ТЕОРЕТИЧЕСКИЙ КУРС ДИСЦИПЛИНЫ «ПРИКЛАДНОЕ ПРОГРАММИРОВАНИЕ»

ТЕМА №1. СОСТАВ И СТРУКТУРА ПАКЕТА: УПРАВЛЯЮЩИЕ, ОБСЛУЖИВАЮЩИЕ И ОБРАБАТЫВАЮЩИЕ МОДУЛИ, ИНФОРМАЦИОННАЯ БАЗА

ВВЕДЕНИЕ В ПРЕДМЕТ. ПОНЯТИЕ ППП

ЦЕЛИ И ЗАДАЧИ ДИСЦИПЛИНЫ

- Изучение основных принципов, используемых в разработке интегрированных программных продуктов.
- Изучение структуры, состава и назначения компонентов интегрированного ПО, а также средств организации взаимодействия между компонентами и инструментальных средств расширения функциональности.
- Формирование навыков работы со средствами автоматизации решения прикладных задач.
- Формирование навыков использования встроенных средств разработки.

Требования к уровню освоения дисциплины

В результате изучения дисциплины студенты должны:

- знать принципы построения прикладных информационных систем
- уметь использовать современные программные средства для обработки разнородной информации;
- уметь автоматизировать процесс решения прикладных задач с помощью встроенных языков программирования;

- иметь представление о современном состоянии и тенденциях развития рынка прикладного ПО.

ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

Информационная система (ИС) - организационно упорядоченная совокупность документов (массивов документов) и информационных технологий, в том числе с использованием средств вычислительной техники и связи, реализующих информационные процессы. Информационные системы предназначены для хранения, обработки, поиска, распространения, передачи и представления информации.

Автоматизированная (информационная) система (АС) - совокупность программных и аппаратных средств, предназначенных для хранения и/или управления данными и информацией и производства вычислений и управляемая человеком-оператором (в этом главное отличие автоматизированной системы от автоматической).

Многоуровневое представление ИС - модель представления информационной системы в виде совокупности взаимосвязанных уровней, разделенных по функциональному назначению (рис. 1).

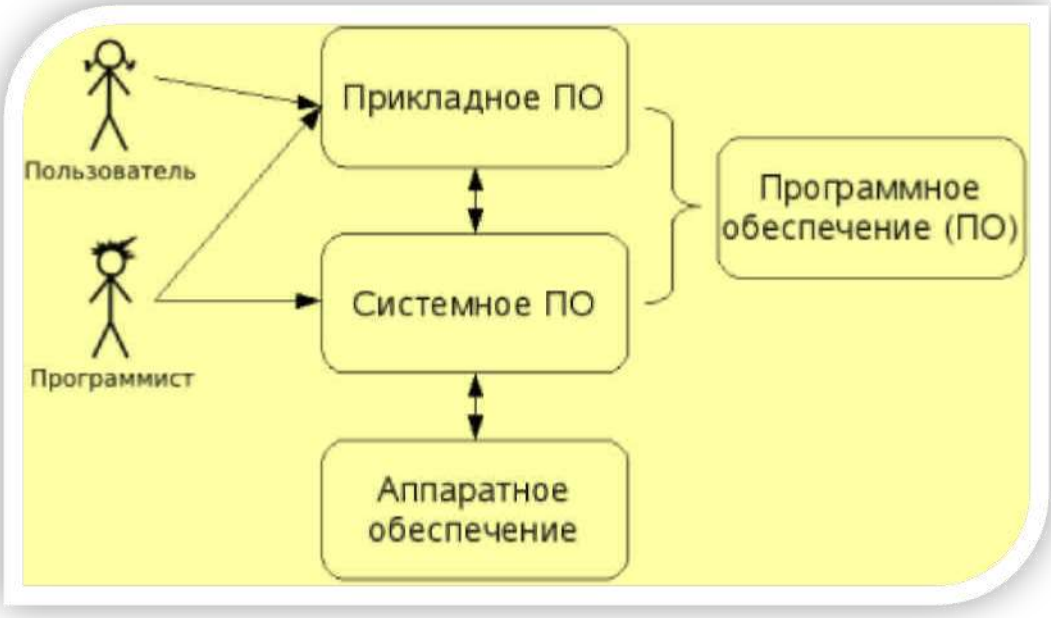


РИСУНОК 1. **МНОГОУРОВНЕВОЕ ПРЕДСТАВЛЕНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ.**

Аппаратное обеспечение ИС - комплекс электронных, электрических и механических устройств, входящих в состав информационной системы или сети.

Программное обеспечение (ПО) - совокупность программ и данных, предназначенных для решения определенного круга задач и хранящиеся на машинных носителях.

Программа - последовательность формализованных инструкций, представляющих алгоритм решения некоторой задачи и предназначенная для исполнения устройством управления вычислительной машины. Инструкции программы записываются при помощи машинного кода или специальных языков программирования. В зависимости от контекста термин «программа» может относится к исходным текстам, при помощи которых записывается алгоритм, или к исполняемому машинному коду.

Программист - специалист, занимающийся разработкой и проверкой программ. Различают системных и прикладных программистов.

Пользователь - человек, принимающий участие в управлении объектами и системами некоторой предметной области и являющийся составным элементом автоматизированной системы.

Прикладное программное обеспечение - программное обеспечение, ориентированное на конечного пользователя и предназначенное для решения пользовательских задач. Прикладное ПО состоит из:

- отдельных прикладных программ и пакетов прикладных программ, предназначенных для решения различных задач пользователей;
- автоматизированных систем, созданных на основе этих пакетов.

Пакет прикладных программ - комплект программ, предназначенных для решения задач из определенной проблемной области. Обычно применение пакета прикладных программ предполагает наличие специальной документации: лицензионного свидетельства, паспорта, инструкции пользователя и т.п.

КЛАССИФИКАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Любая классификация подразумевает выбор некоторого группировочного признака (или нескольких), на основании которого и производится отнесение объектов

к тому или иному классу. Так, при классификации программного обеспечения по способу распространения можно выделить следующие категории список не полный):

- Commercial Software - коммерческое (с ограниченными лицензией возможностями на использование), разрабатываемое для получения прибыли.
- Freeware - свободное ПО, распространяемое без ограничений на использование, модификацию и распространение.
- Shareware - условно-бесплатное ПО, с частичными ограничениями при работе в ознакомительном режиме (например, определенное количество запусков программы).
- Abandonware - «заброшенное» ПО, поддержка которого непосредственным разработчиком прекращена, но продолжается третьими лицами (например, партнерами или энтузиастами).
- Adware - ПО, в код которого включены рекламные материалы. Такое ПО распространяется бесплатно, но для отключения рекламных блоков необходима оплата.
- Careware - «благотворительное» ПО, оплату за которое разработчик (или распространитель) просит переводить на благотворительные нужды.

При классификации программного обеспечения по назначению в качестве критерия используют уровень представления ИС, на который ориентирована та или иная программа. Соответственно выделяют следующие классы ПО:

1. Системное ПО - решает задачи общего управления и поддержания

   работоспособности системы в целом. К этому классу относят операционные

   системы, менеджеры загрузки, драйверы устройств, программные кодеки, утилиты и программные средства защиты информации.

2. Инструментальное ПО включает средства разработки (трансляторы, отладчики, интегрированные среды, различные SDK и т.п.) и системы управления базами данных (СУБД).

3. Прикладное ПО - предназначено для решения прикладных задач конечными пользователями.

Прикладное ПО - самый обширный класс программ, в рамках которого возможна дальнейшая классификация, например по предметным областям. В этом случае группировочным признаком является класс задач, решаемых программой. Приведем несколько примеров:

- Офисные приложения - предназначены для автоматизации офисной деятельности (текстовые редакторы и процессоры, электронные таблицы, редакторы презентаций и т.п.)
- Корпоративные информационные системы - бухгалтерские программы, системы корпоративного управления, системы управления проектами (Project Management), инструменты автоматизации документооборота (EDM-системы) и управления архивами документов (DWM-системы)
- Системы проектирования и производства - системы автоматизированного проектирования (САПР, CAD/CAM-системы), системы управления технологическими (SCADA) и производственными (MES) процессами
- Научное ПО - системы математического и статистического расчета, анализа и моделирования
- Геоинформационные системы (ГИС)
- Системы поддержки принятия решений (СППР)
- Клиенты доступа к сетевым сервисам (электронная почта, веб-браузеры, передача сообщений, чат-каналы, клиенты файлообменных сетей и т.п.)
- Мультимедийное ПО - компьютерные игры, средства просмотра и редактирования аудио- и видеоинформации, графические редакторы и вьюеры, анимационные редакторы и т.п.

С точки зрения конечного пользователя такая классификация оправданна и наглядна, для разработчика же более значимым фактором является структура прикладной программы, в общем случае состоящей из нескольких компонентов. Назначение этих компонентов, связи между ними и способность к взаимодействию

определяют интеграцию прикладного ПО. Чем теснее связаны программные компоненты, тем выше степень интеграции.

В зависимости от степени интеграции многочисленные прикладные программные средства можно классифицировать следующим образом[1]:

1. отдельные прикладные программы;
2. библиотеки прикладных программ;
3. пакеты прикладных программ;
4. интегрированные программные системы.

Отдельная прикладная программа пишется, как правило, на некотором высокоуровневом языке программирования (Pascal, Basic и т.п.) и предназначается для решения конкретной прикладной задачи. Такая программа может быть реализована в виде набора модулей, каждый из которых выполняет некоторую самостоятельную функцию (например модуль пользовательского интерфейса, модуль обработки ошибок, модуль печати и т.п.). При этом доступ к функциям модулей из внешних программ невозможен.

Библиотека представляет собой набор отдельных программ, каждая из которых решает некоторую прикладную задачу или выполняет определенные вспомогательные функции (управление памятью, обмен с внешними устройствами и т.п.). Библиотеки программ зарекомендовали себя эффективным средством решения вычислительных задач. Они интенсивно используются при решении научных и инженерных задач с помощью ЭВМ. Условно их можно разделить на библиотеки общего назначения и специализированные библиотеки.

Пакет прикладных программ (ППП) - это комплекс взаимосвязанных программ, ориентированный на решение определенного класса задач. Формально такое определение не исключает из числа пакетов и библиотеки программ, однако у ППП, как отдельной категории, есть ряд особенностей, среди которых: ориентация на решение классов задач, унифицированный интерфейс, наличие языковых средств.

Интегрированная программная система - это комплекс программ, элементами которого являются различные пакеты и библиотеки программ. Примером служат системы автоматизированного проектирования, имеющие в своем составе несколько

ППП различного назначения. Часто в подобной системе решаются задачи, относящиеся к различным классам или даже к различным предметным областям.

ПОНЯТИЕ ПАКЕТА ПРИКЛАДНЫХ ПРОГРАММ

Итак, пакет прикладных программ (ППП) - это комплекс взаимосвязанных программ для решения определенного класса задач из конкретной предметной области. На текущем этапе развития информационных технологий именно ППП являются наиболее востребованным видом прикладного ПО. Это связано с упомянутыми ранее особенностями ППП. Рассмотрим их подробней:

- Ориентация на решение класса задач. Одной из главных особенностей является ориентация ППП не на отдельную задачу, а на некоторый класс задач, в том числе и специфичных, из определенной предметной области. Так например, офисные пакеты ориентированы на офисную деятельность, одна из задач которой - подготовка документов (в общем случае включающих не только текстовую информацию, но и таблицы, диаграммы, изображения). Следовательно, офисный пакет должен реализовывать функции обработки текста, представлять средства обработки табличной информации, средства построения диаграмм разного вида и первичные средства редактирования растровой и векторной графики.

- Наличие языковых средств. Другой особенностью ППП является наличие в его составе специализированных языковых средств, позволяющих расширить число задач, решаемых пакетом или адаптировать пакет под конкретные нужды. Пакет может представлять поддержку нескольких входных языков, поддерживающих различные парадигмы. Поддерживаемые языки могут быть использованы для формализации исходной задачи, описания алгоритма решения и начальных данных, организации доступа к внешним источникам данных, разработки программных модулей, описания модели предметной области, управления процессом решения в диалоговом режиме и других целей. Примерами входных языков ППП являются VBA в пакете MS Office, AutoLISP/VisualLISP в Autodesk AutoCAD, StarBasic в OpenOffice.org

- Единообразие работы с компонентами пакета. Еще одна особенность ППП состоит в наличии специальных системных средств, обеспечивавших

унифицированную работу с компонентами. К их числу относятся специализированные банки данных, средства информационного обеспечения, средства взаимодействия пакета с операционной системой, типовой пользовательский интерфейс и т.п.

СТРУКТУРА И ОСНОВНЫЕ КОМПОНЕНТЫ ППП

Несмотря на разнообразие конкретных пакетных разработок, их обобщенную внутреннюю структуру можно представить в виде трех взаимосвязанных элементов1 (рис. 2):

1. входной язык (макроязык, язык управления) - представляет средство общения пользователя с пакетом;
2. предметное обеспечение (функциональное наполнение) - реализует особенности конкретной предметной области;
3. системное обеспечение (системное наполнение) – представляет низкоуровневые средства, например, доступ к функциям операционной системы.

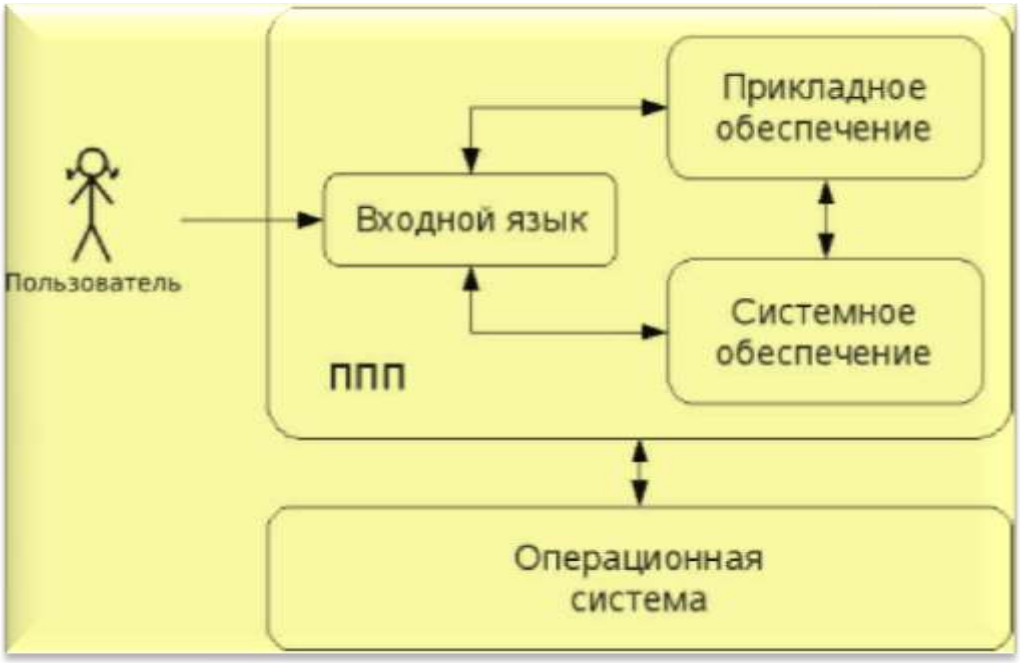


РИСУНОК 2. **СТРУКТУРА ППП.**

Входной язык - основной инструмент при работе пользователя с пакетом прикладных программ. В качестве входного языка могут использоваться как универсальные (Pascal, Basic и т.п.), так и специализированные, проблемно-

ориентированные языки программирования (Cobol - для бизнес-приложений, Lisp - списочные структуры данных, Fortran и MathLAB - математические задачи и т.п.).

Развитый пакет может обладать несколькими входными языками, предназначенными для выполнения различных функций в рамках решаемого класса задач. Так, например в пакете OpenOffice.org поддерживаются языки StarBasic, Python, JavaScript и Java. StarBasic является основным входным языком, предназначенным для

автоматизации работы с пакетом, для этого языка имеется интегрированная среда разработки и встроенный отладчик. Скрипты на языках Python и JavaScript загружаются и исполняются из внешних файлов. На Java (через SDK и функции API OpenOffice) можно создавать модули расширения и полнофункциональные приложения-компоненты.

Входные языки отражают объем и качество предоставляемых пакетом возможностей, а также удобство их использования. Таким образом, именно входной язык является основным показателем возможностей ППП. Однако стоит отметить, что в современных пакетах обращение пользователя к языковым средствам обычно происходит косвенно, через графический интерфейс.

Предметное обеспечение отражает особенности решаемого класса задач из конкретной предметной области и включает:

- программные модули, реализующие алгоритмы (или их отдельные фрагменты) прикладных задач;
- средства сборки программ из отдельных модулей.

Наиболее распространено в настоящее время оформление программных модулей в виде библиотек, подключаемых статически или динамически. В зависимости от использованного разработчиками подхода к проектированию и реализации ППП такие библиотеки содержат встроенные классы и описания их интерфейсов (при использовании объектно-ориентированного программирования). При использовании парадигмы структурного программирования в библиотечных модулях содержатся процедуры и функции, предназначенные для решения некоторых самостоятельных задач. В обоих случаях библиотеки связаны с другими модулями пакета лишь входной и выходной информацией.

Системное обеспечение представляет собой совокупность низкоуровневых средств (программы, файлы, таблицы и т.д.), обеспечивающих определенную дисциплину работы пользователя при решении прикладных задач и формирующих окружение пакета. К системному обеспечению ППП относят следующие компоненты:

- монитор - программа, управляющая взаимодействием всех компонентов ППП;
- транслятор(ы) с входных языков - для ППП характерно использование интерпретируемых языков;
- средства доступа к данным - драйверы баз данных и/или компоненты, представляющие доступ через унифицированные интерфейсы (ODBC, JDBC, ADO, BDE и т.п.);
- информационно-справочный модуль - предоставляет функции поддержки, среди которых информационные сообщения, встроенная справочная системы и т.п.
различные служебные программы, выполняющие низкоуровневые операции (автосохранение, синхронизация совместно используемых файлов и т.д.)

Приведенная логическая структура ППП достаточна условна и в конкретном ППП может отсутствовать четкое разделение программ на предметное и системное обеспечение. Например, программа планирования вычислений, относящаяся к прикладному обеспечению, может одновременно выполнять и ряд служебных функций (информационное обеспечение, связь с операционной системой и т.п.).

Кроме того, одни и те же программы в одном пакете могут относиться к предметному обеспечению, а в другом - к системному. Так, программы построения диаграмм в рамках специализированного пакета машинной графики естественно отнести к предметному обеспечению. Однако те же программы следует считать вспомогательными и относящимися к системному обеспечению, например, в пакете решения вычислительных задач.

ЭВОЛЮЦИЯ ППП. ПРИМЕРЫ СОВРЕМЕННЫХ ППП

ЭТАПЫ РАЗВИТИЯ ППП

Первые ППП представляли собой простые тематические подборки программ для решения отдельных задач в той или иной прикладной области, обращение к ним

выполнялось с помощью средств оболочки ОС или из других программ. Современный пакет является сложной программной системой, включающей специализированные системные и языковые средства. В относительно короткой истории развития вычислительных ППП можно выделить *4 основных поколения* (класса) пакетов. Каждый из этих: классов характеризуется определенными особенностями входящих состав ППП компонентов - входных языков, предметного и системного обеспечения.

ПЕРВОЕ ПОКОЛЕНИЕ

В качестве входных языков ППП первого поколения использовались универсальные языки программирования (Фортран, Алгол-60 и т.п.) или языки управления заданиями соответствующих операционных систем. Проблемная ориентация входных языков достигалась за счет соответствующей мнемоники в идентификаторах. Составление заданий на таком языке практически не отличалось от написания программ на алгоритмическом языке.

Предметное обеспечение первых ППП, как правило, было организовано в форме библиотек программ, т.е. в виде наборов (пакетов) независимых программ на некотором базовом языке программирования (отсюда впервые возник и сам термин «пакет»). Такие ППП иногда называют *пакетами библиотечного типа,* или *пакетами простой структуры*.

В качестве системного обеспечения пакетов первого поколения обычно использовались штатные компоненты программного обеспечения ЭВМ: компиляторы с алгоритмических языков, редакторы текстов, средства организации библиотек программ, архивные системы и т.д. Эти пакеты не требовали сколько-нибудь развитой системной поддержки, и для их функционирования вполне хватало указанных системных средств общего назначения. В большинстве случаев разработчиками таких пакетов были прикладные программисты, которые пытались приспособить универсальные языки программирования к своим нуждам.

ВТОРОЕ ПОКОЛЕНИЕ

Разработка ППП второго поколения осуществлялась уже с участием системных программистов. Это привело к появлению специализированных входных языков на базе универсальных языков программирования. Проблемная ориентация таких языков достигалась не только за счет использования определенной мнемоники, но также

применением соответствующих языковых конструкций, которые упрощали формулировку задачи и делали ее более наглядной. Транслятор с такого языка представлял собой препроцессор (чаще всего макропроцессор) к транслятору соответствующего алгоритмического языка.

В качестве модулей в пакетах этого класса стали использоваться не только программные единицы (т.е. законченные программы на том или ином языке программирования), но и такие объекты, как последовательность операторов языка программирования, совокупность данных, схема счета и др.

Существенные изменения претерпели также принципы организации системного обеспечения ППП. В достаточно развитых пакетах второго поколения уже можно выделить элементы системного обеспечения, характерные для современных пакетов: монитор, трансляторы с входных языков, специализированные банки данных, средства описания модели предметной области и планирования вычислений и др.

ТРЕТЬЕ ПОКОЛЕНИЕ

Третий этап развития ППП характеризуется появлением самостоятельных входных языков, ориентированных на пользователей-непрограммистов. Особое внимание в таких ППП уделяется системным компонентам обеспечивающим простоту и удобство. Это достигается главным образом за счет специализации входных языков и включения в состав пакета средств автоматизированного планирования вычислений.

ЧЕТВЕРТОЕ ПОКОЛЕНИЕ

Четвертый этап характеризуется созданием ППП, эксплуатируемых в интерактивном режиме работы. Основным преимуществом диалогового взаимодействия с ЭВМ является возможность активной обратной связи с пользователем в процессе постановки задачи, ее решения и анализа полученных результатов. Появление и интенсивное развитие различных форм диалогового общения обусловлено прежде всего прогрессом в области технических средств (графическая подсистема ЭВМ и средства мультимедиа, сетевые средства). Развитие аппаратного обеспечения повлекло за собой создание разнообразных программных средств поддержки диалогового режима работы (диалоговые операционные системы, диалоговые пакеты программ различного назначения и т. д.).

Прикладная система состоит из *диалогового монитора* - набора универсальных программ, обеспечивающих ведение диалога и обмен данными, и базы знаний об области. Информация о структуре, целях и форма диалога задает сценарий, в соответствии с который монитор управляет ходом диалога. Носителями процедурных знаний о предметной области являются прикладные модули, реализующие функции собственной системы. Таким образом, создание прикладной системы сводится к настройке диалогового монитора на конкретный диалог, путем заполнения базы знаний. При этом программировать в традиционном смысле этого слова приходится лишь прикладные модули, знания о диалоге вводятся в систему с помощью набора соответствующих средств - редактора сценариев. Логично требовать, чтобы редактор сценариев также представлял собой диалоговую программу, отвечавшую рассмотренным выше требованиям. Благодаря готовому универсальному монитору программист может сосредоточиться на решении чисто прикладных задач, выделение же знаний о диалоге в сценарий обеспечивает в значительной степени необходимая гибкость программного продукта.

Большое внимание в настоящее время уделяется проблеме создания *«интеллектуальных ППП»*. Такой пакет позволяет конечному пользователю лишь сформулировать свою задачу в содержательных терминах, не указывая алгоритма ее решения. Синтез решения и сборка целевой программы производятся автоматически. При этом детали вычислений скрыты от пользователя, и компьютер становится интеллектуальным партнером человека, способным понимать его задачи. Предметное обеспечение подобного ППП представляет собой некоторую базу знаний, содержащую как процедурные, так и описательные знания. Такой способ решения иногда называют концептуальным программированием, характерными особенностями которого является программирование в терминах предметной области использование ЭВМ уже на этапе постановки задач, автоматический синтез программ решения задачи, накопление знаний о решаемых задачах в базе знаний.

КРАТКИЙ ОБЗОР НЕКОТОРЫХ ППП

Для иллюстрации ранее рассмотренных материалов приведем несколько примеров современных пакетов прикладных программ из различных предметных областей. Учитывая, что постоянно появляются новые версии программных продуктов, здесь будут

рассматриваться не возможности конкретных версий, а лишь основные структурные компоненты, входящие в состав того или иного пакета.

AUTODESK AUTOCAD

Основное назначение ППП AutoCAD - создание чертежей и проектной документации. Современные версии этого пакета представляют существенно большие возможности, среди которых построение трехмерных твердотельных моделей, инженерно-технические расчеты и многое другое.

Первые версии системы AutoCAD, разрабатываемой американской фирмой Autodesk, появились еще в начале 80-х годов двадцатого века, и сразу же привлекли к себе внимание своим оригинальным оформлением и удобством для пользователя. Постоянное развитие системы, учет замечаний, интеграция с новыми продуктами других ведущих фирм сделали AutoCAD мировым лидером на рынке программного обеспечения для автоматизированного проектирования.

ЯЗЫКОВЫЕ СРЕДСТВА

В основе языковых средств ППП AutoCAD - технология Visual LISP, базирующаяся на языке AutoLISP (подмножество языка LISP) и используемая для создания приложений и управления в AutoCAD. Visual LISP представляет полное окружение, включающее:

- Интегрированную среду разработки, облегчающую написание, отладку и сопровождение приложений на AutoLISP
- Доступ к объектам ActiveX и обработчикам событий
- Защиту исходного кода
- Доступ к файловым функциям операционной системы
- Расширенные функции языка LISP для обработки списочных структур данных. Для разработчиков совместимых приложений в AutoCAD включена поддержка

ObjectARX. Это программное окружение представляет объектно-ориентированный интерфейс для приложений на языках C++, C# и VB.NET и обеспечивает прямой доступ к структурам БД, графической подсистеме и встроенным командам пакета.

Кроме того, в AutoCAD имеется поддержка языка Visual Basic for Applications (VBA), что позволяет использовать этот пакет совместно с другими приложениями, в частности, из семейства Microsoft Office.

ПРЕДМЕТНОЕ ОБЕСПЕЧЕНИЕ

К предметному обеспечению пакета в первую очередь относятся функции построения примитивов - различных элементов чертежа. Простые примитивы это такие объекты как точка, отрезок, круг (окружность) и т.д. К сложным примитивам относятся: полилиния, мультилиния, мультитекст (многострочный текст), размер, выноска, допуск, штриховка, вхождение блока или внешней ссылки, атрибут, растровое изображение. Кроме того, есть пространственные примитивы, видовые экраны и пр.. Операции построения *большей части* примитивов могут быть выполнены через пользовательский интерфейс, *все* - через команды языка.

Высокоуровневые средства представлены расширениями и приложениями AutoCAD для конкретных предметных областей. Например в машиностроении используется Autodesk Mechanical Desktop - предназначенный для сложного трехмерного моделирования, в том числе валов и пружин. Для проектирования деталей из листовых материалов предназначена система Copra Sheet Metal Bender Desktop (разработчик - Data-M Software GmbH). Моделирование динамики работы механизмов может выполняться в системе Dynamic Designer (Mechanical Dynamics). В числе известных архитектурных и строительных приложений можно отметить системы АРКО (АПИО-Центр), СПДС GraphiCS (Consistent Software), ArchiCAD. Для проектирования промышленных объектов может использоваться система PLANT-4D (CEA Technology). Это лишь некоторые из областей использования AutoCAD.

СИСТЕМНОЕ ОБЕСПЕЧЕНИЕ

Среди системного обеспечения следует отметить основной формат файлов AutoCAD .dwg, который стал стандартом «де факто» для прочих САПР.

К системному же обеспечению обеспечению относятся типовые и специализированные библиотеки деталей и шаблонов, использование которых позволяет существенно ускорить процесс проектирования. Здесь же упомянем требования отраслевых и государственных стандартов, которым должны соответствовать чертежи и спецификации.

Конфигурация и настройки различных режимов AutoCAD устанавливаются через т.н. системные переменные. Изменяя их значения можно задавать пути к файлам, точность вычислений, формат вывода и многое другое.

ADOBE FLASH

Adobe (ранее Macromedia) Flash - это технология и инструментарий разработки интерактивного содержания с большими функциональными возможностями для цифровых, веб- и мобильных платформ. Она позволяет создавать компактные, масштабируемые анимированные приложения (ролики), которые можно использовать как отдельно, так и встраивая в различное окружение (в частности, в веб-страницы). Эти возможности обеспечиваются следующими компонентами технологии: языком Action Script, векторным форматом .swf и видеоформатом .flv, всевозможными flash-плейерами для просмотра и редакторами для создания.

Рассмотрим интегрированную среду Adobe Flash как основное средство создания flash-приложений. При этом отметим, что языковые и системные средства относятся не только к этому пакету, а к технологии в целом.

ЯЗЫК ACTIONSCRIPT

ActionScript - объектно-ориентированный язык программирования, который добавляет интерактивность, обработку данных и многое другое в содержимое Flash-приложений. Синтаксис ActionScript основан на спецификации ECMAScript (сюда же относятся языки JavaScript и JScript). Библиотека классов ActionScript, написанная на C++, представляет доступ к графическим примитивам, фильтрам, принтерам, геометрическим функциям и пр..

ActionScript как язык появился с выходом 5 версии Adobe (тогда еще Macromedia) Flash, которая стала первой программируемой на ActionScript средой. Первый релиз языка назывался ActionScript 1.0. Flash 6 (MX). В 2004 году Macromedia представила новую версию ActionScript 2.0 вместе с выходом Flash 7 (MX 2004), в которой было введено строгое определение типов, основанное на классах программирование: наследование, интерфейсы и т. д. Также Macromedia была выпущена модификация языка Flash Lite для программирования под мобильные телефоны. ActionScript 2.0 является не более чем надстройкой над ActionScript 1.0, то есть на этапе компиляции ActionScript 2.0 осуществляет некую проверку и превращает классы, методы ActionScript 2.0 в прежние прототипы и функции ActionScript 1.0.

В 2005 году вышел ActionScript 3.0 в среде программирования Adobe Flex, а позже в Adobe Flash 9.

ActionScript 3.0 (текущая версия на момент подготовки этого материала) представляет, по сравнению с ActionScript 2.0 качественное изменение, он использует

новую виртуальную машину AVM 2.0 и дает взамен прежнего формального синтаксиса классов настоящее классовое (class-based) Объектно-ориентированное программирование. ActionScript 3.0 существенно производительней предыдущих версий и по скорости приблизился к таким языкам программирования, как Java и C++.

С помощью ActionScript можно создавать интерактивные мультимедиа-приложения, игры, веб-сайты и многое другое.

СИСТЕМНОЕ ОБЕСПЕЧЕНИЕ

ActionScript исполняется виртуальной машиной (ActionScript Virtual Machine), которая является составной частью Flash Player. ActionScript компилируется в байткод, который включается в SWF-файл.

SWF-файлы исполняются Flash Player-ом. Flash Player существует в виде плагина к веб-браузеру, а также как самостоятельное исполняемое приложение. Во втором случае возможно создание исполняемых exe-файлов, когда swf-файл включается во Flash Player.

Для создания и просмотра видеофайлов в формате .flv используются программные кодеки, поддерживающие этот формат.

ПРИКЛАДНОЕ ОБЕСПЕЧЕНИЕ

К прикладному обеспечению в рамках технологии Flash относятся средства создания роликов в форматах .swf, .flv и .exe. Основным инструментом является среда среда Adode Flash, включающая различные средства для создания и редактирования мультимедийного содержания, в т.ч. видео- и аудиофайлов, интегрированную среду разработки на ActionScript и множество дополнительных функций упрощения процесса создания роликов.

ПАКЕТ MATLAB

MatLab (сокращение от англ. «Matrix Laboratory») - пакет прикладных программ для решения задач технических вычислений, и язык программирования,

используемый в этом пакете. По данным фирмы-разработчика, более 1000000 инженерных и научных работников используют этот пакет, который работает на большинстве современных операционных систем, включая GNU/Linux, Mac OS, Solaris и Microsoft Windows.

ЯЗЫК MATLAB

MATLAB как язык программирования был разработан Кливом Моулером (англ. Cleve Moler) в конце 1970-х годов. Целью разработки служила задача использования программных математических библиотек Linpack и EISPACK без необходимости изучения языка Фортран. Акцент был сделан на матричные алгоритмы.

Программы, написанные на MATLAB, бывают двух типов - функции и скрипты. Функции имеют входные и выходные аргументы, а также собственное рабочее пространство для хранения промежуточных результатов вычислений и переменных. Скрипты же используют общее рабочее пространство. Как скрипты, так и функции не компилируются в машинный код, а сохраняются в виде текстовых файлов. Существует также возможность сохранять так называемые pre-parsed программы - функции и скрипты, приведенные в вид, удобный для машинного исполнения и, как следствие, более быстрые по сравнению с обычными.

СИСТЕМНОЕ ОБЕСПЕЧЕНИЕ

Язык MATLAB является высокоуровневым интерпретируемым языком программирования, включающим основанные на матрицах структуры данных, широкий спектр функций, интегрированную среду разработки, объектно-ориентированные возможности и интерфейсы к программам, написанным на других языках программирования. Имеются интерфейсы для получения доступа к внешним данным, клиентам и серверам, общающимся через технологии Component Object Model (COM) или Dynamic Data Exchange (DDE), а также периферийным устройствам, которые взаимодействуют напрямую с MATLAB. Многие из этих возможностей известны под названием MATLAB API.

Встроенная среда разработки позволяет создавать графические интерфейсы пользователя с различными элементами управления, такими как кнопки, поля ввода и другими. С помощью компонента MATLAB Compiler эти графические интерфейсы могут быть преобразованы в самостоятельные приложения.

Для MATLAB имеется возможность создавать специальные наборы инструментов (англ. toolbox), расширяющие его функциональность. Наборы инструментов представляют собой коллекции функций, написанных на языке MATLAB для решения определенного класса задач.

ПРИКЛАДНОЕ ОБЕСПЕЧЕНИЕ

MATLAB предоставляет удобные средства для разработки алгоритмов, включая высокоуровневые с использованием концепций объектно-ориентированного программирования. В нем имеются все необходимые средства интегрированной среды разработки, включая отладчик и профайлер.

MATLAB предоставляет пользователю большое количество (несколько сотен) функций для анализа данных, покрывающие практически все области математики, в частности:

- Матрицы и линейная алгебра — алгебра матриц, линейные уравнения, собственные значения и вектора, сингулярности, факторизация матриц и другие.
- Многочлены и интерполяция — корни многочленов, операции над многочленами и их дифференцирование, интерполяция и экстраполяция кривых и другие.
- Математическая статистика и анализ данных — статистические функции, статистическая регрессия, цифровая фильтрация, быстрое преобразование Фурье и другие.
- Обработка данных — набор специальных функций, включая построение графиков, оптимизацию, поиск нулей, численное интегрирование (в квадратурах) и другие.
- Дифференциальные уравнения — решение дифференциальных и дифференциально-алгебраических уравнений, дифференциальных уравнений с запаздыванием, уравнений с ограничениями, уравнений в частных производных и другие.
- Разреженные матрицы — специальный класс данных пакета MATLAB, использующийся в специализированных приложениях.

В составе пакета имеется большое количество функций для построения графиков, в том числе трехмерных, визуального анализа данных и создания анимированных

роликов, функции для создания алгоритмов для микроконтроллеров и других приложений.

ТЕМА №2. ФУНКЦИОНАЛЬНОЕ И СИСТЕМНОЕ НАПОЛНЕНИЕ ПАКЕТА MS OFFICE. ВИДЫ ИНТЕРФЕЙСОВ: ВНЕШНИЕ, ВНУТРЕННИЕ, СПРАВОЧНЫЕ, УПРАВЛЕНИЯ, ВВОДА-ВЫВОДА, ИНФОРМАЦИОННЫЕ.

СТРУКТУРА И СОСТАВ MS OFFICE. ОСНОВНЫЕ ПРИЛОЖЕНИЯ

СТРУКТУРА MS OFFICE И НАЗНАЧЕНИЕ КОМПОНЕНТОВ

ППП Microsoft Office - это совокупность программных средств автоматизации офисной деятельности. В состав пакета входит множество приложений, каждое из которых предназначено для выполнения определенных функций и может быть использовано автономно и независимо от остальных. Весь набор офисных приложений можно разделить на *основные* и *дополнительные*.

ОСНОВНЫЕ КОМПОНЕНТЫ MICROSOFT OFFICE

Список и назначение основных компонентов, входящих в состав Microsoft Office приведен в таб. 1.

ТАБЛИЦА 1. **ОСНОВНЫЕ КОМПОНЕНТЫ MICROSOFT OFFICE**

Название приложения	Функциональное назначение приложения
Microsoft Word	Текстовый процессор
Microsoft Excel	Табличный процессор
Microsoft PowerPoint	Система подготовки презентаций
Outlook	Система управления персональной информацией
Microsoft Access	Система управления базами данных
Microsoft Binder	Система управления подшивками
Microsoft FrontPage	Система управления Web-узлами
Microsoft PhotoDraw	Графический редактор
Microsoft Publisher	Настольная издательская система
Microsoft Project	Система управления проектами
Microsoft Team Manager	Система управления персоналом

ДОПОЛНИТЕЛЬНЫЕ КОМПОНЕНТЫ MS OFFICE

Кроме основных компонентов, в семейство Microsoft Office входит большое количество вспомогательных приложений, которые устанавливаются (или не устанавливаются) вместе с основными. Ими можно воспользоваться из основных приложений или вызвать независимо. В таб. 2 перечислены некоторые из вспомогательных приложений.

ТАБЛИЦА 2. **НЕКОТОРЫЕ ВСПОМОГАТЕЛЬНЫЕ ПРИЛОЖЕНИЯ MICROSOFT OFFICE**

Название приложения	**Функциональное назначение приложения**
Microsoft Query	Интерпретатор запросов к внешним базам данных
Microsoft Organization Chart	Программа рисования блок-схем
Microsoft WordArt	Программа создания фигурных текстов
Microsoft Equation	Редактор математических формул
Microsoft Map	Программа отображения данных на географических картах
Microsoft Graph	Программа построения диаграмм
Microsoft Photo Editor	Графический редактор
Microsoft Draw	Средство рисования
Microsoft Find Fast	Служба индексации документов
Microsoft Extended Finder	Средство поиска документов в папках файловой системы и электронной почты
Microsoft Script Editor	Редактор сценариев
Microsoft ClipArt	Коллекция картинок и клипов
Панель Microsoft Office	Средство быстрого доступа к приложениям Office

Кроме основных и вспомогательных приложений, могут быть установлены и использованы различные расширения (надстройки). Их можно условно разделить на три группы:

1. *Самостоятельные приложения,* разработанные фирмой Microsoft, которые являются компонентами семейства Microsoft Office, но формально не входят в состав пакета. Примерами являются приложения Microsoft Project и Microsoft Team Manager.

2. *Надстройки* над компонентами Microsoft Office, разработанные фирмой Microsoft и представляющие собой дополнительные функции. Как правило, надстройки оформляются не в виде готовых к выполнению программ, а в виде

документов специального типа: шаблонов, рабочих книг, библиотек динамической компоновки (DLL) и т.п.

3. *Приложения третьих фирм,* разработанные для пользователей Microsoft Office. В этот класс попадают как продукты сторонних фирм, так и собственные разработки пользователей. Сюда можно отнести средства распознавания текстов (OCR), автоматического перевода текста, средства управления большими массивами документов (перечисленные задачи не реализованы или слабо развиты в самом пакете MS Office).

Приведенный перечень основных компонентов носит условный характер, поскольку состав пакета зависит от следующих факторов:

1. *Устанавливаемый комплект (или редакция) пакета*. Пакет выпускается в нескольких редакциях, и состав приложений в разных редакциях различен.

2. *Источник установки. Установка* может быть выполнена с компакт-диска или с сетевого сервера. Наборы файлов, которые устанавливаются на компьютер, существенно различаются.

3. *Операционная система*. Microsoft Office может работать под управлением различных ОС: MS Windows и Mac OS. Эти операционные системы могут иметь разные версии и модификации, что также влияет на состав устанавливаемых компонентов.

4. *Наличие на компьютере в момент установки предшествующих версий*. Некоторые компоненты старых версий автоматически включаются в состав обновляемой версии Microsoft Office (если они уже установлены на компьютере).

5. *Параметры, заданные при установке*. В случае так называемой выборочной (т.е. по выбору пользователя) установки, можно указать несколько десятков независимых параметров, влияющих на состав пакета.

Несмотря на большое число различных приложений в составе пакета, все они в совокупности образуют единое целое. Для каждого из приложений MS Office характерно наличие следующих отличительных признаков:

1. совместимость по данным;

2. унифицированный интерфейс;

3. единые средства программирования.

ДОКУМЕНТЫ MICROSOFT OFFICE

Единица данных самого верхнего уровня структуризации в Microsoft Office называется **документом**.

Документы классифицируются по типам в зависимости от того, какого сорта информация в них хранится. Как правило, документы разных типов обрабатываются разными приложениями Microsoft Office. Основные типы документов, с которыми работают программы Microsoft Office, перечислены в таб. 3.

ТАБЛИЦА 3. **ОСНОВНЫЕ ТИПЫ ДОКУМЕНТОВ MICROSOFT OFFICE**

Название	Расширение	Приложение	Краткое описание
Документ	.doc	Word	Основной тип документов Word. Содержит форматированный текст, т.е. текст с дополнительной информацией о шрифтах, отступах, интервалах и т.п., а также рисунки, таблицы и другие элементы
Рабочая книга	.xls	Excel	Основной тип документов Excel. Содержит данные различных типов: формулы, диаграммы и макросы
База данных	.mdb	Access	Основной тип документов Access. Содержит как собственно базу данных, то есть совокупность таблиц, так и соответствующие запросы, макросы, модули, формы и отчеты
Презентация	.ppt	PowerPoint	Основной тип документов PowerPoint. Содержит презентацию, состоящую из набора слайдов, заметок выступающего, раздаточных материалов и другой информации
Публикация	.pub	Publisher	Основной тип документов Publisher. Как и Word, содержит форматированный текст, рисунки, таблицы и т.п.
План проекта	.mpp	Project	Основной тип документов Project. Содержит календарный план проекта, описание задач, ресурсов и их взаимосвязи

Исходя из вышесказанного, можно сделать следующий вывод: входящие в состав пакета MS Office приложения способны тесно взаимодействовать при решении прикладных задач; они создают единую информационную среду и позволяют обмениваться объектами. Документы Microsoft Office являются частными примерами объектов. Поэтому Microsoft Office является *документо-ориентированным пакетом* (средой).

ПРОГРАММНАЯ СРЕДА

Основным средством разработки приложений в MS Office является комплексное решение на основе языка Visual Basic, а именно - Visual Basic for Application (VBA). Эта технология включает макрорекордер, интерпретатор Visual Basic, интегрированную среду разработки с встроенным отладчиком, библиотеки времени выполнения (runtime library) и библиотеки типов, представляющие объекты пакета. Эти средства позволяют расширять функциональность пакета и адаптировать его к решению специализированных задач.

ИНТЕРФЕЙС MS OFFICE

Приложения Microsoft Office имеют унифицированный интерфейс, суть которого заключается в следующем: сходные функции имеют одинаковое обозначение (название команды или значок на кнопке), а несходные функции имеют различные обозначения.

В большей степени унификация коснулась интерфейсов таких приложений, как Microsoft Word, Microsoft Excel и Microsoft PowerPoint.

Одним из достоинств пакета Microsoft Office является последовательное использование графического интерфейса пользователя (Graphical User Interface, GUI), представляемого операционной системой и различных элементов управления. Как правило, отдельные элементы группируются в более крупные конструкции, такие как окна, панели инструментов, меню. Рассмотрим характеристику каждой из этих групп.

ОКОННЫЙ ИНТЕРФЕЙС

Оконный интерфейс - такой способ организации пользовательского интерфейса программы, когда каждая интегральная часть располагается в *окне* - собственном субэкранном пространстве, находящемся в произвольном месте «над» основным экраном. Несколько окон одновременно располагающихся на экране могут перекрываться, находясь «выше» или «ниже» друг относительно друг

В MS Office использует окна четырех типов:

- окно приложения;
- окно документа;
- диалоговое окно;
- форма.

ПАНЕЛИ ИНСТРУМЕНТОВ

Панели инструментов - это элементы пользовательского интерфейса, на которых могут располагаются такие элементы управления, как кнопки быстрого вызова и раскрывающиеся списки. Панели инструментов разных приложений могут содержать кнопки, сходные по функциям и внешнему виду, что упрощает освоение интерфейса Microsoft Office.

Панели инструментов могут быть:

- пристыкованными вдоль границы окна приложения;
- плавающими, т.е. находится в любой части окна приложения;
- представленными в отдельных окнах; в этом случае форму и размеры панели инструментов можно менять произвольно.

МЕНЮ

Меню представляет доступ к иерархическим спискам доступных команд. Результатом выбора команды из меню может быть:

- непосредственное выполнение некоторого действия;
- раскрытие еще одного меню;
- раскрытие диалогового окна или формы.

Меню интерфейса Microsoft Office, кроме строки меню любого приложения, можно разделить (по способу перехода к ним) на раскрывающиеся и контекстные (или всплывающие).

ЭЛЕМЕНТЫ УПРАВЛЕНИЯ

Элементы управления - это объекты оконного интерфейса, реализующие типовые операции с интерфейсом: щелчок мышью, выбор из списка, выбор вариантов,

прокрутка и т.п. К элементам управления относятся следующие: кнопки, текстовые поля (или поля ввода), флажки, переключатели, списки и раскрывающиеся списки, полосы прокрутки, палитры, счетчики и прочие, специфичные для некоторых приложений или условий.

ТЕМА №3. ВХОДНЫЕ ЯЗЫКИ И ИСПОЛЬЗОВАНИЕ ИХ ДЛЯ ПРОГРАММИРОВАНИЯ В СРЕДЕ ВЫБРАННОГО ПАКЕТА. ИНТЕГРАЦИЯ ВЫБРАННОГО ПАКЕТА С ДРУГИМИ ПРОГРАММАМИ

ВВЕДЕНИЕ В ОФИСНОЕ ПРОГРАММИРОВАНИЕ

Офисное программирование - это процесс разработки приложений, предназначенных для автоматизации офисной деятельности с использованием специализированных пакетов (MS Office, OpenOffice.org или подобных).

Офисное программирование имеет ряд особенностей, отличающих его от программирования в общем смысле:

- цели разработки;
- область применения;
- макроязык;
- среда разработки;
- поддержка объектно-ориентированного программирования.

Рассмотрим эти особенности на примере MS Office.

ЦЕЛИ РАЗРАБОТКИ

В офисной среде *программный проект неразрывно связан с документом,* хранится как часть документа и не может существовать независимо от него. *Документ, а не программа, является целью разработки.*

Стандартные возможности среды по работе с документами велики. Однако возможность изменить типовой документ, снабдив его дополнительными функциями - это одна из важнейших задач офисного программирования. Для ее решения офисная среда представляет совокупность библиотек классов, которые составляют каркас (Framework) текстовых документов, электронных таблиц, презентаций, баз данных и приложений на основе этих документов. Всякий раз, когда создается новый документ, его каркас составляют объекты библиотек, заданные по умолчанию. Этот каркас можно

существенно изменить, добавив в документ новые свойства. Расширение каркаса не требует от программиста значительных усилий - достаточно включить в него необходимые библиотеки классов.

ОБЛАСТЬ ПРИМЕНЕНИЯ

Область применения офисного программирования широка - от настройки отдельных документов до решения задач автоматизации офисной деятельности масштаба предприятия, в т.ч. ориентированных на совместную работу в глобальной сети.

VISUAL BASIC FOR APPLICATION

Visual Basic для приложений (Visual Basic for Application, VBA) - это инструмент разработки приложений, который позволяет создавать программные продукты, решающие практически все задачи, встречающиеся в среде Windows. Эти продукты можно использовать, например, для оформления документов (подготовки текстов) или анализа данных таблиц (электронных таблиц). VBA - уникальное приложение, поскольку оно встраивается в другое приложение и расширяет его функциональные возможности.

Visual Basic for Application (VBA) - стандартный макроязык пакета Microsoft Office, предназначенный для расширения функциональных возможностей приложения в котором используется.

С помощью VBA можно:

- создать собственное диалоговое окно и придать ему требуемый внешний вид;
- создать макросы, расширяющие функциональные возможности приложения, в которое встроен VBA;
- изменить меню приложения Microsoft Office;
- управлять другим приложением Microsoft Office или принадлежащими ему данными;
- объединить данные из нескольких приложений Microsoft Office в одном документе;
- автоматически создавать или изменять страницы Web, совместно используя приложения Microsoft Office и VBA.

Для разработчика доступны следующие инструменты и средства, которые используются при создании проекта VBA:

- отладка приложений без предварительной компиляции;
- средства Win32 API;
- SQL и объекты доступа к данным для управления данными и извлечения их из внешних источников данных, таких как Microsoft SQL Server;
- построение и проверка элементов интерфейса непосредственно в среде разработки VBA (Integrated Development Environment, IDE);
- связывание программ и процедур с событиями, которые возникают в приложениях VBA.

СРЕДА РАЗРАБОТКИ

Среда приложений Office ориентирована в первую очередь на пользователей, а не на программистов и в ней можно создавать документы без всякого программирования. Поэтому программист обычно начинает работать с документами не на пустом месте, а с их заготовками, созданными пользователями, т.е. и сам программист может выступать в роли пользователя. Средства совместной работы над документами Office обеспечивают одновременную работу программистов и пользователей.

Среда MS Office предлагает два способа создания программ, отличающихся подходом к процессу: использование макрорекордера и ручное кодирование (на языке VBA). Эти подходы ориентированы на разные категории: непосредственно пользователей и программистов соответственно.

Макрорекордер (MacroRecorder) - это программный инструмент, записывающий действия пользователя при работе с документами и приложениями, с сохранением записи в виде макроса -исходного кода на языке VBA. При вызове сохраненного макроса воспроизводится вся сохраненная последовательность действий.

Макрорекордер представляет возможность создания программного проекта или, по крайней мере, его отдельных компонентов автоматически, без программирования. Для записи и воспроизведения макроса не требуется специальных знаний, поэтому пользователь может самостоятельно создавать программы (макросы), в общем случае даже не представляя себе как они работают.

Для программиста макрорекордер полезен тем, что позволяет создавать фрагменты программы автоматически, тем самым увеличивая скорость разработки и уменьшая время отладки.

Интегрированная среда разработки на VBA (Visual Basic Environment, VBE) - встроенное в MS Office средство для написания, тестирования и отладки приложений на VBA. Среда VBE представляет все возможности для создания законченных офисных приложений, включая средства визуального проектирования пользовательского интерфейса. VBE ориентирована на использование программистами для разработки офисных приложений (это отнюдь не означает, что пользователи не могут применять VBE).

ПОДДЕРЖКА ООП

Разработка приложений для MS Office тесно связана с парадигмой объектно-ориентированного программирования. Все документы (более того, сами компоненты пакета) в MS Office - суть объекты, наделенные собственными наборами свойств (характеристик объекта), методов (подпрограмм управления свойствами) и событий (подпрограмм, обрабатывающих изменения состояния объекта в результате некоторых действий). Соответственно, для обеспечения более полной интеграции с пакетом, входной язык (VBA) также поддерживает ООП.

Все объекты приложения MS Office образуют иерархическую структуру, которая определяет связь между ними и способ доступа. Такая структура называется объектной моделью (object model). За рамки объектной модели выходят, но также могут использоваться в офисных приложениях, внешние объекты, поддерживающие технологии DDE, OLE/ActiveX и ряд других.

В объектно-ориентированную концепцию удачно вписывается технология *визуального программирования*. Все отображаемые элементы графического интерфейса, такие как формы, элементы управления, меню и панели инструментов являются объектами, наделенными набором свойств и методов и способными реагировать на события (например, щелчки мыши, нажатия клавиш и т.п.). При визуальном подходе не требуется программного задания (хотя это и возможно) их основных свойств (например, ширина или высота, цвет фона и т.п.). Эти свойства можно задать при помощи мыши (например, ширину и высоту формы путем операции

"перетаскивания" маркеров) или установить их в окне свойств (название формы, цвет фона формы и т. д.). Таким образом, визуальное программирование делает проектирование интерфейса программы более наглядным и быстрым. При этом сохраняется возможность управлять всеми объектами и программно.

ПРЕИМУЩЕСТВА ОФИСНОГО ПРОГРАММИРОВАНИЯ

Преимущества, которые получает конечный пользователь, использующий программируемые офисные документы:

- Пользователь получает документы, обладающие новыми функциями и способные решать задачи, характерные для проблемной области пользователя.
- Пользователь находится в единой офисной среде независимо от того, с каким документом он работает в данный момент и какой программист разрабатывал этот документ.
- Большинство доступных при работе с документами функций являются общими для всех документов, поскольку их предоставляет сама офисная среда. Единый стиль интерфейса разных документов облегчает работу с ними.
- Пользователь сам, не будучи программистом, способен создавать простые виды программируемых офисных документов, постепенно совершенствуясь в этой деятельности.
- Преимущества, которые получает программист, работающий в Office:

- В распоряжении программиста находится мощная интегрированная среда. Для него эта среда представлена в виде совокупности хорошо организованных объектов, доступных в языке программирования и по принципу работы ничем не отличающихся от встроенных объектов языка или объектов, создаваемых самим программистом.
- Большинство повседневных задач становятся для него простыми, – чтобы их решить, зачастую достаточно стандартных средств.
- Там, где стандартных средств не хватает, где у документа должны появиться новые функциональные возможности, где необходимо создать документ по заказу, вступает в силу язык программирования – VBA, существенная особенность которого – возможность работы с объектами любого из приложений Office.
- Офисное программирование позволяет применять на практике идеи компонентного программирования. Компонентный подход предполагает взаимодействие

компонентов, создаваемых в разных программных средах, на разных языках, на разных платформах и находящихся на разных машинах. Работа с компонентами (DLL, ActiveX, AddIns, ComAddIns) является неотъемлемой частью офисного программирования.

МАКРОСЫ. ИСПОЛЬЗОВАНИЕ МАКРОРЕКОРДЕРА

МАКРОСЫ

Независимо от используемых операционной системы и программных приложений MS Office пользователь часто выполняет одни и те же последовательности команд для многих рутинных задач. Вместо повторения последовательности команд каждый раз, когда необходимо выполнить какую-либо задачу, можно создать макрос (macro), который вместо пользователя будет выполнять эту последовательность. Термин macro произошел от греческого слова, означающего расширенный или растянутый.

Макрос - это программа (в контексте офисного программирования - созданная автоматически), состоящая из списка команд, которые должны быть выполнены приложением.

Основными преимуществами использования макросов являются:

- повышение точности и скорости работы, поскольку компьютеры лучше приспособлены для выполнения повторяющихся задач, чем человек;
- при выполнении макросов обычно нет необходимости в присутствии человека-оператора; в случае, если макрос очень длинный и выполняет операции, требующие значительного времени (например, поиск в базе данных и сортировка), пользователь может переключиться на другое приложение.

Макрос служит для объединения нескольких различных действий в одну процедуру, которую можно легко вызвать. Этот список команд состоит в основном из макрокоманд, которые тесно связаны с приложением, в котором создается макрос - т.е. с командами Word, Excel или других приложений Microsoft Office.

Можно выделить *три основные разновидности макросов:*

1. *Командные макросы* - это наиболее распространенные макросы, обычно состоящие из операторов, эквивалентным тем или иным командам меню или

параметрам диалоговых окон. Основным предназначением такого макроса является выполнение действий, аналогичных командам меню - т.е. изменение окружения и основных объектов приложения.

2. *Пользовательские функции* - работают аналогично встроенным функциям приложения. Отличие этих функций от командных макросов состоит в том, что они используют значения передаваемых им аргументов, производят некоторые вычисления и возвращают результат в точку вызова, но не изменяют среды приложения.

3. *Макрофункции* - представляют сочетание командных макросов и пользовательских функций. Они могут использовать аргументы и возвращать результат, подобно пользовательским функциям, а также могут изменять среду приложения, как и командные макросы. Чаще всего эти макросы вызываются из других макросов, и активно используются для модульного программирования.

Поддержка макросов позволяет порой обойтись вообще безо всякого программирования: достаточно включить автоматическую запись выполняемых пользователем действий и в результате получить готовый макрос, а затем назначить ему кнопку на панели инструментов или новую команду меню, которые будут использоваться для вызова. Простые макросы удается создавать, не написав вручную ни одной строки программного кода. Для разработки же серьезных приложений приходится программировать.

Таким образом, различают 2 способа разработки макроса:

1. автоматическое создание, с использованием макрорекордера;
2. написание макроса "с нуля", используя язык программирования VBA.

Отметим, что возможен и комбинированный подход: фрагменты будущей программы записываются автоматически, а затем они корректируются и дополняются "рукописным" кодом.

Для записи макросов из приложений Microsoft Office используется **макрорекордер**. Это встроенный инструмент, который фиксирует все действия пользователя, включая ошибки и неправильные запуски. При выполнении макроса интерпретируется каждая записанная макрорекордером команда точно в такой последовательности, в которой пользователь выполнял их во время записи.

Для **записи макроса** в приложении Microsoft Office можно использовать меню "Сервис/Макрос/Начать запись" или выбрать кнопку "Записать макрос" на панели инструментов Visual Basic (для Word 2007 «Вид/Макросы/Начать запись). До начала записи нужно указать имя макроса и определить, где он будет храниться и как будет доступен. Затем выполнить действия, которые требуется сохранить в макросе. Для завершения записи нужно на панели инструментов "Остановка записи" щелкнуть кнопку "Остановить запись".

Для **выполнения макроса** необходимо:

1. Установить курсор в место вставки выполнения макроса.
2. Выбрать пункт меню "Сервис/Макрос/Макросы".
3. В появившемся диалоговом окне "Макрос" выбрать имя нужного макроса и выбрать "Выполнить".

Чтобы **просмотреть код** записанного макроса, надо выбрать меню "Сервис/Макрос/Макросы" (для Word 2007 «Вид/Макросы/Макросы). В появившемся диалоговом окне выбрать имя нужного макроса и щелкнуть кнопку "Изменить". Исходный код указанного макроса будет загружен в окно редактора Visual Basic.

СТРУКТУРА ЗАПИСАННОГО МАКРОСА

Макросы, создаваемые макрорекордером MS Office, сохраняются в специальной части файла данных, называемой *модулем*. Модуль VBA содержит исходный код программы на языке VBA. Фактически макрос является подпрограммой (а точнее, процедурой) VBA. Записанный макрос имеет строго определенную структуру. Ниже представлен исходный код простого макроса, созданного в Microsoft Word.

Листинг 1. Пример макроса

```
Sub Hello()
' Макрос изменяет размер, начертание шрифта, выравнивание абзаца и
' выводит надпись в активный документ MS Word

    Selection.Font.Size = 24
    Selection.Font.Bold = wdToggle
    Selection.ParagraphFormat.Alignment = wdAlignParagraphCenter
```

```
Selection.TypeText Text:="Hello, World!" End Sub
```

В общем виде структуру кода макроса можно представить следующим образом :

```
Sub имяМакроса () 'текст
комментария

    оператор1

    оператор2

    ОператорN

    End Sub
```

Каждый макрос VBA начинается с ключевого слова Sub, за которым следует имя макроса. Строку, содержащую ключевое слово Sub и имя макроса, называют *строкой объявления (declaration)* макроса. За именем макроса всегда следуют пустые круглые скобки (т.к. макрос является процедурой VBA без параметров).

За строкой объявления макроса следуют строки комментариев. *Комментарий (comment)* – это строка в макросе VBA, которая не содержит инструкций, являющихся частью этого макроса. Каждая строка комментария начинается с символа апострофа ('). Комментарии содержат имя макроса и текст, который был введен пользователем в текстовое поле "Описание" ("Description") диалогового окна "Запись макроса" ("Record Macro") в момент записи этого макроса.

Сразу за объявлением макроса следует *тело макроса (body)*. Каждая строка в теле макроса состоит из одного или более операторов VBA. *Оператор VBA (statement)* – это последовательность ключевых слов и других символов, которые вместе составляют одну полную инструкцию для VBA. Макрос VBA состоит из одного или нескольких операторов.

Конец макроса выделяется ключевой строкой End Sub, завершающей тело макроса.

СРЕДА РАЗРАБОТКИ VBE

Visual Basic for Application (VBA) – это система программирования, которая используется как единое средство программирования во всех приложениях Microsoft

Office. Всякая система программирования включает в себя, по меньшей мере, три составные части:

1. Язык (или языки) программирования.
2. Среду разработки, т.е. набор инструментов для написания программ, редактирования, отладки и т.п.
3. Библиотеку (или библиотеки) стандартных программ, т.е. набор готовых программ (процедур, функций, объектов и т.д.), которые можно использовать как готовые элементы при построении новых программ.

Для создания офисных приложений в MS Office имеется *интегрированная среда разработки* (Integrated Development Environment, *IDE*) с унифицированным интерфейсом. VBA IDE – это набор инструментов разработки программного обеспечения, таких как редактор Visual Basic (Visual Basic Editor, VBE), средства отладки, средства управления проектом и т.д.

Вызов VBA IDE из любого приложения выполняется через комбинацию клавиш **Alt+F11** или меню "Сервис/Макрос/Редактор Visual Basic".

СТРУКТУРА VBE

VBE – это стандартное интерфейсное окно, содержащее меню, панели инструментов, другие окна и элементы, которые применяются при создании проектов VBA. Общий вид окна редактора Visual Basic представлен на рис. 3.

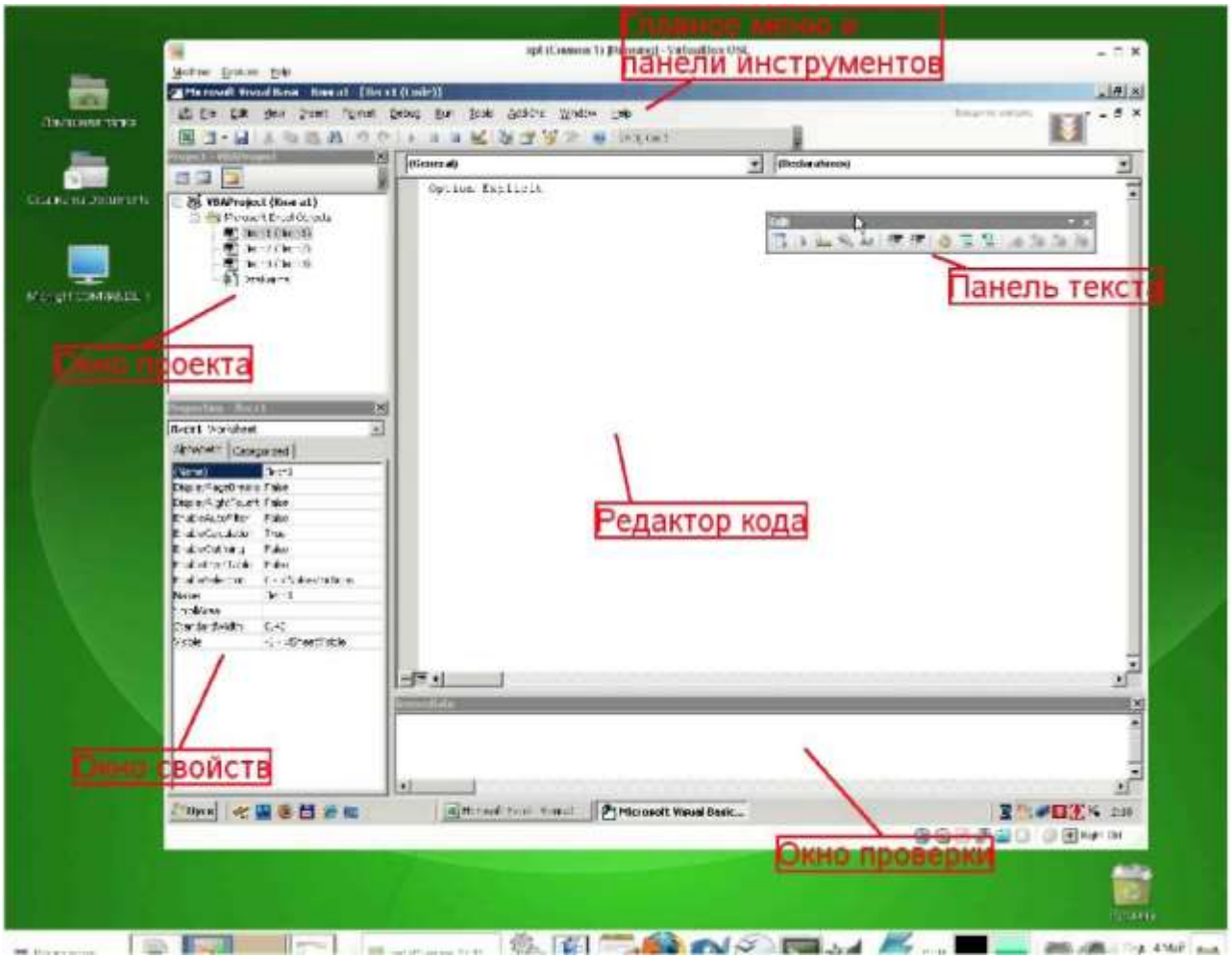


РИСУНОК 3. **ОКНО РЕДАКТОРА VISUAL BASIC**

Основными (открывающимися по умолчанию) являются три окна: окно проекта, окно свойств и окно редактирования кода. Краткое описание этих и некоторых других компонентов VBE приведено в таб. 4. Все они доступны через команды, представленные в меню "Вид".

ТАБЛИЦА 4. **НАЗНАЧЕНИЕ КОМПОНЕНТОВ VBE**

Наименование	**Описание**
Project (Проект)	Предназначено для отображения всех открытых проектов, а также их составляющих: модулей, форм и ссылок на другие
Toolbox (Панель	Содержит элементы управления для конструирования форм
UserForm	Используется для создания форм путем размещения на них элементов управления
Code (Программа)	Предназначено для просмотра, написания и редактирования программы на языке VBA. Поскольку среда разработки является многооконной, то для каждого модуля проекта можно открыть отдельное окно

Properties (Свойства)	Отображает свойства выделенных объектов. В этом окне можно задавать новые значения свойств формы и элементов
Object Browser (Просмотр объектов)	Отображает классы, свойства, методы, события и константы различных библиотек объектов. Используется для быстрого получения информации об объектах
Immediate (Проверка)	Предназначено для быстрого выполнения вводимых в него инструкций. В данном окне также выводятся результаты выполнения вводимых инструкций
Locals (Локальные	Автоматически показывает все переменные данной процедуры
Watches (Контрольные	Применяется при отладке программ для просмотра значений выражений

ХАРАКТЕРИСТИКИ КОМПОНЕНТОВ VBE

ОКНО ПРОЕКТА (PROJECT)

Проект - это совокупность всех программных модулей, связанных с документом Microsoft Office. Окно *Project (Проект)* предназначено для быстрого получения информации о различных составляющих проекта.

Проект может содержать модули следующих видов:

- *Объекты основного приложения*. Проекты VBA выполняются совместно с другими приложениями. Приложение, в котором разрабатывается и выполняется проект VBA, называется основным.
- *Модули форм*. В VBA имеется возможность создавать пользовательские формы, предназначенные для ввода или вывода данных, а также процедуры обработки событий, возникающие в этих формах.
- *Модули кода*. Модульность - один из основных принципов парадигмы структурного программирования. Каждый модуль, как правило, содержит подпрограммы, сходные по назначению. Небольшие модули проще отлаживать и использовать повторно. В частности, в VBE имеются средства импорта/экспорта готового кода.
- *Модули классов*. VBA позволяет создавать и использовать собственные объекты. Описание объектов включается в модули класса. Каждый модуль класса содержит полную информацию об одном типе объекта.

С помощью окна проекта можно добавить или удалить какой-либо объект из проекта. Модули кода добавляются в проект командой "Вставить/Модуль". Формы

создаются командой "Вставить/UserForm", а модули класса командой "Вставить/Модуль класса".

Окно проекта можно использовать также для быстрой навигации по формам проекта и программному коду. Для этого необходимо выбрать в контекстном меню соответственно команды "Объект" или "Программа".

Окно свойств (Properties)

Список свойств выделенного объекта выводится в окне Properties (Свойства). Для того чтобы выделить объект, необходимо с помощью окна проекта выбрать форму и перейти в режим конструктора, используя команду "View Object". Свойства объекта можно упорядочить в алфавитном порядке (Alphabetic (По алфавиту)) или по категориям (Categorized (По категориям)), выбрав соответствующую вкладку. Предусмотрена также возможность получения быстрой справки по какому-либо свойству объекта. Для этого достаточно установить курсор на нужное свойство и нажать клавишу F1.

Окно просмотра объектов(Object Browser)

Окно Object Browser (Просмотр объектов) предназначено для просмотра объектов, доступных при создании программы. Точнее, в этом окне отображаются не сами объекты, а структура соответствующего класса объектов. Окно просмотра объектов может использоваться для поиска метода или свойства объекта.

Окно Code (Окно редактирования кода)

Окно Code (Программа) представляет собой текстовый редактор, предназначенный для написания и редактирования кода процедур приложения. Это окно появляется на экране, например, при создании нового модуля. Код внутри модуля организован в виде отдельных разделов для каждого объекта, программируемого в модуле. Переключение между разделами выполняется путем выбора значений из списка "Object" ("Объект"), который находится в левом верхнем углу окна. Каждый раздел может содержать несколько процедур, которые можно выбрать из списка "Procedure" ("Процедура") в правом верхнем углу.

Интеллектуальные возможности редактора кода:

1. При написании кода пользователю предлагается список компонентов, логически завершающих вводимую пользователем инструкцию.

2. На экране автоматически отображаются сведения о процедурах, функциях, свойствах и методах после набора их имени.
3. Автоматически проверяется синтаксис набранной строки кода сразу после нажатия клавиши Enter. В результате проверки выполняется выделение определенных фрагментов текста:
    - красным цветом - синтаксические ошибки;
    - синим цветом - зарезервированные ключевые слова;
    - зеленым цветом - комментарии.
4. Если курсор расположить на ключевом слове VBA, имени процедуры, функции, свойства или метода и нажать клавишу F1, то на экране появится окно со справочной информацией об этой функции.

Окно редактирования форм (UserForm)

Для создания диалоговых окон, разрабатываемых приложений VBA, используются формы. Редактор форм является одним из основных средств визуального программирования. При добавлении формы в проект (команда "Insert" - "UserForm" ("Вставить" - "UserForm")) на экран выводится незаполненная форма с панелью инструментов Toolbox (Панель элементов).

Используя панель инструментов Toolbox (Панель элементов) из незаполненной формы конструируется требуемое для приложения диалоговое окно. Размеры формы и размещаемых на ней элементов управления можно изменять. Также окно редактирования форм поддерживает операции буфера обмена. Кроме того, команды меню "Format" ("Формат") автоматизируют и облегчают процесс выравнивания элементов управления как по их взаимному местоположению, так и по размерам.

Окна отладочной информации

Окно Immediate (Проверка) позволяет ввести инструкцию и выполнить ее. При этом инструкция должна быть записана в одну строку, директивы которой будут выполнены после нажатия клавиши Enter. Данное окно можно использовать для быстрой проверки действий, выполняемой той или иной инструкцией. Это позволяет не запускать всю процедуру, что удобно при отладке программ.

Окно Locals (Локальные переменные) автоматически отображает все объявленные переменные текущей процедуры и их значения.

Окно Watches (Контрольные значения) применяется при отладке программ для просмотра значений выражений.

СИНТАКСИС VBA

АЛФАВИТ VBA

Алфавит - это полный набор допустимых символов, принятых в языке программирования для обозначения данных и действий над ними. Алфавит языка VBA включает следующий набор символов:

- прописные (A - Z) и строчные (a - z) буквы латинского алфавита[3];
- цифры от 0 до 9;
- машинописные символы и знаки пунктуации: !, @, #, $, %, &;
- знаки арифметических операций (в порядке возрастания приоритета): +, -, *, /, |, ^;
- знаки операций отношения: =, <, >;
- знаки препинания и разделители: <пробел>, <перевод строки>, _ . : ; ( );
- ' - апостроф в качестве символа комментария. В алфавит языка входят также зарезервированные слова, которые не могут быть использованы в качестве идентификаторов. Примеры зарезервированных слов: Dim, Sub, Function, If и т.д.

ТИПЫ ДАННЫХ

Тип данных определяет диапазон возможных значений переменной, количество памяти для ее размещения и набор допустимых операций. Базовые типы данных VBA приведены в таб. 5.

ТАБЛИЦА 5. **ТИПЫ ДАННЫХ VBA**

Тип данных	Описание и диапазон значений
Array	Массив переменных любого встроенного типа данных
Boolean	True (истина) или False (ложь)
Byte	Положительное число от 0 до 255
Currency	Используется для денежных вычислений с фиксированным количеством десятичных знаков. От -922 337 203 685 477,5808 до 922 337 203 685 477,5807
Date	Дата и время. Диапазон дат: от 01.01.0100 г. до 31.12.9999 г. Диапазон времени: от 00:00:00 до 23:59:59

Decimal	Десятичное представление данных в целочисленной или вещественной форме
Double	Число с плавающей точкой двойной точности. Отрицательные числа: от -1,79769313486232E+308 до -4,94065645841247E-324. Положительные числа: 4,94065645841247E-324 до 1,79769313486232E+308
Integer	Целое число от -32 768 до 32 767
Long	Длинное целое число от -2 147 483 648 до 2 147 483 647
Object	Ссылка на объект
Single	Число с плавающей точкой обычной точности. Отрицательные числа от -3,402823E+38 до 1,401298E-45. Положительные числа от 1,401298E-45 до 3,402823E+38
String (переменной длины)	Длина строки от 0 до, приблизительно, 2 миллиардов символов
String (фиксированной длины)	от 0 до ~65 000 символов
Variant	Может использоваться для хранения любого типа данных, кроме строк фиксированной длины. Диапазон зависит от фактически сохраняемых данных.
Определяемый пользователем тип данных	Используется для описания различных структур данных

ИДЕНТИФИКАТОРЫ

В качестве идентификатров в VBA можно использовать произвольные последовательности букв и цифр длиной до 255 символов, которая обязательно должна начинаться с буквы. Эта последовательность может включать также символы подчеркивания и цифр. В качестве идентификаторов нельзя использовать зарезервированные слова языка и имена библиотечных объектов. Пробелы в идентификаторах недопустимы.

Примеры идентификаторов: SalesHistory, Counter, i, BirthDay и т.п.

ОБЪЯВЛЕНИЯ

ПЕРЕМЕННЫЕ

VBA поддерживает 2 способа объявления переменных:

1. **Неявный** - VBA создает переменную и резервирует память для ее хранения,

когда эта переменная в первый раз появляется в каком-либо операторе VBA (обычно в операторе присваивания).

2. **Явный** - имя и тип переменной определяются до первого обращения.
    - *Явное объявление переменных предпочтительнее,* так как:
    - ускоряет выполнение кода;
    - уменьшается количество ошибок;
    - код становится более понятным.

Для явного объявления переменных используется оператор Dim со следующим синтаксисом:

```
Dim <имяПеременной> [As <типДанных>]
```

Здесь:

имяПеременной - любой допустимый идентификатор;
типДанных - любой поддерживаемый VBA тип данных.

Для принудительного включения явного и обязательного объявления переменных используется оператор Option Explicit.Он должен быть расположен в самом начале модуля (раздел Declarations).

При *неявном объявлении переменной* можно задавать ее тип, добавляя в конец имени специальные символы определения типа (type definition character). Эта возможность сохранена для совместимости с устаревшими версиями языка Basic.

КОНСТАНТЫ

VBA поддерживает *неименованные* и *именованные* константы.

Неименованные константы - это фактические значения данных определенного типа. Их можно использовать без какого-либо объявления, непосредственно в выражениях.

Именованные константы - это мнемонические обозначения неименованных констант. Для использования в программе именованные константы должны быть предварительно объявлены с ключевым словом Const.

Синтакис объявления именованных констант:

```
Const <имяКонстанты> [As <типДанных>] = <значение1>
```

Где:

имяКонстанты - имя константы;
значение - значение константы.

Пример:

```
Const intPersonCount As Integer = 100

Const maxLen% = 50
```

Библиотеки типов VBA представляют множество встроенных константы. Такие константы используются обычно при работе с объектами приложения. Эти константы не требуют предварительного описания. Имена встроенных констант начинаются с префикса, который указывает, к объекту какого приложения Microsoft Office они относятся: xl(Excel), wd(Word), ac(Access), pp(Power Point), ol(Outlook), vb(VBA).

ОПЕРАЦИИ

В программах на VBA можно использовать весь типовой для универсального языка программирования набор операций:

- математические (или арифметические) – выполняются над числами и их результатом являются числа;
- отношения – применяются не только к числам и их результатом являются логические значения;
- логические – используются в логических выражениях и их результатом являются логические значения.

ТАБЛИЦА 6. **МАТЕМАТИЧЕСКИЕ ОПЕРАЦИИ**

Операци	**Формат**	**Название**
+	[Операнд1] + [Операнд2]	Сложение
-	[Операнд1] – [Операнд2]	Вычитание
-	- [Операнд1]	Перемена знака (или унарный минус)
*	[Операнд1] * [Операнд2]	Умножение
/	[Операнд1] / [Операнд2]	Деление
\	[Операнд1] \ [Операнд2]	Целочисленное деление
Mod	[Операнд1] Mod [Операнд2]	Остаток от деления
^	[Операнд1] ^ [Операнд2]	Возведение в степень

ТАБЛИЦА 7**. ОПЕРАЦИИ ОТНОШЕНИЯ**

Операция	Формат	Название
<	[Операнд1] < [Операнд2]	Меньше
>	[Операнд1] > [Операнд2]	Больше
<=	[Операнд1] <= [Операнд2]	Меньше или равно
>=	[Операнд1] >= [Операнд2]	Больше или равно
<>	[Операнд1] <> [Операнд2]	Не равно
=	[Операнд1] = [Операнд2]	Равно
Is	[Операнд1] Is [Операнд2]	Сравнение двух операндов, содержащих ссылки на объекты
Like	[Операнд1] Like [Операнд2]	Сравнение двух строковых выражений
And	[Операнд1] And [Операнд2]	Логическое умножение (И)
Or	[Операнд1] Or [Операнд2]	Логическое сложение (ИЛИ)
Xor	[Операнд1] Xor [Операнд2]	Исключающее Or (Исключающее
Not	Not [Операнд1]	Логическое отрицание (НЕ)
Imp	[Операнд1] Imp [Операнд2]	Логическая импликация
Eqv	[Операнд1] Eqv [Операнд2]	Логическая эквивалентность

ТАБЛИЦА 8. **Операции над строками**

Операц	Формат	Название
&	[Строка1] & [Строка2]	Конкатенация (сцепление строк). Для сцепления строк допустимо использование операции вида [Строка1] + [Строка2]

ОПЕРАТОР ПРИСВАИВАНИЯ

Оператор присваивания предназначен для задания ("присваивания") знач переменным, инициализации констант или изменения свойств объекта. Формат опера присваивания:

[Let] <имяЭлемента> = <выражение>

где:

Let - необязательная инструкция, которая сохранена для совместимости; <имяЭлемента> - это идентификатор переменной, константы (при объявлении) и свойства объекта; <выражение> состоит из переменных, констант, операций и функций.

Примеры использования оператора присваивания:

Place="d:\windows\system"

File="GameTree.Exe"

Student_Card(100).Group=33

Для присваивания переменной ссылки на объект применяется инструкция Set:

Set <объектная Переменная> = [New] <объектноеВыражение> | Nothing

где:

New - опция (ключевое слово), которая используется при создании нового экземпляра класса;

Nothing - опция (ключевое слово), которая позволяет освободить все системные ресурсы и ресурсы памяти, выделенные для объекта.

В следующем примере инструкция Set присваивает переменной MyRange ссылку на диапазон ячеек A1:B1.

Set Myrange = Range("A1:B1")

ПРИОРИТЕТЫ ОПЕРАЦИЙ

Порядок выполнения операций определяется расстановкой круглых скобок и приоритетом (старшинством) операций. Это обеспечивает однозначность в трактовании значений выражений. В таб. 9 приведены приоритеты выполнения операций.

ТАБЛИЦА 9. **Приоритеты операции VBA**

Приоритет	Операция
1	Вызов функции и скобки
2	^
3	- (смена знака)
4	*, /
5	\
6	Mod
7	+, -
8	>, <, >=, <=, <>, =
9	Not
10	And
11	Or
12	Xor
13	Eqv
14	Imp

Для иллюстрации вышесказанного приведем пример программы, вычисляющей диаметр, периметр и площадь круга, заданного радиусом (листинг 2).

Листинг 2. Переменные, константы, операции

```
Private R As Single  'радиус
Public D As Single, S As Single, P As Single 'диаметр, периметр, площадь
Const PI = 3.14 'Константа PI (имеется встроенная функция VBA)

Sub sample2()
Dim res As String
R = 20
D = 2 * R 'вычисляем диаметр
S = PI * R ^ 2 'вычисляем площадь
P = D * PI 'вычисляем периметр
 'формируем строку сообщения
res = "Для окружности с радиусом " & CStr(R) & " диаметр равен " & CStr(D) &
_
                ", периметр - " & CStr(P) & ", площадь - " & CStr(S) Debug.Print res ' вывод
                результата в окно отладчика End Sub
```

УПРАВЛЯЮЩИЕ СТРУКТУРЫ

ВЕТВЛЕНИЯ

Для реализации ветвлений в программах на VBA используются следующие операторы:

- условный оператор If;
- оператор выбора Select Case;
- оператор безусловного перехода Goto.

Условный оператор - это структура, которая выбирает ту или иную ветвь кода процедуры на основе некоторого предопределенного условия или группы условий. Общий синтакис условного оператора:

```
If <условие1> Then
```

```
        <Блок операторов1> [ElseIf
<условие2> Then

        <Блок операторов2> [ElseIf
<условиеN> Then

        <Блок операторовN>] [Else

        <Блок операторов_Else>]]
End If
```

где:

<условиеN> - проверяемое условное выражение;

<Блок операторовN>; - операторы, выполняемые при истинности условия;

<Блок операторов_Else> - операторы, выполняемые по умолчанию.

Блоки ElseIf и Else - необязательны.

Помимо приведенной полной формы, поддерживается синтаксис краткой однострочной формы условного оператора следующего вида:

```
If <условие> Then <оператор>
```

Примеры использования условного оператора приведены в листингах 3 и 4.

Листинг 3. Полная форма условного оператора

```
Sub sample4()

        Dim value As Long

        Const MSG = "Вы ввели число, "

        value = InputBox(prompt:="Введите число", Title:="Пример 4")

        If value = 0 Then

                MsgBox (MSG & "равное 0") ElseIf value >
        0 Then

                MsgBox (MSG & "большее 0") Else

    MsgBox (MSG & "меньшее 0") End If End Sub
```

Листинг 4. Краткая форма условного оператора

```
Sub sample5()
        Dim value As Long
        Const MSG = "Вы ввели число, "
        value = InputBox(prompt:="Введите число", Title:="Пример 4")
        If value = 0 Then MsgBox (MSG & "равное 0")
        If value > 0 Then MsgBox (MSG & "большее 0")
        If value < 0 Then MsgBox (MSG & "меньшее 0") End Sub
```

Логические выражения могут быть построены с помощью логических функций Not, And, Or, Xor, Imp, Eqv (листинг 5).

Листинг 5. Составные логические выражения

```
'Программа запрашивает имя пользователя и пароль, 'проверяет
введенную информацию и выводит сообщение.
Sub sample5()
        Const UNAME = "User"
        Const PASSWD = "TopSecret"
        login = InputBox(prompt:="Логин")
        pass = InputBox(prompt:="Пароль")
        If (login = UNAME) And (pass = PASSWD) Then
                MsgBox ("Добро пожаловать!") Else
    MsgBox ("Вы не авторизованы!") End If End Sub
```

Оператор выбора применяется в том случае, если проверяемое условное выражение может принимать много значений. В такой ситуации запись кода получается более компактной и наглядной, чем при использовании условного оператора.

Формат оператора Select Case:

```
Select Case <проверяемоеВыражение> Case
    <списокЗначений1> <блокОператоров1>
    [Case <списокЗначений2>
    <блокОператоров2>]
```

[...][Case Else

<блокОператоров_Else>] End Select

Проверяемое выражение вычисляется в начале работы оператора Select Case.

СписокЗначений - это одно или несколько выражений, разделенных запятой. При выполнении оператора проверяется, соответствует ли хотя бы один из элементов этогосписка проверяемому выражению. Эти элементы списка значений могут иметь одну из трех форм:

1. <выражение> - в этом случае проверяется, совпадает ли значение *проверяемогоВыражения* с этим выражением.
2. <выражение1> To <выражение2> - проверяется, находится ли значение *проверяемогоВыражения* в указанном диапазоне значений.
3. Is <логическаяОперация> <выражение> - *проверяемое выражение* сравнивается с указанным значением с помощью заданной логической операции (или операции отношения).

Отыскивается только первый подходящий элемент списков выражений. Пример использования оператора Select Case приведен в листинге 6.

Листинг 6. Оператор выбора

```
'Процедура выводит на экран окно сообщения,
'содержащее командные кнопки Yes, No и Cancel; затем определяет
'выбранную пользователем кнопку и выводит сообщение,
'поясняющее этот выбор.
Sub sample7()
    Const mTitle = "Демонстрация кнопок MsgBox"
    Dim Resp As Integer
    Resp = MsgBox(prompt:="Выберите кнопку", Title:=mTitle,
    Buttons:=vbYesNoCancel + vbQuestion)
    Select Case Resp
        Case Is = vbYes
```

```
MsgBox prompt:="Вы выбрали кнопку 'Да'", Title:=mTitle,
 Buttons:=vbInformation
Case Is = vbNo
MsgBox prompt:="Вы выбрали кнопку 'Нет'", Title:=mTitle,
Buttons:=vbInformation
Case Is = vbCancel
MsgBox prompt:="Вы выбрали кнопку 'Отмена'", Title:=mTitle,
Buttons:=vbCritical End Select End Sub
```

Оператор безусловного перехода GoTo всегда изменяет порядок выполнения операторов в процедуре или функции VBA без проверки каких-либо условий. Синтаксис оператора GoTo:

GoTo <метка>

где:

<Метка> - это любая допустимая метка в той же процедуре или функции, которая содержит оператор GoTo.

Листинг 7. Использование GoTo

```
'Программа будет запрашивать ввод значения пока не будет введено четное число
Sub sample6()
getValue:  ' метка
value = InputBox(prompt:="Введите четное число")
If (value Mod 2 <> 0) Then GoTo getValue End Sub
```

Для иллюстрации решения задачи из листинга 7 приведем один из альтернативных способов, с помощью оператора цикла (листинг 8).

Листинг 8. Отказ от использования GoTo.

```
Sub sample7()
  Do
value = InputBox(prompt:="Введите четное число")
Loop While value Mod 2 <> 0 End Sub
```

VBA поддерживает циклические структуры двух видов:

1. Циклы с фиксированным числом повторений *(циклы со счетчиком).*
2. Циклы с неопределенными числом повторений *(циклы с условием).*

Для всех видов циклов используется понятие *тело цикла,* определяющее блок операторов, заключенных между начальным и конечным операторами цикла. Каждое повторение выполнения операторов тела цикла называется *итерация.*

ФИКСИРОВАННЫЕ ЦИКЛЫ

VBA предоставляет две управляющие структуры для организации фиксированного цикла: For … Next (цикл со счетчиком) и For Each … Next (цикл с перечислением).

Оператор For … Next это типовой цикл со счетчиком, выполняющий заданное число итераций. Синтаксис оператора For … Next:

```
For <счетчик> = <начЗначение> To <конЗначение> [Step <приращение>]
    <блок операторов> Next
[<счетчик>]
```

Пример использования оператора For … Next.

Листинг 9. Оператор For … Next

```
' ЗАДАЧА: Составить программу, которая получает два числа от пользователя. ' Складывает все
числа в диапазоне, заданном этими двумя числами, а затем ' отображает результирующую сумму.

Sub sample70
    Dim i As Integer ‘счетчик цикла
    Dim sStart              ‘начальное значение счетчика
    Dim sEnd                ‘конечное значение счетчика
    Dim sSum As Long ‘результирующая сумма
    sStart = InputBox("Введите первое число:")
    sEnd = InputBox("Введите второе число:")
    sSum = 0
```

```
For i = CInt(sStart) To CInt(sEnd) sSum = sSum + i
Next i
MsgBox "Сумма чисел от " & sStart & " до " & sEnd & " равна: " & sSum End Sub
```

Оператор цикла For Each ... Next относится к категории операторов объектного типа, т.е. применяется в первую очередь к коллекциям объектов, а также к массивам. Тело цикла выполняется фиксированное число раз, соответствующее числу элементов массиве или коллекции.

Формат оператора For Each ... Next:

```
For Each <элемент> In <группа>
    <блок операторов> Next
[<элемент>]
```

ЦИКЛЫ С УСЛОВИЕМ (НЕОПРЕДЕЛЕННЫЕ ЦИКЛЫ)

Циклы с условием используются в тех случаях, когда повторяющиеся действия нужно выполнять только при определенных условиях. Количество итераций не определено и в общем случае может быть равно нулю (в частности, для циклов с предусловием).

VBA предлагает разработчикам несколько управляющих структур для организации циклов с условием:

- Четыре вида циклов Do..Loop, которые различаются типом проверяемого условия и временем выполнения этой проверки.
- Непрерываемый цикл While ... Wend.

Цикл Do While ... Loop - типичный *цикл с предусловием*. Условие проверяется до того, как выполняется тело цикла. Цикл продолжает свою работу, пока это <условие> выполняется (т.е. имеет значение True). Так как проверка выполняется в начале, то тело цикла может ни разу не выполниться.

Формат цикла Do While ... Loop:

```
Do While <условие>
    <блок операторов> Loop
```

Листинг 10. Цикл Do While ... Loop

```
' ЗАДАЧА: Составить программу, которая предусматривает ввод пользователем '
произвольной последовательности чисел. Ввод должен быть прекращен ' только после того,
как сумма введенных нечетных чисел превысит 100. Sub sample8()

    Dim OddSum As Integer  ‘сумма нечетных чисел
    Dim OddStr As String ‘строка с нечетными числами
    Dim Num                    ‘для приема вводимых чисел

    OddStr = ""                ‘инициализация выходной строки

    OddSum = 0                 ‘инициализация суммы OddSum

    Do While OddSum < 100  ‘начало цикла

        Num = InputBox("Введите число: ")

        If (Num Mod 2) <> 0 Then  ‘проверка на четность

    OddSum = OddSum + Num  ‘изменение суммы OddSum OddStr = OddStr &
    Num & " " End If Loop

 'вывод строки с нечетными числами: MsgBox
prompt:="Нечетные числа: " & OddStr End Sub
```

Оператор Do ... Loop While предназначен для организации *цикла с постусловием*.

Условие проверяется после того, как тело цикла, будет выполнено хотя бы один раз. Цикл продолжает свою работу, пока <условие> остается истинным.

Формат цикла Do ... Loop While:

```
Do

    <блок операторов> Loop
While<условие>
```

Листинг 11. Цикл с постусловием

```
' ЗАДАЧА: Составить программу игры "Угадай число". Программа должна случайным

' образом генерировать число в диапазоне от 1 до 1000, пользователь должен

' угадать это число. Программа на каждое вводимое число выводит подсказку

' "больше" или "меньше".

Sub sample8()
```

```
Randomize Timer ' инициализация генератора случайных чисел

Dim msg As String   ' строка сообщения

Dim SecretNumber As Long, UserNumber As Variant

Begin:

    SecretNumber = Round(Rnd * 1000) ' число, сгенерированное компьютером
    UserNumber = Empty                          ' число, вводимое пользователем

Do ' игровой процесс

    Select Case True

Case IsEmpty(UserNumber): msg = "Введите число" Case UserNumber > SecretNumber: msg =
"Слишком много!" Case UserNumber < SecretNumber: msg = "Слишком мало!" End Select
UserNumber = InputBox(prompt:=msg, Title:="Угадай число") Loop While UserNumber <>
SecretNumber ' проверка If MsgBox("Играть еще? ", vbYesNo + vbQuestion, "Вы угадали!") = vbYes
Then

    Goto Begin End If
End Sub
```

Циклы Do Until ... Loop и **Do ... Loop Until** являются инверсиями ранее рассмотренных циклов с условием. В общем случае они работают аналогично, за исключением того, что тело цикла выполняется при ложном условии (т.е. <условие>=False).

Формат цикла Do Until ... Loop:

```
Do Until <условие>

    <блок операторов> Loop
```

Формат цикла Do ... Loop Until:

```
Do

    <блок операторов> Loop
Until<условие>
```

ПРАКТИЧЕСКОЕ ЗАДАНИЕ:

Перепишите программы из листингов 10 и 11 с использованием инвертированных операторов цикла.

Цикл While ... Wend также относится к циклам с условием. Данный оператор полностью соответствует структуре Do While ... Loop. Формат цикла While ... Wend:

```
While <условие>
    <блок операторов> Wend
```

Отличительной особенностью этого оператора является невозможность принудительного завершения (прерывания) тела цикла (оператор Exit Do не работает в цикле While ... Wend).

Прерывание цикла

Для досрочного завершения итерации и выхода из цикла применяется оператор Exit. Этот оператор применим в любой циклической структуре, кроме While ... Wend. Общий синтаксис использования Exit для прерывания цикла таков:

```
<начало_цикла>
        [<блок операторов1>]
        Exit (For | Do)
        [<блок операторов2>]
        [Exit (For | Do)]
        ..
<конец_цикла>
```

При выполнении оператора Exit цикл прерывается, и управление передается оператору, следующему за оператором <конец_цикла>. В теле цикла может присутствовать несколько операторов Exit.

Листинг 12. Принудительный выход из цикла

```
Sub sample9()
    For i = 1 To 10000000
If i = 10 Then Exit For ' выход из цикла, когда счетчик достигнет 10 Next End Sub
```

ПОДПРОГРАММЫ

VBA поддерживает два типа подпрограмм: процедуры и функции.

- **Функция** - это подпрограмма, которая возвращает результат. Вызов функции является выражением, и может использоваться в других выражениях или в

правой части оператора присваивания.

- **Процедура** - это любая подпрограмма, которая не является функцией. Любой макрос VBA является подпрограммой типа "процедура".

  Для объявления процедуры в VBA используется ключевое слово Sub:

```
Sub <имяПроцедуры> [(<списокПараметров>)]
    <операторы>
End Sub
```

где:

<имяПроцедуры> - любой допустимый идентификатор VBA;

<списокПараметров> - список формальных параметров процедуры, если он пуст, то такая процедура является макросом; <операторы> - любая последовательность операторов VBA.

Листинг №.13 Пример объявления процедуры

```
' Процедура выводит в отладчик максимальное из трех чисел Sub sMax3(A As Long, B
As Long, C As Long) If (A > B) And (A > C) Then
                    Debug.Print "Max is "; A ElseIf (B > A)
    And (B > C) Then
                    Debug.Print "Max is "; B Else
    Debug.Print "Max is "; C End If End Sub
```

Синтаксис объявления функции несколько сложнее, чем синтаксис процедуры:

```
Function <имяФункции> [(<списокПараметров>)] [As <типФункции>] <операторы> . .
    .
    <имяФункции> = <возвращаемое_значение>
    [<операторы>]
End Function
```

где:

<имяФункции> - любой допустимый идентификатор;

<списокПараметров> - список формальных параметров процедуры;

<типФункции> - имя любого поддерживаемого VBA типа данных;

<операторы> - любая последовательность операторов VBA.

<возвращаемое_значение> - результат, передаваемый в вызывающую программу.

Листинг №14. Пример объявления функции

```
' Функция возвращает максимальное из трех чисел Function fMax3(A As Long, B
As Long, C As Long) As Long If (A > B) And (A > C) Then

                    fMax3 = A ElseIf (B > A) And (B >
    C) Then

                    fMax3 = B Else

    fMax3 = C End If End Function
```

Подпрограммы VBA могут принимать для обработки *формальные параметры,*указываемые при объявлении. При вызове они заменяются *фактическими параметрами,*т.е. реально используемыми в вызывающей программе.

В VBA список формальных параметров подпрограммы представляет имена переменных, разделенных запятой. При этом желательно указать тип каждой переменной:

```
Function | Sub <имяПроцедуры> (<параметр1> As <тип>, <параметр2> As
<тип>, ..., <параметрN> As <тип>)
```

Если тип данных параметра не указан, то автоматически будет использован тип Variant.

Список параметров может быть пустым как для процедуры, так и для функции. В этом случае после имени процедуры ставятся пустые круглые скобки.

При передаче фактических параметров в подпрограмму может использоваться один из двух различных способов:

- передача по значению;
- передача по ссылке.

При *передаче по значению,* в подпрограмме создается копия переданного фактического параметра. Все действия внутри подпрограммы выполняются над этой копией и при выходе из подпрограммы все изменения теряются.

Если переменная *передается по ссылке,* то процедуре или функции будет передан адрес этой переменной. Тем самым вызываемая процедура может изменить значение фактического параметра: если будет измен формальный параметр

процедуры, то переданный при вызове ей фактический параметр тоже изменит свое значение.

Способ передачи указывается при описании параметров в строке объявления подпрограммы. Имени параметра может предшествовать один из явных описателей способа передачи:

- ByRef - задает передачу по ссылке;
- ByVal - задает передачу по значению.

  По умолчанию выполняется передача по ссылке.

Листниг 15. Передача параметров в подпрограмму

```
' sample10 - вызывающая программа (макрос) ' ByValByRefDemo
- вызываемая процедура

Sub sample10()

        Dim a As Long, b As Long, c As Long ' фактические параметры

        a = 10

        b = 10

        c = 10

        ByValByRefDemo a, b, c ' передача фактических параметров

        Debug.Print "2: " & "a = " & a & "; b = " & b & "; c = " & c End Sub

Sub ByValByRefDemo(x As Long, ByVal y As Long, ByRef z As Long)

        ' выполнение действий над формальными параметрами

        x = x * 2

        y = y * 3

        z = z * 4

        Debug.Print "1: " & "x = " & x & "; y = " & y & "; z = " & z End Sub
```

Здесь объявлены две процедуры: sample10 и ByValByRefDemo. Процедура sample10

вызывает процедуру ByValByRefDemo и передает ей предварительно

инициализированные переменные a, b и c. Процедура ByValByRefDemo получает значения

переменных a, b и c в виде формальных параметров x, y и z соответственно, выполняет над ними указанные действия, выводит результат и завершается. После возврата из подпрограммы процедура sample10 выводит значения переменных a, b, c в окно отладчика (рис. 4).

РИСУНОК 4. **ПЕРЕДАЧА ПАРАМЕТРОВ В ПОДПРОГРАММУ**

Именованные параметры

При вызове подпрограмм в VBA параметры необходимо передавать в определенном порядке. Отсутствующие необязательные параметры отмечаются запятыми. Нарушение этого правила часто приводит к ошибкам - легко пропустить или поменять местами параметры. Чтобы избежать этих проблем, в VBA можно использовать *именованные параметры* функций. Для этого в вызове подпрограммы явно указываются имена параметров (как это было задано при объявлении подпрограммы), каждому из которых присваивается требуемое значение с помощью оператора ":=" ("двоеточие_равно").

При использовании именованных параметров можно не обозначать отсутствующие параметры и, кроме того, порядок перечисления параметров может быть произвольным.

Следующий пример показывает два обращения к функции MsgBox, которые имеют один и тот же результат:

```
' обычный вызов
MsgBox "Здравствуй, мир!", , "Окно приветствия"

' вызов, с использованием именованных параметров
MsgBox Prompt:= "Здравствуй, мир!", Title:= "Окно приветствия"
```

Вызов подпрограмм

Подпрограммы могут быть вызваны различными способами:

- *Процедуру (Sub)с непустым списком параметров* можно вызвать только из другой процедуры или функции так:

<имяПроцедуры> <списокПараметров> Или так:

Call <имяПроцедуры> (<списокПараметров>) ’использована инструкция Call.

- *Процедура с пустым списком параметров* рассматривается VBA как командный макрос. Ее также можно вызвать двумя способами:

    1. из другой процедуры или функции;
    2. с помощью комбинации клавиш быстрого вызова, команд меню или кнопок панелей инструментов.

- *Функцию (Function)* можно вызывать точно так же, как и процедуру, в виде отдельного оператора. В этом случае возвращаемое функцией значение игнорируется.

Листинг №16. Вызов процедуры

```
Sub sample11() ' вызывающий макрос

    Dim usr As String

    usr = InputBox("Login")

    Hello usr ' вызов процедуры Hello без Call

    Call Hello(usr) ' использование инструкции Call End Sub


' процедура принимает один параметр, формирует строку сообщения, ' выводит сообщение
в окно отладчика Sub Hello(usrname As String)

    Debug.Print "Hello, " & usrname & "!" End Sub
```

Если в проекте используется несколько подпрограмм с одинаковыми названиями (это возможно, если они в разных модулях), то при их вызове перед именем подпрограммы надо указывать (через точку) имя модуля, в котором процедура расположена:

<имяМодуля>.<имяПроцедуры> <списокФактическихПараметров>
Например

MyModule.MySub fArg, sArg ' вызов процедуры из модуля MyModule

Для вызова общих (Public) подпрограмм из другого проекта дополнительно к именам модуля и подпрограммы указывается имя проекта:

<имяПроекта>.<имяМодуля>.<имяПроцедуры> <списокФактическихПараметров> Например

someVal = MyProject.MyModule.MyFunc(fArg, sArg) ' вызов функции из проекта MyProject

Прерывание подпрограммы

В случае необходимости, выполнение процедуры или функции может быть прервано досрочно. Для этого нужно использовать инструкцию прерывания Exit Sub. В этом случае синтаксис объявления примет следующий вид (на примере объявления процедуры):

Sub <имяПроцедуры> [(<списокпараметров>)]

<операторы> Exit Sub

<операторы> End Sub

МОДУЛИ

Любая программа на VBA представлена в виде проекта. Проект - это совокупность программных модулей различных типов. В свою очередь модуль - это основная программная единица уровня проекта, в которой размещаются описания и реализация переменных, констант, типов, подпрограмм и т.д. Имеется три типа модулей:

- **Стандартные модули** - это модули, в которых можно описать доступные во всем проекте процедуры.
- **Модули класса** содержат описание свойств, методов и событий пользовательского класса.
- **Модули форм** содержат процедуры обработки событий, генерируемых элементами управления в форме.

Структура модуля VBA включает два неявных (т.е. не требующих специального описания) раздела: общий (General) и объявлений (Declarations). В общем разделе задаются параметры среды (Option Base, Option Explicit), приводятся описания глобальных переменных, констант и типов. Раздел объявлений предназначен для описания процедур и функций.

Область видимости

VBA поддерживает две области видимости для переменных и подпрограмм: локальную и глобальную.

Локальные переменные определены на уровне подпрограммы с помощью ключевых слов Dim или Static. Они доступны только внутри этой подпрограммы и по выходу из нее уничтожаются.

Глобальные переменные объявляются на уровне модуля. Такие переменные доступны:

- для всех подпрограмм модуля, в котором они объявлены (при объявлении с ключевым словом Private, либо Dim);
- для всего приложения - при объявлении с ключевым словом Public;

Подпрограммы VBA могут быть объявлены на двух уровнях - уровне проекта (Public) и уровне модуля (Private). Например:

```
Public Sub Query(price, count) ' видимость на уровне проекта Private Sub Sub Query(price, count) ' видимость на уровне модуля
```

По умолчанию используется уровень проекта.

При описании локальных переменных можно использовать ключевое слово Static.

Такие переменные являются статическими и сохраняют значения между вызовами.

ВСТРОЕННЫЕ ФУНКЦИИ VBA

В языке программирования VBA предусмотрено несколько десятков *встроенных функций*. Они доступны в любой программе на языке VBA, при этом безразлично, в среде какого программного продукта мы находимся - Excel, Word, Access или, к примеру, AutoCAD. Используются они очень активно, и во многих ситуациях без них не обойтись. Встроенные функции обычно группируют по назначению: математические, строковые, преобразования типов, логические и т.п. В справке по VBA имеется подробная информация о всех встроенных функциях. Здесь же приведем краткое описание только некоторых.

ФУНКЦИИ ПРИВЕДЕНИЯ ТИПОВ

Используются для конвертации типов данных. Вот перечень этих функций:

CBool(), CByte(), CCur(), CDate(), CDbl(), CDec(), CInt(), CLng(), CSng(), CStr(), CVar(), CVDate(), CVErr(). Просмотреть, что в итоге получилось, можно при помощи функции

TypeName(), например:

```
nVar1 = CInt(InputBox("Введите значение")) MsgBox
TypeName(nVar1)
```

Кроме того, еще несколько полезных для конвертации функций:

- *Str()* - позволяет перевести числовое значение в строковое, при этом вставляет пробел впереди для положительных чисел.
- *Val()* - извлекает из строки числовое значение (до первого нечислового символа, кроме точки). Очень удобно, когда вперемежку с числовыми данными прописываются единицы измерения или валюта.

Чтобы не возникло ошибок при конвертации, можно вначале проверять значения на возможность конвертации при помощи функций *IsNumeric()* и *IsDate()*. Для проверки на соответствие специальным значениям можно использовать функции *IsArray(), IsEmpty(),*

IsError(), IsMissing(), IsNull() и *IsObject()*. Все эти функции возвращают True или False в зависимости от результатов проверки переданного им значения.

СТРОКОВЫЕ ФУНКЦИИ

- *Left(), Right(), Mid()* - получить фрагмент строки слева, справа или из середины исходной строки соответственно.
- *Len()* - получить число символов в строке.
- *LCase()* и *UCase()* - перевести строку в нижний и верхний регистры соответственно.
- *LSet()* и *RSet()* - заполнить строку символами без изменения длины (соответственно слева и справа). Лишние символы обрезаются, на место недостающих подставляются пробелы.
- *LTrim(), RTrim(), Trim()* - убрать пробелы соответственно слева, справа или и слева, и справа.
- *Replace()* - заменить в строке одну последовательность символов на другую.
- *StrComp()* - сравнить две строки.
- *StrReverse()* - "перевернуть" строку, разместив ее символы в обратном порядке.

МАТЕМАТИЧЕСКИЕ ФУНКЦИИ

Функций для работы с числовыми значениями в VBA очень много. Приведем только некоторые из них.

- *ABS()* - эта функция возвращает абсолютное значение переданного ей числа.
- *Int(), Fix()* и *Round()* позволяют по разному округлять числа: Int возвращает ближайшее меньшее целое, Fix() отбрасывает дробную часть, Round() округляет до указанного количества знаков после запятой.
- *Rnd()* и команда *Randomize* используются для получения случайных значений. Обычный синтаксис при применении Rnd выглядит так:

  случайное_число = Int(минимум + (Rnd()* максимум)) MsgBox(Int(1 + (Rnd() * 100))) Перед вызовом функции *Rnd()* следует выполнить команду Randomize для

  инициализации генератора случайных чисел.

- *Sgn()* - позволяет вернуть информацию о знаке числа. Возвращает 1, если число положительное, -1, если отрицательное и 0, если проверяемое число равно 0.

ФУНКЦИИ ДЛЯ РАБОТЫ С ДАТОЙ И ВРЕМЕНЕМ

Основные функции VBA для работы с датой/временем:

- *Date()* - возвращает текущую системную дату.
- *Time()* возвращает текущее системное время, а *Now()* - дату и время вместе.
- *DateAdd()* - возможность добавить к дате указанное количество лет, кварталов, месяцев и так далее - вплоть до секунд.
- *DateDiff()* - возможность получить разницу между датами.
- *DatePart()* - возвращает указанную часть даты (например, только год, только месяц или только день недели).
- *DateSerial()* - формирует значение даты на основе передаваемых символьных значений. То же самое делает *DateValue(),* отличия - в формате принимаемых значений. Аналогичным образом (для времени) работают *TimeSerial()* и *TimeValue().*
- *Day()* (а также *Year(), Month(), Weekday(), Hour(), Minute(), Second())* - специализированные заменители функции DatePart(), которые возвращают нужную часть даты.
- *MonthName()* - возвращает имя месяца словами по его номеру. Возвращаемое значение зависит от региональных настроек. Если они русские, то вернется русское название месяца.
- *Timer()* - возвращает количество секунд, прошедших с полуночи.

ФУНКЦИИ ВЗАИМОДЕЙСТВИЯ С ПОЛЬЗОВАТЕЛЕМ

Для организации диалога с пользователем VBA представляет две встроенные функции - MsgBox и InputBox. Окно сообщений MsgBox выводит сообщения для пользователя, а окно ввода InputBox обеспечивает возможность получения информации от пользователя.

Функция *MsgBox()* выводит на экран диалоговое окно, содержащее сообщение, устанавливает режим ожидания нажатия кнопки пользователем, а затем возвращает значение типа Integer, указывающее, какая кнопка была нажата.

Формат функции MsgBox:

MsgBox (Prompt [, Buttons] [, Title] [, HelpFile, Context]) Назначение параметров:

- Prompt - строковое выражение, отображаемое как сообщение в диалоговом окне;
- Title - строковое выражение, отображаемое в строке заголовка диалогового окна. Если этот параметр опущен, в строку заголовка помещается имя приложения, из которого запускается программа VBA;
- HelpFile - строковое выражение, определяющее имя файла справки, содержащего справочные сведения о данном диалоговом окне; обычно это файл, который вы уже должны были создать с помощью Windows Help Compiler. Если этот параметр указан, необходимо также указать параметр Context;
- Context - числовое выражение, определяющее номер соответствующего раздела справочной системы. Если этот параметр указан, необходимо указать также параметр HelpFile;
- Buttons - числовое выражение, которое задает количество и тип кнопок в диалоговом окне MsgBox. Buttons указывает также кнопку по умолчанию в диалоговом окне и содержит ли это диалоговое окно стандартные значки Windows для предупредительных сообщений и запросов пользователя. Значения Buttons можно получить из справочной системы.

  Простой пример использования функции MsgBox:

```
Sub sample3()
    MsgBox "Видите ли вы две кнопки?", vbYesNo + vbInformation, "Сообщение" End Sub
```

В данном примере MsgBox применяется не в виде функции, а в виде процедуры

(т.е. не возвращает никакого значения). Следовательно, код выбранной кнопки нигде не сохраняется и не может быть использован.

Чтобы определить, какая кнопка была нажата, MsgBox необходимо вызвать как функцию, т.е. сохранить возвращаемое значение в переменную (Листинг 17.

Листинг 17. Использование MsgBox

```
Sub sample4()
    Dim res ' объявляем переменную
     ' вызываем MsgBox и сохраняем значение в переменной
    res = MsgBox("Видите ли вы две кнопки?", vbYesNo + vbInformation,
"Сообщение")
    Debug.Print res ' печатаем полученное значение
End Sub
```

При выполнении этого макроса, когда пользователь выбирает кнопку Yes или No в переменной res сохраняется число, соответствующее выбранной кнопке.

Вместо возвращаемых функцией MsgBox целочисленных значений удобнее пользоваться предопределенными константами VBA. В таб. 10 приведены возвращаемые значения констант функции MsgBox.

ТАБЛИЦА 10. **Возвращаемые значения функции Msgbox**

Константа	Означает, что пользователь нажал кнопку
vbAbort	Стоп (Abort)
vbCancel	Отмена (Cancel)
vbIgnore	Пропустить (Ignore)
vbNo	Нет (No)
vbOk	Ок
vbRetry	Повтор (Retry)
vbYes	Да (Yes)

Дополним код листинга 17 проверкой возвращенного значения (листинг 18).

Листинг 18. Проверка возращаемого значения MsgBox

```
Sub sample5()
    ' вызываем MsgBox и сохраняем значение в переменной
    res = MsgBox("Видите ли вы две кнопки?", vbYesNo + vbInformation, "Сообщение")
    ' проверяем, какая кнопка нажата
```

```
If res = vbYes Then : MsgBox "Вы нажали Yes", , "Результат выбора"
    Else : MsgBox "Вы нажали No", , "Результат выбора"
    End If
End Sub
```

Функция *InputBox()* выводит на экран диалоговое окно, содержащее сообщение и поле ввода, устанавливает режим ожидания ввода текста, а затем возвращает значение типа String, содержащее текст, введенный в поле. Формат функции InputBox:

```
InputBox (Prompt [, Title] [, Default] [, XPos] [, Ypos] [, HelpFile, Context])
```

Назначение параметров:

- Prompt - строковое выражение, отображаемое как сообщение в диалоговом окне;
- Title - строковое выражение, отображаемое в строке заголовка диалогового окна. Если этот параметр опущен, в строку заголовка помещается имя приложения;
- Default - строковое выражение, отображаемое в поле ввода и используемое по умолчанию, если пользователь не введет другую строку. Если этот параметр опущен, поле ввода изображается пустым;
- Xpos и Ypos – числовые выражения, которые указывают местоположение окна ввода и являются координатами верхнего левого угла диалогового окна; Xpos – горизонтальное расстояние от левого края окна; Ypos – вертикальное расстояние от верхнего края окна. Если эти параметры опущены, диалоговое окно выравнивается по центру экрана;

Приведем пример использования функции InputBox для получения имени пользователя.

```
Sub sample6()
    Dim username As String
    username = InputBox("Введите ваше имя ", "Пример 6")
    MsgBox ("Здравствуйте, "+username) End Sub
```

В результате выполнения этого макроса на экран последовательно выводятся диалоговые окна ввода и вывода (рис. 5).

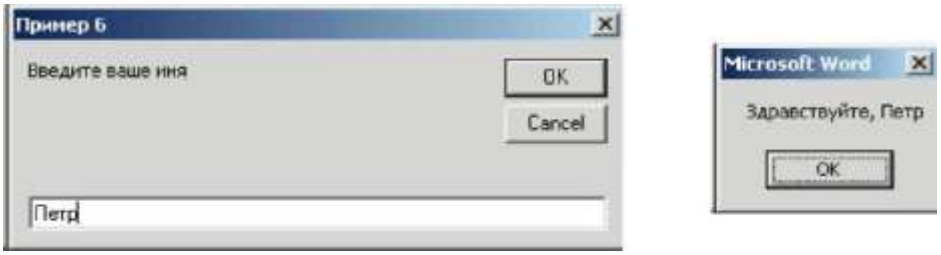


РИСУНОК 5. **ИНТЕРАКТИВНЫЕ ФУНКЦИИ VBA**

СТРУКТУРНЫЕ ТИПЫ ДАННЫХ

Кроме простых типов VBA предоставляет возможность создавать и использовать структурные типы данных. **Структурный тип** данных - это тип данных, который позволяет в одной величине хранить одновременно несколько значений. К структурным типам данных VBA относятся массивы и пользовательские типы данных.

МАССИВЫ

Массив - это упорядоченная совокупность данных одного типа. Порядок элементов массива задается индексами его элементов. Количество элементов определяет *размер массива,* а количество индексов (в VBA - до 60) - его *размерность*.

VBA поддерживает *статические* и *динамические* массивы.

Статический массив имеет фиксированный размер и размерность, заданные при объявлении и неизменяемые в ходе выполнения программы.

Синтаксис объявления статического массива:

(Public | Private | Dim) <имяМассива> (<размер1>, <размер2>, ..., <размер N>) As <типДанных>

Указанные в скобках величины <размер1>, <размер2>, …, <размер N> задают количество индексов и максимально допустимое значение для каждого конкретного индекса (его верхняя граница). Таким образом, определяются *размерность массива* (количество индексов) и *размер массива* - количество элементов данного массива. При этом индексирование элементов массива по умолчанию начинается с нуля. Так, объявление

Dim Array1 (9) As Integer,

определяет одномерный массив из 10 целых чисел, а объявление

Dim Array3 (4, 9) As Variant,

определяет двумерный массив из пятидесяти (5x10) элементов типа Variant.

В VBA имеется возможность изменить индекс нижней границы с помощью оператора Option Base (указание Option Base 1 или Option Base 0 в общем разделе модуля). Более того, при объявлении массива можно явно указать и верхнюю, и нижнюю границы. Синтаксис оператора объявления массива с указанием границ для индексов:

Dim <имяМассива> (<мин1> To <макс1>[, ..., <минN> To <максN>]) As <типДанных> Примеры:

Dim A (1 To 3, 1 To 3) As Single Dim B (1 To 12) As
Integer

Правила инициализации элементов массива такие же, как и для переменных того типа, который использован при объявлении массива. Доступ к элементам массива выполняется по индексу. Листинг 19 иллюстрирует работу с одномерным массивом.

Листинг 19. Обращение к элементам массива

```
' ЗАДАЧА: Сгенерировать 10 случайных целых чисел от 0 до 100,
' записать их в массив и вывести в окно отладчика
Sub sample14()
        Randomize Timer ' запуск генератора случайных чисел Dim myarr(1 To 10)
        As Long ' объявление массива

        ' запись чисел в массив For i = 1 To 10
                myarr(i) = Round(Rnd * 100)
        Next

        ' чтение элементов массива и вывод значений в отладчик For i = 1 To 10
    Debug.Print myarr(i) Next End Sub
```

Удобным способом определения одномерных массивов является функция Array, преобразующая список элементов, разделенных запятыми, в массив из этих значений:

```
Dim A As Variant
A = Array (10, 20, 30)
B = A(2)
```

В данном примере переменная A создается как одномерный массив, состоящий из трех элементов (10, 20, 30), а переменная B принимает значение второго элемента массива A (20).

VBA поддерживает использование *динамических массивов,* размер и размерность которых может изменяться во время выполнения программы. Объявление динамического массива аналогично объявлению статического, но при этом не задаются ни размер, ни размерность:

Dim <имяМассива> () As <типДанных>

Для указания и изменения размеров такого массива используется специальный оператор - ReDim:

ReDim <имяМассива> (<размер1>, <размер2>, ..., <размер N>)

Размерность и размер определяется динамически в той процедуре и в тот момент, когда они становится фактически известной. Обратите внимание, в этом операторе границы изменения индексов можно задать не только как константы, но и как выражения, зависимые от переменных.

Для повторных изменений массива можно снова использовать оператор переопределения ReDim. При каждом переопределении динамического массива все его значения теряются. Чтобы сохранить все ранее полученные элементы необходимо использовать ключевое слово Preserve при переопределении.

Листинг 20 демонстрирует работу с динамическим массивом (нумерация строк приведена только для удобства пояснений).

Листинг20. Работа с динамическим массивом

```
1: Public Vector() As Integer
2: Public Sub DMassiv()
3                   Dim N As Byte, I As Byte
4:
5          N = InputBox("Введите фактическую размерность вектора")
6                   ReDim Vector(N)
```

```
7:                  For I = 1 To N
8:                  Vector(I) = 2 * I + 1
9:                  Next I
10:
11:     'Массив расширяется с сохранением ранее вычисленных элементов
12:                 ReDim Preserve Vector(2 * N + 1)
13:                 For I = N + 1 To 2 * N + 1
14:                 Vector(I) = 2 * I
15:                 Next I
16:                 Debug.Print "Элементы массива Vector:" & Chr(13)
17:                 For I = 1 To 2 * N + 1
18:                 Debug.Print Vector(I)
19:                 Next I
20:                 End Sub
```

Поясним приведенный код. Сначала на уровне модуля объявляется глобальный динамический массив Vector (строка 1). В момент объявления размерность динамического массива не указывается, соответственно не выделяется память. Все это произойдет позже, в процессе выполнения программы. Далее приводится одна из возможных процедур, работающая с этим массивом Vector. В строке 12 массив переопределяется (увеличивается его размер) с сохранением предыдущих значений. Затем массив расширяется (цикл в строках 13-15). В последнем цикле (строки 17-19) значения элементов сформированного массива выводятся в окно отладки (Immediate).

В рассмотренном примере изменялся размер динамического массива, но не его размерность (массив оставался одномерным). Приведем фрагмент кода программы, в котором изменяются и размер, и размерность динамического массива:

```
1:  Sub sample22 ()
2:  Dim dArray ( ) As Variant
3:  ReDim dArray(1,2)
```

```
4:  dArray(0,0) = 2
5:  dArray(0,1) = 3
6:  k = dArray(0,0) + dArray (0,1)
7:  ReDim dArray(k)
8:  dArray(0) = "Строка1"
9:  End Sub
```

В этом примере массив dArray сначала определяется как двумерный массив из шести элементов (2×3) (строка 3), а затем переопределяется как одномерный массив, причем верхняя граница индекса задается значением k (строка 7).

Динамические массивы с успехом можно применять там, где необходимы динамические структуры данных, например списки, стеки, очереди.

ПОЛЬЗОВАТЕЛЬСКИЙ ТИП ДАННЫХ

VBA поддерживает возможность создавать пользовательские типы данных на основе ранее определенных типов. Такой тип в VBA называется User-defined type (UDT) -тип, определенный пользователем. Это соответствует понятиям типа данных record (запись) в языке Pascal или struct (структура) в языке C/C++. Для создания пользовательского типа предназначен оператор Type. Он позволяет на уровне модуля определить структуру данных, включающую другие разнородные, но логически связанные переменные различных типов. После описания типа на его основе можно создавать и использовать переменные.

Синтаксис оператора Type:

```
Type <имяТипа>
    <имяЭлемента1> As <тип>
    <имяЭлемента2> As <тип>
End Type
```

где:

<имяТипа> - имя пользовательского типа данных;

<имяЭлемента> - имя структуры, составляющей новый тип данных.

С помощью ключевых слов Private и Public можно задать область видимости создаваемого типа. Опции Private и Public указываются в строке объявления типа перед ключевым словом Type.

Листинг 21. Пример использования пользовательского типа

```
'Тип TStudent хранит информацию о студенте. Public Type
TStudent
    ID As Long 'идентификатор
    LastName As String 'фамилия
    FirstName As String 'имя
    MiddleName As String 'отчество
    BirthDay As Date 'дата рождения End Type
'Учебная группа Public Type
TGroup
    Num As String * 10 'номер группы
    Students() As TStudent 'список (массив) студентов End Type
'Объявления переменных
Private stud As TStudent ' студент
Public group As TGroup 'группа
```

Для обращения к элементам пользовательского типа (полям структуры) используется точечная нотация:

<имяПеременнойUDT>.<имяЭлемента>

Листинг22. Работа с переменными пользовательского типа

```
Sub sample20()
    ReDim group.Students(10)
    group.Num = "AC-1234"
    group.Students(0).LastName = "Петров"
    group.Students(0).FirstName = "Иван"
```

```
    Debug.Print group.Num & group.Students(0).LastName & " " &
group.Students(0).FirstName
End Sub
```

Широкие возможности, представляемые программисту пользовательским типом

имеют ограничение: все операции должны выполняться на уровне полей. Единственная

разрешенная операция - присваивание (листинг 23).

Листинг 23. Операции над пользовательским типом

```
Sub sample21()
        Dim group1 As TGroup, group2 As TGroup
        ReDim group1.Students(25)
        ReDim group2.Students(28)
        group1.Num = "AS-1234"
        group2.Num = "AS-5678"
        Debug.Print "1: "; group1.Num, group2.Num
        ' If group2 > group1 Then ... - Это вызовет ошибку
        If UBound(group2.Students) > UBound(group1.Students) Then ' Так можно
        group1 = group2 ' Так тоже можно
        End If
        Debug.Print "2: "; group1.Num, group2.Num
End Sub
```

ТЕМА 4. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ В VBA

ПОДДЕРЖКА ООП В VBA

VBA представляет средства для использования имеющихся и создания новых объектов. Класс - это общее описание однородных по структуре объектов. Класс задает характеристики и поведение объектов. Классы в VBA оформляются в виде отдельных модулей.

Концептуально, создание класса начинается с этапа проектирования, где определяются свойства и методы, которыми должны обладать объекты нового класса.

Рассмотрим следующий пример: пусть требуется создать класс, описывающий студента. Объект такого класса может представлять в виде свойств такую информацию о студенте, как его имя, отчество и фамилию, дату и место рождения, контактную информацию, пол, паспортные данные и т.п. В качестве методов можно указать возможность редактирования информации, возможность полного или, наоборот, частичного отображения информации о студенте.

Семантически, такое описание может быть представлено, например следующим образом:

```
// псевдокод описания объекта Студент Студент {
        Фамилия: строка
        Имя: строка
        Отчество: строка
        Дата_рождения: Дата
        Пол: символьный или логический
        Контактная_информация: строка }
Студент.Полное_имя: строка, только чтение
Студент.Полное_описание: строка, только чтение
Студент.Править_описание (Нов_Фамилия, Нов_Имя, Нов_Отчество, Нов_Дата_рожд,
_
        Нов_Пол, Нов_Конт_Инф)
```

Как видно, такое описание очень напоминает описание пользовательского типа данных. Однако отличие объектного типа в том, что не только данные (Фамилия, Имя и пр.), но и код (Студент.Полное_имя и пр.) размещены совместно (инкапсулированы).

Синтаксически, объявление класса в VBA выполняется в специализированной программной единице - модуле класса, куда помещается весь код создаваемого класса.

Отметим, что в VBA не существует специальных языковых конструкций, используемых при описании класса (кроме описания свойств), поэтому порядок создания класса удобнее рассмотреть по шагам, а затем прокомментировать:

1. В редакторе Visual Basic добавляем в проект новый модуль класса (меню "Вставка/ Модуль класса"). Откроется окно нового модуля класса.
2. В окне свойств (F4) задаем имя (Name) модуля (в нашем примере - CStudent). Оно же станет именем класса.
3. В разделе описаний модуля объявляем закрытые члены класса - обычные переменные уровня модуля (Private), которые будут определять значения свойств.
4. Инициализируем начальные значения свойств при помощи метода Class_Initialize (конструктора класса).
5. Определяем свойства для чтения (Property Let), для записи (Property Let) и объектные свойства (Property Set).
6. Создаем методы класса при помощи обычных процедур и функций.
7. Допустимо также создание метода Class_Terminate (деструктора класса) для удаления объекта из памяти по завершении работы с ним.

По этому алгоритму был создан ранее спроектированный класс CStudent, представляющий описание объекта "студент". Полный код модуля с комментариями приведен в листинге 24.

Листинг 24. Модуль класса

```
' Закрытые члены класса, прямой доступ к ним невозможен -
' только через свойства
Private fLastName As String ' фамилия
Private fFirstName As String ' имя
Private fMiddleName As String ' отчество
Private fBirthDay As Date ' дата рождения
Private fGender As String * 3 ' пол
Private fContacts As String ' адрес и телефон

' Конструктор класса. Вызывается в момент создания объектной переменной
Private Sub Class_Initialize()
```

```
        fLastName = "Фамилия не указана"
        fFirstName = "Имя не указано"
        fMiddleName = "Отчество не указано"
        fContacts = "Адрес не указан" End Sub
' Свойства для чтения
Public Property Get LastName() As String
        LastName = fLastName End Property
Public Property Get FirstName() As String
        FirstName = fFirstName End Property
Public Property Get MiddleName() As String
        MiddleName = fMiddleName End Property
Public Property Get BirthDay() As Date
        BirthDay = fBirthDay End Property
Public Property Get Gender() As String
        Gender = fGender End Property
Public Property Get Contacts() As String
        Contacts = fContacts End Property
' Свойства для записи
Public Property Let LastName(ByVal NewValue As String)
        fLastName = NewValue End Property
Public Property Let FirstName(ByVal NewValue As String)
        fFirstName = NewValue End Property
 Public Property Let MiddleName(ByVal NewValue As String)
        fMiddleName = NewValue End
Property
Public Property Let BirthDay(ByVal NewValue As Date)
        fBirthDay = NewValue End Property
```

```
Public Property Let Gender(ByVal NewValue As String)
        fGender = NewValue End
Property

Public Property Let Contacts(ByVal NewValue As String)
        fContacts = NewValue End Property

' Свойства ТОЛЬКО ДЛЯ ЧТЕНИЯ (нет парных Property Let) Public Property Get
FullName() As String
        FullName = fLastName & " " & fFirstName & " " & fMiddleName End Property
Public Property Get FullInfo() As String
        FullInfo = fLastName & Chr(9) & fFirstName & Chr(9) & fMiddleName & Chr(9) _
                & CStr(fBirthDay) & Chr(9) & fGender & Chr(9) & fContacts
End Property

' Метод для изменения свойств объекта.
' Все параметры объявлены как необязательные (Optional).
' Это позволяет изменять только необходимые поля класса.
' Изменять поля можно и через соответствующие свойства для записи (Property Let),
' этот метод приведен для примера

Public Sub EditInfo(Optional LastName As String, Optional FirsName As String,
_
                Optional MiddleName As String, Optional BirthDay As Date,
                Optional Gender As String, Optional Contacts As String)

 fLastName = LastName fFirstName =
FirstName fMiddleName = MiddleName
fBirthDay = BirthDay fGender = Gender
fContacts = Contacts End Sub
```

Дополнительные пояснения к коду листинга 24 в части синтаксиса:

1. Синтаксис описания *членов класса* такой же, как и для обычных переменных. Использование Private позволяет ограничить доступ к членам класса из других модулей и, тем самым, обеспечить целостность данных.

2. *Конструктор* Class_Initialize предназначен для задания начальных значений переменным - членам класса и выполнения других операций инициализации. Он вызывается автоматически - в момент создания экземпляра класса.
3. *Свойства* класса объявляются с ключевым словом Property. При этом функция Property Get создает свойство для чтения, а процедура Property Let служит для записи значений в свойства базовых типов (в т.ч. массивов и пользовательских типов). Процедура Property Set служит для присваивания значений членам объектного типа (см. листинг 25).

Общий синтаксис объявления свойств:

```
'для чтения
[Public | Private] [Static] Property Get <имяФункции>
[(<списокАргументов>)] [As <тип>]
        <операторы>
        имяФункции = возвращаемое значение
        [Exit Property]
        <операторы>
        имяФункции = возвращаемое значение End Property

'для записи
[Public | Private] [Static] Property Let <имяПроцедуры>
[(<списокАргументов>)]
        <операторы>
        [Exit Property]
        <операторы> End Property

'для работы с объектными членами класса
 [Public | Private] [Static] Property Set <имяПроцедуры> [(<списокАргументов>)]
        <операторы>
        [Exit Property]
    <операторы> End Property
```

4. *Методы класса* - это обычные процедуры и функции VBA, которые объявляются в модуле класса и не могут быть использованы самостоятельно.

VBA поддерживает один из основных принципов ООП, *наследование,* косвенным путем - через встраивание объектов. Т.е. нет прямой возможности создавать классы-потомки на основе ранее созданных классов, но можно объявлять членами класса переменные объектных типов. Пример встраивания приведен в листинге 31, где на основе классов CStudent (Студент) и CFaculty (Факультет) создается класс CGroup, описывающий учебную группу указанного факультета.

Листинг 25. Встраивание

```
' Модуль класса CFaculty - объект "Факультет" (приведен фрагмент кода)

Private fTitle As String ' Название факультета Public Property Get Title() As String

        Title = fTitle End Property

Public Property Let Title(NewTitle As String)

        fTitle = NewTitle End Property

...


' Модуль класса CGroup - объект "Учебная группа" (приведен фрагмент кода) '

' Встраивание объектных членов класса Private fFaculty As New CFaculty Dim fStudents() As New CStudent

...
' Использование Property Set для задания значения объектному члену

 Public Property Set Faculty(Title)

        Set fFaculty = Title End Property

' Чтение названия факультета

Public Property Get Faculty() As String

        Faculty = fFaculty.Title End Property

...
```

Поскольку класс - не более чем специфичный тип данных, то для его использования в программе требуются переменные, представляющие экземпляры этого класса. Такие переменные, называемые объектными, создаются одним из способов:

1. явным указанием класса объекта;
2. ссылкой на ранее созданный объект.

При любом способе создания объектная переменная представляет из себя 4-байтовую ссылку на адрес, где хранится объект. При объявлении такой переменной память для самого объекта может и не отводиться, поэтому может быть не определено и значение ссылки. Задание ссылки на объект, т.е. связывание объектной переменной с самим объектом выполняется двумя способами:

1. При *раннем связывании* в момент объявления указывается класс объекта:

   Dim <Переменная> As <классОбъекта>

   Это позволяет еще на этапе трансляции проверять, допустимы ли те или иные операции над создаваемыми объектами.

2. При *позднем связывании* переменная объявляется так:

   Dim <Переменная> As Object

Это объявление говорит о том, что переменная является объектом (ссылкой), но ничего не сказано о классе этого объекта. Он выяснится только динамически при выполнении программы, когда <Переменная> будет связываться с только что созданным или существующим объектом того или иного класса. Поэтому такое связывание и называется поздним, или динамическим.

В целом же, объявление объектных переменных отличается от обычных только указанием ключевого слова New, с помощью которого создаваемому экземпляру класса (при раннем связывании) выделяется память и вызывается его конструктор. Объектные переменные могут быть объявлены и использованы в любых модулях (как в стандартных, так и модулях класса).

Общий синтаксис объявления объектной переменной:

Private | Public | Dim <имяОбъектнойПеременной> As New <имяКласса>

Например:

```
Public Faculty As New CFaculty Private Groups(3) As
New CGroup Dim stud As New CStudent
```

Использование раннего связывания имеет одно преимущество: явное указание класса позволяет получить доступ к его свойствам и методам уже на этапе разработки в VBE. Это выражается в том, что при введении имени объектной переменной появляется всплывающий список доступных операций над объектом.

ИСПОЛЬЗОВАНИЕ ОБЪЕКТОВ

Для обращения к свойствам или методам экземпляра класса в VBA используется точечная нотация:

```
перем = <имяОбъектнойПеременной>.<Свойство> ' чтение свойства Или:
<имяОбъектнойПеременной>.<Свойство> = значение ' запись свойства
```

Для обращения к объектным свойствам следует использовать ключевое слово Set.

Вызов методов и передача параметров аналогичны работе с обычными процедурами и функциями. Примеры работы с экземплярами созданных классов - в листинге 26.

Листинг 26. Работа с объектами

```
' Создание экземпляра класса и работа с его свойствами и методами '
Sub sample30()
    Dim stud As New CStudent ' экземпляр класса CStudent
    ' Обращения к свойствам объекта
    stud.FirstName = "Иван"
    stud.LastName = "Петров"
    stud.Contacts = "г.Омск, пр.Мира, 11, к.8. т/ф (3812) 65-96-11"
    stud.BirthDay = #4/27/1990#
    ' Печать полной информации о студенте, полученной из свойства FullInfo
    Debug.Print stud.FullInfo
```

```
' Редактирование некоторой информации о студенте
stud.EditInfo LastName:="Сидоров", Gender:="муж", BirthDay:=#5/27/1990#
Debug.Print stud.FullInfo End Sub
'
' Создание и использование экземпляра класса с объектными свойствами '
Sub sample31()
        Dim group As New CGroup
        Dim Faculty As New CFaculty
        Faculty.Title = "Информационные системы"
        ' Задание значения свойству объектного типа
        Set group.Faculty = Faculty
        Debug.Print group.Faculty End Sub
```

Использование точечной нотации при работе с объектами сложной структуры (а равно и пользовательскими типами данных) приводит к определенным неудобствам при написании кода и снижению его читабельности. Для решения этой проблемы в VBA поддерживается оператор With… End With, который имеет следующий формат:

```
With <объект> <блок
операторов> End With
```

Действие оператора With состоит в том, что он задает <объект>, над которым выполняются все вложенные операторы. При этом не нужно каждый раз повторять ссылку на объект. В результате текст программы становится короче, понятнее и, самое главное, эффективнее. Так, процедура sample 30 из листинга 26 при использовании этого оператора будет выглядеть так:

```
Sub sample30()
Dim stud As New CStudent ' экземпляр класса CStudent With stud
        .FirstName = "Иван"
        .LastName = "Петров"
        .Contacts = "г.Омск, пр.Мира, 11, к.8. т/ф (3812) 65-96-11"
        .BirthDay = #4/27/1990#
```

```
            Debug.Print .FullInfo

            .EditInfo LastName:="Сидоров", Gender:="муж",
BirthDay:=#5/27/1990#

            Debug.Print .FullInfo

        End With

End Sub
```

ТЕМА №5. ОБЪЕКТНАЯ МОДЕЛЬ КОМПОНЕНТОВ MS OFFICE. БИБЛИОТЕКИ ТИПОВ

На внутреннем уровне пакет MS Office представлен в виде совокупности взаимодействующих объектов. Каждый из них наделен специфичным набором свойств и методов. Совокупность объектов и связей между ними называется **объектной моделью.** Любое приложение MS Office имеет свою объектную модель. В виде объектов представлены все доступные разработчику элементы офисных программ вплоть до приложения как такового (объект Application). Знание структуры объектных моделей Office позволяет создавать профессиональные приложения, выполняющие необходимую обработку данных и подготовку документов.

Компоненты объектной модели каждого приложения Microsoft Office – объекты и семейства – размещаются в одноименных библиотеках (файлы с расширением .old). Существуют стандартная библиотека объектов VBA и стандартная библиотека Office, библиотеки объектов Word, Excel, Access и прочие объектные библиотеки, предоставляющие различные функциональные возможности.

Чтобы просмотреть список объектных библиотек, доступных в конкретном приложении Microsoft Office (например, в Word), необходимо в редакторе VBA выбрать команду меню "Вид/Просмотр Объектов" и раскрыть список "Project/Library". На рис. 5.1.1 показан раскрытый список объектных библиотек, доступных в Microsoft Word[1].

[1] В зависимости от версии пакета, варианта установки и набора установленных офисных компонентов список доступных библиотек может несколько отличаться от приведенного

РИСУНОК 6. ***СПИСОК ОБЪЕКТНЫХ БИБЛИОТЕК, ДОСТУПНЫХ В MICROSOFT WORD***

Коротко охарактеризуем эти библиотеки.

Word - библиотека, задающая основу документов Word. Здесь хранится класс, задающий корневой объект Word.Application, классы объектов, вложенных в корневой объект.

Office - библиотека объектов, общих для всех приложений Microsoft Office. Здесь находятся CommandBar - классы, определяющие инструментальные панели, и классы других общих объектов. Здесь же находятся классы, задающие "помощник" (объект Assistant) и все классы, связанные с ним.

Stdole - библиотека классов, позволяющая работать с ActiveX-объектами (OLE-объектами) и функциями OLE-автоматизации (OLE-Automation).

VBA - библиотека классов, связанных с языком VBA. Здесь хранятся все стандартные функции и константы, встроенные в язык, классы Collection, Err и прочие.

Project - проект по умолчанию, связанный с документом. Классы, которые могут программистом создаваться в этом проекте, методы, свойства - все это доступно для просмотра так же, как и объекты классов, встроенных в стандартные библиотеки.

Normal - проект, доступный для всех документов Word. Здесь могут храниться функции и классы, используемые всеми документами Word.

БИБЛИОТЕКА VBA. ОБЪЕКТЫ ОБЩЕГО НАЗНАЧЕНИЯ

КОЛЛЕКЦИИ VBA

Коллекции VBA - это упорядоченные наборы элементов, представляющие собой единое целое. Любая коллекция VBA - экземпляр класса Collection. Все элементы коллекции (члены) представлены ссылками на фактические объекты. Это позволяет объединять в коллекцию данные различных типов. Коллекция - это списочная структура, поддерживающая операции создания коллекции как таковой, добавления и удаления элементов, получения элемента по его индексу.

В коллекцию можно добавлять элементы методом Add, удалять ранее добавленные элементы методом Remove и обращаться к элементам методом Item, в т.ч. и итеративно, например, с использованием инструкции For Each...Next.

В качестве примера приведем коллекцию UserForms, элементами которой являются все загруженные формы UserForm приложения. Как и все коллекции, UserForms имеет свойства Count и Item, а также метод Add. Свойство Count возвращает количество загруженных форм. Свойство Item возвращает определенный компонент коллекции, т.е. форму. Метод Add добавляет в коллекцию новую форму.

Еще один пример - набор всех элементов управления, размещенных на пользовательской форме UserForm, представляемый коллекцией Controls. Фрагмент кода, приведенный в листинге 27, устанавливает значение свойства Visible каждого элемента коллекции Controls в False (делает элемент невидимым на форме):

Листинг 27. Использование коллекций

```
' Подразумевается, что форма UserForm1 добавлена в проект и загружена ' в процессе
выполнения приложения

-
For Each c in UserForm1.Controls

    c.Visible = False Next

-
```

ОБЪЕКТ DEBUG

Объект Debug направляет вывод приложения в окно отладки (Immediate) во время выполнения. Этот объект был использован в ряде предыдущих примеров. Объект Debug поддерживает 2 метода:

- Print - безусловный вывод указанного выражения в окно отладки;
- Assert - вывод по заданному условию.

ОБЪЕКТ ERR

Объект Err - содержит информацию об ошибках времени выполнения. Ошибки такого рода генерируются системными функциями VBA, либо программистом в коде программы.

При возникновении ошибки времени выполнения (run-time error), свойства объекта Err принимают значения, уникальным образом идентифицирующие эту ошибку и используемые для ее обработки.

Для генерации ошибки программным путем используется метод Raise. Этот метод, совместно с командой Error, используется для генерации системных ошибок в модулях классов. В прочих программных модулях этот метод позволяет генерировать пользовательские ошибки. Основное свойство этого объекта - код ошибки (Number).

Все свойства объекта Err сбрасываются при выходе из подпрограммы, вызвавшей ошибку, кроме случаев, когда инструкция Resume (инструкция передачи управления после обработки ошибки) находится вне блока обработки.

Метод Clear используется для принудительного сброса свойств объекта Err.

Пример использования объекта Err приведен на рис. 7

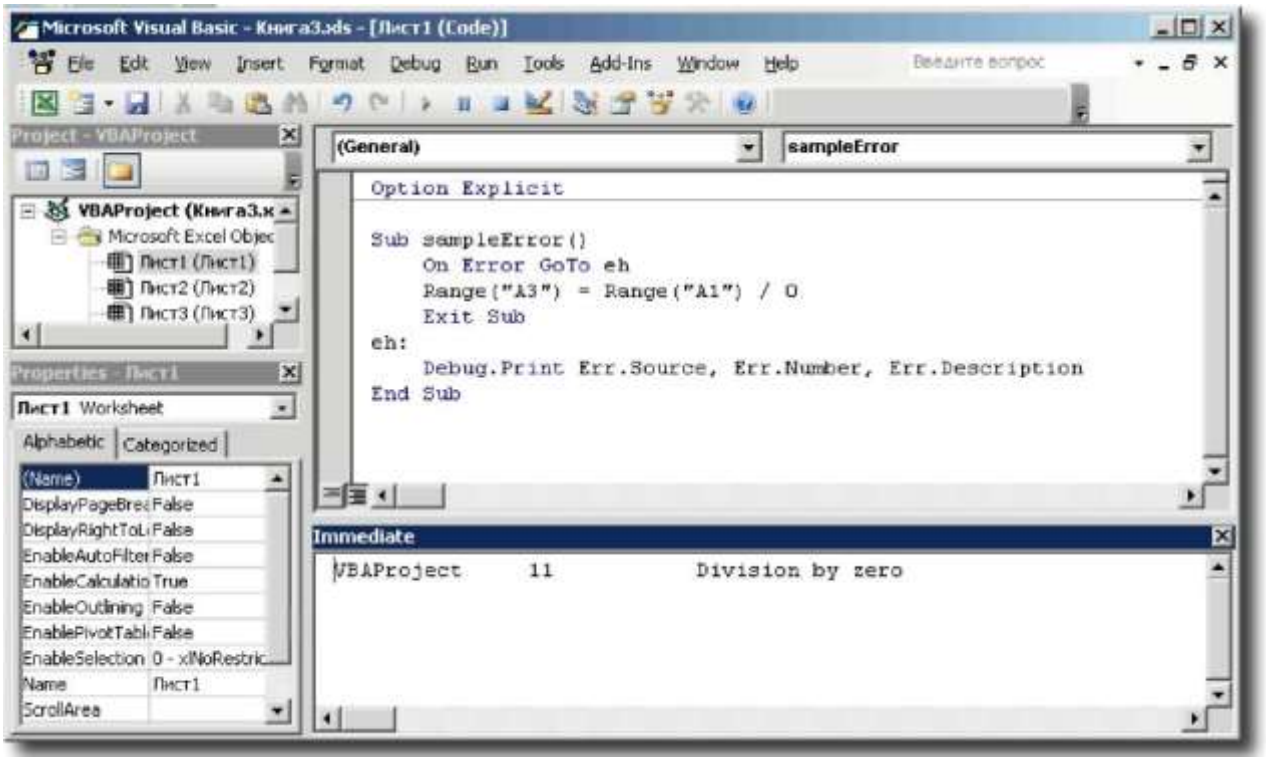


РИСУНОК 7. **ПРИМЕР ИСПОЛЬЗОВАНИЯ ОБЪЕКТА ERR**

ОБЪЕКТНАЯ МОДЕЛЬ EXCEL

Приложение MS Exel представляет разработчику около полусотни объектов и коллекций, доступных разработчику. Подробное описание модели доступно в справочной системе, здесь кратко приведем только некоторые из объектов Excel.

ОБЪЕКТ WORKBOOK

Объект Workbook представляет доступ к открытой рабочей книге Excel. Этот объект является элементом коллекции Workbooks. У объекта имеется множество свойств и методов, основными являются Sheets — коллекция рабочих листов, Worksheet — определенный рабочий лист, Range — диапазон ячеек, Selection — текущее выделение, CellFormat — формат ячеек. Пример использования некоторых объектов рабочей книги приведен в листинге 27.

Листинг 27. Программирование объектов рабочей книги

```
Sub ChangeCellFormat

    Range(“A1”).Select Selection.Interior.ColorIndex =
    36 MsgBox “Ячейка A1 залита жетым цветом” With
    Application
```

```
.FindFormat.Interior.ColorIndex = 36

.ReplaceFormat.Interior.ColorIndex = 36 End With ActiveCell.Replace
What:="", Replacement:="", LookAt:=xlPart, _

SearchOrder:=xlByRows, MatchCase:=False, SearchFormat:=True, _

ReplaceFormat:=True MsgBox "Ячейка A1 залита
зеленым цветом" End Sub
```

ТЕМА №6. РАЗРАБОТКА ПРИЛОЖЕНИЙ ДЛЯ MS OFFICE

Для разработчика на VBA имеется возможность изменять внешний вид офисных приложений. Это реализуется через свойства и методы таких встроенных объектов как, наприемр, CommandBar

КОЛЛЕКЦИЯ COMMANDBARS

Элементами коллекции CommandBars являются объекты CommandBar, которые представляют собой панель команд. Таким образом, коллекция CommandBars содержит все панели команд приложения.

Например, чтобы создать новую панель команд, можно выполнить следующую процедуру:

```
Set myBar = CommandBars.Add(Name:="Личная",Position:=msoBarFloating)
myBar.visible = True
```

СВОЙСТВА КОЛЛЕКЦИИ COMMANDBARS

Свойство	Описание
Application	Позволяет сослаться на активный объект Application и его свойство. Каждый объект и коллекция в MS Office имеют данное свойство
ActionControl	Возвращает объект CommandBarControl, значение свойства OnAction которого определяет запускаемую процедуру. Если процедура не была инициирована элементом управления панели команд, то это свойство возвращает значение Nothing (пусто). Данное свойство можно использовать, например, для проверки того, какая из кнопок на панели инструментов была нажата
ActiveMenuBar	Возвращает объект CommandBar, который представляет собой активную строку меню приложения

Count	Возвращает количество элементов коллекции (строк меню, панелей инструментов и контекстных меню)
DisplayFonts	Определяет способ отображения шрифта в списке Шрифт (Font). Если установлено значение True, то элементы списка отображаются тем шрифтом, который они и представляют
DisplayToolTips	Определяет, отображаются ли экранные подсказки элементов управления панели команд: Да (True) или Нет (False). Обратите внимание, что в случае установки данного свойства в одном из приложений MS Office оно будет оказывать немедленное воздействие на все открытые приложения MS Office, а также на все приложения MS Office, запущенные после этого
Item	Возвращает объект CommandBar, являющийся элементом коллекции CommandBars, или объект CommandBarControl из коллекции
	CommandBarControls. Синтаксис: Expression.Item(Index) где Expression – обязательный элемент, представляющий выражение, которое возвращает объект CommandBar или объект CommandBarControl; Index – обязательный элемент типа Variant, который определяет имя или порядковый номер возвращаемого объекта в коллекции CommandBars
LargeButtons	Определяет размер отображаемых кнопок панели инструментов: большего размера, чем нормальный Да (True) или Нет (False)
MenuAnimationStyle	Определяет тип анимации панели команд. Значение данного свойства должно равняться значению одной из констант, указанных после таблицы

МЕТОДЫ КОЛЛЕКЦИИ COMMANDBARS

ADD

Метод Add позволяет создать панель команд, которая автоматически добавляется в

коллекцию CommandBars. Данный метод возвращает объект CommandBar.

Синтаксис метода Add:

Expression.Add (Name, Position, MenuBar, Temporary)

Expression - обязательный элемент - выражение, которое возвращает объект

ComrnandBar;

Name - имя создаваемой панели команд. Необязательный элемент типа Variant. Если данный аргумент отсутствует, то панели команд будет присвоено имя по умолчанию, например, "Настраиваемая 1";

Position - расположение или тип создаваемой панели команд. Необязательный элемент типа Variant.

MenuBar - необязательный элемент типа Variant. Если данному параметру задать значение True, то активную панель меню можно поменять на создаваемую панель команд. По умолчанию значение данного аргумента равняется False;

Temporary - необязательный элемент типа Variant. Если задать данному параметру значение True, то создаваемая панель команд будет временной и после закрытия приложения будет удалена. По умолчанию значение данного аргумента равняется False.

FINDCONTROLS

Метод FindControls возвращает коллекцию CommandBarControls, которая удовлетворяет определенным критериям.

Синтаксис метода FindControls:

Expression.FindControls(Type, Id, Tag, Visible)

Expression – обязательный элемент, который представляет собой выражение, возвращающее коллекцию Command Bars.

RELEASEFOCUS

Данный метод убирает фокус со всех панелей команд.

Синтаксис метода ReleaseFocus:

Expression.ReleaseFocus

Элемент синтаксиса Expression представляет собой выражение, которое возвращает

объект CommandBars.

ОБЪЕКТ COMMANDBAR

Объект CommandBar предоставляет доступ к панелям команд. Все объекты CommandBar являются членами коллекции CommandBars.

СВОЙСТВА ОБЪЕКТА COMMANDBAR

Свойство	Описание
AdaptiveMenu	Определяет, является ли меню адаптивным (True) или нет (False)
Application	Возвращает объект Application, который является контейнером (container application) для данного объекта
Building	Если данное свойство панели команд или элемента управления равно True, то данная панель является встроенной. В противном случае (False) – это специальная панель команд, элемент управления или встроенный элемент управления, свойство OnActian которого имеет некоторое значение
Controls	Возвращает коллекцию CommandBarControls, которая содержит все элементы управления панели команд или всплывающего элемента управления
Enable	Данное свойство имеет значение True, если панель команд или элемент управления панели команд доступны (в случае панелей команд это аналогично выбору панели в списке доступных панелей). Для встроенных панелей команд присвоение значения True означает, что само приложение определяет, будет ли доступна панель команд. Если значение свойства Enabled равно False, то панель команд будет недоступной
Height	Определяет высоту панели команд в пикселях
Index	Возвращает номер объекта в коллекции
Left	Определяет расстояние (в пикселях) от левого края панели команд или элемента управления панели команд до левого края экрана. Для закрепленных панелей команд это свойство определяет расстояние от панели команд до левого края закрепленной области
Name	Определяет имя панели команд. В случае встроенной панели команд данное свойство возвращает имя, определенное в американской версии приложения (U.S. English)
Parent	Возвращает объект-родитель объекта
Position	Определяет позицию панели команд. Может принимать значения одной из следующих встроенных констант: msoBarLeft, msoBorTop, msoBarRight, msoBarBottom, msoBarFloating, msoBarPopup или msoBarMenuBar

Protection	Определяет способ защиты панели команд от пользовательской настройки. Данное свойство может принимать значение одной из следующих констант: msoBarProtection, msoBarNoProtection, msoBarNoCustomize, msoBarNoResize, msoBarNoMove, msoBarNoChangeVisible, msoBarNoChangeDock, msoBarNoVerticalDock или msoBarNoHorizontalDock
RowIndex	Определяет порядок закрепления панели команд относительно других панелей команд в области стыковки. Данное свойство может быть положительным числом типа Integer или одной из констант: msoBarRowFirst или msoBarRowLast
Top	Определяет расстояние (в пикселях) от верхнего края панели команд или элемента управления панели команд до верхнего края окна. Для закрепленных панелей команд это свойство определяет расстояние от панели команд до верхнего края области стыковки. Для объекта CommandBarControl данное свойство доступно только для чтения
Type	Определяет тип панели команд. Значением данного свойства может быть одна из констант: msoBarTypeNormal – панель инструментов; msoBarTypeMenuBar – строка меню; msoBarTypePopup – контекстное меню
Visible	Позволяет настроить атрибут видимости панели команд или элемента управления панели команд. Для вновь созданной панели команд это свойство по умолчанию равно False. Перед заданием свойству Visible значения True свойство Enabled панели команд должно иметь значение True
Width	Определяет ширину панели команд или элемента управления панели команд в пикселях

МЕТОДЫ ОБЪЕКТА COMMANDBAR

МЕТОД DELETE

Метод Delete удаляет элемент управления из коллекции CommandBars. Данный метод имеет два варианта синтаксиса:

Expression.Delete Expression.Delete
(Temporary)

Expression - обязательный элемент - выражение, которое возвращает объекты

CommandBar (Синтаксис 1) или CommandBarControl (Синтаксис 2);

Temporary - необязательный элемент типа Boolean. Если значение данного параметра равно True, то элемент управления удаляется в текущем сеансе работы, но в следующем будет снова отображен.

МЕТОД FINDCONTROL

Метод FindControl возвращает объект CommandBarControl, который удовлетворяет определенным критериям, задаваемым параметрами метода FindControl. Синтаксис этого метода имеет вид:

```
Expression.FindControl(Type, Id, Tag, Visible, Recursive)
```

Expression - обязательный элемент - выражение, которое возвращает объект CommandBar;

Type - необязательный элемент, определяющий тип элемента управления. Значение данного аргумента может равняться одной из констант, определяющих тип элемента управления;

Id - необязательный элемент типа Variant, выполняющий функцию идентификатора элемента управления;

Tag - необязательный элемент типа Variant, который задает значение свойства Tag элемента управления;

Visible - необязательный элемент типа Variant. По умолчанию данное значение равно False. При задании данному параметру значения True в поиск включаются только видимые элементы управления панели инструментов. Видимые панели команд включают все видимые элементы панели инструментов и любые меню, которые открыты во время выполнения метода FindControl;

Recursive - необязательный элемент типа Boolean. Определяет, включаются (True) или нет (False) в поиск всплывающие панели команд, расположенные на панели команд, к которой применяется метод FindControl. По умолчанию используется значение False.

МЕТОД RESET

Метод Reset сбрасывает настройки встроенных панелей команд к исходным настройкам, установленным по умолчанию, или устанавливает оригинальное функционирование и внешний вид элементов управления панели команд. Синтаксис метода Reset:

Expression.Reset

Элемент синтаксиса Expression метода Reset представляет собой выражение, которое возвращает один из следующих объектов: CommandBar, CommandBarControl, CommandBarButton, CommandBarPopup или CommandBarComboBox.

МЕТОД SHOWPOPUP

Метод ShowPopup выводит на экран панель команд как контекстное меню в заданных координатах или в текущей позиции указателя. Если свойство Position команд имеет значение, отличное от значения константы msoBarPopup, то этот метод не выполняется. Синтаксис метода ShowPopup:

Expression.ShowPopup(X, Y)

Expression - обязательный элемент, представляющий собой выражение,

возвращающее объект CommandBar;

X - необязательный элемент типа Variant, задающий координату расположения контекстного меню по горизонтали. В случае отсутствия этого параметра используется текущая координата указателя;

Y - необязательный элемент типа Variant, задающий координату расположения контекстного меню по вертикали. Если этот параметр отсутствует, то используется текущая координата указателя.

КОЛЛЕКЦИЯ COMMANDBARCONTROLS

Коллекция CommandBarControLs предоставляет доступ ко всем элементам управления панелей команд. Чтобы получить доступ к коллекции CommandBarControLs конкретного объекта CommandBar, необходимо использовать свойство Controls этого объекта.

Для добавления на панель команд нового элемента управления используют метод Add этой коллекции.

В листинге 28 приведен пример создания панели инструментов с последующим добавлением на нее кнопки.

Листинг 28. Создание панели инструментов с добавлением на нее кнопки

```
Sub Examp()

    Dim MyBar As CommandBar

    Dim MyButton As CommandBarButton

    Set MyBar = CommandBars.Add()

    Set MyButton = MyBar.Controls.Add(msoControlButton)

    MyBar.Visible = True End
Sub
```

В приведенном примере объявляются две переменные: MyBar типа CommandBar и MyButton типа CommandBarButton. Затем им присваиваются ссылки на созданную панель инструментов и кнопку панели инструментов соответственно. Последняя инструкция в этой процедуре делает созданную панель инструментов видимой.

Чтобы сослаться на конкретный элемент управления панели команд, являющийся членом коллекции CommandBarControls, можно использовать следующую инструкцию:

```
Set myControl = CommandBars(1).Controls(1)
```

Здесь для ссылки на конкретный элемент управления используется свойство Controls объекта CommandBar.

ОБЪЕКТ COMMANDBARCONTROL

Объект CommandBarControl предоставляет доступ ко всем элементам управления панели команд. Каждый объект CommandBarControl является элементом коллекции CommandBarControls.

Все свойства и методы, которые имеет объект CommandBarControl, также имеют и объекты CommandBarButton, CommandBarComboBox и CommandBarPopup.

В случае объявления в программе переменной типа CommandBarControl, ей можно присвоить ссылку на объекты CommandBarButton, CommandBarComboBox и CommandBarPopup.

СВОЙСТВА ОБЪЕКТА COMMANDBARCONTROL

Свойство	Описание
BeginGroup	Позволяет начать группу элементов управления на панели команд, начиная с данного, при задании данному свойству значения True
Caption	Определяет текст заголовка элемента управления панели команд. Для свойства TooltipText значение данного свойства является значением по умолчанию
Index	Возвращает порядковый номер объекта в коллекции. Первый элемент управления панели команд в коллекции CommandBarControls имеет номер 1. Разделители не включаются в коллекцию CommandBarControls
OnAction	Определяет имя макроса или функции Visual Basic, которая будет выполнена при нажатии на элементе управления или изменении его значения
Parameter	Определяет строку, которую может использовать приложение для выполнения команды при нажатии на элементе управления панели команд. Эта строка может быть, например, параметром функции, определенной свойством OnAction. С помощью данного свойства можно изменить поведение встроенного элемента управления. Для этого ему необходимо присвоить соответствующее значение
Priority	Определяет приоритет элементов управления панелей команд в том случае, если элементы управления не помещаются на одной строке. Допустимыми значениями данного свойства являются числа в диапазоне от 0 до 7. Если значение этого свойство равно 1, то элемент управления не может быть удален с панели команд. При этом все остальные значения данного свойства игнорируются. Обратите внимание, что данное свойство используется только для элементов управления панелей инструментов и не применимо для пунктов меню
TooltipText	Определяет текст экранной подсказки, отображаемой для элемента управления панели команд. По умолчанию в качестве значения данного свойства используется значение свойства Caption
Type	Определяет тип элемента управления панели команд. Значение данного свойства должно равняться одной из констант, определяющих тип элемента управления (msoControlType)

МЕТОДЫ ОБЪЕКТА COMMANDBARCONTROL

Объект CommandBarControl имеет шесть методов:

1. Copy.
2. Delete.
3. Execute.
4. Move.
5. Reset.
6. SetFocus.

Методы Delete и Reset уже были описаны при рассмотрении объекта CommandBar.
Метод Copy копирует элемент управления с одной панели команд на другую.
Синтаксис метода Copy:

Expression.Copy(Bar, Before)

Expression - обязательный элемент, являющийся выражением, которое возвращает один из следующих объектов: CommandBarControl, CommandBarButton, CommandBarPopup или CommandBarComboBox;

Bar - необязательный элемент - объект CommandBar, идентифицирующий панель команд, на которую копируется элемент управления. Если данный аргумент отсутствует, то элемент управления копируется на ту панель команд, где он находится;

Before - необязательный элемент - число, которое указывает позицию копируемого элемента управления на панели команд. Скопированный элемент управления будет вставлен перед элементом управления в позицию, определяемую данным аргументом. Если значение данного аргумента не задано, то создаваемый элемент управления располагается в конце панели команд.

Метод Execute выполняет процедуру или встроенную команду, присвоенную элементу управления панели команд. Для специальных элементов управления запускаемая процедура определяется значением свойства OnAction. Синтаксис метода Execute:

Expression.Execute

Здесь Expression - это выражение, которое возвращает один из следующих объектов: CommandBarControl, CommandBarButton, CommandBarPopup или CommandBarComboBox.

Метод Move перемещает элемент управления в пределах одной и той же панели команд или на другую панель команд. Синтаксис метода Move:

Expression.Move(Bar, Before)

Этот метод имеет те же элементы синтаксиса, что и рассмотренный ранее Copy.

Метод SetFocus устанавливает фокус клавиатуры на элемент управления панели команд, если последний является видимым и доступным. Синтаксис метода SetFocus:

Expression.SetFocus

ОБЪЕКТ COMMANDBARBUTTON

Объект CommandBarButton предоставляет доступ к кнопкам на панели команд.

СВОЙСТВА ОБЪЕКТА COMMANDBARBUTTON

Объект CommandBarButton, имеет все свойства, присущие объекту CommandBarControl, а также ряд специфичных свойств, представленных в таблице.

Определяет идентификатор внешнего вида кнопки панели команд. Это свойство определяет только внешний вид кнопочного элемента управления

Свойство	Описание
FaceId	на панели команд, а выполняемое с помощью нее действие определяется свойством Id объекта CommandBarControl. Для кнопки с внешним видом, определяемым пользователем, значение данного свойства равно 0
ShortcutText	Определяет название горячей клавиши, которое отображается в меню, подменю или контекстных меню. Это свойство можно задать только в том случае, если ранее установлено свойство OnAction
Style	Определяет способ отображения кнопки на панели команд. Значение данного свойства может равняться одной из следующих констант: msoButtonAutomatic, msoButtonIcon, msoButtonCaption, msoButtonIconAndCaption, msoButtonIconAndCaptionBelow, msoButtonIconAndWrapCaption, msoButtonIconAndWrapCaptionBelow или msoButtonWrapCaption

МЕТОДЫ ОБЪЕКТА COMMANDBARBUTTON

Кроме методов объекта CommandBarControl, объект CommandBarButton имеет ряд специальных методов. Метод CopyFace - копирует внешний вид кнопки в буфер обмена. А вызов метода PasteFace позволяет вставить содержимое буфера обмена в кнопочный элемент управления.

ТЕМА №7. ФОРМЫ И КОМПОНЕНТЫ УПРАВЛЕНИЯ. ОБРАБОТКА СОБЫТИЙ

ВИЗУАЛЬНОЕ ПРОГРАММИРОВАНИЕ

В конце XX века широкое распространение получило **визуальное программирование** - технология, предоставляющая программисту наглядные средства конструирования интерфейса. Объектно-ориентированное программирование удачно использует концепцию визуального программирования.

VBA - это интегрированная среда разработки, которая предоставляет программисту возможность создания форм, на которых размещают компоненты (в терминах VBA -controls, элементы управления), имеющиеся в библиотеке VBA или созданные пользователем. Все компоненты (формы, элементы управления, меню и панели инструментов) являются объектами со своими свойствами и методами и способны реагировать на определенные события.

Компоненты могут быть:

- визуальными - т.е. видимыми при работе приложения; немедленно отображаются на экране при проектировании в таком же виде, в каком их увидит пользователь во время выполнения приложения;
- не визуальными - отображаются на форме в процессе проектирования в виде значка, но пользователю во время выполнения программы не видны; выполняют некоторые служебные функции.

Использование визуального проектирования интерфейса предоставляет программисту (пользователю) следующие преимущества:

- можно легко изменять размеры и расположение компонентов на форме (с помощью простых манипуляций мышью);
- в процессе проектирования постоянно виден результат - изображение формы и

расположенных на ней компонентов (не надо запускать приложение для проверки внешнего вида окна и последующего изменения программного кода для подбора более удачного размера и расположения компонентов);

- (основное) во время проектирования формы и размещения на ней компонентов редактор кода автоматически генерирует код программы, включая в нее фрагменты, описывающие данный компонент (далее можно изменять свойства компонентов и писать обработчики событий).

Визуальное проектирование приложения состоит из следующих этапов:

- создание пользовательской формы;
- размещение на созданной форме нужных компонентов (элементов управления);
- задание определенных свойств этих компонентов;
- написание, при необходимости, обработчиков событий.

Рассмотрим каждый из этапов подробнее.

ПОЛЬЗОВАТЕЛЬСКИЕ ФОРМЫ

Пользовательская форма в VBA создается добавлением в проект объекта UserForm, являющегося основой пользовательского диалогового окна. Объект UserForm – это пустое диалоговое окно. Настройку диалогового окна можно выполнить добавлением к объекту UserForm элементов управления. Каждому объекту UserForm присущи определенные свойства, методы и события, которые он наследует от класса объектов UserForm. Каждый объект UserForm включает и модуль класса, в который можно добавлять собственные методы и свойства или код обработки событий формы.

Для добавления к проекту новой формы используется команда редактора VBA "Вставка/UserForm". По умолчанию новой форме присваивается имя UserForm1 и далее используется порядковая нумерация пользовательских форм.

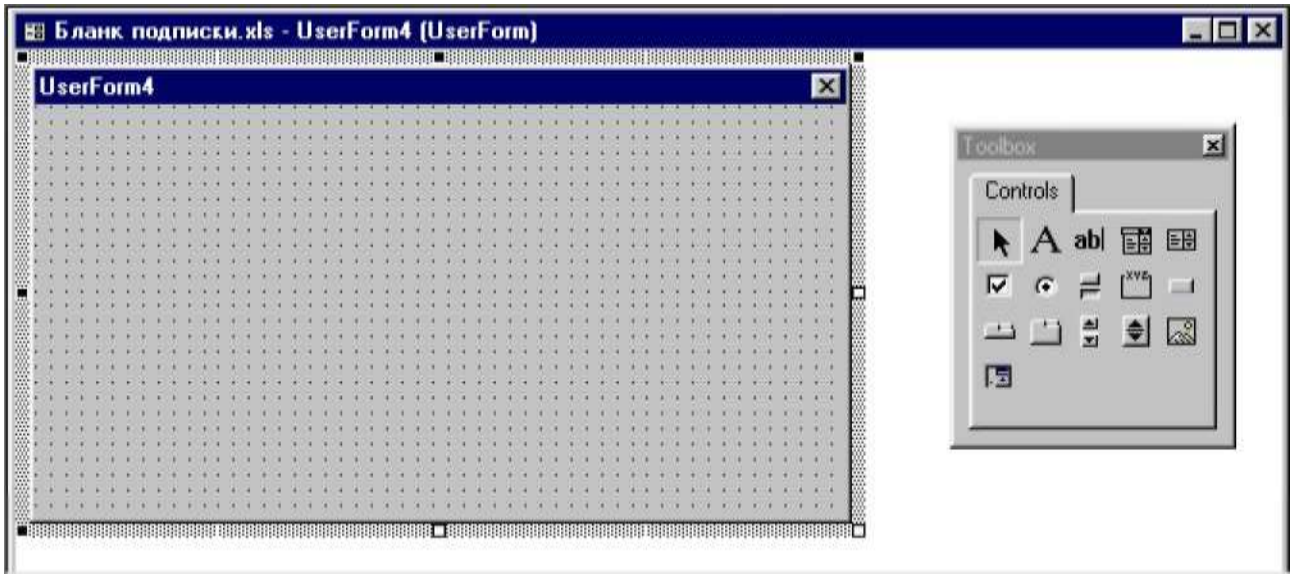


РИСУНОК 8. ***ОБЪЕКТ USERFORM В РЕЖИМЕ КОНСТРУИРОВАНИЯ*** **КАК ПРАВИЛО, ЕСЛИ АКТИВИЗИРОВАНА ФОРМА ИЛИ ОДИН ИЗ ЕЕ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ, НА ЭКРАНЕ ПОЯВЛЯЕТСЯ ПАНЕЛЬ ЭЛЕМЕНТОВ (TOOLBOX), С ПОМОЩЬЮ КОТОРОЙ МОЖНО ДОБАВЛЯТЬ К ФОРМЕ ЭЛЕМЕНТЫ УПРАВЛЕНИЯ.**

Переименовать объект UserForm можно так же, как стандартный модуль или модуль класса. Для этого надо задать значение свойства Name этого объекта.

Каждая добавляемая в проект форма наследует свойства и методы объекта UserForm. Все процедуры и функции, написанные в разделе General (общий) модуля класса формы, становятся дополнительными методами для этой формы. Форме можно придать и новые свойства, добавив в ее модуль класса описания Property Get и Property Let. Копии новой формы можно создавать программно с помощью оператора Dim и опции New.

Все формы VBA являются модальными (modal). Это означает, что вы не сможете выполнить какое-либо другое действие в приложении до тех пор, пока форма диалога не будет закрыта (методами Hide или UnLoad).

СВОЙСТВА ОБЪЕКТА USERFORM

Форма как объект имеет некоторые встроенные свойства, которые можно устанавливать двумя способами:

- программно;
- в окне свойств (Properties Window) редактора VBA.

Программным способом свойства форм устанавливаются путем присвоения свойству нового значения. В таб. 11 перечислены наиболее часто используемые свойства объекта UserForm.

ТАБЛИЦА 11. **Основные свойства объекта UserForm**

Свойство	Описание
ActiveControl	Возвращает объектную ссылку на элемент управления, находящийся в фокусе в данный момент. Свойство только для чтения
BackColor	Возвращает целое значение типа Long, которое определяет цвет фона формы
BorderStyle	Устанавливает тип границы
Caption	Возвращает текст, отображаемый в строке заголовка формы
Controls	Возвращает коллекцию всех элементов управления формы. Только для чтения
Cycle	Определяет, должно ли нажатие клавиши табуляции вызывать последовательный выбор всех элементов управления во всех группах и на каждой странице многостраничных элементов управления или только в пределах текущей группы или страницы. Может принимать значение одной из встроенных констант: fmCycleAllForms или fmCycleCurrentForms
Enabled	Содержит значение типа Boolean, указывающее, доступна ли форма. Если его значение равно False, ни один из элементов управления формы не доступен
Font	Позволяет выбрать параметры шрифта формы или элемента управления
ForeColor	То же самое, что и свойство BackColor, но устанавливает цвет, используемый для переднего плана (обычно это цвет текста) объекта формы
Height и Width	Возвращают высоту и ширину формы в пунктах
Left и Top	Возвращают местоположение левого верхнего угла формы в пунктах
Name	Возвращает имя пользовательской формы
Picture	Указывает рисунок, отображаемый как фон формы
StartUpPosition	Возвращает значение, определяющее положение формы при ее первом отображении на экране. Допустимые значения: • Manual – начальное значение не устанавливается; • CenterOwner – выравнивание по центру объекта, которому принадлежит форма; • CenterScreen – выравнивание по центру экрана; • Windows Default – положение верхнего левого угла экрана

МЕТОДЫ ОБЪЕКТА USERFORM

Исходный объект UserForm обладает рядом методов, наиболее часто используемые из которых приведены в таб. 12. Эти методы доступны для каждой формы, добавляемой в проект.

ТАБЛИЦА 12. **Основные методы объекта UserForm**

Метод	Назначение
Copy	Копирует выделенный в элементе управления текст в буфер обмена Windows
Cut	Вырезает выделенный в элементе управления текст и помещает его в буфер обмена Windows
Hide	Скрывает форму, не выгружая ее из памяти, сохраняя значения элементов управления формы и всех переменных, объявленных в модуле класса формы
Move	Изменяет положение и размер формы
Paste	Вставляет содержимое буфера обмена Windows в текущий элемент управления
PrintForm	Выводит на используемый в Windows по умолчанию принтер изображение формы, включая все данные, введенные в элементы управления
Repaint	Перерисовывает форму, выведенную на экран. Используйте этот метод, если хотите перерисовать форму, не ожидая, когда она будет перерисована через обычный период времени
Show	Выводит форму на экран. Если форма еще не загружена в память, то данный метод сначала ее загружает

СОБЫТИЯ ОБЪЕКТА USERFORM

Событие (event) – это факт, связанный с изменением состояния формы или элемента управления, требующий выполнения некоторых действий (обработки события).

Каждому событию может быть сопоставлена процедура, выполняющая необходимые действия - обработчик события. Обработчики событий могут быть назначены программно или в процессе визуального проектирования формы. При визуальном проектировании заготовка обработчика вида:

```
Private Sub UserForm_Click()
```

формируется автоматически, при двойном клике указателем мыши на форме или элементе управления. Код обработчика создается вручную.

Основная часть кода, который разработчик записывает в модуль класса формы, связана с обработкой событий. В таб.13 перечислены соновные события объекта UserForm, для которых можно написать обработчик в режиме визуального программирования.

ТАБЛИЦА 13. **Основные события объекта UserForm**

Событи	**Описание**
Activate	Происходит, когда окно формы становится активным. Используйте это событие для обновления содержимого диалоговых элементов управления, чтобы отразить любые изменения, которые произошли, пока окно формы было неактивным
Click	Происходит при щелчке мышью по форме (любой ее части, не занятой элементами управления)
DblClick	Происходит при двойном щелчке мышью по форме (любой ее части, не занятой элементами управления)
Deactivat	Происходит, когда форма перестает быть активной
Initialize	Происходит, когда форма впервые загружается в память посредством выполнения оператора Load или с помощью метода Show. Используйте это событие для инициализации элементов управления формы при ее появлении на экране
Resize	Происходит при изменении размеров формы
Terminat e	Происходит при закрытии формы, т.е. когда форма выгружается из памяти. Используйте это событие для осуществления специальных задач, которые необходимо выполнить прежде, чем переменные формы будут выгружены

В дополнение к методам, свойствам и событиям, встроенным в объект UserForm, VBA предоставляет два оператора, которые особенно полезны при работе с объектами форм: Load и Unload.

Синтаксис операторов Load и Unload:

```
Load Object Unload
Object
```

Здесь Object представляет любую допустимую ссылку на объект UserForm.

Оператор Load загружает в память объект UserForm и запускает метод формы Initialize, но не выводит форму на экран. Когда форма загружена, можно использовать написанную на VBA программу для работы с объектом UserForm.

Оператор Unload удаляет из памяти объект UserForm, а также все переменные формы. После того, как форма выгружена, она перестает быть доступной для VBA-кода.

ЭЛЕМЕНТЫ УПРАВЛЕНИЯ

Объект UserForm может содержать те же элементы управления, что и находящиеся в диалоговых окнах Word, Excel или других приложений Windows. Элементы управления (controls) – это элементы диалогового окна, которые дают возможность пользователю взаимодействовать с программой. Используя этот набор и редактор форм не трудно создать любой пользовательский интерфейс, который будет удовлетворять всем требованиям, предъявляемым к интерфейсу в среде Windows. Элементы управления являются объектами. Поэтому, как любые объекты, они обладают свойствами, методами и событиями. Как и для формы, их содержащей, свойства элементов управления можно устанавливать программным путем или с помощью окна свойств (Properties Window) редактора VBA. В программе можно присваивать или восстанавливать значения свойств элементов управления так же, как и для любых других объектов. Элементы управления создаются при помощи панели инструментов Toolbox (Панели элементов).

Создание элементов управления на рабочем листе, в документе или в форме, как правило, происходит на начальном этапе конструирования приложения. Иногда используется программное создание элементов управления в процессе работы приложения. Но этот подход применяется реже. Большинство элементов управления можно располагать как в документе или на рабочем листе, так и в форме. Но существуют такие элементы, как Набор страниц и Набор вкладок, которые можно располагать только в форме. В таб. 14 приведен список стандартных элементов управления, включенных в VBA, и соответствующих кнопок панели инструментов Toolbox, а также описано назначение каждого элемента. Как видно из этой таблицы, к стандартным относятся практически все элементы управления, которые встречаются в приложениях Windows.

ТАБЛИЦА 14. **Стандартные элементы управления, включенные в VBA**

Элемент	**Назначение**
Label (надпись, метка)	Позволяет создавать заголовки элементов управления, которые не имеют собственных встроенных заголовков

TextBox (текстовое поле)	Окно редактируемого текста свободной формы для ввода данных. Может быть одно- и многострочным
ComboBox (поле со списком)	Объединяет окно редактирования и окно списка
ListBox (список)	Отображает список значений, из которых пользователь может сделать выбор
CheckBox (флажок)	Стандартный флажок, который используется для выбора вариантов, не являющихся взаимоисключающими
OptionButton (переключатель)	Стандартная кнопка-переключатель. Используется, когда пользователю необходимо сделать выбор между "включено/выключено"
ToggleButton (выключатель)	Выключатели служат для той же цели, что и флажки, но выводят установки в виде кнопки, находящейся в "нажатом" или "отжатом" состоянии
Frame (рамка)	Визуально и логически объединяет некоторые элементы управления (особенно флажки, переключатели и выключатели)
CommandButton (кнопка)	Используется для выполнения таких действий, как Cancel (Отмена), Save (Сохранить), Ok и т.д. Когда пользователь щелкает по кнопке, выполняется VBA-процедура, закрепленная за данным элементом управления
TabStrip (набор вкладок)	Состоит из области, в которую следует помещать другие элементы управления (такие, как текстовые поля, флажки и т.д.)
MultiPage (набор страниц)	Состоит из нескольких страниц. Можно выбрать любую из них, щелкнув по соответствующей вкладке
ScrollBar (полоса прокрутки)	Позволяет выбирать линейное значение, аналогичное тому, как это можно сделать при помощи счетчика
SpinButton (счетчик)	Специальная разновидность текстового поля. Используется для ввода последовательных величин, которые заведомо находятся в определенном интервале значений (число, дата и т.п.)
Image (рисунок)	Выводит на форме графическое изображение любом из следующих форматов: *.bmp, *.cur, *.gif, *.ico, *.jpg, *.wmf

Для удобства работы с элементами управления в период их конструирования в приложениях Microsoft Office введен режим конструктора, который активизируется нажатием кнопки Режим конструктора (Designe Mode) панели инструментов. В режиме конструктора отключена реакция элемента управления на события. Поэтому при включенном режиме конструктора можно видоизменять элемент управления и задавать его свойства. Размещенный на форме элемент управления можно перемещать, изменять его размеры, копировать в буфер обмена и вставлять из буфера

обмена. Отключается режим конструктора той же кнопкой Режим конструктора (Designe Mode).

СВОЙСТВА ЭЛЕМЕНТОВ УПРАВЛЕНИЯ

Обращение к элементам управления выполняется, в основном, через их свойства и с помощью процедур обработки событий, написанных для каждого элемента. В таб. 15 перечислены наиболее часто используемые свойства элементов управления, которые позволяют изменять заголовок, определять состояние элемента управления (т.е. обнаруживать установки, выполненные пользователем) и так далее.

ТАБЛИЦА 15. **Основные свойства стандартных элементов управления**

Свойство	Описание
Accelerator	Содержит символ, используемый в качестве клавиши быстрого вызова элемента управления. При нажатии Alt+<клавиша быстрого вызова> происходит выбор элемента управления
AutoSize	Значение типа Boolean. Если равно True – устанавливает режим автоматического изменения размеров элемента управления так, чтобы на нем полностью помещался текст, присвоенный свойству Caption
BackColor	Устанавливает цвет фона элемента управления
BorderColor	Устанавливает цвет границы элемента управления
BorderStyle	Устанавливает тип границы элемента управления. Допустимые значения: fmBorderStyleSingle – граница в виде контура; fmBorderStyleNone – граница невидима
Caption	Надпись, отображаемая при элементе управления
Cancel	Задает кнопку отмены диалогового окна. Используется для элемента управления CommandButton. При нажатии на эту кнопку или клавишу Esc диалоговое окно исчезает
ControlTopText	Определяет текст, который отображается в виде всплывающей подсказки, когда указатель мыши помещается на элемент управления. В следующем примере элементу управления CommandButton1 назначен текст всплывающей подсказки "Это кнопка": CommandButton1. ControlTopText = "Это кнопка"

Default	Используется для элемента управления CommandButton. Определяет заданную по умолчанию кнопку. При нажатии на клавишу Enter эта кнопка ведет себя так, как если бы по ней щелкнули мышью
Enabled	Определяет, доступен элемент управления (значение True) или нет (значение False)
ForeColor	Устанавливает цвет для переднего плана элемента управления - как правило, символов текста
Height и Width	Устанавливают геометрические размеры объекта (высоту и ширину).
Left и Top	Устанавливают координаты верхнего левого угла элемента управления, определяющие его местоположение в форме
List	Представляет список, содержащийся в элементе управления (массив типа Variant). Используется для элементов управления ComboBox и ListBox
Max	Переменная типа Long – определяет максимальное значение счетчика или значение, при котором полоса прокрутки находится в самом верху (для вертикальной полосы) или справа (для горизонтальной). Используется для элементов управления ScrollBar и SpinButton
Min	Переменная типа Long – определяет минимальное значение счетчика или значение, при котором полоса прокрутки находится в самом низу (для вертикальной полосы) или слева (для горизонтальной). Используется для элементов управления ScrollBar и SpinButton
Name	Содержит имя элемента управления
Picture (создание картинки)	Внедряет картинку на элемент управления. Например, на поверхности кнопки картинка отображается с помощью следующей инструкции: CommandButto1.Picture = LoadPicture("c:\my_doc\Круг.bmp") Функция LoadPicture (Полное имя файла) считывает графическое изображение
Picture (удаление	После того, как картинка создана на элементе управления, иногда возникает необходимость ее удалить. Это легко достигается присвоением свойству Picture значения LoadPicture("")
RowSource	Задает источник, из которого ComboBox или ListBox "берет" список объекта
SpecialEffect	Устанавливает тип границы. Отличается от свойства BorderStyle тем, что позволяет установить несколько типов, но одного цвета. BorderStyle позволяет установить только один тип, но различных цветов
TabIndex	Определяет число, указывающее положение элемента управления в порядке табуляции. Может иметь значение от 0 до значения, равного количеству элементов управления на форме

TabStop	Значение типа Boolean, которое указывает, может ли элемент управления быть выбран клавишей Tab.
Tag	Используется для хранения дополнительной информации о форме или элементе управления, которая может быть в последующем востребована в программе
Value	Значение текущих установок элемента управления: текст в текстовом поле, какие выбраны флажки и переключатели, индекс выбранного раздела списка или число, указывающее текущее положение полосы прокрутки или счетчика
Visible	Значение типа Boolean, указывающее, является ли элемент управления видимым

Рассмотрим подробнее использование свойства Tag.

В следующем примере (листинг 29) на пользовательской форме расположены три кнопки и одно поле. Свойство Tag каждого из этих элементов управления, за исключением второй кнопки, установлено равным "Показать". Свойство Tag второй кнопки установлено равным "Спрятать". В цикле проверяется свойство Tag всех элементов управления и при инициализации диалогового окна отображаются только те элементы управления, у которых это свойство равно "Показать".

Листинг 29. Использование свойства Tag

```
Private Sub UserForm_Initialize() Dim Элемент As
    Object CommandButton1.Tag = "Показать"
    CommandButton2.Tag = "Спрятать"
    CommandButton3.Tag = "Показать"
    TextBox1.Tag = "Показать" For Each
    Элемент In Controls

        If Элемент.Tag = "Показать" Then

            Элемент.Visible = True Else

    Элемент.Visible = False End If Next Элемент
End Sub
```

В результате выполнения данной процедуры на форме будут отображены первая и третья кнопки, а также поле.

МЕТОДЫ И СОБЫТИЯ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ

В таб. 16 перечислены основные общие методы элементов управления.

ТАБЛИЦА 16. **Основные общие методы элементов управления**

Метод	Описание
Add	Позволяет добавить элемент управления во время выполнения программы
Move	Перемещает элемент управления
SetFocus	Устанавливает фокус на вызвавшем этот метод элементе управления. Часто применяется в программах обработки ошибок
Zorder	Помещает объект до или после всех пересекающихся с ним объектов

В таб. 17 приведены события элементов управления, для которых можно создать процедуры обработки событий. Каждый элемент управления, который вы добавите в свою форму, будет иметь доступ к этим событиям.

ТАБЛИЦА 17. **Наиболее часто используемые события объектов управления**

Событие	Описание
Click	Происходит, когда пользователь выбирает элемент управления с помощью одинарного щелчка кнопкой мыши
DblClick	Происходит, когда пользователь выбирает элемент управления с помощью двойного щелчка кнопкой мыши
KeyDown	Происходит при нажатии пользователем какой-либо клавиши в тот момент, когда форма выполняется и имеет фокус
KeyPress	Происходит, когда пользователь нажимает любую клавишу на клавиатуре, кроме функциональных и клавиш управления
KeyUp	Происходит, когда пользователь отпускает клавишу
Change	Происходит при изменении значения элемента управления
GotFocus	Происходит, когда элемент управления получает фокус
LostFocus	Происходит, когда элемент управления теряет фокус
Error	Используется при уведомлении об ошибке
MouseDown	Происходит при нажатии кнопки мыши
MouseUp	Происходит при отпускании кнопки мыши
MouseMove	Происходит при перемещении указателя мыши
BeforeDragOver	Происходит, пока совершается операция перемещения (drag-and-drop) элемента управления
BeforeDropOrPaste	Происходит перед завершением операции перемещения (drag-and-drop) элемента управления
AfterUpdate	Происходит после обновления значения элемента управления
BeforeUpdate	Происходит после того, как было изменено значение элемента управления, но перед тем, как был обновлен сам элемент управления
Enter	Происходит, когда выделяется элемент управления

Exit	Происходит, когда с элемента управления снимается выделение
SpinDown	Происходит, когда пользователь щелкает стрелку "вниз" ("влево") кнопки счетчика
SpinUp	Происходит, когда пользователь щелкает стрелку "вверх" ("вправо") кнопки счетчика

ПРИМЕР СОЗДАНИЯ ФОРМЫ

Пусть требуется создать форму для ввода информации о студенте (для работы с ранее созданным классом CStudent). Такая форма может содержать несколько элементов управления, связанных с соответствующими свойствами объектной переменной класса CStudent. Кроме того, имеет смысл разместить на форме пару кнопок: подтверждение добавления и отмена операции.

Такая форма может быть создана следующим образом:

1. После добавления заготовки формы (меню "Вставка/UserForm"), ее имя (свойство Name) изменено на frmAddStudent.
2. На форму были добавлены следующие элементы управления: TextBox, Label, OptionButton, CommandButton. Элементам TextBox, предназначенным для ввода данных пользователем, заданы имена (свойство Name) txtLName, txtFName, txtMName, txtContacts, txtBDay. Элементам Label заданы надписи (свойство Caption) "Фамилия:", "Имя:", "Отчество" и т.д. Элементы OptionButton, используемые для указания пола студента, получили имена optMale и optFemale. Эти элементы сгруппированы с помощью рамки (Frame). Для подтверждения ввода или отмены действий на форму добавлены 2 кнопки (CommandButton). Первой из них задано имя (свойство Name) btnAdd и надпись (свойство Caption) "Принять", второй - btnCancel и "Отмена" соответственно.
3. Далее, размещенные элементы были спозиционированы путем изменения размеров и положения через указание значений свойств Left, Top, Width и Height.
4. В завершении были созданы (двойной клик на соответствующем элементе) заготовки обработчиков событий для кнопки btnAdd и btnCancel. Заготовка обработчика события инициализации формы была создана путем выбора этого события из списка доступных методов ("Редактор формы/Declarations/initialize")
5. В проект был добавлен дополнительный модуль с именем General (листинг 30), где описаны общие данные проекта, в частности переменная stud (CStudent) и макрос sample33, из которого вызывается форма frmAddStudent.

6. В созданные обработчики событий добавлен функциональный код, приведенный и прокомментированный в листинге 31.

Листинг 30. Модуль General

```
Option Explicit ' явное описание переменных

Public stud As New CStudent ' экземпляр класса CStudent

Public Const AppName = "БД Деканат"

Sub sample32()

    frmAddStudent.Show ' Показать форму End Sub
```

Листинг 31. Модуль формы frmAddStudent

```
' Событие происходит в момент первого вызова формы Private Sub
UserForm_Initialize()

Caption = AppName & ": Добавление студента" ' Заголовок формы ' Задаем значение по умолчанию в
поле ввода даты рождения txtBDay = Format(Date, "DD.MM.YYYY") End Sub


' Событие происходит по клику на кнопке "Принять" (btnAdd) Private Sub
btnAdd_Click()

With stud
 ' Записываем значения полей ввода в свойства объекта stud

    .LastName = txtLName.Value .FirstName =
    txtFName.Value .MiddleName = txtMName.Value
    .BirthDay = CDate(txtBDay.Value) .Contacts =
    txtContacts.Value

' Проверяем значения элементов OptionButton

' Если установлен optMale, то в свойство stud.Gender пишем "муж",... If optMale.Value Then
    .Gender = "муж" ElseIf optFemale.Value Then '... иначе - "жен"

        .Gender = "жен" End If

    Debug.Print .FullInfo ' отображаем информацию о студенте в отладчике End With

' Уведомляем пользователя о добавлении записи и ждем дальнейших указаний ' Если пользователь
нажмет "Отмена", то скрываем форму (метод Hide) msg = "Запись добавлена." & vbCrLf &
"Продолжить?" If MsgBox(Prompt:=msg, Title:=AppName, Buttons:=vbOKCancel _
```

```
        + vbInformation) = vbCancel Then Hide End Sub

' Событие происходит при выборе кнопки "Отмена" (btnCancel) Private Sub
btnCancel_Click()

    Hide ' скрываем форму End Sub
```

Как видно из приведенного примера, использование визуальных средств разработки

делает процесс проектирования пользовательского интерфейса программы более наглядным и быстрым. Основной акцент переносится непосредственно на решение

прикладной задачи. При этом сохраняется возможность программного управления всеми элементами управления.

ТЕМА №8. ИНТЕГРАЦИЯ С ВНЕШНИМИ ПРИЛОЖЕНИЯМИ

Автоматизация, ранее известная как OLE-автоматизация, - это технология, позволяющая включать функциональные средства любого Windows-приложения в другое приложение посредством программного кода. Другими словами, *автоматизация* - это процесс управления одним приложением посредством другого. Использование другого приложения как источника новых средств и инструментов значительно расширяет возможности вызывающего приложения, причем это даже не всегда требует написания соответствующих процедур VBA. Можно, например, организовать показ слайдов с музыкальным сопровождением (используя средства PowerPoint), которые будут отображать данные рабочей книги Excel.

Для того чтобы использовать автоматизацию, необходимо располагать инсталлированным приложением, средствами которого вы хотите воспользоваться, и это приложения должно поддерживать технологию автоматизации. Большинство приложений, работающих под Windows, полностью поддерживают автоматизацию. В этом случае говорят, что они *открыты,* т.е. доступны все коллекции их объектов, свойств, методов и событий. Возможность использовать открытые объекты - это то же самое, что и возможность использовать средства приложения.

ОСНОВЫ АВТОМАТИЗАЦИИ

При работе со средствами автоматизации надо хорошо понимать роль и назначение каждого участвующего приложения: различают управляющее приложение, или приложение-клиент, и приложение-сервер. *Приложение-клиент* управляет приложением-сервером. Visual Basic - прекрасный пример управляющего приложения. *Приложение-сервер* открывает свои объекты, которые может использовать другое приложение. В рамках этой книги Excel является приложением-клиентом.

Стандарт DDE (Dynamic Data Exchange -динамический обмен данными) и метод **SendKeys** позволяют работать с приложениями, которые не поддерживают технологию OLE-автоматизации.

ССЫЛКА НА БИБЛИОТЕКУ ОБЪЕКТОВ ПРИЛОЖЕНИЯ-СЕРВЕРА

Многие приложения, которые поддерживают технологию автоматизации, имеют библиотеки объектов. *Библиотека объектов* содержит информацию, необходимую приложению-клиенту для управления объектами приложения-сервера. Для получений доступа к этой библиотеке надо создать на нее ссылку из управляющего приложения, такого как Excel.

Рассмотрим, как Excel будет управлять приложением Microsoft Word, т. е. Word будет приложением-сервером, и надо создать ссылку на библиотеку его объектов. Для этого, находясь в редакторе Visual Basic, выполните команду Tools>References (Сервис>Ссылки). Откроется диалоговое окно References, показанное на рис. 9. Установите флажок Microsoft Word 9.0 Object Library (Библиотека объектов Microsoft Word 9.0) и щелкните на кнопке ОК. В текущий проект будет добавлена ссылка на эту библиотеку.

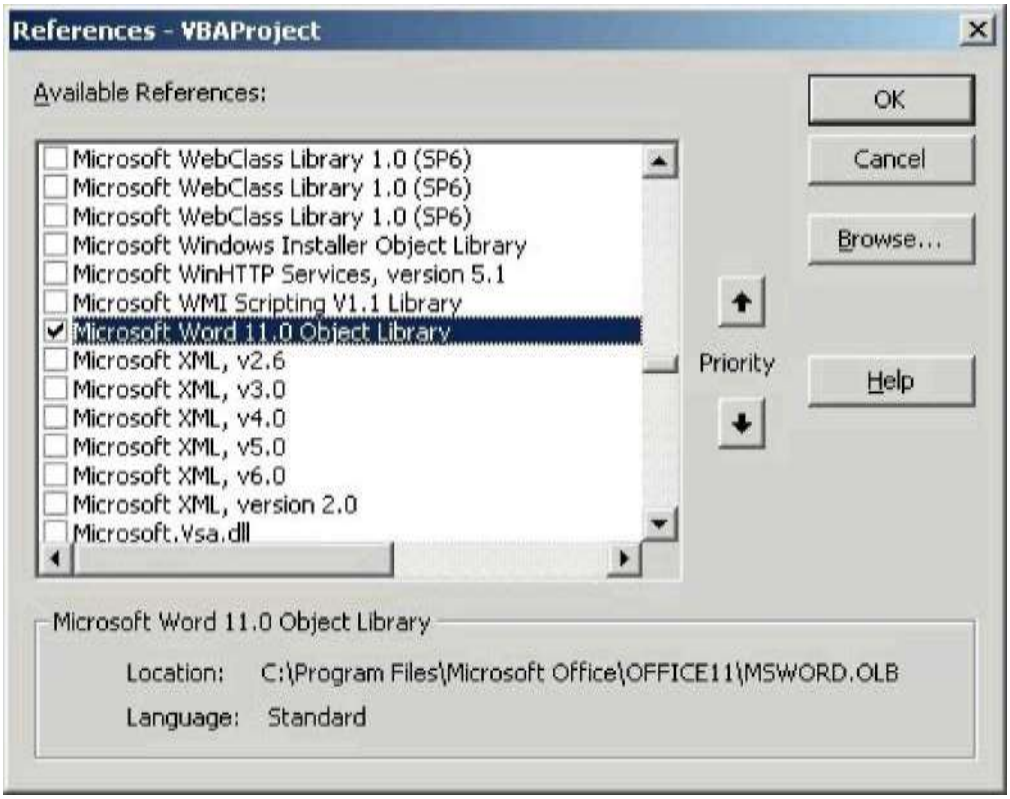


РИСУНОК 9. **СПИСОК ДОСТУПНЫХ ССЫЛОК НА СЕРВЕРЫ АВТОМАТИЗАЦИИ**

ПРОСМОТР БИБЛИОТЕКИ ОБЪЕКТОВ

Окно просмотра объектов Object Browser позволяет просматривать все библиотеки объектов, на которые установлены ссылки. Здесь приведены списки всех доступных объектов и их свойств, событий и методов. Для просмотра библиотеки объектов Microsoft Word сначала выполните команду View > Object Browser (Вид > Просмотр объектов), которая открывает окно Object Browser. Выберите Word из списка проектов и библиотек

(самый верхний раскрывающийся список в окне Object Browser), отобразится содержимое библиотеки объектов Word (рис. 10). Здесь можно выбрать любой объект этой библиотеки и просмотреть все его свойства и методы.

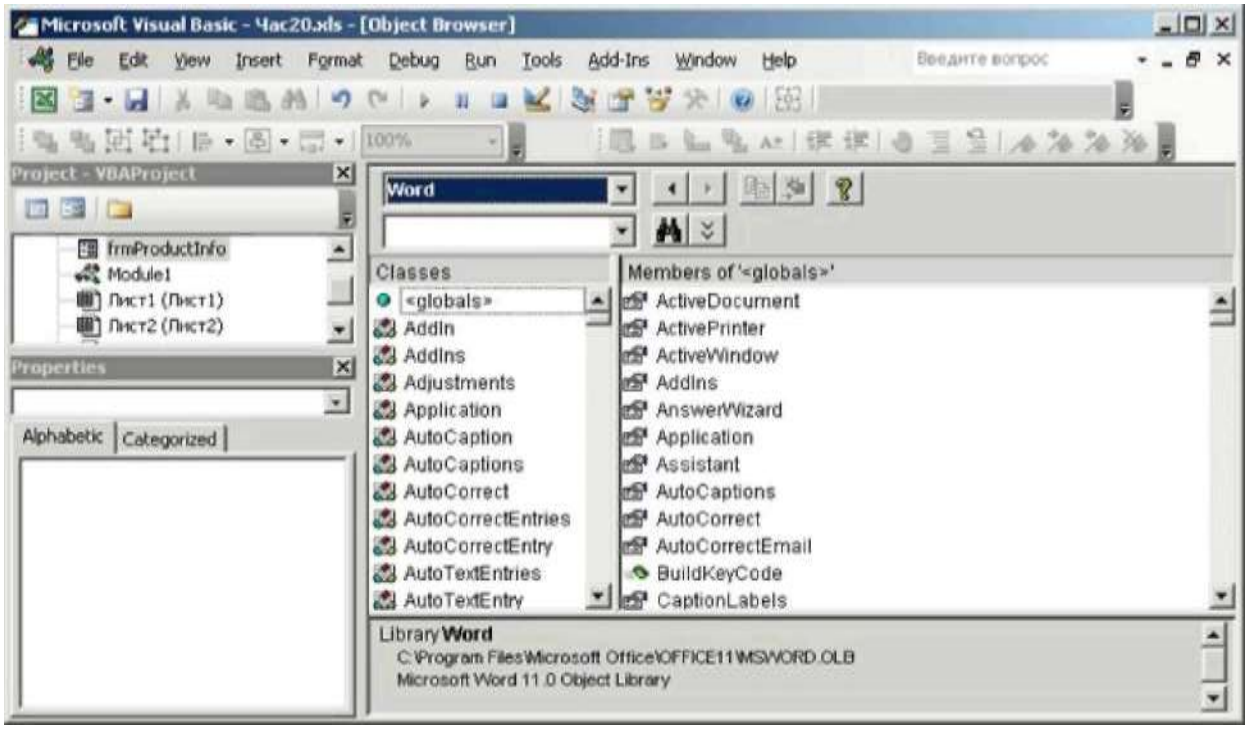


РИСУНОК 10. **ОКНО ПРОСМОТРА ОБЪЕКТОВ**

СОЗДАНИЕ ЭКЗЕМПЛЯРА ПРИЛОЖЕНИЯ-СЕРВЕРА

Задание ссылки на соответствующую библиотеку объектов - это необходимое, но недостаточное условие для использования объектов этой библиотеки. Нужно также с помощью процедуры создать экземпляр приложения-сервера. Для этого используется оператор CreateObject (Создать объект). Только после создания экземпляра приложения-сервера можно использовать все открытые объекты этого приложения, а также их свойства и методы. В листинге 32 приведен код, создающий экземпляр Word.

Листинг 32. Создание экземпляра Word

```
Dim y As Word.Application

Set y = CreateObject("Word.Application")
```

После создания экземпляра приложения-сервера, можно использовать его объекты,

свойства и методы точно так же, как и объекты, свойства и методы Excel. Из этого следует,

что можно записать макрос в Word, а затем полученный код вставить в процедуру Excel.

Далее в код при необходимости можно внести изменения, или как-нибудь еще использовать этот код.

Рассмотрим простое приложение, вставляющее диапазон ячеек рабочего листа Excel в письмо, созданное в Word. В программе Word создадим документ следующего содержания:

Менеджеру по продажам

Посылаю Вам таблицу с итоговыми данными по объемам продаж. Если у Вас есть вопросы, то свяжитесь со мной.

С уважением, Босс

Сохраните этот текст в файле letter.doc.

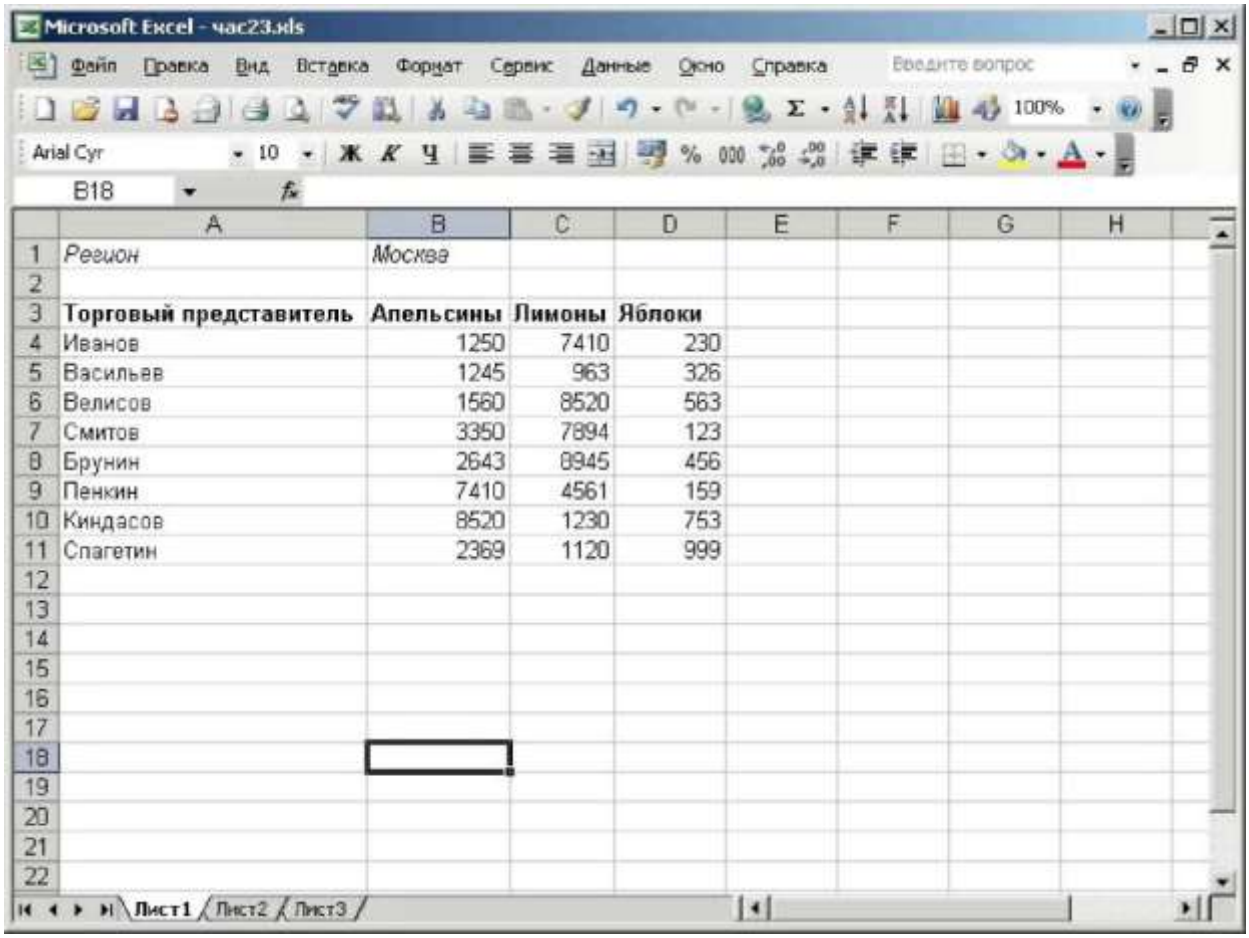
Теперь вставим в письмо две закладки. *Закладка* - это поименованное место в документе Word. Для вставки закладок выполните следующие действия.

1. Установите курсор в начало фразы *Менеджеру по продажам*.
2. Выполните команду Вставка > Закладка, откроется диалоговое окно Закладка.
3. Введите Регион в качестве имени закладки и щелкните на кнопке Добавить.
4. Добавьте пустую строку перед *С уважением*. Установите курсор в эту пустую строку и вставьте закладку Данные.
5. Сохраните и затем закройте документ.

Перейдите в Excel и создайте таблицу с данными, подобную приведенной на рис. 11. Эта таблица далее будет скопирована и вставлена в документ letter.doc.

Вернитесь в Word и начните запись макроса с именем Данные_продажи, предварительно открыв новый документ.

Для начала записи макроса в Word необходимо выполнить те же действия, что и в Excel. Прежде всего выполните команду Сервис > Макрос > Начать запись.



	A	B	C	D	E	F	G	H
1	*Регион*	*Москва*						
2								
3	**Торговый представитель**	**Апельсины**	**Лимоны**	**Яблоки**				
4	Иванов	1250	7410	230				
5	Васильев	1245	963	326				
6	Велисов	1560	8520	563				
7	Смитов	3350	7894	123				
8	Брунин	2643	8945	456				
9	Пенкин	7410	4561	159				
10	Киндасов	8520	1230	753				
11	Спагетин	2369	1120	999				
12								
13								
14								
15								
16								
17								
18								
19								
20								
21								
22								

РИСУНОК 11. **ТАБЛИЦА ДАННЫХ, КОТОРУЮ НАДО ВСТАВИТЬ В ПИСЬМО ВО ВРЕМЯ ЗАПИСИ МАКРОСА ВЫПОЛНИТЕ СЛЕДУЮЩИЕ ДЕЙСТВИЯ.**

1. Выберите команду Файл > Открыть и откройте документ letter.doc.
2. Выполните команду Правка > Перейти. В диалоговом окне Найти и заменить на вкладке Перейти в списке Объект перехода выберите пункт Закладка, а затем в раскрывающемся списке Введите имя закладки выберите Регион. Щелкните на кнопке Перейти.
3. Теперь выберите закладку Данные и щелкните на кнопке Перейти.
4. Щелкните на кнопке Закрыть для удаления диалогового окна Найти и заменить.
5. Остановите запись макроса.

Выполнив команду Сервис > Макрос > Макросы, откроете диалоговое окно Макрос и выберите макрос Данные_продажи. Для просмотра кода этого макроса щелкните на кнопке Изменить. Удалите строки из процедуры так, чтобы у вас осталось то, что показано в листинге 33.

Листинг 33. "Остаток" процедуры Данные продажи

```
Sub Данные_продажи()
```

```
Documents.Open FileName:="Письмо.doc"
Selection.Goto What:=wdGoToBookmark, Name:="Регион"
Selection.Goto What:=wdGoToBookmark, Name:="Данные" End Sub
```

Этот код будет основой процедуры, которую мы создаем в Excel. Скопируйте эту процедуру в буфер обмена. Закройте Word, вернитесь в Excel и откройте редактор Visual Basic. Добавьте модуль в текущую рабочую книгу и вставьте в него скопированную процедуру. Теперь надо создать экземпляр Word, а также перед каждой строчкой процедуры поставить имя переменной, соответствующей экземпляру Word. Самый простой способ сделать это - применить оператор With. Код измененной процедуры показан в листинге 34.

Листинг 34. Процедура Данные продажи с экземпляром Word

```
Sub Данные_продажи() Dim y As Word.Application Set y =
CreateObject("Word.Application") With y .Documents.Open
FileName:="Письмо.doc" .Selection.Goto What:=wdGoToBookmark,
Name:="Регион" .Selection.Goto What:=wdGoToBookmark, Name:="Данные"
End With End Sub
```

Осталось добавить код, копирующий таблицу данных в Excel и вставляющий ее в документ Word. Полный код процедуры приведен в листинге 35.

Листинг 35. Законченная процедура Данные продажи '

```
Sub Данные_продажи()
Dim y As Word.Application
Set y = CreateObject("Word.Application")
With y
        .Visible = True
        .Documents.OpenFileName:="C:\Мои документы\letter.doc"
       Worksheets("Лист1").Range("B1").Copy
```

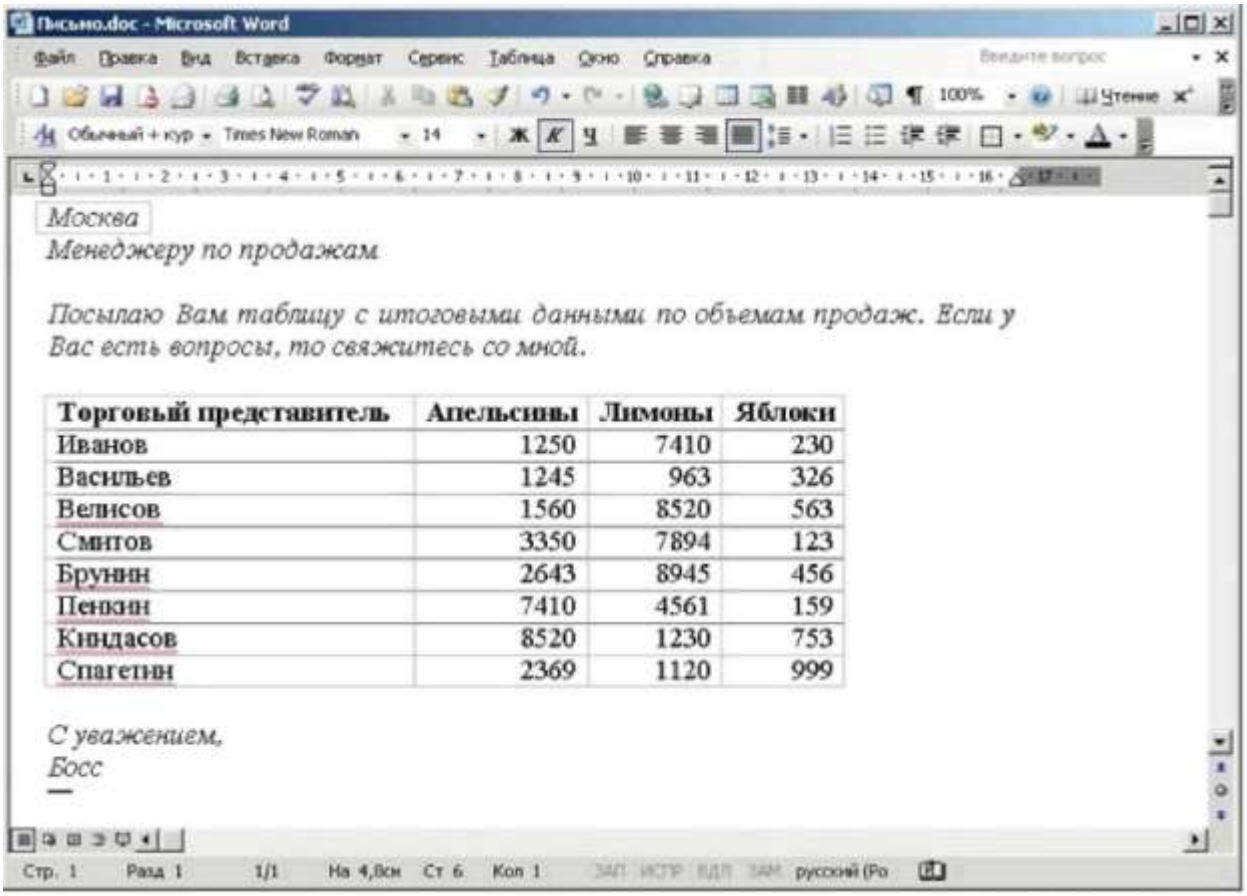
```
.Selection.Goto What:=wdGoToBookmark, Name:="Регион"
.Selection.Paste
Application.CutCopyMode = False
Worksheets("Лист1").Range("A3:D11).Select
.Selection.Copy
.Selection.Goto What:=wdGoToBookmark, Name:="Данные"
.Selection.Paste Application.CutCopyMode = False End With End
Sub
```

Первый новый оператор .Visible = True устанавливает свойство Word Visible (Видимый) как True, тем самым открывая окно программы Word и выводя его на передний план.

Далее открывается документ Письмо. Затем копируется содержимое ячейки B1, содержащей название региона, осуществляется переход к закладке Регион и вставляется содержимое ячейки B1. Установка свойства Excel cutCopyMode (Режим вырезания и копирования) как False снимает выделение с ячейки B1:

```
.Documents. OpenFileName:="C:\Мои документы\letter.doc"
Worksheets("Лист1").Range("B1").Copy
.Selection.GoTo What:=wdGoToBookrnark, Name:="Регион"
.Selection.Paste
Application.CutCopyMode = False
```

Затем аналогичная процедура копирования и вставки выполняется для диапазона A3:D11. Результат работы программы приведен на рис. 12.



Торговый представитель	Апельсины	Лимоны	Яблоки
Иванов	1250	7410	230
Васильев	1245	963	326
Велисов	1560	8520	563
Смитов	3350	7894	123
Брунин	2643	8945	456
Пенкин	7410	4561	159
Киндасов	8520	1230	753
Спагетин	2369	1120	999

РИСУНОК 12. **ДОКУМЕНТ WORD СО ВСТАВЛЕННЫМИ ДАННЫМИ EXCEL В ЗАКЛЮЧЕНИЕ ПЕРЕЧИСЛИМ ЭТАПЫ ВЫПОЛНЕНИЯ ЭТОГО ПРИМЕРА**.

- Создание файла в приложении-сервере.
- Создание таблицы данных в Excel (приложении-клиенте).
- Запись макроса в приложении-сервере.
- Копирование кода записанного макроса и вставка его в процедуру, создаваемую в Excel.
- Изменение процедуры Excel для создания экземпляра приложения-сервера.

Как видно из примера, использование технологии автоматизации позволяет внедрять и использовать объекты других приложений в проектах VBA.

ТЕМА №9. ПЕРСПЕКТИВЫ РАЗВИТИЯ ППП

ОСНОВНЫЕ ТЕНДЕНЦИИ В РАЗВИТИИ ППП

В качестве заключения отметим перспективные направления дальнейшего развития прикладного ПО. На сегодняший день в качестве основных факторов, влияющих на функциональность ППП и сложность их разработки ПО, можно отметить следующие:

- рост производительности персональных компьютеров;
- расширение классов решаемых задач;
- увеличение общего числа пользователей;
- значительное количество ранее созданного (наследованного) ПО;
- развитие Интернет и корпоративных сетей.

Разработка приложений с учетом этих факторов привела к появлению прикладных пакетов и интегрированных сред, которые по своим характеристикам выходят за рамки ППП четвертого поколения. Среди отличительных черт ПО нового поколения следующие:

- интеграция компонентов ППП не только с приложениями пакета, но и с окружением;
- широкое использование отраслевых стандартов;
- использование инфраструктуры Интернет;
- платформонезависимость.

Особую значимость на дальнейший сценарий развития ППП имеет влияние технологий Интернет и, в частности, Web. Возможности, представляемые глобальной сетью позволяют обмениваться любой информацией, которую можно представить в цифровом виде. Это уже сейчас с успехом испольуется в ведущих пакетах прикладных программ, в первую очередь для обеспечения совместной работы пользователей. Практическая реализация общего доступа возможна, например, с использованием промежуточного ПО (middleware). Так, при использовании технологии ActiveX в документ MS Word или таблицу MS Excel можно поместить любой документ, поддерживающий ActiveX. Внедренным может быть документ, размещенный в

Интернет, более того, имеется потенциальная возможность отредактировать его и сохранить изменения в Сети.

Все большей популярностью пользуется концепция “тонких клиентов”. Под “тонким клиентом” подразумевается Интернет-браузер. Современные браузеры позволяют отображать не только гипертекстовые документы, но и изображения в растровых и векторных форматах, видео- и аудиоданные. Кроме этого, браузеры представляют средстваинтерактивного взаимодействия с веб-серверами в виде различных веб-форм (от форм авторизации или поиска до форм загрузки файлов) и поддерживают выполнение программ-скриптов на своей стороне. Это позволяет создавать программа, загружаемые с веб-сервера, но выполняемые в браузере. Примером такого решения являются сервисы Google Docs (Google Документы). Пользователям этого сервиса представляется возможность создавать и редактировать текстовые документы, электронные таблицы и презентации прямо в окне браузера, сохранять их в Интернет и предоставлять в совместныое использование (рис.13).

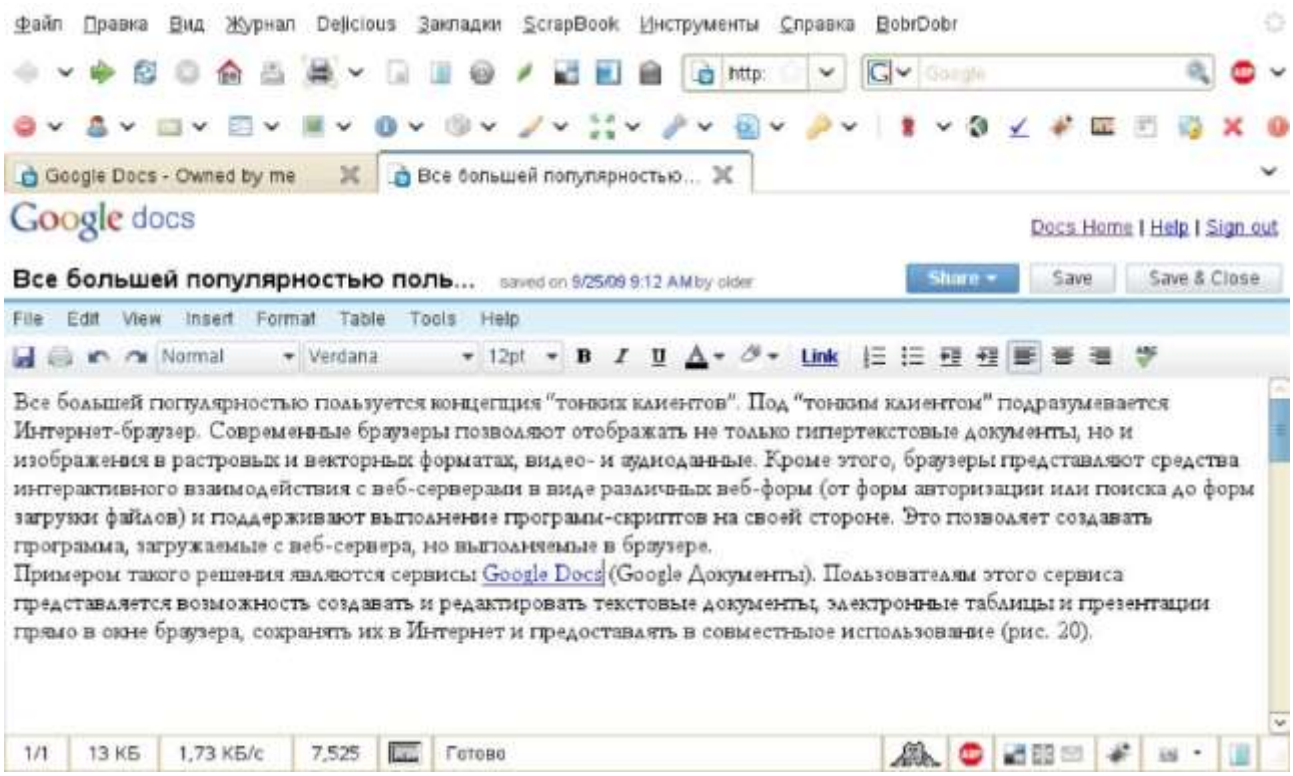


РИСУНОК 13. **GOOGLE DOCS - НАСТРОЙКА ДОСТУПА К ДОКУМЕНТУ**

Ввиду повсеместного проникновения Интернета, можно говорить о том, что прикладное программное обеспечение будет переходить в разряд сервиса, то есть пользователи будут работать с необходимым программным обеспечением через

Сеть, получая на свои компьютеры готовые результаты. Следовательно, необходимость в больших локальных мощностях частично отпадет, что будет способствовать росту спроса на недорогие компьютеры с низким энергопотреблением.

В основе технологий, обеспечивающих подобные возможности, ряд совместных наработок ведущих производителей ПО и организаций по стандартизации. К ним относятся сервисно-ориентированная архитектура корпоративных приложений (веб-сервисы) и стандартизованные форматы документов.

ВЕБ-СЕРВИСЫ

Веб-сервис, веб-служба, (англ. web service) - программная система, доступная через локальную или глобальную сеть по заданному адресу, чьи общедоступные интерфейсы определены на языке XML. Эта программная система доступна другим программными системами, которые могут взаимодействовать с ней посредством XML-сообщений. передаваемых с помощью интернет-протоколов. Веб-служба является единицей модульности при использовании сервисно-ориентированной архитектуры приложения. Иными словами, веб-сервис - это именованный компонент распределенной прикладной системы, доступный по гипертекстовым протоколам (HTTP, HTTPS и подобным).

Сервисно-ориентированные приложения построены на следующих индустриальных стандартах:

- XML: Расширяемый язык разметки, предназначенный для хранения и передачи структурированных данных;
- SOAP: Протокол обмена сообщениями на базе XML;
- WSDL: Язык описания внешних интерфейсов веб-службы на базе XML;
- UDDI: Универсальный интерфейс распознавания, описания и интеграции (Universal Discovery, Description, and Integration). Каталог веб-служб и сведений о компаниях, предоставляющих веб-службы во всеобщее пользование или конкретным компаниям.

  Основными достоинствами веб-сервисов являются:

- интероперабельность;

- открытость архитектуры;
- взаимодействие программных систем через средства защиты информации (прокси-серверы, межсетевые экраны).

Основным недостатком является меньшая производительность приложений и больший объем сетевого трафика по сравнению с другими технологиями распределенных вычислений (RMI, CORBA, DCOM/ActiveX). Еще одним недостатком является повышенная требовательность к аппаратным ресусам на стороне сервера приложений (поставщика веб-сервисов).

УНИФИКАЦИЯ ФОРМАТОВ

Перспективным направлением в развитии ППП является использование унифицированных форматов документов на основе открытых стандартов. Открытый стандарт - общедоступная спецификация, обычно разрабатываемая некоммерческой организацией по стандартизации, свободная от лицензионных ограничений при использовании. Открытые форматы являются подмножеством открытых стандартов и определяют спецификации хранения и представления цифровых данных. Использование открытых форматов в ППП позволяет гарантировать возможность доступа к данным из любого совместимого приложения без оглядки на лицензионные права и технические спецификации. Актуальность концепции открытых форматов подтверждается практикой -правительственные организации многих стран используют их в качестве основного средства.

На сегодняшний день разработаны и применяются открытые форматы практически для всех классов задач, решаемых ППП, начиная от офисных приложений до мультимедийных данных и 3D-графики.

OPENDOCUMENT FORMAT

OpenDocument Format (ODF, сокращённое от OASIS Open Document Format for Office Application - открытый формат документов для офисных приложений) - открытый формат файлов документов для хранения и обмена редактируемыми офисными документами: текстовыми документами, электронными таблицами, рисунками, базами данных, презентациями.

Стандарт был разработан индустриальным сообществом OASIS и основан на XML-формате. 1 мая 2006 года принят как международный стандарт ISO/IEC 26300,

доступен для всех и может быть использован без ограничений. Этот формат поддерживается в таких ППП как OpenOffice.org, IBM Lotus Symphony, Koffice, Scribus, Google Docs, AjaxWrite, Microsoft Office 2007 SP2.

PORTABLE NETWORK GRAPHICS

PNG (англ. portable network graphics) - растровый формат хранения графической информации со сжатием без потерь качества. PNG был создан специально для использования в Интернет как альтернатива формату GIF. Этот формат был разработан в начале 1995 г. по идее Т. Боутелла. В октябре 1996 года спецификация PNG версии 1.0 была рекомендована консорциумом W3C в качестве полноправного сетевого формата и сейчас широко используется в Сети.

ФОРМАТ SVG

SVG (от англ. Scalable Vector Graphics - масштабируемая векторная графика) - язык разметки масштабируемой векторной графики, созданный консорциумом W3C и входящий в подмножество расширяемого языка разметки XML. SVG предназначен для описания двухмерной векторной и смешанной векторно/растровой графики. Формат поддерживает как неподвижную, так анимированную и интерактивную графику.

Поскольку SVG основан на XML, то он представляет все преимущества расширяемого языка разметки. Отметим основные:

- возможность работы в различных средах;
- интернационализация;
- доступность для любых приложений;
- лёгкая модификация через стандартные функции API;
- лёгкое преобразование из других форматов. Приведем пример: используя XSL-трансформацию, можно, например визуализировать химические молекулы, описанных на языке CML (Chemical Markup Language).

ФОРМАТ СЖАТИЯ 7Z

Алгоритм сжатия 7z, лежащий в основе программы-архиватора 7-Zip - еще один пример открытого формата. Среди его достоинств такие:

- открытый код;
- высокая степень сжатия;
- высокая скорость распаковки;
- многопоточное сжатие;
- поддержка криптостойкого шифрования;
- поддержка 64-битных систем.

ПРИЛОЖЕНИЯ ПО ЗАПРОСУ

Распространение веб-сервисов на основе открытых стандартов ведет к ситуации, когда вместо запуска определенных программ корпоративные пользователи смогут получить доступ к любым прикладным средствам, необходимым в данный момент, просто подключившись к Интернет. В таком контексте приложения могут быть представлены как свободно и бесплатно, так и платно, по подписке, в зависимости от объема потребления. Сочетание широкополосного интернета с платформонезависимыми приложениями (написанными, к примеру, на языке Java) в некоторых областях уже сделали модель коммунальных услуг в области ИТ реальностью. Например, Salesforce.com за умеренную месячную плату предлагает в интернете приложения для управления отношениями с клиентами (CRM). Пользователям, количество которых уже составляет около 100 тыс., не нужно устанавливать или поддерживать у себя сложные пакеты CRM. Им достаточно только запустить браузер и подключиться к серверам и услугам Salesforce.com. В свою очередь свободный доступ к офисным приложениям представляет ранее упомянутый сервис Google Docs, число пользователей которого составляет более полумиллиона.