Н. Н. Дмитриев, В. Ю. Сахаров

ВВЕДЕНИЕ В MICROSOFT ACCESS. ПРИМЕНЕНИЕ VBA ДЛЯ РЕШЕНИЯ ПРОСТЕЙШИХ ЗАДАЧ

С а н к т - П е т е р б у р г 2006

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Н. Н. Дмитриев, В. Ю. Сахаров

ВВЕДЕНИЕ В MICROSOFT ACCESS. ПРИМЕНЕНИЕ VBA ДЛЯ РЕШЕНИЯ ПРОСТЕЙШИХ ЗАДАЧ

Учебное пособие

САНКТ-ПЕТЕРБУРГ
2006

ББК 32.81
Д53

Р е ц е н з е н т ы :
проф. О. Н. Граничин (С.-Петерб. гос. ун-т),
доц. М. Н. Охочинский (Балт. гос. техн. ун-т)

Печатается по постановлению
Редакционно-издательского совета
С.-Петербургского государственного университета

Дмитриев Н.Н., Сахаров В.Ю.

Д53 Введение в Microsoft Access. Применение VBA для решения простейших задач: Учеб. пособие. – СПб., 2006. – 72 с.

Учебное пособие посвящено изучению основ Microsoft Access и VBA.

Предназначено для студентов гуманитарных вузов.

ББК 32.81

Введение в MS Access

Вводные замечания

В состав Microsoft Office входит программа MS Access, позволяющая создавать базы данных и управлять ими. При этом большинству пользователей не нужно прибегать к программированию. Средства MS Access достаточно наглядные и понятные при наличии некоторого опыта работы с данной программой.

В MS Access под **базой данных (БД)** понимается файл, в котором хранятся данные и настройки системы управления базами данных.

Хранилищем информации БД выступает таблица, состоящая из **полей** (столбцов) и **записей** (строк). Запись (строка в таблице) – стандартный блок для хранения данных в таблице и выборке данных при запросе.

Еще одним важным элементом, которым обладает таблица, является **ключевое поле**, служащее для однозначного определения записи в таблице.

Одно или несколько ключевых полей, позволяющих идентифицировать запись таблицы и организовать связь между таблицами, называется **ключом**.

Во многих приложениях встречается аббревиатура **SQL** (structured query language). Этот язык встроен во многие программные продукты различных фирм. Поэтому с базой данных созданной в MS Access можно работать, применяя другие специализированные приложения.

Корпорация Microsoft, создав пакет Microsoft Office, позаботилась еще и о том, чтобы составные части этого программного продукта могли взаимодействовать друг с другом. Таким образом, появляется возможность, например, средствами Excel отобразить данные из базы данных, которая создана с помощью MS Access, или таблицу из Excel импортировать в БД MS Access и т.д.

База данных MS Access

Информация в базе данных, созданной в программе MS Access, хранится в таблицах, которые связаны между собой. Поэтому первый шаг в создании БД – это формирование таблиц.

Создание таблиц

Таблицу можно создать в режиме конструктора, с помощью мастера или путем ввода данных (режим таблицы). Для этого следует выполнить процедуру: **Файл → Создать → Новая база данных →** в открывшемся окне **Файл новой базы данных** указать, в какой папке будет располагаться таблица **→** нажать кнопку **Создать.** В результате откроется окно, в котором выбирается способ создания таблицы (рис.1).

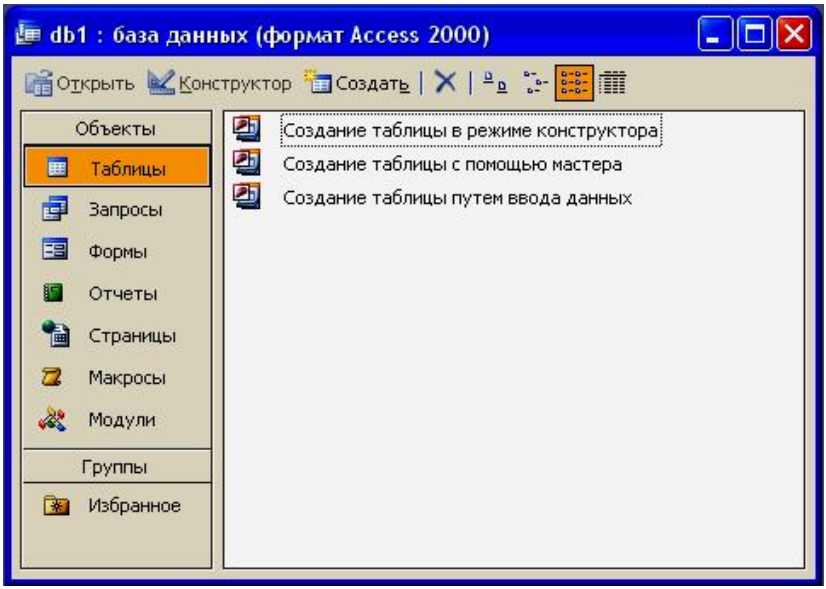


Рис. 1

Режим таблицы путем ввода данных

Выбрав пункт **Создание таблицы путем ввода данных**, пользователь получает доступ к заготовке, представленной на рис. 2.

В этой абстрактной таблице поля следует переименовать, чтобы они несли смысловую нагрузку. Тип полей программа определит автоматически, в зависимости от внесенной в них информации (чем больше данных в таблицу будет введено, тем точнее программа определит тип данных и размеры полей). Сохранение таблицы производится обычным способом: **Файл → Сохранить →** в окне **Сохранение** ввести имя таб-

лицы → нажать кнопку **OK** (при сохранении таблицы все пустые столбцы с неизмененными именами будут удалены).

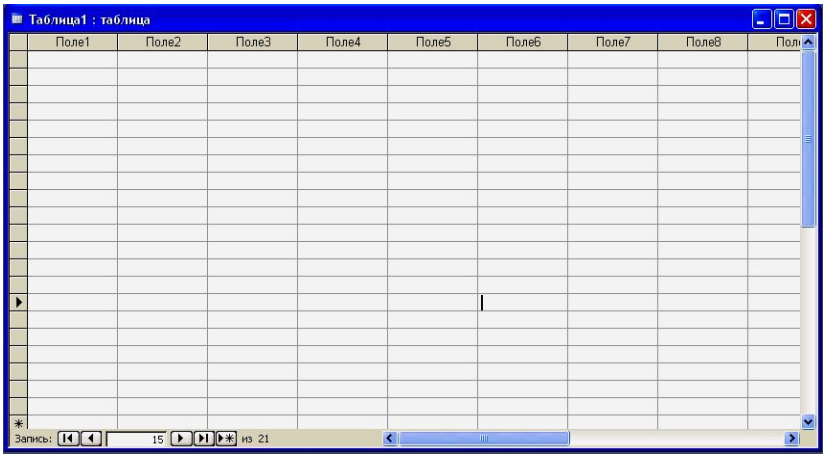


Рис. 2

Замечание. Если MS Access неверно определит какие-либо свойства полей, то исправления можно внести в режиме конструктора.

При работе с программами Microsoft постоянно приходится выделять тот или иной объект. Для выделения ячейки следует передвинуть указатель мыши в ее левую часть так, чтобы он превратился в жирный белый крестик, и нажать левую кнопку мыши. При двойном щелчке в ячейке выделяются только данные. Столбцы выделяются щелчком левой кнопкой мыши на его заголовке. Для выделения смежных столбцов используется клавиша <Shift> и мышь с нажатой левой кнопкой, указатель которой перемещается по нужным заголовкам.

Нажатие правой кнопки мыши вызывает контекстное меню, соответствующее ячейке в которой находится ее указатель. Назначение команд контекстного меню следует из их названия.

Если в таблице не заданы ключевые поля, то эта таблица не готовая база данных. Поэтому при закрытии таблицы MS Access предложит автоматически создать ключевое поле. Имя этого поля по умолчанию будет **Код**, и хранить оно будет номер записи в таблице. Поле-счетчик заполняется автоматически при появлении в таблице новой записи.

Режим конструктора

Более гибкое создание БД возможно в **режиме конструктора**, который, кроме того, позволяет полнее понять, что такое таблица MS Access (рис. 3). В этом случае структура таблицы создается пользователем «с нуля».

В верхней части окна конструктора расположен бланк с перечнем всех полей, заголовков, их типов и описаний. Графа **Описание** не является обязательной и предназначена для текста подсказки, которая появляется во время работы с таблицей.

Тип поля выбирается в ячейке столбца **Тип данных** из списка, который появляется при щелчке в соответствующей ячейке.

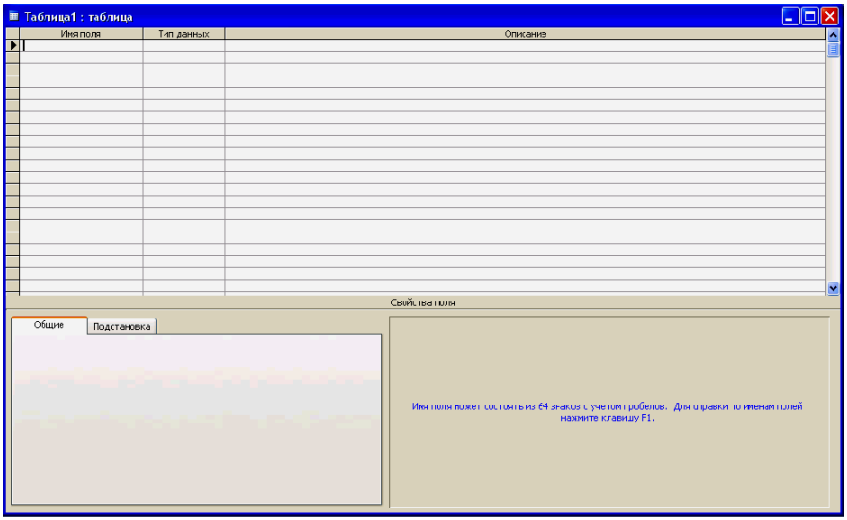


Рис. 3

Одно из полей таблицы должно быть определено как ключевое. Для этого следует установить курсор на соответствующую строку бланка и нажать кнопку **Ключевое поле** (или выбрать эту команду из контекстного меню). Если идентифицировать запись в таблице можно только по нескольким полям, то следует выделить несколько строк в бланке и выбрать команду **Ключевое поле**. Ключевые поля выделяются изображением ключа в области выделения строки. Не может быть более одного ключа. Поэтому изображение символа ключа в нескольких строчках бланка свидетельствует о составном ключе.

Типы (форматы) полей

При щелчке мышью в колонке **Тип данных**, соответствующей некоторому полю, появляется возможность выбрать один из следующих форматов: **текстовый**, **числовой**, **дата/время**, **денежный**, **счетчик**, **логический**, **поле объект OLE**. Кроме того, в этом списке находится **мастер подстановок**.

Текстовый формат

На рис. 4 представлена вкладка *Общие* со свойствами текстовых полей.

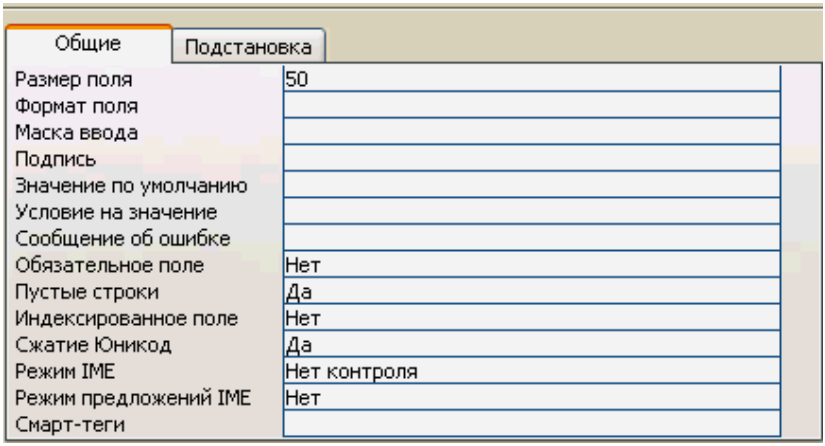


Рис. 4

Замечание. При конструировании базы данных нет необходимости явно указывать все свойства, но следует знать эту возможность и при необходимости ею пользоваться.

В первой строке свойств задается размер текстового поля, который может быть от 1 до 255 символов.

Формат поля позволяет жестко задать вид и размер вводимых строк. Кодовые символы формата имеют вид:

@ – должен быть текстовый символ или пробел;

& – текстовый символ;

> – преобразование символа в верхний регистр;

< – преобразование символа в нижний регистр.

Формат поля может состоять из двух частей, разделенных знаком «**;**». Правая часть – формат ввода, а вторая – определяет значение поля, если данные в него не были введены.

Маска ввода – последовательность кодовых символов:

0 – должна быть цифра от 0 до 9;

9 – цифра или пробел;

– цифра, пробел, плюс или минус;

L – должна быть буква (A,…,Z, A,...,Я);

? – буква;

A – должна быть буква или цифра;

a – может быть буква или цифра;

& – должен быть любой символ или пробел;

C – произвольный символ;

«.», «,», «;», «:»,«-», «/» – разделители, которые сохраняют свой вид в строке данных;

> – преобразование символа в верхний регистр;

< – преобразование символа в нижний регистр;

! – заполнение маски справа налево;

\ – ввод следующего за обратной косой чертой символа как символьной константы.

Подпись – это второй (первый имя) идентификатор поля, который используется программой вместо имени поля при работе с данными в табличной форме для создания заголовка столбца.

Свойство **Значение по умолчанию** позволяет автоматически подставлять заданное значение во все вновь создаваемые поля.

Условие на значение – это фильтр, который разрешает вводить в поле только то, что удовлетворяет заданному условию.

Обязательное поле – логическое значение «Да» означает, что в поле обязательно должны быть введены данные.

Свойство **Пустые строки** – определяет разрешены или нет в данном поле пустые строки.

Свойство **Индексированное поле** может принимать значения *поле неиндексированно*, *индексировано*, *но допускает повторяющиеся значения, индексировано с запретом повторяющихся значений*.

Свойства **Сжатие Юникод**, **Режим IME**, **Режим предложений IME**, **Смарт-теги** обсуждать здесь не будем ввиду их специфичности и ограниченности объема пособия. Сведения о них можно найти в справочной системе программы MS Access (вызывается нажатием <F1>).

Поле **MEMO** – это текстовый формат очень большой длины (сохраняет 65536 знаков), служащий для ввода комментариев, примечаний и т. п.

Числовой формат

Вкладка для определения свойств числового поля представлена на рис. 5.

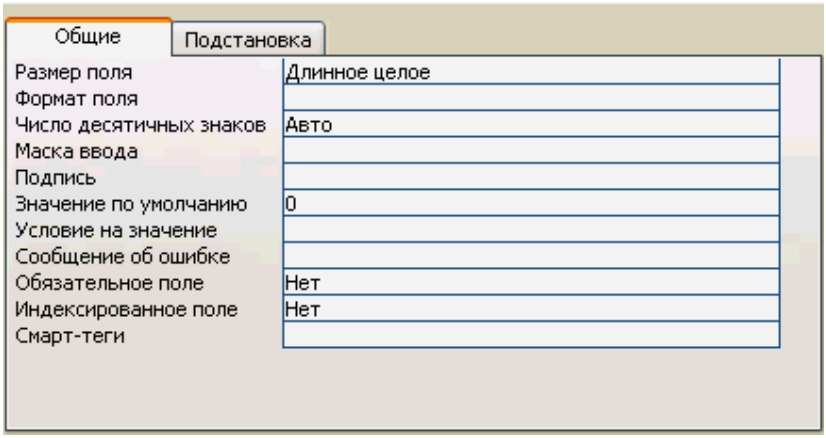


Рис. 5

Свойство **Размер поля** определяет представление числа и может принимать значения:

Байт – целые числа от 0 до 255;

Целое – целые числа от –32768 до 32767;

Длинное целое – целые числа от –2147483648 до 2147483647;

Одинарное с плавающей точкой – числа с семью знаками после запятой от –3.402823 E38 до 3.402823E38;

Двойное с плавающей точкой – числа с 15 знаками после запятой от –1,79769313486231E308 до 1,79769313486231E308;

Код репликации – при конструировании таблицы не используется;

Действительное – числа, содержащие 28 десятичных знаков и располагающиеся в диапазоне от –1E–28 до 1E28–1.

Формат поля определяет вид отражаемых в таблице чисел. Для этого следует выбрать какой-либо формат из списка или использовать следующие кодовые символы для создания нового формата:

. – точка используется в качестве десятичного разделителя;

, – запятая применяется как разделитель групп разрядов;

0 – вывод цифры или нуля, если разряд незначащий;

– вывод цифры;

$ – знак доллара;

% – вывод числа в процентном формате;

E или e – вывод числа в экспоненциальной форме.

При этом могут быть заданы четыре группы кодов: первая для ввода положительных чисел, вторая – для отрицательных, третья – для отражения нулевого значения и четвертая – для пустых полей. Группы разделяются знаком точка с запятой.

Пример. Число 123456,789 при задании формата поля конструкцией # ###,# будет выглядеть следующим образом: 123 456,78900 (при указании числа десятичных знаков, равном 5), Кроме того, в формате можно задать цвет выводимых символов:

#[Красный];-[Синий];0[Зеленый];”Нет данных”

При этом обязательно следует создать группу для нулевого значения, иначе ноль не будет выводиться на экран.

Формат дата/время

Вкладка со свойствами полей **дата/время** представлена на рис. 6.

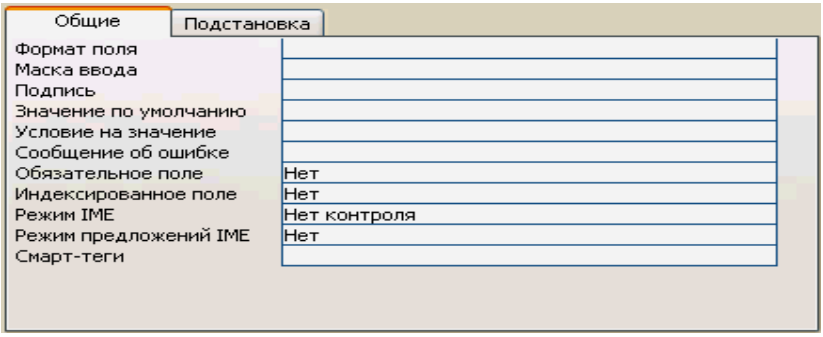


Рис. 6

Следует подчеркнуть, что в случае смены формата поля дата/время часть данных может измениться. Это произойдет, если записи в таблице были сделаны в разных форматах (например, время 15:45 и дата 15/09/05). Поэтому поле данного типа должно быть определено в

самом начале работы. В свойствах целесообразно указать *Условие на значение* и *Сообщение об ошибке*.

Денежный формат

Свойства вкладки *денежного* формата (рис. 7) аналогичны описанным выше.

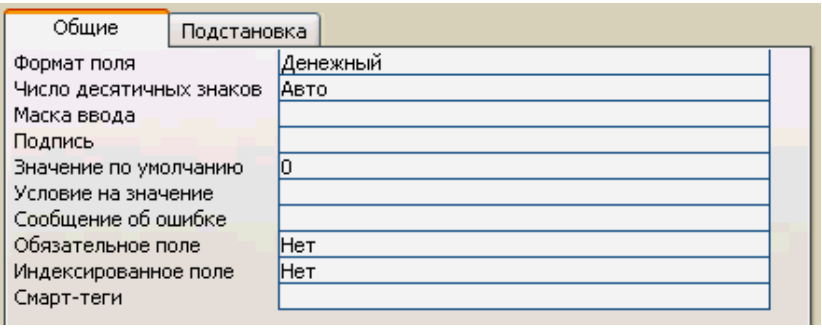


Рис. 7

Денежный тип поля используют для предотвращения округления во время вычислений. В денежных полях обеспечивается 15 знаков слева от десятичной запятой и 4 знака справа. Денежное поле занимает 8 байт на диске.

Формат Счетчик

Свойства поля типа **Счетчик** отображены на рис. 8.

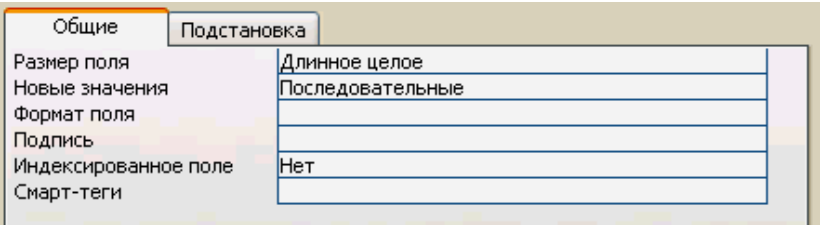


Рис. 8

Следует отметить, что **Счетчик** – это всегда число, предназначенное для идентификации записей в таблице. Изменение этого числа

происходит автоматически последовательно или случайным образом. Первый вариант позволяет нумеровать записи, второй, когда числа выбираются случайным образом, уменьшает вероятность замены одного кода другим.

Логический формат

Ячейка в поле логического типа может содержать только одно из двух значений «Да» или «Нет». Свойства полей логического типа отображены на рис. 9.

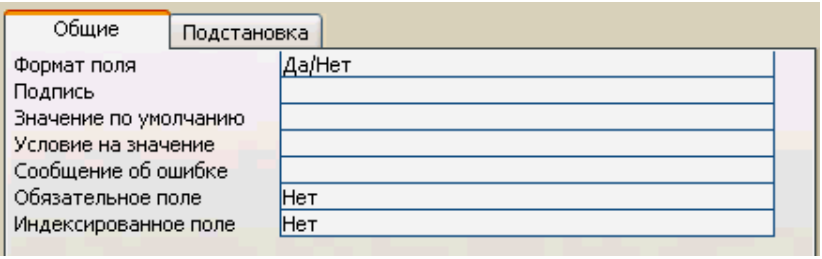


Рис. 9

Два других названия поля логического типа – **Истина/Ложь** и **Включено/Выключено**. Логическое поле представляется в таблице в виде набора флажков. Изображение галочки в квадрате соответствует логическому «Да», отсутствие галочки присваивает ячейке поля логическое значение «Нет» (рис. 10).

Рис. 10

Поле объекта OLE

Поле объекта OLE (Object Linking and Embedding) имеет два свойства (рис. 11).

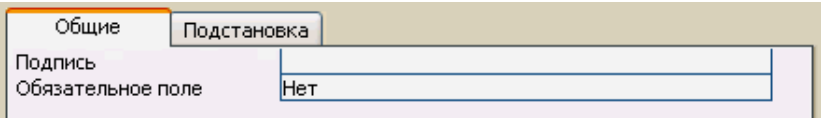


Рис. 11

Это поле не хранит информацию, а содержит ссылку на объект, который включается в базу данных с помощью *OLE-протокола* обмена данными.

Для внедрения объекта в ячейку следует установить в ней курсор → нажать правую кнопку мыши → в появившемся контекстном меню выбрать команду **Вставить объект** → в открывшемся диалоговом окне **Вставка объекта** (рис. 12) при установке переключателя в положение **Создать новый** встраиваемое приложение создается «с нуля»; если встраиваемый файл уже существует, то переключатель надо установить в положение **Создать из файла** → MS Access откроет **окно открытия документа** (рис. 12)→ выбирается документ → **OK** → MS Access автоматически открывает приложение, соответствующее этому файлу → можно произвести редактирование встраиваемого файла → при закрытии программы объект оказывается внедренным.

Замечание. Если нужно внедрить существующий файл без редактирования, то можно воспользоваться процедурой: в проводнике Windows найти и выделить файл → <Ctrl + C> (копирование файла в буфер обмена) → в MS Access выбрать ячейку в поле объекта OLE → активизировать контекстное меню (щелчок правой кнопкой мыши) → выполнить команду **Вставить**.

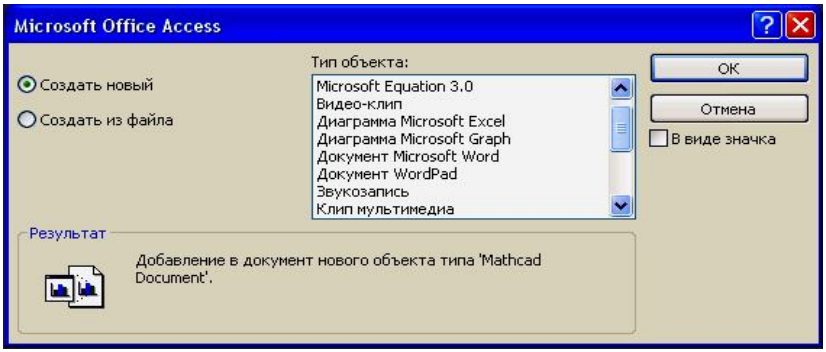


Рис. 12

Активизация работы со встроенными объектами осуществляется двойным щелчком мыши.

Мастер подстановок

В списке типов полей имеется элемент являющийся не названием типа, а командой. Эта команда называется **Мастер подстановок**, создающая связь между таблицами. Действие этой команды рассмотрим на примере. Пусть даны две таблицы (рис.13 и 14), в которых присутствуют поля типа «**Счетчик», числовой и текстовый**.

Таблица12 : таблица

	Код1	Функция	Значение в точке x=2
▶	1	x+1	3
	2	x^2+2	6
	3	x^3+3	11
	4	x^4+4	20
	5	x^5+5	37
*	(Счетчик)		0

Рис. 13

Таблица22 : таблица

	Код2	Вариант	Производные
	1	15	1
	2	24	2x
	3	33	3x^2
	4	42	4x^3
▶	5	51	5x^4
*	(Счетчик)	0	

Рис. 14

Подставим поле **Вариант** из *Таблицы22* в *Таблицу12*, используя следующую процедуру: перейдем в режим конструктора для *Таблицы12* → введем имя нового поля, например, *Номер задания* →в графе **Тип данных** выбираем команду **Мастер подстановок** → на первом

шаге этого мастера выбираем пункт соответствующий использованию данных из другой таблицы → нажмем кнопку **Далее** → на втором шаге выбираем таблицу, из которой осуществляется подстановка (*Таблица22*) → **Далее** → третий шаг предназначен ддля выбора подставляемого поля или нескольких полей (порядок следования полей важен, так как первое указанное поле является источником данных в операции подстановки) → **Далее** → выбираем порядок сортировки списка (по возрастанию или убыванию) → регулируем ширину, подставляемой колонки → **Далее** → задаем подпись столбца → очередное нажатие кнопки **Готово** завершает работу **Мастера подстановки**.

Мастер подстановки произвел следующие действия (рис.15 и рис.16):

Таблица12 : таблица

	Имя поля	Тип данных
🔑	Код1	Счетчик
	Функция	Текстовый
	Значение в точке x=2	Числовой
▶	Номер задания	Числовой

Рис. 15

Общие | Подстановка

Тип элемента управления	Поле со списком
Тип источника строк	Таблица или запрос
Источник строк	SELECT Таблица22.Код2, Таблица22.Вариант FR
Присоединенный столбец	1
Число столбцов	2
Заглавия столбцов	Нет
Ширина столбцов	0см;2,54см
Число строк списка	8
Ширина списка	2,54см
Ограничиться списком	Да

Рис. 16

1. Новому полю присвоен числовой тип (числовой тип относится к ключевому полю **Таблица22.Код2**, а не данным подстановки);
2. В *Таблице12* появился новый столбец, в котором каждая ячейка может быть заполнена данными из списка, содержащие данные вы-

бранного поля *Таблицы22* (заполнение происходит при выделении значения в списке);

3. Если на третьем шаге **Мастера постановки** выбрать два поля (*Вариант* и *Производная*), то список подстановки формируется на основе выбранного первого поля, но при открытии списка будут отражены значения обоих полей, разделенных вертикальной линией. При этом вторая колонка выступает в качестве некоторого комментария и не подставляется в таблицу (рис.17 и рис. 18).

Рис. 17

Рис. 18

При выполнении команды **Сервис → Схема данных** раскрывается схема базы данных (рис. 19), в которой отражается связь между *Таблицей12* и *Таблицей22*. Подстановка на этой схеме обозначена линией без всяких обозначений.

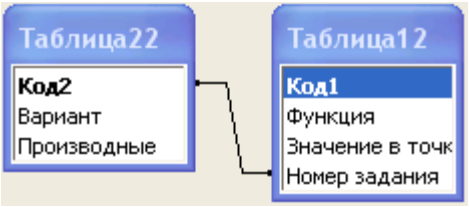


Рис. 19

Замечание 1. Для удаления поля подстановки следует вначале удалить связь между таблицами. Для этого в **Схеме данных** нужно выделить линию, соединяющую таблицы и в контекстном меню выбрать команду **Удалить связь**, и в режиме конструктора убрать поле подстановки.

Замечание 2. Для сохранения структуры таблиц перед выходом из режима конструктора необходимо нажать на панели инструментов кнопку **Сохранить** или выполнить команду **Файл → Сохранить** и нажать кнопку **Закрыть** в верхнем правом углу окна конструктора.

Замечание 3. Отметим, что в MS Access имеются заготовки таблиц**,** доступные через **Мастер таблиц**. Если эти шаблоны не вполне устраивают требованию пользователя, то можно их корректировать в режиме конструктора.

Межтабличные связи

База данных обычно состоит из нескольких таблиц, описания связей между ними, форм, отчетов и других объектов, помогающих пользователю работать с ней. При эксплуатации БД часто возникает необходимость редактирования данных**.** Всякое изменение любого значения удобно делать только в одном месте БД. С другой стороны, в таблицах следует группировать данные по определенной тематике. И, наконец, следует выделять блоки, которые можно совершенствовать по отдельности, а таблицы создавать таким образом, чтобы они могли быть использованы в другой БД. Некоторые из перечисленных замечаний удается удовлетворить с помощью **Мастера по анализу таблиц**.

Мастер по анализу таблиц

Процедура исключения дублирования данных в таблице называется *нормализацией*. Эту процедуру помогает осуществить **Мастер по анализу таблиц**, который запускается из меню таким образом: **Сервис**

→ **Анализ** → **Таблица**. Рассмотрим работу этого мастера на примере преобразований следующей таблицы (рис. 20).

Таблица3 : таблица

	Код	Вариант	Функция	Число	Буква
▶	1	Первый	x^2+1	2	А
	2	Второй	x+11	12	Б
	3	Третий	5x	5	В
	4	Первый	5x^2+2	7	А
	5	Второй	15x	15	Б
	6	Третий	x^3	1	В
	7	Третий	x^2+5	6	В
	8	Второй	2x+2	4	Б
	9	Первый	5x+5	10	А
	10	Первый	x^2+9x+1	11	А
*	(Счетчик)				

Рис. 20

Первые два шага **Мастера** содержат информацию о преобразовании таблицы, которое возможно будет произведено.

Замечание. Исходная таблица при работе **Мастера по анализу таблиц** не меняется, а создаются новые таблицы, содержащие те же данные. В дальнейшем пользователь должен решить какие таблицы оставить в базе данных.

На третьем шаге мастера выбирается таблица, над которой будет произведена работа. Кроме того, третий шаг позволяет отключить показ вводных страниц (первых двух шагов мастера).

На четвертом шаге запрашивается подтверждение распределения данных по двум или более таблицам в случае обнаружения необходимости разбиения таблицы.

Пятый шаг мастера демонстрирует первый результат своей работы. Исходная таблица разбита на две и дублирование данных исключено. Эти таблицы связаны с помощью операции подстановки, при этом связь называется «многие-к-одному». Графически это отображается на рис. 21.

Рис. 21

Здесь видно, что в *Таблице4* поле **Буква** сделано ключевым автоматически. Этот вид связи нас может не устраивать. Поэтому существуют другие виды связи между таблицами, которые будут рассмотрены позже.

На шестом шаге мастера происходит проверка уникальности поля подстановки, как ключевого поля. В случае повторяющихся записей программа предлагает создать еще одно поле, которое станет ключевым, и будет содержать уникальные числовые коды. Для этого следует нажать кнопку **Добавить ключ**  .

Замечание. Если ключевое поле не создавать, хотя программа этого требует, то откроется окно, в котором можно исправить обнаруженный недочет.

Седьмой шаг мастера является завершающим. Остается ответить на вопрос о необходимости создания запроса. При выполнении запроса таблица-запрос будет доступна на вкладке **Запросы**.

Итак, мастер по анализу таблиц создал связанные таблицы (рис. 22 и рис. 23), эквивалентные исходной, и таблицу-запрос (рис. 24).

Таблица2 : таблица

Код	Функция	Число	Подстановка Таблица4
1	x^2+1	2	А, Первый
4	5x^2+2	7	А, Первый
9	5x+5	10	А, Первый
10	x^2+9x+1	11	А, Первый
2	x+11	12	Б, Второй
5	15x	15	Б, Второй
8	2x+2	4	Б, Второй
3	5x	5	В, Третий
6	x^3	1	В, Третий
7	x^2+5	6	В, Третий
(Счетчик)			

Запись: 11 из 11

Рис. 22

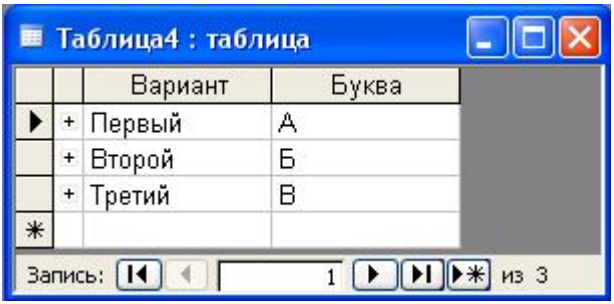


Рис. 23

Рис. 24

Замечание. В *Таблице2* имеется поле *Подстановка Таблица4* – это вычисляемое поле и дублирование в нем не противоречит принципам нормализации данных.

Связь типа «один-ко-многим»

Выше уже было отмечено, что связь таблиц с помощью подстановки не всегда является хорошим решением при создании БД. Поэто-

му опишем как поэтапно создается связь других типов на примере уже созданных таблиц.

Замечание. Связываемые поля должны быть одинакового типа. Числовые поля при этом должны иметь одинаковые значения свойства **Размер поля**. Поле счетчика можно связать с числовым полем типа **Длинное целое**.

Произведем подготовительные действия.

1. В *Таблице2* вставим столбец (**Вставка → Столбец**) с названием *Код Варианта* текстового типа.
2. В режиме конструктора для *Таблицы2* для поля с подписью *Подстановка Таблица4* **установить тип элемента управления** как **Поле** и сохранить изменения.
3. В *Таблице2* выделить поле *Подстановка Таблица4* → **Правка → Копировать** → выделить поле *Код Варианта* → **Правка → Вставить.**
4. Удалить связь между таблицами: **Сервис → Схема данных** → в окне **Схема данных** выделить существующую связь → нажать клавишу <Delete> → закрыть окно **Схема данных**.
5. В *Таблице2* выделить поле *Подстановка Таблица4* → **Правка → Удалить столбец**.

В результате пяти указанных шагов получаем результат, изображенный на рис. 25.

Таблица2 : таблица

	Код	Код_Варианта	Функция	Число
▶	1	А	x^2+1	2
	2	Б	x+11	12
	3	В	5x	5
	4	А	5x^2+2	7
	5	Б	15x	15
	6	В	x^3	1
	7	В	x^2+5	6
	8	Б	2x+2	4
	9	А	5x+5	10
	10	А	x^2+9x+1	11
*	(Счетчик)			

Запись: 1 из 10

Рис. 25

Установим теперь связь *Таблицы4* и *Таблицы2*:

6. **Сервис → Схема данных →** выделить мышью в *Таблице4* поле *Буква* и не отпуская левую кнопку мыши перетащить указатель на поле *Код Варианта Таблицы2* → откроется окно **Связи →** в окне **Связи** установить флажок **Обеспечение целостности данных →** нажать кнопку **Объединение →** в раскрывшемся окне **Параметры объединения** установить переключатель в положение 2 → нажать кнопку **Создать**.

Замечание. Польза от связывания таблиц проявляется при создании форм или отчетов. В зависимости от выбора принципа объединения в окне **Параметры объединения** меняется объем выводимых данных при совместном использовании таблиц.

В рассмотренном примере выбран второй тип объединения. Это означает, что *Таблица4* признана главной и все данные из нее доступны при использовании в форме или отчете. *Таблица2* – это связанная таблица и из нее могут выбираться только данные, для которых установлено соответствие.

В *Таблице4* коды, соответствующие записям, не повторяются дважды. В связанной таблице один код может встречаться несколько раз. Поэтому созданная связь называется «один-ко-многим». Иными словами, одному коду в главной таблице соответствует много записей подчиненной таблицы. На **Схеме данных** эта связь выглядит, как показано на рис. 26, пример формы представлен на рис. 27.

Рис. 26

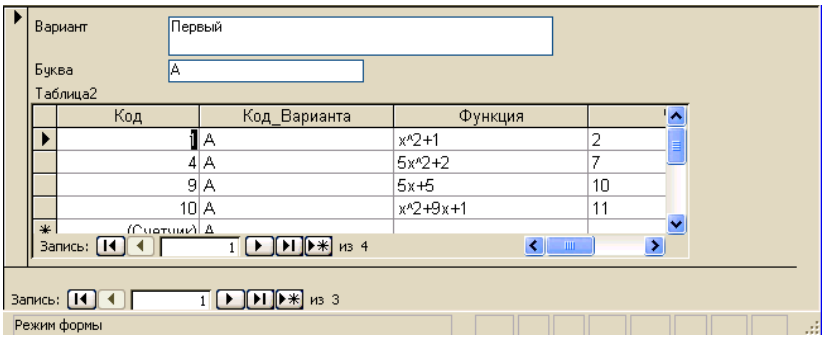


Рис. 27

Связь типа «один-к-одному»

Связь типа «один-к-одному» ставит в соответствие каждой строке одной таблицы единственную строку другой таблицы. При таком сопоставлении данные можно объединить в одну таблицу без ущерба для БД. Но с другой стороны, некоторые данные могут быть взяты из другой БД или модифицированы, а для этого их удобнее разместить в отдельной таблице. Такое размещение данных соответствует принципу модульности СУБД.

Для демонстрации организации связи «один-к-одному» создадим таблицу *Дополнительные задания* (рис. 28):

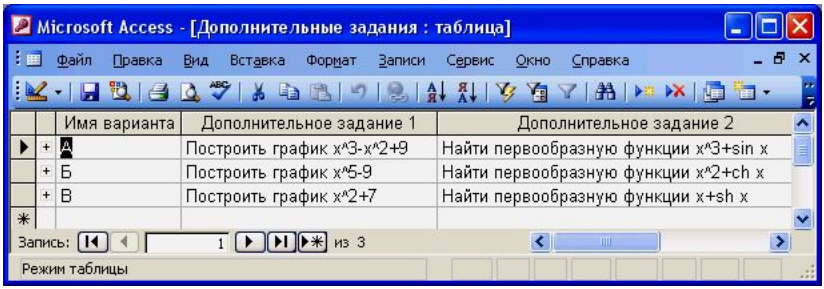


Рис. 28

Связь типа «один-к-одному» между *Таблицей4* и *Дополнительные задания* строится по алгоритму: открыть окно **Схема данных** → нажать кнопку **Отобразить таблицу** (или **Связи** → **Добавить таблицу**)→ выбрать таблицу *Дополнительные задания* → выделить

поле **Код** в *Таблице4* и, не отпуская левую кнопку мыши, переместить указатель на поле **Имя варианта** в таблице *Дополнительные задания* → в открывшемся окне **Связи** установить флажок **Обеспечение целостности данных** (программа определит связь типа «один-к-одному»), нажать кнопку **Объединение** → в раскрывшемся окне **Параметры объединения** установить переключатель в положение 2 → нажать кнопку **Создать**. Схема данных примет вид показанный на рис. 29.

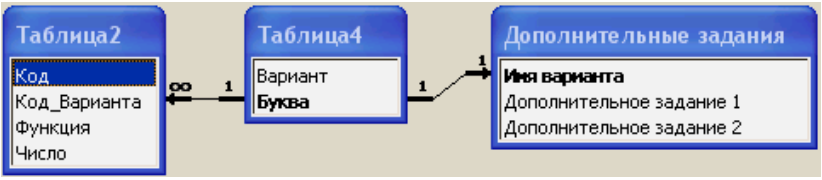


Рис. 29

Связь типа «многие-ко-многим»

При увеличении объема информации в БД количество таблиц возрастает, некоторые поля их могут содержать повторяющиеся значения. Ситуация часто складывается так, что именно по этим полям нужно связывать таблицы. Но попытка осуществит связь на прямую не увенчается успехом, так как тип связи «многие-ко-многим» программно не определен. К счастью, эта связь может быть представлена в виде двух связей типа «один-ко-многим» и некоторой промежуточной таблицей.

В качестве примера рассмотрим таблицу (рис. 30):

	Код1	Код варианта	Теоретический вопрос	Номер теста
▶	1	А	Теорема синусов	101
	2	А	Теорема косинусов	102
	3	Б	Теорема Пифагора	201
	4	Б	Бином Ньютона	202
	5	В	Теорема Фалеса	301
	6	В	Объем конуса	302
*	(Счетчик)			

Рис. 30

Промежуточная таблица должна содержать ключевое поле, которое используется для определения связи. Поэтому в качестве промежу-

точной составим следующую таблицу, в которой поле *Код варианта* является ключевым (рис. 31).

Рис. 31

Схема данных (рис. 32) показывает, как связь «многие-ко-многим» заменена двумя связями «один-ко-многим».

Замечание. Вспомогательная таблица может содержать два поля, через которые осуществляется связь.

Использование мастера форм позволяет быстро составить сводную таблицу, содержащую всевозможные комбинации записей (рис.33).

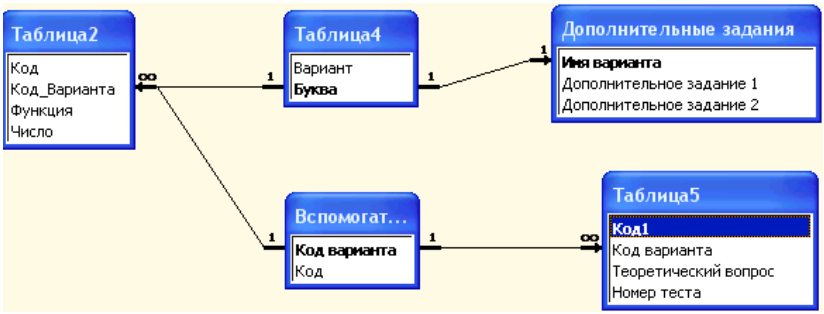


Рис. 32

Рис. 33

Построение форм

Часто представление данных в виде таблиц может не устраивать пользователя. Кроме того, многие документы имеют определенный оригинальный вид, отличный от табличного. Поэтому придуман специальный механизм, названный ***формой***, позволяющий удовлетворить изысканным запросам пользователя.

На рис. 34 представлено окно, которое используется при построении формы.

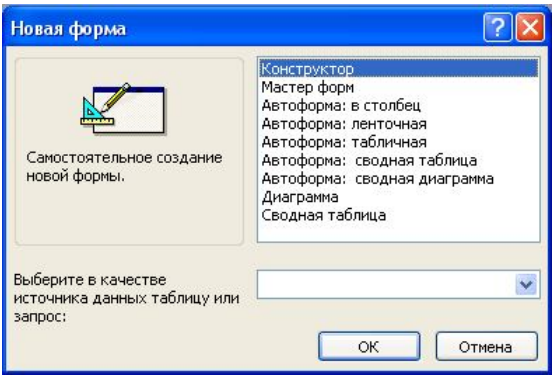


Рис. 34

Видно, что существует три стандартных вида автоформ: в столбец, ленточная и табличная. На рис. 35 и 36 последовательно показаны изображения первых двух автоформ на примере *Таблицы5*.

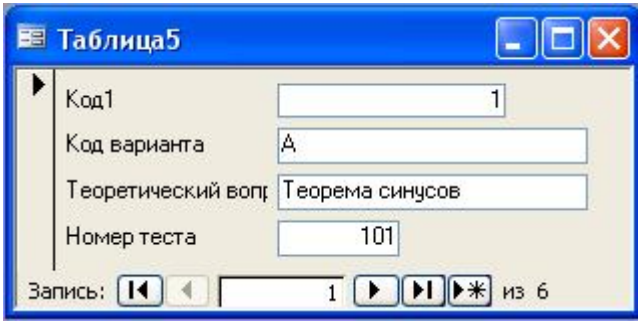


Рис. 35

Рис. 36

При просмотре этих форм в меню **Окно** появляется команда **По размеру формы**, позволяющая автоматически настраивать окно формы.

Табличная форма (рис. 37) имеет два режима: режим таблицы и режим формы.

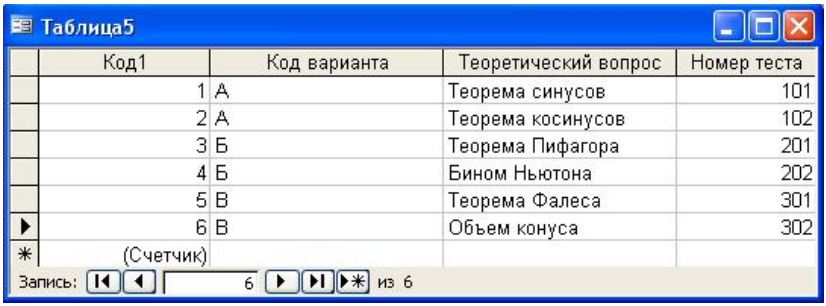


Рис. 37

Переключение между режимами происходит по команде: **Вид → Режим таблицы** (или **Режим формы**). Подчеркнем, что в режиме формы применяется команда **По размеру формы**.

Мастер форм

Форму можно также создать с помощью **Мастера форм**. Это делается в четыре шага: на первом шаге выбираются поля необходимые для построения формы → второй шаг предназначен для выбора вида формы (в один столбец, ленточный, табличный) → третий шаг определяет стиль формы → четвертый шаг задает имя формы.

Замечание. Мастер позволяет быстро создать форму, но ее качество может не удовлетворять пользователя. В этом случае следует создать или доработать форму в режиме конструктора.

Конструктор формы

Для создания новой формы в режиме конструктора можно воспользоваться процедурой: в окне БД (см. рис. 1) нажать кнопку **Формы** панели **Объекты** → нажать кнопку **Создать** → в окне **Новая форма** выбрать пункт **Конструктор** → выбрать имя таблицы, для которой создается форма → **OK**.

Замечание. Если не выбирать пункт **Новая форма**, а сразу нажать кнопку конструктор, то список полей не появится. Для его отображения следует выполнит процедуру: **Вид → Свойства** → в открывшемся окне выбрать вкладку **Все** → в пункте **Источник записей** раскрыть список и выбрать имя нужной таблицы → после появления в окне списка полей заголовка и его заполнения закрыть окно свойств формы.

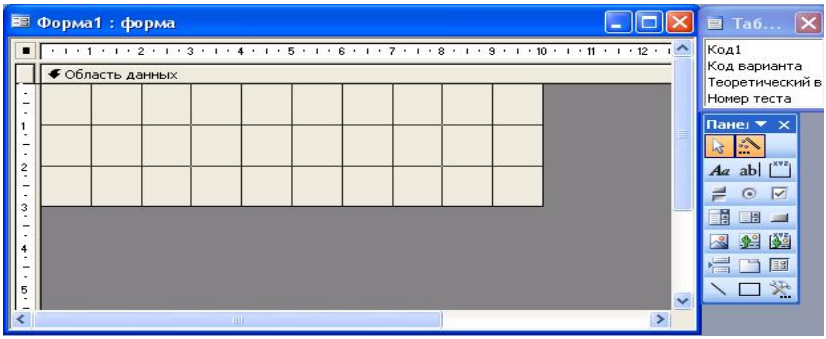


Рис. 38

С открытием окна конструктора разворачивается панель элементов и окно списка полей (рис. 38).

Замечание. Если в режиме конструктора не отобразилась панель элементов или список полей, то следует нажать кнопку **Панель элементов** и **Список полей** панели инструментов **Конструктор форм**.

Замечание. В форме целесообразно вначале разместить поля таблицы, в которые заносятся данные.

В MS Access *элементами управления* называют отдельные конструктивные элементы, которые размещаются в окне формы. При установке элемента в форму ему присваивается уникальное *имя элемента управления* и он обладает набором свойств, просмотр и редактирование которых возможно через диалоговое окно свойств (Открывается при выделении элемента и выборе команды в контекстном меню).

Элементы управления делятся на три типа: *присоединенные*, *свободные* и *вычисляемые*.

Присоединенные элементы связаны с полем исходной таблицы и используются для отображения данных, ввода записей в таблицу и их обновления. Обычно полям таблицы соответствуют присоединенные элементы управления типа **Поле**. Для отображения полей логического типа применяют **переключатели**, **флажки** и **выключатели**. Все вышеуказанные элементы управления обычно применяются с соответствующими им подписями, которые соответствуют значению свойства элемента **Подпись**.

Свободные элементы управления независимы от источников данных и позволяют выводить на экран текст, линии, объекты OLE, которые содержатся не в таблице, а в самой форме.

Вычисляемые элементы не обновляют поля таблицы, но используют из нее данные для вычисления записей по некоторым формулам.

Элементы управления вставляются в форму путем выбора соответствующих кнопок на панели элементов. Описание некоторых из них представлено в табл. 1.

Таблица 1

Имя кнопки	Назначение
Выбор объектов	Позволяет использовать указатель мыши для выделения элемента управления, раздела или формы
Мастера	Включает и отключает мастера по созданию элементов управления
Надпись	Создает элемент управления, который отображает в форме некоторый поясняющий текст
Поле	Создает элемент управления **Поле** (вместе с при-

	соединенной к нему надписью)
Группа переключателей	Создает элемент управления *Группа переключателей*. Элементы группы могут быть флажки, переключатели или выключатели
Выключатель	Создает элемент управления *Выключатель*, присоединяемый к логическому полю таблицы. Отображается в виде кнопки, которая в состоянии «включено» (утопленное положение) соответствует логическому ***Да***, а в состоянии «выключено» (приподнята) – логическому ***Нет***
Переключатель	Создает отдельный элемент управления *Переключатель*, аналогичный по своим функциям выключателю. Отображается в виде круглой кнопки и строки с поясняющим текстом. Если переключатель активизирован, то внутри кнопки появляется черная точка
Флажок	Создает элемент управления *Флажок*, аналогичный по своим функциям выключателю. Отображается в виде строки текста и маленького квадрата, в котором ставится галочка, если результат истинный, в случае ложного результата – квадрат пуст
Поле со списком	Создает элемент управления *Поле со списком*, в котором объединены поле для ввода значений и раскрывающийся список с заранее определенными значениями
Список	Создает раскрывающийся список допустимых опций выбора. Ввод других значений невозможен
Кнопка	Создает кнопку, нажатие которой активизирует выполнение определенных действий
Рисунок	Позволяет добавить в форму рисунок, который не является объектом OLE
Линия	Создает прямую линию, которая помогает визуально отделить родственные данные или разделы формы
Прямоугольник	Создает прямоугольник произвольного размера, который используется для создания графических эффектов

Элемент управления поле

Элементы управления, эквивалентные указанным в списке полям, помещаются в разделе формы, которая называется **Область данных**. Для добавления в форму элемента управления, соответствующего полю, следует выполнить следующие действия: в окне **Списка полей**

выделить нужное поле и, не отпуская левую кнопку мыши, перетащить поле в раздел **Область данных формы** → отпустить кнопку мыши. После этой процедуры в форме появится элемент управления типа **Поле** с присоединенным заголовком. Указанную процедуру следует проделать для переноса в форму всех необходимых полей.

Когда все элементы управления размещены в форме необходимо на панели инструментов **Конструктор форм** нажать кнопку **Сохранить** → в открывшемся диалоговом окне **Сохранение** указать имя формы → нажать кнопку **OK.** Указанные действия сохраняют структуру формы.

Специальные элементы управления

С помощью *специальных элементов управления* обеспечивается возможность выбора вводимых значений из ограниченного набора данных. К *специальным элементам управления* относятся: *список, поле со списком, переключатели, флажки, выключатели, группа переключателей и кнопки.*

Список и поле со списком используется, когда заранее известно, что поле будет содержать значение из определенного множества вариантов. **Список** всегда отображается уже открытым и некоторая его часть уже видна на экране. Пользователь может выбрать только значения из представленного списка. **Поле со списком** выглядит как обычное поле формы, в правой части которого расположена кнопка со стрелкой. **Поле со списком** позволяет вводить значения отличные от предложенных вариантов.

Замечание. Преобразовать элемент **Поле со списком** на **Список**, и наоборот, можно, выделив в режиме конструктора нужный элемент и в контекстном меню выбрав команду **Преобразовать элемент в**, после чего указать новый тип элемента управления.

Создание указанных элементов управления происходит в соответствии с процедурой: активизировать кнопку **Мастера** на **панели элементов** → нажать кнопку **Поле со списком (Список) панели элементов** → поместить указатель в то место формы, где будет находиться выбранный элемент → нажать левую кнопку мыши и вычертить прямоугольник, определяющий границы элемента → отпустить левую кнопку мыши → в открывшемся окне мастера указать способ получения значений для нового элемента управления → нажать кнопку **Далее** → в третьем окне мастера следует указать, какие действия нужно предпринять программе после выбора значения из списка (например,

запомнить значение или **Сохранить в поле**) → **Далее** → указать текст надписи для нового **поля со списком** или **списка** → **Готово**.

Группа переключателей

Для данных логического типа имеется три элемента управления: *переключатели*, *флажки* и *выключатели*. Каждый по отдельности из них возвращает значение типа *Да/Нет*. Если эти элементы поместить в **группу переключателей**, то они будут действовать согласованно. Иными словами, в группе может быть выбран один только элемент, числовое значение которого присваивается всей группе, и затем передается в поле таблицы.

Процедура создания **группы переключателей** состоит в следующем: активизировать кнопку **Мастера** на **панели элементов** → нажать кнопку **Группа переключателей** → переместить указатель мыши в то место, где будет размещаться элемент управления → нажать левую кнопку мыши и, не отпуская ее, нарисовать прямоугольник, в котором будут размещаться элементы **группы переключателей** (не забудьте отпустить кнопку мыши после определения границ группы!) → мастер создания группы, запущен и в первом окне следует указать надписи для каждого переключателя → **Далее** → во втором окне мастера указывается переключатель, используемый по умолчанию → **Далее** → третье окно мастера предназначено для присвоения значений каждому элементу группы при их выборе (эти числовые значения присваиваются всей **Группе переключателей**, и следовательно, связанного с ним поля) → **Далее** → четвертое окно предназначено для указания действия, которое будет выполнено программой MS Access при выборе одного из элементов группы (при необходимости связать группу переключателей с полем таблицы следует отметить пункт **сохранить значение в поле**, при использовании значения присвоенного надписи, следует выбрать пункт **сохранить значение для дальнейшего применения**) → **Далее** → пятое окно мастера предназначено для выбора стиля оформления **группы переключателей** → **Далее** → в заключительном окне мастера вводится текст надписи для группы → **Готово**.

Подкорректировать свойства **группы переключателей** можно, открыв окно свойств этого элемента.

Элемент управления Кнопка

Для запуска конкретного процесса из формы применяется **элемент управления Кнопка**. Категории таких процессов представлены в табл.2.

Таблица 2

Категории	Действия
Переход по записям	Найти далее, найти запись, первая запись, последняя запись, предыдущая запись, последующая запись
Обработка записей	Позволяет восстановить, добавить дублировать, печатать, удалить или сохранить запись
Работа с формой	Можно закрыть форму, изменить фильтр формы, обновить данные, открыть страницу или форму, напечатать форму и применить фильтр формы
Работа с отчетом	Отправить отчет в файл или по почте, печатать или просмотреть отчет
Приложение	Можно выйти из MS Access, запустить Word или Excel или другое приложение
Разное	Запускается автонабор номера, выполняется запрос или макрос, печатается таблица

Создание ***элемента управления Кнопка*** происходит по следующей процедуре: активизировать кнопку **Мастера** на **панели элементов** → активизировать элемент **Кнопка** → отметить указателем место формы, где будет располагаться **Кнопка** (запустится **Мастер**) → в первом окне **Мастера создание кнопок** выбирается **Категория** и **Действие**, которое будет выполняться при нажатии кнопки → **Далее** во втором окне мастера определяется внешний вид кнопки → **Далее** → в заключительном окне указывается имя кнопки → **Готово**.

Подчиненная форма

При существовании между таблицами связей типа «один-к-одному» или «один-ко-многим» в форме могут одновременно размещаться зависимые данные из этих таблиц.

Для встраивания подчиненной формы следует запустить соответствующий мастер и выполнить ряд действий: активизировать кнопку **Мастера** на **панели элементов** → там же нажать кнопку **Подчиненная форма/Отчет** → в режиме конструктора нарисовать мышью область, в которой будет находиться подчиненная форма (запустится **Мастер подчиненных форм**) → первый шаг мастера служит для указания источника данных для новой формы (если форма уже есть, то следует выбрать ее имя, если это таблица, ее имя указывается на втором шаге мастера) → **Далее** → второй шаг служит для указания таблицы и полей подчиненной формы → **Далее** → третий шаг посвящен оп-

ределению характеристик межтабличных связей → **Далее** → четвертый шаг задает имя для подчиненной формы → **Готово**.

На этом мы завершаем краткое введение в Microsoft MS Access. Нерассмотренными остались вопросы, связанные с поиском данных в БД, запросами, отчетами и многими другими важными аспектами СУБД. Но понятие СУБД у читателя данного пособия теперь имеется, как и возможность строить, вначале несложные, свои собственные базы данных.

Введение в программирование

Структурное программирование

С момента зарождения программирования было создано множество языков общения человека с ЭВМ. Сейчас, по видимости, наибольшее распространение имеют языки программирования C++, Delphi, Visual Basic (Visual Basic for Application) и некоторые другие. При этом в каждом из этих языков есть поддержка всех классических управляющих конструкций.

К 70-м годам XX века стало ясно, что программные проекты стали слишком сложными для успешного проектирования, кодирования и отладки в приемлемые сроки. Размер программ достиг величин, при которых программисты не могли с уверенностью сказать, что созданный программный продукт всегда выполняет то, что требуется, и что он не выполняет ничего такого, что не требуется. Назрела проблема изменения подходов к созданию больших программных продуктов.

В 1969 году Э. Дейкстра на международной конференции по программированию впервые использовал термин «структурное программирование» и предложил принципиально новый способ создания программ. Программа рассматривалась им как совокупность иерархических абстрактных уровней, которые позволяют четко структурировать программу, что позволяет лучше ее понимать, доказывать корректность ее работы и тем самым повышать надежность функционирования программы и сократить сроки ее разработки.

Правила структурной методологии разрабатывались такими учеными как Вирт, Дейкстра, Дал, Хоар, Иордан и другими. В этой связи следует отметить знаменитые книги Вирта [1, 2] и сборник [3].

Цели структурного программирования

I. *Обеспечить дисциплину программирования* в процессе создания программных комплексов. Дейкстра дал следующее определение:

«Структурное программирование – это дисциплина, которую программист навязывает сам себе».

II. *Улучшить читабельность программы*. Для этого следует избегать использования языковых конструкций с неочевидной семантикой;

стремиться к локализации действия управляющих конструкций и использования структур данных; разрабатывать программу так, чтобы ее можно было читать от начала до конца без управляющих переходов на другую страницу.

III. *Повышать эффективность программы*. Данное положение достигается при структурировании программы, разбивании ее на модули так, чтобы можно было легко находить и корректировать ошибки, а также чтобы текст любого модуля с целью увеличения эффективности можно было переделать независимо от других.

IV. *Повышать надежность программы.* Надежность обеспечивается хорошим структурированием программы при разбивке ее на модули и выполнением правил написания читабельных программ, что ведет к возможности сквозного тестирования и не создает проблем для организации процесса отладки.

V. *Уменьшать время и стоимость программной разработки*. Этот пункт выполняется, когда повышается производительность труда программиста, чему способствуют правила структурного программирования.

Основные принципы структурной методологии

Принцип абстракции. Абстракция позволяет придумать решение задачи без сиюминутного учета множества деталей. Поэтому проблема может рассматриваться по уровням: верхний уровень показывает большую абстракцию, упрощает взгляд на проект, нижний – показывает мелкие детали реализации задачи.

Принцип формальности. Формальность позволяет перейти от импровизации к строгому инженерному подходу при написании программ. Более того, этот принцип дает основания для доказательства правильности программ, так как позволяет изучать алгоритмы как математические объекты.

Принцип «разделяй и властвуй». Этот принцип означает возможность разделения программы на отдельные модули, которые просты по управлению и допускают независимую отладку и тестирование.

Принцип иерархического упорядочения. Данный принцип тесно связан с принципом «разделяй и властвуй» и выдвигает требования иерархического структурирования взаимосвязей между модулями программного комплекса, что облегчает достижение структурного программирования.

Модульное программирование

Модульное программирование – это организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняется определенным правилам. Первым основные свойства программного модуля сформулировал Парнас (Parnas): «Для написания одного модуля должно быть достаточно *минимальных* знаний о тексте другого». Но только в 1975 году Н. Виртом впервые была предложена специализированная синтаксическая конструкция модуля и включена в новый язык программирования **Modula.** Сейчас аналогичные конструкции имеются во многих языках.

Практические рекомендации

В [4] обсуждаются различные этапы и приемы программирования и разработки алгоритмов. Некоторые из них, применительно к обсуждаемым здесь вопросам сформулированы ниже.

1. *Никаких трюков и заумного программирования*. Не нужно применять сложных методов там, где можно использовать простые.

2. *Как можно меньше переходов*. Программирование без **go to** – это еще не структурное программирование. В некоторых ситуациях переход по **go to** является лаконичным, простым и ясным средством. Но разумное применение конструкций **if-then-else** и **for-do** способствует большей ясности, чем использование оператора **go to.**

3. Выбор с использованием конструкции **if-then-else**. Цель данного положения – обеспечение простоты хода вычисления, без каких то ни было переходов извне внутрь рассматриваемой структуры.

4. *Простота циклов.* Переходы по программе назад почти всегда можно представить как некоторую форму цикла. Существуют формы цикла **for-do**, **while-do**, **repeat-until**.

Можно написать

k:=0; **while** k<1000 **do begin** операторы; k:=k+1 **end**.

Но проще и более ясно записать так

for k:=0 **step** 1 **until** 999 **do** операторы.

Таким образом, каждое средство языка программирования следует применять по назначению.

5. *Сегментация.* Громоздкие программы, состоящие из одной основной программы, как правило, невозможно читать. В отсутствие конструкций **if-then-else** она будет перегружена метками и переходами и ее структура будет неясна. Если имеется возможность использования **if-then-else**, то глубина вложенности циклов может стать настолько

большой, что отыскание для каждого **if-then** соответствующего **else** будет затруднительно.

Чтобы обойти указанные трудности, каждую большую программу можно разбить на множество модулей или процедур (подпрограмм или функций), спроектированных так, что цель для каждой из них определена (логическая часть исходной задачи), и в каждой по возможности используются собственные локальные переменные. Не нужно стремиться разделить программу на равные куски («куски» в этом случае – самое подходящее слово для того, что получится при делении программы на равные части), а следует выделять логические фрагменты. Некоторые из них окажутся достаточно малыми для того, чтобы их можно было в таком виде оставить. Другие же, в свою очередь, потребуют разбиения на процедуры следующего уровня.

6. *Рекурсия*. Все, что можно запрограммировать при помощи простого цикла, как правило, так и следует программировать. Но в некоторых ситуациях рекурсия оказывается естественной и понятной. Примером может служить программа вычисления факториала:

```
Function Nfact(j As Integer)
  Dim ii As Integer
  Dim fff As Long
    If j = 1 Then fff = 1 Else fff = j * Nfact(j - 1)
  Nfact = fff
End Function

Sub factorial()
Dim i, N As Integer
Dim F, FF As Long
F = 1
N = Worksheets("Factorial").Cells(1, 1).Value
For i = 1 To N
  F = F * i
Next i
Worksheets("Factorial").Cells(5, 1).Value = F
Worksheets("Factorial").Cells(5,5).Value= Nfact(N)
End Sub
```

7. *Содержательные обозначения*. Может показаться, что краткие имена ускоряют и упрощают написание программы. Но. Когда через некоторое время программу требуется модифицировать, простота по-

нимания, обеспечиваемая содержательными обозначениями, полностью окупают затраченные при программировании усилия.

Очевидно, что к приведенным семи правилам можно добавить некоторые другие. Но если придерживаться перечисленных правил, то можно писать программы, которые легко понимать и сопровождать.

Некоторые понятия об объектно-ориентированной методологии программирования

Следующим этапом развития науки программирования стало создание объектно-ориентированной методологии.

Объектно-ориентированная методология программирования преследует те же цели, что и структурная, но решает их с другой отправной точки. По определению Г. Буча [1], «объектно-ориентированное программирование – это методология программирования, которая основана на представлении программы в виде совокупности объектов, каждый из которых является реализацией определенного класса (типа), а классы (типы) образуют иерархию на принципах наследуемости».

Одним из принципов управления сложностью проекта является декомпозиция. Г. Буч выделяет две разновидности декомпозиции: алгоритмическую, которую поддерживают структурные методы, и объектно-ориентированную, отличие которой состоит в следующем: «Разделение по алгоритмам концентрирует внимание на порядке происходящих событий, а разделение по объектам придает особое значение факторам, либо вызывающим действия, либо являющимся объектами приложения этих действий». Другими словами, *алгоритмическая декомпозиция учитывает в большей степени структуру взаимосвязей между частями сложной проблемы, а объектно-ориентированная уделяет большее внимание характеру взаимодействий.*

На практике следует использовать обе разновидности декомпозиции. При создании крупных проектов сначала следует применить объектно-ориентированный подход для конструирования общей иерархии объектов, отражающих сущность программной задачи, а затем использовать алгоритмическую декомпозицию на модули для упрощения разработки, отладки и сопровождения программного комплекса.

Следует подчеркнуть [3], что в некоторых областях, таких как интерактивная графика, объектно-ориентированное программирование весьма полезно. В других задачах, таких как классические арифметические типы и вычисления, основанные на них, похоже трудно найти

применение чему-то большему, чем абстракция данных, а средства, необходимые для поддержки объектно-ориентированного программирования, выглядят бесполезными.

Visual Basic for Application

Применение языка программирования VBA для решения некоторых задач диктуется желанием научить студентов создавать приложения для Microsoft Office, а VBA встроен в Word, Excel, MS Access и некоторые другие приложения. Отметим, что с появлением net – технологий эта задача может быть упрощена. Для углубленного изучения VBA можно порекомендовать книгу [7].

Сразу отметим, что ниже не дается всестороннее описание конструкций VBA, а рассматриваются примеры и даются небольшие комментарии к программам. Выбор примеров продиктован интересами авторов данного пособия и курсами, которые они читают на различных факультетах Санкт-Петербургского государственного университета.

Конструкции VBA

Описание переменных

Явное описание переменных позволяет избежать многих ошибок в коде программы. Поэтому предусмотрена команда **Option Explicit**, которая запрещает неявное объявление переменных, но действует только в модуле, в котором она появляется. Для того чтобы эта команда была в каждом модуле, следует выполнить действия: **Tools → Options →** вкладка **Editor →** установить флажок **Require Variable Declaration → OK**. Теперь требование явного объявления переменных действует во всех приложениях Microsoft Office, использующих VBA.

Для явного описания переменных используется оператор **Dim** со следующим синтаксисом: **Dim** *имя переменной* **As** *тип переменной*, [*имя переменной* **As** *тип переменной*]

Пример.

```
Dim n As Integer
Dim s As Double, R As Double
```

Следует помнить, что

1. Переменная, объявленная в процедуре, доступна только в этой конкретной процедуре.

2. Для того чтобы переменная была доступна во всех процедурах модуля, следует поместить объявление переменной в начале модуля перед объявлением процедур.

Во многих приложениях огромную роль играют совокупности переменных имеющий одинаковый математический или физический смысл. Такие переменные часто записывают в виде массивов, под которыми подразумеваются коллекции переменных с общим именем и одинаковым базовым типом. Объявление массива происходит с помощью оператора **Dim**:

Dim имя массива ([измерение массива]) **As** тип компонентов массива

Измерение массива представляет собой верхний и нижний индексы элементов массива, разделенные зарезервированным словом **to**.

Пример.

Объявление одномерного массива из 150 целых чисел

Dim A(1 **to** 150) **As** Integer

Объявление двухмерного массива, эквивалентного матрице 16x21

Dim B(0 **to** 15, 0 to 20) **As** Integer

Обращение к элементам массива происходит по имени, например, A(5) или B(8,7). VBA позволяет создавать массивы, имеющие до 60 измерений.

Арифметические операции

В VBA выполняются все обычные арифметические операции (см. табл. 3).

Таблица 3

Знак	Синтаксис	Описание
+	C+D	Сложение. A=C+D
-	C-D	Вычитание. A=C-D
*	C*D	Умножение. A=C*D
/	C/D	Деление. A=C/D
\	C\D	Целочисленное деление. Делит C на D и отбрасывает любую дробную часть так, чтобы результат был целым числом. A=C\D
Mod	C Mod D	Деление по модулю. Делит C на D, возвращая только остаток операции деления. A=C Mod D. Например, 8 Mod 2 возвращает 0, 5 Mod 3 возвращает 2
^	C^D	Возведение в степень. A=C^D

Операции сравнения

Операции сравнения используются для сравнения переменных значений любого сходного типа и возвращающие логическое значение **Thru** или **False** (табл. 4).

Таблица 4

Операция	Синтаксис	Описание
=	A=B	Равенство. **True.** Если A равно B, иначе **False**
<	A<B	Меньше, чем. **True**, если A меньше B, иначе **False**
<=	A<=B	Меньше, чем или равно. **True**, если A меньше или равно B, иначе **False**
>	A>B	Больше, чем. **True**, если A больше B, иначе **False**
>=	A>=B	Больше, чем или равно. **True**, если A больше или равно B, иначе **False**
<>	A<>B	Не равно. **True**, если A не равно B, иначе **False**

Логические операторы

Результат логической операции является значение типа Boolean (табл. 5).

Таблица 5

Оператор	Синтаксис	Название операции и ее описание
And	A and B	Конъюнкция. **True**, если A и B имеют значения **True**, иначе **False**
Or	A Or B	Дизъюнкция. **True**, если A, B или оба имеют значения **True**, иначе **False**
Not	Not A	Отрицание. **True**, если A имеет значение **False** и **False**, если A имеет значение **True**
Xor	A Xor B	Исключение. **True**, если A равно **Thru** или B равно **True**, иначе **False**
Eqv	A Eqv B	Эквивалентность. **True**, если A имеет то же значение, что и B, иначе **False**
Imp	A Imp B	Импликация. **False**, когда A является равным **True** и B равно **False**, иначе **True**

Приоритеты выполнения операций

При выполнении сложных выражений VBA следует правилам:

1. Части выражения, заключенные в круглые скобки, всегда выполняются в первую очередь.
2. Конкретные операции выполняются в зависимости от иерархии операторов (табл. 6).
3. Когда операторы имеют равный уровень приоритета, они вычисляются в порядке слева направо.

VBA вычисляет выражения в следующем порядке:

1. Знаки арифметических операций.
2. Знаки конкатенации строк.
3. Операторы сравнения.
4. Логические операторы.

Таблица 6

Оператор	Комментарии
^	Возведение в степень наивысший приоритет
-	Унарный минус
*,/	Умножение и деление имеют равные приоритеты
\	
Mod	
+,-	Сложение и вычитание имеют равный приоритет
&	Всякая конкатенация строк выполняется после любых арифметических операций в выражении и перед любыми операциями сравнения или логическими операциями
<,<=,>,>=,Like=,<>,is	Все операции сравнения имеют равные приоритеты
Not	
And	
Or	
Xor	
Eqv	
Imp	

Условный блок if-then-else

Часто решения математических, физических, экономических и других задач содержат ветвления в вычислительном алгоритме, которые происходят при некоторых условиях. Проверка этих условий осуществляется при помощи логических выражений. VBA в своем составе

имеет несколько операторов для организации этих действий. Наиболее общим, по всей видимости, является *условный блок* со следующим синтаксисом:

If <*условие*> **then**
<*Операторы 1*>
else
<*Операторы 2*>
End If

Здесь <*условие*> – любое логическое выражение, <*Операторы1*>, <*Операторы 2*> – совокупности действий, первая из которых выполняется при принятии условием значения true, вторая – при выработке условием значения false. Ключевые слова **End If** указывают на закрытие условного оператора.

Организация циклов

В современных языках программирования обычно присутствуют три оператора для выполнения повторяющихся фрагментов программы. VBA их имеет несколько больше, но наиболее важными являются операторы циклов: **For, While** и **Do … Until**.

Оператор цикла ***For*** имеет синтаксис:

For <*параметр цикла*> = <*начальное значение*> **To** <*конечное значение*>
<*операторы*>
Next <*параметр цикла*>

Здесь <*параметр цикла*> – переменная типа integer, <*начальное значение*> и <*конечное значение*> – выражения того же типа, <*операторы*> – действия, которые повторяются в цикле.

Оператор цикла ***While*** имеет синтаксис:

Do While <*условие*>
<*операторы*>
Loop

Здесь <*условие*> – логическое выражение, <*операторы*> – действия, которые выполняются, если <*условие*> принимает значение **True**. Как только <*условие*> принимает значение **False**, происходит выход из цикла. Операторы, которые стоят внутри цикла, могут вообще не выполняться. Это произойдет, если <*условие*> примет сразу значение **False**.

Оператор цикла ***Do…Until*** с постпроверкой условия имеет синтаксис:

Do

<тело цикла>

Loop Until <условие>

Здесь **Do**, **Loop**, **Until** зарезервированные слова, *<тело цикла>* – последовательность операторов, которая выполняется при работе цикла, <условие> – логическое выражение. Если <условие> принимает значение **False**, то операторы, из которых состоит *<тело цикла>*, повторяются. В случае, когда

<условие> принимает значение **True**, цикл завершает свою работу. Подчеркнем, что *<тело цикла>* выполняется хотя бы один раз, и только потом проверяется условие прекращения работы цикла.

Замечание. В операторах циклов VBA можно заменить ключевое слово **While** на **Until** и наоборот. Этот выбор диктуется правилами:

1. Следует использовать **While**, если цикл продолжает выполняться, пока условие равно **True**.
2. Следует использовать **Until**, если цикл продолжает выполняться, пока условие равно **False**.

Реализация некоторых алгоритмов средствами VBA

Вычисление площади круга

Вход в редактор VBA осуществляется нажатием комбинации клавиш <Alt> + <F11>. Далее следует, исходя из сказанного выше об описании переменных, записать команду **Option Explicit.** Заметим, что всякая вычислительная задача в VBA оформляется как процедура, поэтому следует написать ключевое слово **Sub**, нажать клавишу пробел и ввести имя подпрограммы, после которого в круглых скобках разместить параметры, передающиеся в процедуру извне. Если никакие параметры не передаются, то в круглых скобках ничего не пишется. Нажатие клавиши <Enter> приводит к появлению строчки со словами **End Sub**, которые закрывают процедуру. Текст программы следует вводить между словами **Sub** и **End Sub**.

Замечание. Программы, приведенные ниже, написаны в VBA для Excel.

Первая задача будет наипростейшей. Необходимо вычислить площадь круга по формуле $S = \pi R^2$. Программа, приведенная ниже, запрашивает радиус круга и выдает значение площади.

```
Option Explicit
Sub my_program1()
Const pi = 3.1415926
Dim s, R As Double
Dim BoxTitle As String
BoxTitle = "input date"
R = InputBox("Input Radius", BoxTitle)
s = pi * R ^ 2
MsgBox  s, , "результат"
End Sub
```

Здесь вначале описана постоянная π (правила программирования требуют вначале описать постоянные, так как от них может зависеть описание переменных), затем вещественные s и R, BoxTitle – строковая переменная. Следующий оператор присваивает строковой переменной значение input date (строковые записи заключаются в кавычки). Переменной R мы присваиваем значение радиуса круга с помощью оператора **InputBox**. Параметрами этой функции являются обязательная подсказка, заключенное в кавычки строковое выражение, и необязательная строковая переменная, значение которой отображается в заголовке окна (рис. 39).

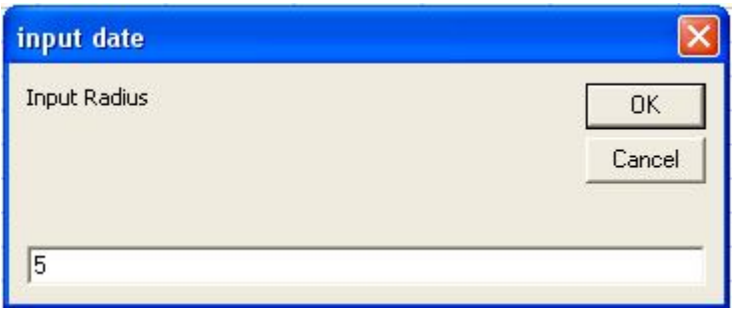


Рис. 39

После задания R, присваиваем переменной s значение площади. Вывод результата производится с помощью диалогового окна, вызываемого функцией **MsgBox** (рис. 40).

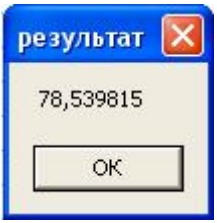


Рис. 40

Замечание. Полное описание функций, доступных пользователю, содержится в справке VBA. Там же присутствуют многочисленные примеры, показывающие применение этих функций.

Произведение матриц

Определение [8]. Пусть даны две матрицы

$$A=\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \text{ и } B=\begin{pmatrix} b_{11} & b_{12} & \dots & b_{1p} \\ b_{21} & b_{22} & \dots & b_{2p} \\ \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & a_{np} \end{pmatrix},$$

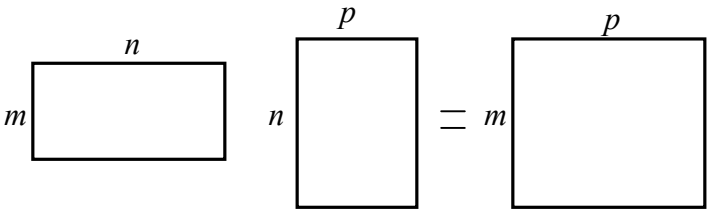
причем число столбцов матрицы A равно числу строк матрицы B. Произведением матрицы A на матрицу B называется матрица C:

$$C=\begin{pmatrix} c_{11} & c_{12} & \dots & c_{1p} \\ c_{21} & c_{22} & \dots & c_{2p} \\ \dots & \dots & \dots & \dots \\ c_{m1} & c_{m2} & \dots & c_{mp} \end{pmatrix},$$

обозначаемая через AB, элементы которой вычисляются по формулам

$$c_{ij}=\sum_{s=1}^{n} a_{is}b_{sj}\text{, где } i=1,2,\dots,m,\ j=1,2,\dots,p.$$

Еще раз отметим, что произведение двух матриц имеет смысл только при условии, которое графически выглядит так:



Программа перемножающая две матрицы представленные на листе Excel записывается так:

```
Option Explicit
Sub multiplcation_matrix()
Dim m, n, p As Integer
Dim A(1 To 20, 1 To 20) As Double
Dim B(1 To 20, 1 To 20) As Double
Dim C(1 To 20, 1 To 20) As Double
Dim i, j, k, ii, jj As Integer
Dim dd, ddddd, s As Double
m = Worksheets("произведение матриц").Cells(1, 1).Value
n = Worksheets("произведение матриц").Cells(1, 2).Value
p = Worksheets("произведение матриц").Cells(1, 3).Value

For i = 1 To m
  For j = 1 To n
    A(i, j) = Worksheets("произведение матриц").Cells(i + 2, j).Value
  Next j
Next i

For ii = 1 To n
  For jj = 1 To p
    B(ii, jj) = Worksheets("произведение матриц").Cells(m + 3 + ii,
jj).Value
  Next jj
Next ii

For i = 1 To m
  For j = 1 To p
    C(i, j) = 0
```

```
      For s = 1 To n
        C(i, j) = C(i, j) + A(i, s) * B(s, j)
      Next s
    Next j
  Next i
  For i = 1 To m
    For j = 1 To p
      Worksheets("произведение матриц").Cells(m + n + 5 + i, j).Value
= C(i, j)
    Next j
  Next i
  End Sub
```

Вычисление определителя квадратной матрицы

Во многих математических дисциплинах используется понятие определителя квадратной матрицы n-го порядка, под которым понимается *сумма всех* $n!$ *произведений элементов этой матрицы, взятых по одному из каждой строчки и по одному из каждого столбца; при этом каждое произведение снабжено знаком плюс или минус по некоторому правилу*.

Произведения, о которых говорится в определении, можно представить в виде

$$P = a_{1\alpha} a_{2\beta} \ldots a_{n\omega}. \qquad (1)$$

Первый индекс у сомножителя a_{ij} в произведении (1) соответствует номеру строки, второй – номеру столбца. Номера столбцов дают перестановку $(\alpha, \beta, \ldots, \omega)$. Если эта перестановка четная, то величина P, входящая в определение определителя, берется со знаком плюс, если нечетная – то со знаком минус.

Замечание. Непосредственное вычисление определителя по определению представляет собой трудоемкий процесс, так как нужно находить $n!$ произведений и определять знак каждого из них. Но можно заметить, что если все элементы, стоящие выше или ниже главной диагонали равны нулю, то определитель равен произведению элементов стоящих на этой диагонали: $a_{11} a_{22} \ldots a_{nn}$.

Полезными при вычислении определителей оказываются следующие их свойства:

- Определитель матрицы n-го порядка не изменится, если к элементам одной ее строки прибавить соответствующие элементы другой строчки, умноженной на одно и то же произвольное число.
- Если все элементы какой-нибудь строчки матрицы n-го порядка умножить на число c, то ее определитель также умножится на число c.

Из сделанного замечания и перечисленных свойств следует, что определитель целесообразно привести к треугольному виду и найти произведение элементов, стоящих на главной диагонали.

Программа, реализующая сказанное, имеет следующую запись:

```
Option Explicit

Sub determinant()
Dim n As Integer
Dim A(1 To 20, 1 To 20) As Double
Dim B(1 To 20, 1 To 20) As Double
Dim i, j, k, ii, jj As Integer
Dim dd, ddddd As Double

dd = 1

n = Worksheets("определитель").Cells(1, 1).Value
For i = 1 To n
  For j = 1 To n
    A(i, j) = Worksheets("определитель").Cells(i + 1, j).Value
  Next j
Next i

For i = 1 To n
  For j = 1 To n
    Worksheets("определитель").Cells(n + 2 + i + 1, j).Value = A(i, j)
  Next j
Next i

For ii = 1 To n
  For jj = 1 To n
    Worksheets("определитель").Cells(n + 2 + ii + 1, jj).Value = A(ii,
jj)
  Next jj
Next ii
```

```
dd = 1

For k = 1 To n - 1
  For ii = 1 To k
  For jj = 1 To n
    B(ii, jj) = A(ii, jj)
    Next jj
   Next ii

  For i = k + 1 To n
    For j = k To n
      B(i, j) = A(k, j) * A(i, k) - A(i, j) * A(k, k)
    Next j
    dd = dd * A(k, k)
  Next i

For ii = 1 To n
  For jj = 1 To n
     A(ii, jj) = B(ii, jj)
  Next jj
Next ii

Next k

For ii = 1 To n
  For jj = 1 To n
    Worksheets("определитель").Cells(n + 2 + ii + 1, jj).Value = B(ii,
jj)
  Next jj
Next ii

ddddd = 1
For k = 1 To n
ddddd = ddddd * B(k, k)
Next k
Worksheets("определитель").Cells(1, 9).Value = ddddd / dd
End Sub
```

Метод Гаусса решения системы линейных алгебраических уравнений

Пусть дана система линейных алгебраических уравнений, которая в матричной записи имеет вид

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}. \qquad (2)$$

Составим расширенную матрицу

$$\begin{matrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{matrix}. \qquad (3)$$

Определение. Под элементарными преобразованиями системы алгебраических уравнений понимаются следующие операции:

1) Умножение какого-либо уравнения системы на число, отличное от нуля.

2) Прибавление к одному уравнению другого уравнения, умноженного на произвольное число.

3) Перемена местами двух уравнений в системе.

В курсе алгебры доказывается теорема: при элементарных преобразованиях система переходит в равносильную систему (две системы линейных уравнений называются равносильными, если каждое решение одной из них является решением другой и наоборот).

Метод Гаусса решения системы линейных уравнений состоит в том, что при помощи элементарных преобразований систему приводят к такому виду, чтобы ее расширенная матрица из коэффициентов оказалась трапециевидной. После чего решение системы становится значительно проще.

Программа, решающая систему линейных алгебраических уравнений записывается так:

```
Sub Gaus()
Dim A(1 To 20, 1 To 21) As Double, X(1 To 20) As Double, _
    N As Integer, i As Integer, j As Integer, _
    n1 As Integer
```

```
N = Worksheets(1).Cells(1, 1).Value

For i = 1 To N
  For j = 1 To N + 1
    A(i, j) = Worksheets(1).Cells(i + 1, j).Value
  Next j
Next i

n1 = N + 1
For k = 1 To N
  k1 = k + 1
  s = A(k, k)
  j = k
  For i = k1 To N
    r = A(i, k)
    If Abs(r) > Abs(s) Then s = r:  j = i
  Next i
  If s = 0 Then MsgBox ("det=0 Решений нет")
  If j = k Then GoTo 1
    For i = k To n1
      r = A(k, i):  A(k, i) = A(j, i): A(j, i) = r
    Next i

For j = k1 To n1
    A(k, j) = A(k, j) / s
  Next j
  For i = k1 To N
    r = A(i, k)
    For j = k1 To n1
      A(i, j) = A(i, j) - A(k, j) * r
    Next j
  Next i
Next k
For i = N To 1 Step -1
  s = A(i, n1)
  For j = i + 1 To N
    s = s - A(i, j) * X(j)
  Next j
X(i) = s
```

```
Next i

For i = 1 To N
  Worksheets(1).Cells(i + 1, 8).Value = X(i)
Next i
  For j = 1 To N
  Worksheets(1).Cells(j + 1, 10).Value = 0

    For k = 1 To N
      Worksheets(1).Cells(j + 1, 10).Value = _
      Worksheets(1).Cells(j + 1, 10).Value + _
      Worksheets(1).Cells(j + 1, k).Value * X(k)
    Next k
  Next j

End Sub
```

Сортировка

Задача сортировки множества формулируется следующим образом [9]: задано конечное множество A, состоящее из n элементов, а на нем задано отношение линейного порядка P. Требуется перенумеровать элементы A числами от 1 до n таким образом, чтобы из неравенства $i < j$ следовало $(a_i, a_j) \in P$.

Обычно задача сортировки сводится к упорядочению чисел по значению.

Сортировка вставкой

Один из простейших способов сортировки состоит в следующем:

1) последовательность из одного элемента множества уже «отсортирована»;

2) берем следующий элемент и сравниваем его с предыдущим, при этом переставляем его вперед до тех пор, пока он не займет свое место.

```
Option Explicit

Sub Inset()
Dim i, j, k, N As Integer
Dim b As Double
```

```
N = Worksheets("сортировка вставкой").Cells(1, 1).Value
ReDim A(1 To N) As Double
'read array
For i = 1 To N
  A(i) = Worksheets("сортировка вставкой").Cells(2, i).Value
Next i

For i = 2 To N
  b = A(i)
 j = 1
  Do While b > A(j)
    j = j + 1
  Loop
  For k = i - 1 To j Step -1
    A(k + 1) = A(k)
  Next k
  A(j) = b
Next i
For i = 1 To N
  Worksheets("сортировка вставкой").Cells(5, i).Value = A(i)
Next i
End Sub
```

Сортировка выбором

Принцип метода:

1) выбираем в массиве элемент с минимальным значением на интервале от 1-го до n-го элемента и меняем его местами с первым элементом;

2) находим элемент с минимальным значением на интервале от 2-го до n-го элемента и меняем его местами со вторым элементом и так далее до $(n-1)$-го элемента.

```
Option Explicit
Sub sort2()
Dim i, N, imin, s As Integer
Dim min As Double
N = Worksheets("сортировка выбором").Cells(1, 1).Value
ReDim A(1 To N) As Double
'read array
For i = 1 To N
  A(i) = Worksheets("сортировка выбором").Cells(2, i).Value
```

```
Next i
For s = 1 To N - 1
  min = A(s)
  imin = s
  For i = s + 1 To N
    If A(i) < min Then min = A(i): imin = i
  Next i
  A(imin) = A(s)
  A(s) = min
Next s
For i = 1 To N
  Worksheets("сортировка выбором").Cells(5, i).Value = A(i)
Next i
End Sub
```

Сортировка обменом («пузырьковая» сортировка)

Содержание метода:

1) слева направо до конца массива поочередно сравниваются два соседних элемента, и если их взаиморасположение не соответствует условию упорядоченности, то они меняются местами;

2) после первого прохода на n-м месте стоит элемент с максимальным значением («всплыл» первый пузырек). Второй проход выполняется до $(n-1)$-го элемента, так как n-й уже на своем месте. Всего требуется $n-1$ проходов.

```
Option Explicit

Sub sort2()
Dim i, N, imin, s As Integer
Dim min As Double
N = Worksheets("сортировка выбором").Cells(1, 1).Value
ReDim A(1 To N) As Double
'read array
For i = 1 To N
  A(i) = Worksheets("сортировка выбором").Cells(2, i).Value
Next i
For s = 1 To N - 1
  min = A(s)
  imin = s
```

```
  For i = s + 1 To N
    If A(i) < min Then min = A(i): imin = i
  Next i
  A(imin) = A(s)
  A(s) = min
Next s
For i = 1 To N
  Worksheets("сортировка выбором").Cells(5, i).Value = A(i)
Next i
End Sub
```

Метод фон Неймана

Пусть даны два упорядоченных массива. Возьмем из них первые элементы и выберем тот, который должен следовать вначале в соответствии с некоторым условием, и запишем его в массив результата. Изменим рассмотрение индексов в массиве, из которого взят элемент. Применим изложенную процедуру к оставшимся массивам. Если один из массивов исчерпан, то остаток другого просто дописывается в результирующий массив. Описанная процедура называется *слиянием массивов*.

Дж. фон Нейман предложил метод сортировки, основанный на операции слияния:

1) исходный массив из n элементов вначале представляется в виде n «отсортированных» массивов, имеющих длину равную 1;

2) сольем эти массивы попарно, что приведет к появлению массивов длины 2;

3) повторим процедуру слияния, получая массивы длиной 4 и так далее. В итоге получается отсортированный массив длиной n.

Замечание. Недостатком этого метода является то, что для почти отсортированного массива тратится столько же времени, как и на самый «перемешанный» массив.

Еще один метод сортировки будет изложен в разделе Рекурсия.

Рекурсия

Рекурсивным называется объект, который частично определяется через самого себя. В программировании рекурсивная функция или процедура – это функция или процедура, которая вызывает сама себя.

Классическим примером сказанного является определения функции факториала. С одной стороны, факториал определяется следующей формулой:

$$n! = 1 \cdot 2 \cdot \ldots \cdot n ,$$

с другой – рекуррентными соотношениями

I. $0! = 1$

II. для $\forall n > 0 \quad n! = n \cdot (n-1)!$

Другим примером может служить определение чисел Фибоначчи

I. $F(1) = 1$

II. $F(2) = 1$

III. для $\forall n > 2 \quad F(n) = F(n-1) + F(n-2)$

Программы, содержащие рекурсивные процедуры, наглядны и содержат компактный код. Однако использование рекурсивных процедур требует большого размера оперативной памяти при выполнении кода, чем нерекурсивные процедуры. Это вызвано тем, что при каждом рекурсивном вызове для параметров процедуры и локальных переменных выделяются новые ячейки памяти.

Определение. Глубиной рекурсии называется максимальное число рекурсивных вызовов, которое происходит во время выполнения программы.

Определение. Текущим уровнем рекурсии называется число рекурсивных вызовов в каждый конкретный момент времени.

Формы рекурсивных процедур

Любая рекурсивная процедура включает в себя некоторое множество операторов S и один или несколько операторов рекурсивного вызова P. Следует особо подчеркнуть, что безусловные рекурсивные процедуры приводят к бесконечным процессам. Еще раз напомним, что каждая копия рекурсивной процедуры требует выделения дополнительной памяти, но память в любом устройстве конечна!

Таким образом, вызов рекурсивной процедуры должен выполняться по условию, которое на каком-то уровне рекурсии станет ложным.

Если условие истинно, то рекурсивный спуск продолжается. В момент принятием условием ложного значения спуск прекращается и начинается поочередный рекурсивный возврат из всех вызванных на данный момент копий рекурсивной процедуры.

Форма с выполнением действий до рекурсивного вызова (действия выполняются на рекурсивном спуске)

```
Sub P()
  S
 if <условие> then P()
End Sub
```

Форма с выполнением действий после рекурсивного вызова (действия выполняются на рекурсивном возврате)

```
Sub P ()
 if <условие> then P()
  S
End Sub
```

Форма с выполнением действий как до, так и после рекурсивного вызова

```
Sub P ()
  S1
 if <условие> then P()
  S2
End Sub
```
или
```
Sub P ()
 if <условие> then
          S1
          P()
          S2
End Sub
```

Многие задачи безразличны к выбору формы рекурсивной процедуры. Но существуют целые классы задач, при решении которых требуется сознательно управлять ходом работы рекурсивных процедур и функций. К таким задачам, в частности, относится обработка списков и древовидных структур данных.

Выполнение действий на рекурсивном спуске

Для реализации алгоритма вычисления факториала, работающего на спуске, вводим параметры: *mult* для выполнения на спуске операции умножения накапливаемого значения факториала на очередной множитель, *m* для обеспечения независимости рекурсивной функции от имени конкретной глобальной переменной.

```
Function Fact1(mult As Long, i, m As Integer) As Long
  mult = mult * i
  If i = m Then Fact1 = mult Else Fact1 = Fact1(mult, i + 1, m)
End Function
Sub FF1()
Dim N As Integer
N = Worksheets("Factorial").Cells(1, 1).Value
Worksheets("Factorial").Cells(15, 1).Value = Fact1(1, 1, N)
End Sub
```

Выполнение действий на рекурсивном возврате

Ниже представлена программа, в которой вычисление факториала происходит на рекурсивном возврате, и при этом рекурсивный вызов и оператор накопления факториала разделены явным образом.

```
Function Fact2(i As Integer) As Long
Dim mult As Long
If i = 1 Then mult = 1 Else mult = Fact2(i - 1)
Fact2 = mult * i
End Function
Sub FF2()
Dim N As Integer
N = Worksheets("Factorial").Cells(1, 1).Value
Worksheets("Factorial").Cells(16, 1).Value = Fact2(N)
End Sub
```

Быстрая сортировка (метод Quicksort)

Английский математик, профессор Оксфордского университета Хоар (Charles Antony Richard Hoare), предложил в 1962 году метод основанный на следующей идее: взять один из элементов массива и поставить его таким образом, что элементы предшествующие ему в упорядочении, размещались до него, а следующие за ним – после. После

этой операции исходный массив разбит на две части, которые можно упорядочить тем же способом.

Реализуем сказанное с помощью рекурсии [2]:

1. Выбираем центральный элемент массива и записываем его в переменную B. Затем элементы массива просматриваются поочередно слева направо и справа налево. При движении слева направо ищем элемент $A(i)$, который будет больше или равен B, и запоминаем его позицию. При движении справа налево ищем элемент $A(j)$, который меньше или равен B, и запоминаем его позицию. Найденные элементы меняем местами и продолжаем встречный поиск по указанным условиям. Этот процесс продолжаем, пока при очередной итерации поиска встречные индексы i и j не пересекутся. Таким образом, относительно значения B массив получается отсортированным. Остается упорядочить левую и правую части массива.

2. Описанную в предыдущем пункте процедуру повторяем для левой и правой частей по отдельности. После чего массив оказывается разбитым уже на четыре непересекающихся по сортировке части, которые можно упорядочить по отдельности. Процесс продолжается пока длина сортируемых частей не станет равной одному элементу и, следовательно, все элементы массива упорядочены.

Так как на каждом этапе выполняемые действия одни и те же, но в разных индексных промежутках, то удобно воспользоваться рекурсией. Программа Quicksort имеет вид:

```
Option Base 1
Option Explicit
Sub QS(A() As Double, L, R As Integer)
Dim i, j, B, tmp, LR As Integer

LR = (L + R) \ 2
B = A(LR)
i = L: j = R
Do While i <= j
  Do While A(i) < B: i = i + 1: Loop
  Do While A(j) > B: j = j - 1: Loop
    If i <= j Then
      tmp = A(i): A(i) = A(j): A(j) = tmp: i = i + 1: j = j - 1
    End If
Loop
```

```
If L < j Then Call QS(A, L, j + 1)
If i < R Then Call QS(A, i, R)
End Sub

Sub qqss()
Dim k, N As Integer
Dim A(1 To 20) As Double
N = 10
For k = 1 To N
 A(k) = Worksheets("Быстрая сортировка").Cells(1, k).Value
Next k
Call QS(A, 1, N)
For k = 1 To N
 Worksheets("Быстрая сортировка").Cells(2, k).Value = A(k)
Next k
End Sub
```

Решение задач механики

В этом разделе покажем, что VBA может быть применен для решения задач механического характера. При этом, размещая результаты расчетов в таблице MS Excel, мы получаем возможность графического отображения, полученных зависимостей.

Движение точки по инерции под действием силы трения

В соответствии со вторым законом Ньютона

$$m\mathbf{w} = \mathbf{F}$$

и законом Кулона

$$\mathbf{T} = -\mu N \frac{\mathbf{v}}{v}$$

уравнение движения точки по горизонтальной плоскости будет иметь вид (здесь μ – коэффициент трения, $\mathbf{v}$ – вектор скорости, v – величина скорости)

$$m\mathbf{w} = -\mu N \frac{\mathbf{v}}{v},$$

и после некоторых преобразований получаем дифференциальное уравнение

$$\ddot{x} = -\mu g\,. \qquad (1)$$

Аналитическое решение этого уравнения при начальных условиях $x(0) = 0$, $\dot{x}(0) = v_0$ имеет вид

$$x(t) = v_0 t - \frac{1}{2}\mu g t^2,\ v = v_0 - \mu g t\,. \qquad (2)$$

Откуда следует, что скольжение будет продолжаться в течении $t_* = \dfrac{v_0}{\mu g}$ с.

Получим численное решение уравнения (1) с помощью метода Рунге-Кутта 4-го порядка с постоянным шагом интегрирования. Реализация этого метода происходит по формулам [10]

$$k_1 = f(t_n, y_n), \quad k_2 = f\left(t_n + \frac{\tau}{2}, y_n + \frac{\tau k_1}{2}\right),$$

$$k_3 = f\left(t_n + \frac{\tau}{2}, y_n + \frac{\tau k_2}{2}\right), \qquad k_2 = f\left(t_n + \tau, y_n + \tau k_3\right)$$

(3)

$$y_{n+1} = y_n + \frac{\tau}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right).$$

Следует подчеркнуть, что величины $k_1, k_2, k_3, k_4, f, y_n$ и y_{n+1} являются векторами, а τ -скалярная величина, характеризующая шаг интегрирования по времени.

Для применения формул (3) уравнение (1) следует привести к нормальному виду:

$$\begin{aligned} \dot{x} &= v \\ \dot{v} &= -\mu g \end{aligned} \quad \text{или} \quad \begin{pmatrix} \dot{x} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} v \\ -\mu g \end{pmatrix}.$$

Программа, выводящая в первый столбец Excel листа время, во второй – значение скорости, а в третий – координату, записывается так:

```
Option Explicit
Const NN = 2
Sub prav(Time As Double, Y() As Double, YY() As Double)
Dim x1 As Double, F As Double
```

```
x1 = Y(2)
F = -9.81 * 0.2
YY(1) = x1
YY(2) = F
End Sub
Sub RK4(T As Double, Tau As Double, Yrk4() As Double)
'Tau-STEP OF INTEGRATION;
 'T-TIME;
 'Y-vector begin value for one step
'YY - resalt prav
Dim i As Integer
Dim K(1 To NN) As Double, K1(1 To NN) As Double, K2(1 To NN) As
Double
Dim K3(1 To NN) As Double, K4(1 To NN) As Double
Call prav(T, Yrk4, K1)
For i = 1 To NN
  K(i) = Yrk4(i) + K1(i) * Tau / 2
Next i
T = T + Tau / 2
Call prav(T, K, K2)
For i = 1 To NN
  K(i) = Yrk4(i) + K2(i) * Tau / 2
Next i
Call prav(T, K, K3)
For i = 1 To NN
  K(i) = Yrk4(i) + K3(i) * Tau
Next i
T = T + Tau / 2
Call prav(T, K, K4)
For i = 1 To NN
  Yrk4(i) = Yrk4(i) + Tau / 6 * (K1(i) + 2 * (K2(i) + K3(i)) + K4(i))
Next i

End Sub
Sub rung(N As Integer, A As Double, B As Double, H As Double, TH
As Double, T As Double, Y() As Double)

Dim THP As Double
Dim i As Integer, j As Integer
```

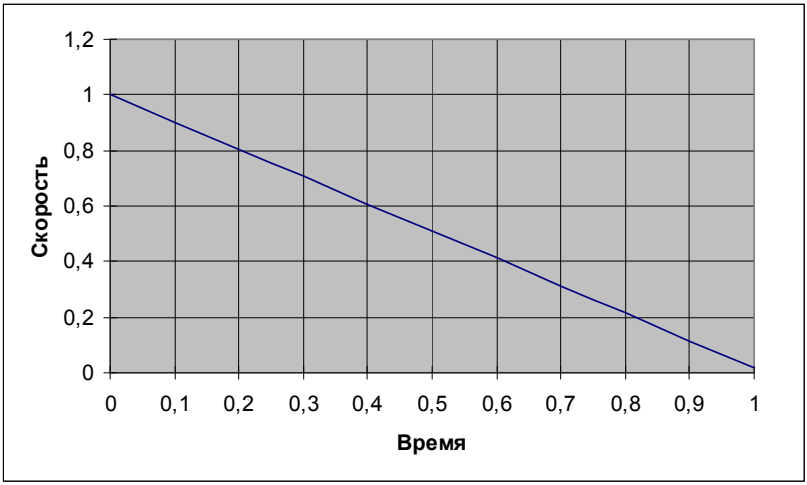
```
T = A
THP = A + TH
j = 2

Worksheets("RK4").Cells(j, 1).Value = T
For i = 1 To NN
   Worksheets("RK4").Cells(j, i + 1).Value = Y(i)
Next i

Do While (T <= B)

Call RK4(T, H, Y)

 If T >= THP Then
   THP = THP + TH
      j = j + 1
      Worksheets("RK4").Cells(j, 1).Value = T
   For i = 1 To NN
      Worksheets("RK4").Cells(j, i + 1).Value = Y(i)
   Next i
 End If

T = T + H
Loop
j = j + 1
      Worksheets("RK4").Cells(j, 1).Value = T
   For i = 1 To NN
      Worksheets("RK4").Cells(j, i + 1).Value = Y(i)
   Next i

End Sub
Sub solv()
Dim Y(1 To 2) As Double
Y(1) = 0: Y(2) = 1
Call rung(NN, 0, 1.0, 0.0001, 0.1, 0, Y)
End Sub
```

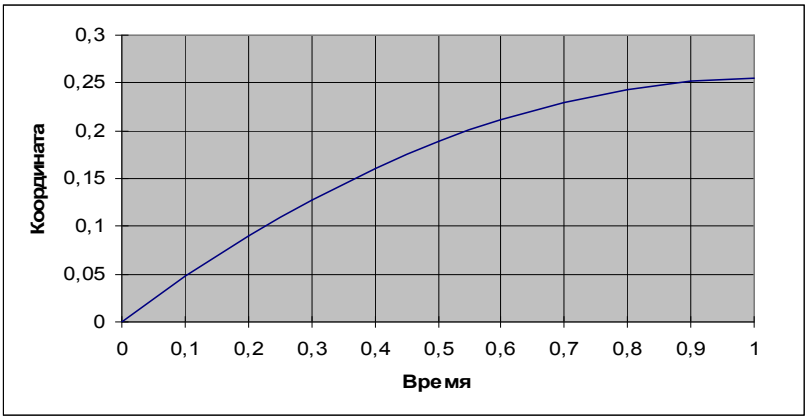
Отметим, что интегрирование происходит с шагом τ , но вывод в ячейки листа с шагом HT.

После получения таблицы с результатами интегрирования

0	0	1
	0,04	0,90
0,1	7638	1704
	0,09	0,80
0,2	027	3604
	0,12	0,70
0,3	7998	5504
	0,16	0,60
0,4	0821	7404
	0,18	0,50
0,5	8738	9304
	0,21	0,41
0,6	1751	1204
	0,22	0,31
0,7	9859	3104
	0,24	0,21
0,8	3062	5004
	0,25	0,11
0,9	1359	6904
	0,25	0,01
1	4752	8804

строим графики $v(t)$, $x(t)$:





Движение точки под действием восстанавливающей силы

Пусть точка массой m может перемещаться вдоль оси X. Предположим, что на точку действует восстанавливающая сила $-cx$.В соответствии с вышеизложенным имеем

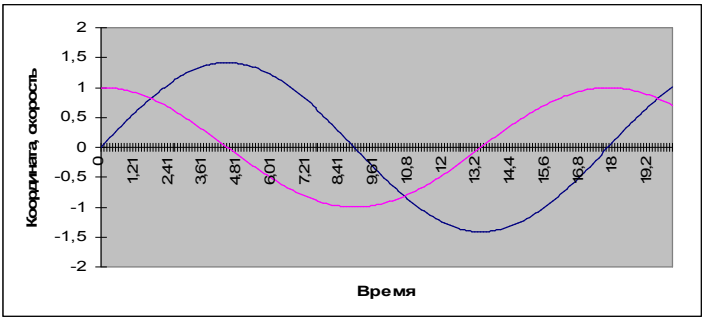
$$m\ddot{x} = -cx$$

или

$$\ddot{x} + k^2 x = 0\,,$$

где $k^2 = \frac{c}{m}$.

Решение данного уравнения для $m = 1$, $c = 0.5$ представлено на следующем графике (начальные условия $x(0) = 0$, $\dot{x}(0) = 1$):



На этом краткое описание реализации некоторых алгоритмов с помощью средств VBA мы завершаем.

Литература

1. *Буч Г.* Объектно-ориентированное проектирование с примерами применения. Киев; М., 1992.
2. *Марченко А. И.* Программирование в среде Borland Pascal 7.0. Киев, 1996.
3. *Страуструп Б.* Язык программирования C++. Специальное издание: Пер. с англ. М., 2004.
4. Практическое руководство по программированию. /Под ред. Б. Мика, П. Хит, Н. Рашби. М., 1986.
5. *Додж М., Стинсон К.* Эффективная работа с Microsoft Excel 2000. СПб., 2000.
6. *Хэлворсон М., Янг М.* Эффективная работа: Office XP. СПб., 2003.
7. *Кузьменко В. Г.* VBA 2002., М. 2002.
8. *Боревич З. И.* Определители и матрицы. СПб., 2004.
9. *Романовский И. В.* Дискретный анализ. СПб., 2003.
10. *Самарский А. А, Гулин А. В.* Численные методы. М., 1989.

Содержание

Учебное издание

Никита Николаевич Дмитриев
Вадим Юрьевич Сахаров

ВВЕДЕНИЕ В MICROSOFS ACCESS
Применение VBA для решения простейших задач

Учебное пособие

Зав. редакцией *Г. И. Чередниченко*
Редактор *Ф. С. Бастиан*
Обложка *А. В. Калининой*

Подписано в печать с оригинала-макета 09.06.2006.
Ф-т 60 × 84$^{1}/_{16}$. Усл. печ. л. 4,18. Уч.-изд. л. 4,38. Тираж 150 экз. Заказ № 532.

РОПИ СПбГУ.
199034, С.-Петербург, Университетская наб., 7/9.

Типография Издательства СПбГУ.
199061, С.-Петербург, Средний пр., 41.