МОСКОВСКИЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ
ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ (МАДИ)

И.Э. СААКЯН, И.А. ЕВСТРАТОВА

ИНФОРМАТИКА

Часть 3. Основы программирования

В печать
Проректор
по учебной работе Татаринов В.В.

Цена 1070 руб.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ
ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (МАДИ)»

Кафедра «Автоматизированные системы управления»

И.Э. СААКЯН, И.А. ЕВСТРАТОВА

ИНФОРМАТИКА

Часть 3. Основы программирования

УЧЕБНОЕ ПОСОБИЕ

Допущено Федеральным учебно-методическим объединением в системе высшего образования по укрупненной группе специальностей и направлений подготовки высшего образования 09.00.00 «Информатика и вычислительная техника» в качестве учебного пособия для студентов высших учебных заведений, обучающихся по направлению подготовки бакалавра 09.03.02 «Информационные системы и технологии»

МОСКВА
МАДИ
2022

УДК 004.43:378.147
ББК 32.97
С120

Рецензенты:
доктор технических наук, профессор МГТУ им. Н.Э. Баумана
В.Ю. Строганов;
доктор технических наук, профессор МАДИ *А.В. Остроух;*

Саакян, И.Э.

С120 Информатика. В 6 частях. Часть 3. Основы программирования: учеб. пособие / И.Э. Саакян, И.А. Евстратова. – М.: МАДИ, 2022. – 212 с.

Представленный материал предназначен для обеспечения учебного процесса по дисциплинам «Информатика», «Информатика и основы программирования», «Информационные технологии» с обучающимися первого и второго курсов всех направлений подготовки в рамках ФГОС ВО третьего поколения (3+) и актуализированных ФГОС ВО (3++).

Третья часть учебного пособия посвящена основам программирования на алгоритмических языках Паскаль (язык со строгой типизацией и интуитивно понятным синтаксисом) и Бейсик (символьный язык программирования общего назначения для начинающих) в реализациях соответственно $PascalABC.NET$ (версия 2015 г.) и VBA для $Microsoft\ Office\ 2010-2019$.

Пособие предназначено для обучающихся всех направлений подготовки факультета управления, оно может быть полезно обучающимся других факультетов, интересующихся информатикой, а также начинающим программистам.

УДК 004.43:378.147
ББК 32.97

Оглавление

ВВЕДЕНИЕ

Основное назначение любого языка как средства общения между субъектами – донести информацию с минимальным искажением и потерями. Смысл использования языка (назовём это общением) обусловлен субъектами общения, т.е. теми, кто «говорит» и «слушает». Нет смысла передавать информацию, которую не способен понять говорящий или слушающий. В век всеобщей компьютеризации такими субъектами полноправно становятся человек и компьютер, т.е. языковые средства необходимы для понимания слушателя (компилятора) и говорящего (программиста). И это неплохо работает пока слушатель и говорящий неизменны. При этом «слушателю» для понимания «говорящий» (программист) передаёт список последовательно записанных инструкций (инструкции называют кодом), который называют – программой, а язык, на котором записан этот список, – языком программирования. В идеале программа – это не способ управления компьютером. Это формализация знаний программиста о задаче и способах её решения. Основное назначение современного языка программирования – позволить программисту сделать это с минимальными искажениями и потерями. И язык должен позволять работать с собой на произвольном уровне понимания программиста. Поэтому он должен по крайней мере в основе своей быть независимым от «говорящего» и «слушающего». Только тогда он сможет адаптироваться под каждого конкретного «пользователя».

С другой стороны, каким бы совершенным ни был компьютер, без программного обеспечения он представляет собой просто груду металла и пластика. Именно программы определяют, что и как делает компьютер, в какой последовательности он выполняет те или иные операции.

Первые языки программирования начали появляться в начале пятидесятых годов прошлого века и использовались для преобразования простых арифметических выражений в машинный код. ***Машинный код*** – это система команд вычислительной машины, интерпретируемых непосредственно микропроцессором. Но человеку писать

программу в машинных кодах очень неудобно. Для того чтобы облегчить труд программиста, и начали создаваться языки программирования.

Языки программирования делятся на языки высокого уровня и низкого. Чем выше уровень языка, тем легче на нем писать программисту. Такой язык более понятен человеку, так как позволяет с помощью простых смысловых конструкций задавать необходимую последовательность действий. После создания программы происходит её компиляция – то есть автоматический перевод в понятный процессору язык машинных кодов. Языки низкого уровня находятся гораздо ближе к языку машинных кодов, поэтому писать на них труднее. Но у них есть своё преимущество – написанные на таком языке программы получаются очень быстрыми и компактными. Наиболее популярным низкоуровневым языком является $Assembler$.

Программирование сегодня – это бурно развивающаяся отрасль производства программных продуктов. В конце прошлого века общаться с компьютерами можно было лишь с помощью программирования, и поэтому программирование изучали фактически во всех учебных заведениях (в т. ч. в школах и вузах). Времена изменились, общаться с компьютерами можно уже с помощью готовых компьютерных программ, и нужда в массовом обучении программированию вроде бы отпала. Однако все оказалось не так просто. В современные прикладные пакеты включаются, как правило, дополнительные средства программирования, обеспечивающие расширение возможностей этих пакетов. Например, практически в любом пакете ($MS\ Office$) есть среда программирования VBA ($Visual\ Basic\ for\ Applications$), обеспечивающая расширение возможностей этого пакета; профессиональная работа с системой «1С Предприятие» требует постоянного программирования для настройки на потребности каждой фирмы. Речь уже идёт о новом подходе, в рамках которого программирование – это обязательная компонента подготовки специалистов, собирающихся профессионально работать в определённой сфере, в которой предполагается использование IT-технологий.

Программирование сегодня – это не только и не столько знание языка программирования. Прежде всего это знание технологии программирования, умение проектировать и разрабатывать программы и программные комплексы на основе этой технологии, умение строить модели, ставить задачи и иметь представление о коллективной разработке программных продуктов. Все это принято называть «***культурой программирования***».

В данном пособии вы познакомитесь с языками программирования: Паскаль (точнее, с его разновидностью $Object\ Pascal$) и VBA – $Visual\ Basic\ for\ Application$ - современным диалектом языка программирования $BASIC$, который был создан в начале 60-х годов (***BASIC*** – сокращение от ***Beginner's All – Purpose Symbolic Instruction Code***, или символьный язык программирования общего назначения для начинающих), с элементами структурного программирования (т. е. программирования без использования оператора безусловного перехода), с технологией проектирования «сверху вниз», с модульным программированием (т. е. с разбиением программы на подпрограммы для удобства отладки и коллективной реализации) и элементами объектно-ориентированного программирования.

Глава 1. ЧТО ТАКОЕ «ПРОГРАММИРОВАНИЕ»

1.1. Что такое «программа» и как её исполняет компьютер

Компьютер представляет собой устройство для исполнения программ. «Мозгом» компьютера является процессор, который призван «понимать» и исполнять эти программы. «Программа» – это последовательность предписаний (команд), записанных на языке, понятном исполнителю.

В нашем случае исполнителем является процессор. А что за язык, который этот процессор понимает? Как известно, компьютер хранит информацию, закодированную с помощью двоичных чисел. Программа для компьютера – это обычная информация, поэтому она хранится в виде набора двоичных чисел (о двоичных числах см. [1], [2]). Но согласно определению, это не простой набор чисел, а именно набор команд, понятных процессору. Сочетание команд, которые понимает процессор, и правил их написания принято называть ***машинным языком,*** или языком программирования низкого уровня. Например, предположим, что некий процессор понимает следующие операции:

00000001 – сложить;

00000111 – разделить;

00000100 – переслать из одной ячейки в другую.

Правило написания команды для современного процессора выглядит следующим образом:

<код операции> <адрес первого операнда> <адрес второго операнда>.

Здесь под ***операндом*** понимается некоторое значение, с которым производится операция; ***адрес операнда*** – это адрес блока памяти, в котором находится операнд.

В таком случае, например, фрагмент программы, определяющей среднее арифметическое четырёх чисел, будет выглядеть следующим образом (табл. 1):

Таблица 1

Пример компьютерной программы в двоичных кодах

Код операции	Адрес 1	Адрес 2	Комментарии
00000001	00011000	00011001	Сложить числа, хранящиеся в блоках памяти по адресам 00011000 и 00011001 и поместить результат в блок памяти с адресом 00011001
00000001	00011001	00011011	Сложить полученную сумму из блока памяти 00011001 и третье число, размещённое в блоке памяти 00011011, и поместить результат в блок памяти с адресом 00011011
00000001	00011011	00011110	Сложить полученную сумму из блока памяти 00011011 и четвёртое число, размещённое в блоке памяти 00011110, и поместить результат в блок памяти с адресом 00011110
00000111	00011110	00111110	Разделить полученную сумму на число, хранящееся по адресу 00111110, и поместить по тому же адресу результат
00000100	00111110	10111000	Переслать содержимое блока памяти с адресом 00111110 в блок памяти с адресом 10111000

Разумеется, у реальной программы есть специальный заголовок, по которому процессор определяет, что это именно программа, и на практике программы существенно сложнее приведённого примера. Однако в редких случаях с помощью машинных кодов пишутся программы для специальных устройств (стиральные машины, управляемые ракеты и т. п.), в которые встроены относительно слабые процессоры и ограниченная оперативная память, но от которых требуется практически мгновенное исполнение команд.

Программы, написанные с помощью машинных кодов, занимают существенно меньшую память, чем созданные с помощью других средств, и, как следствие, быстрее исполняются.

1.2. Языки высокого уровня. Компиляторы и интерпретаторы

Технология написания программ на машинных языках весьма трудоёмка и для создания больших программ фактически непригодна. Именно поэтому стали придумывать языки высокого уровня,

т. е. языки, с помощью которых можно было бы легко и удобно разрабатывать большие программы. Кроме того, подобные языки должны быть доступны большому кругу людей. Это значит, что в конечном итоге язык программирования должен максимально приблизиться к естественному языку человека. Идеал не достигнут, но языки программирования уже стали близки к естественным языкам. Но как в этом случае процессор понимает программы, написанные на этих языках? Да никак. Дело в том, что языки высокого уровня созданы для того, чтобы люди могли естественным образом сформулировать перечень действий, выполняемых компьютером. Ввиду того, что компьютеры языков высокого уровня не понимают, были разработаны специальные программы, называемые ***трансляторами***.

Транслятор (***translator***) – это программа, предназначенная для перевода (трансляции) описания алгоритма с одного формального языка на другой.

Алгоритм создания программы теперь предусматривает использование шага трансляции (рис. 1).

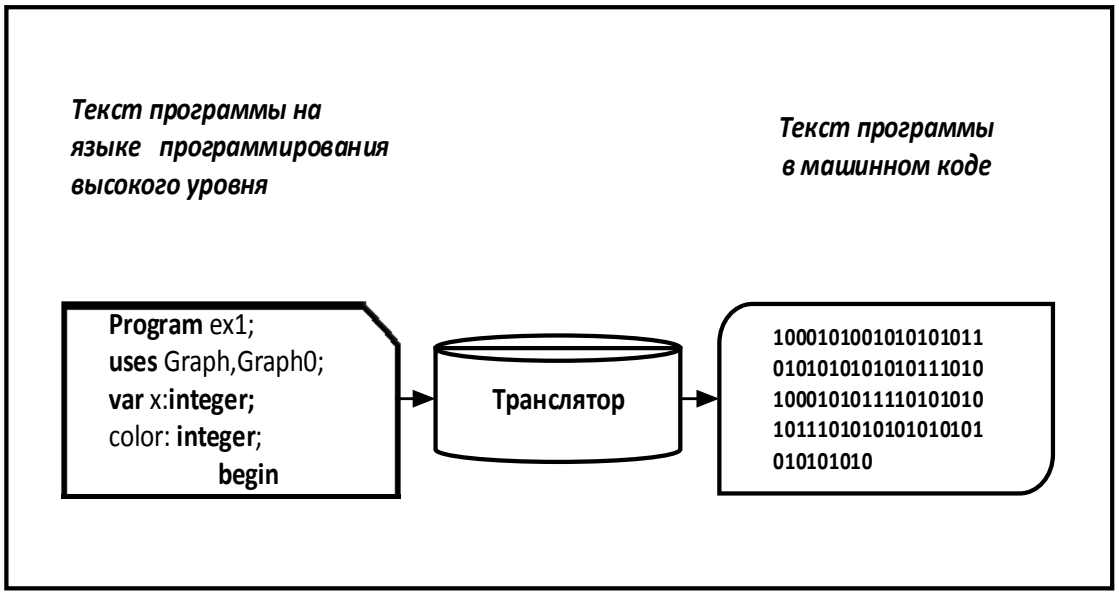


Рис. 1. Алгоритм трансляции программы

Этап превращения программы, написанной на языке высокого уровня, в машинный код реализуется в двух вариантах. В первом случае транслятор берет из файла программу на языке высокого уровня

и переводит её целиком в программу на машинном языке, создавая отдельный файл с расширением *.exe* (или *.com*). Программу, записанную в такой файл, принято называть ***исполняемым модулем***, а транслятор, который выполняет такой перевод, называют ***компилятором***. Программы, написанные на Паскале, используют компилятор.

Во втором случае транслятор берет из файла с программой на языке высокого уровня по одному предписанию (команде), транслирует её в машинный код и передаёт полученную команду процессору для исполнения. Такой транслятор называют ***интерпретатором***. На таком принципе работают программы, написанные на VBA.

Существуют несколько сотен языков программирования высокого уровня. Каждый из них предназначен для решения определённого круга задач. Например, $BASIC$ и $FORTRAN$ – для вычислений, $PROLOG$ – для создания систем искусственного интеллекта, $Pascal$ – для обучения программированию, $COBOL$ – для обработки больших объёмов данных, $C++$ – для профессионального программирования больших приложений и т. п.

Язык программирования – это специально обусловленный набор символов, слов и мнемонических (особым образом организованных и заранее оговорённых) сокращений, используемых для записи набора команд (программы), воспринимаемых компьютером.

Синтаксис языка программирования – это перечень правил записи программ из элементов этого языка.

Программирование – это технология разработки программ с помощью языков программирования.

1.2.1.PascalABC.NET

Одним из современных широко распространённых языков программирования является Паскаль. Он был создан на рубеже 1960–70-х годов швейцарским учёным Никлаусом Виртом и первоначально предназначался для обучения программированию. В настоящее время данный язык используется и для профессиональной разработки малых и средних компьютерных систем.

Язык $PascalABC.NET$ включает в себя практически весь стандартный язык Паскаль, а также большинство языковых расширений языка $Delphi$. Однако этих средств недостаточно для современного программирования. Именно поэтому $PascalABC.NET$ расширен рядом конструкций, а его стандартный модуль - рядом подпрограмм, типов и классов, что позволяет создавать легко читающиеся приложения средней сложности.

Кроме этого, язык $PascalABC.NET$ использует большинство средств, предоставляемых платформой $.NET$: единая система типов, классы, интерфейсы, исключения, делегаты, перегрузка операций, обобщённые типы ($generics$), методы расширения, лямбда-выражения.

Стандартный модуль $PABCSystem$, автоматически подключаемый к любой программе, содержит огромное количество стандартных типов и подпрограмм, позволяющих писать ясные и компактные программы.

В распоряжении $PascalABC.NET$ находятся все средства $.NET$-библиотек классов, постоянно расширяющихся самыми современными возможностями. Это позволяет легко писать на $PascalABC.NET$ приложения для работы с сетью, Web, XML-документами, использовать регулярные выражения и многое другое.

Язык $PascalABC.NET$ позволяет программировать в классическом процедурном стиле, в объектно-ориентированном стиле и содержит множество элементов для программирования в функциональном стиле. Выбор стиля или комбинации этих стилей - дело вкуса программиста, а при использовании в обучении - методический подход преподавателя.

Сочетание богатых и современных языковых средств, возможностей выбора разных траекторий обучения позволяет рекомендовать $PascalABC.NET$, с одной стороны, как язык для обучения программированию (от школьников до студентов младших и средних курсов), с другой - как язык для создания проектов и библиотек средней сложности.

1.2.2.Visual Basic for Application

Несмотря на новизну языка $Visual\ Basic\ for\ Applications$, история его появления почти так же стара, как и вся компьютерная промышленность. Про язык VBA можно сказать, что он является диалектом языка BASIC, который появился в начале 60-х.

Хотя по современным понятиям язык $BASIC$ был довольно ограниченным и, как теперь говорят, варварским, он был прост для изучения и очень скоро получил широкое распространение. Версии языка $BASIC$ выпускались для всех типов компьютеров. Язык $GWBASIC$ производства компании $Microsoft$ был одним из первых языков программирования для современных персональных компьютеров. Он поставлялся со всеми операционными системами $MS\ DOS$ до 5-й версии. Ранние персональные компьютеры производства компании IBM даже имели версию $BASIC$, встроенную в ПЗУ.

С годами первоначальная версия $BASIC$ была существенно доработана. Менялась технология программирования, и вместе с ней под влиянием разработчиков программного обеспечения менялся сам $BASIC$. Современный его диалект включает многие черты и свойства, характерные для более поздних и совершенных языков, таких как $Pascal$, C и $C++$.

В конце 80-х $Microsoft$ выпускает существенно улучшенную версию $BASIC$, названную $QuickBASIC$ во всех версиях $MS\ DOS$, начиная с 6-й (но не $Windows\ 95$).

После нескольких версий $QuickBasic$ в 1992 году $Microsoft$ выпускает $Visual\ Basic\ for\ Windows$. Язык VBA в основном совпадает с $Visual\ Basic\ for\ Windows$, но имеет и существенное отличие. В частности, макросы VBA хранятся в файле документа того приложения, в котором вы создаёте этот макрос.

Внедрив один язык макросов во все свои приложения, $Microsoft$ гарантирует, что большая часть того, что вы выучите о VBA применительно к одному приложению, будет справедлива и для остальных. VBA – мощный язык программирования, инструмент, позволяющий добраться до самых скрытых ресурсов $MS\ Office$.

$Visual\ Basic\ for\ Application$ – это система программирования, которая используется как единое средство программирования во всех приложениях $Microsoft\ Office$.

Программирование в $MS\ Office$ – это прежде всего уменьшение количества повторяющихся действий и ручной работы, которая для этого требуется. Вот примеры некоторых типичных ситуаций:

- вам с определённой периодичностью приходится изготавливать документы, очень похожие друг на друга: приказы, распоряжения в бухгалтерию, договоры, отчёты и т.п. Часто информацию можно взять из базы данных – тогда применение программирования может дать очень большой выигрыш во времени. Иногда её приходится вводить вручную, но и тогда автоматизация даст выигрыш и во времени, и в снижении количества ошибок;
- разновидность той же ситуации: одни и те же данные нужно использовать несколько раз. Например, вы заключаете договор с заказчиком. Одни и те же данные (наименование, адрес, расчётный счёт, номер договора, дата заключения, сумма и т.п.) могут потребоваться во множестве документов: самом договоре, счёте, счёте-фактуре, акте сдачи выполненных работ и так далее. Логично один раз ввести эти данные (скорее всего, в базу данных), а затем автоматически формировать (например, в редакторе $Word$) требуемые документы;
- когда нужно сделать так, чтобы вводимые пользователем данные автоматически проверялись. Вероятность ошибки при ручном вводе данных зависит от многих разных факторов, но, согласно результатам некоторых исследований, она в среднем составляет около двух процентов. "Вылавливать" потом такие ошибки в уже введённых данных – очень тяжёлый труд, поэтому лучше сразу сделать так, чтобы они не возникали.

В общем любое действие, которое вам приходится повторять больше нескольких раз, – это возможный кандидат на автоматизацию. Например, занесение сотен контактов в $MS\ Outlook$, или замена ресурса в десятках проектов $MS\ Project$, или анализ информации из базы данных за разные периоды в таблице $MS\ Excel$ – те ситуации,

когда знание объектных моделей приложений $MS\ Office$ спасёт вас от часов и дней скучного труда.

Конечно, есть ещё практиканты и аналогичный бесплатный трудовой ресурс. Но хочется ли вам потом заниматься ещё и поиском ошибок за них? Кроме того, применение программирования несёт ещё и другие преимущества для сотрудника, который применяет его в работе:

- повышается его авторитет в глазах руководства и других сотрудников;
- если программы этого пользователя активно используются на предприятии (им самим или другими сотрудниками), то этим самым он защищает себя от сокращений, снижения зарплаты и т.п. – ведь поддерживать их и изменять в случае необходимости будет некому.

В принципе, как это не удивительно, при программировании в $MS\ Office$ можно вполне обойтись без языка VBA. Подойдет любой COM-совместимый язык, например, обычный $Visual\ Basic$, $VBScript$, $JScript$, $C++$, $Delphi$, $Java$ и т.п. Можно использовать и $.NET$- совместимые языки программирования – $VB.NET$, C и т.п. Все возможности объектных моделей приложений $MS\ Office$ вполне можно будет использовать. Например, если сохранить код, приведённый на рис. 2, в файле с расширением $*.vbs$ и запустить его на выполнение, то будет запущен $MS\ Word$, в котором будет открыт новый документ и впечатан текст: "Привет от $VBScript$" (рис. 3).

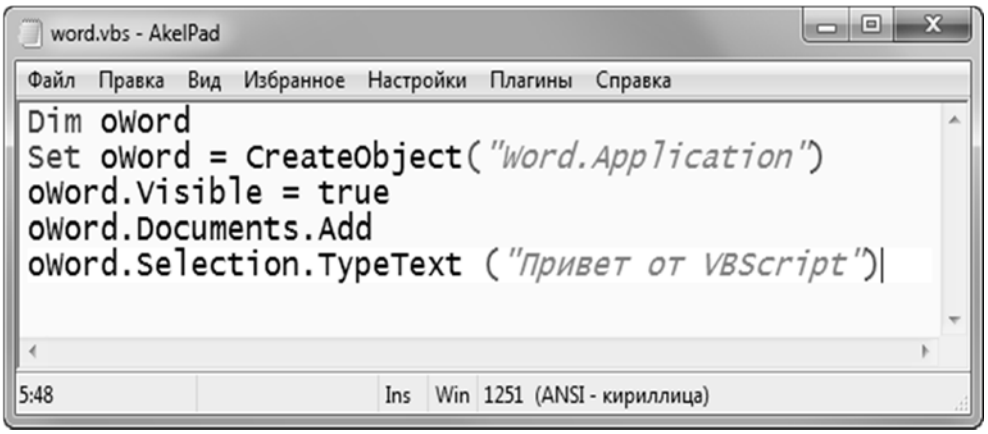


Рис. 2. Пример кода

Однако обычно VBA – самый удобный язык для работы с приложениями $MS\ Office$.

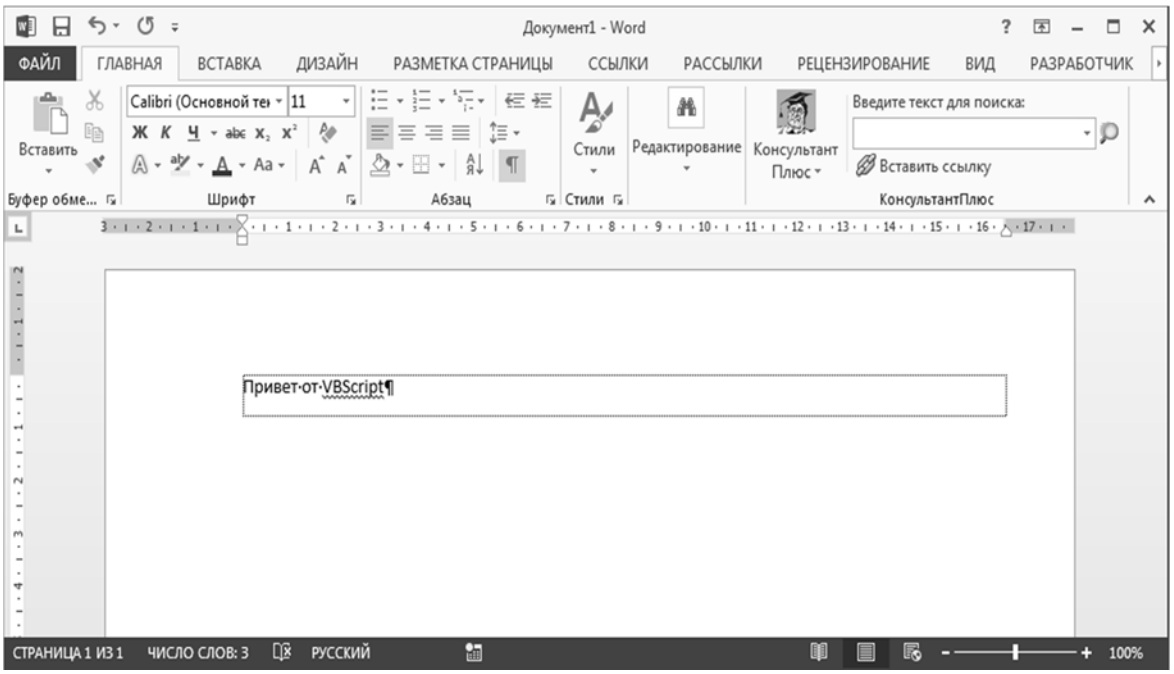


Рис. 3. MS Word 2013, запущенный из скрипта $\boldsymbol{word.vbs}$

Главная причина проста – язык VBA встроен в приложения $MS\ Office$ (и не только). Код на языке VBA можно хранить внутри документов приложений $MS\ Office$ – документах $Word$, книгах $Excel$, презентациях $PowerPoint$ и т.п. Конечно же, этот код можно запускать оттуда на выполнение, поскольку среда выполнения кода VBA (на программистском сленге – хост) встроена внутрь этих приложений.

В настоящее время VBA встроен:

- во все главные приложения $MS\ Office$ – $Word$, $Excel$, $Access$, $PowerPoint$, $Outlook$, $FrontPage$, $InfoPath$;
- в другие приложения $Microsoft$, например, $Visio$ и $Microsoft\ Project$;
- в более чем 100 приложений третьих фирм, например, $CorelDraw$ и $CorelWordPerfect\ Office$ 2000, $AutoCAD$ и т.п.

У VBA есть также множество других преимуществ.

VBA – универсальный язык. Освоив его, вы не только получите

ключ ко всем возможностям приложений $MS\ Office$ и других, перечисленных выше, но и будете готовы к тому, чтобы:

- создавать полноценные приложения на $Visual\ Basic$ (поскольку эти языки – близкие родственники);
- использовать все возможности языка $VBScript$ (это – вообще урезанный VBA). В результате в вашем распоряжении будут универсальные средства для создания скриптов администрирования $Windows$, для создания Web-страниц ($VBScript$ в $Internet\ Explorer$), для создания Web-приложений ASP, для применения в пакетах DTS и заданиях на $MS\ SQL\ Server$, для создания серверных скриптов $Exchange\ Server$ и многое-многое другое.

VBA изначально был ориентирован на пользователей, а не на профессиональных программистов (хотя профессионалы пользуются им очень активно), поэтому создавать программы на нем можно очень быстро и легко. Кроме того, в $MS\ Office$ встроены мощные средства, облегчающие работу пользователя: подсказки по объектам и по синтаксису, макрорекордер и т.п.

При создании приложений на VBA вам скорее всего не придётся заботиться об установке и настройке специальной среды программирования и наличии нужных библиотек на компьютере пользователя – $MS\ Office$ есть практически на любом компьютере.

Несмотря на то, что часто приложения VBA выполняются медленнее, чем бы вам хотелось, они не ресурсоёмки и очень хорошо работают, например, на сервере терминалов. Но, как правило, для программ на VBA особых требований по производительности нет, так как для написания игр, драйверов, серверных продуктов они не используются. Чаще всего проблемы с производительностью VBA-приложений – это не проблемы VBA, а проблемы баз данных, к которым они обращаются. Если проблемы действительно в VBA (обычно тогда, когда вам требуется сложная математика), то всегда есть возможность написать важный код на $C++$ и обращаться к нему как к обычной библиотеке DLL или встраиваемому приложению ($Add-In$) для $Word$, $Excel$, $Access$ и т.п.

Программы на VBA по умолчанию не компилируются, поэтому вносить в них исправления очень удобно. Не нужно разыскивать исходные коды и перекомпилировать программы.

В среде программистов-профессионалов считается, что самый короткий путь «с нуля» (от программ типа «$Hello, World$» до профессиональных программ, которые делаются под заказ) – именно через связку $MS\ Office\ -\ VBA$, а не через $C++$, $Java$ или $Delphi$.

1.3. Технология создания программ

Современное программирование фактически является промышленной технологией, с помощью которой создаются программы и компьютерные системы. Труд этот чаще всего коллективный, хотя не исключены некоторые успехи индивидуальных разработчиков. Например, $MS\ Windows$ и $MS\ Office$ – это результат работы нескольких сотен профессиональных программистов, аналитиков, проектировщиков, менеджеров и дизайнеров фирмы $Microsoft$ (США), в то время как архиватор $WinRAR$ реализовал индивидуальный разработчик Евгений Рошал (Россия).

Схематически технология создания программ проходит следующий ряд этапов:

а) постановка задачи, в рамках которой в общем виде описываются предполагаемые возможности программы;

б) проектирование программы - разработка структуры, интерфейса и детализированных возможностей будущей программы (вплоть до описания конкретных алгоритмов) [3]; в процессе проектирования реализуется разбиение программы на независимые подпрограммы т. е. на функциональные фрагменты, представляющие собой некоторые макрокоманды типа «отредактировать файл», «отправить файл по электронной почте», «найти фрагмент» и т. п.;

в) программирование – задания на разработку подпрограмм выдаются программистам, которые реализуют описанные алгоритмы на языке программирования;

г) отладка и тестирование программ – это обязательный этап, позволяющий выяснить, делает ли программа (подпрограмма) то, на

что она рассчитана, и насколько надёжно она это делает; следует отметить, что важным этапом тестирования является проверка того, как данная подпрограмма работает совместно с другими подпрограммами, входящими в один проект.

В заключение следует отметить, что каждая программа имеет свой жизненный цикл, который тем больше, чем больше у программы возможностей по настройке и чем легче она модифицируется. Именно поэтому сейчас становится все более популярной технология, называемая $OpenSource$, предполагающая предоставление пользователям программы вместе с её кодом на языке программирования высокого уровня.

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

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

1.4. Системы и среды программирования

Изначально инструментарий программистов включал ряд средств разработки, в которые, помимо языка программирования, входили:

- специализированные или обычные текстовые редакторы, с помощью которых писались тексты программ;
- трансляторы, которые проверяли правильность соблюдения синтаксиса языка программирования и, если синтаксис не нарушен, преобразовывали текст программы в машинный код;
- специальные отладчики, которые позволяли, например, покомандно выполнять программу и смотреть получаемые результаты.

Сегодня подобными средствами никто не пользуется, поскольку разработаны более удобные интегрированные инструментальные среды, обеспечивающие выполнение полного комплекса взаимосвязанных работ по созданию программ. Фактически эти среды включают в себя перечисленные выше компоненты. Более того, среды постоянно совершенствуются и все более автоматизируют процесс создания программ.

Можно отметить три поколения подобных сред. К первому поколению относятся Турбо-среды, в которых фактически интегрированы специализированный текстовый редактор, транслятор и отладчик. Повышение производительности обеспечивалось тем, что в рамках одной среды можно было заниматься подготовкой, трансляцией и отладкой программ. К подобным средам можно отнести среды $Borland\ Pascal\ 7.0$ [1], $Turbo\ C$ и др.

Второе поколение – это визуальные среды программирования. Такие среды, помимо того, что обладают всеми возможностями Турбо-сред, предоставляют разработчику огромное количество готовых фрагментов программ. Эти фрагменты сгруппированы в различные подменю в виде отдельных пиктограмм (иконок), и их включение в программу разработчика реализуется перетаскиванием этой пиктограммы в нужное окно с помощью мыши. Подобный подход позволил существенно увеличить скорость разработки программ, имеющих стандартные интерфейсы (кнопки, окна, обработчики событий и т. п.). К данному виду программного обеспечения можно отнести $Delphi$ (язык программирования $Object\ Pascal$), $Borland\ C++Builder$, $JBuilder$ (язык $Java$) и др.

И, наконец, $CASE$-среды программирования, представляющие собой просто конструкторы программ, в рамках которых либо вообще не надо программировать (если не считать рисования каких-либо схем взаимодействия готовых компонентов), либо программировать лишь вид окон, обеспечивающих интерфейс готовой системы. Подобные среды обладают максимальной на сегодняшний день автоматизацией проектирования и реализации программ и позволяют наиболее быстро создавать различные специализированные информационные системы. Среди подобных средств можно

назвать $Vantage\ Team\ Builder$ ($Westmount\ I-CASE$), ($Designer$)/ 2000 , $Erwin\ +\ BPwin$ и $CASE$.Аналитик.

Контрольные вопросы к главе 1

1. Что такое «программа»?
2. В программе на машинном языке (см. табл. 1) выполняются операции с двумя операндами. А куда записывается результат?
3. Чем отличается машинный язык от языка высокого уровня?
4. Предположим, что в программе, описанной в табл. 1, потребуется найти среднее арифметическое не четырёх, а пяти чисел, причём пятое число будет храниться в блоке памяти по адресу 01010101. Как изменится текст программы?
5. В чем назначение транслятора? Чем отличается компилятор от интерпретатора?
6. Что такое язык программирования? Синтаксис языка программирования?
7. Какие этапы включает в себя технология создания программ?
8. В чем особенность Турбо-сред?
9. Чем визуальные среды программирования отличаются от Турбо-сред?
10. Каковы особенности работы в $CASE$-средах?
11. Кто разработал архиватор *WinRAR*?
12. Какой путь в изучении языков программирования в среде профессиональных программистов считается наиболее приемлемым?
13. Для чего служит модуль $PABCSystem$?
14. Какие компоненты необходимы для создания полноценной выполняемой программы на компьютере?
15. Каковы этапы технологии создания компьютерной программы?

Глава 2. СРЕДА РАЗРАБОТКИ

2.1. Среда разработки *PascalABC.NET*

Интегрированная среда разработки *PascalABC.NET* ориентирована на создание проектов малой и средней сложности. Она достаточно легковесна и в то же время обеспечивает разработчика всеми необходимыми средствами, такими как встроенный отладчик, средства *Intellisense* (подсказка по точке, подсказка по параметрам, всплывающая подсказка по имени), переход к определению и реализации подпрограммы, шаблоны кода, автоформатирование кода.

В среду *PascalABC.NET* встроен также дизайнер форм, позволяющий создавать полноценные оконные приложения в стиле *RAD* (*Rapid Application Development* - быстрое создание приложений).

В отличие от многих профессиональных сред, среда разработки *PascalABC.NET* не имеет громоздкого интерфейса и не создаёт множество дополнительных вспомогательных файлов на диске при компиляции программы. Для небольших программ это позволяет соблюсти принцип - «Одна программа - один файл на диске».

В среде *PascalABC.NET* большое внимание уделено связи запущенной программы с оболочкой: консольная программа, запущенная из-под оболочки, осуществляет ввод-вывод в специальное окно, встроенное в оболочку. Можно также запустить несколько программ одновременно - все они будут контролироваться оболочкой.

Интегрированная среда *PascalABC.NET* позволяет переключать в настройках русский и английский язык, при этом локализованы не только элементы интерфейса, но и сообщения об ошибках.

Кроме этого, внутренние представления *PascalABC.NET* позволяют создавать компиляторы других языков программирования и встраивать их в среду разработки с помощью специальных плагинов.

Скачать установочный файл *PascalABCNETWithDotNetSetup.exe* можно с сайта ***PascalABC.NET*. Современное программирование на языке *Pascal*** по адресу: http://pascalabc.net.

После установки среды разработки для начала написания и запуска программ на Паскаль необходимо сделать ещё пару подготовительных шагов.

Шаг 1. Подготовка рабочей папки для хранения файлов с разработанными программами. Определитесь с расположением рабочей папки (спросить у администратора или преподавателя, по умолчанию имя такой папки может быть $Student$). Если таковой нет, создайте её.

В папке $Student$ создайте папку $Pascal$, а в ней – папку с вашей фамилией.

Далее эту папку будем называть «Вашей папкой».

В этой папке создайте следующие папки:

$Prog$ – для хранения исходных кодов разрабатываемых программ;

$Input$ – для файлов с исходными данными;

$Output$ – для откомпилированных файлов разрабатываемых программ.

Шаг 2. Инсталлировать установочный файл среды разработки $PascalABC.NET$. При установке будет задан вопрос о пути к папке с исходными кодами, указать путь к папке $Prog$.

По окончании установки будет сгенерирована папка $PascalABC.NET$ и на рабочем столе появится иконка с тем же именем.

Шаг 3. После окончания установки (*шаг 2*) запустите среду разработки (нажать на иконку $PascalABC.NET$ или **Пуск** → **Программы** → **PascalABC.NET** → **PascalABC.NET**) (рис. 4).

Введём несколько определений, которые будем использовать в данном пособии.

Консольные приложения – это программы, в которых диалог организуется с помощью клавиатуры в специальном консольном окне (на рис. 4 это окно названо просто консолью). Окно может быть единым для операций ввода и вывода или делится на две части. В нашем случае реализован последний вариант.

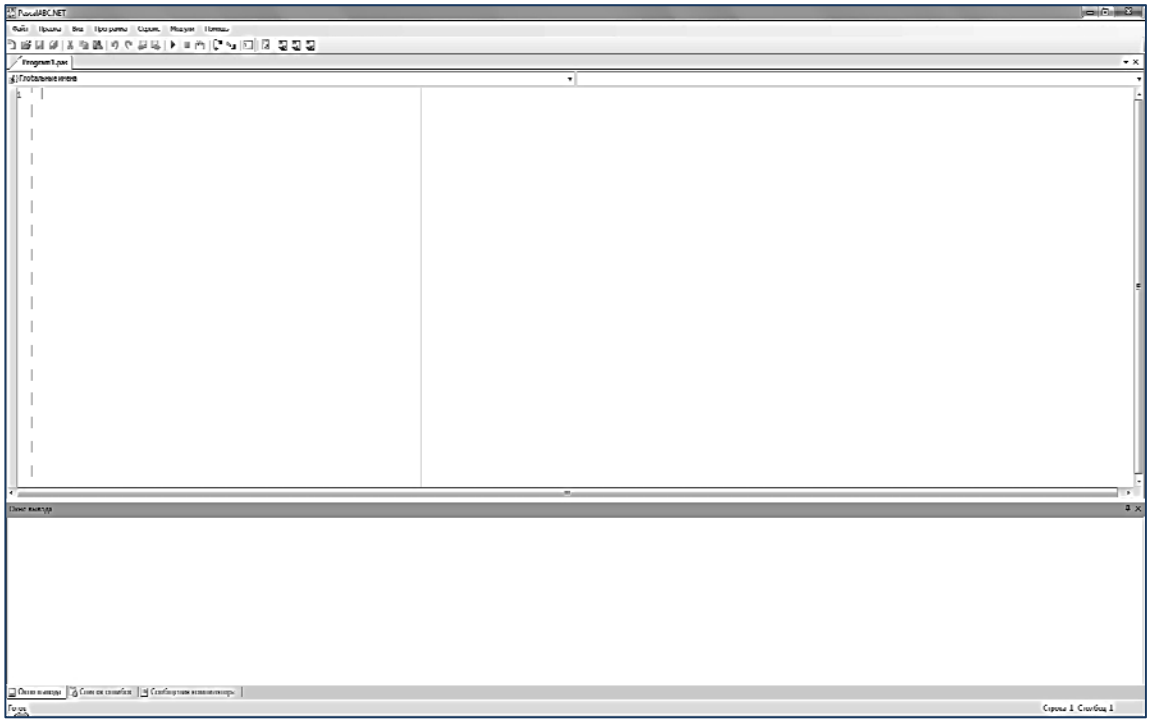


Рис. 4. Основное окно среды разработки PascalABC.NET

Графические приложения – это программы, выполняемые в отдельных окнах, в которых строятся различные изображения. в том числе и диалоговые окна, в которых диалог реализуется как с помощью мыши, так и с помощью клавиатуры.

Лист, на котором размещается программа, будем называть страницей.

Большую часть рабочей области, её верхнюю часть (рис. 5), занимает окно редактора кода. В него вводится исходный текст программы.

Горячие клавиши, которые можно использовать при работе с текстом программы:

$Ctrl + S$- сохранить файл;

$Ctrl + O$- загрузить файл;

$F12$- сохранить файл под новым именем;

$Ctrl + Shift + S$- сохранить все открытые файлы;

$Ctrl + Tab,\ Ctrl + Shift + Tab$ - перейти к следующему / предыдущему окну редактора;

$Ctrl + Shift + I$- увеличить отступ выделенного блока;

$Ctrl + Shift + U$ - уменьшить отступ выделенного блока.

Под окном редактора расположено окно вывода. Оно предназначено для вывода данных процедурами $write$ и $writeln$, а также для вывода сообщений об ошибках и предупреждений во время работы программы.

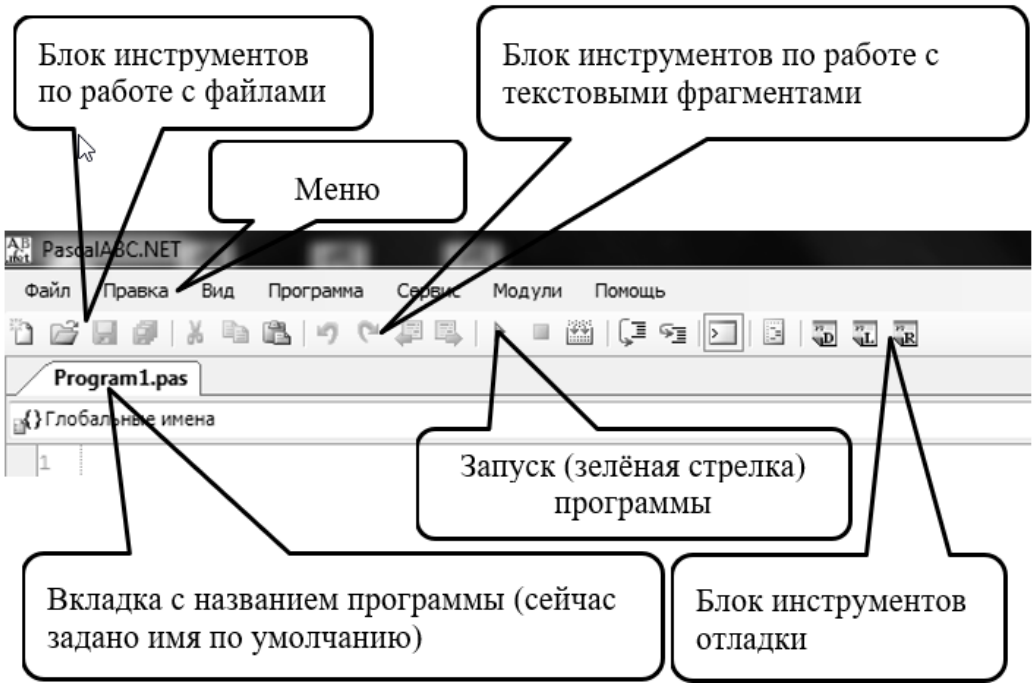


Рис. 5. Основные блоки среды разработки PascalABC.NET

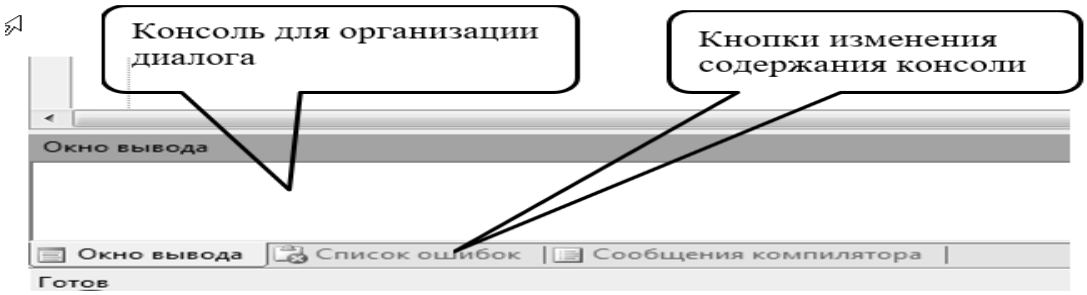


Рис. 6. Консольная часть

Окно вывода может быть скрыто (рис. 6). Клавиша $\boldsymbol{F5}$ показывает/скрывает окно вывода. Для скрытия окна вывода используется также клавиша $\boldsymbol{Esc}$.

Окно вывода обязательно открывается при любом выводе в него.

Для очистки окна вывода следует нажать комбинацию клавиш $\boldsymbol{Ctrl + Del}$.

Окно ввода открывается при выполнении процедур $read$ и $readln$ в ходе работы программы.

Ввод данных в окно ввода сопровождается эхо-выводом в окно вывода (рис. 7). После нажатия клавиши $\boldsymbol{Enter}$ данные из окна ввода попадают в соответствующие переменные, окно ввода закрывается, и программа продолжает работать дальше.

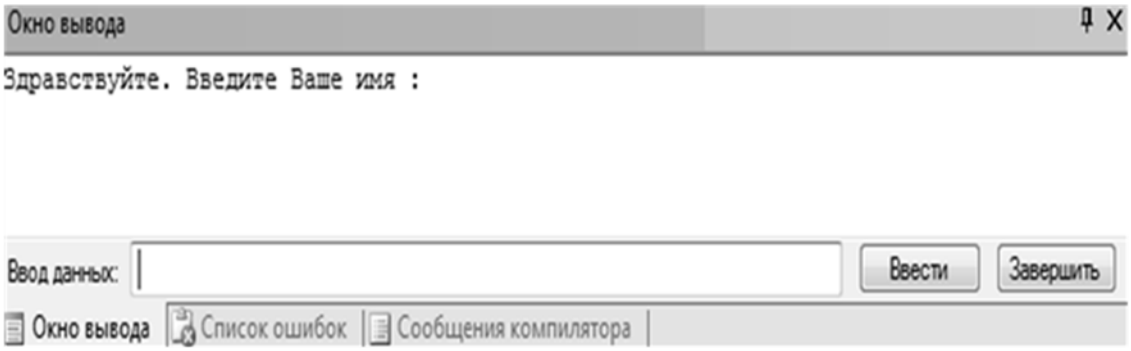


Рис. 7. Окно вывода и ввода

Среда разработки $\boldsymbol{PascalABC.NET}$ проста в использовании. Дополнительные сведения о ней рассмотрим в процессе разработки программ.

2.2 Применение макрорекордера

2.2.1.Макрорекордер в $Word$ и $Excel$: запуск, приемы работы, возможности

В большинстве программ $Microsoft\ Office$, таких как $Excel$, $Word$, $PowerPoint$ и т.п., встроено замечательное средство, которое позволяет создавать программы, вообще ничего не зная о программировании. Это средство называется ***макрорекордером***.

Макрорекордер, как понятно из его названия, – средство для записи макросов. Макрос – всего лишь ещё одно название для VBA-программы, а макрорекордер – средство для его автоматического создания.

Принцип работы макрорекордера больше всего похож на принцип работы магнитофона: мы нажимаем на кнопку – начинается запись тех действий, которые мы выполняем. Мы нажимаем на вторую кнопку – запись останавливается, и мы можем её проиграть (то есть повторно выполнить ту же последовательность действий).

Конечно, макрорекордер позволяет написать только самые простые VBA-программы. Однако и он может принести много пользы. Например, можно «положить» на горячие клавиши те слова, словосочетания, варианты оформления и т.п., которые вам часто приходится вводить (должность, название фирмы, продукт, ФИО директора и ответственного исполнителя и т.п.) – этим вы сэкономите много времени.

Как показывает опыт, подавляющее большинство обычных пользователей и не подозревает о существовании макрорекордера несмотря на то, что его применение позволило бы сэкономить им много времени.

Перед созданием макроса в макрорекордере:

- тщательно спланируйте макрос, хорошо продумав, что необходимо делать и в какой последовательности. Если есть возможность, продумайте подготовительные действия. Например, если нужно вставлять текущую дату в начало документа, то, может быть, есть смысл первой командой макроса сделать переход на начало документа ($\boldsymbol{Ctrl} + \boldsymbol{Home}$);
- посмотрите, нет ли готовой команды, которую можно сразу назначить клавише или кнопке в панели инструментов без изготовления макроса. Посмотреть можно при помощи меню **Файл** → **Параметры** → **Настроить ленту**;
- если собираетесь при помощи макроса менять оформление текста, то правильнее вначале создать новый стиль с вашим оформлением, а потом уже применять этот стиль к тексту. В этом случае опять-таки можно обойтись без макроса, просто назначив стиль комбинации клавиш.

Создать макрос в макрорекордере можно в тех программах $MS\ Office$, для которых такое средство предусмотрено, например, $Excel$, $PowerPoint$, $Project$).

На вкладке **Разработчик**[1] выберите команду **Запись макроса**. В открывшемся окне потребуется определить:

- *Имя макроса.* Оно не должно начинаться с цифры, содержать пробелов и символов пунктуации. Максимальная длина имени макроса в $Excel$ – 64 символа, в *Word* – 80 символов. Можно использовать русский язык.
- *Будет ли макрос назначен кнопке на панели управления или комбинации клавиш.* Назначить макрос кнопке/комбинации клавиш или использовать другие средства для его вызова можно и потом – об этом будет рассказано в следующем разделе.
- *Где сохранить макрос*. В $Word$ в вашем распоряжении текущий файл и шаблон для всех вновь создаваемых документов – $normal.dot$. В $Excel$ в вашем распоряжении текущая книга, возможность создать макрос одновременно с созданием новой книги и личная книга макросов $personal.xls$ (макросы из этой скрытой книги будут доступны в любых книгах). Подробнее про то, где может храниться программный код, мы поговорим в разделе про структуру проектов VBA (с. 80).
- *Описание.* Лучше заполнить – это подарок не только для других, но и для себя (через несколько месяцев).

После нажатия на кнопку OK или назначения кнопки/клавиатурной комбинации начнётся запись макроса. Указатель мыши при этом примет вид магнитофонной кассеты и появится маленькая панель ***Остановить запись***. На ней всего две кнопки – ***Остановить запись*** и ***Пауза***. Если вы случайно закрыли эту панель, то остановить запись можно через меню ***Разработчик*** → ***Остановить запись***.

Самый простой способ запустить макрос, которому не назначена кнопка или клавиатурная комбинация, – на вкладке ***Разработчик*** выбрать ***Макросы*** (или нажать кнопку $Alt + F8$), в списке выбрать нужный макрос и нажать на кнопку ***Выполнить***. Из этого же окна можно просматривать/редактировать макросы, удалять/перемещать их и т.п.

---

[1] *Активацию вкладки Разработчик см. в разделе 2.3.*

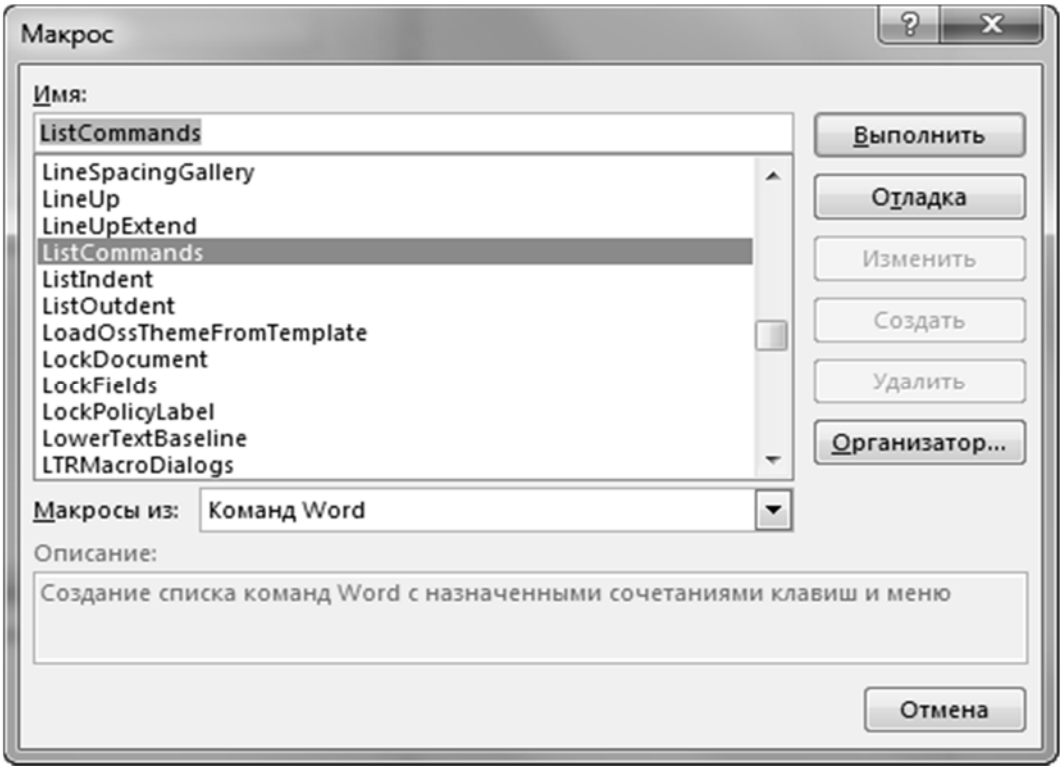


Рис. 8. Окно Word 2013 Макросы

Если макросов создано много, то получить список всех назначений клавиш (включая назначения для встроенных макросов $Word$) можно при помощи меню ***Разработчик*** → ***Макросы***, затем в списке ***Макросы из*** выбрать ***Команды Word*** и выбрать в списке ***Имя*** команду $List\ Commands$. Нажать ***Выполнить*** (рис. 8). В ответ на приглашение нужно выбрать ***Текущие настройки клавиатуры*** (иначе будет выведен полный список команд $Word$ на 26 страниц). В ваш документ будет вставлена таблица с текущими назначениями клавиш, которую можно распечатать.

Если у вас уже есть значительное количество созданных при помощи макрорекордера макросов, то после освоения языка VBA есть смысл подумать над ними и, может быть, внести изменения. Чаще всего есть смысл подумать над следующими моментами:

- если в вашем макросе повторяются какие-либо действия, то возможно, есть смысл организовать цикл;
- может быть, есть смысл в ходе выполнения уточнить что-либо у пользователя (при помощи $Input\ Box$ или элементов управления);

- чтобы в ходе выполнения макроса не возникало ошибок, то, возможно, есть смысл реализовать в нем проверку текущих условий.

Как все это сделать, будет рассказано в следующих главах.

И ещё один очень важный момент, связанный с макрорекордером. Помимо того, что он позволяет создавать простенькие программы, пригодные для самостоятельного использования безо всяких доработок, макрорекордер – это ещё и ваш разведчик в мире объектных моделей приложений $MS\ Office$. Опытные разработчики часто пользуются им для того, чтобы понять, какие объекты из огромных объектных моделей приложений $MS\ Office$ можно использовать для выполнения тех или иных действий.

Конкретный пример: вам нужно автоматизировать создание диаграмм в $Excel$. Поскольку в русской версии $Excel$ для создания диаграммы вручную вы используете команду ***Вставка*** → ***Диаграмма***, то скорее всего, в справке по VBA вы начнете в первую очередь искать объект $Diagram$. И вы его найдете – и скорее всего, потратите определённое время на его изучение, прежде чем поймёте, что это не та диаграмма! Объект $Diagram$ представляет то, что в русской версии $Excel$ называется «Схематическая диаграмма» (доступны из того же меню ***Вставка***), а обычная диаграмма – это объект $Chart$. А вот если бы мы пустили вперед разведчика (то есть создали бы диаграмму с записью в макрорекордере и посмотрели бы созданный код), он бы сразу указал нам нужное направление движения.

2.2.2. Запуск макроса после создания: диалоговые окна, панели инструментов, меню и командная строка

Запустить на исполнение созданный макрос можно несколькими способами. Самый простой, но и самый неудобный способ – воспользоваться окном ***Макрос***, которое можно открыть при помощи меню ***Разработчик*** → ***Макросы*** (рис. 8).

Клавиши в этом окне в разных приложениях могут различаться, однако основные клавиши остаются постоянными:

Выполнить – запустить макрос на выполнение;

Войти – открыть макрос в редакторе $Visual\ Basic$ и начать его пошаговое выполнение;

Изменить – просто открыть макрос в редакторе $Visual\ Basic$;

Создать – необходимо будет ввести имя создаваемого макроса и в редакторе $\boldsymbol{Visual\ Basic}$ будет автоматически создана процедура с определенным вами именем;

Удалить;

Параметры – поменять описание и назначенное сочетание клавиш.

Каждый раз открывать это окно, находить нужный макрос (а их вполне может быть, например, несколько десятков) и нажимать на кнопку ***Выполнить*** – не самый быстрый вариант. Вряд ли он очень понравится вашим пользователям, да и вам самим так работать будет неудобно. Поэтому в вашем распоряжении несколько более удобных вариантов.

Если вы пользуетесь макросом постоянно, то можно использовать самый быстрый способ его вызова – клавиатурную комбинацию. Назначить сочетание клавиш макросу можно очень просто.

1. ***Рассмотрим назначение и удаление сочетаний клавиш при помощи клавиатуры Word 2013.***

Если необходимо, нажмите клавиши $\boldsymbol{ALT} + \Ф, \text{М}$ чтобы открыть диалоговое окно **Параметры Word**, и нажмите клавишу **СТРЕЛКА ВНИЗ** для выбора пункта **Настройка ленты**.

Нажимайте клавишу **TAB** до тех пор, пока не будет выбран пункт **Настроить**, затем нажмите клавишу **ВВОД**. Появится окно настройки клавиатуры (рис. 9).

В поле **Категории** нажмите клавишу **СТРЕЛКА ВНИЗ** или **СТРЕЛКА ВВЕРХ**, чтобы выделить категорию, содержащую команду или иной элемент, для которого нужно назначить или удалить сочетание клавиш.

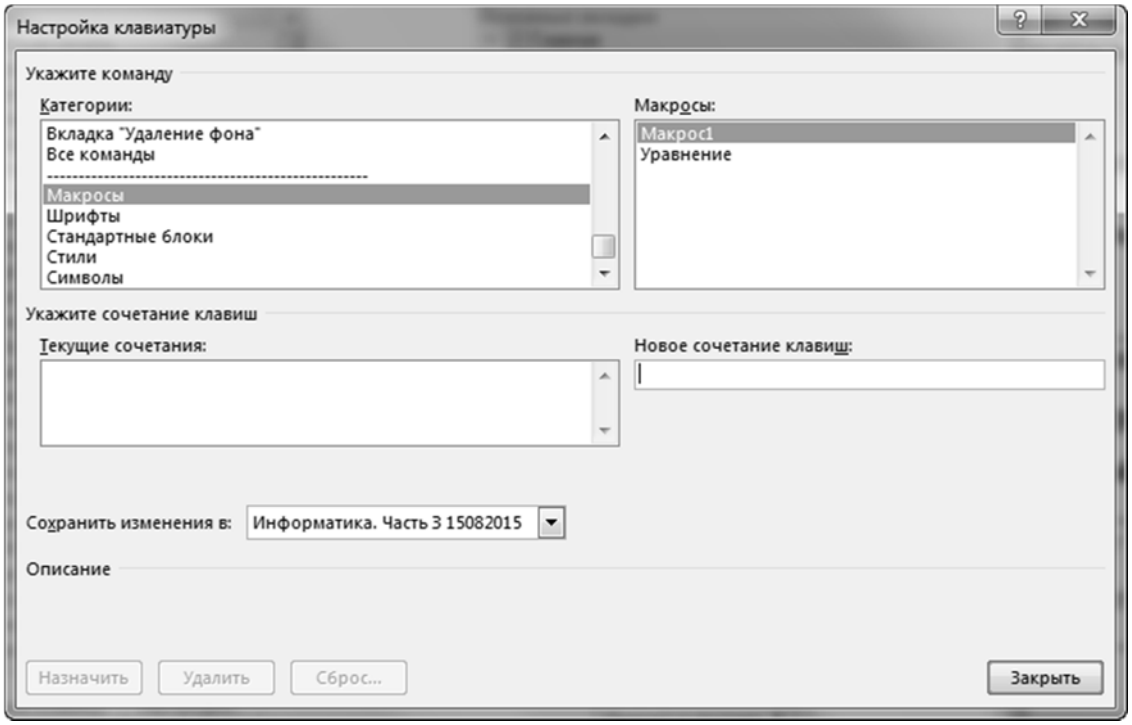


Рис. 9. Окно Настройка клавиатуры

Нажмите клавишу **TAB**, чтобы перейти в поле **Команды**.

Нажмите клавишу **СТРЕЛКА ВНИЗ** или **СТРЕЛКА ВВЕРХ**, чтобы выбрать имя команды или иного элемента, которому нужно назначить или удалить сочетание клавиш.

Все сочетания клавиш, назначенные команде или элементу, отображаются в поле **Текущие сочетания**.

Выполните одно из следующих действий:

а) Назначение сочетаний клавиш

Сочетание клавиш следует начинать с клавиши ***CTRL*** или с функциональной клавиши.

Нажимайте клавишу **TAB** до тех пор, пока курсор не перейдёт в поле **Новое сочетание клавиш**.

Нажмите сочетание клавиш, которое требуется назначить. Например, нажмите клавишу ***CTRL*** и еще какую-либо клавишу.

Если сочетание клавиш уже назначено данной команде или другому элементу, то оно отображается в поле **Текущее назначение**.

Если это сочетание клавиш уже назначено, то введите другое сочетание.

ВАЖНО: Изменение назначения сочетания клавиш делает невозможным его использование по первоначальному назначению. Например, сочетание клавиш $\boldsymbol{CTRL} + \boldsymbol{B}$ используется для оформления выделенного текста полужирным шрифтом. Если назначить сочетание клавиш $\boldsymbol{CTRL} + \boldsymbol{B}$ новой команде или другому элементу, то применение к тексту полужирного стиля при помощи данного сочетания клавиш станет невозможным до тех пор, пока не будет восстановлено исходное назначение этого сочетания клавиш при помощи команды **Сброс**.

Нажимайте клавишу **TAB** до тех пор, пока не будет выбрано поле **Сохранить изменения в**.

Нажмите клавишу **СТРЕЛКА ВНИЗ** или **СТРЕЛКА ВВЕРХ**, чтобы выделить название текущего документа, шаблона или макроса, в котором нужно сохранить изменения сочетания клавиш, затем нажмите клавишу **ВВОД**.

Нажимайте клавишу **TAB** до тех пор, пока не будет выбран пункт **Назначить**, затем нажмите клавишу **ВВОД**.

ПРИМЕЧАНИЕ: Если компьютер оснащён программируемой клавиатурой, то нельзя назначать сочетание клавиш $\boldsymbol{CTRL} + \boldsymbol{ALT} + \boldsymbol{F8}$, поскольку оно зарезервировано для перехода в режим программирования клавиатуры.

б) Удаление сочетаний клавиш

Нажимайте клавишу **TAB** до тех пор, пока не будет выбрано поле **Сохранить изменения в**.

Нажмите клавишу **СТРЕЛКА ВНИЗ** или **СТРЕЛКА ВВЕРХ**, чтобы выделить название текущего документа, шаблона или макроса, в котором нужно сохранить изменения сочетания клавиш, затем нажмите клавишу **ВВОД**.

Нажимайте сочетание клавиш $SHIFT + TAB$ до тех пор, пока курсор не окажется в поле **Текущие сочетания**.

Нажмите клавишу **СТРЕЛКА ВНИЗ** или **СТРЕЛКА ВВЕРХ**, чтобы выбрать сочетание клавиш, которое требуется удалить.

Нажимайте клавишу **TAB** до тех пор, пока не будет выбран пункт **Удалить**, затем нажмите клавишу **ВВОД**.

2. Назначение и удаление сочетаний клавиш при помощи клавиатуры Excel 2013.

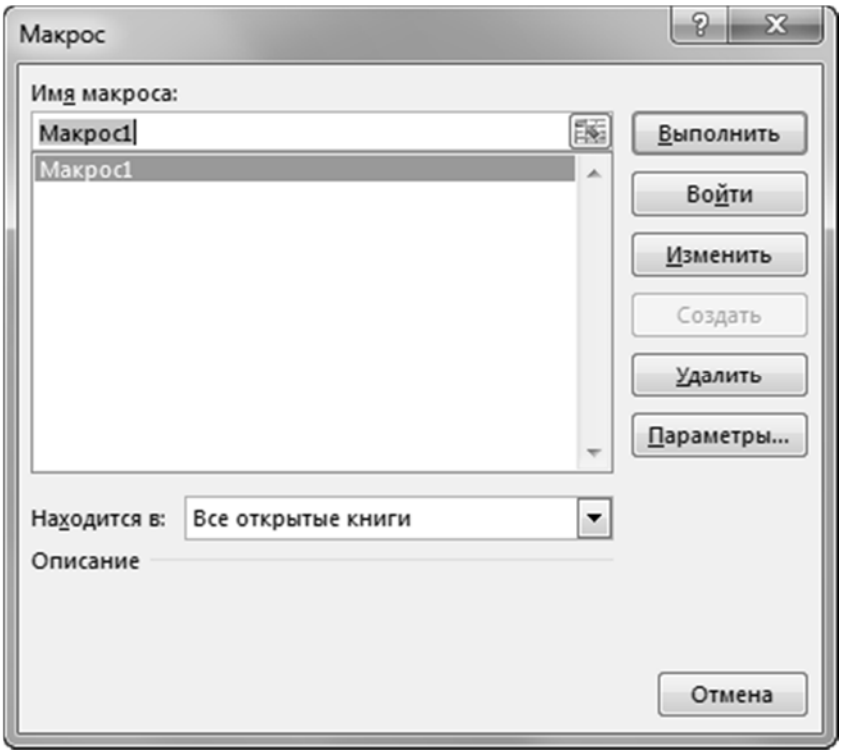


Рис. 10. Окно Макросы Excel 2013

В Excel назначение клавиш можно осуществить следующим образом. Если макрос создаётся только при помощи макрорекордера или вручную после выбора **Разработчик** → **Запись макроса** то появится окно **Запись макроса**, где в поле **Сочетание клавиш** можно ввести необходимое сочетание. Если при назначении клавиш уже существующему макросу после выбора **Разработчик** → **Макросы** появится окно **Макросы**, то необходимо нажать клавишу **Параметры**

(рис. 10) и далее в окне **Параметры макроса** в поле **Сочетание клавиш** ввести необходимую последовательность.

Как уже отмечалось, на клавиатурные комбинации есть смысл назначать только те макросы, которыми вы пользуетесь каждый день. А как же быть с полезными макросами, которые активно используются, к примеру, в отчётный период, а потом опять ждут своего часа целый месяц? Подавляющее большинство пользователей за этот месяц забудет все назначенные клавиши и потеряет те бумажки, на которых им эти клавиатурные комбинации записали. Да и сам пользователь вполне может забыть, что именно нужно нажимать для запуска макроса.

Лучший выход в такой ситуации – назначить макрос пункту меню или кнопке на панели быстрого доступа. Пожалуй, назначение пункту меню даже лучше – больше возможностей упорядочить макросы и есть возможность использовать понятные названия. Однако нажимать на кнопки на панели быстрого доступа быстрее – так что выбирайте сами, что вам больше нравится.

а) Создание кнопки на панели быстрого доступа

Щёлкните панель быстрого доступа правой кнопкой мыши и выберите в контекстном меню команду **Настройка панели быстрого доступа**.

В разделе **Настройка панели быстрого доступа** выберите в списке **Выбрать команды из** пункта **Макросы**.

В списке **"Настройка панели быстрого доступа"** выберите файл с именем текущего редактируемого документа. (Это необходимо, чтобы приложение $Word$ или $Excel$ (либо то в котором сейчас работаете) сохранило кнопку на панели быстрого доступа в файле документа. Если этого не сделать, при копировании файла на другой компьютер кнопка отображаться не будет.)

Выберите созданный макрос и нажмите кнопку **Добавить**.

Нажмите кнопку **Изменить**, чтобы выбрать символ и изменить имя на новое (если нужно).

б) Создание кнопки на ленте

Выбрать вкладку **Файл** → пункт **Параметры** → пункт **Настроить ленту**.

В открывшемся окне справа, в списке **"Настроить ленту"**, выберите вкладку и группу, где будет помещена кнопка, или создайте новые кнопки - **Создать вкладку** и **Создать группу**. Слева в этом же окне в выпадающем списке **"Выбрать команды"** выбрать пункт **"Макросы"**.

Ниже, в списке, выбрать нужный макрос и нажать кнопку **"Добавить >>"**. Чтобы изменить иконку или название кнопки, группы, вкладки, нажмите кнопку **"Переименование..."**.

В подавляющем большинстве остальных приложений $MS\ Office$ ($PowerPoint$, $Project$, $Outlook$ и т.п.) работа с макросами производится так же, как и в $Word$.

Есть ещё один способ предоставить пользователю возможность запускать макросы – самый функциональный, но и самый трудоёмкий: создать специальную графическую форму, на которую можно поместить, например, ниспадающий список макросов. При применении этого способа можно предусмотреть дополнительные элементы управления для ввода параметров, которые макросы смогут «подхватывать» во время выполнения (напрямую параметры макросам передаваться не могут, поскольку макрос – это процедура, не принимающая параметров). Однако применение этого способа потребует написания программного кода. После этого создание такой формы не составит никакого труда.

Есть ещё одна специальная возможность для запуска макросов: сделать так, чтобы они запускались при возникновении специального события. Таким событием может стать, например, внесение изменений на лист $Excel$, открытие книги $Excel$ или документа $Word$ и т.п. Однако можно обеспечить автоматический запуск макроса и без программирования: достаточно просто назначить ему специальное имя. Например, для $Word$ список таких специальных названий представлен в табл. 2.

Таблица 2

Список специальных макросов для Word

Имя процедуры	Когда запускается
AutoExec	При запуске $Word$ (этот макрос должен храниться в $normal.dot$)
AutoNew	При создании нового документа
AutoOpen	При открытии любого документа (если в normal.dot) или открытии документа, в котором находится макрос с таким именем
AutoClose	При закрытии документа
AutoExit	При выходе из $Word$

В $Excel$ предусмотрены специальные имена макросов для рабочей книги $Auto\_Open$, $Auto\_Close$, $Auto\_Activate$ и $Auto\_Deactivate$. Однако $Microsoft$ предупреждает, что эти возможности оставлены только для обратной совместимости и рекомендует пользоваться событийными процедурами.

Ещё один момент, связанный с макросами $Auto$: компания ***Microsoft*** объявила, что макросы такого типа теперь отключены по умолчанию (начиная с версии ***MS Office 2003***), чтобы защитить пользователей от вредоносных документов из соображений безопасности. Для того, чтобы обеспечить им возможность запуска, необходимо изменить установленный уровень безопасности: **Файл** → **Параметры** → **Центр управления безопасностью** справа нажать кнопку **"Параметры центра управления безопасностью…"**. В открывшемся окне Центр управления безопасностью справа выбрать необходимый уровень безопасности (например, **Включить все макросы**)

Ну и последняя, самая малоизвестная, но тем не менее очень полезная возможность для запуска макросов. Вы можете запустить их из командной строки при запуске $Word$ или $Excel$, указав имя макроса в качестве параметра командной строки. Например, чтобы открыть $Word$ и сразу выполнить макрос $MyMacros$ из $normal.dot$, можно воспользоваться командой:

```
winword.exe / mMyMacros
```

Очень удобно использовать эту возможность, если создать несколько ярлыков для запуска приложения $MS\ Office$, например, на

рабочем столе, изменить в них командную строку и использовать для запуска приложения с одновременным запуском макросов.

2.3. Интегрированная среда разработки VBA

Среда разработки VBA называется интегрированной средой разработки или IDE ($Integrated\ Development\ Environment$). $VBA\ IDE$ – это набор инструментов разработки программного обеспечения, таких как редактор $Visual\ Basic$ ($Visual\ Basic\ Editor, VBE$), средства отладки, средства управления проектом и т. д. VBE – это окно, содержащее меню, другие окна и элементы, которые применяются при создании проектов VBA. Все приложения, поддерживающие VBA, работают с одним IDE. Таким образом, при переходе в другое основное приложение не требуется много времени, чтобы научиться применять в нем VBA (в представленном пособии рассматривается интегрированная среда разработки VBA для $MS\ Excel\ 2013$).

Для вывода IDE необходимо на ленте выбрать вкладку **Разработчик** и затем нажать клавишу $Visual\ Basic$ (рис. 11).

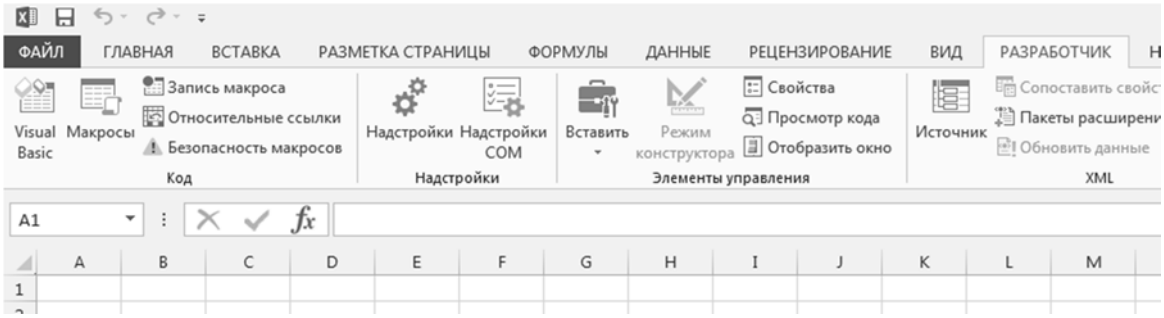


Рис. 11. Запуск VBA IDE в MS Excel 2013

В том случае, если на ленте нет вкладки **Разработчик** необходимо в окне Параметры $Excel$ (Файл-Параметры-Настроить ленту) в правой части (Основные вкладки) отметить строку ***Разработчик*** (рис. 12).

Для перехода из окна основного приложения в редактор VBE достаточно нажать комбинацию клавиш $Alt + F11$.

В русских версиях приложений $MS\ Office$ для редактора

$Visual\ Basic$ предусмотрен англоязычный интерфейс. Справка по языку VBA и объектным моделям приложений $MS\ Office$ – тоже только на английском. К сожалению, русифицированных вариантов не существует. Однако знание английского языка для того, чтобы писать программы в VBA, не обязательно (хотя и очень полезно): программы вполне можно создавать, не зная английского.

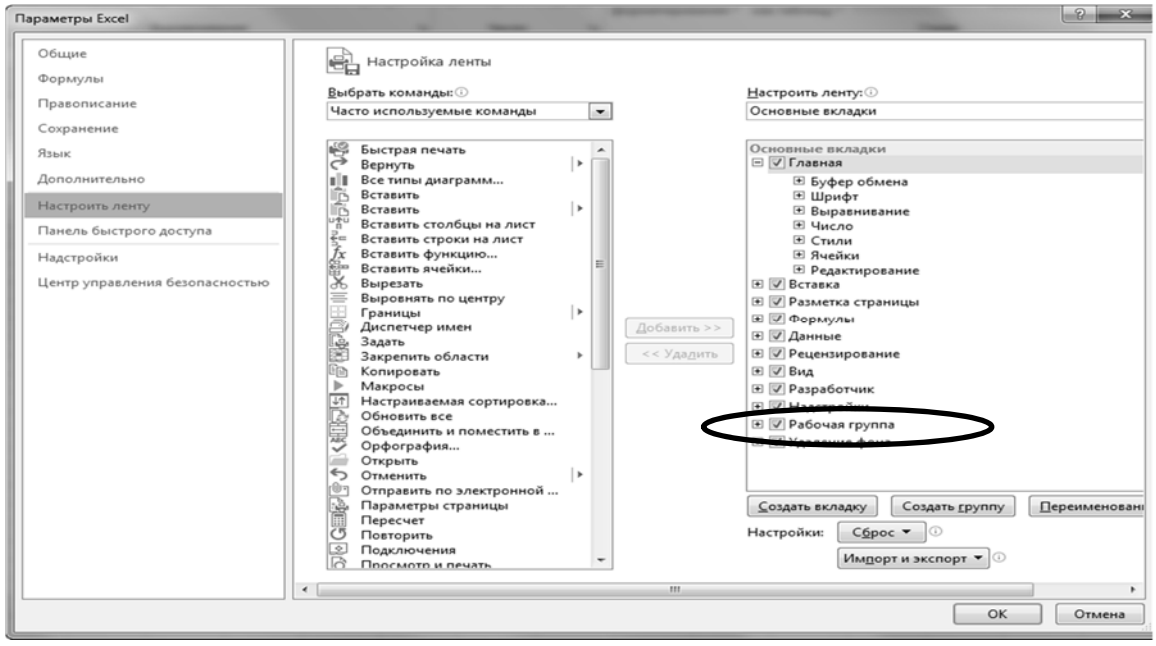


Рис. 12. Окно Параметры Excel

Интерфейс VBE составляют различные окна, панели инструментов и меню. Основными (открывающимися по умолчанию) являются три окна: окно проекта, окно свойств и окно редактирования кода.

Назначение этих и некоторых других компонентов VBE приведено в табл 3. Вызвать на экран тот или иной компонент можно с помощью меню **View** (**Вид**). Вид окна редактора $Visual\ Basic$ представлен на рис. 13.

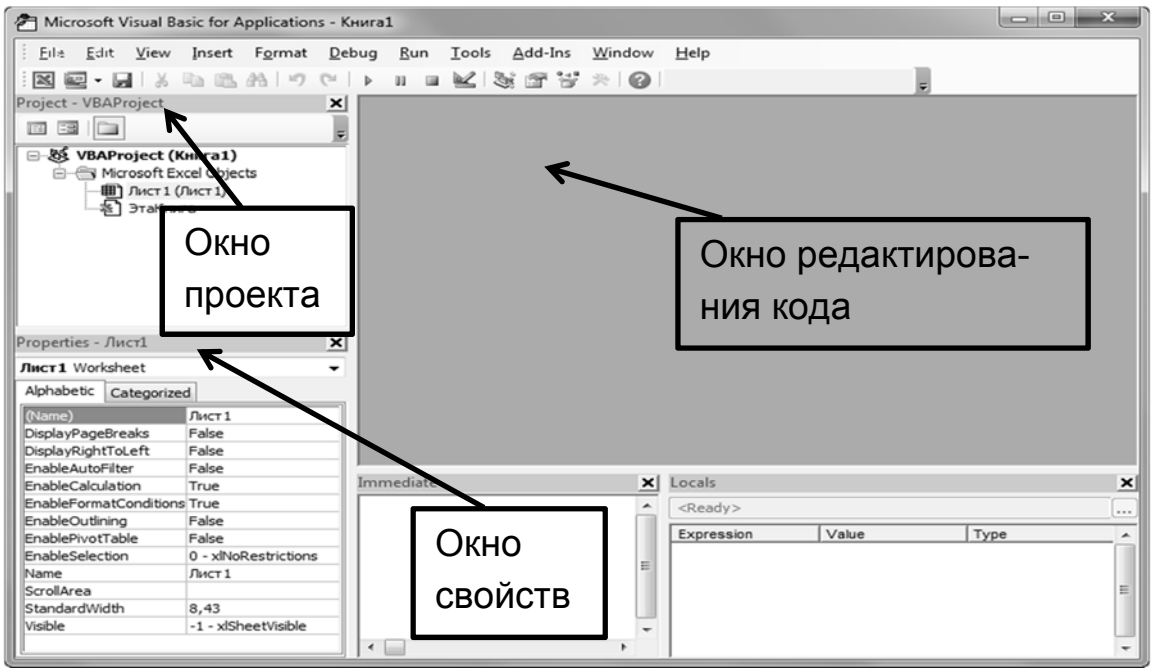


Рис. 13. Основное окно среды разработки VBA

Таблица 3

Назначение компонентов VBE

Наименование окна	Команда View (Вид)	Описание
1	2	3
Project (Проект)	Project Explorer (Окно проекта)	Предназначено для отображения всех открытых проектов, а также их составляющих: модулей, форм и ссылок на другие проекты
Toolbox (Панель элементов)	Toolbox (Панель элементов)	Содержит элементы управления для конструирования форм
UserForm	Object (Объект)	Используется для создания форм путём размещения на них элементов управления
Code (Программа)	Code (Программа)	Предназначено для просмотра, написания и редактирования программы на языке VBA. Поскольку среда разработки является многооконной, то для каждого модуля проекта можно открыть отдельное окно
Properties (Свойства)	Properties (Окно свойств)	Отображает свойства выделенных объектов. В этом окне можно задавать новые значения свойств формы и элементов управления
Object Browser (Просмотр объектов)	Object Browser (Просмотр объектов)	Отображает классы, свойства, методы, события и константы различных библиотек объектов. Используется для быстрого получения информации об объектах

Продолжение табл. Таблица 3

1	2	3
Immediate (Проверка)	Immediate (Окно отладки)	Предназначено для быстрого выполнения вводимых в него инструкций. В данном окне также выводятся результаты выполнения вводимых инструкций
Locals (Локальные переменные)	Locals (Окно локальных переменных)	Автоматически показывает все переменные данной процедуры
Watches (Контрольные значения)	Watches (Окно контрольных значений)	Применяется при отладке программ для просмотра значений выражений

2.3.1. Окно $Project$ (Окно проекта)

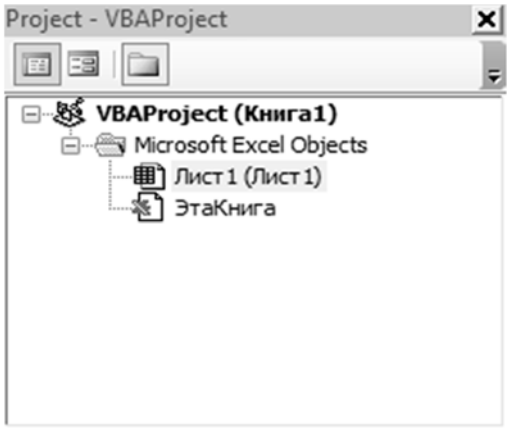


Рис. 14. Окно проекта

Окно ***Project*** (Проект), иногда его называют окно проводника проектов, предназначено для быстрого получения информации о различных составляющих проекта. Такими составляющими являются ***Forms*** (Формы), ***Modules*** (Модули) и ***References*** (Ссылки).

Окно проекта можно использовать также для быстрой навигации по формам проекта и программному коду. Для этого необходимо выбрать в контекстном меню соответственно команды ***View Object*** (**Объект**) или ***View Code*** (**Программа**) (рис. 14).

Окно ***Проводника проектов*** при первой активизации редактора $Visual\ Basic$ обычно открыто. Если оно случайно было закрыто, то вызвать его можно:

- нажав на клавиши $Ctrl + R$;

- нажав на кнопку $Project\ Explorer$ на панели $Standard$;
- воспользовавшись меню $View \rightarrow Project\ Explorer$.

2.3.2. Окно $Properties$ (Окно свойств)

Список свойств выделенного объекта выводится в окне $\boldsymbol{Properties}$ (Свойства) (рис. 15). Для того чтобы выделить объект, необходимо с помощью окна проекта выбрать форму и перейти в режим конструктора, используя команду $\boldsymbol{View\ Object}$. Свойства объекта можно упорядочить в алфавитном порядке ($\boldsymbol{Alphabetic}$ (По алфавиту)) или по категориям ($\boldsymbol{Categorized}$ (По категориям)), выбрав соответствующую вкладку. Предусмотрена также возможность получения быстрой справки по какому-либо свойству объекта. Для этого достаточно установить курсор на нужное свойство и нажать клавишу $F1$.

Рис. 15. Окно свойств

2.3.3. Окно $Object\ Browser$ (Окно просмотра объектов)

Окно $\boldsymbol{Object\ Browser}$ (Просмотр объектов) (рис. 16) предназначено для просмотра объектов, доступных при создании программы. Хотя на самом деле в этом окне просматриваются не объекты, а

структуры соответствующего класса объектов (более подробно понятие объект, класс и другие понятия объектно-ориентированного программирования рассматриваются в дисциплине «Объектно-ориентированное программирование»). Окно просмотра объектов может использоваться для поиска метода или свойства объекта.

Чтобы найти какое-либо свойство или метод, необходимо выполнить следующую последовательность действий:

1. Откройте в редакторе $Visual\ Basic$ нужный модуль.

2. Нажмите на панели инструментов кнопку $\boldsymbol{Object\ Browser}$ (Просмотр объектов).

3. Используя раскрывающийся список $\boldsymbol{Project - Library}$ (Проект – Библиотека), расположенный в верхнем левом углу окна просмотра объектов, выберите нужную библиотеку.

4. Отметьте нужный объект в списке $\boldsymbol{Classes}$ (Классы).

5. Используя список $\boldsymbol{Members\ Of}$ (Компонент), выберите подходящий метод или свойство.

Рис. 16. Окно просмотра объектов

Для получения сведений о выбранном классе, методе, событии или свойстве нажмите кнопку $\boldsymbol{Help}$ (Справка) в окне $\boldsymbol{Object\ Browser}$ (Просмотр объектов).

2.3.4. Окно Code (Окно редактора кода)

Окно ***Code*** (Программа) представляет собой текстовый редактор, предназначенный для написания и редактирования кода процедур приложения. Это окно появляется на экране, например, при создании нового модуля. Код внутри модуля организован в виде отдельных разделов для каждого объекта, программируемого в модуле. Переключение между разделами выполняется путём выбора значений из списка ***Object*** (Объект), который находится в левом верхнем углу окна. Каждый раздел может содержать несколько процедур, которые можно выбрать из списка ***Procedure*** (Процедура) в правом верхнем углу (на рис. 17 этот список раскрыт). В окне редактора доступны два режима представления кода: просмотр отдельной процедуры и всего модуля. Переключение режимов работы окна осуществляется выбором одной из двух кнопок в нижнем левом углу окна редактора кода (см. рис. 17).

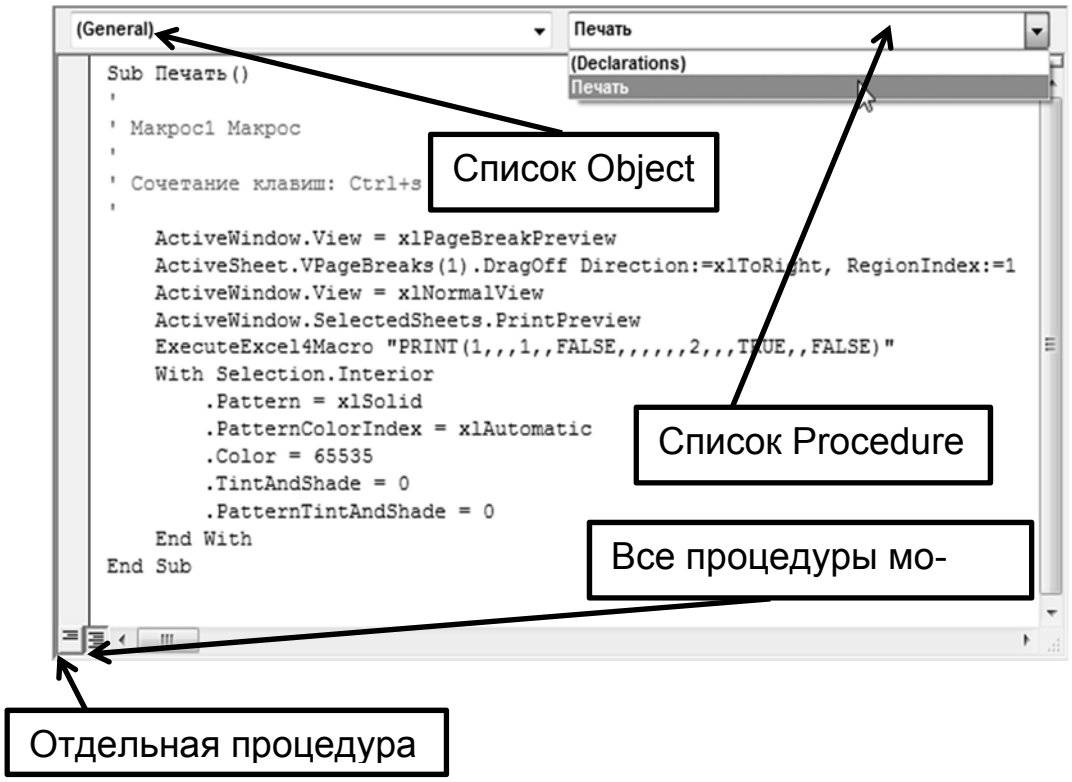


Рис. 17. Окно редактора кода

В верхней части окна редактора кода находятся два списка. Список слева – это список объектов. В нем вы можете выбрать объ-

ект, к которому будет относиться ваш код. Если вы открыли программный код модуля, то здесь будет только пункт ($General$). Другое дело, если открыта форма – в этом списке вы сможете выбрать саму форму или любой ее элемент управления и записать для него код.

Список справа – это список процедур/событий. В нем есть раздел ($Declarations$) – объявления уровня всего модуля и список всех процедур (макросов) для стандартного модуля или событий, если создается код для формы. При выборе нужного события будет автоматически создана нужная процедура, обрабатывающая это событие.

В редакторе кода выполняется основная часть работы по программированию, поэтому знать приёмы работы с ним нужно очень хорошо. Открыть окно редактора кода можно множеством способов:

- выбрать нужный элемент (в $Project\ Explorer$, в дизайнере форм и т.п.) и в контекстном меню выбрать $View \rightarrow Code$;
- нажать на кнопку $F7$;
- выбрать $View \rightarrow Code$ из меню;
- дважды щёлкнуть по объекту модуля в $Project\ Explorer$ (или выделить его и нажать на кнопку $Enter$).

Редактор программного кода – это, по сути, обычный текстовый редактор, и в нем вы вполне можете вырезать и вставлять код, перетаскивать его, скопировать путём перетаскивания с нажатой клавишей $Ctrl$ – в вашем распоряжении почти все те же возможности, что и в редакторе $Word$. Однако он все-таки предназначен для специализированной задачи – создания кода программы.

а) Интеллектуальные возможности редактора кода:

1. При написании кода пользователю предлагается список компонентов, логически завершающих вводимую пользователем инструкцию.
2. На экране автоматически отображаются сведения о процедурах, функциях, свойствах и методах после набора их имени.
3. Автоматически проверяется синтаксис набранной строки кода сразу после нажатия клавиши $\boldsymbol{Enter}$. В результате проверки выполняется выделение определённых фрагментов текста:

- красным цветом – синтаксические ошибки;
- синим цветом – зарезервированные ключевые слова;
- зелёным цветом – комментарии.

4. Если курсор расположить на ключевом слове VBA, имени процедуры, функции, свойства или метода и нажать клавишу $F1$, то на экране появится окно со справочной информацией об этой функции.

б) Применение закладок в редакторе VBA, линия разбивки

Иногда в процессе написания программного кода в одном месте вам в голову приходит идея, относящаяся к другой части кода. Надо бы перепрыгнуть в другое место, но разыскивать потом ту строку, где была прервана работа, очень не хочется. В этом случае опытные программисты используют закладки. Закладка (как и в случае с обычной книгой) – это метка, при помощи которой можно быстро найти нужное место. Работа с закладками производится либо из панели инструментов $Edit$ (вначале панель нужно сделать видимой), либо через меню $Edit \rightarrow Bookmark$. Для того, чтобы включить или отключить закладку, нужно установить указатель ввода на нужную строку и воспользоваться командой $Toggle\ Bookmark$.

Часто бывает очень удобно разделить окно редактирования на две части – для просмотра разных частей модуля, для копирования и т.п. Делается это при помощи линии разбивки – маленького бегунка сразу над полосой прокрутки.

2.3.5. Окно UserForm (Окно редактирования форм)

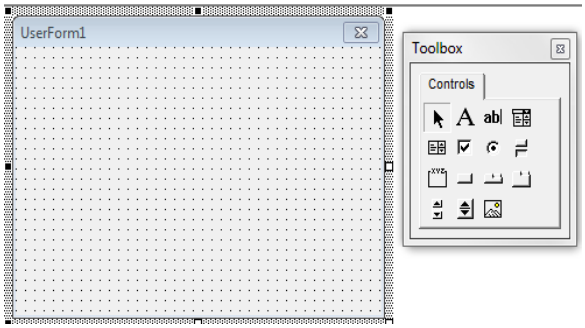


Рис. 18. Окно редактирования форм и панель инструментов Toolbox (Панель элементов)

Для создания диалоговых окон разрабатываемых приложений VBA используются формы. Редактор форм является одним из основных средств визуального программирования. При добавлении формы в проект (команда $\boldsymbol{Insert - UserForm}$ (**Вставить** – $\boldsymbol{UserForm}$)) на экран выводится незаполненная форма (рис. 18) с панелью инструментов $\boldsymbol{Toolbox}$ (Панель элементов).

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

2.3.6. Окно $\boldsymbol{Immediate}$ (Окно проверки)

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

2.3.7. Окно $\boldsymbol{Locals}$ (Окно локальных переменных)

Окно $\boldsymbol{Locals}$ (**Локальные переменные**) автоматически отображает все объявленные переменные текущей процедуры и их значения. Переменные других модулей, объявленные как *Public* и используемые в текущей процедуре, не отображаются.

2.3.8. Окно $\boldsymbol{Watches}$ (Окно контрольных значений)

Окно $\boldsymbol{Watches}$ (**Контрольные значения**) (рис. 19) применяется при отладке программ для просмотра значений выражений. Окно *Watches* представляет большую ценность – в это окно можно просто

"перетащить" нужную переменную или объект, и в этом окне будут отражены все данные об имени переменной, её типе и текущем значении:

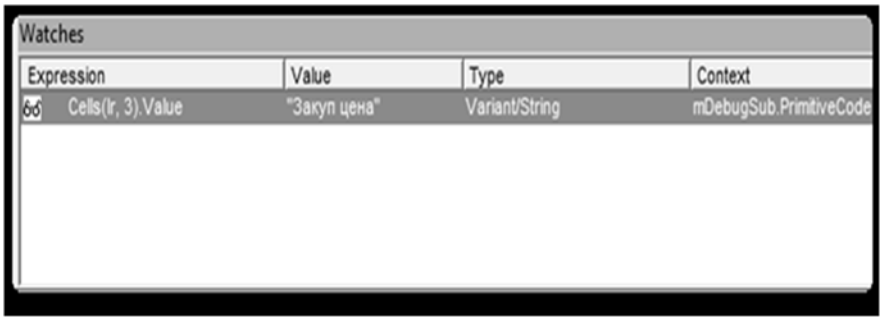


Рис. 19. Окно Watches

2.3.9. Отладчик Debugger (пошаговая отладка кода)

После знакомства с отладкой кода при возникновении ошибки работать с пошаговой отладкой будет проще.

Что такое вообще пошаговая отладка?

Это просмотр этапов выполнения кода строка за строкой.

Для чего это может быть нужно?

Чтобы проанализировать чужой код и понять более точно, что он делает изнутри, а не только увидеть результат его выполнения.

Если вы начинающий программист и часто используете макрорекордер (записываете макросы), то пошаговая отладка поможет понять, какое действия выполняет каждая строка. Это поможет быстрее научиться понимать код и убирать из него лишнее, а также совмещать различные коды.

Если внутри кода есть ошибка логики выполнения, то это, пожалуй, самая сложная ошибка. При такой ситуации *VBA* не останавливает работу и не говорит об ошибке. Код выполняется без ошибок, но результат не такой, как ожидалось. Это может означать, что какой-то переменной назначается не то значение, либо какое-то условие неверно, либо выполняется не в тот момент, в который должно. В общем, по сути, это ошибка разработчика, не приводящая к ошибкам синтаксиса или типов, которые *VBA* может отследить.

Как делать пошаговую отладку? Все просто: устанавливаете курсор в любом месте внутри кода и нажимаете клавишу *F8* (либо выбрать в меню *Degub-Step Into*). Теперь при каждом нажатии клавиши *F8* код будет выполнять одну строку кода за другой в той очередности, в которой они расположены в процедуре. Если внутри процедуры будет вызов второй процедуры или функции, то код пошагово выполнит и её, а затем вернется в основную процедуру.

Можно привести еще сочетания клавиш, которые удобно применять при пошаговой отладке:

- ***Shift+F8 (Degub-Step Over)*** - выполнение вложенной функции/процедуры без захода в неё. Если внутри основной процедуры или функции выполняется другая процедура или функция и вы уверены, что она работает правильно, то просматривать пошагово весь код вложенной процедуры/функции не имеет смысла. Чтобы вложенная процедура/функция выполнилась без пошагового просмотра, надо просто нажать указанное сочетание клавиш тогда, когда строка вызова вложенной процедуры/функции будет подсвечена желтым.
- ***Ctrl+Shift+F8 (Degub-Step Out)*** - завершение вложенной функции/процедуры и выход в основную с остановкой. Если все же перестарались и перешли в пошаговый проход вложенной функции (или сделали это специально, но посмотрели все, что надо), то нажмите это сочетание, и код быстро выполнит вложенную функцию, перейдет в основную и остановится для дальнейшей пошаговой отладки.
- ***Ctrl+F8 (Degub-Run to Cursor)*** - выполнение процедуры до строки, в которой на данный момент установлен курсор.

Часто бывает нужно не просто весь код пройти пошагово, а начать пошаговое выполнение только с какой-либо одной строки, чтобы не мотать множество строк кода (да еще с циклами) ради достижения одной какой-то строки. Еще точки останова очень полезны при отладке событийных процедур (вроде ***Worksheet_Change***, ***Worksheet_BeforeDoubleClick***, событий элементов форм и т.п.) в

следствие того, что они в большинстве своем содержат аргументы, и выполнить по ***F8*** их просто невозможно. Выполняются они только при наступлении самого события, которое они призваны обработать. Тоже самое справедливо для функций пользователя (*UDF*) именно для проверки их работы из листа ввиду того, что эти функции нельзя начать выполнять по ***F5*** – они начинают выполняться только после их пересчета и зачастую ошибки можно выявить исключительно при вызове именно с листа.

Чтобы дать понять *VBA* на какой строке нужно будет остановится, необходимо установить курсор мыши в любое место нужной строки и нажать ***F9*** или ***Debug-Toggle Breakpoint***. Строка будет выделена темно-красным цветом.

Это называется ***установкой точки останова***. Убрать точку останова можно так же, как она была установлена – ***F9*** или ***Debug-Toggle Breakpoint***. Также точку основа можно установить с помощью мыши: для этого необходимо в области левее окна с кодом напротив нужной строки один раз щелкнуть левой кнопкой мыши.

Теперь можно запустить код любым удобным способом (в отладке это, как правило, делается клавишей ***F5*** или с панели ***Run-Run Sub/UserForm***). Как только код дойдет до указанной точки останова, он остановится, и строка будет подсвечена желтым. Дальше можно либо продолжить выполнение в пошаговом режиме (нажимая ***F8***), либо (проверив значения нужных переменных и объектов) нажать опять ***F5*** и код продолжит выполняться автоматически, до тех пор, пока не выполнится или не достигнет другой точки останова. Самих же точек останова может быть сколько угодно и расположены они могут быть в любой процедуре или функции.

Следует помнить, что после закрытия файла с кодом точки останова не сохраняются и при следующем открытии книги их необходимо будет установить заново, если это необходимо.

Контрольные вопросы к главе 2

1. Как запустить среду $\boldsymbol{PascalABC.NET}$?
2. Как создать новую страницу? Как открыть текст ранее созданной программы?
3. Как закрыть текущую страницу с текстом программы? Можно ли закрыть все неактивные страницы?
4. Пусть у вас открыто несколько программ. Как сделать активной какую-либо из них?
5. Как исполнить программу, размещённую на какой-либо странице?
6. Что такое макрорекордер?
7. Как запустить макрорекордер?
8. Что такое макрос? Где его можно сохранить?
9. Какова процедура запуска макросов на выполнение?
10. Какие способы запуска макросов на выполнение вы знаете?
11. Как запустить $\boldsymbol{VBA\ IDE}$?
12. Какие окна используются в $\boldsymbol{VBA\ IDE}$?
13. Для чего нужно окно $\boldsymbol{Project}$?
14. Что показывает окно $\boldsymbol{Properties}$?
15. Как можно узнать вычисляемые значения локальных переменных?
16. Как в окне $Immediate$ произвести вычисления?
17. Что такое $\boldsymbol{UserForm}$?
18. Для чего используется ***Панель элементов***?

Глава 3. ТИПЫ ДАННЫХ

Тип данных (встречается также термин ***«вид данных»***) – фундаментальное понятие теории программирования. Тип данных определяет множество значений, набор операций, которые можно применять к таким значениям и, возможно, способ реализации хранения значений и выполнения операций. Любые данные, которыми оперируют программы, относятся к определённым типам [4].

Концепция типов данных является одной из центральных в любом языке программирования. С типом величины связаны три её свойства: форма внутреннего представления, множество принимаемых значений и множество допустимых операций.

Понятие типа по Ч. Хоару[2]:

- Тип определяет класс значений, которые могут принимать переменная или выражение.
- Каждое значение принадлежит одному и только одному типу.
- Тип значения константы, переменной или выражения можно вывести либо из контекста, либо из самого операнда, не обращаясь к значениям, вычисляемым во время работы программы.
- Каждой операции соответствует некоторый фиксированный тип её операндов и некоторый фиксированный (обычно такой же) тип результата. Разрешение систематической неопределённости в случае, когда один и тот же символ применяется к операндам разного типа, производится на стадии компиляции.
- Для каждого типа свойства значений и элементарных операций над значениями задаются с помощью аксиом.
- При работе с языком программирования знание типа позволяет обнаруживать бессмысленные конструкции и решать вопрос о методе представления данных и преобразования их в ЭВМ.

---

[2] Сэр Чарльз Э́нтони Ри́чард Хо́ар (англ. Charles Antony Richard Hoare или Tony Hoare или C.A.R. Hoare) родился 11 января 1934 в Коломбо, Цейлон, Британская империя, ныне Шри Ланка) — английский учёный, специализирующийся в области информатики и вычислительной техники. Наиболее известен как разработчик алгоритма «быстрой сортировки» (1960), на сегодняшний день являющегося наиболее популярным алгоритмом сортировки.

Типы данных различаются, начиная с нижних уровней компьютера. Так, даже в Ассемблере x86 различаются типы «целое число» и «вещественное число». Это объясняется тем, что для чисел рассматриваемых типов отводятся различные объёмы памяти, используются различные регистры микропроцессора, а для операций с ними применяются различные команды Ассемблера и различные ядра микропроцессора.

Концепция типа данных появилась в языках программирования высокого уровня как естественное отражение того факта, что обрабатываемые программой данные могут иметь различные множества допустимых значений, храниться в памяти компьютера различным образом, занимать различные объёмы памяти и обрабатываться с помощью различных команд процессора.

Как правило, типы языков программирования не всегда строго соответствуют подобным математическим типам. Например, тип «целое число» большинства языков программирования не соответствует принятому в математике типу «целое число», так как в математике указанный тип не имеет ограничений ни сверху, ни снизу, а в языках программирования эти ограничения есть. Как правило, в языках и системах имеется множество целых типов, отличающихся допустимым диапазоном значений (определяемым объёмом занимаемой памяти). Стоит отметить, что в большинстве реализаций языков и систем выход за границу целого типа (переполнение) не приводит к исключительной ситуации.

3.1. Языки без типов

Теоретически не может существовать языков, в которых отсутствуют типы (включая полиморфные - представление набора типов как единственного типа). Это следует из того, что все языки основаны на машине Тьюринга или на лямбда-исчислении (см. [3], раздел 1.2). И в том, и в другом случае необходимо оперировать как минимум одним типом данных – хранящимся на ленте (машина Тьюринга) или передаваемым и возвращаемым из функции (лямбда-исчисление). Ниже перечислены языки программирования по способу определения типов данных:

1. ***Языки с полиморфным типом данных*** - не связывают переменные, константы, формальные параметры и возвращаемые значения функций с определёнными типами, поддерживая единственный полиморфный тип данных. В чистом виде таких языков не встречается, но близкие примеры – VBA – тип $variant$, *Пролог, Лисп* – списки. В этих языках переменная может принимать значение любого типа, в параметры функции можно передавать значения любых типов и функция также может вернуть значение любого типа. Сопоставление типов значений переменных и параметров с применяемыми к ним операциями производится непосредственно при выполнении этих операций. Например, выражение $a + b$ может трактоваться как сложение чисел, если a и b имеют числовые значения, как конкатенация строк, если a и b имеют строковые значения, и как недопустимая (ошибочная) операция, если типы значений a и b несовместимы. Такой порядок называют «динамической типизацией» (соответствует понятию полиморфизм в ООП, полиморфный тип в теории типов). Языки, поддерживающие только динамическую типизацию, называют иногда ***«бестиповыми».*** Это название не следует понимать, как признак отсутствия понятия типов в языке – типы данных всё равно есть.

2. ***Языки с неявным определением типов***. Казалось бы, $BASIC$ является примером языка без типов. Однако это строго типизированный язык: в нём различаются строковые типы (добавляется символ, массивы (добавляется [ ]) и числовые типы (ничего не добавляется).

3. ***Языки с типом, определяемым пользователем***. Также хорошо известны языки, в которых типы данных определяются автоматически, а не задаются пользователем. Каждой переменной, параметру, функции приписывается определённый тип данных. В этом случае для любого выражения возможность его выполнения и тип полученного значения могут быть определены без исполнения программы. Такой подход называют «статической типизацией». При этом правила обращения с переменными, выражениями и параметрами разных типов могут быть как очень строгими $C++$, так и весьма либеральными (C). Например, в классическом языке Си практически

все типы данных совместимы – их можно применять совместно в любых выражениях, присваивать значение переменной одного типа переменной другого почти без ограничений. При таких операциях компилятор генерирует код, обеспечивающий преобразование типов, а логическая корректность такого преобразования остаётся на совести программиста. Подобные языки называют *«языками со слабой типизацией»*. Противоположность им – *«языки с сильной типизацией»*, такие как Ада или Паскаль. В них каждая операция требует операндов строго заданных типов. Никакие автоматические преобразования типов не поддерживаются – их можно выполнить только явно, с помощью соответствующих функций и операций. Сильная типизация делает процесс программирования более сложным, но даёт в результате программы, содержащие заметно меньше трудно обнаруживаемых ошибок.

На практике языки программирования поддерживают несколько моделей определения типов одновременно.

3.2. Преимущества от использования типов данных

Надёжность. Типы данных защищают от трёх видов ошибок:

- *Некорректное присваивание*. Пусть переменная объявлена как имеющая числовой тип. Тогда попытка присвоить ей символьное или какое-либо другое значение в случае статической типизации приведёт к ошибке компиляции и не даст такой программе запуститься. В случае динамической типизации код программы перед выполнением потенциально опасного действия сравнит типы данных переменной и значения и также выдаст ошибку. Всё это позволяет избежать неправильной работы и «падения» программы.
- *Некорректная операция*. Позволяет избежать попыток применения выражений вида «$Hello\ world$» $+\ 1$. Поскольку, как уже говорилось, все переменные в памяти хранятся как наборы битов, то при отсутствии типов подобная операция была бы выполнима (и могла дать результат вроде «$ello\ worldÆ$»). С использованием типов такие ошибки отсекаются опять же на этапе компиляции.
- *Некорректная передача параметров*. Если функция «синус»

ожидает, что ей будет передан числовой аргумент, то передача ей в качестве параметра строки «$Hello\ world$» может иметь непредсказуемые последствия. При помощи контроля типов такие ошибки также отсекаются на этапе компиляции.

Стандартизация. Благодаря соглашениям о типах, поддерживаемых большинством систем программирования, сложилась ситуация, когда программисты могут быстро менять свои рабочие инструменты, а программы не требуют больших переделок при переносе исходных текстов в другую среду. К сожалению, стандартизации по универсальным типам данных ещё есть куда развиваться.

3.3. Контроль типов и системы типизации

Процесс проверки и накладывания ограничений типов – контроля типов, может выполняться во время компилирования (статическая проверка) или во время выполнения (динамическая проверка):

- *Статическая типизация* – контроль типов осуществляется при компиляции.
- *Динамическая типизация* – контроль типов осуществляется во время выполнения.

Контроль типов также может быть строгим и слабым:

- *Строгая типизация* – совместимость типов автоматически контролируется транслятором.
- *Номинативная типизация* (англ. $nominative\ type\ system$) – совместимость должна быть явно указана (наследована) при определении типа.
- *Структурная типизация* (англ. $structural\ type\ system$) – совместимость определяется структурой самого типа (типами элементов, из которых построен составной тип).
- *Слабая типизация* – совместимость типов никак транслятором не контролируется. В языках со слабой типизацией обычно используется подход под названием «утиная типизация» – когда совместимость определяется и реализуется общим интерфейсом доступа к данным типа.

3.4. Классификация типов данных

Каждый язык программирования поддерживает один или несколько встроенных типов данных (базовых типов), кроме того, развитые языки программирования предоставляют программисту возможность описывать собственные типы данных, комбинируя или расширяя существующие.

3.4.1. Простые

Перечислимый тип. Может хранить только те значения, которые прямо указаны в его описании.

Числовые. Хранятся числа. Могут применяться обычные арифметические операции.

Целочисленные: со знаком, то есть могут принимать как положительные, так и отрицательные значения; и без знака, то есть могут принимать только неотрицательные значения.

Вещественные: с запятой (то есть хранятся знак и цифры целой и дробной частей) и с плавающей запятой (то есть число приводится к виду $m \times b\text{^}e$, где m – мантисса, b – основание показательной функции, e – показатель степени (порядок) (в англоязычной литературе экспонента), причём в нормальной форме $0 \leq m < b$, а в нормализованной форме $1 \leq m < b$, e – целое число и хранятся знак и числа m и e).

Числа произвольной точности, обращение с которыми происходит посредством длинной арифметики. Примером языка с встроенной поддержкой таких типов является $UBASIC$, часто применяемый в криптографии.

Символьный тип. Хранит один символ. Могут использоваться различные кодировки.

Логический тип. Имеет два значения: истина и ложь. Могут применяться логические операции. Используется в операторах ветвления и циклах. В некоторых языках является подтипом числового типа, при этом ложь = 0, истина = 1.

Множество. В основном совпадает с обычным математическим понятием множества. Допустимы стандартные операции с множествами и проверка на принадлежность элемента множеству. В некоторых языках рассматривается как составной тип.

3.4.2. Составные (сложные)

Массив. Является индексированным набором элементов одного типа. Одномерный массив – вектор, двумерный массив – матрица.

Строковый тип. Хранит строку символов. Аналогом сложения в строковой алгебре является конкатенация (прибавление одной строки в конец другой строки). В языках, близких к бинарному представлению данных, чаще рассматривается как массив символов, в языках более высокой абстракции зачастую выделяется в качестве простого.

Запись (структура). Набор различных элементов (полей записи), хранимый как единое целое. Возможен доступ к отдельным полям записи. Например, $struct$ в C или $record$ в $Pascal$.

Файловый тип. Хранит только однотипные значения, доступ к которым осуществляется только последовательно (файл с произвольным доступом, включённый в некоторые системы программирования, фактически является неявным массивом).

3.4.3. Класс

Другие типы данных. Если описанные выше типы данных представляли какие-либо объекты реального мира, то рассматриваемые здесь типы данных представляют объекты компьютерного мира, то есть являются исключительно компьютерными терминами.

Указатель. Хранит адрес в памяти компьютера, указывающий на какую-либо информацию, как правило, – указатель на переменную.

3.5. Типы данных в $PascalABC.NET$

Типы в $PascalABC.NET$ подразделяются на простые, строковые, структурированные, типы указателей, процедурные типы и классы.

К *простым* относятся целые и вещественные типы, логический, символьный, перечислимый и диапазонный тип (табл. 4).

Таблица 4

Целочисленные типы данных PascalABC.NET

Целочисленный тип	Диапазон значений			Требуемая память (байт)
Integer	-32 768	..	32 767	2
ShortInt	-128	..	127	1
LongInt	-2 147 483 648	..	2 147 483 647	4
Byte	0	..	255	1
Word	0	..	65 535	2

К *структурированным* типам относятся массивы, записи, множества и файлы.

Все простые типы, кроме вещественного, называются порядковыми. Только значения этих типов могут быть индексами статических массивов и параметрами цикла for. Кроме того, для порядковых типов используются функции Ord, $Pred$ и $Succ$, а также процедуры Inc и Dec.

Все типы, кроме типов указателей, являются производными от типа $Object$. Каждый тип в $PascalABC.NET$ имеет отображение на тип $.NET$. Тип указателя принадлежит к неуправляемому коду и моделируется типом $void*$.

Все типы в $PascalABC.NET$ подразделяются на две большие группы: *размерные и ссылочные*. К *размерным* относятся все простые типы, указатели, записи, статические массивы, множества и строки. К *ссылочным* типам относятся классы, динамические массивы, файлы и процедурный тип.

Размерные типы более эффективны при вычислениях: они занимают меньше памяти и операции, выполняемые над небольшими размерными типами, максимально эффективны. Ссылочные типы обладают большей гибкостью: память под них выделяется динамически в процессе работы программы и освобождается автоматически, когда объект ссылочного типа перестаёт использоваться.

Выделение памяти

Память под переменную размерного типа распределяется на программном стеке в момент её описания. При этом переменная размерного типа хранит значение этого типа.

```
var i: integer;          здесь под i выделяется память
i := 5;
```

Переменная ссылочного типа представляет собой ссылку на объект некоторого класса в динамической памяти. Если она не инициализирована, то хранит специальное значение nil (нулевая ссылка). Для инициализации ссылочных переменных используется вызов конструктора соответствующего класса.

Пример 3.1.

```
type Person = auto class
        name: string;
        age: integer;
end;
var p: Person;                  p хранит значение nil, память
                                под объект не выделена

p := new Person('Иванов',20);   конструктор выделяет память
                                под объект
```

Присваивание

При присваивании переменных размерного типа копируются значения этого типа. Если размерный тип имеет большой размер, эта операция может выполняться долго.

Пример 3.2.

```
var a,a1: array [1..1000000] of integer;
a1 := a;                        копируются все 1000000
                                элементов
```

При присваивании переменных ссылочного типа осуществляется присваивание ссылок, в итоге после присваивания обе ссылки ссылаются на один объект в динамической памяти.

Пример 3.3.

```
var p1: Person;
p1 := p;                        копируется ссылка
```

Сравнение на равенство

Сравнение на равенство объектов размерного типа сравнивает их значения. В частности, две переменные типа запись равны, если равны все поля этих записей.

Пример 3.4.

```
type PersonRec = record
        name: string;
        age: integer;
end;
var p,p1: PersonRec;
p.name := 'Иванов'; p.age := 20;
p1.name := 'Иванов'; p1.age := 20;
writeln(p1=p);                                  True
```

При сравнении на равенство переменных ссылочного типа проверяется, что они ссылаются на один и тот же объект.

Пример 3.5.

```
var p := new Person('Иванов',20);
var p1 := new Person('Иванов',20);
writeln(p1=p);                                  False
```

Управление памятью

Размерные типы распределяются на программном стеке, поэтому не нуждаются в специальном управлении памятью. Под глобальные размерные переменные память распределена всё время работы программы. Под локальные размерные переменные память выделяется в момент вызова подпрограммы, а освобождается в момент завершения работы этой подпрограммы.

Управление памятью для ссылочных типов осуществляется автоматически сборщиком мусора. Сборщик мусора запускается в неопределённый момент времени, когда управляемой памяти перестаёт хватать. Он возвращает в пул неиспользуемой памяти те объекты, на которые больше никто не ссылается, после чего дефрагментирует оставшуюся память, в результате чего динамическая память всегда дефрагментирована и её выделение при вызове конструктора происходит практически мгновенно.

Передача в подпрограммы

При передаче размерных типов по значению происходит копирование значения фактического параметра в переменную-формальный параметр. Если размерный тип имеет большой размер, это может занимать продолжительное время, поэтому размерный тип в этом случае передаётся по ссылке на константу.

Пример 3.6.

```
type Arr = array [1..100] of integer;
…
procedure PrintArray(const a: Arr; n: integer);
begin
   for var i:=1 to n do
   Print(a[i])
end;
```

Ссылочные типы передаются в подпрограмму, как правило, по значению. При передаче таких параметров происходит копирование ссылки, в результате формальный и фактический параметр будут ссылаться на один объект.

Пример 3.7.

```
procedure Change666(a: array of integer);
begin
   a[0] := 666;
end;
```

При этом в результате изменения формального параметра внутри подпрограммы меняется и содержимое соответствующего фактического параметра при вызове подпрограммы.

Детально о типах данных в *PascalABC.NET* говорится в Главе**Ошибка! Источник ссылки не найден.**

3.6. Типы данных в *VBA*

Тип данных – это характеристика переменной, определяющая тип содержащихся в ней данных. К типам данных относятся типы, указанные в табл. 5, а также пользовательские типы и определенные типы объектов.

Таблица 5

Типы данных в VBA

Тип данных	Размер (в байтах)	Описание и диапазон значений
1	2	3
Array	Зависит от числа элементов и их размера	Массив переменных любого встроенного типа данных
Boolean	2	Одно из логических значений: $\boldsymbol{True}$ (истина) или $\boldsymbol{False}$ (ложь)
Byte	1	Положительное число от 0 до 255
Currency	8	Используется для денежных вычислений с фиксированным количеством десятичных знаков. От -922 337 203 685 477,5808 до 922 337 203 685 477,5807
Date	8	Комбинация информации о дате и времени. Диапазон дат: от 01.01.0100 г. до 31.12.9999 г. Диапазон времени: от 00:00:00 до 23:59:59
Decimal	14	Десятичное представление данных в целочисленной или вещественной форме
Double	8	Число с плавающей точкой двойной точности. Отрицательные числа: от -1,79769313486232E+308 до -4,94065645841247E-324. Положительные числа: от 4,94065645841247E-324 до 1,79769313486232E+308
Integer	2	Целое число от -32 768 до 32 76
Long	4	Длинное целое число: от -2 147 483 648 до 2 147 483 647
Object	4	Ссылка на любой определённый объект, распознаваемый VBA
Single	4	Число с плавающей точкой обычной точности. Отрицательные числа от -3,402823E+38 до 1,401298E-45. Положительные числа от 1,401298E-45 до 3,402823E+38
String (переменной длины)	10 байт + длина строки	Длина строки от 0 до 2 миллиардов символов

Продолжение табл.Таблица 5

1	2	3
String (фиксированной длины)	Длина строки (1 байт на символ)	Длина строки от 0 до приблизительно 65 400 символов
Variant	16 байт + 1 байт на каждый символ строки	Может использоваться для хранения любого типа данных, кроме строк фиксированной длины. Диапазон зависит от фактически сохраняемых данных

Контрольные вопросы к главе 3

1. Что такое типы данных?
2. Могут ли существовать языки программирования без типов данных?
3. Что понимается под полиморфным типом данных?
4. Какие преимущества от использования типов данных?
5. Какие бывают системы типизации данных?
6. Что понимается под контролем типов?
7. Что понимается под классификацией типов данных?
8. Какие типы данных относятся к простым, какие к сложным (составным)?
9. Что такое размерные и ссылочные типы данных?
10. Какие типы данных используются в $\boldsymbol{VBA}$?
11. Для чего в VBA используется тип данных $\boldsymbol{Variant}$?
12. Для чего в VBA используется тип данных $\boldsymbol{Object}$?
13. Какова длина в байтах типа данных $\boldsymbol{Single}$?
14. Что понимается под термином «ссылочный тип»?
15. Что происходит при передаче размерных типов по значению?

Глава 4. СТРУКТУРА ПРОГРАММЫ

4.1. Структура программы $PascalABC.NET$

4.1.1. Алфавит

Алфавит языка состоит из множества символов, включающих в себя буквы, цифры и специальные символы.

Латинские буквы: от A до Z (прописные) и от a до z (строчные).

Цифры: $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$.

Шестнадцатеричные цифры: $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$, A, B, C, D, E, F.

Специальные символы: $(+ - * \neq < >$ [ ]., ( ): ; { }^ @$ #)

Следующие комбинации специальных символов являются едиными символами (их нельзя разделять пробелами):

:= знак присваивания;

≤ меньше или равно; >= больше или равно; <> не равно;

$(* *)$ ограничители комментариев, (используются наряду с {┤});

$(..)$ эквивалент [ ].

Пробелы – символ пробела $(ASCII - 32)$ и все управляющие символы кода $ASCII$ (от 0 до 31).

Программа содержит ключевые слова, идентификаторы, комментарии. Ключевые слова используются для выделения синтаксических конструкций и подсвечиваются жирным шрифтом в редакторе. Идентификаторы являются именами объектов программы и не могут совпадать с ключевыми словами.

4.1.2. Идентификатор

Идентификаторы служат в качестве имён программ, модулей, процедур, функций, типов, переменных и констант. Идентификатором считается любая последовательность латинских букв или цифр, начинающаяся с буквы. Буквой считается также символ подчёркивания "_".

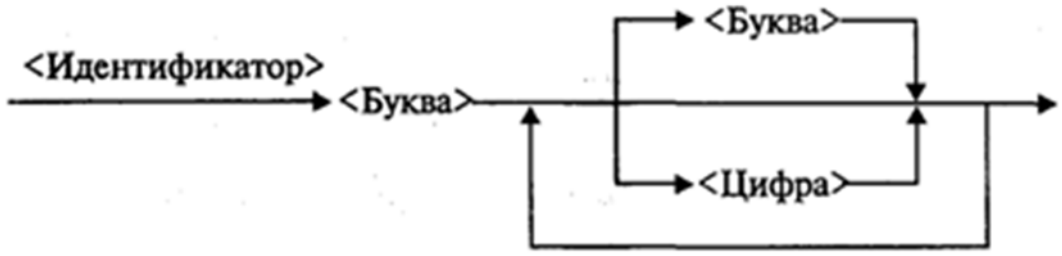


Рис. 20. Синтаксическая диаграмма идентификатора

Например, $a1$, $[\![\_h]\!]$, $b123$ - идентификаторы, а $1a$, $\phi 2$ - нет.

Длина идентификатора может быть произвольной, но значащими являются только первые 63 символа (рис. 20).

С каждым идентификатором связана область действия идентификатора.

Следующие слова являются ключевыми, служат для оформления конструкций языка и не могут использоваться как идентификаторы:

and	array	as	auto	begin
case	class	const	constructor	destructor
div	do	downto	else	end
event	except	file	final	finalization
finally	for	foreach	function	goto
if	implementation	in	inherited	initialization
interface	is	label	lock	mod
nil	not	of	operator	or
procedure	program	property	raise	record
repeat	set	shl	shr	sizeof
template	then	to	try	type
typeof	until	uses	using	var
where	while	with	xor	

Ряд слов являются контекстно ключевыми (они являются ключевыми только в некотором контексте):

abstract	default	external	forward	internal
on	overload	override	params	private
protected	public	read	reintroduce	unit
virtual	write			

Контекстно ключевые слова могут использоваться в качестве имён.

Некоторые ключевые слова совпадают с важнейшими именами платформы .NET. Поэтому в $PascalABC.NET$ предусмотрена возможность использовать эти имена без конфликтов с ключевыми словами.

Первый способ состоит в использовании квалифицированного имени.

Пример 4.1.

```
var a: System.Array;
```

В этом контексте слово $Array$ является именем внутри пространства имён $System$, и конфликта с ключевым словом $array$ нет.

Второй способ состоит в использовании специального символа & перед именем. В этом случае имя может совпадать с ключевым словом.

Пример 4.2.

```
uses System;
var a: &Array;
```

4.1.3. Комментарий

Комментарии – это участки кода, игнорируемые компилятором и используемые программистом для пояснения текста программы.

В $PascalABC.NET$ имеется несколько типов комментариев.

Последовательность символов между фигурными скобками { } или символами $(*$ и $*)$ считается комментарием:

```
{ Это
комментарий }
(* Это
тоже комментарий *)
```

Комментарием также считается любая последовательность символов после символов // и до конца строки:

```
var Version: integer; // Версия продукта
```

Комментарии разных типов могут быть вложенными:

```
{ Это ещё один
(* комментарий *)}
```

4.1.4. Структура программы на Pascal

В целом программа на языке *Pascal* состоит из двух основных частей: описания всех данных, с которыми производятся действия, и описания самих действий (табл. 6). Кроме этого, в самом начале программы может присутствовать ее название – заголовок, который рассматривается как комментарий. В самом конце программы ставится точка.

Таблица 6

Общая структура программы на Pascal

<table>
<tr><th colspan="2">Блок структуры, описание</th><th>Комментарий</th></tr>
<tr><td colspan="2">$PROGRAM$ имя программы;</td><td>Имя программы пишется английскими буквами в одно слово в соответствии с правилами написания идентификаторов. Первая строка называется заголовком программы и не является обязательной</td></tr>
<tr><td colspan="2">$USES$ подключаемые библиотеки (модули);</td><td>Раздел $uses$ начинается с ключевого слова $uses$, за которым следует список имён модулей и пространств имён $.NET$, перечисляемых через запятую[3].</td></tr>
<tr><td rowspan="5">Раздел описаний. Данные подразделы следуют друг за другом в произвольном порядке.</td><td>$LABEL$ список меток;</td><td>Из одного места программы «прыгать» в другое</td></tr>
<tr><td>$CONST$ раздел описания констант;</td><td>Постоянные величины, их нельзя изменять</td></tr>
<tr><td>VAR определение глобальных переменных;</td><td>Описание всех переменных величин, которые в программе могут изменяться</td></tr>
<tr><td>$TYPE$ описание типов переменных;</td><td></td></tr>
<tr><td>ОПРЕДЕЛЕНИЕ ПРОЦЕДУР;</td><td></td></tr>
<tr><td colspan="2">ОПРЕДЕЛЕНИЕ ФУНКЦИЙ;</td><td></td></tr>
<tr><td colspan="2">$BEGIN$
основной блок программы[4]
$END.$</td><td>Внутри блока находятся операторы, отделяемые один от другого символом «точка с запятой». Среди операторов может присутствовать оператор описания переменной, который позволяет описывать переменные внутри блока.</td></tr>
</table>

[3] Раздел $uses$ и раздел описаний могут отсутствовать, подробнее см. в разделе 4.1.9.

[4] Хороший тон – в этом блоке команды формировать с отступом от левой границы.

Пример 4.3.

```
program MyProgram;
var
   a,b: integer;
   x: real;
begin
   readln(a,b);
   x := a/b;
   writeln(x);
end.
```

или

```
uses GraphABC;
begin
   var x := 100;
   var y := 100;
   var r := 50;
   Circle(x,y,r);
end.
```

4.1.5. Раздел описания меток

Раздел описания меток начинается с зарезервированного слова $label$, после которого следует список меток, перечисляемых через запятую. В качестве меток могут быть использованы идентификаторы и положительные целые числа:

```
label a1, l2, 777777;
```

Метки используются для перехода в операторе $goto$.

4.1.6. Раздел описания констант

Раздел описания именованных констант начинается со служебного слова $const$, после которого следуют элементы описания вида:

```
имя константы = значение;
или
имя константы : тип = значение;
```

Пример 4.4.

```
const
        Pi = 3.14;
        Count = 10;
        Name = 'Mike';
```

```
DigitsSet = ['0'..'9'];
Arr: array [1..5] of integer = (1,3,5,7,9);
Rec: record name: string;
age: integer end = (name: 'Иванов'; age: 23);
Arr2: array [1..2,1..2] of real = ((1,2),(3,4));
```

4.1.7. Раздел описания переменных

Переменная - в языках программирования - именованная часть памяти, в которую могут помещаться разные значения. Причём в каждый момент времени переменная имеет единственное значение. В процессе выполнения программы значение переменной может изменяться.

Тип переменных определяется типом данных, которые они представляют.

В программировании переменная ($variable$) – это своего рода ёмкость для хранения данных. Когда информация записана в переменной (или по-другому, когда переменной присвоено значение), тогда эту информацию можно изменять, выводить, преобразовывать и т.д.

Такого типа переменная универсальна:

- в ней можно хранить информацию;
- можно из неё извлекать информацию, что не повлияет на значение самой переменной;
- в неё можно записать новые данные.

Причём эти действия можно выполнять практически сколько угодно раз. Из названия ясно, что переменная – вещь непостоянная. Переменные существуют или содержат в себе значение исключительно во время работы программы. Как только завершается выполнение программы – существование переменных прекращается.

Переменные появились с первыми языками программирования. Результат работы любой программы сводится к выполнению действий над какими-либо данными. Напомним, что память ($memory$) – это последовательность байтов ($byte$), каждый из которых принимает значения от 0 до 255. Так как байтов неимоверно много, единствен-

ный способ различать их – это присвоение каждому из них порядкового номера. Каждый байт оперативной памяти доступен процессору через его порядковый номер. Этот порядковый номер называется ***адресом байта***.

Во времена, когда программы писались на машинном коде, программист должен был запоминать в какой участок памяти он записал нужное значение. Представьте, как усложнялся процесс написания программы, когда возникала необходимость работы с несколькими значениями. Адрес байта памяти есть число, которое мало, о чём говорит. Большой объем памяти создаёт трудности программисту.

С первыми языками программирования появилась полезная возможность связывания определённого участка оперативной памяти с символьным названием (набором символов). По сравнению с адресом название переменной может отражать содержимое этого участка памяти. Но имя переменной – не единственная вещь, которая определяет переменную. Процессор может обрабатывать три вида данных: байт, слово и двойное слово[5]. Поэтому определение вида переменной в языках нижнего и среднего уровней происходит обычно указанием типа переменной. Эти два свойства переменной (название и тип) определяют нужный участок памяти и способ его использования. В большинстве случаев именно тип переменной определяет, сколько байтов памяти захватит переменная. Например, переменной типа $BYTE$ присвоено имя A. Процессор по названию переменной (A) определяет её место в памяти и при извлечении значения этой переменной воспользуется командой, предназначенной для извлечения байта (не слова и не двойного слова).

В общем случае переменная – это поименованный участок оперативной памяти, используемый для временного хранения данных. В зависимости от языка программирования, объявление переменной может сопровождаться указанием его типа.

В современном мире программирования программист должен знать не только имя и тип переменной. Также существуют понятия

---

[5] Термины «слово» (2 байта) и «двойное слово» (4 байта) здесь используются в узкоспециальном смысле, отражая собой размер участка памяти, выделенного под переменную.

пространства имён и область действия переменной. Представьте, что создаётся программа, в которой используются несколько переменных. Имена этих переменных составляют список, который определяет пространство имён. Представим, что в ходе создания программы мы, по ошибке, объявили две переменные с одинаковыми названиями. При попытке запуска программы его компилятор сообщит об этой ошибке. Это было бы невозможно, если бы компилятор не контролировал переменные. То есть контроль безупречности пространства имён возлагается от программиста на компилятор; что облегчает процесс создания и отладки программы. На практике, приведённый пример не во всех языках приводит к ошибке. Каждый компилятор (или интерпретатор) имеет собственные требования к пространству имён. То, что является ошибкой в одном языке, в других языках ошибкой может и не быть.

Если раньше программы были небольшие и их исходный код располагался в одном файле, то сейчас текст кода может состоять из нескольких файлов. И запомнить уникальное имя каждой переменной, использующейся в программе, становится практически невозможным. Поэтому (и не только) было введено понятие «область действия» (или «область существования») переменных. Область действия – понятие абстрактное. Оно применяется только в языках среднего и высокого уровней. Целью применения области действия является разделение пространства имён на несколько независимых частей. Это означает, что переменная, объявленная в одном файле, может быть абсолютно недоступна в других файлах. Например, мы можем объявить переменную с именем $MyVar$ в нескольких файлах проекта, и это не будет ошибкой.

Таким образом, переменная в программировании обладает следующими характеристиками:

- имя;
- адрес;
- тип;
- размер, который обычно определятся типом;
- принадлежность какому-либо пространству имён;
- область действия.

Переменные могут быть описаны в разделе описаний, а также непосредственно внутри любого блока $begin/end$.

Раздел описания переменных начинается с ключевого слова var, после которого следуют элементы описания вида:

```
список имен: тип;
или
имя: тип := выражение;
или
имя: тип = выражение; // для совместимости с Delphi
или
имя := выражение;
```

Имена в списке перечисляются через запятую.

Пример 4.5.

```
var
   a,b,c: integer;
   d: real := 3.7;
   s := 'PascalABC forever';
   al := new List<integer>;
   p1 := 1;
```

В последних трёх случаях тип переменной автоматически определяется по типу правой части. Автовыведение типа активно используется при инициализации переменной вызовом конструктора или функции, возвращающей объект:

```
begin
      var l := new List<integer>;
      var a := Seq(1,3,5);                тип a выводится по типу возвращаемого значения Seq: array of integer
end.
```

Автовыведение типа при описании невозможно при инициализации переменной лямбда-выражением:

```
var f := x -> x*x;                                  так нельзя!
var f : Func<integer,integer> := x -> x*x;
```

Внутриблочные описания используются, чтобы не захламлять раздел описаний описанием вспомогательных переменных. Кроме этого, внутриблочные описания позволяют вводить переменные именно в тот момент, когда они впервые потребовались. Оба этих фактора существенно повышают читаемость программы.

Кроме того, переменные-параметры цикла могут описываться в заголовке операторов for и $foreach$.

4.1.8. Раздел описания типов

Тип данных может быть либо описан непосредственно в разделе описания переменных, либо определяться идентификатором типа. Стандартные типы не требуют описания, в отличие от типов, определенных пользователем. Строго говоря, синтаксис языка *Pascal* не требует обязательного определения идентификатора типа и в последнем случае, так как тип можно задать перечислением в разделе описания переменных. Выбор описания типа зависит, таким образом, только от программиста и специфики программы. Раздел описания типов данных начинается зарезервированным словом *type*, за которым следуют одно или несколько определений типов, разделенных точкой с запятой.

Type <имя типа> = тип [<значение типа>];

Пример 4.6.

```
type
   arr10 = array [1..10] of integer;
   myint = integer;
   pinteger = ^integer;
   IntFunc = function(x: integer): integer;

type
   LatLetter = ('A'..'z');
   Days = 1..31:
   Matr = array[1..10] of integer;
```

Каждое описание задает множество значений и связывает с этим множеством некоторое имя типа. Например, в данном описании тип *LatLetter* определяет множество букв латинского алфавита, *Days* – множество целых чисел от 1 до 31, *Matr* – массив из 10 целых чисел.

4.1.9. Раздел USES

Раздел ***uses*** состоит из нескольких подряд идущих секций ***uses***, каждая из которых имеет вид:

uses список имен;

Имена в списке перечисляются через запятую и могут быть либо именами подключаемых внешних модулей *PascalABC.NET*, либо пространствами имен *.NET*. Например:

uses System, System.Collections.Generic, MyUnit;

Здесь *MyUnit* - модуль *PascalABC.NET*, представленный в виде исходного текста или откомпилированного *.pcu*-модуля, *System* и *System.Collections.Generic* - пространства имен *.NET*.

В модуле или основной программе, которая содержит раздел *uses*, можно использовать все имена из подключаемых модулей *PascalABC.NET* и пространств имен *.NET*. Основное отличие между модулями и пространствами имен *.NET* состоит в том, что модуль содержит код, а пространства имен *.NET* содержат лишь имена - для использования кода его необходимо подключить с помощью директивы компилятора *{$reference ИмяСборки}*, где *ИмяСборки* - имя *dll*-файла, содержащего *.NET*-код. Другое не менее важное отличие состоит в том, что в модуле или основной программе нельзя использовать имена, определенные в другом модуле, без подключения этого модуля в разделе *uses*. Напротив, если сборка *.NET* подключена директивой *$reference*, то можно использовать ее имена, явно уточняя их пространством имен, не подключая это пространство имен в разделе *uses*. Например:

```
begin
  System.Console.WriteLine('PascalABC.NET');
end.
```

По умолчанию в первой секции *uses* неявно первым подключается системный модуль *PABCSystem*, содержащий стандартные константы, типы, процедуры и функции. Даже если раздел *uses* отсутствует, модуль *PABCSystem* подключается неявно. Кроме того, по умолчанию с помощью неявной директивы *$reference* подключаются сборки *System.dll, System.Core.dll* и *mscorlib.dll*, содержащие основные *.NET*-типы.

Поиск глобальных имен осуществляется вначале в текущем модуле или основной программе, затем во всех подключенных модулях и пространствах имен, начиная с самого правого в секции *uses* и заканчивая самым левым. При этом считается, что пространство имен более правого модуля вложено в пространство имен более левого.

Таким образом, конфликта имен не происходит. Если необходимо использовать имя из конкретного модуля или пространства имен, то следует использовать запись

Например, в программе:

```
uses unit1, unit2;
begin
  id := 2;
end.
```

описание переменной *id* будет искаться вначале в основной программе, затем в модуле *unit2*, затем в модуле *unit1*. При этом в разных модулях могут быть описаны разные переменные *id*. Данная ситуация означает, что *unit1* образует внешнее пространство имен, пространство имен *unit2* в него непосредственно вложено, а пространство имен основной программы вложено в *unit2*.

Если в последнем примере оба модуля – *unit1* и *unit2* – определяют переменные *id*, то рекомендуется уточнять имя переменной именем модуля, используя конструкцию *ИмяМодуля.Имя*:

```
uses unit1, unit2;
begin
  unit1.id := 2;
end.
```

В качестве имени модуля может выступать также имя основной программы, если у нее присутствует заголовок *program*.

Во многих случаях стандартные для *PascalABC.NET* возможности ввода/вывода данных с помощью стандартных процедур оказываются недостаточными для разработки удобных в использовании диалоговых программ. В библиотеке (модуле) *CRT* предусмотрено несколько подпрограмм, существенно увеличивающих возможности текстового ввода/вывода (табл.7).

Таблица 7

Команды модуля CRT

Оператор	Назначение	Пример
WINDOW	Процедура. Установить границы текстового окна, относительно левого верхнего угла экрана. Форма записи: WINDOW(X1,Y1,X2,Y2), где X1, Y1 – координаты левого верхнего угла; X2, Y2 – координаты правого нижнего угла.	WINDOW(1,1,10,5);
CLRSCR	Процедура. Очистить экран или текущее текстовое окно, помещая курсор в левый верхний угол (координаты 1,1) и закрасить текущим цветом фона.	CLRSCR;
CLREOL	Процедура. Стирает все символы до конца строки, начиная с позиции курсора.	CLREOL;
INSLINE	Процедура. Вставить пустую строку, все нижестоящие строки перемещаются на одну позицию вниз.	INSLINE;
DELLINE	Процедура. Стереть строку, на которой находится курсор, все нижестоящие строки перемещаются на одну позицию вверх.	DELLINE;
TEXTCOLOR	Процедура. Установить цвет выводимых символов. Форма записи: TEXTCOLOR(COLOR), где COLOR – цвет выводимых символов (диапазон 0 – 15).	TEXTCOLOR(1);
TEXTBACKGROUND	Процедура. Установить цвет фона выводимых символов. Форма записи: TEXTBACKGROUND(COLOR), где COLOR – цвет фона (диапазон 0 – 7).	TEXTBACKGROUND(3);
GOTOXY	Процедура. Установить позицию курсора, относительно левого верхнего угла активного текстового окна. Форма записи: GOTOXY(X,Y) X, Y – координаты.	GOTOXY(2,4);
WHEREX	Функция целого типа. Узнать позицию курсора по X.	X:=WHEREX;

Продолжение табл. 7

Оператор	Назначение	Пример
WHEREY	Функция целого типа. Узнать позицию курсора по Y.	Y:=WHEREY;
KEYPRESSED	Функция логического типа. Возвращает значение типа BOOLEAN, указывающее состояние буфера клавиатуры: FALSE означает, что буфер пуст, а TRUE – что в буфере есть хотя бы один символ, ещё не прочитанный программой. Обращение к функции не задерживает исполнение программы.	REPEAT WRITE('SLOVO') UNTIL KEY-PRESSED;
READKEY	Функция символьного типа, возвращает значение типа CHAR. Останавливает выполнение программы до нажатия любой клавиши и возвращает её код.	A:=READKEY;
DELAY	Процедура. Установить задержку работы программы на заданный интервал времени. Форма записи: DELAY(I), где I – выражение типа WORD, определяющее интервал времени.	DELAY(500);
SOUND	Процедура. Выдать звук заданной частоты. Форма записи: SOUND(I), где I – выражение типа WORD, частота звука в герцах.	SOUND(1500);
NOSOUND	Процедура. Выключить звук.	NOSOUND;

4.2. Структура программы на VBA

В окне $Project\ Explorer$ (см. раздел 2.3.1) представлено дерево компонентов вашего приложения VBA.

Самый верхний уровень – это проект ($Project$), которому соответствует документ $Word$, рабочая книга $Excel$, презентация $PowerPoint$ и прочие файлы, с которыми работает данное приложение. Например, если вы открыли редактор $Visual\ Basic$ из $Word$, то в $Project\ Explorer$ будут представлены все открытые в настоящее время файлы $Word$ и ещё шаблон $Normal.dot$. Если редактор $Visual\ Basic$ открыт из $Excel$, то в $Project\ Explorer$ будут открытые книги $Excel$ и специальная скрытая книга $PERSONAL.XLS$.

Помимо того, что обычно содержится в документах $MS\ Office$ (текст, рисунки, формулы и т.п.), каждый проект (который и есть документ) – это одновременно и контейнер для хранения стандартных модулей, модулей классов, и пользовательских форм. Добавить в проект каждый из этих компонентов можно при помощи меню $Insert$ или через контекстное меню в $Project\ Explorer$. Модули добавляются в проект командой $Insert - Module$ (Вставить – *Модуль*). Формы создаются командой $Insert - UserForm$ (Вставить – *UserForm*), а модули класса – командой $Insert - Class\ Module$ (Вставить – *Модуль класса*).

Стандартные модули – это просто блоки с текстовым представлением команд VBA. В нем может быть только два раздела:

- раздел объявлений уровня модуля (объявление переменных и констант уровня модуля);
- раздел методов модуля (расположение процедур и функций).

При работе макрорекордера в $Word$ в проекте $Normal.dot$ или текущем документе (в зависимости от места сохранения макроса) автоматически создается стандартный модуль $NewMacros$ (в $Excel$ – $Module1$), куда и записываются все создаваемые макрорекордером макросы.

В большинстве проектов VBA используется только один стандартный модуль, куда и записывается весь код. Создавать новые стандартные модули есть смысл только из следующих соображений:

- для удобства экспорта и импорта (из контекстного меню в $Project\ Explorer$). Так можно очень удобно обмениваться блоками кода между приложениями VBA (и обычного VB);
- для повышения производительности. При вызове любой процедуры модуля происходит компиляция всего модуля, поэтому иногда выгоднее разместить процедуры в разных модулях, чтобы компилировать только нужный в данный момент код;
- для улучшения читаемости. Если ваше приложение выполняет разные группы задач, то код, относящийся к каждой группе, лучше поместить в свой модуль.

Модули классов позволяют создавать свои собственные классы – чертежи, по которым можно создавать свои собственные объекты. Обычно используются только в очень сложных приложениях. Применяются они в обычных приложениях VBA редко и здесь рассматриваться не будут.

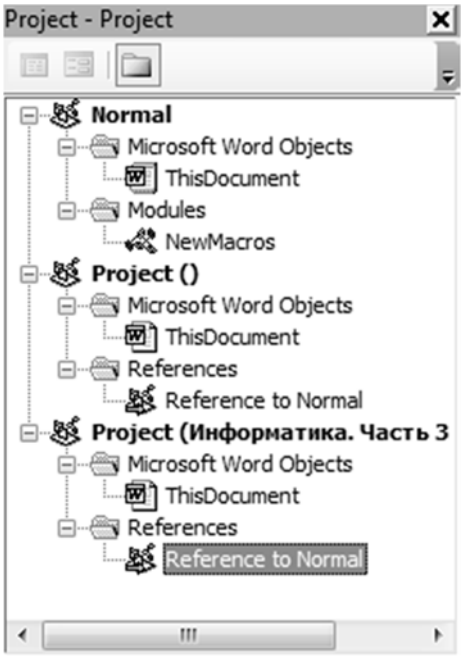


Рис. 21. Контейнеры References

Пользовательские формы являются одновременно хранилищем элементов управления и программного кода, который относится к ним, самой форме и происходящими с ними событиями.

Ещё один важный контейнер в $Project\ Explorer$ – контейнер $References$ (в $Excel$ его нет) (рис. 21), то есть контейнер ссылок. В нем показывается, ссылки на какие другие проекты (документы $Word$) есть в нашем проекте и соответственно какие «чужие» программные модули мы можем использовать. По умолчанию в каждый проект $Word$ помещается ссылка на $Normal$ (то есть шаблон $normal.dot$) – и вы в любом файле можете использовать макросы оттуда.

Обратите внимание, что в этом контейнере, – только ссылки на другие документы.

Ещё одна полезная возможность $Project\ Explorer$ – возможность настроить свойства проекта. Для этого нужно щёлкнуть правой кнопкой мыши по узлу $Project$ ($VBAProject$ в $Excel$) и в контекстном

меню выбрать $Project\ Properties$ (то же окно можно открыть и через меню $Tools \rightarrow Project\ Properties$. В этом окне можно:

- изменить имя проекта. Эта возможность потребуется, если у вас есть ссылки на проект с одинаковыми именами;
- ввести описание проекта, информацию о файле справки и параметры, которые будут использоваться компилятором;
- защитить проект, введя пароль. Не зная этот пароль, проект нельзя будет просмотреть или отредактировать.

И все-таки, что обычно приходится делать в окне $Project\ Explorer$?

Если вам нужно создать свой макрос вручную, а макросов в данном документе ещё нет, то нужно будет щёлкнуть правой кнопкой мыши по узлу проекта (строке, выделенной полужирным цветом) и в контекстном меню дать команду $Insert \rightarrow Module$. В проекте будет создан новый модуль и сразу открыт в окне редактора кода.

Если вы уже создавали макросы в этом проекте (макрорекордером или вручную), то модуль будет уже создан. Его можно будет увидеть под контейнером $Modules$. Чтобы его открыть в окне редактора кода, достаточно щелкнуть по нему два раза мышью. Там можно будет найти созданные вами средствами макрорекордера макросы.

ВАЖНО: Обязательно подумайте, где вам будет нужен создаваемый код – только в одном документе или во всех документах данного приложения. Если он нужен будет только в одном документе, то используйте стандартный программный модуль этого документа. Если во всех, то используйте программные модули проекта $Normal$ (в $Word$) или $PERSONAL.XLS$ (в $Excel$).

Если вам нужно создать графическую форму с элементами управления (кнопками, текстовыми полями, ниспадающими списками и т.п.), то нужно щёлкнуть правой кнопкой мыши по узлу проекта и в контекстном меню выбрать $Insert \rightarrow UserForm$. Новая форма будет создана и открыта в окне дизайнера форм.

К основным понятиям языка VBA относятся: переменные, массивы, константы, операторы, процедуры, функции и т. п.

Данными (data) называются объекты, обрабатываемые программой.

Для хранения временных значений (данных) используются переменные.

Переменной (variable) называется имя, определяющее область памяти для хранения величины, которая может изменяться во время выполнения программы.

Каждая переменная имеет имя, значение и характеризуется типом.

Идентификатор (identifier) – это имя, которое дается элементам в создаваемых процедурах и модулях таким же, как переменным.

Правила выбора имени переменной (идентификатора):

- начинается с буквы, за которой может следовать любая комбинации букв, цифр и символов подчёркивания "_";
- не может превышать 255 символов;
- не может совпадать с ключевыми словами и именами стандартных объектов *VBA*;
- должно быть уникальным в рамках его области действия (*scope*), т.е. в пределах процедуры или модуля, в котором объявлена данная переменная (об области действия переменных будет сказано ниже).

Имена переменных не «чувствительны» к состоянию регистра.

Переменные могут хранить любые типы данных, определённых в VBA.

Константы (constant) – еще один контейнер для хранения данных, но, в отличие от переменных, они не изменяются в ходе выполнения *VBA*-программы. Для чего нужны константы:

- код становится лучше читаемым, убираются потенциальные ошибки;
- чтобы изменить какое-либо значение, которое много раз используется в программе (например, уровень налога), – это можно сделать один раз.

Тип данных (data type) – это характеристика определённого вида данных, которые VBA сохраняет и которыми может манипулировать.

Тип данных определяет:

- формат (размер) хранения данных;
- диапазон допустимых значений данных;
- операции, которые могут выполняться над данными.

В табл. 5 приведена краткая сводка основных типов данных, используемых VBA.

Структурная схема описания переменных и констант приведена ниже (рис. 22).

Рис. 22. Структурная схема терминов

VBA не требует обязательного объявления переменных. Допускается использование переменных без явного описания. VBA создаёт переменную и резервирует память для её хранения, когда эта переменная в первый раз появляется в каком-либо операторе VBA (обычно в операторе присваивания). Создание переменной путём её использования в операторе называется *неявным объявлением переменной*. В случае если переменная не была объявлена, ей автоматически присваивается тип $Variant$. Этот тип является универсальным и может содержать данные различных подтипов: $Integer$, $Long$, $String$ и т.п. Неявное объявление переменных известно так же, как объявление переменных «*на лету*» ($on-the-fly$).

VBA предоставляет возможность выполнять явное объявление переменных, которое имеет много преимуществ:

- Ускоряется выполнение кода. Скорость выполнения кода увеличивается на то количество времени, которое необходимо для анализа и создания неявно объявленных переменных.
- Уменьшается количество ошибок в результате неправильного написания имени переменной.
- Код становится более понятным. Видя все объявления переменных в начале модуля или процедуры, легко определить, какие переменные используются в этом модуле или процедуре и др.

Для явного объявления переменных используется оператор Dim со следующим форматом:

```
Dim     <имяПеременной1> [As <типДанных>],
        [<имяПеременной2>] [As <типДанных>]
```

- $\text{имяПеременной}n$ – это любой допустимый идентификатор переменной;
- типДанных – это любой тип данных, поддерживаемый VBA.

Пример 4.7.

```
Dim A As Integer
Dim B, C, D As Single
Dim E As Variant
Dim F
Dim G As Double, H As String
```

В результате такого объявления переменные будут иметь следующие типы: $A - Integer$, B, C, E и $F - Variant$, $D - Single$, $G - Double$, $H - String$.

Переменные типа $Variant$ могут получать значения любого типа в зависимости от контекста. Кроме того, они могут принимать и некоторые специальные значения:

- $Empty$ – переменная не была инициализирована;
- $Null$ – данные ошибочны;
- $Error$ – значение содержит код ошибки, который может быть использован для её обработки;
- $Nothing$ – переменная типа $Object$ ни на что не ссылается: связь между ней и конкретным объектом прервана или не установлена.

Переменные можно явно объявить и использовать, а можно использовать и без объявления.

В Листинг 1 приведена процедура $HelloWorld$, демонстрирующая явное объявление переменной. Заметим, хорошим тоном среди программистов считается правило ***структурировать исходный код***.

Листинг 1. Процедура $\boldsymbol{HelloWorld}$ *с явным объявлением переменной:*

```
1: Sub HelloWorld()
2:         Dim HelloMsg ‘ переменная для MsgBox
3:         HelloMsg = "Здравствуй, мир!"
4:         MsgBox HelloMsg, , "Окно приветствия"
5: End Sub
```

Исходный код процедуры в реальных модулях VBA не включает номер каждой строки; в этом учебном пособии в некоторых листингах приведены номера строк, чтобы было легче обозначать и объяснять отдельные строки кода.

Разберём подробнее этот листинг (рис. 23).

В строке 1 объявляется процедура с именем $HelloWorld$. Объявление процедуры производится при помощи служебного слова Sub ($subroutine$, $subprogram$). Оператор Dim (строка 2) объявляет переменную $HelloMsg$ типа $Variant$. В строке 3 выполняется присваивание значения переменной $HelloMsg$. В строке 4 переменная $HelloMsg$ используется в качестве первого аргумента процедуры $MsgBox$ – это текст сообщения, выводимого на экран в диалоговом окне. Второй аргумент процедуры $MsgBox$ пропущен, а в третьем задан заголовок диалогового окна ("Окно приветствия").

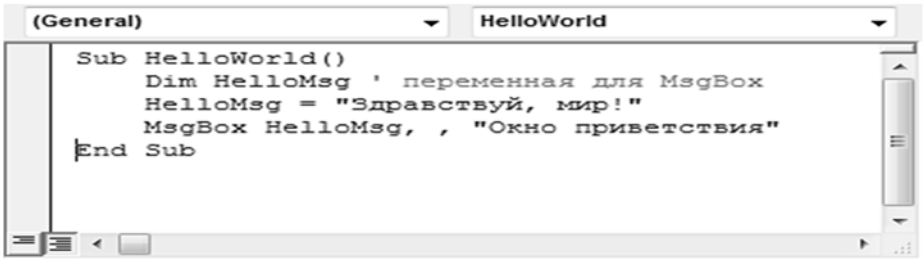


Рис. 23. Листинг

В результате выполнения процедуры $HelloWorld$ на экране появится диалоговое окно, показанное на рис. 24.

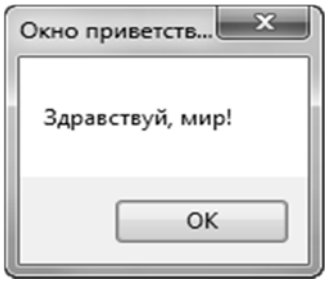


Рис. 24. Окно сообщения

Явно объявить переменную можно как в начале блока, так и в том произвольном месте, где возникла необходимость использовать новую переменную. Лучше придерживаться стратегии, когда все переменные объявляются явно и, как правило, в начале блока. Можно принудительно заставить себя следовать этой стратегии, вставив в начало модуля оператор $Option\ Explicit$ (Опция «Явно») (рис. 25). При этом объявление переменных становится обязательным. Оператор $Option\ Explicit$ должен быть расположен в самом начале модуля – до того, как начнется первая процедура этого модуля.

При отсутствии явного объявления переменной компилятор выдаст соответствующее сообщение и выполнение программы прекратиться до исправления выявленного несоответствия.

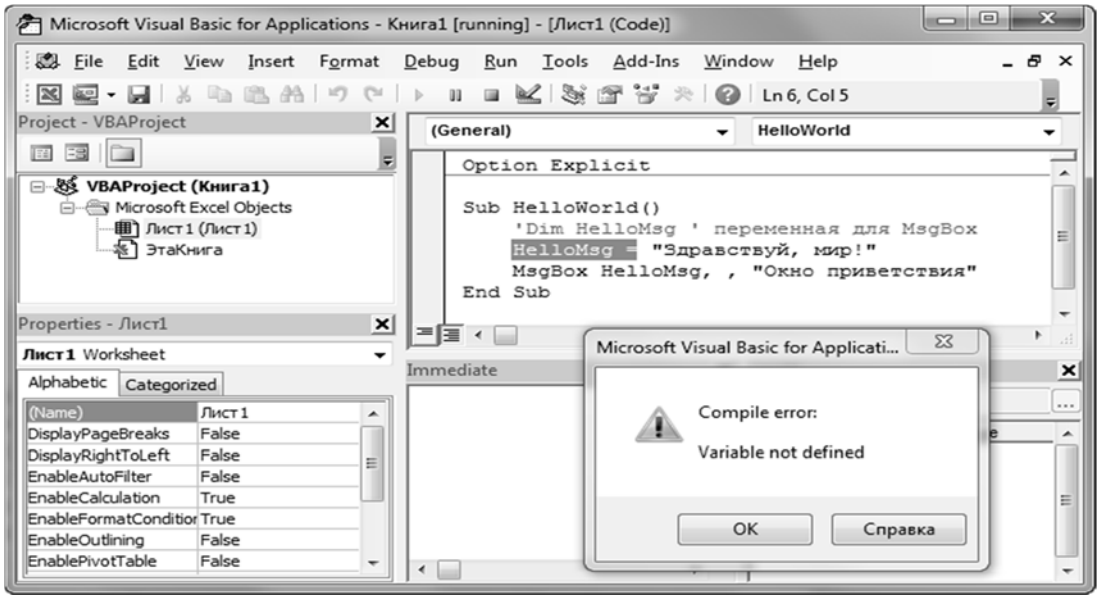


Рис. 25. Результат работы Option Explicit

4.2.1. Процедуры и функции

Процедуры – это самые важные функциональные блоки языка VBA. В VBA вы можете выполнить только программный код, который содержится в какой-либо процедуре – обычной в стандартном модуле, либо событийной для элемента управления на форме и т.п. Иногда начинающие пользователи пытаются записать команды прямо в область объявлений стандартного модуля и не могут понять, почему они не выполняются (сообщений об ошибке при этом не выдаётся – просто этот код становится «невидим» для компилятора). Причина проста – в разделе объявлений модуля (когда в верхних списках показываются значения ($General$) и ($Declarations$) могут быть только объявления переменных уровня модуля и некоторые специальные инструкции для компилятора. Весь остальной программный код должен находится внутри процедур.

В VBA предусмотрены следующие типы процедур:

Процедура типа Sub (подпрограмма) – универсальная процедура для выполнения каких-либо действий:

```
Sub Farewell()
MsgBox “Goodbye”
End Sub
```

Макрос в VBA – это просто процедура типа Sub, не имеющая параметров. Только макросы можно вызывать по имени из редактора VBA или приложения $MS\ Office$. Все другие процедуры нужно вызывать либо из других процедур, либо специальными способами, о которых будет рассказано ниже (в разделе 6.2).

Процедура типа $Function$ (функция) – тоже набор команд, которые должны быть выполнены. Принципиальное отличие только одно: функция возвращает вызвавшей её программе какое-то значение, которое там будет использовано.

Итак, ещё раз подчеркнём: главное отличие между процедурами *Function* и *Sub* состоит в том, что процедура *Function* возвращает результат, процедура *Sub* – нет. Поэтому если требуется выполнить действия и получить какой-то результат (например, просуммировать несколько чисел), то обычно используется процедура *Function*, а для того, чтобы просто выполнить какие-то действия

(например, изменить форматирование группы ячеек), нужно выбрать процедуру *Sub*.

Пример 4.8.

```
Function Tomorrow()
   Tomorrow = DateAdd("d", 1, Date())
End Function
```

и пример её вызова:

```
Private Sub Test1()
   Dim dDate
   dDate = Tomorrow
   MsgBox dDate
End Sub
```

В тексте функции необходимо предусмотреть оператор, который присваивает ей какое-либо значение. В нашем случае это строка Tomorrow = DateAdd(" d", 1, Date()).

В принципе процедуры типа Sub тоже могут возвращать значения при помощи переменных. Зачем же тогда нужны функции? Все очень просто: функцию можно вставлять практически в любое место программного кода. Например, наш последний пример может выглядеть намного проще:

```
Private Sub Test 1()
MsgBox Tomorrow()
End Sub
```

Более подробно о разработке пользовательских процедур и функции можно прочитать в главе 6.

В VBA предусмотрены сотни встроенных функций (и гораздо большее число предусмотрено в объектных моделях приложений $MS\ Office$). Даже в нашем примере используются две встроенные функции: $Date(\dashv)$, которая возвращает текущую дату по часам компьютера и $DateAdd(\dashv)$, которая умеет прибавлять к текущей дате определённое количество дней, недель, месяцев, лет и т.п.

В VBA имеются также процедуры обработки событий ($event\ procedure$) – процедуры типа Sub специального назначения,

которые выполняются в случае возникновении определённого события. Про события подробнее будет рассказано в разделе 6.2.

Есть ещё процедуры типа $Property$ (процедуры свойства). Они нужны для определения свойств создаваемого вами класса.

4.2.2. Редактор VBA: получение списка свойств и методов, список параметров, автоматическое дополнение слов

В редактор кода встроено множество средств, которые облегчают жизнь разработчику. Ниже перечислены наиболее важные из них.

Самое полезное средство – это получение списка свойств и методов. В большинстве VBA-программ используются свойства и методы различных объектов, при этом многие методы принимают параметры. Помнить точное название каждого свойства и метода, очерёдность передачи параметров невозможно, а разыскивать каждый раз справку по этому объекту в документации – непроизводительная трата времени. Пользоваться очень просто: если включён автоматический показ (он включён по умолчанию), то достаточно впечатать имя объекта и за ним – точку. Если автопоказ отключён, то можно воспользоваться командой $List\ Properties/Methods$ в меню $Edit$ или нажать $Ctrl + J$. Выбрав нужное свойство/метод (можно впечатать первые несколько букв или воспользоваться мышью), нужно нажать на клавишу Tab. Это средство работает и для ваших классов/переменных. Если не работает, проверить настройки параметра $Auto\ List\ Members$ в диалоговом окне $Options$ (меню $Tools \rightarrow Options$).

Получить список аргументов для метода и информацию о них можно автоматически после того, как вы напечатали имя метода, принимающего параметры. Вручную вызвать при помощи $Ctrl + I$, включить/отключить можно при помощи $Tools\ Options \rightarrow Auto\ Quick\ Info$. $Ctrl + Shift + I$ – информация о параметрах, показывает список аргументов для самой внешней функции (в случае вложенности).

Получение списка констант (то есть допустимых значений для данного свойства) также появляется автоматически после того, как вы впечатаете знак равенства $(=)$. Можно воспользоваться также комбинацией $Ctrl + Shift + J$.

Ключевые слова VBA и имена доступных в данный момент классов очень удобно вводить при помощи автоматического дополнения слов ($Complete\ Word$). Для этого достаточно нажать на $Ctrl + \text{Пробел}$. Можно предварительно ничего не печатать, а можно впечатать одну – две буквы.

Ещё несколько моментов, связанных с редактором кода:

- если вы напечатаете одну строку кода с отступом, то тот же отступ будет установлен для следующих строк. Изменить поведение можно при помощи параметра $Auto\ Indent$ в том же диалоговом окне $Options$;
- если редактор кода распознает ключевое слово, он автоматически делает его первую букву заглавной и выделяет синим цветом;
- часто бывает необходимо закомментировать или раскомментировать несколько строк сразу. Для этой цели можно включить отображение панели инструментов $Edit$ и воспользоваться кнопками $Comment\ Block$ и $Uncomment\ Block$;
- если при создании процедуры вы пишете ключевое слово Sub или $Function$, то редактор кода автоматически дописывает оператор $End\ Sub$ или $End\ Function$. Между процедурами вставляется строка-разделитель;
- если при переходе на новую строку редактор кода обнаружит синтаксическую ошибку, то вам будет выдано предупреждение. Отменить протесты редактора можно, сняв флажок $Auto\ Syntax\ Check$ в том же диалоговом окне $Options$. Работе это сильно не помешает, потому что синтаксически неверные строки в любом случае будут автоматически выделяться красным цветом;
- в редакторе $Visual\ Basic$ вполне допускается работа сразу с несколькими окнами редактирования кода. Переход между ними – по $Ctrl + Tab$ или $Ctrl + F6$;
- по умолчанию редактор кода работает в режиме $Full\ Module\ View$ который показывает содержимое всего модуля. Если вы хотите просматривать процедуры по отдельности, то переключитесь в режим $Procedure\ View$. Кнопки для переключения – в левом нижнем углу окна редактора кода.

4.2.3. Разделы справки VBA, приёмы нахождения нужной информации

Работа со справкой по программированию в $MS\ Office$ не так очевидна, как может показаться на первый взгляд.

Вызов окна справки производится из редактора кода $Visual\ Basic$ при нажатии на кнопку $F1$. Второй вариант – воспользоваться кнопкой *Справка* на панели инструментов $Standard$.

Ещё одна возможность вызвать справку – установить указатель мыши в нужное место в окне редактора кода (например, на имя вызываемого метода или используемого свойства) и нажать на кнопку $F1$. Преимуществом такого подхода является то, что при наличии нескольких вариантов (например, объект $Range$ и свойство $Range$) вам автоматически открывается нужный.

Справка по программированию в приложении $Microsoft\ Office$ обычно состоит из трёх частей:

- первая часть *Microsoft Excel Visual Basic Reference, Microsoft Word Visual Basic Reference* и т.п.) – это справка по объектной модели самого приложения $MS\ Office$;
- вторая часть ($Microsoft\ Visual\ Basic\ Documentation$, она одинакова во всех приложениях $MS\ Office$) – это справка по синтаксису и встроенным функциям самого языка $Visual\ Basic\ for\ Application$;
- третья часть ($Microsoft\ Office\ Visual\ Basic\ Reference$, она также одинакова во всех приложениях $MS\ Office$) – это справка по общим возможностям приложений $MS\ Office$: программная работа с панелями инструментов и меню, работа с помощником, организация взаимодействия с $Windows\ SharePoint\ Services$ и т.п.

В некоторых приложениях (например, $Microsoft\ Access$) в справку добавлены дополнительные части – по объектной модели ADO, по языку SQL и т.п.

Обычно самая важная часть – это часть, которая посвящена возможностям конкретного приложения $MS\ Office$. Её условно можно разделить на две главные части:

- $Programming\ Concepts$ (концепции программирования) – в ней рассказывается, как программным образом выполнять самые распространённые операции. Например, для $Excel$ это возможность создать или открыть рабочую книгу, найти нужный лист, получить или записать информацию в ячейку и т.п.
- справка по компонентам объектной модели приложения $MS\ Office$: коллекциям, объектам, свойствам и методам и т.п. При этом самые важные моменты, которые относятся скорее к области концепций (какими способами, например, можно создать объект $Range$ в $Excel$) приводятся в справке по соответствующему объекту. Представление обо всех функциональных возможностях данного объекта можно получить, только просмотрев подряд все его свойства и методы.

Найти направление, то есть объект, его свойства и методы, можно тремя способами:

1. Просмотреть раздел $Programming\ Concepts$ (концепции программирования) в справке – не описана ли там наша ситуация.
2. Просто просматривать все подряд объекты, свойства и методы в справке, пытаясь догадаться, что нам может помочь. Это самый неэффективный способ, поскольку объектов в любом приложении $MS\ Office$ сотни (часто используемых – намного меньше). Однако если вам предстоит в течение долгого времени заниматься программированием в каком-либо приложении $MS\ Office$, то есть смысл потратить несколько дней, чтобы подряд читать справку по всем объектам, конспектируя самые важные моменты. Гарантируется, что вы узнаете множество таких возможностей, о которых раньше и не подозревали.
3. Наиболее разумный способ – выполнить нужные вам операции в макрорекордере и потом проанализировать созданный им код. Однако, к сожалению, гарантировать то, что макрорекордер покажет вам самый эффективный путь, невозможно.

И напомним ещё один момент: справки по *Visual Basic for Application* на русском языке, к сожалению, не существует.

Контрольные вопросы к главе 4

1. Какой алфавит используется в языках программирования?
2. Что такое идентификатор?
3. Какова максимальная длина идентификатора?
4. Для чего нужны комментарии?
5. Какова структура программы на $PascalABC.NET$?
6. Для чего используются метки?
7. Что такое переменная?
8. Что хранит переменная?
9. Каков размер переменной?
10. Как происходит автовыведение типа переменной?
11. Какова структура программы в VBA?
12. Для чего нужен раздел объявлений уровня модуля?
13. Что такое раздел методов модуля?
14. Что такое контейнер $References$?
15. Максимальная длина имени переменной в VBA?
16. Каким оператором явно объявляются переменные в VBA?
17. Для чего используются процедуры и функции в VBA?
18. Как работать в редакторе кода VBA?
19. Как пользоваться разделами справки VBA?
20. На каком языке составлены разделы справки VBA?

Глава 5. РАЗНОВИДНОСТИ СТРУКТУР АЛГОРИТМОВ

По структуре алгоритмы разделяют на линейные, разветвляющиеся и циклические.

5.1. Простые типы данных для переменных и констант (*PascalABC.NET*)

Целые типы

Ниже приводится таблица целых типов (табл. 8), содержащая также их размер и диапазон допустимых значений.

Таблица 8

Таблица целых типов (*PascalABC.NET*)

Тип	Размер, байт	Диапазон значений
Shortint	1	-128..127
Smallint	2	-32768..32767
integer, longint	4	-2147483648..2147483647
int64	8	-9223372036854775808..9223372036854775807
byte	1	0..255
word	2	0..65535
longword, cardinal	4	0..4294967295
uint64	8	0..18446744073709551615
BigInteger	Переменный	Неограниченный

Типы $integer$ и $longint$, а также $longword$ и $cardinal$ являются синонимами.

Максимальные значения для каждого целого типа определены как внешние стандартные константы: $MaxInt64$, $MaxInt$, $MaxSmallInt$, $MaxShortInt$, $MaxUInt64$, $MaxLongWord$, $MaxWord$, $MaxByte$.

Для каждого целого типа T, кроме $BigInteger$, определены следующие константы как статические члены:

$T.MinValue$ – константа, представляющая минимальное значение типа T;

$T.MaxValue$ – константа, представляющая максимальное значение типа T.

Для каждого целого типа T определены статические функции:

$T.Parse(s)$ – функция, конвертирующая строковое представление числа в значение типа T. Если преобразование невозможно, то генерируется исключение;

$T.TryParse(s, res)$ – функция, конвертирующая строковое представление числа в значение типа T и записывающая его в переменную res. Если преобразование возможно, то возвращается значение $True$, в противном случае – $False$.

Кроме того, для T определена экземплярная функция $ToString$, возвращающая строковое представление переменной данного типа.

Константы целого типа могут представляться как в десятичной, так и в шестнадцатеричной форме, перед шестнадцатеричной константой ставится знак. Диапазон шестнадцатеричных констант: $0000 0000 .. $FFFF FFFF.

Стандартные константы и переменные

Ниже приведены: в табл.9 – сводка стандартных констант и их размеры; в табл. 10 – стандартные переменные; в табл. 11 – сводка вещественных типов, используемых в *PascalABC.NET*.

Таблица 9

Стандартные константы (*PascalABC.NET*)

Имя константы и ее тип	Размерность
1	2
MaxShortInt = shortint.MaxValue;	Максимальное значение типа shortint
MaxByte = byte.MaxValue;	Максимальное значение типа byte
MaxSmallInt = smallint.MaxValue;	Максимальное значение типа smallint
MaxWord = word.MaxValue;	Максимальное значение типа word
MaxInt = integer.MaxValue;	Максимальное значение типа integer
MaxLongWord = longword.MaxValue;	Максимальное значение типа longword
MaxInt64 = int64.MaxValue;	Максимальное значение типа int64

Продолжение табл. 9

1	2
MaxUInt64 = uint64.MaxValue;	Максимальное значение типа uint64
MaxDouble = real.MaxValue;	Максимальное значение типа double
MinDouble = real.Epsilon;	Минимальное положительное значение типа double
MaxReal = real.MaxValue;	Максимальное значение типа real
MinReal = real.Epsilon;	Минимальное положительное значение типа real
MaxSingle = single.MaxValue;	Максимальное значение типа single
MinSingle = single.Epsilon;	Минимальное положительное значение типа single
Pi = 3.141592653589793;	Константа Pi
E = 2.718281828459045;	Константа E
NewLine: string;	Константа перехода на новую строку

Таблица 10

Стандартные переменные (*PascalABC.NET*)

Имя переменной	Применение
output	Стандартный текстовый файл вывода. По умолчанию связан с экраном, но может быть переназначен процедурой $Assign$
input	Стандартный текстовый файл ввода. По умолчанию связан с клавиатурой, но может быть переназначен процедурой $Assign$

Вещественные типы

Таблица 11

Таблица вещественных типов (*PascalABC.NET*)

Тип	Размер, байт	Количество значащих цифр	Диапазон значений
Real	8	15-16	$-1.8 \cdot 10^{308}$ .. $1.8 \cdot 10^{308}$
Double	8	15-16	$-1.8 \cdot 10^{308}$ .. $1.8 \cdot 10^{308}$
Single	4	7-8	$-3.4 \cdot 10^{38}$ .. $3.4 \cdot 10^{38}$
Decimal	16	30	-79228162514264337593543950335 .. 79228162514264337593543950335

Типы $real$ и $double$ являются синонимами. Самое маленькое положительное число типа $real$ приблизительно равно $5.0 \cdot 10^{-324}$, для типа $single$ оно составляет приблизительно $1.4 \cdot 10^{-45}$.

Максимальные значения для каждого вещественного типа определены как внешние стандартные константы: $MaxReal$, $MaxDouble$ и $MaxSingle$.

Для каждого вещественного типа R кроме $decimal$ определены также следующие константы как статические члены класса:

$R.MinValue$ – константа, представляющая минимальное значение типа R;

$R.MaxValue$ – константа, представляющая максимальное значение типа R;

$R.Epsilon$ – константа, представляющая самое маленькое положительное число типа R;

$R.NaN$ – константа, представляющая не число (возникает, например, при делении 0/0);

$R.NegativeInfinity$ – константа, представляющая отрицательную бесконечность (возникает, например, при делении $(-2)/0$);

$R.PositiveInfinity$ – константа, представляющая положительную бесконечность (возникает, например, при делении $2/0$).

Для каждого вещественного типа R кроме $decimal$ определены следующие статические функции:

$R.IsNaN(r)$ – возвращает $True$, если в r хранится значение $R.NaN$ и $False$ в противном случае;

$R.IsInfinity(r)$ – возвращает $True$, если в параметре r хранится значение $R.PositiveInfinity$ или $R.NegativeInfinity$ и значение $False$ в противном случае;

$R.IsPositiveInfinity(r)$ – возвращает $True$, если в параметре r хранится значение $R.PositiveInfinity$ и значение $False$ в противном случае;

$R.IsNegativeInfinity(r)$ – возвращает $True$, если в параметре r

хранится значение $R.NegativeInfinity$ и значение $False$ в противном случае.

Для каждого вещественного типа R определены следующие статические функции:

$R.Parse(s)$ – функция, конвертирующая строковое представление числа в значение типа R. Если преобразование невозможно, то генерируется исключение;

$R.TryParse(s,res)$ – функция, конвертирующая строковое представление числа в значение типа R и записывающая его в переменную res. Если преобразование возможно, то возвращается значение $True$, в противном случае $False$.

Кроме того, определена экземплярная функция $ToString$, возвращающая строковое представление переменной типа R.

Вещественные константы можно записывать как в форме с плавающей точкой, так и в экспоненциальной форме: 1.7 0.013 2.5e3 (2500) 1.4e-1 (0.14)

Логический тип

Значения логического типа $boolean$ занимают 1 байт и принимают одно из двух значений, задаваемых предопределёнными константами $True$ (истина) и $False$ (ложь).

Для логического типа определены статические методы:

$boolean.Parse(s)$ – функция, конвертирующая строковое представление числа в значение типа $boolean$. Если преобразование невозможно, то генерируется исключение;

$boolean.TryParse(s,res)$ – функция, конвертирующая строковое представление числа в значение типа $boolean$ и записывающая его в переменную res. Если преобразование возможно, то возвращается значение $True$, в противном случае $False$.

Кроме этого, определена экземплярная функция $ToString$, возвращающая строковое представление переменной типа $boolean$.

Логический тип является порядковым. В частности, $False < True$, $Ord(False) = 0$, $Ord(True) = 1$.

Символьный тип

Символьный тип $char$ занимает 2 байта и хранит $Unicode$-символ. Символы реализуются типом $System.Char$ платформы .NET.

Операция + для символов означает конкатенацию (слияние) строк. Например: $'a\hat{}' + \hat{}'b\hat{}' = (\_\hat{}'a\ b\hat{}'$. Как и для строк, если к символу прибавить число, то число предварительно преобразуется к строковому представлению:

```
var s: string := ' '+15;                    s = ' 15'
var s1: string := 15+' ';                   s = '15 '
```

Над символами определены операции сравнения $<>\ <=\ >=\ =\ <>$, которые сравнивают коды символов:

```
'a'<'b'                                     True
'2'<'3'                                     True
```

Для преобразования между символами и их кодами в кодировке $Windows\ (CP1251)$ используются стандартные функции Chr и Ord:

$Chr(n)$ – функция, возвращающая символ с кодом n в кодировке Windows;

$Ord(\mathrm{c})$ – функция, возвращающая значение типа $byte$, представляющее собой код символа с в кодировке Windows.

Для преобразования между символами и их кодами в кодировке $Unicode$ используются стандартные функции $ChrUnicode$ и $OrdUnicode$:

$ChrUnicode(w)$ – возвращает символ с кодом w в кодировке $Unicode$;

$OrdUnicode(\mathrm{c})$ – возвращает значение типа $word$, представляющее собой код символа с в кодировке $Unicode$.

Кроме того, выражение #число возвращает $Unicode$-символ с кодом «число» (число должно находиться в диапазоне от 0 до 65535).

Аналогичную роль играют явные преобразования типов:

$char(w)$ – возвращает символ с кодом w в кодировке $Unicode$;

$word(\mathrm{c})$ – возвращает код символа с в кодировке $Unicode$.

В табл.12 приведена сводка методов статического класса char.

Таблица 12

Методы статического класса ***char*** (*PascalABC.NET*)

Метод	Описание
char.IsDigit(c: char): boolean	Возвращает, является ли символ цифрой
char.IsLetter(c: char): boolean	Возвращает, является ли символ буквой
char.IsWhiteSpace(c: char): boolean	Возвращает, является ли символ пробельным
char.IsUpper(c: char): boolean	Возвращает, является ли символ буквой в верхнем регистре
char.IsLower(c: char): boolean	Возвращает, является ли символ буквой в нижнем регистре
char.IsPunctuation(c: char): boolean	Возвращает, является ли символ знаком препинания
char.IsLetterOrDigit(c: char): boolean	Возвращает, является ли символ буквой или цифрой
char.ToLower(c: char): char	Возвращает символ, преобразованный к нижнему регистру
char.ToUpper(c: char): char	Возвращает символ, преобразованный к верхнему регистру

Перечислимый и диапазонный типы

Перечислимый тип определяется упорядоченным набором идентификаторов:

```
type typeName = (value1, value2, ..., valuen);
```

Значения перечислимого типа занимают 4 байта. Каждое значение $value$ представляет собой константу типа $typeName$, попадающую в текущее пространство имён.

Пример 5.1.

```
type
   Season = (Winter,Spring,Summer,Autumn);
   DayOfWeek = (Mon,Tue,Wed,Thi, Fri, Sat, Sun);
```

К константе перечислимого типа можно обращаться непосредственно по имени, а можно использовать запись $typeName.value$, в которой имя константы уточняется именем перечислимого типа, к которому она принадлежит:

```
Var a: DayOfWeek;
a := Mon;
a := DayOfWeek.Wed;
```

Значения перечислимого типа можно сравнивать на $<$:

```
DayOfWeek.Wed < DayOfWeek.Sat
```

Для значений перечислимого типа можно использовать функции Ord, $Pred$ и $Succ$, а также процедуры Inc и Dec. Функция Ord возвращает порядковый номер значения в списке констант соответствующего перечислимого типа, нумерация при этом начинается с нуля.

Для перечислимого типа определена экземплярная функция $ToString$, возвращающая строковое представление переменной перечислимого типа. При выводе значения перечислимого типа с помощью процедуры $write$ также выводится строковое представление значения перечислимого типа.

Пример 5.2.

```
type
   Season = (Winter,Spring,Summer,Autumn);
var s: Season;
begin
   s := Summer;
   writeln(s.ToString);                                    Summer
   writeln(s);                                             Summer
end.
```

Диапазонный тип представляет собой подмножество значений целого, символьного или перечислимого типа и описывается в виде $a..b$, где a - нижняя, b - верхняя граница интервального типа, $a < b$:

```
Var
   intI: 0..10;
   intC: 'a'..'z';
   intE: Mon..Thi;
```

Тип, на основе которого строится диапазонный тип, называется базовым для этого диапазонного типа. Значения диапазонного типа занимают в памяти столько же, сколько и значения соответствующего базового типа.

Строковый тип

Строки имеют тип $string$, состоят из набора последовательно расположенных символов *char* и используются для представления текста.

Строки могут иметь произвольную длину. К символам в строке можно обращаться, используя индекс: $s[i]$ обозначает i-й символ в строке, нумерация начинается с единицы. Если индекс i выходит за пределы длины строки, то генерируется исключение.

Над строками определены операции сравнения: $<\ >, <=, >=, =,$ $<>$. Сравнение строк на неравенство осуществляется лексикографически: $s1 < s2$, если для первого несовпадающего символа с номером i $s1[i] < s2[i]$ или все символы строк совпадают, но $s1$ короче $s2$.

Операция $+$ для строк означает конкатенацию (слияние) строк. Например: $'\text{Петя}\text{^}' + \text{^}'\text{Маша}\text{^}' = (\_\text{^}'\text{ПетяМаша}\text{^}'$.

Расширенный оператор присваивания $+=$ для строк добавляет в конец строки - левого операнда строку – правый операнд.

Пример 5.3.

```
var s: string := 'Петя';
s += 'Маша';                                    s = 'ПетяМаша'
```

Строка может складываться с числом, при этом число предварительно преобразуется к строковому представлению:

```
s := 'Ширина: '+15;                             s = 'Ширина: 15'
s := 20.5+'';                                   s = '20.5'
s += 1;                                         s = '20.51'
```

Над строками и целыми числами определена операция *: $s * n$ и $n * s$ означает строку, образованную из строки s, повторенной n раз:

```
s := '*'*10;                                    s = '**********'
s := 5*'ab';                                    s = 'abababababab'
s := 'd'; s *= 3;                               s = 'ddd'
```

Строки реализуются типом $System.String$ платформы $.NET$ и представляют собой ссылочный тип. Таким образом, все операции над строками унаследованы от типа $System.String$. Однако, в отличие от $.NET$ - строк, строки в $PascalABC.NET$ изменяемы. Например, можно изменить $s[i]$ (в $.NET$ нельзя). Более того, строки $string$ в $PascalABC.NET$ ведут себя как размерные: после

```
var s2 := 'Hello';
var s1 := s2;
s1[2] := 'a';
```

строка $s2$ не изменится. Аналогично при передаче строки по значению в подпрограмму создается копия строки, т.е. обеспечивается поведение, характерное для $Delphi\ Object\ Pascal$, а не для $.NET$.

Однако строке можно присвоить nil[6], что необходимо для работы с NET-кодом.

Кроме того, в $PascalABC.NET$ реализованы размерные строки. Для их описания используется тип $string[n]$, где n - константа целого типа, указывающая длину строки. Размерные строки, в отличие от обычных, можно использовать как компоненты типизированных файлов. Для совместимости с $Delphi\ Object\ Pascal$ в стандартном модуле описан тип $shortstring = string[255]$.

5.2. Запись данных в память, или оператор присваивания ($PascalABC.NET$)

В предыдущем параграфе вам фактически предложено работать с данными трёх типов: целыми, вещественными и строковыми. Следует запомнить:

- целые числа в программах записываются так же, как принято в математике. Например: 345, -12222;
- вещественные числа могут записываться двумя способами: первый – аналогичный математической записи: -123.234, 92929.3456 (обратите внимание, что здесь используется десятичная точка, а не запятая), второй – в так называемом «плавающем» виде (правильное название: «десятичное число с плавающей точкой»). Например, число $1{,}23 \cdot 10^{12}$ может быть записано следующим образом: 1.23E+12. Здесь $\mathrm{E}+12$ читается как «умножить на десять в двенадцатой степени»;
- строки представляют собой произвольный набор символов, заключённых в апострофы:

  'Какой чудесный был пирог. Я от него ... 2+2=5'

Оператор присваивания предназначен для указания компьютеру *ЗАПИСИ ДАННЫХ В КОНКРЕТНЫЙ БЛОК ПАМЯТИ*. Общий вид оператора:

---

[6] Зарезервированное слово nil представляет собой специальное значение переменной-указателя, не указывающей ни на что конкретное.

<имя блока памяти>:=<выражение>

Обратите внимание на знак присваивания – он состоит из двух значков: двоеточия и равно (:=), которые записываются друг за другом без пробелов.

5.3. Арифметические операции, функции, выражения. Арифметический оператор присваивания (*PascalABC.NET*)

К арифметическим типам данных относятся группы вещественных и целых типов. К ним применимы арифметические операции и операции отношений.

Выражение – это конструкция, возвращающая значение некоторого типа. Простыми выражениями являются переменные и константы, например: 3.14, x, a12.

Более сложные выражения строятся из простых с помощью операций, вызовов функций и скобок. Данные, к которым применяются операции, называются операндами.

В $PascalABC.NET$ имеются следующие операции: @ , not, *, /, div, mod, and, shl, shr, $+, -, or$, xor, $=, >, <, <>, <=, >=, as$, is, in, $=>$, а также операция new и операция приведения типа.

Операции $@, -, +, (\_\hat{}, not$, *операция приведения типа* и *операция* new являются унарными (имеют один операнд), остальные являются бинарными (имеют два операнда), операции $+$ и $-$ являются и бинарными, и унарными. Унарная арифметическая операция одна. Это операция изменения знака. Её формат $-\text{величина}$.

Порядок выполнения операций определяется их приоритетом. В языке $PascalABC.NET$ четыре уровня приоритетов операций, задаваемых таблицей приоритетов.

Для типов, определённых пользователем, ряд операций можно перегружать.

К арифметическим относятся бинарные операции $+, -, *,$ / для вещественных и целых чисел, бинарные операции div и mod для целых чисел и унарные операции $+$ и $-$ для вещественных и целых чисел. Тип выражения $x\ op\ y$, где op – знак бинарной операции $+,\ -$ или $*$, определяется из табл. 13.

Таблица 13

Бинарные арифметические операции*

Знак	Выражение	Тип операндов	Тип результатов	Операция
+	$A + B$	R, R I, I $I, R; R, I$	R I R	Сложение
-	$A - B$	R, R I, I $I, R; R, I$	R I R	Вычитание
*	$A * B$	R, R I, I $I, R; R, I$	R I R	Умножение
/	$A \ / \ B$	R, R I, I $I, R; R, I$	R R R	Деление
div	$A\ div\ B$	I, I	I	Целое деление
mod	$A\ mod\ B$	I, I	I	Остаток от целого деления

Примечание.*I обозначает целые типы, R – вещественные тип.

В соответствии со структурой программы на $PascalABC.NET$ (см. раздел 4.1) напишем первую полноценную программу на $PascalABC.NET$.

Пример 5.4.

```
program example1;              {заголовок программы}
        var num : integer;     {резервирование блока памяти для хранения
                                целого числа}
        day : byte;            {резервирование блока памяти для хранения
                                целого числа от 0 до 255}
        name : string;         {резервирование блока памяти для хранения
                                строки}
Begin
        num:=355;              {запись в блок num числа 355}
        day:=31;               {запись в блок day числа 31}
        name:='Скорпион'       {запись в блок name слова «Скорпион»}
end.
```

К арифметическим величинам могут быть применены стандартные функции $PascalABC.NET$. Функция выступает как операнд в выражении. Например, в следующем операторе присваивания:

$$\mathrm{x} \coloneqq 2 * \mathrm{Sin(A)} \ / \ \mathrm{Ln(3.5)} \ + \mathrm{Cos(C - D)}$$

операндами являются три функции: sin, ln, cos. Их запись такая же, как в математике. Аргументы называются фактическими параметрами и являются в общем случае выражениями арифметического типа. Аргументы записываются в круглых скобках. Результат вычисления функции – величина соответствующего типа.

Как следует из определения оператора присваивания, в правой части оператора присваивания находится некое значение или выражение. В последнем случае в блок памяти (переменную) записывается значение выражения.

Пример 5.5.

```
program example2;          {заголовок программы}
    var num : integer;     {резервирование блока памяти для хранения
                            целого числа}
    day : byte;            {резервирование блока памяти для хранения
                            целого числа от 0 до 255}
Begin
    num:=355 + 145;        {запись в блок num числа 500}
    day:=num - 469;        {при вычислении значения выражения вме-
                            сто num будет подставлено значение из
                            этого блока памяти (500) и вы
                            числен результат (31)}
end.
```

В табл. 14 и табл. 15 приведена сводка бинарных операций для различного типа данных.

Таблица 14

Бинарные операции для всех простых типов данных (часть 1)

Типы	**shortint**	**byte**	**smallint**	**word**	**integer**
shortint	integer	integer	integer	integer	integer
byte	integer	integer	integer	integer	integer
smallint	integer	integer	integer	integer	integer
word	integer	integer	integer	integer	integer
integer	integer	integer	integer	integer	integer
longword	int64	longword	int64	longword	int64
int64	int64	int64	int64	int64	int64
uint64	uint64	uint64	uint64	uint64	uint64
BigInteger	BigInteger	BigInteger	BigInteger	BigInteger	BigInteger
single	single	single	single	single	single
real	real	real	real	real	real

Таблица 15

Бинарные операции для всех простых типов данных (часть 2)

Типы	longword	int64	uint64	BigInteger	single	real
shortint	int64	int64	uint64	BigInteger	single	real
byte	longword	int64	uint64	BigInteger	single	real
smallint	int64	int64	uint64	BigInteger	single	real
word	longword	int64	uint64	BigInteger	single	real
integer	int64	int64	uint64	BigInteger	single	real
longword	longword	uint64	uint64	BigInteger	single	real
int64	uint64	int64	uint64	BigInteger	single	real
uint64	uint64	uint64	uint64	BigInteger	single	real
BigInteger	BigInteger	BigInteger	BigInteger	BigInteger	-	-
single	single	single	single	-	single	real
real	real	real				

То есть, если операнды - целые, то результатом является самый короткий целый тип, требуемый для представления всех получаемых значений.

При выполнении бинарной операции с $uint64$ результирующим типом будет $uint64$, при этом может произойти переполнение, не вызывающее исключения.

Для операции $\boldsymbol{x/y}$ представленная выше табл. 15 исправляется следующим образом: результат деления любого целого на целое имеет тип $real$.

Для операций div и mod выполняются эти же правила, но операнды могут быть только целыми. Правила вычисления операций div и mod - следующие:

$x\ div\ y$ – результат целочисленного деления x на y. Точнее, $x\ div\ y\ =\ x/y$, округленное до ближайшего целого по направлению к 0;

$x\ mod\ y$ - остаток от целочисленного деления x на y. Точнее, $x\ mod\ y\ =\ x - (x\ div\ y) * y$.

Унарная арифметическая операция $+$ для любого целого типа возвращает этот тип. Унарная арифметическая операция $-$ возвращает для целых типов, меньших или равных $integer$, значение типа $integer$, для $longword$ и $int64$ - значение типа $int64$, к $uint64$ унарная

операция – не применима, для типов $single$ и $real$ - соответственно типы $single$ и $real$. То есть также результатом является самый короткий тип, требуемый для представления всех получаемых значений.

5.4. Ввод с клавиатуры и вывод данных на экран дисплея ($PascalABC.NET$)

Ввод данных – это передача информации от внешних устройств в оперативную память. Вводятся, как правило, исходные данные решаемой задачи. Вывод – обратный процесс, когда данные передаются из оперативной памяти на внешние носители (принтер, дисплей, магнитные устройства и т.д.). Результаты решения всякой задачи должны быть выведены на один из этих носителей.

Основными устройствами ввода-вывода у персонального компьютера являются клавиатура и дисплей (экран монитора). Именно через эти устройства главным образом осуществляется диалог между человеком и ПК.

Процедура ввода с клавиатуры имеет следующий формат:

Read(<список ввода>);

где <список ввода> – это последовательность имён переменных, разделённых запятыми. Слово $read$ переводится как читать. (Точнее говоря, $Read$ – это оператор обращения к стандартной процедуре ввода.)

Пример 5.6.

```
Var     T     : real;
        I     : integer;
        S     : char;
Begin
        Read(T, I, S);
End.
```

Набираем на клавиатуре: 0.234␣12␣G↲ Enter

Если в программе имеется несколько операторов $Read$, то данные для них вводятся потоком, т. е. после считывания значений переменных для одного оператора $Read$ данные для следующего оператора читаются из той же строки на экране, что и для предыдущего до окончания строки, затем происходит переход на следующую строку.

Пример 5.7.

```
Var     A, B   : real;
        C, D   : integer;
Begin
        Read(A, B);
        Read(C, D);
End.
```

Набираем на клавиатуре: 2.234E+02␣1.45E-03 ␣12␣23↲ Enter

Но можно и так: 2.234E+02␣1.45E-03↲ Enter12␣23↲ Enter

Другой вариант оператора ввода с клавиатуры имеет вид:

ReadLn(<список ввода>);

Здесь слово $ReadLn$ означает $read\ line$ – читать строку. Этот оператор отличается от $Read$ только тем, что после считывания последнего в списке значения для одного оператора $ReadLn$ данные для следующего оператора будут считываться с начала новой строки. Если в предыдущем примере заменить операторы $Read$ на $ReadLn$:

```
ReadLn(A, B);
ReadLn(C, D);
```

то ввод значений будет происходить из двух строк:

2.234E+02␣1.45E-03↲ Enter

12␣23↲ Enter

Примеры программ, приведённых выше, иллюстрируют команду присваивания и ввод данных в переменные, но выполнение таких программ на компьютере бессмысленно, ведь программа записывает данные в память компьютера и заканчивает работу – без вывода результата работы пользователю. Оператор вывода на экран (обращение к стандартной процедуре[7] вывода) имеет следующий формат:

Write(<список вывода>);

Здесь элементами списка вывода могут быть выражения различных типов (в частности, константы и переменные).

---

[7] Процедурой называется программа, оформленная специальным образом и рассматриваемая как единая команда. Следует иметь в виду, что процедур вывода в PascalABC.Net достаточно много.

Пример 5.8.

```
Write(234);                 {Выводится целая константа}
Write(A+B-2);               {Выводится результат вычисления выражения}
Write(x, Summ, Arg1, Arg2); {Выводятся значения переменных}
```

При выводе на экран нескольких чисел в строку они не отделяются друг от друга пробелами. Программист сам должен позаботиться о таком разделении. Пусть, например, $I = 1$; $J = 2$, $К = 3$. Тогда, написав в программе $write(I,' \sqcup ',J,' \sqcup ',K)$; получим на экране строку: $1\ 2\ 3$. После вывода последнего символа курсор остаётся в той же строке. Следующий вывод на экран будет начинаться с этой позиции курсора.

Второй вариант процедуры вывода на экран:

WriteLn(<список вывода>)

Слово $WriteLn$ – $write\ line$ – означает писать строку. Его действие отличается от оператора $Write$ тем, что после вывода последнего в списке значения происходит перевод курсора к началу следующей строки. Оператор $WriteLn$, записанный без параметров, вызывает перевод строки.

Форматы вывода. В списке вывода могут присутствовать указатели форматов вывода (форматы). Формат определяет представление выводимого значения на экране. Он отделяется от соответствующего ему элемента двоеточием. Если указатель формата отсутствует, то машина выводит значение по определённому правилу, предусмотренному по умолчанию.

Ниже кратко, в справочной форме, приводятся правила и примеры бесформатного и форматированного вывода величин различных типов. Для представления списка вывода здесь будут использованы следующие обозначения:

I, P, Q – целочисленные выражения;

R – выражение вещественного типа;

B – выражение булевского типа;

Ch – символьная величина;

S – строковое выражение;

– цифра;

$*$ – знак + или -

␣ – пробел.

Форматы процедуры $Write$

I – выводится десятичное представление величины I, начиная с позиции расположения курсора:

Значение ***I***	Оператор	Результат
123	$Write(I);$	123
287	$Write(I, I, I);$	287287287

$I: P$– выводится десятичное представление величины I в крайние правые позиции поля шириной P:

Значение ***I***	Оператор	Результат
123	$Write(I: 6);$	␣␣␣ 123
287	$Write((I + I): 7);$	␣␣␣␣ 574

R – в поле шириной 18 символов выводится десятичное представление величины R в формате с плавающей точкой. Если $R \geq 0{,}0$, то используется формат _#.##########E$*$##. Если $R < 0{,}0$, то формат имеет вид ␣ −#.##########E$*$##:

Значение ***R***	Оператор	Результат
715.432	$Write(R);$	␣ $7.1543200000E + 02$
$-1.919E + 01$	$Write(R);$	␣ $-1.9190000000E + 01$

$R: P$ – в крайние правые позиции поля шириной P ***символов*** выводится десятичное представление значения R в нормализованном формате с плавающей точкой. Минимальная длина поля вывода для положительных чисел составляет 7 символов, для отрицательных – 8 символов. После точки выводится по крайней мере одна цифра:

Значение ***R***	Оператор	Результат
511.04	$Write(R: 15);$	$5.110400000E + 02$
46.78	$Write(-R: 12);$	$-4.67800E + 01$

$R: P: Q$ – в крайние правые позиции поля шириной P символов выводится десятичное представление значения R в формате с фиксированной точкой, причем после десятичной точки выводится Q цифр ($0 \leq Q \leq 24$), представляющих дробную часть числа. Если $Q = 0$, то ни дробная часть, ни десятичная точка не выводятся. Если

$Q > 24$, то при выводе используется формат с плавающей точкой:

Значение $\boldsymbol{R}$	Оператор	Результат
511.04	$Write(R:8:4);$	511.0400
-46.78	$Write(R:7:2);$	$\sqcup -46.78$

$Ch:P$ – в крайнюю правую позицию поля шириной P выводится значение Ch:

Значение $\boldsymbol{Ch}$	Оператор	Результат
$'X\hat{}'$	$Write(Ch:3);$	$\sqcup\sqcup X$
$[\![\sqcup\ \hat{}'!\ ]\!]\ \hat{}'$	$Write(Ch:2, Ch:4);$	$\sqcup\ !\ \sqcup\sqcup\sqcup\ !$

S – начиная с позиции курсора выводится значение S:

Значение $\boldsymbol{S}$	Оператор	Результат
$'$День $N\hat{}'$	$Write(S);$	День N
$'RRDD\hat{}'$	$Write(S, S);$	$RRDDRRDD$

$S:P$ – значение S выводится в крайние правые позиции поля шириной P символов:

Значение $\boldsymbol{S}$	Оператор	Результат
$'$День $N\hat{}'$	$Write(S:10);$	$\sqcup\sqcup\sqcup\sqcup$ День $\sqcup\ N$
$'RRDD\hat{}'$	$Write(S:5, S:5);$	$\sqcup\ RRDD\ \sqcup RRDD$

B – выводится результат выражения B: true или false, начиная с текущей позиции курсора:

Значение $\boldsymbol{B}$	Оператор	Результат
$True$	$Write(B);$	$True$
$False$	$Write(B, Not\ B);$	$FalseTrue$

$B:P$ – в крайние правые позиции поля шириной P символов выводится результат булевского выражения:

Значение $\boldsymbol{B}$	Оператор	Результат
$True$	$Write(B:6);$	$\sqcup\sqcup True$
$False$	$Write(B:6, Not\ B:7);$	$\sqcup\ False\ \sqcup\sqcup\sqcup True$

Использование для вывода на экран только процедур $Write$ и $WriteLn$ дает программисту очень слабые возможности для управления расположением на экране выводимого текста. Печать текста может производиться только сверху вниз, слева направо. Невозможны возврат к предыдущим строкам, стирание напечатанного текста, изменение цвета символов и т.д.

Дополнительные возможности управления выводом на экран дают процедуры и функции модуля CRT. Об этом модуле и способах его использования подробно рассказано в «Программирование в Turbo Pascal 7.0. Учебное пособие» [1].

5.5. Главные правила синтаксиса VBA

Синтаксис VBA, как понятно из самого названия этого языка (которое расшифровывается как $Visual\ Basic\ for\ Applications$), почти полностью совпадает с синтаксисом $Visual\ Basic$. Некоторые основные синтаксические принципы этого языка:

- VBA нечувствителен к регистру;
- чтобы закомментировать код до конца строки, используется одинарная кавычка (') или команда REM;
- символьные значения должны заключаться в двойные кавычки;
- максимальная длина любого имени в VBA (переменные, константы, процедуры) – 255 символов;
- начало нового оператора – перевод на новую строку;
- ограничений на максимальную длину строки нет (хотя в редакторе умещается только 308 символов). Несколько операторов в одной строке разделяются двоеточиями:

```
MsgBox "Проверка 1" : MsgBox "Проверка 2"
```

- для удобства чтения можно объединить несколько физических строк в одну логическую при помощи пробела:

```
MsgBox "Сообщение пользователю" _
        & vUserName
```

5.5.1. Типы данных VBA

В VBA, как и в любом другом языке программирования, переменные и константы используются для хранения каких-либо значений. Как следует из названия, переменные могут изменяться, константы же хранят фиксированные значения.

Например, константа *Pi* хранит значение 3,14159265… Число “Пи” не будет изменяться в ходе выполнения программы, но все же хранить такое значение удобнее как константу.

В то же время мы можем использовать переменную *sVAT_Rate* для хранения ставки НДС на покупаемые товары. Величина переменной *sVAT_Rate* может изменяться в зависимости от того, что за товар приобретается.

Все переменные и константы относятся к определённому типу данных. В табл. 16 приведены типы данных, используемые в VBA, с описанием и диапазоном возможных значений.

Таблица 16

Типы данных ***VBA***

Тип данных	Размер	Описание	Диапазон значений
1	2	3	4
Byte	1 байт	Положительные целые числа; часто используется для двоичных данных	от 0 до 255
Boolean	2 байта	Может принимать значения либо True, либо False	True или False
Integer	2 байта	Целые числа (нет дробной части)	от -32 768 до +32 767
Long	4 байта	Большие целые числа (нет дробной части)	от -2 147 483 648 до +2 147 483 647
Single	4 байта	Число с плавающей точкой одинарной точности	от -3.4e38 до +3.4e38
Double	8 байт	Число с плавающей точкой двойной точности	от -1.8e308 до +1.8e308
Currency	8 байт	Число с плавающей точкой, с фиксированным количеством десятичных разрядов	от -922 337 203 685 477.5808 до +922 337 203 685 477.5807
Date	8 байт	Дата и время – данные типа Date представлены числом с плавающей точкой. Целая часть этого числа выражает дату, а дробная часть – время	от 1 января 100 до 31 декабря 9999

Продолжение табл.16

1	2	3	4
Object	4 байта	Ссылка на объект	Любая ссылка на объект
String	Изменяется	Набор символов. Тип String может иметь фиксированную или изменяющуюся длину. Чаще используется с изменяющейся длиной	Фиксированной длины – приблизительно до 65 500 символов. Переменной длины – приблизительно до 2 миллиардов символов
Variant	Изменяется	Может содержать дату, число с плавающей точкой или строку символов. Этот тип используют в тех случаях, когда заранее не известно, какой именно тип данных будет введён	Число – Double, строка – String

Очевидно, что, пользуясь табл. 16 и правильно выбирая тип данных, можно использовать память более экономно (например, выбрать тип данных *Integer* вместо *Long* или *Single* вместо *Double*). Однако, используя более компактные типы данных, нужно внимательно следить за тем, чтобы в коде не было попыток уместить в них не соразмерно большие значения.

5.5.2. Option Explicit

Оператор *Option Explicit* заставляет объявлять все переменные, которые будут использованы в коде *VBA*, и при компиляции выделяет все необъявленные переменные как ошибки (прежде чем будет запущено выполнение кода). Применить этот оператор не сложно – просто запишите в самом верху файла *VBA* такую строку:

```
Option Explicit
```

Если хотите всегда вставлять *Option Explicit* в начало каждого нового созданного модуля VBA, то это можно делать автоматически. Для этого необходимо включить параметр *Require Variable Declaration* в настройках редактора *VBA*.

Это делается так:

- В меню редактора кода *Visual Basic* нажмите *Tools* > *Options*
- В появившемся диалоговом окне откройте вкладку *Editor*
- Отметьте галочкой параметр *Require Variable Declaration* и нажмите *OK*
- При включенном параметре строка *Option Explicit* будет автоматически вставляться в начало каждого нового созданного модуля.

5.5.3. Область действия переменных и констант

Каждая объявленная переменная или константа имеет свою ограниченную область действия, то есть ограниченную часть программы, в которой эта переменная существует. Область действия зависит от того, где было сделано объявление переменной или константы. Возьмём, к примеру, переменную *sVAT_Rate*, которая используется в функции *Total_Cost*. Далее рассматриваются варианты области действия переменной *sVAT_Rate*, объявленной в двух различных позициях в модуле:

```
‘ Вариант №1


Option Explicit

Dim sVAT_Rate As Single

Function Total_Cost() As Double


...


End Function
```

Если переменная *sVAT_Rate* объявлена в самом начале модуля, то областью действия этой переменной будет весь модуль (т.е. переменная *sVAT_Rate* будет распознаваться всеми процедурами в этом модуле).
Следовательно, если в функции *Total_Cost* переменной *sVAT_Rate* будет присвоено некоторое значение, то следующая функция, выполняемая в пределах этого же

модуля, будет использовать переменную *sVAT_Rate* с этим же значением.
Однако, если будет вызвана какая-то функция, расположенная в другом модуле, то для неё переменная *sVAT_Rate* будет не известна.

```
‘ Вариант №2

Option Explicit

Function Total_Cost() As Double
        Dim sVAT_Rate As Single

...

End Function
```

Если переменная *sVAT_Rate* объявлена в начале функции *Total_Cost*, то её область действия будет ограничена только этой функцией (т.е. в пределах функции *Total_Cost*, можно будет использовать переменную *sVAT_Rate*,а за её пределами – нет).
При попытке использовать *sVAT_Rate* в другой процедуре, компилятор *VBA* сообщит об ошибке, так как эта переменная не была объявлена за пределами функции *Total_Cost* (при условии, что использован оператор *Option Explicit*).

В показанном выше примере переменная объявлена на уровне модуля при помощи ключевого слова *Dim*. Однако бывает необходимо, чтобы объявленными переменными можно было пользоваться в других модулях. В таких случаях для объявления переменной вместо ключевого слова *Dim* нужно использовать ключевое слово *Public*.

Кстати, для того чтобы объявить переменную на уровне модуля, вместо ключевого слова *Dim* можно использовать ключевое слово *Private*, которое укажет на то, что данная переменная предназначена для использования только в текущем модуле.

Для объявления констант также можно использовать ключевые слова *Public* и *Private*, но не вместо ключевого слова *Const*, а вместе с ним.

В следующих примерах показано использование ключевых слов *Public* и *Private* в применении к переменным и к константам.

Пример 5.9.

```
Option Explicit

Public sVAT_Rate As Single

Public Const iMax_Count = 5000

...
```

В этом примере ключевое слово *Public* использовано для объявления переменной sVAT_Rate и константы iMax_Count. Областью действия объявленных таким образом элементов будет весь текущий проект.
Это значит, что sVAT_Rate и iMax_Count будут доступны в любом модуле проекта.

Пример 5.10.

```
Option Explicit

Private sVAT_Rate As Single

Private Const iMax_Count = 5000

...
```

В этом примере для объявления переменной sVAT_Rate и константы iMax_Count использовано ключевое слово Private. Областью действия этих элементов является текущий модуль.
Это значит, что sVAT_Rate и iMax_Count будут доступны во всех процедурах текущего модуля, но не будут доступны для процедур, находящихся в других модулях.

5.5.4. Операторы VBA: арифметические, логические, сравнения, присвоения

Оператор – это наименьшая способная выполняться единица кода VBA. Оператор может объявлять или определять переменную, устанавливать параметр компилятора VBA или выполнять какое-либо действие в программе.

Арифметических операторов в VBA всего 7. Четыре стандартных: сложение ($x + y$), вычитание ($x - y$), умножение ($x * y$), деление ($\boldsymbol{x/y}$) и еще три:

- возведение в степень ($x\text{^}y$), например $2\text{^}3 = 8$;
- целочисленное деление (\). Делит первое число на второе, отбрасывая (не округляя) дробную часть. Например, $5\backslash 2 = 2$;

- деление по модулю (Mod). Делит первое число на второе, возвращая только остаток от деления. Например, $5\ Mod\ 2 = 1$.

Оператор присвоения в VBA – знак равенства. Можно записывать так:

```
Let nVar = 10
```

а можно ещё проще:

```
nVar = 10
```

Во втором случае не путайте знак равенства с оператором равенства.

Выражение *nVar = 10* – это значит «присвоить переменной $nVar$ значение 10», а если строка выглядит так: *If (nVar = 10)* – это значит «если значение переменной $nVar$ равно 10».

Если переменной нужно назначить объект, то делается это другими способами.

Операторов сравнения в VBA всего 8:

- равенство ($=$), например, $If\ (nVar = 10)$;
- больше, чем и меньше, чем ($>$ и $<$), например, $If\ (nVar > 10)$;
- больше или равно и меньше или равно ($>=$ и $<=$), например, $If\ (nVar >= 10)$;
- не равно ($<>$), например, $If\ (nVar <> 10)$;
- сравнение объектов (Is). Определяет, ссылаются объектные переменные на тот же объект или на разные, например, $If\ (obj1\ Is\ obj2)$;
- подобие ($Like$). Сравнивает строковый объект с шаблоном и определяет, подходит ли шаблон.

Операторы сравнения всегда возвращают $true$ или $false$ – $true$, если утверждение истинно, и $false$, если ложно.

Немного про сравнение строковых значений:

- при сравнении строковых значений регистр учитывается;
- пробелы в строковых значениях также учитываются;
- при сравнении текстовых строк на больше/меньше по

умолчанию сравниваются просто двоичные коды символов – какие больше или меньше. Если нужно использовать тот порядок, который идёт в алфавите, то можно воспользоваться командой *Option Compare Text.*

Чуть подробнее про оператор $Like$. Общий его синтаксис выглядит как *Выражение1 Like Выражение2*

При этом *Выражение1* – любое текстовое выражение VBA, а *Выражение2* – шаблон, который передаётся оператору $Like$. В этом шаблоне можно использовать специальные подстановочные символы (табл. 17).

Таблица 17

Подстановочные символы для оператора *LIKE*

Подстановочный символ	Значение
#	Любая цифра (только одна) от 0 до 9
*	Любое количество любых символов (включая нулевое)
?	Любой символ (только один)
[a,b,c]	Любой символ (только один) из приведённого списка
[!a,b,c]	Любой символ (только один), кроме приведённых в списке

Очень часто при проверке нескольких условий используются логические операторы:

AND – логическое И, должны быть истинными оба условия;

OR – логическое ИЛИ, должно быть истинным хотя бы одно из условий;

NOT – логическое отрицание, возвращает $TRUE$, если условие ложно;

XOR – логическое исключение. В выражении $E1\ XOR\ E2$ возвращает $TRUE$, если только $E1 = TRUE$ или только $E2 = TRUE$, иначе – $FALSE$;

EQV – эквивалентность двух выражений, возвращает $TRUE$, если они имеют одинаковое значение;

IMP – импликация, возвращает $FALSE$, если $E1 = TRUE$ и $E2 = FALSE$, иначе - $TRUE$.

Помнить нужно про AND, OR, NOT, остальные логические операторы используются редко.

Почти в любой программе VBA используются операторы конкатенации. В VBA их два – $+$ или $\&$. Рекомендуется всегда использовать $\&$, потому что:

- при использовании $\&$ производится автоматическое преобразование числовых значений в строковые - нет опасности допустить ошибку;
- при использовании оператора $+$ сложение строкового значения со значением типа $Null$ дает $Null$.

Пример 5.11.

```
MsgBox "Сообщение пользователю" & vUserName
```

Порядок применения операторов можно регулировать при помощи круглых скобок.

Пример 5.12.

Математическая запись: $y = \ (a + b)/(c - d)$

В *VBA:* y = (a+b)/(c-d)

5.5.5. Переменные и типы данных

Переменные – контейнеры для хранения изменяемых данных. Без них не обходится практически ни одна программа. Для простоты переменную можно сравнить с номерком в гардеробе – вы сдаёте в «гардероб» какие-то данные, в ответ вам выдаётся номерок. Когда вам опять потребовались эти данные, вы «предъявляете номерок» и получаете их.

Пример 5.13.

```
Dim nMyAge As Integer
nMyAge = nMyAge + 10
MsgBox nMyAge
```

Перед работой с переменной настоятельно рекомендуется её объявить. Объявление переменной в нашем примере выглядит так:

```
Dim nMyAge As Integer
```

Как расшифровать эту строку:

Dim – это область видимости переменной. В VBA предусмотрено 4 ключевых слова для определения области видимости переменных:

- Dim – используется в большинстве случаев. Если переменная объявлена как Dim в области объявлений модуля, она будет доступна во всем модуле, если в процедуре – только на время работы этой процедуры;
- $Private$ – при объявлении переменных в VBA значит то же, что и Dim;
- $Public$ – такая переменная будет доступна всем процедурам во всех модулях данного проекта, если вы объявили её в области объявлений модуля. Если вы объявили её внутри процедуры, то она будет вести себя как *Dim или Private;*
- $Static$ – такие переменные можно использовать только внутри процедуры. Эти переменные видны только внутри процедуры, в которой они объявлены, зато они сохраняют своё значение между разными вызовами этой процедуры. Обычно используются для накопления каких-либо значений.

Пример 5.14.

```
Static nVar1 As Integer
nVar1 = nVar1 + 1
MsgBox nVar1
```

Если нет никаких особых требований, то есть смысл всегда выбирать область видимости Dim.

Второе слово в нашем объявлении ($nMyAge$) – это идентификатор (проще говоря, имя) переменной. Правила выбора имен в VBA едины для многих элементов (переменные, константы, функции и процедуры и т.п.). Имя:

- должно начинаться с буквы;

- не должно содержать пробелов и символов пунктуации (исключение – символ подчёркивания);
- максимальная длина – 255 символов;
- должно быть уникальным в текущей области видимости (подробнее – далее);
- зарезервированные слова (те, которые подсвечиваются другим цветом в окне редактора кода) использовать нельзя.

При создании программ VBA настоятельно рекомендуется определиться с правилами, по которым будут присваиваться имена объектам – соглашение об именовании. Чаще всего используется так называемое ***венгерское соглашение*** (в честь одного из программистов $Microsoft$, $Charles\ Simonyi$, венгра по национальности):

имя переменной должно начинаться с префикса, записанного строчными буквами. Префикс указывает, что именно будет храниться в этой переменной:

- str (или s) – $String$, символьное значение;
- fn (или f) – функция;
- c (или сделать все буквы заглавными) – константа;
- b – $Boolean$, логическое значение ($true$ или $false$);
- d – дата;
- obj (или o) – ссылка на объект;
- n – числовое значение.

Имена функций, методов и каждое слово в составном слове должно начинаться с заглавной буквы:

```
MsgBox objMyDocument.Name
Sub CheckDateSub()
```

В ранних версиях $Visual\ Basic$ не было слова $Const$ – все константы определялись как переменные, а для отличия их записывали заглавными буквами, между словами ставили подчёркивания: *COMPANY_NAME*.

Многие программисты используют такой подход для обозначения констант и сейчас (но использование ключевого слова $Const$ теперь обязательно – об этом будет рассказано в разделе 5.5.6).

Третья часть нашего объявления – $As\ Integer$ – это указание на тип данных нашей переменной. Тип данных определяет, данные какого вида можно будет хранить в нашей переменной.

В VBA предусмотрены следующие типы данных:

- числовые ($byte$ – целое число от 0 до 255, $integer$ – целое число от -32768 до 32767, $long$ – большое целое число, $currency$ (большое десятичное число с 19 позициями, включая 4 позиции после запятой), $decimal$ (ещё большее десятичное число с 29 позициями), $single$ и $double$ – значение с плавающей запятой ($double$ в два раза больше));

ВАЖНО: Попытка объявить переменную с типом $Decimal$ (например, $Dim\ n\ As\ Decimal$) приведет к синтаксической ошибке. Чтобы получить возможность работать с типом $Decimal$, переменную нужно изначально объявить, как $Variant$ или вообще объявить без типа ($Dim\ n$), поскольку тип данных $Variant$ используется в VBA по умолчанию.

- строковые ($string$ переменной длины (до примерно 2 млрд символов) и фиксированной длины (до примерно 65400 символов);
- дата и время ($date$ – от 01.01.0100 до 31.12.9999);
- логический ($boolean$ – может хранить только значения $True$ и $False$);
- объектный ($object$ – хранит ссылку на любой объект в памяти);
- $Variant$ – специальный тип данных, который может хранить любые другие типы данных.

Можно ещё использовать пользовательские типы данных, но их вначале нужно определить при помощи выражения $Type$. Обычно пользовательские типы данных используются как дополнительное средство проверки вводимых пользователем значений (классический пример – почтовый индекс).

Некоторые моменты, связанные с выбором типов данных для переменных:

- общий принцип – выбирайте наименьший тип данных, который может вместить выбранные вами значения. Если есть какие-то сомнения – выбирайте больший тип данных во избежание возникновения ошибок;
- если есть возможность, то лучше не использовать типы данных с плавающей запятой ($single$ и $double$). Работа с такими типами данных производится медленнее, кроме того, могут быть проблемы при сравнениях за счёт округлений;
- если есть возможность, то лучше не пользоваться типом $Variant$. Этот тип все равно приводится VBA к одному из других типов, но памяти для него требуется больше. Кроме того, в ходе такого неявного образования могут возникнуть ошибки;
- при определении переменных можно использовать так называемые символы определения типа ($\% - integer$, $\$ - String$ и т.п.). Например, в примере 5.14 нужно закомментировать строку *Static nVar1 As Integer*, а во второй строке написать: *nVar1% = nVar1% + 1.*

Такой подход является устаревшим и к использованию не рекомендуется.

При объявлении переменных можно и не указывать её тип. Например, наше объявление может выглядеть так: $Dim\ nVar1$.

В этом случае переменная будет автоматически объявлена с типом $Variant$.

В принципе в VBA можно работать и без объявления переменных. Например, такой код:

```
nVar1 = nVar1 + 1
MsgBox nVar1
```

будет вполне работоспособным. Если мы используем переменную в программе без её объявления, то будет автоматически создана новая переменная типа $Variant$. ***Однако объявлять переменные нужно обязательно!*** И при этом желательно явно указывать нужный тип данных. Почему:

- сокращается количество ошибок: программа с самого начала откажется принимать в переменную значение неправильно типа (например, строковое вместо числового);

- при работе с объектами подсказка по свойствам и методам действует только тогда, когда мы изначально объявили объектную переменную с нужным типом. Например, в $Excel$ два варианта кода будут работать одинаково:

первый вариант:

```
Dim oWbk As Workbook
Set oWbk = Workbooks.Add()
```

второй вариант:

```
Set oWbk = Workbooks.Add()
```

Но подсказка по свойствам и методам объекта $oWbk$ будет работать только во втором случае.

Все опытные разработчики вообще запрещают использование переменных без явного их объявления. Для этого можно воспользоваться специальной командой компилятора (она помещается только в раздел объявлений модуля) *Option Explicit*, а можно вставлять эту команду во все модули при их создании автоматически – установив в окне редактора кода флажок $Require\ Variable\ Declarations$ (меню $Tools \rightarrow Options$, вкладка $Editor$).

Проиллюстрировать, зачем они это делают, можно на простом примере.

Пример 5.15.

```
Dim n
п = п + 1
MsgBox п
```

С виду код не должен вызывать никаких проблем и просто выводить в окне сообщения единицу. На самом деле он выведет пустое окно сообщения. Причина спрятана очень коварно: в третьей строке n – это вовсе не английская буква N, а русская Π. На вид в окне редактора кода отличить их очень сложно. В то же время компилятор VBA, встретив такой код, просто создаст новую переменную с типом данных $Variant$, у которой будет пустое значение. На выявление такой ошибки может потребоваться определённое время.

Хорошее правило – объявлять переменные заблаговременно, а не когда они потребовались. Это позволяет сделать программу более читаемой и чётко спланированной.

Можно объявить несколько переменных в одной строке, например, так:

```
Dim n1 As Integer, s1 As String
```

Присвоение значений переменным выглядит так:

```
nVar1 = 30
```

Если нужно увеличить уже существующее значение переменной, то команда может выглядеть так:

```
nVar1 = nVar1 + 1
```

В обоих примерах знак равенства означает не «равно», а присвоить.

При присвоении значений переменным нужно помнить о следующем:

- строковые значения всегда заключаются в двойные кавычки:

```
sVar1 = "Hello";
```

- значение даты/времени заключается в «решётки» – символы фунта:

```
dVar1 = #05/06/2020#
```

Обратите внимание, что при присвоении значения даты/времени таким «явным способом» нам придётся использовать принятые в США стандарты: 05 в данном случае – это месяц, 06 – день. Отображение же этого значения (например, в окне сообщения) будет зависеть от региональных настроек на компьютере пользователя.

Если нужно передать шестнадцатеричное значение, то перед ним ставятся символы $\&H$:

```
nVar1 = &HFF00
```

Что содержится в переменных до присвоения им значений?

В переменных всех числовых типов данных – 0.

В строковых переменных переменной длины – "" (строка нулевой длины).

В строковых переменных фиксированной длины – строка данной длины с символами $ASCII$ 0 (эти символы на экран не выводятся).

В $Variant$ – пустое значение.

В $Object$ – ничто (нет ссылки ни на один из объектов).

5.5.6.Константы, объявление, ключевое слово $Const$, встроенные константы, $vbCrLf$

Константы – ещё один контейнер для хранения данных, но, в отличие от переменных, они не изменяются в ходе выполнения VBA-программы. Для чего нужны константы:

- код становится лучше читаемым/убираются потенциальные ошибки;
- чтобы изменить какое-либо значение, которое много раз используется в программе (например, уровень налога) – это можно сделать один раз.

В VBA константы определяются при помощи ключевого слова $Const$:

```
Const COMP_NAME As String = “Microsoft”
```

При попытке в теле процедуры изменить значение константы будет выдано сообщение об ошибке.

Константы очень удобны при работе с группами именованных элементов (дни недели, месяцы, цвета, клавиши, типы окон и т.п.). Они позволяют использовать в коде программы легко читаемые обозначения вместо труднозапоминаемых числовых кодов. Например, строки

```
UserForm1.BackColor = vbGreen
```

и

```
UserForm1.BackColor = 65280
```

функционально одинаковы, но в чем смысл первой строки, догадаться гораздо легче.

В VBA встроено множество служебных констант: календарных, для работы с файлами, цветами, формами, типами дисков и т.п. Просмотреть их можно через справочную систему VBA: $Microsoft\ Visual\ Basic\ Documentation \rightarrow Visual\ Basic\ Reference \rightarrow Constants$. Про одну из констант (она находится в разделе $Miscellaneous$) следует сказать особо: константа $vbCrLf$ позволяет произвести переход на новую строку.

Пример 5.16.

```
MsgBox ("Первая строка" + vbCrLf + "Вторая строка")
```

5.5.7. Функция $MsgBox$

Функция $MsgBox$ выводит сообщение в диалоговом окне, ожидает нажатия кнопки пользователем и возвращает значение типа $Integer$, которое указывает, какая кнопка использовалась (табл. 18–20).

Синтаксис:

MsgBox(Сообщение [, Кнопки] [, Заголовок] [, ФайлСправки] [, Контекст])[8]

Таблица 18

Аргументы функции $MsgBox$

Аргумент	Описание
Сообщение	Обязательный аргумент. строковое выражение, отображаемое как сообщение в диалоговом окне. Максимальная длина строки аргумента *Сообщение* составляет приблизительно 1024 знаков и зависит от их ширины. Если аргумент *Сообщение* содержит несколько строк, их можно разделить с помощью знака возврата каретки ($Chr(13)$), знака перевода строки ($Chr(10)$) или сочетания этих знаков ($Chr(13)$ & $Chr(10)$).
Кнопки	Необязательный аргумент. числовое выражение, которое представляет собой сумму значений, указывающих число и тип отображаемых кнопок, стиль значка, активную кнопку по умолчанию, а также, является ли окно сообщения модальным. Если аргумент *Кнопки* пропущен, по умолчанию используется значение 0.
Заголовок	Необязательный аргумент. Строковое выражение, отображаемое в строке заголовка диалогового окна. Если аргумент *Заголовок* опущен, в строке заголовка выводится имя приложения.
ФайлСправки	Необязательный аргумент. Строковое выражение, определяющее файл справки, в котором содержится контекстная справка данного диалогового окна. Если задан аргумент *ФайлСправки*, необходимо также указать аргумент *Контекст*.
Контекст	Необязательный аргумент. Числовое выражение, представляющее собой номер контекста, присвоенный автором справки соответствующему разделу. Если задан аргумент *Контекст*, необходимо также указать аргумент *ФайлСправки*.

[8] Англ. вариант **MsgBox**(prompt [, buttons] [, title] [, helpfile, context])

Таблица 19

Параметры аргумента Кнопки

Константа	Значение	Описание
1	2	3
vbOKOnly	0	Отображается только кнопка *ОК*.
vbOKCancel	1	Отображаются кнопки *ОК* и *Отмена*.
vbAbortRetryIgnore	2	Отображаются кнопки *Прервать*, *Повторить* и *Пропустить*.
vbYesNoCancel	3	Отображаются кнопки *Да*, *Нет* и *Отмена.*
vbYesNo	4	Отображаются кнопки *Да* и *Нет*.
vbRetryCancel	5	Отображаются кнопки *Повторить* и *Отмена*.
vbCritical	16	Отображается значок важного сообщения.
vbQuestion	32	Отображается значок сообщения с предостережением.
vbExclamation	48	Отображается значок предупреждающего сообщения.
vbInformation	64	Отображается значок информационного сообщения.
vbDefaultButton1	0	По умолчанию активна первая кнопка.
vbDefaultButton2	256	По умолчанию активна вторая кнопка.
vbDefaultButton3	512	По умолчанию активна третья кнопка.
vbDefaultButton4	768	По умолчанию активна четвертая кнопка.
vbApplicationModal	0	Модальность на уровне приложения. Пользователь должен ответить на сообщение, чтобы продолжить работу в текущем приложении.
vbSystemModal	4096	Модальность на уровне системы. В ожидании ответа пользователя на сообщение приостанавливается работа всех приложений.

Продолжение табл.19

1	2	3
vbMsgBoxHelpButton	16384	В диалоговое окно сообщения добавляется кнопка «Справка».
VbMsgBoxSetForeground	65536	Окно сообщения выводится на переднем плане.
vbMsgBoxRight	524288	Текст сообщения выравнивается по правому краю.
vbMsgBoxRtlReading	1048576	Текст сообщения выводится справа налево. Эта возможность предназначена для систем, в которых используется арабский язык или иврит.

Первая группа значений (0 – 5) отражает число и тип кнопок в диалоговом окне. Вторая группа (16, 32, 48, 64) описывает стиль значка сообщения. Третья группа (0, 256, 512) определяет активную по умолчанию кнопку. Наконец, четвертая группа (0, 4096) устанавливает модальность сообщения. При добавлении чисел в итоговое значение аргумента Кнопки следует использовать только один аргумент из группы.

ПРИМЕЧАНИЕ: Приведённые здесь константы определены в $Visual\ Basic$ для приложений, что позволяет заменить ими числовые значения в любом месте программы.

Таблица 20

Возвращаемые значения функции $\boldsymbol{MsgBox}$

Константа	Значение	Описание
vbOK	1	ОК
vbCancel	2	Отмена
vbAbort	3	Прервать
vbRetry	4	Повторить
vbIgnore	5	Пропустить
vbYes	6	Да
vbNo	7	Нет

Если указаны аргументы *ФайлСправки* и *Контекст*, то пользователь может, нажав клавишу $F1$, открыть раздел справки, заданный аргументом *Контекст*. Некоторые ведущие приложения, например, $Microsoft\ Office\ Excel\ 2007$ и старше, автоматически добавляют кнопку Справка в диалоговое окно.

Если в диалоговом окне отображается кнопка *Отмена*, то нажатие кнопки *Отмена* равносильно нажатию клавиши ESC. Если в диалоговом окне выводится кнопка *Справка*, то для данного окна доступна контекстная справка. Однако до нажатия какой-либо другой кнопки никакое значение не возвращается.

ПРИМЕЧАНИЕ: Чтобы указать несколько аргументов, а не только первый из названных, используйте функцию $MsgBox$ в выражении. Чтобы не включать некоторые аргументы, замените их разделителем в виде запятой.

В своей самой простой форме $MsgBox$ используется как оператор с одним аргументом – сообщением, которое должно отображаться. Приведённый ниже макрос создаёт сообщение, показанное на рис. 26.

Пример 5.17.

```
Sub Program ()
        MsgBox "Это - окно сообщений"
End Sub
```

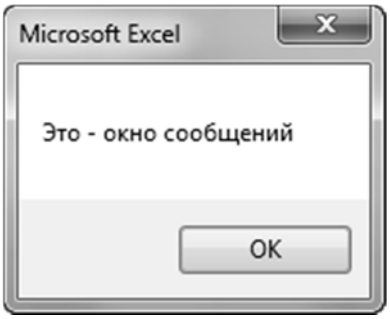


Рис. 26. Простое окно сообщения

MsgBox можно использовать для отображения числового значения (рис. 27).

Пример 5.18.

```
Sub ShowValue()
    Amount = 10
    MsgBox Amount
End Sub
```

Рис. 27. Вывод числа

Переменной $Amount$ присваивается значение 10. На следующей строке для отображения значения $Amount$ используется $MsgBox$. Вокруг $Amount$ нет кавычек, поскольку это – значение переменной, которое нужно выдать на экран, а не слово «$Amount$».

Чтобы использовать вместе две отдельные строки в одном окне сообщения, следует использовать операцию конкатенации $(\&)$ – объединение (слияние) (рис. 28).

Пример 5.19.

```
Sub SayGoodNight()
    Name = "Саша"
    MsgBox "Пожелайте удачи! " & Name
End Sub
```

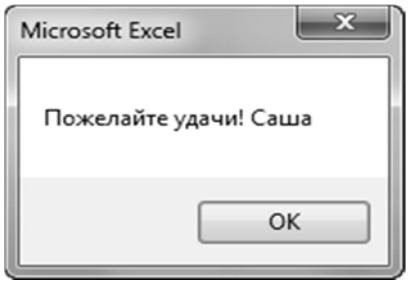


Рис. 28. Слияние

Переменной $Name$ присваивается строка "Саша". В строке кода с $MsgBox$ задаётся текстовая строка "Пожелайте доброй ночи ", за которой следует $\& Name$, указывая $MsgBox$ присоединить значение переменной $Name$ к предыдущей текстовой строке (рис. 29).

Пример 5.20.

```
Sub BestMessage()
    MsgBox "Это - замечательное
    окно сообщений", vbExclamation,
    "Персональное окно"
End Sub
```

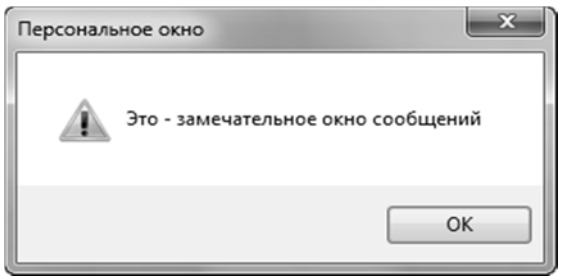


Рис. 29. Использование аргументов

5.5.8. Функция $InputBox$

Синтаксис:

InputBox(Prompt[,Title] [,Default] [,XPos] [,YPos] [,HelpFile,Context])[9]

Функция выводит на экран диалоговое(модальное) окно с кнопкой закрытия, содержащее заданное сообщение, поле ввода, кнопки OK, $Cancel$ и опционально заголовок и/или кнопку $Help$, ожидая от пользователя ввода текста или щелчка кнопки (табл. 21). При задании не только одного первого параметра необходимо использовать функцию $InputBox$ в выражении. Для пропуска некоторых параметров нужно включить соответствующие разделители в виде запятых.

Возвращаемое значение типа $String$, включающее содержимое окна текста.

Таблица 21

Аргументы функции $InputBox$

Аргумент	Описание
1	2
Prompt	Обязательный. Строковое выражение, отображаемое как сообщение в диалоговом окне. Максимальная длина параметра $Prompt$ составляет приблизительно 1024 символа и зависит от ширины используемых символов. Строковое значение $Prompt$ может содержать нескольких физических строк. Для разделения строк допускается использование символа возврата каретки, символа перевода строки или комбинации этих символов
Title	Необязательный. Строковое выражение, отображаемое в строке заголовка диалогового окна. Если этот параметр опущен, в строку заголовка помещается имя приложения. Максимальное число символов заголовка около 50
Default	Необязательный. Строковое выражение, отображаемое в окне текста как ответ, используемый по умолчанию, если пользователь не введёт другую строку. Если этот параметр опущен, окно текста отображается пустым

[9] Русск. вариант **InputBox**(*запрос*, [*название*],[по умолчанию], [*xpos*], [*ypos*], [Файл Справки, *контекст*])

Продолжение табл. 21

1	2
XPos	Необязательный. Числовое выражение, задающее расстояние по горизонтали между левой границей диалогового окна и левым краем экрана (в твипах). Если этот параметр опущен, то диалоговое окно выравнивается по центру экрана по горизонтали
YPos	Необязательный. Числовое выражение, задающее расстояние по вертикали между верхней границей диалогового окна и верхним краем экрана (в твипах). Если этот параметр опущен, то диалоговое окно помещается по вертикали на расстоянии примерно на одну треть высоты экрана.
HelpFile	Необязательный. Строковое выражение, определяющее имя файла Справки, содержащего контекстно-зависимую Справку о данном диалоговом окне. Если этот параметр указан, то необходимо задать также и параметр $Context$
Context	Необязательный. Числовое выражение, определяющее номер соответствующего раздела справочной системы. Если этот параметр указан, то необходимо задать также и параметр $HelpFile$

ПРИМЕЧАНИЕ: Если указаны оба параметра $HelpFile$ и $Context$, то пользователь имеет возможность нажатием клавиши $F1$ вызвать соответствующую контекстную справку. Некоторые главные приложения, например $Excel$, также автоматически добавляют в диалоговое окно кнопку *Справка*. Если пользователь щелкает кнопку OK или нажимает $ENTER$, то функция $InputBox$ возвращает содержимое поля ввода. Если пользователь щелкает кнопку $Cancel$, то функция возвращает пустую строку (␣)

ВАЖНО: Однако если пользователь не введёт в поле ввода строку, то функция тоже вернёт пустую строку. В этом случае будет затруднительно определить - какую кнопку (OK или $Cancel$) нажал пользователь. На самом деле $Cancel$ возвращает $vbNullString$, который и воспринимается $Visual\ Basic$ как пустая строка. Но можно воспользоваться недокументированной функцией $StrPtr$ и получить указатель на строку.

Пример 5.21.

```
Dim strInput As String
strInput = InputBox("")
If StrPtr(strInput) = 0 Then
   MsgBox "Вы нажали Cancel!"
End If
```

Пример 5.22.

Пример демонстрирует различные способы получения сведений от пользователя с помощью функции $InputBox$. Если аргументы x и y опущены, то окно диалога автоматически выравнивается по центру по соответствующим осям. Переменная $MyValue$ содержит значение, введённое пользователем, если была нажата кнопка OK или клавиша $ENTER$. Если пользователь нажмёт кнопку $Cancel$, то функция возвратит пустую строку.

Далее на (рис. 30 и рис. 31) приводятся примеры использования функции *InputBox*.

```
Dim Message, Title, Default, MyValue
Message = "Введите число от 1 до 3"
Title = "Пример"  ' заголовок
Default = "1"  ' значение по умолчанию
```

Рис. 30. Значение по умолчанию

```
MyValue = InputBox(Message, Title, Default)
MyValue = InputBox(Message, Title, , , ,
"DEMO.HLP", 10)
```

Рис. 31. Кнопка "Справка" добавляется автоматически

```
MyValue = InputBox(Message, Title, Default,
100, 100)
```

Размещаем верхний левый угол окна диалога в точке 100, 100.

5.6. Примеры линейных программ на $PascalABC.NET$ и VBA

Линейными называют алгоритмы, в которых операции выполняются последовательно одна за другой, в естественном и единственном порядке следования. В таких алгоритмах все блоки имеют последовательное соединение логической связью передачи информационных потоков. В них могут использоваться все блоки, за исключением блоков проверки условия и модификации. Линейные алгоритмы, как правило, являются составной частью любого алгоритмического процесса.

Пример 5.23. Вычисление площади круга.

а) $PascalABC.NET$

Исходный код

```
program CircleSqr;
const
   Pi = 3.1415;

var
   r: real; // радиус круга
   S: real;// площадь круга

begin
   write('Введите радиус круга: ');
   readln(r);
   S := Pi * r * r;
   writeln('Площадь круга равна ', S);
end.
```

Окно вывода

```
Введите радиус круга: 3
Площадь круга равна 28.2735
```

б) VBA (рис. 32 и рис. 33)

Исходный код

```
Sub CircleSqr()
   Const Pi As Single = 3.14
   Dim r, s As Single
   Dim Message, Title, Default

   Message = "Введите радиус круга"
   Title = "Вычисление площади круга."  ' заголовок
   Default = "1"  ' значение по умолчанию

   r = InputBox(Message, Title, Default)
   Message = "Площадь круга = "
   Title = "Вычисление площади круга."  ' заголовок
```

Окно вывода

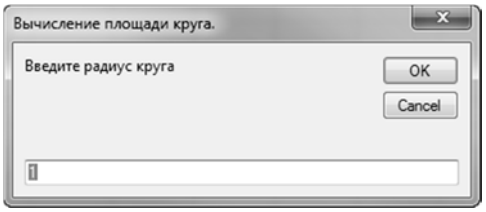


Рис. 32. Ввод значения радиуса

```
    MsgBox Message & Str(Pi * r ^ 2), vbInfor-
mation, Title
End Sub
```

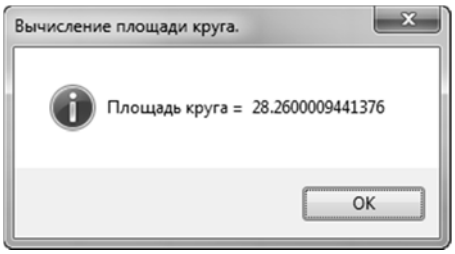


Рис. 33. Окно сообщения

ВНИМАНИЕ: Функция $Str - Str(Number)$. Функция $Str(String)$ используется для приведения числового выражения типа $Long$ в строку(тип $String$). Функция $Str(Expression)$ возвращает значение $Number$, преобразованное в текстовый тип данных $String$. При преобразовании в начале строки возвращаемого значения резервируется место для знака числа. Если число положительно, то в этом месте будет пробел, если число отрицательно, то выводится знак минус. В качестве десятичного разделителя дроби функция Str воспринимает только точку. При использовании других десятичных разделителей (например, запятой) следует использовать функцию $CStr$

```
Dim retval
retval = Str(123)' получаем " 123"
retval=Str(-12.3)' получаем "-12.3"
```

Пример 5.24. Использование вспомогательных переменных, вычислить a^8.

а) $PascalABC.NET$

Исходный код

```
program Exponent;
var r: real;

begin
  write('Введите r: ');
  readln(r);
  var r2,r4,r8: real; // вспомогательные переменные
  r2 := r * r;
  r4 := r2 * r2;
  r8 := r4 * r4;
  writeln(r,' в степени 8 = ',r8);
end.
```

Окно вывода

```
Введите r: 2
2 в степени 8 = 256
```

б) VBA (рис. 34 и рис. 35)

Исходный код

```
' Вариант 1
Sub Exponent_var1()
    Dim r As Single
    Dim Message, Title

    Message = "Введите значение r = "
    Title = "Вычисление степени."

    r = InputBox(Message, Title)
    ' Вспомогательные переменные
    Dim r2, r4, r8 As Single

    r2 = r * r
    r4 = r2 * r2
    r8 = r4 * r4
    Message = "Результат: " & Str(r) _
    & " в степени 8 = "
    MsgBox Message & Str(r8), vbInfor-
mation, Title
End Sub
```

Окно вывода

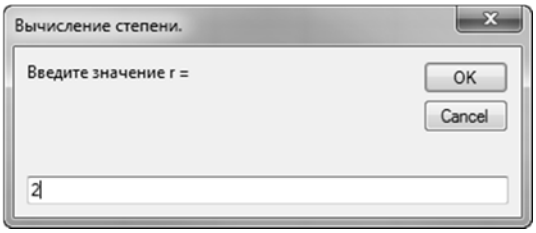


Рис. 34. Ввод значения r

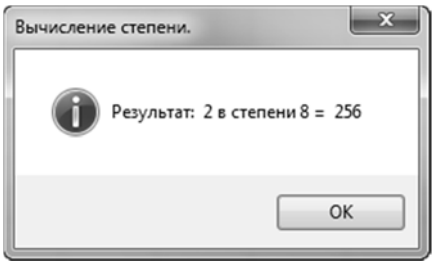


Рис. 35. Окно сообщения

```
' Вариант 2
Sub Exponent_var2()
    Dim r As Single
    Dim Message

    Message = "Введите значение r = "
    r = InputBox(Message, "Вычисление степени.")
    Message = "Результат: " & Str(r) & " в степени 8 = "
    MsgBox Message & Str(r ^ 8), vbInformation, "Вычисление степени."
End Sub
```

Пример 5.25. Вычисление расстояния между двумя точками на прямой.

а) $PascalABC.NET$

Исходный код

```
program Line;

var
    a, b: real; // координаты точек
    r: real;// расстояние между точками на прямой

begin
    write('Введите координату точки a: ');
```

Окно вывода

```
Введите координату точки a: 5
Введите координату точки b: 8
Расстояние между точками = 3
```

```
    readln(a);
    write('Введите координату точки b: ');
    readln(b);
    r := abs(a - b);
    writeln('Расстояние между точками = ', r);
end.
```

ВНИМАНИЕ: Функция *ABS(число)* - возвращает модуль (абсолютную величину) числа. Абсолютная величина числа – это число без знака.

б) VBA (рис. 36 - рис. 38)

Исходный код

```
Sub Line()
    Dim a, b As Single
    Dim Message

    Message = "Введите координату точки a
= "
    a = InputBox(Message, "Координата
точки.")

    Message = "Введите координату точки b
= "
    b = InputBox(Message, "Координата
точки.")

    Message = "Результат: расстояние
между точками = "
    MsgBox Message & Str(Abs(a - b)), vbInfor-
mation, _
    "Вычисление расстояния."
End Sub
```

Окно вывода

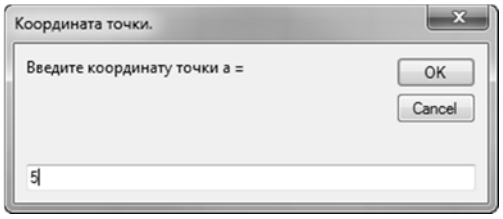


Рис. 36. Ввод значения a

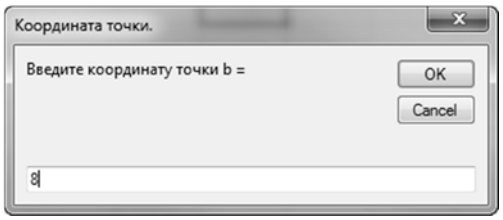


Рис. 37. Ввод значения b

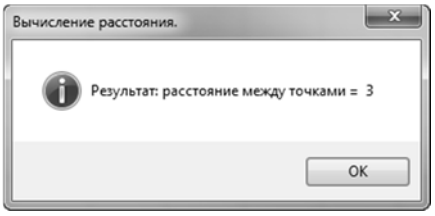


Рис. 38. Окно сообщения

5.7. Разветвляющиеся программы на $PascalABC.NET$ и VBA

При составлении схем алгоритмов часто возникает необходимость проведения анализа исходных данных или промежуточных результатов вычислений и определения дальнейшего порядка выполнения вычислительного процесса в зависимости от результатов этого анализа. Алгоритмы, в которых в зависимости от выполнения некоторого логического условия происходит разветвление вычислений по

одному из нескольких возможных направлений, называют ***разветвляющимися***. Подобные алгоритмы предусматривают выбор одного из альтернативных путей продолжения вычислений. Каждое возможное направление вычислений называется ветвью. Логическое условие называют простым, если разветвляющийся процесс имеет две ветви, и сложным – если процесс разветвляется на три и более ветви.

Разветвляющимся называется алгоритм, при выполнении которого каждый раз последовательность действий может быть разная, т.е. каждый раз выбирается один из нескольких путей прохождения схемы алгоритма. Конкретный путь прохождения алгоритма называется ветвью алгоритма. Схема подобного алгоритма обязательно содержит хотя бы один блок (символ) "решение", который и обеспечивает разветвление вычислительного процесса.

5.7.1. Условный оператор (PascalABC.NET)

Представим себе классическую задачу решения квадратного уравнения: если дискриминант положительный – тогда уравнение имеет два разных решения, если он равен нулю – то одно, а при отрицательном дискриминанте вещественных решений нет вообще. Идея здесь одна – решение квадратного уравнения зависит от дискриминанта, а точнее – от его знака. Или ещё задача: определить количество дней в году. Естественно, что количество дней зависит от того, является год високосным или нет.

А вот пример геометрической задачи подобного рода: выяснить, будут ли три числа a, b и c сторонами треугольника. Мы знаем, что три числа будут сторонами некоего треугольника тогда и только тогда, когда сумма любых двух сторон этого треугольника больше третьей стороны. Здесь тоже, как мы видим, решение задачи зависит от некоторых условий (трех неравенств).

Эти три задачи объединяет одно: здесь нельзя просто так составить программу на Паскале, чтобы задача решалась «одним махом» – из-за наличия определенных условий. Поэтому в таких случаях на помощь приходит условный оператор, который распределяет выполнение операторов в зависимости от условий.

Код на ***PascalABC*** записывается следующим образом:

```
1  .....
2
3  if <условие> then <оператор1>
4  else <оператор2>;
5
6  .....
```

Эта конструкция работает по такому принципу. Сначала проверяется условие (после *if*); если оно истинно, то выполняется оператор1 (после *then*), в противном случае – оператор2 (после *else*). И ещё: *if* означает «если», *then* – «тогда», *else* – «иначе». Все это зарезервированные слова в *Pascal*, всегда выделяются жирным шрифтом. Ещё одно важное замечание: перед *else* никогда не ставится точка с запятой, поскольку оно составляет единое целое с *if* и *then*. Поясним, как используется условный оператор на простом примере:

Пример 5.26.

```
var
  t: real; { Температура воздуха }

begin
  writeln('Введите температуру воздуха:');
  readln(t);
  if t > 0 then writeln('вода не замерзла')
  else writeln('вода замерзла');
  readln;
end.
```

Здесь идет проверка температуры воздуха *t* – если она больше 0 (условие *t* > 0 после *if*), то выполняется оператор после *then* и мы увидим ответ: «вода не замерзла»; в противном случае (то есть при температуре от 0 и ниже: *t* ≤ 0 - это условие противоположное по отношению к предыдущему *t* > 0) выполнится оператор после *else* и ответ будет другим: «вода замерзла» (без кавычек). Это пример простейшей задачи на использование условного оператора.

В следующем примере учтём температуру кипения воды (при нормальном давлении - 100^0C.

Пример 5.27.

```
var
  t: real;

Begin
  writeln('Введите температуру воздуха:');
  readln(t);
  if t <= 0 then writeln('вода замерзла')
  else
    if t < 100 then writeln('вода нагревается')
    else writeln('вода закипела');
  readln
end.
```

Здесь надо сделать некоторые пояснения. Сначала мы проверяем отрицательную или нулевую температуру ($t \leq 0$): если это действительно так, то выполняется оператор вывода *write*('вода замерзла')– это и есть оператор1. В противном случае (то есть, если температура положительная) должен быть выполнен оператор2, стоящий после *else*. Но мы видим, что там находится ещё одна конструкция условного оператора со своими *if*, *then* и *else*.

Тогда встает вопрос: что именно считать оператором2? А как раз эта вся конструкция и будет оператором2, поскольку программой она воспринимается как один оператор (условный оператор - единственный оператор, состоящий из трех частей: *if* - *then* - *else*). Таким образом, здесь мы будем проверять положительные температуры: если температура меньше 100 ($t < 100$), то выполнится оператор вывода *write*('вода нагревается'), иначе (от 100 и выше градусов) вступит в действие *write*('вода закипела'). Все это будет относиться к первому *else*.

Бывает так, что при истинности или ложности условия (после *if*) вместо выполнения одного оператора (оператора1 или оператора2) должна выполнятся целая группа операторов. В этом случае используются операторные скобки *begin* – *end*, в которые записываются все необходимые команды.

В общем случае запись на ***PascalABC*** такова:

```
1   .....................
2
3   if <условие> then
4    begin
5      <оператор1.1>;
6      <оператор1.2>;
7      . . . . . .
8      <оператор1.N>;
9    end
10  else
11   begin
12     <оператор2.1>;
13     <оператор2.2>;
14     . . . . . .
15     <оператор2.M>;
16   end;
17
18  ....................
```

Как видно, при истинности условия выполняется одна группа из *N* операторов, а при ложности – другой набор из *M* операторов. Условие (между *if* и *then*) тоже может состоять из нескольких частей. Для примера решим задачу о существовании треугольника по его сторонам.

Пример 5.28.

Дано три числа: *a*, *b* и *c*. Определить, существует ли треугольник со сторонами, равными этим числам.

Как упоминалось выше, треугольник будет существовать при выполнении трех условий: *a+b>c, b+c>a, c+a>b*. Поскольку все три неравенства должны выполняться одновременно, то мы их объединим оператором *and* (логическое «и»). В итоге получим:

```
Program treugolnik;
var
  a, b, c: real;

begin
  writeln('Введите три положительных числа:');
  readln(a, b, c);
  write('Треугольник со сторонами ', a, ', ', b, ' и ', c);
  { Проверяем условие существования треугольника: }
  if (a+b>c)and(b+c>a)and(c+a>b) then writeln(' существует')
  else writeln(' не существует');
  readln
end.
```

Пример 5.29.

Даны два числа. Вывести большее из них.

```
var
  a, b: real;

begin
  writeln('Введите два числа:');
  readln(a, b);
  if a = b then writeln('Эти числа равны')
  else begin
    write('Большее из этих чисел: ');
    if a < b then writeln(b)
    else writeln(a)
  end;
  readln
end.
```

Пример 5.30.

Упорядочение двух значений по возрастанию

```
var
  x,y: integer;
  v: integer;
begin
  write('Введите x,y: ');
  readln(x,y);
  if x>y then
  begin
    v := x;
    x := y;
    y := v
  end;
  writeln('Результат упорядочения по возрастанию: ',x,'  ',y);
end.
```

Пример 5.31.

Нахождение корней квадратного уравнения

```
var

  a,b,c: real;
  x1,x2,D: real;

begin
  writeln('Введите коэффициенты a, b, c квадратного уравнения
a*x*x+b*x+c=0: ');
  readln(a,b,c);
  D := b*b - 4*a*c;
```

```
  if D<0 then
    writeln('Корней нет')
  else if D=0 then
  begin
    x1 := -b/2/a;
    writeln('Корни совпадют: x1=x2=',x1);
  end
  else
  begin
    x1 := (-b-sqrt(D))/2/a;
    x2 := (-b+sqrt(D))/2/a;
    writeln('Корни: x1=',x1:0:3,' x2=',x2:0:3);
  end;
end.
```

5.7.2. Операторы условного и безусловного перехода (VBA)

Операторы условного перехода – одни из самых важных и часто используемых элементов в языках программирования. Общий принцип их работы прост: проверяется соответствие каким-то условиям (истинность или ложность каких-либо выражений) и в зависимости от этого выполнение программы направляется по одной или другой ветви. В VBA предусмотрено два оператора условного перехода: *If… Then... Else* и *Select Case*.

Оператор *If… Then... Else* - самый популярный у программистов. Полный его синтаксис выглядит так:

```
If Условие Then
   Команды1
[ElseIf Условия N Then
   Команды N]
[Else
   Команды2]
End If
```

При этом:

- *Условие* – выражение, которое проверяется на истинность. Если оно истинно, то выполняются *Команды1*, если ложно – *Команды2*.
- *УсловияN* – дополнительные условия, которые также можно проверить. В случае, если они выполняются (выражение *УсловияN* истинно), то выполняются *КомандыN*.

Оператор *If…Then... Else* применяется:

- когда нужно проверить на соответствие одному условию и в случае соответствия сделать какое-то действие:

```
If nTemperature < 10 Then
   MsgBox "Одеть куртку"
End If
```

- когда нужно сделать то же, что и в предыдущем примере, а в случае несоответствия выполнить другое действие:

```
If nTemperature < 10 Then
   MsgBox "Одеть куртку"
Else
   MsgBox "Одеть ветровку"
End If
```

- когда нужно проверить на соответствие нескольким условиям (обратите внимание на использование логических операторов):

```
If (nTemperature < 10) And (bRain = True) Then
   MsgBox "Одеть куртку и взять зонтик"
End If
```

- когда в случае, если первая проверка вернула *False*, нужно проверить на соответствие еще нескольким условиям (в этом случае удобно использовать *ElseIf*):

```
If (bIGoInCar = True) Then
   MsgBox "Одеться для машины"
ElseIf nTemperature < 10 Then
   MsgBox "Одеть куртку"
Else
   MsgBox "Можно идти в рубашке"
End If
```

В этом примере, поскольку *bIGoInCar* – переменная типа *Boolean* и сама по себе принимает значения *True* или *False*, первая строка может выглядеть так:

```
If bIGoInCar Then …
```

Некоторые замечания по использованию *If…Then... Else*:

- ключевое слово *Then* должно находиться в одной строке с *If* и условием. Если вы перенесете его на следующую строку, то будет выдано сообщение об ошибке;
- если разместить команду, которую нужно выполнить при истинности проверяемого условия, на одной строке с *If* и *Then*,

то *End If* можно не писать:

```
If nTemperature < 10 Then MsgBox "Одеть куртку"
```

- если же вы используете несколько команд или конструкции *Else/ElseIf*, то *End If* в конце нужно писать обязательно – иначе возникнет синтаксическая ошибка.
- для выражения *If…Then* настоятельно рекомендуется использовать отступы для выделения блоков команд. Иначе читать код будет трудно.
- операторы *If…Then* можно вкладывать друг в друга:

```
If MyVar = 5 Then
   MsgBox “MyVar = 5”
   If MyVar = 10 Then
         MsgBox “MyVar = 10”
   End If
End If
```

Пример 5.32.

Дано три числа: a, b и c. Определить, существует ли треугольник со сторонами, равными этим числам

```
Sub treugolnik()
   Dim a As Integer, b As Integer, c As Integer
   a = 2
   b = 3
   c = 2
   If (a + b > c) And (b + c > a) And (c + a > b) Then
      MsgBox "Треугольник со сторонами: a = " & _
         Str(a) & " b = " & Str(b) & " c = " & Str(c) & " существует."
   Else
      MsgBox "Треугольник со сторонами: a = " & _
         Str(a) & " b = " & Str(b) & " c = " & Str(c) & " не существует."
   End If
End Sub
```

Пример 5.33.

Даны два числа. Вывести большее из них

```
Sub primer()
   Dim a As Integer, b As Integer
   a = 21
   b = 3

   If a = b Then
      MsgBox "Числа равны"
   Else
```

```
        If a < b Then
            MsgBox "Число a меньше числа b"
        Else
            MsgBox "Число a больше числа b"
        End If
    End If
End Sub
```

Пример 5.34.

Упорядочение двух значений по возрастанию

```
Sub primer()
    Dim x As Integer, y As Integer, c As Integer
    x = 21
    y = 3

    If x > y Then
        c = x
        x = y
        y = c
    End If

    MsgBox "Результат упорядочения по возрастанию: " & Str(x) & _
            "; " & Str(y)
End Sub
```

Пример 5.35.

Нахождение корней квадратного уравнения

```
Sub primer()
    Dim a As Single, b As Single, c As Single
    Dim x1 As Single, x2 As Single, D As Single

    a = 1
    b = -4
    c = 3
    D = b ^ 2 - 4 * a * c

    If D < 0 Then
        MsgBox " Корней нет"
    ElseIf D = 0 Then
        x1 = -b / 2 / a
        MsgBox "Корни свпадают: x1=x2= " & Str(x1)
    Else
        x1 = (-b - Sqr(D)) / 2 / a
        x2 = (-b + Sqr(D)) / 2 / a
        MsgBox "Корни: x1= " & Str(x1) & ", x2= " & Str(x2)
    End If
End Sub
```

5.8. Циклические программы

Циклом в программировании называют повторение одних и тех же действий (шагов). Последовательность действий, которые повторяются в цикле, называют ***телом цикла***.

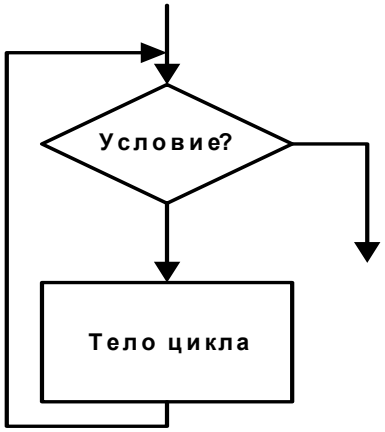


Рис. 39. Цикл с предусловием

Рис. 40. Цикл с постусловием

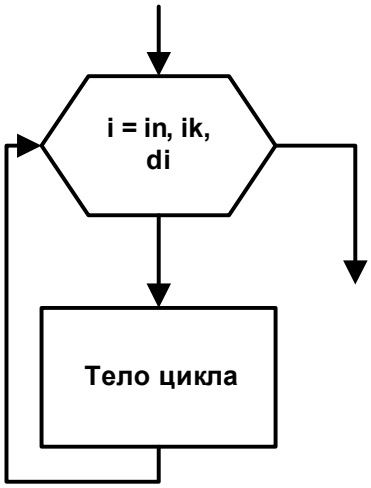


Рис. 41. Цикл без условия

На рис. 39 и рис. 40 представлены блок-схемы циклов с предусловием и с постусловием соответственно, которые называют условными циклическими алгоритмами. Применение блок-схем подробно описано в учебном пособии [3].

Нетрудно заметить, что эти циклы взаимозаменяемы. Но есть и отличия:

- в цикле с предусловием условие проверяется до тела цикла, в цикле с постусловием – после тела цикла;
- в цикле с постусловием тело цикла выполняется хотя бы один раз, в цикле с предусловием тело цикла может не выполняться ни разу;
- в цикле с предусловием проверяется условие продолжения цикла, в цикле с постусловием – условие выхода из цикла.

При написании уловных циклических алгоритмов следует помнить следующее:

- во-первых, чтобы цикл имел шанс когда-нибудь закончиться, содержимое его тела должно обязательно влиять на условие цикла;
- во-вторых, условие должно состоять из корректных выражений и значений, определенных еще до первого выполнения тела цикла.

Кроме того, существует так называемый безусловный циклический алгоритм (рис. 41), который удобно использовать, если известно, сколько раз необходимо выполнить тело цикла.

Выполнение безусловного циклического алгоритма начинается с присвоения переменной, например *i*, стартового значения *in*. Затем следует проверка, не превосходит ли переменная *i* конечного значения *ik*. Если это так, то цикл считается завершенным и управление передается следующему за телом цикла оператору. В противном случае выполняется тело цикла и переменная *i* меняет свое значение в соответствии с указанным шагом *di*. Далее снова производится проверка значения переменной *i*, и алгоритм повторяется. В принципе безусловный циклический алгоритм можно заменить любым условным.

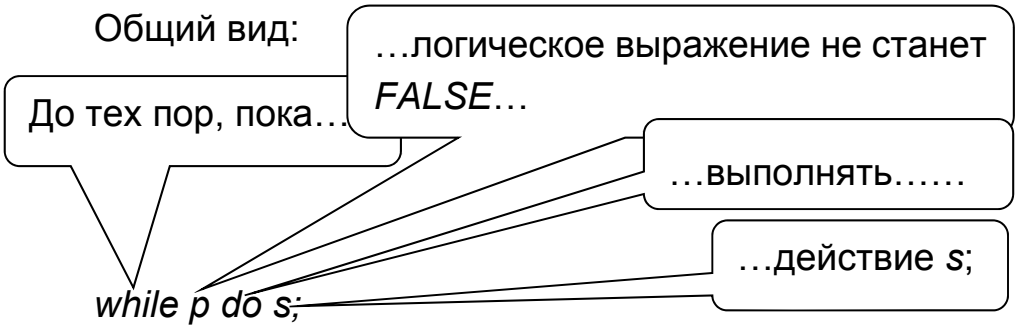
Переменную *i* в данном контексте, а именно при использовании в циклическом алгоритме, называют параметром цикла, так как эта переменная изменяется внутри цикла по определенному закону и влияет на его окончание. Принято также ее называть – счетчик цикла.

5.8.1. Цикл с предусловием (PascalABC.NET)

Синтаксис:

<оператор *while*>::= *while*<логическое выражение> *do* <оператор>

Общий вид:



Логическое выражение управляет количеством повторений цикла. Значение переменных, входящих в логическое, должно задаваться до оператора цикла и изменяться в теле цикла, поэтому тело цикла в основном оформляется как составной оператор. Оператор *while* выполняется до тех пор, пока выражение не станет равным *false*. Если выражение тождественно *true* (*while true do s*;), то цикл будет выполняться бесконечно. Если выражение с самого начала есть *false*, то тело цикла не выполняется ни разу.

Рассмотрим пару примеров использования оператора *while*.

Пример 5.36.

Даны числа x и y (y > 1). Необходимо получить все члены бесконечной последовательности x, x2, x3, …, которые меньше y (рис. 42).

```
program s1;

Uses crt;

var
   x, y, z: real;

begin
   TextBackground(15);        //Задаёт в окне консоли белый фон
   TextColor(0);              //Задаёт шрифту черный цвет
   clrscr;
   read(x, y);
   z := x;
   while z < y do
   begin
      writeln(z);
      z := z * x
```

```
    end;
end.
```

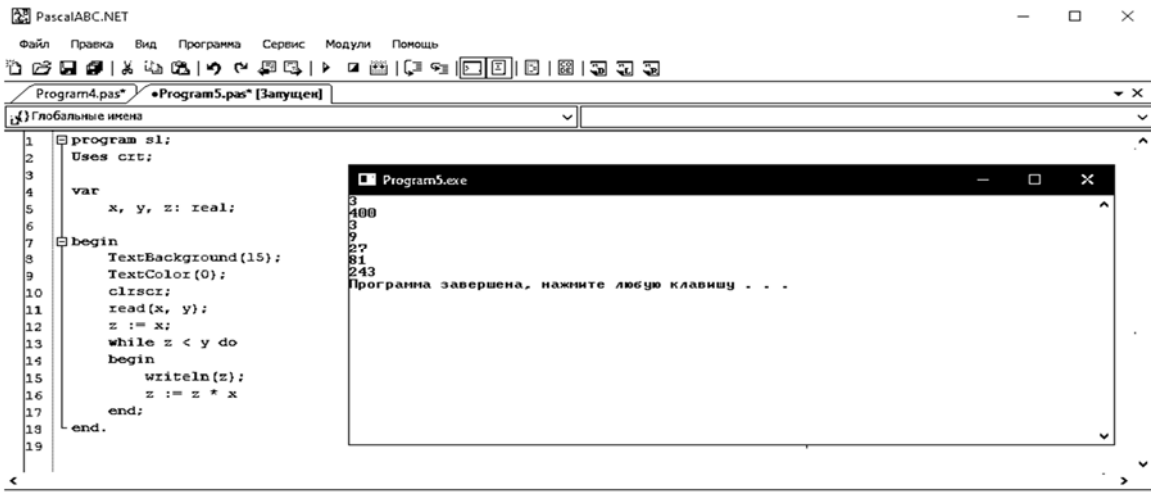


Рис. 42. Программа вычисления последовательностистепеней

В примере в качестве значений x и y заданы числа 3 и 400 соответственно и, согласно условиям задачи, вычислены члены последовательности: 3 (3^1), 9 (3^2), 27 (3^3), 81 (3^4) и 243 (3^5).

Пример 5.37.

Вычисление числа π (пи) методом Грегори.

По формуле Грегори число π можно вычислить следующим образом: $\pi/4 = 1 - 1/3 + 1/5 - 1/7 + \cdots$, причем вычисления будут происходить до тех пор, пока абсолютная величина очередного члена представленного ряда не станет меньше $0{,}5.10^{-7}$. Правило формирования этой последовательности станет яснее, если ее первый член представить в виде $1/1$.

```
program pi;

Uses crt;

const
   c = 0.5E-7;

var
   a, sum: real;
   sign: integer;
   n: longint;

begin
   TextBackground(15);        //Задаёт в окне консоли белый фон
```

```
    TextColor(0);                //Задаёт шрифту черный цвет
    clrscr;
    sign := -1;
    sum := 1.0;
    a := sum;
    n := 1;
    while abs(a) > c do
    begin
       a := sign / (2 * n + 1);
       sum := sum + a;
       sign := -sign;
       n := n + 1;
    end;
    sum := 4 * sum;
    writeln('Pi = ', sum);
end.
```

В представленной программе (рис. 43) константа *c* определяет точность вычисления числа *π*; переменная *a* содержит значение очередного члена ряда $1 - 1/3 + 1/5 - 1/7 + \cdots$; значение переменной *sum* равно текущей сумме вычисленных членов ряда (т.е. приближенному значению ¼ числа π); переменная *sign* содержит знак очередного члена ряда; переменная *n* служит для нумерации членов ряда (и итераций оператора *while*).

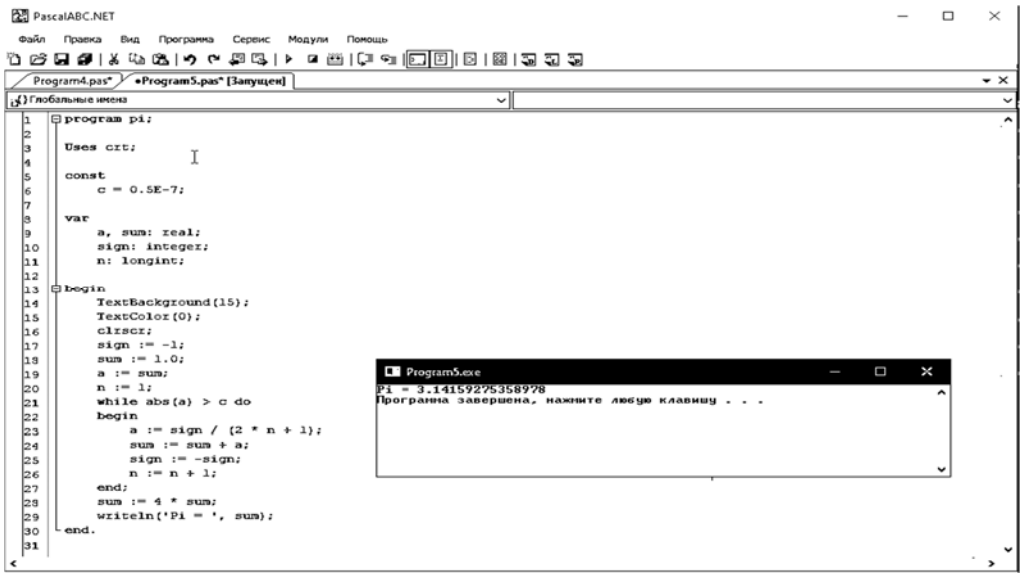


Рис. 43. Расчёт числа π

Переменная *n* объявлена с типом данных *Longint* (см. табл. 4). Это целочисленный тип данных, отличающийся от типа *Integer* только диапазоном допустимых значений. В рассмотренной программе для вычисления числа *π* с точностью $0{,}5.10^{-7}$ оператору *while* потребуется 10 000 001 циклов. Указанное число вне диапазона допустимых значений типа *Integer,* и, если переменную *n* объявить с типом данных *Integer*, то при выполнении программы произойдет так называемое «зацикливание». Для обнаружения этого достаточно ввести (временно) в составной оператор *while* дополнительный оператор вывода *writeln(n)*. В результате номер каждого очередного цикла выводится на экран и это позволит увидеть, что временами данное значение становится отрицательным, т.е. после того, как *n* принимает значение 32 767 (наибольшее из допустимого диапазона для типа *Integer*), при прибавлении очередной единицы при следующей итерации переменная n становится равной -32 768.

В программе используется стандартная функция *abs()* – возвращающая абсолютное значение (модуль) переданного ей.

5.8.2. Цикл с постусловием (PascalABC.NET)

Синтаксис:

<оператор *repeat*>::= *repeat*<оператор>[,<оператор>]…
until<логическое выражение>

Значение логического выражения должно изменяться в цикле, но, в отличие от оператора *while,* в этом операторе логическое выражение вычисляется и проверяется в конце цикла. Последовательность операторов между REPEAT и UNTIL выполняется повторно до тех пор, пока логическое выражение, вычисляемое после каждой итерации, не примет значение *true*. В отличие от оператора *while,* оператор *repeat* выполнится по крайне мере один раз (рис. 44).

Если условие *p* тождественно *false* (*repeat s until false*), то цикл будет бесконечным.

Для иллюстрации разницы между операторами цикла *while* и *repeat* рассмотрим соответствующие фрагменты программ, вычисляющие степени числа 3 в диапазоне между 1 и 300.

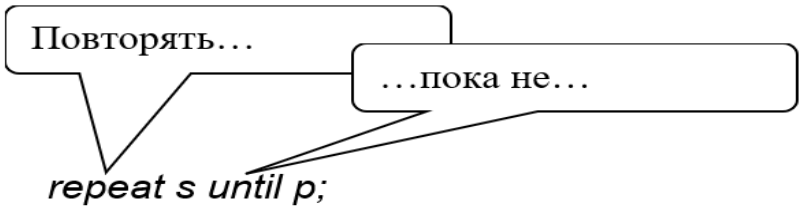


Рис. 44. Цикл с постусловием

Цикл while

```
a := 3;
while a < 300 do
        begin
                writeln(a);
                a := a * 3
        end;
```

Цикл repeat

```
a := 3;
repeat
        writeln(a);
        a := a * 3
until a >= 300;
```

Прежде всего бросается в глаза, что условия повторения цикла для двух случаев противоположны. Это объясняется тем, что в операторе *while* цикл выполняется до тех пор, пока условие не примет значение *false*, а в операторе *repeat* – пока условие не примет значение *true*. Также следует обратить внимание на то, что тело цикла *repeat* не требуется заключать в операторные скобки *begin*…*end*. Если в операторе *while* после ключевого слова *do* выполняется единственный оператор (и при циклическом выполнении нескольких действий приходится несколько операторов объединять в составной оператор), то в операторе *repeat* между ключевыми словами *repeat* и *until* можно ввести любое количество операторов без необходимости заключать их в операторные скобки.

Наконец, в операторе *repeat* после последнего оператора в теле цикла нет точки с запятой (;). Это еще одна особенность оператора *repeat* – перед ключевым словом *until* точка с запятой не обязательна (рис. 45).

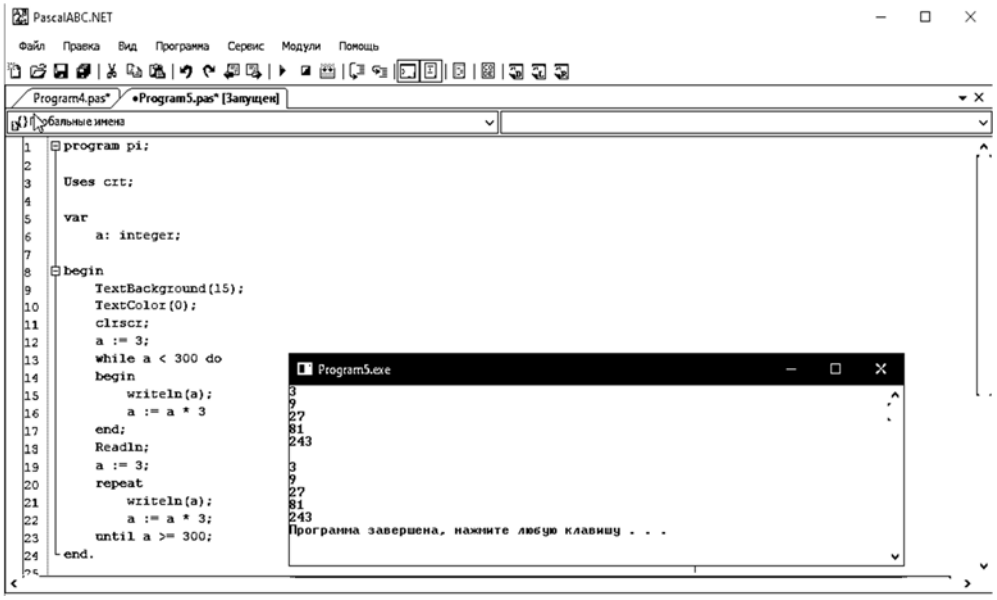


Рис. 45. Применение операторов While и Repeat

Пример 5.38.

Вычисление квадратного корня из неотрицательного числа.

Для вычисления квадратного корня из неотрицательного числа *x* необходим бесконечный ряд $A_1, A_2, A_3, \ldots$, где

$$A_1 = \frac{x+1}{2};\ A_i = \frac{1}{2}(A_{i-1} + \frac{x}{A_{i-1}}),$$ где *i = 2, 3, 4, … .*

Дело в том, что разница между очередным членом этого ряда и величиной корня квадратного от числа *x* по мере увеличения числа *i* становится все меньше.

Предположим, требуется определить корень квадратный от 2 (он приблизительно равен 1,4142135). Вычислим несколько первых членов *ряда* $A_1, A_2, A_3, \ldots$:

$$A_1 = \frac{2+1}{2} = 1.5,$$

$$A_2 = \frac{1}{2}(1.5 + \frac{2}{1.5}) = 1.4166666,$$

$$"A_3 = \frac{1}{2}(1.4166666 + \frac{2}{1.4166666}) = 1.4142156" "."$$

Как видно из приведенных расчетов, величина A*i* становится все ближе значению корня квадратного от двух.

Программа (рис. 46) вычисляет квадратные корни для 10 произвольных неотрицательных чисел. Это осуществляется оператором *while* и переменной *nv*.

Оператором *read(x)* вводим число, из которого предстоит извлечь квадратный корень. Оператор $a := (x + 1)/2$; вычисляет значение первого члена ряда. Далее оператором *repeat* вычисляем значения членов ряда, начиная со второго, до тех пор, пока не будет выполнено условие завершения цикла (абсолютная величина разницы между квадратом значения очередного члена ряда и значением переменной x станет меньше 0.000001 – точность вычисления):

```
program root;

Uses crt;

var
   a, x: real;
   nv: integer = 1;

begin
   Clrscr;
   while nv <= 10 do
   begin
      writeln;
      write('Корень из ');
      read(x);
      a := (x + 1) / 2;
      repeat
         a := 0.5 * (a + x / a)
      until abs(sqr(a) - x) < 0.000001;
      writeln(' = ', a);
      nv := nv + 1
   end;
end.
```

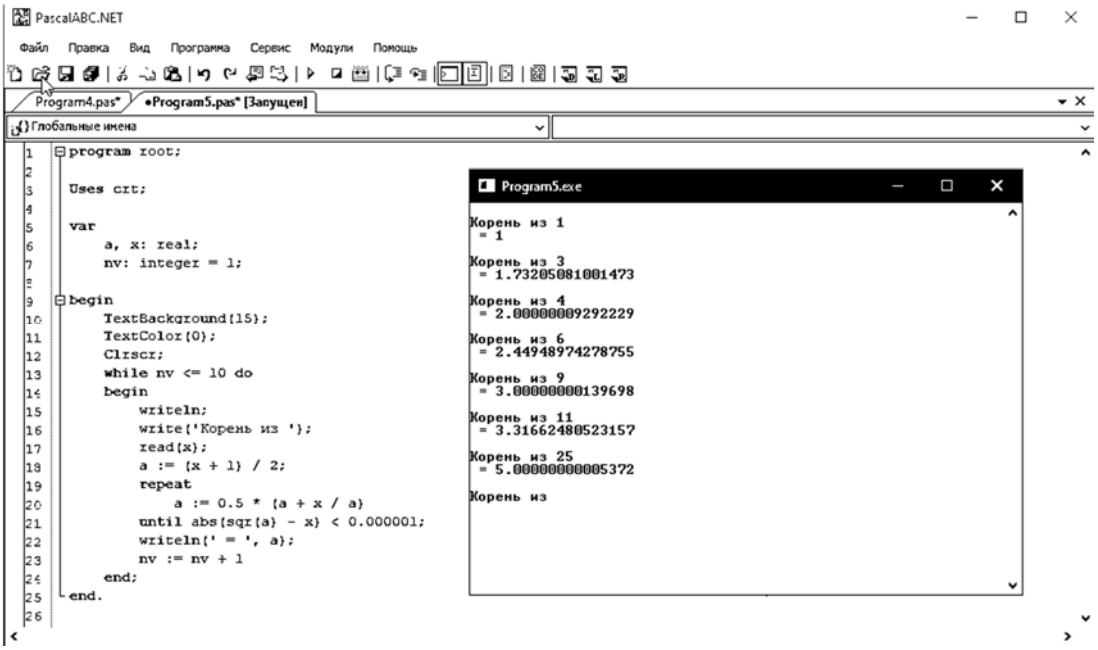


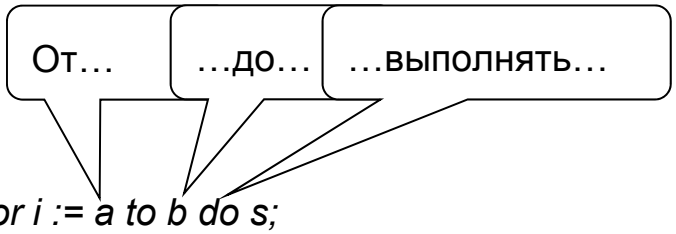
Рис. 46. Программа вычисления квадратного корня

5.8.3. Оператор цикла FOR (PascalABC.NET)

Синтаксис:

<оператор *for*>::=*for*<параметр цикла>:=<список цикла>*do*<оператор>
<параметр цикла>::=<идентификатор>
<список цикла>::=<первое значение>*to*<последнее значение>|
или <первое значение>*downto*<последнее значение>
<первое значение>::=<выражение>
<последнее значение>::=<выражение>
<оператор> - оператор, выполняемый многократно (в цикле).

Оператор цикла без условия *FOR* (его еще называют оператор цикла с параметром) выглядит следующим образом:



for i := a to b do s;

При выполнении этого оператора сначала вычисляется некоторое начальное значение *a*, которое присваивается переменной *i*, называемой *параметром цикла*. Важно помнить – в операторе цикла

параметру цикла ***присваивается*** некоторое значение, т.е. тут употребляется оператор присваивания. Параметр цикла – переменная, следовательно, ее имя – это идентификатор, и он может быть, как любой символ, так и их сочетание, допустимое с точки зрения синтаксиса *PascalABC.NET* (например, *i, j, ii, jj06* и т.д.). Затем вычисляется конечное значение *b* и проверяется, имеет ли место равенство *i* = *b*. Если равенства нет (*i* < *b*), то выполняется оператор *s*, который может быть составным, и переменная *i* увеличивается на единицу. Затем повторяется проверка равности переменной *i* значению *b*. В случае равенства (*i* = *b*) оператор цикла выполняет оператор *s* и передает управление оператору, следующему за оператором *for*, в противном случае оператор цикла повторяется до тех пор, пока не будет достигнуто равенство *i=b*.

Параметр цикла *i*, а также начальное и конечное значения (*a* и *b*) могут принадлежать любому порядковому типу (например, *Integer* или *Char*), но при этом они должны быть все одного типа. Если начальное значение превышает, то оператор *s* не выполняется ни разу.

В теле цикла запрещается явно изменять значение параметра цикла, например, с помощью оператора присваивания.

В рассмотренном случае параметр цикла изменял значения по возрастанию. В случае убывания значений параметра цикла, т.е. когда начальное значение превышает конечное значение, применяют другую форму оператора *for:*

```
for i := a downto b do s;
```

Здесь, чтобы выполнялся оператор *s*, начальное значение *a* должно превышать конечное значение *b*. Кроме того, в этом случае параметр цикла с каждой итерацией уменьшается на единицу, пока не становится равным конечному значению.

Оператор цикла с параметром следует использовать тогда, когда заранее точно известно, сколько раз должно быть выполнено тело цикла:

```
for i := 1 to 5 do z := sqr(x);
for i := m downto c do write(i);
```

Во втором примере (и конкретно в этом примере) оператор цикла с параметром использован для вывода строчных букв латинского алфавита в обратном порядке (от *m* до *c*).

Возможно использование *вложенных циклов*. Это подразумевает, что существуют внешний цикл и один или несколько внутренних циклов. Каждое повторение внешнего цикла означает завершение всех внутренних циклов; при этом всем выражениям, которые управляют внутренними циклами, вновь присваиваются начальные значения.

5.8.4. Оператор Break, Continue (PascalABC.NET)

Оператор *break* осуществляет немедленный выход из циклов *while*, *repeat* и *for*. Его можно использовать только внутри тела цикла.

Оператор *continue* начинает новую итерацию цикла, даже если предыдущая не была завершена. Его можно использовать только внутри тела цикла.

Пример 5.39

Дано натуральное число *N*. Определить, является ли оно простым[10]. Алгоритм решения этой задачи заключается в том, что число *N* делится на параметр цикла *i*, изменяющийся в диапазоне от 2 до *N/2*. Если среди значений параметра не найдется ни одного числа, делящего заданное число нацело, то *N* – простое число, иначе оно таковым не является. В алгоритме (рис. 47) предусмотрено два выхода из цикла: первый – при исчерпании всех значений параметра, второй – досрочный. Нет смысла продолжать цикл, если будет найден хотя бы один делитель из указанной области изменения параметра цикла (рис. 48).

[10] Натуральное число *N* называется простым, если оно делится нацело без остатка только на единицу и *N*.

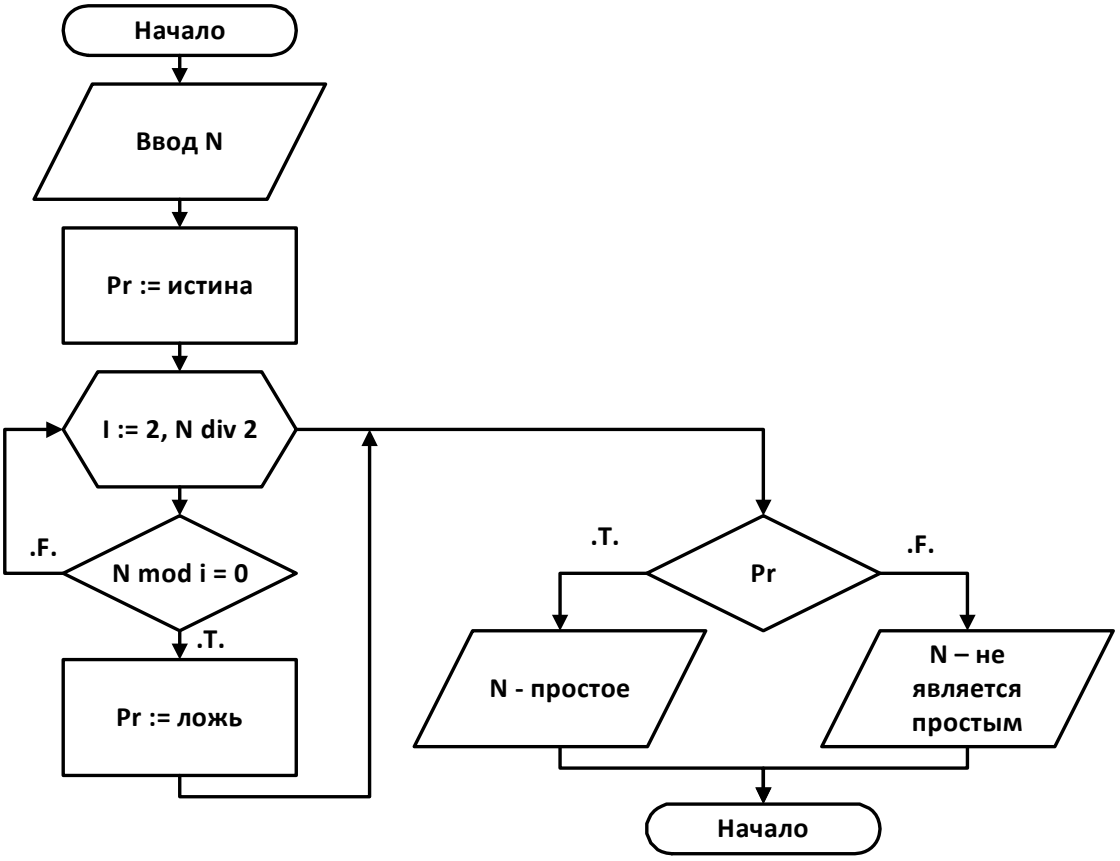


Рис. 47. Блок-схема определения простого числа

```
program prostoe;

Uses crt;

var
   N, i: integer;
   Pr: boolean;

begin
   clrscr;
   write('N=');
   readln(N);
   Pr := true;                  {Предположим, что число простое}
   for i := 2 to N div 2 do     {Операции div и mod}
      if N mod i = 0 then       {Если найден хоть один делитель, то…}
      begin
         Pr := false;           {Число простым не является и}
         break;                 {досрочный выход из цикла}
      end;
   if Pr    then writeln('Число ', N, ' - простое')
   else writeln('Число ', N, ' – простым не является');
end.
```

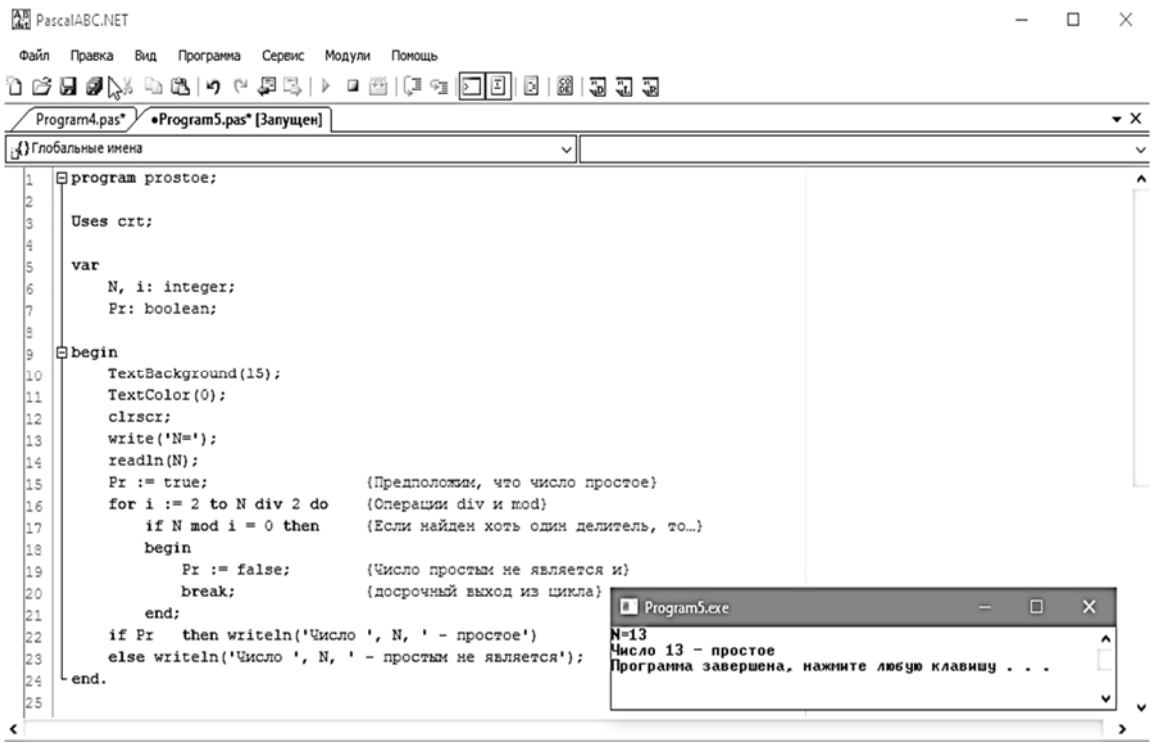


Рис. 48. Программа определения простого числа

5.8.5. Циклы в VBA

Встречаются ситуации, когда от программы *VBA* требуется совершить несколько раз подряд один и тот же набор действий (то есть повторить несколько раз один и тот же блок кода). Это может быть сделано при помощи циклов *VBA*.

К циклам VBA относятся:

- цикл *For;*
- цикл *Do While;*
- цикл *Do Until.*

Далее мы подробно рассмотрим каждый из этих циклов.

Оператор цикла «*For*» в *Visual Basic.*

Структура оператора цикла *For* в *Visual Basic* может быть организована в одной из двух форм: как цикл *For … Next* или как цикл *For Each*.

5.8.5.1. Цикл For…Next

Цикл *For … Next* использует переменную, которая последовательно принимает значения из заданного диапазона. С каждой сменой значения переменной выполняются действия, заключённые в теле цикла. Это легко понять из простого примера:

```
For i = 1 To 10
   Total = Total + iArray(i)
Next i
```

В этом простом цикле *For … Next* используется переменная *i*, которая последовательно принимает значения 1, 2, 3, … 10, и для каждого из этих значений выполняется код *VBA*, находящийся внутри цикла. Таким образом, данный цикл суммирует элементы массива *iArray* в переменной *Total*.

В приведённом выше примере шаг приращения цикла не указан, поэтому для пошагового увеличения переменной *i* от 1 до 10 по умолчанию используется приращение 1. Однако в некоторых случаях требуется использовать другие значения приращения для цикла. Это можно сделать при помощи ключевого слова *Step*, как показано в следующем простом примере:

```
For d = 0 To 10 Step 0.1
   dTotal = dTotal + d
Next d
```

Так как в приведённом выше примере задан шаг приращения равный 0.1, то переменная *dTotal* для каждого повторения цикла принимает значения 0.0, 0.1, 0.2, 0.3, … 9.9, 10.0.

Для определения шага цикла в *VBA* можно использовать отрицательную величину, например, вот так:

```
For i = 10 To 1 Step -1
   iArray(i) = i
Next i
```

Здесь шаг приращения равен -1, поэтому переменная *i* с каждым повторением цикла принимает значения 10, 9, 8, … 1.

5.8.5.2. Цикл For Each

Цикл *For Each* похож на цикл *For … Next*, но вместо того, чтобы перебирать последовательность значений для *переменной-счётчика*, цикл *For Each* выполняет набор действий для каждого объекта

из указанной группы объектов. В следующем примере при помощи цикла *For Each* выполняется перечисление всех листов в текущей рабочей книге *Excel*:

```
Dim wSheet As Worksheet
…
For Each wSheet in Worksheets
   MsgBox "Найден лист: " & wSheet.Name
Next wSheet
```

5.8.5.3. Оператор Exit For

Оператор *Exit For* применяется для прерывания цикла. Как только в коде встречается этот оператор, программа завершает выполнение цикла и переходит к выполнению операторов, находящихся в коде сразу после данного цикла. Это можно использовать, например, для поиска определённого значения в массиве (см. раздел 5.8.6). Для этого при помощи цикла просматривается каждый элемент массива. Как только искомый элемент найден, просматривать остальные нет необходимости – цикл прерывается.

Применение оператора *Exit For* продемонстрировано в следующем примере. Здесь цикл перебирает 100 записей массива и сравнивает каждую со значением переменной *dVal*. Если совпадение найдено, то цикл прерывается:

```
For i = 1 To 100
   If dValues(i) = dVal Then
         IndexVal = i
         Exit For
   End If
Next i
```

5.8.5.4. Цикл Do While

Цикл *Do While* выполняет блок кода до тех пор, пока выполняется заданное условие. Далее приведён пример процедуры *Sub*, в которой при помощи цикла *Do While* выводятся последовательно числа Фибоначчи, не превышающие 1000 (рис. 49).

В приведённом примере условие *iFib_Next* < 1000 проверяется в начале цикла. Поэтому если бы первое значение *iFib_Next* было бы больше 1000, то цикл бы не выполнялся ни разу:

```
'Процедура Sub выводит числа Фибоначчи не превышающие 1000
Sub Fibonacci()
    Dim i As Integer
    'счётчик для обозначения позиции элемента в последовательности
    Dim iFib As Integer
    'хранит текущее значение последовательности
    Dim iFib_Next As Integer
    'хранит следующее значение последовательности
    Dim iStep As Integer
    'хранит размер следующего приращения

    'инициализируем переменные i и iFib_Next
    i = 1
    iFib_Next = 0
    'цикл Do While будет выполняться до тех пор, пока значение
    'текущего числа Фибоначчи не превысит 1000
    Do While iFib_Next < 1000
            If i = 1 Then
            'особый случай для первого элемента последовательности
                    iStep = 1
                    iFib = 0
            Else
            'сохраняем размер следующего приращения перед тем, как
            перезаписать текущее значение последовательности
                    iStep = iFib
                    iFib = iFib_Next
            End If
            'выводим текущее число Фибоначчи в столбце A активного
            рабочего листа в строке с индексом i
            Cells(i, 1).Value = iFib
            'вычисляем следующее число Фибоначчи и увеличиваем ин
            декс позиции элемента на 1
            iFib_Next = iFib + iStep
            i = i + 1
    Loop
End Sub
```

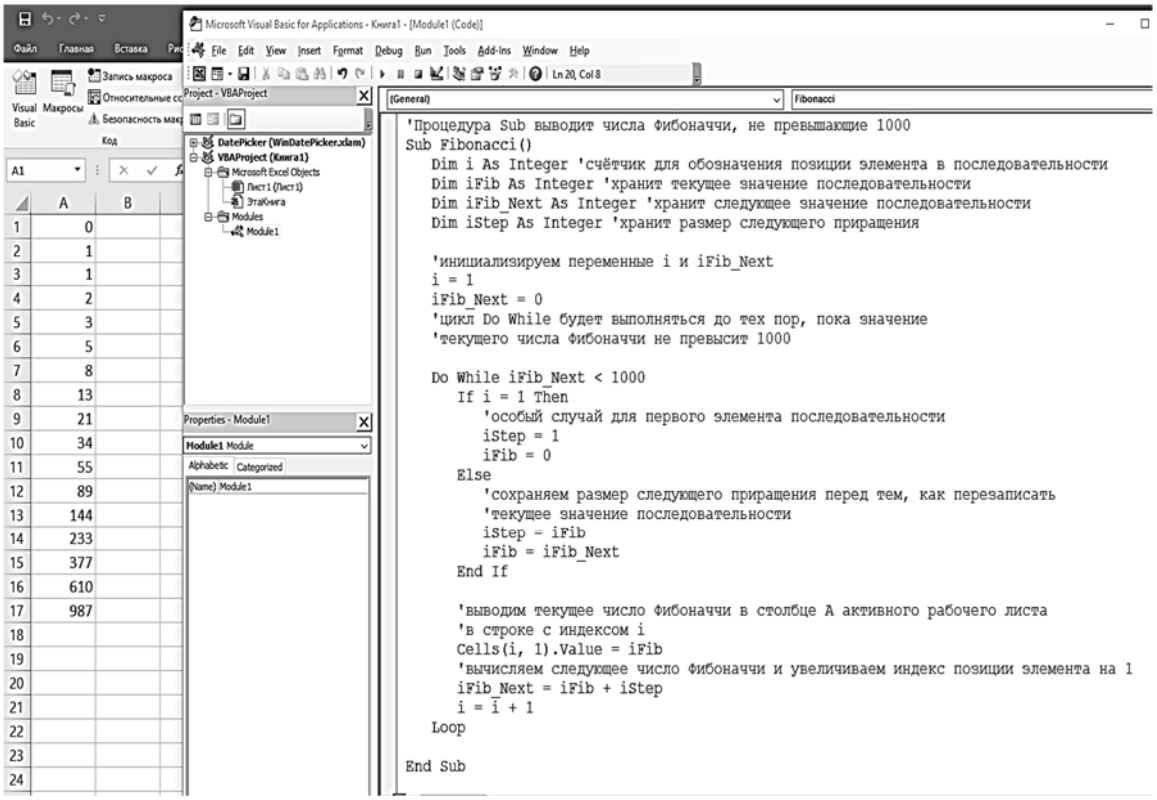


Рис. 49. Программа Числа Фибоначчи (Excel VBA)

Другой способ реализовать цикл *Do While* – поместить условие не в начале, а в конце цикла. В этом случае цикл будет выполнен хотя бы раз не зависимо от того, выполняется ли условие.

Схематично такой цикл *Do While* с проверяемым условием в конце будет выглядеть вот так:

```
Do
   ...
Loop While iFib_Next < 1000
```

5.8.5.5. Цикл Do Until

Цикл *Do Until* очень похож на цикл *Do While*: блок кода в теле цикла выполняется раз за разом до тех пор, пока заданное условие выполняется (результат условного выражения равен *True*). В следующей процедуре *Sub* при помощи цикла *Do Until* извлекаются значения из всех ячеек столбца A рабочего листа до тех пор, пока в столбце не встретится пустая ячейка:

```
iRow = 1
Do Until IsEmpty(Cells(iRow, 1))
```

```
    'Значение текущей ячейки сохраняется в массиве dCellValues
    dCellValues(iRow) = Cells(iRow, 1).Value
    iRow = iRow + 1
  Loop
```

В приведённом выше примере условие *IsEmpty(Cells(iRow, 1))* находится в начале конструкции *Do Until*, следовательно, цикл будет выполнен хотя бы один раз, если первая взятая ячейка не пуста.

Однако, как было показано в примерах цикла *Do While*, в некоторых ситуациях нужно, чтобы цикл был выполнен хотя бы один раз, не зависимо от первоначального результата условного выражения. В таком случае условное выражение нужно поместить в конце цикла вот так:

```
Do
  ...
Loop Until IsEmpty(Cells(iRow, 1))
```

Пример 5.40.

Дано ε. Определить с точностью ε.

$$y = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^n} + \cdots$$

```
Sub ryad()
  Dim y As Single
  Dim n As Integer
  Dim eps As Single
  Dim sl As Single
  y = 0
  n = 1
  eps = Cells(1, 2)

  Do
        sl = 1 / (2 ^ n)
        Cells(n + 1, 10) = sl
        y = y + sl
        n = n + 1
  Loop While Abs(sl) > eps

  Cells(4, 2) = y
End Sub
```

Блок-схема программы представлена на рис. 50.

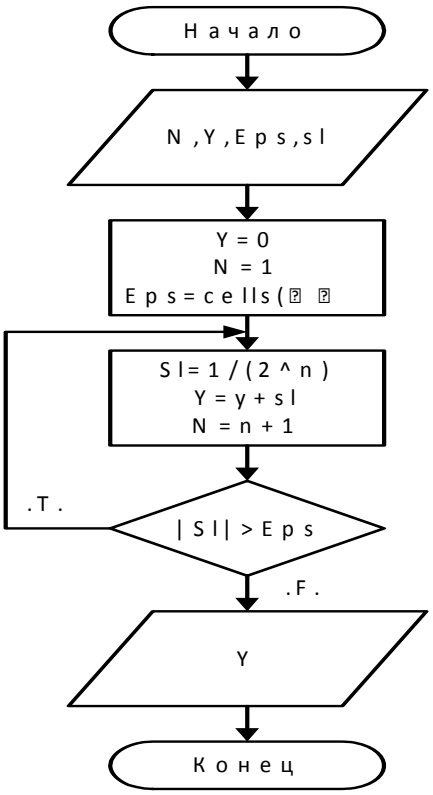


Рис. 50. Блок-схема программы с циклом с постусловием

5.8.6. Массивы (PascalABC.NET)

Массив представляет собой набор элементов одного типа, каждый из которых имеет свой номер, называемый индексом (индексов может быть несколько, тогда массив называется многомерным).

Массивы в *PascalABC.NET* делятся на статические и динамические.

При выходе за границы изменения индекса в *PascalABC.NET* всегда генерируется исключение.

Статические массивы

Статические массивы в отличие от динамических задают свой размер непосредственно в типе. Память под такие массивы выделяется сразу при описании.

Тип статического массива конструируется следующим образом:

```
array [тип индекса1, ..., тип индексаN] of базовый тип
```

Тип индекса должен быть порядковым. Обычно тип индекса является диапазонным и представляется в виде $a..b$, где a и b - константные выражения целого, символьного или перечислимого типа. Например:

```
type
  MyEnum = (w1,w2,w3,w4,w5);
  Arr = array [1..10] of integer;
var
  a1,a2: Arr;
  b: array ['a'..'z',w2..w4] of string;
  c: array [1..3] of array [1..4] of real;
```

При описании можно также задавать инициализацию массива значениями:

```
var
  a: Arr := (1,2,3,4,5,6,7,8,9,0);
  cc: array [1..3,1..4] of real := ((1,2,3,4), (5,6,7,8), (9,0,1,2));
```

Статические массивы одного типа можно присваивать друг другу, при этом будет производиться копирование содержимого одного массива в другой:

```
a1 := a2;
```

Процедура $write$ выводит статический массив, заключая элементы в квадратные скобки и разделяя их запятыми:

```
var a: Arr := (1,2,3,4,5,6,7,8,9,0);
var m := array [1..3,1..3] of integer :=
        ((1,2,3),(4,5,6),(7,8,9));
writeln(a);                                    [1,2,3,4,5]
writeln(m);                                    [[1,2,3],[4,5,6],[7,8,9]]
```

Для доступа к нижней и верхней границам размерности одномерного массива используются функции Low и $High$.

Динамические массивы

Тип динамического массива конструируется следующим образом:

```
array of тип элементов  (одномерный массив)
array [,] of тип элементов  (двумерный массив)
```

и т.д.

Переменная типа «*динамический массив*» представляет собой ссылку. Поэтому динамический массив нуждается в инициализации (выделении памяти под элементы).

Для выделения памяти под динамический массив используется два способа. Первый способ использует операцию new в стиле вызова конструктора класса:

```
var
  a: array of integer;
  b: array [,] of real;
begin
  a := new integer[5];
  b := new real[4,3];
end.
```

Данный способ хорош тем, что позволяет совместить описание массива и выделение под него памяти:

```
var
  a: array of integer := new integer[5];
  b: array [,] of real := new real[4,3];
```

Описание типа можно при этом опускать - тип автовыводится:

```
var
  a := new integer[5];
  b := new real[4,3];
```

Второй способ выделения памяти под динамический массив использует стандартную процедуру $SetLength$:

```
SetLength(a,10);
SetLength(b,5,3);
```

Элементы массива при этом заполняются значениями по умолчанию.

Процедура $SetLength$ обладает тем преимуществом, что при её повторном вызове старое содержимое массива сохраняется.

Можно инициализировать динамический массив при выделении под него память операцией new:

```
a := new integer[3](1,2,3);
b := new real[4,3] ((1,2,3),(4,5,6),(7,8,9),(0,1,2));
```

Инициализацию динамического массива в момент описания можно проводить в сокращённой форме:

```
var
  a: array of integer := (1,2,3);
  b: array [,] of real := ((1,2,3),(4,5,6),(7,8,9),(0,1,2));
  c: array of array of integer := ((1,2,3),(4,5),(6,7,8));
```

При этом происходит выделение памяти под указанное справа количество элементов.

Инициализация одномерного массива проще всего осуществляется стандартными функциями *Seq*..., которые выделяют память нужного размера и заполняют массив указанными значениями:

```
var a := Arr(1,3,5,7,8);                          // array of integer
var s := Arr('Иванов','Петров','Сидоров');  // array of string
var b := ArrFill(777,5);                          // b = [777,777,777,777,777]
var r := ArrRandom(10);                           // заполнение 10 случайными
                                                  // целыми в диапазоне от 0 до
                                                  // 99
```

В таком же стиле можно инициализировать массивы массивов:

```
var a := Arr(Arr(1,3,5),Arr(7,8),Arr(5,6)); // array of array of integer
```

Длина динамического массива

Динамический массив помнит свою длину (*n*-мерный динамический массив помнит длину по каждой размерности). Длина массива (количество элементов в нем) возвращается стандартной функцией *Length* или свойством *Length*:

```
l := Length(a);
l := a.Length;
```

Для многомерных массивов длина по каждой размерности возвращается стандартной функцией *Length* с двумя параметрами или методом *GetLength(i):*

```
l := Length(a,0);
l := a.GetLength(0);
```

Ввод динамического массива

После выделения памяти ввод динамического массива можно осуществлять традиционно в цикле:

```
for var i:=0 to a.Length-1 do
   read(a[i]);
```

Ввод динамического массива можно осуществлять с помощью стандартной функции *ReadSeqInteger*:

```
var a := ReadSeqInteger(10);
```

При этом под динамический массив выделяется память нужного размера.

Вывод динамического массива

Процедура *write* выводит динамический массив, заключая элементы в квадратные скобки и разделяя их запятыми:

```
var a := Arr(1,3,5,7,9);
writeln(a); // [1,3,5,7,9]
```

n-мерный динамический массив выводится так, что каждая размерность заключается в квадратные скобки:

```
var m := new integer[3,3] ((1,2,3),(4,5,6),(7,8,9));
writeln(m); // [[1,2,3],[4,5,6],[7,8,9]]
```

Динамический массив можно выводить также методом расширения *Print* или *Println*:

```
a.Println;
```

При этом элементы по умолчанию разделяются пробелами, но можно это изменить, задав параметр *Print*, являющийся разделителем элементов. Например:

```
a.Print(NewLine);
```

выводит каждый элемент на отдельной строке.

Массивы массивов

Если объявлен массив массивов

```
var c: array of array of integer;
```

то его инициализацию можно провести только с помощью *SetLength*:

```
SetLength(c,5);
for i := 0 to 4 do
   SetLength(c[i],3);
```

Для инициализации такого массива с помощью *new* следует ввести имя типа для *array of integer*:

```
type IntArray = array of integer;
var c: array of IntArray;
...
c := new IntArray[5];
for i := 0 to 4 do
   c[i] := new integer[3];
```

Инициализацию массива массивов можно также проводить в сокращенной форме:

```
var
   c: array of array of integer := ((1,2,3),(4,5),(6,7,8));
```

Присваивание динамических массивов

Динамические массивы одного типа можно присваивать друг другу, при этом обе переменные-ссылки будут указывать на одну память:

```
var a1: array of integer;
var a2: array of integer;
a1 := a2;
```

Следует обратить внимание, что для динамических массивов принята структурная эквивалентность типов: можно присваивать друг другу и передавать в качестве параметров подпрограмм динамические массивы, совпадающие по структуре.

Чтобы одному динамическому массиву присвоить копию другого массива, следует воспользоваться стандартной функцией *Copy*:

```
a1 := Copy(a2);
```

5.8.7. Массивы (VBA)

Массив VBA – это тип переменной. Используется для хранения списков данных одного типа. Примером может быть сохранение списка стран или списка итогов за неделю. В VBA обычная переменная может хранить только одно значение за раз. В следующем примере показана переменная, используемая для хранения оценок ученика:

```
Dim Student1 As Integer
Student1 = 55
```

Переменная *Student* может хранить только 1 значение за раз.

Если мы хотим сохранить оценки другого ученика, то нам нужно создать вторую переменную.

Проблема с использованием одной переменной для каждого учащегося заключается в том, что вам необходимо добавить код для каждого учащегося. Поэтому, если в приведенном выше примере у вас будет тысяча студентов, вам понадобится три тысячи строк кода! К счастью, у нас есть массивы, чтобы сделать нашу жизнь проще. Массивы позволяют нам хранить список элементов данных в одной структуре:

```
Public Sub StudentMarksArr()
   With ThisWorkbook.Worksheets("Лист1")
         ' Объявите массив для хранения оценок для 5 студентов
         Dim Students(1 To 5) As Integer
         ' Читайте оценки учеников из ячеек C3: C7 в массив
         Dim i As Integer
         For i = 1 To 5
               Students(i) = .Range("C2").Offset(i)
         Next i
         ' Распечатывать оценки студентов из массива
         Debug.Print "Оценки студентов"
         For i = LBound(Students) To UBound(Students)
               Debug.Print Students(i)
         Next i
   End With
End Sub
```

Преимущество этого кода в том, что он будет работать для любого количества студентов. Если нам нужно изменить этот код для работы с 1000 студентами, то нам нужно всего лишь изменить (от 1 до 5) на (от 1 до 1000) в декларации.

В VBA есть два типа массивов:

- ***статический*** – массив фиксированного размера;
- ***динамический*** – массив, в котором размер задается во время выполнения.

Разница между этими массивами в основном в том, как они создаются. Доступ к значениям в обоих типах массивов абсолютно одинаков. В следующих разделах мы рассмотрим оба типа.

Статический массив объявляется следующим образом:

```
Public Sub DecArrayStatic()
   ' Создать массив с местоположениями 0,1,2,3
   Dim arrMarks1(0 To 3) As Long
   ' По умолчанию от 0 до 3, то есть местоположения 0,1,2,3
   Dim arrMarks2(3) As Long
   ' Создать массив с местоположениями 1,2,3,4,5
   Dim arrMarks1(1 To 5) As Long
   ' Создать массив с местоположениями 2,3,4
   'Это редко используется
   Dim arrMarks3(2 To 4) As Long
End Sub
```

Как видите, размер указывается при объявлении статического массива. Проблема в том, что вы никогда не можете быть заранее уверены, какой размер вам нужен. Каждый раз, когда вы запускаете макрос, у вас могут быть разные требования к размеру.

Если вы не используете все расположения массива, то ресурсы тратятся впустую. Если вам нужно больше места, то вы можете использовать *ReDim*, но это, по сути, создает новый статический массив.

Динамический массив не имеет таких проблем. Вы не указываете размер, когда объявляете. Поэтому вы можете увеличивать и уменьшать по мере необходимости. При этом стоит помнить, изменение размера массива не изменяет тип массива, который был объявлен вначале (оператор *Dim*):

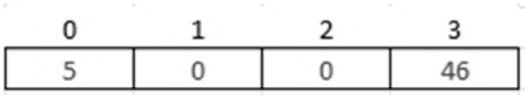
```
Public Sub DecArrayDynamic()
   ' Объявить динамический массив
   Dim arrMarks() As Long
   ' Установите размер массива, когда вы будете готовы
   ReDim arrMarks(0 To 5)
End Sub
```

Динамический массив не выделяется, пока вы не используете оператор *ReDim*. Преимущество в том, что вы можете подождать, пока не узнаете количество элементов, прежде чем устанавливать размер массива. Со статическим массивом вы должны указать размер заранее.

Чтобы присвоить значения массиву, вы используете номер местоположения (пересечении строки и столбца). Вы присваиваете значение для обоих типов массивов одинаково.

```
Public Sub AssignValue()
        ' Объявить массив с местоположениями 0,1,2,3
        Dim arrMarks(0 To 3) As Long
        ' Установите значение позиции 0
        arrMarks(0) = 5
        ' становите значение позиции 3
        arrMarks(3) = 46
        ' Это ошибка, так как нет местоположения 4
        arrMarks(4) = 99
End Sub
```

Заполнение ячеек памяти значениями, присваиваемыми элементам массива:



Номер места называется индексом. Последняя строка в примере выдаст ошибку «Индекс вне диапазона», так как в примере массива нет местоположения 4.

Вы можете использовать функцию *Array* для заполнения массива списком элементов. Вы должны объявить массив как тип *Variant*. Следующий код показывает, как использовать эту функцию.

```
Dim arr1 As Variant
arr1 = Array("Апельсин", Персик","Груша")
```

Заполнение ячеек памяти значениями, присваиваемыми элементам массива:


```
Dim arr2 As Variant
arr2 = Array(5, 6, 7, 8, 12)
```


Массив, созданный функцией *Array*, начнется с нулевого индекса, если вы не используете *Option Base 1* в верхней части вашего модуля. Затем он начнется с первого индекса. В программировании, как правило, считается плохой практикой иметь ваши реальные данные в коде. Однако иногда это полезно, когда вам нужно быстро про-

тестировать некоторый код. Функция *Split* используется для разделения строки на массив на основе разделителя. Разделитель – это символ, такой как запятая или пробел, который разделяет элементы. Следующий код разделит строку на массив из четырех элементов, при этом индексация элементов массива начинается с 0:

```
Dim s As String
s = "Красный,Желтый,Зеленый,Синий"
Dim arr() As String
arr = Split(s, ",")
```

Заполнение ячеек памяти значениями присваиваемыми элементам массива:

0	1	2	3
Красный	Желтый	Зеленый	Синий

Функция *Split* обычно используется, когда вы читаете из *cvs* или *txt*-файла, разделенного запятыми, или из другого источника, который предоставляет список элементов, разделенных одним и тем же символом.

5.8.7.1. Использование циклов с массивами (VBA)

Использование цикла *For* обеспечивает быстрый доступ ко всем элементам массива. Вот где сила использования массивов становится очевидной. Мы можем читать массивы с десятью значениями или десятью тысячами значений, используя те же несколько строк кода. В *VBA* есть две функции: *LBound* и *UBound*. Эти функции возвращают самый маленький и самый большой индекс в массиве. В массиве *arrMarks* (от 0 до 3) *LBound* вернет 0, а *UBound* вернет 3. В следующем примере случайные числа присваиваются массиву с помощью цикла. Затем он печатает эти числа, используя второй цикл:

```
Public Sub ArrayLoops()
   ' Объявить массив
   Dim arrMarks(0 To 5) As Long
   ' Заполните массив случайными числами
   Dim i As Long
   For i = LBound(arrMarks) To UBound(arrMarks)
         arrMarks(i) = 5 * Rnd
   Next i
   ' Распечатайте значения в массиве
   Debug.Print "Место нахождения", "Значение"
   For i = LBound(arrMarks) To UBound(arrMarks)
         Debug.Print i, arrMarks(i)
   Next i
End Sub
```

Функции *LBound* и *UBound* очень полезны. Их использование

означает, что наши циклы будут работать правильно с любым размером массива. Реальное преимущество заключается в том, что, если размер массива изменяется, то нам не нужно менять код для печати значений. Цикл будет работать для массива любого размера до тех пор, пока вы используете эти функции.

Вы можете использовать цикл *For Each* с массивами. Важно помнить, что он доступен только для **чтения**. Это означает, что вы не можете изменить значение в массиве. В следующем коде значение метки изменяется, но оно не меняет значение в массиве:

```
For Each mark In arrMarks
   ' Не изменит значение массива
   mark = 5 * Rnd
Next mark
```

Цикл *For Each* отлично подходит для чтения массива:

```
Dim mark As Variant
For Each mark In arrMarks
   Debug.Print mark
Next mark
```

Функция *Erase* может использоваться для массивов, но она работает по-разному в зависимости от типа массива. Для статического массива функция *Erase* сбрасывает все значения по умолчанию. Если массив состоит из целых чисел, то все значения устанавливаются в ноль. Если массив состоит из строк, то все строки устанавливаются в «» и так далее. Для динамического массива функция удаления стирает память. То есть она удаляет массив. Если вы хотите использовать его снова, то вы должны использовать *ReDim* для выделения памяти.

Давайте рассмотрим пример статического массива. Мы используем *Erase* после установки значений. Когда значения массива будут распечатаны, то все они будут равны нулю:

```
Public Sub EraseStatic()
   ' Объявить массив
   Dim arrMarks(0 To 3) As Long
   ' Заполним массив случайными числами
   Dim i As Long
   For i = LBound(arrMarks) To UBound(arrMarks)
         arrMarks(i) = 5 * Rnd
   Next i
   ' ВСЕ ЗНАЧЕНИЯ УСТАНОВЛЕНЫ НА НОЛЬ
   Erase arrMarks
```

```
    ' Распечатаем значения - там все теперь ноль
    Debug.Print "Место нахождения", "Значение"
    For i = LBound(arrMarks) To UBound(arrMarks)
            Debug.Print i, arrMarks(i)
    Next i
End Sub
```

Теперь мы попробуем тот же пример с динамикой. После того, как мы используем *Erase*, все места в массиве были удалены. Нам нужно использовать *ReDim*, если мы хотим использовать массив снова. Если мы попытаемся получить доступ к членам этого массива, то мы получим ошибку «Индекс вне диапазона»:

```
Public Sub EraseDynamic()
    ' Объявить массив
    Dim arrMarks() As Long
    ReDim arrMarks(0 To 3)
    ' Заполнить массив случайными числами
    Dim i As Long
    For i = LBound(arrMarks) To UBound(arrMarks)
            arrMarks(i) = 5 * Rnd
    Next i
    ' arrMarks теперь освобожден. Места не существуют.
    Erase arrMarks
End Sub
```

Если мы используем *ReDim* для существующего массива, то массив и его содержимое будут удалены. В следующем примере второй оператор *ReDim* создаст совершенно новый массив. Исходный массив и его содержимое будут удалены:

```
Sub UsingRedim()
    Dim arr() As String
    ' Установить массив в слоты от 0 до 2
    ReDim arr(0 To 2)
    arr(0) = "Яблоко"
    ' Массив с яблоком теперь удален
    ReDim arr(0 To 3)
End Sub
```

Если мы хотим расширить размер массива без потери содержимого, то мы можем использовать ключевое слово *Preserve*. Когда мы используем *Redim Preserve*, то новый массив должен начинаться с того же начального размера. Например, мы не можем сохранить от (0 до 2) до (от 1 до 3) или до (от 2 до 10), поскольку они являются различными начальными размерами. В следующем коде мы создаем

массив с использованием *ReDim*, а затем заполняем массив типами фруктов. Затем мы используем *Preserve* для увеличения размера массива, чтобы не потерять оригинальное содержимое:

```
Sub UsingRedimPreserve()
    Dim arr() As String
    ' Установить массив в слоты от 0 до 1
    ReDim arr(0 To 2)
    arr(0) = "Яблоко"
    arr(1) = "Апельсин"
    arr(2) = "Груша"
    ' Изменение размера и сохранение исходного содержимого
    ReDim Preserve arr(0 To 5)
End Sub
```

На рис. 51 приведён снимок экрана с точкой останова выполнения программы для иллюстрации содержимого массива *arr* исходными значениями, их всего три.

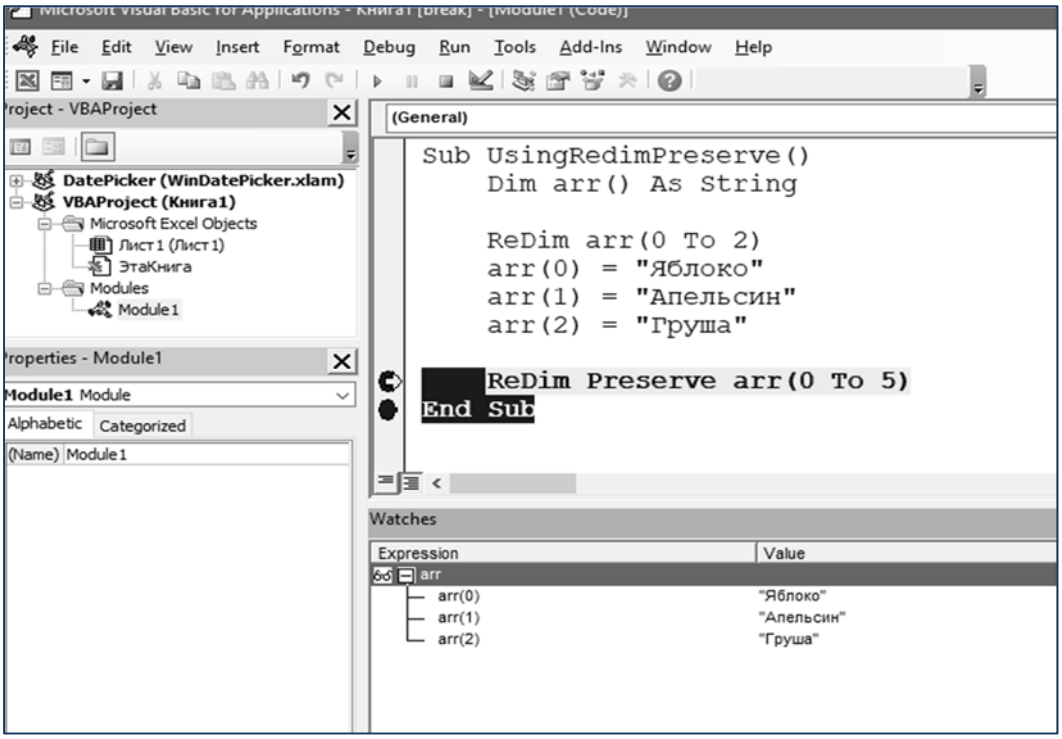


Рис. 51. Исходное состояние массива

На рис. 52 видно, что исходное содержимое массива было сохранено и к исходному содержимому добавлено два новых элемента.

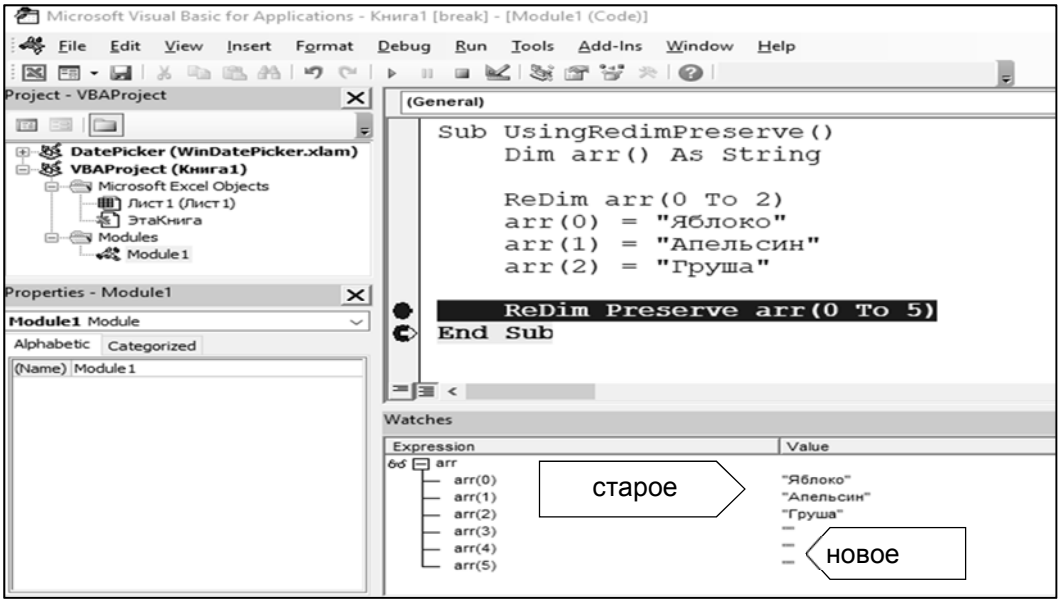


Рис. 52. Содержимое массива «сохранено» и изменён размер

Preserve работает только с верхней границей массива. Например, если у вас есть двумерный массив, то вы можете сохранить только второе измерение, как показано в следующем примере:

```
Sub Preserve2D()
   Dim arr() As Long
   ' Установите начальный размер
   ReDim arr(1 To 2, 1 To 5)
   ' Изменить размер верхнего измерения
   ReDim Preserve arr(1 To 2, 1 To 10)
End Sub
```

Если мы попытаемся использовать *Preserve* на нижней границе, то мы получим ошибку «Индекс вне диапазона». В следующем коде мы используем *Preserve* для первого измерения. Запуск этого кода приведет к ошибке «Индекс вне диапазона»:

```
Sub Preserve2DError()
   Dim arr() As Long
   ' Установите начальный размер
   ReDim arr(1 To 2, 1 To 5)
   ' Ошибка «Вне диапазона»
   ReDim Preserve arr(1 To 5, 1 To 5)
End Sub
```

Когда мы читаем из диапазона в массив, он автоматически создает двумерный массив, даже если у нас есть только один столбец. Применяются те же правила сохранения. Мы можем использовать *Preserve* только на верхней границе, как показано в следующем примере:

```
Sub Preserve2DRange()
   Dim arr As Variant
   ' Назначить диапазон массиву
   arr = Sheet1.Range("A1:A5").Value
   ' Preserve будет работать только на верхней границе
   ReDim Preserve arr(1 To 5, 1 To 7)
End Sub
```

Пример 5.41.

Дан массив, определить в нём все минимальные и максимальные значения, определить количество минимальных и максимальных значений.

```
Sub minmax()
   Dim a(), amin, amax, i, k, l, n
   Dim a1(), a2(), k1, l2
   n = Cells(Rows.Count, 1).End(xlUp).Row
   ReDim a(n)

   For i = 1 To n
         a(i) = Cells(i, 1)
   Next i

   Cells(23, 2) = n
   amin = a(1)
   amax = a(1)

   For i = 2 To n
         If amin > a(i) Then
               amin = a(i)
         End If
         If amax < a(i) Then
               amax = a(i)
         End If
```

```
    Next i

    k = 0
    l = 0

    For i = 1 To n
        If a(i) = amin Then
            k = k + 1
        End If
        If a(i) = amax Then
            l = l + 1
        End If
    Next i

    Cells(23, 3) = k
    Cells(23, 4) = l
    ReDim a1(2, k), a2(2, l)
    k1 = 0
    l2 = 0
    For i = 1 To n
        If a(i) = amin Then
            k1 = k1 + 1
            a1(1, k1) = amin
            a1(2, k1) = i
        End If
        If a(i) = amax Then
            l2 = l2 + 1
            a2(1, l2) = amax
            a2(2, l2) = i
        End If
    Next i

    For k1 = 1 To k
        Cells(a1(2, k1), 3) = a1(1, k1)
    Next k1

    For k1 = 1 To l
        Cells(a2(2, k1), 4) = a2(1, k1)
    Next k1
End Sub
```

На рис. 53 представлен экран выполнения рассматриваемой программы.

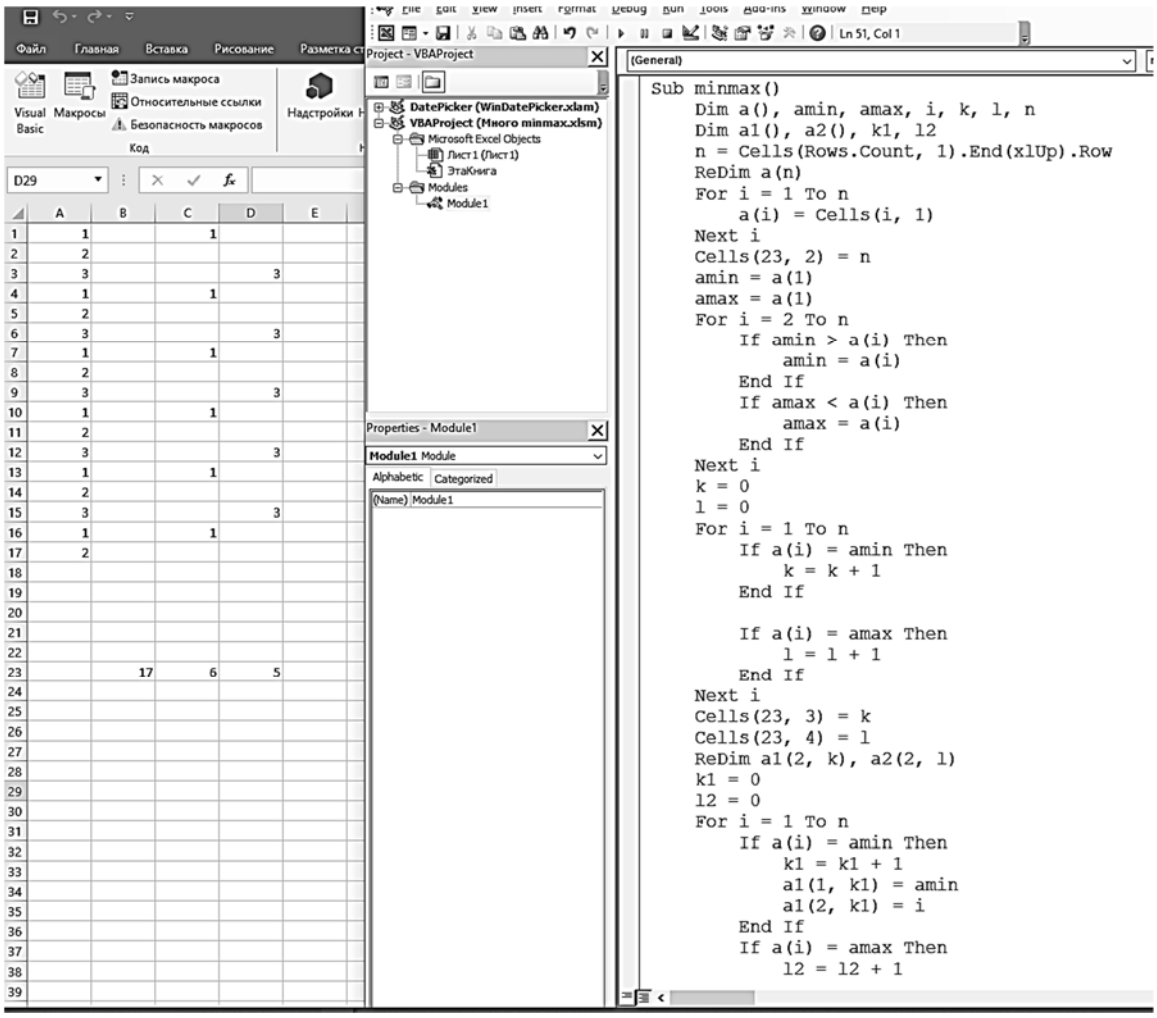


Рис. 53. Определение минимальных и максимальных значений в массиве

5.8.7.2. Краткое руководство по массивам (VBA)

В табл. 22 представлена сводка сведений для корректного использования различных массивов в программах.

Таблица 22

Справка по работе с массивами в ***VBA***

Задача	Статический массив	Динамический массив
1	2	3
Объявление	Dim arr(0 To 5) As Long	Dim arr() As Long Dim arr As Variant
Установить размер	Dim arr(0 To 5) As Long	ReDim arr(0 To 5)As Variant

Продолжение табл. 22

1	2	3
Увеличить размер (сохранить существующие данные)	Только динамический	ReDim Preserve arr(0 To 6)
Установить значения	arr(1) = 22	arr(1) = 22
Получить значения	total = arr(1)	total = arr(1)
Первая позиция	LBound(arr)	LBound(arr)
Последняя позиция	Ubound(arr)	Ubound(arr)
Читать все записи (1D)	For i = LBound(arr) To UBound(arr) Next i Or For i = LBound(arr,1) To UBound(arr,1) Next i	For i = LBound(arr) To UBound(arr) Next i Or For i = LBound(arr,1) To UBound(arr,1) Next i
Читать все записи (2D)	For i = LBound(arr,1) To UBound(arr,1) For j = LBound(arr,2) To UBound(arr,2) Next j Next i	For i = LBound(arr,1) To UBound(arr,1) For j = LBound(arr,2) To UBound(arr,2) Next j Next i
Читать все записи	Dim item As Variant For Each item In arr Next item	Dim item As Variant For Each item In arr Next item
Перейти на Sub	Sub MySub(ByRef arr() As String)	Sub MySub(ByRef arr() As String)
Возврат из функции	Function GetArray() As Long() Dim arr(0 To 5) As Long GetArray = arr End Function	Function GetArray() As Long() Dim arr() As Long GetArray = arr End Function
Получить от функции	Только динамический	Dim arr() As Long Arr = GetArray()
Стереть массив	Erase arr *Сбрасывает все значения по умолчанию	Erase arr *Удаляет массив
Строка в массив	Только динамический	Dim arr As Variant arr = Split(«James:Earl:Jones»,»:»)
Массив в строку	Dim sName As String sName = Join(arr, «:»)	Dim sName As String sName = Join(arr, «:»)
Заполните значениями	Только динамический	Dim arr As Variant arr = Array(«John», «Hazel»)
Диапазон в массиве	Только динамический	Dim arr As Variant arr = Range(«A1:D2»)
Массив в диапазоне	Так же, как в динамическом	Dim arr As Variant Range(«A5:D6») = arr

Контрольные вопросы к главе 5

1. Что такое массив?
2. Что такое регулярные (анонимные) типы?
3. Как используются собственные имена при объявлении массивов?
4. Как осуществляется последовательный доступ к элементу массива?
5. Чем отличаются различные способы сортировки массивов?
6. В чем суть алгоритма удаления элемента из массива?
7. Что такое квадратная матрица?
8. Какими соотношениями связаны расположения элементов в матрице с диагоналями?
9. Какие существуют операции над массивами?
10. Какие существуют операции над элементами массива?
11. Какие типы массивов применяются в программировании?
12. Как можно изменить размер массива?
13. Каков механизм сохранения начальных значений массива при изменении размеров?
14. В чём разница между удалением массива и очищением массива.
15. Каким способом можно определить размер массива?
16. Какая разница между верхней и нижней границами разноразмерных массивов?
17. Сколько типов циклов применяется в программировании?
18. В чём особенности параметрического цикла?
19. В чем принципиальная разница в условных циклах?
20. Какие есть способы досрочного прерывания выполнения цикла?

Глава 6. ПОЛЬЗОВАТЕЛЬСКИЕ ПРОЦЕДУРЫ И ФУНКЦИИ

В языке *PascalABC.NET*, как и в большинстве языков программирования, в том числе и в *VBA*, предусмотрены средства, позволяющие оформлять вспомогательный алгоритм как подпрограмму. Это бывает необходимо когда какой-либо алгоритм неоднократно повторяется в программе или имеется возможность использовать некоторые фрагменты уже разработанных ранее алгоритмов. Кроме того, подпрограммы применяются для разбиения крупных программ на отдельные смысловые части в соответствии с модульным принципом в программировании.

Для использования алгоритма в качестве подпрограммы ему необходимо присвоить имя и описать алгоритм по правилам языка Паскаль. В дальнейшем при необходимости вызвать его в программе делают вызов подпрограммы упоминанием в нужном месте имени соответствующего алгоритма со списком входных и выходных данных. Такое упоминание приводит к выполнению входящих в подпрограмму операторов, работающих с указанными данными. После выполнения подпрограммы работа продолжается с той команды, которая непосредственно следует за вызовом подпрограммы.

В языке Паскаль имеется два вида подпрограмм – процедуры и функции.

Процедуры и функции помещаются в раздел описаний программы. Для обмена информацией между процедурами и функциями и другими блоками программы существует механизм входных и выходных параметров. Входными параметрами называют величины, передающиеся из вызывающего блока в подпрограмму (исходные данные для подпрограммы), а выходными – передающиеся из подпрограммы в вызывающий блок (результаты работы подпрограммы).

Одна и та же подпрограмма может вызываться неоднократно, выполняя одни и те же действия с разными наборами входных данных. Параметры, использующиеся при записи текста подпрограммы в разделе описаний, называют формальными, а те, что используются при ее вызове – фактическими [5].

6.1. Описание и вызов процедур и функций (*PascalABC.NET*)

Структура описания процедур и функций до некоторой степени похожа на структуру Паскаль-программы: у них также имеются заголовок, раздел описаний и исполняемая часть. Раздел описаний содержит те же подразделы, что и раздел описаний программы: описания констант, типов, меток, процедур, функций, переменных. Исполняемая часть содержит собственно операторы процедур.

Формат описания процедуры имеет вид:

```
procedure имя процедуры (формальные параметры);
  раздел описаний процедуры
begin
  исполняемая часть процедуры
end;
```

Формат описания функции:

```
function имя функции (формальные параметры):тип результата;
  раздел описаний функции
begin

  исполняемая часть функции

end;
```

Формальные параметры в заголовке процедур и функций записываются в виде:

```
var имя праметра: имя типа
```

и отделяются друг от друга точкой с запятой. Ключевое слово *var* может отсутствовать (об этом далее). Если параметры однотипны, то их имена можно перечислять через запятую, указывая общее для них имя типа. При описании параметров можно использовать только стандартные имена типов либо имена типов, определенные с помощью команды *type*. Список формальных параметров может отсутствовать.

Вызов процедуры производится оператором, имеющим следующий формат:

```
имя процедуры(список фактических параметров);
```

Список фактических параметров – это их перечисление через запятую. При вызове фактические параметры как бы подставляются вместо формальных, стоящих на тех же местах в заголовке. Таким образом происходит передача входных параметров, затем выполняются операторы исполняемой части процедуры, после чего происходит возврат в вызывающий блок. Передача выходных параметров происходит непосредственно во время работы исполняемой части.

Вызов функции в *PascalABC.NET* может производиться аналогичным способом. Кроме того, имеется возможность осуществить вызов внутри какого-либо выражения. В частности имя функции может стоять в правой части оператора присваивания, в разделе условий оператора *if* и т.д.

Для передачи в вызывающий блок выходного значения функции в исполняемой части функции перед возвратом в вызывающий блок необходимо поместить следующую команду:

```
имя функции := результат;
```

При вызове процедур и функций необходимо соблюдать следующие правила:

- количество фактических параметров должно совпадать с количеством формальных;
- соответствующие фактические и формальные параметры должны совпадать по порядку следования и по типу.

Заметим, что имена формальных и фактических параметров могут совпадать. Это не приводит к проблемам, так как соответствующие им переменные все равно будут различны из-за того, что хранятся в разных областях памяти. Кроме того, все формальные параметры являются временными переменными – они создаются в момент вызова подпрограммы и уничтожаются в момент выхода из нее.

Пример 6.1.

Рассмотрим использование процедуры на примере программы поиска максимума из двух целых чисел (рис. 54).

```
var
  x, y, m, n: integer;
```

```
procedure MaxNumber(a, b: integer; var max: integer);
begin
   if a > b then max := a else max := b;
end;

begin
   write('Введите x,y ');
   readln(x, y);
   MaxNumber(x, y, m);
   writeln('x = ', x, ';  y = ', y, '; m = ', m);
   MaxNumber(2, x + y, n);
   writeln('x + y = ', x + y, ';  n = ', n);
end.
```

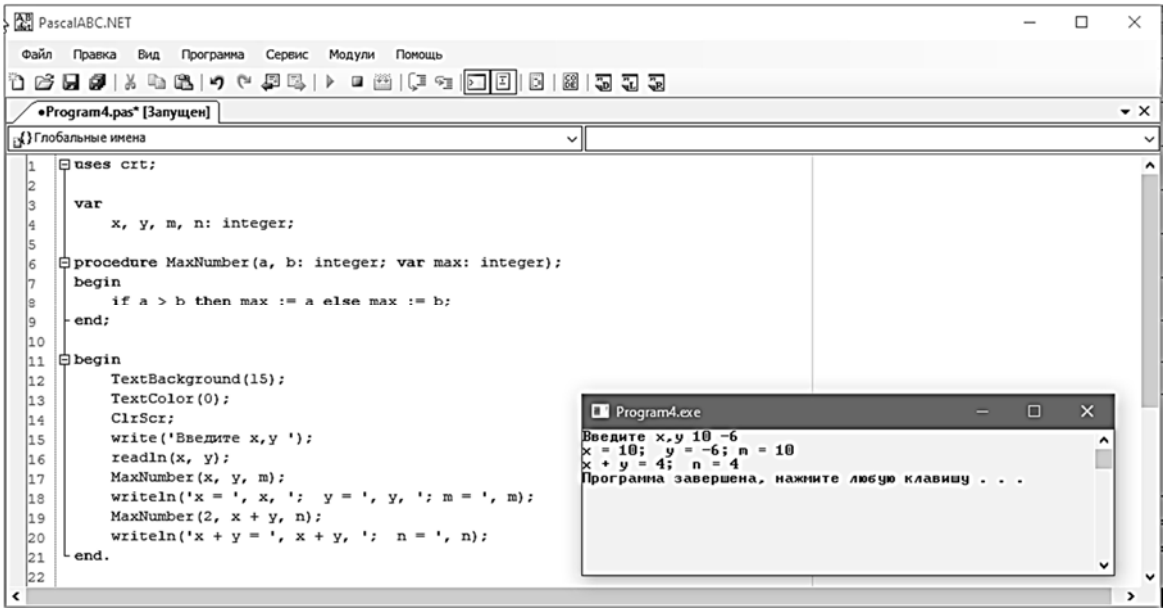


Рис. 54. Пример использования Procedure

Аналогичную задачу, но уже с использованием функций, можно решить так (рис. 55):

```
var
   x, y, m, n: integer;

function MaxNumber(a, b: integer): integer;
var
   max: integer;
begin
   if a > b then max := a else max := b;
   MaxNumber := max;
end;

begin
```

```
    write('Введите x,y ');
    readln(x, y);
    m := MaxNumber(x, y);
    writeln('x = ', x, '; y = ', y, '; m = ', m);
    n := MaxNumber(2, x + y);
    writeln('x+y = ', x+y, '; n = ', n);
end.
```

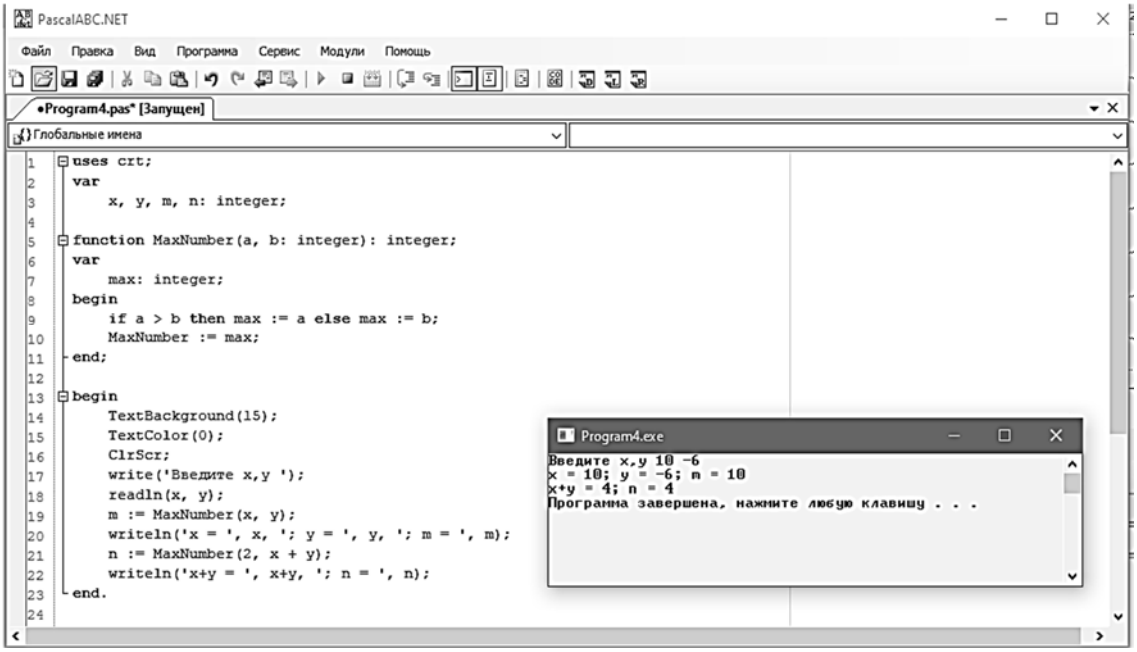


Рис. 55. Пример использования Function

В *PascalABC.NET* допускаются короткие определения для функций, задаваемых одним выражением. Тип возвращаемого значения можно не указывать – он выводится автоматически. Допускаются короткие определения процедур, задаваемых одним оператором:

```
function Куб(x: integer) := x*x*x;
function ПлощадьКруга(r: real) := Pi*r*r;
function Hypot(a,b: real) := Sqrt(a*a+b*b);
procedure DrawP(x,y: real) := Circle(x,y,3);
```

Переменной можно присвоить действие. Такая переменная называется ***процедурной***. Она хранит отложенное действие. Это действие можно вызвать, указав процедурную переменную вместо имени процедуры или функции. Действие можно передать в подпрограмму как параметр. Вызов этого действия в этой подпрограмме называется обратным вызовом (*callback*):

```
procedure Корова := Println('My-y');
```

```
procedure Собака := Println('Гав!');
procedure Кошка := Println('Мяу!');
begin
  var Звук: procedure := Корова;
  Звук;
  Звук := Собака;
  Звук;
  Звук := Кошка;
  Звук;
end.
```

Процедурной переменной можно присвоить функцию. Как и процедуру, функцию можно вызвать через процедурную переменную и передать её как параметр в подпрограмму.

Пример 6.2.

Вызов функции через процедурную переменную.

```
begin
  var a : real -> real := sin;
  Print(a(0));
  a := cos;
  Print(a(0));
end.
```

Пример 6.3.

Передача функции как параметра (рис. 56).

```
uses GraphABC;

begin
  Draw(cos);
end.
```

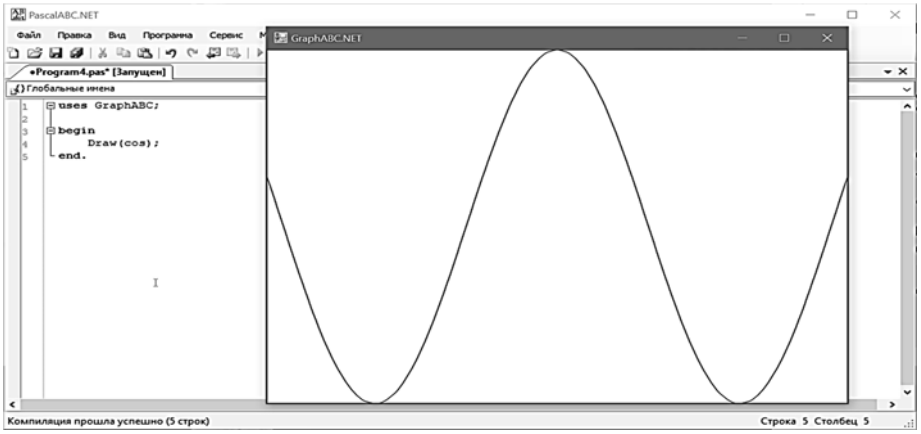


Рис. 56. Передача функции как параметра

6.2. Процедуры и функции. Виды процедур (*VBA*)

Пользовательская функция – это процедура *VBA*, которая производит заданные вычисления и возвращает полученный результат. Используется для вставки в ячейки рабочего листа *Excel* или для вызова из других процедур.

6.2.1. Синтаксис функции

```
[Static] Function Имя ([СписокАргументов])[As ТипДанных]
   [Операторы]
   [Имя = выражение]
   [Exit Function]
   [Операторы]
   [Имя = выражение]
End Function
```

6.2.2. Компоненты функции

- ***Static*** – необязательное ключевое слово, указывающее на то, что значения переменных, объявленных в функции, сохраняются между ее вызовами.
- ***Имя*** – обязательный компонент, имя пользовательской функции.
- ***СписокАргументов*** – необязательный компонент, одна или более переменных, представляющих аргументы, которые передаются в функцию. Аргументы заключаются в скобки и разделяются между собой запятыми.
- ***Операторы*** – необязательный компонент, блок операторов (инструкций).
- ***Имя = выражение*** – необязательный[11] компонент, присвоение имени функции значения выражения или переменной. Обычно значение присваивается функции непосредственно перед выходом из нее.
- ***Exit Function*** – необязательный компонент, принудительный выход из функции, если ей уже присвоено окончательное значение.

[11] Один из компонентов Имя = выражение следует считать обязательным, так как если не присвоить функции значение, то смысл ее использования теряется.

6.2.3. Видимость функции

Видимость пользовательской функции определяется необязательными ключевыми словами *Public* и *Private*, которые могут быть указаны перед оператором *Function* (или *Static*, в случае его использования).

Ключевое слово *Public* указывает на то, что функция будет доступна для вызова из других процедур во всех модулях открытых книг *Excel*. Функция, объявленная как *Public*, отображается в диалоговом окне *Мастера функций*.

Ключевое слово *Private* указывает на то, что функция будет доступна для вызова из других процедур только в пределах программного модуля, в котором она находится. Функция, объявленная как *Private*, не отображается в диалоговом окне Мастера функций, но ее можно ввести в ячейку вручную.

Если ключевое слово *Public* или *Private* не указано, то функция считается по умолчанию объявленной, как *Public*.

Чтобы пользовательская функция всегда была доступна во всех открытых книгах *Excel*, сохраните ее в Личной книге макросов без объявления видимости или как *Public*. Но если вы планируете передать рабочую книгу с пользовательской функцией на другой компьютер, то код функции должен быть в программном модуле передаваемой книги.

Пример 6.4.

Для примера мы рассмотрим простейшую пользовательскую функцию, которой в следующем параграфе добавим описание. Называется функция «Деление», объявлена с типом данных *Variant*, так как ее возвращаемое значение может быть и числом, и текстом. Аргументы функции – Делимое и Делитель – тоже объявлены как *Variant*, так как в ячейках *Excel* могут быть числовые значения разных типов, и функция *IsNumeric* тоже проверяет разные типы данных и требует, чтобы ее аргументы были объявлены как *Variant*:

```
Function Деление(Делимое As Variant, Делитель As Variant) As Variant
  If IsNumeric(Делимое) = False Or IsNumeric(Делитель) = False Then
        Деление = "Ошибка: Делимое и Делитель должны быть числами!"
```

```
        Exit Function
    ElseIf Делитель = 0 Then
        Деление = "Ошибка: деление на ноль!"
        Exit Function
    Else
        Деление = Делимое / Делитель
    End If
End Function
```

Эта функция выполняет деление значений двух ячеек рабочего листа *Excel*. Перед делением проверяются два блока условий:

- Если делимое или делитель не являются числом, то функция возвращает значение: «Ошибка: Делимое и Делитель должны быть числами!» и производится принудительный выход из функции оператором *Exit Function*.
- Если делитель равен нулю, то функция возвращает значение: «Ошибка: деление на ноль!» и производится принудительный выход из функции оператором *Exit Function*.
- Если проверяемые условия не выполняются (возвращают значение *False*), то производится деление чисел и функция возвращает частное (результат деления).

Вы можете скопировать к себе в стандартный модуль эту функцию и она станет доступна в разделе «Определенные пользователем» в диалоговом окне *Мастера функций*. Попробуйте вставить функцию «Деление» в ячейку рабочего листа с помощью *Мастера функций* и поэкспериментируйте с ней.

Практического смысла функция «Деление» не имеет, но она хорошо демонстрирует как объявляются, создаются и работают пользовательские функции в *VBA Excel*. А еще она поможет продемонстрировать, как добавлять к функциям и аргументам описания. С полноценной пользовательской функцией вы можете ознакомиться здесь.

6.2.4. Добавление описания функции

В списке функций, выводимом Мастером, невозможно добавить или отредактировать их описание. Список макросов позволяет добав-

лять процедурам описание, но в нем нет функций. Проблема решается следующим образом:

- Запустите *Мастер функций*, посмотрите, как отображается имя нужной функции и закройте его.
- Откройте список макросов и в поле «*Имя макроса*» впишите имя пользовательской функции.
- Нажмите кнопку «*Параметры*» и в открывшемся окне добавьте или отредактируйте описание.
- Нажмите кнопку «*ОК*», затем в окне списка макросов – «*Отмена*». Описание готово!

Добавление описания на примере функции «Деление» (рис. 57):

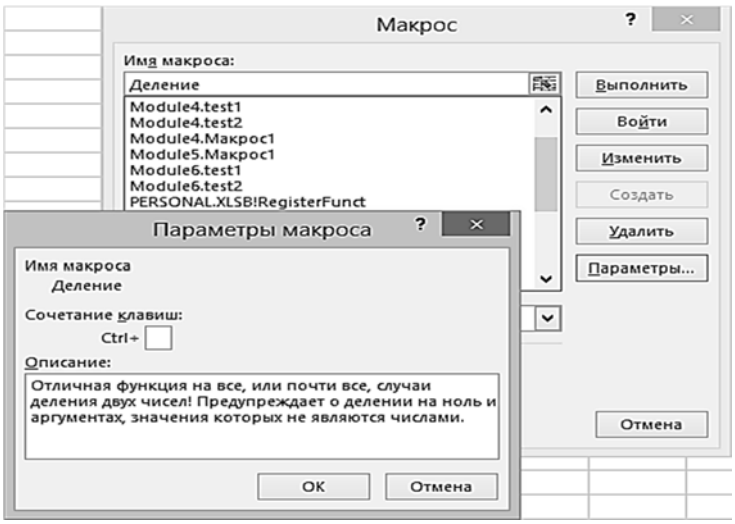


Рис. 57. Добавление описания пользовательской функции

Описание функции «Деление» в диалоговом окне *Мастера функций* «Аргументы функции» (рис. 58):

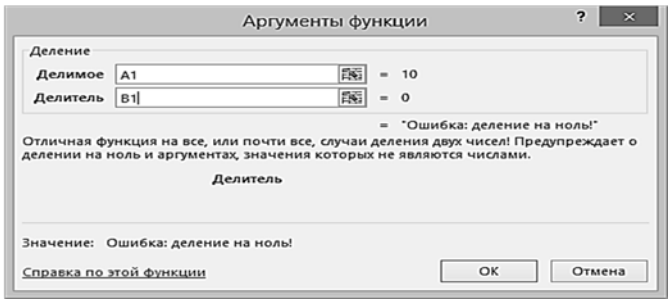


Рис. 58. Описание пользовательской функции в окне «Аргументы функции»

С помощью окна «Список макросов» можно добавить описание самой функции, а ее аргументам нельзя. Но это можно сделать, используя метод *Application.MacroOptions*.

6.2.5. Метод Application.MacroOptions

Метод *Application.MacroOptions* позволяет добавить пользовательской функции описание, назначить сочетание клавиш, указать категорию, добавить описания аргументов и добавить или изменить другие параметры. Давайте рассмотрим возможности этого метода, используемые чаще всего.

Пример 6.5

Пример кода с методом Application.MacroOptions:

```
Sub ИмяПодпрограммы()
   Application.MacroOptions _
         Macro:="ИмяФункции", _
         Description:="Описание функции", _
         Category:="Название категории", _
         ArgumentDescriptions:=Array("Описание 1", "Описание 2",
               "Описание   3", ...)
End Sub
```

- ***ИмяПодпрограммы*** – любое уникальное имя, подходящее для наименования процедур.
- ***ИмяФункции*** – имя функции, параметры которой добавляются или изменяются.
- ***Описание функции*** – описание функции, которое добавляется или изменяется.
- ***Название категории*** – название категории, в которую будет помещена функция. Если параметр *Category* отсутствует, то пользовательская функция будет записана в раздел по умолчанию – «Определенные пользователем». Если указанное *Название категории* соответствует одному из названий стандартного списка, то функция будет записана в него. Если такого *Названия категории* нет в списке, то будет создан новый раздел с этим названием и функция будет помещена в него.

- ***“Описание 1”, “Описание 2”, “Описание 3”,*** … – описания аргументов в том порядке, как они расположены в объявлении пользовательской функции.

Эта подпрограмма запускается один раз, после чего ее можно удалить или использовать как шаблон для корректировки параметров других пользовательских функций.

Сейчас с помощью метода *Application.MacroOptions* попробуем изменить описание пользовательской функции «Деление» и добавить описания аргументов.

```
Sub ИзменениеОписания()
  Application.MacroOptions _
        Macro:="Деление", _
        Description:="Описание функции Деление изменено методом
              Application.MacroOptions", _
        ArgumentDescriptions:=Array("- любое числовое значение", "- число
              вое значение, кроме нуля")
End Sub
```

После однократного запуска этой подпрограммы получаем следующий результат (рис. 59):

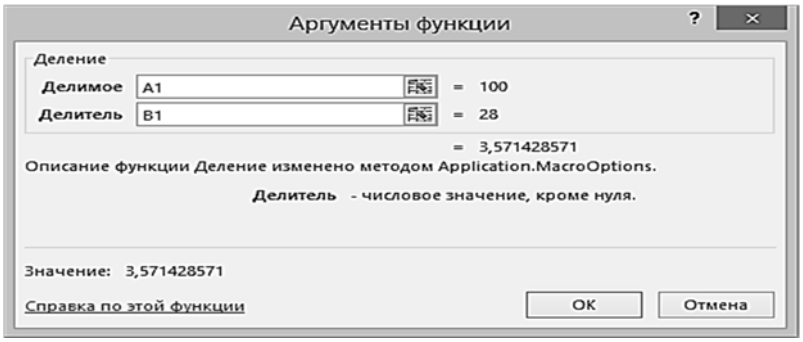


Рис. 59. Новое описание пользовательской функции и её второго аргумента

Метод *Application.MacroOptions* не работает в *Личной книге макросов*, но и здесь можно найти решение. Добавьте описания к пользовательским функциям и их аргументам в обычной книге *Excel*, затем экспортируйте модуль с функциями в любой каталог на жестком диске и оттуда импортируйте в *Личную книгу макросов*. Все описания сохранятся.

Контрольные вопросы к главе 6

1. Где пишутся листинги (коды) процедур?
2. Какова структура листинга процедуры?
3. Что записывается в заголовок процедуры после служебного слова *Sub*?
4. Каким оператором осуществляется присвоение значения переменной в *Visual Basic*?
5. Каким оператором в *Visual Basic* организуется ввод данных? Вывод данных?
6. Какова структура оператора ветвления если / то / иначе в *Visual Basic*?
7. Как организуются циклы с параметром?
8. Как осуществляется обращение к процедуре из другой процедуры?
9. Как описываются переменные в *Visual Basic*? Для чего это необходимо?
10. Где пишутся листинги функций, определяемых пользователем?
11. Какова структура листинга (кода) функции, определяемой пользователем?
12. Что вписывается в заглавную строку листинга пользовательской функции после служебного слова *Function*?
13. Как осуществляется присвоение значения функции в *VBA*?
14. Каковы значения, вычисляемые встроенными функциями *VBA*? Каков смысл аргументов этих функций?
15. В чем различие функций *Excel*, функций, определенных пользователем, и встроенных функций *VBA*? Где могут применяться функции перечисленных видов (рабочий лист, программный код)?

Глава 7. СРАВНИТЕЛЬНЫЙ СИНТАКСИС ЯЗЫКОВ ПРОГРАММИРОВАНИЯ (СООТВЕТСТВИЕ)

Сокращения в табл.23:

вещ – вещественное,

гран – граница,

знач – значение,

кол_элем – количество элементов,

конст – константа,

масс – массив,

нов_имя – новое имя,

объяв – объявление,

операт – оператор,

перем – переменная,

подкл_библиот – подключаемые библиотеки,

прогр – программа,

сущ_тип – существующий тип,

функц – функция.

Таблица 23

Сравнительный синтаксис языков программирования

Операция	Паскаль	Алгоритмический	Бейсик	Си
Комментарий				
	{текст} (*текст*)	|строка текста	'строка текста REM строка текста	//строка текста /*текст*/
Структура программы				

	заголовок
	объяв конст
	объяв типов
	объяв перем
	описание функц
	begin
	тело прогр
	end.
		алг
		нач
		объяв перем
		тело прогр
		кон
		описание функц
			объяв функ
			тело прогр
			END
			описание функц
				подкл_библиот
				объяв конст
				объяв типов
				объяв функц
				void main(){
				тело прогр
				}
				описание функц

Объявление переменных различных типов данных				
Целое (2 байта)	перем: integer;	цел перем	DIM перем AS INTEGER	int перем;
Целое (4 байта)	перем:longint;	цел перем	DIM перем AS LONG	long перем;
вещ (4 байта)	перем:single;	вещ перем	DIM перем AS SINGLE	float перем;
вещ (6 байт)	перем:real;			
вещ (8 байт)	перем: double;	//-//	DIM перем AS DOUBLE	double перем;
символ	перем: char;	сим перем	DIM перем AS STRING	char перем;
строка	перем: string;	лит перем	//-//	-
логическое	перем: boolean;	лог перем	-	-

Продолжение табл. 23

массив целых	масс: array[гран..гран] of integer ;	целтаб масс[гран:гран]	DIM масс() AS INTEGER	int масс[кол_элем];
массив символов	масс: array[гран..гран] of char;	симтаб масс[гран:гран]	DIM масс() AS STRING	char масс[кол_элем+1];
массив строк	масс: array[гран..гран] of string;	литтаб масс[гран:гран]	//-//	-
объявление константы	const имя_конст=знач;	-	CONST имя_конст=знач	#define имя_конст знач
интервал	перем:гран..гран	-	-	-
создание нового типа данных	type имя_типа=опис_типа;			typedef сущ_тип нов_имя
Арифметические операции				
сложение	+	+	+	+
вычитание	-	-	-	-
умножение	*	*	*	*
возведение в степень	exp(степень*ln(основание))	**	^	exp(степень*ln(основание)) pow(основание,степень)
деление	/	/	/	/
целочисленное деление	div	div(делимое,делитель)	\	целое/целое
определение остатка от деления	mod	mod(делимое,делитель)	MOD	%
инкрементация (увеличение на 1)	-	-	-	++

Продолжение табл. 23

декрементация (уменьшение на 1)	-	-	-	--
Операция присваивания				
	:=	:=	=	=
Операция взятия адреса				
	@перем			&перем
Операции сравнения				
равно	=	=	=	==
не равно	<>	<>	<>	!=
больше	>	>	>	>
меньше	<	<	<	<
больше или равно	>=	>=	>=	>=
меньше или равно	<=	<=	<=	<=
Логические операции				
не	not	не	NOT	!
и	and	и	AND	&&
или	or	или	OR	||
Преобразование типов данных				
целое в символ (символ по коду)	chr (целое)	символ(целое)	CHR$(целое)	(char)целое
целое (от 0 до 9) в символ десятичной цифры	if(целое>=0)and (целое<=9) символ:= ord(символ)+ord('0') ;	if(целое>=0)и (целое<=9) символ:= код(символ)+код('0')	if(целое>=0)AND (целое<=9) символ= ASC(символ)+ASC("0")	if((целое>=0)&& (целое<=9)) символ=целое+'0';
код символа	ord(символ)	код(символ)	ASC(символ)	(int)символ

Продолжение табл. 23

символ в десятичную цифру	if(символ>='0'and символ<='9') целое:= ord(символ)-ord('0');	if(символ>='0' и символ<='9') целое:= код(символ)-код('0')	if(символ>='0'AND символ<='9') целое= ASC(символ)-ASC("0")	if((символ>='0')&&(символ<='9')) целое=символ-'0';
целое в строку	str(целое, строка);	цел_в_лит(целое)	STR$(целое)	//подключаем stdlib.h itoa(целое,строка);
вещ в строку	str(вещ: кол_знак: кол_знак_после_зап, строка);	вещ_в_лит(вещ)	STR$(вещ)	//подключаем stdlib.h ftoa(вещ,строка);
строка в целое	val(строка,целое, код_ошиб);	лит_в_цел(строка,успех)	VAL(строка)	//подключаем stdlib.h atoi (строка)
строка в вещ	//-//	лит_в_вещ(строка,успех)	//-//	//подключаем stdlib.h atof (строка);
Функция проверки, является ли символ десятичной цифрой				
	isdigit(символ)	-	-	//подключаем ctype.h isdigit(символ)

Продолжение табл. 23

Ввод и вывод				

ввод перем (через пробел)	read(перем,перем);	ввод перем,перем	-	//подключаем stdio.h
				//ввод целого
				scanf("%d",&перем);
				//ввод вещ
				scanf("%f",&перем);
				//ввод символа
				scanf("%c",&перем);
				//ввод строки //(без пробелов)
				scanf("%s",&строка);
				//ввод строки //(с пробелами)
				gets(строка);

ввод перем (через Enter)	readln(перем,перем);	ввод перем
		ввод перем
			INPUT перем
			INPUT перем
				//подключаем stdio.h
				scanf("%тип", &перем);
				scanf("%тип", &перем);

ожидание программой нажатия любой клавиши	readln			//подключаем conio.h
				getch();

вывод с переводом строки	writeln(перем,перем);	вывод нс, перем,перем	PRINT перем	//подключаем stdio.h
				printf("%тип\n",перем);

вывод без перевода строки	write(перем,перем);	вывод перем,перем	PRINT перем;	//подключаем stdio.h
				printf("%тип%тип",перем, перем);

вывод пустой строки	write(char(13)+char(10));	вывод нс	PRINT	//подключаем stdio.h
				printf("\n");

Продолжение табл. 23

Работа со строками				
сравнение строк	=,<>,>,<	=,<>	=,<>	//подключаем string.h strcmp(строка ,строка)
копирование одной строки на место другой	строка1:=строка2;	строка1:=строка2	строка1=строка2	//подключаем string.h strcpy(строка 1,строка2)
копирование части строки на место другой	-	-	-	//подключаем string.h strncpy(строка1, строка2, позиция)
взятие символа из строки	строка[поз_сим]	строка[поз_сим]	MID$(строка, нач, длина)	строка[поз_сим]
выбор подстроки	copy(строка, позиция, кол_сим)	строка[поз_сим:поз_сим]	//-//	-
слияние строк	строка1+строка2 concat(строка,строка)	строка1+строка2	строка1+строка2	//подключаем string.h strcat(строка, строка)
удаление из строки подстроки	delete(строка, позиция, кол_симв)	-	-	-
добавление подстроки в строку	insert(подстрока, строка, позиция)	-	-	//подключаем string.h strncpy(строка1, строка2, позиция)
длина строки	length(строка)	длин(строка)	LEN(строка)	//подключаем string.h strlen(строка)

Продолжение табл. 23

Операторы ветвления				

если	if условие
	then
	begin
	операторы;
	end
	else
	begin
	операторы;
	end
		если условие
		то
		операторы
		иначе
		операторы
		все
			IF условие
			THEN
			операторы
			ELSE
			операторы
			END IF
				if(условие){
				операторы;
				}
				else{
				операторы;
				}

Выбор	case перем
	of
	конст :
	оператор;
	else оператор;
	end;
		выбор
		при условие:
		оператор
		при условие:
		оператор
		иначе оператор
		все
			SELECT CASE
			перем
			CASE
			пер_выб:операт
			CASE
			пер_выб:операт
			...
			CASE ELSE
			операт
			END SELECT
				switch(перем){
				case
				конст:опер;
				break;
				case
				конст:опер;
				break;
				default : оператор;
				}

Операторы цикла				

цикл "для"	for to
	(downto) do
	begin
	тело цикла
	end;
		нц для от до
		шаг
		тело цикла
		кц
			FOR
			тело цикла
			NEXT
				for(){
				тело цикла
				}

цикл "пока"	while условие do
	begin
	тело цикла
	end;
		нц пока условие
		тело цикла
		кц
			WHILE условие
			тело цикла
			WEND
				while(условие){
				тело цикла
				}

цикл "до тех пор"	repeat тело цикла until(условие)	нц тело цикла кц при условие	DO WHILE условие тело цикла LOOP	do{ тело цикла }while(условие);

Окончание табл. 23

Описание и объявление функции				

определение	Function имя_функц (имена_аргумен-тов:тип): тип;
	begin
	  тело функц
	end;
		алг тип имя_функц (тип имена_аргу-ментов)
		нач
		  тело функц
		кон
			FUNCTION имя_функц (имена_аргу-ментов)
			  тело функц
			END FUNCTION
				тип имя_функц (тип имена_аргу-ментов)
				{
				  тело функц
				}

объявление	-	-	DECLARE FUNCTION имя_функ(аргу-мент)	тип имя_функц (тип)

СПИСОК ЛИТЕРАТУРЫ

1. Саакян, И.Э. Программирование в Turbo Pascal 7.0: учеб. пособие/ И.Э.Саакян, Л.Ф. Макаренко. – М.: МАДИ, 2009. – 504 с.
2. Саакян, И.Э. Информатика. Часть 1. Введение в информатику: учеб. пособие/ И.Э. Саакян, А.Б. Николаев. – М.: МАДИ, 2013. – 185 с.
3. Саакян, И.Э. Информатика. Часть 2. Алгоритмизация: учеб. пособие/ И.Э. Саакян, А.Б. Николаев. – М.: МАДИ, 2015. – 138 с.
4. Словари и энциклопедии на Академике: [сайт]. [© Академик, 2000-2014]. URL: http://dic.academic.ru/dic.nsf/ruwiki/4896 (дата обращения: 21.август.2015).
5. Процедуры и функции [Электронный ресурс] // Волгоградский государственный педагогический университет. Кафедра алгебры, геометрии и информатики: [сайт]. URL: http://mif.vspu.ru/books/pascal/ procedure.html (дата обращения: 31.05.2020).
6. Бесплатные уроки по MS Excel и MS Word от Антона Андронова [Электронный ресурс] // Microsoft Excel для начинающих: [сайт]. [2020]. URL: https://office-guru.ru/excel/peremennye-i-konstanty-v-vba-459.html (дата обращения: 31.03.2020).
7. Все про массивы в VBA [Электронный ресурс] // EcxelPedia.ru Отборные статьи по MS Excel 2010/13/16: [сайт]. [2020]. URL: https:// excelpedia.ru/makrosi-v-excel/massivi-v-vba (дата обращения: 06.05.2020).

Учебное издание

СААКЯН Игорь Эдуардович

ЕВСТРАТОВА Ирина Александровна

ИНФОРМАТИКА

Часть 3. Основы программирования

Редактор И.А. Короткова

Редакционно-издательский отдел МАДИ. E-mail: rio@madi.ru

Подписано в печать 16.03.2022 г. Формат 60×84/16.
Усл. печ. л. 13,25. Тираж 500 экз. Заказ . Цена 1070 руб.
МАДИ, 125319, Москва, Ленинградский пр-т, 64.