Глава 2. Программирование на языке VBA в Microsoft Office

2.1. Общие сведения о языках программирования

Различные языки программирования от самых примитивных языков до языков, близких к естественному языку человека, позволяют создавать совершенные и эффективные программы с продолжительным жизненным циклом. Постоянное повышение сложности задач, решаемых с помощью ЭВМ, а также совершенствование самих ЭВМ требует разработки новых специализированных, ориентированных на проблемную область и более мощных языков программирования.

В развитии инструментального программного обеспечения рассматривают пять поколений языков программирования. Первые три поколения характеризовались более сложным набором зарезервированных слов и синтаксисом. Языки четвертого поколения все еще требуют соблюдения определенного синтаксиса при написании программ, но он значительно легче для освоения. Разрабатываемые в настоящее время естественные языки программирования составляют пятое поколение. В табл. 2.1 приводится общая характеристика языков программирования [5].

Таблица 2.1

Поколения языков	Языки программирования	Характеристика
Первое	Машинные	Ориентированы на использование в конкретной ЭВМ, сложны в освоении, требуют хорошего знания архитектуры ЭВМ
Второе	Ассемблеры, макроассемблеры	Более удобны для использования, но машинно-зависимы
Третье	Языки высокого уровня	Мобильные, человеко-ориентированные, проще в освоении
Четвертое	Непроцедурные, объектно-ориентированные, языки запросов, параллельные	Ориентированы на непрофессионального пользователя и на ЭВМ с параллельной архитектурой
Пятое	Языки искусственного интеллекта, экспертных систем и баз знаний, естественные языки	Ориентированы на повышение интеллектуального уровня ЭВМ и интерфейса с языками

Краткий обзор современных языков программирования можно посмотреть в [6].

В [7] проводится краткая характеристика алгоритмического, структурного, событийно-ориентированного и объектно-ориентированного программирования.

Алгоритмические языки программирования представляют алгоритм в виде последовательности основных алгоритмических структур:

– линейных;

– разветвляющихся;

– циклических.

Алгоритм на выбранном языке программирования описывается с помощью команд описания данных, вычисления значений и управления последовательностью выполнения программы.

Основными элементами алгоритмического языка программирования являются:

– переменные и константы;

– арифметические операции, используемые для создания арифметических выражений;

– логические операции и выражения;

– указатели. В некоторых языках программирования возможно явное указание адреса физической памяти как в Ассемблере;

– сложные данные (списки, деревья, …);

– массивы;

– описания переменных;

– операторы присвоения;

– комментарии;

– условные операторы;

– операторы цикла;

– операторы ввода-вывода.

Структурное программирование предполагает дополнительное использование подпрограмм, процедур и функций. Наличие подпрограмм позволяет вести проектирование и разработку приложений сверху–вниз – такой подход получил название – нисходящее проектирование. Сначала общая задача разбивается на глобальные подзадачи (модули). Затем каждый модуль разбивается на подпрограммы, которые, в свою очередь, могут содержать другие подпро-

граммы. Небольшие подпрограммы значительно проще отлаживать, а это повышает общую надежность всей программы. Кроме того, подпрограммы могут повторно использоваться, а это повышает производительность труда программистов.

Развитием идеи нисходящего проектирования стало *событийно-ориентированное* программирование. Оно возникло в связи с широким распространением визуальных систем типа Windows, идеология которых основана на событиях. Программы при *событийно-ориентированном* программировании имеют следующую структуру: главная часть программы – один бесконечный цикл, который опрашивает Windows, следя за появлением новых сообщений. При обнаружении сообщения вызывается подпрограмма, отвечающая за обработку соответствующего события, а цикл опроса продолжается до получения сообщения «конец работы».

Применение структурного и событийно-ориентированного программирования существенно повысило производительность труда программистов. Но это уже был предел возможностей человека.

Как развитие технологии структурного программирования в середине 1980-х гг. возникло понятие *объекта* и *объектно-ориентированного программирования* (ООП). ООП предполагает создание приложений из объектов, которые можно позаимствовать в готовом виде из разнообразных библиотек, либо при необходимости разрабатывать самостоятельно.

Важное место в технологии ООП занимает *событие* (щелчок клавиши мыши на объекте, нажатие определенной клавиши, открытие документа и т.д.). В качестве реакции на событие может вызываться определенная *процедура*, которая может изменять *свойства* объекта, вызывать его *методы*.

В настоящее время в ООП обычно используется графический интерфейс, позволяющий визуализировать процесс программирования. Появилась возможность создавать объекты, задавать им свойства с помощью мыши.

В среде Microsoft Office в качестве объектов выступают приложения, документы, окна и т.п. Каждый из этих объектов является исполнителем алгоритма, который задается с помощью команд на соответствующем формальном языке, либо другими объектами, функционирующими в данной системе, либо пользователем персонального компьютера. Запись алгоритма для объектов в среде Microsoft Office осуществляется с помощью языка программирования Visual Basic for Application (VBA), который является ядром ООП языка программирования Visual Basic.

Системы программирования содержат разнообразные средства создания программ:

– текстовый редактор;

– компилятор;

– редактор связей;

– библиотека функций;

– отладчик программы.

Из наиболее популярных, использовавшихся в течение длительных периодов времени, можно назвать следующие универсальные языки программирования: Basic, Pascal, C ++.

Для каждого из этих языков сегодня имеется немало систем программирования, ориентированных на разные персональные компьютеры и операционные системы разных поколений**.**

Наиболее популярные системы проектирования программ для непрофессиональных программистов Windows – это Microsoft Visual Basic, Borland Delphi, Borland C++ Bulider и т.д.

2.2. Объектная структура языка VBA

В качестве основного понятия объектно-ориентированного метода программирования используется понятие *объекта*. Данные и программный код, обслуживающий их, заключаются в единый объект. Для отображения этих данных необходимо послать объекту соответствующее сообщение. При внедрении объекта в приложение должно быть указано место его отображения на экране.

Любой *объект* обладает соответствующим набором свойств (характеристиками данного объекта) и набором *методов* (команд для обработки свойств) [8].

Объекты, объединяющие (инкапсулирующие) одинаковый перечень свойств и методов, образуют *классы*. При этом каждый отдельный объект является *экземпляром класса*. Однако экземпляры класса могут иметь отличные *значения* свойств.

Объекты VBA в Microsoft Office – это кнопки, элементы меню, документы, фрагменты документа, символы, интервалы ячеек рабочего листа и даже сам рабочий лист и т.п.

В табл. 2.2 приведены примеры классов объектов VBA для приложений Microsoft Office Word и Excel.

Объекты в приложениях образуют некоторую *иерархию*, на вершине которой находится приложение (*Application*).

Иерархия четырех уровней объектов в приложениях Microsoft Office Word и Excel приведена в табл. 2.3.

Таблица 2.2

Приложение	Класс объектов	Свойства	Методы
Word	*Documents* (документ)	*Name* (имя) *FileName* (полное имя файла) *Add* (создает новый объект в семействе) …	*Open* (открыть) *Close* (закрыть) *Save* (сохранить) …
Excel	*Workbooks* (книга)	*ActiveSheet* (активный лист книги) *Name* (имя) *Path* (полное имя папки, где находится книга) *Add* (создает новый объект в семействе) …	*Open* (открыть) *Close* (закрыть) *Save* (сохранить) …

Таблица 2.3

Word	Excel
Активное приложение (*Application*)	Активное приложение (*Application*)
Документ (*Documents*)	Книга (*Workbook*)
Фрагмент документа (*Selection*)	Лист (*Worksheet*)
Символ (*Character*)	Диапазон ячеек (*Range*)

Чтобы получить *доступ к объекту* в языке VBA, необходимо составить *ссылку* на него. Ссылка на требуемый объект содержит перечень всех объектов по иерархии, разделенных точками. Например, в программе Word ссылка на документ ЛР1.docx выглядит следующим образом:

Application.Documents (”ЛР1.docx”)

Однако, если приложение Word активно, достаточно сделать относительную ссылку на сам документ:

Documents (”ЛР1.docx”)

или, используя общее обращение к активному в данный момент документу без указания его имени:

ActiveDocument

В приложении Excel общее обращение к активной в данный момент рабочей книге выглядит следующим образом:

ActiveWorkbook

Для того чтобы объект выполнил какую-либо *операцию*, необходимо задать *метод*. Многие методы имеют *аргументы*, задающие параметры выполняемых действий.

Синтаксис команды применения метода объекта:

Объект.Метод арг1:=значение, арг2:=значение, ...

Например, операция открытия в приложении Word документа Проба.docx, находящегося на диске H: в папке ФИО, содержит не только название метода *Open*, но и указание пути к открываемому файлу, как значения аргумента *FileName*:

Documents.Open FileName:=("H:\ФИО\Проба.docx")

Для открытия нового документа в Word используется метод *Add* с указанием значения аргумента *DocumentType***:**

Documents.Add DocumentType:=wdNewBlankDocument

Для вывода на печать двух первых страниц документа Проба.docx необходимо задать для метода *PrintOut* значения аргументов *Range* (задает формат диапазона печати), *From* и *To* (задают номер начальной и конечной страниц печати):

Documents("Проба.docx").PrintOut _

Range:=wdPrintFromTo, From:="1", To:="2"

Обратите внимание, что в последнем выражении программная строка кода не поместилась в одной экранной строке, поэтому она перенесена на следующую экранную строку с помощью символа подчеркивания (_) с предшествующим символом пробела.

Для сохранения документа Проба.doc на диске в директории по умолчанию используется метод *Save*:

Documents("Проба.docx").Save

Для изменения *состояния* объекта необходимо задать новые значения его *свойств*.

Синтаксис команды изменения свойств:

Объект.Свойство = ЗначениеСвойства

Например, для установки во фрагменте текста (объект *Selection*) для восьмого символа (объект *Characters(8)*) начертания – полужирный (свойство *Bold*, которое имеет два значения *True* или *False*), введем следующий код:

Selection.Characters(8).Bold = True

Объект может иметь множество свойств. Для сокращения программного кода можно задавать значения сразу нескольким свойствам объекта с помощью инструкции With, не указывая явно его имя:

```
With Объект
    .Свойство1 = ЗначениеСвойства1
    .Свойство2 = ЗначениеСвойства2
        . . .
End With
```

2.3. Основные этапы создания VBA-программ

Процесс создания VBA-программ включает в себя пять этапов.

На первом этапе разработки VBA-программ необходимо проанализировать все требования, которым должна удовлетворять разрабатываемая программа, и прежде всего – что должна делать программа и что представляют собой входные данные, какими ресурсами располагает проектировщик и какими должны быть выходные данные.

На втором этапе проектируется общая структура программы, т.е. определяются те объекты, из которых будет состоять программа (формы, диалоговые окна, элементы управления и т.д.), а также определяются общие принципы управления и взаимодействия между различными компонентами программы. Уже на этом этапе необходимо продумать интерфейс будущей программы.

На третьем этапе осуществляется реализация проекта на языке программирования VBA: проектируется внешний вид форм и связанных с ней элементов управления, задаются их свойства и пишутся процедуры, отвечающие за обработку связанных с ними событий.

На четвертом этапе производится проверка работы VBA-программ. При этом оценивается правильность, вычислительная сложность, а также эффективность ее реализации.

Последний этап – это сопровождение. На этапе эксплуатации программы устраняются обнаруженные при эксплуатации ошибки и находятся «узкие места» или неудачные проектные решения.

2.4. Макросы в Microsoft Office. Редактор Visual Basic

Макросы являются удобным средством для автоматизации выполнения часто повторяемых действий, что позволяет повысить

точность и скорость работы в программах Office. Язык VBA был разработан Microsoft Office для унификации макроязыков в своих приложениях под Windows.

При создании макроса как в Word, так и в Excel макрорекодер записывает все действия пользователя в виде программного кода на языке VBA. При этом макросы сохраняются в файлах документов Word и в файлах рабочих книг Excel в специальной части файла, называемой модулем (Module). Модули, сохраняемые в одном документе Word, называют проектами Project, а в одной рабочей книге Excel называют проектами VBAProject.

Если при создании макроса были допущены ошибочные действия, то его можно отредактировать. Для повышения гибкости, расширения возможностей и оптимизации макросов при их редактировании может добавляться требуемый программный код на языке VBA. Кроме того, VBA позволяет организовать взаимодействие нескольких макросов между собой.

Внимание! Последующие разделы учебного пособия целесообразно изучать с одновременным выполнением соответствующих заданий на компьютере в изучаемой программной среде.

2.4.1. Создание макросов в Microsoft Word

Создадим первый макрос в текущем документе Word (версия 2010) «*Глава2.docx*» под именем *Макрос1*, который выполняет следующие действия:

1) устанавливает с помощью кнопок на панели инструментов начертание шрифта *Курсив* и цвет шрифта *Красный*;

2) выводит в строку, где находится курсор, текст «Я учусь писать макрос».

Для записи макроса в Word необходимо выбрать в разделе меню *Вид* команду *Макросы⇒Запись макроса*. В появившемся окне (рис. 2.1) будет указано имя макроса по умолчанию *Макрос1*, которое при необходимости можно изменить.

Так как в нашем случае макрос должен быть доступен только в текущем файле Word, то в списке *Макрос доступен для*: (см. рис. 2.1) требуется выбрать имя текущего документа. При сохранении макроса в шаблоне *Normal.dotm* он будет доступен для всех документов, созданных на его основе.

После нажатия на клавишу *ОК* необходимо выполнить все действия, описанные в пп. 1–2 данного раздела, а затем выбрать на панели *Макросы* команду *Остановить запись* (рис. 2.2).

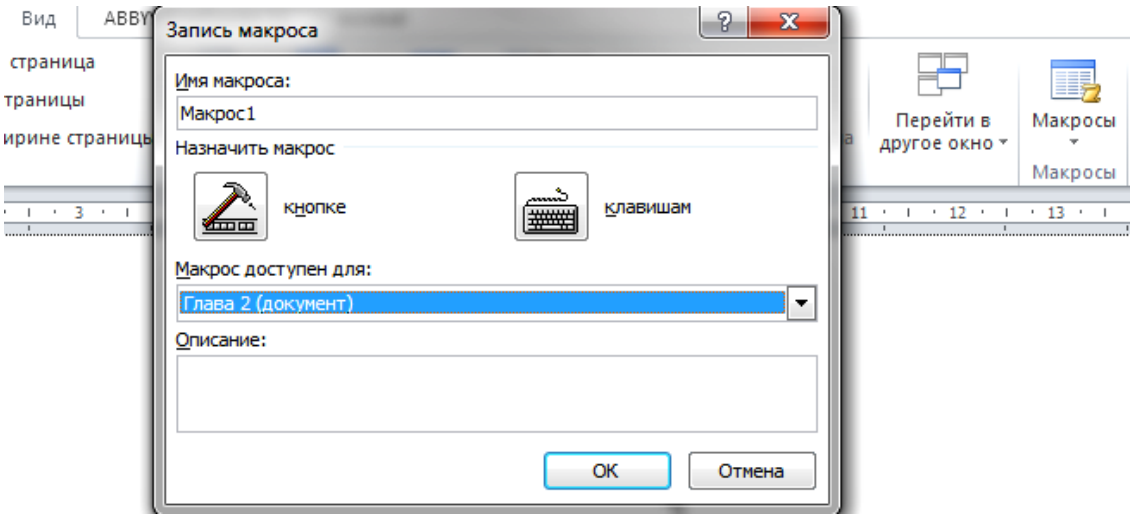


Рис. 2.1

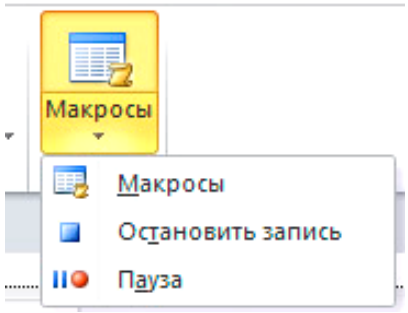


Рис. 2.2

После завершения записи макроса в документе Word добавится модуль под названием *NewMacros*, в котором помещается сгенерированный код на языке VBA.

Для запуска созданного макроса в Word переведем курсор в начало следующей строки и выберем в разделе меню *Вид* команду *Макросы⇒Макрос1⇒Выполнить*.

Создадим второй макрос в текущем документе Word под именем *Макрос2*, который в отличие от первого макроса устанавливает начертание шрифта *Курсив* и цвет шрифта – *Красный* с помощью диалогового окна *Шрифт*. Программный код второго макроса добавится в тот же модуль *NewMacros* текущего документа.

2.4.2. Интегрированная среда проектирования VBA

Для просмотра сгенерированного кода требуется выбрать команду *Макросы⇒Макрос1⇒Изменить* или нажать комбинацию клавиш *Alt+F11*. В результате откроется окно редактора Visual Basic в интерпретации Microsoft (рис. 2.3), которое называется *Интегрированной средой проектирования* (IDE).

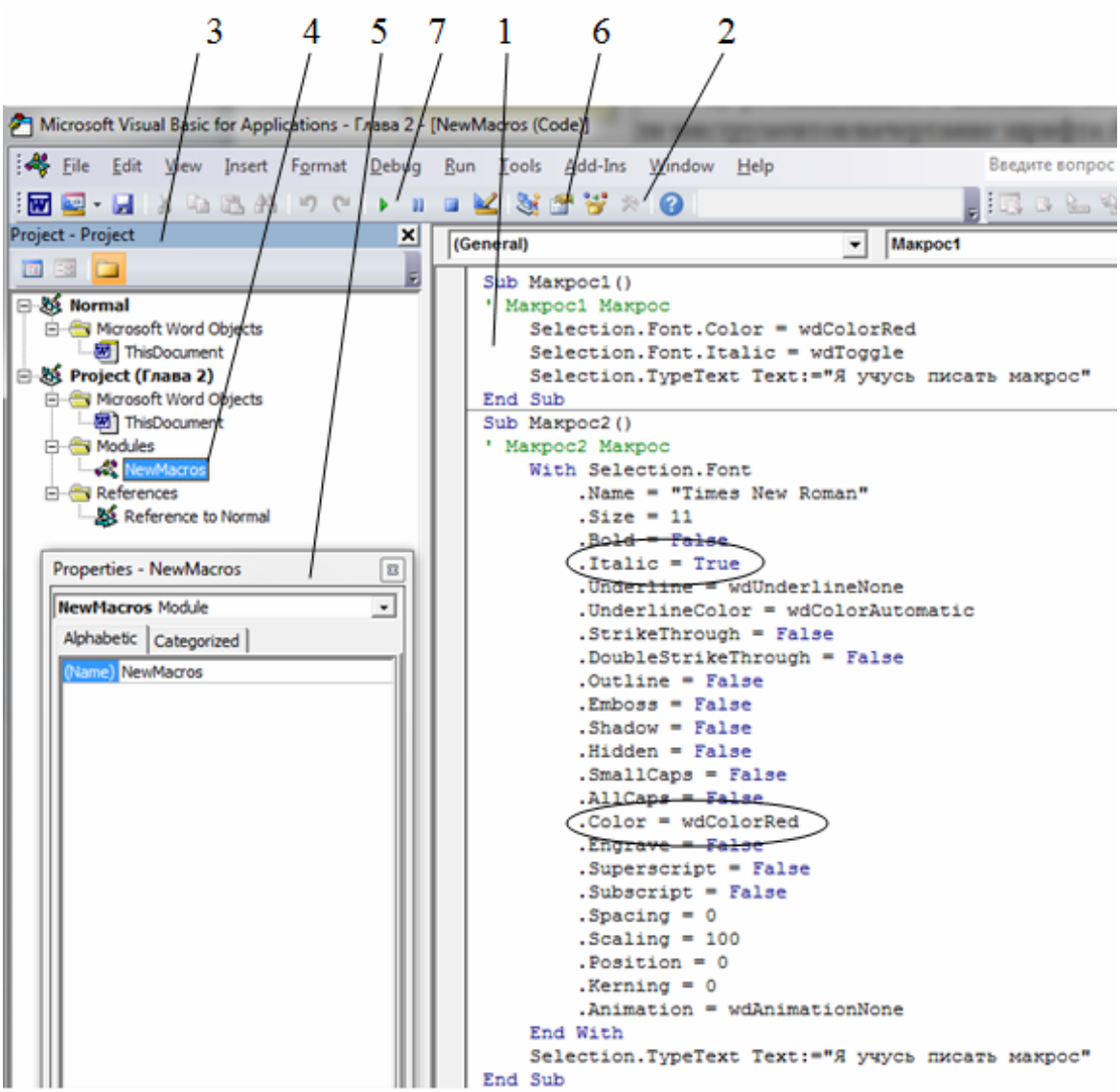


Рис. 2.3

IDE позволяет писать и редактировать программы, проектировать диалоговые окна с различными элементами управления, облегчающими взаимодействие с пользователем, запускать программы, тестировать и выполнять их отладку, контролировать взаимодействие различных VBA-программ.

Самое большое окно в IDE – окно редактирования кода (1), в котором можно посмотреть доступный для редактирования сгенерированный код макроса.

Над окном редактирования кода располагается стандартная панель инструментов (2), которую можно отобразить на экране выбором команды *View⇒Toolbars⇒Standard*.

На рис. 2.3 в окне проектов *Project* (3) содержатся два проекта: первый соответствует общему шаблону *Normal*, второй соответствует открытому документу *Глава 2*.

Каждый проект содержит модуль документа с именем *This-Document*.

Проект *Project* (*Глава 2*) дополнительно содержит модуль *NewMacros* (4), в котором хранятся созданные нами макросы, а также компонент *References*, который содержит ссылку на присоединенный к проекту шаблон (по умолчанию – это шаблон *Normal*).

Окно свойств *Properties* (5) предназначено для установки свойств форм и элементов управления. Окно отображается на экране нажатием кнопки (6)  на стандартной панели инструментов. Список свойств в окне разделен на две колонки.

В левой колонке окна свойств находятся имена свойств, а в правой – значения этих свойств. Установленные по умолчанию значения могут быть изменены. Для некоторых свойств объектов предусмотрена возможность выбора их значений из раскрывающегося списка. Ряд свойств объектов можно задавать и программно.

Для запуска на исполнение созданного макроса из среды IDE VBA необходимо нажать клавишу *F5* или кнопку *Run* (7) на стандартной панели инструментов. При этом в окне проектов должен быть выделен модуль *NewMacros*.

Программный код макроса при необходимости может быть отредактирован в окне редактирования. При этом применяются те же правила редактирования, что и в самом текстовом редакторе Word.

При написании программного кода его целесообразно снабжать *комментариями* – это неисполняемые строки, начинающиеся со знака апострофа (‘).

Кроме того, во время отладки программы апостроф часто используют для временного отключения некоторых программных строк. Апостроф можно ставить вручную, а если надо добавить или удалить сразу несколько комментариев, то удобнее воспользоваться кнопками *CommenBlock* (закомментировать блок)  и *Uncommen-Block* (раскомментировать блок)  панели инструментов *Edit*, которую выбирают в разделе меню *View⇒Toolbars*.

При написании кода в окне для редактирования редактор автоматически предлагает пользователю список компонентов, логически завершающих вводимую им инструкцию.

Например, при наборе кода для объекта *Selection* (определяет выделенный участок текста в Word, или если ничего не выделено, то место, где находится указатель вставки) после ввода точки на экране отобразится список компонентов (свойств или методов данного объекта), которые завершают данную конструкцию. Двойной щелчок на выбранном компоненте вставляет его имя в код программы.

2.4.3. Анализ полученного программного кода макроса

Сопоставим действия, которые были выполнены для написания первого макроса и полученный программный код из окна редактирования (см. рис. 2.3).

Первая и шестая строки *Макроса1* являются началом (*Sub*) и окончанием (*End Sub*) процедуры, соответственно. Вторая строка – это комментарий. В третьей строке устанавливается красный цвет шрифта – свойству *Color* присваивается значение константы *wdColorRed.* В четвертой строке задается начертание шрифта курсив – свойству *Italic* присваивается значение константы *wdToggle*, которая обозначает переключение текущей установки. При установке можно также использовать константы **True** (истина) и **False** (ложь). Пятая строка выводит в документе в строку, где находится курсор (объект *Selection*), текст «*Я учусь писать макрос*».

Сравнив текст первого и второго макросов, можно сделать вывод, что вместо третьей и четвертой строк добавилось 25 строк, в которых с помощью инструкции *With* задается 23 свойства объекта *Selection.Font* из диалогового окна *Шрифт*. Среди перечисленных свойств есть и свойства *Italic* и *Color* (на рис. 2.3 обведены овалами), значения которых менялись. Хотя все остальные настройки шрифта, включенные в текст *Макроса2*, не менялись, но в код макроса были включены все свойства шрифта. Значение этих свойств берется по умолчанию из диалогового окна *Шрифт*.

2.4.4. Сохранение макроса на диске

Для сохранения одного выбранного модуля в текущем файле из среды IDE достаточно нажать клавиши *Ctrl+S* или щелкнуть по кнопке  стандартной панели инструментов, или выбрать команду *File ⇒Save*.

Для сохранения текста программы макроса в отдельном файле с расширением *.bas* необходимо нажать клавиши *Ctrl+E* или выполнить команду *File⇒ExportFile*. А чтобы загрузить в IDE VBA текст программы из файла с расширением **.***bas*, необходимо нажать клавиши Ctrl+M или выполнить команду *File⇒ImportFile*.

Чтобы сохранить сразу все изменения в текущем проекте, включая все связанные с ним модули, необходимо вернуться в активное приложение Word и выбрать команду *Сохранить как:*.

Следует иметь в виду, что при сохранении на диске документа вместе с макросом в окне сохранения документа необходимо выбрать тип документа *Документ Word с поддержкой макросов*. При этом файл будет сохранен с расширением *.docm*.

Сохранить файл с макросами с расширением ***.docm***. Файл назвать по своей фамилии.

2.4.5. Открытие файлов с макросами

При открытии файлов с макросами появляется предупреждение системы безопасности (рис. 2.4). Чтобы иметь возможность работать с макросами в открываемом файле, необходимо нажать на клавишу *Включить содержимое*. Иначе макросы в данном файле будут не доступны.

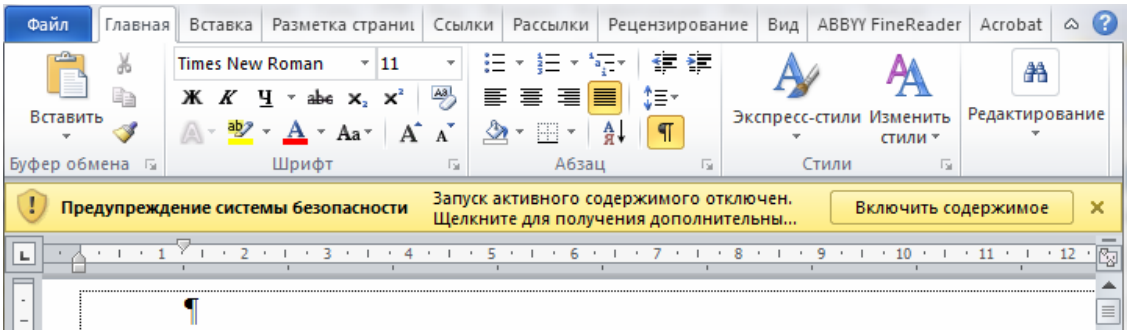


Рис. 2.4

Для настройки параметров макросов в центре управления безопасностью в Microsoft Word необходимо выполнить команду *Файл⇒Параметры⇒Центр управления безопасностью*. В результате откроется окно (рис. 2.5), в котором необходимо выбрать требуемые параметры и нажать клавишу ОК.

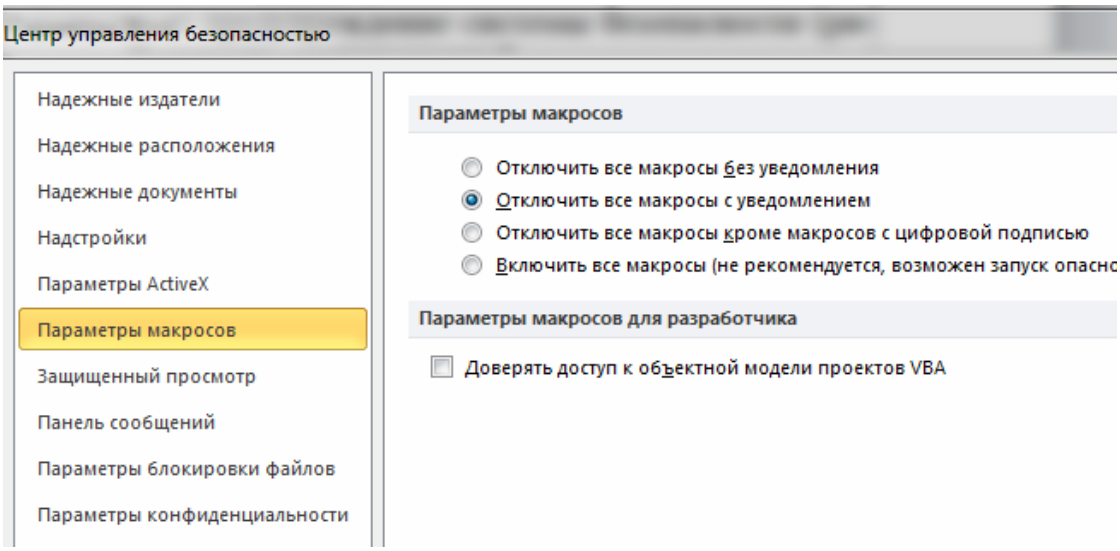


Рис. 2.5

2.4.6. Создание макросов в Excel

Создадим первый макрос в программе Microsoft Excel (версия 2010) в файле *Книга1.xlsm* под именем *Макрос1*, который вычисляет содержимое текущей ячейки как сумму двух ячеек, расположенных левее текущей ячейки и устанавливает денежный формат ячейки с точностью два знака после запятой (рис. 2.6).

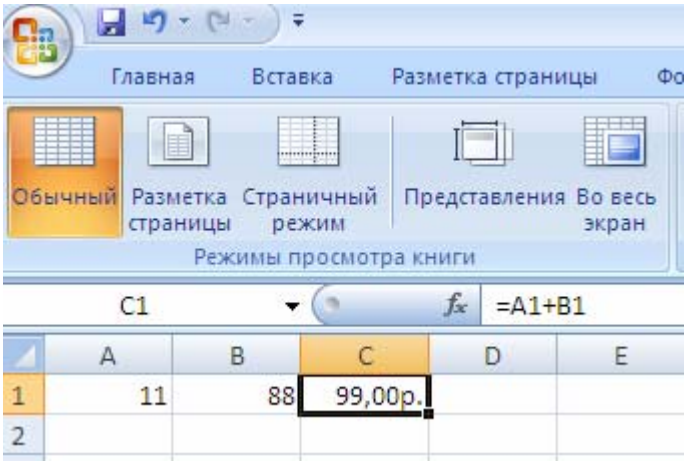


Рис. 2.6

Для записи макроса в Microsoft Excel необходимо выбрать в разделе меню *Вид* команду *Макросы⇒Запись макроса*. В появившемся окне (рис. 2.7) будет указано имя макроса по умолчанию *Макрос1*, которое при необходимости можно изменить.

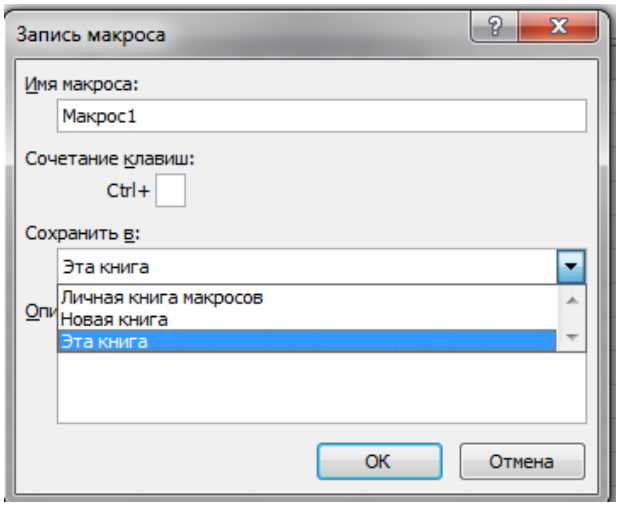


Рис. 2.7

Если макрос должен быть сохранен только в текущем файле Excel, то в списке *Сохранить в:* (см. рис. 2.7) требуется выбрать *Эта книга*. При сохранении макроса с настройкой *Личная книга макросов* он будет доступен для всех книг, так как запишется в спе-

циальный скрытый файл *Personal.xlsb*, который загружается при каждом запуске Excel. При выборе в списке строки *Новая книга* макрос запишется во вновь открытую книгу Excel.

Запуск макроса и просмотр его кода выполняется аналогично, как и в Word.

Созданный макрос помещается в модуле *Modules1* активного проекта *VBAProject (Книга1.xlsm)* (рис. 2.8).

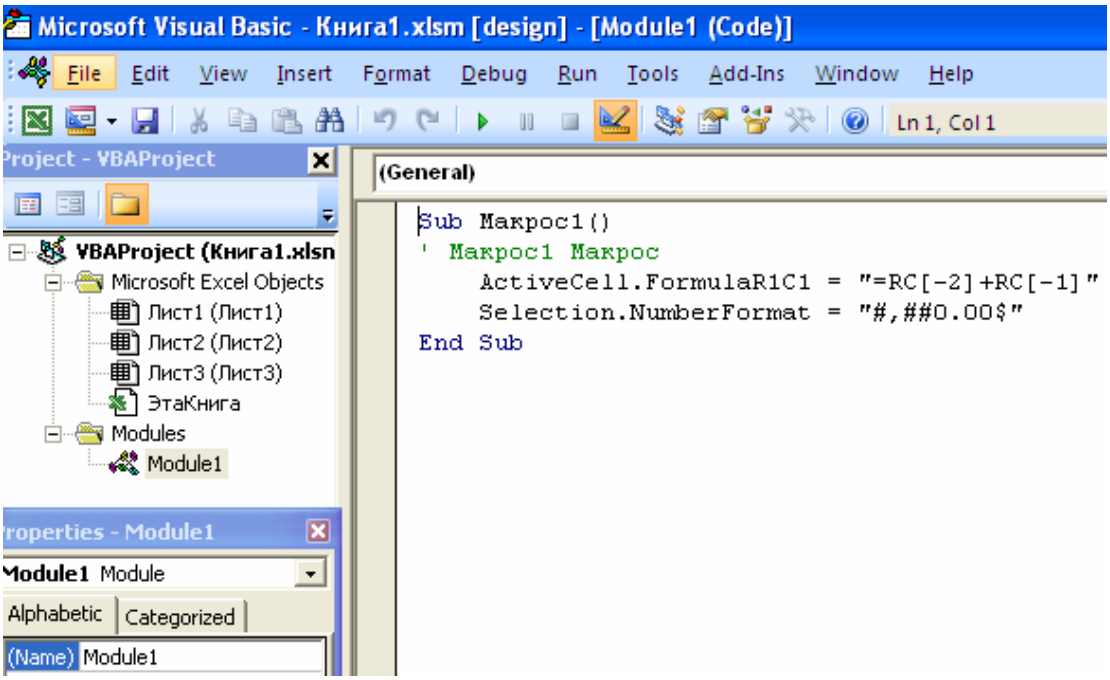


Рис. 2.8

Сопоставим действия, которые были выполнены для написания макроса, и полученный программный код макроса из окна редактирования (см. рис. 2.8).

В третьей строке формируется содержимое активной ячейки таблицы по формуле. При этом используется не прямая ссылка на ячейку, а относительная адресация через приращения по отношению к текущей строке (*Row*) и текущему столбцу (*Colomn*).

Символьная константа *"=RC[-2]+RC[-1]"*, задающая значение свойства *FormulaR1C1* активной ячейки *ActiveCell*, обозначает, что содержимое активной ячейки будет рассчитано как сумма содержимого ячеек в той же строке, с номерами столбцов на две и на одну единицу меньше номера столбца активной ячейки соответственно.

В четвертой строке устанавливается денежный формат активной ячейки с точностью два знака после запятой.

Сохранение листинга программы в отдельном файле, а также всего проекта выполняется аналогично, как и в разд. 2.4.4. Сохранить созданный макрос в текущем файле Excel.

2.4.7. Организация проекта в среде VBA

Программа на языке VBA состоит из одного или нескольких модулей. В VBA проект, помимо макросов, которые хранятся в соответствующем модуле, может состоять из нескольких форм, связанных с соответствующими программными модулями. Кроме того, в проекте могут присутствовать стандартные модули и модули классов.

Обычно модуль начинается с опций, которые управляют описанием переменных.

Инструкция *Option Explicit* используется для принудительного объявления всех переменных в области *General Declarations* (Общее описание) модуля, которая располагается в верхней части модуля.

Эта инструкция исключает возможность случайного создания новых переменных. Ее использование полезно при отладке программы, так как если в программе допущена ошибка в имени переменной, то будет выдано соответствующее сообщение об ошибке.

Затем в модуле располагается область объявления переменных и констант уровня модуля или проекта, которые могут быть использованы во всех процедурах модуля либо проекта, например:

Dim bytA As Byte
Dim strA1 As String
Private i As Integer
Private Среднее As Double

Далее располагается код процедур или функций, составляющий саму программу.

Процедура представляет собой поименованную часть кода, выполняющую определенные действия. Процедура может иметь параметры, которые в результате выполнения последовательности инструкций могут менять свои значения.

При общем описании формата (синтаксиса) команд принято использовать следующие обозначения:

[ ] (квадратные скобки) обозначают, что заключенная в них часть конструкции является необязательной, но если она есть, то присутствует без самих квадратных скобок;

| (вертикальная черта) разделяет возможные варианты выполнения команды.

Процедура имеет следующий синтаксис:

[PublicPrivate |Friend][Static] Sub name[(arglist)]
[statements]

[Exit Sub]

[statements]

End Sub

где *Public* – указывает, что процедура доступна для всех других процедур во всех модулях;

Private – указывает, что процедура доступна для других процедур только того модуля, в котором она объявлена;

Friend – используется только в модулях классов. Позволяет вызывать процедуру из другого модуля проекта;

Static – указывает, что локальные переменные процедуры сохраняются в промежутках времени между вызовами этой процедуры;

arglist – список аргументов, разделенных запятой, значения которых передаются в процедуру или возвращаются из процедуры при ее вызове;

statements – любая группа инструкций, выполняемых в процедуре Sub;

Exit Sub – инструкция, которая приводит к немедленному выходу из процедуры Sub.

Программные модули могут включать в себя процедуры двух типов: *общие* и *событийные*.

Общая процедура представляет собой подпрограмму, которая начинает выполняться после ее вызова из другой процедуры. Выполнение общих процедур не связывается с какими-либо событиями, они вызываются на выполнение с помощью оператора *Call* и имеют следующий формат:

Call ИмяПроцедуры (Список параметров)

где *Список параметров* представляет собой набор переменных, значения которых устанавливаются до начала выполнения процедуры.

Событийная процедура представляет собой подпрограмму, которая начинает выполняться после реализации определенного события. В рассмотренной в следующем разделе программе Калькулятор содержатся лишь событийные процедуры, большинство из которых обрабатывает событие (*Click*), произошедшее при щелчке левой клавиши мыши на соответствующем элементе управления.

Кроме процедуры *Sub* в VBA используется и процедура *Function*, которая имеет следующий формат:

[Public │Private │Friend][Static] Function name[(arglist)]

statements
[name=expression]
[Exit Function]
statements
[name=expression]
End Function

где *name* – имя процедуры *Function*;

expression – любое выражение, возвращающее значение того же типа, что и процедура *Function*.

Инструкция *Exit Function* приводит к немедленному выходу из процедуры *Function*.

Если требуется использовать возвращаемое *Function* значение, в отличие от процедуры *Sub* процедура *Function* может применяться в правой части выражения, как и любая другая встроенная функция

Переменная = name()

В свою очередь, процедуре *Function* может присваиваться значение какого-либо выражения

name() = expression

Для вызова процедуры *Function* может использоваться так же, как и для процедуры *Sub* инструкция *Call*, когда интересует не возвращаемое процедурой значение, а осуществляемое ей действие.

В VBA возможно создание рекурсивных процедур *Function* и *Sub*, т.е. процедур вызывающих самих себя.

2.5. Основы программирования на VBA

2.5.1. Основные элементы языка программирования VBA

Программа, написанная в среде VBA, представляет собой совокупность программных строк, имеющих следующий формат:

[номер строки] оператор [:оператор] ... ['комментарий]

или

метка:

или

$метаоператор

Операторы представляют собой действия, выполняемые программой. В строке может быть *один* или *несколько* операторов, разделенных двоеточием. Длина программной строки в VBA может

быть до 249 символов. Если оператор не помещается в одной экранной строке, то его можно разбить на несколько экранных строк, поставив в конце строки символ подчеркивания (_). При этом следующая строка будет рассматриваться как продолжение предыдущей. Следует иметь в виду, что перед символом подчеркивания должен стоять пробел, если только строка не разрывается внутри слова.

Номер строки – это целое десятичное число от 0 до 65535. Номера необязательно использовать в программе, но если они есть, то используются для идентификации (распознавания) строк программы. Никаких ограничений, за исключением того, что все строки должны иметь разные номера, на взаимное расположение номеров строк не накладывается.

Метка должна быть в строке без операторов (хотя комментарий в этой строке может быть). Она служит для идентификации оператора, следующего непосредственно за ней. Метка должна начинаться с буквы и может содержать произвольное число букв и цифр. Заглавные и строчные буквы в метке не различаются. За меткой должно следовать двоеточие.

Основными элементами языка VBA являются символы, с помощью которых записываются константы, переменные, массивы, выражения, встроенные функции и операторы.

При написании программы на VBA можно использовать символы из определенного набора, называемого алфавитом языка. Он включает в себя заглавные и строчные латинские буквы от A до Z, цифры от 0 до 9, пробел, а также специальные символы: = (знак равенства), + (плюс), – (минус), * (звездочка, знак умножения), / (слэш, знак деления), \ (знак целочисленного деления), ^ (крышка, знак возведения в степень), % (знак процента), & (амперсанд), ! (восклицательный знак), # (знак номера), $ (знак доллара), ( ) (круглые скобки), [ ] (квадратные скобки), , (запятая), . (точка), ‘ (апостроф), ; (точка с запятой), : (двоеточие), ? (вопросительный знак), > (знак больше), < (знак меньше), “ (кавычки), _ (подчеркивание). Кроме того, в комментариях, символьных константах и именах переменных могут использоваться заглавные и строчные буквы кириллицы (русские буквы).

2.5.2. Типы данных, константы и переменные

Программы на VBA используют различные типы данных, которые определяют область возможных значений данного типа, структуру организации данных и разрешенные операции для данных этого типа.

VBA разделяет обрабатываемые данные на числа, даты, строки, логические значения и объекты. Подробный список типов данных в VBA можно посмотреть, например, в табл. 4.1 самоучителя [9]. В данном пособии рассмотрены наиболее распространенные типы данных, используемых в VBA.

Среди *числовых* данных выделим следующие типы:

– *целое Integer* (число без десятичной точки в диапазоне от –32768 до 32767);

– *длинное целое Long* (в диапазоне от –2 147 483 648 до 2 147 483 647);

– *вещественное Single* (число с плавающей точкой одинарной точности):

для отрицательных чисел в диапазоне от $-3{,}402823\ 10^{38}$ до $-1{,}401298\ 10^{-45}$;

для положительных чисел в диапазоне от $1{,}401298\ 10^{-45}$ до $3{,}402823\ 10^{38}$.

Внимание. В VBA целая часть числа отделяется от дробной точкой!

Любые *строковые (текстовые) данные* в языке VBA используют тип данных *String*. Они всегда заключаются в кавычки (“ “).

По умолчанию все строковые данные относятся к строкам переменной длины. Для объявления этого типа используют оператор *Dim*:

Dim имя_переменной As String

Если в программе требуется использовать строки фиксированной длины (от 1 до 65 400 символов), то необходимо использовать оператор *Dim* следующей конструкции:

*Dim имя_переменной As String * длина_строки*

Если в VBA переменная была создана без указания типа, то ей назначается тип *Variant***.** Однако при этом переменная будет занимать много места в памяти компьютера, и программа будет выполняться дольше.

Константы – это элементы языка VBA, значения которых определены и неизменны. Они бывают *числовыми* и *символьными***.**

Числовые константы, в свою очередь, могут быть целыми и вещественными. Они используют цифры, знак минус и необязательный знак плюс. *Целые* числа, например число *257*, допускается записывать в двоичном (&B100000001), восьмеричном (*&*401) и в шестнадцатеричном (*&*H101) представлении.

Вещественные константы можно задавать как в десятичном формате (*0.000235988*), так и в формате с плавающей точкой (*235.988E-6*).

Символьные константы в VBA могут быть *литеральные* и *символические***.** Значение литеральной символьной константы задается прямо в тексте программы в кавычках (“*Пример*“). Изменить его можно только при редактировании программы.

Символические константы, так же как и переменные, должны быть объявлены, прежде чем их использовать в любом месте программы. Тип данных константы при ее объявлении указывать не обязательно:

Dim НомерЗадачи As String
Dim Задача As String
Const НомерЗадачи = 2
Const Задача = “Решение системы уравнений”

Константы, объявленные в процедуре, могут использоваться только в ней. При определении константы в области определений модуля она будет доступна всем процедурам данного модуля.

Переменные – это элементы языка для хранения данных указанного типа в поименованной области компьютера. Значения переменных заранее не определены, но в процессе выполнения программы им можно присваивать значения констант, других переменных либо выражений. Переменные разделяются на *числовые* переменные и *символьные* (строковые). Числовой переменной можно присвоить значение любого числа, а символьной – значение строки символов.

Имя переменной должно быть уникальным и начинаться с буквы, хотя может содержать и цифры. Заглавные и строчные буквы не различаются. Длина имени переменной не должна превышать 255 символов. Зарезервированные слова не могут применяться в качестве имен переменных. Кроме того, имя переменной не может начинаться с букв *FN*, которые используются в начале имени пользовательской функции.

Для объявления переменной используется оператор *Dim*, который резервирует в памяти компьютера поименованную область для хранения данных указанного типа. При этом тип переменной определяется типом данных, хранящихся в ней:

Dim имя_переменной As тип_данных

Переменные, объявленные в процедуре, могут использоваться только в ней. При определении переменных в области определений

модуля с помощью ключевых слов *Private* и *Dim* они будут доступны всем процедурам данного модуля. Если при объявлении переменных в области определений модуля использовалось ключевое слово *Public*, то она будет доступна для всех модулей во всех проектах.

Массив – это последовательность данных одинакового типа, имеющих общее имя. Элемент массива является отдельной переменной. Он идентифицируется путем указания в скобках после имени для одномерного массива индекса его элемента, а для многомерного массива разделенных запятой индексов его элемента.

Для объявления массива необходимо указать его размерность. При этом в VBA по умолчанию индексы элементов массива начинаются с нуля. Для удобства можно изменить начальный индекс элементов массива с помощью оператора *Option Base*. Определим, например, матрицу A(i, j) размерностью пять на пять, элементы которой имеют тип данных *Integer*:

Dim A (1 to 5, 1 to 5) As Integer

2.5.3. Арифметические выражения и функции VBA

Выражения в VB состоят из элементов и операций, связывающих их. В одном выражении нельзя использовать несовместимые типы данных. Элементами выражений могут быть константы, переменные и функции.

Функции в VBA реализуют стандартные математические функции, функции преобразования данных, функции даты и времени, строковые функции, а также некоторые функции самих host-приложений Word и Excel [8, с. 151].

Элементы числовых выражений могут связываться между собой с помощью арифметических и логических операций, важнейшие из которых приведен в табл. 2.4 в порядке убывания их приоритета (устанавливающего порядок выполнения операций в выражении).

Операции с одинаковым приоритетом выполняются в порядке просмотра выражения слева направо. Для изменения порядка выполнения операций в выражении можно использовать скобки.

Все арифметические операции имеют традиционную форму записи для математики.

Запись 31 *MOD* 4 обозначает остаток от целочисленного деления 31 на 4 и возвращает в результате число 3.

Таблица 2.4

Обозначение операции	Наименование операции
^	Возведение в степень
–	Унитарная операция минус (для обозначения отрицательных чисел или изменения знака)
* , /	Умножение, деление
\	Целочисленное деление
MOD	Вычисление остатка от деления
+ , –	Сложение, вычитание
<, <=, =, >=, >	Операции сравнения
<>	Операция не равно
NOT	Логическое НЕ
AND	Логическое И
OR	Логическое ИЛИ

Результатом операции сравнения является *True* (истина или 1) либо *False* (ложь или 0).

В VBA есть ряд встроенных функций, упрощающих вычисления и операции, которые включают в себя следующие группы [11]:

– математические функции;

– функции преобразования форматов;

– функции проверки типов;

– функции обработки строк;

– функции даты и времени.

Все функции подробно рассмотрены в справочной системе VBA, а также в [8, 11].

Чтобы получить список функций VBA при написании кода в окне редактирования, введите *VBA* и точку (.).

Особое место среди встроенных функций VBA занимают функции, которые не только возвращают одно значение, но и отображают диалоговое окно, в котором пользователь может выполнять определенные действия либо выводить сообщения.

В VBA также имеется встроенное диалоговое окно ввода *InputBox*, которое позволяет ввести данные в программу через *текстовое поле на диалоговой панели*. Соответствующая функция имеет следующий синтаксис:

InputBox (Приглашение$, Заголовок$, [По Умолчанию$])

В процессе выполнения этой функции появляется диалоговая панель с текстовым полем. В строке заголовка панели будет печа-

таться значение второго аргумента *Заголовок$*, на самой панели печатается значение аргумента *Приглашение$*, а в текстовом поле печатается значение аргумента *По Умолчанию$* (если это значение отсутствует, то содержимое текстового окна также отсутствует).

Введенная пользователем в текстовое поле строка становится значением функции.

На рис. 2.9 показано диалоговое окно, которое появится на экране в результате выполнения следующего программного кода:

x = InputBox("x = ", "Введите значение")

и введения в текстовое поле значения «1» с клавиатуры.

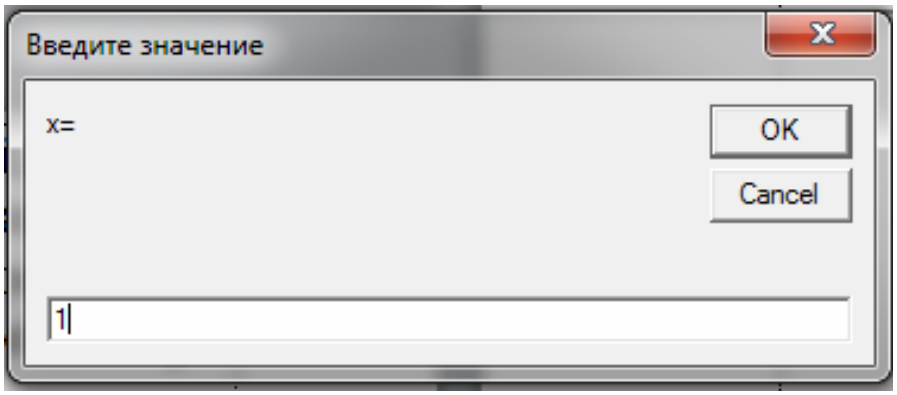


Рис. 2.9

После нажатия клавишу *ОК* переменной *x* присвоится значение, равное строковой константе «1».

Для преобразования строковых значений (*x*), вводимых в текстовые поля, в десятичное число необходимо использовать функцию *Val(x)*. При этом строка *x* должна содержать только цифры и одну десятичную точку, иначе VBA не может преобразовать ее в число и возвращает значение 0.

С помощью функции *MsgBox* можно добавлять в проект *панели сообщений*. В простейшем случае функция *MsgBox* работает в режиме оператора и используется для вывода сообщений на специальной панели сообщений. Синтаксис функции в режиме оператора

MsgBox Сообщение$ [,ЧислоКод1+ЧислоКод2] [,Заголовок$]

В результате выполнения данной команды на экране появится панель сообщений со строкой *Сообщение$*, в строке заголовка будет выведено *Заголовок$*, а аргумент *ЧислоКод1 + ЧислоКод2* определяет внешний вид панели.

С помощью одного числа, являющегося суммой чисел *ЧислоКод1* и *ЧислоКод2*, можно одновременно установить определенную пиктограмму и определенную комбинацию кнопок, размещенных на панели сообщений. Значения чисел *ЧислоКод1* и *ЧислоКод2* приведены в табл. 2.5 и 2.6.

Таблица 2.5

ЧислоКод1	Пиктограмма
16	
32	
48	
64	

Таблица 2.6

ЧислоКод2	Набор кнопок
0	*OK*
1	*OK*, *Cancel*
2	*Abort*, *Retry*, *Ignore*
3	*Yes*, *No*, *Cancel*
4	*Yes*, *No*
5	*Retry*, *Cancel*

Например, число 36 можно рассматривать как сумму числа 4 (код комбинации кнопок Yes, No) и числа 32 (код пиктограммы  «Вопрос»).

Для вывода на панели сообщений с заголовком *"MsgBox в режиме оператора"* сообщения *"Ввод закончен"* (рис. 2.10) требуется набрать следующий программный код:

```
MsgBox "Ввод закончен", 48, "MsgBox в режиме оператора"
```

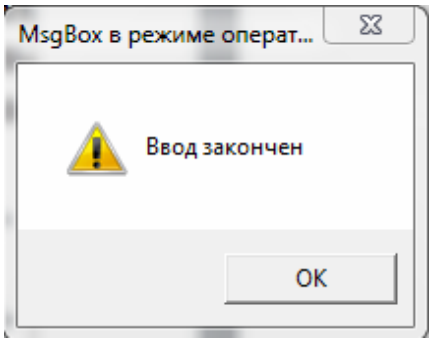


Рис. 2.10

В случае записи аргументов функции *MsgBox* в скобках функция возвращает, в зависимости от нажатой клавиши, одно из значений из табл. 2.7.

Возвращаемое функцией *MsgBox* значение может быть использовано в программе, что позволяет организовывать интерактивный режим работы программы.

Таблица 2.7

Нажатая кнопка	Возвращаемое значение
OK	1
Cancel	2
Abort	3
Retry	4
Ignore	5
Yes	6
No	7

2.5.4. Основные операторы языка программирования VBA для создания линейных и разветвляющихся программ

Наименьшей единицей программного кода VBA является *оператор*.

Оператор присваивания значения имеет следующий формат:

$$a = b,$$

где *a* – имя числовой, символьной или индексной переменной (должно стоять слева от знака равенства), которой присваивается значение;

b – выражение, значение которого присваивается переменной *a*, причем тип выражения должен соответствовать типу переменной (для числовой переменной числовое выражение, а для символьной – символьное выражение).

Для переменных типа объект при ссылке на него необходимо добавить ключевое слово *Set*, например

Set obj1 = obj2

При необходимости в VBA можно использовать оператор безусловного перехода *GoTo*, который изменяет последовательность выполнения операторов программы.

Он имеет следующий формат:

GoTo Метка

Для организации разветвляющихся программ в VBA используют, в частности, оператор условного перехода *If…Then.* Существуют два его варианта: *строчный* и *блочный*.

Строчный оператор *If…Then* имеет следующий формат:

If условие1 [And условие2…] Then операторы [Else операторы]

где *условие1* (*условие2*) – это условное выражение;

операторы – оператор или последовательность разделенных двоеточием выполняемых операторов.

Если *условие1* истинно (*True*), то выполняются операторы, следующие за *Then***,** после чего выполнение строчного оператора *If…Then* прекращается и программа переходит к следующему оператору.

Если *условие1* ложно (*False*), то операторы за *Then* пропускаются и выполняются операторы за *Else*, если они есть, затем программа также переходит к следующему оператору.

После *If* может проверяться выполнение нескольких условий:

условие1 And условие2

Открыть файл с макросами, созданный в Word, и перейти в среду IDE, нажав клавиши *Alt+F11*. Войти в модуль *NewMacros* проекта *Project* текущего файла. Перейти в окно редактирования и записать в нем следующий программный код:

```
Sub Макрос3()
x = InputBox("Введите 1 – если строчный, 2 – если Блочный ", _
"Введите значение")
If x = 1 Then x = “Строчный If...Then" Else x = “Блочный If...Then"
    MsgBox x
End Sub
```

Обратить внимание на то, что строчный оператор *If…Then* и все связанные с ним операторы, включая следующие за *Else*, должны быть в одной программной строке.

При запуске на исполнение макроса *Макрос3* на экране появится диалоговое окно. На рис. 2.11 показано это окно, после введения в текстовое поле значения «1» с клавиатуры.

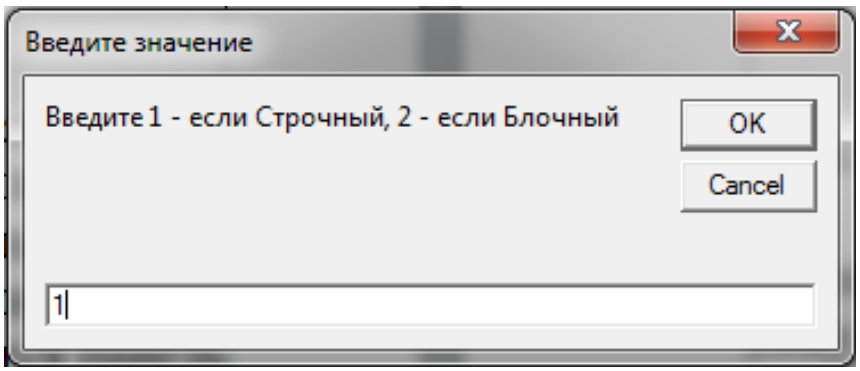


Рис. 2.11

Если нажать на клавишу *OK*, то на экране появится панель сообщений (рис. 2.12).

Если же в текстовое поле ввести с клавиатуры цифру 2 и нажать на клавишу *OK*, то на экране появится панель сообщений (рис. 2.13).

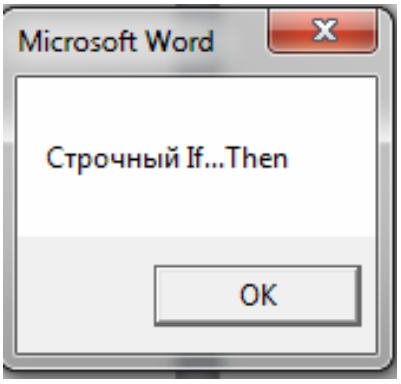


Рис. 2.12

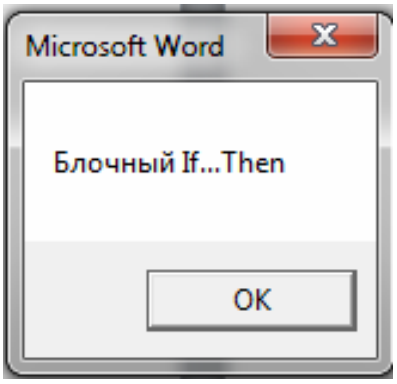


Рис. 2.13

В VBA допускается вложенность операторов *If...Then*, ограниченная только одной строкой, при этом оператор *Else* соответствует последнему не закрытому оператору *Then*.

Если в операторе *If...Then* после слова *Then* помещаются несколько операторов, то удобнее использовать *блочный* оператор *If...Then*, имеющий следующий формат:

```
If условие1 Then
    . . . операторы
    [ElseIf условие2 [ ... ] Then
    . . . операторы ]
      . . .
    [Else
    . . . операторы ]
End  If
```

При выполнении этого блока сначала проверяется истинность *условие1*. Если оно ложно, то по порядку проверяются условия в каждом из следующих за ним операторов *ElseIf* (их может быть сколько угодно).

Как только находится истинное условие, выполняются операторы, следующие за соответствующим оператором *Then*, после чего выполнение блока *If...Then* прекращается и управление передается на оператор, следующий за оператором *End If*.

Если никакое из проверенных условий *ElseIf* несправедливо, то выполняются операторы за *Else*, если они есть, после чего выполнение блока *If...Then* прекращается.

В блочном операторе *If…Then* после ключевых слов *Then* и *Else* не должно быть других операторов. *If*-блок может быть вложенным, т.е. любой из операторов, следующий за всяким *Then* или *Else*, может содержать другой *If*-блок.

Рассмотрим примеры использования строчного и блочного операторов *If…Then*. Определить, чему будет равна переменная *A* после выполнения оператора *If…Then*.

При использовании строчного оператора *If…Then*, программный код будет иметь следующий вид:

```
If x>y THEN A=A+1 Else If z<y AND z>x Then A=A+2 Else  A=A*2
```

Для реализации того же самого алгоритма, но с помощью блочного оператора *If…Then* требуется записать следующий программный код:

```
If x>y THEN
   A=A+1
   ElseIf  z<y  AND  z>x  Then
   A=A+2
   Else
   A=A*2
End If
```

Если $A = 4$, $z = 1$, $x = 2$, $y = 3$, то в результате выполнения как первого, так и второго программного кода переменная *A* будет равна 8.

2.5.5. Операторы VBA для создания циклических программ

Часто при составлении программ приходится повторять некоторую часть программы *заданное число раз*. Пример такого циклического алгоритма приведен на рис. 1.13,*б*. Он предназначен для вычисления и вывода на экран значения функции $Y(x) = \sin x$ для значений аргумента *x*, изменяющегося в интервале от 0 до 20 с шагом 2.

В VBA для организации цикла с заданным числом повторений используется оператор *For…Next*, имеющий следующий формат:

```
For переменная = x To y [Step z]
. . . операторы
Next [переменная [ ,переменная] . . . ]
```

где *переменная* – числовая переменная, используемая как счетчик повторов; *x*, *y*, *z* – числовые выражения, определяющие соответственно начальное, конечное значение и приращение счетчика.

Оператор *For* задает начальное *x* и конечное *y* значение *переменной* цикла, а также шаг *z* (по умолчанию *1*), с которым она будет изменяться.

Конструкция выполняет операторы, следующие за оператором *For*, пока в программе не встретится оператор *Next*. Тогда к текущему значению *переменной* цикла прибавляется значение шага *z* и полученное значение сравнивается с конечным значением **y**.

Если значение *переменной* цикла больше конечного значения *y*, то выполняется оператор, следующий за оператором *Next*, иначе управление снова передается оператору, находящемуся за оператором *For*.

Например, процедура, реализующая алгоритм, приведенный на рис. 1.13,*б*, с помощью оператора *For…Next* имеет следующий вид:

```
Sub Макрос6()
   For x = 0 To 20 Step 2
      y = Sin(x)
     y = Round(y, 2)
          Ответ = Ответ + CStr(y) + "  "
   Next x
         MsgBox Ответ
End Sub
```

Результат работы программы показан на рис. 2.14.

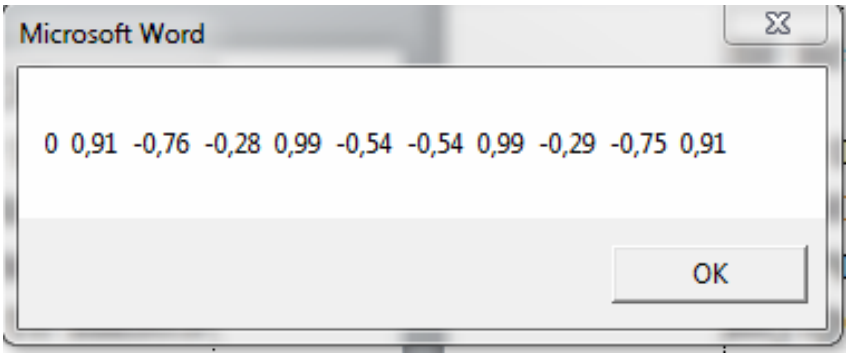


Рис. 2.14

В четвертой строке процедуры после вычисления значения функции *y* для текущего значения переменной цикла *x* происходит округление полученного значения до двух знаков после запятой с помощью функции *Round*.

В пятой строке путем накапливания формируется строковая константа *Ответ*, включающая в себя 11 значений функции *y*, разделенных пробелами.

Циклы с неопределенным количеством повторений позволяет задавать конструкция *While…Wend*, в которой указывается условие продолжения цикла:

```
While x
        ...[операторы]
Wend
```

где *x* – числовое выражение.

Цикл повторяется до тех пор, пока выражение *x* не станет равным нулю (*False*). При $x = 0$ все операторы до оператора *Wend* включительно пропускаются, и программа выполняется дальше.

Например, если в процедуре *Макрос6* для организации цикла использовать оператор *While…Wend*, то программа будет иметь вид

```
Sub Макрос7()
  x = 0
    While x <= 20
          y = Sin(x)
          y = Round(y, 2)
          Ответ = Ответ + CStr(y) + "  "
          x = x + 2
    Wend
  MsgBox Ответ
End Sub
```

Изучите самостоятельно наиболее общую форму организации циклов с помощью операторов *Do…Loop*, которая позволяет проверять условие окончания цикла в начале или в конце цикла [8, 10].

Циклы могут быть вложенными. Всякий оператор *For (Wend)* относится к ближайшему оператору *Next* (*While*).

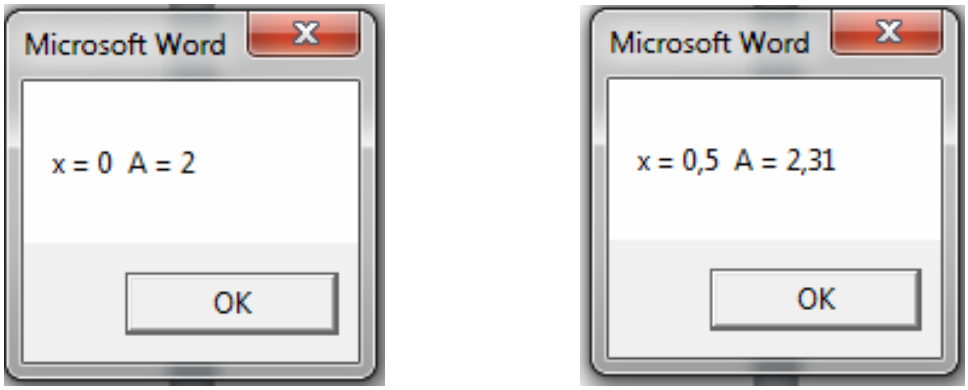
Рассмотрим пример программы, вычисляющей и выводящей на экран монитора значения функции

$$A = \prod_{k=1}^{n} \sqrt{x^2 + k^2}$$

при *x*, изменяющемся от 0 до 5 с шагом 0,5 и для любого значения *n*. Соответствующая схема алгоритма приведена на рис. 1.24. Она может быть реализована с помощью макроса *Макрос8*:

```
Sub Макрос8()
n = InputBox("Введите значение n")
n = Val(n)
      For x = 0 To 5 Step 0.5
         A = 1
            For k = 1 To n
                  A = A * Sqr(x ^ 2 + k ^ 2)
                 A = Round(A, 2)
            Next k
MsgBox "x = " + CStr(x) + "  A = " + CStr(A)
      Next x
End Sub
```

В результате выполнения данного макроса на экране сначала появится диалоговое окно, куда необходимо ввести требуемое значение переменной *n*, например равное «2», а затем панель сообщений (рис. 2.15,*а*) с результатом вычисления функции *A* для аргумента $x = 0$. После нажатия на клавишу ОК появится следующая панель сообщений (рис. 2.15,*б*) и т.д., всего 11 панелей.



а) *б*)

Рис. 2.15

Обратите внимание, что вычисление произведения *A* организуется в цикле путем последовательного перебора и умножения отдельных сомножителей как *A = A*f(x,k)*. При этом, когда вычисляется первое произведение, переменная *A* должна равняться «1», т.е. перед внутренним циклом необходимо присвоить переменной *A* значение, равное единице.

Если необходимо организовать вычисление суммы, $A = \sum_{k=1}^{n} f(x,k)$, то в цикле необходимо последовательно перебирать и складывать отдельные слагаемые по формуле *A = A + f(x, k)*.

При этом первая сумма получается путем сложения первого слагаемого с нулем, т.е. перед внутренним циклом необходимо присвоить переменной *A* значение, равное нулю.

Если несколько циклов имеют общий конечный оператор, можно указать для них один оператор *Next*, перечислив в нем переменные циклов в порядке, обратном порядку операторов *For*.

2.5.6. Циклические программы по обработке массивов

Для работы с массивами также удобно использовать циклические программы. Например, процедура *Макрос9*, присваивающая с помощью окна ввода значения элементам матрицы А размерностью 2 на 3 и выводящая затем матрицу на панель сообщения, имеет вид

```
Sub Макрос9()
  Dim i As Integer: Dim j As Integer
  Dim A(2, 3) As Integer
  Dim Матрица As String
  Матрица = ""
   For i = 1 To 2
            For j = 1 To 3
            Aij = "A(" + CStr(i) + "," + CStr(j) + ")"
A(i, j) = InputBox (Aij, " Введите значения элементов матрицы")
           Матрица = Матрица + CStr(A(i, j)) + vbTab
            Next j
   Матрица = Матрица + vbLf
   Next i
MsgBox Матрица
End Sub
```

В результате выполнения данного макроса на экране последовательно выводятся шесть диалоговых окон (рис. 2.16), в которых указываются индексы элементов матрицы *A(i, j),* для которых необходимо ввести требуемое значение. Затем появляется панель сообщений (рис. 2.17), в которой помещаются все введенные значения элементов матрицы *A(i, j)* в виде матрицы.

Во внутреннем цикле процедуры *Макрос9* для вывода элементов по столбцам (через равные интервалы) при формировании текущего значения строковой константы *Матрица* в конце добавляет-

ся системная переменная *vbTab* (знак табуляции), которая добавляет в конце константы недостающие пробелы. Что позволяет вывести все значения в строке матрицы через равные интервалы.

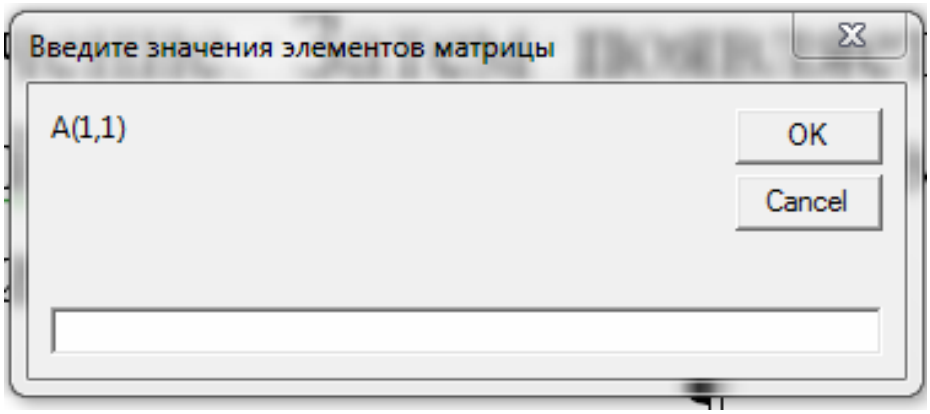


Рис. 2.16

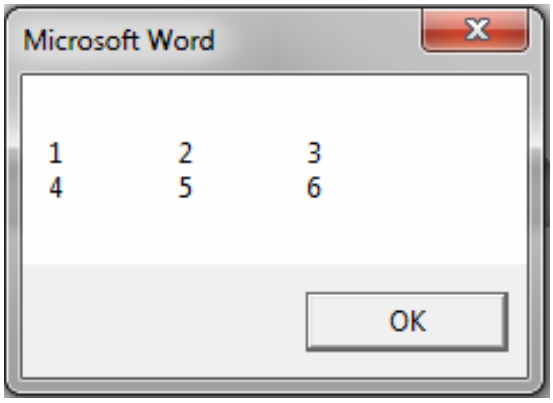


Рис. 2.17

Во внешнем цикле процедуры *Макрос9* для вывода элементов по строкам при формировании текущего значения строковой константы *Матрица* в конце добавляется системная переменная *vbLf*, которая выполняет перевод строки.

Для присвоения произвольных значений элементам матрицы можно использовать функцию генератора случайных чисел *RND*, которая возвращает случайные положительные числа из диапазона от 0 до 1. *Макрос9а* автоматически формирует и выводит в панель сообщений матрицу, значения элементов которой – случайные числа из диапазона от –50 до 50:

```
Sub Макрос9а()
Dim i As Integer: Dim j As Integer: Dim A(2, 3) As Integer
Dim Матрица As String:Матрица = ""
   For i = 1 To 2
            For j = 1 To 3
```

```
            A(i, j) = (Rnd - 0.5) * 100
            Матрица = Матрица + CStr(A(i, j)) + vbTab
             Next j
        Матрица = Матрица + vbLf
        Next i
    MsgBox Матрица
    End Sub
```

В результате выполнения данного макроса в панели сообщений (рис. 2.18) выводятся значения элементов матрицы *A(i, j)* в виде матрицы.

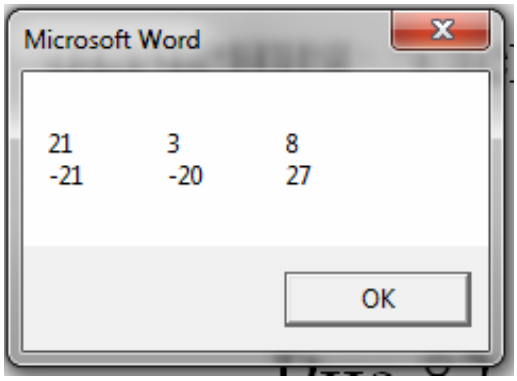


Рис. 2.18

2.6. Использование форм и элементов управления VBA для организации интерфейса программ

2.6.1. Форма и элементы управления VBA

Основным объектом в VBA является *форма*, представляющая собой окно приложения, в котором можно разместить различные элементы управления. Форма может быть добавлена в проект выбором команды *Insert→UserForm* или с помощью кнопки *UserForm* на стандартной панели инструментов в редакторе VBA (рис. 2.19). В результате на экране появляется сама форма, которая добавляется в проект с именем *UserForm1* по умолчанию (см. рис. 2.19). Если в проект добавить следующую форму, то она будет иметь имя *UserForm2* и т.д.

Размеры формы можно менять, перетаскивая мышкой ее правую или нижнюю границу. Как и любой объект в VBA, форма обладает свойствами, методами и событиями. Наиболее часто используемые свойства, методы и инструкции форм приведены в табл. П.А.1–П.А.3.

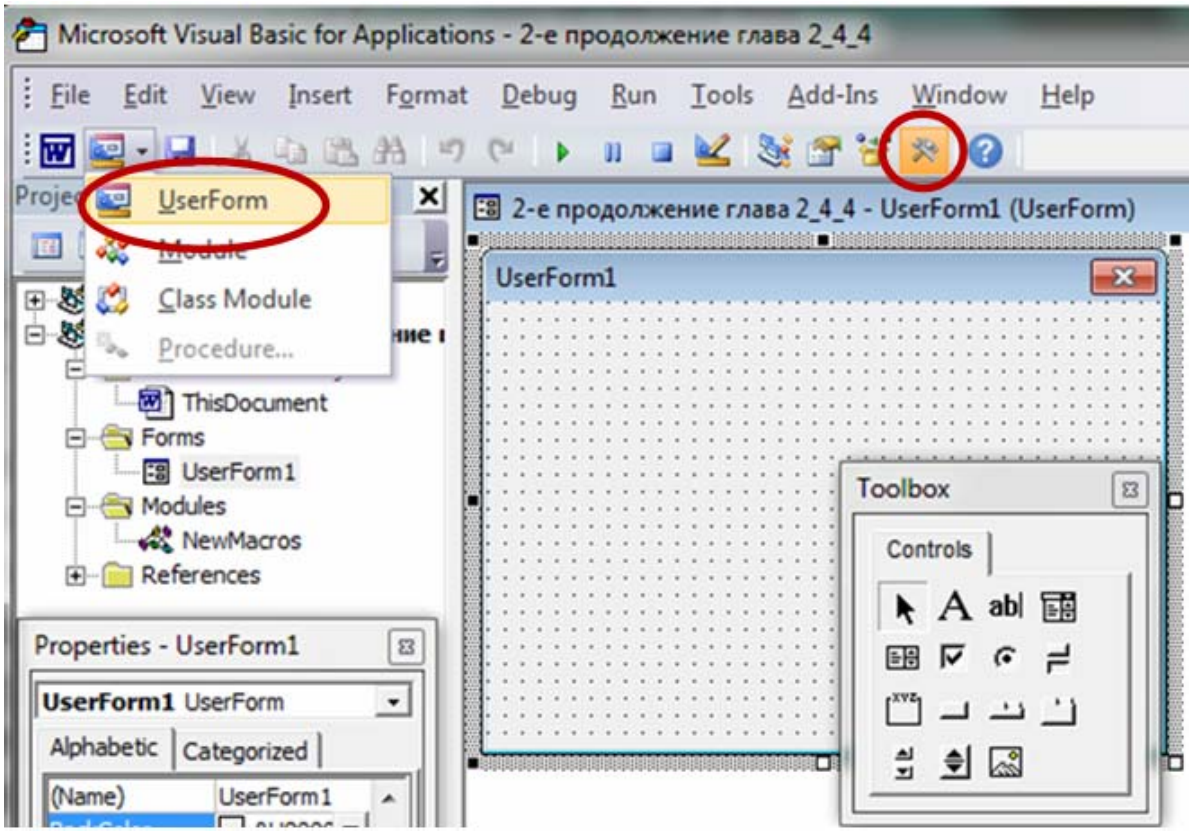


Рис. 2.19

Первоначально форма пуста, в дальнейшем в процессе создания графического интерфейса приложения в ней размещаются *элементы управления*. Используя эти элементы, нетрудно создавать любой пользовательский интерфейс в среде Windows. Пиктограммы элементов управления помещаются на *Панели элементов управления Toolbox*, которая либо появляется одновременно с появлением формы, либо может быть вызвана с помощью команды *View→ Toolbox→Edit* или с помощью кнопки  на стандартной панели инструментов в редакторе VBA (см. рис. 2.19).

Список основных элементов управления VBA приведен в табл. П.А.4. Все кнопки панели инструментов, за исключением первой, служат для создания элементов управления. Первая кнопка называется *Указатель* (*Point*); щелкнув по нему, можно выбрать уже созданный в форме элемент управления, изменить его размер или переместить. Данный режим конструирования формы включается автоматически после размещения элементов управления в форме. Для размещения элементов управления в форме необходимо, выбрав щелчком мыши элемент, поместить его в нужное место на форму проектируемого приложения. После этого элемент управления можно перемещать, изменять его размеры, копировать в буфер обмена и вставлять из буфера обмена. Для установки свойств эле-

ментов управления вручную при его конструировании необходимо его выделить и нажать кнопку *Properties*.

Все элементы управления формы образуют семейство *Controls*. Общий способ обращения к любому элементу управления в семействе *Controls* осуществляется путем указания его индекса:

Controls (номер элемента управления в проекте)

Индексы объектов *Control* начинаются с 0, затем 1 и т.д.

С формой связывается программный модуль, содержащий событийные и общие процедуры.

У каждого элемента управления есть целый набор событий, связанных с теми или иными действиями. Некоторые общие для форм и элементов управления события приведены в табл. П.А.5.

Для вывода программного модуля на экран можно воспользоваться контекстным меню, выбрав там команду *View→Code* или просто выполнить двойной щелчок по форме клавишей указания мыши, а для возвращения в окно форм необходимо выбрать в контекстном меню команду *Hide*, либо выбрать требуемую форму в окне проектов. Кроме того, любое из открытых в данном проекте окон может переключаться и изменяться средствами Windows.

2.6.2. Пример создания приложения в Microsoft Word

Зайти в среду VBA и создать приложение *Калькулятор*. Виртуальный калькулятор должен выполнять четыре арифметических действия – сложение, вычитание, умножение и деление.

Создание приложения включает в себя три этапа:

– виртуальное программирование графического интерфейса калькулятора;

– задание значений свойствам объектов графического интерфейса калькулятора;

– создание программного кода, связанного с элементами управления на форме.

1. Вставить в проект форму и разместить на ней необходимые для функционирования приложения элементы управления (рис. 2.20):

– первых три текстовых поля (объекты *TextBox1*, *TextBox2*, *TextBox3*): два поля для ввода числовых данных и третье – для вывода результатов вычислений;

– четыре кнопки (объекты *CommandButton1*, *CommandButton2*, *CommandButton3*, *CommandButton4*).

Расположить управляющие элементы в форме, таким образом, как это показано на рис. 2.20.

2. Для изменения свойств объектов вызвать окно свойств *Properties*. Последовательно выделяя каждый объект, включая саму форму, задать для них значения свойства *Caption* (Надпись) в соответствии с табл. 2.8.

В первом столбце таблицы в скобках указаны имена объектов по умолчанию – свойство *Name* (Имя).

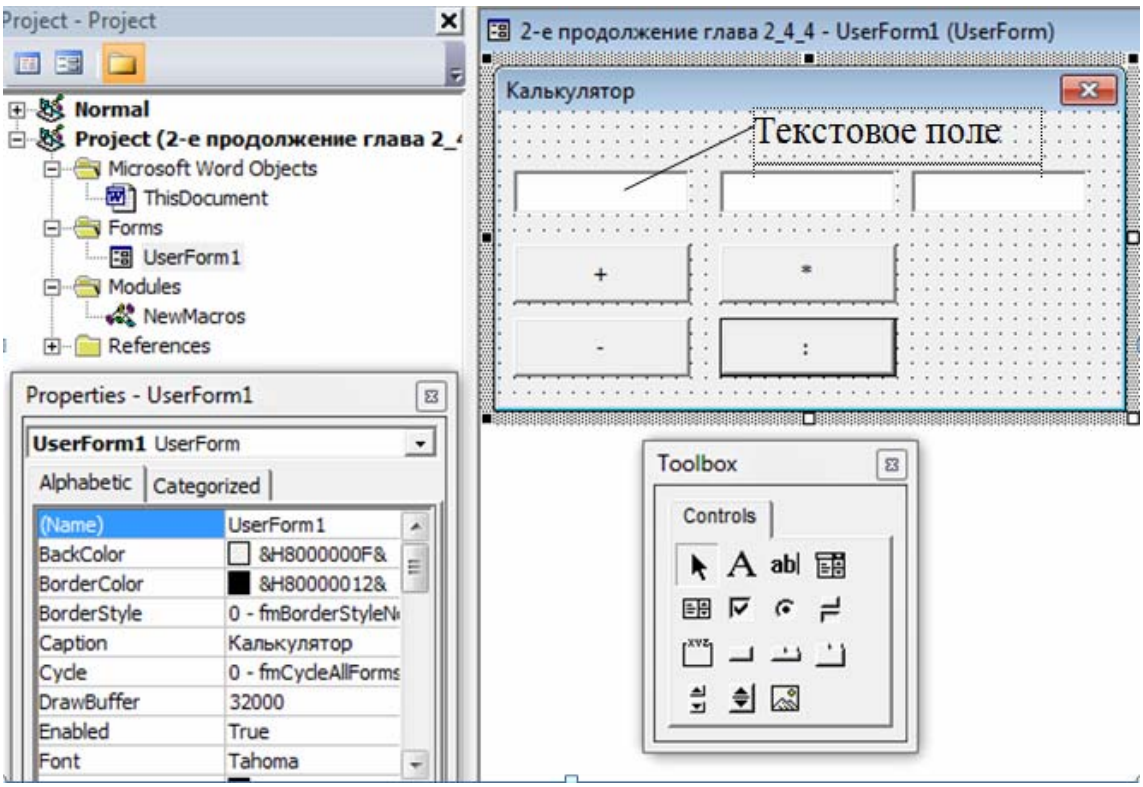


Рис. 2.20

Таблица 2.8

Класс объектов (свойство *Name*)	Значение свойства *Caption* по умолчанию	Новое значение свойства *Caption*
Форма (*UserForm1*)	*UserForm1*	*Калькулятор*
Кнопка (*CommandButton1*)	*CommandButton1*	+
Кнопка (*CommandButton2*)	*CommandButton2*	–
Кнопка (*CommandButton3*)	*CommandButton3*	*
Кнопка (*CommandButton4*)	*CommandButton4*	/

Для объектов *TextBox1 – TextBox3* изменить значения свойства *Font*, которое задает нужный шрифт, начертание и размер шрифта, размещаемого в текстовом поле. Для этого выделить каждое текстовое окно, найти для него в окне свойств *Properties* свойство *Font* и

навести курсор на правую часть соответствующей ячейки второго столбца таблицы (рис. 2.21). В результате раскроется диалоговое окно *Шрифт*, в котором установить требуемые параметры шрифта.

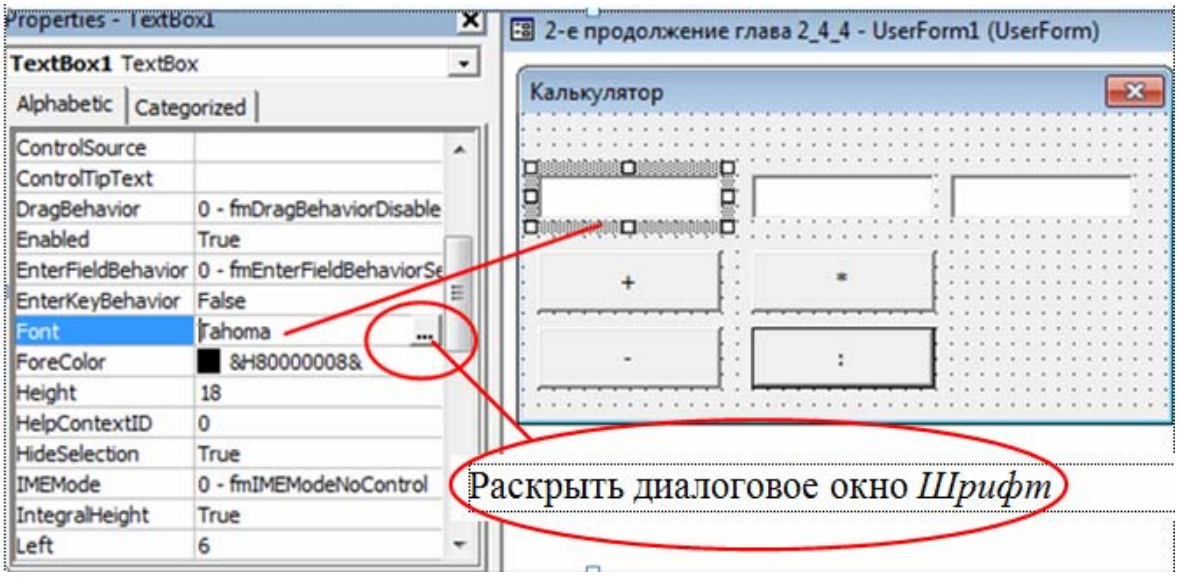


Рис. 2.21

3. На этапе создания программного кода происходит кодирование событийных процедур в приложении. В рассмотренном приложении *Калькулятор* содержатся лишь событийные процедуры, обрабатывающие событие (*Click*), произошедшее при щелчке левой клавиши мыши на соответствующем элементе управления.

Процедура сложения реализует сложение двух чисел. Эта процедура должна изменить свойство *Text* объекта *TextBox3* так, чтобы оно явилось суммой числовых значений свойства *Text* объектов *TextBox1* и *TextBox2*. Эта процедура должна вызываться, если пользователь щелкнет по кнопке с надписью «+». Для ее написания необходимо выполнить двойной щелчок по кнопке «+», в результате откроется окно редактора кода, в котором будет помещена заготовка для процедуры обработки события Click объекта *CommandButton1*:

```
Private Sub CommandButton1_Click()
End Sub
```

Внутри процедуры необходимо с клавиатуры добавить команду:

```
TextBox3.Text=Val(TextBox1.Text)+Val(TextBox2.Text)
```

Функция *Val* используется в процедуре для преобразования строковых значений, вводимых в текстовые поля, в десятичное число.

При написании программ целесообразно проверять работоспособность процедур по мере их написания. Для этого надо нажать клавишу *F5* или кнопку  или выбрать команду *Run→Run Sub*.

В результате на экране появится форма (рис. 2.22). Ввести любые числовые значения в первое и второе текстовые окна, например «100» и «22» на рис. 2.22. После нажатия на кнопку «+» в третьем окне появится числовое значение, равное сумме чисел из первых двух текстовых окон – «122» на рис.2.22.

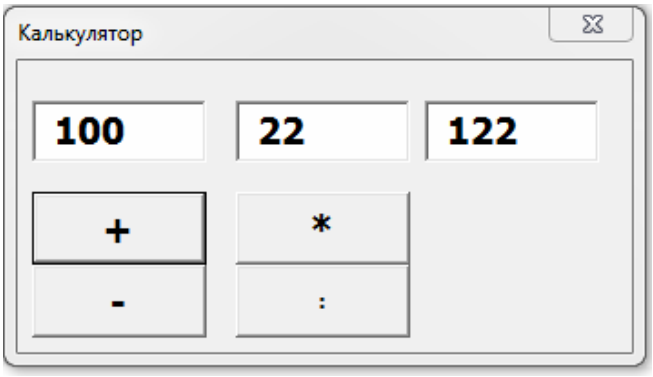


Рис. 2.22

Для прекращения работы приложения нажать стандартную кнопку Windows для закрытия окон  в правом верхнем углу формы. В результате приложение *Калькулятор* будет закрыто, а VBA вернется обратно в режим редактирования (ввода кода.)

Убедившись в правильности работы процедуры сложения *Private Sub CommandButton1_Click()*, использовать ее при написании процедуры вычитания.

Для этого выполнить двойной щелчок по кнопке «–», расположенной в форме.

В результате откроется окно редактора кода, в котором будет помещена заготовка процедуры *CommandButton2_Click()*. Скопировать внутрь процедуры вычитания программный код из процедуры сложения и заменить в нем символ «+» на символ «–». В результате получим следующий программный код:

```
Private Sub CommandButton2_Click()
TextBox3.Text = Val(TextBox1.Text) - Val(TextBox2.Text)
End Sub
```

Аналогично запишем процедуры умножения *CommandButton3_Click()* и деления *CommandButton4_Click()*.

Проверить работу вновь созданных процедур.

На этом создание приложения *Калькулятор* завершено.

4. Записать в процедуре сложения имя функции *Val с синтаксической ошибкой*, например, *Vala*. Запустить приложение *Калькулятор* и нажать на кнопку «+». В результате окно с кодом программы станет активным и появится диалоговое окно с указанием ошибки, а также выделится то место в программе, где есть ошибка (рис. 2.23).

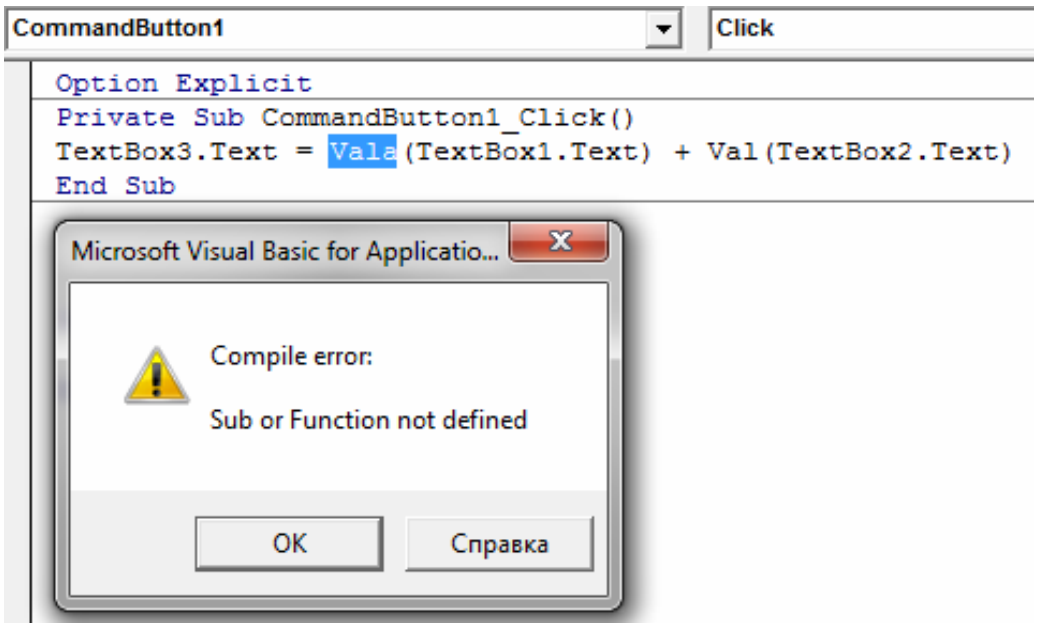


Рис. 2.23

Синтаксические ошибки отслеживаются VBA не только при трансляции кода, но и при его вводе.

При попытке запуска процедуры, содержащей ошибки *компиляции*, которые он не в состоянии правильно откомпилировать, появляется окно с указанием ошибки.

Если не понятна причина ошибки, то можно вызвать контекстную справку (кнопка *Справка*).

При нажатии на кнопку *OK* окно с сообщением об ошибке закроется и автоматически включится режим отладки *Debug*. При этом выделится (по умолчанию желтым цветом) начало процедуры, на которой произошло прерывание выполнения программы.

В программах VBA могут возникать и *ошибки времени исполнения* или *Run-time error-ошибки*, которые возникают в случае несоответствия типов переменных в выражениях присвоения значения либо в аргументах подпрограммы, а также при попытке деления на нуль либо при переполнении и т.д. Они проявляются в ходе выполнения программного кода. Например, на рис. 2.24 приведено диалоговое окно с указанием кода ошибки, которая соответствует делению на нуль.

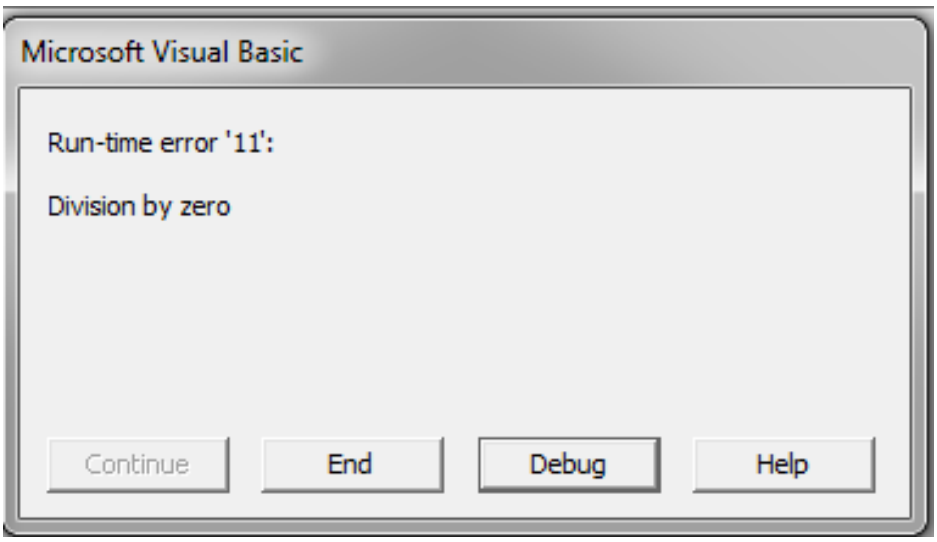


Рис. 2.24

При нажатии на кнопку *Debug* окно с сообщением об ошибке закроется и автоматически включится режим отладки *Debug*. При этом выделится (по умолчанию желтым цветом) начало процедуры, на которой произошло прерывание выполнения программы. Для того, чтобы вновь можно было бы запустить программу необходимо выйти из режима отладки, нажав кнопку *Reset* (рис. 2.25).

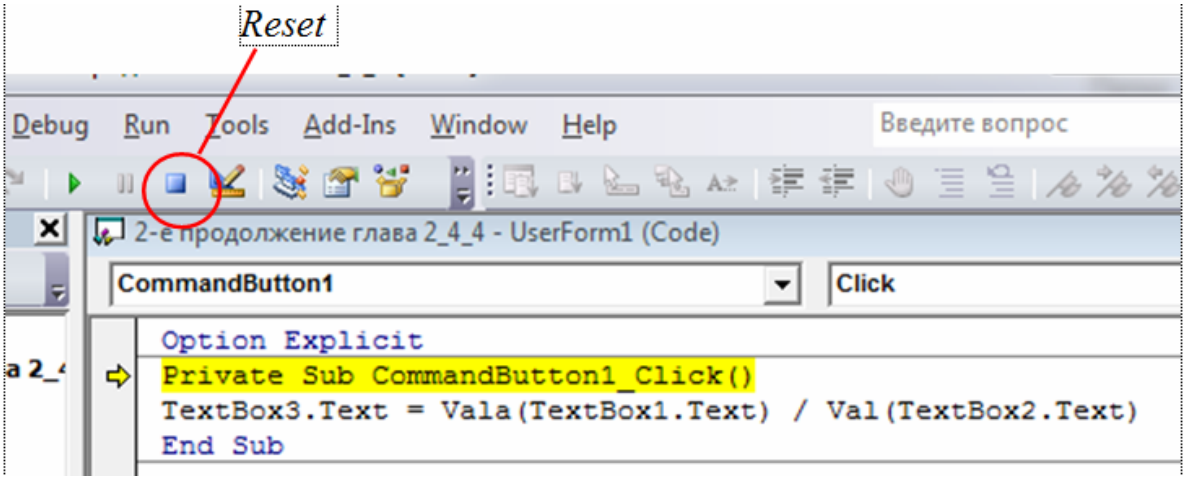


Рис. 2.25

При нажатии на кнопку *End* в окне с сообщением об ошибке (см. рис. 2.24) программа прерывается.

Исправить *синтаксическую ошибку* в процедуре сложения и убедиться, что процедура работает по-прежнему.

5. Расширить форму и добавить на нее с панели *Toolbox* следующие элементы управления: *Label* (надпись), *ComboBox* (комбинированное поле) и *CommandButton* (кнопку). Разместить элементы в форме в соответствии с рис. 2.26.

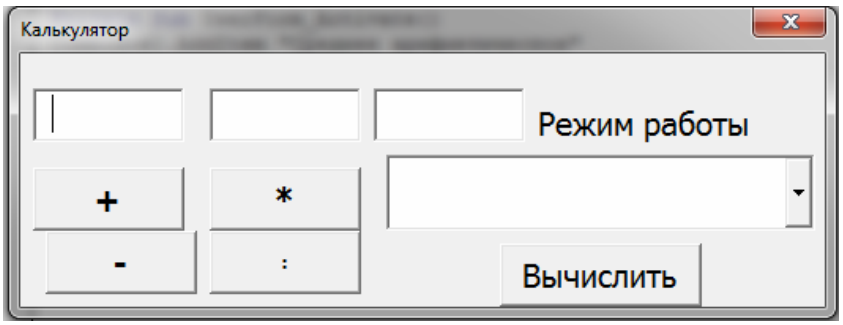


Рис. 2.26

Значения свойства *Name* элементов *Label*, *ComboBox* и *CommandButton* оставить по умолчанию: *Label1*, *ComboBox1* и *CommandButton5*.

Задать значение свойства *Caption* для объекта *Label1* – «Режим работы», а для объекта *CommandButton5* – «Вычислить».

6. Открыть окно кода, относящееся к *UserForm1* и добавить в его начале инструкцию *Option Explicit*, а также описание типа переменных модуля:

```
'Переменные уровня модуля
Private СрАрифм As Double
Private СрГеом As Double
Dim regim As String
```

7. Элемент управления *ComboBox1* будет использоваться в программе для выбора режима работы калькулятора. Он хранит список значений, из которых пользователь может выбрать только одно значение, а также позволяет вводить значение через поле ввода, также как и в элементе управления *TextBox*.

Для автоматического формирования списка значений в элементе *ComboBox* при появлении формы *UserForm1* на экране, включим в программу процедуру активизации формы:

```
Private Sub UserForm1_Activate()
ComboBox1.AddItem «Среднее арифметическое»
ComboBox1.AddItem «Среднее геометрическое»
End Sub
```

8. Записать процедуру *Private Sub CommandButton5_Click(),* которая будет запускаться при нажатии на клавишу *Вычислить* и в зависимости от выбранного в списке *ComboBox1* режима работы вычислять среднее арифметическое или среднее геометрическое значение чисел, введенных в текстовое поле объектов *TextBox1* и *TextBox2*:

```
Private Sub CommandButton5_Click()
        regim = ComboBox1.Text
        If regim = "Среднее арифметическое" Then
СрАрифм = (Val(TextBox1.Text) + Val(TextBox2.Text)) / 2
СрАрифм = Round(СрАрифм, 2)
TextBox3.Text = СрАрифм
        ElseIf  regim = "Среднее геометрическое" Then
СрГеом = (Val(TextBox1.Text)^2 + Val(TextBox2.Text)^2)^ 0.5
СрГеом = Round(СрГеом, 2) : TextBox3.Text = СрГеом
        End If
End Sub
```

9. Добавить в форму элемент управления *Image* (*Рисунок*), который используется для отображения в форме графических файлов в формате BMP, CUR, GIF, ICO, JPG и WMF. Для этого найдем в окне свойств *Properties* для объекта *Image* свойство *Picture* и наведем курсор на правую часть соответствующей ячейки второго столбца таблицы. В результате откроется диалоговое окно *Load* Picture (рис. 2.27), в котором необходимо выбрать требуемый рисунок и нажать кнопку *Открыть*.

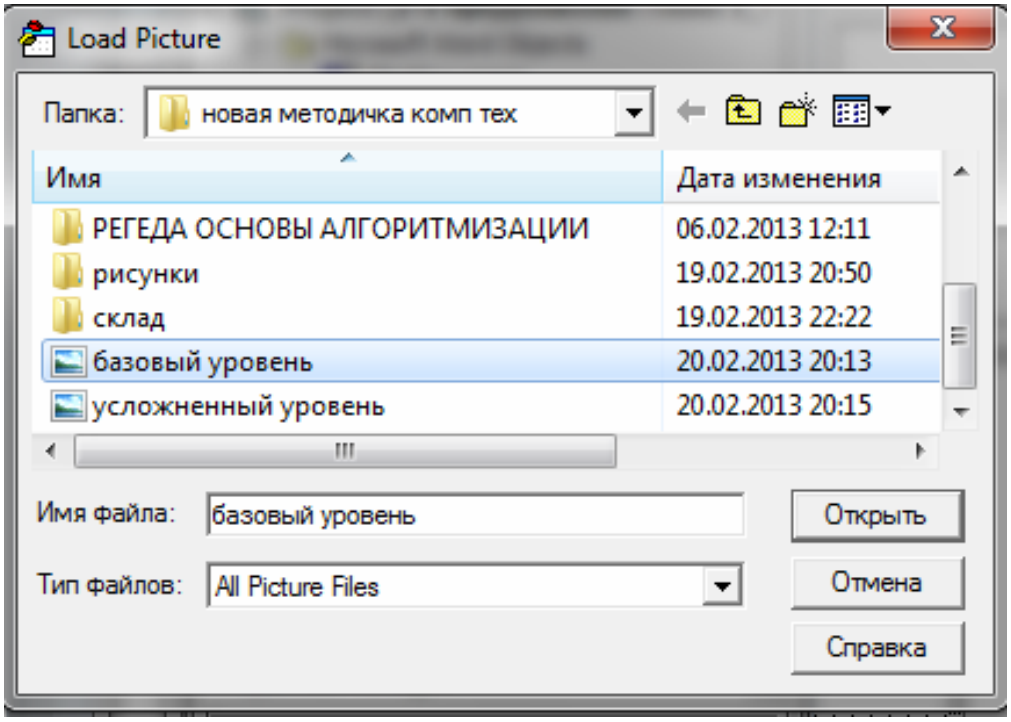


Рис. 2.27

Установить в окне *Properties* для свойства *AutoSize* значение *False*, что позволяет вписать рисунок в выбранное окно *Image*, зна-

чение *True* выводит часть рисунка, которая помещается в окне *Image*, либо уменьшает размеры окна до размеров рисунка, если рисунок оказался меньше окна.

10. Для доступа к созданному проекту из среды Microsoft Word удобно поместить в документе Word соответствующую кнопку на панели инструментов. Для этого необходимо создать пустой макрос *Макрос10()* и назначить ему кнопку на требуемой панели инструментов, а затем добавить в макросе всего одну строку, отображающую форму *UserForm1* на экране:

```
Sub Макрос10()
UserForm1.Show
End Sub
```

11. Проверить работу макроса и посмотреть в окне проектов *Project* структуру созданного проекта.

2.6.3. Пример создания приложения в Microsoft Excel

С помощью Microsoft Excel можно решить множество разнообразных задач. Использование средств VBA позволяет дополнительно создавать максимально удобный и гибкий интерфейс, приспособленный для решения конкретной задачи, предусмотреть средства защиты от несанкционированных действий, а также автоматизировать процесс создания электронных таблиц с помощью написания и использования собственных макрокоманд.

Создать в Microsoft Excel программу для построения графика функции $y = x^k$.

1. Войти в программу Microsoft Excel и сохранить книгу в файле с поддержкой макросов под именем *Построение графиков.xlsm* .

В случае, если на ленте меню нет раздела *Разработчик*, выбрать в меню команду *Файл⇒Параметры*.

В появившемся диалоговом окне *Параметры Excel* (рис. 2.28) выбрать раздел *Настройка ленты* и поставить галочку в элементе флажок слева от раздела *Разработчик*.

2. Заполнить ячейки на Листе1 текущей книги в соответствии с рис. 2.29.

3. Выделить ячейки *A2:B23*, выбрать в контекстном меню команду *Формат ячеек⇒Граница* и задать внешние и внутренние границы для выбранных ячеек.

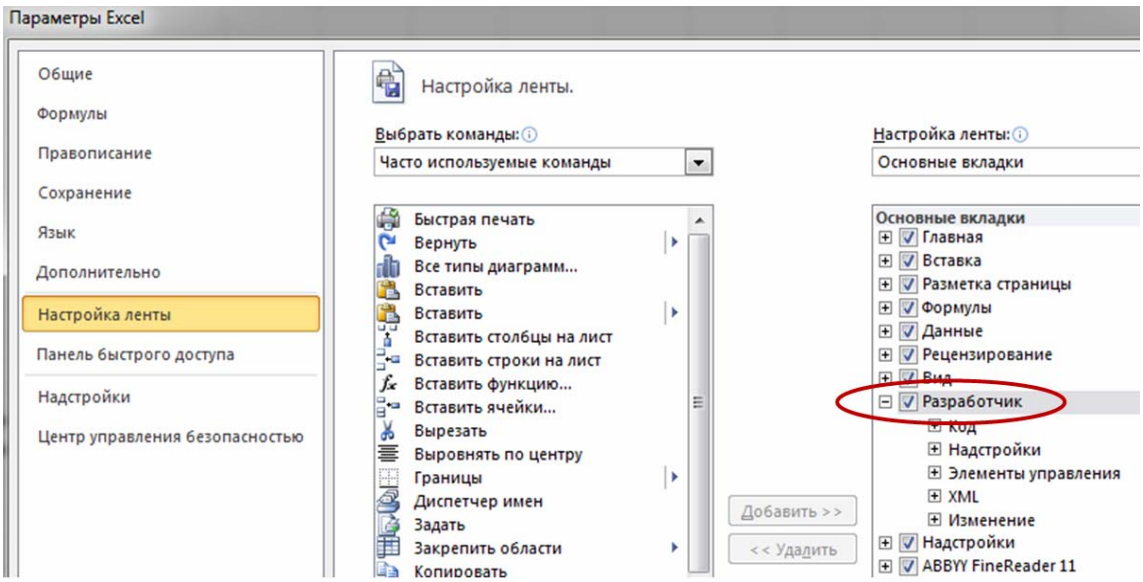


Рис. 2.28

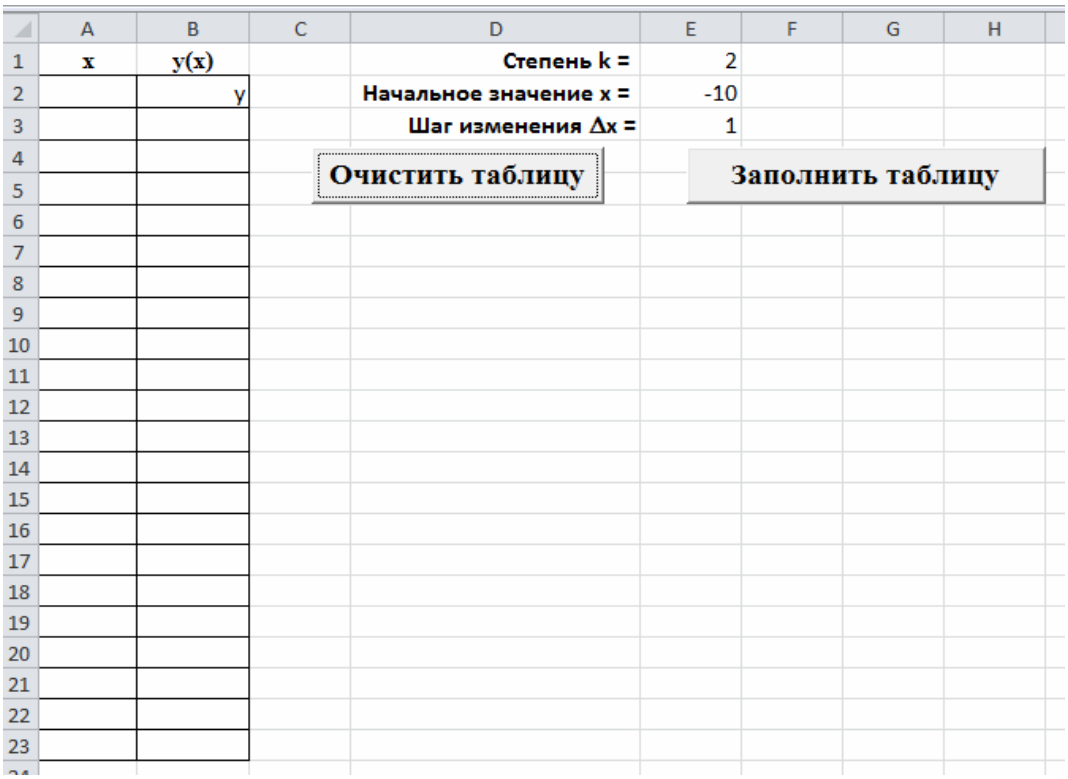


Рис. 2.29

4. Перейти на закладку меню *Разработчик* и включить *Режим конструктора*  , нажав на соответствующую кнопку (рис. 2.30). Добавить на лист две кнопки *CommandButton1* и *CommandButton2* с помощью панели *Элементы ActiveX* и назначить в соответствующем окне *Properties* свойству *Caption* для левой кнопки значение *Очистить таблицу*, а для правой кнопки значение *Заполнить таблицу*.

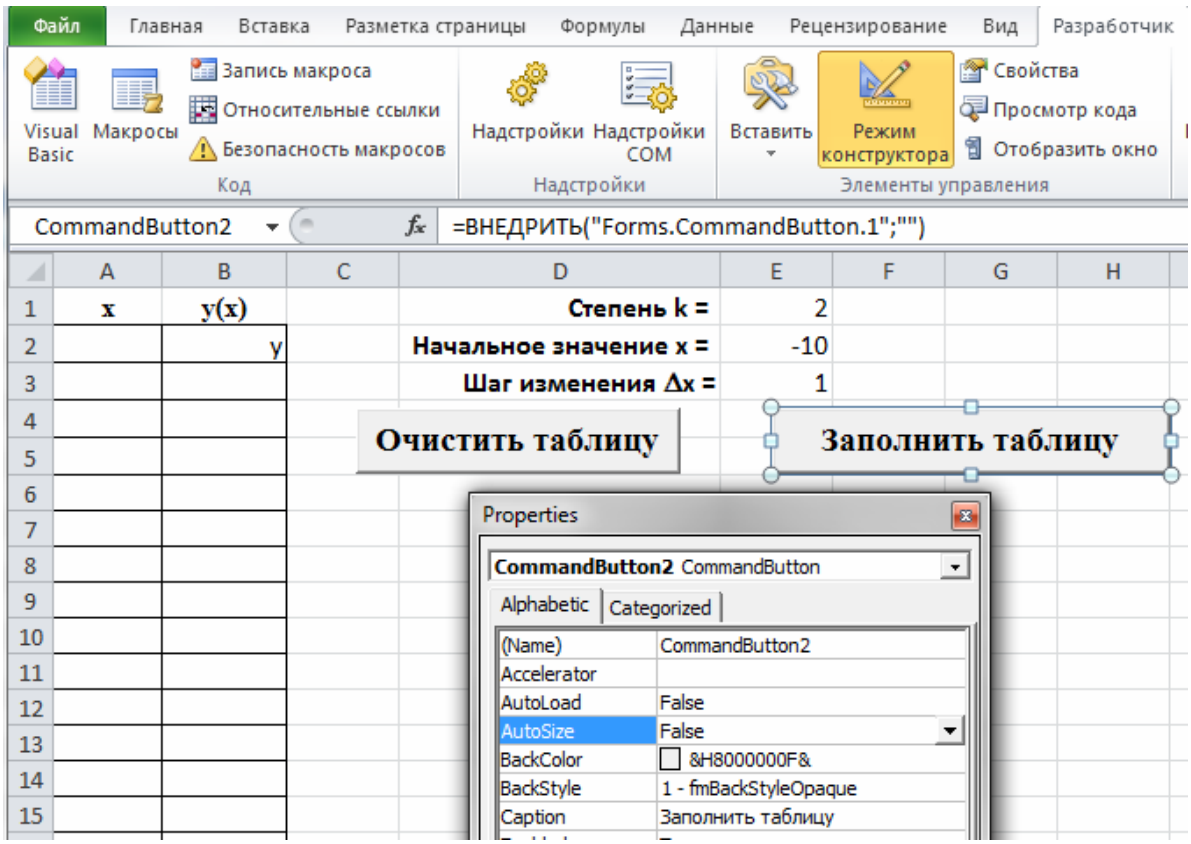


Рис. 2.30

5. Выполнить двойной щелчок на клавише *CommandButton1*, в результате откроется окно редактора кода (рис. 2.31), в котором будет помещена заготовка для процедуры обработки события *Click* объекта *CommandButton1*.

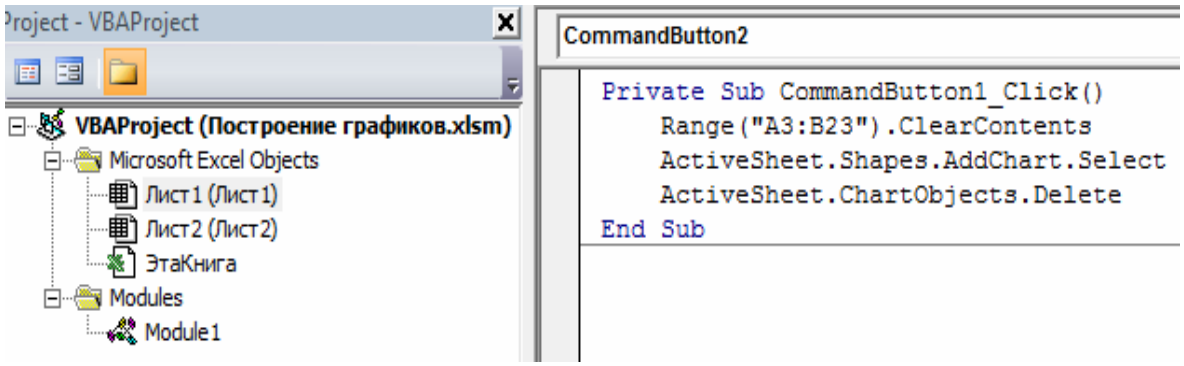


Рис. 2.31

Внутри процедуры необходимо дописать следующий код:

```
Private Sub CommandButton1_Click()
    Range("A3:B23").ClearContents
    ActiveSheet.Shapes.AddChart.Select
    ActiveSheet.ChartObjects.Delete
End Sub
```

Вторая строка записанной процедуры очищает ячейки таблицы в диапазоне ячеек *A3:B23*, а четвертая строка удаляет диаграмму *ChartObjects* из рабочего листа. Если при нажатии на кнопку *Очистить таблицу* диаграммы не было на рабочем листе, то в этой строке появится сообщение об ошибке. Поэтому в процедуру для этого случая добавлена третья строка, в которой сначала создается объект диаграмма *Chart*, а затем в четвертой строке он удаляется. Объект *Chart* является членом семейства всех графических объектов рабочего листа *Shapes*.

6) Дважды щелкнуть по клавише *CommandButton2*. В появившемся окне редактора кода необходимо дописать внутри заготовки процедуры *Sub CommandButton2_Click()* код, с помощью которого происходит заполнение ячеек таблицы в диапазоне ячеек A3:B23, а затем строится соответствующий график.

```
Private Sub CommandButton2_Click()
   x = Range("E2").Value
   k = Range("E1").Value
      For i = 3 To 23
         Cells(i, 1).Value = x
         Cells(i, 2).Value = x ^ k
         x = x + Range("E3").Value
      Next i
ActiveSheet.Shapes.AddChart.Select
      With ActiveChart
         .ChartArea.Left = 110
         .ChartArea.Top = 100
      End With
ActiveChart.ChartType = xlLine
ActiveChart.SetSourceData Source:=Range("A2:B23")
      Name = "y(x^" & CStr(k) & ")"
ActiveChart.ChartTitle.Text = Name
Range("A24").Select
End Sub
```

7. Отключить *Режим конструктора* и проверить работу программы для различных значений параметров.

Результаты работы программы приведены на рис. 2.32.

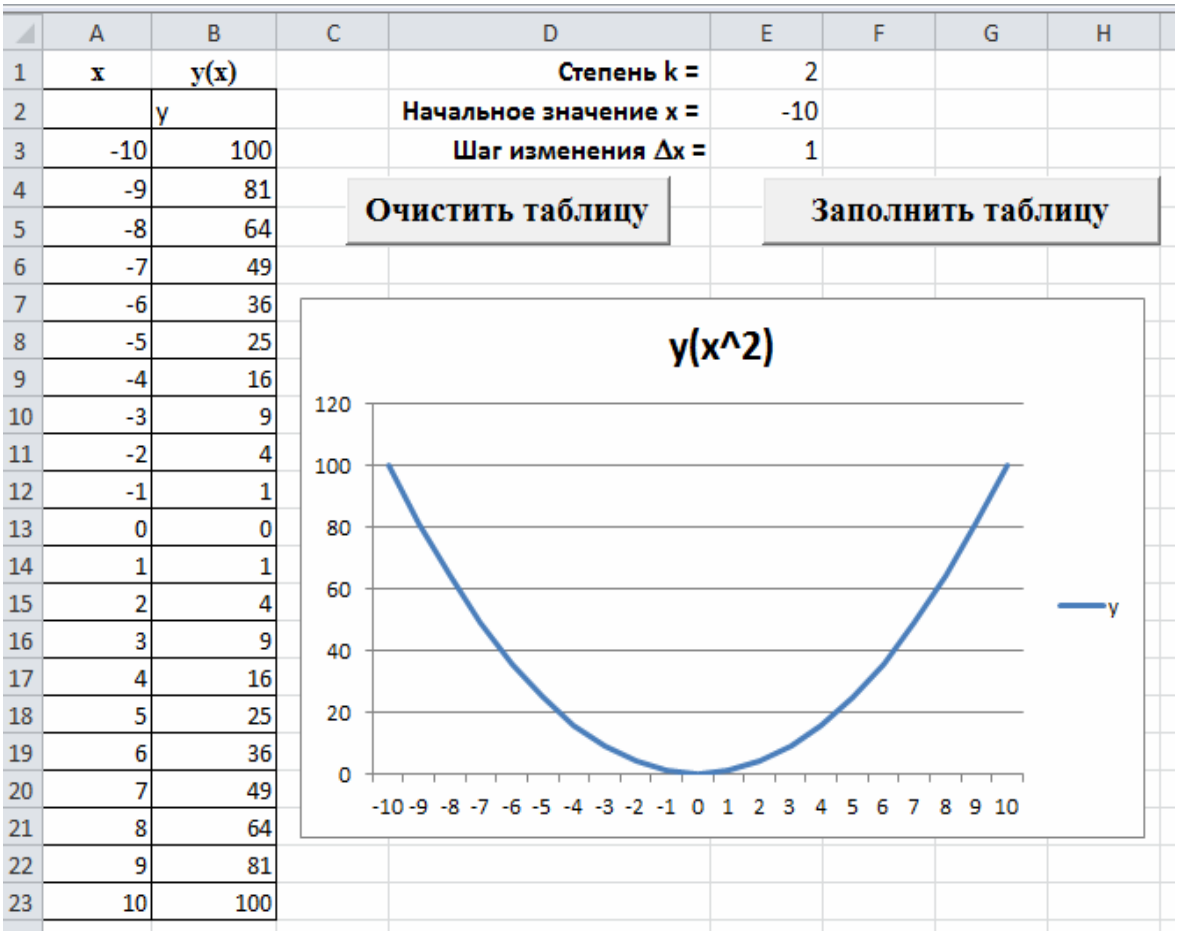


Рис. 2.32

2.7. Дополнительные возможности по созданию автоматизированных приложений в Microsoft Word

Прежде чем начать создавать автоматизированное приложение с использованием VBA, следует определить степень необходимой автоматизации и выработать определенный подход к дизайну элементов проекта.

Если документ должен быть определенным образом отформатирован или иметь заданные параметры страницы, то это удобнее осуществить средствами самого Word.

Если требования к вводимым в документ данным достаточно строги (должны выбираться из определенного списка, располагаться в определенном мести на листе документа и т.д.), то удобно применять пользовательские формы и соответствующие им VBA-программы.

После формулировки требований к приложению приступают к созданию *шаблона*, который определит требуемое форматирование, параметры страницы, а также будет содержать коды макрокоманд

и необходимые пользовательские формы. Начав с обычного документа или документа, основанного на похожем шаблоне, вставить в новый шаблон необходимые рисунки, таблицы и другие элементы.

Рассмотрим пример разработки документа-шаблона в среде Microsoft Word, предназначенного для добавления в документ рамки со штампом (для первого листа текстовых технических документов) и заполнения штампа из специально разработанной пользовательской формы с соответствующими элементами управления [10]. После заполнения формы файл требуется переименовать, чтобы оставить файл документа-шаблона без изменений.

Среда проектирования VBA предлагает удобный способ разработки собственных диалоговых окон, которые позволяют ввести текст в документ сразу в необходимом формате. Это дает возможность пользователю не заботиться о правильном положении текста в документе или использовать переменные документа для сохранения текста. Используя различные списки, переключатели, кнопки и другие управляющие элементы пользовательских форм, можно оказывать также информационную помощь пользователю.

Можно написать свою программу так, что прежде чем допустить пользователя к следующему этапу работы, будет проверяться, например, правильность введенных данных и использованных параметров. Благодаря наличию хорошо разработанных пользовательских форм, ввод данных будет происходить гораздо быстрее.

1. Создать в текстовом редакторе Microsoft Word файл с поддержкой макросов под именем «Формат А4.docm». Установить в диалоговом окне *Параметры страницы* следующие значения полей:

– левое – 2 см;

– правое – 0,5 см;

– верхнее и нижние – минимально возможные.

Затем с помощью таблицы создать в Microsoft Word рамку со штампом заданных размеров и вписать в соответствующие ячейки неизменяемый текст (рис. 2.33).

2. Пометить ключевые области документа, установив *закладки*. Закладки дают возможность VBA-программе быстро и безошибочно переходить к данной части документа. Такой подход обеспечивает быстрое выполнение различных действий в выделенной области, например, ввод текста, применение различных команд форматирования или перемещение точки вставки или курсора в то место, где пользователь должен ввести текст.

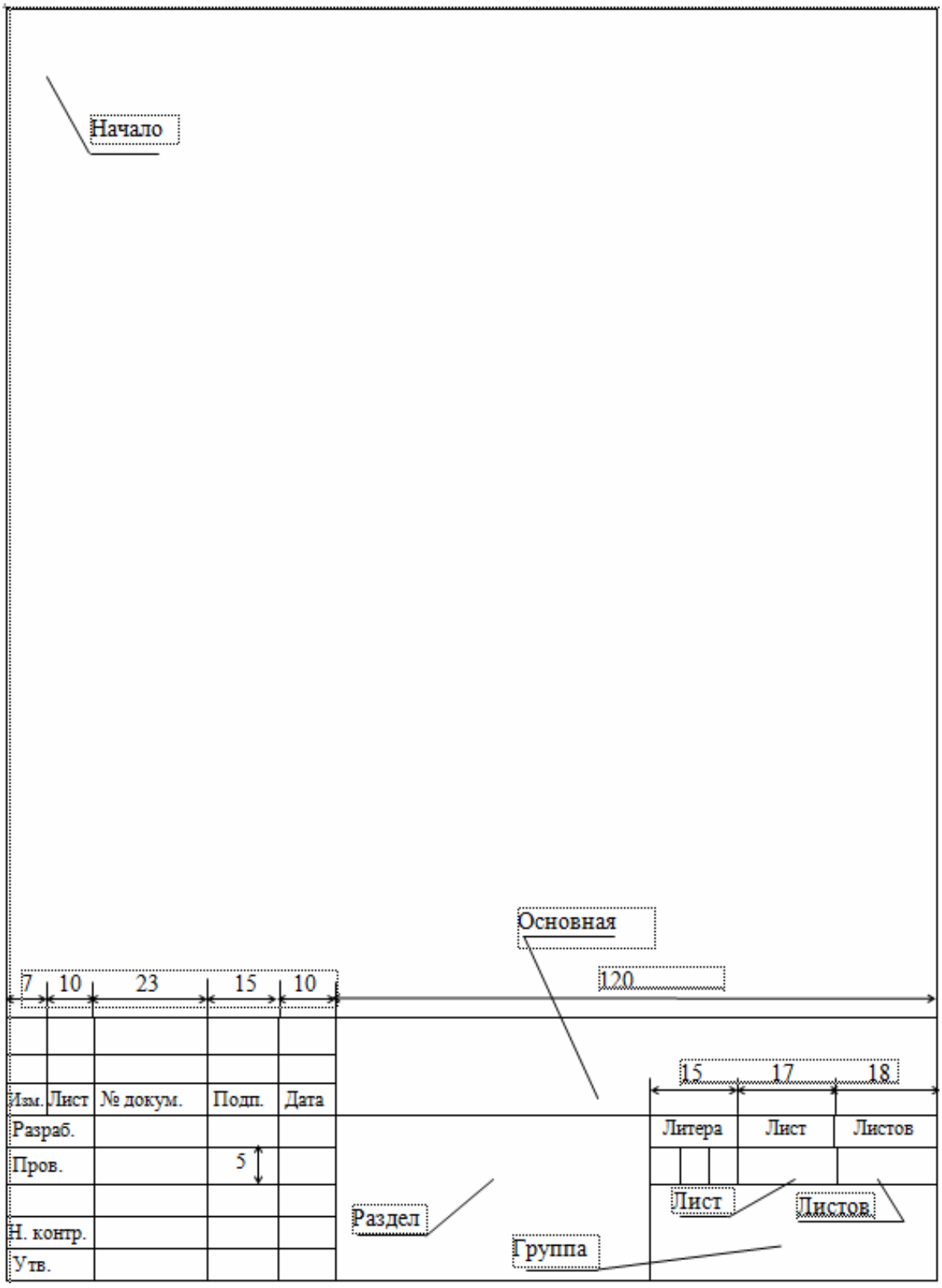


Рис. 2.33

Для создания закладки в шаблоне необходимо выполнить следующие действия:

– установить курсор в место для закладки;

– в меню *Вставка* выбрать команду *Закладка*;

– ввести имя закладки в поле *Имя закладки* и щелкнуть на кнопке *Добавить* (рис. 2.34).

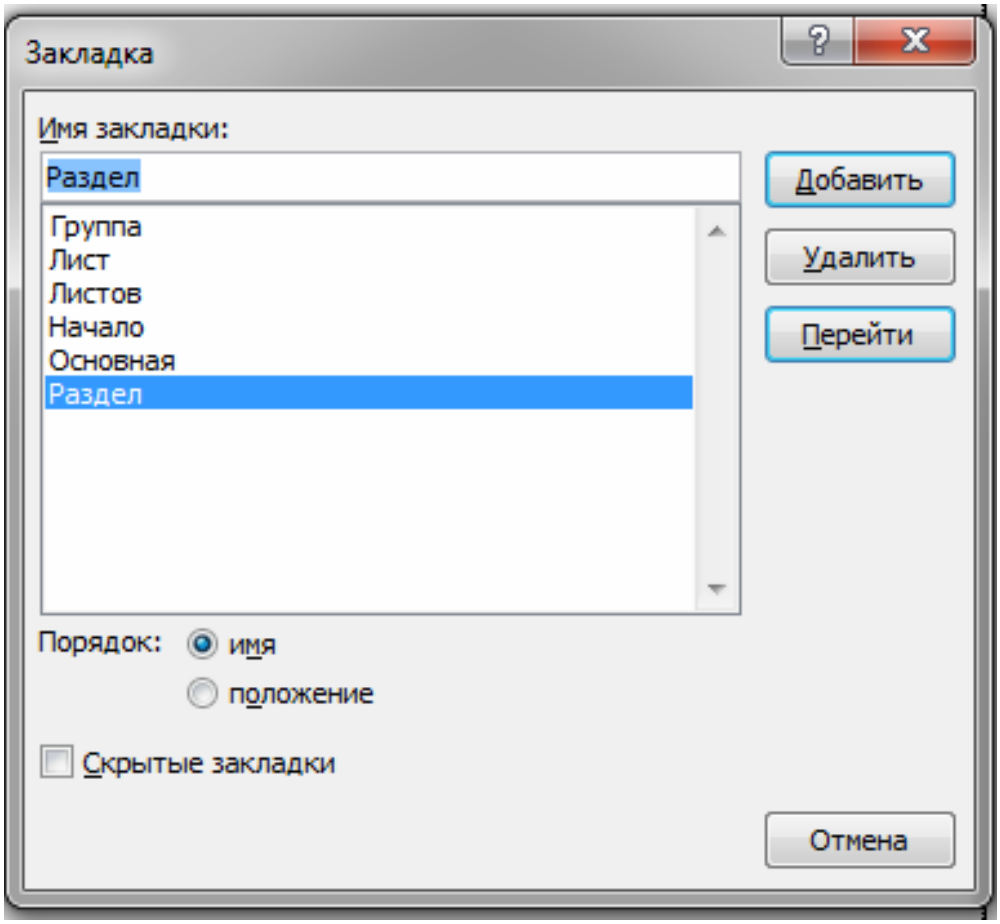


Рис. 2.34

Добавить в шаблон следующие закладки:

Основная – указывает место, куда вводится основная надпись;

Раздел – указывает место, куда вводится название раздела;

Лист – указывает место, куда вводится номер листа;

Листов – указывает место, куда вводится количество листов;

Группа – указывает место, куда вводится номер группы;

Начало – указывает место, куда по завершении макроса будет помещена точка вставки. Именно здесь должен начать вводиться текст технического документа.

Установить на месте всех закладок параметры шрифта и абзаца, указанные в табл. 2.9.

3. Прежде чем приступать к разработке пользовательской формы, которая будет определять интерфейс нашего проекта, необходимо продумать, какого рода информация должна быть выведена на

нее и определить, какие элементы управления процессом создания документа можно предоставить пользователю для заполнения штампа.

Таблица 2.9

Закладка	Размер шрифта	Вариант выравнивания
Основная	*20*	*По центру*
Раздел	*15*	То же
Лист	*10*	*–"–*
Листов	*10*	*–"–*
Группа	*15*	*–"–*
Начало	*15*	*По ширине*

Перейти в редактор IBA, вставить в проект *Project (Формат А4)* форму *UserForm1* и разместить на ней четыре надписи *Label1 – Label4* и одну кнопку *CommandButton1* (рис. 2.35).

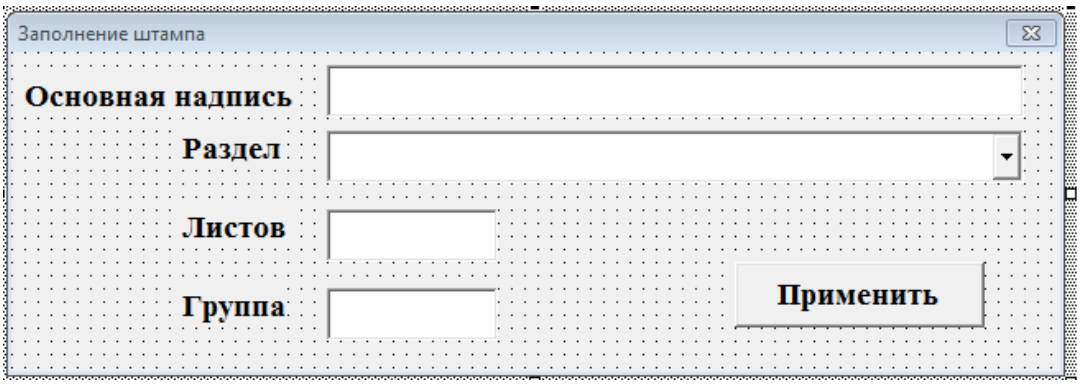


Рис. 2.35

Задать для вставленных в проект объектов значения свойства Caption в соответствии с табл. 2.10.

Таблица 2.10

Класс объектов (свойство *Name*)	Новое значение свойства *Caption*
Форма **(***UserForm1***)**	*Заполнение штампа*
Надпись (Label1)	*Основная надпись*
Надпись (Label2)	*Раздел*
Надпись (Label3)	*Листов*
Надпись (Label4)	*Группа*
Кнопка *(CommandButton1)*	*Применить*

Напротив надписи «Раздел» (см. рис. 2.31) вставить комбинированное поле *ComboBox1*.

Добавить в форму три *текстовых поля* (объекты *TextBox1*, *TextBox2*, *TextBox3*) и задать для них значения свойства *Name* в соответствии с табл. 2.11.

Таблица 2.11

Класс объектов (свойство *Name*)	Новое значение свойства *Name*
Текстовое поле *TextBox1*	*Основная*
Текстовое поле *TextBox2*	*Листов*
Текстовое поле *TextBox3*	*Группа*

4. После разработки документа и формы необходимо создать несколько процедур VBA, связанных с формой.

В модуле *ThisDocument* проекта *Project (Формат А4)* запишем процедуру *Document_Open*, которая выполняется каждый раз, когда открывается документ с заданным шаблоном.

```
    Private Sub Document_Open()
' Формирование списка ComboBox1
    UserForm1.ComboBox1.AddItem "Схема структурная"
    UserForm1.ComboBox1.AddItem _
                "Схема электрическая функциональная"
    UserForm1.ComboBox1.AddItem _
                "Схема электрическая принципиальная"
  UserForm1.Show ' Отображение формы на экране
'Выбор закладки Основная
        ActiveDocument.Bookmarks("Основная").Select
    Selection.TypeText Text:=UserForm1.Основная.Text
        ActiveDocument.Bookmarks("Раздел").Select
    Selection.TypeText Text:=UserForm1.ComboBox1.Text
        ActiveDocument.Bookmarks("Листов").Select
    Selection.TypeText Text:=UserForm1.Листов.Text
        ActiveDocument.Bookmarks("Группа").Select
    Selection.TypeText Text:=UserForm1.Группа.Text
'Выбор закладки Лист
        ActiveDocument.Bookmarks("Лист").Select
'Вставка номера страницы из поля PAGE
    Selection.Fields.Add Range:=Selection.Range, _
        Type:=wdFieldEmpty, Text:="PAGE\*Arabic ", _
        PreserveFormatting:=True
'Переход на закладку Начало
```

ActiveDocument.Bookmarks("Начало").Select
End Sub

Для ячейки таблицы «*Лист*» удобно использовать *поле Page*, которое в Microsoft Word вставляет номер текущей страницы. Для переключения между отображением результата кодов полей и отображением самих кодов нажмите комбинацию клавиш *Alt+F9*; чтобы включить отображение кодов полей нажмите клавиши *Alt+F9* еще раз.

В модуле *UserForm1* проекта *Project (Формат А4)* записать процедуру *CommandButton1_Click*, которая будет выполняться при нажатии на клавишу *Применить*, расположенную в форме:

```
Private Sub CommandButton1_Click()
Dim vvod As Boolean : vvod = True
If Len (Me.Основная.Text) = 0 Then vvod = False
If Len (Me.ComboBox1.Text) = 0 Then vvod = False
If Len (Me.Листов.Text) = 0 Then vvod = False
If Len (Me.Группа.Text) = 0 Then vvod = False
If vvod = True Then
Me.Hide
Else
MsgBox "Введите информацию во все поля формы", 48, _
"Ошибка ввода данных"
End If
MsgBox "Не забудьте сохранить файл под требуемым именем"
End Sub
```

Процедура начинается с объявления переменной *vvod* с начальным значением True. Далее функция *Len* проверяет длину введенных строк в текстовых полях формы. Если хотя бы одно из полей будет пустым, то на экран выводится сообщение, предупреждающее пользователя, что необходимо заполнить форму до конца.

Обратите внимание, что в процедуре используется краткое обращение к форме по умолчанию как «*Me*».

5. На этом процесс создания документа-шаблона завершен. Сохранить окончательный проект полностью в том же *файле «Формат А4.docm»*.

Выйти из Microsoft Word и повторно открыть тот же файл, разрешив загрузку макросов при открытии, и проверить его работу.