Программное обеспечение информационных технологий

Аппаратное обеспечение информационных технологий создаёт возможность автоматизированной обработки данных. Однако сам характер и возможности обработки данных определяются программным обеспечением (ПО). В конечном итоге программы предназначены для решения задач связанных с обработкой данных и получения результатов, которые используются для эффективного управления различными объектами. В то же время, для работы этих программ требуются вспомогательные программы, которые обеспечивают их взаимодействие с аппаратным обеспечением, передачу данных в сетях и другие сервисные функции. Такое разделение функций программных средств закреплено разделением их на три большие группы:

- Системное ПО
- Прикладное ПО
- Инструментальное ПО.

По способу распространения и использования ПО делится на:

- Закрытое ПО, которое распространяется только по платным лицензионным соглашениям без доступа к исходному программному коду.
- Открытое программное обеспечение (англ. open-source software) — программное обеспечение с открытым исходным кодом. Исходный код таких программ доступен для просмотра, изучения и изменения, что позволяет пользователю принять участие в доработке самой открытой программы, использовать код для создания новых программ и исправления в них ошибок — через заимствование исходного кода, если это позволяет совместимость лицензий, или через изучение использованных алгоритмов, структур данных, технологий, методик и интерфейсов (поскольку исходный код может существенно дополнять документацию, а при отсутствии таковой сам служит документацией).
- свободное или FREEWARE(свободное программное обеспечение может распространяться, устанавливаться и использоваться на любых компьютерах дома, в офисах, школах, вузах, а также коммерческих и государственных учреждениях без ограничений)

В системное ПО входят такие программные средства как:

- Операционные системы
- Тестовые программы
- Драйвера
- Средства сетевого взаимодействия компьютеров
- Программы защиты от вредоносного ПО
- Системы управления базами данных

К прикладному ПО относятся программные средства (ПС), позволяющие решать задачи управления экономическими объектами. К таким объектам можно отнести предприятия, региональные структуры, человека и т.д. То есть все виды деятельности человека, где требуется обработка информации. Часто такие программы называют приложениями. Использование прикладных программных средств обладает рядом особенностей, которые связаны с технологией их эксплуатации для решения конкретных задач.

1. По этому признаку можно выделить программные средства для решения целевых задач. Например, бухгалтерского учёта, складского учёта, управления торговыми операциями (программы линейки 1С) , математические статистические пакеты прикладных программ и т.д.
2. Другой группой программных средств, являются программы, работающие с широким классом однотипных объектов, например, с изображениями, презентациями, документами, издательскими макетами и т.д. К ним можно отнести Word, Power Point,Publisher, Photo Shop , системы автоматизированного проектирования (САПР) и т.д.
3. И, наконец, существует широкий класс программных средств, позволяющих создавать программы пользователей для решения конкретных задач без обладания профессиональными навыками программиста. Наиболее распространённой из таких программ является программа Excel. Более мощным и гибким в плане поддержки хранения, обработки и визуализации данных является программа Access.

Последняя группа программных средств позволяет создавать при сравнительно небольших профессиональных навыках программы, которые можно отнести к первой группе ППО. Однако для разработки профессиональных программных продуктов этих средств недостаточно. Поэтому для профессиональной работы были созданы специальные средства для разработки профессиональных программ. К ним можно отнести трансляторы, интерпретаторы и такие интегральные средства разработки программ как Visual Studio фирмы Microsoft, Delphi фирмы Borland и другие. Их относят к инструментальному ПО.

Системное ПО

Операционная система - это совокупность программных средств, которые обеспечивают управление аппаратными ресурсами вычислительной системы и взаимодействие программных процессов с аппаратурой, другими процессами и пользователем.

Операционная система выполняет следующие функции:

- управление памятью, вводом - выводом информации, файловой системой, взаимодействием процессов;
- диспетчеризацию процессов; защиту информации; учет использования ресурсов;

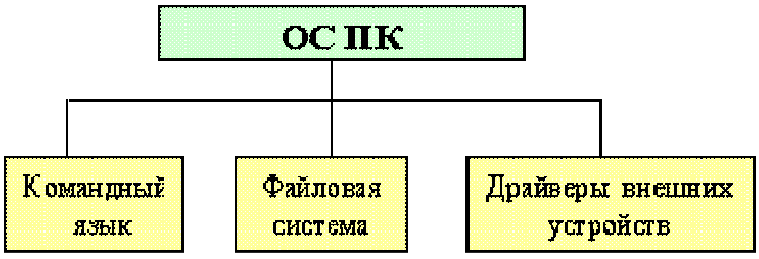
- обработку командного языка; выявление разных моментов, которые возникают в процессе работы, и соответствующую реакцию на них (например, при ошибочных ситуациях).

ОС координирует работу всех устройств компьютера. При параллельной работе процессора, памяти и внешних устройств операционная система обеспечивает разделение ресурсов, чем предотвращает возможность возникновения конфликтов между компонентами вычислительной системы.

Одной из важнейших функций ОС является автоматизация процессов ввода-вывода информации, управления выполнением прикладных задач, решаемых пользователем. ОС загружает нужную программу в память компьютера и следит за ходом ее выполнения; анализирует ситуации, препятствующие нормальным вычислениям, и дает указания о том, что необходимо сделать, если возникли затруднения. Пользователь взаимодействует с персональным компьютером через внешний интерфейс, организуемый операционной системой. В зависимости от своих целей, он вводит задания и получает результаты их выполнения либо, работая в диалоговом режиме, использует для общения с компьютером специальный интерфейс диалога. Диалоговый интерфейс - это совокупность программных средств, предназначенных для обмена информацией между пользователем и операционной системой. Существуют два типа диалоговых интерфейсов между пользователем и компьютером: текстовый, как, например, в операционной системе MS DOS, и графический, характерный для операционной системы Windows. В графических интерфейсах информация и команды представляются в виде пиктограмм (значков), и пользователь выполняет те или иные действия, указывая на эти пиктограммы и манипулируя ими определенным образом

Составные части операционной системы

Термин "операционная система" не имеет строгого определения, поскольку в различных операционных системах в ее состав входят различные системные программы. Наиболее важными частями операционной системы являются: файловая система, драйверы внешних устройств, загрузчик, системная библиотека.



Командный язык

Командный язык - это набор команд, которые вводятся пользователем для непосредственного их исполнения. Командный язык поддерживает связь пользователя со всеми ресурсами персонального компьютера.

Файловая система

Файловая система, являясь одним из основных элементов операционной системы, представляет собой способ организации хранения файлов в дисковой памяти. Тип файловой системы и организация хранения данных на носителях устройств внешней памяти (накопители на гибких и жестких магнитных дисках) определяют удобство работы пользователя, скорость доступа к файлам, организацию многозадачной работы, возможность создания хороших баз данных.

Драйверы внешних устройств

Поддержка широкого спектра периферийных устройств - важная функция любой операционной системы. Для управления внешними устройствами предназначены короткие программы - драйверы. Драйвер – это программа ОС обслуживающая отдельное периферийное устройство ПК. Каждое устройство располагает своим драйвером, который реализует обмен информации между памятью компьютера и внешним устройством. Драйвер должен учитывать все характеристики и элементы конструкции периферийного устройства.

Драйверы стандартных устройств хранятся в микросхеме флэш-памяти BIOS или в микросхемах, смонтированных на контроллерах устройств ввода-вывода. Драйверы модернизированных или новых устройств представлены отдельными программами, которые при запуске компьютера динамически подключаются к операционной системе.

Виды операционных систем

Операционные системы подразделяются на две большие категории - *стандартные* и *сетевые*. **Стандартные операционные системы**, или операционные системы общего назначения, предназначены для реализации следующих задач.

- Управление аппаратными средствами компьютера.
- Создание рабочей среды и интерфейса пользователя.
- Выполнение команд пользователя и программных инструкций.
- Организация ввода-вывода, хранение информации и управление файлами и данными.

Сетевые операционные системы выполняют функции стандартных операционных систем, а также, дополнительно к ним, реализуют задачи, связанные с управлением работой с файлами, данными и ресурсами, находящи-

мися на различных узлах сети. Сетевые операционные системы позволяют организовать управление работой компьютерной сети и совместный доступ пользователей к сетевым файлам и ресурсам.

Сетевые операционные системы бывают одноранговыми и серверными. Серверные системы отличаются от одноранговых большей сложностью и мощностью и полностью заменяют собой стандартную операционную систему.

Одноранговые операционные системы могут устанавливаться на любой рабочей станции. Серверные операционные системы состоят из двух частей: одна часть располагается на сервере, а другая - на рабочих станциях.

Одноранговые сетевые ОС применяются на ПК самостоятельно в виде отдельных программных средств либо входят в состав пакетов, другую половину которых представляют программы, обслуживающие мощные компьютеры управления сетями, - серверы. Операционные системы, например OS/2 WarpConnect, Windows NT Workstation, Windows for Workgroups, Artisoft LANtastic Network Operating System, Performance Technologies PowerLan - одноранговые.

К числу серверных операционных систем с высокой производительностью и широкими сетевыми возможностями относятся: Novell NetWare, Windows Server 2008, OS/2 LAN Server, OS/2 SMP, VINES, UNIX Ware, LINUX, SC

Исходя из выполняемых функций, ОС можно разбить на группы

- Однозадачные и многозадачные
- Однопользовательские и многопользовательские
- Сетевые и локальные
- Системы с разделением времени и реального времени
- Однопроцессорные и многопроцессорные

Сервисные программы это совокупность программных продуктов, предоставляющих пользователю дополнительные услуги в работе с компьютером и расширяющих возможности операционных систем

- улучшающие пользовательский интерфейс;
- защищающие данные от разрушения и несанкционированного доступа;
- восстанавливающие данные;
- ускоряющие обмен данными между диском и ОЗУ;
- архивации - разархивации;
- антивирусные средства.

Интерфейсные системы являются естественным продолжением операционной системы и модифицируют как пользовательский, так и программный интерфейсы, а также реализуют дополнительные возможности по управлению ресурсами компьютеров. В связи с тем, что развитая интерфейсная система может изменить весь пользовательский интерфейс, часто их также

называют операционными системами. Эго относится, например, к Windows 3.11 и Windows 3.11 for Work Groups (для рабочих групп).

Оболочки операционных систем, в отличие от интерфейсных систем, модифицируют только пользовательский интерфейс, предоставляя пользователю качественно новый интерфейс по сравнению с реализуемым операционной системой. Такие системы существенно упрощают выполнение часто запрашиваемых функций, например, таких операций с файлами, как копирование, переименование и уничтожение, а также предлагают пользователю ряд дополнительных услуг. В целом, программы-оболочки заметно повышают уровень пользовательского интерфейса, наиболее полно удовлетворяя потребностям пользователя. На ПК широко используются такие программы-оболочки, как Norton Commander и DOS Navigator.

Утилиты предоставляют пользователям средства обслуживания компьютера и его ПО. Они обеспечивают реализацию следующих действий:

- обслуживание магнитных дисков;
- обслуживание файлов и каталогов;
- предоставление информации о ресурсах компьютера;
- шифрование информации;
- защита от компьютерных вирусов;
- архивация файлов и др.

Существуют отдельные утилиты, используемые для решения одного из перечисленных действий, и многофункциональные утилиты. В настоящее время для персональных компьютеров среди многофункциональных утилит одним из наиболее совершенных является комплект утилит Norton Utilities. Существуют его версии для использования в среде DOS и Windows.

Прикладное ПО

Как уже отмечалось, приложения, предназначенные для решения целевых задач (первая группа программ в приведенной выше классификации), разрабатываются профессиональными постановщиками задач и программистами. При этом разрабатывать сложную и дорогостоящую программу для решения узкоспециализированной задачи и только для одного объекта, например предприятия, не выгодно. Программа должна обладать определённой степенью универсальности, то есть должна решать поставленную задачу в различных условиях работы предприятия и для различных предприятий. Подобного рода универсальность достигается введением настроечных параметров, которые позволяют настроить программу для условий работы объекта (предприятия). Чем шире сфера применения программы, тем больше и сложнее набор настраиваемых параметров. В наиболее сложных случаях система параметров управления программой может превращаться в специализиро-

ванный язык. Так, например, произошло с бухгалтерской системой 1С . Сначала она имела набор настаиваемых параметров для работы с конкретным предприятием и стандартный список документов. С течением времени предприятия перестали удовлетворять существующие возможности программы. Им хотелось в рамках программы 1С создавать свои документы. Поэтому разработчики ввели в неё специальный язык позволяющий создавать новые документы. Это в свою очередь потребовало введение в штат предприятия (или привлечения специализированных фирм) программистов, умеющих работать с этим языком. Приведенные соображения показывают, что первоначальная идея приобретения предприятием, разработанной профессионалами программы, и дальнейшая её эксплуатация без привлечения ИТ специалистов не работает. К такого рода программным продуктам можно отнести: системы автоматизации документооборота (EDM), аудиторские и бухгалтерские программы, системы логистической поддержки изделий и т.д.

Вторая группа программ предназначена для создания решений пользователями различных уровней квалификации от начинающего пользователя до программиста. Работа с ними требует от пользователя хорошего знания предметной области, для работы с которой предназначено программное средство. В то же время такие ПС обладают возможностью добавления программных кодов пользователя, которые расширяют их функциональность. Но для использования этих инструментов от пользователя требуются навыки программирования. Эта группа программных продуктов представлена: текстовыми процессорами, табличными процессорами, редакторами презентаций, системами автоматизации проектных работ (САПР, CAD), редакторами изображений и т.д.

Третья группа программ предназначена для создания программных решений пользователями, не обладающими профессиональными навыками программиста, но имеющими представление о той предметной области, в которой им нужно создать приложения и знающие основные методы программирования, например, такие как рассмотрены в этом курсе. К таким программным средствам относятся Excel и Access.

Инструментальное ПО

К этому виду ПО относятся средства разработки программ, которые используются для разработки нового программного обеспечения как системного, так и прикладного.

Транслятор языка программирования - это программа, которая переводит текст программы с языка программирования в машинный код.

Комплекс средств, включающих в себя входной язык программирования, транслятор, машинный язык, библиотеки стандартных программ, средства отладки оттранслированных программ и компоновки их в единое целое, называется системой программирования. В системе программирования транслятор переводит программу, написанную на входном языке программи-

рования, на язык машинных команд конкретного компьютера. В зависимости от способа перевода с входного языка (языка программирования) трансляторы подразделяются на компиляторы и интерпретаторы.

В компиляции процессы перевода и выполнения программы разделены во времени. Сначала компилируемая программа преобразуется в набор объектных: модулей на машинном языке, которые затем собираются (компонуются) в единую машинную программу, готовую к выполнению и сохраняемую в виде файла на магнитном диске. Эта программа может быть выполнена многократно без повторной трансляции.

Интерпретатор осуществляет пошаговую трансляцию и немедленное выполнение операторов исходной программы: каждый оператор входного языка программирования транслируется в одну или несколько команд машинного языка, которые тут же выполняются без сохранения на диске. Таким образом, при интерпретации программа на машинном языке не сохраняется и поэтому при каждом запуске исходной программы па выполнение ее нужно (пошагово) транслировать заново. Главным достоинством интерпретатора по сравнению с компилятором является простота.

Особое место в системе программирования занимают Ассемблеры, представляющие собой комплекс, состоящий из входного языка программирования ассемблера и ассемблер-компилятора. Ассемблер представляет собой мнемоническую (условную) запись машинных команд и позволяет получить высокоэффективные программы на машинном языке. Однако его использование требует высокой квалификации программиста и больших затрат времени на составление и отладку программ. Поэтому разработчики программного обеспечения пользуются более удобными и производительными языками. Это языки высокого уровня. Они гораздо понятнее человеку и удобнее для разработки программ.

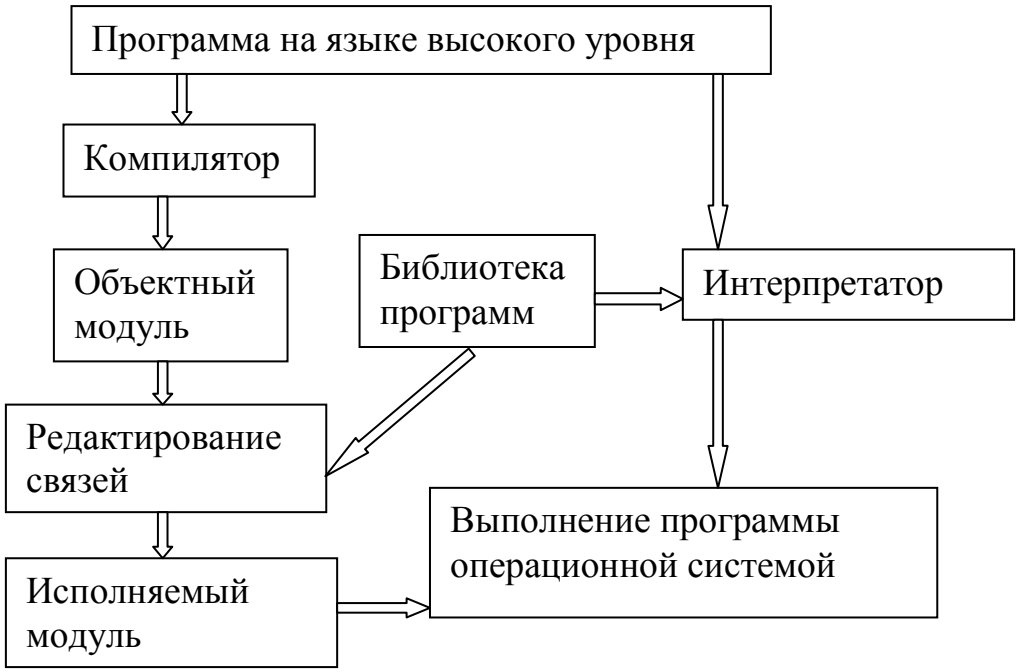
Общие характеристики языков высокого уровня

Языки программирования высокого уровня (в дальнейшем *язык*) были названы так, потому что в отличие от машино - ориентированных языков оперировали ограниченным набором операторов, с помощью которых можно было реализовать любой алгоритм. В общем, каждый из существующих языков имеет следующие группы операторов:

- описания типов данных
- присваивания
- условные операторы
- операторы цикла
- операторы ввода – вывода
- операторы использования подпрограмм и функций

Для реализации всего многообразия возможностей языка используются подпрограммы и функции организованные в библиотеки программ. Отличие

языков только в синтаксисе и количестве и разнообразии программ в библиотеках. Есть и другие различия, но они влияют только на удобство программирования. За всю историю языков программирования их существовало несколько десятков. Начиная с языка Fortran, Cobol [1], которые использовались на в 60 – 70 –х годах прошлого столетия, и до современных языков менялись не только сами языки, но и способы их использования. (Здесь и далее понятия, помеченные верхними индексами, подробно описаны в приложении). Сам язык и соответствующую ему программную среду стали называть *системой программирования*. Системы программирования менялись в зависимости от предметной области, для которой разрабатывалась программа, уровня разработчика программных средств. Так язык Fortran использовался для решения математических и инженерных задач, Cobol – для экономических задач и т.д. Обобщённо систему программирования с использованием языков высокого уровня можно отобразить следующей схемой.



Исполнение программы в режиме компиляции требует больше времени на подготовку программы к исполнению, но затем программа (исполняемый модуль) может многократно выполняться с очень большой скоростью. Важной особенностью в этой цепочке является подключение заранее подготовленных проверенных фрагментов программ, реализующих разнообразные функции и хранящихся в библиотеке программ, к объектному модулю. Богатство библиотеки программ определяет мощность системы программирования для того или иного языка.

В режиме интерпретации каждый оператор исходной программы переводится на машинный язык, при необходимости из библиотеки программ извлекается подпрограмма и запускается процесс её выполнения. Результаты сохраняются и используются при выполнении следующего оператора.

Долгое время системы программирования ориентировались на технологию программирования называемую процедурной.

В такой программе все действия описывались последовательным выполнением операторов программы. При этом характер этих действий можно разделить на две группы: действия связанные с расчётным алгоритмом и отображение интерфейса с пользователем на экране монитора. В дальнейшем это обстоятельство подготовило почву для объектно - ориентированного программирования. Процедурно ориентированный подход был связан с большой трудоёмкостью и необходимостью исправлять большое число ошибок. Попытки сделать структуру программы такой, чтобы снизить количество ошибок, связанных с невнимательностью или забывчивостью программиста, привели к созданию структурного программирования. Примером языка, реализующим эти принципы, является Pascal.

Схема разработки программы состоит из следующих этапов:

1. Постановка задачи
2. Разработка алгоритма
3. Кодирование программы
4. Подготовка тестовых примеров
5. Трансляция (компиляция или интерпретация) программы
6. Отладка программы
7. Анализ ошибок и повторение процесса в зависимости от их типа с пунктов 3,2 или 1
8. Документирование программы
9. Релиз (выпуск программного продукта)

Технология разработки программных продуктов промышленного масштаба является сложным многоэтапным действием и называется *программной инженерией*.

В настоящее время наиболее распространёнными языками программирования являются

- Visual Basic [2]
- C, C++ [3]
- Java [4]
- C# [5]

Использование этих языков программирования связано с объектными технологиями программирования [7]. В дальнейшем все вопросы связанные с программированием мы будем рассматривать с позиций объектно ориентированного подхода.

Системы программирования и связанные с ними технологии исторически использовались для разработки программ, которые функционировали на компьютерах и использовали данные размещённые на том же компьютере или в пределах доступной ему сети. При этом основной формой хранения данных являются базы данных, организующих хранение данных на центральном узле или в распределённой среде. Базы данных хранят данные во

внутренних форматах и программы, использующие их, «понимают» эти форматы. С появлением такой информационной среды как интернет[8] появилась необходимость представлять данные в универсальном формате, основной задачей которого было представлять любые типы данных в виде последовательности символов. Поэтому сначала возникает язык HTML, позволяющий отображать на экране различные визуальные объекты: тексты и изображения. Затем для представления данных в универсальном символьном формате создан язык XML. Создаваемые на их основе WEB формы или сайты позволяли на конечных узлах отображать информацию предоставляемую сервером. С течением времени этого становится недостаточно. Появляется необходимость сделать так, чтобы сайты функционировали как полновесные приложения, то есть работали в интерактивном режиме, позволяя формировать и исполнять запросы по требованию пользователя. Таким образом, к HTML были добавлены возможности включения в WEB формы фрагментов программных кодов (скриптов) на одном из языков программирования. Так, язык Java был специально разработан для этих целей [4]. Язык HTML с такими возможностями стали называть *динамическим HTML*.

Такой широкий спектр требований к создаваемым программным продуктам обусловил появление универсальных систем программирования. К таким системам можно отнести Delphi фирмы Borland и Visual Studio фирмы Microsoft.

Поскольку в дальнейшем все практические примеры программирования будут рассматриваться на программных продуктах фирмы Microsoft, рассмотрим более подробно среду программирования Visual Studio. В основе этой среды лежит анонсированная в 2000 году .Net (читается Dot Net) стратегия [6]. Её идея, вкратце, заключается в следующем. Была разработана огромная структурированная библиотека программ (платформа) Framework (в настоящее время доступна версия 4.0). Каждый из модулей библиотеки описывает класс объектов [7], и может быть использован программами практически на любом из широко распространённых языков. Более полно система рассмотрена в приложении 9.

Основы программирования

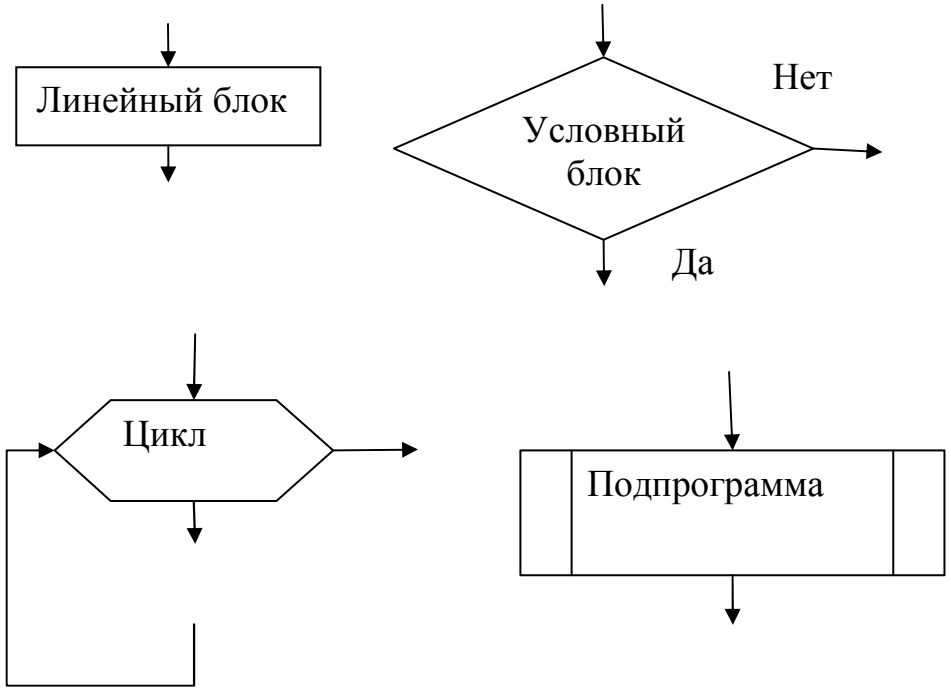
Разработка программ достаточно сложное дело. И сложность заключается в том, что человеку пытающемуся программировать, необходимо поменять способ мышления. Разработка программ требует навыков разложения любого процесса на последовательность этапов, что весьма трудно сделать человеку, который с детства привык мыслить образами. Поэтому обучение программированию требует от человека значительных усилий.

Разработка программы включает три этапа:

- Разработка алгоритма
- Написание программы на языке программирования

- Отладка программы

На первом из них необходимо разделить процесс решения задачи на последовательность этапов с учётом типовых процедур, используемых в языках программирования: присваивание, циклы и условные операторы. Это один из самых творческих и трудных этапов. Наиболее известным способом изображения алгоритма является блок схема. Блок – схема это графическое изображение алгоритма. Основными элементами блок схем являются следующие.

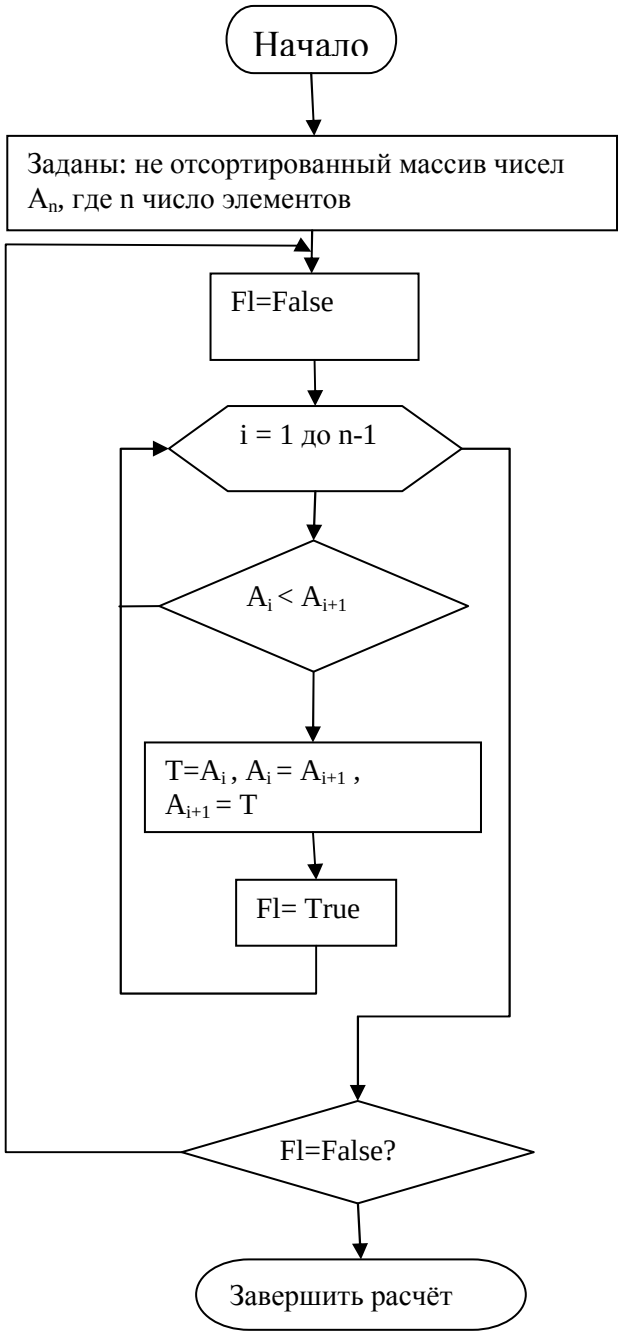


Описание алгоритма с помощью блок схем имеет следующие преимущества.

- Алгоритм не зависит от языка программирования
- Процессы описываются более наглядно
- Переход от блок схемы к программе на любом языке программирования становится формальной процедурой
- С помощью блок схем удобно описывать сложные алгоритмы

Широкое распространение блок схемы получили при процедурном программировании, когда программы создавались в виде единого модуля довольно большого размера. С переходом к объектно – ориентированному программированию роль блок-схем снизилась, так как вместо одного большого модуля программы, программисту приходится писать много небольших фрагментов реализующих отклики на события происходящие с объектами. И в дальнейшем все примеры будут рассматриваться применительно к объектно ориентированному подходу. Но прежде рассмотрим пример описания алгоритма с помощью блок схемы.

Пусть требуется отсортировать массив чисел в порядке убывания. Блок схема одного из алгоритмов для решения этой задачи будет выглядеть так.



Эта блок схема описывает следующие действия. Предполагается, что в памяти имеется массив чисел A, содержащий n неупорядоченных чисел. Далее, устанавливаем переменную Fl (флажок) в состояние False (Ложь). Эта переменная регистрирует: были ли произведены обмены данными при просмотре массива чисел. Следующий блок организует цикл по перебору чисел массива от 1 до n-1. Далее следует проверка: i-е число меньше i+1 го. Если да, то i-й и i+1 – й элементы нужно поменять местами, что делается в следующем блоке. В противном случае переходим к следующему элементу массива. Если обмен элементов имел место, то флажок F1 устанавливаем в значение Истина (True). После завершения цикла, когда просмотрены все элементы массива, проверяем значение F1. Если оно равно значению Ложь

(False), сортировка закончена. Иначе цикл повторяется. После выполнения алгоритма числа в массиве будут отсортированы в убывающем порядке.

На языке Basic этот алгоритм будет выглядеть так.

```
Dim n As Integer, i As Integer T As Integer
Dim A(n) As Integer
Dim F1 As Boolean
NewCicl:
F1= False
For i= 1 To n-1
   If A(i) < A(i+1) Then
      T=A(i)
       A(i)=A(i+1)
       A(i+1)=T
       F1=True
  End If
Next
If  F1= True Then GoTo NewCicl
….
….
```

Более подробно язык Basic рассматривается в приложении 11. Однако здесь вкратце рассмотрим основные параметры языка, а затем особенности его использования в объектно ориентированном программировании. В общем, любой язык программирования состоит из 5 групп операторов:

1. Операторы описания или декларирования переменных;
2. Операторы присваивания;
3. Условные операторы;
4. Операторы цикла;
5. Операторы описания процедур и функций.

Операторы описания переменных имеют дело с переменными. Переменная это поименованное место в памяти компьютера, предназначенное для временного хранения данных. Переменная характеризуется уникальным именем и типом. Уникальность имени заключается в том, что оно не должно повторяться в программе, в которой определена. Тип переменной показывает, какие данные будут храниться в переменной. По типу переменной определяются операции, которые могут над ней производиться. Так над переменными, описанными как числовые (целые, с плавающей точкой, денежные), могут выполняться все арифметические операции; над переменными типа даты могут выполняться операции вычитания одной даты из другой, вычитание и сложение даты с целым числом; над символьными переменными могут выполняться операции объединения (конкатенации) и разъединения текстов. Переменные играют роль записной книжки при проведении сложных расчётов.

Операторы присваивания позволяют заносить в память переменных различные значения. Синтаксис оператора следующий:

<переменная> = <выражение>

Знак равенства это и есть оператор присваивания. Он читается как - *присвоить значение*. Действие его следующее: сначала вычисляется выражение стоящее справа от знака присваивания, затем полученное значение переносится в переменную слева от него. В процессе выполнения оператора могут возникать различные ситуации. Так, если переменной имеющей тип с плавающей точкой мы присвоим целое число, то в процессе выполнения оператора будет выполнено неявное преобразование формата числа. Однако, если переменной целого типа будет присвоено значение числа с плавающей точкой, оператор выдаст ошибку, так как непонятно, куда девать дробную часть числа. С другой стороны, числовой переменной можно присвоить символьное значение, если оно содержит набор цифровых символов и знаки. Но для этого нужно использовать явное преобразование.

Условные операторы позволяют выполнять ту или иную группу операторов в зависимости от значения некоторого выражения. Синтаксис оператора для языка Basic следующий:

If <условное выражение> Then
<1 я группа операторов>
Else
<2 я группа операторов >
End If

Оператор выполняется следующим образом. Вычисляется условное выражение, если его результат –«Истина», то выполняется первая группа операторов, иначе вторая. Оператор может иметь сокращённые формы. Например, такие.

If <условное выражение> Then
< группа операторов>
End If

If <условное выражение> Then < группа операторов>

И в том и в другом случае, если условное выражение – «Истина», то выполняется группа операторов, иначе следующие операторы. Условный оператор существует во всех языках высокого уровня. Например, в C# он имеет следующий синтаксис.

if (<условное выражение>)
{ Блок операторов 1}
else
{ Блок операторов 2}

Как видно они очень похожи.

Операторы цикла позволяют повторять одну и ту же группу операторов несколько раз с разными параметрами. Существует несколько типов операторов цикла. Здесь рассмотрим один из них. Это цикл For с заданным числом

повторений. За остальными отсылаю к приложению 11. Рассмотрим простейший пример цикла.

```
For i = 1 To 10
   MsgBox (i)
Next i
```

Операторы *For* и *Next* обозначают начало и конец цикла, i – это управляющая переменная цикла, которая может принимать значения от 1 до 10 с шагом 1. MsgBox (i) это оператор, который выводит на экран значение переменной i. Вместо него могут располагаться несколько операторов, которые будут выполняться для разных значений управляющей переменной. Если значения i нужно менять с шагом 2, фрагмент кода цикла будет выглядеть так.

```
For i = 1 To 10 Step 2
      MsgBox (i)
Next i
```

Кроме этого типа цикла используют циклы с предусловиями и циклы с постусловиями. Приведу пример первого из них.

```
A = 1
While A < 10
   A = Int(Rnd() * 20)
   MsgBox A
Wend
```

Цикл начинается с ключевого слова While (пока) и заканчивается Wend. Семантика цикла такова. В начале цикла проверяется условие. В данном случае это A < 10. Если оно выполняется, а это действительно так – A в предыдущем операторе установлено равным 1, то выполняются операторы тела цикла. Эти операторы должны включать действия связанные с изменением переменных, входящих в условие. В примере это получение случайного числа от 0 до 1 оператором Rnd(), умножение его на 20 и преобразование в целое число. А так же вывод на экран получившегося числа A (оператор MsgBox A). Если в процессе выполнения тела цикла образуется A больше или равно 10, то цикл будет завершён. Цикл с постусловием имеет тот же смысл, но условие завершения цикла проверяется не в начале, а в конце цикла. Синтаксис цикла поясняется следующим примером.

```
Dim var_A
Do
   var_A = InputBox("Введите число")
Loop Until IsNumeric(var_A)
```

Здесь объявлена переменная типа variant , которая может принимать данные любого типа. Цикл начинается ключевым словом Do (выполнить) и заканчивается выражением Loop Until , за которым следует условное выражение. Если результат вычисления условного выражения ИСТИНА, то цикл

завершается, иначе повторяется снова. В примере в качестве условного выражения используется функция IsNumeric(var_A) , которая проверяет – является ли введённый текст числом. Если да, то цикл заканчивается, иначе запрос на ввод числа повторяется. Функция *InputBox("Введите число")* выводит на экран подсказку и ждёт ввода данных. После завершения ввода, введённое значение присваивается переменной *var_A* .

Подпрограммы содержат операторы, которые так же могут выполняться многократно, но только при обращении к подпрограмме. Обращение к подпрограмме может быть выполнено из любой части программы. Особенностью подпрограмм является то, что при обращении к подпрограмме в неё передаются параметры. Параметры, как правило, представляют собой список переменных разных типов, значения которых используются в подпрограмме для расчётов. Параметры заданные при описании подпрограммы называются *формальными*, а параметры определённые для обращения к подпрограмме – *фактическими*. Подпрограммы делятся на *процедуры* и *функции*. Процедуры это подпрограммы, которые обмениваются данными с вызывающей программой только через параметры. Функции это подпрограммы, которые получают значения через параметры, а возвращают только одно значение, как правило, описанное переменной, являющейся именем функции. Приведу пример процедуры.

```
Public Sub UserInput(UserNumber As Integer)
    MsgBox _
    ("Здравствуйте, вы пользователь № " + _
    Str(UserNumber))
End Sub
```

Здесь *Sub* и *End Sub* начало и конец процедуры, *UserInput* - имя процедуры, а *UserNumber As Integer* – описание переменной формального параметра. А модификатор доступа Public означает, что эту процедуру можно вызвать из любого модуля проекта. Существуют и другие модификаторы доступа, в частности, Private. Он дает доступ к процедуре только из того модуля, где она объявлена. Обращение к процедуре может выглядеть так *UserInput (1)*. В результате такого обращения оператор в теле процедуры выведет на экран сообщение: *Здравствуйте, вы пользователь № 1*

В качестве функции рассмотрим пример вычисления квадрата числа. Описание функции будет выглядеть так.

```
Public Function Square(num_One As Double) As Double
   Square = num_One ^ 2
End Function
```

Здесь *Function* и *End Function* начало и конец описания функции, *Square* имя функции, *num_One As Double* описание переменной формального параметра. В отличие от процедуры здесь появляется описание типа функции *As Double*, то есть имя функции рассматривается как переменная с плавающей запятой двойной точности. Поэтому в теле функции появляется опера-

тор, который присваивает имени функции результат возведения в квадрат параметра функции. Обращение к функции может выглядеть так.

```
Dim mum_Rez  As Double
mum_Rez=Square(2.3)
```

Одной из важнейших особенностей подпрограмм, как процедур, так и функций является то, что область «видимости» переменной ограничено подпрограммой. То есть переменная, если она не объявлена с модификатором Public, за пределами подпрограммы недоступна. Это позволяет использовать процедуры и функции для решения автономных задач. Особенно наглядно это проявляется при рекурсивном использовании функций. Например, нужно вычислить факториал числа A=n!, что равнозначно A=1*2*3*….*n . Вычисление факториала можно реализовать следующей рекурсивной функцией.

```
Function Factorial(n As Integer) As Integer
   If n=1 Then
      Factorial = 1
   Else
      Factorial = n* Factorial(n -1)
   End If
End Function
```

Для проверки работы функции обратимся к ней с помощью выражения A= Factorial(4). Обращение к функции приводит к последовательному выполнению выражений

```
Factorial = 4* Factorial(3)
Factorial = 3* Factorial(2)
Factorial = 2* Factorial(1)
```

При последнем обращении функция вернёт Factorial = 1 и далее в обратном порядке.

Объектно-ориентированное программирование

Основные понятия объектно-ориентированного программирования (ООП). Основополагающей идеей одного из современных подходов к программированию — объектно-ориентированному — является объединение данных и обрабатывающих их процедур в единое целое — объекты.

Объектно-ориентированное программирование — это методология программирования, которая основана на представлении программы в виде совокупности объектов, каждый из которых является реализацией определенного класса (типа особого вида), а классы образуют иерархию, основанную на принципах наследуемости. При этом объект характеризуется как совокупностью всех своих свойств и их текущих значений, так и совокупностью допустимых для данного объекта действий.

Несмотря на то что в различных источниках делается акцент на те или иные особенности внедрения и применения ООП, три основных (базовых) понятия ООП остаются неизменными. К ним относятся:

• наследование (Inheritance);
• инкапсуляция (Encapsulation);
• полиморфизм (Polymorphism).

Эти понятия как три кита лежат в основе ООП. При процедурном подходе требуется описать каждый шаг, каждое действие алгоритма для достижения конечного результата. В отличие от него объектно-ориентированный подход оставляет за объектом право решать, как отреагировать и что сделать в ответ на поступивший вызов. Достаточно в стандартной форме поставить перед ним задачу и получить ответ. Объект состоит из следующих трех частей:

• имени объекта;
• состояния (переменных состояния);
• методов (операций).

Можно дать обобщающее определение: объект ООП— это совокупность переменных состояния и связанных с ними методов (операций). Упомянутые методы определяют, как объект взаимодействует с окружающим миром.

Под методами объекта понимают процедуры и функции, объявление которых включено в описание объекта и которые выполняют действия. Возможность управлять состояниями объекта посредством вызова методов в итоге и определяет поведение объекта. Совокупность методов часто называют интерфейсом объекта.

Инкапсуляция — это механизм, который объединяет данные и методы, манипулирующие этими данными, и защищает и то и другое от внешнего вмешательства или неправильного использования. Когда методы и данные объединяются таким способом, создается объект.

Применяя инкапсуляцию, мы защищаем данные, принадлежащие объекту, от возможных ошибок, которые могут возникнуть при прямом доступе к этим данным. Кроме того, применение указанного принципа очень часто помогает локализовать возможные ошибки в коде программы. А это намного упрощает процесс поиска и исправления этих ошибок. Можно сказать, что инкапсуляция обеспечивает сокрытие данных, что позволяет защитить эти данные. Однако применение инкапсуляции ведет к снижению эффективности доступа к элементам объекта. Это обусловлено необходимостью вызова методов для изменения внутренних элементов (переменных) объекта. Но при современном уровне развития вычислительной техники подобные потери в эффективности не играют существенной роли.

Наследование — это процесс, посредством которого один объект может наследовать свойства другого объекта и добавлять к ним черты, характерные только для него. В итоге создается иерархия объектных типов, где

поля данных и методов «предков» автоматически являются и полями данных и методов «потомков».

Смысл и универсальность наследования заключаются в том, что не надо каждый раз заново («с нуля») описывать новый объект, а можно указать «родителя» (базовый класс) и описать отличительные особенности нового класса. В результате новый объект будет обладать всеми свойствами родительского класса плюс своими собственными отличительными особенностями.

Пример 1. Можно создать базовый класс «транспортное средство», который универсален для всех средств передвижения, к примеру, на четырех колесах. Этот класс «знает», как двигаются колеса, как они поворачиваются, тормозят и т.д. Затем на основе этого класса создадим класс «легковой автомобиль». Поскольку новый класс унаследован из класса «транспортное средство», унаследованы все особенности этого класса, и нам не надо в очередной раз описывать, как двигаются колеса и т. д. Мы просто добавим те черты, которые характерны для легковых автомобилей. В то же время мы можем взять за основу этот же класс «транспортное средство» и построить класс «грузовые автомобили». Описав отличительные особенности грузовых автомобилей, получим новый класс «грузовые автомобили». А, к примеру, на основании класса «грузовой автомобиль» уже можно описать определенный подкласс грузовиков и т.д. Таким образом, сначала формируем простой шаблон, а затем, усложняя и конкретизируя, поэтапно создаем все более сложные шаблоны.

Полиморфизм — это свойство, которое позволяет одно и то же имя использовать для решения нескольких технически разных задач. Полиморфизм подразумевает такое определение методов в иерархии типов, при котором метод с одним именем может применяться к различным родственным объектам. В общем смысле концепцией полиморфизма является идея «один интерфейс — множество методов». Преимуществом полиморфизма является то, что он помогает снижать сложность программ, разрешая использование одного интерфейса для единого класса действий. Выбор конкретного действия в зависимости от ситуации возлагается на компилятор.

Пример 2. Пусть есть класс «автомобиль», в котором описано, как должен передвигаться автомобиль, как он поворачивает, как подает сигнал и т.д. Там же описан метод «переключение передачи». Допустим, что в этом методе класса «автомобиль» описана автоматическая коробка передач. А теперь необходимо описать класс «спортивный автомобиль», у которого механическое (ручное) переключение скоростей. Конечно, можно было бы описать заново все методы для класса «спортивный автомобиль». Вместо этого указываем, что класс «спортивный автомобиль» унаследован из класса «автомобиль», а следовательно, он обладает всеми свойствами и методами, описанными для класса-родителя. Единственное, что надо сделать — это переписать метод «переключение передач» для механической коробки передач. В ре-

зультате при вызове метода «переключение передач» будет выполняться метод не родительского класса, а самого класса «спортивный автомобиль».

Механизм работы ООП в таких случаях можно описать примерно так: при вызове того или иного метода класса сначала ищется метод в самом классе. Если метод найден, то он выполняется, и поиск этого метода завершается. Если же метод не найден, то обращаемся к родительскому классу и ищем вызванный метод в нем. Если он найден, то поступаем, как при нахождении метода в самом классе. А если нет, то продолжаем дальнейший поиск вверх по иерархическому дереву, вплоть до корня (верхнего класса) иерархии. Этот пример отражает так называемый механизм раннего связывания.

В настоящее время процедурное программирование практически не встречается. Ему на смену пришёл объектно-ориентированный подход к разработке программ (см. приложение 7). Практически все продукты MS Office используют этот подход для программных реализаций. Программирование в среде MS Office хотя и является объектно-ориентированным, но носит упрощённую форму. Существуют среды для профессиональных программистов, принципы которых изложены в приложении 9. Далее коснусь основных методов объектно-ориентированного подхода в MS Office. Их основы подробно излагаются в приложении 10.

В отличие от процедурного программирования объектно - ориентированный подход к разработке программ предполагает не программную реализацию алгоритма, а написание фрагментов программных кодов, обслуживающих объекты. Под объектом понимается либо некоторое экранное изображение, несущее функциональную нагрузку, либо программный код, описывающий логически завершённую совокупность действий. Их принято называть элементами управления (ЭУ). Каждый элемент управления имеет набор свойств, которые характеризуют его вид; данные, которые он отражает, а так же набор событий, которые с ним могут происходить. Кроме того с ЭУ связаны методы, которые выполняют стандартные действия с объектом. С точки зрения программной реализации каждый объект описывается специальной программой, которая называется *классом*. В классе описывается сам объект, его свойства и методы. Большинство существующих программных систем построено с помощью классов. При этом классы представлены в виде откомпилированных программ, собранных в файлах называемых библиотеками программ. В системах программирования MS Office установлены ссылки на эти библиотеки, благодаря чему может быть использован любой из классов библиотеки. В этих системах, в основном, используются классы созданные разработчиками. Однако можно создавать новые классы и добавлять их в библиотеку программ. Для профессиональных разработок фирмой Microsoft были разработаны хорошо структурированные библиотеки, которые были объединены в единый каркас среды разработки Framework .Net, рассмотренный в приложении 9.

Реализация объектно – ориентированного подхода в MS Excel

Для работы с объектами используют объектные переменные. Они объявляются так же как и простые и служат для временного хранения объектов. Главным объектом при программировании в среде MS Office является Application (приложение). Этот объект имеет набор свойств и методов, принципы работы с которыми для Excel следующие.

Microsoft Office Excel - это популярные электронные таблицы. Обычно, программируя для этой программы, преследуют такие цели:

- Автоматизация вычислений.
- Автоматизация ввода и обработки информации.
- Работа с простейшими базами данных - вывод, ввод, анализ, визуализация информации.
- Анализ финансовой и другой информации.
- Создание систем для организации автоматизированного ввода данных
- Математическое моделирование.

В Excel рабочая область листа разбита на ячейки, каждая из которых имеет собственное имя. Имена ячеек могут быть двух видов.

- Первый вид (стиль A1) - это имя, состоящее из буквенного имени столбца и номера строки. Например, A1 - ячейка, находящаяся на пересечении столбца A (первого) и первой строки.
- Другой вид - это индексы ячеек (стиль именования R1C1). Для адресации ячейки в таком стиле указывают номер строки (R - Row - строка) и номер столбца (C - Column - столбец), на пересечении которых расположена ячейка. Строки изначально пронумерованы, а номера столбцов начинаются с 1 - первому столбцу соответствует столбец A, второму - B и т.д. Например, (2, 3) - это адрес ячейки, находящийся на пересечении второй строки и третьего столбца, то есть, если переложить это на стиль A1 - получим ячейку C2 .

Для выполнения большинства операций в MS Excel применяются следующие объекты.

- Excel.Application (Приложение) - объект, представляющий приложение Microsoft Excel.
- Workbook (Рабочая книга) - представляет рабочую книгу - аналог документа Microsoft Word.
- Worksheet (Рабочий лист) - книга в MS Excel разбита на рабочие листы. Именно на листе расположены ячейки, которые могут хранить информацию и формулы.
- Range (Диапазон) - может быть представлен в виде одной ячейки или группы ячеек. Работа с листом удобна - чтобы работать с ка-

кой-либо ячейкой, надо знать лишь ее имя (в формате A1) или адрес (R1C1).

- QueryTable (Таблица запросов) - этот объект используют для импорта в Microsoft Excel информации из баз данных. Подключение к базе данных, запрос информации и т.д. производятся средствами объекта, а итоги запроса выгружаются на лист MS Excel в виде обычной таблицы.
- PivotTable (Сводная таблица) - это особый вид электронной таблицы Excel - она позволяет в интерактивном режиме обобщать и анализировать большие объёмы информации, в частности, взятой из базы данных.
- Chart (Диаграмма) - представляет собой диаграмму. Обычно их используют для визуализации данных.

В основном нам придётся иметь дело с объектом Excel.Application . Этот объект имеет следующие свойства:

- активные объекты - ActiveCell, ActiveChart, ActivePrinter, ActiveSheet, ActiveWindow, ActiveWorkbook
- наборы объектов и коллекции - Cells, Columns, Rows, Sheets, Workbooks, Worksheets, Names

Это только некоторые из свойств объекта Excel.Application. При использовании свойства отделяются от объекта точкой. Например, адрес активной ячейки даёт выражение `Application.ActiveCell.Address` . Смысл этого выражения следующий: объект – *активная ячейка* внутри объекта - *приложение* имеет свойство `Address,` которое возвращает адрес (номер строки и столбца) ячейки. Под активной ячейкой понимается ячейка выделенная пользователем. Так же понимается и свойство `Value` , но оно возвращает значение, а не адрес ячейки. Использование этих свойств показано в следующем примере:

```
MsgBox ("В ячейке с именем " + Application.ActiveCell.Address + " хранится значение " +  Application.ActiveCell.Value)
ActiveCell.Value = InputBox("Введите новое значение для ячейки " + ActiveCell.Address)
```

Как видно из примера, указанные свойства могут участвовать в выражениях, как справа, так и слева от знака присваивания. К наиболее распространённым свойствам объектов в Excel относятся Range (интервал), Cells (ячейки), Columns(столбцы), Rows(строки), Sheets(страницы). Эти свойства возвращают соответствующие наборы объектов и коллекции. Определим их основное предназначение.

- `Cells, Columns, Rows` - возвращают наборы объектов `Range`, содержащие, соотвественно, ячейки, столбцы, строки. При вызове этих свойств можно указывать, какие именно объекты нужно возвратить, а можно, вызвав без параметров, получить все объекты нужного вида.
- `Sheets`, `Worksheets` - возвращают коллекции, которые содержат листы активной книги. В коллекции `Sheets` будут содержаться листы, которые содержат диаграммы и обычные листы, а в коллекции `Worksheets` - лишь обычные листы.

- `Workbooks` - возвращает коллекцию открытых книг.
- `Names` - возвращает коллекцию именованных диапазонов.

Рассмотрим более подробно объект `Range` .Обращение к нему возвращает ячейку или диапазону ячеек, указанных при его вызове. Собственно говоря, `Range` наряду с `Cells` - это основные инструменты для работы с ячейками *листа*. Например, так можно вставить число 4 в ячейку E1:

```
ActiveSheet.Range("E1") = 4
```

А такая конструкция позволяет прибавить по 1 к значению каждой из ячеек диапазона A1:K100

```
Dim MyCell As Variant
For Each MyCell In ActiveSheet.Range("A1:K100")
    MyCell.Value = MyCell.Value + 1
Next
```

При использовании конструкции `For-Each` обход ячеек осуществляется по строкам.

Можно адресовать *ячейку* или диапазон ячеек, указав их адреса в стиле `A1`. Здесь и далее мы используем метод `Select` *объекта* `Range`, который выделяет *ячейки*

```
ActiveSheet.Range("A2").Select
```

Для обращения к диапазону ячеек нужно знать верхнюю правую и нижнюю левую границы диапазона. Например, для обращения к диапазону высотой в одну строку от `A2` до`E2` или к диапазону `A2:E4` - понадобится такой код

```
ActiveSheet.Range("A2:E2").Select
ActiveSheet.Range("A2:E4").Select
```

Можно воспользоваться конструкцией с использованием объекта `Cells`, который позволяет обращаться к отдельной *ячейке* по ее индексу в формате `R1C1`. Чтобы обратиться к *ячейке* A5 таким способом, нужно заметить, что она расположена в пятой строке и первом столбце:

```
ActiveSheet.Cells(5,1).Select
```

Можно объединить использование `Range` и `Cells`, указав координаты ячеек при адресации диапазона с помощью `Cells` .

```
ActiveSheet.Range(Cells(5, 4), _
    Cells(7, 5)).Select
```

Нам уже встречалось использование `Cells` для доступа к группам ячеек в цикле - в качестве индексов ячеек можно использовать переменные

```
For i = 1 To 3
    For j = 1 To 3
        ActiveSheet.Cells(i, j).Select
        Application.Wait (Now + _
        TimeValue("0:00:01"))
        p = p + 1
        Selection = p
    Next j
Next i
ActiveSheet.Range("A1:E5").Clear
```

Здесь мы циклически выделяем *ячейки* диапазона `A1:C3`, делая задержку на 1 секунду после каждого выделения и выводя количество прошедших с начала работы программы секунд. Здесь мы воспользовались для выделения *ячейки* уже знакомым вам мето-

дом `Select`, а для ввода данных в выделенную *ячейку* применили объект `Selection`, который в данном случае ссылается на выделенную *ячейку*. В конце мы очистили диапазон `A1:E5` от введенных данных.

Приложение

1

Несмотря на то, что было создано несколько сотен языков высокого уровня, лишь немногие из них получили широкое признание. В середине 1950-х годов корпорация IBM создала язык Fortran для написания научных и инженерных приложений, выполняющих сложные математические расчеты.

В конце 1950-х годов группа производителей компьютеров совместно с некоторыми правительственными и коммерческими организациями создала язык COBOL. Он используется преимущественно в экономических приложениях, где требуются манипуляции большими объемами данных. Значительная часть современного программного обеспечения с экономической направленностью создана на COBOL.

2

Visual Basic является результатом развития языка программирования BASIC, разработанного в середине 60-х годов XX века преподавателями Дартмутского колледжа Джоном Кемени и Томасом Куртцем BASIC был задуман как язык, позволяющий легко и быстро писать простые программы, и его основным предназначением было обучение новичков базовым навыкам программирования.

Когда Билл Гейтс основал в 1970-е годы корпорацию Microsoft, он реализовал BASIC на ранних моделях персональных компьютеров. В конце 80-х и начале 90-х годов прошлого века корпорация Microsoft разработала графический пользовательский интерфейс Microsoft Windows, представляющий собой видимую часть операционной системы, с которой пользователи могут взаимодействовать. С появлением графического интерфейса Windows язык BASIC естественным образом эволюционировал в Visual Basic, который был представлен корпорацией Microsoft в 1991 году для облегчения программирования Windows приложений.

До этого программирование под Windows было довольно трудоемким процессом. Современный Visual Basic является объектно-ориентированным языком визуального программирования, управляемым событиями. Программы на нем создаются с помощью специального программного инструмента, называемого интегрированной средой разработки. Пользуясь интегрированной средой Microsoft Visual Studio, вы можете легко и быстро писать, выполнять, тестировать и отлаживать программы на Visual Basic.

Последняя версия Visual Basic является полностью объектно-ориентированной. Visual Basic управляется событиями. Это означает, что написанные в программы реагируют на события, исходящие от пользователя, такие как щелчки мыши нажатие клавиш и показания таймеров. Это язык визуального программирования, т. е. можете не только писать операторы языка при разработке программы, но и пользоваться графическим интерфейсом Visual Studio и перетаскивать на рабочее поле стандартные объекты (например, кнопки или текстовые поля), а затем снабжать их надписями и изменять их размер. Значительную часть кода, реализующего пользовательский интерфейс, за вас пишет среда Visual Studio.

3

Язык C, разработанный Деннисом Ричи (Dennis Ritchie) из Bell Laboratories в 1970-х, получил широкое распространение как язык, на котором была написана операционная система UNIX. Язык C++, представляющий собой дальнейшее развитие C, был создан Бьярном Страуструпом (Bjarne Stroustrup) из Bell Laboratories в начале 1980-х. Этот язык обладает возможностями объектно-ориентированного программирования (ООП). Многие современные важнейшие операционные системы (например, Microsoft Windows) написаны на C или C++.

4

В начале 1990-х годов многие организации, в том числе Sun Microsystems, предсказывали резкий скачок на рынке интеллектуальной потребительской электроники, т. е. бытовых приборов со встроенными микропроцессорами (электронными устройствами, благодаря которым работают компьютеры). Однако рынок развивался не так быстро, как ожидала компания Sun. Когда в 1993 году произошел "взрыв" популярности Всемирной паутины, компания Sun поняла перспективность своего нового языка программирования Java, предназначенного для создания интерактивного анимированного содержимого Web-страниц. Язык Java был анонсирован в 1995 году и сразу привлек внимание деловых кругов благодаря широчайшему интересу к Всемирной паутине. В настоящее время разработчики используют Java для написания Web-страниц с динамическим содержимым (содержимым, генерируемым в ответ на действия пользователя), для создания крупных коммерческих приложений, для расширения функциональности Web-серверов (компьютеров, предоставляющих содержимое вашему браузеру, когда вы посещаете Web-сайты), для создания приложений, работающих в бытовой электронике (например, в сотовых телефонах, пейджерах и портативных компьютерах), а также для многих других целей.

5

В 2000 году корпорация Microsoft анонсировала язык C# (произносится "си-шарп") одновременно со своей стратегией .NET. Язык программирования C# был разработан специально для платформы .NET. Он уходит корнями в C, C++ и Java, унаследовав от них самые лучшие черты. Как и Visual Basic, язык

C# является объектно-ориентированным. Он предоставляет программисту доступ к обширной библиотеке стандартных компонентов .NET, что Ускоряет процесс разработки приложений. Языки C#, Java и Visual Basic имеют примерно одинаковые функциональные возможности.

6

В июне 2000 года корпорация Microsoft анонсировала .NET-инщиативу (www.microsoft.com/net)l новое широкое видение использования Интернета и Всемирной паутины в проектировании, создании, распространении и применении программного обеспечения. Не принуждая программистов к работе с каким-то одним языком программирования, .NET-инициатива позволяет им создавать приложения на любом из .NET-совместимых языков (в число которых входят Visual Basic, Visual C++, Visual C# и др.). Частью этой инициативы является технология ASP. NET позволяющая создавать Web-приложения (приложения, работающие во Всемирной паутине). Стратегия .NET является распространением идеи многократно используемого программного обеспечения до масштабов Интернета. Она позволяет программистам сосредоточиться на специальных задачах, не отвлекаясь на реализацию всех компонентов приложения. Визуальное программирование стала популярным благодаря тому, что оно позволяет без лишнего труда создавать Web- и Windows-приложения, пользуясь готовыми графическими компонентами, такими как кнопки, текстовые поля и полосы прокрутки.

В основе .NET-стратегии Microsoft находится платформа .NET Framework. Она служит средой выполнения приложений и Web-служб, содержит библиотеку классов (называемую Framework Class Library) и предоставляет программисту множество других технических возможностей.

7

Объектная технология — это схема осмысленных программных модулей. Бывают объекты даты и времени, объекты-документы, объекты-автомобили, объекты-люди, аудиообъекты, видеообъекты, объекты-файлы, объекты-записи и т. д. Практически любое существительное может быть приемлемым представлено в виде программного объекта. Объекты имеют свойства (также называемые атрибутами), например, цвет, размер и вес; они могут выполнять действия (также называемые моделями поведения или методами). Классы — это типы сходных объектов. Например, все автомобили принадлежат классу "автомобиль", хотя отдельные экземпляры различаются по изготовителю, цвету и пакету опций. Класс описывает общий формат своих объектов, а свойства вия, специфичные для объекта, зависят от его класса. Объект имеет к классу пример отношение, какое имеет здание к чертежам, по которым оно возведено.

До появления объектно-ориентированных языков процедурные языки программирования такие как Fortran, Pascal, BASIC и C) были нацелены на

действия (глаголы), а не на объекты (существительные). Это отчасти затрудняло программирование. Однако с помощью современных языков, таких как Visual Basic, C++, Java и C#, вы можете программировать в объектно-ориентированном стиле, который более естественно отражает наше восприятие мира. В результате значительно повышается производительность труда программиста.

Когда применяется объектная технология, правильно спроектированные классы могут быть многократно использованы в разных приложениях. Библиотеки классов заметно уменьшают объём работы при создании новых систем. Некоторые фирмы заявляют, что главная выгода, которую они имеют от объектно-ориентированного программирования, на самом деле состоит не в возможности многократно использовать код, а в создании продукта, который лучше организован, более понятен и прост в сопровождении.

Ориентированность на объекты позволяет вам сосредоточиться на "общей картине. Вы не беспокоитесь о частных деталях реализации многократно используемых объектов, а направляете свои усилия на поведение объектов и взаимодействие между ними.

Очевидно, что в ближайшие десятилетия объектно-ориентированное программирование будет основной методологией написания приложений. Visual Basic является одним из самых распространенных объектно-ориентированных языков.

Объектно-ориентированный подход к программированию

Важнейшим шагом на пути к совершенствованию языков программирования стало появление *объектно-ориентированного подхода* к программированию (или, сокращенно, *ООП*) и соответствующего класса языков.
При ***объектно-ориентированном подходе*** программа представляет собой описание ***объектов***, их *свойств* (или ***атрибутов***), совокупностей (или ***классов***), отношений между ними, способов их взаимодействия и операций над *объектами* (или ***методов***).
Несомненным преимуществом данного подхода является концептуальная близость к предметной области произвольной структуры и назначения. Механизм ***наследования*** *атрибутов* и *методов* позволяет строить производные понятия на основе базовых и таким образом создавать модель сколь угодно сложной предметной области с заданными свойствами.
Еще одним теоретически интересным и практически важным свойством *объектно-ориентированного подхода* является поддержка механизма ***обработки событий***, которые изменяют *атрибуты объектов* и моделируют их взаимодействие в предметной области.
Перемещаясь по *иерархии классов* от общих понятий предметной области к более конкретным (или от более сложных – к более простым) и наоборот, программист получает возможность изменять степень абстрактности или конкретности взгляда на моделируемый им реальный мир.
Использование ранее разработанных (возможно, другими коллективами про-

граммистов) библиотек *объектов* и *методов* позволяет значительно сэкономить трудозатраты при производстве программного обеспечения, в особенности, типичного.
Объекты, *классы* и *методы* могут быть полиморфными, что делает реализованное программное обеспечение более гибким и универсальным.
Сложность адекватной (непротиворечивой и полной) формализации *объектной* теории порождает трудности тестирования и верификации созданного программного обеспечения. Пожалуй, это обстоятельство является одним из самых существенных недостатков *объектно-ориентированного подхода* к программированию.
Наиболее известным примером *объектно-ориентированного* языка программирования является язык C++, развившийся из императивного языка C. Его прямым потомком и логическим продолжением является язык C#, который изучается в данном курсе. Другие примеры *объектно-ориентированных* языков программирования: Visual Basic, Java, Eiffel, Oberon.
Переход от структурно-процедурного подхода к *объектно-ориентированному* программированию, подобно переходу от низкоуровневых языков программирования к языкам высокого уровня, требует значительных затрат на обучение. Естественно, что платой за это является повышение производительности труда программистов при проектировании и реализации программного обеспечения. Другое преимущество *ООП* перед императивным подходом – более высокий процент повторного использования уже разработанного программного кода. При этом, в отличие от предыдущих подходов к программированию, *объектно-ориентированный подход* требует глубокого понимания основных принципов, или, иначе, концепций, на которых он базируется. К числу основополагающих понятий *ООП* обычно относят *абстракцию данных*, *наследование*, *инкапсуляцию* и *полиморфизм*.
Поясним качественно фундаментальные принципы *ООП*. *Наследование* конкретных *атрибутов объектов* и функций оперирования *объектами* основано на иерархии. ***Инкапсуляция*** означает "сокрытие" *свойств* и *методов* внутри *объекта*. ***Полиморфизм***, как и в функциональном программировании, понимается как наличие функций с возможностью обработки данных переменного типа.
В *объектно-ориентированном* программировании каждый ***объект*** представляет собой принципиально динамическую сущность, т.е. изменяется в зависимости от времени (а также от воздействия внешних по отношению к нему факторов). Иначе говоря, *объект* обладает тем или иным образом поведения. В отношении *абстракции* как *объекта*, поведение заключается в приложении функции к аргументу.
Как мы уже отмечали, концепция *абстракции* в *объектно-ориентированном* программировании адекватно моделируется посредством ламбда-исчисления. Точнее говоря, операция *абстракции* в полной мере является моделью одноименного понятия *ООП*.

Другой фундаментальной составляющей концепции объектно-ориентированного программирования является интуитивно ясное понятие на-

следования.
В неформальной постановке под наследованием понимается свойство того или иного объекта, который является производным от некоего базового, сохранять поведение (а именно, атрибуты и операции над ними), характерное для родительского объекта.
С точки зрения языков программирования понятие наследования означает применимость всех или лишь некоторых свойств и/или методов базового (или родительского) класса для всех классов, производных от него. Кроме того, сохранение свойств и/или методов базового класса должно обеспечиваться и для всех конкретизаций (т.е. конкретных объектов) любого производного класса.
В математике концепцию наследования принято моделировать, например, отношением частичного порядка (которое представляет собой вид иерархии).
Концепция наследования адекватно формализуется математически посредством одной из следующих нотаций:

1. фреймовой нотации Руссопулоса (названной так по имени своего создателя, N.D. Roussopulos);
2. диаграмм Хассе (получивших название по имени ученого, который впервые предложил этот способ наглядного представления наследования, H. Hasse).

Рассмотрим пример программы на языке программирования C#, иллюстрирующий концепцию наследования:

```
class A {               // базовый класс
int a;
public A() {...}
public void F() {...}
}

// подкласс (наследует свойства
//класса A, расширяет класс A)
class B:A {
int b;
public B() {...}
public void G() {...}
}
```

Пример представляет собой описание базового класса A и производного от него класса B.
Класс A содержит целочисленный атрибут (т.е. переменную) a, а также два метода (т.е. функции), A() и F(). Класс B содержит целочисленный атрибут (т.е. переменную) b, а также два метода (т.е. функции) B() и G().
Двоеточие B:A в описании класса B означает наследование.
Отметим, что выше был рассмотрен простейший случай наследования, а именно, единичное наследование. Язык программирования C# позволяет реализовать механизмы, поддерживающие и более сложный случай наследования – множественное наследование.

Рассмотрим более подробно особенности наследования, которые реализует данный пример программы на языке C#.
Производный класс B наследует от базового класса A свойство a и метод F().
При этом к классу B добавляются собственные свойство b и метод G().
Заметим, что в отношении операции наследования справедливы следующие ограничения:

1. конструкторы (т.е. функции создания и инициализации классов) не наследуются;
2. в языке C# существует возможность замещения наследуемых методов (этот языковой аспект будет рассмотрен более подробно в ходе дальнейших лекций).

Как уже отмечалось, наиболее простым случаем наследования является так называемое единичное наследование. При таком наследовании производный класс (или, иначе, подкласс) может наследовать свойства только одного базового класса. Однако при этом производный класс может реализовывать множественные интерфейсы (т.е. использовать описания объектов и методов других классов, напрямую минуя механизм наследования).
Один класс языка программирования C# может наследовать лишь свойства другого класса (но не структуры – типа данных, аналогичного кортежу языка программирования SML).
Структура не может наследовать свойства другого типа данных, однако может при этом реализовывать как один, так и несколько интерфейсов.
Подкласс с неявным базовым классом наследует свойства наиболее абстрактного класса, известного под названием "объект" (object).
Еще одним фундаментальным компонентом концепции объектно-ориентированного программирования является понятие *инкапсуляции*.
Неформально говоря, под инкапсуляцией понимается возможность доступа к объекту и манипулирования им исключительно посредством предоставляемых именно этим объектом свойств и методов.
Таким образом, свойствами объекта (безразлично, являются ли они явно описанными или производными от других объектов) возможно оперировать исключительно посредством методов, которые содержатся в описании данного объекта (или родительских по отношению к нему объектов, при условии, что эти методы унаследованы).
Инкапсуляция является весьма важным свойством, поскольку обеспечивает определенную (а точнее, определяемую программистом) степень доступности объекта.
Хотя инкапсуляция как таковая является фундаментальным свойством ООП, степень инкапсуляции при наследовании может варьироваться в зависимости от типа области видимости объекта, который определяется модификатором видимости. Так, используемый в предыдущем примере модификатор видимости public обеспечивает доступность свойств и методов объекта из произвольного места программы.

К основным свойствам инкапсуляции относятся следующие возможности:

1. совместное хранение данных и функций (т.е. свойств и методов) внутри объекта;
2. сокрытие внутренней информации от пользователя (что обеспечивает большую безопасность приложения);
3. изоляция пользователя от особенностей реализации (что обеспечивает независимость от машины и потенциально дружественный интерфейс приложений).

Как нам уже известно, важная позитивная особенность языка программирования SML заключается в том, что в нем поддерживается так называемая полиморфная типизация.
В объектно-ориентированном программировании под полиморфизмом понимается возможность оперировать объектами, не обладая точным знанием их типов.
Рассмотрим пример простейшей полиморфной функции:

```
void Poly(object o) {
Console.WriteLine(o.ToString());
}
```

Данная функция реализует отображение на экране объекта (метод Console.WriteLine) с предварительным преобразованием его к строковому типу (метод ToString()) .
Все приведенные ниже варианты вызова функции:

```
Poly(25);
Poly("John Smith");
Poly(3.141592536m);
Poly(new Point(12,45));
```

успешно пройдут компиляцию и завершатся выдачей корректного результата.

8

В конце 1960-х агентство ARPA, имеющее непосредственное отношение к Министерство обороны США, приступило к реализации плана по созданию сети из компьютерных систем примерно в десятке университетов и исследовательских институтов, которые оно финансировало. Планировалось соединить компьютеры линиями связи с поразительной по тем временам пропускной способностью в 56 Кбит/с (1 Кбит/с равен 1024 битам в секунду). Заметим, что тогда лишь немногие пользователи вообще имели выход в сеть по телефонным линиям со скоростью 110 бит/с. Академические исследования стояли на пороге грандиозного прорыва. Агентство ARPA создало сеть, получившую название ARPAnet и ставшую прародителем современного Интернета.

Однако события стали развиваться не совсем так, как было запланировано. Хотя сеть ARPA действительно позволила исследователям объединить

компьютеры, оказалось, что ее основным достоинством является возможность быстрого и легкого общения с помощью электронной почты. И по сей день электронная почта, моментальные сообщения и передача файлов Интернете позволяют более чем миллиарду людей во всем мире общаться друг с другом Протокол (т. е. набор правил) связи в ARPAnet получил название TCP (Transmission CoП^ Protocol, протокол управления передачей). Он гарантировал, что сообщения, разбитые части (называемые пакетами), будут посланы по маршруту от отправителя к получателю прибудут в целости и сохранности и будут собраны в правильном порядке. Параллельно с ранним этапом развития Интернета организации по всему миру стали создавать собственные сети, как для связи внутри организаций, так и для связи их друг с другом. Появилось огромное количество разнообразной аппаратуры и программ, обеспечивавших сетевую связь. Вскоре возникла проблема общения между различными сетями. Проблему шило агентство ARPA, разработавшее протокол IP (Internet Protocol, межсетевой прото-ол), который позволил создать "сеть сетей", архитектуру современного Интернета. Объеденный набор протоколов получил название TCP/IP.

Коммерческие компании быстро осознали, что с помощью Интернета они смогут усовершенствовать свои операции и предложить клиентам новые, более качественные услуги. Фирмы стали вкладывать большие средства в развитие и усиление своего присутствия в Интернете. Возникла конкуренция между владельцами линий связи, а также между поставщиками аппаратного и программного обеспечения за удовлетворение растущего спроса на инфраструктуру. В результате резко возросла пропускная способность (т. е. производительность) линий передачи информации, а цены на аппаратное обеспечение упали.

Всемирная паутина представляет собой совокупность аппаратного и программного обеспечения, связанную с Интернетом и позволяющую пользователям компьютеров находить и просматривать мультимедийные документы (документы, сочетающие в себе текст, графику, анимацию, звуки и видеоизображения) практически по любой теме. Хотя Интернет появился более трех десятилетий тому назад, Всемирная паутина (или World Wide Web, сокращенно WWW) возникла сравнительно недавно. В 1989 году Тим Бернерс-Ли (Tim Berners-Lee) из ICERN (Conseil Europeen pour la Recherche Nucleaire, Европейская организация по ядерным исследованиям) приступил к разработке технологии, которая позволила бы обмениваться информацией с помощью текстовых документов, связанных "гиперссылками". Бернерс-Ли назвал свое изобретение HTML (Hypertext Mark-up Language, язык разметки гипертекста). Он также разработал протоколы связи, в частности, HTTP (Hypertext Transfer Protocol, протокол передачи гипертекстовых файлов), образующие основу новой гипертекстовой информационной системы, которую он и назвал "World Wide Web" (Всемирная паутина).

В октябре 1994 года Бернерс-Ли основал организацию World Wide Web Consortium (W3C, www.w3.org), занявшуюся разработкой технологий для

Всемирной паутины. Одной из своих задач эта организация считает обеспечение доступной Всемирной паутины каждому человеку, независимо от его языка, культуры и физических возможностей.

9

Visual Studio .Net - открытая среда разработки

Среда разработки *Visual Studio .Net* - это уже проверенный временем программный продукт, являющийся седьмой версией Студии. Но новинки этой версии, связанные с идеей .Net, позволяют считать ее принципиально новой разработкой, определяющей новый этап в создании программных продуктов. Выделю две важнейшие, на мой взгляд, идеи:

- *открытость* **для языков программирования;**
- принципиально новый подход к построению **каркаса среды - Framework .Net**.

Открытость

Среда разработки теперь является открытой языковой средой. Это означает, что наряду с языками программирования, включенными в среду фирмой Microsoft - **Visual C++ .Net** (с управляемыми расширениями), **Visual C# .Net**, **J# .Net**, **Visual Basic .Net**, - в среду могут добавляться любые языки программирования, компиляторы которых создаются другими фирмами-производителями. Таких расширений среды **Visual Studio** сделано уже достаточно много, практически они существуют для всех известных языков - **Fortran** и **Cobol**, **RPG** и **Component Pascal**, **Oberon** и **SmallTalk**. Я у себя на компьютере включил в среду компилятор одного из лучших объектных языков - языка**Eiffel**.

Открытость среды не означает полной свободы. Все разработчики компиляторов при включении нового языка в среду разработки должны следовать определенным ограничениям. Главное ограничение, которое можно считать и главным достоинством, состоит в том, что все языки, включаемые в среду разработки *Visual Studio .Net*, должны использовать единый *каркас - Framework .Net*. Благодаря этому достигаются многие желательные свойства: легкость использования компонентов, разработанных на различных языках; возможность разработки нескольких частей одного приложения на разных языках; возможность бесшовной отладки такого приложения; возможность написать класс на одном языке, а его потомков - на других языках. Единый каркас приводит к сближению языков программирования, позволяя вместе с тем сохранять их индивидуальность и имеющиеся у них достоинства. Преодоление языкового барьера - одна из важнейших задач современного мира. Благодаря единому каркасу, *Visual Studio .Net* в определенной мере решает эту задачу в мире программистов.

Framework .Net - единый каркас среды разработки

В *каркасе Framework .Net* можно выделить два основных компонента:

- статический - ***FCL (Framework Class Library)*** - *библиотеку классов* каркаса;
- динамический - ***CLR (Common Language Runtime)*** - общеязыковую исполнительную среду.

Библиотека классов FCL - статический компонент каркаса

Понятие каркаса приложений - Framework Applications - появилось достаточно давно; по крайней мере оно широко использовалось еще в четвертой версии Visual Studio. Несмотря на то, что каркас был представлен только статическим компонентом, уже тогда была очевидна его роль в построении приложений. Уже в то время важнейшее значение в библиотеке классов MFC имели классы, задающие архитектуру строящихся приложений. Когда разработчик выбирал один из возможных типов приложения, например, архитектуру Document-View, то в его приложение автоматически встраивались класс Document, задающий структуру документа, и класс View, задающий его визуальное представление. Класс Form и классы, задающие элементы управления, обеспечивали единый интерфейс приложений. Выбирая тип приложения, разработчик изначально получал нужную ему функциональность, поддерживаемую классами каркаса. Библиотека классов поддерживала и более традиционные для программистов классы, задающие расширенную систему типов данных, в частности, динамические

Единство каркаса

Каркас стал единым для всех языков среды. Поэтому, на каком бы языке программирования не велась разработка, она использует классы одной и той же библиотеки. Многие классы библиотеки, составляющие общее ядро, используются всеми языками. Отсюда единство интерфейса приложения, на каком бы языке оно не разрабатывалось, единство работы с коллекциями и другими контейнерами данных, единство связывания с различными хранилищами данных и прочая универсальность.

Встроенные примитивные типы

Важной частью *библиотеки FCL* стали классы, задающие ***примитивные типы*** - те типы, которые считаются *встроенными* в язык программирования. Типы каркаса покрывают все множество *встроенных типов*, встречающихся в языках программирования. Типы языка программирования проецируются на соответствующие типы каркаса. Тип, называемый в языке Visual Basic - Integer, а в языке C# - int, проецируется на один и тот же тип каркаса System.Int32. В каждом языке программирования, наряду с "родными" для языка названиями типов, разрешается пользоваться именами типов, принятыми в каркасе. Поэтому, по сути, все языки среды разработки могут пользоваться единой системой *встроенных типов*, что, конечно, способствует облегчению взаимодействия компонентов, написанных на разных языках.

Структурные типы

Частью библиотеки стали не только простые *встроенные типы*, но и *структурные типы*, задающие организацию данных - строки, массивы, перечисления, структуры (записи). Это также способствует унификации и реальному сближению языков программирования.

Архитектура приложений

Существенно расширился набор возможных архитектурных типов построения приложений. Помимо традиционных Windows- и консольных приложений, появилась возможность построения Web-приложений. Большое внимание уделяется возможности создания повторно используемых компонентов - разрешается строить библиотеки классов, библиотеки элементов управления и библиотеки Web-элементов управления. Популярным архитектурным типом являются Web-службы, ставшие сегодня благодаря открытому стандарту одним из основных видов повторно используемых компонентов. Для языков C#, J#, Visual Basic, поддерживаемых Microsoft, предлагается одинаковый набор из 12 архитектурных типов приложений. Компиляторы языков, поставляемых другими фирмами, создают проекты, которые удовлетворяют общим требованиям среды, сохраняя свою индивидуальность. Так, например, компилятор Eiffel допускает создание проектов, использующих как *библиотеку FCL*, так и собственную библиотеку классов.

Модульность

Число классов *библиотеки FCL* велико (несколько тысяч). Поэтому понадобился способ их структуризации. Логически классы с близкой функциональностью объединяются в группы, называемые пространством имен (Namespace). Для динамического компонента *CLR* физической единицей, объединяющей классы и другие ресурсы, является сборка (assembly).

Основным пространством имен *библиотеки FCL* является пространство System, содержащее как классы, так и другие вложенные пространства имен. Так, уже упоминавшийся *примитивный тип* Int32 непосредственно вложен в пространство имен System и его полное имя, включающее имя пространства - `System.Int32`.

В пространство `System` вложен целый ряд других пространств имен. Например, в пространстве `System.Collections` находятся классы и интерфейсы, поддерживающие работу с коллекциями объектов - списками, очередями, словарями. В пространство `System.Collections`, в свою очередь, вложено пространство имен `Specialized`, содержащие классы со специализацией, например, коллекции, элементами которых являются только строки. Пространство `System.Windows.Forms` содержит классы, используемые при создании Windows-приложений. Класс `Form` из этого пространства задает форму - окно, заполняемое элементами управления, графикой, обеспечивающее интерактивное взаимодействие с пользователем.

Общеязыковая исполнительная среда CLR - динамический компонент каркаса

Наиболее революционным изобретением *Framework .Net* явилось создание **исполнительной среды** *CLR*. С ее появлением процесс написания и выполнения приложений становится принципиально другим. Но обо всем по порядку.

Двухэтапная компиляция. Управляемый модуль и управляемый код

Компиляторы языков программирования, включенные в *Visual Studio .Net*, создают модули на промежуточном языке ***MSIL (Microsoft Intermediate Language)***, называемом далее просто - IL. Фактически компиляторы создают так называемый ***управляемый модуль*** - переносимый исполняемый файл (Portable Executable или PE-файл). Этот файл содержит код на IL и ***метаданные*** - всю необходимую информацию как для *CLR*, так и конечных пользователей, работающих с приложением. О *метаданных* - важной новинке*Framework .Net* - мы еще будем говорить неоднократно. В зависимости от выбранного типа проекта, PE-файл может иметь расширения exe, dll, mod или mdl.

Заметьте, PE-файл, имеющий расширение exe, хотя и является exe-файлом, но это не совсем обычный исполняемый Windows файл. При его запуске он распознается как специальный PE-файл и передается *CLR* для обработки. Исполнительная среда начинает работать с кодом, в котором специфика исходного языка программирования исчезла. Код на IL начинает выполняться под управлением *CLR* (по этой причине **код** называется ***управляемым***). Исполнительную среду можно рассматривать как своеобразную виртуальную IL-машину. Эта машина транслирует "на лету" требуемые для исполнения участки кода в команды реального процессора, который в действительности и выполняет код.

Виртуальная машина

Отделение каркаса от студии явилось естественным шагом. *Каркас Framework .Net* перестал быть частью студии, а стал надстройкой над операционной системой. Теперь компиляция и создание PE-модулей на IL отделены от выполнения, и эти процессы

могут быть реализованы на разных платформах. В состав *CLR* входят трансляторы JIT (Just In Time Compiler), которые и выполняют трансляцию IL в командный код той машины, где установлена и функционирует исполнительная среда *CLR*. Конечно, в первую очередь Microsoft реализовала *CLR* и *FCL* для различных версий Windows, включая Windows 98/Me/NT 4/2000, 32 и 64-разрядные версии Windows XP и семейство .Net Server. Для операционных систем Windows CE и Palm разработана облегченная версия *Framework .Net*.

В 2001 году ECMA (Европейская ассоциация производителей компьютеров) приняла язык программирования C#, *CLR* и *FCL* в качестве стандарта, так что *Framework .Net* уже функционирует на многих платформах, отличных от Windows. Он становится свободно распространяемой виртуальной машиной. Это существенно расширяет сферу его применения. Производители различных компиляторов и сред разработки программных продуктов предпочитают теперь также транслировать свой код в IL, создавая модули в соответствии со спецификациями *CLR*. Это обеспечивает возможность выполнения их кода на разных платформах.

Microsoft использовала получивший широкое признание опыт виртуальной машины Java, улучшив процесс за счет того, что, в отличие от Java, промежуточный код не интерпретируется исполнительной средой, а компилируется с учетом всех особенностей текущей платформы. Благодаря этому создаются высокопроизводительные приложения.

Следует отметить, что *CLR*, работая с IL-кодом, выполняет достаточно эффективную оптимизацию и, что не менее важно, защиту кода. Зачастую нецелесообразно выполнять оптимизацию на уровне создания IL-кода - она иногда может не улучшить, а ухудшить ситуацию, не давая *CLR* провести оптимизацию на нижнем уровне, где можно учесть даже особенности процессора.

10

Синтаксические описания

Синтаксис языка программирования — это правила написания программ.

Рассмотрим пример — возьмем из справочной системы описание команды. Мы пока не будем вдаваться в смысл ключевых слов (перед нами — описание *оператора* цикла), приведенных здесь, остановимся на основных частях описания.

```
For counter = start To end [Step step]
[statements]
[Exit For]
[statements]
Next [counter]
```

В синтаксических описаниях VBA жирным шрифтом выделены ключевые слова языка, курсивом даны названия *переменных* и других элементов языка. В квадратных скобках располагаются необязательные элементы – то есть такие элементы, без описания которых можно обойтись. Например, в приведенном описании блоки `[statements]` будут содержать в себе "полезную нагрузку" цикла – *операторы*, которые будут выполняться многократно, однако цикл будет работать и без них.

Элементы, которые допускают альтернативный выбор, разделяются знаком "`|`". Например, такая запись: "`one | two`" обозначает "one или two".

Ниже мы будем останавливаться на существенно важных особенностях тех или иных конструкций. Если вам понадобятся подробности — вы сможете найти их в справке к VBA.

Теперь рассмотрим понятие *переменной* — важнейшее понятие в любом языке программирования.

Переменные

Переменная — это именованная область памяти, где могут храниться различные данные, которые можно изменять во время выполнения программы. *Переменные*— это одна из основ любой программы. В них можно сохранять введенную пользователем информацию, их можно использовать для накопления данных, обработанных в программе и так далее. Если бы не было *переменных* — программирования не существовало бы.

У *переменной* есть несколько важнейших характеристик. Первая — имя *переменной*. Используя имя, мы можем обращаться к *переменной* в программе. Вторая характеристика — это *тип данных*, которые могут храниться в *переменной*. Тип определяет характер данных, которые мы можем хранить в *переменной*. Например, это могут быть числовые данные (возраст пользователя) и строковые данные (имя пользователя).

Давая *переменным* имена, следует придерживаться следующих правил.

- Имя *переменной* должно состоять из букв и цифр, причем оно должно начинаться с буквы.
- Имя не может быть длиннее 255 символов
- Имя не должно содержать специальных знаков (`#`, `$`, `%` и т.д.) и пробелов.
- В качестве имен нельзя использовать зарезервированные слова VBA (например – `if`, `dim`, `for` и т.д.).

При именовании *переменных* старайтесь пользоваться латинскими буквами. Иначе возможны проблемы при работе ваших программ на различных версиях Office.

Например, такие имена *переменных* выглядят вполне корректно: `str_Name`, `num_Age`, `str_Name1`, `num_Item2` и т.д. Обратите внимание на префиксы, которыми мы снабдили имена *переменных*. Это признак особого стиля именования *переменных*. Он очень напоминает стиль именования элементов управления, который мы рассматривали выше.

Типы данных в VBA

Тип данных определяет важнейшие свойства *переменной*. А именно, следующее:

- что может храниться в *переменной* (текст, число, некоторые другие виды данных);
- размер памяти, необходимый для хранения *переменной* (измеряется в байтах);
- операции, которые можно производить с *переменной* (например, невозможно извлечь квадратный корень из слова "Привет" так как эта операция не определена для строковых данных);

Вы уже можете предположить как минимум два *типа данных*, которые могут хранить *переменные* — строковые и числовые. На самом деле список типов данных VBA гораздо обширнее.

В следующей таблице вы найдете информацию об основных типах данных VBA.

Таблица Типы данных в VBA

Тип данных	Размер, байт	Описание	Диапазон значений
`Variant`	16 (числа)	Может хранить данные	

	22+длина строки (строки)	любых типов	
Integer	2	Целое число	от -32768 до 32767
Long	4	Длинное целое	от -2147483648 до 2147483647
Single	4	Число с плавающей запятой обычной точности до значения и для положительных	Для отрицательных: от -3.402823E38 до -1.401298E-45 Для положительных: от 1.401298E-45 до 3.402823E38
Double	8	Число с плавающей запятой двойной точности, для отрицательных значений и для положительных	Для отрицательных от -1.79769313486231E308 до -4.94065645841247E-324 Для положительных: от 4.94065645841247E-324 до 1.79769313486232E308
Byte	1	Байт	от 0 до 255
Currency	8	Денежный формат	от -922,337,203,685,477.5808 до 922,337,203,685,477.5807
Decimal	14	Масштабируемое целое	29-значное число с 28 знаками справа от запятой
Boolean	2	Логический	True или False
Date	8	Дата и время	от 1 января 100 г. до 31 декабря 9999 г.
Object	4	Ссылка на объект	Различные виды объектов
String	Зависит от длины строки	Строка	

В VBA существует немало типов данных. Может показаться, что нет ничего удобнее *типа данных* `Variant`. Однако надо учесть, что за универсальность типа `Variant` приходится платить производительностью и системными ресурсами. Чем больше места требуется для хранения *переменной* — тем ниже скорость работы с ней.

Если вы хотите, чтобы ваши программы работали как можно быстрее — выбирайте *типы данных*, которые используют для хранения вашей информации минимум системных ресурсов. Если вы сомневаетесь, хватит ли размерности выбранного *типа данных* для решения ваших задач — возьмите более емкий тип.

Как правило, для работы с дробными числами и различных вычислений используют *тип данных* `Double`. Для *переменных*, которые используются в циклах, счетчиках — `Integer` или `Long`. В работе со строками используют `String`. Объектные *переменные* имеют самые разные типы, которые определяются объектами, для обращения к которым они создаются. В небольших проектах вполне оправдано использование *переменных* типа `Variant` – это не слишком замедлит работу. Однако, если размер проекта растет или вы нуждаетесь в ускорении ресурсоемких вычислений – подходите к выбору типов данных ответственно.

Если при объявлении *переменной* не указывать ее тип — он автоматически устанавливается в `Variant`. Объявляя *переменную* без указания типа, вы пользуетесь преимуществами работы с объявленными *переменными*, и, в то же время, можете гибко использовать *переменную* для хранения различных типов данных (например – чисел с плавающей запятой и целых чисел). Такой подход — объявление *переменных* без указания типа — допустим при создании небольших проектов, а так же — на начальном этапе работы над достаточно масштабными программами, когда вы не знаете точно, какой *тип данных* понадобится для той или иной *переменной*.

Выше мы уже говорили о венгерской нотации как о способе именования объектов. Те же правила действуют и для *переменных*. Префикс имени обычно говорит о типе*переменной*. В следующей таблице приведены некоторые префиксы.

Таблица Префиксы и типы переменных

Префикс имени переменной	Пояснения
`s, str`	Строковая переменная
`n, num`	Числовая переменная (существует множество числовых типов данных, название каждого из которых может быть использовано в качестве префикса, например, int для Integer, dbl для Double и т.д.)
`b, bool`	Логическая переменная
`O, obj`	Объектная переменная

Для работы с *переменными* можно использовать два подхода. При первом мы выбираем имя *переменной* и используем ее в программе без каких-либо подготовительных действий. Такая *переменная* приобретет тип `Variant`. Второй подход предусматривает объявление *переменных* перед использованием.

Преимущества объявления переменных

Как вы уже знаете, в VBA *переменные* можно использовать без предварительного объявления и указания типа. Но такой подход допустим лишь при разработке небольших программ, состоящих буквально из пары десятков строк. Когда программа увеличивается, контролировать работу с *переменными*, которые спонтанно появляются в коде, становится сложно.

Необъявленные *переменные* могут вызывать трудно диагностируемые ошибки. Например, рассмотрим такой код:

```
txt_CompanyName = "Типография"
MsgBox (txt_CompanyName)
```

Как вы думаете, что получится при его исполнении? Очевидно, должно вывестись окно сообщения с текстом "Типография". Но это далеко не факт. По внешнему виду *переменных* можно предположить, что их имена написаны латинскими буквами. Предположим, что в строке, где мы присваиваем *переменной* значение "Типография", это так и есть. А во второй строке мы ошиблись — и вместо латинской написали русскую букву "С" (они и на клавиатуре занимают одну клавишу, а внешне различить их просто невозможно). Как результат – совершенно непонятная ошибка – все выглядит правильно, а работать не хочет.

Если бы редактор кода не давал нам пользоваться необъявленными *переменными* — подобного рода ошибки были бы пресечены на корню. Чтобы заставить редактор автоматически требовать объявление *переменных*, можно поступить одним из двух способов. Первый — вставить в раздел объявлений модуля (то есть — вне кода процедур и обработчиков событий) команду `Option Explicit`. Второй — включить запрещение работы с необъявленными *переменными* в настройках редактора. Для этого выполните команду главного меню `Tools • Options` (Инструменты • Опции) и в появившемся диалоговом окне на вкладке `Editor` (Редактор) включите параметр `Require Variable Declaration` (Требовать объявление *переменных*). Подобные правила являются элементом структурного программирования, которое в своё время явилось правилом хорошего тона при написании программ.

Объявление переменных: Dim и Static

Лучше всего объявлять *переменные* перед использованием. Так вы избежите ненужных ошибок, и, при работе с объектными *переменными*, сможете получать подсказки при их использовании.

Рассмотрим пример. Объявление *переменной* с именем `num_MyAge` типа `Byte` выглядит так:

```
Dim num_MyAge As Byte
num_MyAge = 23
```

Этот код можно перевести так: "Объявить *переменную* типа `Byte` с именем `num_MyAge` и сделать ее доступной в модуле, в котором она объявлена". Второй строкой мы присваиваем *переменной* число 23.

Доступность *переменной*, или, как говорят, область видимости, задается ключевым словом `Dim`. Оно означает, что мы сможем пользоваться объявленной *переменной* только внутри того модуля, в котором она объявлена.

Обратите внимание на то, что если вы присваиваете значение строковой *переменной* — передаваемое ей значение должно быть заключено в кавычки. При присваивании значений даты соответствующим *переменным*, эти значения должны быть заключены в значки `#`. Например, так:

```
Dim str_MyName as String
Dim date_MyBirthDate as Date
str_MyName = "Alexander"
date_MyBirthDate = #January 1, 2012#
```

Помимо ключевого слова `Dim` могут использоваться и другие слова. В частности, будет полезна команда `Static` — *переменная*, объявленная с использованием этой команды, будет хранить свое значение между вызовами процедуры, область ее видимости аналогична *переменной*, объявленной с `Dim`. Такую *переменную* можно использовать для накопления каких-либо значений.

Рассмотрим пример. Создайте новый документ Microsoft Excel, добавьте в него две кнопки. В обработчике события `Click` одной из них с именем `cmd_Dim`, на которой будет написано `Dim`, напишите такой код

```
Dim num_Counter As Integer
num_Counter = num_Counter + 1
MsgBox (num_Counter)
```

Здесь мы сначала объявляем *переменную* типа `Integer`, потом присваиваем этой *переменной* ее же значение, увеличенное на 1, а дальше — выводим *переменную* в окне сообщения.

Сколько бы вы не нажимали на эту кнопку — всякий раз в окне сообщения будет выводиться единица. Ведь каждый раз, когда запускается обработчик, все происходит как бы в первый раз — информация об изменениях *переменной* не сохраняется при выходе из обработчика.

Вторую кнопку назовите cmd_Static, подпишите ее как Static, и в обработчик нажатия добавьте следующее:

```
Static num_Counter As Integer
num_Counter = num_Counter + 1
MsgBox (num_Counter)
```

Единственное отличие этого текста от предыдущего — ключевое слово Static вместо Dim. Но благодаря такому объявлению, *переменная* хранит свое значение между запусками обработчика. Попробуйте несколько раз нажимать на кнопку Static — каждый раз вы будете видеть в окне сообщения число, увеличенное на 1.

Арифметические операторы и работа с числовыми переменными

Операторы — это команды, которые используются в VBA для работы с данными. Если вы программировали когда-нибудь, например, на Basic'e или на каком-нибудь другом языке, вам уже знакомы основные *операторы*. Ну а если вы раньше не сталкивались с *операторами* в программировании, то уж со школьным курсом математики вы сталкивались точно. Большинство *операторов* VBA действуют точно так же, как знакомые всем арифметические *операторы*. В таблице вы можете видеть описание арифметических *операторов*, применимых в VBA

Таблица Арифметические операторы VBA

Оператор	Описание
+	Сложение. Например, в результате вычисления выражения 4+3 получится 7.
–	Вычитание.
*	Умножение
/	Деление
^	Возведение в степень, 5^2 равняется 25
\	Целочисленное деление. От результата деления отбрасывается дробная часть. Например, 10 \ 3 равняется 3.
mod	Деление по модулю. Возвращает остаток от деления. Например, 10 mod 3 равняется 1

В VBA применяется тот же порядок обработки выражений, который принят в математике. Например, результат вычисления 2*3+7 равняется 13-ти. Так же, в математических выражениях могут использоваться круглые скобки. Однако, круглые скобки могут встретиться вам не только в математических выражениях. Скобки, как вы могли заметить из примеров, которые мы рассматривали выше, часто употребляются при вызове функций VBA, при работе с объектами и т.д.

Если снова вернуться к математике, нельзя не заметить, что редкая математическая запись обходится без знака =. Этот знак используется в VBA в нескольких ролях.

Во-первых, это оператор присваивания. Вы уже сталкивались с ним, когда присваивали какие-то значения свойствам элементов управления или *переменным*. Например, такая запись: num_a = 10 + 2 означает: "Присвоить *переменной* num_a результат сложения чисел 10 и 2" или, проще, "a равно 10+2". Вторая роль *оператора* = заключается в том, что он используется в командах сравнения выражений — о них мы поговорим в следующей главе.

Конкатенация

Несколько ролей и у *оператора* +. Во-первых — это арифметический оператор сложения. А во-вторых — оператор конкатенации строк. Конкатенация — это нечто вроде "склеивания" строк. В качестве *оператора* конкатенации можно использовать и оператор &. Считается, что & использовать предпочтительнее так как он в любом случае обрабатывает операнды как строковые данные.

Давайте рассмотрим пример, который охватывает арифметические операции и конкатенацию строк.

Напишем программу, которая запрашивает имя пользователя и два числа, после чего выводит такой текст в окне сообщения (если имя введено как "Александр", первое число 2, второе – 21): "Здравствуйте, Александр. Вы ввели числа 2 и 21, их сумма равняется 23."

Добавим кнопку в документ Microsoft Word, назовем ее `cmd_Experiments`, надпишем ее как Работа с *операторами*.

Теперь решим, какие *переменные* нам нужны.

Для имени пользователя это *переменная* типа `String`. Дадим ей имя `str_UserName`. Для чисел нам понадобится пара *переменных* одного из числовых типов.

Какие числа введет пользователь? Этого мы не знаем. Конечно, можно ограничить ввод проверками, но это дела будущих примеров. Предполагается, что пользователь может ввести практически любое число — целое или дробное, положительное или отрицательное. Поэтому воспользуемся типом `Double`. Назовем пару числовых *переменных* `num_First` и `num_Second`.

Создадим отдельную *переменную* того же типа `Double` для хранения суммы введенных значений – назовем ее `num_Summ`.

В условии нашей задачи присутствует требование вывести все введенные данные в строку, скомбинировав с определенными словами. Нам понадобится *переменная* типа`String` для хранения этой строки. Назовем ее `str_Result`.

Также добавим в обработчик нажатия кнопки команду `Option Explicit` — тогда система запретит использование необъявленных *переменных*.

Вы уже знакомы с функцией `InputBox`, которая выводит окно для ввода данных пользователем. Как вы могли убедиться, `InputBox` отлично справляется со вводом строковых значений. А как насчет чисел? Можем ли мы написать что-то вроде:

```
num_First = InputBox("Введите первое число")
```

Вполне можем.

Пожалуй, сейчас для вас самое сложное — это построить строку вывода, однако пользуйтесь следующими правилами и вы всегда будете правильно и быстро строить подобные строки.

Во-первых — помните, что все, что вы хотите вывести в виде неизменного текста, должно быть включено в кавычки, а имена *переменных*, наоборот, пишутся без кавычек. Во-вторых — используйте оператор & при "склеивании" отдельных частей строки

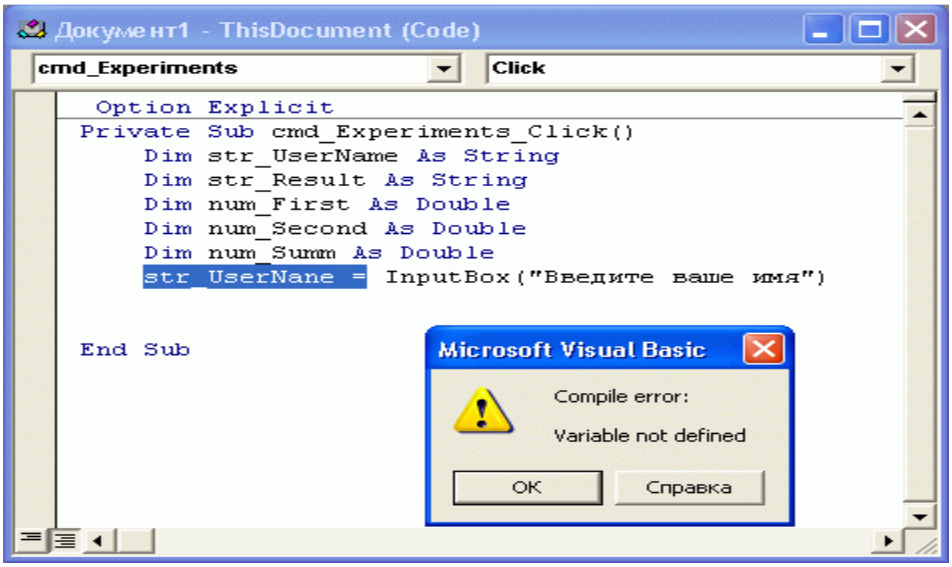
или оператор + и функцию Str (о ней вы прочтете ниже), которая конвертирует числовые*переменные* в строки. Так вы будете гарантированы от неожиданностей и ошибок.

Например, пусть в *переменной* str_UserName хранится имя пользователя "Александр", а *переменная* str_Result должна содержать результат вывода. Напишем код, помещающий в str_Result строку "Привет, Александр":

```
str_Result = "Привет, " & str_UserName
```

Видите? Ничего сложного, правда?

Обратите внимание на то, как реагирует система на использование необъявленных *переменных* при добавленной в модуль команде Option Explicit (см. рисунок ниже).



Здесь вместо str_UserName мы ошибочно использовали str_UserNane. Но благодаря Option Explicit появление необъявленных *переменных* воспринимается как ошибка. При попытке запуска программы мы видим сообщение об ошибке: "Variable not defined" - "Переменная не определена".

Однако вернемся к нашей задаче. Мы определились с особенностями работы наиболее сложных ее частей, в итоге у нас получилось следующее (листинг 5.7.):

```
Dim str_UserName As String
Dim str_Result As String
Dim num_First As Double
Dim num_Second As Double
Dim num_Summ As Double
str_UserName = InputBox("Введите ваше имя")
num_First = InputBox("Введите первое число")
num_Second = InputBox("Введите второе число")
num_Summ = num_First + num_Second
    str_Result = "Здравствуйте, " & str_UserName _
& ". Вы ввели числа " & num_First & " и " & num_Second _
& ". Их сумма равняется " & num_Summ
MsgBox (str_Result)
```

Обратите внимание на то, что строку сборки строки вывода `str_Result` мы разбили на несколько частей *оператором* переноса строки.

Преобразование типов данных

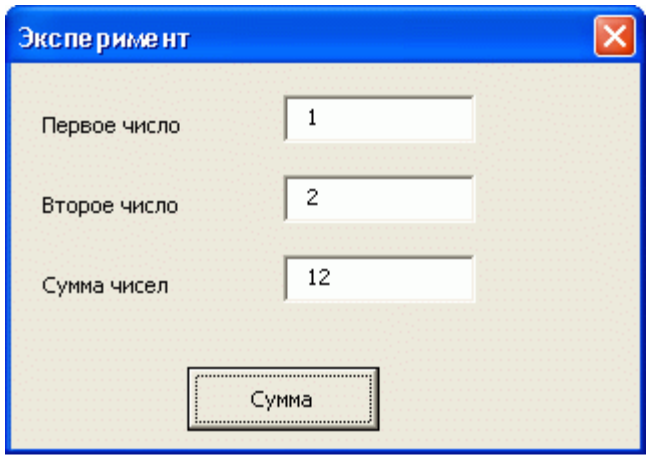
Нередко возникают ситуации, когда данные одного типа надо преобразовать в данные другого. Ярче всего это проявляется тогда, когда нужно преобразовать число, представленное в виде строки и хранящееся в строковой *переменной*.

Давайте проведем эксперимент. Создадим форму, назовем ее `frm_First`, разместим на ней три текстовых поля — `txt_First` и `txt_Second` для ввода чисел, и `txt_Summ` для вывода суммы этих чисел. Добавим на форму кнопку `cmd_First` с надписью `Сумма` и создадим для нее обработчик события `Click` (листинг 5.8.).

```
txt_Summ = txt_First + txt_Second
```

Листинг 5.8. Обработчик события Click кнопки cmd_First

На рисунке ниже вы можете видеть форму после того, как мы ввели в первое и второе поля числа 1 и 2 и нажали на кнопку.



Как видите, вместо сложения мы получили конкатенацию. Любые значения, вводимые в текстовые поля, по умолчанию рассматриваются как строковые. Чтобы все-таки получить сумму введенных чисел в поле "Сумма чисел", нам нужно превратить строки, пусть и содержащие числовые символы, в настоящие числа, преобразовать их из *типа данных* `String` в один из числовых типов.

Для таких "превращений" существуют специальные функции *преобразования типов*.

Val — тип String в тип Double

Функция `Val` применяется для конверсии строковых *переменных* в числовые, а именно – *переменных* типа `String` в тип `Double`.

Чтобы наш пример заработал, код обработчика нажатия кнопки можно переписать так (листинг 5.9.):

```
txt_Summ = Val(txt_First) + Val(txt_Second)
```

Листинг 5.9. Измененный обработчик события Click кнопки cmd_First

Давайте рассмотрим еще несколько примеров использования этой функции.

```
Val (" 12345привет") возвратит число 12345.
```

Val читает предлагаемую ей строку слева направо, игнорируя пробелы. Она считывает все числовые знаки до первого символьного знака и преобразует считанное в число. В качестве дробных символов функция понимает лишь точки.

```
Val ("1 2 3") возвратит число 123
Val ("1 2 и 3") возвратит число 12.
```

Иногда нужно провести обратное преобразование — превратить число в строку.

Str — числовые типы в String

Функция Str конвертирует данные различных числовых типов в тип String.

Особенность функции заключается в том, что первый символ полученного строкового значения зарезервирован для знака числа. Если в строку конвертируется число отрицательное — первый символ полученной строки — знак -. Если конвертируется положительное число, первым символом полученной строки будет пробел, а дальше будут идти числовые символы.

Например, функция Str (12) возвратит строку " 12". Мы рассмотрим пример с использованием функции Str немного ниже, когда будем говорить о работе со строками.

Существуют и другие функции, предназначенные для конверсии типов данных. Их названия состоят из сокращенного слова "Convert" и сокращенного же названия *типа данных*, в который они конвертируют входные значения. Например, это CBool, CByte, CCur, CDate, CDbl, CDec, CInt, CLng, CSng, CStr, CVar. Скажем, функция CIntконвертирует данные в формат Integer. Учитывая особенности этого *типа данных*, корректно могут быть сконвертированы лишь значения от -32768 до 32767. Причем, дробные числа округляются при конверсии до ближайшего четного числа — 0.5 округляется до 0, 1.5 — до 2. Если вам понадобятся подробности о каждой из этих функций — обратитесь к справочной системе VBA.

Функции проверки типа данных

Если вам нужно узнать *тип данных переменной*, вы можете воспользоваться функцией TypeName.

Добавим в документ Microsoft Word кнопку, назовем ее cmd_DataType, надпишем как Проверка типа и внесем в ее обработчик Click такой код (листинг 5.10.):

```
Dim num_MyAge as Byte
num_MyAge = 24
MsgBox (TypeName(num_MyAge))
```

Листинг 5.10. Обработчик события Click кнопки cmd_DataType

На рисунке вы можете видеть результат выполнения этого кода.

Чтобы проверить, являются ли данные, хранимые в *переменной* типа Variant, числом, можно воспользоваться функцией IsNumeric.

Для точного определения *типа данных*, которые хранятся в *переменной* типа `Variant`, вы можете воспользоваться функцией `VarType`.

Встроенные математические функции

Вы хотите вычислить квадратный корень, округлить число или сделать с ним еще что-нибудь подобное? Для этого VBA имеет специализированные функции, вы можете найти их в следующей.

Таблица Встроенные математические функции

Функция	Описание
`Abs`	Абсолютное значение
`Atn`	Арктангенс
`Cos`	Косинус числа
`Exp`	Возвращает число e (2.718282), возведенное в степень аргумента функции.
`Fix`	Отбрасывает дробную часть числа и возвращает целую. В результате для положительных чисел получается число меньшее, чем входное (Fix(2.5) возвратит 2), для отрицательных - большее (Fix(-2.5) возвратит -2)
`Int`	Отбрасывает дробную часть числа и возвращает целую. Для положительных получается число меньшее введенного (Int(2.5) возвратит 2), для отрицательных - так же меньшее (Int(-2.5) возвратит -3).
`Log`	Возвращает натуральный логарифм числа
`Rnd`	Возвращает случайное число типа Single, причем, это число находится между 0 и 1. Для инициализации генератора случайных чисел используйте директиву`Randomize` - ее надо вызвать до вызова `Rnd`.
`Sgn`	Функция предназначена для определения знака числа. Если число положительное - она возвращает 1. Для нуля функция возвратит 0, для отрицательного числа -1.
`Sin`	Синус
`Sqr`	Квадратный корень
`Tan`	Тангенс

Давайте рассмотрим пример. Добавим в документ Microsoft Word кнопку, назовем ее `cmd_Calc`, надпишем ее как Вычисления и добавим следующий код (листинг 6.1.), иллюстрирующий работу рассмотренных функций.

```
Dim dblNumber As Double
'Переменная, используемая в вычислениях
Dim varResult
'Переменная типа Variant
dblNumber = Val(InputBox("Введите число"))
'Вычисляем абсолютное значение введенного числа
'Сначала присвоим результат переменной varResult
'Далее - выведем подписанный результат в окне
'сообщения, воспользуемся знаком "+" для
'конкатенации строк, в других случаях
'будем вызывать функции непосредственно
'в MsgBox'e
'Обратите внимание на то, что мы конвертируем
'числовые значения в строки с помощью функции Str
varResult = Abs(dblNumber)
MsgBox ("Абсолютное значение " + _
Str(dblNumber) + " равняется " + Str(varResult))
'Арктангенс
```

```
    MsgBox ("Арктангенс " + _
    Str(dblNumber) + " равняется " + _
    Str(Atn(dblNumber)))
    'Косинус
    MsgBox ("Косинус " + _
    Str(dblNumber) + " равняется " + _
    Str(Cos(dblNumber)))
    'e в степени введенного числа
    MsgBox ("Число e в степени " + _
    Str(dblNumber) + " равняется " + _
    Str(Exp(dblNumber)))
    'Функция Fix
    MsgBox ("Результат работы функции Fix для " + _
    Str(dblNumber) + " равняется " + _
    Str(Fix(dblNumber)))
    'Функция Int
    MsgBox ("Результат работы функции Int для " + _
    Str(dblNumber) + " равняется " + _
    Str(Int(dblNumber)))
    'Натуральный логарифм
    MsgBox ("Натуральный логарифм " + _
    Str(dblNumber) + " равняется " + _
    Str(Log(dblNumber)))
    'Получим несколько случайных чисел
    'первое число - от 0 до 1
    'второе - от 0 до 10.
    'Третье - от 25 до 100
    'Четвертое - целое то 0 до 34
    Randomize
    MsgBox ("Группа случайных чисел: " + _
    Str(Rnd()) + ", " + _
    Str(Rnd() * 10) + ", " + _
    Str(Rnd() * 75 + 25) + ", " + _
    Str(Int(Rnd() * 34)))
    'Функция Sgn
    MsgBox ("Результат работы Sng для " + _
    Str(dblNumber) + " равняется " + _
    Str(Sgn(dblNumber)))
    'Синус
    MsgBox ("Синус " + _
    Str(dblNumber) + " равняется " + _
    Str(Sin(dblNumber)))
    'Квадратный корень
    MsgBox ("Квадратный корень " + _
    Str(dblNumber) + " равняется " + _
    Str(Sqr(dblNumber)))
    'Тангенс
    MsgBox ("Тангенс " + _
    Str(dblNumber) + " равняется " + _
    Str(Tan(dblNumber)))
```

Листинг 6.1. Обработчик события Click кнопки cmd_Calc

Обратите внимание на алгоритм получения случайного числа, находящегося в определенном диапазоне, с помощью функции `Rnd`. Предположим, нам нужно получить случайное число от `15` до `40`. Получим, для начала, число от `0` до `40`. Очевидно, что для этого нам понадобится такой вызов: `Rnd()*40`.

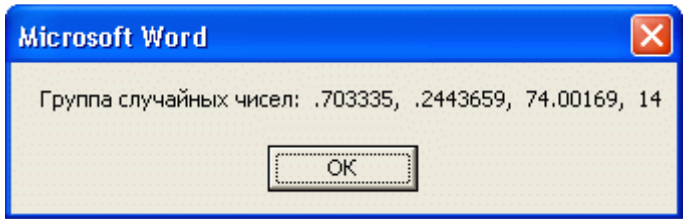
Чтобы "поднять" уровень наименьшего случайного числа, возвращаемого выражением, до `15`, сделаем следующее.

Во-первых, вычислим разность `40` и `15` - у нас получится `25`. Значит, чтобы получить случайное число от `0` до `25`, можно использовать вызов `Rnd()*25`.

Во-вторых, прибавим к полученному случайному числу 15. Теперь выражение для получения случайного числа от 15 до 40 выглядит так: Rnd()*25+15.

Проверим это высказывание на правильность. Функция Rnd, как известно, возвращает случайные числа от 0 до 1. Если функция возвратит 0 - результат вычисления выражения будет равен 15 (0*25+15). Если функция возвратит 1 - результат будет равен 40 (25*1+15). Промежуточные значения Rnd дадут искомые случайные числа между15 и 40.

На следующем рисунке вы можете видеть окно сообщения, содержащее результаты вызовов функции Rnd.



Выше мы работали, в основном, с числами, теперь поговорим о строковых переменных.

Строковые функции

В следующей таблице вы можете найти информацию об основных строковых функциях VBA.

Таблица Строковые функции

Функция	Описание
Len(string)	Возвращает длину строки. Например, длина строки "Добрый день" составляет 11 символов - учитывая пробел. Выходное значение имеет тип Long
LCase(string)	Возвращает строку, все символы которой записаны в нижнем регистре. Например, строка "Привет" превратится в "привет"
UCase(string)	Возвращает строку, все символы которой записаны в верхнем регистре. Например, для "Привет" мы получим "ПРИВЕТ"
String(number, character)	Возвращает строку, состоящую из number символов character
Left(string, length)	Возвращает length символов, начиная с первого левого символа строки string
Right(string, length)	Возвращает length символов, начиная с самого правого символа строки string
LTrim(string)	Возвращает строку, в которой вырезаны все пробелы слева
RTrim(string)	Вырезает из строки все пробелы справа
Trim(string)	Вырезает из строки все пробелы слева и справа
Mid(string, start[, length])	Вырезает из строки string с позиции start length символов
Asc(string)	Возвращает ASCII-код первого символа строки
Chr(charcode)	Возвращает символ, соответствующий коду символа

Работа со строками традиционно считается сложным разделом программирования для начинающих, поэтому проиллюстрируем вышеописанные функции примерами.

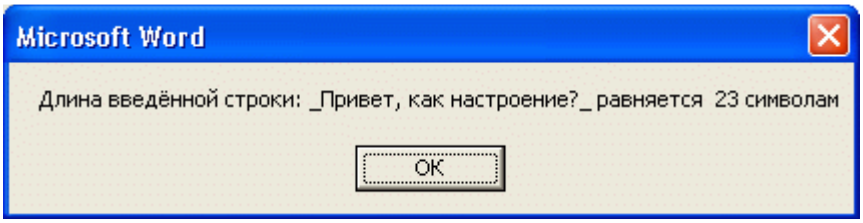
Создадим документ Microsoft Word и добавим на него следующие кнопки

Таблица Кнопки на листе		
Имя кнопки	**Надпись**	**Номер листинга**
cmd_Len	Длина строки	листинг 6.1.
cmd_Conv	Конверсия	листинг 6.2.
cmd_Mid	Вырезание	листинг 6.3.
cmd_Asc	Коды	листинг 6.5.

Обработчик нажатия кнопки cmd_Len будет содержать решение следующей задачи: сообщить пользователю длину введенного текста. Очевидно, для решения этой задачи нам понадобится функция Len.

```
'Переменная для хранения входной строки
Dim str_InpStr As String
'Переменная для хранения найденной длины строки
Dim lng_StrLen As Long
str_InpStr = InputBox("Введите строку")
'Вычисляем длину строки
lng_StrLen = Len(str_InpStr)
MsgBox ("Длина введенной строки: _" + _
str_InpStr + "_ равняется " + Str(lng_StrLen) + _
" символам")
```

На рисунке вы можете видеть результат вычисления длины строки.



Теперь займемся конверсией символов - функциями LCase и UCase .

```
'Переменная для хранения входной строки
Dim str_InpStr As String
'Переменная для хранения измененной строки
Dim str_NewStr As String
str_InpStr = InputBox("Введите текст")
'В str_NewStr окажется введеная строка
'в которой все прописные буквы заменены строчными
str_NewStr = LCase(str_InpStr)
MsgBox ("Измененная строка: " + str_NewStr)
'Теперь в str_NewStr будет та же строка
'в которой все буквы стали прописными
str_NewStr = UCase(str_InpStr)
MsgBox ("Измененная строка: " + str_NewStr)
```

На очереди - вырезание символов - функции Mid, LTrim, Rtrim, Left, Right . Среди этих функций наиболее мощной является Mid - используя ее, можно делать со строками очень много всего.

```
'Переменная для хранения входной строки
Dim str_InpStr As String
'Переменная для хранения вырезанных символов
Dim str_NewStr As String
'Зададим строку, с которой удобно будет работать
str_InpStr = "  Здравствуйте, Александр  "
'Функции удаления пробелов
'При выводе строки ставим перед ее началом
```

```
'и концом символ "_" для того чтобы
'лучше было видно наличие
'или отсутствие пробелов
MsgBox ("Мы работаем с такой строкой: " + _
"_" + str_InpStr + "_")
'LTrim - присваиваем результаты работы
'переменной и выводим информацию в MsgBox
str_NewStr = LTrim(str_InpStr)
MsgBox ("Результат работы LTrim: " + _
"_" + str_NewStr + "_")
'RTrim
MsgBox ("Результат работы RTrim: " + _
"_" + RTrim(str_InpStr) + "_")
'Trim
MsgBox ("Результат работы Trim: " + _
"_" + Trim(str_InpStr) + "_")
'Left - вырезаем из строки str_InpStr 12
'символов предварительно очистив ее
'от начальных пробелов
str_NewStr = Left(LTrim(str_InpStr), 12)
MsgBox ("Первые 12 символов слева: " + _
str_NewStr)
'Right - аналогично Вырезаем 9 символов справа
str_NewStr = Right(RTrim(str_InpStr), 9)
MsgBox ("Первые 9 символов справа: " + _
str_NewStr)
'Функция Mid - для начала выведем по одному символу
'со второй и пятнадцатой позиции строки
'преварительно очищенной от лишних пробелов
'в начале и в конце
str_NewStr = Mid(Trim(str_InpStr), 2, 1)
MsgBox ("Второй символ введенной строки: " + _
str_NewStr)
str_NewStr = Mid(Trim(str_InpStr), 15, 1)
MsgBox ("Пятнадцатый символ введенной строки: " + _
str_NewStr)
'Выведем 5 символов, начиная с 15 символа
str_NewStr = Mid(Trim(str_InpStr), 15, 5)
MsgBox ("Пять символов строки с 15-й позиции: " + _
str_NewStr)
```

Теперь рассмотрим примеры работы функций `Asc`, `Chr` и функции `String`. Чтобы работать с функциями `Asc` и `Chr` нам нужно познакомиться с понятием таблицы символов ASCII.

ASCII расшифровывается как American Standard Code For Information Intercharge - американский стандартный код для обмена информацией. Каждый символ в ASCII закодирован 8-мибитным кодом. В результате получается таблица, в которой каждому управляющему символу, цифре, букве латинского или национального алфавитов сопоставлен свой код. Коды записывают в различных представлениях - в основном - в шестнадцатеричном и десятичном. Мы будем пользоваться десятичной записью.

Мы не будем приводить здесь таблицу ASCII полностью, приведем лишь некоторые полезные коды и диапазоны кодов.

Коды в диапазоне 0-31 имеют управляющие символы. Символ возврата каретки (тот самый, который вставляется в документ при нажатии клавиши **Enter**) имеет код 13.

Коды в диапазоне 32-127 имеют латинские символы, цифры, знаки препинания - эта часть таблицы остается постоянной для различных кодовых таблиц. Например, пробел имеет код 32, точка - 46. Диапазон 48-57 занимают цифры от 0 до 9, диапазон 65-90 занимают заглавные латинские буквы от A до Z, диапазон 97-122 - прописные буквы a-z.

В диапазоне 128-225 расположены символы национальной кодировки. В русифицированных версиях MS Windows обычно применяется кодовая страница MS Windows 1251. В ней коды 192-223 имеют заглавные буквы от А до Я, 224-255 - прописные буквы от а до я.

Используя коды символов можно вводить в документы (или записывать в файлы, создаваемые программно) символы, которые нельзя ввести с клавиатуры.

В коде обработчика `Click` для `cmd_Asc` создадим программу, которая сначала запрашивает у пользователя ввод символа, после чего выводит его ASCII код, потом - ввод кода, после чего выводит соответствующий ему символ. Так же здесь мы посмотрим на то, как работает функция `String`:

```
'Переменная для хранения кода символа
Dim num_AscNumber
'Переменная для хранения символа
Dim str_Char
'Начало блока перевода кода в символ
num_AscNumber = Val(InputBox("Введите код символа"))
'В переменной str_Char теперь хранится символ
'с кодом num_AscNumber
str_Char = Chr(num_AscNumber)
MsgBox (str_Char + " - символ с кодом " + _
Str(num_AscNumber))
'Начало блока перевода символа в код
str_Char = InputBox("Введите символ")
'Теперь в переменной num_AscNumber хранится
'Код символа, введенного в str_Char
num_AscNumber = Asc(str_Char)
MsgBox ("Символу " + str_Char + " соответствует код" + _
Str(num_AscNumber))
'А теперь - пример функции String.
'выведем в окне сообщения 15 символов *
MsgBox ("15 символов *: " + String(15, "*"))
```

Пользовательские процедуры и функции

Выше мы пользовались *встроенными* процедурами и функциями VBA. Может показаться, что их достаточно много для решения любой задачи. Но даже при таком разнообразии порой нужны совершенно особые процедуры, которые пользователь может создать самостоятельно.

Напомним, что главное отличие процедуры от функции заключается в том, что функция возвращает в точку вызова некое значение, которое, как правило, является результатом обработки переданной функции информации. А процедура лишь выполняет какие-либо действия, но ничего в точку вызова не возвращает.

Например, очевидно, что при обработке такой последовательности команд: `num_A = num_B + val(str_C)` вместо выражения `val(str_C)` подставляется числовое значение переменной `str_C`, найденное благодаря функции `Val`. Или, скажем, такое выражение: `num_A = InputBox("Введите число")`. Здесь функция `InputBox` возвращает введенное пользователем число в переменную `num_A`.

Процедуры, как уже было сказано, лишь выполняют какие-то действия, не возвращая никаких значений в точку вызова. Например, если мы вызываем окно сообщения с надписью, не настраивая никаких дополнительных параметров, то мы можем решить, что имеем дело с процедурой. Такой вызов: `MsgBox ("Привет")`, будет отлично работать. В то же время, `MsgBox` может вести себя как функция, возвращая в точку вызова код нажатой кнопки при настройках кнопок окна сообщения отличных от стандартных.

Разобравшись в отличиях процедур и функций, поговорим о том, зачем они нам нужны. Во-первых, процедуры удобно использовать для сокращения объема программы, выделяя в них часто используемые блоки операторов. Если вам придется создавать реальные программы, вы очень скоро убедитесь в том, что некоторые операции (например - запись каких-либо значений на лист MS Excel, создание и сохранение документов и т.д.) могут быть нужными в различных местах программы и занимать порой несколько десятков строчек кода. Нерационально каждый раз копировать эти участки кода в нужное место. Во-первых - увеличивается размер текста программы, во-вторых - если вам понадобится изменить что-нибудь в часто используемом наборе операторов - придется искать все такие участки и редактировать каждый из них. Гораздо удобнее будет выделить часто используемый участок в виде самостоятельной процедуры и каждый раз, когда он будет нужен, вызывать его с помощью одной лишь строчки кода.

Пользовательская процедура

Рассмотрим пример создания и использования *пользовательской процедуры*. Предположим, нам нужно опросить пятерых пользователей, обработав и записав их ответы в файл. Самое главное в этом примере не то, куда мы будем записывать ответы, и как их будем обрабатывать, а то, что блок кода будет повторяться для каждого пользователя. Сначала напишем этот блок в расчете на одного пользователя. Для этого добавим в документ Microsoft Word кнопку cmd_User_Data и создадим для нее такой обработчик события:

```
Private Sub cmd_UserData_Click()
    Dim str_Name As String
    Dim byte_Age As Byte
    MsgBox ("Здравствуйте, вы пользователь № 1")
    str_Name = InputBox("Введите ваше имя")
    byte_Age = InputBox("Введите ваш возраст")
    'Здесь находится блок обработки и
    'записи введенных данных
End Sub
```

Для наглядности мы оставили операторы начала и конца процедуры.

Давайте подумаем, что нужно сделать с кодом при переходе ко второму пользователю. Очевидно, что мы сможем работать с теми же переменными, что и в первом случае, будем использовать те же операторы. Единственным, что изменится, будет надпись в первом окне сообщения. Вместо "Здравствуйте, вы пользователь №1" будет выведено "Здравствуйте, вы пользователь №2".

Это значит, что мы можем использовать весь код, который находится в данный момент в обработчике, для создания собственной процедуры.

Чтобы создать процедуру, добавим в проект новый модуль, например, командой **Insert o Module** (Вставить о Модуль).

В модуле создадим для начала "скелет" процедуры. Он будет выглядеть так:

```
Public Sub UserInput(UserNumber As Integer)
```

```
    'пользовательская процедура
    'для ввода и обработки данных
End Sub
```

Внутри процедуры - там, где будет располагаться ее тело, мы разместили комментарии. Вы уже знаете, что `End Sub` означает конец процедуры. А вот первая строка объявления выглядит гораздо интереснее. Давайте разберем ее составные части.

`UserInput` - это имя процедуры. Мы выбираем его самостоятельно. В скобках после имени находится объявление переменной, которую можно передать процедуре в качестве параметра. В нашем случае это переменная, имеющая имя `UserNumber` и тип `Integer`. Вспомните, как мы работаем с функцией `MsgBox` - вызывая ее мы пишем такой код:`MsgBox ("Привет")`. В данном случае мы передаем функции `MsgBox` текстовый параметр "Привет", который она выведет на экран. Точно так же, мы сможем передать нашей процедуре числовой параметр типа `Integer`, который будет "виден" ей под именем `UserNumber`. Вызов нашей процедуры для первого пользователя из обработчика события`Click` будет выглядеть так:

```
UserInput (1)
```

Тот номер, который мы указали в скобках, будет присвоен переменной `UserNumber` и мы сможем пользоваться ей внутри процедуры. Давайте проиллюстрируем это. Как вы помните, выше мы определили, что в общем коде будет изменяться лишь номер пользователя. Поэтому перепишем строку приглашения таким образом, чтобы для каждого номера пользователя она выводила бы собственный текст. После этого текст процедуры будет выглядеть так:

```
Public Sub UserInput(UserNumber As Integer)
     MsgBox _
     ("Здравствуйте, вы пользователь № " + _
     Str(UserNumber))
End Sub
```

Теперь рассмотрим остальные части первой строки процедуры. Ключевое слово `Sub` означает, что перед нами процедура. А модификатор доступа `Public` означает, что эту процедуру можно вызвать из любого модуля проекта. Существуют и другие модификаторы доступа, в частности, `Private`. Он дает доступ к процедуре только из того модуля, где она объявлена.

Теперь доработаем процедуру и обработчик события так, чтобы с их помощью мы могли решить поставленную выше задачу. Ниже вы можете видеть код обработчика события`Click` для кнопки и процедуры.

```
Private Sub cmd_UserData_Click()
    UserInput (1)
    UserInput (2)
    UserInput (3)
    UserInput (4)
    UserInput (5)
End Sub

Public Sub UserInput(UserNumber As Integer)
    Dim str_Name As String
    Dim byte_Age As Byte
    MsgBox _
     ("Здравствуйте, вы пользователь № " + _
     Str(UserNumber))
```

```
    str_Name = InputBox("Введите ваше имя")
    byte_Age = InputBox("Введите ваш возраст")
End Sub
```

Теперь поговорим о функциях.

Пользовательская функция

Рассмотрим пример создания и использования функции, которая выполняет очень простую задачу: возводит переданное ей число во вторую степень.

Функция, реализующая эту задачу, выглядит так:

```
Public Function Square(num_One As Double) As Double
    Square = num_One ^ 2
End Function
```

Объявление функции очень похоже на объявление процедуры. Здесь мы используем модификатор доступности `Public`, дающий доступ к функции из всех модулей. Ключевое слово `Function` означает, что мы объявляем функцию. Следом за ним, в скобках, идет объявление параметров, которые мы можем передать функции. В нашем случае это - одна переменная `num_One` типа `Double`. После объявления имени и параметров функции следует объявление типа самой функции, или, точнее - типа возвращаемого функцией значения. В нашем случае с помощью конструкции `As Double` мы задали тип функции `Double`. При объявлении функции можно не указывать тип - тогда он автоматически будет установлен в `Variant`.

Последняя строка объявления функции - `End Function`, означает конец функции. Как и в случае с процедурой, внутри тела функции можно разместить операторы, необходимые для выполнения задачи. Однако, здесь есть одна очень важная особенность. После того, как найдено значение, которое функция должна возвратить, в тексте кода функции нужно присвоить это значение переменной, которая имеет то же имя, что и функция. В нашем случае, при имени функции `Square` это присвоение выглядит как`Square = num_One ^ 2`.

Для испытания функции добавим в документ еще одну кнопку - `cmd_UserCalc`, подписанную как Функция с таким кодом:

```
Private Sub cmd_UserCalc_Click()
    Dim num_Res As Double
    Dim num_Input As Double
    num_Input = CDbl(InputBox("Введите число"))
    num_Res = Square(num_Input)
    MsgBox (Str(num_Input) + " во 2-й степени = " + _
    Str(num_Res))
End Sub
```

Как видите, здесь мы работаем с *пользовательской функцией* точно так же, как работали бы с одной из *встроенных функций*. Для наглядности мы использовали переменную, в которую записываем значение, возвращаемое функцией.

Пользовательские типы данных

Иногда возникает необходимость в работе с данными, имеющими особенную структуру. Например, какой тип должна иметь переменная, которая может хранить код пользова-

теля, его ФИО, возраст, адрес, телефон, дату рождения? Очевидно, что встроенными типами данных тут не обойтись. Можно, конечно, представить себе строковую переменную, которая содержит все эти данные, записанные подряд, но работать с такой переменной будет очень неудобно. В подобных случаях на помощь приходят*пользовательские типы данных*. Для определения нового типа данных используется конструкция `Type - End Type`.

Рассмотрим пример. Создадим документ Microsoft Word, добавим с помощью редактора Visual Basic новый модуль и разместим в нем такой код:

```
Type Worker ' начало объявления типа данных
    Usercode As Integer
    Name As String
    Phone As String
    BirthDate As Date
End Type
Public Sub WorkWithUser()
    Dim wrk_NewUser As Worker
    Dim wrk_TestUser As Worker
    Dim str_UserName As String
    wrk_NewUser.Usercode = 1
    wrk_NewUser.Name = "Петров Петр Петрович"
    wrk_NewUser.Phone = "8(928)8888888"
    wrk_NewUser.BirthDate = #8/12/1980#
    'Обмен данными между переменными
    'пользовательского типа
    'адекватен обычному обмену
    wrk_TestUser = wrk_NewUser
    'Присвоим строковой переменной
    'значение одной из частей
    'пользовательской переменной
    str_UserName = wrk_TestUser.Name
    MsgBox ("Имя пользователя: " + str_UserName)
End Sub
```

Новый тип данных должен быть объявлен вне процедуры - на уровне модуля. Мы специально привели здесь, вместе с текстом объявления типа, процедуру, в которой используется этот тип. Как видите, внутри объявления типа (между `Type` и `End Type`), находятся объявления переменных, а сразу после ключевого слова `Type` следует имя типа. Мы дали типу данных имя `Worker`, а это значит, что объявляя переменную в модуле, мы будем использовать это имя для указания ее типа данных. Имя нашего типа данных используется при объявлении переменной точно так же, как и имена встроенных типов - `String`, `Byte` и т.д. Мы объявили три переменные. `wrk_NewUser` типа `Worker`,`wrk_TestUser` того же типа и `str_UserName` - обычную строковую переменную.

После того, как переменная типа `Worker` объявлена, мы можем работать с ней. А именно, обращение к элементам переменной ведется через точку с использованием имен "внутренних" переменных типа. Сначала мы по очереди присваиваем каждой части переменной `wrk_NewUser` соответствующие значения. Далее мы присваиваем значение переменной `wrk_NewUser` переменной `wrk_TestUser`. Как видите, работа с пользовательскими переменными одинакового типа ни чем не отличается от работы с обычными переменными. Несмотря на сложную структуру переменной, обычный оператор присваивания отлично справляется с переносом значений из одной переменной в другую. Следующий этап нашей программы - извлечение значения одной из частей переменной типа `Worker`, а конкретно - имени пользователя, в строковую переменную`str_UserName`. После этого мы выводим переменную `str_UserName` в окне сообщения.

Константы

Константы очень похожи на переменные. Они имеют имена, которые строятся по тем же правилам, в них могут храниться различные значения и т.д. Главное отличие *констант*от переменных в том, что *константы* нельзя изменять в ходе выполнения программы. Это удобно в тех случаях, когда вы собираетесь много раз использовать в программе какое-то значение и хотите, чтобы оно гарантированно оставалось неизменным.

Для объявления *констант* используется ключевое слово `Const`. При объявлении *константы* нужно обязательно указать ее имя и присвоить ей нужное значение. При необходимости можно указать и тип *константы*. Вот как выглядит работа с *константами* в процедуре (листинг 5.24.):

```
Const str_Name As String = "Александр"
Const int_Size As Double = 18000
Dim num_NewSize As Double
MsgBox ("Здравствуйте, " + str_Name)
num_NewSize = int_Size * 2
```

Здесь мы объявили пару *констант* - одну строковую для хранения имени пользователя, вторую - типа `Double`. Строковую константу мы используем для построения строки вывода в операторе `MsgBox`, а числовую применяем для построения выражения, результат вычислении которого записывается в переменную `num_NewSize`.

В VBA имеется обширный набор встроенных *констант*. Они используются для работы с цветом (например, `vbRed` - красный и т.д.) для задания типов окон сообщений и во многих других случаях. Как правило, *константы*, допустимые в том или ином случае, можно найти в справочном материале, который появляется при наборе команд. Так же, подробные сведении о встроенных *константах* содержатся в справочной системе VBA.

Массивы

Решим простую задачу: ввести в ответ на вопросы программы фамилии сотрудников. Если фамилий немного - 5 или 10 - использование переменных, с которыми вы уже знакомы, вполне оправдано. Программа будет состоять из нескольких строк такого вида:

```
a = InputBox("Введите фамилию сотрудника").
```

Как быть, если нужно работать со списком из 20 фамилий? А если их будет 50, 1000 или их количество должно быть определено в ходе выполнения программы?

Для обработки больших объемов информации использовать переменные неудобно. Что же делать? Ответ прост: использовать *массивы*.

Массив - это именованный набор индексированных ячеек. Ячейки так же называют элементами или индексированными переменными.

У каждого *массива* 5 основных характеристик: имя, размерность, число элементов, номер первого элемента и тип элементов.

- **Имя** - правила именования *массивов* аналогичны правилам именования переменных.
- **Размерность** - одномерные *массивы* напоминают одну строку таблицы, каждая ячейка которой содержит какие-то данные. Многомерные *массивы* имеют больше измерений. Их можно сравнивать с таблицами, имеющими множество строк и столбцов и с наборами таблиц.

- **Нумерация элементов** подчиняется следующим правилам:
  - По умолчанию нумерация элементов *массива* начинается с 0. Первый по счету элемент получит индекс 0, второй - 1 и т.д.
  - В объявлении отдельного *массива* можно явно указать индекс его первого и последнего элемента, разделив их ключевым словом `To`.
  - Если вы хотите, чтобы индексы всех *массивов* начинались с 1, добавьте в раздел объявлений модуля (вне процедур, функций и обработчиков событий) команду`OptionBase 1`.
- **Тип** - подчиняется тем же правилам, которые мы ранее рассмотрели для переменных. Если пользователь не указал его явно, массив получит тип по умолчанию - `Variant`. Это требует больше системных ресурсов, но позволяет обрабатывать значения различных типов.

Не всегда количество элементов и размерность *массива* известны до начала работы программы. VBA умеет работать с *динамическими массивами*, параметры которых можно менять в ходе выполнения программы. О них мы поговорим ниже.

Одномерные массивы

Для объявления *массивов* используют оператор `Dim`. Объявить массив можно двумя способами. Первый заключается в указании общего количества элементов. Например, так:

```
Dim MyArrayA(30) As Single
```

Объявленный массив `MyArrayA` содержит 31 элемент (с индексами от 0 до 30) типа `Single`. Поскольку нумерация явно не задана, элементы получат индексы по обычным правилам.

Можно объявить массив и другим способом:

```
Dim MyArrayB(1 To 25)
```

Массив `MyArrayB` содержит 25 элементов. Границы нумерации заданы явно - первый элемент получит индекс 1, второй - 2 и т.д. Тип не указан - в *массиве* можно хранить любые данные.

Работа с элементами *массива* ничем не отличается от работы с переменными. Давайте решим следующую задачу:

1. Объявить одномерный массив на 3 элемента
2. Внести в первый элемент число 5 в программе, во второй - запросив значение с помощью окна ввода
3. Вычислить в третьем элементе *массива* произведение значений, хранящихся в первом и втором элементах.
4. Вывести полученное значение в окне сообщения.

Решение задачи следующее.

```
Dim A(2)
A(0) = 5
A(1) = InputBox("Введите значение второго элемента")
A(2) = A(0) * A(1)
MsgBox A(2)
```

Листинг 7.1. Работа с массивом

В конце работы программы, если на вопрос о вводе числа мы введем число 2, массив `A` будет иметь такой вид.

Таблица Массив A после работы программы

Индекс	0	1	2

Значение	5	2	10

Как видите, серьезное отличие *массивов* от переменных - использование индексов для работы с различными элементами *массива*. Однако, можно заметить, что эта задача легко решается с использованием переменных, без *массивов*.

Возможно, сейчас вы еще не вполне осознаете полезность *массивов*. Если сейчас вы напишите программу для ручного ввода данных в несколько элементов *массива*, она получится такой же громоздкой, как раньше. Так чем же *массивы* лучше? Ответ на этот вопрос кроется в использовании циклов.

Циклы

Циклы позволяют выполнять одни и те же команды много раз. В следующей таблице собрана информация об основных видах циклов.

Таблица Виды циклов

Название цикла	Вид
`For - Next`	С фиксированным количеством повторов. Выполняется заданное количество раз
`While - Wend`	С предусловием. Если не верно условие, заданное на входе в цикл, может не выполниться ни разу
`Do - Loop`	С постусловием. Выполняется по меньшей мере один раз.

Цикл For - Next

Решим задачу: вывести цифры от 1 до 10 в окнах сообщений. Ее можно решить и без использования циклов, написав 10 строк такого вида: `MsgBox ("1")`. А вот конструкция `For - Next` позволяет делать то же самое гораздо изящнее. В следующем листинге вы можете найти решение задачи.

```
For i = 1 To 10
    MsgBox (i)
Next i
```

Ключевое слово `For` задает начало цикла. После него следует переменная `i`, которая увеличивается при каждом проходе цикла. В ходе работы конструкции `For - Next`значение `i` изменяется от 1 до 10.

Начальное значение счетчика устанавливается при входе в цикл. Мы просто приравниваем `i` какому-либо числу. Конечное значение переменной задается после ключевого слова `To`. Переменная `i` доступна внутри цикла - ведь именно ее мы выводим в окне сообщения.

Ключевое слово `Next` с указанием переменной, к которой оно относится, закрывает цикл.

В качестве первого и последнего значения счетчика цикла можно использовать какую-нибудь переменную. Она может быть определена в ходе выполнения программы, но до входа в конструкцию `For - Next`.

Вы могли заметить, что переменная цикла меняется с приращением 1. Это приращение можно задать в явном виде с помощью ключевого слова `Step`. Ниже вы можете видеть пример оператора `For - Next`, выводящего нечетные числа в диапазоне от 1 до 10.

```
For i = 1 To 10 Step 2
    MsgBox (i)
Next i
```

С помощью ключевого слова `Step` можно создать не только возрастающий, но и убывающий счетчик. Для этого в `Step` надо указать отрицательное число и проследить за тем, чтобы первое значение переменной цикла было больше последнего. Например, так: `For i=10 to 1 Step -1`. Первое значение переменной в таком цикле будет равняться 10, второе - 9 и т.д. - до 1.

Теперь перейдем к совместному использованию циклов и *массивов*. Напишем программу, которая предлагает пользователю ввести 10 фамилий, сохраняет их в *массиве*, а потом выводит в окнах сообщений.

Готовая программа имеет следующий вид.

```
Dim MyArray(9)
For i = 0 To 9
MyArray(i) = InputBox("Введите фамилию №" & i + 1)
Next i
For i = 0 To 9 'Начало еще одного цикла
MsgBox ("Фамилия №" & (i + 1) & " " & MyArray(i))
Next i 'Конец цикла
```

Обсудив цикл типа `For-Next`, продолжим разговор о *массивах*. На очереди - многомерные *массивы*.

Многомерные массивы

Многомерные *массивы* имеют несколько измерений. Чаще всего применяются двумерные *массивы* (матрицы). Матрицу можно представить в виде обычной таблицы с несколькими строками и столбцами.

Для того чтобы объявить двумерный массив, нужно воспользоваться командой `Dim` с указанием размерности каждого из измерений. Остальные правила объявления таких-*массивов* и работы с ними аналогичны таковым для одномерных *массивов*. Например, мы можем указать лишь размеры измерения *массива*:

```
Dim MyArrayA(10, 1) As Single
```

Массив `MyArrayA` содержит 11 строк и 2 столбца типа `Single`.

Можно в явном виде задать границы размерностей:

```
Dim MyArrayB(1 To 25, 1 To 5)
```

Массив `MyArrayB` содержит 25 строк и 5 столбцов. Границы нумерации заданы явно. Тип не указан - в *массиве* можно хранить любые данные.

В листинге ниже приведен пример программы, которая объявляет двумерный массив 5x2 и предлагает ввести в него фамилии и номера телефонов сотрудников.

```
Dim MyArray(1 To 5, 1 To 2)
For i = 1 To 5
    MyArray(i, 1) = InputBox("Введите фамилию №" & i)
    MyArray(i, 2) = InputBox("Введите Телефон №" & i)
```

```
Next i
```

В таблице вы можете видеть массив `MyArray` после заполнения его фамилиями и номерами телефонов. В `MyArray (1,1)` мы внесли фамилию "Иванов", в `MyArray (1,2)` - телефон Иванова 898989898 и т.д.

Таблица Заполненный массив MyArray

Индекс	1	2
1	Иванов	898989898
2	Петров	343434343
3	Сидоров	565656565
4	Александров	121111212
5	Маринин	545454544

Эта программа очень похожа на те, которые мы писали для работы с одномерными *массивами*. В цикле, тело которого повторяется 5 раз, мы поочередно запрашиваем фамилию и номер телефона.

Один цикл неудобно использовать для работы с *массивами* больших размерностей. Нетрудно представить себе, какой громоздкой получится решение задачи копирования одной матрицы 100x100 в другую такую же. Кстати, при обработке данных в Microsoft Excel вам постоянно придется обращаться с большими двумерными матрицами. К счастью, существует механизм вложенных циклов, который помогает решать подобные задачи.

Вложенные циклы For-Next

Принцип работы вложенных циклов кроется в их названии. Все очень просто - один цикл вкладывается в другой. Например, для заполнения *массива* 10x10 случайными целыми числами от 1 до 10 можно написать такую программу .

```
Dim MyArray(1 To 10, 1 To 10)
For i = 1 To 10
    For j = 1 To 10
        MyArray(i, j) = Int(Rnd(1) * 10)
    Next j
Next i
```

Внешний цикл (`i`) выполняется один раз, после чего внутренний (`j`) - десять раз. За один проход внешнего цикла внутренний выполняет десять - заполняется первая строка*массива* (с индексами от 1,1 до 1,10) и т.д.

Динамические массивы

Когда вы используете массив, не всегда известно заранее, сколько элементов он будет иметь. В такой ситуации можно объявить массив, который содержит заведомо больше элементов, чем может понадобиться, но это приведет к нерациональному использованию системных ресурсов. Что же делать? Динамические *массивы* - вот достойный ответ на этот вопрос.

Решим задачу. Программа просит пользователя ввести количество сотрудников, которое сохраняет в переменной `ArraySize`, а потом создает массив, одна из размерностей которого равняется `ArraySize`.

Чтобы воспользоваться *динамическим массивом*, сначала нужно объявить пустой массив, например, командой `Dim MyArray()`, а потом задать размерность *массива* командой`ReDim` .

```
Dim MyArray()
ArraySize = InputBox("Введите количество сотрудников")
ReDim MyArray(1 To ArraySize, 1 To 2)
```

В итоге, если на вопрос программы о количестве сотрудников мы ввели число 15, будет создан двумерный массив размерностью 15x2.

Если в программе возникла ситуация, когда последней размерности объявленного и заполненного *массива* не хватает для хранения данных, вы можете увеличить его командой`ReDim` с ключевым словом `Preserve`. Благодаря ему данные, внесенные ранее в массив, будут сохранены. Например, для добавления двух дополнительных столбцов в*динамический массив* из листинга 7.7. нужно использовать такую команду:

```
ReDim Preserve MyArray(1 To ArraySize, 1 To 4)
```

Дополнительные команды работы с массивами

Для работы с *массивами* вы можете использовать еще некоторые команды.

`Array` (Список аргументов)- позволяет быстро заполнять массив. Например, в листинге 7.8. массив `MyArray` заполняется числами 1, 2, 6, 9 и 19, после чего первый элемент*массива* выводится в окне сообщения.

```
Dim MyArray
MyArray = Array(1, 2, 6, 9, 10)
MsgBox MyArray(0)
```

`IsArray` (Имя переменной) - возвращает `True` если переменная является массивом. Например, в листинге ниже мы объявляем две переменные - одну из них как массив, вторую - как обычную переменную. Далее мы используем оператор `IsArray` для проверки того, является ли переменная массивом. После чего программа выводит соответствующее сообщение. Здесь мы использовали оператор сравнения `If`, подробности о котором мы рассмотрим ниже.

```
Dim MyArray(10)
Dim MyArr
If IsArray(MyArray) Then _
MsgBox ("Переменная MyArray - массив") _
Else MsgBox ("Переменна MyArray - не массив")
If IsArray(MyArr) Then _
MsgBox ("Переменная MyArr - массив") _
Else MsgBox ("Переменна MyArr - не массив")
```

`LBound` (Имя Массива, Размерность) - возвращает нижнюю границу для указанной размерности массива.

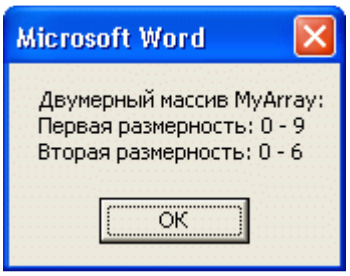
`UBound` (Имя Массива, Размерность) - возвращает верхнюю границу для указанной размерности массива.

Рассмотрим пример. Создадим *динамический двумерный массив*, размерности которого заданы с помощью генератора случайных чисел. После этого с помощью

операторовLBound и UBound узнаем размерности *массива* и выведем их в окнах сообщений. Далее - используем двойной цикл для заполнения *массива* случайными числами .

```
Dim MyArray()
ReDim MyArray(Int(Rnd * 5 + 5), Int(Rnd * 5 + 5))
MsgBox ("Двумерный массив MyArray:" + Chr(13) + _
"Первая размерность:" + _
Str(LBound(MyArray, 1)) + " -" + _
Str(UBound(MyArray, 1)) + Chr(13) + _
"Вторая размерность:" + _
Str(LBound(MyArray, 2)) + " -" + _
Str(UBound(MyArray, 2)))
For i = LBound(MyArray, 1) To UBound(MyArray, 1)
    For j = LBound(MyArray, 2) To UBound(MyArray, 2)
        MyArray(i, j) = Int(Rnd * 100)
    Next j
Next i
```

В нашем случае команда LBound для обеих размерностей *массива* возвращает 0 так как по умолчанию нумерация элементов *массива* начинается с 0. А вот Uboundвозвращает границу каждой из размерностей, которая установлена случайным образом с помощью оператора ReDim. На рисунке вы можете видеть окно сообщения с информацией о границах *массива*.



Erase (Имя массива) - очистить массив. Элементы обычных *массивов*, содержащих числовые данные, обнуляются. Если мы применим команду Erase к *массиву* строк - каждый его элемент будет хранить строку нулевой длины (""). Применяя команду Erase к *динамическому массиву*, мы очищаем память, выделенную этому *массиву* командой ReDim. Причем, для повторного использования *динамического массива*, придется снова устанавливать его размерности. Если команда Erase применяется к объектному *массиву*, в каждый его элемент записывается специальное значение Nothing, которое означает пустую ссылку на объект.

Теперь, когда мы обсудили циклы For-Next и работу с *массивами*, поговорим о других типах циклов.

Цикл с предусловием

Как вы уже знаете, *цикл с предусловием* While - Wend выполняется до тех пор, пока условие, указанное на входе, верно.

Ниже представлено решение такой задачи: выводить на экран случайные числа от 0 до 20 до тех пор, пока не будет выведено число больше 10.

```
A = 1
While A < 10
    A = Int(Rnd() * 20)
```

```
        MsgBox A
    Wend
```

Сначала мы приравниваем переменной A число 1. Проверка при входе в цикл находит, что A меньше 10 и запускает первый проход. Переменной A приравнивается целое случайное число, это число выводится в окне сообщения. Дальше следует новая проверка - если A все еще меньше 10 - все повторяется снова. Если A больше или равно 10 - число выводится, после чего выполнение цикла прекращается.

Теперь рассмотрим *цикл с постусловием*.

Цикл с постусловием

Цикл Do-Loop While выполняется до тех пор, пока значение на выходе из цикла верно. Подобные циклы используют, например, для проверки правильности ввода каких-либо данных пользователем. Если данные введены неверно - цикл выполняется снова.

Аналогично действует цикл Do-Loop Until - он будет выполняться до тех пор, пока условие цикла неверно (то есть равно False).

В листинге ниже вы можете найти пример такого цикла. Здесь пользователю предлагается ввести какое-нибудь число. Если введено не число (то есть функция IsNumericвозвратит False), программа выведет окно ввода снова.

```
Dim var_A
Do
    var_A = InputBox("Введите число")
Loop Until IsNumeric(var_A)
```

Принятие решений: If-Then-Else

Программы на VBA умеют принимать решения - для этого существуют операторы условного перехода. Они объединены в конструкцию If - Then - Else.

В этой конструкции могут быть использованы следующие операторы сравнения .

Таблица Операторы сравнения

Оператор	Описание
=	Равно
<>	Не равно
>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно
Like	Сравнение строки с шаблоном
Is	Сравнение объектов

Напишем простую программу, которая спрашивает у пользователя его возраст. Если введенный возраст меньше 18 - программа должна вывести надпись "Вам менее 18 лет", если больше или равен 18 - надпись "Вам 18 или больше".

```
a = InputBox("Введите ваш возраст")
If a < 18 Then MsgBox ("Вам меньше 18")
```

```
If a >= 18 Then MsgBox ("Вам 18 или больше")
```

Здесь представлен оператор в простейшем виде - проверка условия и выполнение однострочной команды. А что если нужно выполнить не одну команду, а несколько? Для этого служит команда End If

```
a = InputBox("Введите ваш возраст")
If a < 18 Then
    MsgBox ("Вам меньше 18")
    MsgBox ("Вам не следует смотреть этот фильм")
End If
If a >= 18 Then MsgBox ("Добро пожаловать")
```

Здесь программа выводит два сообщения, если пользователю меньше 18 лет.

Оператор может иметь вид If - Then - Else . Благодаря ему два оператора If - Then можно объединить в один.

```
a = InputBox("Введите ваш возраст")
If a < 18 Then MsgBox ("Вам меньше 18") _
Else MsgBox ("Вам больше 18")
```

Применение команды Else так же позволяет исполнять многострочные команды. Например, вот так .

```
a = InputBox("Введите ваш возраст")
If a < 18 Then
    MsgBox ("Вам меньше 18")
    MsgBox ("Вам не следует смотреть этот фильм")
Else
    MsgBox ("Вам больше 18")
    MsgBox ("Добро пожаловать")
End If
```

В операторе If возможно использование сложных условий. Например, вы просите пользователя ввести имя и пароль. Если они соответствуют данным, хранящимся в системе, программа выводит приветствие, иначе - сообщает о том, что пользователь ввел неправильные сведения. Очевидно, что нам нужно проверить два условия, причем важно, чтобы и то и другое выполнялось. Для этого можно воспользоваться логическим оператором And, который трактуется как "И". Конструкция с And выглядит так.

```
UserName = InputBox("Введите имя пользователя")
UserPass = InputBox("Введите ваш пароль")
If UserName = "Александр" And UserPass = "12345" Then
    MsgBox ("Добро пожаловать в систему")
Else
    MsgBox ("Неверное имя пользователя или пароль")
End If
```

Словесно вышеприведенную конструкцию можно описать так: "Если имя равно Александр и пароль равен 12345…".

Логический оператор Or (переводится как "Или") позволяет принять положительное решение, если выполняется хотя бы одно из условий. Например, нам нужно узнать имя пользователя, и если оно соответствует одному из имен, зарегистрированных в систе-

ме, вывести приветствие, иначе - вывести информацию об отсутствии в системе такого пользователя.

```
UserName = InputBox("Введите имя пользователя")
If UserName = "Александр" Or UserName = "Сергей" Or _
UserName = "Николай" Then
    MsgBox ("Добро пожаловать в систему, " & UserName)
Else
    MsgBox ("В системе нет такого пользователя!")
End If
```

And и Or можно использовать вместе.

Оператор Not (Не) позволяет задавать условия с отрицанием. Например, вы можете пропустить в систему всех пользователей кроме пользователя с именем "Владимир".

Для проверки дополнительных условий можно использовать оператор If - Then - Else.

Выше, в таблице, есть пара необычных операторов - Like для сравнения строк с шаблоном, и Is - для сравнения объектных переменных.

Сравнение с использованием Like и Is

Оператор Like используется для сравнения строк с шаблонами. Шаблон - это особым образом записанная последовательность символов. При построении шаблонов используются специальные символы, приведенные в таблице.

Таблица Символы для построения шаблонов

Символы	Описание
?	Любой одиночный символ
*	Любое количество любых символов
#	Любая одиночная цифра
[список символов]	Любой одиночный символ, входящий в список символов
[!список символов]	Любой одиночный символ, не входящий в список

После построения шаблона его заключают в кавычки.

Давайте рассмотрим пример, реализующий следующие проверки.

- Узнать, есть ли в строке прописные и заглавные буквы латинского алфавита.
- Проверить, состоит ли введенное слово из четырех символов (цифр или букв)
- Проверить, состоит ли введенная последовательность из двух любых символов (цифр или букв) и двух цифр
- Проверить, нет ли во введенной строке русских букв "а" и "о"
- Если введенная строка начинается двумя буквами "d" и заканчивается тремя буквами "f", сообщить об этом

Далее вы можете найти решение этой задачи

```
Dim str_Inp As String
str_Inp = InputBox("Введите строку")
'Есть ли латинские буквы в строке
If str_Inp Like "[a-z]" Or _
str_Inp Like "[A-Z]" _
Then MsgBox ("В строке есть латинские буквы")
'Состоит ли введенное слово из 4-х символов
```

```
If str_Inp Like "????" Then _
MsgBox ("Введенное слово состоит из 4-х символов")
'Состоит ли введенная последовательность
'из 2-х любых символов и 2-х цифр
If str_Inp Like "??##" Then _
MsgBox ("Введены два любых символа и две цифры")
'Проверка на отсутствие букв
'"а", "о"
If str_Inp Like "[!а]" And _
str_Inp Like "[!о]" Then
MsgBox ("В строке нет букв " + Chr(34) + _
"а" + Chr(34) + " и " + Chr(34) + "о" + Chr(34))
End If
'Проверка на наличие в начале
'введенной строки двух букв "d", а в конце
'трех "f"
If str_Inp Like "dd*fff" _
Then MsgBox ("Строка имеет вид: dd*fff")
```

Как вы можете видеть, шаблон для сравнении с текстом всегда заключается в двойные кавычки. На основе информации об использовании шаблонных символов вы можете самостоятельно построить выражения для проверки различных последовательностей.

Оператор `Is` используется для работы с объектными переменными. Мы будем подробно рассматривать их в начале следующей главы. Оператор `Is` проверяет, ссылаются ли две объектные переменные на один и тот же объект. Если это так - он возвращает `True`, если нет - то `False`.

Принятие решений: Select Case

Если вам предстоит проверить данные на множество значений, для каждого из которых надо выполнить какое-то особое действие, можно использовать множество операторов`If` или один `Select Case`.

Далее приведен пример использования `Select Case` - в зависимости от введенного имени программа здоровается с пользователями по-разному.

```
Dim str_UserName As String
str_UserName = InputBox("Введите имя пользователя")
Select Case str_UserName
    Case "Александр"
        MsgBox ("Привет")
    Case "Сергей"
        MsgBox ("Здравствуй")
    Case "Николай"
        MsgBox ("Добро пожаловать")
    Case Else
        MsgBox ("Не существует такого пользователя")
End Select
```

После `Select Case` указано имя переменной, анализ которой осуществляется. Дальше расположено произвольное количество вариантов - каждый вариант после ключевого слова `Case`. Если не выполнен ни один из `Case` - выполняется `Case Else`. В конце этой конструкции находится оператор `End Select`.

Оператор безусловного перехода

В VBA есть один оператор, которым не рекомендуется пользоваться при написании программ. Это *оператор безусловного перехода* `GoTo`. Он позволяет передать управление в определенное место программы, заданное номером строки или меткой. Меткой строки может быть любое слово (подчиняющееся правилам именования переменных), после которого следует двоеточие.

Например, *цикл с постусловием* можно создать таким способом:

```
    Dim num_Cycle
    num_Cycle = 0
Cycle_Start:
    num_Cycle = num_Cycle + 1
    MsgBox ("Проход цикла номер: " + Str(num_Cycle))
    If num_Cycle < 5 Then GoTo Cycle_Start
```

Здесь мы создаем метку `Cycle_Start`, которая означает начало цикла. После этого увеличиваем на единицу переменную, которая является счетчиком цикла, выводим ее в окне сообщения. В последней строке мы проверяем переменную. Если она меньше 5 - команда `GoTo Cycle_Start` передает управление на соответствующую метку. В такой простой конструкции, как наша, особенных сложностей в чтении кода не видно. Но стоит программе хотя бы немного увеличиться, несколько подобных циклов (особенно - вложенных, а еще хуже - использующих номер строки для перехода) превратят ее в сложный для чтения и правки текст.

`GoTo` делает программы трудночитаемыми, его практически всегда можно заменить, используя другие программные конструкции. Единственное место, где `GoTo` пока незаменим - это использование его при написании обработчиков ошибок времени выполнения. В следующей лекции мы поговорим о таких ошибках и об использовании `GoTo` для организации подпроцедур в исполняемой процедуре.

Работа с файлами

Программы, написанные на VBA, умеют работать с внешними *файлами*. В частности, на практике могут возникнуть задачи по поиску *файлов* в директориях, по открытию, обработке, сохранению *файлов*. Открытие, обработка, сохранение - дело отдельных приложений (например, MS Word, MS Excel) - то есть эти задачи решаются с помощью объектных моделей этих приложений. А вот поиск *файлов* осуществляется общими для всех методами VBA.

Как правило, чтобы открыть *файл*, нужно знать его имя. Иными словами, поиск *файлов* заключается в получении имен *файлов*, находящихся в определенной директории. Для этого можно использовать команду `Dir`. Она возвращает строку, содержащую имя *файла*, используя путь, заданный при вызове. Давайте рассмотрим конструкцию (листинг 7.23.), которая позволяет найти все *файлы*, находящиеся в корневой директории диска C.

```
    var_Doc = Dir("C:\*.*")
    Do While var_Doc <> ""
        MsgBox var_Doc
        var_Doc = Dir()
    Loop
```

Сначала мы присваиваем переменной `var_Doc` первое найденное имя *файла*. Очевидно, что узнав имя *файла*, мы можем сказать, что нашли этот файл на диске. В самом про-

стом варианте использования функции Dir в качестве параметров мы передаем ей путь и маску имени *файла*. Знак * в маске означает любое количество любых символов. Следовательно, *.* означает "все файлы" - то есть *файлы* с любыми именами и любыми расширениями. В маске можно так же использовать знак ? - он символизирует один любой символ. Если не указать путь к *файлам*, а лишь маску - Dir будет искать их в текущей директории. Например, для Microsoft Word по умолчанию это папка Мои документы.

Помимо пути и маски при поиске *файлов* можно указать некоторые дополнительные параметры. Так, по умолчанию функция ищет лишь обычные *файлы*, не обращая внимания на папки, скрытые и системные *файлы*. Чтобы функция нашла по заданному пути не только *файлы*, но и папки, ее нужно вызвать так:

```
var_Doc = Dir("C:\*.*", vbDirectory)
```

Обратите внимание на то, что после пути и маски указан параметр vbDirectory - он указывает функции, что она должна включить в поиск и директории.

После того, как первое найденное имя присвоено переменной, мы запускаем *цикл с предусловием,* в котором проверяем, не пуста ли эта переменная. Если Dir не обнаружил по указанному пути ничего подходящего под заданную маску, он возвратит, пустую строку. Следовательно, цикл в таком случае не выполнится ни разу.

В цикле есть две строки. Первая выводит найденное имя на экран, а вторая - вызывает функцию Dir еще раз - без параметров. Такой вызов возвращает следующее имя*файла*, подходящее под заданный при первом вызове Dir шаблон. После этого все повторяется. В реальной программе в такой цикл можно вставить команды для работы с найденными *файлами*.

Помимо Dir полезной может оказаться команда ChDir. Она позволяет перейти в указанную при ее вызове директорию, которая будет использоваться в качестве директории по умолчанию. Такая конструкция, предшествующая циклу из предыдущего примера позволит найти все *файлы* в папке "Документы", которая расположена по пути "C:\Документы":

```
ChDir ("C:\Документы")
var_doc = Dir("*.*")
```

Существует множество других функций работы с *файлами*, вы можете найти информацию о них в справочной системе VBA.