

Е. А. СИДОРОВА, С. П. ЖЕЛЕЗНЯК

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ VBA

ОМСК 2021

Министерство транспорта Российской Федерации
Федеральное агентство железнодорожного транспорта
Омский государственный университет путей сообщения

Е. А. Сидорова, С. П. Железняк

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ VBA

Учебное пособие

Омск 2021

УДК 004.42(075.8)
ББК 32.973я73
С34

Основы программирования на языке VBA: Учебное пособие /
Е. А. Сидорова, С. П. Железняк; Омский гос. ун-т путей сообщения. Омск,
2021. 118 с.

Рассмотрены приемы работы в интегрированной среде разработки приложений VBA, программирование базовых структур алгоритмов и их применение для решения различных вычислительных задач.

Учебное пособие представлено в виде блоков теоретического и практического материала, в которых излагаются общая концепция и особенности алгоритмизации и программирования вычислительных процессов на VBA в приложении Microsoft Excel.

Предназначено для студентов и аспирантов очной и заочной форм обучения всех направлений подготовки (специальностей), изучающих основы программирования. Может быть использовано в качестве самоучителя по программированию для любых категорий пользователей.

Библиогр.: 5 назв. Табл. 10. Рис. 62.

Рецензенты: доктор техн. наук, профессор В. Н. Горюнов;
доктор техн. наук, профессор А. А. Кузнецов;
канд. техн. наук, доцент А. А. Руппель.

ISBN 978-5-949-41276-3

ОГЛАВЛЕНИЕ

Введение	5
1. Язык программирования VBA	6
1.1. Общие сведения о VBA	6
1.2. Основные понятия VBA	6
2. Интегрированная среда разработки приложений VBA.....	7
2.1. Запуск редактора VBA в Excel.....	8
2.2. Работа с окнами редактора VBA	8
2.3. Работа в стандартном модуле	10
2.4. Сохранение файла с программным кодом.....	17
2.5. Создание пользовательской формы	18
3. Элементы VBA	24
3.1. Алфавит VBA.....	24
3.2. Общие правила именования элементов VBA.....	24
3.3. Типы данных	25
3.4. Константы	27
3.5. Переменные.....	28
3.6. Встроенные функции	29
4. Запись выражений	33
4.1. Арифметические операции.....	33
4.2. Операция слияния строк (конкатенация).....	34
4.3. Операции отношения (сравнения).....	34
4.4. Сравнение строк	35
4.5. Логические операции.....	35
4.6. Порядок действий в выражении	35
5. Управление вводом и выводом данных	38
5.1. Выбор и очистка рабочего листа Excel	38
5.2. Операторы ввода данных	39
5.3. Операторы вывода данных.....	42
5.4. Взаимодействие с элементами управления на форме	46
6. Линейные вычислительные процессы	48
7. Разветвляющиеся вычислительные процессы.....	55
7.1. Оператор условного перехода <i>If</i>	55
7.2. Объединение условий с помощью логических операций.....	60
7.3. Оператор безусловного перехода <i>GoTo</i>	64
7.4. Оператор выбора <i>Select Case</i>	66

8. Циклические вычислительные процессы	70
8.1. Понятие цикла	70
8.2. Арифметический цикл	70
8.3. Вычисление сумм и произведений	81
8.4. Итерационные циклы	84
9. Модульное программирование. Процедуры-функции и процедуры-подпрограммы	89
9.1. Понятие процедуры.....	89
9.2. Формальные и фактические параметры процедур	90
9.3. Разработка процедур-функций	91
9.4. Разработка процедур-подпрограмм.....	95
9.5. Срочный выход из процедур и циклов	97
10. Переменные с индексами. Одномерные массивы.....	98
10.1. Понятие массива.....	98
10.2. Ввод массива.....	102
10.3. Вывод массива	102
10.4. Решение задач с использованием массивов	103
11. Отладка программы и обработка ошибок.....	109
11.1. Ошибки компиляции.....	109
11.2. Ошибки времени выполнения.....	110
11.3. Логические ошибки.....	110
12. Контрольные вопросы.....	111
Библиографический список.....	112
Приложение 1. Основные операторы VBA	113
Приложение 2. Наиболее распространенные ошибки при работе на VBA.....	116

ВВЕДЕНИЕ

Visual Basic for Applications (VBA) представляет интегрированную среду программирования, встроенную во множество программных пакетов, в том числе приложения Microsoft Office. К преимуществам VBA можно отнести сравнительную простоту освоения, благодаря которой пользователи, не программирующие профессионально, могут автоматизировать решение практически любой задачи: от простых рутинных операций, например, при подготовке текстов в Word или выполнении расчетов в Excel, до сложнейших вычислений над большими объемами данных.

Структура данного пособия ориентирована на последовательное изучение VBA – от организации ввода-вывода данных до программирования сложных алгоритмических структур. Наряду с описанием основных конструкций языка VBA пособие содержит типовые примеры, необходимые для его практического освоения. Для удобства изучения представленного материала в каждом разделе пособия использована отдельная нумерация примеров, в которой буквенный префикс соответствует тематике раздела:

- Л – линейные процессы;
- У – условные (разветвляющиеся) процессы;
- Ц – циклические процессы;
- ЦС – циклы вычисления сумм и произведений;
- ЦИ – циклы итерационные;
- ПФ – процедуры-функции;
- ПП – процедуры-подпрограммы;
- ОМ – одномерные массивы.

Многие действия на VBA можно реализовать различными способами. В пособии рассмотрены простейшие и самые наглядные из них. Библиографический список в конце издания содержит литературу для углубленного изучения материала по представленной тематике.

1. ЯЗЫК ПРОГРАММИРОВАНИЯ VBA

1.1. Общие сведения о VBA

Visual Basic for Applications – одна из версий языка программирования Visual Basic, встроенная в Word, Excel, Access, PowerPoint и другие приложения Microsoft Office, а также программные пакеты других фирм, например, CorelDRAW, AutoCAD и т. п. Язык VBA изначально был ориентирован не на профессиональных программистов, а на обычных пользователей, поэтому несложные компьютерные программы на нем можно создавать довольно легко и быстро, ничего не зная о программировании. Для этого в приложениях Microsoft Office служит *макрорекордер* – встроенный инструмент автоматического кодирования действий пользователя для их дальнейшего повторения. Записанные с помощью макрорекордера последовательности команд называются макрокомандами, или *макросами*. Однако макрорекордер имеет ограниченные возможности: не умеет проверять значения, чтобы в зависимости от них выбирать какое-либо действие, не работает с циклами, не умеет перехватывать и обрабатывать ошибки и т. д. VBA-программирование помогает решить эти проблемы и значительно повысить возможности макросов, позволяя улучшить интерфейс приложений, создавать пользовательские меню, диалоговые окна и др.

VBA считается объектно-ориентированным языком программирования, ключевым понятием которого является совокупность объектов и связей между ними. VBA включает в себя не только собственно язык программирования для разработки программного кода (текста программы), но и графическую среду, позволяющую конструировать экранные формы.

В данном пособии программирование на VBA рассмотрено применительно к Microsoft Excel 2010 (далее – Excel).

1.2. Основные понятия VBA

Проект на VBA представляет собой совокупность объектов и программных модулей, связанных с документом (рабочей книгой Excel).

Объект – это именованный элемент, представляющий собой объединение данных с программным кодом, предназначенным для их обработки. Объектами являются рабочая книга и рабочий лист Excel, пользовательская форма, элемент управления и др.

Объект имеет:

свойства (характеристики), которые можно проверить или изменить (например, заголовок, размер шрифта, цвет фона и др.). Доступ к информации, хранящейся в объекте, реализуется через его свойства;

методы, т. е. действия, которые может выполнить объект (например, осуществлять какие-либо операции, принимать и возвращать значения и др.);

события, т. е. возможные для объекта ситуации (например, щелчок кнопкой мыши, нажатие клавиши на клавиатуре и др.), на которые он может ответить заранее predetermined действиями.

Оператор – это наименьшая способная выполняться единица программного кода. Оператор может объявлять или определять переменную или выполнять какое-либо действие в программе. Другими словами, оператор – это обобщенная инструкция (команда), которая содержит всю информацию, необходимую для выполнения определенных действий. Операторы в общем случае состоят из ключевых слов, операндов и выражений.

Ключевые (зарезервированные) слова – слова, распознаваемые как часть языка VBA. К ним относятся имена операторов, типов данных, методов, свойств, встроенных функций и др.

Операнды – различные данные (константы, переменные, функции, массивы и др.), над которыми выполняют определенные действия (операции).

Выражение – любая комбинация операндов и знаков операций.

Процедура – наименьшая единица программного кода, на которую можно ссылаться по имени и которая может выполняться независимо. Основные типы процедур – Sub (процедура-подпрограмма) и Function (процедура-функция).

Модуль – именованная единица, состоящая из одной или нескольких процедур и раздела объявлений, относящихся ко всем процедурам в модуле.

2. ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ ПРИЛОЖЕНИЙ VBA

Проекты на языке VBA создаются с помощью *редактора VBA* – интегрированной среды разработки программного обеспечения, включающей в себя средства редактирования текстов программ, средства отладки, средства управления проектом и другие инструменты.

2.1. Запуск редактора VBA в Excel

Открыть окно редактора VBA в Excel можно следующими способами:

1) самый быстрый способ – нажать клавиши левый Alt + F11;

2) выполнить на ленте команду **Разработчик** → [Код] → Visual Basic¹. Если вкладка **Разработчик** отсутствует, то нужно ее включить, выполнив команду **Файл** → Параметры → Настройка ленты → ☒ **Разработчик**.

В результате открывается главное окно редактора VBA (рис. 2.1), имеющее стандартный для Windows-приложений интерфейс: заголовок окна, где выводится имя текущей рабочей книги, главное меню и панель инструментов.

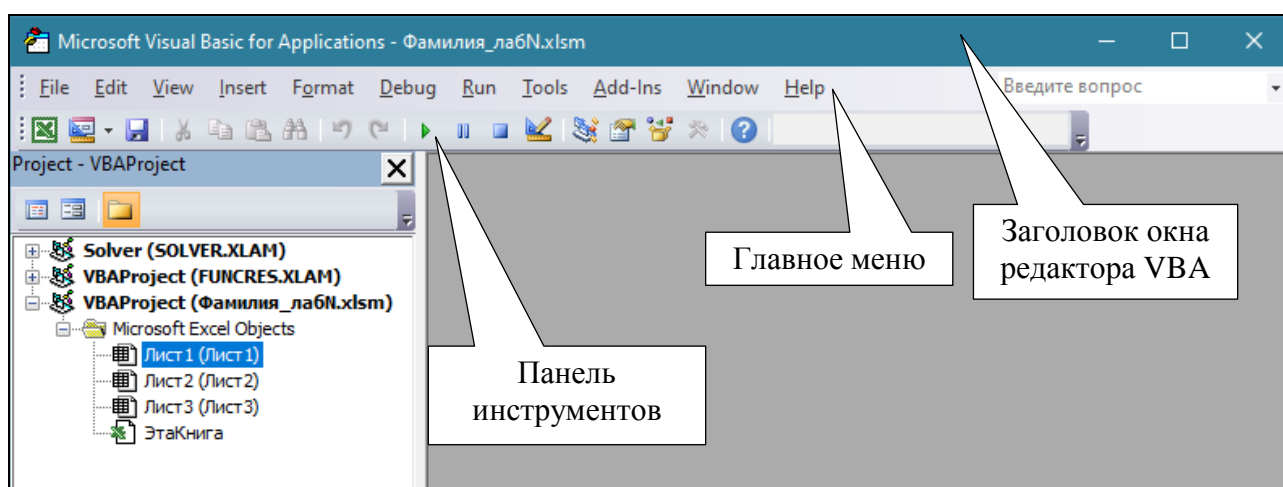



Рис. 2.1. Вид главного окна редактора VBA

Чтобы закрыть редактор VBA, достаточно щелкнуть кнопку  в строке заголовка окна или нажать клавиши Alt + F4.

Список всех доступных макросов открытых рабочих книг отображается в диалоговом окне **Вид** → [Макросы] → Макросы или **Разработчик** → [Код] → Макросы.

2.2. Работа с окнами редактора VBA

Всего в редакторе VBA предусмотрено девять дополнительных окон. Открыть любое из них можно из меню **View**, выполнив соответствующую команду. При работе в редакторе VBA наиболее часто используются следующие окна (рис. 2.2):

¹ В данном пособии месторасположение команды на ленте Excel указывается в виде: **Вкладка** → [Группа] → Элемент управления → *Команда (опция меню)*

Project – окно проводника VBA-проекта, в котором отображается иерархическая структура его объектов (модулей, форм и др.);

Properties – окно свойств, которое используется для просмотра или задания характеристик выбранного элемента проекта и играет важную роль при программировании пользовательских форм;

Code – окно редактора программного кода выбранного объекта;

Immediate – окно отладки процедур, предназначенное для вывода текущих значений переменных и тестирования отдельных строк программного кода.

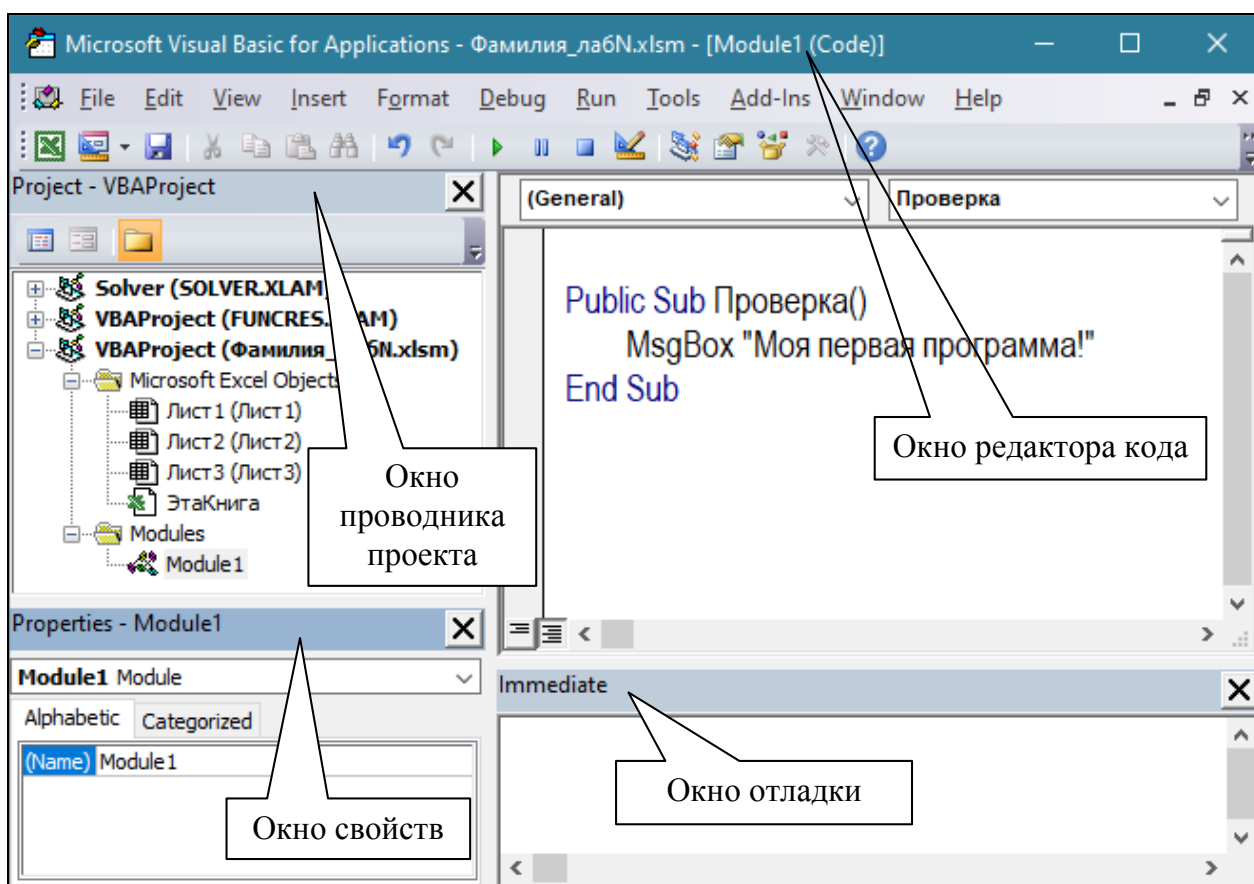




Рис. 2.2. Наиболее часто используемые окна редактора VBA

Закрыть любое окно редактора VBA можно, нажав в его правом углу кнопку  или отменив соответствующую команду в меню **View**.

Для быстрого перехода из любого другого окна редактора VBA в окно Properties можно нажать клавишу F4, в окно Code – клавишу F7, в окно Immediate – клавиши Ctrl + G, на рабочий лист Excel – клавиши левый Alt + F11 или крайнюю левую кнопку  на панели инструментов (см. [рис. 2.1](#)).

При запуске редактора VBA окно Project по умолчанию открыто и находится в левой части окна редактора VBA (см. [рис. 2.1](#)). В окне проекта отображается список проектов всех открытых рабочих книг и иерархическая структура (дерево компонентов) каждого проекта. Проект является не только документом, содержащим данные, формулы, диаграммы и т. д., но и контейнером для хранения модулей, предназначенных для создания программ на VBA.

По назначению модули бывают двух типов: стандартные модули и модули объектов. К *стандартным* относятся модули, содержащие пользовательские программы, к *модулям объектов* – модули, связанные с рабочей книгой, рабочими листами (эти модули создаются автоматически при запуске редактора VBA), пользовательскими формами и др.

2.3. Работа в стандартном модуле

2.3.1. Создание и удаление стандартного модуля

Добавить в проект новый стандартный модуль можно командой главного меню **Insert** → **Module** или аналогичной командой контекстного меню окна Project. В результате в проекте будет создан стандартный модуль с именем Module1, одновременно откроются пустое окно редактора кода этого модуля [Module1 (Code)] и окно свойств модуля Properties – Module1. Выполняя аналогичные действия, в проекте можно создать новые модули Module2, Module3 и т. д. При необходимости имя выбранного модуля можно изменить в окне его свойств в поле Name (см. [рис. 2.2](#)). Имя модуля должно соответствовать общим правилам именования элементов VBA (см. [подразд. 3.2](#)).

Открыть окно редактора кода любого модуля можно двойным щелчком мыши на его имени в окне Project.

Для удаления выбранного модуля нужно выполнить команду главного меню **File** → **Remove Module...** или аналогичную команду в контекстном меню этого модуля. При этом на запрос системы о сохранении модуля для последующего восстановления обычно отвечают «Нет» ([рис. 2.3](#)).

Пользовательская программа на языке VBA записывается в стандартном модуле и оформляется в виде процедуры, которая является логически самостоятельным функциональным блоком.

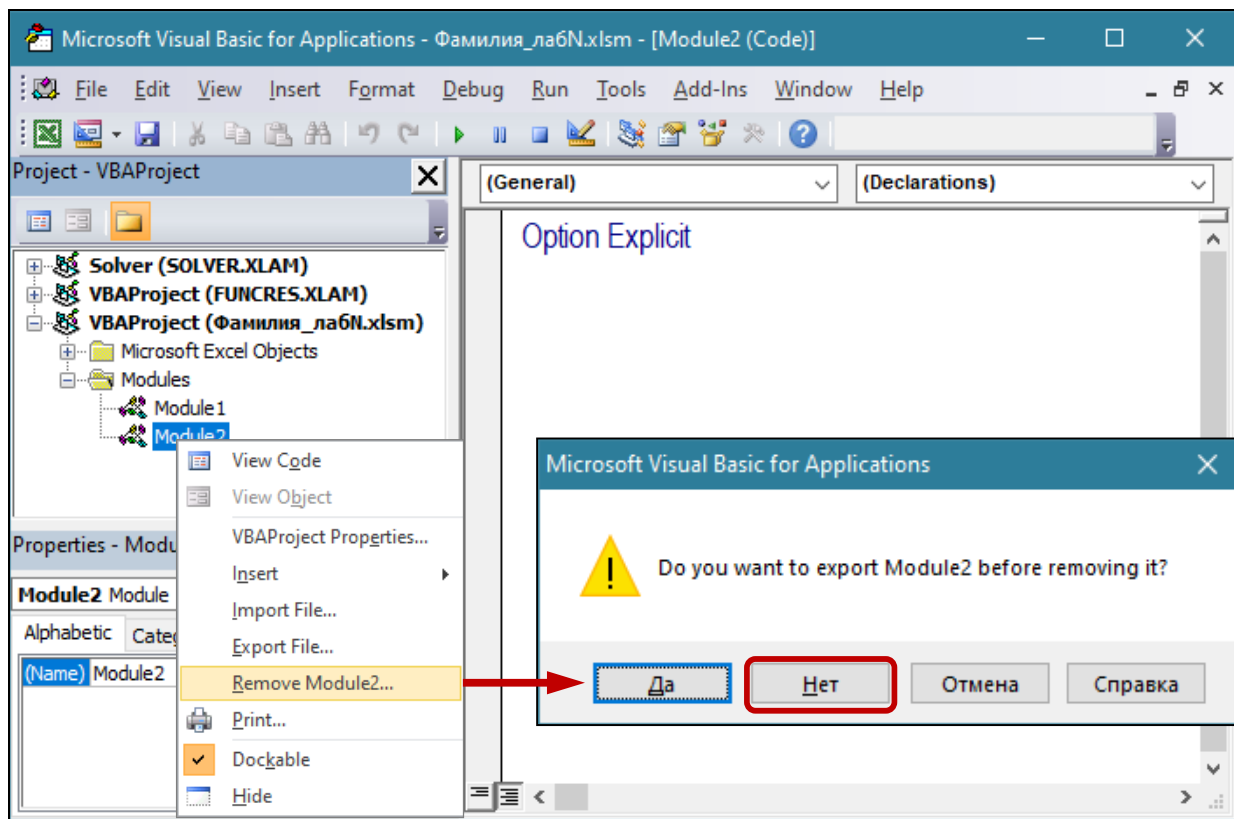


Рис. 2.3. Вид окна команды контекстного меню для удаления модуля Module2

Стандартный модуль может включать в себя одну или несколько процедур, а также объявления уровня модуля и в общем случае имеет следующую структуру:

'Раздел объявлений уровня модуля

Option Explicit

... 'Инструкции уровня модуля

'Раздел методов модуля (расположение процедур)

Sub Имя_процедуры1()

... 'Тело процедуры1

End Sub

Sub Имя_процедуры2()

... 'Тело процедуры2

End Sub

2.3.2. Создание процедуры в стандартном модуле

Универсальной процедурой для выполнения каких-либо программных действий является процедура типа Sub (подпрограмма), для создания которой в окне Code выбранного модуля нужно выполнить команду главного меню **Insert** → Procedure. В открывшемся диалоговом окне Add Procedure (рис. 2.4, а) выполнить следующие действия:

в поле Name ввести имя процедуры в соответствии с общими правилами именования элементов VBA (см. подразд. 3.2), например, Проверка;

в группе Type выбрать тип процедуры – Sub (процедуры типа Function рассмотрены в подразд. 9.3);

в группе Scope указать зону видимости процедуры: Public – открытая процедура, вызываемая не только из того модуля, в котором она записана, но и из любого другого модуля текущей рабочей книги и (или) других книг; Private – локальная процедура, ее можно вызывать только из того модуля, в котором она расположена.

После нажатия кнопки Ok в окне Code автоматически появляются (рис. 2.4, б):

строка начала (заголовок) процедуры с заданными установками, например:

```
Public Sub Проверка();
```

пустая строка-разделитель, определяющая место для ввода инструкций языка VBA, составляющих «тело» процедуры;

```
строка завершения процедуры End Sub.
```

Этот же набор строк можно ввести и непосредственно с клавиатуры в окне Code. При этом достаточно записать только заголовок процедуры и нажать клавишу Enter, после чего пустая строка и строка завершения процедуры появятся автоматически. В заголовке процедуры зону ее видимости можно не указывать. В этом случае по умолчанию принимается Public. В дальнейшем будем рассматривать процедуры только такого типа.

Структура процедуры в общем случае имеет следующий вид:

```
Sub Имя_процедуры(Аргументы)  'Заголовок процедуры
    ... 'Объявление констант и переменных
    ... 'Последовательность инструкций
End Sub      'Конец процедуры
```

Круглые скобки после имени процедуры обязательны. В них перечисляются параметры, необходимые для работы процедуры (см. подразд. 9.2). При отсутствии параметров круглые скобки остаются пустыми.

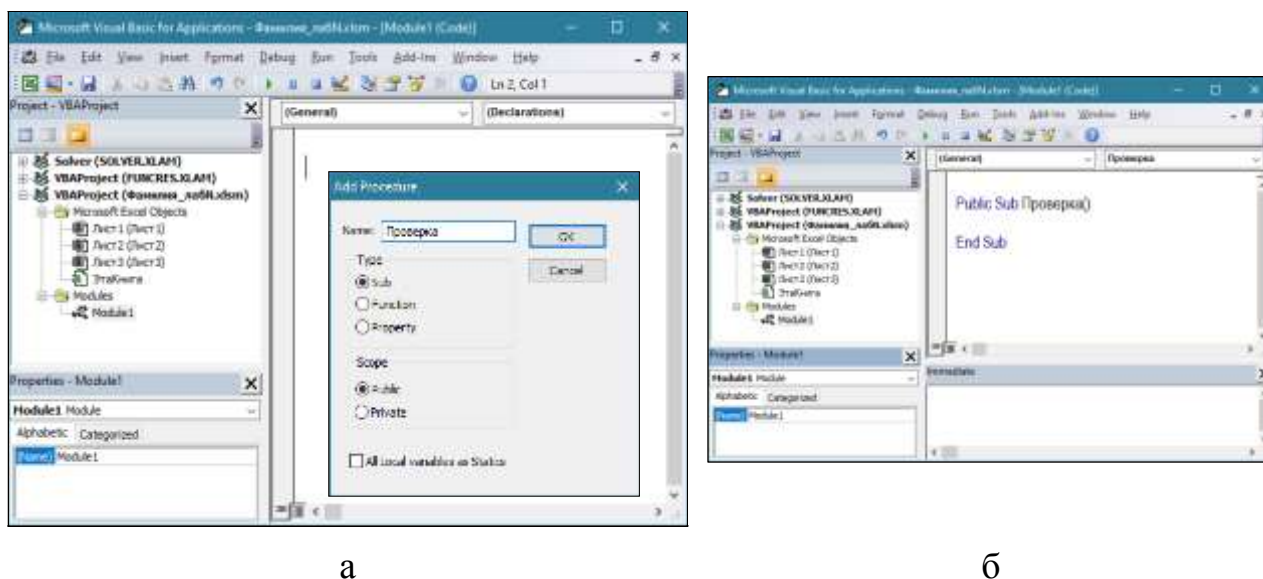


Рис. 2.4. Создание процедуры Проверка в стандартном модуле Module1:
а – параметры диалогового окна Add Procedure; б – окно редактора кода

2.3.3. Общие правила записи процедур

VBA может выполнить только тот программный код, который содержится в какой-либо процедуре, т. е. между строками Sub... и End Sub. В противном случае программный код выполняться не будет, поскольку он становится «невидимым» для транслятора VBA.

Как правило, каждая инструкция программного кода записывается в отдельной строке. Допускается указывать в одной строке несколько инструкций, разделяя их двоеточием. Чтобы сделать процедуру хорошо читаемой, при наборе текста программы следует делать отступы (для этого рекомендуется использовать клавишу Tab), вводить пустые строки между функционально самостоятельными частями программы и помещать *комментарии*, поясняющие ход выполнения программы.

Комментарии вводятся в программу с помощью оператора Rem или одинарной кавычки (апострофа), после которой записывается текст комментария. Включение операторов Rem в программу никак не влияет на ее выполнение, так как содержимое строки после оператора комментария транслятором игнорируется и не исполняется.

Формат записи оператора Rem:

Rem Комментарий (или 'Комментарий')

Для удобства при записи длинной программной строки ее можно разделить на несколько фрагментов – физических строк. Чтобы при трансляции программы эти строки были объединены в одну логическую строку, в конце каждой физической строки (кроме последней) вставляют пробел и после него – знак подчеркивания, например:

```
MsgBox "Сумма равна " _  
      & Summa
```

Такая запись равнозначна строке

```
MsgBox "Сумма равна " & Summa
```

Инструкции на VBA должны записываться в точном соответствии с их синтаксисом (правилами построения). При составлении программного кода редактор VBA автоматически отображает список компонентов, логически завершающих вводимую инструкцию. Целесообразно применять режим дополнения ключевых слов VBA: ввести несколько начальных букв и нажать Ctrl + Пробел. При этом произойдет автоматический ввод слова до конца (если оно единственное) или будет выдан перечень слов VBA, начинающихся на введенные буквы (рис. 2.5), тогда можно выбрать из этого списка нужное слово, после чего нажать клавишу Tab (чтобы остаться в этой же строке программы) или Enter (для перехода на следующую строку).

При вводе имени процедуры, функции, свойства или метода на экране автоматически отображается всплывающая подсказка с описанием структуры вводимой инструкции (рис. 2.6).

При необходимости копирования текста из программного модуля в другие приложения (например, на лист Excel или в документ Word) следует предварительно переключить клавиатуру на русский регистр. В противном случае после операции копирования возможна неверная кодировка русских букв, например, вместо слова «ввод» будет «ââîä».

Для приобретения навыков работы в стандартном модуле VBA рекомендуется набрать простейшую программу, последовательно выполнив следующие действия, более подробно рассмотренные выше:

- 1) запустить приложение MS Excel;
- 2) с помощью клавиш левый Alt + F11 открыть редактор VBA;
- 3) командой меню **Insert** → **Module** создать стандартный модуль Module1;
- 4) выполнить команду **Insert** → **Procedure** и создать процедуру типа Public Sub с именем Проверка;
- 5) ввести программный код в соответствии с образцом, представленным на [рис. 2.7, а](#) (при этом следует указать свою фамилию).

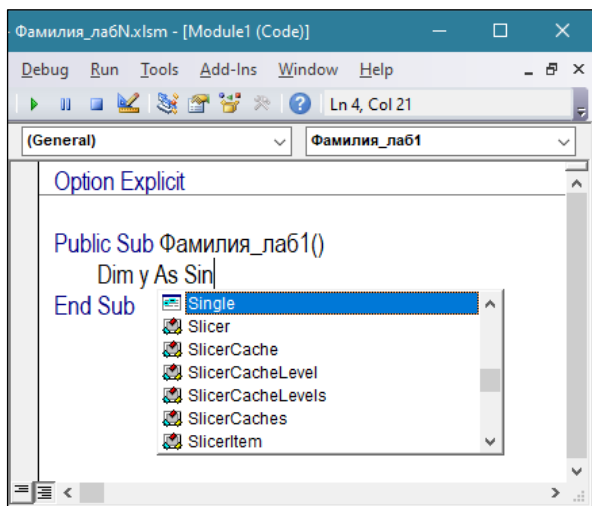


Рис. 2.5. Список возможных слов для дополнения начального фрагмента Sin

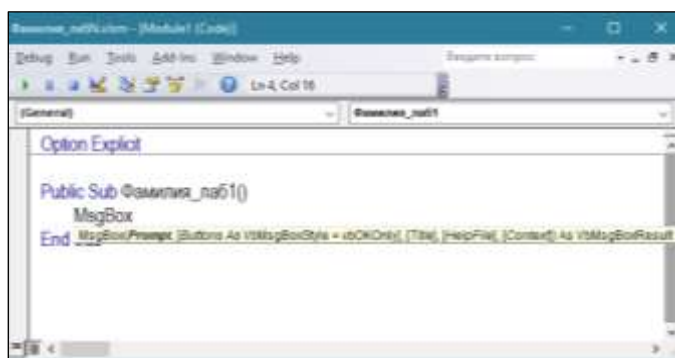



Рис. 2.6. Всплывающая подсказка при вводе функции MsgBox

Программа готова. Осталось запустить ее на выполнение, нажав клавишу F5 (см. [пункт 2.3.4](#)), и получить результат ([рис. 2.7, б](#)). При нажатии на кнопку ОК происходит возврат в окно редактора кода [Module1 (Code)].

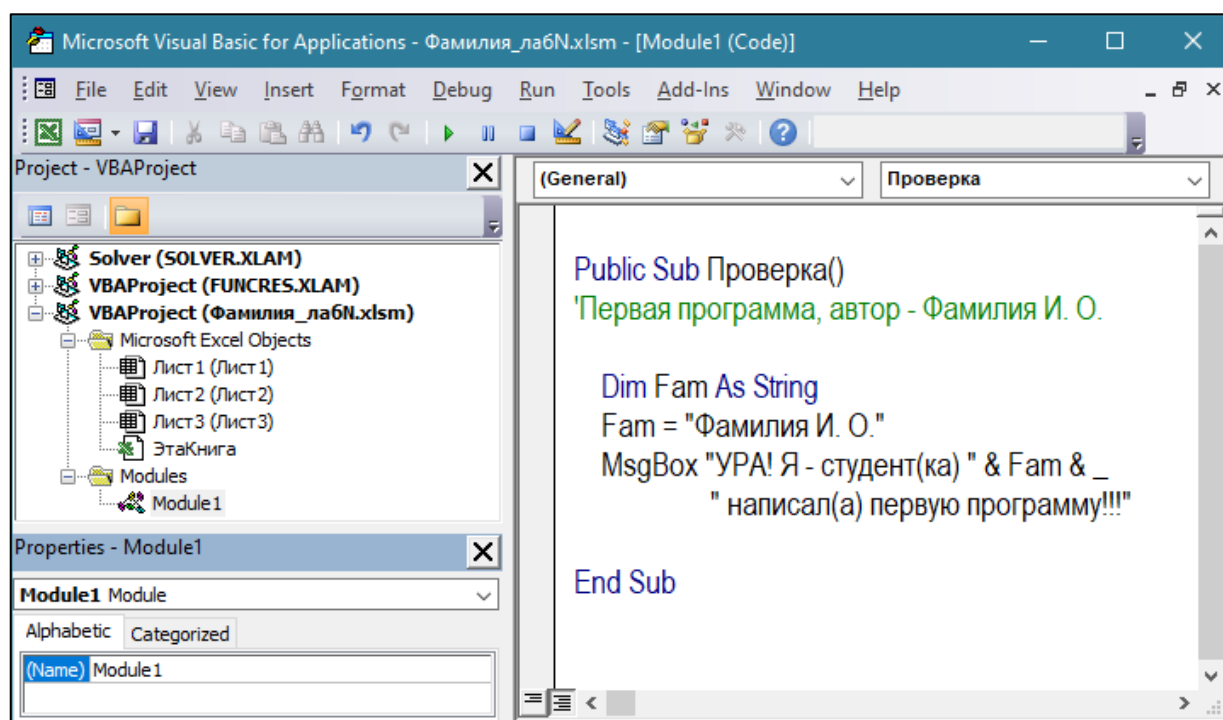
2.3.4. Выполнение процедуры

Запуск готовой процедуры осуществляется командой Run, выполнить которую можно следующими способами:

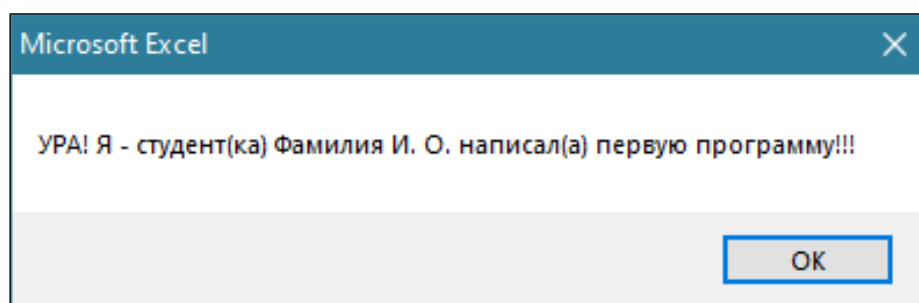
- 1) нажать кнопку  на панели инструментов окна редактора VBA;
- 2) нажать на клавиатуре клавишу F5;
- 3) в главном меню редактора VBA выбрать команду **Run** → **Run Sub/UserForm**.

При этом запускается та процедура, в которой находится курсор. Если курсор установлен за пределами процедуры, то команда Run сначала выдаст запрос на выбор имени запускаемой процедуры. При попытке запустить процедуру

в проекте с отключенными макросами будет выдано соответствующее предупреждение (см. подразд. 2.4).




а



б

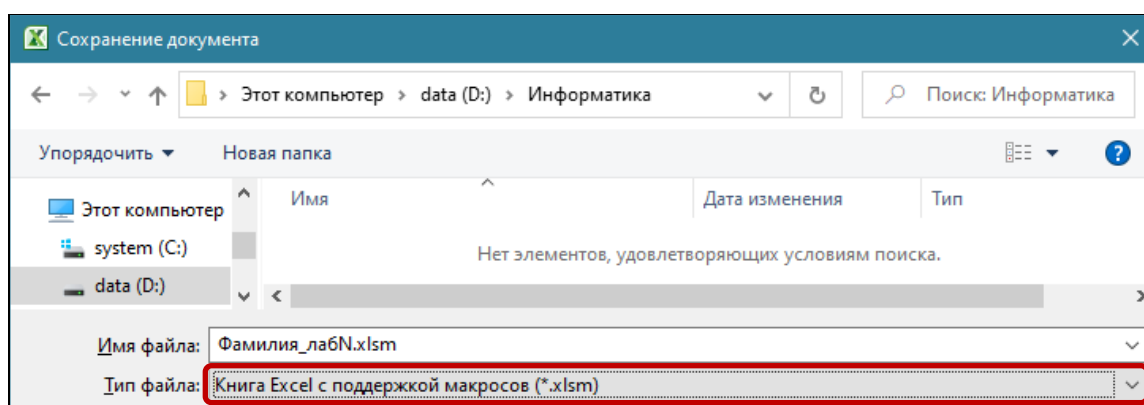
Рис. 2.7. Программный код процедуры Проверка (а) и результат ее выполнения (б)

При обнаружении ошибки во время выполнения программы (см. разд. 11) VBA переходит в режим ожидания (прерывания), для выхода из которого нужно выполнить команду главного меню **Run** → **Reset** или нажать кнопку  на панели инструментов редактора VBA.

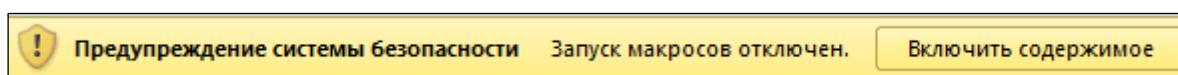
2.4. Сохранение файла с программным кодом

Сохранение на диске файла с программным кодом VBA выполняется обычным для документов Microsoft Office способом. При этом необходимо указывать тип файла *Книга Excel с поддержкой макросов (*.xlm)* (рис. 2.8, а).

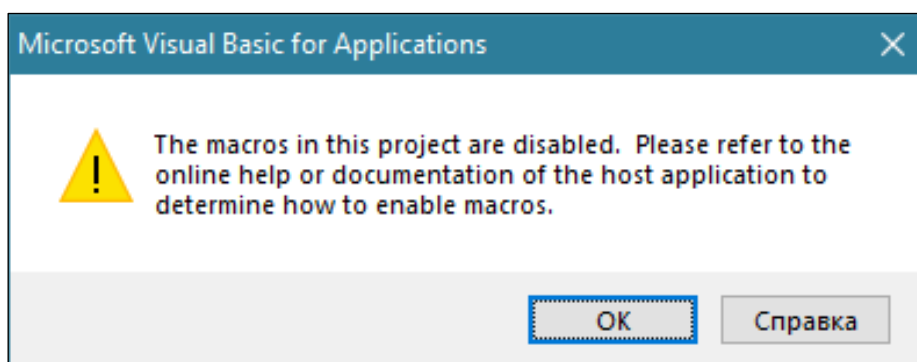
Если при открытии файла, содержащего программный код, на Панели сообщений Excel (под лентой) появилась панель желтого цвета с предупреждением об отключении макросов (рис. 2.8, б), то для продолжения работы следует нажать кнопку Включить содержимое, иначе при запуске программы в редакторе VBA будет выдано сообщение об отключении макросов в данном проекте (рис. 2.8, в) и программа выполняться не будет. В этом случае необходимо закрыть файл и при повторном его открытии включить макросы (см. рис. 2.8, б).



а



б



в

Рис. 2.8. Сохранение файла Excel с программным кодом VBA (а), предупреждение об отключении макросов при открытии рабочей книги (б) и предупреждение об отключении макросов в проекте (в)

2.5. Создание пользовательской формы

В VBA для создания диалоговых окон пользовательских приложений применяются *формы*. Пользовательская форма является одновременно хранилищем элементов управления (ЭУ) и программного кода, который относится к самой форме, размещенным на ней ЭУ и происходящим с ними событиям. *Элементы управления* – специализированные объекты, с помощью которых осуществляется взаимодействие программы с пользователем.

Процесс разработки приложения применительно к организации взаимодействия программы с пользователем состоит из трех этапов:

- 1) создание пользовательского интерфейса;
- 2) установка свойств объектов и определение методов;
- 3) составление программного кода.

2.5.1. Создание пользовательского интерфейса

Разработка пользовательского интерфейса заключается в создании пользовательской формы и наполнении ее элементами управления.

Добавить в проект новую форму (объект UserForm) можно, выполнив команду главного меню редактора VBA **Insert** → UserForm. В результате откроются *окно дизайнера формы* – пустое серое окно формы с именем UserForm1, панель инструментов Toolbox (Элементы управления) и окно свойств формы Properties – UserForm1 (рис. 2.9).

Элементы управления, как и любые другие объекты, имеют свойства, методы и события. Создание ЭУ на форме происходит на начальном этапе разработки приложения и осуществляется с помощью соответствующих кнопок панели инструментов Toolbox.

Основными элементами управления являются следующие.

1. **Label** (надпись, метка) – самый простой ЭУ, представляет собой область для вывода текста,

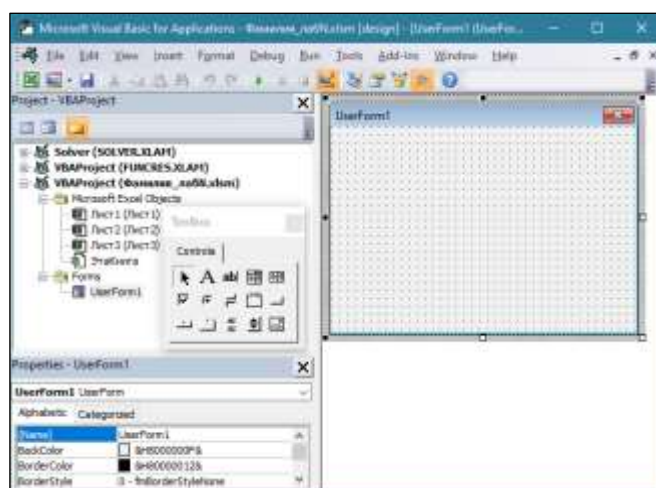


Рис. 2.9. Вид окна дизайнера формы

например, какого-либо пояснения, приглашения для ввода данных и т. п. Во время работы программы пользователь изменять этот текст не может. Как правило, для таких элементов в программе не предусматривается никаких действий.

2. **TextBox** (текстовое поле) – один из наиболее часто применяемых ЭУ. Обычно текстовое поле применяется для приема данных, вводимых пользователем, или для вывода информации. В таких ЭУ возможно изменение данных пользователем.

3. **CommandButton** (командная кнопка) – самый распространенный ЭУ на формах. На большинстве форм обычно имеются кнопки OK и Cancel (Отмена).

Кроме того, панель Toolbox содержит такие ЭУ, как различные списки, переключатели, флажки, вкладки, полосы прокрутки и др. При конструировании формы можно просто перетаскивать ЭУ с панели Toolbox на форму и затем настраивать их размеры, но удобнее работать в следующем порядке:

- 1) на панели Toolbox щелкнуть левой кнопкой мыши на значке ЭУ, который необходимо разместить на форме;
- 2) поместить указатель мыши на то место, где будет располагаться ЭУ;
- 3) при нажатой левой кнопке мыши растянуть появившийся прямоугольник до требуемых размеров;
- 4) отпустить кнопку мыши, завершив создание элемента управления.

Размеры формы и расположенных на ней ЭУ можно изменять. Технология изменения размеров стандартная для Windows: выделить изменяемый элемент, поместить указатель мыши на один из размерных маркеров и протянуть его при нажатой левой кнопке мыши так, чтобы объект принял требуемые размеры. Следует избегать двойного щелчка мышью на форме или элементе управления, поскольку это автоматически влечет за собой создание процедуры обработки этого события.

При проектировании пользовательского интерфейса целесообразно предусмотреть, чтобы в диалоговом режиме пользователь мог перемещаться по ЭУ в заданном порядке нажатием клавиши Enter. Для установления последовательности такого автоматического перехода по ЭУ нужно щелчком правой кнопки мыши на свободном поле формы вызвать контекстное меню, выбрать в нем опцию Tab Order, а затем с помощью кнопок Move Up и Move Down переместить требуемый объект выше или ниже по списку.

Окно редактирования формы поддерживает операции буфера обмена. Как и любые объекты, элементы управления можно копировать, перемещать и удалять обычными способами, применяемыми в приложениях Windows.

2.5.2. Установка свойств объектов на форме

Переход в окно свойств формы (рис. 2.10) или выбранного ЭУ осуществляется по клавише F4. В этом окне можно просматривать и при необходимости изменять установленные свойства формы или ЭУ. Окно свойств состоит из двух частей. В его верхней части находится раскрывающийся список, из которого можно выбрать саму форму или любой расположенный на ней ЭУ. Рабочая часть окна свойств содержит две вкладки – **Alphabetic** (По алфавиту) и **Categorized** (По категориям), отображающие набор свойств соответственно в алфавитном порядке или с систематизацией по категориям, но в любом случае первым всегда представлено свойство **Name**.

Значение свойства **Name** характеризует имя объекта, по которому в программе осуществляется обращение к нему. Это свойство имеют все ЭУ, но для пользователя имя объекта на форме визуально никак не отображается. При создании нового ЭУ его имя формируется автоматически и включает в себя тип объекта и его порядковый номер, например, *Label1*, *TextBox5*, однако не рекомендуется оставлять такие имена без изменения. Имя назначается в соответствии с общими правилами именования элементов VBA (см. подразд. 3.2) и должно быть информативным. Поскольку ЭУ разного типа имеют различные наборы свойств, для удобства программирования, наглядности работы и уменьшения количества ошибок целесообразно в начале имени указывать префикс,

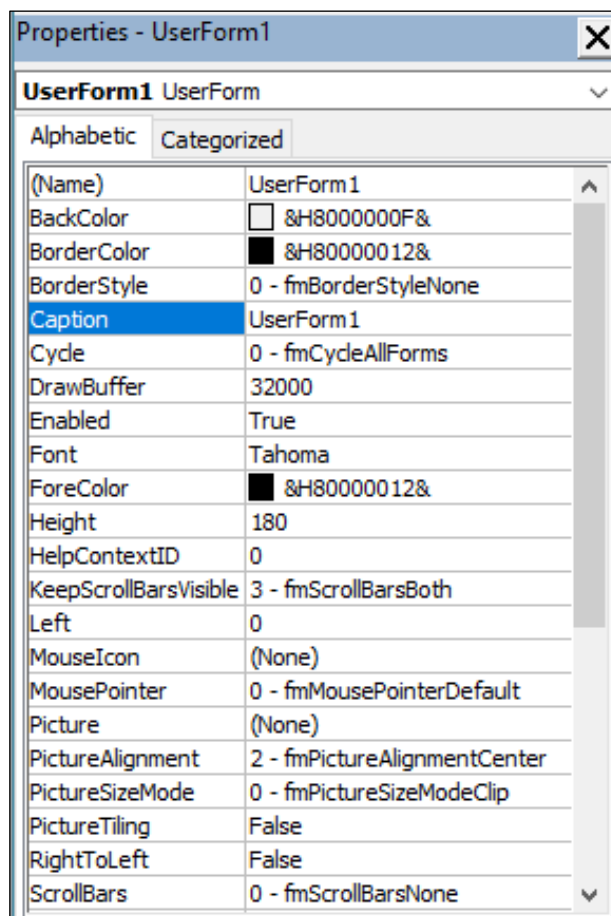


Рис. 2.10. Вид окна свойств формы

обозначающий тип ЭУ (например: Н – надпись, Т – текстовое поле, К – командная кнопка), а смысловое содержание имени должно соответствовать назначению объекта. Например, свойство *Name* объекта *TextBox*, предназначенного для ввода даты, можно задать как *Т_Ввод_даты*.

Набор свойств зависит от типа объекта. Наиболее часто используются следующие свойства:

Caption – заголовок, который визуальнo отображается непосредственно на поверхности ЭУ. Это свойство является одним из основных для элементов управления *Label* и *CommandButton*, но у объекта *TextBox* это свойство отсутствует;

Text – текст, который содержится в поле элемента управления *TextBox*, объекты *Label* и *CommandButton* такого свойства не имеют;

Font – характеристики шрифта (тип, начертание, размер);

ForeColor – цвет шрифта;

Back... – фон (*BackColor* – цвет фона, *BackStyle* – стиль фона);

Border... – граница (*BorderColor* – цвет границы, *BorderStyle* – стиль границы);

TextAlign – выравнивание текста.

Некоторые свойства имеют логические значения: *True* – да, *False* – нет.

В зависимости от вида свойств их значения изменяются в соответствующем поле разными способами:

- вводом значения свойства с клавиатуры;
- выбором значения из раскрывающегося списка, который активизируется щелчком мыши в поле свойства.

Для установки цвета какого-либо элемента (свойство *...Color*) в соответствующем поле в раскрывающемся списке следует выбрать вкладку *Palette* с визуальнo отображаемой палитрой возможных цветов (по умолчанию открыта вкладка *System* с перечнем системных цветовых схем).

Для вставки на форму рисунка на панели *Toolbox* предусмотрен специальный элемент управления *Image*, но удобнее для этой цели применять обычный элемент управления *Label* (в этом случае рисунок удобнее масштабировать). Имя графического файла-источника рисунка указывается в поле свойства *Picture*.

2.5.3. Создание процедуры обработки события

Процесс составления программного кода формы заключается в создании процедур для работы с элементами управления и практически не отличается от работы в стандартном модуле. Для перехода в окно редактора программного

кода открытой формы можно нажать клавишу F7, для возврата в окно дизайнера формы – комбинацию клавиш Shift + F7.

Для создания процедуры обработки события, связанного с определенным ЭУ на форме, необходимо перейти в окно программного кода, выбрать в его верхней части в раскрывающемся списке слева нужный ЭУ, в раскрывающемся списке справа – соответствующее программируемое событие, например, Click (щелчок) (рис. 2.11). В результате этого автоматически создается заготовка процедуры для ввода программного кода, описывающего перечень действий при совершении указанного события. Этот перечень может включать в себя самые разные операции – ввод и вывод данных, проверку условий, вычисления и др. Пример работы с формой рассмотрен в пункте 2.5.4 и разд. 6.

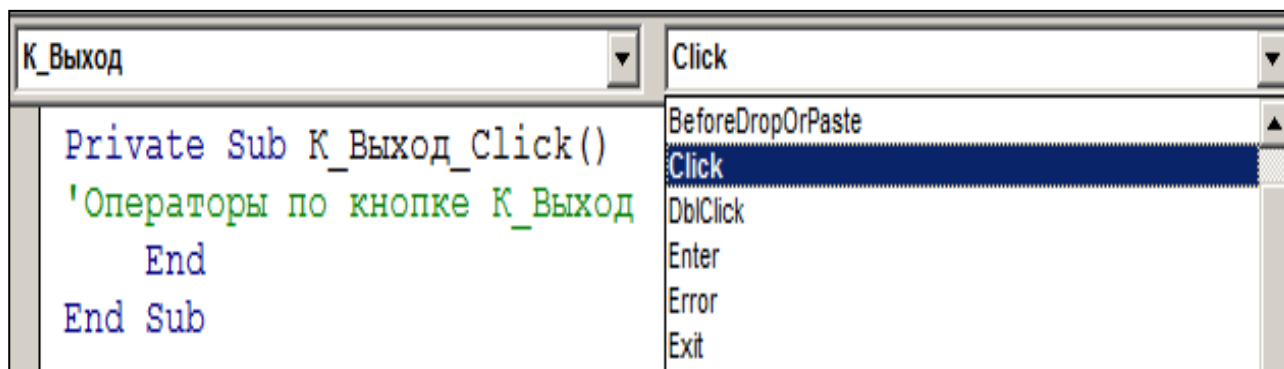


Рис. 2.11. Создание процедуры обработки события (щелчка мышью) для командной кнопки с именем К_Выход

2.5.4. Пример создания пользовательской формы

Рассмотрим создание пользовательской формы на примере организации диалогового режима для сложения двух произвольных чисел.

В редакторе VBA командой меню **Insert** → **UserForm** добавим в проект новую форму UserForm1. На первом этапе разместим на ней элементы управления аналогично рис. 2.12, на котором заголовки объектов заданы по умолчанию. Затем для удобства работы с формой изменим свойства (имена и заголовки) каждого объекта в соответствии со значениями, указанными в табл. 2.1. Кроме того, для всех объектов по желанию можно изменить тип и размер шрифта, а также цвет фона, например, для элемента *Н_Вывод_результата* зададим свойство **ForeColor** (цвет шрифта) – красный, для элемента *Т_Результат* определим свойство **BackColor** (цвет фона) – светло-оранжевый. После установления указанных свойств объектов форма приобретет вид, показанный на рис. 2.13.

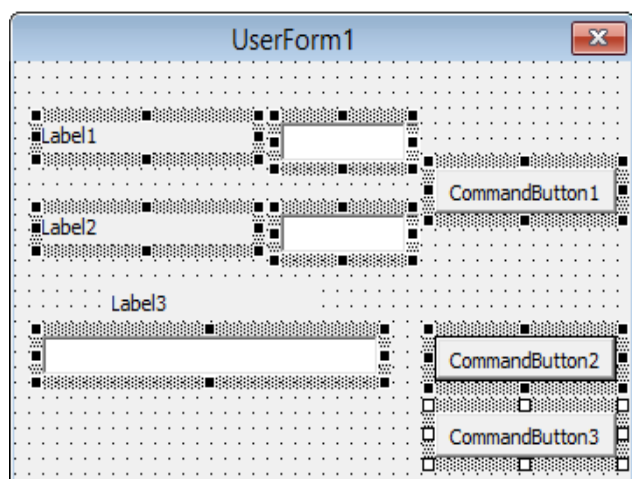


Рис. 2.12. Вид окна формы на первом этапе ее создания




Рис. 2.13. Вид окна формы «Сложение чисел»

Таблица 2.1

Характеристики элементов управления формы

Элемент	Свойство	Значение
UserForm1	Name	Расчет
	Caption	Форма «Сложение чисел»
Label1	Name	Н_Ввод_числа1
	Caption	Введите первое число
TextBox1	Name	T_Число1
Label2	Name	Н_Ввод_числа2
	Caption	Введите второе число
TextBox2	Name	T_Число2
Label3	Name	Н_Вывод_результата
	Caption	Результат
TextBox3	Name	T_Результат
CommandButton1	Name	K_Сумма
	Caption	Сумма
CommandButton2	Name	K_Очистка
	Caption	Очистка
CommandButton3	Name	K_Выход
	Caption	Выход

Запуск формы выполняется командой Run (см. пункт 2.3.4). Закрывать окно формы можно, нажав кнопку  в правом верхнем углу окна. Программный код обработки событий для командных кнопок будет рассмотрен в разд. 6.

3. ЭЛЕМЕНТЫ VBA

3.1. Алфавит VBA

Текст программы составляется и записывается по определенным правилам с помощью *алфавита* языка программирования – разрешенного к использованию набора символов, с помощью которых записываются служебные слова данного языка. Алфавит VBA включает в себя:

- прописные и строчные латинские буквы (A – Z, a – z) и буквы кириллицы (А – Я, а – я);

- десятичные цифры от 0 до 9;

- знаки арифметических операций (+, –, *, /, \, ^);

- знаки пунктуации: пробел, точка, запятая, точка с запятой, двоеточие, круглые скобки, кавычки, апостроф;

- специальные символы (?, !, \$, #, %).

3.2. Общие правила именования элементов VBA

При записи имен переменных, констант и прочих идентификаторов VBA следует придерживаться определенных правил:

- длина имени не должна превышать 255 символов (для форм и элементов управления – 40 символов);

- имя должно начинаться с буквы, допускается использовать цифры и знак подчеркивания;

- имя не может содержать пробелы, пунктуационные знаки (кроме подчеркивания), специальные символы и знаки операций;

- имя должно быть уникальным и не должно повторять зарезервированных слов VBA, имен процедур, операторов, функций и др.;

- имя нельзя использовать более одного раза в одной зоне видимости. Например, все процедуры одного модуля или переменные одной процедуры должны иметь разные имена, однако можно использовать одинаковые имена для переменных, объявленных внутри разных процедур.

Примеры имен:

Alfa, *X*, *summa*, *B25*, *KOR1_3*.

В среде профессиональных программистов идентификаторам принято присваивать информативные имена. Чаще всего для этого используется венгерская нотация (в честь одного из программистов фирмы Microsoft Чарльза Симиони, венгра по национальности), согласно которой имена идентификаторов

предваряются заранее оговоренными префиксами. Например, *nMyAge* – имя переменной, обозначающей «Мой возраст», *n* – целочисленный тип данных. Для облегчения восприятия учебного материала в примерах, приведенных в пособии, используются упрощенные имена идентификаторов.

3.3. Типы данных

Элементарными конструкциями языка VBA являются *данные*. Данные в программе могут использоваться в виде переменных, констант, массивов.

К фундаментальным понятиям любого языка программирования относят *тип данных*, который определяет область допустимых значений из множества данных, набор операций, которые можно применять к этим значениям, правила выполнения операций и способ хранения данных (место хранения и объем в памяти компьютера). Самыми распространенными типами данных являются числовой и строковый (текстовый). Основные числовые типы данных VBA приведены в табл. 3.1. Наиболее часто для хранения целых чисел используют тип *Integer*, для дробных чисел – *Single*.

Таблица 3.1

Основные числовые типы данных VBA

Тип данных	Объем памяти, байт	Описание типа данных	Диапазон допустимых значений
Byte	1	Достаточно малое целое число	От 0 до 255
Integer	2	Не очень большое целое число	От -32768 до 32767
Long	4	Большое целое число	От -2147483648 до 2147483647
Single	4	Вещественное число одинарной точности (до семи десятичных знаков)	От $\approx -3,4 \cdot 10^{38}$ до $\approx 3,4 \cdot 10^{38}$
Double	8	Вещественное число двойной точности (до 15 десятичных знаков)	От $\approx -1,79 \cdot 10^{308}$ до $\approx 1,79 \cdot 10^{308}$

Строковые (символьные) данные содержат текст. Для их хранения используется тип данных *String*. При записи текстовые данные заключаются в кавычки, например: *F* = "Информатика", *ZT* = "Телефон 123456".

Для хранения данных о дате и времени используется тип данных *Date*, такие данные заключаются в символы #, при этом по умолчанию дата записывается в формате #ММ/ДД/ГГГГ#, например: #05/19/2021# (19 мая 2021 г.).

Для хранения логических значений *True* и *False* применяется тип данных Boolean.

В VBA имеется специальный тип Variant, который может содержать данные практически любого вида. Такой тип автоматически устанавливается для данных, если для них в программе в явной форме не задан другой тип. Однако следует избегать такой ситуации, поскольку данные типа Variant требуют большого объема памяти (16 байт), их обработка может приводить к замедлению выполнения программы и возможным ошибкам несоответствия типов данных при вычислениях.

3.3.1. Формы записи вещественных чисел в программе

Целые числа записывают в программе обычным образом, например: 45, –3.

При записи вещественного числа разделителем целой и дробной частей является точка, а не запятая, как в математике. Для записи вещественных чисел на VBA применяются две формы:

основная (естественная) форма записи с фиксированной точкой. Знак плюс и нулевую целую часть можно опустить, например: 5.38, .874, 0.012, –973.6;

экспоненциальная (показательная, нормализованная, константа с порядком) форма записи числа с плавающей точкой в виде:

$$mEr, \tag{3.1}$$

где m – мантисса (число с фиксированной точкой);

E – основание показательной функции (экспонента, означает «...умножить на десять в степени...»);

p – порядок (показатель степени), в стандартном виде представляет собой двузначное число с добавлением при необходимости ведущего нуля.

Математически формулу (3.1) можно интерпретировать как $m \cdot 10^p$.

Мантисса m и порядок p могут быть как отрицательными, так и положительными числами.

В нормализованной научной записи порядок p выбирается такой, чтобы абсолютная величина мантиссы оставалась не меньше единицы, но строго меньше десяти, т. е. $1 \leq |m| < 10$. В таком виде представляет мантиссу и Excel.

В инженерной практике, математике и информатике мантисса обычно выбирается в пределах $0,1 < |m| \leq 1$.

VBA позволяет работать с числами в любой из указанных форм записи.

Число в экспоненциальной форме записывается без пробелов, знак «+» и незначащие нули можно опускать, мантиссу опускать нельзя. Экспоненциальную форму применяют при записи в программе очень больших или очень малых чисел. Аналогично подобные значения, полученные в результате работы программы, VBA отображает также в экспоненциальном представлении.

Примеры записи чисел в экспоненциальной форме:

$37000 = 0,37 \cdot 10^5 \rightarrow 0.37E5$ (или $3.7E4$);

$-2480000 = -0,248 \cdot 10^7 \rightarrow -0.248E7$ (или $-2.48E6$);

$-0,00045 = -0,45 \cdot 10^{-3} \rightarrow -0.45E-3$ (или $-4.5E-4$);

$0,000001 \rightarrow 1E-6$;

$10^{25} \rightarrow 1E25$.

3.4. Константы

Константы – это объекты, значения которых остаются постоянными и не могут быть изменены во время выполнения программы. Константы бывают числовые и строковые, их имена выбираются по общим правилам (см. подразд. 3.2).

Перед первым использованием константы в программе ее необходимо объявить (декларировать), т. е. указать имя константы, тип данных и значение, которое будет храниться в ней. Раздел объявлений констант обычно размещается в начале процедуры (см. пункт 2.3.2), в этом случае константы будут доступны только в данной процедуре. Если константа объявлена в разделе объявлений уровня модуля (см. пункт 2.3.1), то она будет доступна во всех процедурах модуля.

Константы объявляются с помощью оператора `Const`, который имеет следующий формат записи:

`Const Имя_константы As Тип_данных = Значение`

Примеры объявления констант:

`Const k5 As Integer = -937` 'Целочисленная константа $k5 = -937$

`Const w As Single = 1.2E-5` 'Вещественная константа $w = 1.2E-5$

В одном операторе `Const` можно объявлять несколько констант, например:

`Const g As Single = 9.81, n As Byte = 2, txt As String = "Информатика"`

Значение числа π VBA «не знает», его нужно задавать в программе, например:

`Const pi As Single = 3.1415926` или `Const pi As Single = 3.14`

3.5. Переменные

Переменная – это объект, который предназначен для хранения данных. С технической точки зрения переменная – это отдельный именованный участок памяти для хранения данных определенного типа. Переменная характеризуется именем, типом данных и значением, которое в отличие от константы может изменяться в ходе выполнения программы.

3.5.1. Объявление переменных

Перед первым использованием переменной в программе ее необходимо объявить – указать ее имя и тип данных. Аналогично константам раздел объявлений переменных обычно располагается в начале процедуры. Объявление переменных осуществляется оператором `Dim`, который имеет следующий формат записи:

`Dim Имя_переменной As Тип_данных`

Примеры объявления переменных:

`Dim Tok As Byte` 'Целочисленная переменная *Tok* типа `Byte`

`Dim Fam As String` 'Строковая переменная *Fam*

В одном операторе `Dim` можно объявлять несколько переменных, например:

`Dim V As Single, Pos As Integer` 'Вещественная переменная *V* одинарной
'точности и целочисленная переменная
'*Pos* типа `Integer`

При объявлении переменной каждого типа резервируется соответствующий объем памяти для ее хранения (см. [табл. 3.1](#)), что способствует оптимизации обработки данных, обеспечивает дополнительный контроль имени, множества допустимых значений и набора выполняемых операций. Это делает программу более понятной, надежной, ускоряет ее работу, уменьшает количество ошибок. Кроме того, только в этом случае будут действовать всплывающие подсказки VBA при написании и отладке программы.

При объявлении переменные инициализируются, т. е. принимают некоторое значение «по умолчанию», например: числовые переменные – значение 0, символьные переменные – "" (строка пустой длины). Для записи в переменную необходимого значения применяется операция присваивания (см. [пункт 5.2.1](#)).

3.5.2. Запрет использования необъявленных переменных

Для запрета использования переменных без их объявления необходимо в разделе объявлений уровня модуля (см. [пункт 2.3.1](#)) указать инструкцию

Option Explicit

В этом случае при попытке использования предварительно необъявленной переменной редактор VBA остановит выполнение программы и сообщит об ошибке (см. [разд. 11](#)).

Для автоматического размещения команды Option Explicit при загрузке редактора VBA необходимо в меню Tools (Инструменты) → Options (Настройки) → Editor (Редактор) активировать команду Require Variable Declaration (Требуется объявление переменной). Следует иметь в виду, что эта установка начнет действовать только после нового запуска Excel.

3.6. Встроенные функции

В VBA имеется большой набор встроенных функций, использование которых существенно упрощает программирование. Эти функции можно разделить на следующие основные категории: математические функции, функции обработки строк, проверки, определения и преобразования типов данных, функции даты и времени, финансовые функции и др.

Общий формат записи любой функции²:

Имя_функции[(Аргументы)]

Обращение к функции осуществляется указанием ее имени (идентификатора), сразу после которого в круглых скобках через запятую записываются ее аргументы – исходные данные для расчета значения функции. Некоторые функции не требуют аргументов, например: функция Now возвращает текущую системную дату и время, функция Rnd возвращает случайное значение в диапазоне [0; 1). При этом возможно применение общего формата записи таких функций с пустым списком аргументов в скобках: Now(), Rnd().

² Квадратными скобками в описаниях инструкций здесь и далее выделены составляющие инструкции, которые могут отсутствовать в зависимости от особенностей их использования (например, они могут принимать значения по умолчанию).

Аргументами функций могут быть константы, переменные, арифметические выражения, другие функции.

3.6.1. Математические функции

Основные встроенные математические функции VBA приведены в табл. 3.2.

Таблица 3.2

Основные встроенные математические функции VBA

Функция	Математическая запись	Запись на VBA	Примечание
Целая часть числа	—	Fix(x)	Выделение целой части числа x ; дробная часть при этом отбрасывается, число не округляется, например: Fix(1.2) = 1; Fix(-2.7) = -2
Наибольшее целое число, не превосходящее исходное	—	Int(x)	Int(9.7) = 9; Int(-2.7) = -3
Абсолютное значение	$ x $	Abs(x)	
Квадратный корень	\sqrt{x}	Sqr(x)	$x > 0$
Показательная функция	e^x	Exp(x)	
Логарифм натуральный	$\ln x$	Log(x)	$x > 0$
Синус	$\sin x$	Sin(x)	
Косинус	$\cos x$	Cos(x)	
Тангенс	$\operatorname{tg} x$	Tan(x)	
Арктангенс	$\operatorname{arctg} x$	Atn(x)	

Аргумент тригонометрической функции должен быть в радианах, формула для перевода градусов в радианы: $x_{\text{рад}} = x_{\text{град}} \cdot \pi / 180^\circ$.

Например, $\sin 60^\circ \rightarrow \operatorname{Sin}(60 * 3.14 / 180)$.

Функции, отсутствующие в числе встроенных, необходимо записывать с помощью имеющихся встроенных функций, используя известные математические соотношения (табл. 3.3).

Примеры записи функций в VBA

Функция	Математическая формула для расчета	Запись на VBA
Корень n -й степени $\sqrt[n]{x}$	$x^{1/n}$	$x ^ (1 / n)$
Логарифм числа b по основанию a ($\log_a b$)	$\frac{\ln b}{\ln a}$	$\text{Log}(b) / \text{Log}(a)$
Десятичный логарифм $\lg x$	$\log_{10} x$	$\text{Log}(x) / \text{Log}(10)$
Арксинус $\arcsin x$	$\arctg \left(\frac{x}{\sqrt{1-x^2}} \right)$	$\text{Atn}(x / \text{Sqr}(1 - x ^ 2))$
Арккосинус $\arccos x$	$\arctg \left(\frac{\sqrt{1-x^2}}{x} \right)$	$\text{Atn}(\text{Sqr}(1 - x ^ 2) / x)$
Арккотангенс $\text{arcctg } x$	$\frac{\pi}{2} - \arctg x$	$\text{pi} / 2 - \text{Atn}(x)$
Котангенс $\text{ctg } x$	$\frac{1}{\text{tg } x}$	$1 / \text{Tan}(x)$

3.6.2. Строковые функции

VBA содержит встроенные функции, предназначенные для обработки строковых данных. Основными из них являются следующие.

$\text{Len}(\text{Строка})$ – возвращает (определяет) количество символов в *Строке*, т. е. длину текста. Пробелы в строке учитываются, например:

$C = \text{Len}(\text{"день"})$ Результат: $C = 4$.

$C = \text{Len}(\text{" день "})$ Результат: $C = 7$ (четыре буквы и три пробела).

$\text{Val}(\text{Строка})$ – выдает число по его символьной записи. При этом число выделяется только в том случае, если оно начинается с первой позиции в *Строке* или ему предшествуют пробелы. Если в строке имеются посторонние символы (не числа), то выдается 0. Пробелы между цифрами игнорируются. Функция Val обычно используется, если с числами, встречающимися в тексте, нужно выполнить какие-либо арифметические действия. Примеры:

$C = \text{Val}(\text{"Информатика"})$ Результат: $C = 0$.

$C = \text{Val}(\text{"12 3"})$ Результат: $C = 123$.

Str(Число) – функция, обратная Val, выдает символьную запись числа и обычно используется, если в текст нужно вставить какие-либо числа, например:

C = Str(123) → C = "123".

Asc(Строка) – возвращает (определяет) числовой код первого символа в *Строке*, например: C = Asc("Test") Результат: C = 84.

Chr(Число) – функция, обратная функции Asc, возвращает (выдает) символ по его числовому машинному коду, например:

C = Chr(196) Результат: C = "Д".

Команда Chr(13) выдает символ конца строки и обычно используется в операторах вывода данных для перевода позиции печати на новую строку.

Instr([n], Строка, Фрагмент) – выдает начальную позицию первого вхождения *Фрагмента* в *Строку*. Поиск ведется, начиная с позиции *n* (если *n* не задано, то с первой позиции). Если *Фрагмент* не найден, то возвращается 0, например:

A = "Добрый день, студенты!"

B = "де"

C = Instr(A, B) или C = Instr(A, "де") Результат: C = 8.

C = Instr(10, A, "де") Результат: C = 17.

C = Instr(A, "утро") Результат: C = 0.

Left(Строка, k) – выделяет *k* левых крайних символов в *Строке*, например:

C = Left("Добрый день", 6) Результат: C = "Добрый".

Right(Строка, k) – выделяет *k* правых крайних символов в *Строке*, например:

C = Right("Добрый день", 4) Результат: C = "день".

Mid(Строка, n, [k]) – выделяет подстроку из *k* символов, начиная с позиции *n*. Если *k* не задано или количество символов справа от *n* меньше *k*, то выделяются все символы, начиная с позиции *n*, до конца *Строки*, например:

A = "Добрый день, студенты!"

C = Mid(A, 8, 4) Результат: C = "день".

C = Mid(A, 8) Результат: C = "день, студенты!".

Помимо функции Mid в VBA имеется оператор Mid, формат записи которого следующий:

Mid(Строка, n, [k]) = Фрагмент

Этот оператор заменяет в *Строке*, начиная с позиции *n*, *k* символов на *Фрагмент* (но не более количества символов в *Строке*). Оператор Mid в отличие от функции Mid записывается слева от знака равенства. Если *k* не указано, *Фрагмент* берется целиком. Пример:

A = "Добрый день"

Mid(A, 1, 6) = "Плохой"

Результат: A = "Плохой день".

4. ЗАПИСЬ ВЫРАЖЕНИЙ

Выражения представляют собой последовательную запись операндов, т. е. констант, переменных, функций, или любых их комбинаций, образованную при помощи знаков арифметических операций, конкатенации, операций отношения и логических операций.

4.1. Арифметические операции

В VBA всего семь арифметических операций (табл. 4.1): первые пять – стандартные и две дополнительные.

Таблица 4.1

Арифметические операции, используемые в VBA

Знак операции	Описание операции	Примеры	
		выражение	результат Y при A = 17, B = 5
+	Сложение	$Y = A + B$	22
–	Вычитание	$Y = A - B$	12
*	Умножение	$Y = A * B$	85
/	Деление	$Y = A / B$	3.4
^	Возведение в степень	$Y = A ^ B$	1419857
\	Целочисленное деление (деление без остатка, при этом дробная часть отбрасывается, число не округляется)	$Y = A \setminus B$	3
Mod	Остаток от деления целых чисел	$Y = A \text{ Mod } B$	2

Арифметические выражения соответствуют общепринятым алгебраическим выражениям. Результатом вычисления является число.

К строкам арифметические операции неприменимы. Если требуется разбить строку на отдельные части или выполнить другие операции со строками, используются строковые функции (см. пункт 3.6.2).

4.2. Операция слияния строк (конкатенация)

При слиянии (объединении) текстовых данных формируются более длинные строки (этот процесс напоминает составление поезда из вагонов). Операция слияния текстовых строк называется *конкатенацией* и в VBA реализуется с помощью символов & или +. Рекомендуется применять &, так как в этом случае производится автоматическое преобразование числовых значений в строковые, например:

$A = \text{"Лекция "}$ $B = \text{"№ 2"}$ $C = \text{"Информатика"}$

$D = A \& B \& \text{" – "}$ & C Результат: $D = \text{"Лекция № 2 – Информатика"}$.

Конкатенация используется практически в любой программе VBA, например, в операторе вывода данных MsgBox "y=" & y.

4.3. Операции отношения (сравнения)

Для сравнения значений выражений применяются операции отношения, представленные в табл. 4.2. Результатом выполнения операции отношения является логическое значение *True* (да), если утверждение истинно, или *False* (нет), если утверждение ложно.

Таблица 4.2

Операции отношения (сравнения), используемые в VBA

Знак операции	Описание операции	Примеры		
		выражение	результат вычислений при $X = 18, Y = 3$	результат сравнения
=	Равно	$X = Y$	$18 = 3$	<i>False</i>
		$X \text{ Mod } 2 = 0$	$0 = 0$	<i>True</i>
		$X \text{ Mod } Y = 1$	$0 = 1$	<i>False</i>
		$Y = X \setminus 6$	$3 = 3$	<i>True</i>
< >	Не равно	$X \text{ Mod } 4 < > 0$	$2 < > 0$	<i>True</i>
<	Меньше	$X < Y$	$18 < 3$	<i>False</i>
>	Больше	$\text{Abs}(Y - X) > 0$	$15 > 0$	<i>True</i>
< =	Меньше или равно	$\text{Fix}(Y / X) < = 1$	$0 < = 1$	<i>True</i>
> =	Больше или равно	$X \setminus 10 > = 1$	$1 > = 1$	<i>True</i>

4.4. Сравнение строк

Каждый символ имеет свой машинный десятичный код. Так, буквы латинского алфавита от А до z кодируются числами от 65 до 122, буквы кириллицы от А до я – числами от 192 до 255 соответственно. Набор символов с кодами 1 – 127 одинаков и соответствует общепринятым международным кодам. Набор символов с кодами 128 – 255 может быть разным в зависимости от принятой кодировки. Узнать код любого символа в VBA можно с помощью строковой функции Asc(), описание которой приведено в [пункте 3.6.2](#).

Строки можно указывать в сочетании со всеми операциями отношения. Символьные переменные сравниваются в десятичных кодах посимвольно, начиная с первого символа. Знак «<» означает выше в алфавитном порядке (т. е. с меньшим кодом), знак «>» – старше (т. е. ниже) в алфавитном порядке. Если длины сравниваемых переменных разные, то при сравнении меньшая по длине переменная дополняется пробелами (до одинаковой длины). При проверке на совпадение строки должны быть полностью идентичными с учетом как ведущих, так и концевых пробелов.

Примеры:

«информатика» < «физика» (так как буква «и» выше (т. е. раньше) по алфавиту, чем «ф»);

«информатика» < «история» (так как первые буквы в словах одинаковые, то сравниваются между собой вторые символы – «н» раньше по алфавиту, чем «с»);

«мы » < > «мы» (так как в конце первого слова есть пробел).

4.5. Логические операции

При записи выражений на VBA, например, при одновременной проверке нескольких условий, могут применяться *логические операции*:

Not – логическое отрицание;

And – логическое И;

Or – логическое ИЛИ.

Пример их использования рассмотрен в [подразд. 7.2](#).

4.6. Порядок действий в выражении

Выражение содержит операции из нескольких категорий, которые выполняются в порядке их приоритета.

В первую очередь всегда вычисляются функции. При этом следует различать операции, которые относятся непосредственно к функции или к ее аргументу. Операция, которая относится к функции, записывается после скобки, закрывающей список аргументов функции. Операция, которая относится к аргументу функции, записывается непосредственно после него внутри скобок, ограничивающих список аргументов функции.

После функции сначала выполняются арифметические операции, затем конкатенация и операции отношения, последними выполняются логические операции. При этом соблюдается следующая последовательность действий (в порядке снижения их приоритета):

- возведение в степень (^);
- умножение и деление (*, /);
- целочисленное деление (\);
- вычисление остатка от деления (Mod);
- сложение и вычитание (+, -);
- конкатенация (&);
- операции отношения (<, >, =, <> и др.);
- логическая операция Not;
- логическая операция And;
- логическая операция Or.

Для изменения порядка действий используются круглые скобки. На одном уровне скобок операции одинакового приоритета выполняются последовательно слева направо. Запись выражения в VBA выполняется в одну строку, в выражении нельзя опускать знак умножения, ставить подряд два знака операций. Если в числителе или знаменателе дроби имеется многочлен (вычисление суммы или разности), то его обязательно нужно заключать в скобки.

Рассмотрим на конкретном примере влияние приоритета операций.

Пусть нужно вычислить $X = \text{Fix}(8.7 + 3) \setminus 2 - 11 \text{ Mod } 3 ^ 2$.

Выполним действия в порядке их приоритета:

- 1) $\text{Fix}(8.7 + 3) = 11$;
- 2) $3 ^ 2 = 9$;
- 3) $11 \setminus 2 = 5$;
- 4) $11 \text{ Mod } 9 = 2$;
- 5) $X = 5 - 2 = 3$. Результат: $X = 3$.

Теперь вычислим значение X по формуле с аналогичным содержанием, но с другой расстановкой скобок: $X = \text{Fix}(8.7) + 3 \setminus 2 - (11 \text{ Mod } 3) ^ 2$.

Выполним действия в порядке их приоритета:

- 1) $\text{Fix}(8.7) = 8$;
- 2) $3 \setminus 2 = 1$;
- 3) $11 \text{ Mod } 3 = 2$;
- 4) $2 ^ 2 = 4$;
- 5) $X = 8 + 1 - 4 = 5$. Результат: $X = 5$.

Примеры записи арифметических и логических выражений на языке VBA приведены соответственно в табл. 4.3 и 4.4. В последнем столбце табл. 4.3 представлены примеры характерных ошибок.

Таблица 4.3

Примеры записи арифметических выражений на VBA

Математическая запись	Правильная запись на языке VBA	Неправильная запись (примеры ошибок)
ab	$a * b$	ab
$(x+y)(z+k)$	$(x + y) * (z + k)$	$(x + y) (z + k)$
$\frac{a}{bc}$	$a / (b * c)$ или $a / b / c$	$a / b * c$
$\frac{bx+w}{5+\sqrt{kx}}$	$(b * x + w) / (5 + \text{Sqr}(k * x))$	$b * x + w / 5 + \text{Sqr}(k * x)$
$\sqrt[5]{x}$	$x ^ (1 / 5)$	$x ^ 1 / 5$
$\sqrt[3]{(x+a)^2}$	$(x + a) ^ (2 / 3)$	$(x + a) ^ 2 / 3$
$\sin^2 x^3$	$\text{Sin}(x ^ 3) ^ 2$	$\text{Sin} ^ 2 (x ^ 3)$
e^{x+y}	$\text{Exp}(x + y)$	$\text{Exp} ^ (x + y)$
$\lg^3 x^2$	$(\text{Log}(x ^ 2) / \text{Log}(10)) ^ 3$	$\text{Log}(x ^ 2) ^ 3 / \text{Log}10$
$\cos \sqrt[5]{e^{2x}}$	$\text{Cos}(\text{Exp}(2 * x) ^ (1 / 5))$	$\text{Cos} * ((\text{Exp} ^ 2x) ^ 1 / 5))$

Примеры записи операций сравнения и логических выражений на VBA

Математическая запись	Запись на языке VBA
$b^2 - 4ac > 0$	<code>b ^ 2 - 4 * a * c > 0</code>
$k \neq m$	<code>k <> m</code>
$-5 \leq x \leq 7$	<code>x >= -5 And x <= 7</code>
$x > y$ и $y > 0$	<code>x > y And y > 0</code>
$x = 0$ или $x = 1$	<code>x = 0 Or x = 1</code>

Пример. Записать на VBA математическое выражение:

$$z = \frac{\sin 6^\circ + e^{a+b} + (2^a)^b}{|ab| - \frac{a}{bc} + \sqrt[5]{a^2}} + \sin^2 a^3.$$

В соответствии с рассмотренными выше правилами записи выражений на VBA результат будет иметь вид:

$$z = (\sin(6 * 3.14 / 180) + \exp(a + b) + (2 ^ a) ^ b) / (Abs(a * b) - a / (b * c) + a ^ (2 / 5)) + \sin(a ^ 3) ^ 2.$$

5. УПРАВЛЕНИЕ ВВОДОМ И ВЫВОДОМ ДАННЫХ

Ввод и вывод информации могут осуществляться разными способами, например, путем обмена данными между программой и диалоговым окном Windows, между программой и ячейками Excel и др.

5.1. Выбор и очистка рабочего листа Excel

В программе на VBA при необходимости можно обращаться к разным рабочим листам Excel. Для выбора нужного рабочего листа используется метод `Select` (выбор) для объекта `Worksheets` (рабочий лист):

```
Worksheets("Имя листа").Select
```

Пример:

```
Worksheets("Лист1").Select
```

'Выбор рабочего листа с именем Лист1.

Для полной очистки листа от всех данных и форматов применяется инструкция `Worksheets("Имя листа").Cells.Clear`.

Очистка заданного диапазона ячеек текущего рабочего листа выполняется с помощью объекта `Range` и соответствующего метода, например:

`Range("A:F").Clear` 'Очистка содержимого столбцов с А по F;

`Range("3:8").Clear` 'Очистка содержимого строк с третьей по восьмую;

`Range("A1:C5").Clear` 'Очистка содержимого диапазона ячеек с А1 по С5;

`Range("A1:C5").Delete` 'Удаление ячеек с А1 по С5 целиком, включая все объекты в них.

5.2. Операторы ввода данных

Для работы практически любой программы необходимы исходные данные. Передать их в программу можно несколькими способами.

5.2.1. Ввод данных с помощью оператора присваивания

Запись значений переменных непосредственно в тексте программы осуществляется с помощью оператора присваивания `Let`. Он служит для вычисления значения выражения и сохранения этого значения в переменной.

Общий формат записи оператора:

`[Let] Имя_переменной = Выражение`

При записи программы ключевое слово `Let` можно опускать.

Примеры:

`A = 2.1 : X = 6`

`Summa = A + Cos(X) ^ 2`

Следует различать оператор присваивания и алгебраическое равенство. Оператор `Y = A + B` означает для ЭВМ: сложить содержимое ячеек памяти, отведенных для размещения значений переменных *A* и *B*, и поместить результат в ячейку памяти, отведенную для значения переменной *Y*.

В программировании широко используется конструкция оператора присваивания типа `X = X + 1` (с точки зрения математики это равенство не имеет смысла). При выполнении записанной таким образом команды в программе к текущему значению переменной *X* прибавляется 1 и результат помещается в ту же ячейку, заменив бывшую там информацию на новую.

При присвоении переменным строковых значений их необходимо заключать в кавычки:

$T = \text{"Параметр 1"}$,

а значения типа Дата/Время – заключать в символы #:

$D = \text{\#11/29/2021\#}$.

5.2.2. Ввод данных из ячеек рабочего листа Excel

Для ввода информации с текущего листа Excel в программу используется инструкция $\text{Cells}(i, j)$, которая в данном случае выступает как функция ввода данных. Формат записи:

Имя_переменной = $\text{Cells}(i, j)$,

где i, j – порядковые номера соответственно строки и столбца, на пересечении которых находится ячейка рабочего листа Excel и куда предварительно введено исходное значение.

Пример:

$z = \text{Cells}(1, 2)$

После выполнения этой команды переменной z присвоится значение, которое хранится в ячейке, находящейся в первой строке (первая цифра аргумента функции Cells) и во втором столбце (вторая цифра), т. е. в ячейке B1 рабочего листа Excel.

В командах ввода и вывода данных на лист Excel вместо инструкции $\text{Cells}(i, j)$ с указанием номера строки и столбца можно непосредственно указывать адрес нужной ячейки, записывая его в квадратных скобках, например, предыдущей операции равнозначна команда $z = [B1]$.

5.2.3. Ввод данных в диалоговом окне

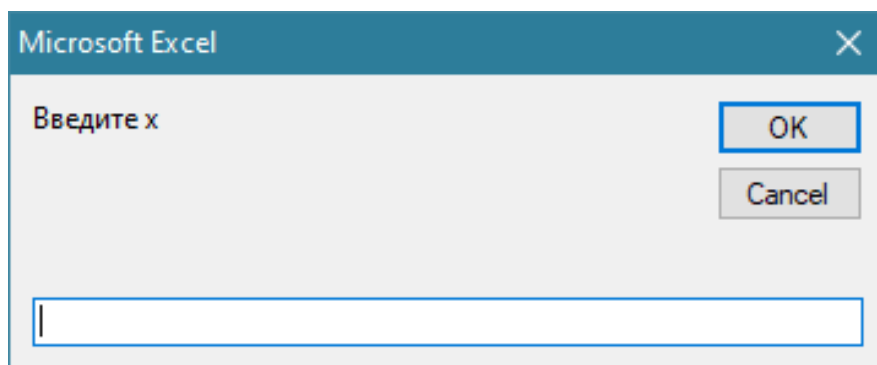
Ввод данных непосредственно в ходе выполнения программы (т. е. в интерактивном режиме с пользователем) выполняется с помощью диалогового окна ввода информации, реализуемого функцией InputBox , простейший формат которой имеет вид:

Имя_переменной = $\text{InputBox}(\text{"Сообщение"})$

В ходе работы программы при выполнении данной команды на экране монитора появляется диалоговое окно, содержащее текст *Сообщения*, а также поле для ввода одного значения. Например, по команде

```
x = InputBox("Введите x")
```

на экран выводится диалоговое окно, представленное на [рис. 5.1](#).



[Рис. 5.1](#). Вид диалогового окна для ввода данных

По команде `InputBox` выполнение программы приостанавливается в ожидании ввода данных с клавиатуры и нажатия одной из кнопок. После ввода информации и нажатия на кнопку `OK` (или клавишу `Enter`) переменной `x` присваивается значение типа `String` (строковый тип данных), содержащее символы, введенные в поле ввода. Если в поле ввода в появившемся диалоговом окне ввести, например, цифры 23, то в переменную `x` будет записан текст "23", а не число 23. Текстовая строка "23" представляет собой лишь визуальное отображение числа, но не является числом в буквальном смысле слова, т. е. с ним, как и с любым другим текстом, нельзя производить никаких арифметических действий.

Для преобразования строкового типа данных в числовой тип служит функция `Val(Строка)` (см. [пункт 3.6.2](#)).

При записи в программе команды

```
x = Val(InputBox("Введите x"))
```

и последующем вводе в поле ввода цифр 23 переменной `x` будет присвоено числовое значение 23.

Следует иметь в виду, что при использовании подобной конструкции с функцией `Val` при вводе дробного числа в качестве разделителя целой и дробной частей обязательно используется точка.

5.2.4. Генерирование случайных чисел

Для генерирования случайного числа используется функция Rnd, которая возвращает значение в диапазоне [0; 1).

Простейший формат записи:

Имя_переменной = Rnd

Для формирования целого случайного числа x в диапазоне $[Min; Max]$ можно использовать формулу: $x = \text{Fix}(\text{Rnd} * (Max - Min + 1) + Min)$.

Пример:

$z = \text{Fix}(\text{Rnd} * 101 - 50)$ Генерирование числа z в диапазоне $[-50; +50]$.

Каждый раз при запуске VBA формируется одна и та же последовательность случайных чисел. Чтобы сформировать другую последовательность, нужно переустановить базу генератора случайных чисел с помощью оператора Randomize [база], который обычно указывается в начале программы. В качестве базы целесообразно использовать функцию Timer, которая возвращает количество секунд текущего времени суток от полуночи:

Randomize Timer

5.3. Операторы вывода данных

5.3.1. Вывод данных в диалоговое окно

Вывод информации в диалоговое окно выполняется с помощью оператора MsgBox, простейший формат которого имеет вид:

MsgBox Сообщение

В результате выполнения этой команды на экране появляется диалоговое окно с заголовком *Microsoft Excel*, содержащее текст *Сообщения*. Выполнение программы приостанавливается до нажатия пользователем кнопки ОК.

Обычно *Сообщение* представляет собой выводной список, содержащий различные данные и сопровождающие их текстовые пояснения, которые заключаются в кавычки. Для объединения нескольких элементов выводного списка в одну строку в операторе вывода их разделяют знаком &, слева и справа от которого обязательно ставят пробел:

"Текст1 " & X & "Текст2 " & Y

Изменить заголовок окна MsgBox можно следующим образом:

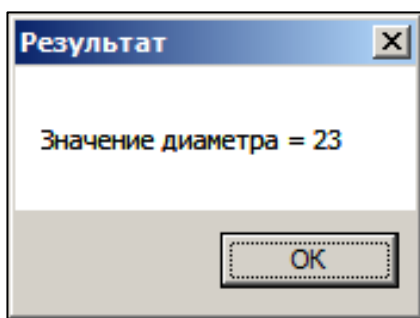


Рис. 5.2. Пример вывода
в диалоговое окно

MsgBox Сообщение, , "Заголовок"

Пример:

MsgBox "Значение диаметра = " & d, , "Результат"

В данном случае в окно с заголовком *Результат* выводятся комментарий и текущее значение переменной *d* (рис. 5.2).

5.3.2. Вывод данных на рабочий лист Excel

Вывод данных на текущий рабочий лист Excel выполняется с использованием инструкции `Cells(i, j)`, которая в этом случае выступает как оператор вывода (в отличие от рассмотренной в пункте 5.2.2 функции ввода):

`Cells(i, j) = Сообщение`

В результате этой команды указанное *Сообщение* помещается в ячейку с адресом, определяемым номером строки *i* и номером столбца *j*.

Примеры:

`Cells(1, 1) = "x="` 'Вывод в ячейку A1 текста *x=*

`Cells(1, 2) = x` 'Вывод в ячейку B1 текущего значения переменной *x*

Другая форма записи вывода этих данных в ячейки листа Excel:

`[A1] = "x="`

`[B1] = x`

5.3.3. Вывод данных в окно отладки Immediate

Окно отладки *Immediate* обычно располагается под окном программного кода. Если этого окна нет, то его можно вывести, нажав `Ctrl + G` или выбрав в главном меню VBA команду **View** → Immediate Window.

Для вывода данных в окно отладки применяется метод `Print` (печать) объекта `Debug` (отладчик). Формат записи:

`Debug.Print [Сообщение]`

Пример: `Debug.Print "Значение диаметра =" & d`

Как видно из этого примера, команда `Debug.Print` используется аналогично команде `MsgBox`.

Пустой (т. е. без *Сообщения*) метод `Debug.Print` выводит пустую строку.

Помимо знака `&` в методе `Print` возможно использование разделителей списка выводимых данных. При этом знак `«;»` означает вывод очередного значения непосредственно за предыдущим, знак `«,»` – переход к началу новой зоны печати (окно отладки делится на вертикальные зоны по 14 позиций каждая). При вводе данных знак `«;»` между элементами выводного списка можно опускать, VBA добавит его автоматически.

Запятая или точка с запятой в конце выводного списка в команде `Debug.Print` подавляет переход на новую строку (следующий `Debug.Print` начнет печать в той же строке).

Пример: `Debug.Print "Результат y="; y;`

В любом операторе вывода (`MsgBox`, `Cells`, `Debug.Print`) возможен не только вывод готовых результатов, но и одновременный расчет и вывод:

Примеры:

`MsgBox "Если диаметр = " & d & ", то радиус = " & d / 2`

`Cells(1, 2) = Sin(w) + 5`

`Debug.Print "s = "; s, " k1+k2 = "; k1 + k2`

5.3.4. Форматированный вывод данных

Для вывода числового значения в заданном формате в диалоговое окно или в окно отладки служит функция `Format` с указанием требуемого количества десятичных знаков.

Примеры:

`Debug.Print z` 'Вывод переменной `z` в окно отладки обычным образом;

`Debug.Print Format(z, "0.00")` 'Вывод переменной `z` в окно отладки с двумя десятичными знаками;

`MsgBox Format(z, "0.000")` 'Вывод переменной `z` в диалоговое окно с тремя десятичными знаками.

На [рис. 5.3](#) представлены разные способы записи операторов вывода и соответствующие им результаты при $x = 5$, $y = x / 3$.

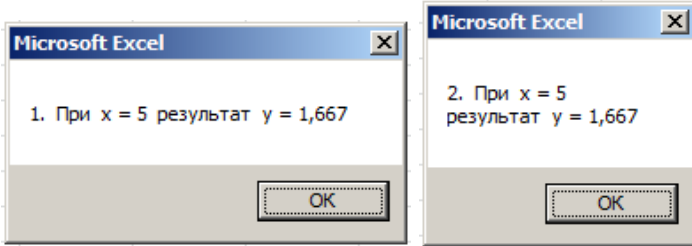
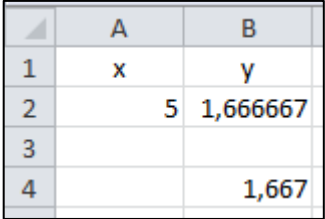
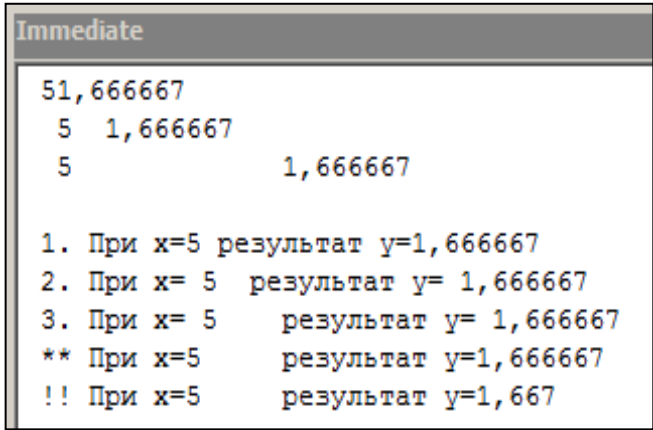
<p>'Вывод в диалоговое окно MsgBox "1. При x = " & x & " результат y = " & Format(y, "0.000") 'Chr(13) - переход к началу следующей строки MsgBox "2. При x = " & x & Chr(13) & "результат y = " & Format(y, "0.000")</p>	
<p>'Вывод на лист Excel Cells(1, 1) = "x" : Cells(1, 2) = "y" 'Вывод текстовых заголовков Cells(2, 1) = x : Cells(2, 2) = y 'Вывод значений x и y 'Вывод в ячейку B4 значения y в формате с тремя десятичными знаками Cells(4, 2).NumberFormat = "0.000" : Cells(4, 2) = y</p>	
<p>'Вывод в окно отладки Debug.Print x & y 'Сцепление значений Debug.Print x; y '1 разряд для знака, 1 между значениями Debug.Print x, y '1 разряд для знака, с начала след. зоны Debug.Print 'Пустая строка (переход к началу след. строки) 'Вывод в окно отладки с комментариями Debug.Print "1. При x=" & x & " результат y=" & y Debug.Print "2. При x="; x; " результат y="; y Debug.Print "3. При x="; x, " результат y="; y Debug.Print "*** При x=" & x, " результат y=" & y Debug.Print "!! При x=" & x, " результат y=" & Format(y, "0.000")</p>	

Рис. 5.3. Примеры записи операторов вывода и соответствующие им результаты

5.3.5. Форматирование данных на листе Excel

Для вывода чисел в заданном формате на лист Excel функцию Format применять нельзя, так как она преобразует данные в текстовый тип, что ограничивает возможности их дальнейшей обработки. Для установки заданного числового формата в ячейке листа Excel используется свойство NumberFormat. Пример записи:

`Cells(2, 1).NumberFormat = "0.000"` 'Установка в ячейке A2 числового формата с тремя знаками после запятой.

Для цветового оформления ячеек на листе Excel применяются следующие инструкции:

`Cells(i, j).Font.Color = RGB(r, g, b)` 'Установка цвета шрифта в ячейке;

`Cells(i, j).Interior.Color = RGB(r, g, b)` 'Установка цвета фона (заливка) ячейки, где r, g, b – интенсивность (целое число в диапазоне от 0 до 255) соответственно красной, зеленой и синей составляющих определяемого цвета. Необходимые значения r, g, b можно узнать в режиме выбора цветового спектра в любом приложении Windows. Например, инструкция `Cells(1, 2).Font.Color = RGB(255, 0, 0)` устанавливает в ячейке B1 красный цвет шрифта.

Для изменения цвета одновременно в нескольких ячейках необходимо в приведенных выше командах вместо указания одной ячейки `Cells(i, j)` задать диапазон требуемых ячеек с помощью ключевого слова Range, например:

`Range("D3:F7").Interior.Color = RGB(190, 255, 202)` 'Заливка ячеек с D3 по F7 светло-зеленым цветом.

Для форматирования содержимого в ячейке служат команды:

`Cells(i, j).Font.Bold = True` 'Жирный шрифт в ячейке;

`Cells(i, j).Font.Italic = True` 'Курсивный шрифт в ячейке;

`Cells(i, j).Borders.LineStyle = xlContinuous` 'Задание границ ячейки (обрамление);

`Cells(i, j).HorizontalAlignment = xlCenter` 'Выравнивание содержимого ячейки по горизонтали;

`Cells(i, j).VerticalAlignment = xlCenter` 'Выравнивание содержимого ячейки по вертикали.

5.4. Взаимодействие с элементами управления на форме

При организации диалоговых окон для установки свойства объекта или использования его метода в программе применяются следующие инструкции:

Объект.Свойство = Значение

Объект.Метод [Параметры...]

Здесь *Объект* – имя настраиваемого объекта, *Свойство* – характеристика, которую нужно изменить, *Значение* – новая установка свойства, *Метод* – команда, которая используется для изменения объекта.

Извлечение свойства объекта осуществляется с помощью инструкции

Значение = Объект.Свойство

Указанные инструкции применительно к организации ввода и вывода данных рассмотрены ниже.

5.4.1. Ввод данных с использованием формы

Для ввода данных с использованием формы в большинстве случаев применяется элемент управления *TextBox* (текстовое поле).

Присвоение переменной значения, введенного в текстовое поле, т. е. извлечение информации из окна ввода осуществляется по команде:

Имя_переменной = *Имя_TextBox*.Text

Пример для строковой переменной: *fat* = *Фамилия*.Text.

Пример для числовой переменной: *z* = Val(*Число_z*.Text).

Свойство Text является основным для текстового поля и предполагается по умолчанию, поэтому при записи команд ввода его можно опускать, например:

z = Val(*Число_z*.Text) равнозначно *z* = Val(*Число_z*).

Для помещения фокуса клавиатуры, т. е. установки курсора на требуемый объект, применяется инструкция *Имя_объекта*.SetFocus.

5.4.2. Вывод данных с использованием формы

Вывод данных на форму с использованием элементов управления выполняется следующим образом.

Для элемента управления *Label*:

Вывод заданного текста в надпись:

Имя_надписи.Caption = "Текст"

Пример: *Фамилия*.Caption = "Иванов С.Н."

Очистка надписи:

Имя_надписи.Caption = "" '(пара кавычек, внутри ничего нет)

Вывод значения переменной в надпись:

Имя_надписи.Caption = Имя_переменной

Пример: *Значение_x.Caption = x*

Для элемента управления *TextBox*:

Вывод в текстовое поле значения переменной, в том числе при установке значения переменной по умолчанию:

Имя_TextBox.Text = Имя_переменной

Пример: *Число_z.Text = z*

Очистка текстового поля:

Имя_TextBox.Text = "" (пара кавычек, внутри ничего нет)

Свойства *Caption* для надписи и *Text* для текстового поля являются основными свойствами этих элементов управления и предполагаются по умолчанию, при записи команд вывода их можно опускать, например:

Фамилия.Caption = "Иванов С.Н." равнозначно *Фамилия = "Иванов С.Н."*,

Число_z.Text = z равнозначно *Число_z = z*.

Пример ввода и вывода данных с использованием формы рассмотрен в [разд. 6](#).

6. ЛИНЕЙНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ ПРОЦЕССЫ

Программы, реализующие алгоритм линейного вычислительного процесса, являются самыми простыми. Они содержат только операторы ввода данных, присваивания и вывода результатов. Операторы записываются последовательно друг за другом в естественном порядке их следования и выполняются только один раз. В общем случае процедура линейного вычислительного процесса имеет следующую структуру:

```
Sub Имя_процедуры()  
    'Объявление переменных (Dim ...)  
    'Объявление констант (Const ...)  
    'Тело процедуры:  
    'Ввод исходных данных  
    'Вычисления  
    'Вывод результатов  
End Sub
```

Пример Л1. Вычислить сумму чисел Z и R , где $Z = 2a - b$, $R = \pi + x$, $a = 5$, $b = 1,3$, x – произвольное.

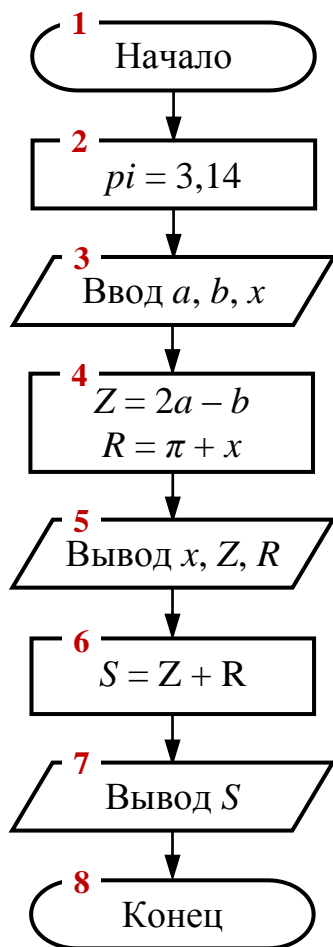


Рис. 6.1. ГСА решения примера Л1

Составим графическую схему алгоритма (ГСА) решения задачи (рис. 6.1). Она содержит следующие блоки:

- 1 – начало алгоритма;
- 2 – определение значения константы π ;
- 3 – ввод значений a , b , x ;
- 4 – расчет Z и R по заданным формулам;
- 5 – вывод значений x , Z и R ;
- 6 – расчет суммы чисел Z и R и присвоение полученного значения переменной S ;
- 7 – вывод результата расчета S ;
- 8 – конец алгоритма.

В соответствии с представленным алгоритмом составим программу расчета на VBA. Для ознакомления с различными способами ввода и вывода данных реализуем в программе все рассмотренные в подразд. 5.2 и 5.3 операторы. При этом предусмотрим ввод значений переменных a и b с листа Excel, значения x – с клавиатуры во время работы программы. Текст программы с подробными комментариями приведен на рис. 6.2.

До запуска программы на исполнение на листе Excel должны быть введены исходные данные: в ячейку A2 – значение переменной a (число 5), в ячейку B2 – значение переменной b (число 1,3).

Рассмотрим другие примеры линейных задач, демонстрирующие также применение строковых функций.

Пример Л2. Ввести произвольный текст, состоящий из двух – трех предложений. В этом тексте найти позицию первой точки, вывести весь текст после этой точки и определить его длину.

ГСА решения примера Л2 и текст программы с подробными комментариями представлены на рис. 6.3.

```

Option Explicit      'Запрет использования необъявленных переменных
Sub Пример_Л1()      'Начало процедуры
'Вычисление суммы чисел Z и R

'Оъявление переменных
Dim a As Integer      'a – целочисленная
Dim b As Single, x As Single 'b, x – вещественные
Dim Z As Single, R As Single 'Z, R – вещественные
Dim S As Single      'S – вещественная (сумма Z и R)

Const pi As Single = 3.14      'Объявление константы pi

Worksheets("Лист1").Select      'Выбор листа Excel с именем "Лист1"
Range("3:6").Clear      'Очистка строк Excel с третьей по шестую

'Тело процедуры
a = Cells(1, 2)      'Ввод a из ячейки B1 листа Excel
b = Cells(2, 2)      'Ввод b из ячейки B2 листа Excel
x = Val(InputBox("Введите x"))      'Ввод x в диалоговое окно

Z = 2 * a - b      'Вычисление Z
R = pi + x      'Вычисление R

'Вывод текстовых заголовков в ячейки Excel
Cells(3, 1) = "x=" : Cells(4, 1) = "Z=" : Cells(5, 1) = "R="
Cells(6, 1) = "Сумма Z + R ="

Cells(3, 2) = x      'Вывод значения x в ячейку B3

'Вывод значений Z и R в формате с двумя десятичными знаками
Cells(4, 2).NumberFormat = "0.00" : Cells(4, 2) = Z      'в ячейку B4
Cells(5, 2).NumberFormat = "0.00" : Cells(5, 2) = R      'в ячейку B5

S = Z + R      'Вычисление суммы S

'Вывод значения S в формате с двумя десятичными знаками
Cells(6, 2).NumberFormat = "0.00" : Cells(6, 2) = S      'в ячейку B6

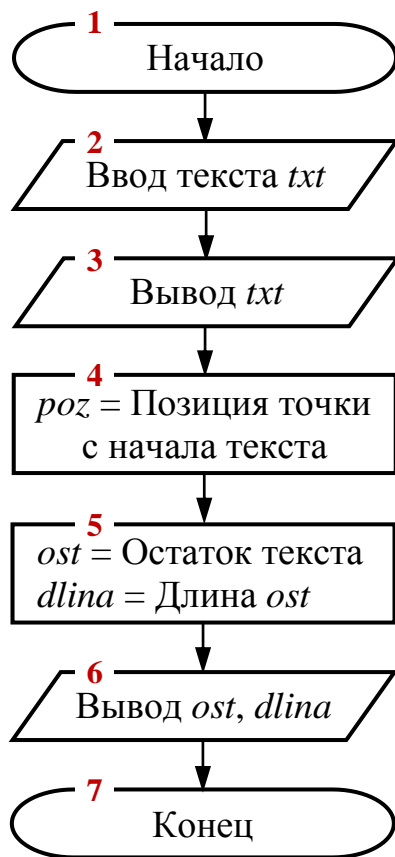
'Вывод S в диалоговое окно
MsgBox "Сумма Z + R =" & Format(S, "0.00")

'Вывод всех значений расчетов в окно отладки Immediate
Debug.Print "При a ="; a, "b ="; b, "x ="; x
Debug.Print "Z ="; Z, "R ="; R, "Сумма Z + R ="; S

End Sub      'Конец процедуры

```

Рис. 6.2. Листинг программы решения примера Л1



а

```

Sub Пример_Л2()
'Применение строковых функций (поиск позиции точки в тексте)

'Объявление переменных
Dim txt As String      'Исходный текст
Dim poz As Byte        'Позиция первой точки
Dim ost As String      'Остаток текста после точки
Dim dlina As Byte      'Длина остатка текста

txt = InputBox("Введите текст") 'Ввод текста в диалоговое окно

'Вывод исходного текста в окно отладки Immediate
Debug.Print "Исходный текст:"; txt

poz = InStr(txt, ".")      'Поиск позиции точки с начала текста
ost = Mid(txt, poz + 1)    'Выделение остатка текста после точки
dlina = Len(ost)          'Определение длины остатка текста

'Вывод результата работы программы в диалоговое окно в две строки
MsgBox "ost=" & ost & Chr(13) & "Длина =" & dlina

End Sub
  
```

б

Рис. 6.3. Решение примера Л2: ГСА (а) и листинг программы (б)

Пример Л3. Преобразовать текст «Имя Отчество Фамилия» в текст «Фамилия И.О.».

ГСА решения **примера Л3** представлена на **рис. 6.4**, текст программы с подробными комментариями – на **рис. 6.5**. Характерной особенностью данной задачи является то, что в ней применяются практически все основные функции обработки текстовых данных.

Рассмотрим общие принципы ввода и вывода информации на форму.

Пример Л4. Составить программный код для реализации вычислений с использованием формы «Сложение чисел», рассмотренной в **пункте 2.5.4**.

Для решения данной задачи необходимо создать процедуры обработки нажатия кнопок *Сумма*, *Очистка* и *Выход*.

ГСА решения **примера Л4** и окно редактора VBA с листингом программного кода процедур обработки событий, связанных с элементами управления пользовательской формы, представлены на **рис. 6.6**.

По кнопке *Сумма* осуществляются чтение исходных чисел с формы, расчет их суммы и вывод результата, по кнопке *Очистка* – очистка всех текстовых полей, по кнопке *Выход* – завершение работы с помощью оператора End.

Для удобства использования программы в ней с помощью метода SetFocus предусмотрена автоматическая установка курсора:

- после вывода результата – на кнопку *Очистка*;
- после очистки – в поле ввода первого числа.

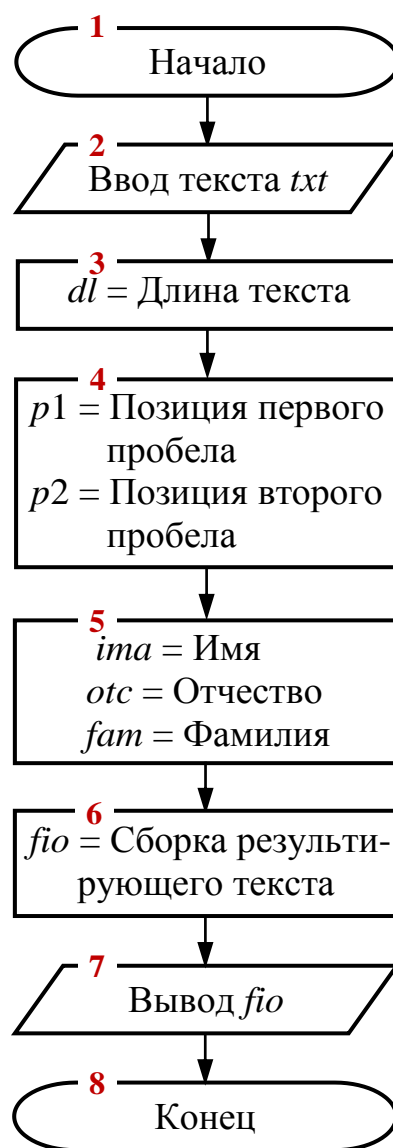


Рис. 6.4. ГСА решения **примера Л3**

```

Sub Пример_Л3()
'Преобразование текста "Имя Отчество Фамилия" в "Фамилия И.О."

'Объявление переменных
Dim txt As String      'Исходный текст
Dim dl As Byte         'Длина исходного текста
Dim p1 As Byte         'Позиция первого пробела
Dim p2 As Byte         'Позиция второго пробела
Dim fam As String      'Фамилия
Dim ima As String      'Имя
Dim otc As String      'Отчество
Dim fio As String      'Результирующий текст

'Комментарии приведены для текста txt = "Петр Сергеевич Иванов"

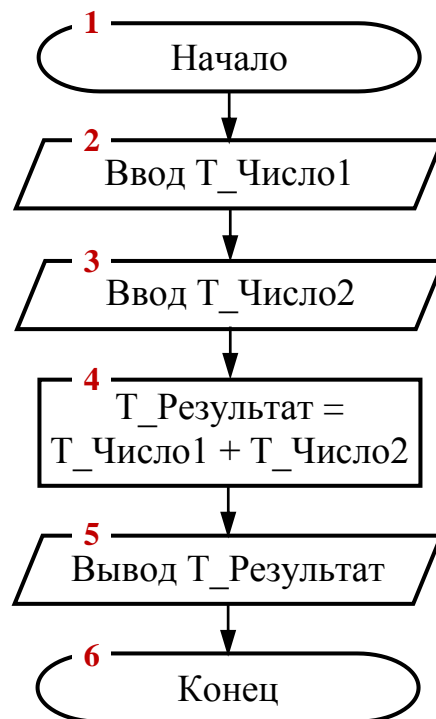
txt = InputBox("Имя Отчество Фамилия", "Введите текст")
dl = Len(txt)           'Длина текста ( = 21)
p1 = InStr(txt, " ")    'Позиция первого пробела ( = 5)
p2 = InStr(p1 + 1, txt, " ") 'Позиция второго пробела ( = 15)
ima = Left(txt, p1 - 1) 'Имя ( = Петр)
otc = Mid(txt, p1 + 1, p2 - p1 - 1) 'Отчество ( = Сергеевич)
fam = Right(txt, dl - p2) 'Фамилия ( = Иванов)

'Сборка результирующего текста
fio = fam + " " + Left(ima, 1) + "." + Left(otc, 1) + "."

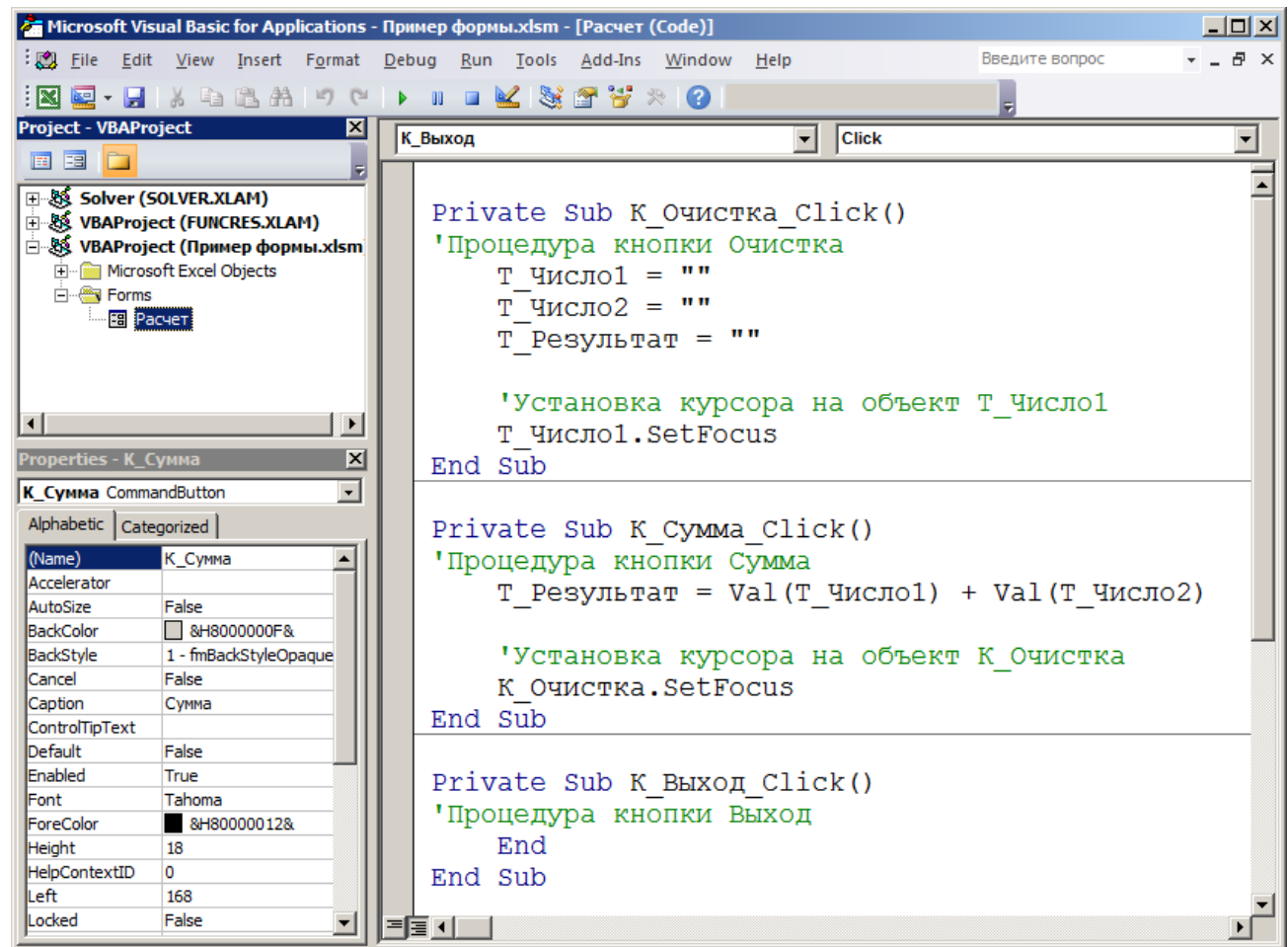
'Вывод результатов работы программы в окно отладки Immediate
Debug.Print "Исходный текст: "; txt; " Длина текста = "; dl
Debug.Print "Позиция первого пробела = "; p1
Debug.Print "Позиция второго пробела = "; p2
Debug.Print "Имя "; ima
Debug.Print "Отчество "; otc
Debug.Print "Фамилия "; fam
Debug.Print "Результат "; fio
End Sub

```

Рис. 6.5. Листинг программы решения примера Л3



а



б

Рис. 6.6. Решение примера Л4: ГСА (а) и окно редактора VBA с листингом программы (б)

7. РАЗВЕТВЛЯЮЩИЕСЯ ВЫЧИСЛИТЕЛЬНЫЕ ПРОЦЕССЫ

Большинство задач невозможно реализовать только с помощью линейной структуры, поскольку они обычно содержат различные условия, в зависимости от выполнения которых выбирается то или иное направление решения.

Любое изменение естественной (линейной) последовательности выполнения блоков называется *переходом*. Переходы бывают двух видов:

условные – реализуются с помощью условного оператора *If*, который имеет блочный и однострочный формат записи;

безусловные – реализуются с помощью оператора *GoTo*.

7.1. Оператор условного перехода *If*

7.1.1. Блочная форма оператора *If*

Блочная форма условного оператора применяется, когда после проверки условия необходимо выполнить программные операторы, записанные в нескольких строках (блок операторов).

Самая простая – безальтернативная (усеченная) форма записи, которая применяется тогда, когда в случае истинности условия необходимо выполнить набор операторов, а при невыполнении условия никаких действий нет:

```
If Условие Then  
    ... 'Операторы при выполнении (истинности) Условия  
End If
```

Условие – это логическое выражение, записанное с помощью операций отношения (см. [подразд. 4.3](#)) и принимающее значение *Истина* или *Ложь*. Если условие истинно, то выполняются операторы, записанные после слова *Then*. Если условие ложно, то никаких действий не выполняется, управление передается оператору, следующему за *End If*.

В тех случаях, когда при истинности условия необходимо выполнить один набор операторов, а при невыполнении условия – другой, применяется альтернативная форма оператора *If*, которая записывается в таком виде:

```
If Условие Then  
    ... 'Операторы при выполнении (истинности) Условия  
Else  
    ... 'Операторы при невыполнении Условия  
End If
```


Пример У1 (Разветвление на две ветви). Вычислить значение функции z для произвольного значения аргумента x :

$$z = \begin{cases} \sin \pi x & \text{при } x \leq 0; \\ \ln x & \text{при } x > 0. \end{cases}$$

ГСА решения **примера У1** приведена на [рис. 7.1, а](#). Для выполнения расчетов в алгоритме определена константа pi (блок 2). Для наглядности и контроля работы программы предусмотрена дополнительная переменная nv , предназначенная для хранения номера ветви, по которой выполняется расчет z при конкретном значении x . В данном примере при истинности условия $x \leq 0$ (блок 4) выполняется вычисление по первой формуле (блок 5). Если условие ложно, а значит $x > 0$, выполняется вычисление по второй формуле (блок 6). Во всех случаях после расчета осуществляется вывод результата (блок 7).

Листинг программы решения **примера У1**, составленной в соответствии с ГСА (см. [рис. 7.1, а](#)), с подробными комментариями приведен на [рис. 7.1, б](#).

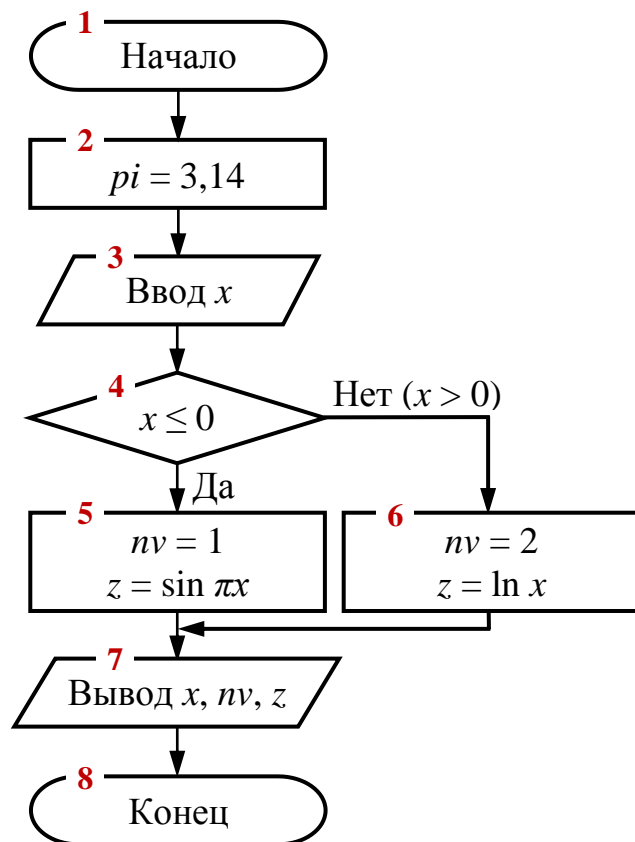
При решении многовариантных задач необходимо последовательно проверять каждое условие. Как только встретится истинное, будет выполнен соответствующий набор команд и осуществлен выход за End If. Если ни одно из условий не окажется истинным, выполняются команды после End If.

Пример У2 (Разветвление на три ветви). Вычислить значение функции z для $b = -4,5$ и произвольного значения аргумента x :

$$z = \begin{cases} b \sin x & \text{при } x < 3; \\ \sqrt{x} & \text{при } 3 \leq x \leq 5; \\ \cos x & \text{при } x > 5. \end{cases}$$

ГСА решения **примера У2** приведена на [рис. 7.2, а](#). При истинности первого условия $x < 3$ (блок 4) производится расчет по первой формуле (блок 5). Если первое условие не выполняется, значит, $x \geq 3$, т. е. левая часть двойного условия $3 \leq x \leq 5$ уже выполнена и осталось проверить только его правую часть $x \leq 5$ (блок 6). При истинности условия $x \leq 5$ производится расчет по второй формуле (блок 7), в противном случае – по третьей (блок 8). Независимо от варианта расчета после него всегда осуществляется вывод результата (блок 9).

Листинг программы решения **примера У2**, составленной в соответствии с ГСА (см. [рис. 7.2, а](#)), с подробными комментариями приведен на [рис. 7.2, б](#).



а

```

Sub Пример_У1()
'Разветвление на 2 ветви (блочная альтернативная форма If)
'Объявление переменных
'x – аргумент функции, nv – номер ветви, z – функция
Dim x As Single, nv As Byte, z As Single

Const pi As Single = 3.14 'Объявление константы pi

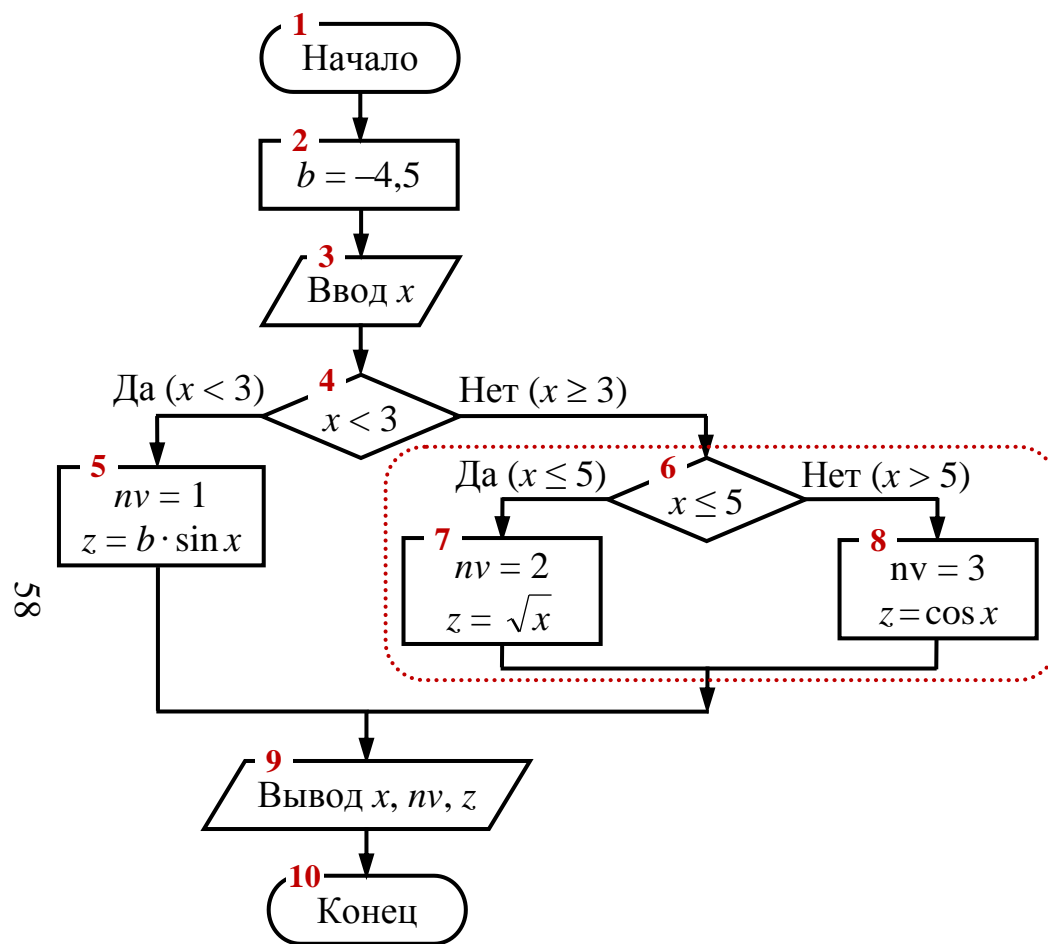
'Ввод значения x в диалоговое окно
x = Val(InputBox("Введите x"))

If x <= 0 Then           'Проверка условия
    nv = 1 : z = Sin(pi * x) 'ветвь 1, вычисление значения z
Else
    nv = 2 : z = Log(x)      'ветвь 2, вычисление значения z
End If                   'Завершение If

'Вывод результата в окно отладки
Debug.Print "При x ="; x, "ветвь"; nv, "z ="; z
End Sub
  
```

б

Рис. 7.1. Решение примера У1: ГСА (а) и листинг программы (б)



а

```

Sub Пример_У2()
'Разветвление на 3 ветви (блочная альтернативная форма If)
'Объявление переменных
'x – аргумент функции, nv – номер ветви, z – функция
Dim x As Single, nv As Byte, z As Single

Const b As Single = -4.5 'Объявление константы b

'Ввод значения x в диалоговое окно
x = Val(InputBox("Введите x"))

If x < 3 Then 'Проверка условия 1
    nv = 1 : y = b * Sin(x) 'вычисление z по первой ветви
Else
    If x <= 5 Then 'Проверка условия 2
        nv = 2 : z = Sqr(x) 'вычисление z по второй ветви
    Else
        nv = 3 : z = Cos(x) 'вычисление z по третьей ветви
    End If 'Завершение вложенного If
End If 'Завершение внешнего If

'Вывод результата в окно отладки
Debug.Print "При x ="; x, "ветвь"; nv, "z ="; z
End Sub
  
```

б

Рис. 7.2. Решение примера У2: ГСА (а) и листинг программы (б)

При составлении ГСА и программы следует руководствоваться правилом: количество условных блоков (операторов If) всегда на единицу меньше числа ветвей, на которые расходуется вычислительный процесс.

Как видно из [примера У2](#), операторы If могут быть вложенными (см. объединенные пунктирной линией блоки 6, 7, 8 в ГСА на [рис. 7.2, а](#)). На любом уровне вложенности структура каждого внутреннего оператора If должна полностью входить во внешний для него If, например:

```
If Условие1 Then
    ...
Else
    ...
    {
        If Условие2 Then
            ...
        Else
            ...
        End If
    }
    ...
End If
```

Если между внешним Else и внутренним If нет других операторов, то их обычно соединяют в одно слово Elseif, а команду End If, соответствующую внутреннему If, исключают. Записанная таким способом оптимизированная структура оператора If для [примера У2](#) представлена на [рис. 7.3](#).

If x < 3 Then	'Проверка условия 1
nv = 1 : y = b * Sin(x)	'вычисление z по первой ветви
Elseif x <= 5 Then	'Проверка условия 2
nv = 2 : z = Sqr(x)	'вычисление z по второй ветви
Else	
nv = 3 : z = Cos(x)	'вычисление z по третьей ветви
End If	'Завершение If

[Рис. 7.3](#). Оптимизированная структура оператора If для решения [примера У2](#)

7.1.2. Однострочный (линейный) оператор If

В VBA возможна запись оператора If в одну строку. Такая форма записи называется *линейной*, при этом ключевая фраза End If отсутствует. В общем случае линейная форма условного оператора If имеет вид:

If *Условие* Then *Операторы1* [Else *Операторы2*],

где *Операторы1* – операторы при выполнении *Условия*,

Операторы2 – операторы при невыполнении *Условия*.

Линейный оператор If также может быть записан в безальтернативной и альтернативной формах, например:

If t > z Then z = t ^ 2 'Безальтернативная форма

If t > z Then z = t ^ 2 Else z = 0 'Альтернативная форма

Линейный If применяется редко, обычно в тех случаях, если проверяется не более двух условий, а выполняемые операторы достаточно кратки.

7.2. Объединение условий с помощью логических операций

В одном операторе If можно объединять несколько одновременно проверяемых простых условий в более сложные логические формулы с помощью операций And (и), Or (или). Независимо от этого при составлении ГСА рекомендуется каждое условие записывать в отдельном блоке.

В общем случае при использовании блочной формы оператора If совместную проверку условий можно записать так:

С помощью операции And:

```
If Условие1 And Условие2 Then
    ... 'Операторы1
[ Else
    ... 'Операторы2 ]
End If
```

С помощью операции Or:

```
If Условие1 Or Условие2 Then
    ... 'Операторы1
[ Else
    ... 'Операторы2 ]
End If
```

Запись составных условий с применением линейного оператора If:

If *Условие1* And *Условие2* Then *Операторы1* [Else *Операторы2*]

If *Условие1* Or *Условие2* Then *Операторы1* [Else *Операторы2*]

Использование подобной записи позволяет сократить число строк программного кода, но иногда может привести к ухудшению восприятия программы.

Объединять с помощью логических операций можно только условия в операторе If, но не последовательное выполнение операторов (в этом случае они разделяются двоеточием – см. [пункт 2.3.3](#)), например:

If x > 3 And x < 5 Then z = x ^ 2 : s = 0	'Правильная запись
If x > 3 And x < 5 Then z = x ^ 2 And s = 0	'Неправильная запись

Пример У3 (Объединение условий связкой And). Дано произвольное число X. Если оно положительное и его дробная часть больше 0,5, то возвести его в квадрат, в противном случае изменить знак этого числа.

ГСА решения [примера У3](#) и листинг программы с подробными комментариями приведены на [рис. 7.4](#). Проверяемые условия (блоки 5 и 6 в ГСА) реализованы в программе одним оператором If с помощью логической операции And.

Для вычисления дробной части числа X в программировании на VBA в общем случае применяется выражение Abs(X – Fix(x)).

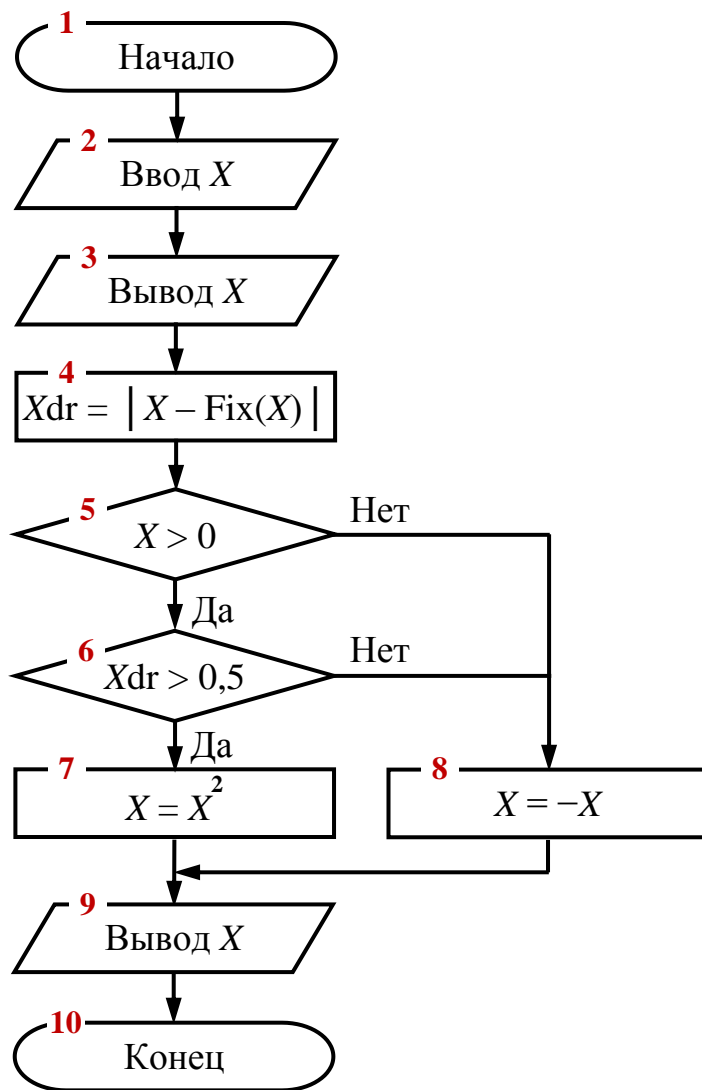
Пример У4 (Объединение условий связкой Or). С помощью генератора случайных чисел сформировать целые числа W и Z в диапазоне от 20 до 80. Если число W окажется больше 50 или число Z будет кратно 6, то вычислить модуль разности этих чисел, в противном случае оба числа обнулить.

ГСА решения [примера У4](#) приведена на [рис. 7.5, а](#).

Данная задача предусматривает проверку кратности чисел. В программировании на VBA она реализуется следующим образом. Число N кратно числу K, если выполняется любое из следующих условий:

$$\begin{aligned} N \bmod K &= 0, \\ N / K &= \text{Fix}(N / K), \\ N / K &= N \backslash K. \end{aligned}$$

Листинг программы решения [примера У4](#), составленной в соответствии с ГСА (см. [рис. 7.5, а](#)), с подробными комментариями приведен на [рис. 7.5, б](#). Проверяемые условия (блоки 4 и 5 в ГСА) реализованы в программе одним оператором If с помощью логической операции Or.



а

```

Sub Пример_У3()
'Объединение условий с помощью логической операции And
'Объявление переменных
Dim X As Single      'X – число
Dim Xdr As Single     'Xdr – дробная часть X

X = Val(InputBox("Введите X"))      'Ввод значения X
Debug.Print "Исходное число X="; X   'Вывод исходного X

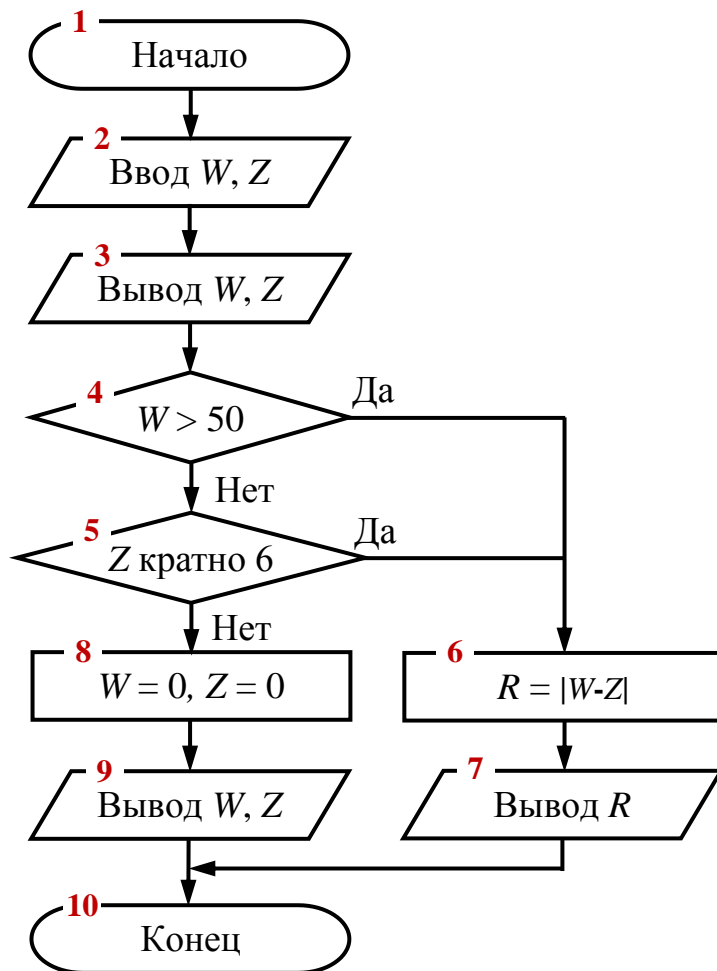
Xdr = Abs(X - Fix(X))  'Вычисление дробной части X

If X > 0 And Xdr > 0.5 Then 'Проверка условий
    'При одновременном выполнении условий
    X = X ^ 2                'вычисление квадрата X
Else
    'При невыполнении хотя бы одного условия
    X = -X                  'изменение знака X
End If                      'Завершение If

'Вывод результирующего значения X в окно отладки
Debug.Print "Результат X="; X
End Sub
  
```

б

Рис. 7.4. Решение примера У3: ГСА (а) и листинг программы (б)



а

Sub Пример_У4()

'Объединение условий с помощью логической операции Or

'Объявление переменных

'W, R – исходные числа, R – модуль разности

Dim W As Integer, Z As Integer, R As Integer

W = Fix(Rnd * (81 - 20) + 20)

'Генерирование значения W

Z = Fix(Rnd * (81 - 20) + 20)

'Генерирование значения Z

Debug.Print "Исходное число W ="; W

'Вывод значения W

Debug.Print "Исходное число Z ="; Z

'Вывод значения Z

If W > 50 Or Z Mod 6 = 0 Then 'Проверка условий

'При выполнении хотя бы одного условия

'вычисление R и вывод результата в окно отладки

R = Abs(W - Z) : Debug.Print "Abs(W - Z) ="; R

Else

'При одновременном невыполнении условий

'обнуление значений W и Z

W = 0 : Z = 0

'вывод результирующих значений W и Z в окно отладки

Debug.Print "Результат W ="; W, "Результат Z ="; Z

End If

'Завершение If

End Sub

б

Рис. 7.5. Решение примера У4: ГСА (а) и листинг программы (б)


7.3. Оператор безусловного перехода GoTo

Оператор GoTo используется для передачи управления в определенное место программы без проверки каких-либо условий и имеет следующий формат:

GoTo Метка

Метка обозначает место в программе, куда осуществляется переход. Имя метки выбирается по обычным правилам для имен переменных и оканчивается двоеточием. Метка может находиться как до оператора GoTo, так и после него.

Например:



```
GoTo m1      'Переход на строку с меткой m1
.....
m1:  y = cos(x)
```

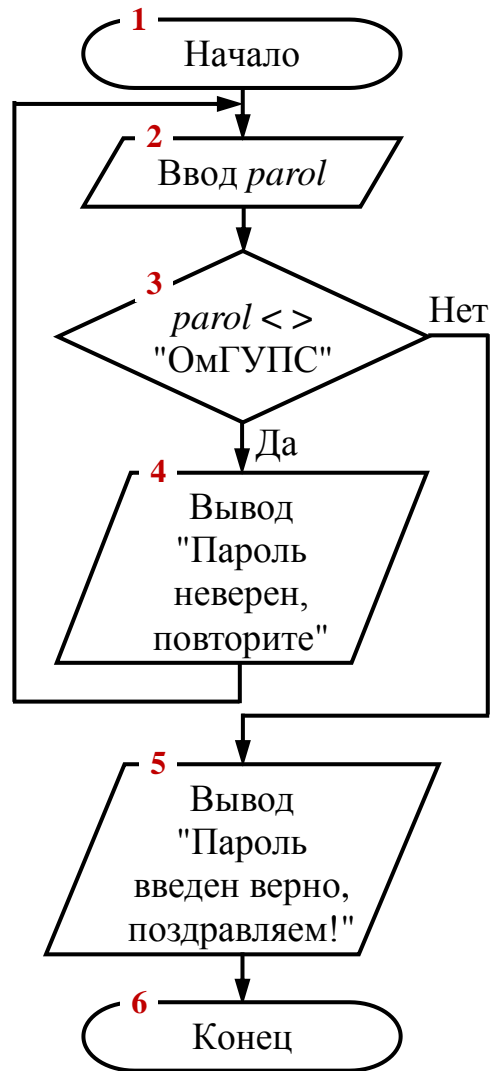
В ГСА оператор GoTo отображается просто линией потока.

В следующем примере рассмотрено применение в одной программе операторов условного (If) и безусловного (GoTo) переходов.

Пример У5 (Безусловный переход). Составить программу запроса и проверки пароля. Если введенный пароль не совпадает с заданным (например, «ОмГУПС»), то повторить его ввод.

ГСА решения [примера У5](#) и листинг программы с подробными комментариями приведены на [рис. 7.6](#).

Несмотря на то, что оператор GoTo иногда очень удобен, использовать его в больших программах не рекомендуется, так как они становятся плохо читаемыми. Обычно вместо GoTo используют циклические конструкции While ... Wend или Do While ... Loop, которые рассмотрены в [подразд. 8.4](#).



а

Sub Пример_У5()

'Оператор безусловного. перехода GoTo

Dim parol As String 'Объявление строковой переменной parol

vr: parol = InputBox("Введите пароль") 'Ввод пароля (vr – метка строки)

If parol <> "ОмГУПС" Then 'Проверка пароля

'При несовпадении введенного и проверяемого паролей

'вывод сообщения в диалоговое окно

MsgBox "Пароль неверен, повторите"

GoTo vr 'переход к метке vr (возврат на строку ввода пароля)

End If 'Завершение If

'При правильном вводе пароля вывод сообщения в диалоговое окно

MsgBox "Пароль введен верно, поздравляем!"

End Sub

б

Рис. 7.6. Решение примера У5: ГСА (а) и листинг программы (б)

7.4. Оператор выбора Select Case

Если нужно многократно проверить одно и то же выражение и сравнить его с разными значениями, то оператор If может стать слишком сложным. В этом случае идеально подходит оператор выбора Select Case. Он имеет вид:

```
Select Case Проверяемое_выражение
  Case Список1
    Операторы1 'Операторы, которые выполняются, если
                'Проверяемое_выражение соответствует Списку1
  Case Список2
    Операторы2 'Операторы, которые выполняются, если
                'Проверяемое_выражение соответствует Списку2
  Case Список n...
    ОператорыN 'Операторы, которые выполняются, если
                'Проверяемое_выражение соответствует СпискуN
[ Case Else
  ОператорыElse ] 'Операторы, которые выполняются, если
                  'Проверяемое_выражение не соответствует
                  'ни одному из списков, перечисленных ранее
End Select
```

Проверяемое_выражение, Список1, Список2, ... могут содержать имена переменных, числовые или строковые выражения.

При выполнении оператора Select Case вычисляется значение проверяемого выражения. Затем по порядку сверху вниз проверяются все Case, т. е. значение проверяемого выражения сравнивается со значениями, записанными в списке условий Case. После того как нужный Case будет найден, выполняются его операторы и сразу осуществляется выход за End Select. Реализация случая Case Else не обязательна в структуре Select Case, но рекомендуется ее предусматривать для обработки значений, отсутствующих в списках всех других Case.

В том случае, когда при нескольких значениях проверяемого выражения необходимо выполнять один и тот же перечень действий, список этих значений можно указать после ключевого слова Case, разделяя их запятыми. Например:

```
Case 3, 5, 6, 9 To 12, Is > 16
```

Такая запись проверяет, равняется ли значение выражения одному из чисел: 3, 5, 6, или оно находится в диапазоне от 9 до 12, или оно больше 16. Ключевое слово `Is` используется для обозначения проверяемого выражения, указанного в заголовке оператора `Select Case`. При вводе условия слово `Is` можно не набирать, редактор VBA добавит его автоматически.

После ключевого слова `Case` допустимо использовать составные условия подобно тому, как это делается в условном операторе `If`.

Пример У6 (Множественный выбор). Составить программу для перевода количества информации, заданного в битах, в более крупные единицы измерения.

Для решения задачи введем следующие обозначения:

kbit – количество информации в битах;

edizm – код единицы измерения результата: 1-байт, 2-КБ, 3-МБ, 4-ГБ;

kp – коэффициент перевода из битов в другие единицы измерения;

txt – текстовое сообщение.

ГСА решения **примера У6** приведена на [рис. 7.7](#). В блоках 2 и 3 вводятся исходные данные *kbit* и *edizm*. Затем выполняется анализ введенного значения *edizm*, при котором последовательно проверяется каждое условие (блоки 4, 6, 8, 10). Как только встретится истинное, будет выполнен соответствующий набор действий (блоки 5, 7, 9, 11): определение коэффициента *kp* для выбранной единицы измерения и формирование соответствующего сообщения *txt*. После их выполнения осуществляется вывод результатов расчета (блок 13). Если ни одно из условий не окажется истинным, будет выведено соответствующее сообщение (блок 12) и произведен возврат к блоку 2 для повторного ввода исходных данных.

Листинг программы решения **примера У6**, составленной в соответствии с ГСА (см. [рис. 7.7](#)), с подробными комментариями представлен на [рис. 7.8](#).

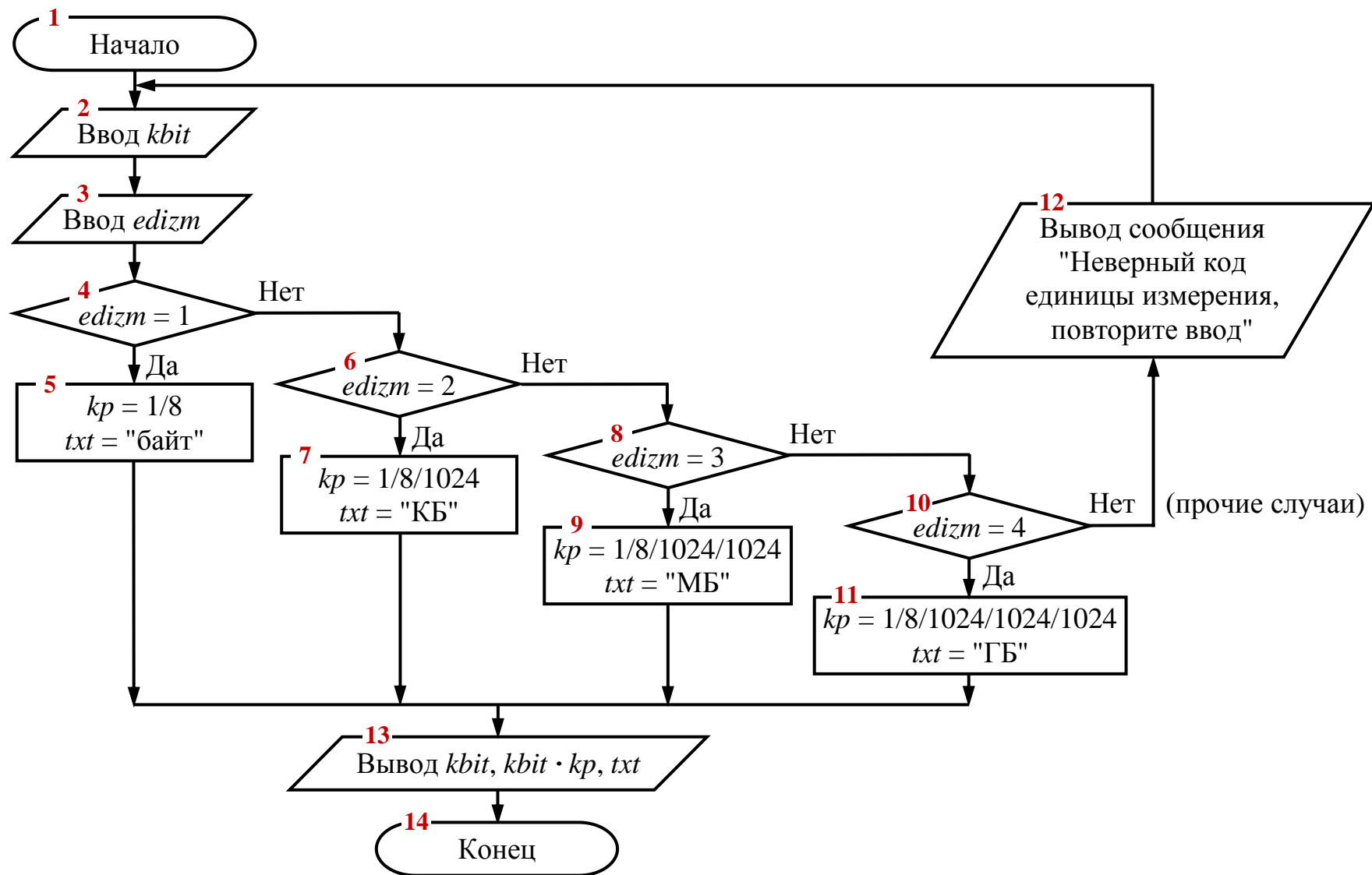


Рис. 7.7. ГСА решения примера У6

```

Sub Пример_У6()
'Перевод единиц измерения информации (оператор Select Case)
'Объявление переменных
Dim kbit As Single      'Количество информации в битах
Dim edizm As Byte       'Код единицы измерения информации
Dim kp As Single        'Коэффициент перевода
Dim txt As String       'Текстовое сообщение

vvod:      'Ввод исходных данных в диалоговое окно (vvod - метка строки)
kbit = Val(InputBox("Введите количество информации в битах"))
edizm = Val(InputBox("В какие единицы перевести? " + _
    Chr(13) + "(1-байт, 2-КБ, 3-МБ, 4-ГБ)"))

Select Case edizm      'Анализ введенного значения edizm
Case 1                  'Проверка edizm = 1
    'определение коэффициента перевода в байты
    'и формирование текстового сообщения
    kp = 1 / 8 : txt = "байт"
Case 2                  'Проверка edizm = 2
    'определение коэффициента перевода в КБ
    'и формирование текстового сообщения
    kp = 1 / 8 / 1024 : txt = "КБ"
Case 3                  'Проверка edizm = 3
    'определение коэффициента перевода в МБ
    'и формирование текстового сообщения
    kp = 1 / 8 / 1024 / 1024 : txt = "МБ"
Case 4                  'Проверка edizm = 4
    'определение коэффициента перевода в ГБ
    'и формирование текстового сообщения
    kp = 1 / 8 / 1024 / 1024 / 1024 : txt = "ГБ"
Case Else               'Все прочие случаи
    MsgBox "Неверный код единицы измерения, повторите ввод"
    GoTo vvod           'Возврат на ввод kbit и edizm
End Select             'Завершение Select Case

'Вывод результата в окно отладки: исходного количества информации в битах
'и в заданных единицах измерения
Debug.Print kbit; "бит ="; kbit * kp; txt
End Sub

```

Рис. 7.8. Листинг программы решения примера У6

8. ЦИКЛИЧЕСКИЕ ВЫЧИСЛИТЕЛЬНЫЕ ПРОЦЕССЫ

8.1. Понятие цикла

Часто при решении задач приходится многократно выполнять вычисления по одним и тем же формулам с разными исходными данными. Например, нужно составить программу расчета значений функции $y = \sin x$ при $x = 0; 0,1; 0,2; \dots; 1$. Для определения всех значений функции y необходимо вычислять и выводить 11 раз значение $\sin x$, начиная с $x = 0$ и увеличивая каждый раз аргумент x на 0,1. Данный пример является примером *табулирования* функции – определения значений функции при изменении ее аргумента от начального до конечного значения с определенным шагом (приращением). При решении подобных задач целесообразно использовать циклический алгоритм, реализуемый с помощью специальных операторов циклов.

Цикл – это конструкция в программировании, позволяющая неоднократно выполнять одну и ту же последовательность операторов внутри программы.

Циклические алгоритмы делятся на арифметические и итерационные.

Цикл называется *арифметическим*, если количество его повторений заранее известно или может быть легко вычислено (другие названия арифметического цикла – цикл с известным числом повторений, регулярный, счетный, цикл с параметром).

Цикл называется *итерационным*, если число его повторений заранее неизвестно, например, он заканчивается при выполнении некоторого условия.

Любой цикл содержит в себе элементы разветвления, поскольку в цикле всегда нужно проверять условие его окончания.

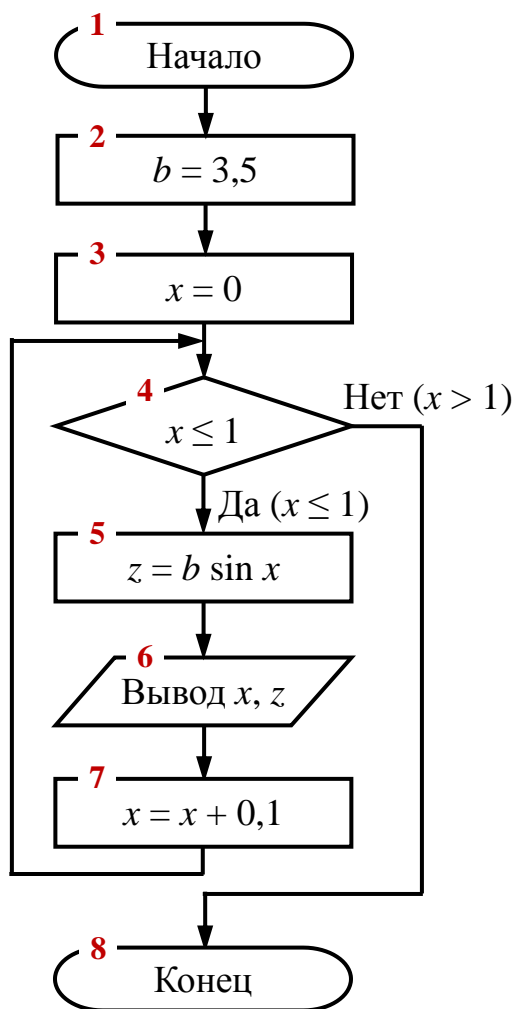
8.2. Арифметический цикл

Арифметический цикл предназначен для организации выполнения последовательности операторов заданное число раз. Рассмотрим реализацию арифметического цикла на простом примере.

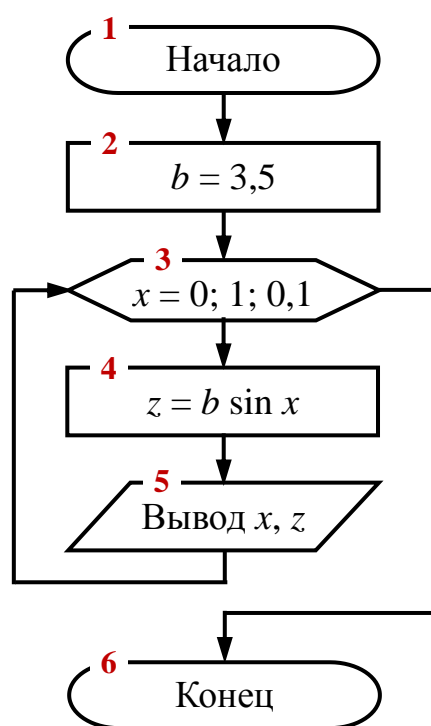
Пример Ц1 (Арифметический цикл). Протабулировать функцию $z = b \sin x$ в диапазоне изменения аргумента x от 0 до 1 с шагом $\Delta x = 0,1$, если $b = 3,5$.

Составим ГСА решения **примера Ц1** (рис. 8.1, а) на основе знаний, полученных при изучении темы «Разветвляющиеся вычислительные процессы»

(см. [разд. 7](#)). В блоке 2 задается значение константы ($b = 3,5$), в блоке 3 – начальное значение аргумента ($x = 0$), в блоке 4 выполняется сравнение текущего значения аргумента с конечным ($x \leq 1$). В случае истинности условия вычисляется z (блок 5) и выводятся текущие значения x и z (блок 6). В блоке 7 значение аргумента x наращивается на величину Δx , выполняется возврат к блоку 4, и весь процесс повторяется до тех пор, пока текущее значение x не превысит конечное (т. е. условие в блоке 4 перестанет выполняться).



а



б

Рис. 8.1. ГСА решения [примера Ц1](#): в полной форме с применением условного блока (а) и в краткой форме с помощью блока подготовки (б)

Представленная в полной форме ГСА (см. [рис. 8.1, а](#)) отображает логику циклического процесса, который можно реализовать, используя операторы условного перехода `If` и безусловного перехода `GoTo`. Однако в программировании для реализации арифметического цикла предусмотрены более эффективные

инструменты. В ГСА такой цикл обычно изображают в компактном виде (рис. 8.1, б) с использованием символа *подготовки* (блок 3), в котором описывают закон изменения (модификации) параметра цикла (в данном примере x), указывая его начальное и конечное значения и шаг изменения. Таким образом, блок 3, реализующий цикл в краткой форме ГСА, объединяет в себе блоки 3, 4 и 7 полной формы ГСА.

Для организации арифметического цикла в VBA предусмотрен оператор For ... Next, который имеет следующий формат записи:

For $x = x_{\text{нач}}$ To $x_{\text{кон}}$ [Step Δx]	'Заголовок (начало) цикла с параметром x
...	'Операторы (тело цикла)
Next x	'Возврат к началу цикла

Заголовок цикла задает перебор значений параметра цикла x от начального значения $x_{\text{нач}}$ до конечного значения $x_{\text{кон}}$ с шагом Δx . Значения $x_{\text{нач}}$, $x_{\text{кон}}$, Δx могут быть константами, переменными или арифметическими выражениями. Если шаг Δx равен 1 или -1 , то ключевое слово Step и параметр Δx можно опустить.

При обработке дробных чисел может накапливаться погрешность вычислений, которая приводит к тому, что цикл завершится до достижения последнего значения параметра цикла. Для исключения такой ситуации рекомендуется увеличить конечное значение параметра цикла, например, на десятую долю шага, т. е. вместо $x_{\text{кон}}$ принимать значение $x_{\text{кон}} + \Delta x/10$.

В теле цикла записывается перечень действий, которые повторяются для каждого значения параметра цикла. Этот перечень может включать в себя фрагменты линейной структуры, разветвления и вложенные циклы. После завершения циклического процесса управление передается оператору, следующему за Next.

Простейший вариант программы решения [примера Ц1](#), составленной в соответствии с ГСА (см. [рис. 8.1, б](#)), с использованием оператора For...Next приведен на [рис. 8.2](#). При этом в заголовке цикла вместо исходного конечного значения параметра цикла, равного 1, принято значение 1,01. В противном случае значение функции при $x = 1$ не будет получено за счет погрешности вычислений.

Обычно для наглядного представления результатов циклического процесса и дальнейшей работы с ними (сортировки, выборки, построения диаграмм и др.) их вывод осуществляют на лист Excel. На [рис. 8.3](#) приведена соответствующая программа решения [примера Ц1](#), в которой вывод результатов расчета выполнен на рабочий лист Excel. Результат работы программы приведен на [рис. 8.4](#).

```

Sub Пример_Ц1()
'Арифметический цикл (табуляция функции)
'Объявление переменных
'x – аргумент функции (параметр цикла), z – функция
Dim x As Single, z As Single

Const b = 3.5 'Объявление константы b

For x = 0 To 1.01 Step 0.1 'Заголовок цикла с параметром x
    z = b * Sin(x) 'вычисление значения z
    Debug.Print "x ="; x, "z ="; z 'вывод значений x, z в окно отладки
Next x 'Возврат к заголовку цикла
End Sub

```

Рис. 8.2. Листинг программы решения примера Ц1

```

Sub Пример_Ц1_2способ()
'Арифметический цикл (табуляция функции на лист Excel)
'Объявление переменных
'x – аргумент функции (параметр цикла), z – функция
'nstr - счетчик номеров строк на листе Excel
Dim x As Single, z As Single, nstr As Byte

Const b = 3.5 'Объявление константы b

nstr = 1 'Установка вывода с 1-й строки
Cells(nstr, 1) = " x" 'Вывод заголовка 1-го столбца
Cells(nstr, 2) = " z" 'Вывод заголовка 2-го столбца

'Заголовок цикла с параметром x,
'с учетом погрешности вывода конечного значения
For x = 0 To 1.01 Step 0.1
    z = b * Sin(x) 'вычисление значения z
    'Вывод текущих значений расчета
    nstr = nstr + 1 'наращивание номера строки
    Cells(nstr, 1) = x 'вывод x в 1-й столбец
    'Вывод z во 2-й столбец в формате
    'с двумя десятичными знаками
    Cells(nstr, 2).NumberFormat = "0.00"
    Cells(nstr, 2) = z
Next x 'Возврат к заголовку цикла
End Sub

```

Рис. 8.3. Листинг программы решения примера Ц1 с выводом данных на лист Excel

	A	B
1	x	z
2	0	0,00
3	0,1	0,35
4	0,2	0,70
5	0,3	1,03
6	0,4	1,36
7	0,5	1,68
8	0,6	1,98
9	0,7	2,25
10	0,8	2,51
11	0,9	2,74
12	1	2,95

Рис. 8.4. Результат работы программы решения примера Ц1 с выводом данных на лист Excel

Таким образом, при составлении ГСА и программ циклических процессов нужно уметь выделять:

- параметр цикла;
- начальное значение параметра цикла;
- конечное значение параметра цикла или условие окончания цикла;
- закон изменения (приращение) параметра цикла;
- тело цикла.

При программировании арифметических циклов рекомендуется придерживаться следующих правил:

параметр цикла в инструкциях For и Next должен быть одинаковым (или после Next параметр цикла можно не указывать);

не рекомендуется изменять внутри цикла значение параметра цикла или пределы его изменения;

запрещается вход в цикл, минуя оператор For, т. е. недопустима передача управления извне на операторы, составляющие тело цикла. Это приведет к ошибке Next without For (Next без For);

при выходе из цикла сохраняется текущее значение параметра цикла, которое обычно на один шаг отличается от конечного значения;

допускается выход из цикла в любое время с помощью оператора GoTo или Exit For (см. [подразд. 9.5](#)).

Цикл не выполняется в случаях, если:

$x_{\text{нач}} > x_{\text{кон}}$ и $\Delta x \geq 0$;

$x_{\text{нач}} < x_{\text{кон}}$ и $\Delta x < 0$.

При $x_{\text{нач}} \leq x_{\text{кон}}$ и $\Delta x = 0$ цикл будет выполняться бесконечно долго (произойдет заикливание).

Если $x_{\text{нач}} = x_{\text{кон}}$, то при любом отличном от нуля значении Δx цикл выполнится один раз.

8.2.1. Цикл со счетчиком

Если в повторяющихся вычислениях аргумент функции изменяется не с постоянным шагом, а произвольным образом, но количество расчетов заранее известно, то организуют цикл со счетчиком. В этом случае в качестве параметра цикла используют переменную-счетчик, которая определяет количество повторений цикла. Когда счетчик превысит конечное значение, цикл завершится.

Пример Ц2 (Цикл со счетчиком). Вычислить функцию $z = b \sin x$ для пяти произвольных значений x , если $b = 3,5$.

В отличие от решения **примера Ц1** в этой задаче в переменную x поочередно вводятся пять чисел. Поскольку эти числа произвольные и изменяются с неизвестным шагом, то использовать переменную x в качестве параметра арифметического цикла невозможно. Поэтому параметром цикла в данном случае определен счетчик k – порядковый номер вводимого числа.

ГСА решения **примера Ц2** приведена на **рис. 8.5, а**. Она отличается от ГСА решения **примера Ц1** (см. **рис. 8.1, б**) тем, что в блоке 3 параметром цикла является счетчик k (его значения изменяются от 1 до 5 с шагом 1), а в тело цикла дополнительно введен блок 4 для ввода значения переменной x . После расчета соответствующего значения z (блок 5) выполняются вывод текущих значений k, x, z (блок 6) и возврат к блоку 3 для вычисления следующего значения k . Когда k превысит конечное значение 5, цикл завершится выходом из блока 3.

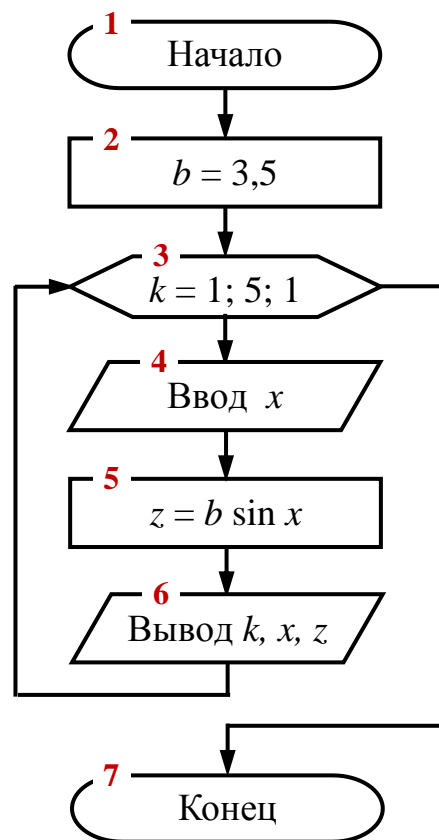
Листинг программы решения **примера Ц2**, составленной в соответствии с ГСА (см. **рис. 8.5, а**), с подробными комментариями приведен на **рис. 8.5, б**.

8.2.2. Вычисление максимума и минимума

Среди разных типов вычислительных задач особое место занимает определение экстремума – максимального и (или) минимального значения функции на рассматриваемом интервале изменения ее аргумента. Решение такой задачи основано на последовательном сравнении каждого значения функции с текущим значением экстремума, которое при выполнении соответствующего условия становится равным проверяемому значению функции. В программировании для осуществления таких действий организуют цикл, в который вложена разветвляющаяся структура, реализующая вычисление экстремума.

Пример Ц3 (Поиск максимума и минимума). Вычислить максимальное и минимальное значения функции $z = b \sin^2 x^3$ в диапазоне изменения аргумента x от 2 до 5 с шагом $\Delta x = 0,2$, если $b = 0,13$.

Вычисление значений z в этой задаче аналогично решению **примера Ц1**. Дополнительно вместе с расчетом функции z выполняется вычисление экстремумов функции (ее максимального и минимального значений). ГСА решения **примера Ц3** приведена на **рис. 8.6**.



а

```

Sub Пример_Ц2()
'Арифметический цикл со счетчиком
'Объявление переменных
'k – счетчик чисел (параметр цикла), x – аргумент функции, z – функция
Dim k As Byte, x As Single, z As Single

Const b = 3.5 'Объявление константы b

For k = 1 To 5 Step 1 'Заголовок цикла с параметром k
    x = Val(TextBox("Введите x")) 'ввод значения x в диалоговое окно
    z = b * Sin(x) 'вычисление значения z
    Debug.Print "k="; k, "x="; x, "z="; z 'вывод значений k, x, z в окно отладки
Next k 'Возврат к заголовку цикла
End Sub
  
```

б

Рис. 8.5. Решение примера Ц2: ГСА (а) и листинг программы (б)

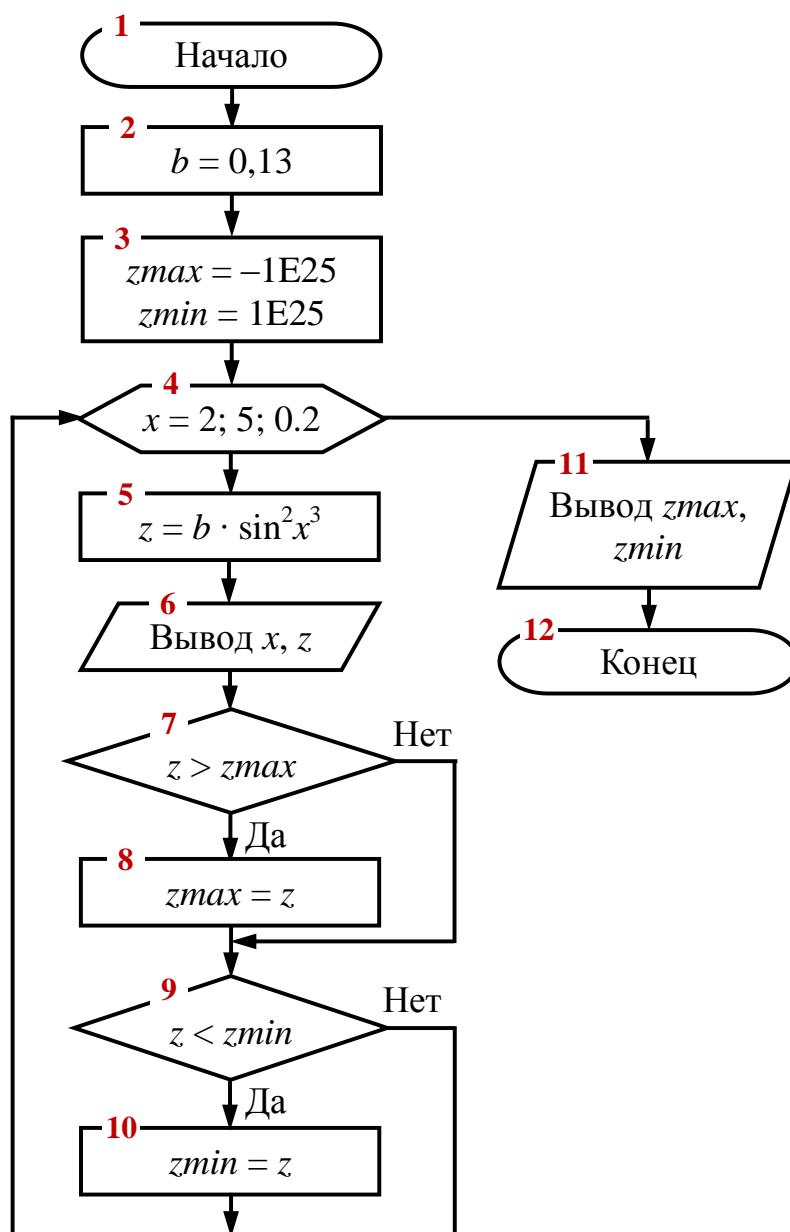


Рис. 8.6. ГСА решения примера ЦЗ

Процесс поиска значений экстремумов функции z заключается в следующем. Предусматриваются отдельные переменные (блок 3):

для хранения максимума – z_{max} , в качестве ее начального значения принимается любое число, которое заведомо меньше хотя бы одного из анализируемых значений функции (в идеальном случае это должно быть минимально возможное из области допустимых значений z). Условно примем в качестве начального значения z_{max} очень малое число, например, $-1E25$;

для хранения минимума – z_{min} , в качестве ее начального значения принимается любое число, которое заведомо больше хотя бы одного из анализируемых значений функции (в идеальном случае это должно быть максимально

возможное из области допустимых значений z). Условно примем в качестве начального значения $zmin$ очень большое число, например, $1E25$.

После расчета и вывода текущего значения z (блоки 5 и 6) выполняется его сравнение со значением $zmax$ (блок 7), и если оно окажется больше, то переменной $zmax$ присваивается это значение z (блок 8). Аналогично каждое текущее значение z сравнивается со значением $zmin$ (блок 9), и если оно окажется меньше, то переменной $zmin$ присваивается это значение z (блок 10). Вывод результата вычисления экстремумов функции z выполняется после выполнения расчетов всех значений z и выхода из цикла (блок 11 в ГСА).

Листинг программы решения [примера Ц3](#), составленной в соответствии с ГСА (см. [рис. 8.6](#)), с подробными комментариями приведен на [рис. 8.7](#).

```
Sub Пример_Ц3()  
'Арифметический цикл вычисления экстремумов функции  
'Объявление переменных  
'x – аргумент функции (параметр цикла), z – функция  
'zmax – максимум функции, zmin – минимум функции  
Dim x As Single, z As Single  
Dim zmax As Single, zmin As Single  
  
Const b As Single = 0.13      'Объявление константы b  
  
zmax = -1E25 : zmin = 1E25    'Начальные значения максимума и минимума  
  
For x = 2 To 5 Step 0.2      'Заголовок цикла с параметром x  
    z = b * Sin(x ^ 3) ^ 2    'вычисление значения z  
    Debug.Print "x ="; x, "z ="; z 'вывод значений x, z в окно отладки  
    If z > zmax Then zmax = z  'вычисление максимального значения z  
    If z < zmin Then zmin = z  'вычисление минимального значения z  
Next x                       'Возврат к заголовку цикла  
  
'Вывод минимального и максимального значений z в окно отладки  
Debug.Print "zmax ="; zmax, "zmin ="; zmin  
End Sub
```

Рис. 8.7. Листинг программы решения [примера Ц3](#)

Таким образом, можно сформулировать общий порядок вычисления максимума и (или) минимума:

- определить отдельную переменную для хранения значения, например:
для максимума – $zmax$, для минимума – $zmin$;
- до цикла задать начальное значение:
для максимума – $zmax = -1E25$, для минимума – $zmin = 1E25$;

- в цикле записать проверку условия и вычисление текущего значения:
 для максимума `If z > zmax Then zmax = z;`
 для минимума `If z < zmin Then zmin = z;`
- после цикла вывести результирующее значение экстремума.

8.2.3. Цикл с разветвлением

При решении практических задач имеются случаи, когда на рассматриваемом интервале изменения аргумента функции в зависимости от выполнения накладываемых на него ограничений значения функции требуется определять по разным формулам, т. е. выполнять табулирование разрывной функции. В программировании решение подобных задач осуществляется с помощью организации цикла, в который вложена разветвляющаяся структура.

Пример Ц4 (Цикл с разветвлением). Протабулировать разрывную функцию $z(x)$, изменяя аргумент x с шагом $\Delta x = 0,2$, если $b = -4,5$:

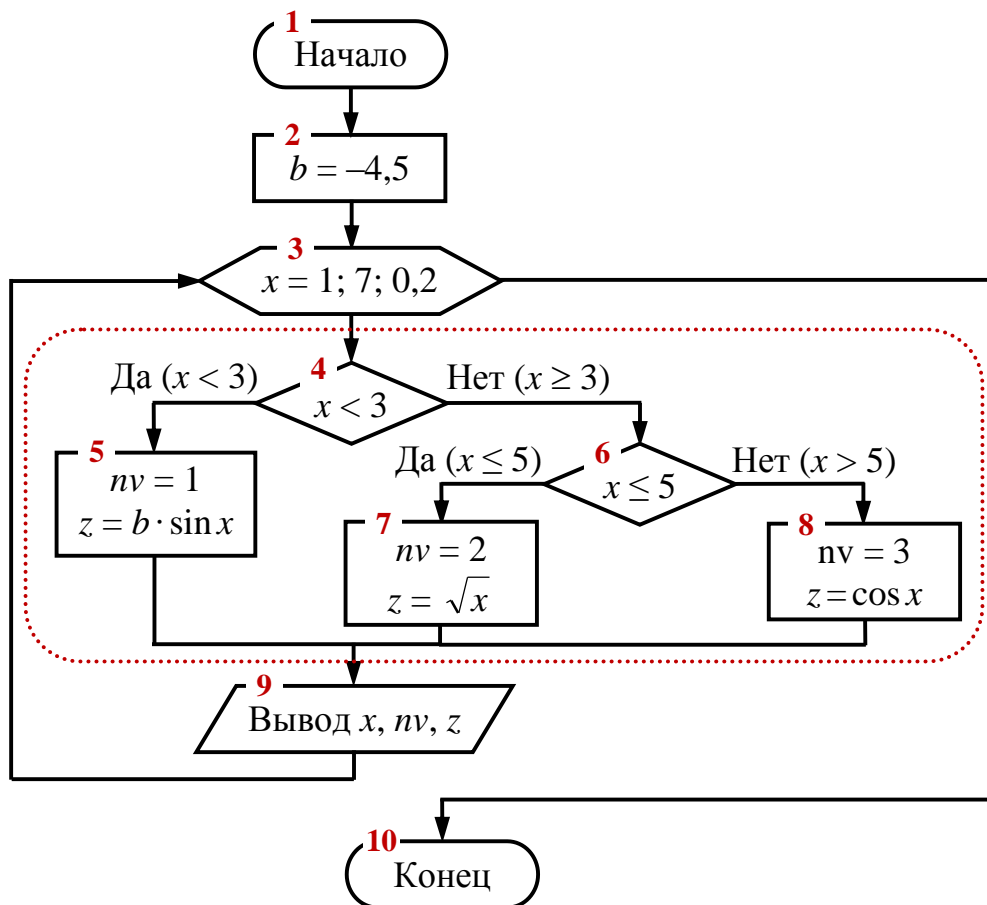
$$z = \begin{cases} b \sin x & \text{при } x < 3; \\ \sqrt{x} & \text{при } 3 \leq x \leq 5; \\ \cos x & \text{при } x > 5. \end{cases}$$

Заданная функция z состоит из трех частей и рассчитывается по разным формулам в зависимости от диапазона изменения аргумента x . Поскольку общий интервал изменения x в задании не указан, определим его исходя из равенства длин всех трех диапазонов.

По условию задачи известны начальное и конечное значения среднего диапазона, что позволяет определить его длину: $|5 - 3| = 2$. Тогда начальное значение общего интервала изменения аргумента x равно $3 - 2 = 1$, а конечное значение общего интервала изменения аргумента x равно $5 + 2 = 7$.

Таким образом, примем общий интервал изменения x от 1 до 7 с шагом $\Delta x = 0,2$, что позволяет решить задачу аналогично [примеру Ц1](#). Отличие заключается в том, что в теле цикла необходимо предусмотреть проверку условий изменения значения x с целью выбора формулы для расчета значения функции z .

ГСА решения [примера Ц4](#) и листинг программы с подробными комментариями приведены на [рис. 8.8](#). Пунктиром в ГСА и программе выделена структура, реализующая разветвление на три ветви, как в [примере У2](#).



а

```

Sub Пример_Ц4()
'Цикл с разветвлением на 3 ветви
'Объявление переменных
'x – аргумент функции (параметр цикла), nv – номер ветви, z – функция
Dim x As Single, nv As Byte, z As Single

Const b As Single = -4.5      'Объявление константы b

For x = 1 To 7 Step 0.2      'Заголовок цикла с параметром x
    If x < 3 Then             'Проверка условия 1
        nv = 1 : z = b * Sin(x) 'вычисление z по первой ветви
    ElseIf x <= 5 Then        'Проверка условия 2
        nv = 2 : z = Sqr(x)    'вычисление z по второй ветви
    Else                      'вычисление z по третьей ветви
        nv = 3 : z = Cos(x)
    End If                   'Завершение If

    Debug.Print "x ="; x, "ветвь"; nv, "z ="; z 'Вывод результата в окно отладки
Next x                       'Возврат к заголовку цикла
End Sub
  
```

б

Рис. 8.8. Решение примера Ц4: ГСА (а) и листинг программы (б)

8.3. Вычисление сумм и произведений

На практике часто приходится решать задачи по вычислению математических выражений вида $\sum_{i=n}^k f_i$, т. е. накапливать сумму, или $\prod_{i=n}^k f_i$ – накапливать произведение.

Нахождение суммы $S = \sum_{i=n}^k f_i$ заключается в циклическом вычислении промежуточных сумм S_i посредством определения очередного слагаемого f_i и добавления его к значению суммы предыдущих слагаемых S_{i-1} . Таким образом, накопление суммы можно представить в виде последовательности однотипных вычислений (для наглядности примем $n = 1$):

$$\begin{aligned} S_1 &= S_0 + f_1 = f_1; \\ S_2 &= S_1 + f_2 = (f_1) + f_2; \\ S_3 &= S_2 + f_3 = (f_1 + f_2) + f_3; \\ &\dots\dots\dots \\ S_k &= S_{k-1} + f_k = \sum_{i=1}^k f_i. \end{aligned} \tag{8.1}$$

Поскольку в памяти компьютера обычно не требуется сохранять значения всех слагаемых и промежуточных сумм, то их можно представить простыми переменными, т. е. хранить в памяти ЭВМ в одних и тех же ячейках. Тогда общая формула для накопления суммы будет иметь вид:

$$S = S + f_i. \tag{8.2}$$

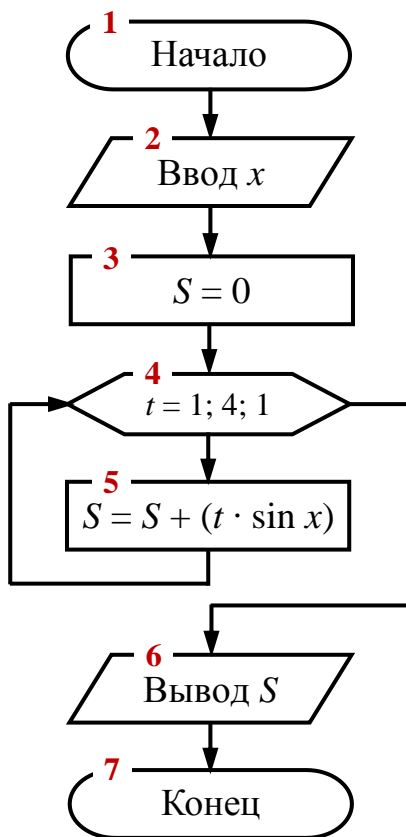
Характерной особенностью формулы (8.2) является то, что и в правой, и в левой ее частях фигурирует одна и та же переменная S . Суть такой формулы заключается в том, что к вычисленному ранее промежуточному значению суммы S (в правой части формулы) прибавляется очередное слагаемое f_i и полученное текущее значение суммы присваивается этой же переменной S (в левой части). Чтобы обобщенную формулу (8.2) можно было использовать и для расчета первого промежуточного значения суммы (равного первому ее слагаемому), следует принять начальное значение суммы S равным нулю.

Пример ЦС1 (Накопление суммы). Вычислить $S = \sum_{t=1}^4 (t \cdot \sin x)$ для произвольного x .

извольного x .

Общая формула накопления суммы для заданного выражения имеет вид: $S = S + (t \cdot \sin x)$. ГСА решения **примера ЦС1** приведена на **рис. 8.9, а**. В блоке 3 в ГСА переменной S присваивается начальное нулевое значение. Затем организуется цикл (блок 4 в ГСА), в котором на каждом шаге к значению переменной S , полученному на предыдущем шаге, добавляется очередное слагаемое $t \cdot \sin x$ (блок 5 в ГСА). После завершения цикла выводится результат накопления S (блок 6 в ГСА).

Листинг программы решения **примера ЦС1**, составленной в соответствии с ГСА (см. **рис. 8.9, а**), с подробными комментариями приведен на **рис. 8.9, б**.



а

```

Sub Пример_ЦС1()
'Арифметический цикл (накопление суммы)
'Объявление переменных
'x – произвольное число, t – параметр цикла, S – сумма
Dim x As Single, t As Byte, S As Single

x = Val(InputBox("Введите x")) 'Ввод значения x

S = 0 'Начальное значение суммы

For t = 1 To 4 'Заголовок цикла с параметром t
    S = S + (t * Sin(x)) 'формула накопления суммы
Next t 'Возврат к заголовку цикла

'Вывод результата накопления суммы в окно отладки
Debug.Print "S ="; S
End Sub
  
```

б

Рис. 8.9. Решение **примера ЦС1**: ГСА (а) и листинг программы (б)

Аналогичным образом вычисляется произведение $P = \prod_{i=n}^k f_i$. Отличие

от расчета суммы состоит лишь в том, что общая формула имеет вид $P = P \cdot f_i$, а начальное значение произведения, которое задается перед циклом, должно быть равно единице.

Пример ЦС2 (Накопление произведения). Вычислить $W = x^2 + \prod_{t=1}^4 (t \cdot \sin x)$

для произвольного x . Вывести результат пошагового накопления произведения.

Обозначим $P = \prod_{t=1}^4 (t \cdot \sin x)$, тогда $W = x^2 + P$. По аналогии с [примером](#)

[ЦС1](#) общая формула накопления произведения имеет вид: $P = P \cdot (t \cdot \sin x)$. ГСА решения [примера ЦС2](#) приведена на [рис. 8.10, а](#). Отличие от [примера ЦС1](#) состоит в том, что начальное значение $P = 1$ (блок 2), на каждом шаге цикла к значению переменной P , полученному на предыдущем шаге, добавляется очередной множитель $t \cdot \sin x$ (блок 5) и выводится текущее промежуточное значение P (блок 6). После цикла вычисляется значение выражения W (блок 7) и выводится его результат (блок 8).

Листинг программы решения [примера ЦС2](#), составленной в соответствии с ГСА (см. [рис. 8.10, а](#)), с подробными комментариями приведен на [рис. 8.10, б](#).

Таким образом, можно сформулировать общий порядок вычисления

суммы $S = \sum_{i=n}^k f_i$ или произведения $P = \prod_{i=n}^k f_i$:

– задать перед циклом начальное значение накапливаемой переменной.

Если не указано иначе, то в качестве него обычно принимают:

при накоплении суммы: $S = 0$;

при накоплении произведения: $P = 1$;

при вычислении количества (счетчик): $C = 0$;

– организовать цикл (For i = n To k);

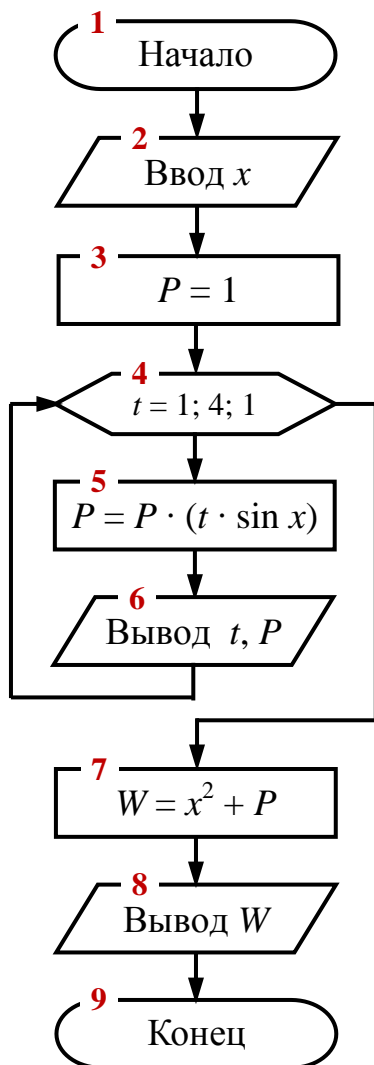
– записать в теле цикла закон изменения переменной (накапливаемая переменная обязательно должна быть и в левой, и в правой частях формулы):

при накоплении суммы: $S = S + f_i$;

при накоплении произведения: $P = P \cdot f_i$;

при вычислении количества: $C = C + 1$;

– вывести после цикла итоговое значение накапливаемой переменной.



а

```

Sub Пример_ЦС2()
'Арифметический цикл (накопление произведения)
'Объявление переменных
'x – произвольное число, t – параметр цикла
'P – произведение, W – результат
Dim x As Single, t As Byte
Dim P As Single, W As Single

x = Val(InputBox("Введите x")) 'Ввод значения x
P = 1 'Начальное значение произведения

For t = 1 To 4 'Заголовок цикла с параметром t
    P = P * (t * Sin(x)) 'формула накопления произведения
    'вывод результата пошагового накопления произведения
    Debug.Print "t="; t, "P="; P
Next z 'Возврат к заголовку цикла

W = x ^ 2 + P 'Вычисление значения W
Debug.Print "W="; W 'Вывод результата W в окно отладки
End Sub
  
```

б

Рис. 8.10. Решение примера ЦС2: ГСА (а) и листинг программы (б)

8.4. Итерационные циклы

Если число повторений цикла заранее не известно и решение о его завершении принимается на основе анализа заданного условия, то такой повторяющийся вычислительный процесс называется *итерационным циклом*.

Итерационные циклы организуются с помощью конструкции While ... Wend или Do ... Loop.

8.4.1. Оператор цикла While ... Wend

Наиболее простым вариантом реализации итерационного цикла является использование конструкции While ... Wend:

While *Условие*

... 'Операторы, которые выполняются, пока *Условие* истинно (тело цикла)

Wend

Когда *Условие*, записанное справа от ключевого слова While, перестанет выполняться, осуществляется выход из цикла, т. е. переход к оператору, следующему за ключевым словом Wend.

В ГСА итерационный цикл изображается только в полной форме.

Пример ЦИ1 (Цикл с предусловием). Найти наименьшее положительное целое трехзначное число W , при котором значение выражения $3 \cdot W - 350$ становится неотрицательным числом.

ГСА решения **примера ЦИ1** приведена на [рис. 8.11, а](#). Так как в задаче требуется найти трехзначное число, то начальное значение W принимается равным 100 (блок 2). Пока значение выражения $3 \cdot W - 350$ отрицательно (т. е. меньше нуля, блок 3), выводится результат вычисления выражения (блок 4) и определяется следующее число W (блок 5). Действия повторяются до тех пор, пока значение выражения $3 \cdot W - 350$ станет неотрицательным (т. е. больше нуля или равно нулю), а значит, искомое число W найдено (блок 6).

Листинг программы решения **примера ЦИ1**, составленной в соответствии с ГСА (см. [рис. 8.11, а](#)), с подробными комментариями приведен на [рис. 8.11, б](#).

Пример ЦИ2 (Накопление суммы в итерационном цикле). Через сколько месяцев удвоится сумма на вкладе в банке, если каждый месяц банк начисляет 1 % на текущий остаток вклада?

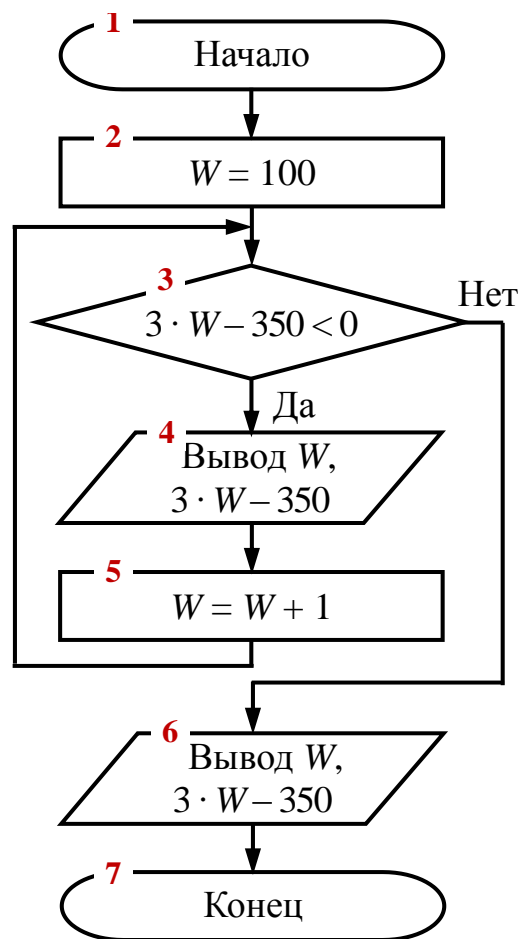
Итоговая сумма определяется по формуле:

$$S_{\text{итог}} = \sum_{m=1}^n (S_m + D_m), \quad (8.3)$$

где S_m – сумма вклада на конец m -го месяца;

$D_m = 0,01 \cdot S_m$ – доход по вкладу в m -м месяце.

ГСА решения **примера ЦИ2** и листинг программы с подробными комментариями приведены на [рис. 8.12](#).



а

```

Sub Пример_ЦИ1()
'Итерационный цикл с предусловием
'Вычисление наименьшего трехзначного числа, при котором
'значение выражение 3 * W - 350 станет неотрицательным

Dim W As Integer      'Объявление целочисленной переменной W
W = 100                'Начальное значение W

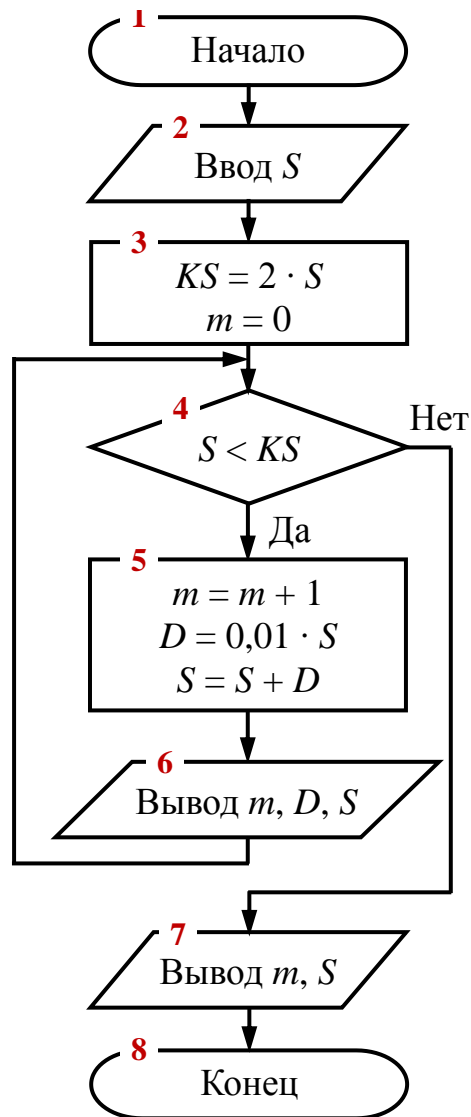
'Заголовок цикла (проверка условия):
'пока значение выражение 3 * W - 350 < 0
While 3 * W - 350 < 0
    'вывод текущего числа W и значения выражения в окно отладки
    Debug.Print "W ="; W, "3 * W - 350 ="; 3 * W - 350

    W = W + 1          'следующее значение W
Wend                  'Возврат к заголовку цикла

'Вывод результата в окно отладки
Debug.Print "Наименьшее трехзначное W ="; W, "3 * W - 350 ="; 3 * W - 350
End Sub
  
```

б

Рис. 8.11. Решение примера ЦИ2: ГСА (а) и листинг программы (б)



а

Sub Пример_ЦИ2()

'Итерационный цикл с предусловием (накопление суммы вклада)

'Объявление переменных

'S – текущая сумма вклада, KS – конечная сумма вклада

'D – текущий доход вклада, m – счетчик месяцев

Dim S As Single, KS As Single, D As Single, m As Integer

S = Val(InputBox("Введите S")) 'Ввод начальной суммы вклада S

KS = 2 * S 'Расчет конечной (удвоенной) суммы вклада

m = 0 'Начальное значение счетчика месяцев

'Заголовок цикла (проверка условия)

'пока текущая сумма вклада S меньше конечной суммы KS

While S < KS

m = m + 1 'наращивание счетчика месяцев

D = 0.01 * S 'расчет текущего дохода

S = S + D 'увеличение (накопление) текущей суммы вклада

'вывод текущих результатов накопления суммы в окно отладки

Debug.Print "m = "; m, "D = "; D, "S = "; S

Wend 'Возврат к заголовку цикла

'Вывод итогового результата в окно отладки

Debug.Print "Итог m = "; m, "S = "; S

End Sub

б

Рис. 8.12. Решение примера ЦИЗ: ГСА (а) и листинг программы (б)

8.4.2. Операторы цикла *Do ... Loop*

Другим способом реализации итерационного цикла является использование оператора *Do ... Loop*, который может быть записан в разных формах.

Основной формат записи:

Do While Условие

... 'Операторы, которые выполняются, пока *Условие* истинно (тело цикла)
Loop

Эта форма полностью аналогична рассмотренному выше оператору *While ... Wend*. В данном случае вместо ключевого слова *While* записывается *Do While*, вместо *Wend* – ключевое слово *Loop*.

Все рассмотренные выше формы записи циклических операторов, включая арифметический цикл, являются *циклами с предусловием*, т. е. условие в них обязательно проверяется до выполнения тела цикла.

Существует и другая форма записи оператора *Do ... Loop*:

Do

... 'Операторы, которые выполняются, пока *Условие* истинно (тело цикла)
Loop While Условие

Приведенная форма записи отличается от предыдущей только тем, что в ней ключевое слово *While* и условие перенесены из заголовка цикла в конец. Такой цикл называется *циклом с постусловием* и отличается от циклов с предусловием тем, что в нем цикл обязательно выполняется хотя бы один раз. ГСА циклов с предусловием и постусловием приведены на [рис. 8.13](#).

Существует еще одна форма записи оператора *Do ... Loop*, в ней вместо слова *While* записывается *Until*. Такая форма применяется значительно реже и предполагает повторение циклического процесса пока *Условие* ложно:

Do Until Условие

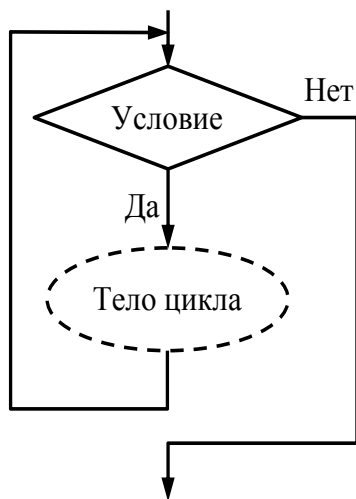
... 'Операторы, которые выполняются, пока *Условие* ложно (тело цикла)
Loop

Если в заголовке цикла *Do While* проверяется какое-либо условие, то для получения такого же результата выполнения цикла в заголовке *Do Until* условие

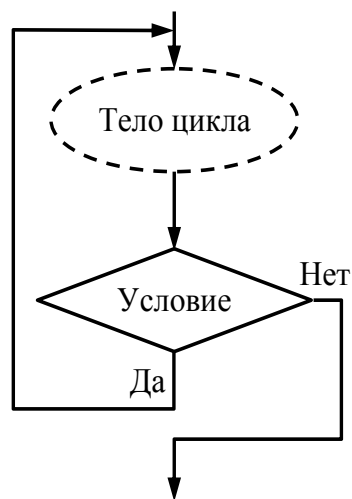
должно быть противоположным. Приведем пример, демонстрирующий различие форм записи оператора Do ... Loop: если нужно выполнять цикл, пока x положительное, то циклы организуются следующим образом:

Do While $x > 0$
... 'Тело цикла
Loop

Do Until $x \leq 0$
... 'Тело цикла
Loop



а



б

Рис. 8.13. ГСА итерационных циклов с предусловием (а) и постусловием (б)

9. МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ.

ПРОЦЕДУРЫ-ФУНКЦИИ И ПРОЦЕДУРЫ-ПОДПРОГРАММЫ

9.1. Понятие процедуры

Все современные языки программирования реализуют модульный, т. е. блочный, принцип программирования. Самыми важными функциональными блоками языка VBA являются процедуры, поскольку в VBA может выполняться только тот программный код, который содержится в какой-либо процедуре.

Процедура – это поименованная группа обособленных операторов, выполняющих нужные действия, не зависящая от других частей программного кода. Процедура позволяет один раз записать последовательность действий, а затем многократно обращаться к ней по имени с разными исходными данными в разных местах программы. Применение процедур уменьшает объем программы,

улучшает ее структурированность. Облегчается отладка программ, так как работу каждой процедуры можно проверить по отдельности. По сути, процедуры – это новые операторы, или операции языка, определенные программистом.

Процедуры делятся на процедуры-функции и процедуры-подпрограммы.

Процедуры-функции (Function) (функции пользователя, нестандартные функции) аналогичны встроенным (стандартным) функциям языка и, как правило, предназначены для вычисления некоторого значения и передачи его в главную программу. Как и стандартные функции, функции пользователя являются частью вычисляемого выражения.

Процедуры-подпрограммы (Subroutine) представляют собой группу операторов, выполняющих определенные действия. В отличие от функций процедуры-подпрограммы могут вычислять не одно значение, а выполнять сразу несколько действий и получать не один, а несколько результатов. В связи с этим процедуры-подпрограммы не могут являться частью математического выражения.

9.2. Формальные и фактические параметры процедур

В операторе объявления любой процедуры после ее имени в скобках указывается список параметров (аргументов процедуры), с которыми связан вычислительный процесс в данной процедуре. Имена параметров в списке перечисляются через запятую, их количество зависит от конкретной процедуры. В тех случаях, когда использование параметров в процедуре не требуется, наличие скобок (пустых) после имени процедуры все равно обязательно. Аргументы, указываемые при объявлении процедуры, являются ее *формальными* параметрами, поскольку в момент объявления процедуры их значения не заданы.

Перед обращением к процедуре ее аргументы должны иметь конкретные значения, при вызове процедуры они называются *фактическими* параметрами. В качестве фактических параметров могут выступать константы, переменные, арифметические выражения. Фактические параметры должны строго соответствовать формальным по количеству, типу данных и порядку следования. Фактические параметры автоматически подставляются в процедуру вместо формальных параметров при обращении к ней в программе.

При использовании процедур рекомендуется в разделе общих объявлений программного модуля обязательно записывать оператор Option Explicit. Это позволит избежать ошибок при передаче параметров в процедуру.

9.3. Разработка процедур-функций

Если в расчетах неоднократно встречается выражение одного и того же вида, целесообразно описать его в отдельной процедуре как пользовательскую функцию и затем применять ее в программе наравне со встроенными функциями (Sin, Cos и др.).

Описание процедуры-функции (ПФ) имеет следующий синтаксис:

```
Function Имя_Функции([a1, a2, ..., an]) [As Тип]
    [Операторы ...]
    Имя_функции = Выражение 'Оператор вычисления значения функции
    [Операторы ...]
End Function
```

где a_1, a_2, \dots, a_n – список формальных параметров (аргументов) ПФ, представляет собой перечень имен переменных, значения которых должны быть переданы ПФ для ее выполнения. Все аргументы обязательно должны использоваться в теле функции, иначе выделять их не имеет смысла.

Тип значения ПФ указывается в ее заголовке после списка аргументов.

В теле процедуры-функции обязательно должен присутствовать оператор присваивания результата вычислений переменной с именем, точно совпадающим с именем функции в заголовке этой процедуры. ПФ возвращает вызвавшей ее программе единственное значение, которое определяется *Выражением*.

В программе обращение к ПФ осуществляется по ее имени:

```
Имя_функции([f1, f2, ..., fn])
```

где f_1, f_2, \dots, f_n – фактические параметры.

В ГСА процедуру-функцию обычно не отражают.

Пример ПФ1. Вычислить:

$$z = \arcsin 0,5 - \arcsin^3 h + \arcsin (b^2 - 1) + \cos b, \quad \text{где } b = 0,3, \quad h = \pi/6.$$

Для решения этого примера нужно использовать математическую формулу

$$\arcsin x = \arctg \frac{x}{\sqrt{1-x^2}}, \quad \text{которая на VBA записывается } \arcsin = \text{Atn}(x / \text{Sqr}(1 - x^2)),$$

и поочередно подставлять в нее вместо x значения 0,5, h и $(b^2 - 1)$. Опишем эту формулу как процедуру-функцию с формальным параметром x , а затем три раза используем ее с разными фактическими параметрами.

Рассматриваемые здесь и далее примеры на применение ПФ имеют простую линейную структуру, ГСА которой аналогична [примеру Л1](#), поэтому на [рис. 9.1](#) представлен только листинг программы решения [примера ПФ1](#) с подробными комментариями.

Пример ПФ2. Вычислить: $z = \frac{\log_2 x + \log_b y}{2 \log_{b+2}(x+y)}$ для произвольных значений x, y, b .

Для решения этого примера нужно использовать универсальную формулу вычисления логарифма: $\log_a p = \frac{\ln p}{\ln a}$, которая на VBA записывается `lra = log(p) / log(a)`, и поочередно подставлять в нее вместо p и a значения $(x, 2)$, (y, b) , $(x+y, b+2)$.

Листинг программы решения [примера ПФ2](#) с подробными комментариями приведен на [рис. 9.2](#).

Любое имя переменной, указанное в списке формальных параметров при определении функции, является *локальным* по отношению к этой функции (эту переменную даже не обязательно объявлять). Это означает, что данная переменная отличается от переменной с тем же именем, указанной в другом месте программы. Данная переменная принимает значение, передаваемое ей во время обращения к функции. Другие переменные, фигурирующие в процедуре-функции при ее определении, но не включенные в состав формальных параметров, ничем не отличаются от обычных переменных программы, и во время обращения к ПФ используются их текущие значения.

Пример ПФ3. Для произвольных значений переменных a, b, c, t вычислить:

$$z = \frac{\overset{\textcircled{1}}{\sqrt{a^2 - b^2 + \sin^2 a + t^2}}}{\underset{\textcircled{2}}{\sqrt{c^2 - a^2 + \sin^3 c + t^2}}} + \overset{\textcircled{3}}{\sqrt{b^2 - a^2 + \sin^4 b + t^2}}.$$

Вид выражений 1, 2 и 3, входящих в заданную формулу, одинаков, за исключением степени синуса. Чтобы записать выражение в общем виде, введем для обозначения степени синуса дополнительную переменную n .

Option Explicit

```
Function arcsin(x) As Single 'Объявление функции arcsin(x)
'Процедура-функция вычисления арксинуса
    arcsin = Atn(x / Sqr(1 - x ^ 2)) 'Вычисление значения arcsin
    'Вывод значений аргумента и функции в окно отладки
    Debug.Print "x ="; x, "arcsin ="; arcsin
End Function
```

```
Sub Пример_ПФ1() 'Основная программа
'Использование процедуры-функции и встроенной функции
'Объявление переменных
Dim h As Single, z As Single

'Объявление констант
Const b As Single = 0.3, pi As Single = 3.14

h = pi / 6 'Вычисление значения h

'Вычисление значения z с использованием
'процедуры-функции arcsin
z = arcsin(0.5) - arcsin(h) ^ 3 + arcsin(b ^ 2 - 1) + Cos(b)

Debug.Print "z ="; z 'Вывод результата в окно отладки
End Sub
```

Рис. 9.1. Листинг программы решения примера ПФ1

Option Explicit

```
Function lpa(p, a) As Single 'Объявление функции lpa(p, a)
'Процедура-функция вычисления логарифма
    lpa = Log(p) / Log(a) 'Вычисление значения lpa
    'Вывод значений аргументов и функции в окно отладки
    Debug.Print "p ="; p, "a ="; a, "lpa ="; lpa
End Function
```

```
Sub Пример_ПФ2() 'Основная программа
'Использование процедуры-функции
'Объявление переменных
Dim x As Single, y As Single, b As Single, z As Single

x = Val(InputBox("Введите x")) 'Ввод значения x
y = Val(InputBox("Введите y")) 'Ввод значения y
b = Val(InputBox("Введите b")) 'Ввод значения b

'Вычисление значения z с использованием
'процедуры-функции lpa
z = (lpa(x, 2) + lpa(y, b)) / (2 * lpa(x + y, b + 2))

Debug.Print "z ="; z 'Вывод результата в окно отладки
End Sub
```

Рис. 9.2. Листинг программы решения примера ПФ2

Переменная t во всех выражениях встречается идентично, поэтому нет необходимости вводить ее в список формальных параметров ПФ. При обращении к ПФ всегда будет использоваться текущее значение t , но чтобы эта переменная была известна всем процедурам, ее следует объявить в разделе объявлений уровня модуля.

Запишем выражения 1, 2, 3 в общем виде как функцию трех аргументов:

$F(x, y, n) = \sqrt{x^2 - y^2 + \sin^n x + t^2}$ (исходя из сказанного выше переменная t не включена в список формальных параметров функции). Проверим эту запись:

1) при $x = a, y = b, n = 2$ $F(a, b, 2) = \sqrt{a^2 - b^2 + \sin^2 a + t^2}$;

2) при $x = c, y = a, n = 3$ $F(c, a, 3) = \sqrt{c^2 - a^2 + \sin^3 c + t^2}$;

3) при $x = b, y = a, n = 4$ $F(b, a, 4) = \sqrt{b^2 - a^2 + \sin^4 b + t^2}$.

Листинг программы решения [примера ПФ3](#) с подробными комментариями приведен на [рис. 9.3](#).

```
Option Explicit
Dim t As Single 'Объявление переменной t в разделе объявлений уровня модуля
'-----
Function F(x, y, n) As Single 'Объявление функции F(x, y, n)
    F = Sqr(x ^ 2 - y ^ 2 + Sin(x) ^ n) + t ^ 2 'Вычисление значения функции
    'Вывод значений аргументов и функции в окно отладки
    Debug.Print "x="; x, "y="; y, "n="; n, "F="; F
End Function
'-----
Sub Пример_ПФ3() 'Основная программа
    'Использование процедуры-функции
    'Объявление переменных
    Dim a As Single, b As Single, c As Single, z As Single
    'Ввод значений переменных
    a = Val(InputBox("Введите a")) : b = Val(InputBox("Введите b"))
    c = Val(InputBox("Введите c")) : t = Val(InputBox("Введите t"))
    'Вычисление значения z с использованием процедуры-функции F(x, y, n)
    z = F(a, b, 2) / F(c, a, 3) + F(b, a, 4)
    Debug.Print "z="; z 'Вывод результата в окно отладки
End Sub
```

Рис. 9.3. Листинг программы решения [примера ПФ3](#)

9.4. Разработка процедур-подпрограмм

При составлении сложных программ часто повторяющиеся действия или фрагменты вычислений выделяют в отдельные подпрограммы, к которым можно многократно обращаться по имени из разных мест программы. Процедура-подпрограмма (ПП) может содержать аргументы, получать входные значения, выполнять любые действия и возвращать любое количество значений в отличие от функций, которые возвращают только одно значение. Подпрограммы обычно используют для получения или обработки входных данных, отображения результатов или установки свойств объектов.

Синтаксис процедуры-подпрограммы следующий:

```
Sub Имя_процедуры([a1, a2, ..., an])  
    ... 'Операторы (тело процедуры)  
End Sub
```

где a_1, a_2, \dots, a_n – список формальных параметров ПП (аналогично ПФ).

Для обращения к ПП из любого места программы служит оператор Call:

```
Call Имя_процедуры([f1, f2, ..., fn])
```


где f_1, f_2, \dots, f_n – фактические параметры ПП (аналогично ПФ).

К подпрограмме можно обратиться другим способом:

```
Имя_процедуры [f1, f2, ..., fn]
```

В этом случае фактические параметры записываются без круглых скобок.

При использовании ПП оператор Option Explicit обязателен. Если переменные подпрограммы объявлены на уровне модуля, то их можно не включать в число параметров ПП.

В ГСА основной программы обращение к подпрограмме изображается символом «предопределенный процесс» – , а ГСА подпрограммы изображается отдельно, как для обычной программы.

Пример ПП1. Вычислить: $u = e^{z_1 + y_1} - e^{z_2 - y_2}$,

где z_1, z_2 – корни квадратного уравнения $5z^2 + 20z - 1,5 = 0$;

y_1, y_2 – корни квадратного уравнения $2y^2 - y - 7,3 = 0$.

Составим подпрограмму *korni* для решения квадратного уравнения в общем виде $ax^2 + bx + c = 0$. В число ее параметров включим коэффициенты a, b, c и корни x_1, x_2 , которые при каждом обращении к подпрограмме будут иметь конкретные значения. ГСА основной программы [примера ПП1](#) и подпрограммы вычисления корней уравнения приведены на [рис. 9.4](#), листинг программного кода с подробными комментариями – на [рис. 9.5](#).

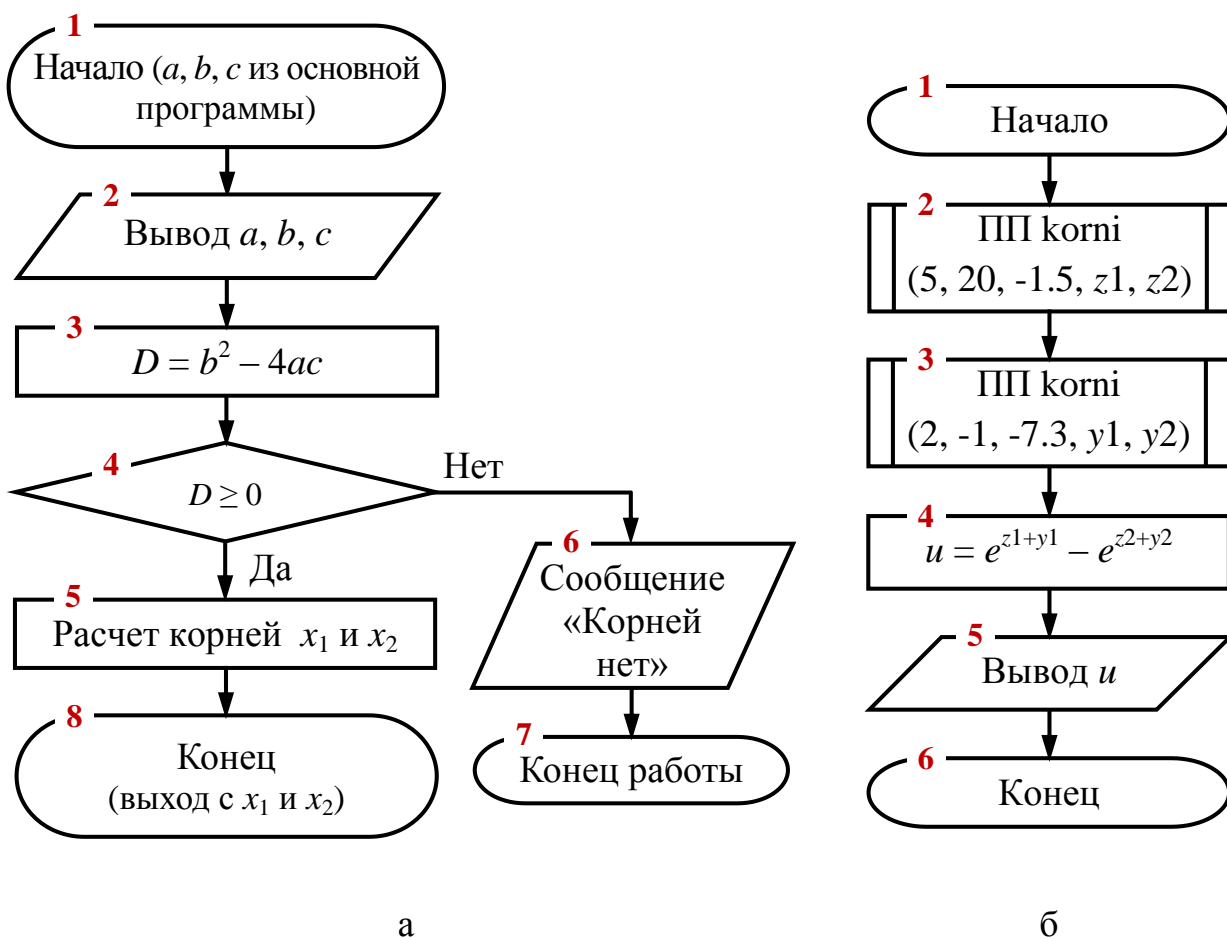


Рис. 9.4. ГСА решения [примера ПП1](#): подпрограммы *korni* (а), основной программы (б)

Следует отметить, что коэффициенты уравнений a, b, c можно было объявить переменными и вводить их значения в подпрограмме, исключив их из числа параметров подпрограммы *korni*.

```

Option Explicit

Sub korni(a, b, c, x1, x2)      'Подпрограмма вычисления корней уравнения
    Dim D As Single          'Объявление переменной D – дискриминант
    'Вывод значений параметров a, b, c в окно отладки
    Debug.Print "При коэффициентах "; a; b; c

    D = b ^ 2 - 4 * a * c     'Вычисление дискриминанта

    If D >= 0 Then            'Проверка дискриминанта
        'вычисление и вывод корней уравнения
        x1 = (-b + Sqr(D)) / (2 * a)
        x2 = (-b - Sqr(D)) / (2 * a)
        Debug.Print "корни x1 ="; x1, "x2 ="; x2
    Else
        'вывод сообщения и завершение работы, если корней нет
        Debug.Print "Корней нет" : End
    End If                    'Завершение If
End Sub

'-----
Sub Пример_ПП1()              'Основная программа
    'Объявление переменных
    Dim z1 As Single, z2 As Single 'z1, z2 – корни 1-го уравнения
    Dim y1 As Single, y2 As Single 'y1, y2 – корни 2-го уравнения
    Dim u As Single              'u – результат вычисления

    Call korni(5, 20, -1.5, z1, z2) 'Обращение к ПП для решения 1-го уравнения
    Call korni(2, -1, -7.3, y1, y2) 'Обращение к ПП для решения 2-го уравнения

    'Вычисление и вывод результата в окно отладки
    u = Exp(z1 + y1) - Exp(z2 - y2)
    Debug.Print "u ="; u
End Sub

```

Рис. 9.5. Листинг программы решения примера ПП1

9.5. Срочный выход из процедур и циклов

Для срочного выхода из модульной или циклической структуры до ее окончания (например, при получении недопустимых значений) служит оператор `Exit <структура>`, который записывается непосредственно в теле соответствующей структуры. В результате его выполнения осуществляется переход к оператору, следующему за последним оператором структуры:

Exit Sub – выход из ПП к оператору, следующему за оператором ее вызова;
Exit Function – выход из ПФ к оператору, следующему за оператором ее вызова;
Exit For – выход из цикла For ... Next к оператору, следующему за Next;
Exit Do – выход из цикла Do ... Loop к оператору, следующему за Loop.

Во многих случаях прервать работу программы, например, при ее заикливании или зависании, можно, нажав клавиши Ctrl + Pause/Break.

10. ПЕРЕМЕННЫЕ С ИНДЕКСАМИ. ОДНОМЕРНЫЕ МАССИВЫ

10.1. Понятие массива

Рассмотрим понятие массива на простом примере.

Пример OM1. Вычислить сумму шести произвольных чисел.

Первый способ. Организуем простой линейный вычислительный процесс. Каждое число обозначим отдельной переменной. Листинг программы с подробными комментариями представлен на [рис. 10.1](#).

```
Sub Пример_OM1_способ1()  
'Вычисление суммы чисел  
'Объявление переменных для каждого из чисел  
Dim x As Single, y As Single, z As Single, n As Single  
Dim g As Single, f As Single, S As Single  
  
'Ввод значений переменных  
x = Val(InputBox("Введите x")) : y = Val(InputBox("Введите y"))  
z = Val(InputBox("Введите z")) : n = Val(InputBox("Введите n"))  
g = Val(InputBox("Введите g")) : f = Val(InputBox("Введите f"))  
  
'Вычисление значения S  
S = x + y + z + n + g + f  
Debug.Print "S ="; S      'Вывод результата в окно отладки  
End Sub
```

Рис. 10.1. Листинг программы решения [примера OM1](#) (способ 1)

Преимущество этого способа заключается в том, что в конце работы сохраняются все введенные значения, недостаток – объем программы зависит от количества чисел (например, для 500 чисел он будет очень большим).

Второй способ. Каждое число будем вводить в цикле в одну и ту же переменную x , а для вычисления суммы организуем цикл накопления (см. подразд. 8.3). Листинг программы с подробными комментариями представлен на рис. 10.2.

```
Sub Пример_OM1_способ2()
'Вычисление суммы чисел
'Объявление переменных
Dim x As Single      'произвольное число
Dim i As Byte        'параметр цикла
Dim S As Single      'сумма чисел

S = 0                'Начальное значение S

For i = 1 To 6      'Заголовок цикла с параметром i
    x = Val(InputBox("Введите x" & i)) 'ввод i-го значения x
    S = S + x        'накопление суммы S
Next i              'Возврат к заголовку цикла

Debug.Print "S ="; S 'Вывод результата в окно отладки
End Sub
```

Рис. 10.2. Листинг программы решения примера OM1 (способ 2)

Преимущество второго способа состоит в том, что объем программы не зависит от количества чисел (программа вводит каждое новое значение в одну и ту же переменную x и добавляет его к сумме предыдущих чисел), недостаток – в конце цикла переменная x хранит только последнее введенное значение, все предыдущие значения оказываются утраченными. Таким образом, положительные и отрицательные моменты рассмотренных способов решения данного примера прямо противоположны.

Чтобы объединить положительные свойства обоих способов, существуют специальные средства. В тех случаях, когда необходимо манипулировать множеством значений одной переменной с сохранением этих значений в памяти ЭВМ, используют переменную с индексом. *Индекс* – порядковый номер очередного значения переменной, он может быть записан в виде константы, переменной или арифметического выражения и может принимать целые положительные значения и быть нулем. Индексы записывают в скобках после имени переменной, например: $z(5)$, $x1(i)$, $pr(n+2)$.

Переменные с индексами являются элементами массивов.

Массив – упорядоченный по возрастанию индексов набор значений одной переменной.

Массив характеризуется:

- именем (записывается по обычным правилам – см. [подразд. 3.2](#));
- количеством измерений (индексов): одномерные, двумерные, многомерные массивы;
- размером (верхней границей каждого из индексов).

Примерами одномерных массивов в математике являются векторы, двумерных – матрицы.

Отдельными элементами массива можно пользоваться так же, как и простыми переменными, не забывая при этом в скобках указывать индекс, например: $x(5) = 2$, $r(7) = 2 * x + 4$, $x(i) = y(k) + 1$, `Debug.Print x(n)`.

По умолчанию элементы массива в VBA нумеруются, начиная с нуля. Чтобы установить нумерацию элементов всех массивов с единицы, в тексте программы на уровне общих объявлений модуля необходимо записать оператор `Option Base 1`.

До использования массива в программе его необходимо объявить, это можно сделать с помощью оператора `Dim` с указанием имени массива, максимально возможного индекса и типа данных в нем:

`Dim Имя_массива(Max_индекс) As Тип`

При этом в качестве максимального индекса может быть указано только конкретное числовое значение или объявленная ранее константа, например:

`Dim A(7) As Single`

или: `Const n As Byte = 7 : Dim A(n) As Single`

При выполнении оператора `Dim` в памяти ЭВМ резервируется необходимое для размещения массива количество ячеек (в данном случае 8 – от 0 до 7, при наличии в программе оператора `Option Base 1` – 7 ячеек, от 1 до 7).

Работа с каждым элементом массива может включать в себя любые операции с ним: ввод, вывод, расчет, проверку выполнения какого-либо условия и др. Обычно работа с элементами массива осуществляется в цикле.

Для удобства дальнейшего изложения введем следующие обозначения:

i – индекс элемента массива;

ni – начальное значение индекса;

ki – конечное значение индекса;

Δi – шаг изменения индекса;

x_i – значение i -го элемента массива x .

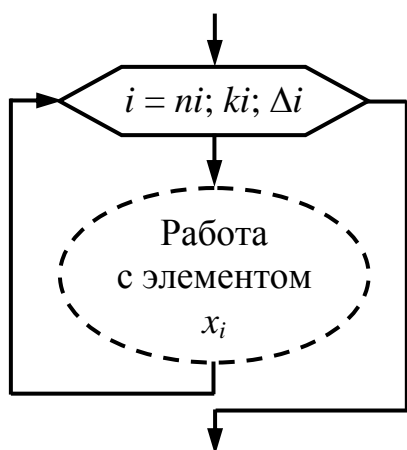
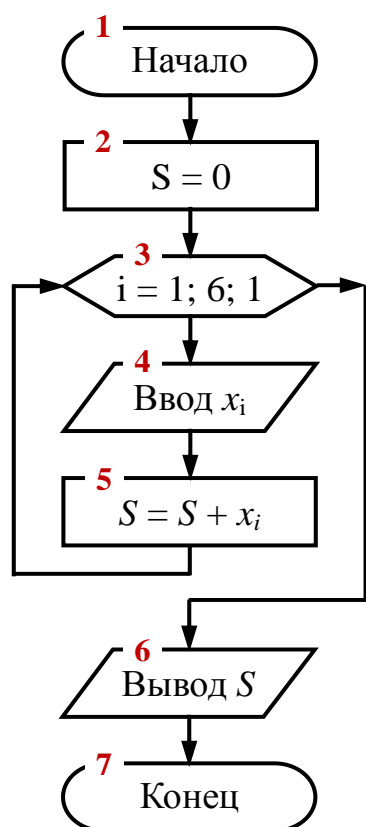


Рис. 10.3. ГСА обработки массива в общем виде

Графическая схема алгоритма обработки массива в общем виде приведена на рис. 10.3.

Третий способ. Запишем решение примера ОМ1 с использованием массива. ГСА и листинг программы с подробными комментариями представлены на рис. 10.4 (отличия от второго способа решения задачи отмечены в листинге программы жирным подчеркнутым шрифтом).

В этом примере ввод элементов массива выполняется с помощью функции InputBox. Однако такой способ используется в редких случаях, поскольку при каждом запуске программы все значения приходится вводить заново. Поэтому чаще ввод элементов массива осуществляют другими способами, изложенными в подразд. 10.2.



а

```

Sub Пример_ОМ1_способ3()
'Вычисление суммы чисел
'Объявление переменных
'x(6) – массив размерностью 6 элементов
'i – индекс элемента массива (параметр цикла)
'S – сумма чисел
Dim x(6) As Single
Dim i As Byte
Dim S As Single

S = 0           'Начальное значение S

For i = 1 To 6  'Заголовок цикла с параметром i
  'ввод значения i-го элемента массива x
  x(i) = Val(InputBox("Введите x" & i))

  S = S + x(i)  'накопление суммы S
Next i          'Возврат к заголовку цикла

'Вывод результата в окно отладки
Debug.Print "S ="; S
End Sub
  
```

б

Рис. 10.4. Решение примера ОМ1 (способ 3): ГСА (а) и листинг программы (б)

10.2. Ввод массива

10.2.1. Считывание массива с листа Excel

Заполнение массива числовыми данными из ячеек электронной таблицы осуществляется в цикле с использованием команды Cells(i, j), где i, j – соответственно номер строки и столбца, на пересечении которых находится ячейка с данными. При небольшом размере массива данные для него обычно набирают на листе Excel подряд либо в одной строке, либо в одном столбце, тогда фрагмент программы для ввода (чтения) элементов массива с листа Excel имеет вид:

```
'Ввод массива с листа Excel
Dim a(6) As Single, i As Byte
For i = 1 To 6
    a(i) = Cells(3, i) 'Ввод из 3-й строки листа Excel
или: a(i) = Cells(i, 5) 'Ввод из 5-го столбца листа Excel
Next i
```

10.2.2. Заполнение массива случайными числами

Этот способ обычно используется при отладке программ.

Для генерирования случайных чисел применяется функция Rnd (см. [пункт 5.2.4](#)). Программный код для заполнения массива из шести элементов случайными числами от –50 до +50 можно записать в следующем виде:

```
'Заполнение массива случайными числами
Randomize Timer 'Установка базы генератора случайных чисел
Dim a(6) As Single, i As Byte
For i = 1 To 6
    a(i) = Fix(Rnd * 101 – 50) 'Случайные числа от –50 до +50
Next i
```

10.3. Вывод массива

Вывод элементов массива на экран монитора должен выполняться таким образом, чтобы обеспечить наглядность и удобство восприятия полученных результатов.

Вывод одномерного массива в строку или столбец на рабочий лист Excel:

```
For i = 1 To 6
```

```
    Cells(4, i) = a(i)    'Вывод в 4-ю строку листа Excel
```

```
или: Cells(i, 8) = a(i)    'Вывод в 8-й столбец листа Excel
```

```
Next i
```

Аналогичным образом массив выводится в окно отладки Immediate:

```
For i = 1 To 6
```

```
    Debug.Print a(i);    'Вывод в окно отладки в строку
```

```
или: Debug.Print a(i)    'Вывод в окно отладки в столбец
```

```
Next i
```

10.4. Решение задач с использованием массивов

В общем случае работа с массивом производится следующим образом: организуются циклы ввода, обработки и вывода элементов массива, как показано на [рис. 10.5](#). Однако если для решения задачи нет необходимости одновременно иметь все элементы массива, то ввод, вывод элементов массива и работу с ними можно совмещать в одном цикле, как показано на [рис. 10.6](#) и выполнено в [примере ОМ1](#).

Пример ОМ2. Массив $X(10)$ заполнить числовыми данными, расположенными в первой строке листа Excel. Найти количество и произведение положительных элементов массива с четными индексами. Исходные данные и результаты расположить на листе Excel в соответствии с [рис. 10.7](#).

ГСА решения [примера ОМ2](#) приведена на [рис. 10.8](#), листинг программы с подробными комментариями – на [рис. 10.9](#). С целью уменьшения объема программы и повышения быстродействия ее работы все операции с элементами массива реализованы в одном цикле.

Пример ОМ3. Массив $Y(12)$ заполнить случайными числами в диапазоне от -50 до 50 и разместить его индексы во втором столбце, элементы – в третьем столбце листа Excel. Найти минимальный элемент среди отрицательных элементов массива и его порядковый номер. Найденные значения вывести в строке рядом с соответствующим элементом массива.

ГСА решения [примера ОМ3](#) приведена на [рис. 10.10](#), листинг программы с подробными комментариями – на [рис. 10.11](#).

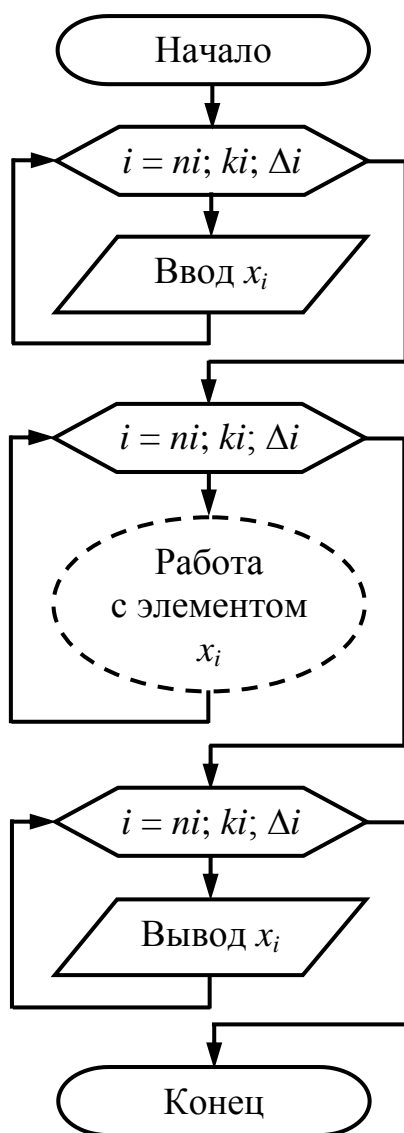


Рис. 10.5. ГСА работы с массивом в общем случае

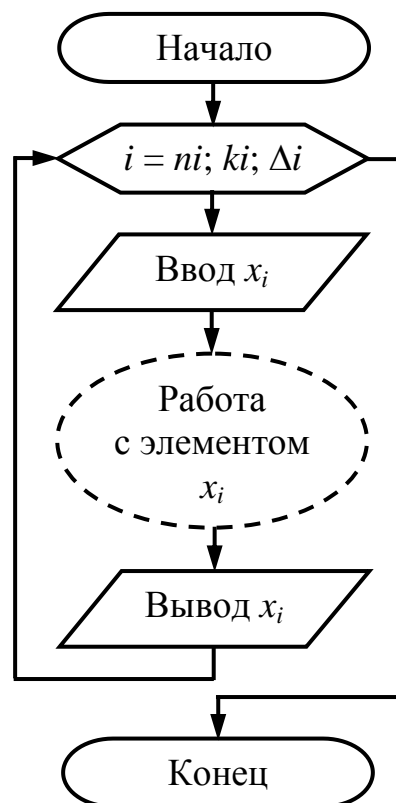


Рис. 10.6. ГСА работы с массивом в одном цикле

	A	B	C	D	E	F	G	H	I	J	K
1	-2	1	4	-15	8	14	12	16	15	32	Исх. данные
2											
3	Работа с массивом										
4	1	2	3	4	5	6	7	8	9	10	Индекс i
5	-2	1	4	-15	8	14	12	16	15	32	Знач. X(i)
6											
7		1				14		16		32	Знач. X(i) > 0
8											
9	k =	4									
10	p =	7168									

Рис. 10.7. Фрагмент листа Excel с данными примера OM2

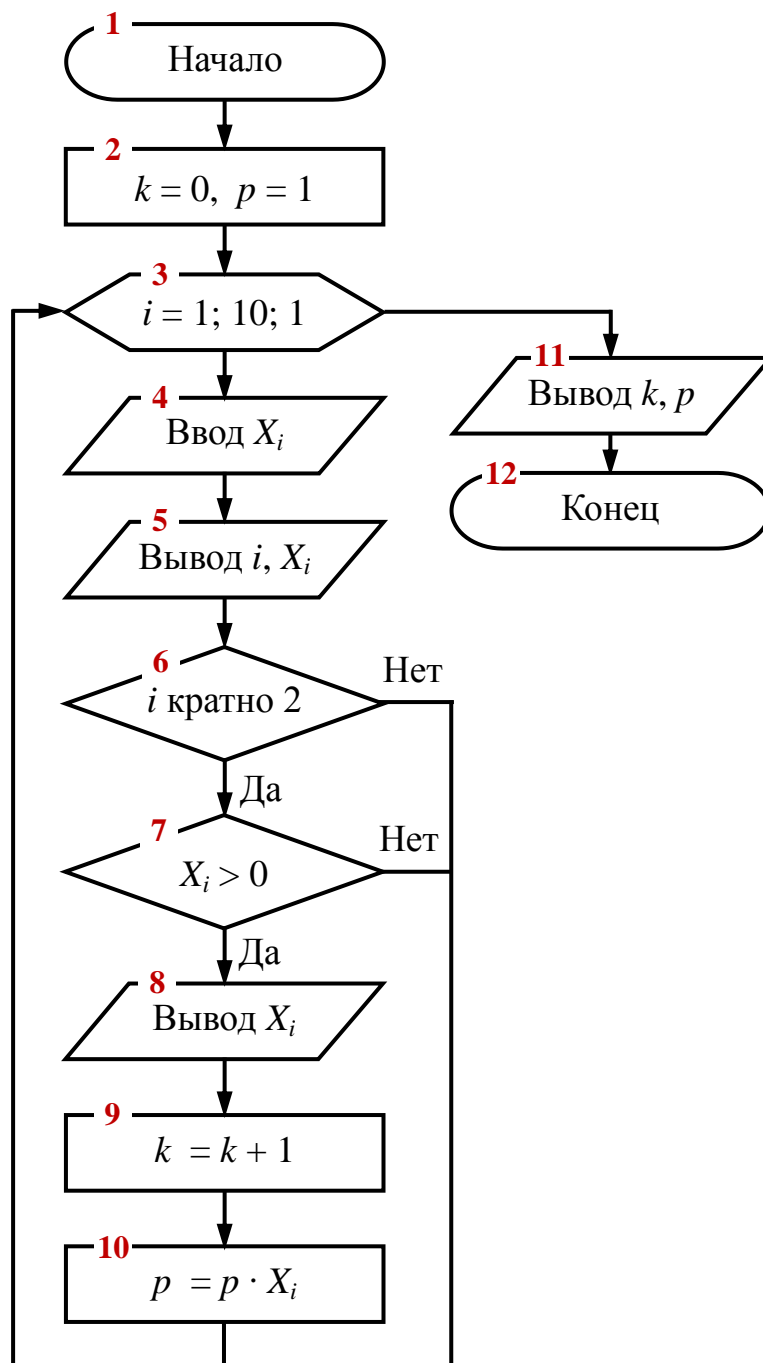


Рис. 10.8. ГСА решения примера ОМ2

```

Sub OM2()
'Одномерный массив (количество и произведение элементов)

'Объявление переменных
'X(10) – массив размерностью 10 элементов,
'i – индекс элемента массива (параметр цикла),
'k – счетчик количества положительных элементов с четными индексами,
'p – произведение положительных элементов с четными индексами
Dim X(10) As Single, i As Integer
Dim k As Integer, p As Single

k = 0 : p = 1      'Начальные значения k и p

Cells(3, 1) = "Работа с массивом"   'Вывод текстового сообщения

For i = 1 To 10      'Заголовок цикла с параметром i
    X(i) = Cells(1, i)      'заполнение массива из первой строки листа Excel

    'вывод индекса элемента массива i в 4-ю строку листа Excel,
    'значения элемента массива X(i) – в 5-ю строку
    Cells(4, i) = i : Cells(5, i) = X(i)

    'проверка условий с помощью функции And,
    'не путать индекс элемента i и его значение X(i)
    If i Mod 2 = 0 And X(i) > 0 Then
        'вывод элемента массива X(i), удовлетворяющего условиям
        Cells(7, i) = X(i)

        k = k + 1      'вычисление количества элементов
        p = p * X(i)   'вычисление произведения элементов
    End If             'завершение If
Next i                 'Возврат к заголовку цикла

'Вывод результатов на лист Excel
Cells(9, 1) = "k =" : Cells(9, 2) = k      'в 9-ю строку
Cells(10, 1) = "p =" : Cells(10, 2) = p    'в 10-ю строку
End Sub

```

Рис. 10.9. Листинг программы решения примера OM2

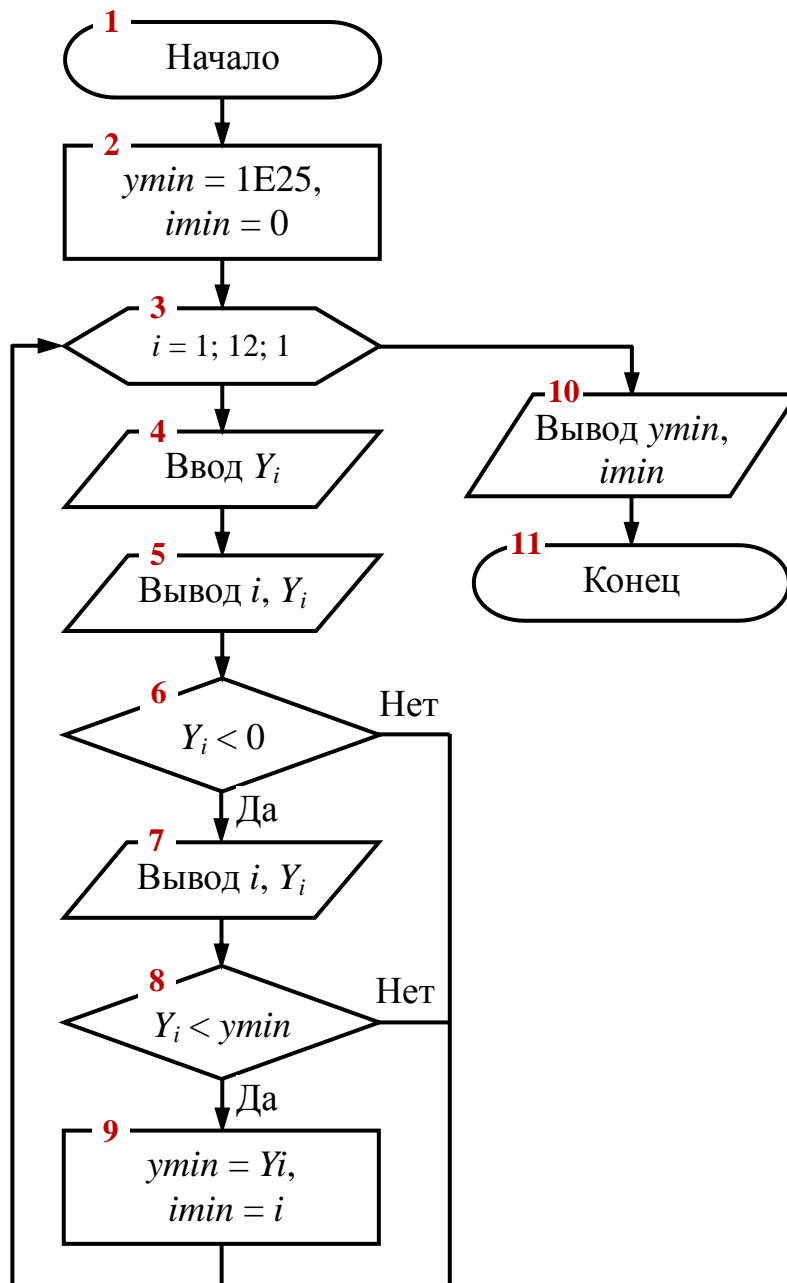


Рис. 10.10. ГСА решения примера ОМЗ

```

Sub OM3()
'Одномерный массив (поиск минимума)

'Объявление переменных
'Y(12) – массив размерностью 12 элементов,
'i – индекс элемента массива (параметр цикла)
'ymin – минимальное значение элемента массива
'imin – индекс минимального элемента массива
Dim Y(12) As Single, i As Integer
Dim ymin As Single, imin As Integer

'Начальные значения минимального элемента массива и его порядкового номера
ymin = 1E+25 : imin = 0

Cells(1, 1) = "Работа с массивом"    'Вывод текстового сообщения

For i = 1 To 12    'Заголовок цикла с параметром i
    'заполнение массива случайными числами от -50 до + 50
    Y(i) = Fix(Rnd * 101 - 50)

    'вывод индекса элемента массива i во 2-й столбец,
    'значения элемента Y(i) – в 3-й столбец
    Cells(i, 2) = i : Cells(i, 3) = Y(i)

    If Y(i) < 0 Then    'проверка условия (отрицательный элемент)
        Cells(i, 4) = Y(i)    'вывод отрицательного элемента в 4-й столбец

        If Y(i) < ymin Then    'проверка условия (минимальный элемент)
            ymin = Y(i)    'сохранение значения минимального элемента
            imin = i    'сохранение индекса минимального элемента
        End If    'завершение внутреннего If (Y(i) < ymin)
    End If    'завершение внешнего If (Y(i) < 0)
Next i    'Возврат к заголовку цикла

'Вывод индекса и значения минимального элемента
'в строке соответствующего элемента массива
Cells(imin, 5) = "imin=" : Cells(imin, 6) = imin
Cells(imin, 7) = "ymin=" : Cells(imin, 8) = ymin
End Sub

```

Рис. 10.11. Листинг программы решения примера OM3

Примеры OM1 – OM3 включают в себя наиболее часто встречающиеся программные структуры, которые целиком или отдельными фрагментами могут быть использованы при решении широкого круга задач.

Для удобства применения на практике рассмотренные в данном пособии операторы VBA представлены в систематизированном виде в [прил. 1](#).

11. ОТЛАДКА ПРОГРАММЫ И ОБРАБОТКА ОШИБОК

При работе с программой достаточно часто возникают ошибки, причем ошибки разного рода и уровня сложности. При возникновении ошибки VBA выводит на экран окно сообщений с запросом на прерывание или продолжение выполнения программы. Условно ошибки можно поделить на три типа: ошибки компиляции, ошибки выполнения и логические ошибки.

11.1. Ошибки компиляции

Ошибки компиляции (*Compile error*) возникают, если VBA не может интерпретировать введенный код. Такие ошибки компилятор VBA обнаруживает при вводе текста программы или при ее запуске. Ошибки компиляции относятся либо к одной строке (ошибки синтаксиса), либо к группе строк (ошибки структуры).

Редактор кода выполняет автоматическую проверку синтаксиса каждой набранной строки программы. Если после нажатия клавиши Enter строка выделяется красным цветом, это указывает на наличие ошибки в набранной строке, например, использование недопустимого имени, некорректное количество скобок, неполный ввод инструкции и т. д.

Другие ошибки компиляции обнаруживаются при запуске программы. В этом случае предполагаемое местоположение ошибки выделяется синей заливкой (рис. 11.1). Помимо выделения цветом VBA отображает на экране окно *Visual Basic for Applications* с сообщением о характере и возможной причине ошибки, например, попытке использовать переменную, которая ранее не была объявлена (при использовании в программном модуле инструкции Option Explicit), дублировании имен констант и (или) переменных в одной процедуре и т. д.

Примеры типичных ошибок компиляции приведены в прил. 2.

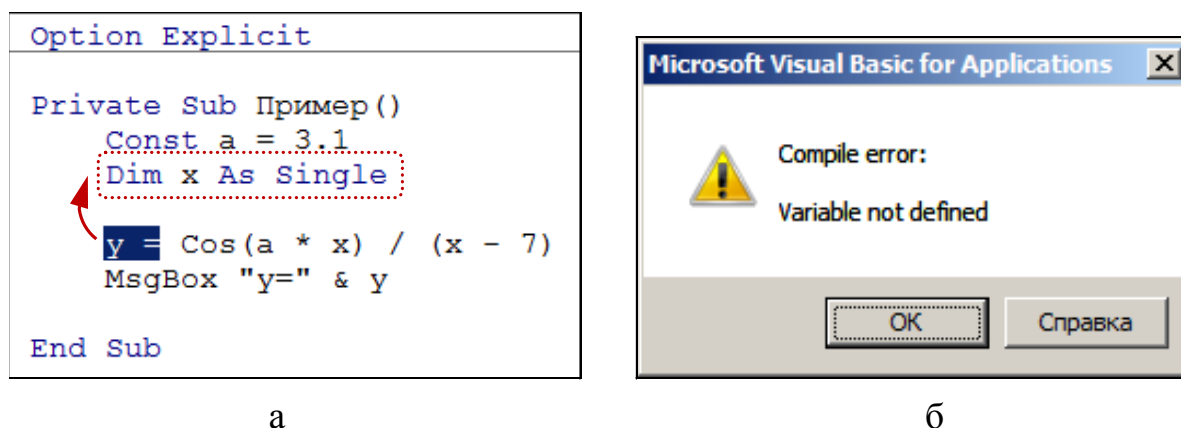


Рис. 11.1. Пример ошибки компиляции (переменная *y* не определена):

а – листинг программы; б – окно с сообщением

11.2. Ошибки времени выполнения

Ошибки времени выполнения (*run-time error*) возникают после успешной компиляции программы при ее выполнении. Причинами таких ошибок могут быть, например, переполнение памяти, некорректные данные или некорректность вычислений. В этих случаях VBA отображает на экране окно *Microsoft Visual Basic* с сообщением о номере ошибки и возможной причине, ее вызвавшей (рис. 11.2). Если в окне сообщения нажать кнопку **Debug** (прерывание программы для отладки), то в модуле желтым цветом и стрелкой будет выделена строка, по причине которой выполнение программы было прервано. Режим прерывания предоставляет возможность узнать текущее значение переменных. Для этого достаточно навести указатель мыши на имя переменной, что вызовет появление всплывающей подсказки с текущим значением этой переменной.

Примеры типичных ошибок времени выполнения приведены в [прил. 2](#).

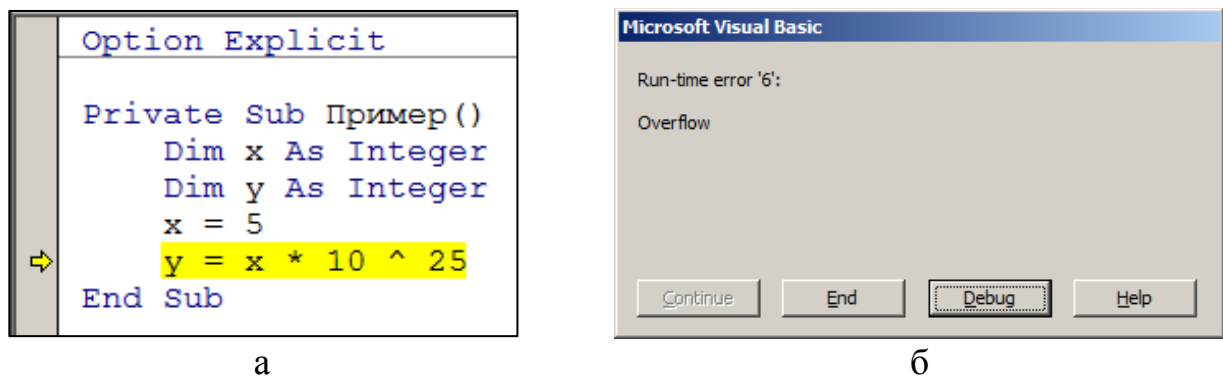


Рис. 11.2. Пример ошибки времени выполнения (переполнение):
а – листинг программы; б – окно с сообщением

11.3. Логические ошибки

Логические ошибки труднее всего обнаружить и устранить. Эти ошибки не приводят к прерыванию выполнения программы, т. е. визуально все выглядит так, как будто программа работает безупречно. Однако программа выдает неверные результаты. Логическая ошибка является следствием плохо разработанного алгоритма, недоучета диапазона возможных значений входных данных и т. д. Локализация логических ошибок связана с привлечением дополнительных средств отладки VBA (пошаговое выполнение программы, точки останова, вывод контрольных значений и др.).

12. КОНТРОЛЬНЫЕ ВОПРОСЫ

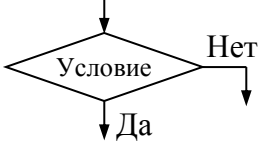
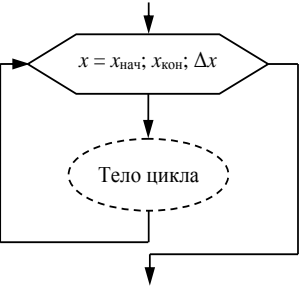
- 1) Какие форматы используются для записи вещественных чисел?
- 2) Перечислите арифметические операции VBA.
- 3) Какими способами можно осуществить ввод данных в программу?
- 4) Перечислите правила записи встроенных функций на VBA.
- 5) Как программно установить цвет шрифта или заливки на листе Excel?
- 6) Какие операторы могут использоваться в программах, реализующих алгоритмы линейного вычислительного процесса?
- 7) Назовите приоритет выполнения операций в арифметическом выражении.
- 8) Какие элементы управления могут размещаться на пользовательской форме?
- 9) Как на пользовательской форме изменить последовательность автоматического перехода по объектам?
- 10) В каких случаях применяется оператор If?
- 11) Могут ли операторы If быть вложенными?
- 12) Каковы особенности записи однострочного оператора If?
- 13) В каких случаях используется сокращенная форма разветвления?
- 14) В каких случаях используется логическая операция And?
- 15) В каких случаях используется логическая операция Or?
- 16) С помощью какого оператора реализуется множественный выбор?
- 17) Опишите синтаксис оператора Select Case.
- 18) Опишите синтаксис оператора GoTo.
- 19) В каких случаях применяется арифметический цикл?
- 20) Что такое параметр арифметического цикла?
- 21) Опишите формат записи оператора For ... Next.
- 22) В чем заключается особенность цикла со счетчиком?
- 23) Какая алгоритмическая конструкция применяется для вычисления экстремума функции?
- 24) Опишите порядок вычисления максимума и минимума функции.
- 25) Как реализуются циклы накопления сумм и произведений?
- 26) Чем характеризуется массив данных?
- 27) Как задать нумерацию элементов массива, начиная с единицы?
- 28) Что такое процедура-подпрограмма? Какие параметры в процедуре-подпрограмме являются формальными, а какие фактическими?

Библиографический список

1. Лебедев, В. М. Программирование на VBA в MS Excel : учебное пособие / В. М. Лебедев. – Москва : Юрайт, 2020. – 306 с. – Текст : непосредственный.
2. Казанский, А. А. Прикладное программирование на Excel 2019 : учебное пособие / А. А. Казанский. – Москва : Юрайт, 2020. – 171 с. – Текст : непосредственный.
3. Моисеева, Н. А. Языки и технологии программирования / Н. А. Моисеева. – Омск : Омский гос. ун-т путей сообщения, 2019. – 29 с. – Текст : непосредственный.
4. Основы алгоритмизации / Е. А. Сидорова, С. П. Железняк [и др.]. – Омск : Омский гос. ун-т путей сообщения, 2020. – 35 с. – Текст : непосредственный.
3. ГОСТ 19.701-90 (ИСО 5807-85). Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. – Москва : Изд-во стандартов, 1990. – 36 с. – Текст : непосредственный.

Основные операторы VBA

Блок ГСА	Назначение	Оператор, функция	Формат записи	Пример применения
1	2	3	4	5
	Начало процедуры	Sub	Sub Имя_процедуры()	Sub Иванов_Лин_з1_в5()
	Конец процедуры	End Sub	End Sub	End Sub
	Ввод данных в диалоговое окно	InputBox	Имя_переменной = InputBox("Сообщение")	x = Val(InputBox("Введите x"))
	Ввод данных с листа Excel	Cells	Имя_переменной = Cells(i, j)	x = Cells(1, 2)
	Вывод данных в диалоговое окно	MsgBox	MsgBox Сообщение	MsgBox "При x =" & x & " y =" & y
	Вывод данных на лист Excel	Cells	Cells(i, j) = Сообщение	Cells(1, 1) = "x =" Cells(1, 2) = x
	Вывод данных в окно отладки Immediate	Debug.Print	Debug.Print [Сообщение]	Debug.Print "При x ="; x, "y ="; y
	Вычисление (присваивание значения)	Let	[Let] Имя_переменной = Выражение	Let z = 4.35 y = z + pi * Cos(x) ^ 2

1	2	3	4	5
	Проверка условия и выбор направления выполнения	If	<p><i>Блочный If:</i></p> <p>If <i>Условие</i> Then 'Операторы' [Else 'Операторы'] End If</p> <p><i>Линейный If:</i></p> <p>If <i>Условие</i> Then Операторы_ [Else Операторы]</p>	<p>If $x = 0$ Then MsgBox "Деление на ноль" End If</p> <p>If $x < > 0$ Then $y = \text{Log}(\text{Abs}(x))$ Else $y = \text{Exp}(x)$ End If</p>
	Арифметический цикл (цикл с параметром)	For ... Next	For $x = x_{\text{нач}}$ To $x_{\text{кон}}$ [Step Δx] 'Операторы (тело цикла)' Next $[x]$	For $x = 0$ To 1 Step 0.1 $y = \sin(x)$ Debug.Print x, y Next x

1	2	3	4	5
	Итерационный цикл с предусловием	While ... Wend	While <i>Условие</i> 'Операторы (тело цикла) Wend	While $x > 0$ $y = \sin(x)$ Debug.Print x, y $x = x - 1$ Wend
		Do While ... Loop	Do While <i>Условие</i> 'Операторы (тело цикла) Loop	Do While $x > 0$ $y = \sin(x)$ Debug.Print x, y $x = x - 1$ Loop
	Итерационный цикл с постусловием	Do ... Loop While	Do 'Операторы (тело цикла) Loop While <i>Условие</i>	Do $y = \sin(x)$ Debug.Print x, y $x = x - 1$ Loop While $x > 0$

Наиболее распространенные ошибки при работе на VBA
(представлены в алфавитном порядке по тексту сообщения об ошибке)

Текст сообщения об ошибке	Перевод на русский язык	Причина ошибки
1	2	3
<i>Ошибки компиляции (Compile error)</i>		
Ambiguous name detected: ...	Повтор имени	Определено две процедуры с одинаковым именем
Assignment to constant not permitted	Изменение значения константы не разрешено	Значение константы не может быть изменено
Block If without End If	Блочный If без End If	Конструкция блочного оператора If должна завершаться ключевым словом End If
Duplicate declaration in current scope	Дублирование объявления в текущей области	Одинаковое имя для константы и переменной
Expected:)	Ожидалась скобка	Несоответствие парности открывающих и закрывающих скобок в выражении
Expected: end of statement	Ожидалось завершение оператора	В линейном операторе If не должно быть ключевого слова End If
Expected: identifier	Ожидался идентификатор	Недопустимое имя для переменной
Expected: list separator or)	Ожидался разделитель списка или скобка	Отсутствует закрывающая скобка функции
Expected: Then or Go To	Ожидалось ключевое слово Then или Go To	В строке оператора If отсутствует ключевое слово Then
For without Next	For без Next	Конструкция For должна завершаться ключевым словом Next

1	2	3
Next without For	Next без For	Нарушена вложенность конструкций, например, в теле цикла в блочной конструкции If отсутствует ключевое слово End If
Select Case without End Select	Select Case без End Select	Конструкция Select Case должна завершаться ключевым словом End Select
Sub or Function not defined	Процедура или функция не определена	Неверная запись функции на VBA
User-defined type not defined	Пользовательский тип не определен	Ошибка типа данных при объявлении переменной
Variable not defined	Переменная не определена	Переменная не объявлена оператором Dim
Wrong number of arguments or invalid property assignment	Неправильное количество аргументов или неверно назначено свойство	Несоответствие количества формальных аргументов фактическим аргументам при вызове функции
<i>Ошибки времени выполнения (Run-time error)</i>		
Run-time error '5': Invalid procedure call or argument	Недопустимый вызов процедуры или аргумент	Значение аргумента функции выходит за пределы допустимых значений
Run-time error '6': Overflow	Переполнение	Попытка присвоить переменной указанного типа значение, превышающее ограничение для этого типа данных
Run-time error '9': Subscript out of range	Индекс выходит за пределы допустимого диапазона	Попытка вывести элемент массива с индексом, находящимся за пределами объявленного диапазона
Run-time error '11': Division by zero	Деление на ноль	При вычислении выражения делитель равен нулю
Run-time error '13': Type mismatch	Несоответствие типов	Попытка присвоить переменной указанного типа значение переменной типа String

Учебное издание

СИДОРОВА Елена Анатольевна,
ЖЕЛЕЗНЯК Светлана Петровна

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ VBA

Учебное пособие

Редактор Н. А. Майорова

Подписано в печать 27.04.2021. Формат $60 \times 84 \frac{1}{16}$.
Офсетная печать. Бумага офсетная. Усл. печ. л. 7,4. Уч.-изд. л. 8,3.
Тираж 100 экз. Заказ .

**

Редакционно-издательский отдел ОмГУПСа
Типография ОмГУПСа

*

644046, г. Омск, пр. Маркса, 35