МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТРАНСПОРТА»

Кафедра «Информационные технологии»

Н. А. МАРЬИНА, С. А. МАРЬИН

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА VBA

Учебно-методическое пособие

Гомель 2010

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТРАНСПОРТА»

Кафедра «Информационные технологии»

Н. А. МАРЬИНА, С. А. МАРЬИН

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА VBA

Учебно-методическое пособие

Одобрено методической комиссией факультета промышленного и гражданского строительства

Гомель 2010

УДК 004.438(075.8)
ББК 32.973 – 018
М25

Рецензент – канд. техн. наук, доцент кафедры «Информационные технологии»
Ю. А. Пшеничнов (УО “БелГУТ”).

Марьина, Н. А.

М25 Основы программирования на *VBA* : учеб.-метод. пособие / Н. А. Марьина, С. А. Марьин ; М-во образования Респ. Беларусь, Белорус. гос. ун-т трансп. – Гомель : БелГУТ, 2010. – 97 с.
ISBN 978-985-468-611-0

Изложены базовые элементы языка *VBA*, возможности реализации диалога с пользователем (организация ввода – вывода), а также основные алгоритмические конструкции, возможности создания формы с элементами управления, отладочные средства и возможности для настройки интегрированной среды разработки *IDE* редактора *VBA*.

Предназначено для студентов технических специальностей, осваивающих основы программирования на объектно-ориентированном языке *Visual Basic for Applications*, который позволяет программировать почти во всех приложениях фирмы *Microsoft*.

УДК 004.438 (075.8)
ББК 32.973 – 018

ISBN 978-985-468-611-0

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Visual Basic for Application (VBA) – это объектно-ориентированный язык, предназначенный для визуального проектирования программ, приложений и сложных документов. *VBA* доступен из любого приложения фирмы *Microsoft*, например, *MS Word*, *MS Excel* и т. д., и составляет платформу для создания сложных интерактивных документов. Например, можно создать документ, который содержит не только текст, но и программы, которые вычисляют данные для этого документа, автоматически вычисляют и строят графики и таблицы по введенным данным.

Помимо традиционной технологии программирования, заключающейся в использовании переменных, массивов, ветвлений и циклов, для реализации сложных алгоритмических задач *VBA* поддерживает технологии структурированного, модульного, событийного, объектно-ориентированного и визуального программирования. Рассмотрим вкратце особенности каждой технологии и те преимущества, которые предоставляет каждая технология при решении алгоритмических задач.

Традиционное программирование представляет любую программу как "черный ящик", на вход которого поступают входные данные, а на выходе – результат решения задачи. Между входом и выходом стоят операторы обработки данных (операторы ввода – вывода), операторы присваивания и операторы управления вычислительным процессом.

Технология **структурированного программирования** подразумевает программирование без использования оператора `GOTO` и представлена в *VBA* набором структурированных операторов управления вычислительным процессом (операторы ветвления и операторы цикла). Использование этой технологии позволяет писать легко читаемые программы и обеспечить простоту сопровождения и модернизации программ.

Модульное программирование предлагает для упрощения структуры сложных программ использовать независимые модули, которые могут содержать произвольное число программ и данных. Разделение программ и данных на модули можно и удобно строить по функциональному признаку. Некоторые из этих программ можно вызывать из других модулей (это общие переменные и программы), а некоторые можно использовать только внутри модуля. В модульном программировании введено понятие видимости программ и данных. Если программа или данные должны быть видимы из других программ других модулей, то этим программам или данным присваивается атрибут `Public`, в противном случае – атрибут `Private`. Про-

граммы или данные с атрибутом `Private` видимы только в том модуле, где они определены и недоступны в других модулях. Использование модульного программирования позволяет ввести иерархию в программный продукт и скрыть от пользователя несущественные детали реализации программ и улучшить переносимость программ и алгоритмов.

Концепция **технологии событийного программирования** предполагает наличие программных возможностей оперативного отслеживания событий, исходящих от пользователя программы или других устройств, которые подключены к программе (компьютеру). Эта технология позволяет создавать гибкие и мощные программные средства для решения сложных задач при тесном взаимодействии с пользователем программы. Эта технология предполагает наличие в языке программирования *VBA* специальных программ, которые привязаны к событиям, генерируемым операционной системой.

Объектно-ориентированное программирование рассматривает все предметы "реального" (компьютерного) мира как объекты. Объект – это предмет (одушевленный или неодушевленный), который обладает набором свойств (характеристик) и методов (действий). Свойство – это качество или характеристика объекта. Например, для самолета основными свойствами являются мощность двигателя, максимальная грузоподъемность, максимальная высота полета или максимальная скорость. **Метод** – это действие, которые они могут выполнять над объектом. Например, самолет может взлетать, летать и приземляться. Если принять аналогию с конструкцией предложения на любом языке, то **свойства** – это существительные (переменные), а методы – это глаголы (программы). Набор свойств и методов каждого объекта, в общем случае уникален. Используя свойства и методы, мы можем, например, написать программу, которая управляет полетом самолета.

Технология визуального программирования предполагает наличие специальной среды, которая позволяет проектировать (размещать) элементы управления на форме проектируемого приложения при помощи визуальных средств, предоставляемых интегрированной средой разработки (*IDE – Integrate Design Environment*). Характерной чертой *IDE* является использование мыши для размещения элементов управления на форме и задание их свойств, а также предоставление помощи при редактировании свойств и методов существующих объектов. Цель этой технологии – дружественность на основе *Windows*-интерфейса для создания гибких программ обработки, ввода и вывода данных.

1 БАЗОВЫЕ ЭЛЕМЕНТЫ ЯЗЫКА VBA

1.1 Алфавит, словарь, идентификаторы

Для написания программы на каком-либо языке программирования необходимо знать алфавит языка, который формируется на основе конечного набора знаков. Алфавит языка *VBA* образуют: прописные и строчные буквы латинского алфавита; десятичные цифры; специальные символы: “ _ ”, “+”, “-”, “*”, “/”, “=”, “>”, “<”, “[”, “]”, “(”, “)”, “.”, “!”, “?”, “,”, “:”, “;”, “ ‘ ”, “#”, “&”, “$”, “^”, “\”, “%”, “ ”. Комбинации специальных символов образуют составные символы: “<>”, “<=”, “>=”, “. .”, “: =”.

Знаки алфавита используются для составления слов – неделимых последовательностей знаков алфавита, отделенных друг от друга разделителями и образующими тело программы на языке программирования. Для разделения слов используются пробел, комментарии или символ конца строки.

Различают две группы слов – ключевые слова и идентификаторы пользователя. *Ключевые слова* являются составной частью языка, имеют фиксированное представление и определенный смысл. К ним, как к элементам языка *VBA*, относятся имена инструкций, типов данных, методов, свойств, операторов, встроенных констант, объектов и стандартных функций. *Идентификаторы пользователя* применяются для обозначения переменных, констант, процедур, функций, различных объектов, определенных пользователем.

При выборе идентификаторов целесообразно руководствоваться следующими правилами:

– длина имени не должна превышать 255 символов;

– имя может содержать любую комбинацию букв, цифр и допустимых символов, начинающуюся с буквы;

– допустимыми символами являются знак подчеркивания “_”, который может быть размещен в любом месте идентификатора, кроме первого, единичные знаки %, !, $, &, @, которые могут располагаться только в конце идентификатора;

– имена должны быть уникальны внутри области, в которой они определены;

– имена пользователя не должны совпадать со служебными словами VBA и именами встроенных функций и процедур. Например, допустимые имена: `X15, FIO, Nomer_Zakaza, A%, B@`; недопустимые имена: `15x, Ф.И.О., № Tel, A%_1, B@@.`

Кроме этого, имя должно быть не только правильным, но и понятным, т. е. отражать назначение объекта.

Поскольку регистр букв не имеет значение, то лучше пользоваться такими именами `NomToma, Nom_Zayavki`. Например, выражение `R=T/M` станет более понятным, если его заменить:

```
Result=Tek_trafik/Max_trafik.
```

Идентификаторы *VBA* не зависят от состояния регистра: написание идентификатора прописными или заглавными буквами не имеет значения, например, идентификаторы пользователя `Name` и `name` или `Sin` и `sin` для *VBA* идентичны. Если же переменная объявлена явно, то все обращения к идентификатору переменной преобразуются в соответствии с типом ее объявления.

1.2 Типы данных

Понятие типа является одним из фундаментальных понятий любого языка программирования. Каждый элемент данных или объект (константа, переменная, выражение, функция), которым оперирует программа, относится к определенному типу.

Тип данных определяет: множество значений, которые могут принимать объекты программ, совокупность операций, допустимых над этими объектами, и объем выделяемой памяти и форму представления данных в ней.

1.2.1 Классификация типов данных

Существует несколько критериев классификации типов данных. Например, типы данных можно разделять на *простые* и *сложные* в зависимости от того, как устроены их данные. У простых (скалярных) типов возможные значения данных едины и неделимы. Сложные типы характеризуются способом структуризации данных, – одно значение сложного типа состоит из множества значений данных, организующих сложный тип. Есть и другие критерии классификации типов. Так, типы разделяются на *встроенные* типы и типы, *определенные программистом* (пользователем). *Встроенные* типы изначально принадлежат языку программирования и составляют его базис. В основе системы типов любого языка программирования всегда лежит базисная система типов, встроенных в язык. На основе *встроенных* типов программист может строить собственные, им определенные типы данных. Но способы (правила) создания таких типов являются базисными, встроенными в язык.

Типы данных разделяются также на *статические* и *динамические* типы. Для данных *статического* типа память отводится в момент объявления, требуемый размер данных известен при их объявлении. Для данных *динамического* типа размер данных в момент объявления не известен и память им выделяется динамически в процессе выполнения программы по запросу.

1.2.2 Тип Variant

В общем случае отметим, что среди языков программирования выделяются два крайних случая – строго типизированные и бестиповые языки. В первом случае, каждая переменная имеет строго фиксированный в момент объявления тип и он не может изменяться в процессе выполнения програм-

мы. Такие языки являются более надежными, поскольку обеспечивают жесткий контроль типов и позволяют обнаруживать ошибки еще на стадии компиляции программы. Классическим примером является язык Паскаль. Бестиповые языки – это скорее языки с одним единственным типом. В таких языках одна и та же переменная по ходу программы может хранить данные фактически разных типов. Тип `Variant` языка *VBA* является примером такого обобщающего типа. Бестиповые языки обеспечивают более быстрое исполнение программ, предоставляют программистам большую гибкость, – это достигается за счет надежности программ.

Таким образом, в *VBA* можно обрабатывать числа, строки, логические значения, даты и объекты, использовать обобщающий тип данных – тип `Variant`, а также создавать собственные типы данных. Встроенные простые типы данных представлены в таблице 1.1. В *VBA* имеется также возможность создать определяемый пользователем тип данных с помощью инструкции `Type`.

Тип `Variant`, использованный для описания некоторой переменной, позволяет присваивать и обрабатывать с его помощью данные разных типов. С одной стороны это хорошо: не надо помнить об ограничениях на множество допустимых значений и операций. Но это может вызвать ошибки, а также значительно увеличить время обработки переменных и необходимые для выполнения программы ресурсы памяти.

Отметим, что переменные типа `Variant` могут получать значения любого типа в зависимости от контекста. Кроме того, они могут принимать и некоторые специальные значения:

- `Empty` – переменная не была инициализирована;
- `NULL` – данные ошибочны;
- `ERROR` – значение содержит код ошибки, который может быть использован для ее обработки в программе;
- `Nothing` – переменная типа `Object` ни на что не ссылается: связь между ней и конкретным объектом прервана или не установлена.

1.2.3 Простые типы данных и область видимости переменных

При объявлении переменной определяется ее тип и область видимости – область, где имя переменной видимо, и значит, возможен доступ к ее значению. Переменные можно объявлять на двух уровнях – *процедуры* и *модуля*. Для объявления переменных используются операторы `Dim`, `Public`, `Private` и `Static`. Первый можно использовать на обоих уровнях, `Public` и `Private` – на уровне модуля, `Static` – только на уровне процедуры.

Переменные, объявленные на уровне процедуры, называются **локальными** по отношению к данной процедуре. Их областью видимости является

только та процедура, в которой они объявлены. Переменные уровня модуля являются **глобальным**. Они объявляются в разделе `Declarations`, который есть у каждого модуля. Область видимости глобальных переменных может распространяться:

– на все процедуры одного модуля, в котором они объявлены; такие глобальные переменные, называемые закрытыми (`Private`), должны быть объявлены на уровне модуля, либо оператором `Private`, либо оператором `Dim`;

– весь программный проект – все процедуры всех модулей данного проекта; такие глобальные переменные, называемые открытыми (`Public`), должны быть объявлены оператором `Public`.

Локальные переменные уровня процедуры могут быть объявлены оператором *`Static`*, что делает их статическими. У таких переменных увеличивается время жизни. Обычные локальные переменные рождаются при входе в процедуру, видимы только в ней и "умирают", выходя из процедуры. Это значит, что память под переменные отводится при входе в процедуру, а при выходе она освобождается. Область видимости статических переменных по-прежнему – процедура, но время жизни иное, так как у них не отбирается память при выходе, – она просто становится временно недоступной. Поэтому при повторном входе в процедуру статические переменные восстанавливают те значения, что у них были при последнем выходе. Статические переменные – это хранители информации между многократными вызовами одной и той же процедуры. Чтобы статические переменные имели смысл, необходима первоначальная инициализация переменных, т. е. они должны иметь хоть какие-нибудь значения уже при первом вхождении в процедуру.

Таблица 1.1 – **Описание простых типов данных языка VBA**

Тип данных	Описание	Символ	Размер, байт	Диапазон значений
`Byte`	Байт	–	1	От 0 до 255
`Boolean`	Логический	–	2	`True` или `False`
`Integer`	Целое число	%	2	От – 32 768 до 32 767
`Long`	Длинное целое число	&	4	От – 2 147 473 648 до 2 147 483 647
`Single`	Число с плавающей точкой обычной точности	!	4	От – 3,402823E38 до – 1,401298E – 45 для отрицательных чисел; от 1,401298E – 45 до 3,402823E38 для положительных чисел

Окончание таблицы 1.1

Тип данных	Описание	Символ	Размер, байт	Диапазон значений
`Double`	Число с плавающей запятой двойной точности	#	8	От –1,79769313486232E 308 до – 4,9406564584124E–324 – для отрицательных значений; от 4,94065645841247E –324 до 1,79769313486232E 308 – для положительных значений
`Currency`	Денежный	@	8	От – 922337230685477,5808 до 922337203685477,5807
`Date`	Дата и время	–	8	От 01. 01. 100 до 31. 12.9999
`String`	Строка	$	1 на каждый символ	От 0 до 65535 символов
`Variant`	Числовые подтипы	–	16 + N байт (согласно типу)	Переменная любого числового типа
`Object`	Объект	–	4	Ссылка на объект (указатель)

1.3 Синтаксис объявления переменных

Переменными называются элементы данных, значения которых при выполнении программы могут принимать различные значения в соответствии с указанным типом.

1.3.1 Явное объявление переменных

Для явного объявления переменных обычно используется инструкция объявления `Dim`:

```
Dim имяПеременной [As тип] [, имяПеременной [As тип]
```

Элементы синтаксиса:

`имяПеременной` – обязательный элемент, определяющий имя переменной, удовлетворяющее стандартным правилам именования переменных;

`тип` – необязательный элемент, определяющий тип данных объявляемой переменной. Для каждой описываемой переменной следует использовать отдельное предложение `As тип`. В случае отсутствия параметра переменная будет иметь тип `Variant`.

Например, инструкции

```
Dim A As Integer
Dim B As Single
Dim C As Boolean
Dim Str As String*5
```

описывают переменные `A` – целого, `B` – вещественного и `C` – логического типа, `Str` – строка из пяти символов.

Это же объявление переменных можно выполнить с помощью одной инструкции `Dim`:

```
Dim A As Integer, B As Single, C As Boolean, Str As String*5
```

а в инструкции

```
Dim A, B, C As Integer
```

только переменная `C` – целого типа, а `A`, `B` – типа `Variant`.

При объявлении любой переменной *VBA* в памяти выделяется область, размер которой определяется в соответствии с типом объявляемой переменной. Выделенная область памяти связывается с идентификатором объявляемой переменной. При обращении к переменной фактически происходит обращение к области памяти, где хранится текущее значение переменной. Переменная инициализируется, т. е. в соответствии с типом объявляемой переменной ей присваивается начальное значение. Число инициализируется значением 0, переменные `Boolean` – значением `False`, строки – пустыми строками (не содержат символов).

Объявить переменную можно в любом месте программы, но обязательно до ее использования. Однако правила хорошего стиля программирования все-таки предписывают делать это в самом начале программы.

Переменные, описанные с помощью инструкции `Dim`, называются *явно описанным.* Переменные являются *неявно описанными*, если они используются в программе без объявления их с помощью инструкции `Dim` или в инструкции объявления типа переменной был опущен параметр *тип*. Неявно описанные переменные связываются с типом `Variant`, а затем они используются как любые другие переменные этого типа. Однако при этом, во-первых, потребуется больше ресурсов памяти по сравнению с другими типами данных и времени, т. к. компилятор вынужден сначала определить настоящий тип переменной, затем преобразовать его к этому типу, и только потом использовать в вычислениях. При работе с достаточно большими программами это может привести к значительной потере во времени и ресурсах. Во-вторых, не описывая переменные явно, можно получать неправильные ответы или ошибки при выполнении программы (зацикливание и др.).

С переменной будет связан тип `Variant` и тогда, когда при объявлении переменной была допущена ошибка.

Чтобы избежать части ошибок, *VBA* позволяет наложить требование на явное описание всех переменных в модуле. В этом случае при использовании необъявленной переменной во время компиляции генерируется ошибка, которую можно легко исправить.

Чтобы наложить требование на явное описание переменных, необходимо в разделе описаний модуля указать директиву компилятора `Option Explicit`, например:

```
Option Explicit
  Sub Primer()
      Dim K As Byte
      K=K+10
  End Sub
```

1.3.2 Объявление переменных по умолчанию

Отметим также, что в *VBA* есть средства, позволяющие не объявлять переменную явно, но устанавливать ее тип по первому или последнему символу имени переменной. Имена переменных *VBA* могут заканчиваться специальным символом, позволяющим установить тип этой переменной (колонка «символ» в таблице 1.1).

Например,

```
Message$ = txtDisplay.Text
IntVar% =5
```

Означает определение значения переменной `Message$` строкового типа, присваивание переменной целого типа `IntVar%` значения 5.

Есть еще одна возможность определения типа по первой букве имени. С этой целью в язык введена серия операторов `DefType` (по одному на каждый тип `DefBool`, `DefInt` и т. д.), определяющих свой диапазон букв для каждого типа. Если первая буква имени необъявленной переменной входит в тот или иной диапазон, ей приписывается соответствующий тип. Эти операторы устанавливаются на уровне модуля и действуют на все его процедуры. Концевой символ установления типа сильнее, чем `DefType`, а тот сильнее стандартного "умолчания" `Variant`. Но заметим, что данные варианты объявления типов в некоторой степени архаичны, и не рекомендуются при современном стиле программирования.

1.4 Описание констант

Константами называются элементы данных, значения которых определены при их описании и в процессе выполнения программ не изменяются.

В *VBA* существуют константы двух типов:

1 *Литералы* – константы, определяемые их значениями и используемые в выражениях. Различают числовые (3.14; 16; 4.2E+02), строковые (“ГЭФ”; “Иванов И.И.”) и логические константы (`True`, `False`).

2 *Именованные*, имеющие собственные уникальные идентификаторы, среди которых различают *встроенные (стандартные) константы*, их имена и количество определяются используемым приложением, например, к таким константам относятся `vbOKOnly`, `fmAltMask` и т. д. и *пользовательские константы, которые* объявляются самим пользователем с помощью инструкции `Const`. В момент объявления таким константам присваиваются значения. Попытка переопределения значения константы с помощью оператора присваивания вызывает ошибку.

Сокращенный **синтаксис** инструкции `Const`:

```
Const ИмяКонстанты [As тип] = выражение
```

Элементы синтаксиса:

`ИмяКонстанты` – обязательный параметр, удовлетворяющий стандартным правилам именования;

`тип` – необязательный параметр. Связывает объявляемую константу с одним из поддерживаемых типов данных. Для каждой описываемой константы следует использовать отдельное предложение `As` тип. В случае отсутствия параметра тип константы определяется типом *выражения*;

`выражение` – обязательный параметр, литерал (числовое, строковое или логическое значение), другая константа или любое сочетание, которое включает арифметические и логические операторы.

Например, следующие инструкции:

```
Const Year As Integer=2003
Const Plan%=129
```

объявляют две целочисленные константы: `Year` и `Plan`. Первая константа имеет явно описанный целый тип, а тип второй определяется ее именем, символ ’%’ в конце имени.

Инструкция

```
Const NameGrup = "ПС - 21"
```

осуществляет объявление строковой константы `NameGrup`, которой присвоено значение строкового литерала `"ПС - 21"`.

Логическая константа `Flag`, с которой связывается значение `True` (Истина), может быть объявлена следующим образом:

```
Const Flag = True
```

Как и переменные, именованные константы можно объявлять на уровне процедуры или модуля. В первом случае используется только ключевое слово `Const`, во втором – дополнительно можно задать спецификаторы `Public` или `Private`, позволяющие объявить константу общей для всех модулей или закрытой. По умолчанию глобальные константы имеют статус `Private`. У локальных констант, объявленных в процедурах, областью видимости является процедура. Если не определить тип константы, он будет восстановлен по значению константного выражения.

Встроенных констант огромное количество. Есть, например, встроенные константы, связанные с тем или иным приложением, например, *Excel* или *Word*. Обычно они имеют соответствующий префикс (`xl`, `wd` и т. д.). Но есть и встроенные константы самого *VBA*. Они позволяют задавать тип календаря, дни недели, цвет, клавиши и могут применяться в конкретной ситуации, скажем, при работе с окном сообщений. Большинство таких констант начинается с префикса `vb`, например: `vbWhite`, `vbSunday`, `vbKeyEscape`, `vbOKOnly`.

1.5 Разделы объявлений

В классическом языке программирования Паскаль, созданном Николасом Виртом, раздел объявлений имел четкую структуру и, в свою очередь, делился на разделы, содержащие объявления констант, типов, процедур, переменных. В языке *VBA* такого формального разделения нет, поэтому возможно, что объявление переменных пользовательского типа предшествует объявлению самого типа. Поэтому основная рекомендация хорошего стиля программирования состоит в том, что необходимо поддерживать структуру раздела объявлений и разделять объявления констант, типов и переменных. Выделим основные части раздела объявлений:

- опций;
- констант;
- типов;
- переменных;
- `Declare`.

1.5.1 Раздел опций

Опции являются указаниями для транслятора. Они могут задаваться только на уровне модуля и должны начинать раздел объявлений. Заметьте, это не пожелание, а синтаксическое требование. Опции задаются ключевым словом `Option`, после которого идет имя опции и возможно параметры. Рассмотрим и охарактеризуем все возможные опции.

`Explicit.` При указании этой опции транслятор требует, чтобы все переменные модуля были явно описаны. Правильно включить эту опцию раз и навсегда в опциях редактора *VBA*.

`Base`. Эта опция имеет два значения: 0 и 1, указывающие нижнюю границу индекса массивов, задаваемую по умолчанию. Рекомендуется не пользоваться этой опцией, всегда явно указывать нижнюю границу индекса массивов.

`Private`. Эту опцию, достаточно поместить в один из модулей проекта, обычно, в главный модуль проекта, который неявно всегда выделяется программистом. При ее задании проект делается закрытым и недоступен для других проектов в системе документов.

`Compare`. Опция говорит транслятору, как он должен выполнять сравнение строк в процедурах модуля. Параметр опции может принимать одно из трех возможных значений: {`Binary`|`Text`|`DataBase`}.

По умолчанию *VBA* применяет **метод `Binary`**, при котором строки сравниваются по внутренним кодам соответствующих символов. В ОС семейства *Windows* порядок сортировки определяется кодовой страницей. Вот часть типичного такого порядка: `A < Z < a < z < A < Я < a < я.`

Метод сравнения `Text` основан на сравнении, не чувствительном к регистру, так что при сравнении заглавные и строчные буквы не различаются. Для тех же символов порядок при этом сравнении будет такой:

`(A = a)<(Z = z)<(A = a)<(Я = я)`

Метод `DataBase` допустим лишь при работе с *MS Access*. Сравнение при этом базируется на порядке, задаваемом локализацией той базе данных, для которой проводится сравнение.

1.5.2 Разделы констант, типов и переменных

Основные разделы объявлений уже рассмотрены. Однако отметим, что разумно создавать такие разделы явно, отделяя комментариями один раздел от другого в описанном выше порядке. Тогда уже объявленные константы будут появляться при описании границ массивов, типы переменных будут предшествовать объявлению самих переменных.

Для констант и типов ключевые слова `Const` и `Type` однозначно определяют описываемый объект. Для переменных ключевые слова могут быть разными. Заметим, что для переменных уровня модуля не целесообразно использовать ключевое слово `Dim`, лучше использовать спецификаторы `Public` или `Private`, явно указывающие область действия переменной.

1.5.3 Раздел Declare

Этот раздел появляется в тех случаях, когда модули проекта используют динамически присоединяемые библиотеки – `DLL`. Если `DLL` имеет библио-

теку типов TypeLib и она доступна проекту, то нет необходимости описывать компоненты библиотеки, они будут найдены автоматически. Но если TypeLib недоступна или не определена, то в этой ситуации каждая из функций и процедур библиотеки, вызываемая в модуле, должна быть предварительно описана специальным оператором Declare. Вот его синтаксис:

```
[Public | Private] Declare {Sub | Function} имя Lib
"имя_библиотеки" [Alias "псевдоним"] [([параметры])]
[As возвращаемый_тип]
```

Здесь указывается имя библиотеки, имя процедуры или функции, возможный псевдоним, параметры и возвращаемое значение для функций. Подробно о DLL, операторе Declare с примерами на эту тему изложено в литературе [3–5].

1.5.4 Правила именования

Есть несколько простых правил, которые следует выполнять при написании и оформлении текста программ:

- использование комментариев;
- соблюдение правил именования;
- соблюдение структуры текста;
- соблюдение "подъемного" размера модулей в программе.

Основное правило именования констант и переменных состоит в следующем: имя должно отражать содержательный смысл, и состоять из одного или нескольких слитно написанных слов, каждое из которых начинается с большой буквы.

Разработчики от *Microsoft* рекомендуют придерживаться более строгих правил. Имя должно отражать не только смысл, но и тип переменной и ее область действия. Поэтому имя должно состоять из префикса и собственного имени. Префикс также является составным, две его части отражают область действия и тип переменной. В таблицах 1.2–1.4 показаны возможные значения префикса.

Таблица 1.2 – **Префикс, задающий область действия**

Первая часть префикса	Область действия
g	Global – весь проект
m	Module – для Private переменных модуля отсутствует
p	Procedure – для локальных переменных

Таблица 1.3 – **Префикс, задающий тип переменной**

Вторая часть префикса	Тип переменной
str	String
int	Integer
byt	Byte
lng	Long
sng	Single
dbl	Double
cur	Currency
var	Variant
obj	Object
bln	Boolean

Вот несколько примеров, построенных по такому принципу имен: `gstrOneWord`, `mintNumberOne`, `strAnswer`, `curSalary`. Так же, как по значению константы можно восстановить ее тип, по правильно построенным именам можно однозначно восстановить их объявление. Например:

```
Public gstrOneWord As String
Private mintNumberOne As Integer
Dim strAnswer As String
Dim curSalary As Currency
```

Согласно этим же рекомендациям имена констант следует строить только из заглавных букв. Если имя константы состоит из нескольких слов, то для их объединения используется знак подчеркивания, например: `MY_DIRECTORY_PATH`. Заметим, что при построении констант, начиная с *Office* 2000, используется префикс, указывающий, какому из приложений принадлежит константа.

Таблица 1.4 – **Префиксы констант**

Приложение	Префикс констант
Access	ac
Excel	xl
FrontPage	fp
Office	mso
OfficeBinder	bind
Outlook	ol
Power Point	pp
Word	wd
VBA	vb

Префиксы следует использовать и при именовании объектов, для форм обычно используется префикс `frm`, для командных кнопок – `cmd` и т. д. Следует понимать, что если уж пользоваться префиксами, то общеупотребительными, не следует заниматься самодеятельностью в этом вопросе.

1.6 Операции и операторы VBA

В программах на *VBA* можно использовать стандартный набор операций над данными. Для обозначения различных операций используются соответствующие операторы. Например, оператор "+" выполняет операцию сложения двух чисел или выражений, являющихся операндами. Большинство операций *VBA* требуют наличия двух операндов. Все операции *VBA* можно разделить на следующие группы:

- арифметические;
- отношения;
- конкатенации (сцепления);
- логические.

Им соответствуют группы операторов с аналогичными названиями.

Арифметические операторы используются для выполнения математических вычислений с операндами числовых типов. Примеры использования арифметических операторов представлены в таблице 1.5.

Таблица 1.5 – **Примеры использования арифметических операторов**

Выражение	Операция	Пример		
		A	B	Результат
`A + B`	Сложение	5	2,75	7,75
`A - B`	Вычитание	5	2,75	2,15
`A * B`	Умножение	2	6	12
`A / B`	Деление	7	2	3,5
`A \B`	Целочисленное деление	7	2	3
`A mod B`	Остаток от деления по модулю	7	2	1
`A ^ B`	Возведение в степень	2	3	8

Операторы конкатенации используются для объединения строковых значений. С их помощью можно формировать строки, состоящие из компонентов различного типа. *VBA* имеет два оператора для реализации операции конкатенации (таблица 1.6). Это операторы «+» и «&». Поскольку оператор

«+» используется и для обозначения операции сложения и конкатенации, то в последнем случае предпочтительнее использовать оператор «&», так как он однозначно определяет требуемую операцию.

Таблица 1.6 – **Операторы конкатенации**

Оператор	Пример	Возвращаемое значение
+	Y = "46" + "8" Y = 46 + 8 Y = "Сегодня " + 1 + " Мая "	"468" – строка 54 – число "Сегодня 1 Мая"
&	Y = "46" & "8" Y = 46 & 8 Y = "Сегодня " & 1 &" сентября "	"468" – строка "468" – строка "Сегодня 1 сентября"

Операторы сравнения (таблица 1.7) позволяют сравнить два выражения. Результатом сравнения являются логические значения `True` (истина) либо `False` (ложь). Например: `a<=N; x<>(2*z-b)/3; i+1>j-5.`

Таблица 1.7 – **Операторы конкатенации**

Оператор	Операция	Пример	Результат
<	Меньше	2 < 5 2 < 2	True False
<=	Меньше или равно	2 <= 2 2 <= 0	True False
>	Больше	5 > 2 2 > 5	True False
>=	Больше или равно	2 >= 2 5 >= 2	True False
=	Равно	2 = 2 2 = 5	True False
<>	Не равно	2 <> 5 2 <> 2	True False

Операндами логического выражения могут быть логические константы, переменные логического типа, отношения. В *VBA* чаще используют 4 логические операции: отрицание – `NOT`, логическое умножение – `AND`, логическое сложение – `OR`, исключающее "или" – `XOR`. Результаты логических операций для различных значений операндов приведены в таблице 1.8. Использованы обозначения: `T` – `True`, `F` – `False`. Примеры логических выражений: `(z+1)<>(x>3) and (y<5); a>0 or b<0.`

Таблица 1.8 – **Результаты логических операций**

Оператор	Операция	Выражение	Значение A	Значение B	Результат
And	Конъюнкция (логическое умножение или логическое И)	A And B	True True False False	True False True False	True False False False
Or	Дизъюнкция (логическое сложение или логическое ИЛИ)	A Or B	True True False False	True False True False	True True True False
Not	Логическое отрицание	Not A	True False		False True
Xor	Исключающее ИЛИ	A Xor B	True True False False	True False True False	False True True False

1.7 Встроенные функции VBA

VBA предоставляет большой набор встроенных функций и процедур, использование которых существенно упрощает программирование. Все функции можно разделить на следующие основные категории: математические функции, функции преобразования и проверки типов, преобразования форматов, обработки строк, времени и даты, финансовые функции.

1.7.1 Основные математические функции

Рассмотрим некоторые математические функции (таблица 1.9).

Таблица 1.9 – **Основные математические функции**

Ключевое слово	Функция	Пример	Примечание
Atn(x)	Арктангенс числа	pi=4*Atn(1)	Вычисление значения π
Cos(x)	Косинус угла	M=1/cos(x)	Вычисление секанса
Sin(x)	Синус угла	M=1/sin(x)	Вычисление косеканса
Tan(x)	Тангенс угла	M=1/tan(x)	Вычисление котангенса
Exp(x)	Число e в степени x	M=Exp(A)	Вычисление e^A
Log(x)	Логарифм натуральный	M=Log(A)	Вычисление Ln(A), A>0
Sqr(x)	Квадратный корень числа	M=Sqr(A)	Вычисление $\sqrt{A}$, A≥0
Abs(x)	Абсолютное значение числа	M=Abs(A)	Вычисление |A|

Рассмотрим старшинство операций в порядке убывания их приоритета, принятого в *VBA*:

- операции в скобках;
- вычисление функции;
- ^ (возведение в степень);
- смена знака;
- *, /, \, `mod`;
- +, –;
- = , >,<,>=,<=,<>;
- `Not`;
- `And`;
- `Or`;
- `Xor`.

Рассмотрим некоторые примеры записи математических выражений в виде арифметических выражений на *VBA* (таблица 1.10).

Таблица 1.10 – **Примеры математических выражений**

Математическое выражение	Выражение на *VBA*
$x^2 - \sin(x) + 6$	`x^2-sin(x)+6`
$\frac{x+4y}{x^2+\sqrt{x}}$	`(x+4y)/(x^2+sqr(x))`
$\sqrt{e^{x+\lvert xy\rvert} + \ln(\lvert x+xy\rvert)}$	`Sqr(Exp(x+Abs(x*y))+Log(Abs(x+x*y)))`

1.7.2 Функции преобразования типов

Функции преобразования типов выполняют преобразование переменной некоторого типа в заданный тип. Использование этих функций способствует устранению ошибок несовпадения типов, а также позволяет обеспечивать явный контроль над преобразованием типов данных в выражениях.

Для преобразования из строки символов в число и числа в его строковое представление можно использовать функции, приведенные в таблице 1.11.

Таблица 1.11 – **Функции преобразования число-строка и наоборот**

Функция	Назначение
`Val` (строка)	Возвращает числа, содержащиеся в строке, как числовое значение соответствующего типа
`Str` (число)	Возвращает значение типа `variant` (`String`), являющееся строковым представлением числа
`Asc` (строка)	Возвращает число кода символа, соответствующее первой букве строки. Буква "А", например, имеет код символа 65

Окончание таблицы 1.11

Функция	Назначение
`Chr` (число)	Возвращает строку из одного символа, соответствующего коду символа (параметр является числом), который может принимать значения от 0 до 255. (`Chr(13)` – символ возврата каретки, `Chr(10)` – символ смещения на одну строку)
`Hex` (число)	Возвращает строку, содержащую шестнадцатеричное представление числа
`Oct` (число)	Возвращает строку, содержащую восьмеричное представление числа

В качестве допустимого разделителя функция `Str` воспринимает только точку. При наличии другого десятичного разделителя следует использовать функцию `CStr`. Остальные функции преобразования из данного типа в указанный тип приведены в таблице 1.12.

Таблица 1.12 – **Функции преобразования выражения**

Функция	Тип, в который преобразуется выражение
`CBool()`	`Boolean`
`CByte()`	`Byte`
`CDate()`	`Date`
`CDbl()`	`Double`
`CInt()`	`Integer`
`CLng()`	`Long`
`CSng()`	`Single`
`CStr()`	`String`

Функции проверки типа позволяют узнать, является ли переменная выражением определенного типа (таблица 1.13).

Таблица 1.13 – **Проверки типа переменной**

Функция	Проверка
`IsNumeric(x)`	Является ли переменная числовым значением
`IsNull(x)`	Является ли переменная пустым значением (`Null`)
`IsError(x)`	Является ли переменная кодом ошибки

1.7.3 Функции обработки даты и времени

Для выполнения операций над данными типа `Date`, а также для получения текущей даты и времени, разбиения значения даты на ее составляющие части или для преобразования строковых и численных данных в значения типа

Date в *VBA* используются специальные функции обработки даты и времени:

Date()	Возвращает системную дату. Можно также использовать эту функцию как процедуру для установки системных часов
Time()	Возвращает системное время как значение типа Date. Можно также использовать эту функцию для установки системных часов
Now ()	Возвращает системную дату и время
Year(дата)	Возвращает целое значение, являющееся частью выражения типа Date и содержащее год. Год возвращается как число между 100 и 9999
Month(дата)	Возвращает целое значение, являющееся частью выражения типа Date, содержащее месяц. Месяц возвращается как число от 1 до 12
Day(дата)	Возвращает целое, являющееся частью выражения типа Date и содержащее день. День возвращается как число от 1 до 31
Weekday(дата)	Возвращает целое значение, содержащее день недели для выражения типа Date. День недели возвращается как число между 1 и 7, включительно; 1 – воскресенье, 2 – понедельник и т. д.
Hour(дата)	Возвращает целое значение, содержащее часы как часть времени, содержащегося в выражении типа Date. Часы возвращаются как число от 0 до 23. Если аргумент не содержит значения времени, то возвращается 0
Minute(дата)	Возвращает целое значение, содержащее минуты как часть времени в выражении типа Date. Минуты возвращаются как число от 0 до 59. Если аргумент не содержит значения времени, то возвращается 0
Second(дата)	Возвращает целое, содержащее секунды как часть времени в выражении типа Date. Секунды возвращаются как число между 0 и 59
DateAdd(S,N,D)	Возвращает значение [тип Variant (Date)], содержащее дату, к которой добавлен заданный интервал времени
DateDiff(S,D1, D2[,N1 [, N2]])	Возвращает значение [тип Variant (Long)] числа временных интервалов между двумя определенными датами
DateValue(данные)	Возвращает значение типа Date, эквивалентное дате, заданной аргументом, который должен быть строкой, числом или константой, представляющей дату

Более подробно об использовании этих функций можно узнать из справочной системы *VBA*.

1.8 Совместимость и преобразование типов данных

Не все типы данных совместимы друг с другом, и нельзя использовать несовместимые типы данных в одном и том же выражении. Например, не имеет смысла арифметическое сложение строкового типа данных и числа, такое выражение не может быть вычислено и *VBA* не может его оценить. Многие типы данных совместимы друг с другом. Например, вы можете объединять различные численные типы данных в одном и том же выражении; *VBA* автоматически выполняет необходимые преобразования типа различных численных типов. Кроме этого, возможно также иногда автоматически преобразовывать другие типы данных так, чтобы все типы в выражении были совместимы, хотя это не всегда возможно. Очень важно контролировать и знать тип выражения, потому что если выражения содержат несовместимые типы, *VBA* выдает ошибку времени исполнения – ошибку несовпадения типов (*type – mismatch*).

При обработке выражения, содержащего различные типы данных, *VBA* сначала «пытается» устранить любое различие типов, преобразуя значения в выражении в совместимые типы данных. Если устранить какие-либо различия преобразованием типов не удается, отображается ошибка времени исполнения и процедура прекращает выполняться. Например, в выражении 52 & "Информатика" *VBA* всегда выполняет строковую конкатенацию (соединяет две строки), независимо от типов переменных; результатом является тип *String*; это выражение никогда не вызывает ошибки несовпадения типов. *VBA* обычно преобразует все численные типы данных в выражении в тип наибольшей точности, а затем дает этот тип результату выражения. Например, если выражение содержит численные значения с типами *Integer* и *Single*, результат выражения является типом *Single* – тип наибольшей точности в этом выражении. Если вы присваиваете результат численного выражения переменной с наименьшей точностью, чем фактический тип результата выражения, *VBA* округляет результат выражения до тех пор, пока его точность не совпадет с ожидаемым типом. Например, если вы присваиваете численное выражение, имеющее результатом число типа *Double*, переменной типа *Integer*, *VBA* округляет число двойной точности до типа Integer.

При преобразовании числа в строку *VBA* создает строку, содержащую все цифры этого числа и десятичный знак. Число 3413.72 (точка используется для записи числа в коде), например, преобразуется в строку "3413,72". Если число очень большое или очень маленькое, *VBA* может создать строковое представление числа в экспоненциальной записи; например, число 0.0000000004927 преобразуется в строку "4,927E–11".

Следует отметить, можно преобразовывать строку в число, если только эта строка содержит символьное представление числа в десятичном формате или экспоненциальном. Строки "988,6", "812", "-186,7", "1,3E10" пред-

ставляют числа, и VBA может преобразовать их в числа. Строки "1.045", "$74.550" и "За словом – дело!" не могут быть преобразованы в числа.

Когда *VBA* преобразует значения типа *Boolean* в числа, значение *True* преобразуется в 1, а значение *False* – в 0. Когда *VBA* преобразует число в тип *Boolean*, нуль преобразуется в *False*, а любое другое значение преобразуется в True. Когда VBA преобразует значения типа *Boolean* в строки, *VBA* использует строку "*True*" для *True* и "*False*" – для *False*.

Когда *VBA* преобразует тип данных *Date* в число, результатом является численное значение – число типа *Double*, которое содержит количество дней от 23 декабря 1899 (отрицательное число представляет дату, более раннюю, чем 12/23/1899). Десятичная часть числа (если имеется) выражает время дня как часть дня; 0 – это полночь, а 0.5 – это полдень. В *VBA* преобразование численных типов данных в типы *Date* является просто обратным преобразованию типа *Date* в число.

1.9 Оператор присваивания

В *VBA* существует несколько видов операторов присваивания: `Let` (прямое присваивание), `Lset` (левое присваивание), `Rset` (правое присваивание), `Set` (объектное присваивание). Каждый из этих операторов имеет свою специфику в применении.

Рассмотрим наиболее часто используемый оператор прямого присваивания `Let`. Этот оператор осуществляет присваивание результата вычисления некоторого выражения переменной или константе.

Синтаксис:

```
[Let] <идентификатор> = <выражение>
```

Элементы синтаксиса:

`Let` – ключевое слово;

`идентификатор` – обязательный параметр. Идентификатор переменной, константы или свойства объекта, удовлетворяющий стандартным правилам именования;

`выражение` – обязательный параметр. Любое выражение, удовлетворяющее требованиям его составления.

Отметим, что ключевое слово `Let`, как правило, опускается. Оператор присваивания предписывает сначала вычислить значение выражения, заданное в его правой части, и присвоить полученный результат идентификатору, имя которого указано в левой части оператора. В результате, например, действия следующих операторов присваивания:

```
x=3
y=2+x-x^2
Let MyStr = "Начать работу!" 'С ключевым словом Let
```

```
MyInt = 5' Обычный вариант объявления.
```
переменной *y* будет присвоено значение, равное – 4, переменной строкового типа `MyStr` – строка “Начать работу!”, а переменной `MyInt`– значение 5.

Ограничений на тип идентификатора и выражения в операторе присваивания нет, т. е. идентификатору может быть присвоено значение любого выражения, даже если их типы не совпадают. Исключение: значение строкового выражения нельзя присвоить идентификатору не строкового типа.

При вычислении значения выражения тип результата выбирается в зависимости от типа того компонента выражения, который позволяет получить наиболее точное значение. Затем результат полученного вычисления преобразуется в соответствии с типом идентификатора.

1.10 Общая структура программы

Решение задач с помощью *VBA* требует создания проекта. *Проект* – это совокупность нескольких элементов. Основными из них являются: приложение (*MS Excel, MS Word* и т. д.), среда разработки *VBA* (редактор *VBA*) и совокупность модулей, в которых записывается программный код (совокупность программных единиц – процедур проекта).

Любая процедура представляет собой последовательность инструкций. *Инструкция* – это синтаксически полный компонент программы, представляющий собой операцию, описание или определение. Инструкция может содержать ключевые слова, операторы, переменные, константы и выражения.

В *VBA* различают *три типа инструкций:*

1 *Инструкции объявления.* Они используются для объявления процедур, переменных, массивов и констант.

2 *Инструкции присваивания.* Эти инструкции присваивают значение, результат выполнения функции или результат вычисления выражения переменной или константе. Инструкция присваивания всегда содержит символ «=».

3 *Выполняемые инструкции.* Инструкции этого типа выполняют какие-либо действия. Например, выполнение разветвления, повторяющихся действий, метода и т. д.

1.11 Правила оформления кода программ

VBA не накладывает на структуру программы (процедуры) каких-либо особых ограничений. Однако желательно придерживаться следующих рекомендаций:

1 В соответствии с правилами хорошего стиля программирования *размещать инструкции описания в начале процедуры*, т. е. структурно выделив

в ней описательную и исполнительную части.

2 *Использование комментариев*. Комментарии, т. е. пояснения к фрагменту текста процедуры, не являются программным кодом и поэтому компилятором игнорируются. Комментарии выполняют две важные функции:

– делают программу легко читаемой, поясняя смысл кода и алгоритма. Комментарии могут располагаться в любом месте процедуры. Закомментированный текст в начале программы используется для указания действия, выполняемого ею, и краткой справке об авторе программы;

– комментарии по тексту процедуры обычно используются для пояснения ключевых фрагментов кода;

– временно отключают от выполнения закомментированные фрагменты программы, что бывает очень полезно при ее отладке.

Для ввода комментариев используется символ (´) апостроф. Его можно использовать в любом месте строки. При этом все символы, начиная от апострофа до конца строки, будут восприниматься компилятором как комментарий.

Например:

```
Sub Lab1 ()
 ´программирование линейных алгоритмов
 ´задание 2.2
 ´выполнил студент гр. ПС-11 Иванов И.И.
   Dim S As Integer ´S – сумма положительных чисел
   Dim i As Byte ´i – переменная цикла
End Sub
```

3 Для размещения длинных инструкций следует осуществлять *перенос строк кода*.

Длинные инструкции можно размещать в нескольких строках. Для этого используют признак продолжения строки, состоящей из двух символов: пробела и подчеркивания (_). Например,

```
y = 2 * Sqr(x+3) - (log(x^2) +2) _
/ (sin(x-1))
```

При переносе строк необходимо помнить:

– нельзя разбивать переносом строковые константы. Если строковая константа длинная, и ее все же необходимо разбить, следует использовать операцию конкатенации (сцепления). Например,

```
y= "В обрабатываемом массиве нет" & _
" положительных чисел"
```

– за признаком продолжения строки нельзя ставить комментарий;

– допустимо не более семи продолжений одной и той же строки;
– строка не может состоять более чем из 1024 символов.

4 Следует *избегать* расположения *нескольких операторов в одной строке*.

Использование знака двоеточия (:) позволяет разместить несколько операторов (инструкций) на одной строке. Это всегда короткие операторы, как правило, операторы присваивания. Например, конструкции

```
x=x+a
y=x-b
```

эквивалентны такой

```
x=x+a: y=x-b
```

5 Для большей читабельности программного кода необходимо выделять логические уровни отступами.

Любая информация воспринимается лучше, если она имеет иерархическую структуру (процедура, цикл, вложенный цикл, ветвление, вложенное ветвление, описательные инструкции и т. д.). В *VBA* такая структура создается с помощью отступов в коде в нужных местах. Это не только облегчит чтение кода, но и поможет увидеть его структуру. Правила отступа просты. Основной текст программы следует вводить с отступом, например, в три пробела. Каждый вложенный блок (например, инструкции, расположенные внутри оператора цикла) сдвиньте еще на три пробела и т. д.

Контрольные вопросы

1 Какие требования предъявляются к идентификаторам пользователя?
2 Какова основная структура типов данных?
3 Как выполняются логические операции и каков их приоритет?
4 Что такое простые типы данных? Каковы основные диапазоны принимаемых значений?
5 Каков порядок явного объявления переменных?
6 Каким образом объявляются константы?
7 Какова общая структура программы на *VBA*?
8 Каким образом указываются комментарии в программе?
9 Каковы основные требования к оформлению кода программы?
10 Что означает опция `Option Explicit`?

2 ОРГАНИЗАЦИЯ ВВОДА – ВЫВОДА ДАННЫХ

Для обмена информацией с пользователем в *Windows* используются специальные формы, которые называются диалоговыми окнами.

В проектах *VBA* при организации диалога с пользователем используются две разновидности встроенных диалоговых окон: окна сообщений и окна ввода. Окно ввода (`InputBox`) обеспечивает ввод информации, а окно сообщений (`MsgBox`) выводит сообщения для пользователя.

Кроме того, в программах *VBA*, написанных для приложения *MS Excel*, есть возможность ввод и вывод данных осуществить непосредственно из ячеек (в ячейки) рабочего листа.

2.1 Окно сообщения. Стандартная процедура `MsgBox`

Вывод – это процесс переноса информации из оперативной памяти компьютера на внешний носитель (экран, принтер, файл). Практически любое информационное сообщение или предупреждение пользователь может вывести в специальном диалоговом окне.

Программная поддержка окна сообщения осуществляется встроенной процедурой `MsgBox`, которая:

– выводит на экран диалоговое окно, содержащее сообщение и одну или более кнопок (рисунки 2.1–2.5). Кнопка `OK` выводится по умолчанию. Выбор другого вида кнопок определяется разработчиком программы;

– устанавливает режим ожидания нажатия кнопки пользователем.

Сокращенный **синтаксис** процедуры `MsgBox`:

```
MsgBox  сообщение [, атрибуты] [, заголовок]
```

Элементы синтаксиса:

`сообщение` – обязательный параметр, который содержит строковое выражение, отображаемое как сообщение в диалоговом окне. Максимальная длина строки 1024 символа;

`атрибуты` – необязательный параметр, который содержит числовое выражение, с помощью которого можно установить:

– число и тип отображаемых кнопок;

– тип используемого информационного значка;

– основную кнопку.

Значение по умолчанию этого параметра равняется 0. Значения встроенных констант, определяющих число, тип кнопок, а также вид используемых значков, приведенных в таблицах 2.1, 2.2**.** Для вывода в окне сообщения

нескольких кнопок и информационных значков значения констант указываются в виде элементов операции конкатенации.

Например: `vbOKCancel + vbExclamation;`

`заголовок` – необязательный параметр, содержащий строковое выражение, отображаемое в строке заголовка диалогового окна. Если этот параметр опущен, то в строку заголовка помещается имя приложения (см. рисунок 2.1).

Наличие запятых, соответствующих отсутствующим не последним аргументам, является обязательным.

Таблица 2.1 – **Значения параметра `Buttons` процедуры и функции `MsgBox`, определяющие отображаемые кнопки в окне сообщения**

Константа	Значение	Отображаемые кнопки
`vbOKOnly`	0	OK
`vbOKCancel`	1	OK, Отмена
`vbAbortReplyIgnore`	2	Стоп, Повтор, Пропустить
`vbYesNoCancel`	3	Да, Нет, Отмена
`vbYesNo`	4	Да, Нет
`vbReplyCancel`	5	Повтор, Отмена

Таблица 2.2 – **Значения параметра `Buttons` процедуры и функции `MsgBox`, определяющие отображаемые информационные значки в окне сообщения**

Константа	Значение	Значок сообщения
`vbCritical`	16	
`vbQuestion`	32	
`vbExclamation`	48	
`vbInformation`	64	

Параметры атрибуты и заголовок могут отсутствовать. Однако если в процедуре используется параметр `Заголовок`, то перед ним необходимо указать два разделительных символа “,” (запятая).

Например, для вывода простого информационного сообщения необходимо ввести следующее:

```
Sub Pr2_1a()
  'Простое информационное сообщение
   MsgBox "Лабораторная работа № 1"
End Sub
```

Результатом работы данной процедуры является информационное сообщение, выведенное в виде диалогового окна (см. рисунок 2.1, *а*). Поскольку в процедуре `MsgBox` отсутствуют параметры атрибуты и заголовок, то в диалоговом окне выведена только одна обязательная кнопка `OK`, которая используется для закрытия окна, и в заголовке окна выведено имя приложения – *Microsoft Word*.

Усложним выводимое информационное сообщение. На рисунке 2.1, *б* показан результат вывода строковой константы "`Лабораторная работа №1`" и значения строковой переменной `Name`. Для их вывода в качестве единого информационного сообщения использована операция конкатенации (сцепления). Данный вывод реализован с помощью следующей процедуры:

```
Sub Pr2_1b()
' Простое информационное сообщение
' с выводом заголовка
    Dim Name As String
     Name = "(пример 1)"
     MsgBox "Лабораторная работа" & Name, , "Пример"
End Sub
```

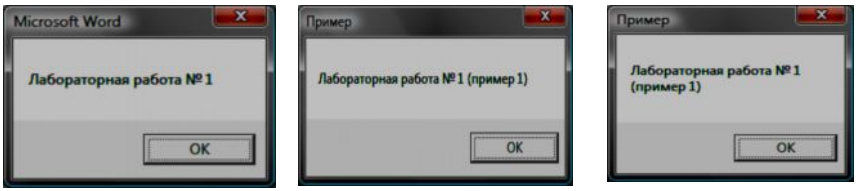
Для вывода сообщения в нескольких строчках используется стандартная функция `Chr()`. Она позволяет получать символы, генерируемые при нажатии различных клавиш. Например, `Chr(9)` – соответствует нажатию клавиши `Tab`, `Chr(13)` – клавиши `Enter`.

Поскольку символы, используемые для начала новой строки являются очень важными при форматировании сообщений и других строковых данных, которыми манипулирует *VBA*-процедуры, *VBA* имеет несколько предопределенных констант для этих символов, чтобы не было необходимости использовать функцию `Chr()`:

- `vbCr` – символ возврата каретки, эквивалент выражения `Chr(13)`;
- `vbTab` – символ табуляции, эквивалент выражению `Chr(9)`. Символы табуляции включают в строки для выравнивания данных в столбцах.

На рисунке 2.1, *в* приведены результаты работы процедуры `Pr2_1_b()`.

```
Sub Pr2_1c()
  'Простое информационное сообщение
  'с выводом заголовка
  Dim Name As String
  Name = "(пример 1)"
  MsgBox "Лабораторная работа № 1" & Chr(13) & Name, ,_
"Пример"
End Sub
```



а) *б)* *в)*

Рисунок 2.1 – Примеры вывода информационного сообщения

Рассмотрим пример вывода информационного сообщения совместно с информационным значком в окне сообщения. Процедура `Pr2_2` осуществляет вывод диалогового окна, представленного на рисунке 2.2.

```
Sub Pr2_2()
'Вывод сообщения в сочетании с
'предупреждающим информационным значком
  MsgBox "Процент выполнения плана выпуска" & _
  "продукции-" & vbCr & " ниже нормы",
  vbExclamation, "Отчетная информация"
End Sub
```

Для вывода информационного значка  (предупреждение) используется в качестве параметра *атрибуты* встроенная константа `vbExclamation` (см. таблицу 2.2). Рассмотрим еще один пример вывода диалогового окна с кнопками `OK`, `Отмена` и со значком  (предупреждающий запрос). Данное окно (см. рисунок 2.3) можно вывести с помощью следующей процедуры:

```
Sub Pr2_3()
'Вывод сообщения с двумя кнопками OK и Отмена
'в сочетании с предупреждающим значком.
  MsgBox "Процент выполнения плана выпуска" & _
  "продукции -" & Chr(13)& "ниже нормы",vbOKCancel_
  + vbQuestion + vbDefaultButton1,"Отчетная информация"
End Sub
```

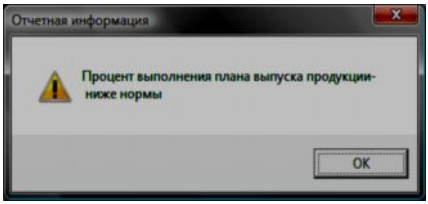


Рисунок 2.2 – Пример вывода сообщения с информационным значком

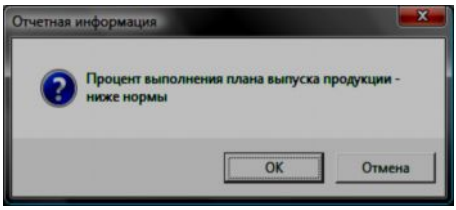


Рисунок 2.3 – Пример вывода предупреждающего информационного сообщения

2.2 Окно ввода. Стандартная функция `InputBox()`

Ввод – это процесс переноса информации с внешнего носителя (клавиатура, файл) в оперативную память компьютера. Программная поддержка окна ввода обеспечивается функцией `InputBox()`, которая:

– выводит на экран диалоговое окно, содержащее сообщение, поле ввода и две кнопки: `OK` и `Cancel` (см. рисунок 2.4);

– устанавливает режим ожидания ввода текста пользователем и нажатия кнопки;

– возвращает значение типа `String` по нажатию кнопки `OK`, содержащее текст, введенный в поле ввода;

– возвращает пустую строку (значение `Empty`) при нажатии кнопки `Cancel`.

Сокращенный синтаксис:

```
InputBox (сообщение [, заголовок] [, умолчание]
```

Элементы синтаксиса:

`сообщение` – обязательный параметр. Строковое выражение, отображаемое как сообщение в диалоговом окне. Может содержать несколько строк. Максимальная длина строки 1024 символа;

`заголовок` – необязательный параметр. Строковое выражение, отображаемое в строке заголовка диалогового окна. Если этот параметр опущен, то в строку заголовка помещается имя приложения;

`умолчание` – необязательный параметр. Строковое выражение, отображаемое в поле ввода. Используется по умолчанию, если пользователь не введет другую строку. Если параметр опущен, то поле ввода изображается пустым.

Наличие запятых, соответствующих отсутствующим не последним аргументам, является **обязательным**.

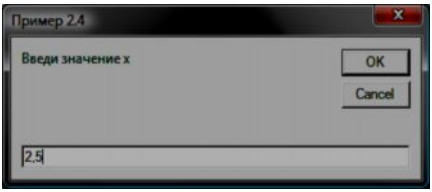


Рисунок 2.4 – Пример окна диалога

Например, инструкция:

```
X = InputBox("Введите значение x", "Пример 2.4", "2.5")
```

с помощью окна ввода (см. рисунок 2.4) осуществляет ввод некоторого значения с клавиатуры. Затем введенное значение преобразуется в соответствии с типом переменной *x*. При вводе вещественных чисел используется знак десятичная запятая.

2.3 Ввод – вывод данных на рабочий лист *Excel*

Для организации ввода – вывода данных непосредственно с (или на) рабочий лист приложения *MS Excel* используется объект `Range` (Диапазон ячеек) или свойство `Cells` (Ячейка) объекта `Worksheets` (Рабочий лист). С их помощью можно работать с любой ячейкой рабочего листа *Excel*.

Используя объект `Cells`, можно содержимое ячейки рабочего листа присваивать в качестве значения переменной и наоборот, значение выражения выводить в любой ячейке.

Например,

```
x = WorkSheets(1).Cells(5,3).Value или
x = Cells(5,3) или
x = Range("C5")
```

Переменной *x* присваивается значение свойства `Value` ячейки *C5*.

```
x = Cells(1+i, 1+j)
```

Переменной *x* присваивается значение ячейки, номер строки и столбца которой определяется значениями выражения $1+i$ и $1+j$ соответственно.

```
Cells(8,4)= x+2*y
```

В ячейке *D8* выводится значение выражения $x+2y$.

```
p = InputBox("Введи номер строки")
q = InputBox("Введи номер столбца")
Cells(p,q) = a + Range("A1")
```

В ячейку рабочего листа, расположенную в *p* строке и *q* столбце, выводится значение выражения `a + Range("A1")`. Номера строки и столбца вводятся с клавиатуры.

Пример 2.1. Найти площадь треугольника, зная длину трех его сторон. Таким образом, необходимо вычислить значение площади по формуле Герона $S=\sqrt{p(p-a)(p-b)(p-c)}$, где $p=(a+b+c)/2$ – полупериметр, при этом ввод – вывод данных осуществить на рабочий лист. Фрагмент рабочего листа с исходными данными представлен на рисунке 2.5. Расчеты получены с помощью процедуры `Pr2_5()` и выведены на тот же рабочий лист. Отметим, что в данной про-

	A	B	C	D
1	Исходные данные:			
2	Введите длину сторон треугольника			
3		A =	2,5	
4		B =	3	
5		C =	4	
6	Площадь треугольника:			
7			3,74531	
8				

Рисунок 2.5 – Пример ввода – вывода на рабочий лист *MS Excel*

грамме не производится проверка корректности исходных данных, а для корректной работы программы это следует делать.

```
Sub Pr2_5 ()
 Dim a As Single, b As Single, c As Single
 Dim S As Single, p As Single
 a = WorkSheets(1).Cells(3, 3)
 b = WorkSheets(1).Cells(4, 3)
 c = WorkSheets(1).Cells(5, 3)
 p = (a + b + c) / 2
 S = Sqr(p * (p - a) * (p - b) * (p - c))
 WorkSheets(1).Cells(6, 1) = "Площадь треугольника:"
 WorkSheets(1).Cells(7, 3) = S
End Sub
```

Контрольные вопросы

1 Для чего используется функция `MsgBox`?

2 Каков формат данной функции `MsgBox` и сколько у нее обязательных параметров и для чего они предназначены?

3 Каким образом влияют на работу функции `MsgBox()` системные константы?

4 Как вывести сообщение в несколько строк?

5 Для чего используется функция `InputBox()`?

6 Каким образом в функции `InputBox()` задается значение по умолчанию?

7 Значение какого типа возвращается функция `InputBox()`?

8 В каком контексте используется функция `InputBox()` для обработки вводимого числового значения?

9 Является обязательным наличие запятых, соответствующих отсутствующим не последним аргументам?

10 Каков порядок ввода – вывода данных на рабочий лист *MS Excel*?

11 Как вывести значение в ячейку B2 второго листа рабочей книги?

3 ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ: ЛИНЕЙНЫХ, РАЗВЕТВЛЯЮЩИХСЯ И ЦИКЛИЧЕСКИХ

Логическая структура любого алгоритма может быть представлена комбинацией трех базовых структур:

– *следование* или *линейный* алгоритм, образуемый последовательности действий, следующих одно за другим;

– *ветвление* или *разветвляющийся* алгоритм, который выполняется в зависимости от результата проверки условия (да или нет) и выбора одного из альтернативных путей работы алгоритма. Каждый из путей ведет к общему выходу так, что работа алгоритма будет продолжаться независимо от того, какой путь будет выбран;

– *цикл* или *циклический* алгоритм, который заключается в многократном выполнении некоторой совокупности действий, которая называется телом цикла.

3.1 Программирование линейных алгоритмов

Линейный алгоритм, который заключается в последовательном (линейном) порядке выполнения действий, рассмотрим на примере вычисления значения функции $f(x, y) = |2x| + \sin^2(y + 5)$. Заметим, что эта функции двух переменных x, y, которые в программе описаны как переменные вещественного типа, следовательно, для результата также объявлена переменная вещественного типа (`Double`). Пример программы с пояснениями представлен ниже:

`Public Sub prog1()`	Заголовок процедуры `prog1` (начало программы).
`Dim x As Double, y As Double` `Dim f As Double`	Описание переменных: переменные `x, y, f` вещественного типа (`Double`).
`x=CDbl(InputBox("Введите x"))` `y=CDbl(InputBox("Введите y"))`	Ввод значений `x` и `y`. Функция `InputBox` выводит на экран окно с полем ввода и сообщением «Введите *x*» и возвращает значение типа строка (`String`). Для преобразования вводимого значения к вещественному типу – `Double` используется функция `CDbl`.
`f = Abs(2*x) + Sin(y + 5)^2`	Вычисление значения переменной `f`: функция `Abs`(аргумент) возвращает модуль аргумента, `Sin`(аргумент) –

`MsgBox "Результат = " & f`	синус аргумента, ^ – степень числа. Процедура `MsgBox` выводит на экран окно сообщений с текстом «Результат = 12» (если `f = 12`).
`End Sub`	Конец программы.

На рисунке 3.1 приведена общая блок-схема данной линейной программы.

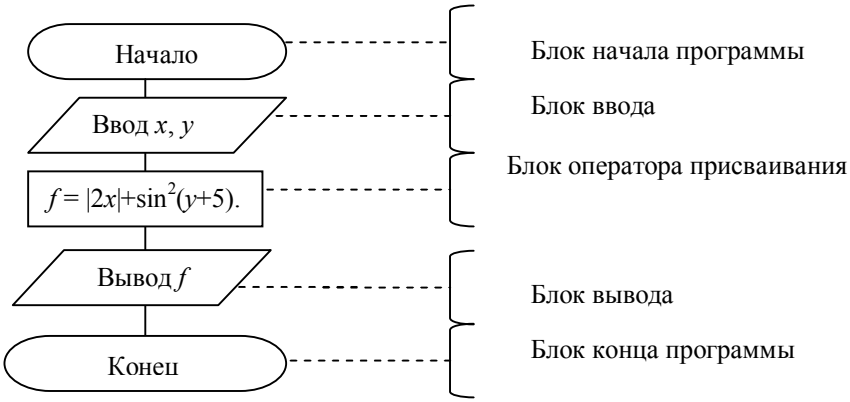


Рисунок 3.1 – Блок-схема программы `prog1`

3.2 Программирование разветвляющихся алгоритмов

3.2.1 Условный оператор `if`

Условный оператор позволяет выбирать и выполнять действия в зависимости от истинности некоторого условия.

Синтаксис: условный оператор имеет два варианта:

а) однострочная форма записи условного оператора:

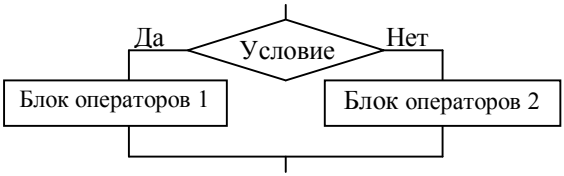
```
If <условие> Then [<операторы 1>] [Else [<операторы 2>]]
```

б) блочная форма записи условного оператора:

```
If <условие 1> Then
      [<операторы 1>]
         . . .
  [ElseIf <условие n> Then
        [<операторы n> ]…
     [Else]
        [<Иначе_Операторы>]]
End If.
```

Порядок выполнения: вычисляется значение <условие>. Оно может принимать значения **TRUE (Истина)** или **FALSE (Ложь)**. Если <условие> принимает значение TRUE, то выполняются [<операторы 1>] (операторы ветки **Then**), в противном случае – [<операторы 2>] (операторы ветки **Else**).

Изображение в блок-схеме:



Пример 3.1. Для введенного числа выполнить следующие вычисления:

$$rez = \begin{cases} 2x, & если\ x > 0 \\ x^2, & если\ x < 0 \\ 0, & если\ x = 0 \end{cases}.$$

Ввод значений аргумента *x* должен осуществляться из листа рабочей книги *MS Excel* из ячейки A1, вывод – в ячейку B1. Блок-схема алгоритма программы представлена на рисунке 3.2.

```
Public Sub prog2()
Dim x As Double
x=Worksheets(1).Range("A1")


If x > 0 Then
  s = 2*x
 ElseIf x < 0 Then
       s = x^2
     Else
       s = 0
End If


Worksheets(1).Range("B1")=s

End Sub
```

Ввод значения переменной x из ячейки A1 (Range ("A1")) на Листе1 (Worksheets(1))

Условный оператор. Если условие принимает значение True (Истина), то выполняется оператор присваивания ветки Then (s = 2x) и завершается условный оператор, в противном случае проверяется второе условие: x < 0. Если условие True, то s = 2*x , иначе x = 0 .

Вывод результата s – на Лист1 в ячейку B1.

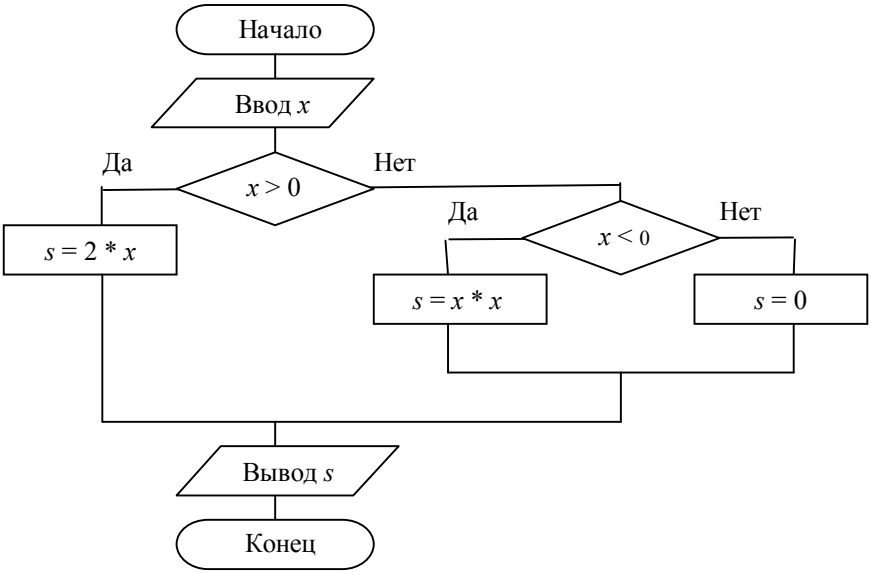


Рисунок 3.2 – Блок-схема программы *prog2*

3.2.2 Оператор `Select Case`

Данный оператор применяется в том случае, если во всех логических выражениях (условиях) участвует одна и та же величина (переменная).

Синтаксис оператора `Select Case`:

```
Select Case <проверяемое_выражение>
  Case <список_значений_1>
     <блок_операторов_1>
  Case <список_значений_2>
     <блок_операторов_2>
     . . .
  [Case Else
     <блок_операторов_Else>]
End Select
```

Элементы синтаксиса:

`проверяемое_выражение` – выражение, которое вычисляется в начале работы оператора `Select Case`. Это выражение может быть логического, числового или строкового типа;

`список_значений` – это одно или несколько выражений, разделенных запятой. При выполнении оператора проверяется, соответствует ли хотя бы

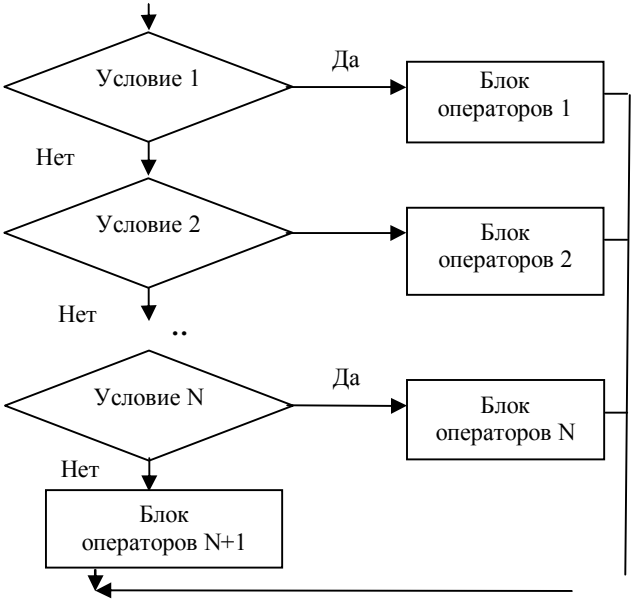
один из элементов этого списка проверяемому выражению. Эти элементы списка значений могут иметь одну из трех форм:

1 <выражение>. В этом случае проверяется, совпадает ли значение проверяемого_выражения с этим выражением. Например: `Case 45`.

2 <выражение_1> **To** <выражение_2>. В этом случае проверяется, находится ли значение проверяемого_выражения в указанном диапазоне значений. Например: `Case 5 To 12`.

3 **Is** <логическая_Операция> <выражение>. В этом случае проверяемое_выражение сравнивается с указанным значением с помощью заданной логической операции (или операции отношения). Например: `Case Is >= 10`. В данном случае условие считается выполненным, если проверяемое значение не меньше 10.

Блок-схема данного оператора:



Порядок работы данного оператора: вычисляется проверяемое выражение. Далее, если хотя бы один из элементов списка соответствует проверяемому выражению, то выполняется соответствующая группа операторов, а затем управление передается оператору, находящемуся после ключевых слов `End Select`. При этом остальные списки выражений не проверяются, т. е. отыскивается только первый подходящий элемент списков выражений. Если же ни один из элементов всех этих списков не соответствует прове-

ряемому выражению, выполняются операторы группы Else, если блок Else отсутствует, управление передается оператору, следующему за End Select.

Пример использования:

```
Select Case City
  Case "МИНСК"
     KOD = 017
  Case "ГОМЕЛЬ"
     KOD = 0232
End Select
```

Пример 3.2. Составить процедуру, которая выводит на экран окно сообщения, содержащее командные кнопки Yes, No и Cancel; затем определяет выбранную пользователем кнопку и выводит сообщение, поясняющее этот выбор. Далее представлен текст процедуры Demo_ВыборКнопки(), которая демонстрирует использование оператора выбора.

```
Sub Demo_ВыборКнопки()
 Dim Otvet As Integer
 Otvet = MsgBox(prompt:="Выберите кнопку",Title:=mTitle,
_Buttons:=vbYesNoCancel + vbQuestion)
 Select Case Otvet
  Case Is = vbYes
  MsgBox prompt:="Выбрана кнопка'Да'",Title:=mTitle,
_Buttons:=vbInformation
  Case Is = vbNo
  MsgBox prompt:="Выбрана кнопка'Нет'",Title:=mTitle,
_Buttons:=vbInformation
  Case Is = vbCancel
  MsgBox prompt:="Выбрана кнопка'Отмена'", Title:=mTitle,
_Buttons:=vbCritical
 End Select
End Sub
```

3.2.3 Функции – заменители синтаксических конструкций разветвления Choose(), IIF(), Switch()

В *VBA* предусмотрено несколько функций, которые позволяют заменять синтаксические конструкции условного перехода, например, IF … THEN … ELSE или SELECT … CASE. Особенных преимуществ применение таких функций не дает, только программный код становится немного короче, но профессиональные программисты часто их используют. Начинающим же программистам рекомендуются обычные синтаксические конструкции. Однако для чтения чужого кода необходимо знать такие функции. Приведем некоторые из них:

Choose() – принимает число (номер значения) и несколько значений. Возвращает значение, порядковый номер которого соответствует передаваемому числу. Например, `Choose (2, "Первый", "Второй", "Третий")` вернет `"Второй"`.

IIF() – расшифровывается как `Immediate IF`, то есть "Немедленный IF". Представляет из себя упрощенный вариант `IF`, когда проверяется условие и возвращается одно из двух значений. Например, `IIf (n > 10, "Значение > 10", "Значение <= 10")`

Switch() – принимает неограниченное количество пар типа выражение/значение, проверяет каждое выражения на истинность и возвращает значение для первого выражения, которое оказалось истинным.

Например:

```
Function Language (CityName As String)
Language = Switch(CityName = "Москва", "русский",
CityName = "Париж", "французский", CityName _
= "Берлин", "немецкий")
End Function
```

3.2.4 Оператор безусловного перехода

Оператор безусловного перехода всегда изменяет порядок выполнения операторов в процедуре или функции *VBA*. При этом *VBA* не проверяет никаких условий, а просто переходит к выполнению кода с другого места. *VBA* имеет только один оператор безусловного перехода: `GoTo`. Для его использования на практике существует очень мало причин; процедуры, которые используют несколько операторов `GoTo`, трудны для понимания. Однако следует знать, как работает оператор `GoTo` главным образом для того, чтобы понимать, как действует обработчик ошибок `GoTo`.

Синтаксис оператора `GoTo`:

```
GoTo <метка>
```

Элементы синтаксиса:

`<метка>` – это любая допустимая метка в той же процедуре или функции, которая содержит оператор `GoTo`. При выполнении оператора `GoTo` *VBA* немедленно переходит к выполнению оператора в строке, определенной с помощью метки строки. Метка строки (`line label`) – это особый тип идентификатора, который задает определенную строку по имени.

Метка строки имеет следующий формат:

`<имя_строки>:`

Выражение `имя_строки` – это любой допустимый идентификатор *VBA*. Метка строки может начинаться в любом столбце строки, если она является

первым непустым символом в строке.
Например:

```
Sub F()
...
 GoTo EndSubCancel
...
 EndSubCancel: MsgBox "Работа завершена"
...
End Sub
```

Следует отметить, что рекомендуется избегать использования инструкции GoTo, так как программный код, содержащий данную инструкцию достаточно сложно отладить. Тем более, что существует много возможностей исключить использование данного оператора, прибегая к различным вариантам инструкции Exit: Exit Do, Exit For, Exit Function, Exit Sub и др.

3.3 Программирование циклических алгоритмов

3.3.1 Оператор цикла с параметром (For…Next)

Оператор цикла For позволяет повторять группу операторов фиксированное количество раз.
Синтаксис:

```
For <счётчик_цикла> = <начало> To <конец> [Step <шаг>]
<тело цикла>
[Exit For]
  . . .
Next [<счётчик_цикла>],
```

где

<счётчик_цикла> – числовая переменная;

<начало> – начальное значение (выражение) переменной <счётчик_цикла>;

<конец> – заключительное значение (выражение) переменной <счётчик_цикла>;

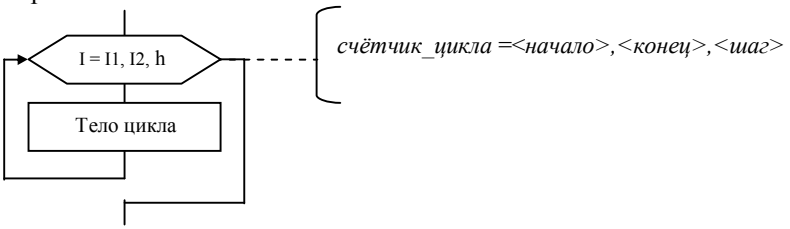
<тело цикла> – это последовательность операторов, которая будет выполнена заданное число раз.

Порядок выполнения: переменной <счётчик_цикла> присваивается значение <начало> и проверяется условие: <начало> ≤ <конец>; если

условие неверно, то <тело цикла> не выполняется и управление передается на оператор, следующий за Next. Если же условие выполняется, то выполняется <тело цикла>, затем значение <счётчик_цикла> изменяется на значение <шаг> (увеличится в случае положительного значения <шаг> и уменьшится при отрицательном значении <шаг>). Данный процесс будет выполняться, пока значение <счётчик_цикла> не достигнет значения <конец> (если шаг положителен, цикл завершится, когда впервые выполнится условие: <счетчик_цикла> **>** <конец>. Если шаг цикла отрицателен, условие его завершения: <счетчик_цикла> **<** <конец>).

Досрочно завершить цикл For...Next можно и с помощью оператора **Exit For**. Такие операторы могут быть расположены в тех местах тела цикла, где требуется из него выйти, не дожидаясь выполнения условия завершения.

Изображение в блок-схемах:



Пример 3.3. Вычислить *n*-й член последовательности, заданной формулой $a_n = a_{n-1} + a_{n-2}$, если $a_1 = 1,\ a_2 = 1$.

`Public Sub prog3()`	
`  Dim n As Byte`	Описание переменной типа `Byte`.
`  Dim an As Integer,`	
`  a1 As Integer,`	
`  a2 As Integer`	
`  n=CByte(InputBox("n ="))`	Ввод значения переменной `n` (номера элемента последовательности) через окно ввода и преобразование введенного значения к типу `Byte`.
`  a1 = 1: a2 = 1`	Присваивание начальных значений переменным `a1` и `a2` (двоеточием разделяются операторы, записанные на одной строке).
`  For i = 3 To n`	Организация цикла `For…Next`, в котором счетчик `i` изменяется от `3` до `n` с шагом `1`.
`    an = a1 + a2`	Вычисление следующего члена последовательности как суммы двух преды-

`a1 = a2: a2 = an`	дущих членов последовательности. Изменение последнего и предпоследнего значений последовательности на данный момент.
`Next i`	Увеличение счетчика на 1 (конец тела цикла).
`MsgBox an` `End Sub`	Вывод значения `an`.

3.3.2 Циклы с предусловием и постусловием

Циклы данного вида используются, когда заранее неизвестно, сколько раз будет выполняться тело цикла. Циклы с предусловием (`Do While…Loop, While…Wend, Do Until…Loop`) представлены в таблице 3.1, а операторы циклов с постусловием (`Do…Loop While, Do…Loop Until`) – в таблице 3.2.

Отличие циклов с предусловием от циклов с постусловием заключается в том, что тело цикла первых может не выполниться ни разу, в то время как тело цикла с постусловием всегда выполнится хотя бы один раз.

Таблица 3.1 – **Циклы с предусловием**

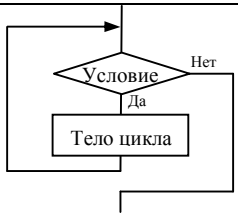
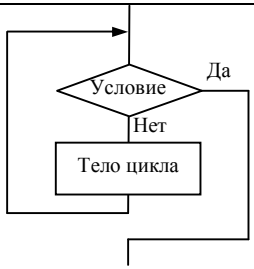
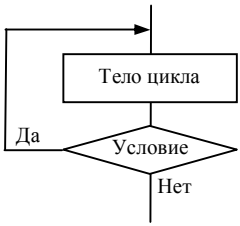
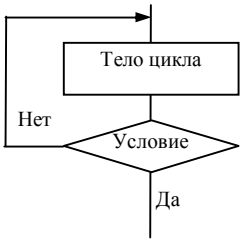
Синтаксис	Порядок выполнения	Изображение в блок-схемах
Do While <условие> <тело цикла> [**Exit Do**] … **Loop**	<Тело цикла> будет выполняться в том случае, когда <условие> имеет значение истина (TRUE) (цикл продолжается при истинном значении <условия>). Если <условие> ложно (FALSE), то выполняются операторы, стоящие за циклом. В первом случае есть возможность досрочного выхода из цикла (это реализовано через **Exit Do**)	
While <условие> <тело цикла> **Wend**		
Do Until <условие> <тело цикла> [**Exit Do**] … **Loop**	<Тело цикла> выполняется до тех пор, пока <условие> не примет значение истина (цикл продолжается при ложном значении <условия>). Есть возможность досрочного выхода из цикла (это реализовано через **Exit Do**)	

Таблица 3.2 – **Операторы циклов с постусловием**

Синтаксис	Порядок выполнения	Изображение в блок-схемах
Do <тело цикла> [**Exit Do**] … **Loop While** <условие>	<Тело цикла> будет выполняться в том случае, когда <условие> имеет значение истина (цикл продолжается при истинном значении <условия>). Если <условие> ложно, то выполняются операторы, стоящие за циклом. Предоставлена возможность досрочного выхода из цикла (это реализовано через **Exit Do**)	
Do <тело цикла> [**Exit Do**] … **Loop Until** <условие>	<Тело цикла> выполняется до тех пор, пока <условие> не примет значение истина (цикл продолжается при ложном значении <условия>). Есть возможность досрочного выхода из цикла (это реализовано через **Exit Do**)	

Рассмотрим пример, который реализуется с помощью различных операторов цикла.

Пример 3.4. Вычислить значение функции $f(x) = \cos^3 \frac{x}{2y}$ при $y = 1{,}32$, на отрезке $1{,}5 \le x \le 3{,}5$ с шагом $\Delta x = 0{,}1$. Исходные данные для расчетного интервала и значение параметра y задавать на листе *MS Excel*. Расчетные значения также выводить на лист *Excel*.

Для решения данной задачи вычисления значения функции на отрезке написать программу с использованием операторов цикла с предусловием, постусловием, а также оператора `For`.

Реализация с помощью оператора **`for`**:

```
Public Sub Primer_3_3_1()
Dim dx As Single
Dim y As Single
Dim f As Single
Dim begin_x As Single, end_x As Single
Dim i As Integer
```

```
i = 3 'номер строки, с которой начать вывод данных
'ввод исходных данных
begin_x = Worksheets(2).Cells(2, 5)
end_x = Worksheets(2).Cells(2, 7)
dx = Worksheets(2).Cells(2, 9)
y = Worksheets(2).Cells(3, 5)
'проверка корректности исходных данных
If y = 0 Then
 MsgBox "Некорректное значение параметра y",,"Ошибка!"
 Exit Sub
End If
'Организация вычисления значений функции на отрезке
For x = begin_x To end_x Step dx
 f = Cos(x / (2 * y)) ^ 3 'вычисление значения функции
 'вывод на лист Excel
 Worksheets(2).Cells(1 + i, 1) = x
 Worksheets(2).Cells(1 + i, 2) = f
 i = i + 1 'переход на следующую строку
Next x
End Sub
```

Рассмотрим фрагменты программного кода для организации вычисления значения рассматриваемой функции на отрезке с помощью операторов циклов с предусловием **Do While**…**Loop** (1), **While**…**Wend** (2), **Do Until**…**Loop** (3):

1 Организация вычисления значений функции на отрезке с помощью оператора **Do While**…**Loop**:

```
x = begin_x
Do While x <= end_x
  f = Cos(x / (2 * y))^3 'вычисление значения функции
  'вывод на лист Excel
  Worksheets(2).Cells(1 + i, 4) = x
  Worksheets(2).Cells(1 + i, 5) = f
  x = x + dx 'переход к следующему значению на отрезке
  i = i + 1  'переход на следующую строку
Loop
```

2 Организация вычисления значений функции на отрезке с помощью оператора **While**…**Wend**:

```
x = begin_x
While x <= end_x
  f = Cos(x / (2 * y))^3 'вычисление значения функции
```

```
  'вывод на лист Excel
  Worksheets(2).Cells(1 + i, 4) = x
  Worksheets(2).Cells(1 + i, 5) = f
  x = x + dx 'переход к следующему значению на отрезке
  i = i + 1  'переход на следующую строку
Wend
```

3 Организация вычисления значений функции на отрезке с помощью оператора **Do Until**…**Loop**:

```
x = begin_x
Do Until x > end_x
  f = Cos(x / (2 * y))^3 'вычисление значения функции
  'вывод на лист Excel
  Worksheets(2).Cells(1 + i, 7) = x
  Worksheets(2).Cells(1 + i, 8) = f
  x = x + dx 'переход к следующему значению на отрезке
  i = i + 1  'переход на следующую строку
Loop
```

Рассмотрим фрагменты программного кода для организации вычисления значения функции на отрезке с помощью операторов циклов с предусловием **Do**…**Loop While** (1), **Do**…**Loop Until** (2):

1 Организация вычисления значений функции на отрезке с помощью оператора **Do**…**Loop While**

```
x = begin_x
Do
  f = Cos(x / (2 * y))^3 'вычисление значения функции
  'вывод на лист Excel
  Worksheets(2).Cells(1 + i, 7) = x
  Worksheets(2).Cells(1 + i, 8) = f
  x = x + dx 'переход к следующему значению на отрезке
  i = i + 1  'переход на следующую строку
Loop While x <= end_x
```

2 Организация вычисления значений функции на отрезке с помощью оператора **Do**…**Loop Until**:

```
x = begin_x
Do
  f = Cos(x / (2 * y))^3 'вычисление значения функции
  'вывод на лист Excel
  Worksheets(2).Cells(1 + i, 7) = x
  Worksheets(2).Cells(1 + i, 8) = f
```

```
  x = x + dx 'переход к следующему значению на отрезке
  i = i + 1  'переход на следующую строку
Loop Until x > end_x
```

Таким образом, в зависимости от того, какой оператор цикла используется, указывается свое условие выполнения цикла.

3.4 Объявление и обработка массивов

3.4.1 Понятие массива и способы его объявления

Массив – совокупность однотипных элементов данных (чисел, логических данных, символов), которой при обработке присвоено определенное имя. Массивы бывают *статические* и *динамические*. Статическими называются массивы, количество элементов в которых заранее известно и не изменяется в ходе выполнения программы. *Динамические массивы* – массивы, в которых либо не известно начальное количество элементов, либо размерность массива (количество элементов) изменяется при выполнении программы.

Описание (объявление) массивов:

1) одномерный статический массив

```
Dim <имя массива> (<начальное значение индекса> To <конечное значение индекса>) [As <тип элементов массива>]
```

или

```
Dim <имя массива> (<количество элементов массива>) [As <тип элементов массива>]
```

2) двумерный статический массив

```
Dim <имя массива> (<начальное значение индекса по строкам> To <конечное значение индекса по строкам >, < начальное значение индекса по столбцам> To < конечное значение индекса по столбцам>) [As <тип элементов массива>]
```

или

```
Dim <имя массива> (<количество строк>, <количество столбцов>) [As <тип элементов массива>].
```

Первый способ отличается от второго тем, что в первом случае указывается индекс первого и последнего элементов, во втором же – только количество элементов, нумерация которых может начинаться как с 0, так и с 1. Это зависит от опции `Base` (задает базовый индекс). Если опция не указана, то нумерация элементов массива начинается с нуля. Для изменения базового индекса в начале листа модуля необходимо написать `Option Base 1`. Например:

а) `Dim A(1 To 10) As Integer` – массив A состоит из 10 элемен-

тов целого типа, индексы которых 1, 2, …, 10;

б) `Dim A(10) As Integer` – массив состоит из 10 значений целого типа. Индексация зависит от опции `Base`. Если опция не указана, то номера элементов – от 0 до 9, если же указана (т. е. вначале модуля записано `Option Base 1`), то номера элементов изменяются от 1 до 10;

3) динамический массив:

Dim <имя массива> **( ) [As** <тип элементов массива>**].**

После определения количества элементов массива выполняется его переопределение:

ReDim <имя массива> **(**<задается размерность массива (одномерного/двумерного >**).**

Например:

`Dim A() As Single` – динамический массив *A* вещественных элементов.

`n=7`.

`ReDim A(1 To n)` – переопределение одномерного массива из `n` значений.

`ReDim A(5,n)` – переопределение двумерного динамического массива, состоящего из `5` строк и `n` столбцов (начало индексации элементов определяется по опции `Base`).

Обращение к элементу массива осуществляется следующим образом: указывается имя массива, а затем в круглых скобках указывается номер элемента в массиве. Если массив двумерный – указывается вначале номер строки, затем через запятую – номер столбца.

3.5 Типовые алгоритмы обработки одномерного массива

Пример 3.5. Ввод – вывод элементов одномерного массива с помощью функций `InputBox()`, `MsgBox()`

```
Option Base 1
Option Explicit
Public Sub Vvod1()
Const N = 10
Dim A(N) As Integer
Dim i As Integer
Dim str As String
'Ввод элементов массива
For i = 1 To N
    A(i) = CInt(InputBox("Введите A[" & i & "]=",
"Ввод  массива"))
Next i
```

```
'Вывод элементов массива
For i = 1 To N
   str = str & A(i) & "  "
Next i
MsgBox "Исходный массив:" & Chr(13) & str, , "Вывод
массива"
End Sub
```

Пример выполнения данной процедуры представлены на рисунке 3.3.

а) б)

Рисунок 3.3 – Пример выполнения программы по формированию элементов одномерного массива: а – ввод элементов; б – вывод элементов массива

Пример 3.6. Заполнение одномерного массива с помощью генератора случайных чисел.

```
Public Sub vvod2()
Const N = 10
Dim A(N) As Integer
Dim i As Integer
Dim str As String
Randomize ' Инициализация генератора случайных чисел
For i = 1 To N
  A(i)=Int((10 * Rnd)+1) '[случайное число от 1 до 10]
Next i
For i = 1 To N
   str = str & A(i) & "  "
Next i
MsgBox "Исходный массив:" & Chr(13) & str, , "Вывод
массива"
End Sub
```

Заметим, что порядок выполнения этой процедуры от описанной в примере 3.5 состоит в том, что в данном примере происходит заполнение массива случайными числами в диапазоне от 1 до 10. Формирование диапазона генерации случайных чисел происходит согласно принципу:

```
(End * Rnd) + Begin,
```

где [Begin, End] – задаваемый диапазон для генерации случайных чисел.

Пример 3.7. Поиск максимального (минимального) элемента (индекса) массива.

```
Public Sub Min_Max()
Const N = 10
Dim A(N) As Integer
Dim i As Integer, i_max As Integer, i_min As Integer
Dim str As String, str1 As String

Randomize ' Инициализация генератора случайных чисел
For i = 1 To N
    A(i) = Int((10 * Rnd) + 1)
Next i
For i = 1 To N
   str = str & A(i) & "  "
Next i
'Поиск минимального (максимального) элемента
i_max = 1
i_min = 1
For i = 2 To N
    If A(i) > A(i_max) Then i_max = i
    If A(i) < A(i_min) Then i_min = i
Next i
str1 = "Максимальный элемент: A[" & i_max & "] = " &_
A(i_max) & Chr(13) & "Минимальный элемент:A["&i_min &_
"] = " & A(i_min)

MsgBox "Исходный массив:" & Chr(13) & str & Chr(13) &
str1, , "Вывод массива"
End Sub
```

Пример выполнения данной процедуры представлен на рисунке 3.4.

Пример 3.8. Сортировка элементов массива (методом выбора)

```
Public Sub Sort()
Const N = 10
Dim A(N) As Integer, vs As Integer
Dim i As Integer, max As Integer, j As Integer
Dim str As String, str1 As String
Randomize ' Инициализация генератора случайных чисел
```

```
For i = 1 To N
    A(i) = Int((10 * Rnd) + 1) '[случайное число от 1
до 10]
Next i
'формирование строки для вывода исходного массива
For i = 1 To N
   str = str & A(i) & "  "
Next i
For j = 1 To N
    'поиск максимального элемента, начиная с j-го
    max = j
    For i = j To N
        If A(i) > A(max) Then max = i
   Next i
'поменять местами j-й и найденный максимальный элементы
   vs = A(j)
   A(j) = A(max)
   A(max) = vs
Next j
'формирование строки для вывода отсортированного
'массива
For i = 1 To N
   str1 = str1 & A(i) & "  "
Next i
MsgBox "Исходный массив:" & Chr(13) & str & Chr(13) &_
"Отсортированный массив: " & Chr(13) & str1, , "Вывод_
массива"
End Sub
```

Пример выполнения данной процедуры сортировки массива представлен на рисунке 3.5.

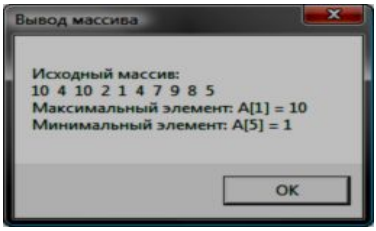


Рисунок 3.4 – Пример работы программы поиска максимального и минимального элементов

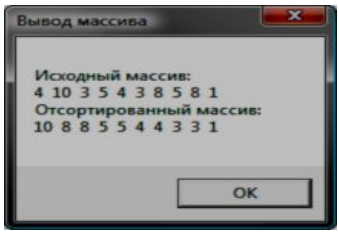


Рисунок 3.5 – Пример работы программы сортировки

3.5.1 Пример программы обработки двумерного массива

Пример 3.9. Определить сумму и количество положительных элементов в массиве целых чисел *A* размерности (5×8).

```
Public Sub prog4()
'Описание целочисленного массива (5 строк, 8 столбцов)
Dim a(1 To 5, 1 To 8) As Integer
Dim s As Integer, kol As Integer
s = 0              'Обнуление переменной s
kol = 0            'Обнуление переменной kol
For i = 1 To 5     'Цикл по строкам (от 1 до 5)
 For j = 1 To 8    'Цикл по столбцам (от 1 до 8)
   'Ввод элементов массива с листа Excel
   a(i,j)=Worksheets(1).Cells(i,j)
   'Сравнение элементов массива с нулем
    If a(i, j) > 0 Then
       'Добавление элемента >0 к текущей сумме
       s = s + a(i, j)
       'Увеличение количества элементов >0 на 1
       kol = kol + 1
   End If
 Next j      'счетчик по j
Next i       'счетчик по i
'Вывод результатов на лист Excel
Worksheets(1).Range("A12")= s
Worksheets(1).Range("A13")= kol
End Sub
```

Пример 3.10. Рассмотрим пример обработки одномерного *динамического* массива, в котором необходимо определить максимальный и минимальный элементы и поменять их местами.

```
Public Sub prog5()
 Dim b() As Double 'Описание динамического массива
                   'вещественных значений
 Dim nom_max As Integer 'номер максимального элемента
 Dim nom_min As Integer 'номер минимального элемента
 Dim vs As Double    'вспомогательная переменная
 'Ввод размерности массива
 '(вводимое значение преобразуется к целому типу)
 n = CInt(InputBox("Введите размерность массива"))
 ReDim b(1 To n)  'Переопределение массива
 'Цикл с параметром для ввода элементов массива
 For i = 1 To n
  b(i) = CDbl(InputBox("Введите "& i &"-й элемент"))
```

```
 Next i
 'начинаем поиск максимального и минимального
 'элементов начиная с первого
 nom_max = 1: nom_min = 1
 'Начиная со второго, просматриваются все элементы
 For i = 2 To n
  'Каждый элемент сравнивается с максимальным
  'на данный момент элементом. Если текущий элемент
  'оказался больше максимального, то запоминается его
  'номер
   If b(i) > b(nom_max) Then nom_max = i
   'поиск минимального элемента
   If b(i) < b(nom_min) Then nom_min = i
 Next i
 'С помощью переменной vs меняются местами
 'минимальный и максимальный элементы
 vs = b(nom_max)
 b(nom_max) = b(nom_min):b(nom_min) = vs
 'вывод преобразованного массива
 For i = 1 To n
   Worksheets(1).Range("D" & i)= b(i)
 Next i
End Sub
```

3.6 Использование процедур и функций

При решении сложной задачи ее разделяют на более простые и обозримые подзадачи. В программировании сложный код программы разбивают на подпрограммы. *Подпрограмма* – это группа операторов, выполняющих законченное действие. *Основная программа* – программа, реализующая основной алгоритм решения задачи и содержащая в себе обращения к подпрограммам (вызов подпрограмм). В точке вызова функции выполнение программы переходит к подпрограмме, и, выполнив все действия подпрограммы, возвращается в основную программу. В подпрограмме могут быть объявлены собственные переменные, а также параметры подпрограммы для обмена значений с основной программой.

3.6.1 Формат описания и вызова процедур и функций

В *VBA* существуют два типа подпрограмм: подпрограммы-функции и подпрограммы-процедуры. Функция, в отличие от процедуры, возвращает значение и может входить в состав выражений. В приведенных ниже табли-

цах представлены основные форматы описания и вызова подпрограмм – процедур (таблица 3.3) и подпрограмм – функций (таблица 3.4).

Таблица 3.3 – **Формат описания и вызова процедур**

	Подпрограмма-процедура
Описание	**Sub** <имя> **(**[<список параметров>]**)** <операторы> **[Exit Sub]** <операторы> **End Sub**
Вызов в основной программе	1.<имя процедуры> <список аргументов>; 2.**Call** <имя процедуры> [**(**<список аргументов>**)**]

Таблица 3.4 – **Формат описания и вызова функций**

	Подпрограмма-функция
Описание	**Function <**имя функции> [(<список параметров>]) [As <тип функции>] <операторы> **[Exit Function]** <операторы> **<**имя функции> = **<**выражение> **End Function**
Вызов в основной программе	<имя функции> (<список аргументов>)

<Список параметров> отличается от <списка аргументов> тем, что первый указывается при описании подпрограммы, второй – при ее вызове в основной программе.

<Список параметров> позволяет передать в подпрограмму требуемые значения из вызывающей программы и имеет следующий синтаксис:

[**ByRef**|**ByVal**] <имя параметра1> [**As** <Тип>],
[**ByRef**|**ByVal**] <имя параметра2> [**As** <Тип>],
[**ByRef**|**ByVal**] <имя параметра3> [**As** <Тип>], …

<Тип> позволяет явно задать тип передаваемых значений. Если тип опущен, то по умолчанию принимает значение Variant.

Ключевое слово **ByVal** (передача по значению) используется в тех случаях, когда желают, чтобы изменение параметров внутри процедуры не приводило к изменению соответствующих аргументов процедуры в основной программе. Использование **ByRef** (передача по ссылке) или, если ключевое слово опущено, означает, что при изменении параметров внутри про-

цедуры происходит изменение соответствующих параметров в основной программе.

<Список аргументов> перечисляется через запятую. Количество и типы параметров и аргументов должны соответствовать. Аргументы, сответсвующие параметрам с ключевывым словом **ByRef** (или по умолчанию), должны быть переменными.

Подпрограммы принимают для обработки *формальные параметры*, указываемые при объявлении. При вызове они заменяются *фактическими параметрами*, т. е. реально используемыми в вызывающей программе. В *VBA* список формальных параметров подпрограммы представляет имена переменных, разделенных запятой. При этом желательно указать тип каждой переменной.

Для досрочного выхода из подпрограммы и возврата в основную программу, опуская оставшиеся операторы, используется **Exit Sub** (в процедурах) и **Exit Function** (в функциях).

3.6.2 Способы передачи параметров по ссылке и по значению

При передаче переменных в качестве фактических параметров процедуры или функции может использоваться один из двух различных способов: по ссылке (ByRef), по значению (ByVal). Способ передачи указывается при описании аргументов процедуры в строке объявления этой процедуры, т. е. имени аргумента может предшествовать один из явных описателей способа передачи.

Если переменная передается *по ссылке* (по умолчанию), значит, процедуре или функции будет передан адрес этой переменной в памяти. При этом происходит отождествление формального аргумента процедуры и переданного фактического параметра. Тем самым вызываемая процедура может изменить значение фактического параметра: если будет изменен формальный аргумент процедуры, то переданный при вызове ей фактический параметр тоже изменит свое значение.

Если же фактический параметр передается по значению, то формальный аргумент вызываемой процедуры или функции получает только значение фактического параметра, но не саму переменную, используемую в качестве этого параметра. Тем самым изменения формального аргумента не сказывается на значении переменной, являющейся фактическим параметром. Если же явное указание способа передачи параметра отсутствует, то по умолчанию подразумевается передача по ссылке.

Поясним порядок передачу параметров на примере**:**

```
Public Function dist(ByRef x As Double, ByVal y As _
Double) As Double
  x = x + 1
  y = y + 1
  dist = x ^ 2 + y ^ 2
End Function
Public Sub Pr_par()
  Dim x As Double
  Dim y As Double
  x = Val(InputBox("Введите x", "Пример", 0))
  y = Val(InputBox("Введите y", "Пример", 0))
  z = dist(x, y)
  MsgBox ("Для заданных значений x="&x&"и y=" &y &_
Chr(13) & "Получен результат " & z)
End Sub
```

Порядок и пояснение работы программы: если при выполнении программы введены значения: $x = 2$, $y = 5$, то после вызова функции `Dist(2,5)` значения x, y увеличиваются на 1 и становятся соответственно 3 и 6. Далее вычисляется значение: $3^2+6^2 = 9 + 36 = 45$. По возвращении в основную процедуру `Pr_par()` выводится сообщение, представленное на рисунке 3.6. Так как значение переменной x передавалось по ссылке, то ее значение в основной процедуре после вызова функции `Dist()` изменилось (увеличилось на 1) и при выводе сообщения ее значение стало равным 3, а значение переменной y передавалось в функцию `Dist()` по значению, и хотя при вызове функции также увеличилось на 1, но при возврате в основную процедуру – не изменилось.

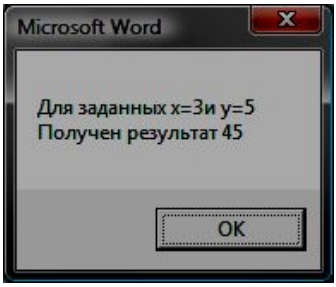


Рисунок 3.6 – Информационное окно работы программы

3.6.3 Создание собственных функций рабочего листа *MS Excel*

Функции рабочего листа, определенные пользователем, – это процедура `Function()`, которую можно указать в формуле, хранящейся в ячейке. Эти функции работают точно так же, как и другие функции, используемые на рабочем листе *Excel*. Таким образом, можно создавать собственные функции, которые будут производить специальные расчеты согласно требуемому пользователем алгоритму.

Функция, определяемая пользователем, разрабатывается точно так же,

как и любая функция *VBA*. Они хранятся в *модуле* и описываются с помощью ключевого слова `Public`.

Приведем пример функции, которая вычисляет значение выражения: $f(x) = x^2 + \sin(x)$.

```
Public Function Func(x As Double)
 Func = x ^ 2 + Sin(x)
End Function
```

Для вызова данной функции, описанной в модуле, вызывается мастер функций и категория **Определенные пользователем**. Далее указывается аргумент данной функции и выполняется автозаполнение ячеек. На рисунке 3.7 представлен пример вызова функции, определенной пользователем и вычисление значений данной функции на интервале [–2; 0,8] с шагом 0,2.

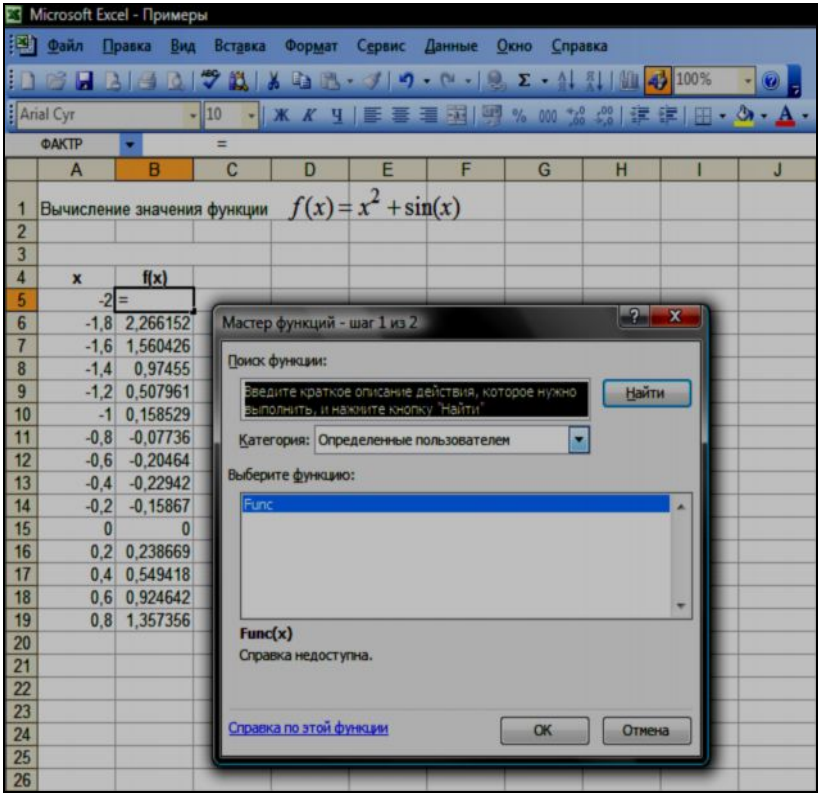


Рисунок 3.7 – Пример вызова функции, определенной пользователем

3.6.4 Использование в *VBA* функций *MS Excel*

Использование в *VBA* функций *MS Excel* можно рассмотреть в двух аспектах:

– вызов функций рабочего листа из программы;

– вставка функции в ячейки рабочего листа с помощью программы *VBA*.

В первом случае, чтобы вызвать функцию рабочего листа из программы *VBA*, следует использовать объект `WorksheetFunction`. Проиллюстрируем технологию вызова функций рабочего листа из программы с помощью строки программы:

```
T_Range = Worksheets("Лист1").Range("A1:A10")
Maximum = Application.WorksheetFunction.Max(T_Range)
```

В данном примере переменная `Maximum` принимает значение наибольшего из чисел, находящихся в ячейках `A1:A10`. Для этого используется функция рабочего листа `Max`. Следует отметить, что в качестве аргумента функции должен использоваться диапазон ячеек, описываемый объектом `Range`.

Во втором случае, чтобы вставить функцию или просто формулу в рабочий лист, следует использовать свойство `Formula` объекта `Range`. Ниже представлена инструкция, которая помещает формулу `"=A1+B1"` в ячейку `C1`:

```
Worksheets("Лист1").Range("C1").Formula = "=A1+B1".
```

3.6.5 Рекурсия. Пример программы с рекурсией

Рекурсия – такой способ организации вспомогательного алгоритма (подпрограммы), при котором эта подпрограмма (процедура или функция) в ходе выполнения обращается сама к себе. Один из классических примеров подпрограмм с рекурсией – это вычисление факториала. Так как с одной стороны факториал определяется как: $n! = 1 \cdot 2 \cdot 3 ... (n-1) \cdot n$, а с другой стороны $n! = \begin{cases} 1, & если\ n = 1 \\ (n-1)! \cdot n, & если\ n > 1 \end{cases}$, то граничным условием в данном случае является $n = 1$. Рассмотрим данную процедуру, используемую для вычисления суммы членов ряда.

Пример 3.11. Вычислить сумму членов ряда $\sum_{i=1}^{n} \frac{1}{i!}$, где $i!$ – факториал числа i (произведение натуральных чисел от 1 до i). Для вычисления факториала применяется рекурсивная функция `faktor()`.

```
Public Sub prog6()
  Dim i As Integer, n As Integer
  Dim s As Double
  n = CInt(InputBox("Введите n"))
  For i = 1 To n
    s = s + 1 / faktor(i)
  Next i
  MsgBox s
End Sub
'рекурсивная функция вычисления факториала
Public Function faktor(x As Integer) As Long
If x = 1 Then
    faktor = 1
    Else: faktor = faktor(x - 1) * x
End If
End Function
```

В приведенной программе при вычислении искомой суммы производится вызов рекурсивной функции `faktor()`. При описании функции типу ее результата присваивается тип длинный целый (`Long`), т. к., например 10! = 3 628 800, что выходит за диапазон типа `Integer`. При выполнении функции результат присваивается ее имени.

3.7 Обработка строк

3.7.1 Формат объявления переменных строкового типа

Строка – упорядоченная последовательность символов. Каждый символ строковой величины занимает 1 байт памяти (код ASCII). Количество символов в строке называется ее длиной.

Строковая константа – последовательность символов, заключенных в кавычки. Например: “это строковая константа” , “272”. Две следующих друг за другом кавычки (“”) обозначают “пустую строку”, т. е. строку нулевой длины.

Строковая переменная описывается в разделе описания переменных:

Dim <идентификатор> **As String**

Например: `Dim Name As String`

3.7.2 Операции над строками

Основные операции, которые применимы над строками, описаны в разд. 1.6, а именно, это операция сцепления (конкатенация) (&) или (+) применяется для соединения нескольких строк в одну результирующую строку, опе-

рации отношения (=, <, >, <=, >=, <>). Отметим, что операции отношения имеют приоритет более низкий, чем операции сцепления. Сравнение строк производится слева направо до первого несовпадающего символа, и та строка считается больше, в которой первый несовпадающий символ имеет больший номер в таблице символьной кодировки. Если строки имеют различную длину, но в общей части символы совпадают, считается, что более короткая строка меньше, чем более длинная. Строки равны, если они полностью совпадают по длине и содержат одни и те же символы.

Кроме этого, существует достаточно большое количество стандартных функций обработки строк. Приведем некоторые из них.

Функция **Left**(S, N) выделяет из строки S подстроку длиной N символов, начиная с левого края строки. Например, S = "Окончательная стоимость" P = Left(S, 14) Результат – строка p = "стоимость ".

Функции **Right**(S, N). Результат – строка длиной N символов, взятых подряд из строки S, начиная с правого края. Например, Right ("ЭВМ IBM-PC",6). Получим результат строку "IBM-PC".

Функция **Len**(S) определяет текущую длину строки S. Результат – значение целого типа. Например, Len("Смета") будет равен 5.

Функция **InStr**(N, S1, S2) обнаруживает первое появление в строке S2 подстроки S1. Поиск начинается с позиции N, этот аргумент необязателен. Результат – целое число, равное номеру позиции, где находится первый символ подстроки S1. Если в S2 подстроки S1 не обнаружено, то результат равен 0. Например, InStr("Abcdef","cd") будет равен 3.

Функция **InStrRev**(Sl, S2[, Nl[, N2]]) возвращает позицию появления строки S2 внутри S1, в направлении от конца (или N1) к началу строки. N2 определяет тип сравнения. В том случае, если N2 опускается, то для поиска используется текущая установка Option Compare.

Функция **Mid**(S, Poz, N). Результат – новая строка длины N из символов строки S, взятых подряд, начиная с позиции Poz. Например, Mid("abcdefg",1,3). Результат: строка "abc".

Функция **LCase**(S) возвращает строку (тип String), содержащую копию S со всеми символами верхнего регистра, преобразованными в символы нижнего регистра. Например, LCase("ПРОВЕРКА"). Результат – "проверка".

Функция **UCase**(S). Возвращает S со всеми символами нижнего регистра, преобразованными в символы верхнего регистра. Например, UCase("ответ"). Результат – "ОТВЕТ ".

Функция **StrConv**(S, N) возвращает строку, преобразованную в новую форму в зависимости от числового кода, заданного аргументом N. *VBA* предоставляет внутренние константы для использования с функцией

StrConv(); наиболее полезными являются: vbProperCase (преобразует строку так, что каждая буква, начинающая слово, становится заглавной), vbLowerCase (преобразует строку в буквы нижнего регистра) и vbUpperCase (преобразует строку в буквы верхнего регистра). Например, StrConv("введите данные", vbProperCase). Результат – "Введите данные".

Функция **LTrim**(S) возвращает копию строки S после удаления начальных пробелов из левой части строки. Например, LTrim(" Вход"). Результат – "Вход".

Функция **StrComp**(S1, S2[, N]) позволяет сравнивать S1 с S2 и возвращает число, обозначающее результат сравнения: –1, если SI < S2; 0, если SI = S2; 1, если SI > S2. Параметр N является необязательным и указывает, следует ли выполнять сравнение с учетом регистра. Если N опускается, строки сравниваются с использованием текущей установки Option Compare.

3.7.3 Примеры программ обработки строк

Пример 3.12. Из данной символьной строки выбрать все цифры и сформировать другую строку из этих цифр, сохранив их последовательность. Например, для исходной строки: "df56gha789yt6u8k88w" должен быть получен результат: "567896888".

Идея алгоритма состоит в следующем: просматриваются все символы исходной строки и проверяется принадлежность каждого символьному интервалу от 0 до 9. Если это условие выполняется, то такой символ присоединяется к строке S2.

```
Sub stroki()
  'Объявление переменных строкового типа
  Dim S1 As String, S2 As String, S As String
  Dim i As Integer
  S1 = InputBox("Введите исходную строку")
  S2 = ""
 'Просмотр всех символов строки
  For i = 1 To Len(S1)
    S= Mid(S1, i, 1) 'Выделение символа из строки S1
    If S >= "0" And S <= "9" Then S2 = S2 + S
  Next i
  MsgBox "Результат: " & S2
End Sub
```

Пример 3.13. Определить, является ли заданная строка палиндромом, т. е. читается одинаково слева направо и справа налево. Например, строка

“abcdcba” является палиндромом, а строка “abcabc” – нет.

Идея созданного алгоритма заключается в просмотре строки одновременно слева направо и справа налево и сравнении соответствующих символов. Если в какой-то момент символы не совпадают, делается вывод о том, что строка не является палиндромом, если же удается достичь середины строки и при этом все соответствующие символы совпали, то строка является палиндромом. Граничные условия – строка является палиндромом, если она пустая или состоит из одного символа.

```
'рекурсивная функция
Public Function pal(Str As String) As Boolean
If Len(Str) <= 1 Then
  pal = True
  Else
   pal = (Mid(Str,1,1))= Mid(Str,Len(Str),1) And _
pal(Mid(Str, 2, Len(Str) - 2))
End If
End Function
Public Sub stroka1()
 Dim Stroka As String
 Stroka = InputBox("Введите строку ", "Пример")
 If pal(Stroka) Then
    MsgBox "Введенная строка - палиндром"
    Else:MsgBox "Введенная строка - непалиндром"
 End If
End Sub
```

Контрольные вопросы

1 Каков формат использования инструкции `if`?
2 Каков формат использования инструкции `Select Case`?
3 Какова альтернатива использования стандартных конструкций разветвления?
4 Как организовать циклический пересчет на *VBA*?
5 Чем отличаются циклы с предусловием от циклов с постусловием?
6 Каков формат использования циклов с предусловием?
7 Каков формат использования циклов с постусловием?
8 Как описать процедуру (функцию) на *VBA*? Что такое рекурсия?
9 Что такое список формальных (фактических) параметров?
10 Что означают служебные слова `ByRef`, `ByVal`?
11 Как объявить переменную строкового типа? Как выделить подстроку из строки?

4 СОЗДАНИЕ ФОРМЫ И ЭЛЕМЕНТОВ УПРАВЛЕНИЯ

Заметим, что *Visual Basic for Application* относится к числу языков программирования, реализующих современные принципы программирования. Одним из основных таких принципов является концепция визуального программирования, ориентированная на ускорение процесса разработки и программ и создание удобного пользовательского интерфейса. Отметим, что языки визуального программирования получили широкое распространение именно благодаря широким возможностям по созданию удобного интерфейса с минимальными усилиями, так как раннее на это уходило от 60 до 90 % всего времени разработки программы, а остальная часть на программирование расчетов. Основным средством для создания пользовательского интерфейса является пользовательская форма с различными элементами управления на ней.

Элемент управления – это объект, помещаемый пользователем в форму, который имеет собственный набор свойств, методов и событий. Элементы управления используются для приема данных, вводимых пользователем, отображения данных и запуска процедур (подпрограмм – функционально завершенных частей команд языка, имеющих одно имя) обработки события. Большинство элементов управления могут обрабатываться с помощью методов. Некоторые элементы управления являются интерактивными (откликающимися на действия пользователя), тогда как другие являются статическими (доступными только в программе).

Сущность концепции объектно-ориентированного программирования заключается в том, что вся программа создается из объектов (это элемент программы, который минимально зависит от данных), через которые пользователь или другой объект выполняет определенные команды программы для достижения результата. Объекты характеризуются свойствами (характеристики объекта), методами (определяют действия, которые можно производить над объектом) и событиями (возникают при работе с объектом или других определенных действиях пользователя, иногда как результат действия операционной системы, это реакция объекта). Объекты объединяются в классы (совокупность объектов, различающихся только значениями свойств).

4.1 Размещение элемента управления на рабочем листе *MS Excel*

Элементы управления являются объектами. Поэтому, как любые объекты, они обладают свойствами, методами и событиями. Элементы управления создаются при помощи панели инструментов **Элементы управления**

(*Control Toolbox*). На этой панели представлены кнопки, позволяющие конструировать элементы управления, а также кнопки вызова окна свойств и перехода в режим конструктора и редактор кода.

При этом отметим, что элементы управления можно размещать не только на форме, но и на рабочем листе или в форме. Иногда используется программное создание элементов управления в процессе работы приложения, но это применяется достаточно редко.

Большинство элементов управления можно располагать, как уже говорилось, как на рабочем листе, так и в форме, но существуют такие элементы управления, например как `RefEdit` (набор страниц и набор вкладок), которые можно располагать только в форме.

Для размещения элемента управления на листе необходимо нажать на соответствующую кнопку на панели инструментов **Элементы управления** и с помощью мыши перетащите рамку элемента управления в нужное место. После этого элемент управления можно перемещать, изменять его размеры, копировать в буфер обмена и вставлять из буфера обмена.

Для удобства работы с элементами управления в период их конструирования в *Excel* введен режим конструктора, который активизируется нажатием соответствующей кнопки. В режиме конструктора (*Design Mode*) отключена реакция элемента управления на события, поэтому при включенном режиме конструктора можно видоизменять элемент управления и задавать его свойства. После того как пользователь решит, что созданный элемент управления имеет тот вид, который ему нужен, и все требуемые свойства элемента управления установлены, он должен отключить режим конструктора повторным нажатием кнопки **Режим конструктора** (*Design Mode*).

Рассмотрим порядок создание элемента управления на примере кнопки, размещенной на рабочем листе.

Пример 4.1. Пусть необходимо построить таблицу значений функции $f(x) = (x-5)^2 + \sqrt[3]{(x-1)^2}$ на некотором отрезке и с указанным шагом. Вычисления значений функции производить по нажатию кнопки, размещенной на рабочем листе *Excel*.

После определения исходных данных и размещения в режиме конструктора кнопки (рисунок 4.1), необходимо набрать соответствующую процедуру выполнения вычислений. Далее необходимо назначить данную процедуру кнопке. Для этого необходимо в режиме конструктора в контекстнозависимом меню выбрать команду **Назначить макрос** и выбрать соответствующую процедуру, текст которой приведен ниже.

```
Sub z1()
  X0 = Sheets(1).Cells(4, 2)
  XN = Sheets(1).Cells(5, 2)
  DX = Sheets(1).Cells(6, 2)
  X = X0
  N = 10 'начальная строка
 For X = X0 To XN Step DX
   Y = (X - 5)^2 + (X - 1)^(2/3)
   Sheets(1).Cells(N, 1) = X
   Sheets(1).Cells(N, 2) = Y
   N = N + 1:X = X + DX
 Next X
End Sub
```

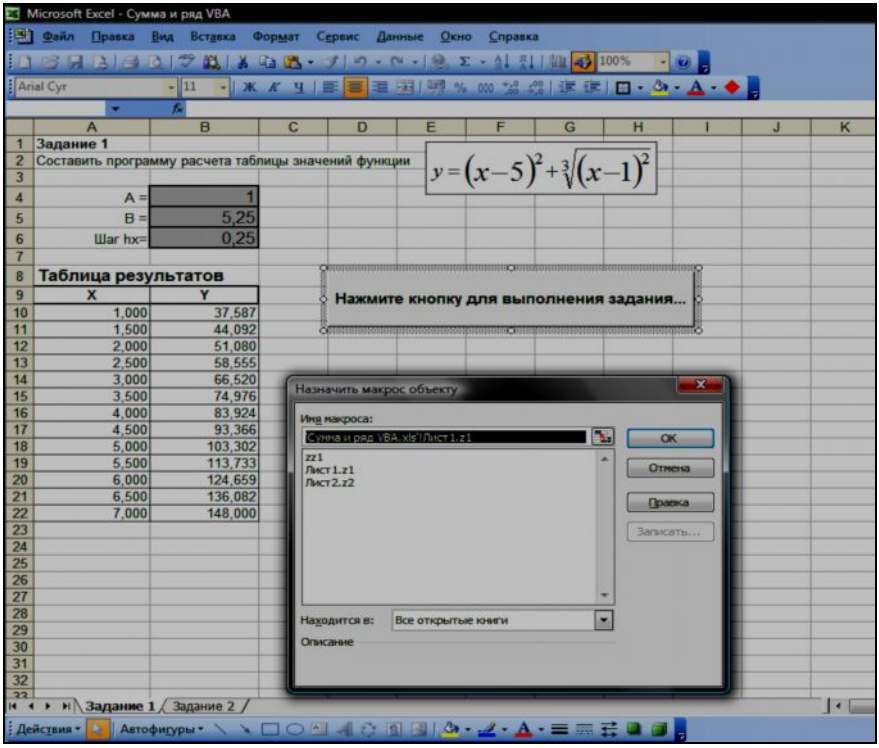


Рисунок 4.1 – Создание элемента управления `CommandButton` на рабочем листе

4.2 Создание формы

Формой называют пользовательское окно или объект пользовательского интерфейса класса `UserForm`. Для создания этого объекта необходимо вы-

брать команду `UserForm` из меню **Вставка**. В результате появится окно с названием «`UserForm1`» (рисунок 4.2).

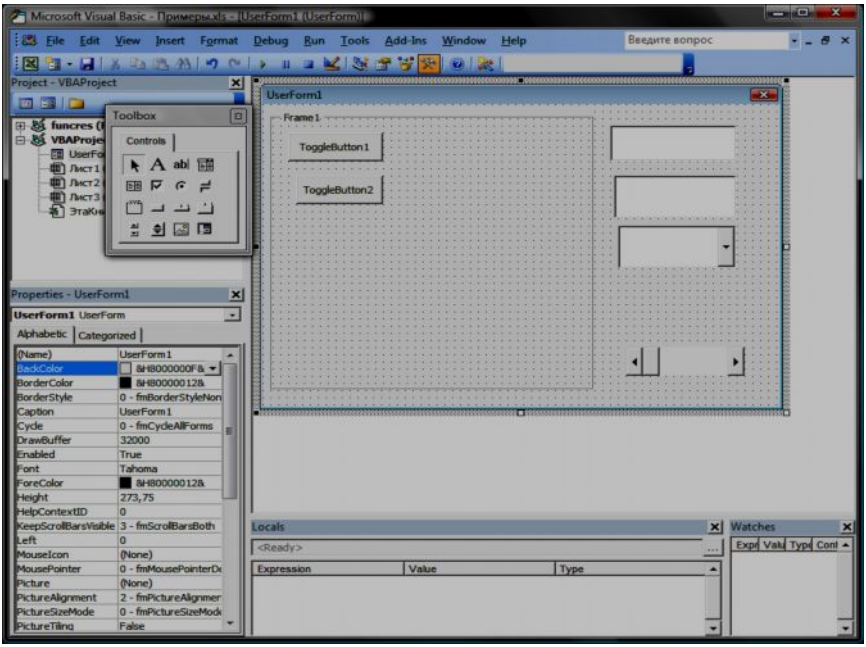


Рисунок 4.2 – Пользовательская форма в режиме конструктора

Это и есть объект, на котором могут располагаться любые другие объекты пользовательского интерфейса. По своей сути форма (или пользовательская форма) представляет собой диалоговое окно, в котором можно размещать различные элементы управления. В приложении может быть как одна, так и несколько форм.

4.2.1 Основные свойства интерфейсного объекта *UserForm*

Для просмотра и изменения свойств любого объекта необходимо расположить на экране окно свойств (рисунок 4.3). Если его нет, то необходимо выбрать из меню **Вид** команду **Окно свойств** или нажать клавишу **F4** либо кнопку **Окно свойств**  на панели инструментов **Стандарт**. В данном окне есть заголовок, где располагается название окна и название объекта, свойства которого отображаются (на рисунке 4.3). Далее следует выпадающее меню, где можно выбрать все созданные доступные объекты. Надпись в выпадающем меню «**`UserForm2`** `UserForm`» означает, что объект

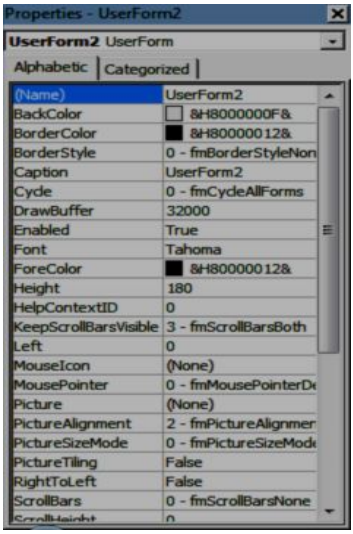


Рисунок 4.3 – Окно свойств объекта *UserForm*

`UserForm2` относится к классу `UserForm`. Ниже расположены две вкладки, которые все свойства могут показать свойства: *по алфавиту* и *по категориям*.

Затем в окне отображается область из двух колонок. В левой колонке отображаются имена свойств, а в правой – их значение. Например, свойство `BorderColor` (цвет границы) имеет значение `&H80000012&` (черный цвет). Нажав на кнопку справа от этого значения, можно вызвать палитру цветов и выбрать другой необходимый цвет. Отметим, свойство `Name` – имя объекта, является особенным, поэтому оно заключено в круглые скобки. Именно по этому имени (точнее по значению этого свойства, в нашем случае – `UserForm2`) другие объекты будут распознавать данный объект. Существует также другое название данного свойства – *идентификатор*. Значение этого свойства можно изменить. По умолчанию *Visual Basic* формирует имена следующим образом: к названию класса, к которому относится объект, добавляется первый свободный номер (начиная с единицы). При изменении имени свойства `Name`, необходимо соблюдать требования, предъявляемые к идентификаторам.

Следующее свойство, которое обычно изменяют, – это заголовок окна. Заголовок окна хранит свойство `Caption`. Отметим, измененные свойства сразу отражаются на виде объекта, что характерно только для языков визуального программирования.

Свойства `Width` и `Height` задают ширину и высоту объекта соответственно. С помощью свойства `BackColor` можно изменить цвет фона. Для изменения свойства `BorderColor` (цвет границы) необходимо изменить значение свойства – `BorderStyle`. Если оно установлено в значение 0, то это означает, что границы нет, если – 1, то появится граница окна, что тут же отразится на виде нашего объекта.

Рассмотрим еще одно интересное свойство окна – `StartUpPosition` – положение окна на экране при появлении. Свойство может принимать одно из четырех значений:

– 0 (Manual) – положение задается вручную, с помощью свойств Left – отступ от левого края и Top – отступ от верхнего края окна, в котором появляется данное окно (или другой объект);

– 1 (CenterOwner) – окно будет располагаться по центру окна, в котором оно появляется;

– 2 (CenterScreen) – окно будет появляться по центру экрана;

– 3 (WindowsDefault) – окно будет появляться на основании установок операционной системы.

Свойство Picture указывает рисунок, отображаемый как фон формы.

Заметим, что объект UserForm1 в режиме редактирования (конструктора) покрыт сеткой точек (сетку при необходимости можно убрать). К этим точкам привязываются все объекты, которые будут располагаться на UserForm1.

Приведем наиболее часто используемые методы объекта UserForm:

- Show – отображает форму на экране;
- Hide – закрывает форму;
- Move – изменяет положение и размер формы;
- PrintForm – печатает изображение формы.

Чтобы скрыть элементы управления формы UserForm, необходимо свойству visible элемента управления присвоить значение False:

```
TextBox1.Visible = False
```

4.3 Основные элементы управления и их свойства

В *VBA* имеется обширный набор встроенных элементов управления. Используя этот набор и редактор форм, нетрудно создать любой пользовательский интерфейс, который будет удовлетворять всем требованиям, предъявляемым к интерфейсу в среде *Windows*. Элементы управления являются объектами. Как любые объекты, они обладают свойствами, методами и событиями. Элементы управления создаются при помощи Панели элементов, которая отображается на экране либо выбором команды **Вид (***View***) – Панель элементов (***Toolbox***),** либо нажатием кнопки  на панели инструментов Standard**.** На этой панели представлены кнопки, позволяющие конструировать элементы управления. Для создания элементов управления служат все кнопки панели инструментов, за исключением кнопки **Выбор объекта**  **.** Щелкнув по кнопке **Выбор объекта**, можно выбрать уже соз-

данный в форме элемент управления для последующего его редактирования (изменения размеров или редактирования).

Таким образом, используя **Панель элементов** из незаполненной формы, можно сконструировать любое требуемое для приложения диалоговое окно. Размещение нового управляющего элемента в форме осуществляется следующем порядке:

1 Щелкните значок того элемента, который необходимо разместить на форме.

2 Поместите указатель мыши на то место, где будет располагаться управляющий элемент.

3 Нажмите левую кнопку мыши и, не отпуская ее, растяните появившийся прямоугольник до требуемых размеров.

4 Отпустите кнопку мыши. Элемент управления – создан.

Размеры формы и расположенных на ней элементов управления можно изменять. Технология изменения размеров стандартная для *Windows*: выделить изменяемый элемент, разместить указатель мыши на одном из размерных маркеров и протащить его при нажатой левой кнопки мыши так, чтобы объект принял требуемые размеры. Окно редактирования форм поддерживает операции буфера обмена. Таким образом, можно копировать, вырезать и вставлять элементы управления, расположенные на поверхности формы. Для облегчения размещения и выравнивания элементов управления используется сетка. Активизировать ее можно с помощью вкладки **Общие** (*General*) диалогового окна **Параметры** (*Options*), вызываемого командой **Сервис**, **Параметры** (*Tools, Options*), там же устанавливается шаг сетки. Кроме того, команды меню **Формат** (*Format*) автоматизируют и облегчают процесс выравнивания элементов управления как по их взаимному местоположению, так и по размерам (рисунок 4.4).

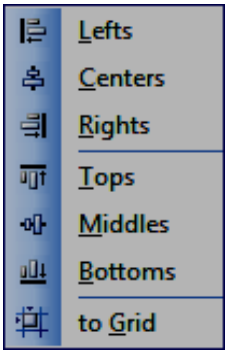


Рисунок 4.4 – Панель выравнивания взаимоположения элементов управления на форме

В таблице 4.1 приведен список основных элементов управления, которые могут быть размещены на пользовательской форме.

Таблица 4.1 – **Список основных элементов управления**

Элемент управления	Имя	Кнопка
Поле	`TextBox`	ab|
Надпись	`Label`	A

Окончание таблицы 4.1

Элемент управления	Имя	Кнопка
Кнопка	`CommandButton`	
Список	`ListBox`	
Поле со списком	`ComboBox`	
Полоса прокрутки	`ScrolBar`	
Счетчик	`SpinButton`	
Переключатель	`OptionButton`	
Флажок	`CheckBox`	
Выключатель	`ToggleButton`	
Рамка	`Frame`	
Рисунок	`Image`	
Набор страниц	`MultiPage`	
Набор вкладок	`TabStrip`	

Для размещения элемента управления на лист или в форму необходимо нажать соответствующую кнопку на панели элементов и с помощью мыши перетащить рамку элемента управления в нужное место. После этого элемент управления можно перемещать, изменять его размеры, копировать в буфер обмена, вставлять из буфера обмена и удалять из формы. В таблице 4.2 приведены основные свойства элементов управления.

Таблица 4.2 – **Основные общие свойства элементов управления**

Свойство	Описание
`Caption`	Надпись, отображаемая при элементе управления
`AutoSize`	Допустимые значения: `True` (устанавливает режим автоматического изменения размеров элемента управления так, чтобы на нем полностью помещался текст, присвоенный свойству `Caption`) и `False` (в противном случае)
`Visible`	Допустимые значения: `True` (элемент управления отображается во время выполнения программы) и `False` (в противном случае)
`Height и Width`	Устанавливают геометрические размеры объекта (высоту и ширину)
`Enabled`	Допустимые значения: `True` (пользователь вручную может управлять элементом управления) и `False`

Окончание таблицы 4.2

Свойство	Описание
Left и Top	Устанавливают координаты верхнего левого угла элемента управления, определяющие его местоположение в форме
ControlTipText	Устанавливает текст в окне всплывающей подсказки, связанной с элементом управления. В следующем примере элементу управления CommandButton назначен текст, всплывающей подсказки "Это кнопка": CommandButton1.ControlTipText = "Это кнопка"
BackColor, ForeColor, BorderColor	Устанавливают цвет заднего и переднего планов элемента управления, а также его границы
BackStyle	Устанавливает тип заднего фона
BorderStyle	Устанавливает тип границы. Допустимые значения: fmBorderStyleSingle (граница в виде контура); fmBorderStyleNone (граница невидима)
SpecialEffect	Устанавливает тип границы. Отличается от свойства BorderStyle тем, что позволяет установить несколько типов, но одного цвета. BorderStyle позволяет установить только один тип, но различных цветов
Picture (удаление картинки)	После того как картинка создана на элементе управления, иногда возникает необходимость ее удалить. CommandButton1.Picture = LoadPicture("")
Picture (создание картинки)	Внедряет картинку на элемент управления. Например, на поверхности кнопки картинка отображается с помощью следующей инструкции: CommandButton1._ Picture =LoadPicture("c:\my doc\Круг.bmp") Функция LoadPicture (ПолноеИмяФайла) считывает графическое изображение. Аргумент ПолноеИмяФайла указывает полное имя графического файла
Picture (удаление картинки)	После того как картинка создана на элементе управления, иногда возникает необходимость ее удалить. Это легко достигается присвоением свойству Picture значения LoadPicture(""): CommandButton1.Picture = LoadPicture("")

Кроме общих свойств у каждого элемента управления есть свои индивидуальные свойства, которые определяют специфику функционирования каждого элемента. В таблице 4.3 приведены свойства наиболее часто используемых элементов управления, описано их назначение и основные свойства.

Таблица 4.3 – **Описание свойств основных элементов управления**

Свойство	Описание
TextBox(поле) используется для ввода текста пользователем или для вывода в него результатов расчетов программ	
Text	Возвращает текст, содержащийся в поле
Multiline	Допустимые значения: True (устанавливает многострочный режим ввода текста в поле) и False (однострочный режим)
WordWrap	Допустимые значения: True (устанавливает режим автоматического переноса) и False (в противном случае)
Label(надпись) используется для отображения надписей, например, заголовков элементов управления, не имеющих свойства Caption	
Multiline	Допустимые значения: True (устанавливает многострочный режим ввода) и False (однострочный режим)
WordWrap	Допустимые значения: True (устанавливает режим автоматического переноса) и False (в противном случае)
CommandButton (кнопка) используется для инициирования выполнения некоторых действий, вызываемых нажатием кнопки, например запуск программы или остановка ее выполнения, печать и т. д.	
Cancel	Допустимые значения: True (устанавливаются отменяющие функции для кнопки, т. е. нажатие клавиши <Esc> приводит к тем же результатам, что и нажатие кнопки) и False
Accelerator	Назначает клавишу, при нажатии на которую одновременно с клавишей <Alt> происходит запуск действий, связанных с кнопкой. Например: CommandButton1.Accelerator="C"
Default	Задаёт кнопку по умолчанию, т. е. устанавливает ту кнопку, для которой действия, связанные с ней, будут выполняться при нажатии клавиши <Enter>
Frame(рамка) используется для визуальной группировки элементов управления	
OptionButton(переключатель) позволяет выбрать один из нескольких взаимоисключающих параметров. Переключатели обычно отображаются группами, обеспечивая возможность выбора альтернативного варианта	
Value	Возвращает True, если переключатель выбран и False – в противном случае

4.4 Общие методы и события элементов управления

Событийное программирование предполагает наличие возможностей по управлению выполнением задач со стороны пользователя посредством воз-

можностей операционной системы (в нашем случае *Windows*) через события объектов. **Событие (прерывание)** – способность объекта реагировать на события, которые могут исходить от пользователя программы и представляют собой программы. Программы обслуживания событий программируются в отличие от методов объектов теми, кто непосредственно использует эти объекты в своих программах.

Наиболее часто используемым событием является событие `Click`. Это событие наступает, когда пользователь производит щелчок кнопкой мыши. Рассмотрим ситуацию, когда при щелчке мышью на форме Form1, необходимо изменить цвет фона формы по случайному закону. Программа обработки этого события:

```
Private Sub UserForm_Click( )
  UserForm1.BackColor = QBColor(15*Rnd( ))
End Sub
```

Можно организовать считывание координат курсора мыши и вывода их в активные элементы управления типа метка `Label1` и `Label2`. Для решения этой задачи существует событие `MouseMove`. Программа обработки этого события:

```
Private Sub UserForm_MouseMove(ByVal Button As _
Integer, ByVal Shift As Integer, ByVal X As Single,_
ByVal Y As Single)
  Label1.Caption = "X=" + Str(Int(X))
  Label2.Caption = "Y=" + Str(Int(Y))
End Sub
```

В этой программе используется функция `Str`() для преобразования числовых значений координат курсора мыши в строковое выражение.

Приведем наиболее часто используемые события объектов *VBA* и *Windows* (таблица 4.4).

Таблица 4.4 – **Основные общие события для элементов управления**

Событие	Описание
`Click`	Происходит, когда пользователь выбирает элемент управления с помощью одинарного щелчка кнопкой мыши
`KeyPress`	Происходит, когда пользователь нажимает любую клавишу на клавиатуре, кроме функциональных и клавиш управления курсором
`DblClick`	Происходит, когда пользователь выбирает элемент управления с помощью двойного щелчка кнопкой мыши

Окончание таблицы 4.4

Событие	Описание
`Change`	Происходит при изменении значения элемента управления
`GotFocus (LostFocus)`	Происходит, когда элемент управления получает (теряет) фокус
`Error`	Используется при уведомлении об ошибке
`Activate`	Загрузка объекта для выполнения (для объекта `UserForm`)
`Terminate`	Закрытие объекта (для объекта `UserForm`)
`MouseMove`	Передвижение указателя мыши по объекту

VBA является "дружественной" средой программирования и для исключения ошибок при использовании свойств и методов объектов после ввода имени и точки существующего объекта предоставляет список свойств и методов этого объекта, что существенно облегчает использование объектно-ориентированной технологии. В таблице 4.5 представлен список методов, используемых для многих элементов управления.

Таблица 4.5 – **Основные общие методы элементов управления**

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

4.5 Пример создания пользовательской формы с элементами `TextBox`, `CommandBotton`

Пример 4.2. В качестве примера работы с формой создадим простое приложение, вычисляющее значение функции $f(x) = 2\cos(2x) + \sin(x)$ в заданной точке. На форме задается значение аргумента x и по нажатию кнопки вычисляется значение $f(x)$. Для выполнения задания необходимо:

1 Добавить пользовательскую форму в проект. Для этого необходимо выполнить команду `Insert – UserForm`.

2 Расположить на форме следующие элементы управления (рисунок 4.5).

Рисунок 4.5 – Вид формы в режиме конструктора

На форму внесено пять элементов управления: кнопка, два поля и надписи. Опишем их назначение и основной механизм взаимодействия:

CommandButton1(кнопка)– при нажатии на кнопку запускается процедура обработки события (Private Sub Command-Button1 _Click()), которая считывает значение аргумента из поля TextBox1. Проверяется, введено ли в это поле число. Если введено не число, то на экране отображается соответствующее сообщение, прерывается выполнение процедуры, и фокус (курсор) устанавливается на поле TextBox1, предлагая исправить вводимые данные. Если введено число, то находится значение функции при введенном значении аргумента, результат выводится во второе поле TextBox2;

TextBox1(поле) – поле для ввода значения аргумента. При вводе значения должна появиться всплывающая подсказка “Введите число”;

Label1(надпись)– пояснительная надпись для поля ввода аргумента;

TextBox2(поле) – поле вывода значения функции. Поле сделаем недоступным для пользователя, т. е. пользователь не сможет ни ввести, ни скорректировать данные в этом поле;

Label2(надпись) – пояснительная надпись для поля вывода.

3 Набрать соответствующие процедуры для каждого элемента управления и инициализации формы в том числе. Для написания кода программы, связанного с пользовательской формой, достаточно дважды щелкнуть, например кнопку CommandButton1. Откроется редактор кода на листе модуля UserForm4. Более того, он откроется на том месте, где программируются действия, связанные с элементом управления, который был дважды нажат. Если код еще не набран, то при открытии редактора кода появятся инструкции заголовка и окончания процедуры, которая будет связана с элементом управления.

Приведем код программы, который должен выполняться при нажатии кнопки:

```
Private Sub CommandButton1_Click()
 'Проверка, является ли введенное значение числом
  If Not IsNumeric(TextBox1.Text) Then
     'Вывод окна сообщения
    MsgBox "Аргумент должен быть числом", vbExclamation
     'Фокус (курсор) устанавливается на поле TextBox1
    TextBox1.SetFocus
    Exit Sub 'Досрочный выход из процедуры
  End If
'При считывании числа из поля ввода при помощи функции
'CDbl строку преобразуем в число
x = CDbl(TextBox1.Text)
```

```
‘Вычисление значение функции
y = 2*Cos(2*x)+sin(x)
‘Вывод результата в поле
TextBox2.Text = CStr(y) 'переводим число в строку
End Sub
```

Процедура инициализации формы, которая конструирует форму до ее загрузки:

```
Private Sub UserForm_Initialize()
  'Изменение заголовка формы
  UserForm1.Caption = "Вычисление значения функции"
  'Пояснения к полям ввода и вывода
  Label1.Caption = "Введите значение x"
  TextBox1.ControlTipText = “Введите число”
  Label2.Caption = "Значение функции y(x)"
  'Изменение надписи на кнопке
  CommandButton1.Caption = "Вычислить"
  'Сделать поле TextBox2 недоступным для пользователя
  TextBox2.Enabled = False
End Sub
```

Отметим, что при создании формы можно обойтись без процедуры инициализации, изменив названия надписей (Caption) элементов управления непосредственно в окне свойств.

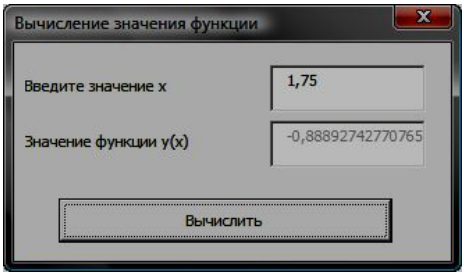


Рисунок 4.6 – Работа с пользовательской формой

После конструирования формы и написания кода в модуле формы выберем команду Run – Run Sub/UserForm и форма отобразится поверх активного рабочего листа *Excel* (рисунок 4.6). Для работы с формой необходимо ввести значение аргумента и нажать на кнопку для выполнения вычисления значения функции **Вычислить**.

4.6 Использование элемента управления – список ListBox

Приведем часто используемые свойства элемента управления ListBox (список), применяемого для хранения списка значений. Из списка пользователь может выбрать одно или несколько значений, которые в последующем будут использоваться в тексте программы (таблица 4.6).

Таблица 4.6 – **Основные свойства элемента управления `ListBox`**

Свойство	Описание
`ListIndex`	Возвращает номер текущего элемента списка. Нумерация элементов списка начинается с нуля
`ListCount`	Возвращает число элементов списка
`TopIndex`	Возвращает элемент списка с наибольшим номером
`ColumnCount`	Устанавливает число столбцов в списке
`TextColumn`	Устанавливает столбец в списке, элемент которого возвращается свойством `Text`
`Enabled`	Допустимые значения: `True` (запрещен выбор значения из списка пользователем) и `False` (в противном случае)
`Text`	Возвращает выбранный в списке элемент
`List`	Возвращает элемент списка, стоящий на пересечении указанной строки и столбца: `List(row, column)`
`RowSource`	Устанавливает диапазон, содержащий элементы списка
`ControlSource`	Устанавливает диапазон (ячейку), куда возвращается выбранный элемент из списка
`MultiSelect`	Устанавливает способ выбора элементов списка. Допустимые значения: – `fmMultiSelectSingle` (выбор только одного элемента); – `fmMultiSelectMulti` (разрешен выбор нескольких элементов посредством либо щелчка, либо нажатием клавиши <Пробел>); – `fmMultiSelectExtended` (разрешено использование клавиши <Shift> при выборе ряда последовательных элементов списка)
`Selected`	Допустимые значения: `True` (если элемент списка выбран) и `False` (в противном случае). Используется для определения выделенного текста, когда свойство `MultiSelect` имеет значение `fmMultiSelectMulti` или `fmMultiSelectExtended`

`ColumnWidths`	Устанавливает ширину столбцов списка. Синтаксис:
	`ColumnWidths = String`,
	`String` – строка, устанавливающая ширину столбцов. В следующем примере устанавливается ширина каждого из трех столбцов списка:
	`With ListBox1`
	`. ColumnCount = 3`
	`. ColumnWidths = "20;30;30"`
	`End With`

Окончание таблицы 4.6

Свойство	Описание
ColumnHeads	Допустимые значения: True (выводятся заголовки столбцов раскрывающегося списка) и False (в противном случае)
ListStyle	Устанавливает способ выделения выбранных элементов. Допустимые значения: – fmListStylePlain (выбранный элемент из списка выделяется цветом); – fmListStyleOption (перед каждым элементом в списке располагается флажок и выбор элемента из списка соответствует установке этого флажка)
MatchEntry	Выводит первый подходящий элемент из списка при наборе его имени на клавиатуре. Допустимые значения: – fmMatchEntryNone (режим вывода подходящего элемента в списке отключен); – fmMatchEntryFirstLetter (выводит подходящий элемент по набранной первой букве. В этом случае, предпочтительно, чтобы элементы списка были упорядочены в алфавитном порядке); – fmMatchEntryComplete (выводит подходящий элемент по полному набранному имени)
BoundColumn	Устанавливает тип, возвращаемый свойством Value. А именно, если свойство BoundColumn равно 0, то свойство Value возвращает индекс выбранной строки, т. е. в этом случае оно действует как свойство ListIndex. Если свойство BoundColumn принимает значение из диапазона от 1 до количества столбцов в списке, то свойство Value возвращает элемент из выбранной строки, стоящий в столбце, определенном свойством BoundColumn

Наиболее часто используемые методы элемента управления ListBox приведены в таблице 4.7.

Таблица 4.7 – **Методы элемента управления ListBox**

Метод	Описание
Clear	Удаляет все элементы из списка
RemoveItem	Удаляет из списка элемент с указанным номером. Синтаксис: RemoveItem (index), где index – номер удаляемого из списка элемента

Окончание таблицы 4.7

Метод	Описание
`AddItem`	Добавляет элемент в список. Синтаксис: `AddItem ( [ Item [, varIndex]])`, где `Item` – элемент (строковое выражение), добавляемый в список;

Для заполнения списка можно воспользоваться одним из следующих способов:

1) поэлементно, если список состоит из одной колонки:

```
With ListBox1
    .AddItem "Июнь"
    .AddItem "Июль"
    .AddItem "Август"
End With
```

2) массивом, если список состоит из одной колонки:

```
With ListBox1
    .List = Array("Июнь", "Июль", "Август")
End With
```

3) из диапазона, в который предварительно введены элементы списка. Результат выбора (индекс выбранной строки) выводится в ячейку `C1`:

```
With ListBox1
    .ColumnCount=2
    .RowSource="A1:B4"
    .ContrSource="C1"
    .BoundColumn=0
End With
```

4) поэлементно, если список состоит из нескольких колонок, например двух:

```
With ListBox1
   ColumnCount =2
   AddItem
   List(i, 0) = XX(i)
   List(i, 1) = Y(i)
End With
```

4.6.1 Пример создания приложения с обработкой выбранных в списке значений

Пример 4.3. Создадим приложение, которое обрабатывает список значений и позволяет подсчитать сумму, произведение, минимальное или мак-

симальное значения среди выбранных в списке элементов. Проектируемая пользовательская форма представлена на рисунке 4.7.

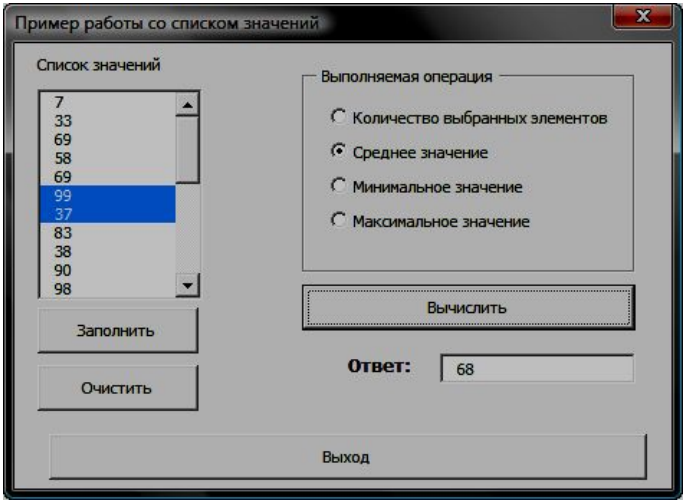


Рисунок 4.7 – Пользовательская форма для работы со списком значений

Опишем назначение и принцип взаимодействия элементов управления на представленной форме:

`ListBox` (список) – список значений из целых чисел, которые можно выделять. На основе выделенных значений и выбранных переключателей формируется ответ;

`CommandButton1` (кнопка) – кнопка с надписью "**Заполнить**". Нажатие на эту кнопку запускает процедуру обработки события (`Private Sub CommandButton1_Click()`), которая должна заполнить список случайными целыми числами в интервале от 1 до 100;

`CommandButton2` (кнопка) – кнопка с надписью "**Очистить**". Нажатие на эту кнопку запускает процедуру обработки события (`Private Sub CommandButton2_Click()`), которая должна очистить список, что можно реализовать, используя метод `ListBox1.Clear`;

`CommandButton3` (кнопка) – кнопка с надписью "**Выполнить**". Нажатие на эту кнопку запускает процедуру обработки события (`Private Sub CommandButton3_Click()`), которая выполняет заполнение поле `TextBox1` с пояснением "**Ответ**:" в зависимости от выбранного переключателя: `OptionButton1`, `OptionButton2`, `OptionButton3`, `OptionButton4`;

CommandButton4 (кнопка) – кнопка с надписью “**Выход**”. Нажатие на кнопку прячет форму;

TextBox1 (поле) с пояснением “**Ответ**:”. В это поле будет выводиться результат. Поле должно быть недоступным для пользователя, т. е. пользователь не сможет ни ввести, ни скорректировать данные в нем;

Label1 (надпись) – пояснительная надпись для поля вывода ответа;

Frame1 (рамка). Используется для группировки переключателей;

OptionButton1 (переключатель) с надписью “Количество выбранных элементов”;

OptionButton2 (переключатель) с надписью “Среднее значение”;

OptionButton3 (переключатель) с надписью “Минимальное значение”;

OptionButton4 (переключатель) с надписью “Максимальное значение”.

На основе созданной в конструкторе формы необходимо набрать для данной формы программный код, представленный ниже.

Процедура заполнения списка при нажатии на кнопку “**Заполнить**”:

```
Private Sub CommandButton1_Click()
Randomize
For i = 0 To N - 1
    'генерация массива случайных чисел
    A(i) = Rnd * 100 + 1
    'добавление элемента массива в список
    ListBox1.AddItem (A(i))
Next i
End Sub
```

Процедура “очищения” списка значений при нажатии на кнопку “**Очистить**”:

```
Private Sub CommandButton2_Click()
   ListBox1.Clear
End Sub
```

Процедура выполнения вычислений в зависимости от значения переключателей при нажатии на кнопку “Вычислить”:

```
Private Sub CommandButton3_Click()
  Dim i As Integer, k As Integer
  Dim b(0 To N - 1) As Integer
  k = 0
  'выход из процедуры, если нет выделенных элементов
  If ListBox1.ListIndex = 0 Then Exit Sub
  'формирование нового массива из выделенных элементов
  'списка
  For i = 0 To N - 1
```

```
        If ListBox1.Selected(i) = True Then
          b(k) = A(i)
          k = k + 1
        End If
  Next i
'k – количество выделенных элементов
'Организация вычислений в зависимости от
'переключателей
  'проверка, выбран ли переключатель: количество
  'выбранных элементов
  If Frame1.OptionButton1.Value = True Then
    TextBox1.Text = CStr(k)
  End if
  'проверка, выбран ли переключатель: среднее значение
  If Frame1.OptionButton2.Value = True Then
     TextBox1.Text = CStr(avr(b, k))
  End if
  'проверка, выбран ли переключатель: максимальное
  'значение
  If Frame1.OptionButton3.Value = True Then
     TextBox1.Text = CStr(min_max(0, b, k))
  End if
  'проверка, выбран ли переключатель: минимальное
  'значение
  If Frame1.OptionButton4.Value = True Then
     TextBox1.Text = CStr(min_max(1, b, k))
  End if
End Sub
```

Функция `min_max()` определения минимального или максимального значения в массиве $b()$, размерности N в зависимости от параметра p (при $p = 1$ функция возвращает максимальное значение, в противном случае минимум):

```
Public Function min_max(p As Byte, b() As Integer, N_
As Integer) As Integer
   Dim min As Integer
   Dim max As Integer
   Dim i As Integer
   min = 0 : max = 0
   For i = 1 To N - 1
      If b(i) > b(max) Then max = i
      If (b(i) < b(min)) Then min = i
   Next i
If p = 1 Then min_max = b(max) Else min_max = b(min)
```

```
End Function
```

Функция `avr()` определения среднего значения в массиве выделенных элементов *b*(), размерности *N*:

```
Public Function avr(b() As Integer, N As Integer) As_
Double
   Dim i As Integer
   Dim s As Integer
   s = 0
   For i = 0 To N - 1
      s = s + b(i)
   Next i
   avr = s / N
End Function
```

Процедура закрытия формы при нажатии на кнопку "**Выход**":

```
Private Sub CommandButton4_Click()
   UserForm2.Hide
End Sub
```

Процедура инициализации пользовательской формы:

```
Private Sub UserForm_Initialize()
    'Установить значение первого переключателя на форме
  OptionButton1.Value = True
   'Установить для списка свойство для выделения
   'нескольких значений
  ListBox1.MultiSelect = 1
End Sub
```

Контрольные вопросы

1 Что такое объект, свойство, событие, метод, элемент управления?

2 Как создается пользовательская форма?

3 Объясните концепцию визуального и объектно-ориентированного программирования?

4 Объясните назначение объекта класса UserForm.

5 В чем заключается назначение окон свойств и проекта?

6 Что такое проект? Что находится в окне проекта?

7 Объясните назначение свойства *Name*, *BackColor, Font, Caption.* У каких объектов имеется данные свойства?

8 Объясните назначение свойства *TextAlign*. У каких объектов имеется данное свойство? Можно ли с помощью этого свойства расположить буквы текста перевернутыми?

9 Какие существуют способы добавления элементов в список?

5 ВОЗМОЖНОСТИ ИНТЕГРИРОВАННОЙ СРЕДЫ *VBA*

Visual Basic представляет пользователю обширные возможности для настройки интегрированной среды разработки *IDE* (*Integrated Development Environment*). Кроме этого, возможно редактирование стандартной панели инструментов, а также создание новых панелей инструментов и их редактирование. Среда позволяет иметь на рабочей поверхности до пяти панелей инструментов. В исходном состоянии отображается только одна панель инструментов, называемая *стандартной.* Если же выбрать команду `Customize`, то появившееся диалоговое окно также позволит вывести панель и, кроме того, позволит при желании создать новую панель инструментов (с помощью кнопки `New`).

Появившееся окно панели инструментов можно пристыковать к другим окнам, выполнив двойной щелчок на его титульной строке. Отстыковать окно панели инструментов можно, выполнив двойной щелчок на свободном от кнопок месте этого окна. С помощью диалогового окна `Customize` можно не только создавать новые панели инструментов, но и изменять имеющиеся панели инструментов, то есть помещать на них полезные кнопки и удалять ненужные.

5.1 Работа в среде VBA

Для написания и отладки программы (приложения) или отдельного модуля в среде необходимо запустить одно из приложения *MS Office*. Принципы работы в среде *VBA* не зависят от базового приложения, но рациональнее использовать то приложение, максимальное количество объектов которого используется в программе. В данном случае, за базовое приложение принят табличный процессор *Excel*. Таким образом, для начала работы в среде *VBA* запустить *MS Excel* и открыть редактор *VBA,* выполнить команду **Сервис** – **Макрос** – **Редактор Visual Basic** или комбинацию клавиш `Alt+F11`**.** Для быстрого запуска редактора можно вынести кнопку для вызова редактора VBA на панель инструментов **Стандартная,** для этого следует выполнить:

- команду **Вид – Панели инструментов – Настройка***;*
- перейти на вкладку **Команды**;
- в категории **Сервис** найти команду **Редактор Visual Basic**;
- перетащить кнопку **Редактора** на панель инструментов.

5.1.1 Структура окна редактора

В окне редактора могут находиться три окна (рисунок 5.1): проекта (`Project Explorer`), свойств (`Properties Window`) и программы (`Code Window`). Если нужное окно отсутствует на экране, его можно ото-

бразить с помощью соответствующей команды меню `View` (**Вид**). Можно изменять размеры окон и перемещать их по экрану.

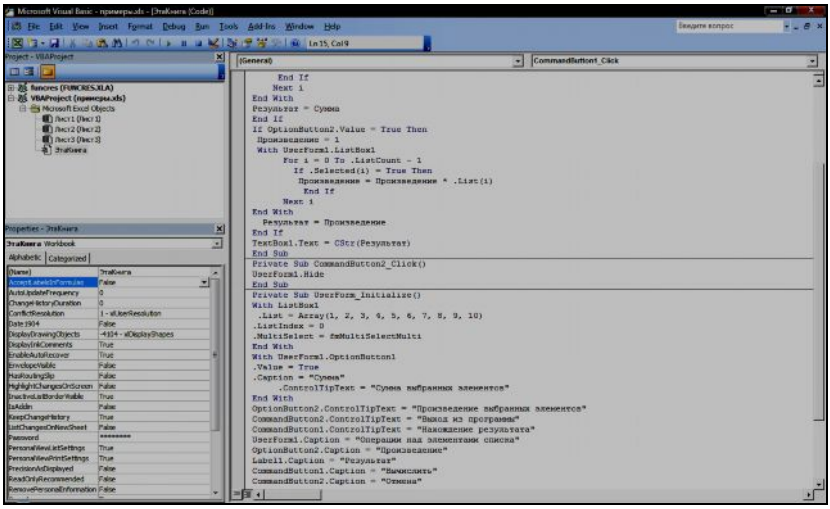


Рисунок 5.1 – Окно редактора *VBA*

Окно проекта содержит дерево открытых файлов и объектов, содержащихся в этих файлах. Если открыта только одна рабочая книга, то в окне проекта будет только один проект, если открыто несколько рабочих книг, то в окне проекта каждой рабочей книге соответствует свой проект, содержащий объекты и модули книги.

Окно свойств содержит свойства текущего объекта, выделенного в окне проекта. Список свойств может быть выведен в алфавитном порядке (вкладка `Alphabetic`) и по категориям (вкладка `Categorized`).

Окно программы – это окно, в котором можно создавать, просматривать и редактировать текст процедур *VBA* (исходный код). Можно просматривать в окне текст всех процедур модуля (кнопка `Full Module View`) или одну выбранную процедуру (кнопка `Procedure View`). Для перемещения между процедурами в первом режиме и для вывода в окно нужной процедуры во втором используется список `Procedure` в правом верхнем углу окна.

При **создании** первой программы в проекте или при создании нового модуля необходимо:

– выделить нужный проект в окне проекта;

– добавить модуль к проекту команда `Insert – Module` или щелкнуть на кнопке на панели инструментов **Стандартная** (`Standard`);

– ввести текст новой процедуры в окне программы.

Для загрузки существующей процедуры или ввода новой в существующий модуль необходимо:

– открыть окно программы для модуля: выполнить двойной щелчок по имени модуля в окне проекта или выделить модуль в окне проекта и выполнить команду `View – Code` (`F7`);

– ввести текст новой процедуры или изменить существующую процедуру (функцию).

5.1.2 Ошибки и их исправление

Все ошибки можно разделить на три большие группы:

– *синтаксические*. Такие ошибки могут возникать по причине неправильного написания оператора, имя переменной и т. д., они не требуют больших усилий по их поиску и исправлению;

– *логические*. Ошибки подобного рода проявляют себя в ходе выполнения программы. Для выявления и исправления ошибок такого типа и предназначены приемы отладки программы;

– ошибки времени выполнения (*run – time error*). Такого рода ошибки возникают в процессе выполнения, когда программа столкнулась с проблемой, решить которую она не в состоянии (файл с таким именем уже существует, возник конфликт записей при вставке в базе данных, произведена попытка записать информацию на переполненный диск и т. п.).

После того как *VBA* успешно завершит анализ и компиляцию строки кода в процедуре и не будет обнаружено никаких ошибок, выполнится цветовое кодирование различных частей строки. Заметим, ключевые слова в редакторе *VBA* отображаются синим цветом, комментарии – зеленым, а данные или другие операторы отображаются в виде черного текста. Если обнаружена ошибка синтаксиса в строке в процессе анализа или компиляции, *VBA* отображает всю строку красным цветом и выводит на экран диалоговое окно с сообщением об ошибке. Для доступа к справочной системе *VBA* и получения сведений об ошибке щелкните по кнопке **Справка**, для удаления с экрана диалогового окна – по кнопке **ОК**. Для получения справки по конкретному ключевому слову нужно поместить курсор на это слово и нажать клавишу `F1`. Рекомендуется сразу же исправлять обнаруженные ошибки синтаксиса до ввода или редактирования следующей строки или до перехода к следующему этапу работы.

Для **сохранения созданного проекта** следует выполнить команду `File – Save` или кнопка `Save` на панели инструментов **Стандартная**. Если рабочая книга ни разу не сохранялась и имеет имя **Книга1**, то появится окно сохранения документа, в котором необходимо указать имя рабочей книги и папку для ее сохранения.

При повторном использовании команды или кнопки `Save` проект сохра-

няется вместе с рабочей книгой под существующим именем.

Если необходимо изменить папку или имя документа, нужно перейти в окно рабочей книги и использовать команду **Файл – Сохранить как -** ....

Для выполнение (запуска) процедуры следует установить курсор на текст нужной процедуры и выполнить команду `Run – Run Sub\ UserForm` (`F5`) или использовать кнопку `Run Sub\UserForm` на панели инструментов **Стандартная**.

5.1.3 Запуск программы на выполнение

При выполнении процедур VBA выявляет ошибки, которые не могли быть обнаружены при вводе и редактировании программ. Такие ошибки называются ошибками времени исполнения (*run – time errors*) или *run – time* – ошибками. При обнаружении таких ошибок выполнение процедуры прекращается и на экран выводится диалоговое окно с сообщением об ошибке и кнопками `Continue` (продолжить), `End` (завершить), `Debug` (отладка) и `Help` (помощь). Для большинства ошибок кнопка `Continue` отключена, `End` завершает работу процедуры. Кнопка `Debug` позволяет перейти в режим отладки или продолжить работу процедуры (например повторить ввод данных), при этом команда и кнопка `Run Sub\UserForm` превращаются соответственно в команду и кнопку `Continue` (продолжить).

Для **прерывания работы процедуры** можно также использовать (например для выхода из зацикливающейся программы) команды меню `Run` и соответствующие кнопки на панели инструментов **Стандартная**:

– `Break` прерывает выполнение процедуры и позволяет перейти в режим отладки;

– `Reset` устанавливает процедуру в исходное состояние, выйдя из режима отладки.

При выполнении процедур исходные данные (если они есть) обычно вводятся в режиме диалога в текстовое окно ввода, а результаты могут быть размещены в ячейках рабочего листа, в текстовом файле или выведены в диалоговое окно (которое имеет, по крайней мере, одну кнопку *OK*). Если результаты выведены в диалоговое окно, то после их анализа необходимо нажать кнопку *OK* и вернуться в окно редактора.

5.1.4 Распечатка текста процедур

В редакторе VBA можно напечатать текст всех процедур одного модуля или всего проекта. Если требуется напечатать отдельную процедуру необходимо:

1) выделить текст нужной процедуры в окне программы или имя модуля в окне проекта;

2) выполнить команду *File – Print*…появится диалоговое окно *Print*;

3) в группе переключателей *Range* (диапазон) указать, что нужно напечатать: выделенный текст процедуры (*Selection*), текущий модуль (*Current Module*) или проект (*Current Project*);

4) в группе переключателей *Print What* (печатать) установить флажок *Code*;

5) щелкнуть по кнопке *Setup*, выбрать принтер и, при необходимости, установить альбомную ориентацию, щелкнуть по кнопке *OK*.

Для закрытия окна редактора и перехода в *Excel* следует выполнить команду *File – Close and Return to Microsoft Excel* или нажать комбинацию клавиш `ALT+Q`.

5.2 Настройка параметров интегрированной среды разработки

Доступ к большинству настраиваемых параметров можно получить с помощью диалогового окна *Options*, если выполнить команды *Tools*, *Options*.

Окно диалога *Options* (рисунок 5.2) содержит четыре вкладки, на каждой из которых группируются параметры определенной категории.

На вкладке *Editor* (Редактор) представлены параметры, влияющие на режимы ввода кода (группа *Code Settings*), и параметры окна редактора (группа *Window Settings*). Установка флажка слева от параметра устанавливает или отменяет действие функции соответствующего параметра. Назначение параметров:

– *Auto Syntax Check* – проверка синтаксиса строки;

– *Require Variable Declaration* – вставка в секцию *General* создаваемой формы или модуля инструкции *Option Explicit*, обязывающей пользователя в этой форме или модуле объявлять все переменные;

– *Auto List Members* – вывод подсказки в окне кода – списка свойств и методов объекта во время ввода программы, когда после имени объекта поставлена точка;

– *Auto Quick Info* – отображение в окне кода программы подсказки о функциях и параметрах во время ввода программы;

– *Auto Data Tips* – отображение в режиме отладки значения переменной, находящейся под указателем мыши;

– *Auto Indent* – установка в следующей строке такого же отступа, как и в текущей;

– *Drag-and-Drop Text Editing* – разрешает перетаскивание элементов текста внутри окна кода, а также из окна кода в окна *Watch* и *Immediate*;

– *Default to Full Module View* – прокрутка в окне кода, как обычного текста, или пролистывание по процедурам;

– *Procedure Separator* – вставка разделительных линий между процедурами в окне кода.

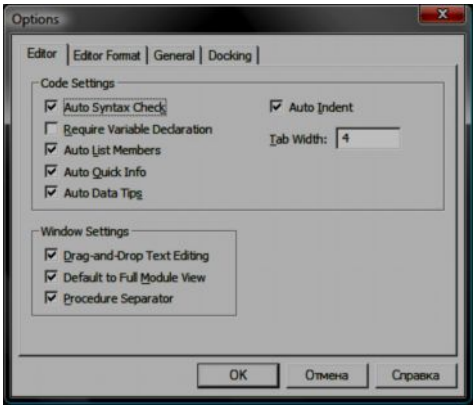


Рисунок 5.2 – Окно *Options* – вкладка *Editor*

Вкладка *Editor Format* (Параметры текста) позволяет выбрать шрифт и цвет символов разных категорий программного кода.

Вкладка *General* (Общие параметры) позволяет установить реакцию на ошибки, выравнивание по сетке, шаг сетки и др.

Вкладка *Docking* (Стыковка окон) – содержит список окон, для каждого из которых можно включить режим стыковки.

5.3 Средства отладки

Для облегчения поиска ошибок в программе, т. е. когда программа работает неправильно, в *VBA* предусмотрены специальные средства отладки, которые позволяют по шагам отслеживать выполнение инструкций программы, а также наблюдать за изменением значений переменных, выражений и свойств объектов.

Средства отладки становятся доступными в режиме прерывания с помощью команды меню *Debug* или специальной панели инструментов *Debug* (Отладить).

При выполнении программы режим прерывания будет установлен:

– при запуске проекта нажатием клавиши `F8`;

– если после обычного запуска проекта щелкнуть на кнопке панели инструментов *Break* (Прервать);

– по достижении инструкции с установленной точкой прерывания.

Точки прерывания могут быть установлены и сняты в программе в режиме разработки или в режиме прерывания. Для установки точки прерывания в режиме разработки достаточно щелкнуть на сером поле слева от той строки программы, перед выполнением которой она должна остановиться (рисунок 5.3). Следующий щелчок в этом же месте позволит снять точку прерывания. Если в режиме прерывания поместить указатель мыши в программе на имени переменной или названии свойства объекта, то можно увидеть чему равно значение этой переменной или свойства.

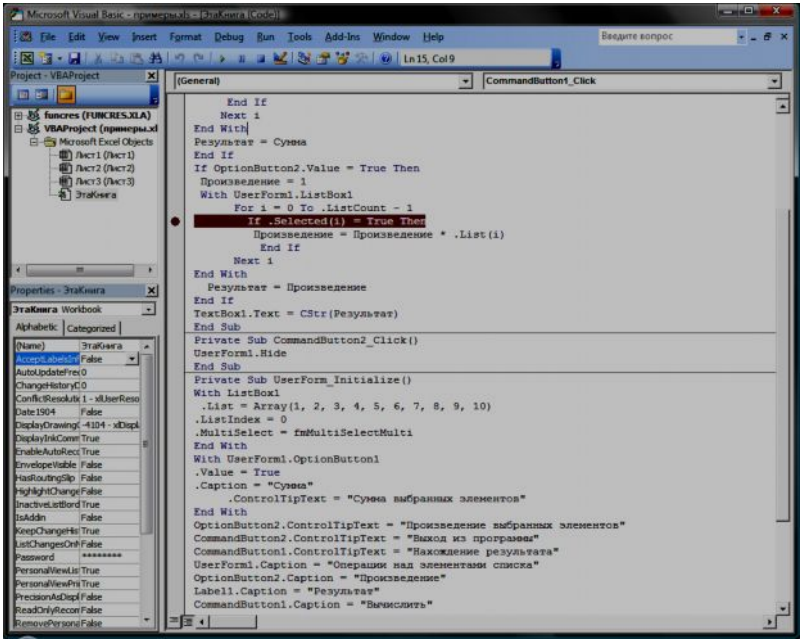


Рисунок 5.3 – Установка точки прерывания

Остановимся на назначении отдельных кнопок специальной панели инструментов *Debug* (Отладить), изображенной на рисунке 5.4.

В режиме прерывания с помощью кнопки *Step Into* (Выполнить шаг) этой панели инструментов или с помощью соответствующей команды меню *Debug* можно выполнять программу по шагам оператор за оператором.

Если очередной выполняемый оператор содержит обращение к процедуре или функции, то предоставляются две возможности:

а) войти в процедуру или функцию и остановиться (кнопка *Step Into* – выполнить шаг);

б) войти в процедуру или функцию, выполнить ее до конца и остановиться (кнопка *Step Over* – выполнить процедуру).

Кнопка *Step Out* (Закончить процедуру) позволяет выполнить оставшиеся операторы текущей процедуры и остановиться.

Кнопка *Toggle Breakpoint* (Точка прерывания) переключает состояние точки прерывания в текущей строке программы.

Кнопка *Locals Window* (Окно локальных переменных) автоматически выводит значения всех локальных переменных, а также может отображать значения всех свойств объектов.

Кнопка *Immediate Window* открывает окно непосредственных вычислений. Если в режиме прерывания в этом окне поставить вопросительный знак, набрать некоторое выражение (например, ? x + y) и нажать на клавишу *Enter*, то будет вычислено и выведено значение этого выражения. Если же набрать оператор и нажать на клавишу *Enter*, то этот оператор будет немедленно выполнен.

Кнопка *Watch Window* (Окно отладки) открывает окно, в котором можно наблюдать за текущими значениями отслеживаемых выражений, а также свойств объектов. Выражение, которое как отслеживаемое вам нужно включить в окно, достаточно выделить и перетащить в окно мышью. Значения переменных можно принудительно изменять.

Кнопка *Quick Watch* (Быстрый просмотр) открывает окно с текущим значением выделенного выражения. Чтобы было возможно следить за значением выделенного выражения в будущем, следует его включить в окно кнопкой *Add*.

Кнопка *Call Stack* (Стек) вызывает окно со списком всех незавершенных процедур и функций.

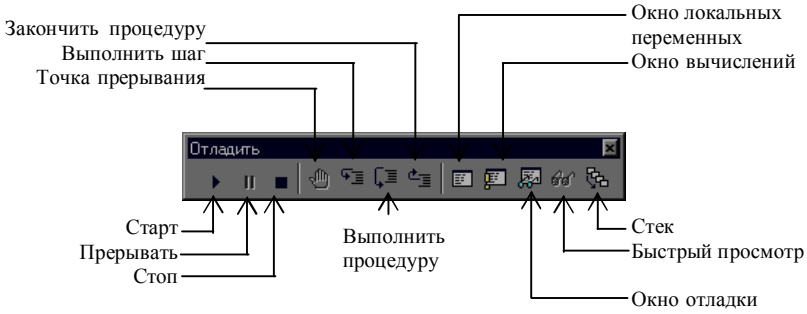


Рисунок 5.4 – Кнопки специальной панели инструментов *Debug*

Поскольку меню и панель инструментов могут быть недоступны во время работы приложения, например, если форма развернута на весь экран, то в процессе отладки могут оказаться весьма полезными «быстрые» клавиши, назначение которых приведено в таблице 5.1.

Таблица 5.1 – **«Горячие» клавиши для отладки**

Сочетание клавиш	Оказываемое действие
`F5`	Запускает проект
`Ctrl + Pause`	Прерывает выполнение программы
`F8`	Выполняет шаг или запускает программу в пошаговом режиме
`F9`	Устанавливает или снимает точку прерывания в текущей строке
`Shift + F8`	То же, что и `F8`, но процедуры и функции выполняются за один шаг
`Ctrl + Shift + F8`	Выполняет текущую процедуру или функцию до конца и выходит на точку, следующую за точкой, откуда процедура или функция была вызвана
`Ctrl + G`	Открывает окно *Locals*
`Shift + F9`	Показывает текущее значение
`Ctrl +L`	Показывает стек (список всех вызванных, но незавершенных процедур и функций)
`F6`	Переключается между окнами *Immediate* и *Watch*

ЗАКЛЮЧЕНИЕ

Visual Basic for Applications (VBA) является общей языковой платформой для всех приложений (*Excel, Word, Power Point* и др.). *VBA* соблюдает основной синтаксис и правила программирования языков – диалектов *Basic (BASICA, GW-BASIC)*, т.е. *VBA* является встроенным в приложения пакета *MS Office* с целью автоматизации их использования и организации оперативного обмена данными между ними. Он ориентирован на активное использование объектов диалога и библиотек объектов приложений пакета. Программный код *VBA*, выполняется в режиме интерпретации и хранится в составе файла документа приложения и является удобным средством для расширения функциональных возможностей базового приложения, а также автоматического создания составных документов.

Visual Basic for Applications (VBA) – развитая система визуального программирования для создания прикладных программ в среде *Microsoft Office.*

С помощью *VBA* можно создавать объекты управления графического интерфейса пользователя, задавать и изменять свойства объектов, подключать к ним соответствующий программный код. Методика программирования с использованием средств *VBA* сводится к следующему:

– создание объектов управления и контроля (диалоговые окна, пиктограммы, меню);

– разработка процедур, используемых при вызове объектов.

Прикладные программы на языке *VBA* оперируют со следующими понятиями:

– объект управления и контроля – экранные формы, графические элементы внутри форм, в том числе текстовые окна, линейки прокрутки, пиктограммы, окна-списки, командные кнопки и др.;

– свойство (параметр) – характеристика или атрибут объекта управления;

– значение свойства;

– событие – действие, которое распознается объектом управления;

– метод доступа – аналогичное понятиям функция, оператор, который воздействует всегда на объект;

– процедура – подпрограммы и функции, произвольная последовательность операторов VBA; процедуры делятся на событийные (запускаются при наступлении событий) и общие процедуры.

СПИСОК ЛИТЕРАТУРЫ

1 MS Office XP : Разработка приложений / под ред. Ф. А. Новикова. – СПб. : БХВ-Петербург, 2005. – 345 с.

2 **Биллинг, В. А.** VBA и Office 2000. Офисное программирование / В. А. Биллинг. – М. : Русская редакция, 1999. – 423 с.

3 **Браун, С.**Visual Basic 6: учебный курс / С. Браун. – СПб. : Питер, 2004. – 576 с.

4 **Васильев, А. А.** VBA в Office 2000: учебный курс / А. А. Васильев. – СПб. : Питер, 2002. – 234 с.

5 **Васильев, А. А.** VBA в Office 2000: учеб. курс / А. А. Васильев, А. И. Андреев. – СПб. : Питер, 2001. – 432 с.

6 **Гарнаев, А. Ю.** Использование MS Excel и VBA в экономике и финансах / А. Ю. Гарнаев. – СПб. : БХВ – Санкт-Петербург, 2006. – 256 с.

7 **Гарнаев, А. Ю.** Самоучитель VBA / А. Ю. Гарнаев. – СПб. : БХВ – Санкт-Петербург, 2006. – 345 с.

8 **Карпов, Б.** Visual Basic 6: специальный справочник / Б. Карпов. – СПб.: Питер, 2003. – 416 с.

9 **Кузьменко, В. Г.** VBA 2002 / В. Г. Кузьменко. – М. : БИНОМ, 2004. – 344 с.

10 **Кузьменко, В. Г.** Программирование на VBA 2002 / В. Г. Кузьменко. – М. : БИНОМ, 2003. – 880 с.

11 **Райтингер, М**. Visual Basic 6.0 / М. Райтингер, Г. Муч ; пер. с нем. – Киев : Издательская группа BHV, 2000. – 288 с.

12 **Санна, П.** Visual Basic для приложений (версия 5) в подлиннике / П. Санна, : пер. с англ. – СПб. : BHV, 1998. – 704 с.

13 Отладка приложений на VB6 [Электронный ресурс]. – Режим доступа: http://www.vbrussian.com/programs.asp – Дата доступа: 20.02.2009.

14 Меню на VBA. Готовим профессионально [Электронный ресурс]. – Режим доступа: http://www.vbstreets.ru/VBA/Articles/66509.aspx – Дата доступа: 24.03.2009.

15 **Хореев, В. Д.** Самоучитель программирования на VBA в Microsoft Office / В. Д. Хореев. – Киев : Юниор, 2001. – 296 с.

16 Основы офисного программирования и документы MS Word[Электронный ресурс]. – Режим доступа: http://www.intuit.ru/department/office/vbaword/ – Дата доступа: 24.03.2009.

17 Основы офисного программирования и документы MS Excel [Электронный ресурс]. – Режим доступа: http://www.intuit.ru/department/office/vbaexcel/ – Дата доступа 2.03.2009.

18 Основы офисного программирования и язык VBA [Электронный ресурс]. – Режим доступа http://www.intuit.ru/ department/office /vba2000/ – Дата доступа: 2.03.2009.

19 Курс лекций по VBA [Электронный ресурс]. – Режим доступа: http://www.mini-soft.ru/soft/vba/ – Дата доступа: 12.03.2009.

20 **Фризен, И. Г.** Офисное программирование [Электронный ресурс] / И. Г. Фризен. – Режим доступа :http://vsebook.ru/gumanitar/23770-ofisnoe-programmirovanie.html – Дата доступа: 23.05.2009.

21 Специальный курс по офисному программированию [Электронный ресурс]. – Режим доступа: http://www.askit.ru/custom/vba_office – Дата доступа: 10.02.2010

Учебное издание

МАРЬИНА Наталья Александровна

МАРЬИН Сергей Александрович

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА VBA

Учебно-методическое пособие

Редактор *Т. М. Ризевская*
Технический редактор *В. Н. Кучерова*

Подписано в печать 14.06.2010 г. Формат 60×84 1/16 .
Бумага офсетная. Гарнитура Times. Печать на ризографе.
Усл. печ. л. 5,81. Уч.-изд. л. 5,97. Тираж 120 экз.
Зак. № . Изд. № 67.

Издатель и полиграфическое исполнение
Белорусский государственный университет транспорта:
ЛИ № 02330/0552508 от 09.07.2009 г.
ЛП № 02330/0494150 от 03.04.2009 г.
246653, г. Гомель, ул. Кирова, 34.