он в фокусе (помечен).

#### ComboBox - комбинированный список

Комбинированный список - это элемент управления, соединяющий поле ввода с кнопкой и раскрывающимся списком. Работая с ним, пользователь может ввести значение непосредственно в поле ввода либо выбрать одно из значений в списке. Список состоит из строк данных. Данные в строке могут располагаться в одном или нескольких столбцах. Рассмотрим основные свойства объекта ComboBox.

* ListCount определяет, сколько элементов находится в списке. Свойство доступно только для чтения и изменяется автоматически вместе с добавлением (удалением) элементов в список.
* ListRows задает число одновременно видимых строк списка.
* ListIndex задает номер выбранной строки; возможные значения от -1 до ListCount - 1, т. е. значение ListCount всегда на 1 больше максимального значения свойства ListIndex, так как нумерация строк начинается с 0. Если ни один элемент в списке не выбран, ListCount равно 0, а значением ListIndex будет -1.
* ColumnCount определяет число столбцов в выводимом на экран списке; если оно равно 0, столбцы не выводятся, при значении -1 выводятся все столбцы.
* ColumnWidths задает ширину каждого столбца для списков с несколькими столбцами. Значение этого свойства - строка, в которой размеры столбцов перечислены через точку с запятой. Пустое значение или -1 означают, что ширина столбца вычисляется автоматически, 0 - отсутствие столбца, значения > 0 задают ширину столбца в точках. Можно также рядом с числовым значением указывать другую единицу измерения. По умолчанию ширина столбца не меньше 1 дюйма (72 точек). Например, если значение ColumnCount (число столбцов) равно 4, а ширина всего окна списка 12 см, значение ColumnWidth вида "56; 112; 56; 112" устанавливает ширину первого и третьего столбцов - 56 точек (2 см), а второго и четвертого - 112 точек (4 см), то же разбиение можно получить, задав "2 cm; 4 cm; 2 cm; 4 cm". Если же задать "4 cm; 0;;8 cm", второго столбца на экране не будет, третий будет иметь минимальную вычисляемую ширину - 1 дюйм, а в четвертом будет видна лишь левая часть шириной около 5. 5 см (при этом появится горизонтальная полоса прокрутки). Отсутствие значения (пустая строка) означает, что все столбцы имеют одинаковую ширину.
* TextColumn задает номер столбца, видного пользователю. Нумерация столбцов начинается с 1. 0 означает, что будет виден только выбранный элемент из строки ListIndex. При значении 1 выводится первый столбец, чья ширина, установленная свойством ColumnWidths, больше 0.
* BoundColumn указывает столбец со значением данных (свойством Value ) в списке со многими столбцами. Если свойство равно 0, значением свойства Value будет номер ListIndex выбранной пользователем на экране строки. Если BoundColumn > 0, значение свойства Value берется из указанного столбца. Таким образом, пользователь может выбрать на экране один элемент, а в качестве значения этого выбора в программе можно задать другой. Например, список может содержать названия месяцев в первом столбце и их номера - во втором. Установив TextColumn = 1, а BoundColumn = 2, мы получим элемент, в котором пользователь выбирает месяц по названию, а программа получает в качестве значения объекта номер выбранного месяца.
* Задать элементы списка можно программно, используя метод AddItem (например, в процедуре инициализации диалогового окна), либо установив при проектировании свойство RowSource. Его значение - строка, задающая диапазон ячеек Excel, из которых будут браться элементы списка. Например, следующие операторы задают список с 4 столбцами для элемента ComboBox1. Элементы этого списка берутся из ячеек b1:e6 текущего рабочего листа Excel:
* ComboBox1. ColumnCount = 4

ComboBox1. RowSource = "b1:e6"

* List - двухмерный массив с элементами списка. Обращение к нему:

объект. List (строка, столбец)

а элементы этого массива имеют тип Variant. Нумерация строк и столбцов начинается с 0. Это свойство можно использовать для инициализации списка с несколькими столбцами в элементах управления ComboBox и ListBox. Допустим, на уровне модуля определен двухмерный массив строк ListItems (4, 3) , содержащий 3 столбца значений, которые мы хотим предложить пользователю для выбора. Тогда этот массив можно связать с комбинированным списком CBox в процедуре инициализации диалогового окна.

Private Sub UserForm\_Initialize ()

СBox. ColumnCount = 3 ' задание числа столбцов списка

CBox. List () = ListItems ' присвоение значения списку

End Sub

* ListStyle определяет внешний вид списка. Если оно равно fmListStylePlain = 0, элементы списка выводятся в обычном виде без кнопок слева. Если же его значение - fmListStyleOption = 1, слева от каждого элемента списка выводится кнопка. Для списков с единственным выбором это кнопка-переключатель (OptionButton), а для списков с множественным выбором - кнопка-флажок (CheckBox). При выборе элемента в соответствующей кнопке появляется метка.
* MatchRequired и MatchEntry определяют поведение комбинированного списка при вводе пользователем данных в поле ввода. Если MatchRequired равно True, вводимый пользователем текст может стать значением элемента, лишь когда он совпадает с одним из элементов списка. По умолчанию ему присвоено False, что не требует от вводимых пользователем данных совпадения с элементами списка. Свойству MatchEntry по умолчанию устанавливается 1, означающее, что при вводе пользователем *очередного символа* в списке ищется первый элемент, для которого введенное слово является префиксом и выводится в качестве значения. Если MatchEntry равно 0, поиск происходит по первой букве слова, т. е. при повторном выборе одной и той же буквы в качестве значений перебираются все элементы списка, начинающиеся на нее. При MatchEntry равном 2 список не реагирует на набираемый пользователем текст. Этот текст доступен через свойство Text.

13. Задание и изменение свойств объектов при написании программы.

Выше уже говорилось, что, используя встроенные объекты, с помощью VBA мы можем управлять приложением. Прежде чем перейти к изучению встроенных объектов, надо понять, что же такое объект и как с ним работать. В объектно-ориентированном программировании объект обычно определяется через понятие класс. ***Класс*** – это сложный тип, объединяющий данные и код (программы) для их обработки, а ***объект*** – это переменная этого типа (экземпляр класса). Примерами объектов могут служить кнопка (объект класса CommandButton), рабочая книга (объект класса Workbook), форма (объект класса UserForm). В литературе обычно опускают слово класс и называют рабочую книгу объектом Workbook, а кнопку – объектом CommandButton.

Каждый объект имеет имя, с помощью которого иа него можно сослаться в тексте программы. У элементов управления и формы это значение свойства (Name). В отличие от других свойств, это свойство нельзя изменить во время выполнения программ.

Состояние объекта характеризуется текущим значением его атрибутов. Атрибутами могут быть не только простейшие величины, по и другие объекты. Атрибуты объекта называются свойствами. Свойства элементов управления и формы можно задать в **Окне свойств**. Для доступа к значению свойства в программе нужно указать имя объекта и, через точку, имя свойства. Если значение свойства в свою очередь является объектом, то нужно снова поставить точку и указать свойство этого свойства и т.д. Например:

Selection. Font.Bold = True

Объекты выполняют обработку данных с помощью методов. Метод – это процедура, связанная с объектом. Метод вызывается аналогично свойству. Если нужно, при вызове указываются значения фактических параметров:

Forml.Hide

Forml.Move 20, 20, 200, 100

***Событие –*** это действие, распознаваемое объектом (например, щелчок мышью, нажатие клавиши или переход в другое окно). Возникают события в результате действий пользователя или программы, могут быть инициированы операционной системой. У каждого объекта свой набор событий, созданный разработчиком. Если вы хотите, чтобы при наступлении события ваша программа что-то сделала, напишите процедуру обработки события. Имя процедуры всегда состоит из имени объекта и имени события, между которыми ставится знак подчеркивания. Например, процедура cmdOK Click будет выполняться, когда пользователь нажмет на кнопку с именем cmdOK, так как в этот момент для нее возникнет событие Click. Для форм процедуры обработки событий имеют имена Form\_событие, т.е. указывается не имя формы, а слово Form, например FormClick. Процедура обработки события размещается в том модуле, в котором используется объект.

Для создания процедуры обработки события перейдите в окно кода нужной формы, например с помощью кнопки **View Code** окна проекта. В верхней части окна имеются два списка. В левом списке выберите нужный объект, в правом – событие для этого объекта. Система автоматически создаст заготовку процедуры с именем процедуры в начале и инструкцией End Sub в конце. Допишите в нее нужные инструкции. Если процедура уже существует, она будет показана в окне. Не пишите заготовки процедур обработки событий вручную, так как некоторые события имеют параметры и вы можете ошибиться. Также имейте в виду, что двойной щелчок по форме или элементу управления приводит к переходу в окно кода формы и созданию процедуры обработки самого типичного события этого объекта (события по умолчанию).

Как мы уже выяснили, объект – это переменная специального типа. Работа с такими переменными в программе похожа на работу с переменными других типов: мы можем их объявлять, присваивать им значения, сравнивать, – но в то же время имеет свою специфику. На самом деле объектная переменная хранит ссылку на область памяти, в которой размещена вся информация, относящаяся к объекту. Поэтому любая объектная переменная всегда занимает 4 байта. Это определяется моделью памяти, а не классом. Используя объектную переменную, мы получаем доступ к свойствам, методам и событиям объекта.

Для описания объектной переменной используются обычные инструкции Dim, Public, Private или Static. В качестве типа можно указать Variant, Object или имя класса. Тип Object позволяет создать универсальную ссылку на любой объект. Он используется в том случае, если объектный тип не известен до выполнения процедуры. Например:

Dim 01 'Описывает 01 с типом Variant

Dim 02 As Object 'Описывает 02 с типом Object

Dim 03 As Font 'Описывает 03 с типом Font

Важно понимать, что описание объектной переменной не означает создание объекта. Чтобы переменная ссылалась на объект, объект нужно создать и присвоить эту ссылку в качестве значения переменной.

Присвоение значения объектной переменной выполняется с помощью инструкции **Set**:

**Set** ***Переменная*** = ***Выражение*** | **New** ***ИмяКласса*** | **Nothing**

Слева от знака "равно" может стоять в том числе имя свойства, если свойство является объектом. Выражение, стоящее справа, может быть именем другой переменной того же типа, функцией, методом или свойством, которые возвращают объект того же типа. Ключевое слово New используется для создания нового экземпляра объекта указанного класса. Присваивание Nothing (пустая ссылка) разрывает связь переменной с объектом. Когда переменная получает значение Nothing, все системные ресурсы и ресурсы памяти, выделенные для объекта, на который имелась ссылка, освобождаются, если никакие другие переменные на него не ссылаются.

При использовании инструкции Set копия объекта обычно не создается. Значением переменной является ссылка на объект. Следовательно, несколько объектных переменных могут ссылаться на один и тот же объект и любые изменения объекта отражаются на всех переменных, которые ссылаются на него.

В качестве примера рассмотрим создание нового рабочего листа (объекта Worksheet). В первом случае лист создается с помощью метода Add, который возвращает ссылку на новый лист, и эту ссылку мы присваиваем переменной newSheetl. Во втором случае лист создается с помощью New. Использовать переменные newSheetl и newSheet2 мы можем одинаково:

Set newSheetl = Worksheets.Add

newSheetl.Name = "Отчет"

Set newSheet2 = New Worksheet

newSheet2.Name = "Вычисления"

Для сравнения двух переменных, содержащих ссылки на объекты, нельзя использовать обычную операцию "равно". Для этих целей предназначена специальная операция Is. Если переменные содержат ссылки на один объект, результат имеет значение True, в противном случае результат имеет значение False.

В приведенном ниже примере выполняется поиск слова "Образец" в столбце В листа "Лист1". В случае обнаружения нужного слова метод Find вернет ссылку на найденную ячейку, в противном случае – пустую ссылку. Результат поиска сохраняется в переменной FC. Так мы можем проверить, чем закончился поиск:

Set FC = Worksheets("JlMCTl ").Columns("B").Rnd(O6pa3eH")

If FC Is Nothing Then

MsgBox "Слово не найдено"

Else

MsgBox "Слово находится в ячейке" & FC.Address

End If

Иногда во время выполнения программы требуется узнать класс объекта, на который ссылается переменная. В этом случае следует использовать операцию TypeOf. Так, например, мы можем проверить тип активного элемента управления в форме (TextBox, PictureBox – это имена классов, свойство ActiveControl хранит ссылку на активный объект):

If TypeOf UserForml.ActiveControl Is TextBox Then

'Активный элемент – TextBox

Elself TypeOf UserForml .ActiveControl Is PictureBox Then

'Активный элемент – PictureBox

Else

MsgBox "Другой тип данных"

End If

Инструкция With дает возможность выполнить последовательность действий над объектом, нс повторяя его имени. Обращение к свойствам и методам объекта внутри инструкции начинается с точки. Инструкция не только облегчает чтение и написание программы за счет уменьшения объема текста, по и позволяет транслятору создать более эффективный код, так как ссылка па объект вычисляется один раз. Синтаксис инструкции:

**With** ***Объект***

[***Инструкции***]

**EndWith**

Пример использования инструкции:

With Worksheets("Листl").Range("A1:C10")

.Font.Bold = True .Interior.Color = RGB(255, 255, 0)

End With

В примере форматируются ячейки А1:С10 листа "Лист1" (шрифт – полужирный, цвет фона – желтый). Таким образом, ссылка на необходимый диапазон указывается один раз, хотя изменяются значения двух свойств объекта.

Отдельные объекты можно объединять в коллекции, подобно тому как из отдельных элементов можно создать массив. Коллекция – это специальный объект-контейнер, содержащий другие объекты. Создание коллекций, так же как и создание классов, выходит за рамки этого курса, поэтому нам нужно только научиться работать с существующими коллекциями. В приложениях Microsoft используется очень много коллекций, так как они позволяют легко обработать все объекты заданного класса.

Добавление, удаление и другие операции с элементами коллекции реализуются с помощью методов соответствующих классов. Доступ к конкретному элементу коллекции возможен по номеру или по ключу. Нумерация элементов начинается с единицы. При использовании номера нужно указать имя коллекции и в круглых скобках – номер элемента, например Worksheets(1). Доступ по ключу возможен в том случае, если при добавлении элемента в коллекцию ключ был задан. Ключ – это строка. В коллекции рабочих листов ключом является имя листа, поэтому мы можем обратиться к нужному листу, например, так: Worksheets("Peшение"). В данном случае совершенно не важно, какой помер у этого листа, тем более что номер во время выполнения программы может измениться из-за добавления или удаления листов. Количество элементов в коллекции всегда можно узнать с помощью свойства Count. Это свойство доступно только для чтения.

Перебор всех элементов коллекции удобно выполнять с помощью специальной разновидности цикла For Each ... Next, которая имеет следующий синтаксис:

Переменная цикла может иметь тип Variant, Object или тип конкретного объекта. Для досрочного выхода из цикла используется инструкция Exit For. Цикл выполняется для каждого элемента коллекции. Имя переменной после Next

рекомендуется указывать в тех случаях, когда используются вложенные циклы.

В качестве примера рассмотрим, как подсчитать количество пустых ячеек в выделенном диапазоне. В данном случае объект Selection – это коллекция, состоящая из объектов типа Range:

N = О

Dim с as Range

For Each c In Selection

If c.Value = "" Then N = N + 1

Next c

14. События, их роль в программе на языке C++ Builder.

ы неоднократно рассматривали применение различных событий компонентов в примерах программ. Настало время рассмотреть каждое из них в отдельности. В табл. 22.1 приведены основные события компонентов и даны описания ситуаций, когда они происходят.

**Таблица**. Основные события компонентов и описание ситуаций, когда они происходят

|  |  |
| --- | --- |
| **Событие** | **Когда происходит событие** |
| **OnActivate** | При активизации объекта |
| **OnCanResize** | При изменении размеров |
| **OnContextPopup** | При вызове контекстного меню |
| **OnClick** | По щелчку мыши на компоненте и при другой активизации компонента |
| **OnClose** | При закрытии |
| **OnDblClick** | По двойному щелчку мыши на компоненте |
| **OnChange** | После изменения графического объекта |
| **OnDragDrop** | В момент отпускания перетаскиваемого компонента над данным компонентом |
| **OnDragOver** | Когда пользователь  перемещает  перетаскиваемый  объект над компонентом |
| **OnEndDrag** | В момент прерывания или окончания перетаскивания компонента |
| **OnEnter** | В момент получения элементом фокуса |
| **OnExit** | В момент потери элементом фокуса |
| **OnKeyDown** | При нажатии пользователем любой клавиши |
| **OnKeyPress** | При нажатии пользователем клавиши символа |
| **OnKeyUp** | При отпускании пользователем любой клавиши |
| **OnMouseUp** | В момент нажатия пользователем клавиши мыши над компонентом |
| **OnMouseMove** | При перемещении курсора мыши над компонентом |
| **OnPaint** | При получении сообщения Windows о необходимости перерисовать изображение |
| **OnProgress** | Во   время   медленных   процессов  изменения   графического изображения |
| **OnStartDrag** | Когда пользователь начал перетаскивание объекта |
| **OnEnter** | При вводе |
| **OnExit** | При выходе |

15. Функции обработки событий и их написании в программе.

Обработчики событий (*event handlers*) – это оркестр, исполняющий интерактивную пьесу вашего фильма. Каждый обработчик содержит *скрипт*, выполняемый только тогда, когда происходит определенное событие. Каждый *скрипт* в фильме запускается событием – *пользователь* навел *указатель* мыши на кнопку или нажал клавишу, или фильм достиг определенного кадра на монтажном столе, и так далее.

В *ActionScript* обработчик события (за исключением событий, связанных с кадрами) обычно указан в первой строке всякого скрипта. Примерно так:

Когда происходит (типСобытия) {

делать то-то;

делать то-то;

}

События, привязанные к кадрам, происходят, когда монтажный стол при воспроизведении достигает кадра, содержащего *скрипт*. Для скрипта, помещенного в *кадр*, не нужен обработчик события – такой *скрипт* будет выполнен при достижении данного кадра. Так, если бы приведенный выше *скрипт* был привязан к кадру, он выглядел бы так:

делать то-то;

делать то-то;

Чем лучше вы поймете концепции обработчиков событий, тем более полным будет ваш *контроль* над интерактивными элементами. Правильно используя обработчики, вы сможете создать такую интерактивную среду, в которой *пользователь* будет чувствовать себя как дома.

Многие компьютерные программы позволяют пользователю перетаскивать элементы на экране, менять размер окон, делать те или иные настройки с помощью регуляторов, создавать собственные "шедевры искусства" при помощи виртуальных инструментов. Все эти режимы интерактивного взаимодействия зависят от того, как реализована в программном обеспечении обработка различных событий (нажатий кнопок мыши, перемещения мыши, ввода с клавиатуры и т.п.).

### Выбор нужного обработчика

Правильное использование обработчиков событий – одна из наиболее сложных проблем, стоящих перед пользователем *ActionScript*, ведь большинство из нас не задумывается о том, как именно и почему происходят события вокруг нас. А ведь при создании интерактивной среды важно именно это – четко представлять себе, что с чем и как взаимодействует.

Знакомя вас с обработчиками событий *Flash*, мы будем рассматривать их *связь* с событиями реального мира. Не забывайте, что во *Flash* каждому событию соответствует действие (и ответная *реакция*).

**Примечание** Далее в этом уроке мы обсудим методы обработчиков событий – под ними мы подразумеваем расширения стандартных обработчиков.

### События мыши

Обработчики этой группы управляют выполнением скриптов для событий, связанных с манипуляциями мышью над экранными кнопками и экземплярами фильмов-символов.

**Примечание** Если вы знакомы с реализацией событий мыши во *Flash* 5, то знаете, что там их можно было применять только к кнопкам. Во *Flash* *MX* же события мыши можно присоединять также к экземплярам фильмов-символов (об этом будет детально рассказано в упражнении).

В реальном мире, если вы к чему-то прикасаетесь или на что-то нажимаете (будь то человек или кубик льда), вы ожидаете реакции: кубик льда начинает таять, человек тоже так или иначе откликается. Обработчик события on(press) служит для имитации событий вроде прикосновений, захватов, нажатий. Этот обработчик используется для запуска скрипта в ситуации, когда указатель наведен на кнопку или фильм-символ и кнопка мыши нажата.

#### Отпускание, освобождение: on (release)

Отпуская что-либо, прекращая контакт, вы обычно тем самым завершаете взаимодействие с предметом. Данный обработчик имитирует всякого рода освобождение – от нажатия или *захвата*. Применяется для запуска скрипта в ситуации, когда пользователь отпустил кнопку мыши (которую он до этого нажал на кнопке или экземпляре фильма-символа).

#### Сдвиг, перетаскивание: on (releaseOutside)

[увеличить изображение](https://www.intuit.ru/EDI/29_10_14_1/1414531052-28803/tutorial/231/objects/2/files/02_04.jpg)

Вообразите колоду карт, лежащую на столе. Вы нажимаете пальцем на верхнюю карту, сдвигаете ее с колоды и отпускаете палец. Во Flash вы можете представить себе кнопку, которую пользователь нажал, переместил мышь и отпустил кнопку. Этот обработчик запускает скрипт в случае, когда пользователь нажал кнопку мыши на объекте (экранной кнопке или фильме-символе), а отпустил за пределами объекта – полезно для программирования всякого рода перемещений и сдвигов.

#### Управление с клавиатуры: on (keyPress)

Этим событием можно запрограммировать скрипт, который будет выполняться при нажатии пользователем какой-либо клавиши на клавиатуре. Клавишу можно задать любую – с буквой, цифрой, знаками препинания и *специальными символами*, клавиши-стрелки, *Backspace*, Insert, Home, End, Page Up и Page Down.

#### Указал, но не нажал: on (rollOver)

Для того, чтобы почувствовать жар от горячей плиты, не обязательно к ней прикасаться – достаточно поднести руку. Будем считать, что этот обработчик имитирует объекты, которые способны воздействовать на расстоянии – излучая тепло, холод, свет и тому подобное. Также этот обработчик можно использовать для отображения информации (всплывающей подсказки) при наведении указателя на кнопку или экземпляр фильма-символа. Итак, этот обработчик используется для запуска скрипта в ситуации, когда пользователь навел указатель мыши на кнопку или экземпляр фильма-символа.

#### Указатель отведен в сторону: on (rollOut)

Отведя руку от плиты, вы, естественно, перестаете чувствовать ее жар. Именно такую ситуацию имитирует этот обработчик. Вы можете использовать этот обработчик для запуска скрипта, когда пользователь отвел указатель от кнопки или фильма-символа (на который он перед этим указатель навел).

#### "Желобки", "впадины", трение: on (dragOver)

Перемещение вперед-назад в пределах какой-либо области – как, например, движется щетка, которой чистят ботинок. Данное событие позволяет имитировать такой тип деятельности, запуская скрипт каждый раз, когда указатель мыши проходит над какой-либо кнопкой или фильмом-символом, пока кнопка мыши остается нажатой.

#### "Ой": on (dragOut)

Данное событие позволяет имитировать ситуацию, когда вы, прикоснувшись к чему-либо нечаянно, тут же отдергиваете руку. Этот обработчик можно использовать для запуска скрипта в случае, когда пользователь навел указатель на кнопку или фильм-символ, нажал кнопку мыши и тут же убрал указатель в сторону (а кнопка мыши при этом осталась нажатой).

16. Как записать передачу данных из текстового окна в программу.

Для вывода текстовой информации наиболее часто применяются контролы Label и TextBox, которые мы уже использовали в параграфе "Простейший Windows Application проект". Контрол TLabel предназначен для вывода однострочной текстовой информаци и имеет стандартный набор свойств и методов, большинство из которых прозрачны или описаны в предыдущих параграфах.

Отметим, что в контрол можно помещать не только текст, но, чего нет в Delphi и Borland C++ Builder, и рисунки, аналогично, как это мы делалали для кнопок - он имеет такойже набор свойств(Image, ImageAlign, ImageList).

Интересным является свойство UseMnemonic, позволяющее определить, как будет интерпретироваться символ амперсанда (&), размещенный в свойстве Text. Если установлено значение True, то амперсанд из текста удаляется, а символ, перед которым он расположен, подчеркивается. Эта возможность применяется для определения клавиш быстрого доступа для выбора именно контрола Label - пользователь может комбинацией клавиш <А1t>+<подчеркнутая клавиша в метке> устанавливать фокус на контрол, а затем, используя клавишу Tab, на контрол, следующий по таб номеру за контролом Label. Например, если в форме расположено несколько контролов, среди которых имеется текстовое поле, в которое необходимо перейти нажатием клавиши быстрого доступа, то достаточно установить свойства Tabindex метки и текстового поля так, чтобы для метки это значение было на единицу меньшее, чем у текстового поля. Теперь, в программе, Вы можете выбрать клавишами быстрого доступа метку и, далее, нажать клавишу Tab - будет выбрано текстовое поле для ввода информации.

Вывод текста в контрол не вызывает трудностей, для этого контрол имеет свойство Text:

label1.Text="My Text";

17. Что такое комментарий, для чего он служит и как оформляется?

Коммента́рии — пояснения к исходному тексту программы, находящиеся непосредственно внутри комментируемого кода. Синтаксис комментариев определяется языком программирования. С точки зрения компилятора или интерпретатора, комментарии — часть текста программы, не влияющая на её семантику. Комментарии не оказывают никакого влияния на результат компиляции программы или её интерпретацию. Помимо исходных текстов программ, комментарии также применяются в языках разметки и языках описания.

18. Что представляет собой оператор, какие функции выполняет операторы в программе?

### Операторы цикла

*Операторы цикла* используются для организации многократно повторяющихся вычислений.

Любой цикл состоит из *тела цикла*, то есть тех операторов, которые выполняются несколько раз, *начальных установок*, блока *модификации* *параметра* *цикла* и проверки *условия выхода* из *цикла* ([рис. 3.3](https://www.intuit.ru/studies/courses/626/482/lecture/10935?page=2#image.3.3)), которая может размещаться либо до *тела цикла* (тогда говорят о цикле *с предусловием* ), либо после *тела цикла* (цикл *с постусловием* )

Один проход *цикла* называется *итерацией*. Переменные, принудительно изменяющиеся в цикле и использующиеся при проверке условия выхода из него, называются *параметрами цикла*. Целочисленные параметры *цикла*, изменяющиеся на *целое число* на каждой *итерации*, называются *счетчиками цикла*.

Нельзя передавать управление извне внутрь *цикла*. *Выход* из *цикла* возможен как при выполнении условия выхода, так и *по* операторам *break*, return или *безусловного перехода*.

#### Цикл с предусловием (while)

**Цикл с предусловием** реализует *структурную схему*, приведенную на [рис. 3.3](https://www.intuit.ru/studies/courses/626/482/lecture/10935?page=2#image.3.3)(а), и имеет вид:

**while ( выражение ) оператор;**

Выражение определяет условие повторения *тела цикла*, представленного простым или составным оператором. Если выражение не равно 0 (истинно), выполняется оператор цикла, после чего опять вычисляется выражение. Если при первой проверке выражение равно 0, цикл не выполнится ни разу. Тип выражения должен быть арифметическим или приводимым к нему.

Пример (программа печатает таблицу *значений функции* y=x2+1 во введенном диапазоне):

#include <cstdio>

using namespace std;

int main()

{

float Xn, Xk, Dx;

printf("Введите диапазон и шаг изменения аргумента: ");

scanf("%f%f%f", &Xn, &Xk, &Dx);

printf("| X | Y |\n"); //шапка таблицы

int X = Xn; //начальные установки цикла

while (X<=Xk)

{ //проверка условия выхода

printf("| %5.2f | %5.2f |\n", X, X\*X + 1); //тело

X += Dx; //модификация

}

}

#### Цикл с постусловием (do while)

**Цикл с постусловием** реализует *структурную схему*, приведенную на [рис. 3.3](https://www.intuit.ru/studies/courses/626/482/lecture/10935?page=2#image.3.3)(б), и имеет вид:

**do оператор while выражение;**

Сначала выполняется простой или *составной оператор*, составляющий тело цикла, а затем вычисляется выражение. Если оно не равно 0 (истинно), тело цикла выполняется еще раз, и так далее, пока выражение не станет равным нулю или в теле цикла не будет выполнен какой-либо *оператор передачи управления*. Тип выражения должен быть арифметическим или приводимым к нему.

Пример (программа осуществляет проверку ввода):

#include <iostream>

using namespace std;

int main(){

char answer;

do{

cout << "\nКупи слоника! ";

cin >> answer;

}while (answer != 'y');}

#### Цикл с параметром (for)

Цикл с параметром имеет следующий формат:

for ( инициализация; выражение; модификации) оператор;

*Пример*:

for ( int i = 1; i < 6; ++i )

printf( " %d ", i );

printf( "вышел зайчик погулять" );

*Инициализация* используется для объявления и присвоения начальных значений величинам, используемым в цикле. В этой части можно записать несколько операторов, разделенных запятой.

*Выражение* определяет условие выполнения цикла: если оно не равно 0 (истинно), цикл выполняется.

*Модификации* выполняются после каждой *итерации* цикла и служат обычно для изменения параметров цикла. В части модификаций можно записать несколько операторов через запятую.

Простой или составной *оператор* представляет собой тело цикла. Любая из частей оператора for может быть опущена (но точки с запятой надо оставить на своих местах!). В любой части можно использовать операцию "запятая" (последовательное вычисление), например:

for (int i = 1, s = 0; i<=100; i++)

s += i; // сумма чисел от 1 до 100

Пример (программа печатает таблицу *значений функции* y=x2+1 во введенном диапазоне):

#include <cstdio>

using namespace std;

int main()

{

float Xn, Xk, Dx;

printf("Введите диапазон и шаг изменения аргумента: ");

scanf("%f%f%f", &Xn, &Xk, &Dx);

printf("| X | Y |\n");

for (int X = Xn; X<=Xk; X += Dx)

printf("| %5.2f | %5.2f |\n", X, X\*X + 1);

}

### Операторы передачи управления

В С++ есть пять операторов, изменяющих естественный порядок выполнения вычислений:

* оператор выхода из цикла и *переключателя* *break* ;
* *оператор перехода* к следующей *итерации* цикла *continue* ;
* оператор возврата из функции return ;
* оператор *безусловного перехода* *goto* ;
* оператор генерации исключения *throw*.

*Оператор выхода* *break* используется для немедленного завершения *оператора цикла* или *switch*. Выполнение передается следующему после завершенного оператору.

*Оператор перехода к следующей итерации* *цикла* *continue* пропускает все *операторы*, оставшиеся до конца *тела цикла*, и передает управление на модификацию параметров *цикла* (в случае *цикла* for ) и начало следующей *итерации*.

*Оператор возврата из функции*return завершает выполнение функции и передает управление в точку ее вызова. Мы рассмотрим его вместе с функциями на следующей лекции.

*Оператор безусловного перехода**goto* имеет формат:

goto метка;

В теле той же функции должна присутствовать ровно одна конструкция вида:

метка: оператор;

Оператор *goto* передает управление на помеченный оператор. *Метка* - это обычный *идентификатор*, *областью видимости* которого является *функция*, в теле которой он встречается.

Использование *оператора безусловного перехода* оправдано в двух случаях:

* принудительный выход вниз по тексту программы из нескольких *вложенных циклов* или переключателей;
* переход из нескольких мест функции в одно (например, если перед выходом из функции необходимо всегда выполнять какие-либо действия).

В остальных случаях для записи любого алгоритма существуют более подходящие средства.

*Исключительную ситуацию* (или просто *исключение* ) генерирует либо программист с помощью оператора *throw*, либо сама среда выполнения. Это происходит, когда во *время выполнения* программы возникают какие-либо ошибки, например, *деление* на ноль или *переполнение*. Механизм *обработки исключений*, реализованный в С++, позволяет реагировать на подобные ошибки и таким образом избегать аварийного завершения программы.

Пример использования оператора *break*. *Программа* вычисляет *значение* синуса вещественного аргумента х с заданной точностью *eps* с помощью разложения в бесконечный ряд:

sh x = x - x3/3! + x5/5! - x7/7! + …

*Точность* достигается при |Rn| < *eps*, где Rn —остаточный член ряда, который для данного ряда можно заменить величиной очередного члена ряда, прибавляемого к сумме.

#include <cmath>

#include <iostream>

using namespace std;

int main()

{

const int MaxIter = 500; //ограничитель количества итераций

double x, eps;

cout << "\nВведите аргумент и точность: ";

cin >> x >> eps;

bool done = true; //признак успешного вычисления

double y = x, ch = x; // сумма и первый член ряда

for (int n = 0; fabs(ch) > eps; n++)

{

ch \*= -x \* x /(2 \* n + 2)/(2 \* n + 3); //очередной член ряда

y += ch;

if (n > MaxIter)

{

cout << "\nРяд расходится!";

done = false; break;

}

}

if (done)

cout << "\nЗначение функции: " << y;}