О.В. Гаркуша

Инструментальные средства MS Office

Учебное пособие

```
Private Sub CB_Summ_Click()
Dim A, B, C As Integer
A = Val(TextBox_A.Value)
B = Val(TextBox_B.Value)
C = A + B
TextBox_C.Value = C
End Sub
```

Краснодар
2022

Министерство науки и высшего образования
Российской Федерации

КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

О.В. ГАРКУША

ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА MS Office

Учебное пособие

Краснодар
2022

УДК 519.682(075.8)
ББК 32.972.13я73
Г 204

Рецензенты:
Доктор физико-математических наук, профессор
Е.Н. Калайдин
Кандидат физико-математических наук, доцент
С.Е. Рубцов

Гаркуша, О.В.
Г 204 Инструментальные средства MS Office: учебное пособие / О.В. Гаркуша; Министерство науки и высшего образования Российской Федерации, Кубанский государственный университет. – Краснодар: Кубанский гос. ун-т, 2022. – 165 с. – 500 экз.
ISBN 978-5-8209-2049-3

Содержит материал о возможностях использования средств визуального программирования и решения задач линейного программирования в приложениях MS Office.

Адресуется студентам III курса направления бакалавриата 01.03.02 «Прикладная математика и информатика», 02.03.03 «Математическое обеспечение и администрирование информационных систем», направления бакалавриата 02.03.02 «Фундаментальные информатика и информационные технологии», 09.03.03 «Прикладная информатика», а также магистрам и аспирантам для изучения основ визуального программирования и решения задач линейного программирования в приложениях MS Office.

УДК 519.682(075.8)
ББК 32.972.13я73

ISBN 978-5-8209-2049-3

ВВЕДЕНИЕ

Учебное пособие «Инструментальные средства MS Office» предназначено для изучения и практического освоения возможностей MS Word и MS Excel.

В РАЗДЕЛЕ 1 рассматриваются приемы использования инструментальной среды Visual Basic for Applications (VBA) в приложениях MS Word и MS Excel, создания несложных программ для решения задач на персональном компьютере. Учебное пособие подготовлено с учетом того, что читатель в достаточной степени освоил приемы использования макросов в MS Excel и Word.

РАЗДЕЛ 2 предлагает варианты решения задач линейного программирования средствами MS Excel.

Под линейным программированием понимают раздел прикладной математики, имеющий дело с теорией и численными методами минимизации линейных функций при наличии ограничений, описываемых конечными системами линейных неравенств.

Первые постановки задач линейного программирования были сформулированы известным советским математиком Л.В. Канторовичем, которому за эти пионерские работы была присуждена Нобелевская премия по экономике.

Значительное развитие теория и алгоритмический аппарат линейного программирования получили с изобретением и распространением ЭВМ и формулировкой американским математиком Дж. Данцингом симплекс-метода.

В настоящее время линейное программирование является одним из наиболее употребительных аппаратов математической теории оптимального принятия решения. Для решения задач линейного программирования разработано сложное программное обеспечение, дающее возможность эффективно и надежно решать практические задачи больших объемов.

Возможности MS Excel предоставляют возможность решения задач оптимизации.

Предлагаемое издание адресуется студентам всех специальностей направлений бакалавриата факультетам компьютерных технологий и прикладной математики, экономического факультета, а также желающим изучить основы

программирования в среде VBA самостоятельно. Оно может быть также рекомендовано преподавателям для проведения практических занятий.

При подготовке пособия автор не ставил цель описать все возможности VBA, а исходил только из требований учебных программ различных вузов. Для более глубокого освоения программирования на VBA следует пользоваться дополнительными источниками.

Данное учебное пособие написано с учетом опыта преподавания на факультете компьютерных технологий и прикладной математики и на экономическом факультете Кубанского государственного университета.

РАЗДЕЛ 1. СРЕДА ПРОГРАММИРОВАНИЯ VISUAL BASIC FOR APPLICATIONS (VBA)

1. VBA КАК СИСТЕМА ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

Из всех офисных программ (не считая Access, который изначально является средством разработки) для создания пользовательских приложений чаще всего используется Excel, так как именно данный пакет предназначен для широкого круга прикладных задач по обработке данных.

Всего несколько строчек кода, включенных в Excel, смогут создать программу для выполнения серьезных вычислений и оригинального анализа с использованием графики и выдачей отчетов. Однако для разработки собственных приложений необходимы следующие условия:

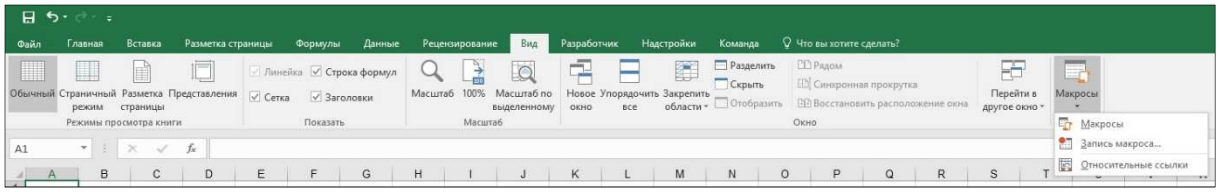
– нужно научиться работать с Excel в его традиционном варианте, что позволит с помощью минимального программного кода использовать максимум встроенных возможностей пакета. Чем лучше вы знаете сам продукт, тем проще создавать приложения;

– требуется освоить иерархическую модель объектов Excel вместе с соответствующими свойствами, методами и событиями, через которые производится управление средой Excel при разработке пользовательского приложения;

– следует изучить среду разработки VBA, где вы можете писать код программ, создавать пользовательские формы и отлаживать свое приложение.

1.1. НАСТРОЙКА ПРИЛОЖЕНИЙ MS OFFICE ДЛЯ РАБОТЫ С МАКРОСАМИ И В СРЕДЕ VBA

Как в MS Excel, так и в MS Word, доступ к возможности записи макроса и задания его свойств (имя, варианты активизации и т.п.) находится в пункте главного меню «Вид». Здесь приведен фрагмент главного меню MS Excel. Аналогично можно получить такую возможность в MS Word.



Вкладка «Разработчик» как в Excel, так и в Word является одним из наиболее значимых, очень полезных и используемых параметров в Excel. Есть несколько приложений со вкладкой «Разработчик» в Excel:

– с помощью редактора Visual Basic на вкладке разработчика можно создавать, записывать и редактировать макросы;

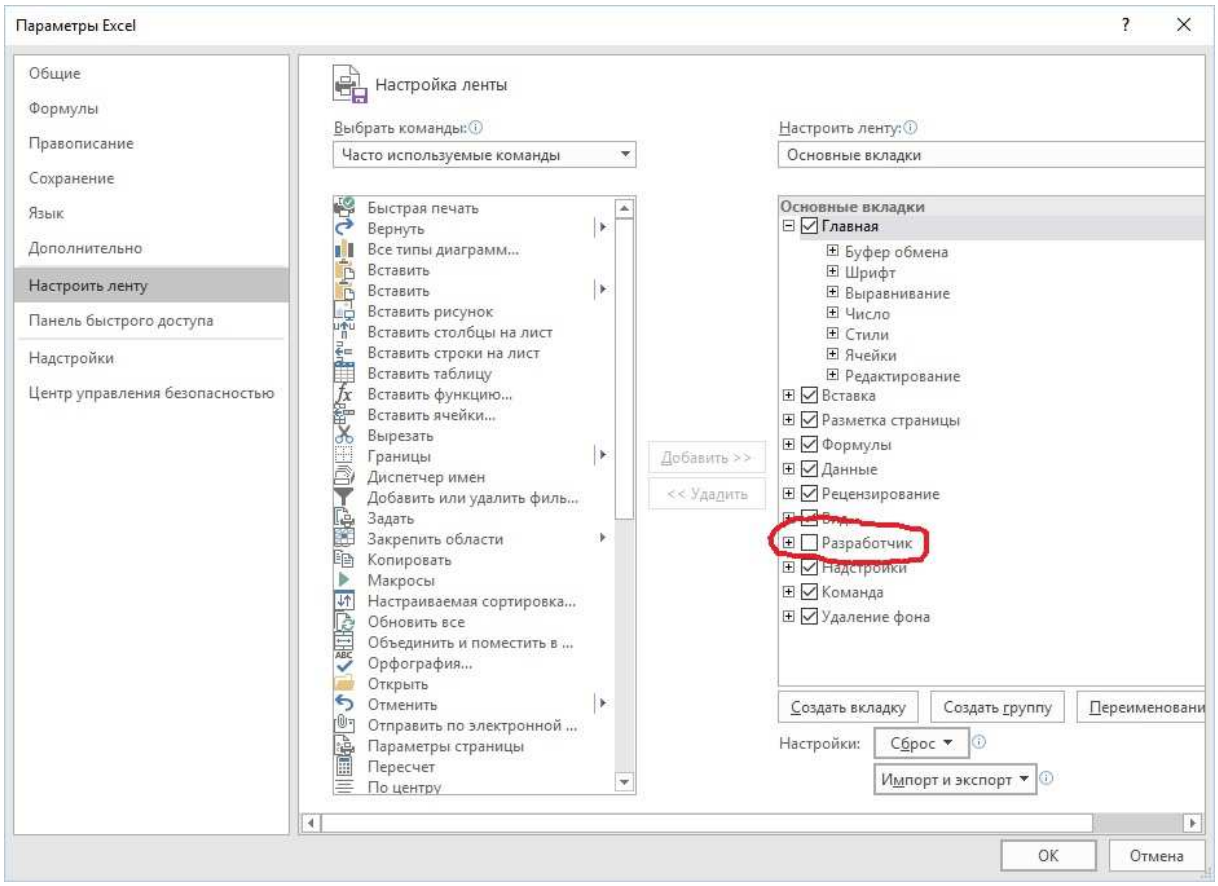
– можно запустить макросы, которые были записаны ранее;

– использовать относительную ссылку для записи макроса (применение этой ссылки к другому диапазону ячеек или рабочему листу, или рабочей книге для запуска макроса);

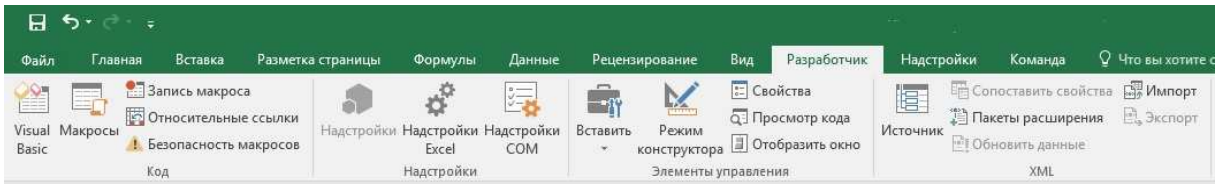
– использовать команды XML (для импорта и экспорта данных расширяемого языка разметки (XML), созданные из других баз данных и приложений;

– применять формы и элементы управления ActiveX в MS Excel, например. Кнопки, полоса прокрутки и флажки.

Для отображения вкладки «Разработчик» нужно:
(Файл > Параметры > Настроить ленту > Основные вкладки) и установить флажок «Разработчик».

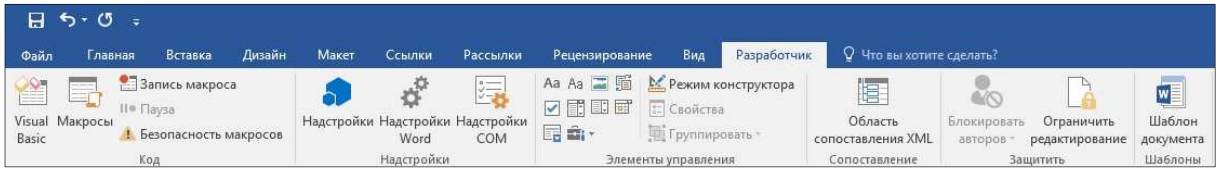


В результате главное меню примет вид

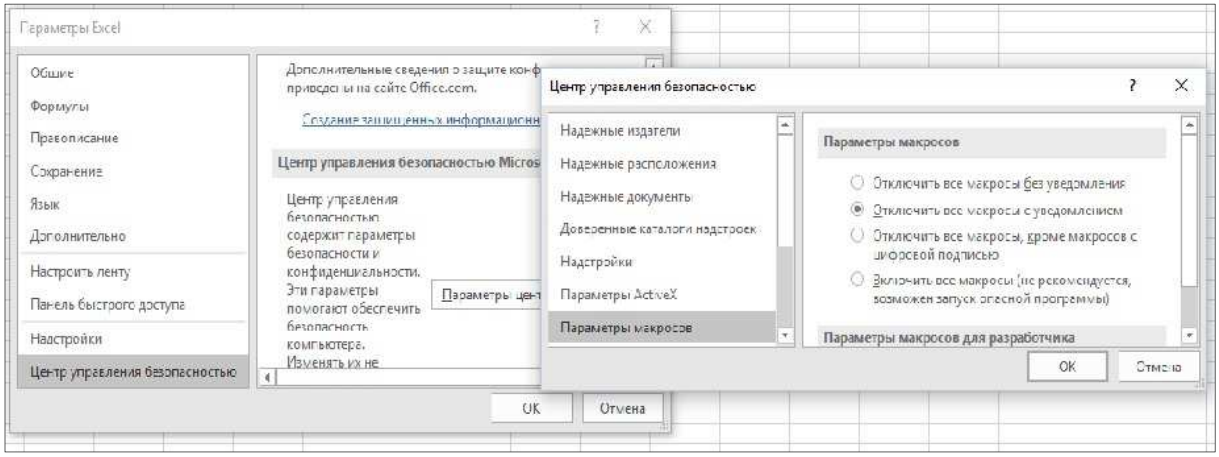


и мы получаем доступ к средствам работы с макросами и возможностям «Режима конструктора».

Аналогичные действия в MS Word позволяют получить такой вид главного меню:



Изменить параметры безопасности макросов можно в центре управления безопасностью:
(Файл > Параметры > Центр управления безопасностью > Параметры центра управления безопасностью > Параметры макросов) и выбрать «Отключить все макросы с уведомлением».



1.2. Общие сведения о VBA

Существует целый ряд систем программирования, позволяющих в той или иной степени реализовать концепцию объектно-ориентированного подхода при разработке программных средств. Наиболее полные возможности имеют такие системы, как Delphi, C++, Java, Visual Basic. В отличие от Visual Basic, VBA не является языком объектно-ориентированного программирования в строгом смысле этого слова, тем не менее в нем очень широко используются элементы ООП и связанные с ним понятия.

Visual Basic for Applications — это подмножество Visual Basic (VB), которое включает почти все его средства создания приложений, структуры данных и управляющие структуры, возможность создания пользовательских типов данных. VBA, как и VB, является языком визуального и событийно-управляемого программирования — в нем есть возможность создания форм со стандартным набором элементов управления и написания процедур, обрабатывающих события, которые возникают при тех или иных действиях системы и конечного пользователя. Кроме того, он позволяет применять элементы ActiveX. В общем, это

полноценный язык программирования, хотя и не обладающий всеми возможностями последних версий Visual Basic. VBA позволяет работать с огромным набором объектов — по существу, в нем определены все объекты приложений MS Office.

Отметим одну, может быть главную, особенность программирования на VBA. Программист, создавая проект на каком-либо языке программирования, работает в соответствующей среде, где язык — главное *средство,* а создание кода — главная *цель* действия программиста. А при работе на VBA целью является создание документа в широком смысле (документа Word, рабочей книги Excel, презентации и т. д.). Проект на VBA — результат побочной деятельности по созданию документа. Более того, проект на VBA, независимо от какого-либо документа, создать нельзя, даже если никакие свойства этого документа не используются.

Приступая к очередному сеансу работы, программист открывает одно из приложений MS Office, и в этот момент в языке VBA автоматически становится доступным объект Application, определяющий это приложение, а также все встроенные в него объекты. Можно определить и создать объект Application для любого приложения MS Office, получив тем самым доступ ко всем его объектам.

Итак, VBA отличается от Visual Basic и прочих языков программирования тем, что он предоставляет возможность непосредственной работы с объектами MS Office. Это позволяет эффективно использовать его для автоматизации деятельности, связанной с обработкой различных типов документов. Обычные средства VBA, унаследованные от Visual Basic, — важная, но не определяющая часть языка.

VBA позволяет существенно расширить вычислительные возможности Excel. С помощью VBA можно легко и быстро создавать различные приложения, даже не являясь специалистом в области программирования. VBA имеет графическую инструментальную среду, позволяющую создавать экранные формы и управляющие элементы. С его помощью можно создавать собственные функции для Excel, вызываемые мастером функций, разрабатывать макросы, создавать собственные меню и др. VBA придает документам элегантность и позволяет с

легкостью решать многие задачи, о возможности выполнения которых средствами Excel можно только мечтать.

Как и Visual Basic, VBA реализует концепцию визуального программирования, управляемого событиями.

Другим подмножеством VBA является язык описания сценариев VBScript. Его основное назначение — создание интерактивных Web-страниц.

Таким образом, изучив программирование в среде VBA, вы без больших усилий можете освоить программирование на Visual Basic или VBScript.

1.3. Объектная модель Excel

Excel состоит из более чем 100 объектов — от ячеек и диаграмм до рабочих книг и электронных таблиц. Каждый объект имеет набор свойств, управляющих его внешним видом и поведением, а также содержит методы, обеспечивающие определенные действия с помощью этого объекта. Отдельные объекты имеют события, которые выдают сообщение пользователю, когда происходит что-нибудь интересное.

Например, объект **`Workbook`** представляет собой конкретный файл Excel. Он имеет свойства **`Name`** (имя файла), **`Path`** (имя каталога) и **`Author`** (имя автора файла).

Существует два способа ссылок на объекты: можно сослаться непосредственно на имя одного из объектов или на индекс в коллекции. Согласно самому простому определению, коллекция — это группа похожих объектов. Все объекты Excel разделяются на два класса: единичные объекты и объекты в коллекции. Для первых ссылка осуществляется непосредственно по их имени, для вторых — по индексу в данной группе. Коллекции предоставляют возможность иерархической организации объектов. Например, коллекция **`Workbooks`** содержит все объекты **`Workbook`**. Чтобы сослаться на конкретную рабочую книгу, можно указать имя: **`Workbooks(«Лист1.xls»)`** или номер **`Workbooks(1)`**.

Ключевыми объектами в Excel являются **`Application`**, **`Workbook`**, **`Worksheet`** и **`Range`**, которые образуют иерархию.

Объект Application представляет собой саму программу Excel. Все приложения Excel/VBA реализуются в Excel, поэтому можно рассматривать этот объект в качестве среды, в которой они осуществляются. Любые установки свойств или вызовы методов, совершенные с объектом **`Application`**, воздействуют на весь Excel и соответственно на все приложения VBA, выполняемые в его среде.

Объект **`Workbook`** является файлом рабочей книги Excel. В терминах разработки приложений его можно рассматривать в качестве механизма доставки или контейнера для любого приложения VBA, созданного при помощи Excel. Любые установки свойств или вызовы методов, совершенные с объектом **`Workbook`**, воздействуют на данное приложение.

Объект **`Worksheet`**, содержащийся в **`Workbook`**, служит нескольким целям в приложении Excel/VBA. Он используется в качестве основы для разработки форм — большинство пользовательских форм в Excel создано с его помощью. Эти объекты также предоставляют многофункциональную сетку (**`grid`**), которая предназначена для вывода и обработки данных; и она содержит ячейки, куда пользователь может включать формулы для выполнения вычислений. При этом свойства и методы объекта **`Worksheet`** обрабатывают электронную таблицу как единое целое.

Объект **`Range`** представляет собой одну или несколько ячеек в электронной таблице. Он используется в основном для хранения и вывода фрагментов данных: чисел, строк или формул. Ячейки электронной таблицы (**`Cells`**), которые представлены в объекте **`Range`**, обладают широкими возможностями. Например, пользователь способен получить доступ из ячейки к более чем 400 встроенным функциям Excel и вызвать функции VBA. Можно также установить связи с другими ячейками в той же самой электронной таблице, других электронных таблицах или других рабочих книгах. Гибкость и мощность объекта **`Range`** позволяют совершенствовать встроенный вычислительный блок Excel и создавать более сложные приложения для анализа данных.

1.4. Основные понятия объектно-ориентированного программирования

В основе объектно-ориентированного программирования (ООП), управляемого событиями, лежат понятия *класс, инкапсуляция, объект, наследование, полиморфизм, событие и сообщение.*

В качестве объектов могут рассматриваться конкретные предметы, а также абстрактные или реальные сущности. Например, объектами могут быть покупатель, фирма, производящая товары, банк, заказ на поставку и др.

В частном случае, в VBA *объектом* являются элементы пользовательского интерфейса, которые создаются на форме пользователя (UserForm) или на рабочем листе, а также рабочая книга и ее элементы.

Объект является комбинацией состояния и поведения. Состояние описывается переменными экземпляра, а его возможное поведение характеризуется присущими ему методами.

Каждый объект — представитель некоторого *класса* однотипных объектов, т.е. объект является экземпляром класса. Класс определяет общие для всех его объектов методы и свойства.

Методы — это программные процедуры, реализующие некоторый алгоритм, определяющий взаимодействие объектов класса с внешней средой.

Свойства представляют собой характеристики (атрибуты), присущие объектам (например, размер шрифта, название и др.).

Инкапсуляция — это сокрытие информации. При объектно-ориентированном программировании возможен доступ к объекту только через его методы и свойства. Внутренняя структура объекта скрыта от пользователя, т.е. объекты — это самостоятельные сущности, отделенные от внешнего мира. Инкапсуляция позволяет изменять реализацию объектов любого класса без опасений, что это вызовет нежелательные побочные эффекты в программной системе. Это мощное средство обеспечивает многократное использование одного и того же программного кода, позволяя собирать программу из готовых модулей, как здание из отдельных кирпичиков, но различной архитектуры и функционального назначения. *Наследование* — это

возможность выделить свойства, методы и события одного объекта и приписать их другому объекту, иногда с их модификацией. С точки зрения программиста, новый класс должен содержать только коды и данные для новых или изменяющихся методов.

Полиморфизм — это способность объектов выбирать операцию на основе данных, принимаемых в сообщении. Каждый объект может реагировать по-своему на одно и тоже сообщение. Например, команда Print будет по-разному воспринята черно-белым или цветным принтером.

В заключение следует сказать, что ООП — это технология программирования, позволившая перейти к индустриальным методам разработки программного обеспечения.

1.5. Объекты, методы, свойства, события

Объектам VBA присуща функциональность — они действуют определенным образом и могут откликаться на определенные ситуации. При этом если свойства объекта определяют его внешний вид и состояние, методы объекта определяют те задачи, которые может выполнить данный объект. Методы, по сути дела, представляют собой сегмент программного кода, внедренный в объект.

Существует определенный формат программного кода, задающего установку свойства и использование метода:

```
Объект.Свойство = Значение
Объект.Метод [Параметр1 [ . . . ]
```

Здесь **Объект** — имя настраиваемого объекта; **Свойство** — характеристика, которую нужно изменить; **Метод** — команда, которая используется для изменения объекта; **Значение** — новая установка свойства; **Параметр** — аргумент, используемый методом.

Объекты могут реагировать на события — действия пользователя или другие внешние действия, например, щелчок по кнопке, изменение текста, нажатие клавиши и др. Событие представляет собой действие, распознаваемое объектом, для которого можно запрограммировать отклик.

Иногда свойства и методы объекта оказываются связанными в том смысле, что выполнение некоторого метода приводит к изменению свойств объекта. В свою очередь изменение некоторых свойств может вызвать наступление событий. Программа может обрабатывать два основных типа событий: инициируемые пользователем и генерируемые системой. События, инициируемые пользователем, возникают в результате его действий: нажатие клавиши, щелчки кнопками мыши. Но есть события, являющиеся следствием действий пользователя.

Таким образом, любое действие пользователя может вызвать целый набор событий, и порядок их вызова может быть важным. Основные действия пользователя, генерирующие вызов событий в программе:

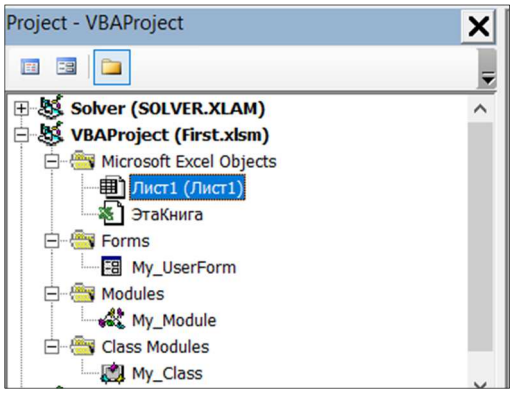
- запуск программы;
- нажатие клавиши;
- щелчок кнопкой мыши;
- перемещение мыши;
- выход из программы.

Суть программирования на VBA заключается в двух понятиях: событие и отклик на него. Если пользователь производит какое-то воздействие на систему, допустим, щелкает на кнопке, тогда в качестве отклика выполняется код созданной пользователем процедуры. Этот специальный вид процедур, генерирующих отклик на события, называется процедурами обработки событий. Если такой отклик не создан (не написана соответствующая процедура), то система никак не будет реагировать на это событие. Таким образом, задачей пользователя является разработка программного кода процедур для обработки различных событий, которые являются важными с точки зрения функционирования программы.

2. ПРОЕКТ VBA И ЕГО ЭЛЕМЕНТЫ

2.1. Структура проекта VBA

Проект — эта та часть программы, которая видима на экране при ее создании. На рисунке приведен пример проекта VBA для приложения Excel.



Как видно из рисунка, проект VBA имеет иерархическую структуру и включает: объекты Excel, формы, стандартные модули и модули класса. Объектами Excel, входящими в проект, являются рабочие книги (**`WorkBooks`**), рабочие листы (**`Worksheets`**), диаграммы (**`Charts`**). С каждым из этих объектов связан специальный модуль, в который может быть помещен программный код, выполняющий определенные действия.

В модулях класса размещается программный код, описывающий методы класса и его члены — данные для хранения значений свойств. В модулях могут быть размешены программные коды макросов, отдельно выполняемых процедур и функций.

В модулях форм записываются коды процедур обработки событий формы и элементов управления, размещенных на ней. В модулях рабочих листов помещаются процедуры обработки событий рабочих листов и элементов управления, размещаемых на рабочих листах.

Таким образом, проект включает две части: интерфейсную, т.е. видимую при выполнении программы, и программную, которая сосредоточена в различных модулях и реализует выполнение заданных действий.

Весь проект представляет собой один файл — рабочую книгу, и ***сохраняется*** вместе с ней.

2.2. Структура программы VBA

Программа VBA представляет собой совокупность процедур и функций, размещенных, в зависимости от особенностей решаемой задачи, в одном или нескольких модулях. Каждый

модуль имеет две области: общую область и область подпрограмм. В общей области помещаются операторы описания переменных, которые являются общими для всех процедур и функций этого модуля. В области подпрограмм помещается код программы.

В VBA программный код, реализующий какие-либо действия, оформляется в виде процедур и функций. Благодаря этому создаваемые программы имеют хорошую структурированность и наглядность. Разработанные отдельные функции или процедуры можно накапливать в библиотеках и в дальнейшем по мере необходимости использовать их.

2.3. Типы процедур (функций) и их определение

Различают следующие типы процедур:

- процедуры обработки событий;
- процедуры макросов;
- процедуры (функций) пользователя.

Процедуры обработки событий связаны с каким-либо объектом и имеют следующий синтаксис:

```
Private Sub ИмяОбъекта_Событие()
    «Код обработки события»
End Sub
```

где **Private**[1] — ключевое слово, определяющее область видимости подпрограммы;

Sub[2] — ключевое слово, определяющее тип подпрограммы;

ИмяОбъекта — имя объекта, с которым связывается процедура;

Событие — вид обрабатываемого события;

End Sub — ключевые слова, указывающие на окончание блока.

Например, процедура обработки события **click** для объекта Кнопка имеет вид:

```
Private Sub CommandButton1_Click()
    «Код обработки события»
End Sub
```

[1] Private – частный.

[2] Сокр. от Subroutine – подпрограмма.

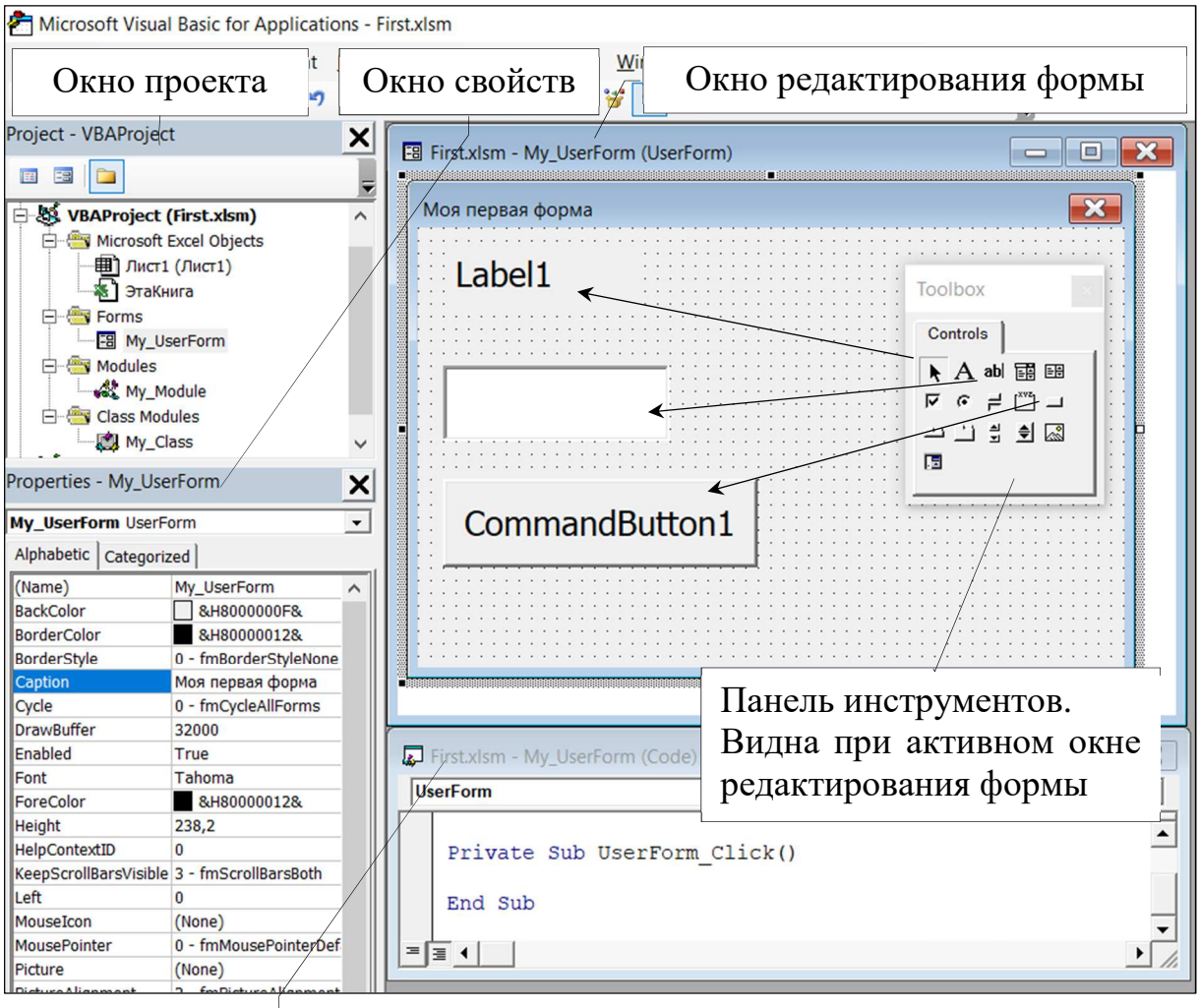
3. СРЕДА РАЗРАБОТКИ

3.1. Структура редактора VBA

Главное окно редактора обычно занимает весь экран. В окне имеются: строка заголовка, меню и панель инструментов. В строке заголовка выводится имя текущей рабочей книги. В главном окне размещаются все другие окна редактора.

3.1.1. Окно проекта

Окно проекта активизируется выполнением команды меню View > Project Window или щелчком на кнопке Project Window панели инструментов редактора. В окне проекта отображаются список проектов всех открытых рабочих книг и иерархическая структура каждого проекта.



В проекте автоматически создается модуль для каждого рабочего листа и для всей рабочей книги. Кроме того, модули создаются для каждой пользовательской формы, макросов и классов. По назначению модули бывают двух типов — модули объектов и стандартные. К стандартным относятся модули, содержащие макросы или функции. Стандартные модули добавляются в проект командой меню Insert > Module или соответствующей командой контекстного меню. Стандартный модуль также может быть создан автоматически при записи макроса.

К модулям объектов относятся модули, связанные с рабочей книгой, рабочими листами, формами, и модули класса.

Чтобы удалить какой-либо модуль, его нужно выделить в окне проекта и выполнить команду меню File > ReMove.

Благодаря тому что в окне проекта выводятся проекты всех открытых рабочих книг, можно легко копировать программные коды из одного проекта в другой.

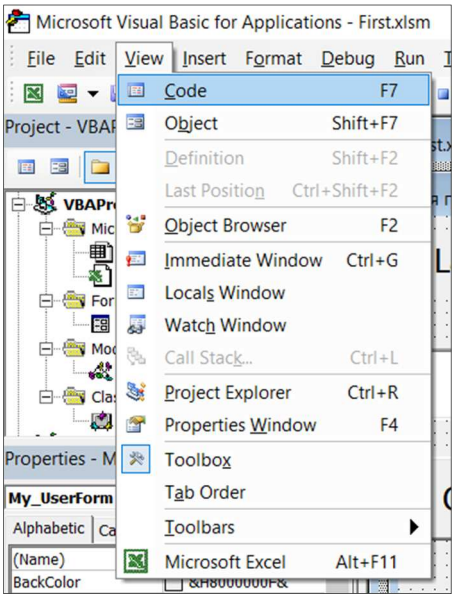
3.1.2. Окно редактирования кода

Окно редактирования кода выполняет функции текстового редактора для ввода и изменения кода процедур и функций проекта. Открыть окно редактирования кода для соответствующего модуля можно, если выполнить одно из следующих действий:

– в окне проекта сделать двойной щелчок на выбранном модуле;

– в окне проекта выделить модуль и выполнить команду меню View > Code;

– в окне проекта выделить модуль и нажать клавишу <F7>.



В окне редактирования можно просмотреть код отдельной процедуры или код всего модуля. Выбор режима просмотра окна редактирования производится одним из двух способов:

– нажатием одной из двух кнопок, размещенных в левом нижнем углу окна редактирования кода;

– установкой или снятием флажка Просмотр всего модуля (Default to Full Module View) в диалоговом окне Options, которое открывается при выполнении команды меню Сервис > Параметры (Tool > Options).

В верхней части окна редактирования кода размещены два раскрывающихся списка. Левый список содержит имена объектов, правый — перечень событий, допустимых для объекта, выбранного в левом списке.

3.1.3. Окно редактирования формы

Для создания диалоговых приложений VBA служат пользовательские формы. Пользовательская форма (UserForm) служит базой диалогового окна, на которой в зависимости от решаемой задачи размещают нужные элементы управления.

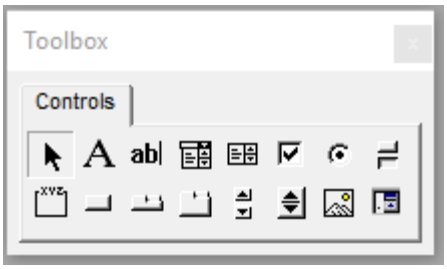
Редактор форм является основным инструментом визуального программирования. Чтобы добавить форму в проект, нужно выполнить команду меню Вставка > Форма пользователя

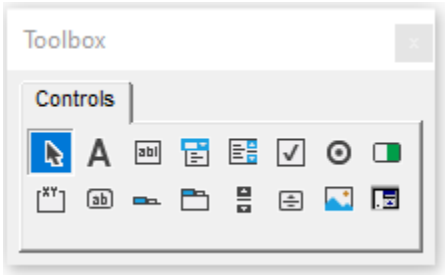
(Insert > UserForm). После этого появится пустое окно редактирования формы и панель элементов.

3.2. Панель элементов (ToolBox)

Панель элементов содержит элементы управления, которые можно поместить в форме. Открыть панель элементов можно с помощью команды Вид > Панель элементов (View > Toolbox).

В исходном состоянии на рабочей поверхности главного окна присутствует панель элементов, называемая стандартной. В разных версиях MS Office панель элементов может выглядеть по-разному, однако смысл и назначение инструментов остается прежним:





Стандартные элементы управления представлены в таблице.

		Выбор объектов			Рамка
		Надпись			Кнопка
		Поле			Набор вкладок
		Поле со списком			Набор страниц
		Список			Полоса прокрутки
		Флажок			Счетчик
		Переключатель			Рисунок
		Выключатель			RefEdit

3.3. Свойства, методы и события элементов управления

Элементы управления — это объекты, которые можно поместить в окне формы. Как все объекты, они имеют свойства и методы. Свойства элементов управления определяют их внешний вид (положение, размер, цвет) и поведение. Изменять свойства элементов управления можно как во время проектирования, так и во время выполнения программы. Метод — это процедура,

которая воздействует на объект во время выполнения. Например, для перемещения элемента управления используется метод Move.

Свойство Name определяет имя, которое используется для ссылок на элемент управления в программе. Имена должны удовлетворять условиям, предъявляемым к именам в языке VBA. Можно использовать русские буквы. Рекомендуется сразу после того, как вы поместили элемент управления в форму, изменить имя, заданное по умолчанию, на другое, отражающее назначение объекта. Если вы где-нибудь в программе используете имя элемента управления, а потом поменяете значение свойства Name, то в тексте оно не изменится. В программе может быть много элементов управления, поэтому рекомендуется давать им имена, состоящие из двух частей: префикса, определяющего тип элемента, и собственно имени.

Существуют свойства, которые для всех или для многих элементов управления называются одинаково и имеют один и тот же смысл. Эти свойства приводятся далее. В дальнейшем они не будут указываться для элементов, а будут описываться только специфические свойства каждого элемента.

3.3.1. Общие свойства элементов управления

Name	Имя, которое используется для ссылок на элемент управления в программе. Нельзя изменить во время выполнения программы
AutoSize	По умолчанию имеет значение **`False`**. Если изменить на **`True`**, то элемент будет автоматически менять свои размеры в соответствии с текстом
Left	Позиция элемента управления относительно левого края формы или рамки
Top	Позиция элемента управления относительно верхнего края формы или рамки
Height	Высота элемента управления
Width	Ширина элемента управления
Caption	Текст заголовка или надписи
Enabled	Определяет, является ли элемент управления доступным. Возможные значения **`True/False`**. Если значение свойства равно **`False`**, элемент не доступен пользователю

Visible	Определяет, будет ли элемент управления виден на экране во время выполнения программы (**True/False**). Если значение свойства равно **False**, элемент не виден на экране
TabIndex	Определяет порядок перемещения от объекта к объекту с помощью клавиш <Tab> или <Shift> + <Tab>
BorderStyle	Определяет стиль обрамления
BackColor	Определяет цвет фона
BorderColor	Определяет цвет рамки
ForeColor	Определяет цвет шрифта
Font	Используется для установки параметров шрифта

3.3.2. Общие методы стандартных элементов управления

SetFocus	Передает фокус объекту
Drag	Служит для перетаскивания элемента управления по технологии Drag&Drop (перетащить и отпустить)
Move	С помощью этого метода элемент управления перемещают по форме, при этом можно изменять его размеры

3.3.3. Элемент Кнопка (CommandButton)

Элемент Кнопка (CommandButton) очень часто используется при разработке интерфейса. На поверхности кнопки можно разместить надпись или рисунок, или и то и другое. Далее описываются основные свойства и события элемента.

Свойства элемента Кнопка

Cancel	Только одна из кнопок формы может иметь значение этого свойства **True**. Это должна быть кнопка, выполняющая функции кнопки Cancel (Отмена). Если форма активна и пользователь нажмет <Esc>, то будут выполнены действия, связанные с этой кнопкой
Default	Только одна из кнопок формы может иметь значение этого свойства **True**. Если ни одна из кнопок формы не находится в фокусе и пользователь нажал <Enter>, то будут выполнены действия, связанные с этой кнопкой
Picture	Определяет рисунок на поверхности кнопки

События элемента Кнопка

Click	Возникает при нажатии пользователем кнопки мышью или на клавиатуре
DblClick	Возникает при двойном щелчке мыши на кнопке

3.3.4. Элемент Поле (TextBox)

Элемент Поле (TextBox) обеспечивает возможность ввода текста пользователем. Текстовые окна поддерживают ввод и редактирование текста без всякого вмешательства с вашей стороны. Вырезать, копировать и вставлять текст можно с помощью стандартных для Windows клавиш: <Ctrl> + <X>, <Ctrl> + <C>, <Ctrl> + <V>. Ниже описываются основные свойства элемента.

Свойства элемента Поле

Text	Содержит текст. Тип значения **`String`**. Если вы используете элемент для ввода чисел или дат, то это значение нужно преобразовать в значение соответствующего типа, используя функции **`Val, Cdate, CInt, CDbl`** и др.
Value	Возвращает или задает значение типа **`Variant`**.
SelText	Содержит выделенный в поле текст.
Locked	При Locked = **`True`** пользователь не может ввести текст в поле.
Multiline	При MultiLine = **`True`** поле может содержать более одной строки.
ScrollBars	Определяет наличие полос прокрутки при многострочном режиме.
PasswordChar	Если текстовое поле используется для введения пароля, то в это свойство записывается замещающий символ. Если свойство содержит пустую строку, то режим обычный.

3.3.5. Элемент Надпись (Label)

Элемент Надпись (Label) обычно используется для вывода различных текстов в форме. Он может содержать и рисунок. Пользователь не может изменить надпись, но программа во время выполнения может изменять значение надписей.

Свойства элемента Надпись

Caption	Содержит текст надписи.
Picture	Определяет рисунок, помещаемый в элемент.
BackStyle	Задает режим для фона.

4. ОСНОВНЫЕ ПОНЯТИЯ ЯЗЫКА VBA

4.1. Инструкции

Инструкция Visual Basic является полной командой. Она может содержать ключевые слова, операторы, переменные, константы и выражения. Любая инструкция относится к одной из следующих категорий:

– инструкции описания, именующие переменные, константы или процедуры, а также могут задавать типы данных;

– инструкции присвоения, которые присваивают значение переменной или константе;

– исполняемые инструкции, инициирующие действие. Они могут выполнить метод или функцию, а также могут организовать повторение или ветвление блоков программы. Эти инструкции часто содержат математические или условные операторы.

Если несколько инструкций располагаются в одной строке, то они разделяются двоеточием (:).

Продолжение инструкций на несколько строк.

Обычно инструкция располагается в одной строке, но ее можно продолжить на следующую строку с помощью символа продолжения строки (_) — знака подчеркивания.

Комментарии

Комментарии позволяют объяснить процедуру или определенную инструкцию всем читателям программы. Когда процедура выполняется, Visual Basic игнорирует комментарии. Строки комментария начинаются с апострофа (') или со слова **Rem**, за которым следует пробел. Их можно вносить в любое место процедуры. Чтобы внести комментарий в строку, на которой расположена инструкция, после инструкции следует поставить апостроф, а за ним комментарий. По умолчанию комментарии выделяются в тексте зеленым цветом.

```
' Это комментарий
Rem Это тоже комментарий
```

Проверка синтаксических ошибок

Если после набора строки и нажатия клавиши ENTER строка выделяется красным цветом (может появиться также сообщение об ошибке), необходимо выяснить, что неправильно в данной инструкции, и исправить ее.

4.2. Имена и идентификаторы

Идентификатор — весьма важное и употребительное понятие. Этот термин происходит от слова «идентифицировать», т.е. «отождествлять». Поскольку алгоритм, определяющий процесс обработки данных, оперирует с различными программными объектами — переменными величинами, функциями и т.д., то при записи алгоритма приходится ссылаться на используемые объекты. Для этой цели программным объектам даются индивидуальные имена, которые и представляют соответствующие объекты. Именами обозначаются и некоторые атрибуты используемых объектов, например, тип значений, которые могут принимать программные объекты. Роль таких имен и выполняют идентификаторы.

При присвоении имен процедурам, константам, переменным, и аргументам в модуле Visual Basic используются следующие правила:

– имена должны начинаться с буквы;

– имя не может содержать пробел, точку (.), восклицательный знак (!) или символы @, &, $, #;

– имена не должны содержать более 255 символов;

– как правило, не следует использовать имена, совпадающие с названиями функций, инструкций и методов языка Visual Basic, так как при этом прекращается выделение в тексте одноименных ключевых слов языка. Чтобы использовать встроенные функции языка, инструкции или методы, имена которых конфликтуют с присвоенным пользователем именем, их необходимо явно указывать. Для этого перед именем встроенной функции, инструкции или метода должно стоять имя связанной с

ними библиотеки типов. Например, если имеется переменная с именем Left, то функция Left должна вызываться как VBA.Left;

– не допускается использование повторяющихся имен на одном уровне области определения. Нельзя, например, описать две переменные с именем **age** в одной процедуре. Однако описание личной переменной **age** и переменной уровня процедуры **age** внутри одного модуля допустимо.

В языке Visual Basic не различаются строчные и прописные буквы, однако в инструкции описания сохраняются прописные буквы.

Чтобы иметь более наглядные имена, целесообразно использовать строчные и прописные буквы, знак подчеркивания «_»: Kol_Iter.

4.3. Время жизни переменной

Временем жизни переменной называется время, в течение которого переменная может иметь значение. Значение переменной может меняться на протяжении ее времени жизни, но в течение этого времени она обязательно имеет какое-либо значение. Когда переменная теряет область определения, она более не имеет значения.

В начале выполнения процедуры все переменные инициализируются. Числовая переменная получает значение 0, строка переменной длины получает значение пустой строки (“”), а строка фиксированной длины заполняется ASCII символом 0 или Chr(0). Переменные типа **Variant** получают при инициализации значение **Empty**. Каждый элемент массива переменных с определяемым пользователем типом при инициализации получает значение, которое он получил бы, если бы являлся одиночной переменной.

При описании объектной переменной для нее выделяется память, но ее значение определяется как **Nothing** до тех пор, пока ей не присвоена ссылка на объект с помощью инструкции **Set**.

Если значение переменной не изменяется во время выполнения программы, она сохраняет значение, полученное при инициализации, до тех пор, пока не потеряет область определения.

Переменная, описанная с помощью инструкции **Dim** на уровне процедуры, сохраняет значение до окончания выполнения процедуры. Если процедура вызывает другие процедуры, переменная сохраняет свое значение, пока не закончится выполнение и этих процедур.

Если переменная уровня процедуры описана с помощью ключевого слова **Static**, она сохраняет свое значение до тех пор, пока программа выполняется в каком-либо модуле. По завершении работы всей программы переменная теряет свою область определения и свое значение. Ее время жизни совпадает со временем жизни переменной уровня модуля.

Если перед инструкциями **Sub** или **Function** имеется ключевое слово **Static**, значения всех переменных уровня процедуры сохраняются между вызовами процедуры.

4.4. Описания

Описание — важное понятие языка (впрочем, суть этого понятия точнее отражал бы термин *объявление*). Необходимость этого понятия связана с тем, что операторы задают правила обработки данных, т.е. определяют действия над программными объектами, а прежде чем задавать такие действия, программист должен ввести в употребление нужные ему программные объекты и точно определить необходимые атрибуты (свойства) каждого из них. Если, например, это массив, то надо указать его размерность, размеры по каждому измерению, а также указать, что представляют собой его элементы.

Для введения в употребление нужных программных объектов, описания их атрибутов, присваивания имен объектам, а также для некоторых других целей и служат описания.

4.4.1. Инструкция Dim

При описании переменных обычно используется инструкция **Dim**[1]. Для создания переменной на уровне процедуры инструкция описания помещается внутри процедуры. Чтобы создать

[1] Сокр. от Dimension — размерность.

переменную на уровне модуля, инструкция описания располагается в начале модуля, в разделе описаний.

В следующем примере создается переменная **strName** и задается тип данных **String**.

```
Dim strName As String
```

Когда эта инструкция располагается в процедуре, переменная **strName** может использоваться только в данной процедуре. Если же такая инструкция находится в разделе описаний модуля, то переменная **strName** доступна для всех процедур данного модуля, но не может использоваться процедурами из других модулей проекта. Чтобы сделать переменную доступной для всех процедур проекта, перед ней надо поставить инструкцию **Public**:

```
Public strName As String
```

Переменные могут описываться как один из следующих типов данных: **Boolean, Byte, Integer, Long, Currency, Single, Double, Date, String** (для строк переменной длины), **String * длина** (для строк фиксированной длины), **Object**, или **Variant**. Если тип данных не задан, по умолчанию переменная приобретает тип **Variant**. Имеется также возможность создать определяемый пользователем тип данных с помощью инструкции **Type**.

Допускается также описание нескольких переменных в одной строке. В этом случае, чтобы задать тип данных, надо указать определенный тип для каждой переменной. В следующем примере переменные **X, Y,** и **Z** описываются как **Integer**.

```
Dim X As Integer, Y As Integer, Z As Integer
```

В следующей строке **X** и **Y** описываются как **Variant**; и только **Z** описывается как **Integer**.

```
Dim X, Y, Z As Integer
```

Можно не указывать тип данных переменной в описании. Если тип данных не указан, переменная приобретает тип **Variant**.

Тип переменных и констант можно задавать суффиксом:

% — Integer, & — Long, ! — Single, # — Double, @ — Currency, $ — String.

Например:

10%, Num% — константа и переменная целого типа;

Res_Str$ —переменная строкового типа;

99#, Res_Real# — константа и переменная вещественного типа.

Такой подход является устаревшим и к использованию не рекомендуется.

4.4.2. Инструкция Public

Инструкция **Public** используется для описания общих переменных на уровне модуля.

```
Public strName As String
```

Общие переменные могут использоваться в любой процедуре проекта. Если общая переменная описана в стандартном модуле или в модуле класса, она также может использоваться в любом проекте, в котором имеется ссылка на проект, где описана эта переменная.

4.4.3. Инструкция Private

Инструкция **Private** используется для описания личных переменных уровня модуля.

```
Private MyName As String
```

Личные переменные доступны только для процедур одного и того же модуля.

На уровне модуля инструкция **Dim** эквивалентна инструкции **Private**. Использование инструкции **Private** может упростить чтение и отладку программы.

4.4.4. Инструкция Static

Переменные, описанные с помощью инструкции **Static** вместо инструкции **Dim**, сохраняют свои значения при выполнении программы.

4.4.5. Инструкция Option Explicit

В языке Visual Basic можно неявно описать переменную, просто используя ее в инструкции присвоения. Все неявно описанные переменные имеют тип **`Variant`**. Переменные типа **`Variant`** более требовательны к ресурсам памяти, чем большинство других переменных. Программа будет более эффективной, если переменные явно описаны с определенным типом данных. Явное описание всех переменных уменьшает вероятность конфликтов имен и ошибок, связанных с опечатками.

Если неявные описания нежелательны, инструкция **`Option Explicit`** должна предшествовать в модуле всем процедурам. Эта инструкция налагает требование явного описания всех переменных этого модуля. Если модуль содержит инструкцию **`Option Explicit`**, при попытке использования неописанного или неверно введенного имени переменной возникает ошибка во время компиляции.

4.5. Переменная

При синтаксических определениях ряда понятий языка, в том числе операторов и описаний, часто используется понятие *переменная.* Переменная — программный объект, способный принимать значение. Это значение переменная получает уже в процессе выполнения программы, обычно в результате выполнения оператора присваивания. Присвоенное значение переменная охраняет до тех пор, пока ей не будет присвоено новое текущее значение — при этом предыдущее ее значение (если оно было определено) безвозвратно теряется. С каждой переменной связывается определенный тип значений, которые она может принимать. Попытка присвоить переменной значение иного типа квалифицируется как ошибка в программе.

С точки зрения синтаксиса, переменная (в простейшем случае) — это идентификатор, который сопоставлен переменной в качестве ее имени. Это имя используется для ссылки на значение переменной. Другими словами, имя в тексте программы представляет значение этой переменной.

Что касается семантики понятия (переменная), то можно считать, что в вычислительной системе имеется несколько типов «запоминающих ячеек», каждая из которых способна хранить значения определенного типа. К началу выполнения программы каждой из используемых в ней переменных выделяется ячейка соответствующего типа и этой ячейке дается имя, совпадающее с именем самой переменной.

С алгоритмической точки зрения весьма важно такое действие, как присваивание переменной некоторого значения. Удобно считать, что выполнение этого действия означает помещение присваиваемого переменной значения в выделенную для нее ячейку. При описании процессов обработки данных часто приходится присваивать переменной величине ее новое текущее значение. Поскольку это действие встречается особенно часто, то для его обозначения используется символ « **=** ». Например, запись **z = x + y** означает, что переменной *z* должно быть присвоено новое текущее значение, равное значению **x + y**.

4.6. ФУНКЦИИ И ПРОЦЕДУРЫ

Понятие *функция* хорошо известно из школьного курса математики. С помощью функций задаются самые различные зависимости одних значений (значений функции) от других значений (аргументов функции). Таким образом, в алгоритмическом языке любая функция задается некоторой процедурой, выполнение которой и дает значение функции.

5. ТИПЫ ДАННЫХ

Под типом данных понимается множество *значений,* которые может принимать переменная и, как следствие, множество *операций,* допустимых над данной переменной.

VBA допускает использование переменных, тип которых не описан. Неописанные переменные приобретают тип данных **Variant**.

Variant является особым типом данных. Переменные этого типа могут содержать любые данные, за исключением строк (тип **String**) фиксированной длины и определяемых пользователем типов. Переменная типа **Variant** может также содержать специальные значения **Empty**, **Error**, **Nothing** и **Null**.

Этот тип данных упрощает написание программ, но его использование не всегда является эффективным.

Следует предусмотреть применение других типов данных, если:

– программа имеет большой размер и очень много переменных;

– требуется как можно более быстрое выполнение программы;

– выполняется прямая запись данных в файлы с произвольным доступом.

Кроме типа **Variant** поддерживаются типы данных, размеры, требуемые для сохранения значений и диапазоны допустимых значений (см. таблицу).

Тип данных	Размер	Диапазон значений
Целый		
Byte (байт)	1 байт	0 .. 255.
Integer (целое)	2 байт	–32 768 .. 32 767.
Long (длинное целое)	4 байт	–2 147 483 648 .. 2 147 483 647.
Decimal	14 байт	±79 228 162 514 264 337 593 543 950 335 без дробн. части; ±7,9228162514264337593543950335 с 28 знаками справа от запятой.
Вещественный (с плавающей точкой)		
Single	4 байт	От –3,402823E38 до –1,401298E-45 для отрицательных значений; от 1,401298E-45 до 3,402823E38 для положительных значений.

Тип данных	Размер	Диапазон значений
Double (двойной точности)	8 байт	От –1,79769313486232E308 до – 4,94065645841247E-324 для отрицательных значений; от 4,94065645841247E-324 до 1,79769313486232E308 для положительных значений.
Логический		
Boolean	2 байт	**True** (истина) или **False** (ложь).
Строковый		
String	Длина строки	От 1 до приблизительно 65 400 символов.
Денежный		
Currency	8 байт	От -922 337 203 685 477,5808 до 922 337 203 685 477,5807.
Дата и время		
Date	8 байт	От 1 января 100 г. до 31 декабря 9999 г.

Константа вещественного типа 1,234E2 имеет значение $1{,}234 \cdot 10^2$, т.е. 123,4.

5.1. Операции и функции для данных целого типа

Символом описания типа для **Integer** служит символ (%); для **Long** — (&).

Для данных целого типа определены следующие операции:

+, -, *, ^ \, Mod.

Первые четыре — общепринятые и в комментариях не нуждаются.

Операция ^ — возведение в степень: 3^2 = 9.

Операция \ — целочисленное деление: в качестве результата принимается частное от деления, а остаток игнорируется.

Например: 7 \ 2 = 3; 3 \ 5 = 0; (–7) \ 2 = –3.

Операция **Mod** — остаток от деления. m **Mod** n определено только для n>0.

При m>0, m **Mod** n = m – ((m \ n)*n).

m<0, m **Mod** n= m – ((m \ n)*n)+n.

Например: 7 Mod 2 = 1; 3 Mod 5 = 3; (–7) Mod 2 = 1.

При вычислении арифметических выражений используется общепринятый приоритет: сначала выполняется операция (^), затем —операции типа «умножения» (*, \, **Mod**) и в последнюю

очередь — типа «сложения» (+, –). Как обычно, скобки могут изменить порядок выполнения операций.

5.2. Операции и функции для данных вещественного типа

Следующие операции дают вещественный результат, если хотя бы один аргумент является значением вещественного типа (другой может быть и целого типа):

+, –, *, /, возведение в степень (^).

Для операции деления заметим, что результат будет вещественного типа и для двух целочисленных аргументов. Все эти операции имеют общепринятый приоритет.

Перечислим *стандартные* функции:

Abs(x) ≡ $|x|$;

Sin(x) ≡ синус ($sin\ x$);

Cos(x) ≡ косинус ($cos\ x$);

Tan (x) ≡ тангенс ($tg\ x$);

Atn (x) ≡ арктангенс ($arctg\ x$);

Log(x) ≡ натуральный логарифм ($ln\ x = log_e x$);

Exp(x) ≡ экспонента (e^x);

Sqr[1](x) ≡ квадратный корень ($\sqrt{x}$).

Int(x), **Fix**(x) отбрасывают дробную часть числа и возвращают целое значение.

Различие между функциями **Int** и **Fix** состоит в том, что для отрицательного значения аргумента число функция **Int** возвращает ближайшее отрицательное целое число, меньшее либо равное указанному, а **Fix** ближайшее отрицательное целое число, большее либо равное указанному. Например, функция **Int** преобразует –8.4 в – 9, а функция **Fix** преобразует –8,4 в –8.

Sgn[2](x) возвращает знак числа x (т.е. ±1 или 0).

[1] Sqr – сокр. от Square root (англ.) – квадратный корень.

[2] Sgn – сокр. от Signum – знак.

Rnd[1][(x)] возвращает значение типа **Single**, содержащее случайное число.

Необязательный аргумент **x** представляет значение типа **Single** или любое допустимое числовое выражение.

Rnd возвращает значение из интервала [0, 1). **x** определяет способ генерации случайного числа функцией **Rnd**. Перед вызовом функции **Rnd** рекомендуется использовать инструкцию **Randomize** для инициализации генератора случайных чисел значением, возвращаемым системным таймером.

Далее приведена формула, предназначенная для получения случайных целых чисел в заданном диапазоне [a, b]:

```
Int((b – a) * Rnd + a)
```

Примеры вещественных констант:
3.1415926; -543.0123; 2.3E12=2.3·1012; 62.376E-24=62.376·10-24.

5.3. Операции и функции для данных логического типа

Переменные логического, или булевского[2], типа занимают в памяти один байт и могут принимать два значения: **True** (истина) и **False** (ложь).

Результатом операций отношения (=, ≠, <, ≤, >, ≥) будет одна из булевских констант. Нестрогие неравенства записываются парой символов, причем (≥) записывается как (>=), (≤) – (<=), (≠) – (< >).

Над значениями булевского типа допустимы операции сравнения, причем считается, что **False**<**True**.

Имеются четыре стандартные логические операции:

And — логическое умножение (конъюнкция — &);

Or — логическое сложение (дизъюнкция — ∨);

Xor — исключающее «или» (сложение по модулю 2 — ⊕);

---

[1] Rnd – сокр. от Random – случайный.

[2] Дж. Буль (Bool) — английский математик, заложивший в середине XIX в. основы формальной логики.

Eqv — логическая эквиваленция (≡);

Imp — логическая импликация (→);

Not — отрицание.

Результаты этих операций приведены в таблице.

A	B	A And B	A Or B	A Xor B	A Eqv B	A Imp B	Not A
False	False	False	False	False	True	True	True
False	True	False	True	True	False	True	True
True	False	False	True	True	False	False	False
True	True	True	True	False	True	True	False

Логические операции используются для формирования достаточно сложных логических выражений. Так условие принадлежности x интервалу

$x \in [a, b)$ есть **a <= x And x < b**

$x \in (-\infty, a) \cup [b, \infty)$ есть **x < a Or x >= b**

5.4. Операции и функции для строковых данных (String)

Существует два типа строковых значений:

– строки переменной длины, которые могут содержать до приблизительно 2 млрд (2^31) символов;

– строки постоянной длины, которые могут содержать от 1 до приблизительно 64K (2^16) символов.

Кодами для символов, образующих значения типа **String**[1], служат целые числа в диапазоне от 0 до 255. Первые 128 символов (0–127) набора символов соответствуют буквам и символам стандартной американской клавиатуры. Эти первые 128 символов совпадают с набором символов ASCII[2]. Следующие 128 символов (128–255) представляют буквы национальных алфавитов, буквы с надстрочными символами, символы денежной единицы и дроби. Символом описания типа для типа **String** служит символ ($).

[1] String – строка.

[2] ASCII – аббревиатура от American Standart Code for Information Interchange – Американский стандартный код для обмена информацией.

5.4.1. Функции, результаты которых имеют числовой тип

Val[1](строка) возвращает числа, содержащиеся в строке, как числовое значение соответствующего типа. Обязательный аргумент строка является любым допустимым строковым выражением.

Функция **Val** прекращает чтение строки на первом символе, который она не может распознать в качестве части числа. Символы, которые часто рассматриваются в качестве частей числовых значений, типа знака доллара и запятых, не распознаются. Однако эта функция распознает префикс основания &O (для восьмеричных) и &H (для шестнадцатеричных значений). Пробелы, символы табуляции и символы перевода строк удаляются из значения аргумента. Функция **Val** распознает в качестве разделителя целой и дробной части только **точку** (.).

```
Dim MyValue
MyValue = Val(«2457»)          ' Возвращает 2457
MyValue = Val(« 2 45 7»)       ' Возвращает 2457
MyValue = Val(«24 and 57»)     ' Возвращает 24
```

Len[2](строка) возвращает значение типа **Long**, содержащее число символов в строке или число байт, необходимое для размещения переменной.

Эквивалентной функцией обработки строк по байтам является функция **LenB**, которая возвращает число байт, используемое для представления указанной строки, а не число символов в строке. Для определяемых пользователем типов функция **LenB** возвращает размер, занимаемый в памяти, включая заполнение промежутков между элементами.

Asc[3](строка) возвращает **Integer**, представляющее код первого символа строки.

```
Dim MyValue
MyValue = Asc("A")         ' Возвращает 65
MyValue = Asc("a")         ' Возвращает 97
```

[1] Value – величина.
[2] Length – длина.
[3] Asc – сокращение от ASCII.

```
MyValue = Asc("Apple")    ' Возвращает 65
```

Допускается сравнение строк, которое происходит посимвольно, начиная от первого символа в строке. Строки равны, когда имеют одинаковую длину и посимвольно эквивалентны:

```
"abcd" = "abcd"   True,
"abcd" <> "abcde" True,
"abcd" <> "abcd"  True.
```

При посимвольном сравнении может один символ оказаться больше другого (его код больше), значит, и строка, его содержащая, считается большей. Остатки строк и их длины не играют роли. Любой символ всегда больше «пустого места»:

```
"abcd" > "abcD"   (так как "d" > "D" ),
"abcd" > "abc"    (так как "d" > "" ),
"aBcd"" < "ab"    (так как "B" < "b" ),
" " > ""          (так как «пробел» (код #32) >"") .
```

Можно, конечно, использовать нестрогие отношения: >= и <=.

5.4.2. Функции, результаты которых имеют тип String

Некоторые функции имеют по две версии, одна из которых возвращает тип данных **Variant**, а другая — тип данных **String**. Первая версия является более удобной, так как при этом для значений типа **Variant** преобразование типов данных выполняется автоматически. В ней также допускается передача значения **Null** через выражение. Вторая версия, возвращающая тип **String**, более эффективна, так как она использует меньше памяти.

Следующие функции возвращают значения типа **String**, если к имени функции добавляется символ доллара ($). Эти функции имеют такое же применение и синтаксис, как и их эквиваленты без символа доллара, возвращающие тип **Variant**.

Операции «**+**» или «**&**» используются для слияния (склеивания) строк. При этом операция «**+**» выполняется только для данных типа **String**.

При выполнении операции «**&**» происходит преобразование к типу **String**, если выражение не содержит строкового значения.

```
Dim S, S1, S2
```

```
S1 = "Склеивание"
S2 = " строк"
S = S1 + S2       ' Возвращает «Склеивание строк»
S = S1 & S2       ' Возвращает «Склеивание строк»
S = S1 & 2 & S2   ' Возвращает «Склеивание 2 строк»
```

Str(число) возвращает значение типа **Variant** (**String**), являющееся строковым представлением числа. Обязательный аргумент число имеет тип **Long** и может задаваться любым допустимым числовым выражением. При преобразовании числа в строку в начале строки обязательно резервируется позиция для знака числа. Если число является положительным, возвращенная строка будет содержать пробел на месте знака. В качестве допустимого десятичного разделителя функция **Str** воспринимает только **точку** (.).

```
Dim MyString
MyString = Str(459)         ' Возвращает « 459»
MyString = Str(-459.65)     ' Возвращает «-459.65»
MyString = Str(459.001)     ' Возвращает « 459.001»
```

Mid[1](строка, n1[, n2]) возвращает значение типа **Variant** (**String**), содержащее указанное число символов строки. n1 — позиция символа в строке строка, с которого начинается нужная подстрока. Если n1 больше числа символов в строке строка, функция **Mid** возвращает пустую строку. n2 — количество возвращаемых символов. Если этот аргумент опущен или превышает число символов, расположенных справа от позиции n1, то возвращаются все символы от позиции n1 до конца строки.

```
Dim MyString, FirstWord, LastWord, MidWords
MyString = "Пример функции Mid"'Инициализирует строку
FirstWord = Mid(MyString, 1, 6)'Возвращает «Пример»
LastWord = Mid(MyString, 16, 3)'Возвращает «Mid»
MidWords = Mid(MyString, 8) 'Возвращает »функции Mid»
```

[1] Mid – середина.

Left[1](строка, n), **Right**[2](строка, n) возвращают значения типа **Variant** (**String**), содержащие указанное число n первых символов **(Left)** или n последних символов **(Right)** заданной строки. Если n = 0, возвращается пустая строка (""). Если значение n больше либо равняется числу символов в строке, возвращается вся строка.

Эквивалентными функциями обработки строк по байтам являются функции **LeftB** и **RightB**. В этом случае n указывает число байт (а не символов), которые следует вернуть.

```
Dim AnyString, MyStr
AnyString = "Всем привет"   ' Задает строку
MyStr = Left(AnyString, 1)  ' Возвращает «В»
MyStr = Left(AnyString, 6)  ' Возвращает «Всем п»
MyStr = Left(AnyString, 20) ' Возвращает «Всем привет»
MyStr = Right(AnyString, 1) ' Возвращает «т»
MyStr = Right(AnyString, 7) ' Возвращает « привет»
MyStr = Right(AnyString, 20)' Возвращает «Всем привет»
```

String(n, символ) возвращает значение типа **Variant** (**String**), содержащее повторяющийся символ n раз.

В данном примере функция **String** используется для генерации строк, содержащих указанное число повторяющихся символов.

```
Dim MyString
MyString = String(5, "*")    ' Возвращает «*****»
MyString = String(5, 42)     ' Возвращает «*****»
MyString = String(10, "ABC") 'Возвращает «AAAAAAAAAA»
```

Chr[3](код) возвращает значение типа **String**, содержащее символ, соответствующий указанному коду символа.

Date, **Time** возвращает значение типа **Variant** (**Date**), содержащее текущую системную дату или время.

---

[1] Left – левый, слева.
[2] Right – правый, справа.
[3] Chr – сокр. от Character – знак, символ.

LTrim(строка) **RTrim**(строка) **Trim**[1](строка)
Возвращают значение типа **Variant** (**String**), содержащее копию строки, из которой удалены пробелы, находившиеся в начале строки (**LTrim**) , в конце строки (**RTrim**) или в начале и конце строки (**Trim**).

UCase(строка) возвращает значение типа **Variant** (**String**), содержащее строку, преобразованную к верхнему регистру. К верхнему регистру преобразуются только строчные буквы; прописные буквы и прочие символы остаются неизменными.

5.5. Переменные типа денежные значения (Currency)

Переменные типа **Currency** (денежные значения) сохраняются как 64-разрядные (8-байтовые) целые числа, которые после деления на 10000 дают число с фиксированной десятичной точкой с 15 разрядами в целой части и 4 разрядами в дробной. Такое представление позволяет отобразить числа в диапазоне от – 922 337 203 685 477,5808 до 922 337 203 685 477,5807. Символом описания типа для типа **Currency** служит символ (@).

Тип данных **Currency** используется для денежных расчетов, а также для проведения расчетов с фиксированной десятичной точкой, в которых требуется обеспечить высокую точность.

6. ОПЕРАТОРЫ ЯЗЫКА VBA

6.1. Оператор присваивания

Процесс решения задачи распадается на ряд последовательно выполняемых этапов, на каждом из которых по некоторым значениям, известным к началу выполнения этого этапа, вычисляется новое значение.

Для задания правил вычисления новых значений в VBA служит такое понятие, как *выражение,* причем каждое выражение задает правила вычисления только одного значения, вычисленное же значение необходимо запомнить для его использования на

[1] Trim – сокращать, урезать.

последующих этапах вычислительного процесса. Такое запоминание достигается путем присваивания вычисленного значения некоторой переменной с помощью *оператора присваивания,* который относится к числу основных операторов. Синтаксически оператор присваивания определяется следующим образом:

A = B, здесь **A** – некоторая переменная, **B** – выражение.

Основной символ « = » обозначает операцию присваивания и читается как «присвоить значение». Этот символ не следует путать с символом « = », обозначающим операцию отношения (сравнения). Выполнение оператора присваивания сводится к вычислению значения выражения, заданного справа от символа « = » с последующим его присваиванием переменной, указанной слева от этого символа. Таким образом, оператор присваивания определяет некоторый самостоятельный, логически завершенный этап вычислительного процесса: в результате выполнения оператора присваивания некоторая переменная принимает новое текущее значение, доступное для последующего его использования (при этом предыдущее значение этой переменной безвозвратно теряется).

```
Dim a, b, c
a = 3: b = 5: c = a + b   ' c получает значение 8
```

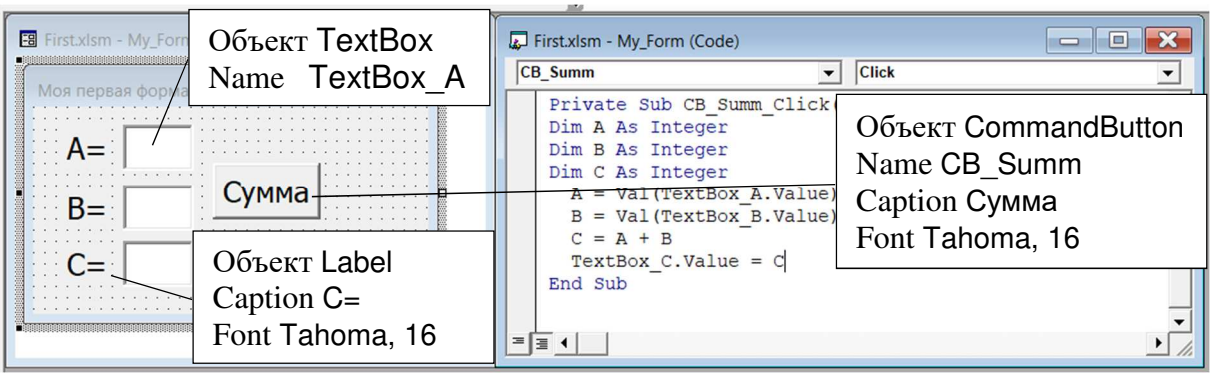
6.2. Ввод и вывод данных

Для обмена данными будем использовать элемент управления TextBox. Свойство этого элемента Text или Value (типа **String**) можно использовать как для ввода данных, так и для получения результата.

В качестве примера рассмотрим получение суммы **C** двух переменных **A** и **B**.

На форме поместим три элемента управления TextBox и кнопку CommandButton. Меняя свойство Name каждого из них, определим их имена – соответственно TextBox_A, TextBox_B, TextBox_C и CB_Summ. Изменение имен необязательно, но использование таких имен позволит без труда определить назначение каждого из элементов. Поместив на форму элементы

Надпись (Label), можно «подписать» соответствующие поля (Свойство Caption).

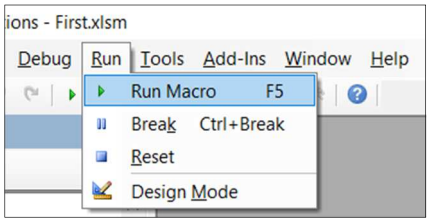


По двойному щелчку на кнопке **CB_Summ**, автоматически создается подпрограмма обработки события нажатия кнопки **CB_Summ_Click**, куда мы вписываем соответствующие описания преобразования переменных.

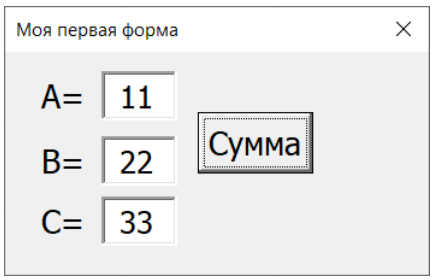
```
Private Sub CB_Summ_Click()
Dim A As Integer
Dim B As Integer
Dim C As Integer
' Значение из TextBox_A преобразуется в число и
' присваивается переменной A
  A = Val(TextBox_A.Value)1

' Значение из TextBox_B преобразуется в число и
' присваивается переменной B
  B = Val(TextBox_B.Value)
  C = A + B                ' Вычисление суммы
' Результат C присваивается TextBox_C
  TextBox_C.Value = C
End Sub
```

Для запуска программы на выполнение нужно в основном меню редактора VBA выбрать пункт Run или нажать функциональную клавишу F5.

[1] После нажатия [CTRL + пробел] редактор VBA выводит список всех доступных переменных и объектов. Поэтому полностью вводить текст **TexBox_A** необязательно. Достаточно нажать [CTRL + пробел] и первые буквы имени. Затем из выпадающего списка выбрать нужный объект.



После ввода данных и нажатия кнопки [Сумма] получим



6.2.1. Функция InputBox

Выводит на экран диалоговое окно, содержащее сообщение и поле ввода, устанавливает режим ожидания ввода текста пользователем или нажатия кнопки, а затем возвращает значение типа **`String`**, содержащее текст, введенный в поле. Используются следующие параметры функции **`InputBox`**[1]. В квадратных скобках перечисляются необязательные параметры.

`InputBox`(сообщение [,заголовок окна] [, значение] [, xpos] [, ypos] [, helpfile, context])

Здесь:

сообщение -- строка, отображаемая как сообщение в диалоговом окне. Сообщение может состоять из нескольких строк. Для разделения строк допускается использование символа возврата каретки (Chr(13)), символа перевода строки (Chr(10)) или комбинации этих символов.

заголовок окна -- строка, отображаемая в заголовке диалогового окна. Если этот аргумент опущен, в строку заголовка помещается имя приложения.

значение -- строковое выражение, отображаемое в поле ввода как используемое по умолчанию, если пользователь не введет другую строку. Если этот аргумент опущен, поле ввода изображается пустым.

[1] Input box – поле ввода.

xpos -- горизонтальная координата диалогового окна. Если этот аргумент опущен, диалоговое окно выравнивается по центру экрана по горизонтали.

ypos -- вертикальная координата диалогового окна. Если этого аргумента нет, то окно помещается по вертикали примерно на одну треть высоты экрана.

helpfile -- имя файла справки, содержащего справочные сведения о данном диалоговом окне. Если этот аргумент указан, необходимо указать также аргумент context.

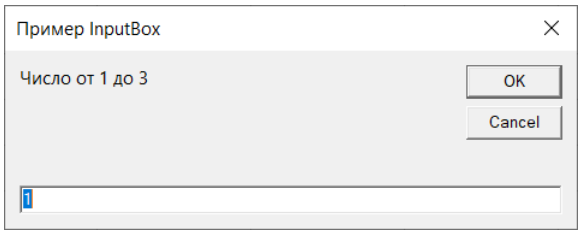
context -- числовое выражение, определяющее номер соответствующего раздела справочной системы. Если этот аргумент указан, необходимо указать также аргумент helpfile.

Если указаны оба аргумента, helpfile и context, пользователь имеет возможность нажатием клавиши F1 вызвать контекстную справку. Некоторые главные приложения, например Microsoft Excel, также автоматически добавляют в диалоговое окно кнопку «Справка». Если пользователь нажимает кнопку «OK» или клавишу ENTER, функция **InputBox** возвращает содержимое поля ввода. Если пользователь нажимает кнопку «Отмена», функция возвратит пустую строку ("").

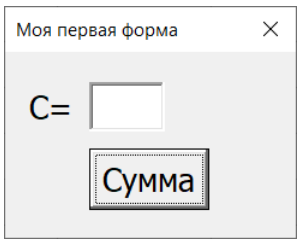
Функцию **InputBox** с двумя или большим числом аргументов можно использовать только в выражении. Наличие запятых, соответствующих отсутствующим аргументам, является обязательным.

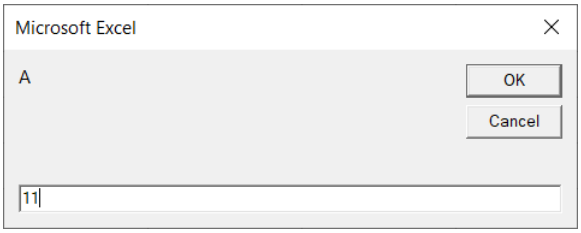
В данном примере приведен способ получения сведений от пользователя с помощью функции **InputBox**. Переменная **MyValue** содержит введенное значение, если была нажата кнопка OK или клавиша ENTER. Если нажать кнопку Отмена (Cancel), функция возвратит пустую строку.

```
Dim Message, Title, Default, MyValue
Message = "Число от 1 до 3"   ' Сообщение-подсказка
Title = "Пример"              ' Заголовок
Default = "1"                 ' Значение по умолчанию

' Выводит на экран сообщение, заголовок
MyValue = InputBox(Message, Title, Default)
```



Далее приведен вариант решения задачи **C = A + B.** Для ввода данных используется функция **InputBox**. При этом нет необходимости иметь на форме поля TextBox_A и TextBox_B.






```
Private Sub CB_Summ_Click()
Dim A As Integer
Dim B As Integer
Dim C As Integer
 A = Val(InputBox("A"))
 ' Значение из InputBox преобразуется в число и
 ' присваивается переменной A
 B = Val(InputBox("B"))
 ' Значение из InputBox преобразуется в число и
 ' присваивается переменной B
 C = A + B                ' Вычисление суммы
 TextBox_C.Value = C 'Результат присваивается TextBox_C
End Sub
```

Обмен данными может осуществляться и с использованием ячеек листа. Эту возможность мы рассмотрим при обсуждении возможностей обработки массивов.

6.2.2. Функция MsgBox

Выводит на экран диалоговое окно, содержащее сообщение, устанавливает режим ожидания нажатия кнопки пользователем, а затем возвращает значение типа **Integer**, указывающее, какая кнопка была нажата.

MsgBox[1](сообщение [, кнопки] [, заголовок окна] [, helpfile, context])

[1] MsgBox – сокр. от Message box – поле сообщения.

Смысл и назначение параметров сообщение, заголовок окна, helpfile, context такие же, как и функции **`InputBox`**.

Параметр кнопки имеет является необязательным. Представляет числовое выражение, равное сумме значений, которые указывают число и тип отображаемых кнопок, тип используемого значка, основную кнопку. Значение по умолчанию этого аргумента равняется 0.

Далее перечислены основные допустимые значения аргумента кнопки:

Константа	Значение	Описание
vbOKOnly	0	Отображается только кнопка «OK».
VbOKCancel	1	Отображаются кнопки «OK» и «Отмена» (Cancel).
VbYesNoCancel	3	Отображаются кнопки «Да» (Yes), «Нет» (No) и «Отмена» (Cancel).
VbYesNo	4	Отображаются кнопки «Да» (Yes) и «Нет» (No).
VbRetryCancel	5	Отображаются кнопки «Повторить» (Retry) и «Отмена» (Cancel).
VbCritical	16	Используется значок «Критическое сообщение».
VbQuestion	32	Используется значок «Предупреждающий запрос».
VbExclamation	48	Используется значок «Предупреждение».

Первая группа значений (0–5) указывает число и тип кнопок, отображаемых в окне диалога, вторая группа (16, 32, 48, 64) задает тип используемого значка. При определении значения аргумента кнопки следует суммировать не более одного значения из каждой группы.

Использование имен этих констант вместо их значений допускается в любом месте программы.

Возвращаемые значения

Константа	Значение	Нажатая кнопка
vbOK	1	OK
vbCancel	2	Отмена (Cancel)
vbAbort	3	Прервать (Abort)

Константа	Значение	Нажатая кнопка
vbRetry	4	Повторить (Retry)
vbIgnore	5	Пропустить (Ignore)
vbYes	6	Да (Yes)
vbNo	7	Нет (No)

Функцию **MsgBox** с двумя или большим числом аргументов можно использовать только в выражении. Наличие запятых, соответствующих отсутствующим аргументам, является обязательным.

В данном примере функция **MsgBox** используется для вывода окна диалога с сообщением об ошибке и кнопками «Да» (Yes) и «Нет» (No). Основной является кнопка «Нет» (No). Значение, возвращаемое функцией **MsgBox**, зависит от того, какая кнопка была нажата пользователем. Предположим, что DEMO.HLP является файлом справки, содержащим раздел с номером, равным 1000.

```
Dim Msg, Style, Title, Help, Ctxt, Res, MyString
Msg = "Обнаружена ошибка. Продолжить?" ' Сообщение
Style = vbYesNo + vbCritical + vbDefaultButton2
Title = "Пример"           ' Заголовок
Help = "DEMO.HLP"          ' Файл справки
Ctxt = 1000                ' Контекст
Res = MsgBox(Msg, Style, Title, Help, Ctxt)
    'Выводит сообщение

If Res = vbYes Then        ' Нажата кнопка «Да» (Yes)
    MyString = "Да"        ' Выполняет действие
Else                       ' Нажата кнопка «Нет» (No)
    MyString = "Нет"       ' Выполняет действие
End If
```

6.3. Условная инструкция (If ... Then ... Else)

В разветвляющихся вычислительных процессах отдельные этапы вычислений (операторы) выполняются не всегда в одном и том же порядке, а в зависимости от некоторых условий, проверяемых уже по ходу вычислений, выбираются для

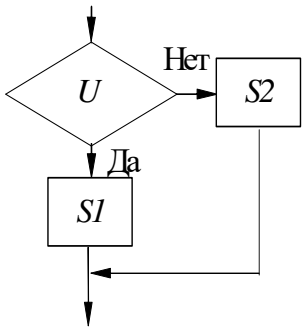
исполнения различные их последовательности. Если, например, в программе используются переменные **x**, **y** и **z**, и на каком-то этапе решения задачи требуется вычислить **z = max(x,y)**, то желаемый результат получается в результате выполнения либо оператора присваивания **z = x**, либо оператора присваивания **z = y**. Поскольку значения переменных **x** и **y** заранее неизвестны, а определяются в процессе вычислений, то в программе необходимо предусмотреть оба эти оператора присваивания. Однако на самом деле должен выполняться только один из них. Поэтому нужно, чтобы в программе содержалось указание на то, в каком случае надо выбирать для исполнения тот или иной оператор присваивания.

Для выполнения альтернативных действий используется условный оператор

```
If U Then S1 Else S2
```

Здесь **If** (если), **Then** (то) и **Else** (иначе) являются служебными словами, **U** — булевское (логическое) выражение, а **S1** и **S2** — инструкции.

Семантику условного оператора можно описать схемой.



Выполнение такого условного оператора сводится к выполнению одной из входящих в него инструкций **S1** или **S2**: если заданное условное выражение **U** принимает значение **True**, то выполняется **S1**, в противном случае — **S2**. Таким образом, решение задачи вычисления **z = max(x,y)** можно задать в виде

```
If x>y Then z=x Else z=y
```

При формулировании алгоритмов весьма типична такая ситуация, когда на определенном этапе вычислительного процесса какие-либо действия надо выполнять только с учетом некоторого условия, а если оно не выполняется, то на данном этапе вообще не нужно выполнять никаких действий. Пример такой ситуации — вычисление $|x|$. Записывая определение модуля на VBA, получим:

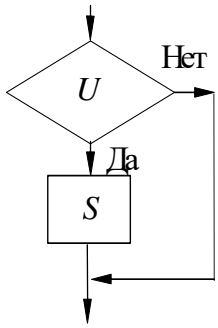
$$|x| = \begin{cases} x, & x \geq 0 \\ -x, & x < 0 \end{cases}$$

```
If x<0 Then x = -x Else x = x
```

Понятно, что присваивание **x = x** не имеет никакого смысла.

В подобных ситуациях весьма удобна *сокращенная форма* условного оператора

```
If U Then S
```



Достаточно очевидно: если значение логического выражения **U** есть **True**, то выполняется инструция **S**; в противном случае никаких иных действий, кроме вычисления выражения **U**, не производится.

Допускается также использование блоковой формы:

```
If U Then
  S11: S12: … S1n
Else
  S21: S22: … S2n
End If
```

Простую однострочную форму рекомендуется использовать для коротких, простых проверок. Однако блочная форма обеспечивает более структурированный подход и большую гибкость по сравнению с однострочной формой. Блочная форма обычно проще для чтения, обработки и отладки.

Однострочная форма допускает выполнение нескольких инструкций в результате проверки одного условия **If...Then**, но все инструкции должны находиться на одной строке и разделяться двоеточием, как в следующем примере:

```
If A > 10 Then A = A + 1: B = B + A: C = C + B
```

В блоковой форме инструкция **If** должна быть первой инструкцией в строке. Компоненты **Else** и **End If** могут иметь перед собой только номер строки или метку строки. Блок **If** должен заканчиваться инструкцией **End If**.

Допускаются вложенные блоки инструкций **If**, то есть блоки **If**, содержащиеся в других блоках.

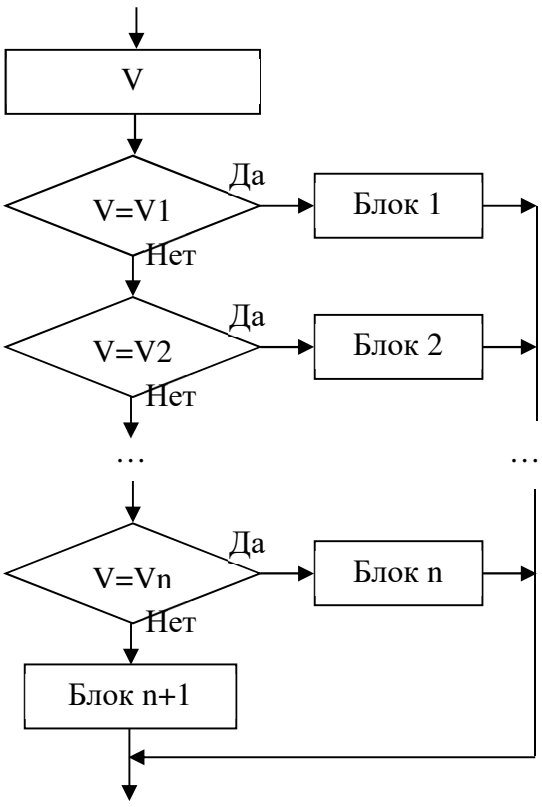
6.4. Оператор выбора варианта (Select Case)

Данный оператор — обобщение условного оператора для произвольного числа альтернатив. На практике довольно часто встречаются случаи, когда вычислительный процесс надо разветвить не по двум, а по **n** (**n** >2) возможным путям. Это можно сделать и с помощью условного оператора:

```
If U1 Then
    S1
Else
    ...
    If Un
    Then
        Sn
    End If
    ...
End If
```

В этом случае запись условного оператора может оказаться весьма громоздкой и не наглядной.

Во многих случаях более удобен оператор выбора варианта, который выполняет одну из нескольких групп в зависимости от значения выражения.




```
Select Case V
    [Case V1
        [Блок 1]
    [Case V2
        [Блок 2]
        ...
    [Case Vn
        [Блок n]
    [Case Else
        [Блок n + 1]]
End Select
```

V — любое числовое или строковое выражение.

V1, V2, … Vn обязательны при наличии предложения **Case**. Список с разделителями, состоящий из одной или нескольких форм следующего вида:

выражение; выражение **To** выражение; **Is** оператор сравнения выражения.

Ключевое слово **To** задает диапазон значений. При использовании ключевого слова **To** перед ним должно находиться меньшее значение. Ключевое слово **Is** с операторами сравнения задает диапазон значений.

Блок 1, …, Блок n — одна или несколько инструкций, выполняемых в том случае, если V совпадает с любым компонентом списка V1, …, Vn.

Блок n+1 — выполняется в том случае, если выражение V не совпадает ни с одним из V1, …, Vn.

Допускаются вложенные инструкции **Select Case**. Каждой вложенной инструкции **Select Case** должна соответствовать инструкция **End Select**.

```
Dim N
N = Val(InputBox("N=")    ' Инициализирует переменную
Select Case N             ' Анализирует число
    Case 1 To 5           ' Число между 1 и 5
        MsgBox "Между 1 и 5"
    Case 6, 7, 8          ' Число между 6 и 8
        MsgBox "Между 6 и 8"
    Case Is > 8 And N < 11    ' 9 или 10
        MsgBox "Больше 8"
    Case Else             ' Другие значения
        MsgBox "Вне интервала 1 – 10"
End Select
```

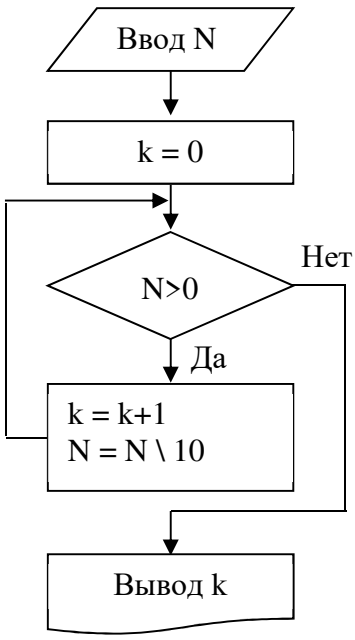
Пример 1. По заданной сумме N получить строку, содержащую это число и единицы измерения в рублях (3 рубля, 25 рублей, 41 рубль и т.п.).

При условии, что на форме у нас имеются TextBox_N – для ввода значения N, TextBox_Res – для вывода результата Res и кнопка CB_Res – для активизации вычислений, текст программы будет выглядеть так:

```
Private Sub CB_Res_Click()
```

```
Dim N As Integer, Nr As Integer
Dim Res As String ' Строка для формирования результата
' Значение из TextBox_N преобразуется в число и
' присваивается переменной N
  N = Val(TextBox_N.Value)
  If 10 <= N And N <= 20 Then
    Res = " рублей"
  Else
    Nr = N Mod 10     ' Выделяем младшую цифру числа N
    Select Case Nr
      Case 1
        Res = " рубль"
      Case 2 To 4
        Res = " рубля"
      Case 0, 5 To 9
        Res = " рублей"
    End Select
  End If
  TextBox_Res.Value=Str(N)+Res'Результат в TextBox_Res
End Sub
```

6.5. Программирование циклов



Рассмотренные операторы задают явно все те операции, которые должны быть выполнены, причем каждая из них выполняется не более одного раза. Поэтому ясно, что с помощью таких операторов можно задать лишь простейшие вычисления, а в этом случае эффект использования компьютера на самом деле ничтожен, потому что время, затрачиваемое на доставление такой программы, сравнимо со временем выполнения всех заданных операций вручную или с использованием простейших вычислительных средств типа калькулятора.

Рассмотрим организацию циклического алгоритма на примере поиска количества цифр заданного целого положительного числа N.

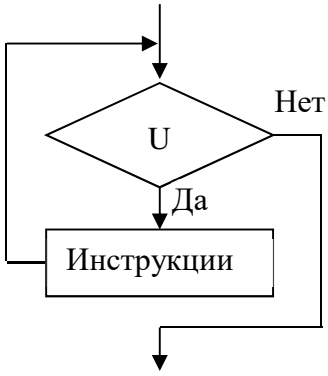
После ввода значения переменной N счетчику (переменная k) присваиваем нулевое значение. Затем в цикле (пока N > 0) увеличиваем счетчик k на единицу и удаляем младший разряд N (N = N \ 10). При N = 0 выходим из цикла и выводим полученное значение k.

В приведенной блок-схеме можно выделить блок, который реализуется инструкцией **While … Wend**.

6.5.1. Оператор цикла с предусловием (While … Wend)

Оператор цикла с предусловием имеет следующий синтаксис:

```
While U
    [инструкции]
Wend
```



где **While**, **Wend** – ключевые слова, ограничивающие **инструкции**, которые выполняются в цикле; **U** — логическое выражение, принимающее значение **True** (Истина) или **False** (Ложь).

Цикл **While** выполняется до тех пор, пока условие **U = True**.

При условии, что на форме у нас имеются **TextBox_N** – для ввода значения **N**, **TextBox_k** – для вывода результата **k** и кнопка **CB_Res** – для активизации вычислений, текст программы будет выглядеть так:

```
Private Sub CB_Res_Click()
Dim N As Integer
Dim k As Integer
  N = Val(TextBox_N.Value) ' Значение из TextBox_N
' преобразуется в число и присваивается переменной N
  k = 0
  While N > 0  ' Начало цикла, проверка условия
    k = k + 1  ' Увеличение счетчика
    N = N \ 10 ' Удаление младшего разряда
  Wend         ' Конец цикла
  TextBox_k.Value = k ' k присваивается TextBox_k
End Sub
```

6.5.2. Операторы цикла с условиями (Do … Loop)

Более общий случай реализуется циклами **Do … Loop**, которые имеют две формы.

С предусловием	С постусловием
Do While | Until U **инструкции** **[Exit Do]** **Loop**	**Do** **инструкции** **[Exit Do]** **Loop While | Until U**

где **Do, Loop** – ключевые слова, ограничивающие начало и конец цикла;
While, Until – ключевые слова, указывающие тип цикла;
U – логическое выражение, принимающее значение **True** (Истина) или **False** (Ложь);
Exit Do – прекращает выполнение цикла;
символ **|** означает выбор одного из вариантов.

При этом конструкция **Do While U** выполняется пока **U=True**, а цикл **Do Until U** - пока **U=False**.

Тогда решение задачи о поиске количества цифр заданного целого положительного числа N можно представить в четырех вариантах:

k = 0 **Do While N>0** **k = k + 1** **N = N \ 10** **Loop**	**k = 0** **Do Until N=0** **k = k + 1** **N = N \ 10** **Loop**	**k = 0** **Do** **k = k + 1** **N = N \ 10** **Loop While N>0**	**k = 0** **Do** **k = k + 1** **N = N \ 10** **Loop Until N=0**

6.5.3. Оператор цикла с параметром (For ... Next)

Пусть, например, требуется вычислить суммы первых N членов натурального ряда

y = 1 + 2 + 3 + … + N.

Как видно, в этом случае процесс вычислений будет носить циклический характер: оператор **y = y + I** должен выполняться многократно, т.е. циклически, при различных значениях **I** — **N** раз.

Этот пример циклического вычислительного процесса весьма типичен; его характерные особенности состоят в том, что:

– число повторений цикла известно к началу его выполнения;

– управление циклом осуществляется с помощью переменной, которая в этом циклическом процессе принимает последовательные значения от заданного начального до заданного конечного значений.

Для компактного задания подобного рода вычислительных процессов и служит *оператор цикла с параметром*.

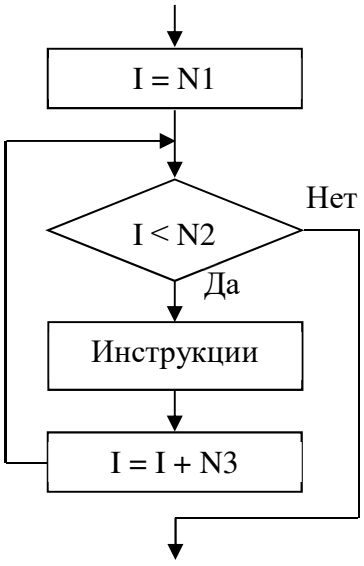
```
For I = N1 To N2 [Step N3]
     [инструкции]
     [Exit For]
     [инструкции]
Next [I]
```

где **For**[1], **To**, **Step**, **Next**, **Exit For** – ключевые слова.

I — числовая переменная, используемая в качестве счетчика цикла. Эта переменная не может принадлежать к типу **Boolean** или быть элементом массива.

N1, **N2** — соответственно начальное и конечное значение переменной **I**.

N3 — шаг изменения переменной **I** при каждом выполнении тела цикла. Может быть как положительным, так и отрицательным. Если это значение не задано, по умолчанию шаг равен единице.



Выполнение цикла при **N3** > 0 можно описать следующей схемой.

После выполнения всех инструкций цикла значение **N3** добавляется к текущему значению переменной **I**. После этого инструкции цикла либо выполняются еще раз (на основе того же условия, которое привело к начальному выполнению цикла),

[1] **For** – для; **To** – к; **Step** – шаг; **Next** – следующий; **Exit For** – выход из цикла.

либо цикл завершается и выполнение продолжается с инструкции, следующей за инструкцией **Next**.

Изменение значения переменной **I** внутри цикла усложняет чтение и отладку программы.

Альтернативный способ выхода из цикла предоставляет инструкция **Exit For**. В любых местах цикла может размещаться любое число таких инструкций. Инструкция **Exit For** часто применяется вместе с проверкой некоторого условия (например, **If...Then**). Эта инструкция передает управление инструкции, непосредственно следующей за инструкцией **Next**.

Следует обратить внимание на тот факт, что **инструкции** могут не выполняться ни разу.

Допускается организация вложенных циклов **For...Next** (один цикл **For...Next** располагается внутри другого). Счетчик каждого цикла должен иметь уникальное имя. Допускаются следующие конструкции:

```
For I = 1 To 10
    For J = 1 To 10
        For K = 1 To 10
            ...
        Next K
    Next J
Next I
```

Если опустить переменную **I** в инструкции **Next**, выполнение продолжается, как и при ее наличии. При обнаружении инструкции **Next** до соответствующей инструкции **For** возникает ошибка.

Пример 2. Получить произведение двух натуральных чисел **M** и **N** без использования операции умножения.

Очевидно, что: $k = \underbrace{m + m + ... + m}_{n}$. Этот несложный алгоритм приведен в программе.

```
Private Sub CB_Res_Click()
Dim N As Integer
Dim I As Integer
Dim k As Integer
```

```
  N = Val(TextBox_N.Value)
' Значение из TextBox_N преобразуется в число
' и присваивается переменной N

  M = Val(TextBox_M.Value) ' То же для M
  k = 0
  For I = 1 To N          ' Начало цикла
    k = k + M             ' Суммируем
  Next I                  ' Конец цикла
  TextBox_k.Value = k     ' k присваивается TextBox_k
End Sub
```

Замечание.

Проверив выполнение этой программы, например, при **N** = 3 и **M** = 200, легко выяснить, что за 3 выполнения тела цикла **k** = **k** + **M** получаем ответ **k** = 600. При **N** = 200 и **M** = 3 получаем тот же ответ, но за 200 шагов. Из этого можно сделать вывод о том, что для эффективности работы программы необходимо, чтобы количество повторений цикла **N** было меньше (точнее — не больше) **M**.

За счет введения условного оператора с блоком обмена значениями переменных **N** и **M** гарантированно обеспечивается меньшее количество выполнений операции тела цикла **k = k + M**.

```
If N > M Then
  r = N:  N = M:  M = r
End If
```

6.5.4. Использование операторов цикла

Приведем примеры решения некоторых задач с использованием операторов цикла.

Пример 3. Для заданного натурального значения переменной N вычислить k = N!!. По определению,

$$N!! = \begin{cases} 1 \cdot 3 \cdot \ldots \cdot N, \text{ если N — нечетное} \\ 2 \cdot 4 \cdot \ldots \cdot N, \text{ если N — четное} \end{cases}$$

Для решения этой задачи необходимо выяснить, является ли введенное N четным или нет. Для этого следует проверить остаток от деления на 2 — если он равен 0, то число четное, если 1 — нечетное. Это нужно сделать для того, чтобы обеспечить

изменение множителя либо от 1 до N с шагом 2 (т.е. все нечетные), либо от 2 до N с шагом 2 (т.е. все четные)

```
Private Sub CB_Res_Click()
Dim N As Integer
Dim I As Integer
Dim k As Integer
   ' Значение из TextBox_N преобразуем в число и
   ' присваиваем переменной N
  N = Val(TextBox_N.Value)
  If N Mod 2 = 0 Then
    r = 2           ' T.e. N – четное
  Else
    r = 1           ' T.e. N – нечетное
  End If
  k = 1
  For I = r To N Step 2      ' Начало цикла
    k = k * I                ' Получаем произведение
  Next I                     ' Конец цикла
  TextBox_k.Value = k        ' k присваивается TextBox_k
End Sub
```

Эту же задачу можно решить без использования проверки на четность (объясните почему).

```
  k = 1
  For I = N To 1 Step -2     ' Начало цикла
    k = k * I                ' Получаем произведение
  Next I                     ' Конец цикла
```

Пример 4. Является ли заданное натуральное число N простым? (Простым называется число, которое в качестве делителей имеет 1 и само себя.)

При решении этой задачи необходимо проверять все возможные делители. (Известно, что делители могут быть в диапазоне от 2 до целой части от $\sqrt{N}$.)

Логическая переменная F — признак того, что N — простое (**`F = True`**) или составное (**`F = False`**).

Переменная S будет иметь соответствующий текст.

```
Private Sub CB_Res_Click()
Dim N As Integer, I As Integer,
Dim F As Boolean
Dim S As String
  N = Val(TextBox_N.Value)
  ' Значение из TextBox_N преобразуем в число и
  ' присваиваем переменной N
  r = Int(Sqr(N)) ' Верхняя граница возможных делителей
  F = True         ' Предполагаем, что N – простое
  For I = 2 To r              ' I – делители
    If N Mod I = 0 Then       ' Если N делится на I,
      F = False               ' то N – составное
      Exit For                ' досрочный выход из цикла
    End If
  Next I
  If F Then s = "Простое" Else s = "Составное"
  TextBox_Res.Value = s
End Sub
```

Пример 5. Выяснить, за сколько лет в некотором банке при начальном вкладе R рублей и процентах годового прироста P будет накоплена сумма S.

Введем переменные k – число лет, за которые будет накоплена требуемая сумма и d – величину прироста за год.

Для вывода числа лет используется **TextBox_k**, а для вывода соответствующего текста (3 года; 1 год; 5 лет) используется метка **Label_Year**. Изменяя свойство **Caption** можно получить соответствующее название.

В этом примере на форме помещены кнопки **CB_Clear** и **CB_Exit**, которые используются для очистки используемых полей и завершения работы.

```
Private Sub CB_Res_Click()
Dim R As Single
Dim S As Single
Dim d As Single
Dim k, Nr As Integer
  R = Val(TextBox_R.Value)
  ' Значение из TextBox_R преобразуется в число
  ' и присваивается переменной R
```

```
  P = Val(TextBox_P.Value) ' TextBox_P -> P
  S = Val(TextBox_S.Value) ' TextBox_S -> S
  While R < S               ' Начало цикла
    d = R * P / 100         ' Прирост за год
    R = R + d               ' Вклад в конце года
    k = k + 1               ' Число лет
  Wend                      ' Конец цикла
  TextBox_k.Value = k       ' Результат в TextBox_k
  If 10 <= k And k <= 20 Then
    Res = "лет"
  Else
    Nr = k Mod 10       ' Выделяем младшую цифру k
    Select Case Nr
      Case 1
        Res = "год"
      Case 2 To 4
        Res = "года"
      Case 0, 5 To 9
        Res = "лет"
    End Select
  End If
  Label_Year.Caption = Res
End Sub

Private Sub CB_Clear_Click()
  TextBox_R.Value = ""         ' Очищаем поля
  TextBox_P.Value = ""
  TextBox_S.Value = ""
  TextBox_k.Value = ""
  Label_Year.Visible = False   ' Делаем невидимым
End Sub

Private Sub CB_Exit_Click()
  End                          ' Завершение работы
End Sub
```

7. ОСНОВНЫЕ ОБЪЕКТЫ MS EXCEL

В рамках настоящего пособия не ставится цель изучения всех возможностей. Для разработки большинства программ достаточно изучить основные из них.

7.1. Объект Application. Свойства, методы и события

7.1.1. Свойства объекта Application

Свойства	Описание и допустимые значения
ActiveWorkbook, ActiveSheet, ActiveCell, ActiveChart	Возвращают активный объект: рабочую книгу, лист, ячейку, диаграмму. Свойство ActiveCell содержится в ActiveSheet, а свойства ActiveSheet и ActiveChart — в ActiveWorkbook.
ThisWorkbook	Возвращает рабочую книгу, содержащую выполняющийся в данный момент макрос.
Calculation	Устанавливает режим вычислений. Возможные значения.
Caption	Возвращает текст в строке имени активного окна.

7.1.2. Основные методы объекта Application

Методы	Действия
Calculate	Вызывает принудительные вычисления во всех открытых рабочих книгах.
Run	Запускает на выполнение подпрограмму или макрос: **`Run (ИмяМакроса, Аргументы).`**
Wait	Временно приостанавливает работу приложения: **`Wait(Time).`**
OnTime	Назначает выполнение процедуры на определенное время: **`OnTime (ВремяЗапуска, ИмяПроцедуры, ...).`**
Quit	Закрывает приложение.

7.1.3. События объекта Application

Событие	Когда происходит
NewWorkBook	При создании новой рабочей книги.
WorkbookActivate	При активизации рабочей книги.
WorkbookBeforeClose	Перед закрытием рабочей книги.
WorkbookBeforeSave	Перед сохранением рабочей книги.
WorkbookDeactivate	Когда рабочая книга теряет фокус.
WorkbookNewSheet	При добавлении нового листа.
WorkbookOpen	При открытии рабочей книги.

7.2. Основные свойства, методы и события семейства WorkBooks

Объект **Workbook** — это файл рабочей книги. Получить объект **Workbook** можно используя свойства **Workbooks**, **ActiveWorkbook** и **ThisWorkbook** объекта **Application**.

7.2.1. Основные свойства объектов семейства Workbooks

Свойства	Описание и допустимые значения
ActiveSheet	Возвращает активный лист книги.
ActiveChart	Возвращает активную диаграмму.
Count	Возвращает количество объектов семейства.
Sheets, Worksheets, Charts	Возвращают семейства всех рабочих листов книги и всех диаграмм соответственно.

7.2.2. Основные методы объектов семейства Workbooks

Методы	Действия
Activate	Активизирует рабочую книгу (первый лист становится активным).
Add	Создает новую рабочую книгу.
Close, Open, OpenText	Закрытие (открытие) рабочей книги, открытие текстового файла с таблицей данных.
Save, SaveAs	Сохранение рабочей книги (сохранение в другом файле). Например, запросить у пользователя имя файла и сохранить активную рабочую книгу можно кодом программы: **fName = Application.GetSaveAsFilename ActiveWorkbook.SaveAs Filename := fName**

7.2.3. События объектов семейства Workbooks

Событие	Когда происходит
BeforeClose BeforeSave Deactivate	Перед закрытием рабочей книги. Перед сохранением рабочей книги. Когда рабочая книга теряет фокус.

7.3. Основные свойства и методы объектов семейства Worksheets

Объект **Worksheet** представляет собой рабочий лист. Объект **Worksheet** можно получить, используя свойства **ActiveSheet** или **Worksheets** объекта **Workbook**.

7.3.1. Свойства объектов семейства Worksheets

Свойства	Описание и допустимые значения
Name	Возвращает имя рабочего листа: **Worksheets(1).Name = "Итоги"**
Visible	**True** (**False**) — рабочий лист видим (невидим).
Range	Возвращает ссылку на указанный диапазон ячеек. Например: **ActiveSheet.Range("B1")**
UsedRange	Возвращает диапазон ячеек рабочего листа.
ActiveCell	Возвращает активную ячейку рабочего листа.

7.3.2. Методы объектов семейства Worksheets

Методы	Выполняемые действия
Activate	Активизирует рабочий лист: **Worksheet(2).Activate**
Add	Создает новый рабочий лист. Параметры: **Before** — лист, перед которым будет размещен новый лист; **After** — лист после которого будет помещен новый лист; **Count** — число добавляемых листов; **Type** — тип добавляемого листа. Например: **ActiveWorkbook.Worksheets.Add**
Delete	Удаляет рабочий лист: **Worksheets(1).Delete**
Evaluate	Преобразует текстовую строку в объект Excel или значение. Используется, например, для ввода ссылок на ячейки: **MyCell = InputBox("Введите имя ячейки")** **Evaluate(MyCell).Value = "Новое значение"**
Copy	Копирование активного рабочего листа в другое место рабочей книги: **Worksheets("Лист2").Copy After :=** **Worksheets("Лист3")**
Move	Перемещение активного рабочего листа в другое место рабочей книги: **Worksheets("Лист2").Move After :=** **Worksheets("Лист3")**

7.3.3. События объекта Worksheet

Событие	Когда происходит
Before Close	Перед закрытием рабочей книги.
Before Save	Перед сохранением рабочей книги.
Deactivate	Когда рабочая книга теряет фокус.
NewSheet	При добавлении нового листа в рабочую книгу.
Open	При открытии рабочей книги.
SheetActivate	При активизации рабочего листа.
SheetDeactivate	Когда рабочий лист теряет фокус.

7.4. Объект Range

7.4.1. Адресация ячеек в Excel

Для ссылок на ячейки в Excel используются два формата:

Формат A1. Ссылка состоит из имени столбца (обозначаются буквами от **A** до **IV**, **256** столбцов максимально) и номера строк (от 1 до 65536). Для ссылки на диапазон ячеек указываются адреса левой верхней и правой нижней ячейки диапазона, разделенных двоеточием. Например, **B10:B20**, **7:7** (все ячейки в 7-й строке), **5:10** (все ячейки между 5-й и 10-й строкам включительно), **D:D** (все ячейки в столбце **D**), **H:J** (все ячейки между столбцами **H** и **J** включительно). Признаком абсолютной ссылки является знак доллара ($) перед именем строки или столбца.

Формат R1C1. В формате **R1C1** после буквы «**R**» (Row) указывается номер строки ячейки, после буквы «**C**» (Colonum) — номер столбца. Например, абсолютная ссылка **R1C1** эквивалентна абсолютной ссылке **$A$1** для формата **A1**. Для задания относительной ссылки указывается смещение по отношению к активной ячейке. Смещение указывается в квадратных скобках. Знак указывает направление смешения. Например, **R[-3]C** (относительная ссылка на ячейку, расположенную на три строки выше в том же столбце), **R[2]C[2]** (относительная ссылка на ячейку, расположенную на две строки ниже и на два столбца правее), **R2C2** (абсолютная ссылка на ячейку, расположенную во второй строке и во втором столбце), **R[-1]** (относительная ссылка на строку, расположенную выше текущей ячейки), **R** (абсолютная ссылка на текущую строку).

Полный адрес ячейки может содержать также имя рабочего листа и адрес книги. После имени листа ставится знак «**!**», а адрес книги заключается в квадратные скобки. Например:

```
[МояКнига.xls]Лист1!D2.
```

Объект **Range** используется для работы с ячейками, строками, столбцами, а также их группами. Для доступа к объекту чаще всего используются свойства **Range** и **Cells**, хотя есть и другие возможности.

Если используется свойство **Range**, то в качестве аргумента указывается любая допустимая в Excel ссылка в формате **A1**. Если имя листа не указывается, то используется активный лист. Например:

```
' Ячейке C2 листа Лист1 присвоить значение 10
Worksheets("Лист1").Range("C2").Value = 10
' Ячейке C2 листа Лист1 присвоить значение 10
Range("C2").Value = 10
```

Свойство **Cells** используется для доступа к отдельной ячейке. В качестве аргументов указываются номер строки и столбца. Например, так можно присвоить значение ячейке **A5** первого рабочего листа:

```
Worksheets(1).Cells(2, 3).Value = 10
```

Можно также использовать свойство **Cells** для альтернативного указания диапазона. Например:

Range("A2:C3") и **Range(Cells(2,1), Cells(3,3))** определяют один и тот же диапазон.

7.4.2. Основные свойства объекта Range

Свойства	Описание и допустимые значения
Value	Возвращает значение из ячейки или диапазона: `X = Range("C2").Value`
Name	Возвращает имя диапазона: `Range("B1:B4").Name="Итого"`
CurrentRagion	Возвращает количество строк текущего диапазона.
WrapText	**True** (**False**) — разрешает (не разрешает) перенос текста при вводе в диапазон.
EntireColumn, EntireRow	Возвращает строку и столбец.

Свойства	Описание и допустимые значения
ColumnWidth, RowHeight	Возвращает ширину столбцов и высоту строк диапазона.
Font	Возвращает объект **Font** (шрифт). Например: **With** **Worksheets("Лист1").Range("B5").Font** **.Size = 14** **.Bold = True** **.Italic = True** **End With**
Formula	Формула в формате **A1** . Например, так можно ввести формулу в ячейку **B5**: **Range("B5").Formula = "=$A$4+$A$10".** При считывании значения возвращается текстовая строка (как в строке формул).
Formula Local	Формула в формате **A1** с учетом языка пользователя (для неанглоязычных версий Excel). Например: **Range("B5").FormulaLocal = "=ПИ()"**
FormulaR1C1	Формула в формате **R1C1**. Например, **Range("B1").FormulaR1C1 = "=R1C1+1"**
FormulaR1C1Local	Формула в формате **R1C1** с учетом языка пользователя (для неанглоязычных версий Excel).

Методы объекта **Range** можно разделить на две большие группы. Методы, относящиеся к самому объекту, и методы, реализующие команды. Многие из них имеют параметры, которые здесь описываются лишь частично. Подробнее о параметрах этих методов можно прочитать, например, в справочной системе Excel. Для изучения методов, реализующих команды, рекомендуется записать макрос, выполняющий нужную команду, и проанализировать полученный код.

7.4.3. Основные методы объекта Range

Методы	Действия
Adress	Возвращает адрес ячейки.
Auto Fit	Автоматически настраивает ширину столбца и высоту строки. Например: **`Range ("B1:B3").Columns.AutoFit`**. Использование свойства Columns или Rows в данном случае необходимо, так как значением диапазона должны быть строки или столбцы, иначе будет выдаваться ошибка.
Clear	Очищает диапазон. Например: **`Range("A1:C5").Clear`**
Delete	Удаляет диапазон. Параметр **`Shift`** определяет направление сдвига ячеек. Например: **`Range("A6:D6").Delete Shift:=xlShiftToLeft`**
Insert	Вставляет ячейку или диапазон ячеек. Например, так можно вставить строку перед шестой строкой на листе «Лист2»: **`Worksheets("Лист2").Rows(6).Insert`**
Select	Выделяет диапазон: **`Range("A1:C7").Select`**

7.4.4. Методы объекта Range (команды Excel)

Кроме методов, реализующих команды, объект Range имеет методы, которые используют команды Excel.

Метод	Действия
DataSeries	Создает прогрессию. **`DataSeries (rowcol, date, step, stop, trend)`** Вручную метод выполняется с помощью команды Правка > Заполнить > Прогрессия.
AutoFill	Автозаполнение. Автоматически заполняет ячейки дна пазона элементами последовательности: Объект (Диапазон, Тип).
Consolidate	Объединение данных из нескольких диапазонов в одну итоговую таблицу. Соответствует команде Данные > Консолидация.
Find	Поиск данных. Вручную вызывается командой Правка > Найти.
tblGoalSeek	Подбор параметра. Вручную выполняется с помощью команды Сервис > Подбор параметра.
Sort	Сортировка данных. Вручную выполняется с помощью команды Данные > Сортировка.
Subtotal	Добавляет промежуточные итоги. Вручную вызывается командой Данные > Промежуточные итоги.

7.5. Инструкция With

Инструкция **With**[1] позволяет указывать объект или определяемый пользователем тип данных только один раз для последовательности инструкций. Инструкция **With** ускоряет выполнение процедур и помогает избежать повторного задания имени объекта.

В следующем примере диапазон ячеек **A1:C10** заполняется числовым значением **30**, затем устанавливается полужирный шрифт и для внутренней области ячеек задается желтый цвет.

```
Sub FormatRange()
    With Worksheets("Лист1").Range("A1:C10")
        .Value = 30
        .Font.Bold = True
        .Interior.Color = RGB(255, 255, 0)
    End With
End Sub
```

Для увеличения эффективности программы возможно создание вложенных инструкций **With**.

8. МАССИВЫ

До сих пор мы рассматривали лишь простые типы данных: значение любого из них — отдельное данное (отдельная ячейка памяти).

В VBA имеется возможность описать массив для работы с набором значений одного типа данных. Массив представляет собой одну переменную с множеством ячеек памяти для хранения значений, тогда как обычная переменная имеет только одну ячейку, в которой может храниться только одно значение. При необходимости сослаться на все элементы массива можно ссылаться на массив как целое. Возможны также ссылки на его отдельные элементы.

[1] With — указывает на связь, совместность с …, вместе с …

8.1. Одномерные массивы

Необходимость в массивах возникает всякий раз, когда при решении задачи приходится иметь дело с большим, но конечным количеством однотипных упорядоченных данных. Эта структура представляет собой упорядоченный набор перенумерованных компонент, при этом индивидуальное имя получает весь набор (вся структура данных), а для компонент этого набора определяется лишь порядок следования и общее их количество. Дадим упорядоченному набору из десяти вещественных компонент имя **X**. Тогда для указания той или иной компоненты этого набора в качестве ее «имени» можно использовать имя самого набора и номер нужной компоненты в этом наборе. В обычной математической символике обозначаются X_1, X_2, ... X_{10}. Здесь имеет место полная аналогия с обычным понятием одномерного числового вектора. Числовой вектор как единый объект имеет индивидуальное имя и структурно состоит из фиксированного числа упорядоченных однотипных компонент — чисел.

Итак, *массив* — это упорядоченный набор фиксированного количества некоторых значений (компонент массива). Все компоненты должны быть одного и того же типа. Так, в примере с суммированием можно ввести в употребление вещественный массив — вектор, компонентами которого являются вещественные числа. Массив с компонентами — отдельными литерами — может интерпретироваться как строка текста; целочисленная матрица может быть представлена как массив, компонентами которого являются целочисленные векторы и т.д.

Как обычно, каждому используемому в программе конкретному массиву должно быть дано имя. Каждая компонента массива может быть явно обозначена путем указания имени массива, за которым следует *индекс*. При ссылке на компоненты массива индекс записывается на одном уровне с именем и заключается в круглые скобки.

В нашем примере массив получит имя **X**, ссылки на отдельные его компоненты производятся с помощью переменных **X(1), X(2), ..., X(10).**

В общем случае в качестве индекса может быть использовано выражение, значение которого и определяет номер компоненты массива. При этом важно, что в индексное выражение могут входить переменные, так что при изменении их значений меняется и значение индекса, определяемое номером компоненты массива. Таким образом, одна и та же переменная с индексом в процессе выполнения программы может обозначать различные компоненты массива.

Например, для записи денежных затрат на каждый день календарного года можно описать один массив с 365 элементами, вместо того, чтобы описывать 365 переменных. Каждый элемент массива содержит одно значение. Следующая инструкция описывает массив **Y** с 365 элементами. По умолчанию индексация массива начинается с нуля, так что верхняя граница массива — 364, а не 365.

```
Dim Y(364) As Currency
```

Чтобы задать значение отдельного элемента, надо указать его индекс. В следующем примере всем элементам массива присваивается значение 20.

```
Dim Y(364) As Currency
Dim I As Integer
    For I = 0 to 364
        Y(I) = 20
    Next I
```

8.1.1. Изменение нижней границы индексов (Option Base)

По умолчанию индекс первого элемента массива равен 0. Для изменения номера первого элемента используется инструкция **Option Base** в начале модуля. В следующем примере инструкция **Option Base** изменяет индекс первого элемента, а инструкция **Dim** описывает массив **Y** с 365 элементами.

```
Option Base 1
Dim Y(365) As Currency
```

Допускается также явное задание нижней границы индексов массива с помощью предложения **To**, как показано в следующем примере.

```
Dim Y(1 To 365) As Currency
Dim Z(7 To 13) As String
```

8.1.2. Изменение границ массивов (ReDim)

Инструкция **ReDim** используется для задания или изменения размера динамического массива, который уже был формально описан с помощью инструкции **Private**, **Public** или **Dim** с пустыми скобками (без индексов размерностей).

```
ReDim [Preserve] имяМассива(индексы) [As тип],
```

где **Preserve** — ключевое слово, используемое для сохранения данных в существующем массиве при изменении значения последней размерности;

имяМассива — имя переменной, удовлетворяющее стандартным правилам именования переменных;

индексы — размерности переменной массива; допускается описание до 60 размерностей.

Имеется возможность повторно использовать инструкцию **ReDim** для изменения числа элементов и размерностей массива. Однако не допускается описание массива с одним типом данных и использование инструкции **ReDim** для последующего изменения типа данных этого массива, если массив не содержится в переменной типа **Variant**. Тип элементов массива, содержащегося в переменной типа **Variant**, может быть изменен с помощью предложения **As** тип.

```
Dim X()
N = Val(TextBox_N.Value)
ReDim X(1 To N)
    …
```

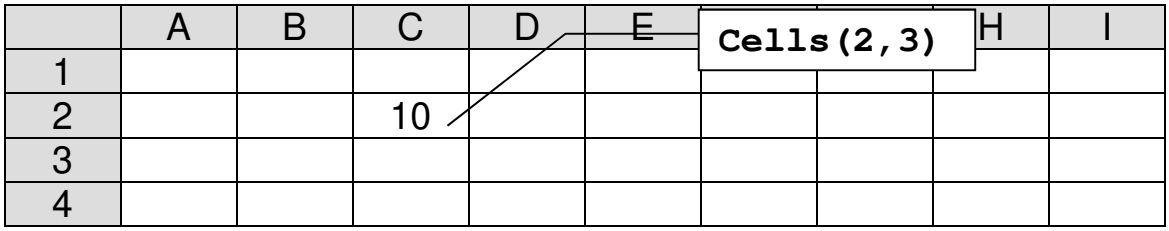
8.1.3. Ввод – вывод элементов массива

Ранее для ввода исходных данных и получения результатов мы использовали свойство **Value** объекта **TextBox**.

Для организации обмена данными с элементами массива удобно использовать ячейки активного листа Exel. Свойство **Cells** позволяет получить доступ к отдельной ячейке активного листа.

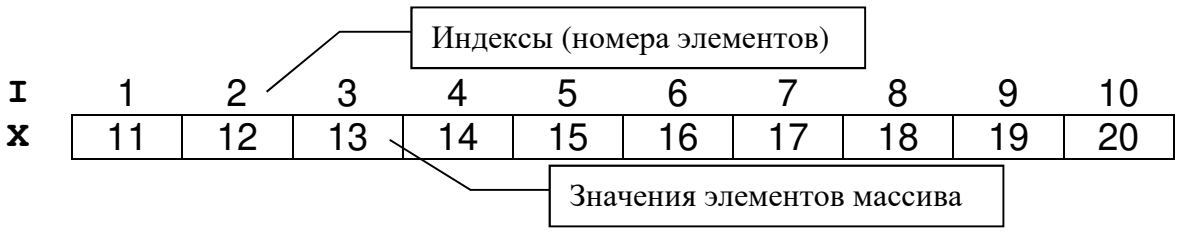
Далее приведен фрагмент рабочего листа. Для того чтобы ячейке **C2** присвоить значение **10**, достаточно записать:

```
Cells(2, 3) = 10
```



Пусть необходимо заполнить массив **X** из десяти элементов числами от 11 до 20. Это можно сделать следующим образом:

```
Dim X(1 To 10) As Integer
  For I = 1 To 10 ' Перебираем все индексы
    X(I) = I + 10 ' Присваиваем значения от 11 до 20
  Next I
```



Следующий шаг: нам нужно «увидеть» полученные в оперативной памяти значения. Для этого элементы массива **X** выведем на активный лист, например, во **вторую** строку. Это выполняет следующий фрагмент программы:

```
  …
  For I = 1 To 10      ' Перебираем все индексы
   Cells(2, I) = X(I) ' Присваиваем ячейкам листа
  Next I               ' значения элементов X
```

Результат вывода элементов массива во вторую строку.

	A	B	C	D	E	F	G	H	I	J
1										
2	11	12	13	14	15	16	17	18	19	20
3										

В том случае, когда нам нужно ввести и обработать данные, которые располагаются во **второй** колонке (пусть их будет 10), можно сделать так:

```
…
For I = 1 To 10      ' Перебираем все индексы
 X(I) = Cells(I, 2) ' Значения ячеек листа
Next I              ' присваиваем элементам X
```

Для определения количества непустых ячеек в строке с заданным номером **N_Row**, начиная с колонки **N_Col**, можно воспользоваться конструкцией.

```
…
I = 1            ' Индекс
' Пока не пусто
While1 Cells(N_Row, N_Col + I - 1) <> ""
  I = I + 1      ' Переходим к следующей ячейке
Wend
N = I - 1        ' Т.е. заполненных — на 1 меньше
```

Позиция ячейки в колонке — (**N_Col + I - 1**), номер элемента (индекс) — **I**.

Часто для формирования элементов массива используется датчик случайных чисел **Rnd**.

Далее приведена программа, в которой формируется массив, состоящий из случайных чисел из диапазона [**A, B**]. Затем полученные элементы массива выводятся в первую строку активного листа.

```
…
```

[1] Возможно использование конструкции:
While Not IsEmpty(Cells(N_Row, N_Col + I - 1)),
которая имеет такой же смысл.

```
Dim X() As Integer          ' Описание массива
Dim N As Integer
Dim I, A, B As Integer
' Количество элементов массива
  N = Val(TextBox_N.Value)
' Переопределяем размерность массива
  ReDim X(1 To N) As Integer
  A = -20     ' Нижняя граница диапазона
  B = 30      ' Верхняя граница диапазона
  Randomize  ' Инициализация датчика случайных чисел
  ' Присваиваем элементам массива случайные значения
  For I = 1 To N
    X(I) = Int((B - A) * Rnd + A)
  Next I
  ' Выводим элементы массива в первую строку листа
  For I = 1 To N
    Cells(1, I) = X(I)
  Next I
```

8.2. Многомерные массивы

В языке Visual Basic допускается описание массивов, имеющих до 60 размерностей. Например, следующая инструкция описывает двумерный массив (матрицу), в котором 5 строк и 10 колонок.

```
Dim A(1 To 5, 1 To 10) As Single
```

Для обработки многомерных массивов используются вложенные инструкции **For ... Next**. В следующей процедуре двумерный массив заполняется значениями типа данных **Single** и выводится на активный лист.

```
Sub FillArrayMulti()
  Dim I As Integer, J As Integer
  Dim A(1 To 5, 1 To 10) As Single
```

```
  ' Заполнение массива
  For I = 1 To 5
    For J = 1 To 10
      A(I, J) = I * 10 + J
    Next J
  Next I
  ' Вывод на активный лист
  For I = 1 To 5
    For J = 1 To 10
      Cells(I, J) = A(I, J)
    Next J
  Next I
End Sub
```

8.3. Примеры использования массивов

Пример 6. Вычислить сумму **S** элементов одномерного массива **X**, состоящего из **N** целочисленных случайных значений из диапазона **[A, B]**.

```
Dim X() As Integer          ' Описание массива
Dim N As Integer, I As Integer, S As Integer
' Количество элементов массива
  N = Val(TextBox_N.Value)
 ' Определяем размерность массива
 ReDim X(1 To N) As Integer

' Нижняя граница диапазона
  A = Val(TextBox_A.Value)
' Верхняя граница диапазона
  B = Val(TextBox_B.Value)

  Randomize  ' Инициализация датчика случайных чисел
  ' Присваиваем элементам массива случайные значения
  For I = 1 To N
    X(I) = Int((B - A) * Rnd + A)
  Next I
```

```
' Вывод массива на активный лист
For I = 1 To N
  Cells(I, 1) = X(I)
Next I
' Вычисление суммы
For I = 1 To N
  S = S + X(I)
Next I
' Вывод в заранее заготовленный TextBox
TextBox_S.Value = S
…
```

Пример 7. Вычислить **S1** и **S2** – соответственно суммы элементов с нечетными и четными номерами одномерного массива **X**, состоящего из **N** вещественных значений.

```
` Описание массива, определение размерности массива
` Задаем исходные данные
…
' Вычисление суммы
For I = 1 To N Step 2
  S1 = S1 + X(I):   S2 = S2 + X(I+1)
Next I
' Вывод S1, S2
…
```

В каком случае эта программа работает некорректно и как нужно видоизменить код программы?

Пример 8. Определить значение и номер минимального элемента одномерного массива **A**, состоящего из **N** целочисленных значений.

Алгоритм поиска минимального элемента состоит в следующем. Первоначально считаем, что минимальным является первый элемент, т.е. (**Min = A(1)**; **Num = 1**). Затем, начиная со 2-го, сравниваем **Min** со всеми остальными. Если встречается элемент, значение которого меньше, чем **Min**, «запоминаем» его значение и номер (**Min = A(I), Num = I**).

```
` Описание массива, определение размерности массива
` Задаем исходные данные
…
' Поиск минимального элемента и его номера
Num = 1
Min = A(Num)
For I = 2 To N
  If A(I) < Min Then Min = A(I): Num = I
Next I
' Вывод Min, Num
…
```

Пример 9. Решить задачу примера 8 без использования переменной **Min**.

Решение этой задачи основывается на том, что по номеру элемента массива **Num** мы можем получить его значение **A(Num)**. Поэтому, повторяя алгоритм предыдущей задачи, следует при выполнении условного оператора **Min** заменить на **A(Num)**.

```
` Описание массива, определение размерности массива
` Задаем исходные данные
…
' Поиск минимального элемента и его номера
Num = 1
For I = 2 To N
  If A(I) < A(Num) Then Num = I
Next I
' Вывод A(Num), Num
…
```

Пример 10. Определить сумму положительных элементов вещественной квадратной матрицы **B** размерности **N**, расположенных выше главной диагонали.

Для организации суммирования возможны два подхода:

1) перебирать все элементы и суммировать только те, индексы которых удовлетворяют соотношению **i<j**;

2) сразу выбираются только те элементы, которые необходимо суммировать.

```
` Вариант решения для подхода 2)
…
For I = 1 To N–1
  For J = I + 1 To N
    If B(I, J) > 0 Then S = S + B(I, J)
  Next J
Next I
' Вывод S
…
```

Пример 11. В матрице **B** размерности **N×M** найти номер строки, содержащей наибольшее количество положительных элементов.

Решение задачи заключается в следующем. Внешний цикл (**For I = …**) задает номер обрабатываемой строки. Во внутреннем цикле (**For J = …**) определяется количество положительных элементов в **I**-й строке. По аналогии с поиском минимального элемента определяется **k_max** и **i_max**.

```
` Описание массива, определение размерности массива
` Задаем исходные данные
…
k_max = 1: i_max = 1
For I = 1 To N ' Цикл по строкам
  k = 0         ` Количество >0 в i–й строке
  For J = 1 To M    ' Цикл по колонкам
    ` Считаем количество положительных в i–й строке
    If B(I, J) > 0 Then k = k + 1
  Next J
  If k > k_max Then
    k_max = k       ` Определяем максимальное k
    i_max = I       ` Сохраняем номер строки
  End If
Next I
' Вывод i_max
…
```

9. ОБРАБОТКА СИМВОЛОВ И СТРОК

Далее приведены примеры использования обработки символов и строк с использованием операций и функций для строковых переменных.

9.1. ПРИМЕР ПОИСКА ПЕРВОГО СЛОВА, НАЧИНАЮЩЕГОСЯ ЗАДАННОЙ БУКВОЙ

Пример 12. Предложением будем называть любой набор символов. «Слова» в предложении разделяются пробелом. Найти первое слово, начинающееся заданной буквой.

Решение задачи основывается на поиске начала и конца «слова» в предложении. Используется следующий алгоритм: (**i+1**)-й – номер первой буквы «слова» — это когда (**i+1**)-й символ пробел, а **i**-й символ НЕ пробел; **i**-й — номер последней буквы «слова» — это когда **i**-й символ НЕ пробел, а (**i+1**)-й символ пробел.

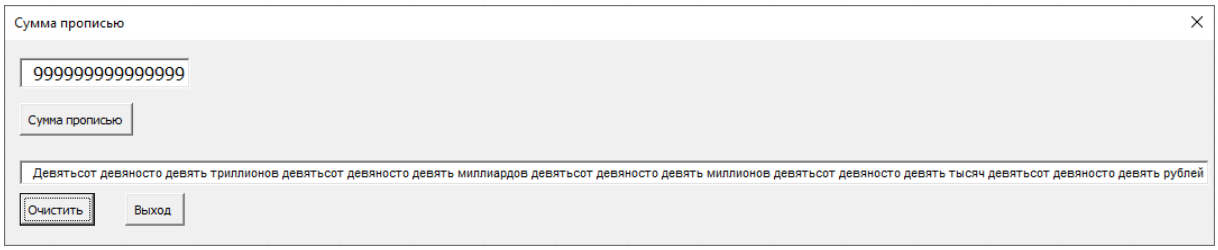
```
Private Sub CB_Res_Click()
Dim R_Str As String, Res As String
Dim Simb As String, rs As String
Dim N_Let As Integer, i As Integer
Dim k0 As Integer, k1 As Integer
Dim Ok As Boolean ' =True- слово найдено, False - нет
  R_Str = TB_Str.Text
  R_Str = LTrim(R_Str)' Убираем пробелы в начале строки
  R_Str = RTrim(R_Str)' Убираем пробелы в конце строки
  N_Let = Len(R_Str)  ' Длина строки

  Simb = TB_Simb.Text ' Буква для поиска

  Ok = Mid(R_Str, 1, 1)= Simb ' Это буква первого слова?
  If Ok Then
    k0 = 1            ' Если - да, то k0 = 1
  Else ' Просматриваем предложение и ищем начало слова
    For i = 1 To N_Let - 1
      If (Mid(R_Str, i, 1) = « «) And _
```

```
        (Mid(R_Str, i+1, 1)<>« «) Then ' Нашли начало
        rs = Mid(R_Str, i+1, 1)' выделяем первую букву
        Ok = rs = Simb          ' Совпадение есть?
        If Ok Then              ' Если – да, то
          k0 = i + 1            ' k0=i+1 – Номер буквы
          Exit For              ' Выход из цикла
        End If
      End If
    Next i
  End If

  If Ok Then ' Если совпадение было, то ищем конец слова
    For i = k0 + 1 To N_Let – 1
      If Mid(R_Str, i, 1) <> " " And _
         Mid(R_Str, i + 1, 1) = " " Then' Нашли конец
        k1 = i                          ' Номер буквы
        If Ok Then Exit For
      End If
    Next i
  End If

  If Ok Then
    If k1 = 0 Then k1 = N_Let      ' т.е. это последнее
    Res = Mid(R_Str, k0, k1 – k0 + 1)
    TB_Res.Text = Res
  Else
    TB_Res.Text = "Нет такого слова"
  End If
End Sub
```

9.2. Пример преобразования целого в символьное: «Сумма прописью»

Пример 13. «Сумма прописью» — преобразование числа из цифрового вида в символьное.

Пример вывода максимально возможного числа для преобразования.



Далее приведен текст процедур и функций.

```
Dim Calculate As Boolean
Dim Left_Form As Integer

Sub SummaString(Summa$, Source As Double, _
                Rod%, w1$, w2to4$, w5to10$)
' Исходные данные:
' Source – заданное число
' далее нужно задать информацию о единице изменения
' Rod%    = 1 – мужской, = 2 – женский, = 3 – средний
'           название единицы изменения:
' w1$     – именительный падеж единств. число (=1)
' w2to4$ – родительный падеж единств. число (=2–4)
' w5to10$ – родительный падеж множеств. число (=5–10)
' Rod% должен быть задано обязательно,
' название единицы может быть не задано = ""
' ------------------------------
' Результат: Summa$ – запись прописью
'==================================
   Dim TempValue As Double
   If Not Calculate Then Exit Sub

   If Source = 0 Then
     Summa$ = RTRIM$("ноль " + w5to10$): Exit Sub
   End If
   TempValue = Source: Summa$ = ""

   ' Единицы
   Call SummaStringThree(Summa$, TempValue, Rod%, _
        w1$, w2to4$, w5to10$)
   If TempValue = 0 Then Exit Sub
```

```
   ' Тысячи
   Call SummaStringThree(Summa$, TempValue, 2, _
       "тысяча", "тысячи", "тысяч")
   If TempValue = 0 Then Exit Sub

   ' Миллионы
   Call SummaStringThree(Summa$, TempValue, 1, _
        "миллион", "миллиона", "миллионов")
   If TempValue = 0 Then Exit Sub

   ' Миллиарды
   Call SummaStringThree(Summa$, TempValue, 1, _
        "миллиард", "миллиарда", "миллиардов")
   If TempValue = 0 Then Exit Sub
   '
   ' Триллионы
   Call SummaStringThree(Summa$, TempValue, 1, _
        "трилллион", "триллиона", "триллионов")
   If TempValue# = 0 Then Exit Sub
   '
   ' Что идет после триллионов, не знаю...
End Sub

Sub SummaStringThree(Summa$, TempValue As Double, _
                     Rod%, w1$, w2to4$, w5to10$)
Dim Rest As Double, R_w As Double
'  Формирование строки для трехзначного числа:
'  (последних трех знаков TempValue)
   If Not Calculate Then Exit Sub
 If TempValue > 999999999999999# + 1 Then
    MsgBox "Извините, не помещается на экране"
    TB_Summa.Value = ""
    TB_Summa.SetFocus
    Calculate = False
    Exit Sub
 Else
   Rest = TempValue Mod 1000
   TempValue = TempValue \ 1000
   If Rest = 0 Then    ' последние три знака нулевые
```

```
    If Summa$ = "" Then Summa$ = w5to10$
    Exit Sub
  End If
  ' начинаем подсчет с Rest
  EndWord$ = w5to10$
  ' Сотни
  Select Case Rest \ 100
    Case 0: s100$ = ""
    Case 1: s100$ = "сто "
    Case 2: s100$ = "двести "
    Case 3: s100$ = "триста "
    Case 4: s100$ = "четыреста "
    Case 5: s100$ = "пятьсот "
    Case 6: s100$ = "шестьсот "
    Case 7: s100$ = "семьсот "
    Case 8: s100$ = "восемьсот "
    Case 9: s100$ = "девятьсот "
  End Select
  '
  ' Десятки
  Rest = Rest Mod 100: Rest1% = Rest \ 10
  s1$ = ""
  Select Case Rest1%
    Case 0: s10$ = ""
    Case 1  ' особый случай
      Select Case Rest
        Case 10: s10$ = "десять "
        Case 11: s10$ = "одиннадцать "
        Case 12: s10$ = "двенадцать "
        Case 13: s10$ = "тринадцать "
        Case 14: s10$ = "четырнадцать "
        Case 15: s10$ = "пятнадцать "
        Case 16: s10$ = "шестнадцать "
        Case 17: s10$ = "семнадцать "
        Case 18: s10$ = "восемнадцать "
        Case 19: s10$ = "девятнадцать "
      End Select
    Case 2: s10$ = "двадцать "
    Case 3: s10$ = "тридцать "
```

```
  Case 4: s10$ = "сорок "
  Case 5: s10$ = "пятьдесят "
  Case 6: s10$ = "шестьдесят "
  Case 7: s10$ = "семьдесят "
  Case 8: s10$ = "восемьдесят "
  Case 9: s10$ = "девяносто "
End Select
'
If Rest1% <> 1 Then  ' единицы
  Select Case Rest Mod 10
    Case 1
      Select Case Rod%
        Case 1: s1$ = "один "
        Case 2: s1$ = "одна "
        Case 3: s1$ = "одно "
      End Select
      EndWord$ = w1$
    Case 2
      If Rod% = 2 Then
        s1$ = "две " Else s1$ = "два "
      End If
      EndWord$ = w2to4$
    Case 3: s1$ = "три ": EndWord$ = w2to4$
    Case 4: s1$ = "четыре ": EndWord$ = w2to4$
    Case 5: s1$ = "пять "
    Case 6: s1$ = "шесть "
    Case 7: s1$ = "семь "
    Case 8: s1$ = "восемь "
    Case 9: s1$ = "девять "
  End Select
End If
'
' Сборка строки
Sum3$ = s100$ + s10$ + s1$ + EndWord$
If Summa$ = "" Then
  Summa$ = Sum3$
Else: Summa$ = RTRIM$(Sum3$) + " " + Summa$
End If
Summa$ = Trim(Summa$)
```

```
 End If
End Sub

Private Sub CB_Clear_Click()
  TB_Summa.Value = ""
  TB_Summa_Propis.Value = ""
  Call Sub_Width_Form
  TB_Summa.SetFocus
End Sub

Private Sub Sub_Width_Form()
Dim Width_Form As Integer
  TB_Summa_Propis.AutoSize = True
  Width_Form = TB_Summa_Propis.Width + 25
  If Width_Form < 200 Then
    Width_Form = 200
    UserForm_Summ.Left = Left_Form
  Else
    UserForm_Summ.Left = 10
  End If
  UserForm_Summ.Width = Width_Form
End Sub

Private Sub CB_Exit_Click()
  End
End Sub

Private Sub TB_Summa_Change()
  TB_Summa_Propis.Text = ""
  Call Sub_Width_Form
End Sub
Private Sub UserForm_Activate()
  Left_Form = UserForm_Summ.Left
End Sub

Private Sub CB_Summa_Propis_Click()
Dim Source As Double
Dim Result$
Calculate = True
```

```
If IsNumeric(TB_Summa.Value) Then
  Source = CDbl(TB_Summa.Value)
  Call SummaString(Result$, Source, 1, _
                   "рубль", "рубля", "рублей")
  '  Установка первой буквы в формате прописной
  If Calculate Then
    Mid$(Result$, 1, 1) = UCase(Mid$(Result$, 1, 1))
  Else
    Call CB_Clear_Click
    Exit Sub
  End If
  TB_Summa_Propis.Text = Result$
  CB_Clear.SetFocus
Else
  MsgBox "Извините, только числа", 0, _
         "Будьте внимательнее..."
  Call CB_Clear_Click
  TB_Summa.SetFocus
End If
Call Sub_Width_Form
End Sub
```

10. ПОДПРОГРАММЫ

Ранее были рассмотрены структура и примеры процедур обработки некоторых событий (как правило, это были события нажатия кнопки **Click**). Эти процедуры автоматически создаются при двойном щелчке на кнопке в окне редактирования формы.

Однако в программировании весьма типична ситуация, когда в разных местах программы приходится выполнять по сути дела один и тот же частичный алгоритм, который имеет достаточно самостоятельное значение, т.е. предназначен для решения некоторой подзадачи, выделенной из основной решаемой задачи, например, нахождение наибольшего общего делителя двух натуральных чисел, упорядочение компонент вектора, решение системы линейных алгебраических уравнений и т.д. Если этот частичный алгоритм достаточно сложен и представляется большим фрагментом программы, то было бы нерационально

выписывать его каждый раз заново в том месте программы, где этот частичный алгоритм должен использоваться.

Для обеспечения большей компактности программы и повышения ее наглядности можно выделить любой частичный алгоритм из основного ее текста и записать его только один раз, представив этот частичный алгоритм в качестве самостоятельного программного объекта, называемого процедурой (или подпрограммой).

С помощью списка фактических параметров конкретизируются значения формальных параметров.

Различные особенности описаний процедур и способы их использования будем иллюстрировать на примере следующей задачи: по заданным вещественным *x* и *y* вычислить:

$$u = \max(x + y, x * y),\ v = \max(u, 10).$$

Программу решения этой задачи без использования процедуры для поиска максимума можно записать в виде:

```
Private Sub CB_Res_Click()
Dim x, y, U, V    ' Описываем используемые переменные
  ' Задаем исходные данные x, y
  x = Val(TB_x.Value)
  y = Val(TB_y.Value)

  If x + y > x * y Then U = x + y Else U = x * y
  If U > 10 Then V = U Else V = 10

  ' Выводим результат U, V
  TB_U.Value = U
  TB_V.Value = V
End Sub
```

В этой программе используются два условных оператора, каждый из которых предназначен для решения, по сути дела, одной и той же задачи: нахождения большего из двух заданных вещественных значений и присваивания некоторой переменной полученного результата. Поэтому алгоритм решения этой частичной задачи целесообразно объявить процедурой.

10.1. Процедуры без параметров

Чтобы подчеркнуть сходство двух упомянутых условных операторов, запишем программу несколько иначе:

```
Private Sub CB_Res_Click()
Dim a, b, r  ' Вспомогательные переменные
Dim x, y, U, V   ' Описываем используемые переменные
  ' Задаем исходные данные x, y
…
  a = x + y: b = x * y
  If a > b Then r = a Else r = b
  U = r
  a = 10: b = U
  If a > b Then r = a Else r = b
  V = r
  ' Выводим результат U, V
…
End Sub
```

Условные операторы в точности совпадают друг с другом, поэтому можно считать, что один и тот же условный оператор присутствует в программе дважды. Чтобы избежать двукратного выписывания этого оператора в разделе операторов программы, оформим его в виде процедуры:

```
Private Sub Max1()
  If a > b Then r = a Else r = b
End Sub
```

Для вызова этой процедуры в нужном месте программы необходимо указать оператор вызова **`Call Max1`**.

Для того чтобы значения вспомогательных переменных были доступны всем используемым процедурам, они должны быть глобальными, т.е. их описание необходимо поместить в начало программного кода.

Тогда программа примет вид:

```
Dim a, b, r  ' Глобальные переменные
Private Sub Max1()
  If a > b Then r = a Else r = b
End Sub
```

```
Private Sub CB_Res_Click()
Dim x, y, U, V   ' Описываем используемые переменные
  ' Задаем исходные данные x, y
…
  a = x + y: b = x * y
  Call Max1
  U = r
  a = 10: b = U
  Call Max1
  V = r
  ' Выводим результат U, V
…
End Sub
```

Выполнение каждого из фигурирующих здесь операторов процедуры **Max1** сводится к выполнению тела процедуры с этим именем.

Поскольку в данном случае фрагмент текста программы, объявленный процедурой, весьма невелик, то существенного выигрыша в размере общего текста программы мы не получили, в противном случае выигрыш был бы очевиден. Но даже и в нашем случае основная часть программы — ее раздел операторов — стала более компактной и наглядной по сравнению с предыдущим вариантом.

10.2. Процедуры с параметрами

10.2.1. Параметры-значения

Введенная нами в употребление процедура **Max1** не очень удобна для использования, поскольку ее назначение зафиксировано слишком жестко. В частности, исходными данными для нее могут служить только значения переменных **a** и **b**. Поэтому перед каждым обращением в процедуре приходится предварительно присваивать этим переменным те значения, из которых нужно выбирать большее. Чтобы снять это ограничение и тем самым обеспечить общность процедуры и повысить удобство ее использования, можно не фиксировать те исходные

значения, к которым должна применяться процедура, а сделать их параметрами процедуры, для того, чтобы было удобно конкретизировать их при каждом обращении к ней.

С этой целью не будем заранее фиксировать те значения, из которых процедура должна выбрать большее, а обозначим их формально некоторыми идентификаторами, не используемыми в теле процедуры.

Такие идентификаторы называются *формальными параметрами* процедуры, поскольку они представляют не какие-то конкретные значения, а значения «вообще». При каждом обращении к процедуре ее формальные параметры должны конкретизироваться, поэтому для упрощения последующих обращений к процедуре ее формальные параметры явно указываются в заголовке процедуры и тем самым упорядочиваются по их перечислению. При этом для каждого формального параметра может быть указан тип значения, представляемого этим параметром.

```
Private Sub Max2(r1, r2)
  If r1 > r2 Then r = r1 Else r = r2
End Sub
```

При обращении к такой процедуре в соответствующем операторе процедуры вслед за именем процедуры необходимо в круглых скобках задать список фактических параметров, конкретизирующих те значения, к которым должна применяться процедура и которые в ее теле были обозначены формальными параметрами. При этом соответствие между фактическими и формальными параметрами устанавливается путем их сопоставления в обоих списках: первый по порядку фактический параметр соответствует первому формальному параметру, второй фактический параметр — второму формальному параметру и т.д.

Теперь программу можно записать так:

```
Dim r    ' Глобальная переменная

Private Sub Max2(ByVal r1, ByVal r2)
  If r1 > r2 Then r = r1 Else r = r2
End Sub
Private Sub CB_Res_Click()
Dim x, y, U, V   ' Описываем используемые переменные
  ' Задаем исходные данные x, y
…
  Call Max2(x + y, x * y)
  U = r
  Call Max2(10, U)
  V = r
  ' Выводим результат U, V
…
End Sub
```

Таким образом, при обращении к процедуре в ней вводятся в употребление свои внутренние переменные **r1, r2**. Эти переменные существуют только во время выполнения процедуры. При входе в процедуру этим внутренним переменным присваиваются значения, заданные соответствующими фактическими параметрами в операторе процедуры, — они и используются при выполнении процедуры.

Как видно, теперь в основной программе отпала необходимость вводить в употребление переменные **a** и **b**, а перед обращением к процедуре — присваивать этим переменным соответствующие значения. Благодаря этому раздел операторов в основной программе стал еще более компактным и наглядным.

Рассмотренные здесь формальные параметры процедуры носят название *параметры-значения* (**ByVal**[1]), поскольку каждый из них в теле процедуры представляет некоторое значение, задаваемое при обращении к процедуре с помощью соответствующего фактического параметра. Фактическим параметром в этом случае может быть любое выражение того же типа, что и тип формального параметра, в частности, константа

[1] Сокр. от By Value – по значению.

или переменная соответствующего типа как частный случай выражения.

Следует подчеркнуть, что в данном случае фактические параметры используются только при входе в процедуру с целью передачи задаваемых ими значений тем внутренним переменным процедуры, которые поставлены в соответствие ее формальным параметрам-значениям. После этого фактические параметры недоступны из процедуры, так что она не в состоянии ни использовать их каким-либо иным способом, ни изменить значение переменной, являющейся фактическим параметром.

10.2.2. Параметры-переменные

Недостаток процедуры **Max2** заключается в том, что найденное большее из двух значений она всегда присваивает одной и той же переменной **r**. В нашей же задаче в одном случае найденный результат надо присвоить переменной **U**, а в другом — переменной **V**. Поэтому после оператора процедуры в программе приходилось записывать дополнительные операторы присваивания.

Этот недостаток можно устранить, если в процедуре не фиксировать переменную, которой присваивается найденное значение, а сделать ее тоже параметром процедуры, обозначив эту переменную, например, идентификатором **Res**, который также будет формальным параметром процедуры. Однако этот параметр существенно отличается от формальных параметров **r1** и **r2**: он в теле процедуры должен представлять не значение, являющееся одним из исходных данных этой процедуры, а некоторую переменную, существующую вне тела процедуры. И чтобы процедура могла присвоить значение такой переменной, необходимо обеспечить непосредственный доступ к ней из процедуры.

Перед *параметром-переменной* в списке формальных параметров записывается ключевое слово **ByRef**[1], а перед *параметром-значением* — **ByVal**. После формального параметра-переменной по-прежнему должен быть указан ее тип. В отличие от

[1] Сокр. от By Reference – по ссылке.

формального параметра-значения, фактическим параметром для которого может быть любое выражение соответствующего типа, для параметра-переменной фактическим параметром может быть только переменная, но не любое выражение. В этом случае нашу программу можно записать так:

```
Private Sub Max3(ByVal r1, ByVal r2, ByRef Res)
  If r1 > r2 Then Res = r1 Else Res = r2
End Sub
```

```
Private Sub CB_Res_Click()
Dim x, y, U, V   ' Описываем используемые переменные
  ' Задаем исходные данные x, y
…
  Call Max3(x + y, x * y, U)
  Call Max3(10, U, V)
  ' Выводим результат U, V
…
End Sub
```

Теперь раздел операторов основной программы получился предельно компактным и наглядным — в нем не осталось никаких вспомогательных операторов.

Итак, если формальный параметр объявлен параметром-переменной, то процедура получает непосредственный доступ к той переменной, которая задана в качестве соответствующего фактического параметра. В связи с этим процедура может непосредственно использовать и изменять значение этой переменной и тем самым передавать в основную программу вырабатываемые ею результаты. Если говорить более точно, то при обращении к процедуре ей передается *ссылка* на переменную, заданную в качестве фактического параметра — эта ссылка и используется процедурой для доступа к этой переменной.

10.3. Синтаксис процедур

После того как мы познакомились с понятием процедуры и рассмотрели примеры описаний процедур, дадим более точные определения этих понятий.

```
[Public | Private] [Static] Sub имя [(списокАрг)]
    [инструкции]
    [Exit Sub]
    [инструкции]
End Sub
```

Синтаксис инструкций содержит следующие элементы:

Public Используется по умолчанию. Указывает, что процедура доступна для всех других процедур во всех модулях.

Private Необязательный. Указывает, что процедура доступна для других процедур только того модуля, в котором она описана.

Static Необязательный. Указывает, что локальные переменные процедуры сохраняются в промежутках времени между вызовами этой процедуры. Атрибут **Static** не действует на переменные, описанные вне процедуры, даже если они используются в процедуре.

имя Указывать обязательно. Имя процедуры, удовлетворяющее стандартным правилам именования переменных.

списокАрг Необязательный. Список переменных (формальных параметров), которые передаются в процедуру при ее вызове.

инструкции Любая группа инструкций, выполняемых в процедуре.

Аргумент **списокАрг** имеет следующий синтаксис и элементы:

```
[ByVal | ByRef] имяПерем [( )] [As тип]
```

ByVal Необязательный. Указывает, что этот аргумент передается по значению.

ByRef Необязательный. Указывает, что этот аргумент передается по ссылке. Описание **ByRef** используется в Visual Basic по умолчанию.

имяПерем Обязательный. Имя переменной, удовлетворяющее стандартным правилам именования переменных.

тип Необязательный. Тип данных аргумента, переданного в процедуру; поддерживаются типы **Byte**, **Boolean**, **Integer**,

Long, **Currency**, **Single**, **Double**, **Decimal**, **Date**, **String** (только строки переменной длины), **Object**, **Variant**.

Инструкция **Exit Sub** приводит к немедленному выходу из процедуры **Sub**. Выполнение программы продолжается с инструкции, следующей за инструкцией, содержащей вызов процедуры.

10.4. Вызов процедур

Процедура **Sub** вызывается в выражении по своему имени, за которым следует список аргументов в скобках.

[Call] имя [списокАрг]

Call Необязательное ключевое слово. Если указано, необходимо заключить **списокАрг** в круглые скобки.

имя Обязательный. Имя вызываемой процедуры.

списокАрг Необязательный. Разделяемый запятыми список переменных, массивов или выражений, передаваемых в процедуру. Компоненты **спискаАрг** могут включать ключевые слова **ByVal** или **ByRef** для описания того, каким образом аргументы будут рассматриваться вызываемой процедурой. Однако ключевые слова **ByVal** и **ByRef** могут использоваться с инструкцией **Call** только при вызове процедуры из библиотеки динамической компоновки.

Не обязательно применять ключевое слово **Call** при вызове процедуры. Однако, если ключевое слово **Call** используется для вызова процедуры с аргументами, **списокАрг** должен быть заключен в скобки. Если ключевое слово **Call** не используется, **списокАрг** записывается без скобок.

Для передачи в процедуру полного массива следует воспользоваться именем массива с пустыми скобками.

10.5. Правила соответствия между формальными и фактическими параметрами

1. Число фактических параметров в операторе процедуры должно быть равно числу формальных параметров в описании процедуры. Если у процедуры нет параметров, то оператор процедуры представлен только ее именем.

2. При записи оператора процедуры соответствие между фактическими и формальными параметрами устанавливается путем их сопоставления слева направо в соответствующих списках: первый фактический параметр ставится в соответствие первому формальному, второй фактический параметр — второму формальному и т.д.

3. Тип каждого фактического параметра должен соответствовать типу формального параметра.

4. При задании фактического параметра надо следить за тем, что представляет в процедуре соответствующий ему формальный параметр — значение или переменную.

Если это параметр — переменная (перед ним записывается ключевое слово **`ByRef`**), то фактическим параметром может быть только переменная, причем переменная того же типа, что и тип формального параметра. При этом допускается как полная, так и частичная переменная, например, переменная с индексами.

Если же формальный параметр является параметром-значением (перед ним записывается ключевое слово **`ByVal`**), то в качестве соответствующего ему фактического параметра можно задавать любое выражение соответствующего типа, в частности, константу или переменную, которые являются частным случаем выражения.

10.6. Принцип локализации

Переменные, используемые в подпрограммах, разбиваются на две категории: явно описанные внутри процедуры (*локальные параметры*) и не описанные внутри процедуры (*глобальные параметры*). Переменные, которые явно описаны в процедуре (с помощью ключевого слова **`Dim`** или эквивалентного ему), всегда

являются локальными для этой процедуры. Переменные, которые используются, но явно не описаны в процедуре, также являются локальными, если они явно не описаны на более высоком уровне.

Глобальные параметры — это переменные, не являющиеся формальными параметрами и не описанные в теле процедуры.

Локальные параметры — это переменные, не являющиеся формальными параметрами, но описанные в теле процедуры.

Принцип локализации заключается в том, что если какая-либо переменная описана в теле процедуры, то внутри процедуры она приобретает смысл в соответствии с этим описанием.

Если же эта переменная была описана вне тела процедуры, то действие предыдущего описания временно (до выхода из процедуры) приостанавливается. При выходе из процедуры утрачивает свою силу описание, содержащееся в теле процедуры, и этот идентификатор вновь приобретает тот же смысл, который он имел до входа в процедуру (если этот идентификатор был описан вне процедуры).

Как видно, принцип локализации предоставляет достаточно большую свободу в выборе обозначений при описании процедур и тем самым обеспечивает параллельную и независимую работу различных исполнителей.

11. ФУНКЦИИ

В математике понятие функции хорошо известно — с ее помощью задаются самые различные зависимости одних значений (являющихся значением функции) от других, называемых аргументами функции. Как известно, эти зависимости могут задаваться различными способами: таблично, графически, аналитически и т.д. В отличие от общематематического понятия функции, в алгоритмических языках рассматриваются только такие, для которых можно задать алгоритм определения их значений.

При этом в VBA допустимы только такие функции, значения которых относятся к простым типам, так что значением функции не может быть, например, массив. В частности, каждое арифметическое или логическое выражение, даже не

использующее понятия функции, определяет некоторую функциональную зависимость.

Однако далеко не каждую функциональную зависимость, допустимую в алгоритмическом языке, можно задать в виде такого выражения. В ряде случаев значение функции определяется достаточно сложным вычислительным процессом (процедурой). Например, хорошо известную функциональную зависимость, обозначаемую в математике через *N!*, невозможно задать в виде арифметического выражения (заметим, что часто используемая в математике запись вида $1 \cdot 2 \cdot 3 \cdot \ldots \cdot N$ на алгоритмическом языке невозможна, поскольку синтаксис языка не допускает использования многоточия). Поэтому алгоритм вычисления значения *y=N!* придется задать некоторой последовательностью операторов, например:

```
nf = 1
For I = 1 To N
  nf = nf * I
Next I
```

Если необходимость использования какой-либо функциональной зависимости встречается в нескольких местах программы, то было бы нерационально каждый раз выписывать соответствующий алгоритм. Как и в случае процедур, удобнее однажды определить требуемую функциональную зависимость, дав ей некоторое имя, а при необходимости использовать эту зависимость путем указания ее имени и задания конкретных значений аргументов.

Процедуры, предназначенные для определения функциональных зависимостей, будем называть *подпрограммы-функции,* или *функции*.

11.1. Описание функций

Синтаксис описания функции очень похож на синтаксис описания процедуры:

```
[Public | Private] [Static] Function имя [(списокАрг)]
[As тип]
    [инструкции]
    имя = выражение
    [Exit Function]
    [инструкции]
    имя = выражение
End Function
```

Назначение и смысл элементов описания функций полностью соответствует описанию процедур.

Как обычно, вводимому в употребление программному объекту дается **имя**. За именем в общем случае следует взятый в круглые скобки список формальных параметров, представляющих аргументы описываемой функции. Параметрами функции также могут быть как параметры-значения, так и параметры-переменные. При этом аргументы могут иметь разные типы, так что и в этом отношении функции ничем не отличаются от процедур.

Заголовок функции завершается указанием типа значения описываемой функции. Возможность явного указания типа значений описываемой функции диктуется двумя обстоятельствами. Во-первых, для определения типа арифметического выражения следует знать тип каждой из используемых в нем функций. Во-вторых, это необходимо для контроля правильности использования данной функции в программе, т.е. для повышения надежности программы.

Как уже было отмечено, в общем случае значение функции определяется некоторой вычислительной процедурой, в процессе выполнения которой может вычисляться довольно много различных, в том числе и промежуточных, результатов. Для однозначного определения результата выполнения функции в ее

разделе операторов обязательно должен присутствовать хотя бы один оператор присваивания вида

имя = выражение

который и означает, что в качестве значения функции принимается значение заданного в нем выражения. Операторов присваивания указанного вида может быть и несколько, но хотя бы один из них должен выполняться в процессе выполнения тела процедуры. Результат последнего выполнения оператора присваивания и принимается в качестве окончательного значения.

Например, функцию F(N) = N! можно описать следующим образом:

```
Function Fact(N As Integer) As Integer
Dim nf As Integer
  nf = 1
  For I = 1 To N
    nf = nf * I
  Next I
  Fact = nf
End Function
```

Сразу же обратим внимание на возможные типичные ошибки.

1. В левой части оператора присваивания, определяющего значение функции, наряду с именем функции записывается и ее аргумент, например, **Fact(N) = nf**. Во-первых, такая запись недопустима по синтаксису (в левой части фактически записано выражение, каковым является вызов функции). Во-вторых, запись **Fact(N)** означает, что надо вычислить значение функции **Fact** при указанном в скобках аргументе. Но ведь здесь речь идет не о том, чтобы вычислить значение функции, а об указании того, что в качестве значения функции надо принять текущее значение переменной **nf**.

2. Имя функции используется в правой части оператора присваивания как имя переменной, например:

```
Function Fact(N As Integer) As Integer
  Fact = 1
  For I = 1 To N
    Fact = Fact * I
  Next I
End Function
```

Такое описание функции **Fact** неверно, потому что запись **Fact*I** — не арифметическое выражение. В самом деле, в этой записи **Fact** может быть либо переменной, либо вызовом функции без параметров. Однако согласно данному описанию идентификатор **Fact** есть имя функции, а не переменной. Вызовом функции запись **Fact** тоже быть не может, так как не содержит указания на количество фактических и формальных параметров в заголовке функции при ее описании.

11.2. Вызов функции

Как мы уже знаем, в VBA предусмотрен определенный набор стандартных функций — эти функции используются в любой программе без их явного описания.

Если для вызова процедуры (с целью ее активации) служит оператор **Call** (или просто указание имени процедуры), то для обращения к функции — вызов функции. Заметим, что термин «функция» означает определенный программный объект, а «вызов функции» — некоторую синтаксическую конструкцию, с помощью которой в программе задается обращение к этому программному объекту с целью его активации. Однако ради краткости изложения мы будем иногда использовать термин «функция» и в качестве синонима термина «вызов функции», где это не приводит к неоднозначности понимания.

Для получения значения функции (т.е. ее вызова) необходимо в выражении соответствующего типа указать ее имя с требуемыми фактическими параметрами. Так, если требуется вычислить значение P = 5! – a + (k + 2)!, то, используя функцию **Fact**, это можно сделать так:

```
  P = Fact(5) - a + Fact(k+2)
```

Здесь предполагается, что значения переменных **a** и **k** определяются до выполнения этого оператора.

Преимущества или недостатки применения процедуры или функции можно оценить только в каждом конкретном случае.

Если вычисляемое значение чаще принимается за промежуточное значение при вычислении выражений, то лучше использовать функцию. Если же это значение надо присвоить некоторой переменной, то это можно сделать с использованием процедуры, поручив ей и выполнение присваивания вычисленного значения задаваемой переменной.

Конечно, о предпочтительности того или иного вида подпрограмм можно говорить только в том случае, если в результате получаем единственное значение. Если же результат — значение производного типа (например, массив), то здесь функции вообще не применимы из-за ограничения на допустимые типы значений функций (не говоря уже о процедурах, результатом выполнения которых является некоторое действие — печать какого-то сообщения на принтере, ввод данных и т.п.).

11.3. Побочный эффект функции

Термином «главный эффект функции», который не употребляется, можно было бы назвать вычисление значения функции, поставляемое в программу в качестве значения вычисляемого операнда в каком-либо выражении.

До сих пор мы исходили из того, что этот «главный эффект» функции — единственный в том смысле, что вне процедуры-функции невозможно обнаружить каких-либо последствий ее выполнения. Все рассматривавшиеся примеры функций обладали именно таким свойством (заметим, что локализованные в процедуре переменные существуют только на время ее выполнения, а факт их возникновения и присваивания им каких-то значений невозможно обнаружить вне процедуры).

На самом деле Basic допускает и такие функции, которые — наряду с определением значения функции — могут выполнять такие действия, результат которых обнаруживается и вне процедуры: изменять значения глобальных для нее переменных,

производить вывод и т.п. Результат выполнения подобного рода действий и называется побочным эффектом функции.

Термин «побочный эффект» взят из фармакологии: каждое лекарство направлено на устранение какой-то определенной болезни, но попутно способствует лечению и других болезней либо обостряет другие болезни. При этом значимость побочного эффекта может не уступать главному эффекту. Побочные эффекты функций тоже могут быть как весьма удобными и полезными, так и вызывать различные неприятности. Поэтому функции с побочным эффектом надо использовать очень осторожно, а без достаточного опыта работы следует избегать вводить их в употребление. Они особенно нежелательны в программах для широкого распространения и требующих последующего их сопровождения.

Приведем простейший пример функции с побочным эффектом.

```
Dim A, B

Function Fun(x)
 Fun = x * x – 1: A = 0
End Function

Private Sub CB_Res_Click()
  A = 3           ' Задаем исходные данные
  B = Fun(A)      ' Вызов функции

  TB_a.Value = A ' Выводим результат
  TB_b.Value = B
End Sub
```

Может показаться, что результатом вывода будут числа **3** и **8**. Однако это не так.

Рассмотрим подробнее выполнение приведенного примера. В первом операторе присваивания переменной **A** получает значение **3**. При вычислении арифметического выражения в правой части второго оператора присваивания производится обращение к функции **Fun**. При этом локальной переменной **x**

(формальный параметр) присваивается значение **3**. При выполнении первого оператора тела процедуры определяется значение функции, равное **8**. И поскольку переменная **A** для функции является глобальной, то во втором операторе тела процедуры этой переменной (которой до обращения к процедуре было присвоено значение **3**) присваивается новое значение, равное нулю — это и есть побочный эффект функции. Таким образом, в результате выполнения оператора присваивания **B = Fun(A)** переменной **B** будет присвоено значение **8**, а (побочный эффект функции) переменной **A** — значение **0**. Так что на самом деле будут выведены числа **0** и **8**.

Побочный эффект функции может проявляться и в том, что функция изменит значение фактического параметра-переменной за счет получения непосредственного доступа к этому фактическому параметру для использования и изменения его значения.

11.4. Рекурсивные функции

С понятием рекурсии мы уже встречались при рассмотрении металингвистических формул, используемых для синтаксического определения понятий языка — это случай, когда какое-либо понятие определяется с использованием этого же самого понятия.

Рекурсивным может быть и определение функции. Классический пример: факториал — функция F(N) = N! Эту функцию можно определить различными способами, в том числе и рекурсивно:

$$N!=\begin{cases}1, & N=0,\\ N\cdot(N-1)!, & N>0.\end{cases}$$

Здесь N! определяется через (N – 1)!, т.е. через эту же самую функцию.

Особенность рекурсивного описания функции состоит в том, что в теле такой процедуры-функции содержится обращение к этой же описываемой функции. Например, можно дать следующее рекурсивное описание функции N!:

```
Function Fact_R(N As Integer) As Integer
Dim nf As Integer
  nf = 1
  If N=0 Then
    Fact_R = 1
  Else
    Fact_R = N * Fact_R(N-1)
  End If
End Function
```

Отметим, что здесь имя функции **Fact_R** встречается как в левой, так и в правой части оператора присваивания. Однако вхождение имени в левую часть — это еще не рекурсия: по этому имени не производится обращения к функции, а лишь указывается, что значение выражения в правой части этого оператора должно быть принято в качестве значения определяемой функции. А вот наличие вызова определяемой функции в правой части какого-либо оператора присваивания (именно вызова функции с заданием необходимого количества фактических параметров), в данном примере вызова функции **Fact_R(N-1)**, свидетельствует об обращении к той же самой функции в процессе вычисления ее значения, т.е. о рекурсивности определения.

В данном случае имеет место явная рекурсия: обращение **Fact_R(N-1)** к описываемой функции определенно есть в теле описания этой функции.

Процесс вычисления рекурсивной функции рассмотрим на примере вызова функции **Fact_R(3)**. При входе в функцию по этому вызову, как обычно, вводится в употребление локальная переменная **N**, соответствующая формальному параметру-значению, и ей присваивается значение фактического параметра, после чего выполняется раздел операторов процедуры. Поскольку здесь **N = 3**, то на самом деле выполняется оператор **Fact_R = 3 * Fact_R(2).**

В процессе вычисления арифметического выражения в правой части этого оператора опять производится обращение к функции для вычисления **Fact_R(2)**. При обращении к функции **Fact_R** опять вводится в употребление теперь уже новая локальная переменная, соответствующая формальному

параметру-значению. Чтобы подчеркнуть, что это другая переменная, обозначим ее через **n1**. Этой переменной присваивается значение фактического параметра, так что **n1=2**, и поскольку **n1≠0**, то выполнение условного оператора в теле процедуры сводится к выполнению оператора **Fact_R = 2 * Fact_R(1)**.

При этом для вычисления **Fact_R(1)** снова производится обращение к процедуре **Fact_R**, вводится в употребление новая локальная переменная **n2** с присвоенным значением **n2=1**, и поскольку **n2≠0**, выполнение тела процедуры сводится к выполнению оператора **Fact_R = 1 * Fact_R(0)**.

Заметим, что при этом каждый раз откладывалось завершение вычисления выражения в правой части оператора присваивания. При очередном обращении к функции получим **n3=0**, а выполнение тела процедуры сведется к выполнению оператора **Fact_R=1**.

По получении нами этого значения завершается последовательное выполнение операторов **Fact_R = 1 * Fact_R(0)** со значением **Fact_R(1) = 1**; далее **Fact_R = 2 * Fact_R(1)**, что дает равным **Fact_R(2) = 2**, и наконец, **Fact_R = 3 * Fact_R(2)**, что и дает искомый результат **Fact_R(3) = 3 * 2 =6**.

Любое рекурсивное определение функции можно заменить и нерекурсивным, с использованием оператора цикла.

Как видно, рекурсивность — это свойство не самой функции, а ее описания. При выборе описания (рекурсивного или нерекурсивного) нужно помнить: рекурсивное описание обычно короче и нагляднее, но на вычисление рекурсивной функции затрачивается больше машинного времени (за счет повторных обращений к функции) и памяти (за счет дублирования локализованных в процедуре переменных).

11.5. Пример использования функций

Рассмотрим задачу определения «счастливого» числа. Счастливым числом будем называть число, состоящее из **2k** цифр, таких, что сумма первых **k** цифр равнялась сумме последних **k**

цифр. Например, **326407** — счастливое **(3+2+6 = 4+0+7**), а **123456** – нет.

Для решения этой задачи нам нужно определять количество цифр и сумму цифр числа. Для определения количества цифр воспользуемся уже имеющимся алгоритмом, который оформим в виде функции **Kol_vo_Number**. Для вычисления суммы цифр произвольного числа создадим функцию **Summ_Number**.

```
Private Sub CB_Happy_Click()
Dim N As Long          ' Заданное число
Dim k0 As Integer      ' Количество цифр в числе
Dim st As Integer      ' Степень 10
Dim N1 As Long         ' Число из первых k цифр
Dim N2 As Long         ' Число из последних k цифр
Dim k1 As Integer      ' Сумма первых k цифр
Dim k2 As Integer      ' Сумма последних k цифр
  N = Val(TB_N.Value)
  ' Если ничего не ввели
  ' выводим сообщение, завершаем работу
  If (N = 0) Or (Trim$(TB_N.Value) = "") Then '
    MsgBox Chr(13) + "!!! Введите число !!! " + Chr(13)
    TB_N.SetFocus
    Exit Sub
  End If

' Вызываем функцию для определения
' количества цифр в исходном числе
  k0=Kol_vo_Number(N)' Количество цифр в исходном числе
  If k0 Mod 2 <> 0 Then
  ' Если количество цифр нечетное, то
  ' выводим сообщение, очищаем поля, завершаем работу
    MsgBox " Количество цифр числа " + TB_N.Text + _
           Chr(13) + " должно быть ЧЕТНЫМ !!! "
    TB_N.Value = ""
    TB_N.SetFocus
    Exit Sub
  End If
```

```
  st = 10 ^ (k0 \ 2)  ' Степень 10
' Выделяем первые k цифр и находим их сумму
  N1 = N \ st
  k1 = Summ_Number(N1)
  TB_N1.Value = k1

' Выделяем последние k цифр и находим их сумму
  N2=N Mod st:  k2=Summ_Number(N2): TB_N2.Value=k2

  If k1 = k2 Then
    L_Happy.Caption = "!!! СЧАСТЬЕ !!! "
  Else
    L_Happy.Caption = "В следующий раз... "
  End If
  TB_N1.Visible = True
  TB_N2.Visible = True
End Sub
' Функция для определения количества цифр числа
Public Function Kol_vo_Number(ByVal Nr As Long) As
Integer
Dim k_N As Integer    ' Количество цифр числа
  k_N = 0
  While Nr <> 0       ' Начало цикла, проверка условия
    k_N = k_N + 1     ' Увеличение счетчика
    Nr = Nr \ 10      ' Удаление младшего разряда
  Wend                ' Конец цикла
  Kol_vo_Number = k_N ' Результат присваивается имени
End Function

' Функция для определения суммы цифр числа
Public Function Summ_Number(ByVal Nr As Long) As
Integer
Dim s_N As Integer    ' Сумма цифр числа
Dim b As Integer      ' Цифра младшего разряда
  s_N = 0
  Do                  ' Начало цикла
    b = Nr Mod 10' Выделяем цифру младшего разряда
    s_N = s_N + b' Суммируем полученную цифру
    Nr = Nr \ 10 ' Удаление младшего разряда
```

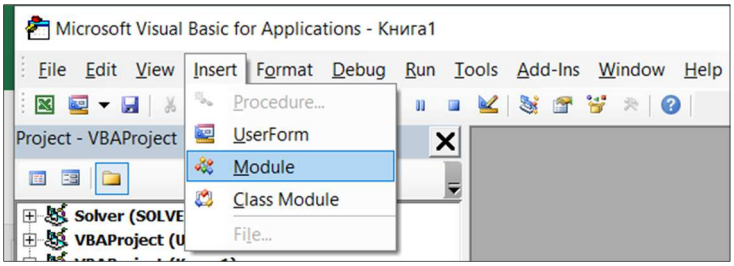
```
  Loop Until Nr = 0   ' Конец цикла, проверка условия
  Summ_Number = s_N   ' Результат присваивается имени
End Function

Private Sub CB_Exit_Click()
  End
End Sub
```

11.6. Создание процедур (функций) пользователя

Мы рассмотрели процесс разработки форм и создания процедур обработки событий.

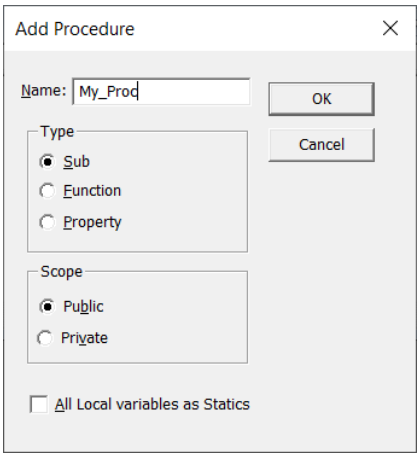
VBA позволяет создавать процедуры или функции пользователя, которые располагаются в модулях. Для создания процедуры пользователя нужно выполнить следующие действия:

– если в проекте нет модуля, то нужно создать его, выполнив команду в меню редактора Вставка > Модуль (Insert > Module);

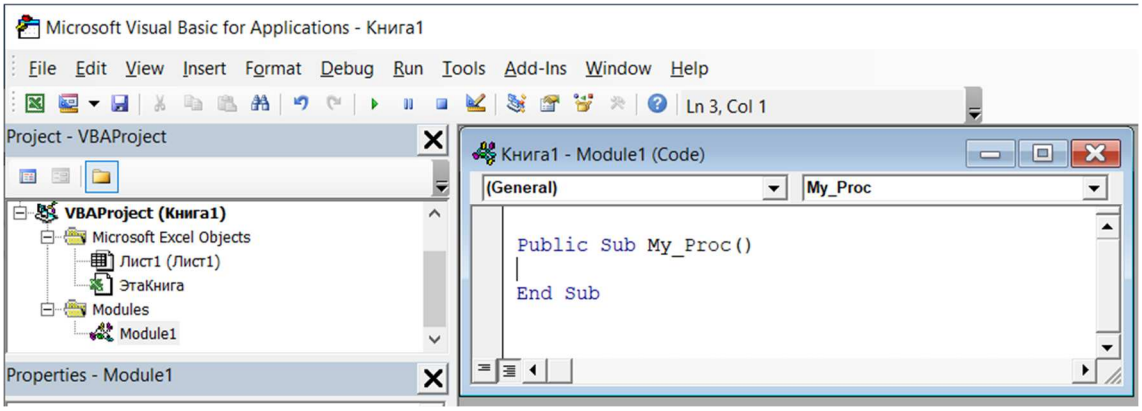


– двойным щелчком активизировать его, перейти в окно редактирования кода модуля и выполнить команду из меню редактора Вставка > Процедуры (Insert > Procedure);

– в появившемся диалоговом окне Add Procedure ввести имя процедуры (или функции), определить ее тип, область действия и нажать OK.



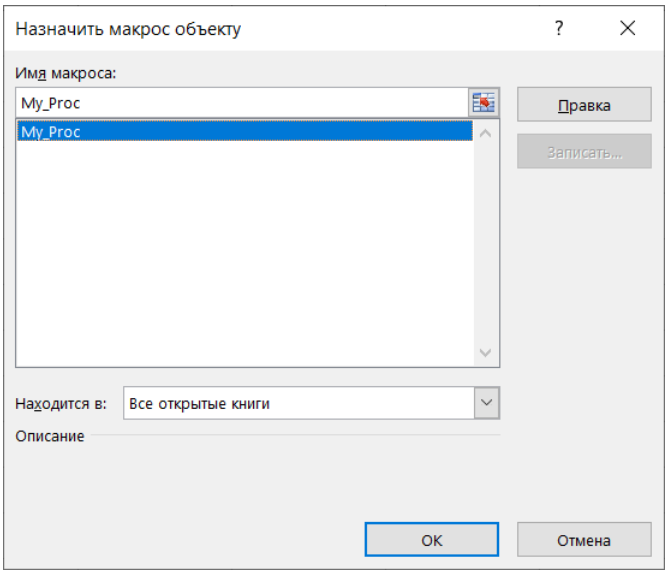
После выполнения этих действий в окне кода модуля появится заготовка процедуры, куда нужно поместить код программы.



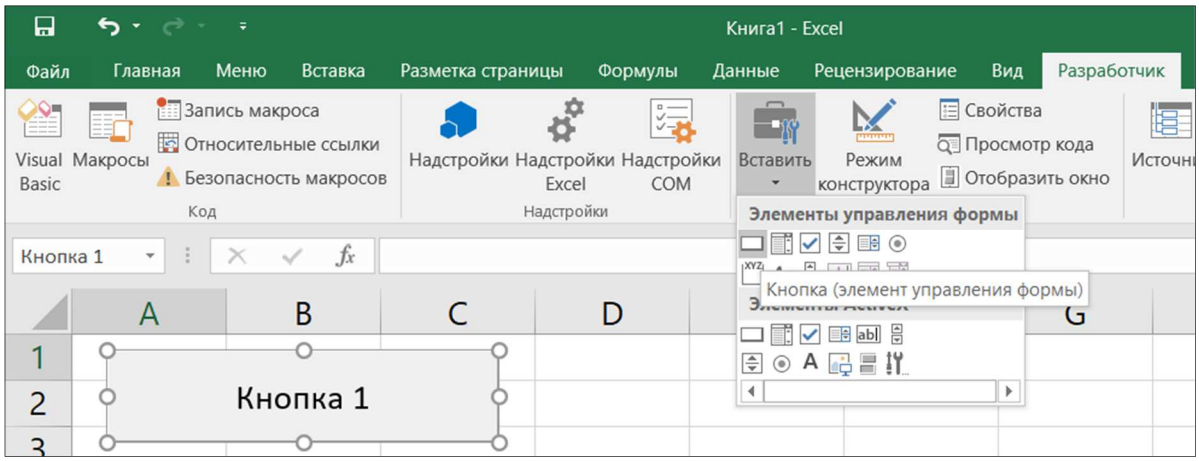
Для активизации (вызова для выполнения) созданной процедуры (функции) возможны два подхода.

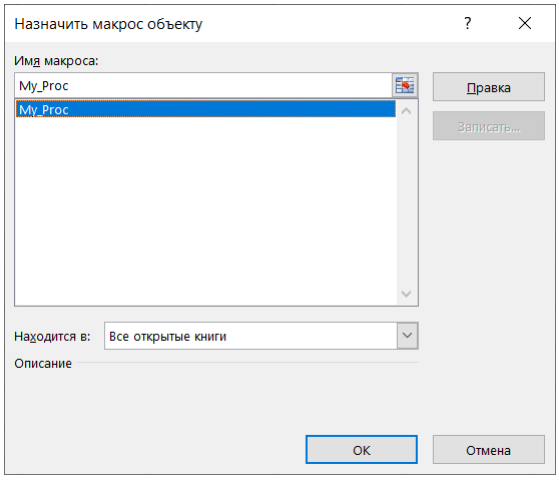
Из среды MS Excel запустить на выполнение макрос Разработчик > Макросы или нажать комбинацию клавиш ALT + F8 и выбрать соответствующую процедуру.

Второй подход состоит в том, чтобы поместить на рабочем листе MS Excel кнопку и назначить ей созданную процедуру.



Для этого из главного меню MS Excel нужно активизировать панель Формы, выбрать элемент Кнопка и поместить его на лист.





12. VBA В MS WORD

MS Word позволяет также использовать средства VBA. Это позволяет с успехом реализовывать автоматизацию создания документов.

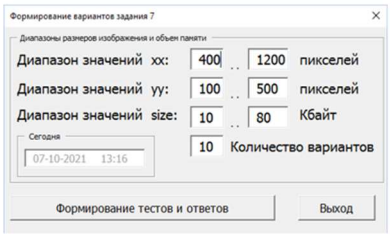
Так, например, формирование вариантов тестирования удобно создавать непосредственно в Word.

В качестве примера рассмотрим генерацию вариантов тестирования задания 7 ЕГЭ по информатике.

Этот документ используется для формирования тестов задания 7.

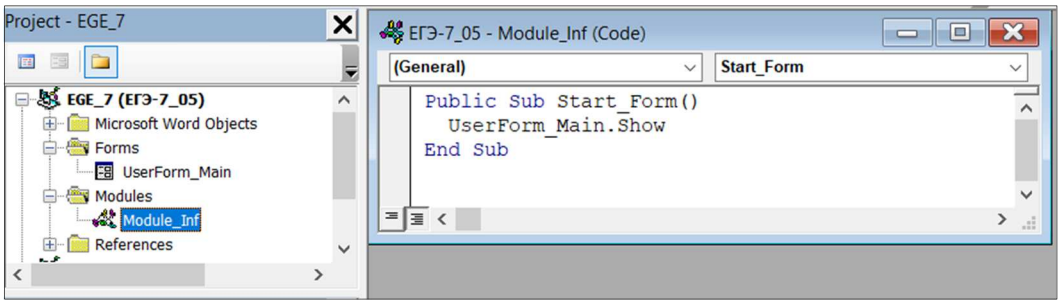
Для хранения произвольного растрового изображения размером **xx** × **yy** пикселей отведено **size** Кбайт памяти без учёта размера заголовка файла.
Для кодирования цвета каждого пикселя используется одинаковое количество бит, коды пикселей записываются в файл один за другим без промежутков.
Какое максимальное количество цветов можно использовать в изображении?

После того, как нажали кнопку Старт, появится форма, на которой можно задать данные.

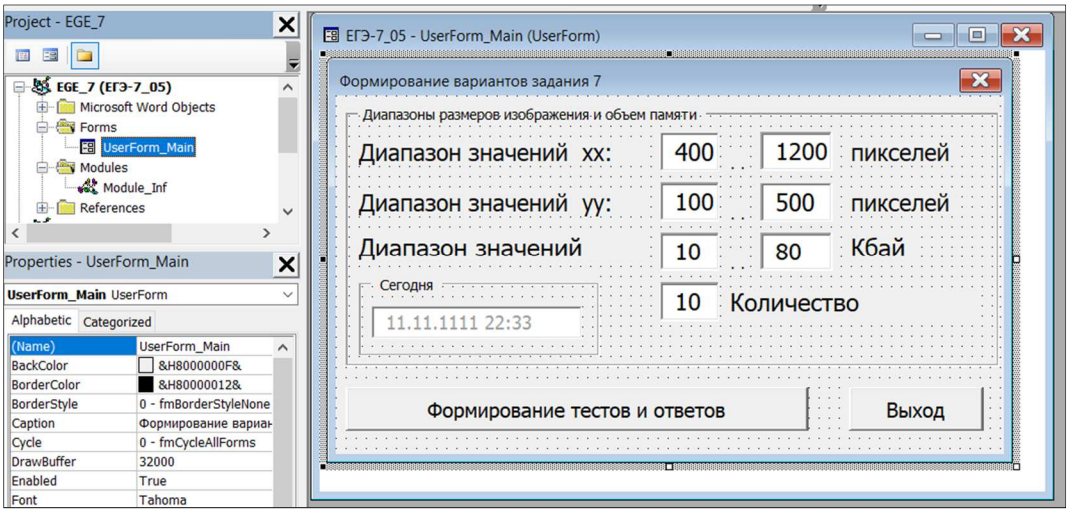


Полученные задания и ответы формируются в отдельных файлах.

При нажатии кнопки Старт происходит вызов из модуля Module_Inf процедуры Start_Form.



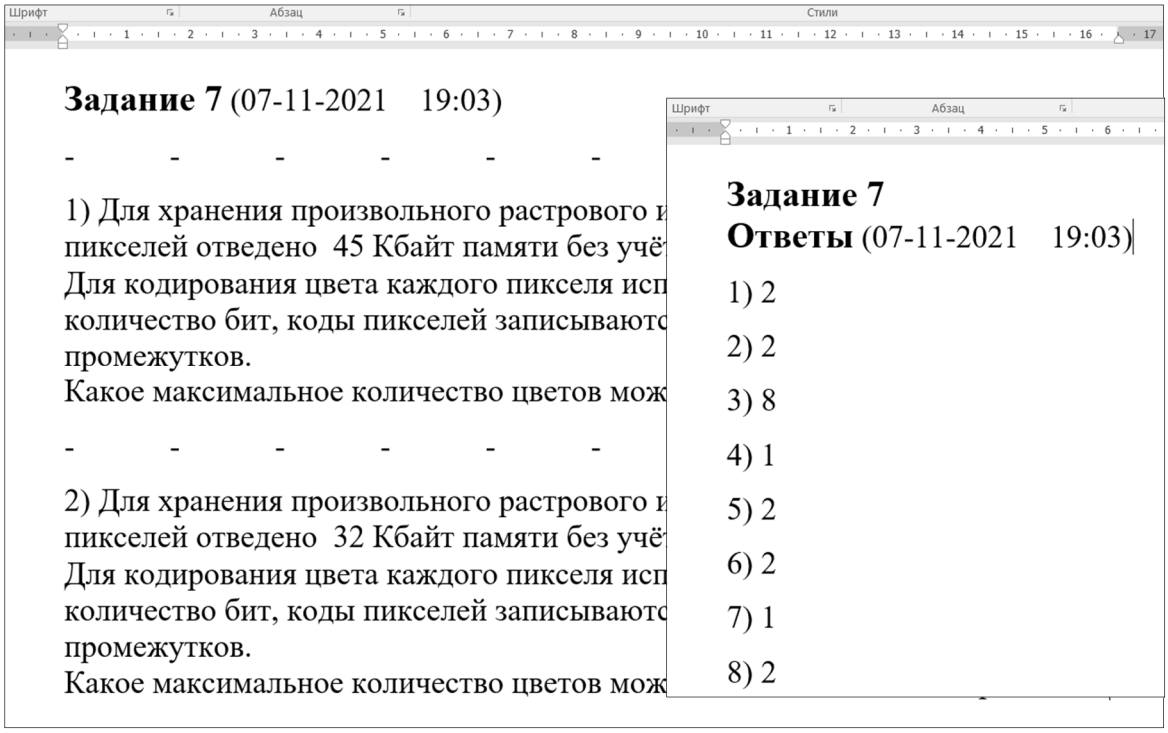
В редакторе VBA создаем форму



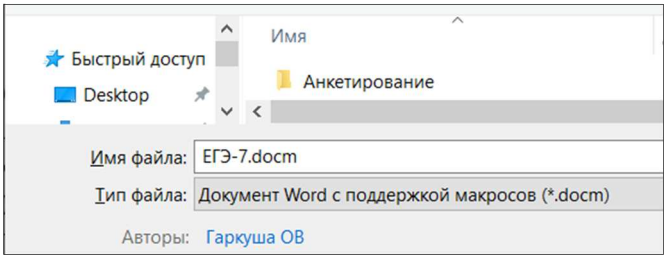
Формирование вариантов тестирования и ответов к ним происходит в процедуре **CB_Create_Test_Click.**

```
' Инициализация датчика случайных чисел
Randomize
' Случайные значения xx из [xx0..xx1]
xx = Int((xx1 - xx0) * Rnd + xx0)

' Случайные значения yy из [yy0..yy1]
yy = Int(Rnd * (yy1 - yy0)) + yy0

' Случайные значения size из [size0..size1]
size = Int(Rnd * (size1 - size0)) + size0

' Формируем ответ для полученных случайных значений
Count_Bit = Int(size * 1024 * 8 / (xx * yy))
Count_Color = 2 ^ Count_Bit

' Сохраняем ответ в массиве
Arr_Otvet(i) = Count_Color
```

Заданное количество тестов и ответов к ним формируются в файлах *.docx



После создания формы документ необходимо сохранить как [… с поддержкой макросов (*.docm)]



Полный текст процедур приведен в прил. 2.

РАЗДЕЛ 2. ЗАДАЧИ ЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ

1. ЗАДАЧА О РАСПРЕДЕЛЕНИИ РЕСУРСОВ

1.1. Общая постановка задачи

Необходимо определить план производства одного или нескольких видов продукции, который обеспечивает наиболее рациональное использование имеющихся материальных, финансовых и других видов ресурсов. Такой план будет оптимальным с точки зрения какого-либо выбранного критерия — максимума прибыли, минимума затрат на производство и т.д.

Задачи распределения ресурсов возникают на разных уровнях в системе экономического управления: на уровне отдельных производственных участков и бригад, предприятий, отраслей, на уровне народного хозяйства в целом:

$$\sum_j c_j x_j \to \max, \tag{1}$$

$$\sum_j a_{ij} x_j \le b_i, \quad i = 1,...,m \tag{2}$$

$$x_j \ge 0, \qquad j = 1,..., n \tag{3}$$

Обозначения:

n — количество видов выпускаемой продукции;

m — количество видов производственных ресурсов (производственные мощности, сырье, рабочая сила);

a_{ij} — объем i-го ресурса на выпуск единицы j-й продукции;

c_j — прибыль от выпуска единицы j-й продукции;

b_i — количество имеющегося ресурса i-го вида;

x_j — объем выпуска j-й продукции (переменная);

(1) — целевая функция (максимум прибыли);

(2) — группа ограничений на объем имеющихся в наличии ресурсов;

(3) — ограничения на неотрицательность переменных.

Если финансы, оборудование, сырье и даже людей полагать ресурсами, то значительное число задач в экономике можно рассматривать как задачи распределения ресурсов. Достаточно

часто математической моделью таких задач является задача линейного программирования.

Рассмотрим следующий пример.

Требуется определить, в каком количестве надо выпускать продукцию четырех типов Прод1, Прод2, Прод3, Прод4, для изготовления которой требуются ресурсы трех видов: трудовые, сырье, финансы. Количество ресурса каждого вида, необходимое для выпуска единицы продукции данного типа, называется нормой расхода. Нормы расхода, а также прибыль, получаемая от реализации единицы каждого типа продукции, приведены на рис. 2.1. Там же приведено наличие располагаемого ресурса.

	A	B	C	D	E	F	G
1	Ресурс	Прод1	Прод2	Прод3	Прод4	Знак	Наличие
2	Прибыль	60	70	120	130	max	-
3	Трудовые	1	1	1	1	<=	16
4	Сырье	6	5	4	3	<=	110
5	Финансы	4	6	10	13	<=	100

Рис. 2.1

Составим математическую модель, для чего введем следующие обозначения:

x_j — количество выпускаемой продукции j-го типа, $j = 1,\ldots,4$;

b_j — количество располагаемого ресурса i-го вида, $i = 1,\ldots,3$;

a_{ij} — норма расхода i-го ресурса для выпуска единицы продукции j-го типа;

c_j — прибыль, получаемая от реализации единицы продукции j-го типа.

Как видно из рис. 2.1, для выпуска единицы Прод1 требуется 6 единиц сырья, значит, для выпуска всей продукции Прод1 требуется $6x_1$ единиц сырья, где x_1 — количество выпускаемой продукции Прод1. С учетом того что для других видов продукции зависимости аналогичны, ограничение по сырью будет иметь вид

$$6x_1 + 5x_2 + 4x_3 + 3x_4 <= 110.$$

В этом ограничении левая часть равна величине *потребного* ресурса, а правая показывает количество *имеющегося* ресурса.

	A	B	C	D	E	F	G	H
1		Переменные						
2	Ресурс	Прод1	Прод2	Прод3	Прод4			
3	Значения							
4	Нижн. гр.							
5	Верхн. гр.							
6	Коэфф. ЦФ	60	70	120	130	0	Макс	
7		Ограничения						
8	Вид					Левая часть	Знак	Правая часть
9	Трудовые	1	1	1	1	0	<=	16
10	Сырье	6	5	4	3	0	<=	110
11	Финансы	4	6	10	13	0	<=	100

Рис. 2.2

Аналогично можно составить ограничения для остальных ресурсов и написать зависимость для целевой функции. Тогда математическая модель задачи будет иметь вид

$$\left.\begin{aligned} & F = 60x_1 + 70x_2 + 120x_3 + 130x_4 \to max \\ & x_1 + x_2 + x_3 + x_4 \le 16 \\ & 6x_1 + 5x_2 + 4x_3 + 3x_4 \le 110 \\ & 4x_1 + 6x_2 + 10x_3 + 13x_4 \le 100 \\ & x_j \ge 0, \qquad j = 1,...,4 \end{aligned}\right\} \qquad (4)$$

1.2. Ввод условий задачи о распределении ресурсов

Ввод условий задачи состоит из следующих основных шагов:

1. Создание формы для ввода условий задачи.
2. Ввод исходных данных.
3. Ввод зависимостей из математической модели.
4. Назначение целевой функции.
5. Ввод ограничений и граничных условий.

1. Для задачи, приведенной на рис. 2.1, требуется создать форму для ввода условий задачи (рис. 2.2).

Весь текст на рис. 2.2 (и в дальнейшем) является комментариями и на решение задачи не влияет.

2. Ввести исходные данные в форму (рис. 2.3). Необходимые исходные данные приведены на рис. 2.1. и 2.3.

3. Ввести зависимости из математической модели (2.4).

	A	B	C	D	E	F	G	H
1		Переменные						
2	Ресурс	Прод1	Прод2	Прод3	Прод4			
3	Значения							
4	Нижн. гр.							
5	Верхн. гр.							
6	Коэфф. ЦФ	60	70	120	130	0	Макс	
7		Ограничения						
8	Вид					Левая часть	Знак	Правая часть
9	Трудовые	1	1	1	1	0	<=	16
10	Сырье	6	5	4	3	0	<=	110
11	Финансы	4	6	10	13	0	<=	100

Рис. 2.3

Для наглядности (но не обязательно!) можно перейти к режиму представления формул. При этом ввод данных приводится на рис. 2.3, а режим представления формул (Формулы > Показать формулы или CTRL+`) — на рис. 2.4.

	A	B	C	D	E	F	G	H
1		Переменные						
2	Ресурс	Прод1	Прод2	Прод3	Прод4			
3	Значения	10	0	6	0			
4	Нижн. гр.	0	0	0	0			
5	Верхн. гр.							
6	Коэфф. ЦФ	60	70	120	130	=СУММПРОИЗВ(B3:E3;B6:E6)	Макс	
7		Ограничения						
8	Вид					Левая часть	Знак	Правая часть
9	Трудовые	1	1	1	1	=СУММПРОИЗВ(B3:E3;B9:E9)	<=	16
10	Сырье	6	5	4	3	=СУММПРОИЗВ(B3:E3;B10:E10)	<=	110
11	Финансы	4	6	10	13	=СУММПРОИЗВ(B3:E3;B11:E11)	<=	100

Рис. 2.4

4. Ввести зависимости для целевой функции:

> Курсор в ячейку F6;
> Вызываем «Вставка функции»
> Категория – «Математические»
> Функции – «СУММПРОИЗВ»

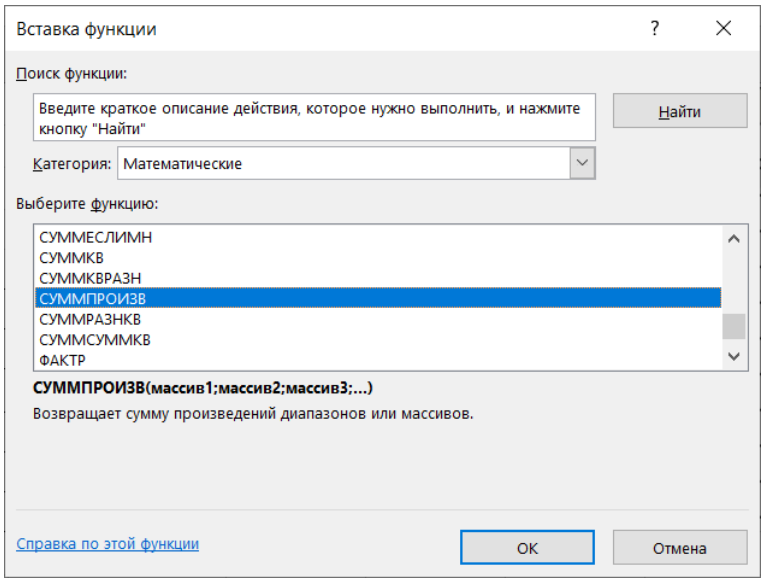


Рис. 2.5. Диалоговое окно «Вставка функции» шаг 1 из 2

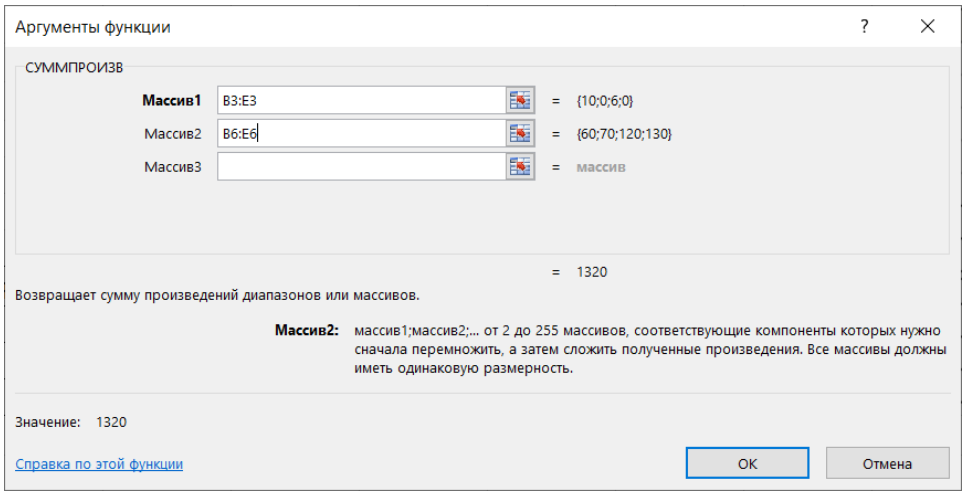


Рис. 2.6 Диалоговое окно «Вставка функции» шаг 2 из 2

> В массив 1 ввести B$3:E$3.

Заметим, что во все диалоговые окна адреса ячеек удобно вводить не с клавиатуры, а протаскивая мышь по ячейкам, адреса которых следует ввести.

> В массив 2 ввести B6:E6.

Ввести зависимости для левых частей ограничений:

> Курсор в F6.
> Копировать.
> Курсор в F9.
> Вставить.

На экране: в F9 введена функция, как это показано на рис. 2.4.

> Скопировать F9 в F10:F11.

Диалоговое окно «Поиск решения»

1. Данные > Поиск решения...

В том случае, когда в главном меню «Данные» нет пункта «Поиск решения», следует указать этот инструмент в пункте меню «Надстройки…»: Файл > Параметры > Надстройки.

Выбрать «Поиск решения» (рис. 2.7)

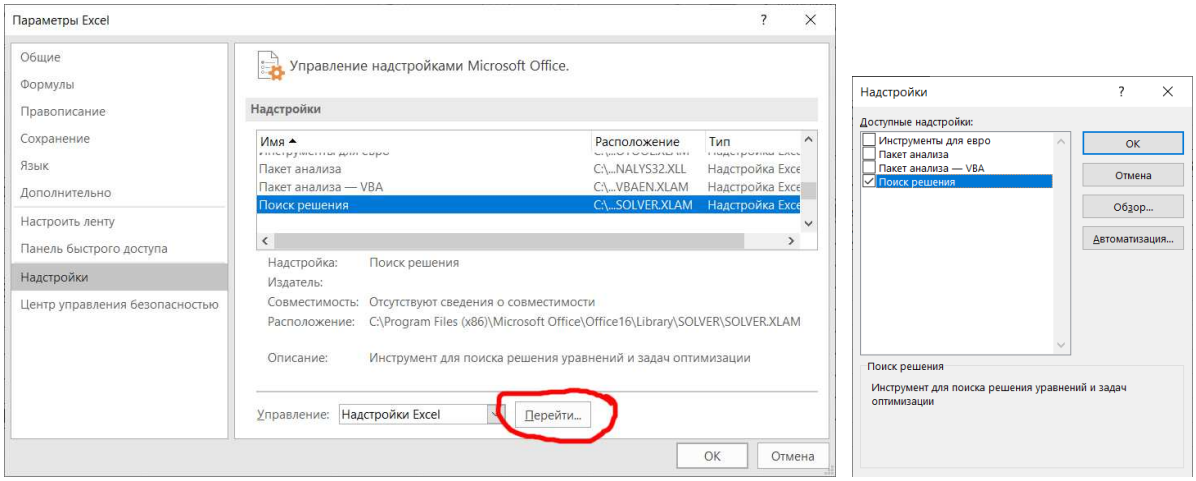
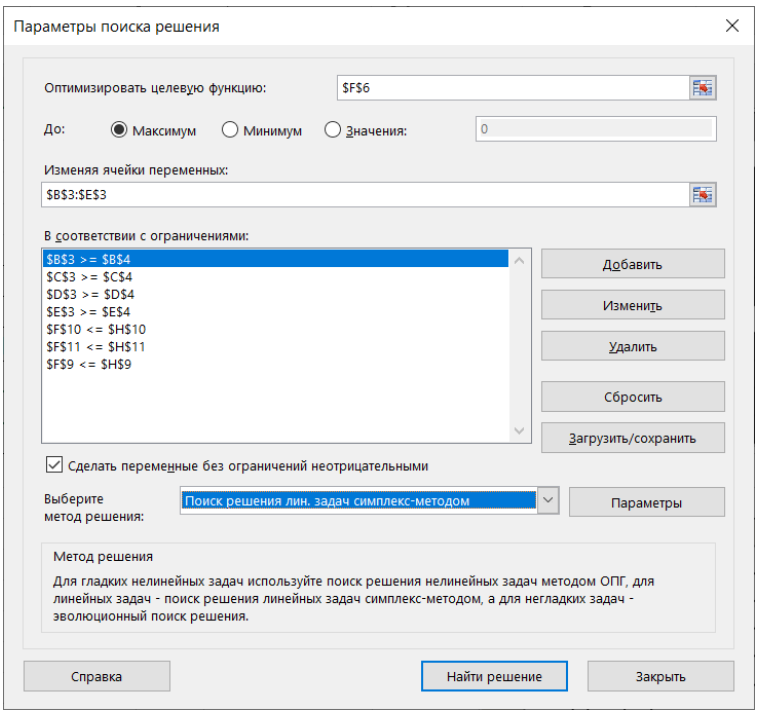


Рис. 2.7. Диалоговое окно «Поиск решения».

2. Назначить целевую функцию:

> Курсор в окно «Оптимизировать целевую функцию»: $F$6.

Направление целевой функции: «Максимум»



3. Ввести адреса искомых переменных:

> «Изменяя ячейки переменных»: $B$3: $E$3. «Добавить»

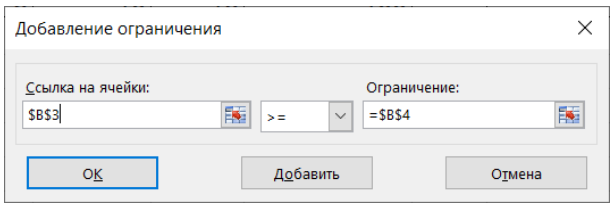


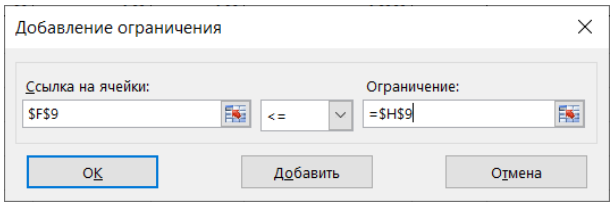
Рис. 2.8. Диалоговое окно «Добавление ограничения»

4. Ввести граничные условия на переменные (Прод1 — Прод4): B3 >= B4, C3 >= C4, D3 >= D4, E3 >= E4.

Также вводим граничные условия для остальных переменных.

5. Аналогично ввести ограничения:

F9 <= H9, F10 <= H10, F11 <= H11.



После ввода последнего ограничения вместо «Добавить» ввести «ОК».

Если при вводе задачи возникает необходимость в изменении или удалении внесенных ограничений или граничных условий, то это делается с помощью команд «Изменить», «Удалить».

Решение задачи производится сразу же после ввода данных, когда на экране находится диалоговое окно «Поиск решения» (рис. 2.7). Предварительно можно задать параметры поиска решения.

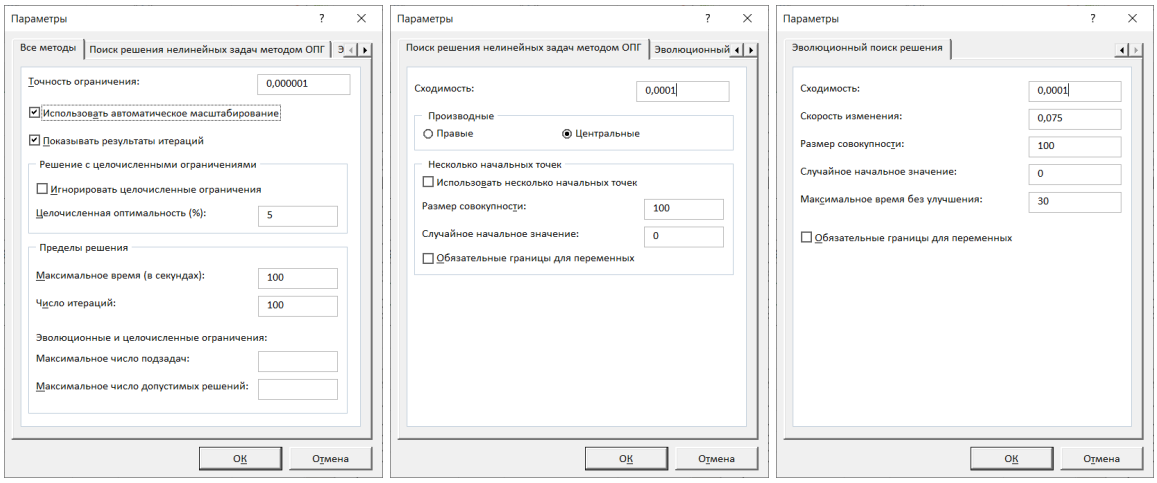


Рис. 2.9. Диалоговое окно «Параметры поиска решения»

С помощью команд, находящихся в этом диалоговом окне, можно вводить условия для решения задач оптимизации всех классов. Приведем наиболее важные параметры, которые применяются при решении конкретных задач. Вместе с тем команды, используемые по умолчанию, подходят для решения большей части практических задач.

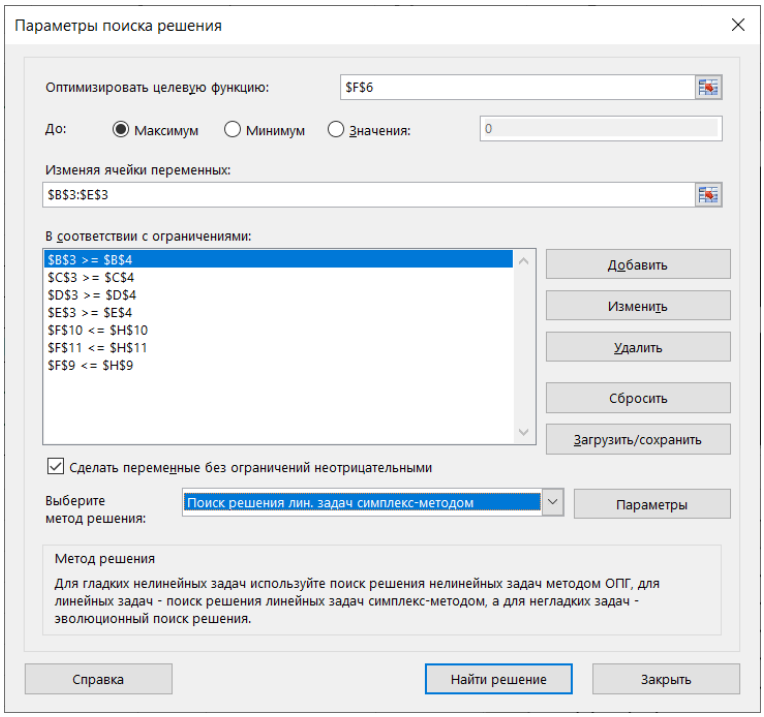
Максимальное время

Служит для назначения времени в секундах, выделяемого на поиск решения задачи. В поле можно ввести время, не превышающее 32767 секунд (более 9 часов!). Значение 100, используемое по умолчанию, подходит для решения большинства задач.

Предельное число итераций

Служит для назначения числа итераций. Используемое по умолчанию значение 100 подходит для решения большинства задач.

В диалоговом окне «Параметры поиска решения» выбираем «Найти решение»



Решение найдено (рис. 2.10) и результат оптимального решения задачи приведен на рис. 2.11.

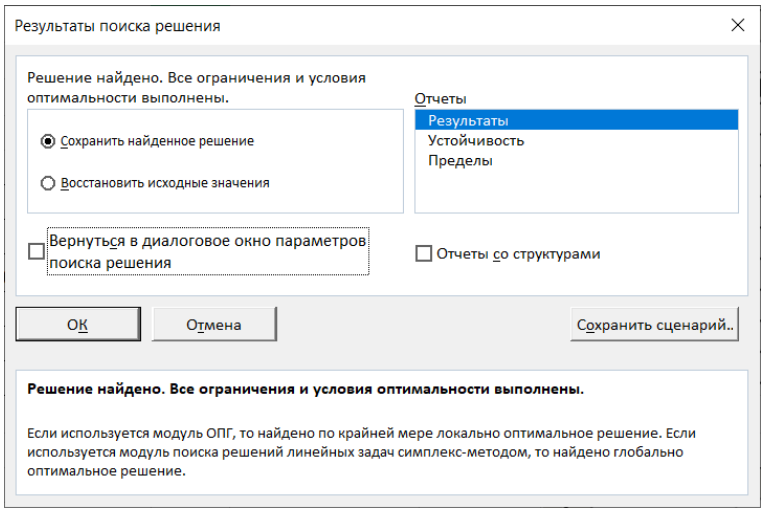


Рис. 2.10. Диалоговое окно «Результаты поиска решения»

На листе с исходными данными можно видеть полученное решение.

	A	B	C	D	E	F	G	H
1		Переменные						
2	Ресурс	Прод1	Прод2	Прод3	Прод4			
3	Значения	10	0	6	0			
4	Нижн. гр.	0	0	0	0			
5	Верхн. гр.							
6	Коэфф. ЦФ	60	70	120	130	1320	Макс	
7		Ограничения						
8	Вид					Левая часть	Знак	Правая часть
9	Трудовые	1	1	1	1	16	<=	16
10	Сырье	6	5	4	3	84	<=	110
11	Финансы	4	6	10	13	100	<=	100

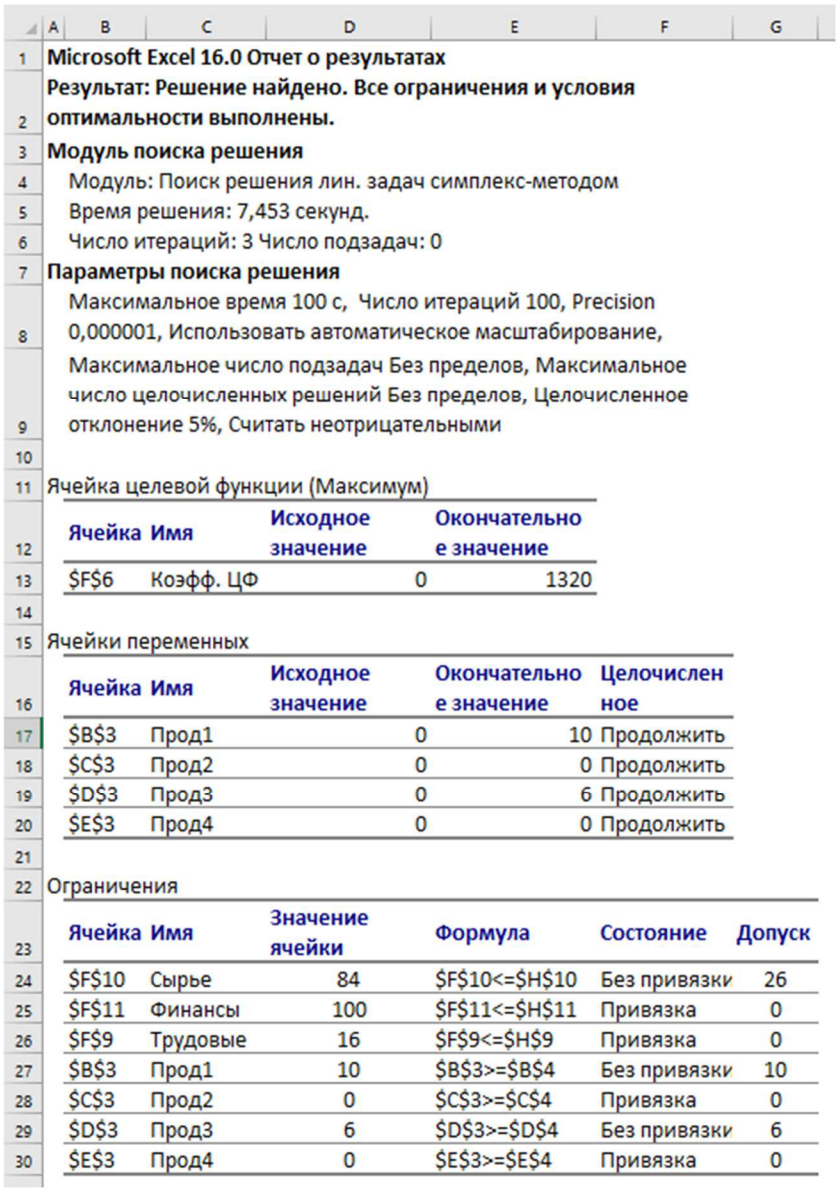
Рис. 2.11. Оптимальное решение

На рис. 2.11 видно, что в оптимальном решении
Прод1 = B3 = 10,
Прод2 = C3 = 0,
Прод3 = D3 = 6,
Прод4 = E3 = 0.

При этом максимальная прибыль будет составлять F6 = 1320, а количество использованных ресурсов равно
Трудовые = F9 = 16,
Сырье = F10 = 84,
Финансы = F11 = 100.

Таково оптимальное решение рассматриваемой задачи распределения ресурсов.

Выбирая «Результаты» во вкладке «Отчеты» на листе «Отчет о результатах», получим



Microsoft Excel 16.0 Отчет о результатах

Результат: Решение найдено. Все ограничения и условия оптимальности выполнены.

Модуль поиска решения

Модуль: Поиск решения лин. задач симплекс-методом

Время решения: 7,453 секунд.

Число итераций: 3 Число подзадач: 0

Параметры поиска решения

Максимальное время 100 с, Число итераций 100, Precision 0,000001, Использовать автоматическое масштабирование, Максимальное число подзадач Без пределов, Максимальное число целочисленных решений Без пределов, Целочисленное отклонение 5%, Считать неотрицательными

Ячейка целевой функции (Максимум)

Ячейка	Имя	Исходное значение	Окончательное значение
$F$6	Коэфф. ЦФ	0	1320

Ячейки переменных

Ячейка	Имя	Исходное значение	Окончательное значение	Целочисленное
$B$3	Прод1	0	10	Продолжить
$C$3	Прод2	0	0	Продолжить
$D$3	Прод3	0	6	Продолжить
$E$3	Прод4	0	0	Продолжить

Ограничения

Ячейка	Имя	Значение ячейки	Формула	Состояние	Допуск
$F$10	Сырье	84	$F$10<=$H$10	Без привязки	26
$F$11	Финансы	100	$F$11<=$H$11	Привязка	0
$F$9	Трудовые	16	$F$9<=$H$9	Привязка	0
$B$3	Прод1	10	$B$3>=$B$4	Без привязки	10
$C$3	Прод2	0	$C$3>=$C$4	Привязка	0
$D$3	Прод3	6	$D$3>=$D$4	Без привязки	6
$E$3	Прод4	0	$E$3>=$E$4	Привязка	0

2. ЗАДАЧА О РАЦИОНЕ

2.1. Общая постановка задачи

Прародительницей этих задач была так называемая задача о диете: найти наиболее дешевую смесь пищевых продуктов 1,2,…,*m* (хлеба, мяса, молока и пр.), которая удовлетворяла бы определенным биологическим ограничениям на содержание жиров, белков, углеводов, микроэлементов, витаминов и тому подобных биологически активных веществ.

Если обозначить через x_j содержание по весу (возможно процентное) j-го продукта в смеси, через a_{ij} — весовое содержание i-го вещества в j-м продукте, d_i — допустимую верхнюю границу содержания i-го вещества в смеси, b_j — нижнюю, а через c_j — стоимость j-го продукта, то задача о наиболее дешевой диете приобретает вид

$$\sum_{j=1}^{n} c_j x_j \to \min \tag{5}$$

$$b_i \le \sum_{j=1}^{n} a_{ij} x_j \le d_i, \quad i = 1,...m \tag{6}$$

$$x_j \ge 0, \qquad j = 1,...,n \tag{7}$$

Для этой задачи характерно наличие двусторонних ограничений (6) на значение определенных линейных комбинаций переменных. Эта особенность весьма часто встречается в практике линейного программирования и специальным образом учитывается как в алгоритмах, так и в формах представления данных.

В приведенной интерпретации задачи в качестве продуктов могут выступать, например, различные виды нефти, полученные с разных месторождений. Эти виды отличаются по составу: они содержат различные концентрации примесей серы, парафинов, воды и прочих веществ, существенно влияющих на процесс термического разложения нефти на бензины, керосин и другие нефтепродукты.

Для наилучшей эффективности и безопасности технологического процесса концентрации упомянутых примесей должны находиться в определенных пределах, что достигается смешиванием различных видов сырой нефти. Учитывая то, что стоимости различных видов нефти существенно отличаются, задача подбора наиболее дешевой смеси, укладывающейся в технологические допуски, может дать существенный экономический эффект, преумноженный многомиллионными объемами переработки.

Аналогичные проблемы возникают, например, и при производстве металлургического кокса из углей различных

месторождений, разработке рациона питания скота и пр. В более реалистичных постановках возникают также и так называемые производственно-транспортные задачи, когда в расходах учитывают и транспортные затраты.

За неизвестные в моделях о рационе принимаются доли или количества ингредиентов, идущие на приготовление рациона.

Различаются типы моделей:

а) с одним или с большим числом рационов;

б) с ограниченным или неограниченным количеством ингредиентов;

в) по критерию производства рациона.

Рассмотрим пример поиска оптимального рациона для однопродуктовой модели.

Определить рацион, стоимость которого была бы минимальной, если предельные нормы суточной выдачи сена — не более 18 кг, силоса — не более 24 кг, концентратов — не более 16 кг. Данные о содержании белка, кальция и витаминов приведены на рис. 2.12.

	Белок, г/кг	Кальций, г/кг	Витамины, уд.ед/кг	Цена 1кг, р.
Сено	40	5	2	300
Силос	20	4	1	200
Концентраты	160	4	2	500
Нормы потребления	2000	120	40	

Рис. 2.12

Построим математическую модель этой задачи.

Как видно из рис. 2.12, в 1 кг сена содержится 40 г белка, значит, для белкового рациона требуется $40x_1$ единиц сена, где x_1 — количество сена, входящего в суточный рацион. С учетом того, что для силоса и концентратов зависимости аналогичны, ограничение по белковому рациону будет иметь вид

$$40x_1 + 20x_2 + 160x_3 \geq 2000.$$

В этом ограничении левая часть равна величине *потребного* количества белкового рациона, а правая показывает *необходимое* количество белкового рациона.

Аналогично можно составить ограничения для остальных рационов и написать зависимость для целевой функции.

Тогда математическая модель задачи будет иметь вид:

$$\begin{cases} F = 300x_1 + 200x_2 + 500x_3 \to min \\ 40x_1 + 20x_2 + 160x_3 \geq 2000 \\ 5x_1 + 4x_2 + 4x_3 \geq 120 \\ 2x_1 + x_2 + 2x_3 \geq 40 \\ x_1 \leq 18; \quad x_2 \leq 24; \quad x_3 \leq 16 \\ x_j \geq 0, \qquad j = 1,...,3 \end{cases}$$

2.2. Ввод условий задачи о рационе

Для задачи, приведенной на рис. 2.12, сделать форму для ввода условий задачи (рис. 2.13).

	A	B	C	D	E	F	G
1		Переменные					
2	Имя	Сено	Силос	Концентраты			
3	Значения						
4	Нижн. гр.	0	0	0			
5	Верхн. гр.	18	24	16			
6	Коэфф. ЦФ	300	200	500	0	Мин	
7		Ограничения					
8	Вид				Левая часть	Знак	Правая часть
9	Белок	40	20	160	0	>=	2000
10	Кальций	5	4	4	0	>=	120
11	Витамины	2	1	2	0	>=	40

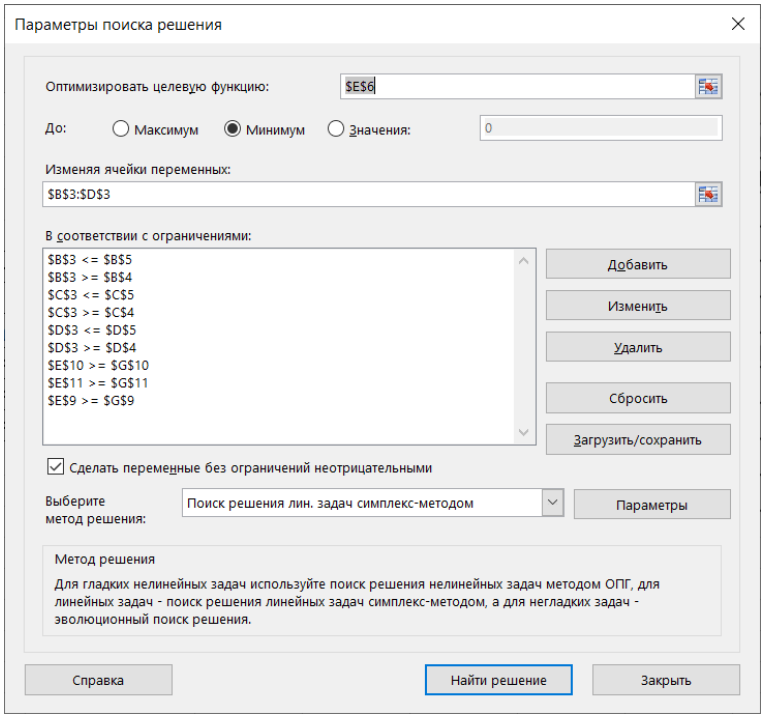
Рис. 2.13

Для наглядности можно перейти к режиму представления формул — на рис. 2.14.

	A	B	C	D	E	F	G
1		Переменные					
2	Имя	Сено	Силос	Концентраты			
3	Значения						
4	Нижн. гр.	0	0	0			
5	Верхн. гр.	18	24	16			
6	Коэфф. ЦФ	300	200	500	=СУММПРОИЗВ(B3:D3;B6:D6)	Мин	
7		Ограничения					
8	Вид				Левая часть	Знак	Правая часть
9	Белок	40	20	160	=СУММПРОИЗВ(B3:D3;B9:D9)	>=	2000
10	Кальций	5	4	4	=СУММПРОИЗВ(B3:D3;B10:D10)	>=	120
11	Витамины	2	1	2	=СУММПРОИЗВ(B3:D3;B11:D11)	>=	40

Рис. 2.14

По аналогии с вводом данных и ограничений для предыдущей задачи (Данные > Поиск решения...)



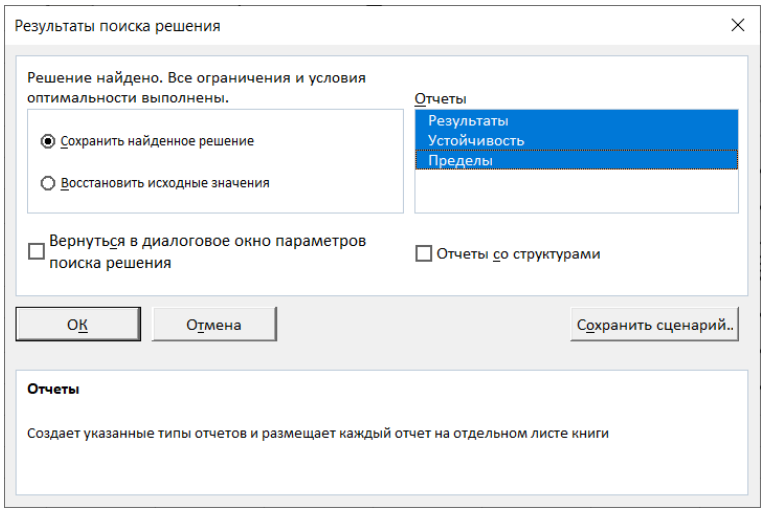
можно получить оптимальный рацион:

	A	B	C	D	E	F	G
1		Переменные					
2	Имя	Сено	Силос	Концентраты			
3	Значения	0	20	10			
4	Нижн. гр.	0	0	0			
5	Верхн. гр.	18	24	16			
6	Коэфф. ЦФ	300	200	500	9000	Мин	
7		Ограничения					
8	Вид				Левая часть	Знак	Правая часть
9	Белок	40	20	160	2000	>=	2000
10	Кальций	5	4	4	120	>=	120
11	Витамины	2	1	2	40	>=	40

Рис. 2.15. Оптимальный рацион

2.3. Анализ оптимального решения

Проведем анализ чувствительности задачи об оптимальном рационе. Для этого необходимо в окне «Результаты поиска решения» выделить типы отчетов – «Результаты», «Устойчивость», «Пределы». При этом в рабочей книге будут созданы новые листы с соответствующими отчетами.



2.3.1. Отчет по результатам

Отчет по результатам (рис. 2.16) состоит из трех таблиц:

1) таблица 1 содержит информацию о ЦФ;

2) таблица 2 содержит информацию о значениях переменных, полученных в результате решения задачи;

3) таблица 3 показывает результаты оптимального решения для ограничений и для граничных условий.

Microsoft Excel 16.0 Отчет о результатах

Результат: Решение найдено. Все ограничения и условия оптимальности выполнены.

Модуль поиска решения

Модуль: Поиск решения лин. задач симплекс-методом

Время решения: 0,016 секунд.

Число итераций: 6 Число подзадач: 0

Параметры поиска решения

Максимальное время 100 с, Число итераций 100, Precision 0,000001, Использовать автоматическое масштабирование

Максимальное число подзадач Без пределов, Максимальное число целочисленных решений Без пределов, Целочисленное отклонение 5%,

Ячейка целевой функции (Минимум)

Ячейка	Имя	Исходное значение	Окончательное значение
$E$6	Коэфф. ЦФ	0	9000

Ячейки переменных

Ячейка	Имя	Исходное значение	Окончательное значение	Целочисленное
$B$3	Сено	0	0	Продолжить
$C$3	Силос	0	20	Продолжить
$D$3	Концентраты	0	10	Продолжить

Ограничения

Ячейка	Имя	Значение ячейки	Формула	Состояние	Допуск
$E$10	Кальций Левая часть	120	$E$10>=$G$10	Привязка	0
$E$11	Витамины Левая часть	40	$E$11>=$G$11	Привязка	0
$E$9	Белок Левая часть	2000	$E$9>=$G$9	Привязка	0
$B$3	Значения Сено	0	$B$3<=$B$5	Без привязки	18
$B$3	Значения Сено	0	$B$3>=$B$4	Привязка	0
$C$3	Значения Силос	20	$C$3<=$C$5	Без привязки	4
$C$3	Значения Силос	20	$C$3>=$C$4	Без привязки	20
$D$3	Значения Концентраты	10	$D$3<=$D$5	Без привязки	6
$D$3	Значения Концентраты	10	$D$3>=$D$4	Без привязки	10

Рис. 2.16. Лист отчета по результатам

Столбец «Состояние» значением «Привязка» отмечает ограничения (≥ или ≤), которые в результате решения превратились в строгие равенства, прочие ограничения имеют статус «Без привязки».

Столбец «Допуск» показывает, на какую величину ограничения отклонились от строгого равенства, т.е. показана разность между значением переменной в найденном оптимальном решении и заданным для нее граничным условием.

Таблица 3 отчета по результатам дает информацию для анализа возможного изменения запасов *несвязанных* ресурсов при сохранении полученного оптимального значения ЦФ. Так, если на ресурс наложено ограничение типа ≥, то в графе «Допуск» дается количество ресурса, на которое была превышена минимально необходимая норма.

Если на ресурс наложено ограничение типа ≤, то в графе «Допуск» дается количество ресурса, которое не используется при реализации оптимального решения.

Анализ строки 28 показывает, что 4 кг силоса оказывается неиспользуемым, а из анализа строки 30 следует, что неиспользованных концентратов — 6 кг. Из этого следует, что количество силоса можно уменьшить на 4 кг, а концентратов — на 6 кг, и это никак **не повлияет** на оптимальное решение.

2.3.2. Отчет по устойчивости

Отчет по устойчивости (рис. 2.17) состоит из двух таблиц.

Таблица 1 содержит информацию, относящуюся к ячейкам переменных.

1. Окончательное значение.

2. Приведенная стоимость, которая показывает, на сколько может измениться значение ЦФ в случае принудительного включения единицы этой продукции в оптимальное решение.

3. Коэффициенты целевой функции.

4. Предельные значения приращения целевых коэффициентов Δc_j, при которых сохраняется первоначальное оптимальное решение.

	A	B	C	D	E	F	G	H
1	**Microsoft Excel 16.0 Отчет об устойчивости**							
2								
3	Ячейки переменных							
4				Окончательное	Приведенн.	Целевая функция	Допустимое	Допустимое
5		Ячейка	Имя	Значение	Стоимость	Коэффициент	Увеличение	Уменьшение
6		$B$3	Значения Сено	0	0	300	100	17,85714286
7		$C$3	Значения Силос	20	0	200	15,625	50
8		$D$3	Значения Концентраты	10	0	500	166,6666667	233,3333333
9								
10	Ограничения							
11				Окончательное	Тень	Ограничение	Допустимое	Допустимое
12		Ячейка	Имя	Значение	Цена	Правая сторона	Увеличение	Уменьшение
13		$E$10	Кальций Левая часть	120	33,33333333	120	0	30
14		$E$11	Витамины Левая часть	40	27,77777778	40	11,25	0
15		$E$9	Белок Левая часть	2000	1,944444444	2000	0	1200

Рис. 2.17. Лист отчета по устойчивости

Таблица 2 (рис. 2.17) содержит информацию, относящуюся к ограничениям.

1. Величина использованных ресурсов в колонке «Окончательное значение».

2. Теневая цена (ценность дополнительной единицы i-го ресурса) рассчитывается только для *связанных* ресурсов.

3. Предельные значения приращения ресурсов Δc_j. В графе «Допустимое Уменьшение» показывают, на сколько можно уменьшить (устранить излишек) или увеличить (повысить минимально необходимое требование) ресурс, сохранив при этом оптимальное решение.

2.3.3. Отчет по пределам

Отчет по результатам (рис. 2.18) состоит из двух таблиц:

1) таблица 1 в очередной раз содержит информацию о ЦФ;

2) таблица 2 содержит информацию о диапазонах значений переменных и соответствующих значениях целевой функции.

	A	B	C	D	E	F	G	H	I	J
1	Microsoft Excel 16.0 Отчет о пределах									
2		Целевая функция								
3		Ячейка	Имя	Значение						
4		$E$6	Коэфф. ЦФ	9000						
5										
6		Переменная				Нижний	Целевая функция		Верхний	Целевая функция
7		Ячейка	Имя	Значение		Предел	Результат		Предел	Результат
8		$B$3	Сено	0		0	9000		18	14400
9		$C$3	Силос	20		20	9000		24	9800
10		$D$3	Концентраты	10		10	9000		16	12000

Рис. 2.18. Лист отчета по пределам

3. ТРАНСПОРТНАЯ ЗАДАЧА

3.1. Общая постановка задачи

В транспортной задаче (ТЗ) требуется найти оптимальный план перевозок некоторого продукта от заданного множества производителей, пронумерованных числами 1, …, M, к множеству потребителей, также занумерованных числами 1, …, N.

Производственные возможности i-го производителя заданы объемом производимого продукта — a_i. Спрос j-го потребителя на этот продукт задается числом b_j.

Обозначим планируемый объем перевозок от i-го производителя к j-му потребителю как x_{ij}. В этих условиях должны быть выполнены балансовые соотношения:

$$\begin{aligned} \sum_{j=1}^{N} x_{ij} &= a_i, \quad i = 1,2,...,M. \\ \sum_{i=1}^{M} x_{ij} &= b_j, \quad j = 1,2,...,N. \end{aligned} \tag{8}$$

Для существования допустимого плана перевозок должен выполняться общий баланс между спросом и потреблением:

$$\sum_{i=1}^{M} a_i = \sum_{j=1}^{N} b_j = \mathrm{D}.$$

При этом транспортную задачу называют сбалансированной.

Можно убедиться, например, что в сбалансированной транспортной задаче

$$x_{ij} = a_i b_j / \mathrm{D} \tag{9}$$

являются допустимым вариантом перевозок, т.е. удовлетворяющим ограничениям (8).

Целью решения транспортной задачи является минимизация суммарных транспортных расходов. Если предположить, что стоимость перевозки продукта линейно зависит от объема перевозки и характеризуется числами c_{ij}, где c_{ij} — стоимость перевозки единицы продукта от i-го производителя к j-му потребителю, а x_{ij} — объемы перевозок, то целевая функция в транспортной задаче принимает вид

$$T(x) = \sum_{i=1}^{M} \sum_{j=1}^{N} c_{ij} x_{ij} \qquad (10)$$

и задача заключается в минимизации (10) при выполнении ограничений (8) и условия неотрицательности переменных x_{ij}.

Переменные x_{ij} можно представить в виде матрицы объемов перевозок.

Поставщики \ Потребители	1	2	…	N
1	x_{11}	x_{12}	…	x_{1N}
2	x_{21}	x_{22}	…	x_{2N}
…			…	…
M	x_{M1}	x_{M1}	…	x_{MN}

Рассмотрим решение транспортной задачи, суть которой заключается в оптимальной организации транспортных перевозок штучного товара со складов в магазины

Тарифы, р./шт.	1-й магазин	2-й магазин	3-й магазин	Запасы, шт.
1-й склад	2	9	7	25
2-й склад	1	0	5	50
3-й склад	5	4	100	35
4-й склад	2	3	6	75
Потребности	45	90	50	45+90+50=25+50+35+75, т.е. сбалансированная задача.

Рис. 2.19. Исходные данные транспортной задачи

Целевая функция и ограничения данной задачи примут вид

$$
\begin{aligned}
T(x)= \quad & 2\cdot x_{11}+ \quad 9\cdot x_{12}+ \quad 7\cdot x_{13}+ \\
& 1\cdot x_{21}+ \quad\quad\quad 5\cdot x_{23}+ \\
& 5\cdot x_{31}+ \quad 4\cdot x_{32}+ \quad 100\cdot x_{33}+ \\
& 2\cdot x_{41}+ \quad 3\cdot x_{42}+ \quad 6\cdot x_{43} \quad\quad \rightarrow min
\end{aligned}
$$

$$
\begin{cases}
x_{11}+x_{12}+x_{13}=25 \\
x_{21}+x_{22}+x_{23}=50 \\
x_{31}+x_{32}+x_{33}=35 \\
x_{41}+x_{42}+x_{43}=75 \\
x_{11}+x_{21}+x_{31}=45 \\
x_{12}+x_{22}+x_{32}=90 \\
x_{13}+x_{23}+x_{33}=50 \\
\forall x_{ij}\geq 0, \forall x_{ij}-целые\ (i=\overline{1,4};\ j=\overline{1,3})
\end{cases}
\qquad (11)
$$

3.2. Решение транспортной задачи

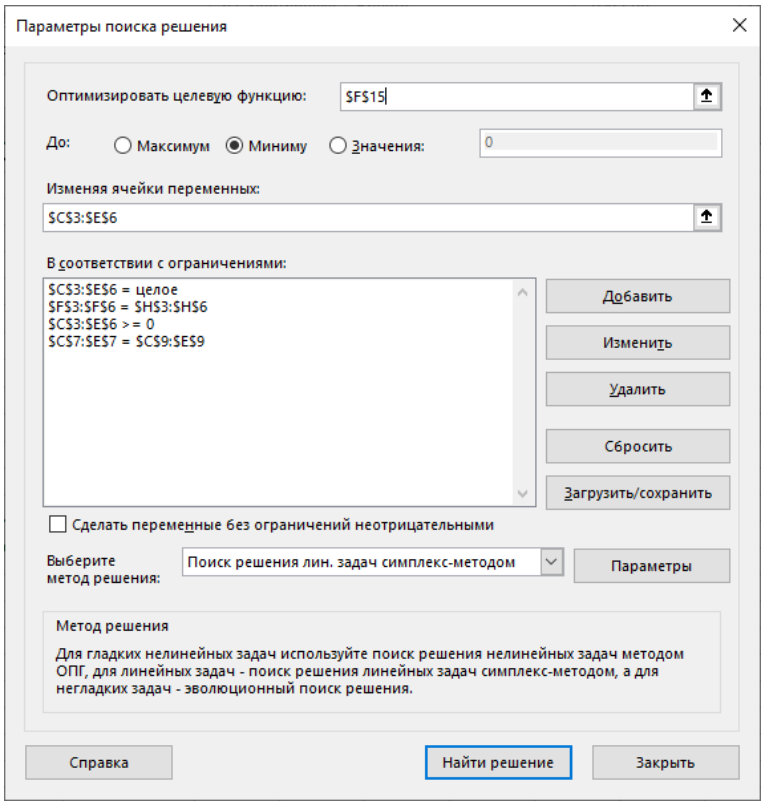
Ввод условий задачи

Экранная форма для задания переменных, целевой функции, ограничений и граничных условий транспортной задачи (8) может иметь вид

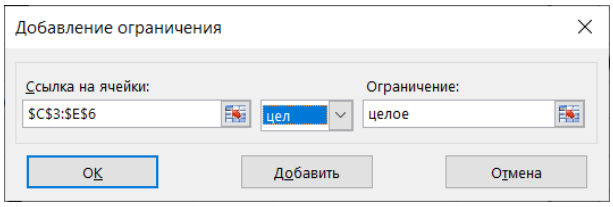
	A	B	C	D	E	F	G	H
1		**Переменные**				**Ограничения**		
2		целые	xi1	xi2	xi3	Лев. Часть	Знак	Прав. Часть
3		x1j				0	=	25
4		x2j				0	=	50
5		x3j				0	=	35
6		x4j				0	=	75
7	**Ограничения**	Лев. Часть	0	0	0			
8		Знак	=	=	=			185
9		Прав. Часть	45	90	50	185		**Баланс**
10								
11		**Тарифы**	xi1	xi2	xi3			
12		x1j	2	9	7			
13		x2j	1	0	5			
14		x3j	5	4	100	Значение	Направление	
15		x4j	2	3	6	0	Min	

Формулы экранной формы

Объект математической модели	Выражение в Excel
Переменные задачи x_{ij}	C3:E6
Формула ЦФ в ячейке F15	=СУММПРОИЗВ(C3:E6;C12:E15)
Ограничения по строкам в ячейках F3 F4 F5 F6	 =СУММ(C3:E3) =СУММ(C4:E4) =СУММ(C5:E5) =СУММ(C6:E6)
Ограничения по столбцам в ячейках C7 D7 E7	 =СУММ(C3:C6) =СУММ(D3:D6) =СУММ(E3:E6)
Суммарные запасы и потребности H8 G9	 =СУММ(H3:H6) =СУММ(C9:E9)



Для ввода ограничения на целочисленность переменных x_{ij} следует в окне Добавление ограничения в поле ввода знака ограничения установить «целое».

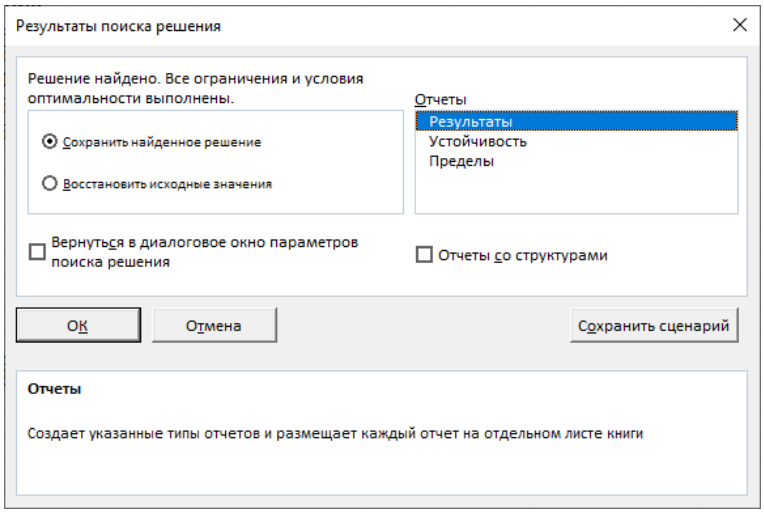


В результате поиска решения экранная форма для транспортной задачи (11) будет иметь вид

	A	B	C	D	E	F	G	H
1		**Переменные**				**Ограничения**		
2		целые	xi1	xi2	xi3	Лев. Часть	Знак	Прав. Часть
3		x1j	25	0	0	25	=	25
4		x2j	0	50	0	50	=	50
5		x3j	0	35	0	35	=	35
6		x4j	20	5	50	75	=	75
7	**Ограничения**	Лев. Часть	45	90	50			
8		Знак	=	=	=			185
9		Прав. Часть	45	90	50	185		**Баланс**
10								
11		**Тарифы**	xi1	xi2	xi3			
12		x1j	2	9	7			
13		x2j	1	0	5			
14		x3j	5	4	100	Значение	Направление	
15		x4j	2	3	6	545	Min	

3.3. Анализ оптимального решения

Для проведения анализа чувствительности задачи об оптимальном плане перевозок необходимо в окне «Результаты поиска решения» выделить тип отчета – «Результаты». При этом в рабочей книге будет создан новый лист «Отчет по результатам».



Лист «Отчет по результатам»

Параметры поиска решения

Максимальное время 100 с, Число итераций 100, Precision 0,000001
Максимальное число подзадач Без пределов, Максимальное число целочисленных решений Без пределов, Целочисленное отклонение 5%,

Ячейка целевой функции (Минимум)

Ячейка	Имя	Исходное значение	Окончательное значение
$F$15	x4j Значение	545	545

Ячейки переменных

Ячейка	Имя	Исходное значение	Окончательное значение	Целочисленное
$C$3	x1j xi1	25	25	Целочисленное
$D$3	x1j xi2	0	0	Целочисленное
$E$3	x1j xi3	0	0	Целочисленное
$C$4	x2j xi1	0	0	Целочисленное
$D$4	x2j xi2	50	50	Целочисленное
$E$4	x2j xi3	0	0	Целочисленное
$C$5	x3j xi1	0	0	Целочисленное
$D$5	x3j xi2	35	35	Целочисленное
$E$5	x3j xi3	0	0	Целочисленное
$C$6	x4j xi1	20	20	Целочисленное
$D$6	x4j xi2	5	5	Целочисленное
$E$6	x4j xi3	50	50	Целочисленное

Ограничения

Ячейка	Имя	Значение ячейки	Формула	Состояние	Допуск
$F$3	x1j Лев. Часть	25	$F$3=$H$3	Привязка	0
$F$4	x2j Лев. Часть	50	$F$4=$H$4	Привязка	0
$F$5	x3j Лев. Часть	35	$F$5=$H$5	Привязка	0
$F$6	x4j Лев. Часть	75	$F$6=$H$6	Привязка	0
$C$7	Лев. Часть xi1	45	$C$7=$C$9	Привязка	0
$D$7	Лев. Часть xi2	90	$D$7=$D$9	Привязка	0
$E$7	Лев. Часть xi3	50	$E$7=$E$9	Привязка	0
$C$3	x1j xi1	25	$C$3>=0	Без привязки	25
$D$3	x1j xi2	0	$D$3>=0	Привязка	0
$E$3	x1j xi3	0	$E$3>=0	Привязка	0
$C$4	x2j xi1	0	$C$4>=0	Привязка	0
$D$4	x2j xi2	50	$D$4>=0	Без привязки	50
$E$4	x2j xi3	0	$E$4>=0	Привязка	0
$C$5	x3j xi1	0	$C$5>=0	Привязка	0
$D$5	x3j xi2	35	$D$5>=0	Без привязки	35
$E$5	x3j xi3	0	$E$5>=0	Привязка	0
$C$6	x4j xi1	20	$C$6>=0	Без привязки	20
$D$6	x4j xi2	5	$D$6>=0	Без привязки	5
$E$6	x4j xi3	50	$E$6>=0	Без привязки	50
$C$3:$E$6=Целочисленное					

4. ЗАДАЧА О НАЗНАЧЕНИЯХ

4.1. Общая постановка задачи

Задача о назначениях – это ресурсная задача, в которой для выполнения каждой работы требуется один и только один ресурс (один человек, одна автомашина и т.д.), а каждый ресурс может быть использован на одной и только одной работе. То есть ресурсы не делимы между работами, а работы не делимы между ресурсами. Таким образом, задача о назначениях является частным случаем транспортной задачи. Задача о назначениях имеет место при назначении людей на должности или работы, автомашин на маршруты, водителей на машины, при распределении групп по аудиториям, научных тем по научно-исследовательским лабораториям и т.п.

Рассмотрим следующий пример.

Некоторая организация проводит конкурс на занятие пяти вакансий (V_1, V_2, V_3, V_4, V_5). В конкурсе участвуют семь претендентов (P_1, P_2, P_3, P_4, P_5, P_6, P_7).

Результаты тестирования каждого претендента на соответствующие вакансии представлены в виде матрицы C (тестирование производилось по десятибалльной системе).

	V_1	V_2	V_3	V_4	V_5
P_1	7	5	7	6	7
P_2	6	4	8	4	9
P_3	8	6	4	3	8
P_4	7	7	8	5	7
P_5	5	9	7	9	5
P_6	6	8	6	4	7
P_7	7	7	8	6	4

Требуется определить, какого претендента и на какую вакансию следует принять, причем так, чтобы сумма баллов всех претендентов оказалась максимальной.

Ведем переменные x_{ij}, которые могут принимать два значения:

$$x_{ij} = \begin{cases} 1, \text{если претендент } P_i \text{ принимается на вакансию } V_j \\ 0, \text{если претендент } P_i \text{ не принимается на вакансию } V_j \end{cases}$$

Очевидно, что все переменные задачи неотрицательные и целые числа: $x_{ij} \geq 0$ и x_{ij} — целые.

Кроме этого, так как каждый претендент может занять только одну вакансию и все вакансии должны быть заняты, должны удовлетворяться следующие ограничения (аналог ограничений 8):

$$\sum_{j=1}^{N} x_{ij} = 1, \quad i = 1,2,...,M.$$

$$\sum_{i=1}^{M} x_{ij} = 1, \quad j = 1,2,...,N.$$

В нашем случае N = 5, M = 7.

Другими словами, в матрице (x_{ij}) суммы элементов в каждой строке и суммы элементов в каждом столбце должны быть равны единицам. Это условие означает, что выбор претендентов должен быть таким, чтобы в матрице (x_{ij}), представляющей решение задачи, было бы по одной единице в каждой строке и по одной единице в каждом столбце, остальные элементы матрицы должны равняться нулю.

Целью задачи является выбор претендентов таким образом, чтобы суммарное число баллов, набранное ими, было бы максимальным. Суммарное число набранных баллов вычисляется по формуле

$$Z = \sum_{i=1}^{M} \sum_{j=1}^{N} c_{ij} x_{ij}.$$

Тогда окончательно математическая модель задачи записывается так:

$$\begin{aligned}
&Z = \sum_{i=1}^{M} \sum_{j=1}^{N} c_{ij} x_{ij} \quad \rightarrow \max \\
&\sum_{j=1}^{N} x_{ij} = 1, \quad i = 1,2,...,M \\
&\sum_{i=1}^{M} x_{ij} = 1, \quad j = 1,2,...,N \\
&\forall x_{ij} \geq 0, \forall x_{ij} - \text{целые} \, (i = \overline{1,M};\ j = \overline{1,N}).
\end{aligned} \tag{12}$$

4.2. Ввод условий задачи

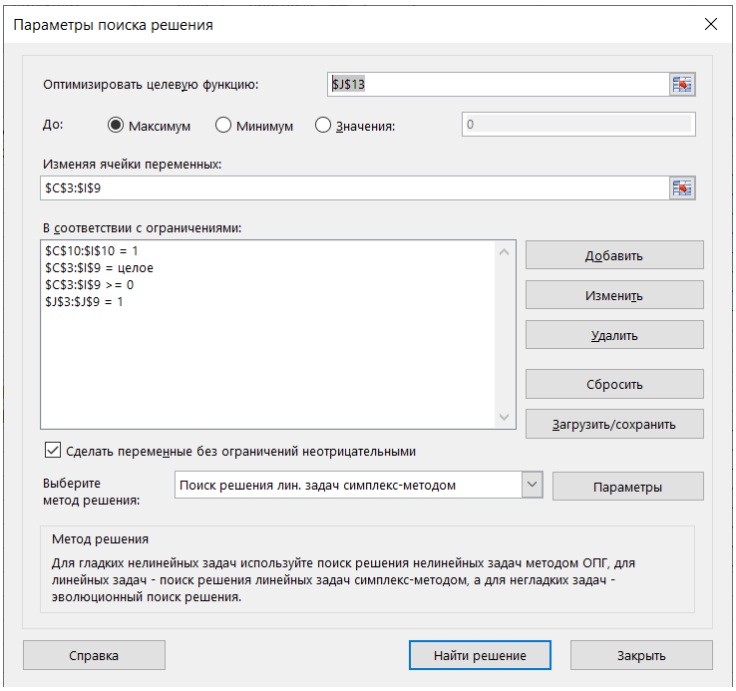
В силу того, что задача должна быть сбалансированной, в таблицу баллов и, соответственно в матрицу переменных, вводим два фиктивных столбца (6-й и 7-й).

Экранная форма для задания переменных, целевой функции, ограничений и граничных условий может иметь вид

	A	B	C	D	E	F	G	H	I	J	K	L
1		**Переменные**								**Ограничения**		
2		целые	xi1	xi2	xi3	xi4	xi5	xi6	xi7	Лев. Часть	Знак	Прав. Часть
3		x1j								0	=	1
4		x2j								0	=	1
5		x3j								0	=	1
6		x4j								0	=	1
7		x5j								0	=	1
8		x6j								0	=	1
9		x7j								0	=	1
10	**Ограничения**	Лев. Часть	0	0	0	0	0	0	0			
11		Знак	=	=	=	=	=	=	=	**ЦФ**		
12		Прав. Часть	1	1	1	1	1	1	1	**Значение**	Направление	
13										0	Max	
14		**Баллы**				Вакансии						
15		Претенденты	1	2	3	4	5	6	7			
16		1	7	5	7	6	7	0	0			
17		2	6	4	8	4	9	0	0			
18		3	8	6	4	3	8	0	0			
19		4	7	7	8	5	7	0	0			
20		5	5	9	7	9	5	0	0			
21		6	6	8	6	4	7	0	0			
22		7	7	7	8	6	4	0	0			

Формулы экранной формы

Объект математической модели	Выражение в Excel
Таблица баллов (матрица С) C16:G22 Фиктивные вакансии H16:I22	 Заданные значения =0
Переменные задачи x_{ij}	C3:I9
ЦФ в ячейке J13	=СУММПРОИЗВ(C16:I22;C3:I9)
Ограничения по строкам в ячейках J3 J4 … J9	 =СУММ(C3:I3) =СУММ(C4:I4) … =СУММ(C9:I9)
Ограничения по столбцам в ячейках C10 D10 … I10	 =СУММ(C3:C9) =СУММ(D3:D9) … =СУММ(I3:I9)



Окно «Поиск решения» заполняется по аналогии с предыдущими задачами.

Выполнив процедуру «Поиск решения», получим следующие результаты.

	A	B	C	D	E	F	G	H	I	J	K	L
1		**Переменные**								**Ограничения**		
2		целые	xi1	xi2	xi3	xi4	xi5	xi6	xi7	Лев. Часть	Знак	Прав. Часть
3		x1j							1	1	=	1
4		x2j					1			1	=	1
5		x3j	1							1	=	1
6		x4j						1		1	=	1
7		x5j				1				1	=	1
8		x6j		1						1	=	1
9		x7j			1					1	=	1
10	**Ограничения**	Лев. Часть	1	1	1	1	1	1	1			
11		Знак	=	=	=	=	=	=	=	**ЦФ**		
12		Прав. Часть	1	1	1	1	1	1	1	**Значение**	Направление	
13										42	Max	
14		**Баллы**				Вакансии						
15		Претенденты	1	2	3	4	5	6	7			
16		1	7	5	7	6	7	0	0			
17		2	6	4	8	4	9	0	0			
18		3	8	6	4	3	8	0	0			
19		4	7	7	8	5	7	0	0			
20		5	5	9	7	9	5	0	0			
21		6	6	8	6	4	7	0	0			
22		7	7	7	8	6	4	0	0			

Результаты расчетов показывают, что претенденты P_1 и P_4 попадают на фиктивные вакансии и не принимаются на работу. P_2 принимается на пятую вакансию, P_3 — на первую, P_5 — на четвертую, P_6 — на вторую, P_7 — на третью.

Сумма баллов, полученная при данном решении, равна 9 + 8 + 8 + 9 + 8 = 42.

5. ЗАДАЧА О РАСКРОЕ

5.1. Общая постановка задачи

Основы линейного программирования были заложены в 1930-х гг. молодым ленинградским математиком В.Л.Канторовичем при работе над социальным заказом — минимизацией отходов авиационной фанеры.

Исходная проблема заключается в нахождении оптимального раскроя партии листов (фанеры, бумаги, металла и т.п.) на заготовки заданных размеров.

Рассмотрим эту задачу на примере раскроя прутьев.

В производстве железобетонных изделий (блоков) прутья арматуры стандартной длины режут на детали указанной в проекте длины, из которых вяжут арматурный каркас. Детали имеют различную длину. Необходимо так разрезать заготовки, чтобы, затратив наименьшее их количество, изготовить заданное количество комплектов арматуры, либо из заданного количества заготовок получить наибольшее количество комплектов.

Введем обозначения:

$i = 1,\ldots, m$ – номера деталей в комплекте;

k_i – количество i-х деталей в комплекте;

L – длина заготовки;

l_i – длина i-й детали;

Q – количество заготовок;

N – количество комплектов.

Каждую заготовку можно разрезать на различное количество деталей. Нужно составить всевозможные варианты раскроя.

$j = 1, \ldots, n$ – номера вариантов раскроя;

x_j – количество заготовок, раскраиваемых по j-му варианту;

a_{ij} – количество i-х деталей в j-м варианте раскроя.

Матрица вариантов раскроя (рис. 2.20) формируется на основе исходных данных.

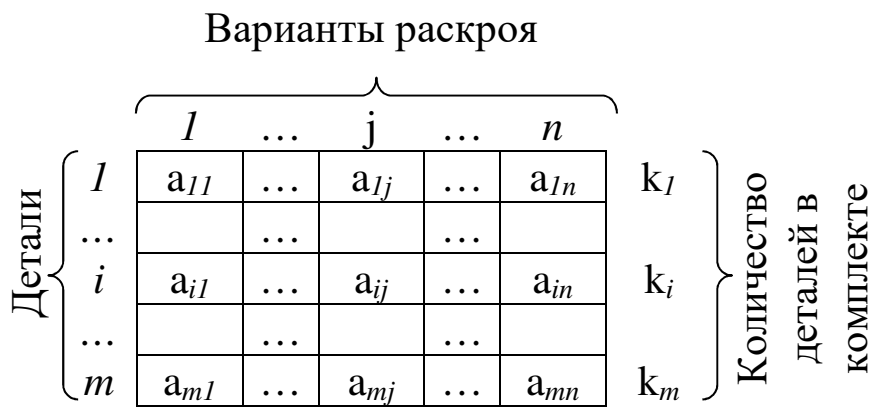


Рис. 2.20

Задача может решаться в двух вариантах:

1. Задано количество заготовок *Q*, нужно максимизировать количество комплектов *N*.

Математическая модель этой задачи будет выглядеть так:

$$\begin{cases} N \to \max \\ \sum_{j=1}^{n} a_{ij} x_j \ge k_i N, \quad i = 1,2,...,m \\ \sum_{j=1}^{n} x_j \ge Q \\ x_j \ge 0; \ x_j - целые. \end{cases} \qquad (13)$$

При этом нельзя требовать, чтобы количество комплектов *N* было целым. Конечно же, можно использовать только целое число комплектов, тогда некоторое количество деталей будет лишним (т.е. в полученном решении выделим целую часть).

2. По заданному количеству комплектов *N* требуется получить минимальное количество разрезаемых заготовок *Q*.

В этом случае задача (13) преобразуется к виду

$$\begin{cases} Q = \sum_{j=1}^{n} x_j \to \min \\ \sum_{j=1}^{n} a_{ij} x_j \ge k_i N, \quad i = 1,2,...,m \\ x_j \ge 0; \ x_j - целые. \end{cases} \qquad (14)$$

Поясним смысл ограничений:

$a_{ij}x_j$ – количество i-х деталей, которые можно получить, раскроив x_j заготовок по j-му варианту;

$\sum_{j=1}^{n} a_{ij}x_j$ – общее количество i-х деталей, получаемое по всем вариантам раскроя;

k_iN – количество i-х деталей, которое требуется для N комплектов;

$x_j \geq 0$ – по j-му варианту разрезается x_j заготовок либо нисколько не разрезается;

x_j – целое; здесь появляется требование целочисленности решения, так как резать нужно целое количество заготовок.

5.2. Решение задачи о раскрое

Пусть имеются заготовки длиной 7,15 м. Из них нужно нарезать детали длиной 1,4 и 2 м. В комплект входят 5 деталей длиной 1,4 м и 3 детали длиной 2 м. Требуется получить $N = 300$ комплектов. Нужно минимизировать количество разрезаемых заготовок Q.

Составим варианты раскроя.

Из заготовки можно нарезать 0, 1, 2 или 3 длинных деталей (2 м). Если не резать ни одной длинной детали, то из заготовки можно нарезать 5 коротких (1,4 м). При одной длинной детали из остатка можно получить 3 коротких, при двух длинных – 2 коротких, при трех длинных – 0 коротких. Это все 4 варианта.

Варианты раскроя запишем в виде таблицы (матрицы вариантов раскроя):

Номер варианта (j)	1	2	3	4
Количество первых деталей (a_{1j})	5	3	2	0
Количество вторых деталей (a_{2j})	0	1	2	3

Рис. 2.21. Матрица вариантов раскроя

Тогда математическая модель (14) для этой конкретной задачи будет выглядеть так:

$$x_1 + x_2 + x_3 + x_4 \to \min$$
$$5x_1 + 3x_2 + 2x_3 + 0x_4 \geq 5N;$$
$$0x_1 + 1x_2 + 2x_3 + 3x_4 \geq 3N;$$
$$x_j \geq 0; \quad x_j \text{ – целые.}$$

Ввод условий задачи

Экранная форма для задания переменных, целевой функции, ограничений и граничных условий задачи о раскрое может иметь вид:

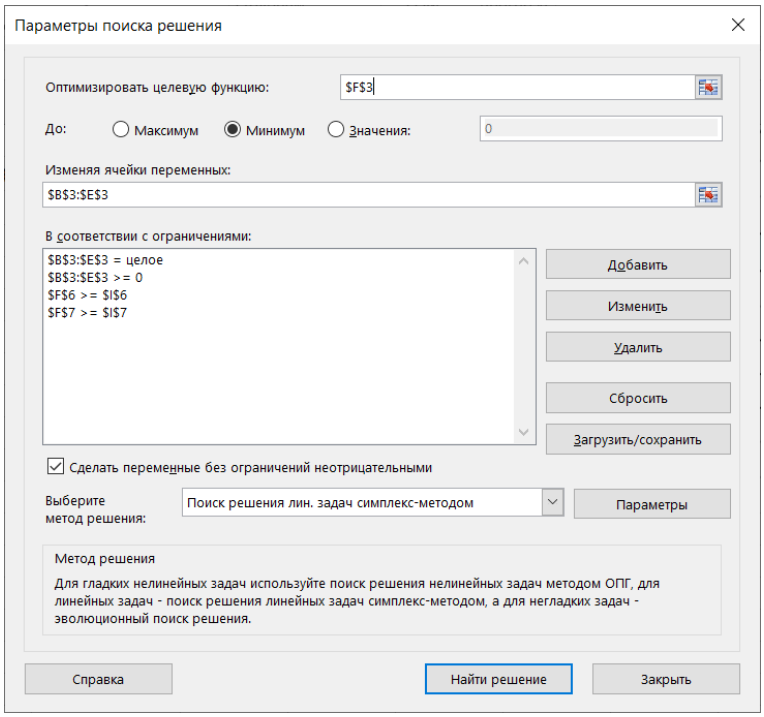
	A	B	C	D	E	F	G	H	I
1	**Переменные**					**ЦФ**			
2	Целые	x1	x2	x3	x4	**Значение**	Направление		
3						0	Min		
4	Ограничения							k	kN
5		Вар1	Вар2	Вар3	Вар4	Лев. часть	Знак		Прав. часть
6	a1j	5	3	2	0	0	>=	5	1500
7	a2j	0	1	2	3	0	>=	3	900
8									
9	**N=**	**300**							

Рис. 2.22. Экранная форма

Формулы экранной формы

Объект математической модели	Выражение в Excel
Переменные задачи x_j	B3:E3
Матрица деталей по вариантам	B6:E7
N – количество комплектов B9	300
Количество деталей в 1-м комплекте H6 во 2-м комплекте H7	 5 3
Количество деталей в заданном количестве комплектов: в 1-м комплекте I6 во 2-м комплекте I7	 =$B$9*H6 =$B$9*H7
Количество нарезанных деталей: в 1-ом комплекте F6 во 2-ом комплекте F7	 =СУММПРОИЗВ(B$3:E$3;B6:E6) =СУММПРОИЗВ(B$3:E$3;B7:E7)
ЦФ - Количество заготовок F3	=СУММ(B3:E3)
Ограничения по количеству деталей в 1-м комплекте во 2-м комплекте	 F6 ≥ I6 F7 ≥ I7
Целочисленность x_j, $x_j \geq 0$	

Выполняя Данные > Поиск решения... и заполнив соответствующие поля в форме Поиск решения...



Получим результат решения задачи о раскрое, который представлен на экранной форме (рис. 2.23):

	A	B	C	D	E	F	G	H	I
1	**Переменные**					**ЦФ**			
2	Целые	x1	x2	x3	x4	**Значение**	Направление		
3		120	0	450	0	570	Min		
4	Ограничения							k	kN
5		Вар1	Вар2	Вар3	Вар4	Лев. часть	Знак		Прав. часть
6	a1j	5	3	2	0	1500	>=	5	1500
7	a2j	0	1	2	3	900	>=	3	900
8									
9	**N=**	**300**							

Рис. 2.23. Оптимальное решение

На рис. 2.23 видно, что в оптимальном решении:

x_1 = В3 = 120, x_2 = С3 = 0, x_3 = D3 = 450, x_4 = Е3 = 0.

Получено значение целевой функции, равное 570, это значит, что нужно разрезать 570 заготовок, чтобы получить 300 комплектов.

При этом:

по 1-му варианту нужно разрезать 120 заготовок;

по 3-му – 450 заготовок;

2-й и 4-й варианты не используются.

5.3. Построение матрицы вариантов раскроя

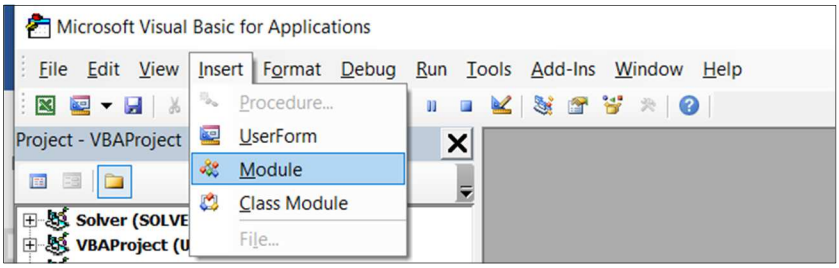
В приведенном примере использовались 2 детали определенной длины. В результате несложных рассуждений были получены 4 варианта раскроя.

В общем случае для получения матрицы вариантов раскроя придется решить следующую задачу.

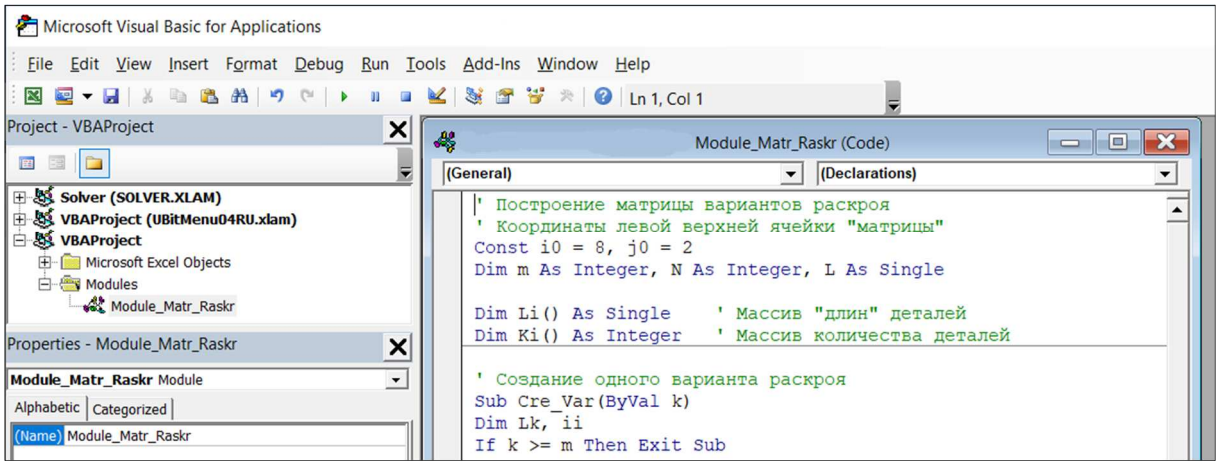
Пусть имеются заготовки длиной **L** м. Из них нужно нарезать **m** деталей, длины которых задаются элементами массива **Li**. Требуется получить матрицу A вариантов раскроя.

Это можно сделать средствами VBA.

Чтобы вставить приведенный текст программы, необходимо перейти в редактор Visual Basic: Разработчик > Visual Basic или нажать комбинацию клавиш ALT + F11. Далее — вставить модуль



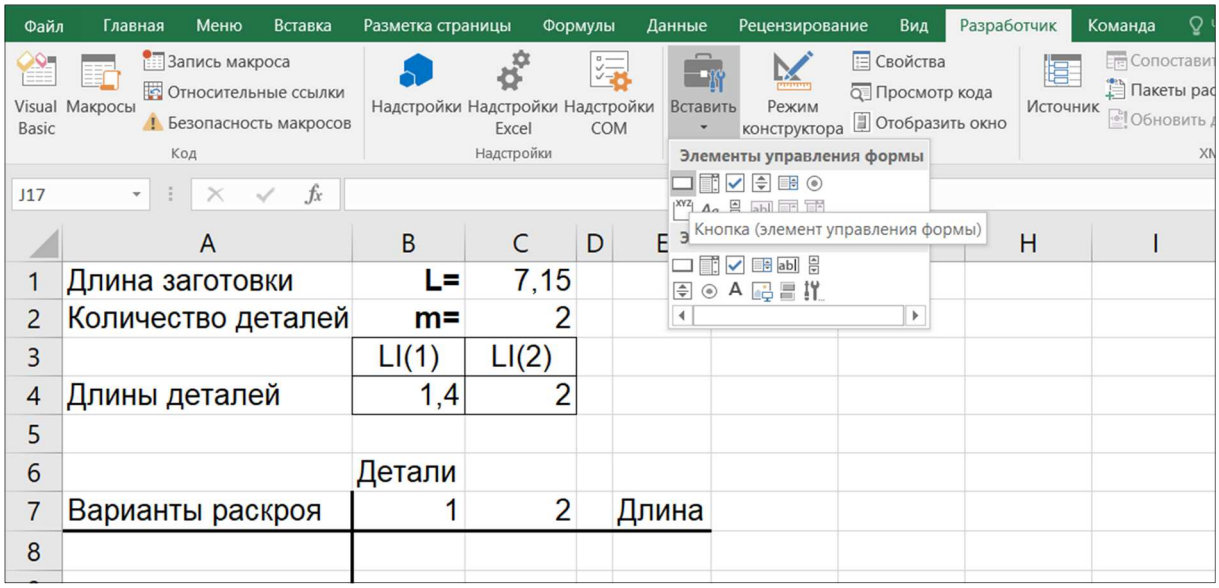
Дать этому модулю имя, например, Module_Matr_Raskr.



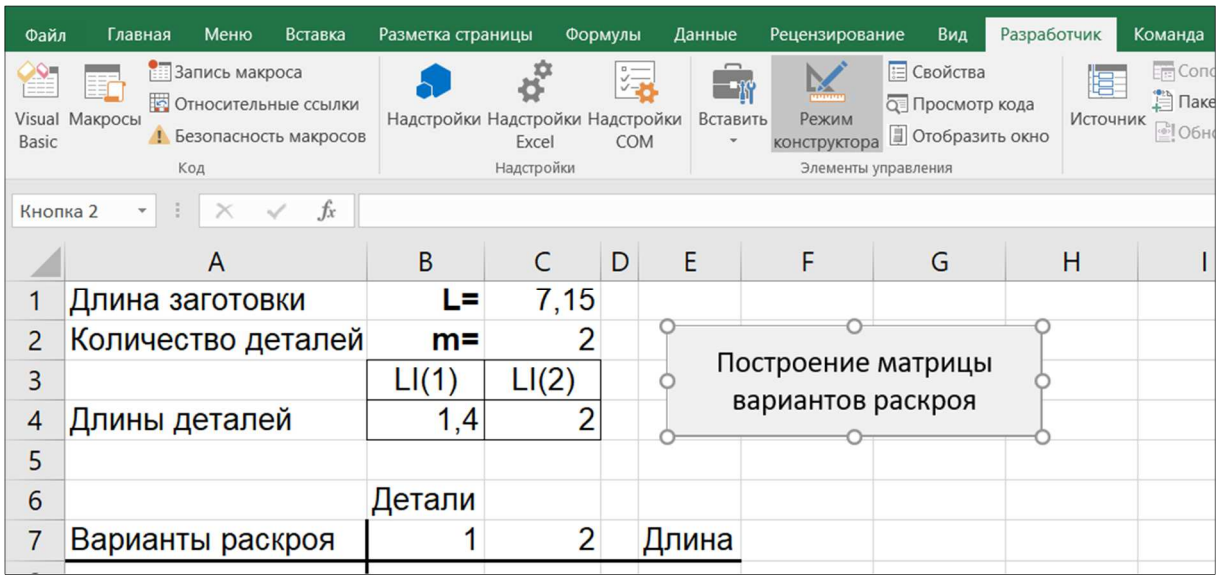
Вставить приведенный далее текст программы в окно редактирования кода Module_Matr_Raskr (Code).

Активизировать макрос построения матрицы вариантов раскроя можно несколькими способами.

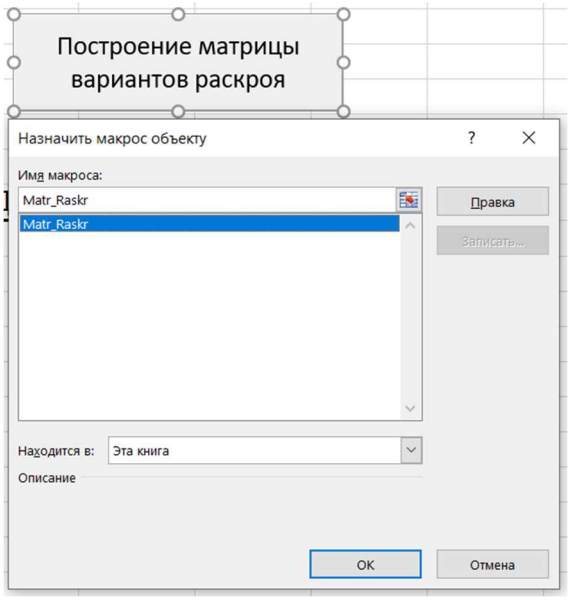
Поместить кнопку ла лист.



В режиме конструктора изменить имя.



И назначить макрос этому объекту (кнопке) — на кнопке нажать правую кнопку мыши и выбрать Назначить макрос объекту и указать имя нашего макроса Matr_Raskr.



Иначе запустить макрос на выполнение можно, выбрав пункт меню Разработчик > Макросы или нажав комбинацию клавиш ALT + F8.

Также из главного меню: Вид > Макросы.

Так как количество вариантов раскроя заранее неизвестно, удобно использовать ячейки листа для вывода очередного варианта, чтобы получить вид, приведенный на рис. 2.21. Точнее, на листе для удобства вывода получаем транспонированную матрицу.

На рис. 2.24 приведен фрагмент листа с результатами работы программы.

	A	B	C	D	E
1	Длина заготовки	**L=**	7,15		
2	Количество деталей	**m=**	2		
3		LI(1)	LI(2)		
4	Длины деталей	1,4	2		
5					
6		Детали			
7	Варианты раскроя	1	2		Длина
8	1	5	0		7
9	2	4	0		5,6
10	3	3	1		6,2
11	4	2	2		6,8
12	5	1	2		5,4

Рис. 2.24. Результаты работы программы

Далее приведены процедуры модуля для построения матрицы вариантов раскроя.

```
' Построение матрицы вариантов раскроя
' Координаты левой верхней ячейки «матрицы»
Const i0 = 8, j0 = 2
Dim m As Integer, N As Integer, L As Single

Dim Li() As Single    ' Массив «длин» деталей
Dim Ki() As Integer   ' Массив количества деталей

' Создание одного варианта раскроя
Sub Cre_Var(ByVal k)
Dim Lk, ii
If k >= m Then Exit Sub
  ' Какая общая длина предыдущих вариантов
  Lk = 0
  For ii = 1 To k
    Lk = Lk + Ki(ii) * Li(ii)
  Next ii
  L0 = L - Lk ' Остаток
  k = k + 1
  ' Max количество одной след. детали
  Ki(k) = Int(L0 / Li(k))
  Call Cre_Var(k)   ' Рекурсивный вызов
End Sub

Sub Matr_Raskr()
Dim Lk, ii
L = Cells(1, 3)              ' Длина заготовки
m = Cells(2, 3)              ' Количество деталей
' Описываем массив «длин» деталей
ReDim Li(1 To m) As Single
' Описываем массив количества деталей
ReDim Ki(1 To m) As Integer
For i = 1 To m
  ' Формируем массив «длин» деталей
  Li(i) = Cells(4, i + 1)
Next i
N = 0
' Max количество одной 1-й детали
Ki(1) = Int(L / Li(1))
For i = 1 To m - 1
```

```
  Lk = 0
  For ii = 1 To i
    Lk = Lk + Ki(ii) * Li(ii)
  Next ii
  ' Max количество одной i-й детали
  Ki(i) = Int(Lk / Li(i))
  While Ki(i) > 0
    N = N + 1
    ' Очистка одного варианта
    For ii = i + 1 To m
      Ki(ii) = 0
    Next ii
    ' Создание нового варианта
    Call Cre_Var(i)
    ' Вывод варианта
    Call Out_Variant(N, Ki, Li)
    ' Очистка одного варианта
    For ii = 1 To i - 1
      Ki(ii) = 0
    Next ii
    Ki(i) = Ki(i) - 1
  Wend
Next i
End Sub

' Вывод варианта
Sub Out_Variant(N, Ki, Li)
  Cells(i0 + N - 1, j0 - 1) = N
  Ls = 0                        ' Общая длина варианта
  For j = 1 To m
    Cells(i0 + N - 1, j0 + j - 1) = Ki(j)
    Ls = Ls + Ki(j) * Li(j)
  Next j
  Cells(i0 + N - 1, j0 + m + 1) = Ls
End Sub
```

В приведенном модуле имеется три процедуры.

Варианты вызова процедуры **Sub Matr_Raskr()** мы рассмотрели ранее. В этой процедуре организован перебор вариантов с использованием рекурсивной процедуры **Sub Cre_Var(ByVal k).**

Процедура **Sub Out_Variant** используется для вывода очередного варианта на лист.

Сравнивая полученные результаты с теми, которые приведены на рис. 2.24, можно заметить, что вариант 2 является заведомо худшим, чем вариант 1, т.е. его можно не включать в матрицу раскроя.

ПРИЛОЖЕНИЯ

Приложение 1

Финансово-математические функции

Имя функции, параметры	Возвращаемое значение
DDB(Стоимость, Остаточная_стоимость, Время__эксплуатации, Период, Кратность)	Вычисляет амортизацию фондов в течение заданного интервала времени.
FV(Ставка, Кпер, Плата [, Нз[, Тип]])	Вычисляет накопленную стоимость при известном размере регулярного взноса и постоянной процентной ставке.
IPmt(Ставка, Период, Кпер, Нз[, Бз[, Тип]])	Вычисляет сумму выплат при известном размере регулярного взноса и постоянной процентной ставке.
IRR (Величина()[, guess])	Вычисляет внутреннюю норму доходности при известной последовательности выплат и поступлений.
MIRR(Величина(), Ставка_финанс, Ставка_реинвест)	Вычисляет модифицированную внутреннюю норму доходности.
NPer(CiaBKa, Платеж, Нз, Бз, Тип)	Вычисляет количество периодов времени, необходимых для достижения заданной фактической стоимости при постоянном размере выплат и постоянной процентной ставке.
NPV(CT3BKa, ВеличинаО)	Вычисляет чистую приведенную стоимость инвестиционного проекта при известном размере выплат и поступлений при постоянной дисконтной ставке.
PPMT(Ставка, Период, Кпер, Нз, Бз, Тип)	Вычисляет величину постоянного взноса для достижения определенной суммы при постоянной процентной ставке.
Ргт(Ставка, Кпер, Нз[, Бз[, Тип]])	Аналогична PPMT, но позволяет вычислить величину выплаты в зависимости от того, когда она производится (в начале или в конце периода).
РУ(Ставка, Кпер, Плата[, Бз[, Тип]])	Вычисляет приведенную стоимость при известном и постоянном размере выплаты, периоде и постоянной процентной ставке.
Rate(Knep, Плата, Нз[, Бз[, Тип]])	Вычисляет процентную ставку, необходимую для достижения заданной стоимости при известном периоде выплат.
SLN (Стоимость, Ликвидная_ стоимость. Жизнь)	Вычисляет величину амортизации фондов линейным методом.
SYD(Стоимость, Ликвидная_ стоимость, Жизнь, Период)	Вычисляет величину годовой амортизации фондов за определенный период.

Приложение 2

Текст процедур и функций для формирования вариантов тестирования задания 7 ЕГЭ по информатике.

```
Const Str_Begin = "Задание 7"
Dim Curr_Date As String, Curr_Time As String
Dim Curret_Path As String

Dim Str_All As String
Dim Enter As String

Dim Line As String, Line_Break As String
Dim CountTest
Dim Arr_Otvet()

Dim xx, yy, size
Dim xx0, yy0, size0
Dim xx1, yy1, size1
Dim First_Time As Boolean

Private Sub CB_Create_Test_Click()

' Задаем количество тестов
CountTest = Val(TB_CountTest.Value)

' Переопределяем размерность массива ответов
ReDim Arr_Otvet(1 To CountTest)

First_Time = True

' Формируем нужное количество тестов
For i = 1 To CountTest
    ' Выполняется один раз для каждого запуска
    If First_Time Then
      Call First_Paragraph  ' Вывод первого заголовка
      First_Time = False
    End If

  ' Формируем текст задания
  str1 ="Для хранения произвольного растрового "+_
        "изображения размером "
  str2 =" x"
  str3 =" пикселей отведено "
```

```
  str4 =" Кбайт памяти без учёта размера "+_
        "заголовка файла." + Line_Break
  str5 ="Для кодирования цвета каждого пикселя " + _
        "используется одинаковое количество бит," +_
        "коды пикселей записываются в файл " + _
        "один за другим без промежутков." + _
         Line_Break + _
        "Какое максимальное количество цветов " + _
        "можно использовать в изображении?"
  Randomize
  ' Случайные значения xx из [xx0..xx1]
  xx = Int((xx1 - xx0) * Rnd + xx0)

  ' Случайные значения yy из [yy0..yy1]
  yy = Int(Rnd * (yy1 - yy0)) + yy0

  ' Случайные значения size из [size0..size1]
  size = Int(Rnd * (size1 - size0)) + size0

  ' Формируем ответ для полученных случайных значений
  Count_Bit = Int(size * 1024 * 8 / (xx * yy))
  Count_Color = 2 ^ Count_Bit

  ' Сохраняем ответ в массиве
  Arr_Otvet(i) = Count_Color

  Str_xx = Str(xx)         ' Преобразуем xx в String
  Str_yy = Str(yy)         ' Преобразуем yy в String
  Str_size = Str(size)     ' Преобразуем size в String

   Str_All = str1 + Str_xx + str2 + Str_yy + str3 + _
             Str_size + str4 + str5 + _
             Line             ' Выводим "линию"
  ' Выводим полученный тест (Str_All) в документ Word
  Selection.TypeText i & ") " & Str_All

  ' Выводим 5 тестов на страницу
  If (i Mod 5 = 0) And (i <> CountTest) Then
    Selection.InsertBreak Type:=wdPageBreak
  End If
Next i
MsgBox (CountTest & " тестов готовы..." + Enter)
Call CB_Create_Otvet_Click
End Sub
```

```
Public Sub First_Paragraph()
  Documents.Add Template:="Normal",_
    NewTemplate:=False, DocumentType:=0
  Selection.ParagraphFormat.LineSpacingRule = _
    wdLineSpaceSingle

  Selection.Font.Name = "Times New Roman"
  Selection.Font.size = 16
  Selection.Font.Bold = wdToggle

  Selection.TypeText Str_Begin
  Selection.ParagraphFormat.LineSpacingRule = _
    wdLineSpaceSingle
  Selection.Font.size = 14
  Selection.Font.Bold = wdToggle
  Selection.TypeText " (" + TB_Data_Curr + ")" + Line
End Sub

Private Sub CB_Create_Otvet_Click()
  Documents.Add Template:="Normal", _
    NewTemplate:=False, DocumentType:=0
  Selection.ParagraphFormat.LineSpacingRule = _
    wdLineSpaceSingle

  Selection.Font.Name = "Times New Roman"
  Selection.Font.size = 16
  Selection.Font.Bold = wdToggle
  Selection.TypeText Str_Begin + " Ответы"
  Selection.Font.Bold = wdToggle

  Selection.Font.size = 14
  Selection.TypeText " (" +TB_Data_Curr + ")" + Enter

  For i = 1 To CountTest
    Selection.TypeText i & ")" & Arr_Otvet(i) & Enter
  Next i

  MsgBox (CountTest & " ответов готовы..." + Enter)
End Sub

Private Sub CB_Exit_Click()
  End
End Sub
```

```
Private Sub Enable_CB()
Dim Ok As Boolean
  Ok = (TB_xx0.Text <> "") And _
       (TB_xx1.Text <> "") And _
       (TB_yy0.Text <> "") And _
       (TB_yy1.Text <> "") And _
       (TB_size0.Text <> "") And _
       (TB_size1 <> "") And _
       (TB_CountTest.Text <> "")
End Sub

Private Sub UserForm_Activate()
  Enter = Chr(13) + Chr(10) '= CR+LF - Возврат
каретки и перевод строки
  Line_Break = Chr(11) ' VT - Вертикальная табуляция
  Line = Enter + "-            -" + Enter
  ' Начальные значения переменных
  xx0 = TB_xx0.Value: xx1 = TB_xx1.Value
  yy0 = TB_yy0.Value: yy1 = TB_yy1.Value
  size0 = TB_size0.Value: size1 = TB_size1.Value

  Curret_Path = ThisDocument.Path

  ' Текущие дата и время
  Curr_Date = Format(Date, "dd-mm-yyyy")
  Curr_Time = Format(Time, "hh:mm")
  TB_Data_Curr.Text = Curr_Date + "    " + Curr_Time
  ' Активность кнопки CB_Create_Data
  Call Enable_CB
End Sub
```

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Гарнаев А.Ю. Самоучитель VBA. 2-е изд., перераб. и доп. СПб.: БХВ, 2004.
2. Комолова Н.В., Яковлева Е.С. Программирование на VBA в Excel 2016. Самоучитель. СПб.: БХВ-Петербург, 2017.
3. Лебедев В.М. Программирование на VBA в MS Excel: учеб. пособие для СПО. М.: Издательство Юрайт, 2017.
4. Гаркуша О.В. Вычислительная математика и программирование. Краснодар: Кубанский гос.ун-т, 2000. Ч. 1.
5. Берндт Г., Каинка Б. Измерение, управление и регулирование с помощью макросов VBA в Word и Excel. СПб.: «КОРОНА-ВЕК», 2008.
6. Практикум по экономической информатике: в 3 ч.: учеб. пособие для студентов вузов / Е.Л. Шуремов, Н.А. Тимакова, Е.А. Мамонтова; под ред. Е.Л. Шуремова и др. М.: Перспектива, 2004. Ч. 1.
7. Практикум по экономической информатике: в 3 ч.: учеб. пособие для студентов вузов / П.П. Мельников, И.В. Миронова, И.Ю. Шполянская и др.; под ред. П.П. Мельникова. М.: Перспектива, 2002. Ч. 3.
8. Решение задач оптимизации в Microsoft Excel 2010: учеб. пособие / Н.И. Шадрина, Н.Д. Берман; науч. ред. Э.М. Вихтенко. Хабаровск: Изд-во Тихоокеан. гос. ун-та, 2016.
9. Акулич И.Л. Математическое программирование в примерах и задачах. М.: Высшая школа, 1986.
10. Кузнецов А.В., Сакович В.А., Холод Н.И. и др. Сборник задач и упражнений по высшей математике. Математическое программирование. Минск: Вышэйшая школа, 1995.
11. Курицкий Б. Решение оптимизационных задач средствами Excel. М.: BHV, 1997.
12. Эддоус М., Стенсфилд Р. Методы принятия решений. М.: Аудит, ЮНИТИ, 1997.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 3

РАЗДЕЛ 1. СРЕДА ПРОГРАММИРОВАНИЯ VISUAL BASIC FOR APPLICATIONS (VBA) 5

1. VBA КАК СИСТЕМА ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ 5

1.1. Настройка приложений MS Office для работы с макросами и в среде VBA 5

1.2. Общие сведения о VBA 8

1.3. Объектная модель Excel 10

1.4. Основные понятия объектно-ориентированного программирования 12

1.5. Объекты, методы, свойства, события 13

2. ПРОЕКТ VBA И ЕГО ЭЛЕМЕНТЫ 14

2.1. Структура проекта VBA 14

2.2. Структура программы VBA 15

2.3. Типы процедур (функций) и их определение 16

3. СРЕДА РАЗРАБОТКИ 17

3.1. Структура редактора VBA 17

3.1.1. Окно проекта 17

3.1.2. Окно редактирования кода 18

3.1.3. Окно редактирования формы 19

3.2. Панель элементов (ToolBox) 20

3.3. Свойства, методы и события элементов управления 20

3.3.1. Общие свойства элементов управления 21

3.3.2. Общие методы стандартных элементов управления 22

3.3.3. Элемент Кнопка (CommandButton) 22

3.3.4. Элемент Поле (TextBox) 23

3.3.5. Элемент Надпись (Label) 23

4. ОСНОВНЫЕ ПОНЯТИЯ ЯЗЫКА VBA 24

4.1. Инструкции 24

4.2. Имена и идентификаторы 25

4.3. Время жизни переменной 26

4.4. Описания 27

4.4.1. Инструкция Dim 27

4.4.2. Инструкция Public 29
4.4.3. Инструкция Private 29
4.4.4. Инструкция Static 29
4.4.5. Инструкция Option Explicit 30
4.5. Переменная 30
4.6. Функции и процедуры 31
5. ТИПЫ ДАННЫХ 31
5.1. Операции и функции для данных целого типа 33
5.2. Операции и функции для данных вещественного типа 34
5.3. Операции и функции для данных логического типа 35
5.4. Операции и функции для строковых данных (String) 36
5.4.1. Функции, результаты которых имеют числовой тип 37
5.4.2. Функции, результаты которых имеют тип String 38
5.5. Переменные типа денежные значения (Currency) 41
6. ОПЕРАТОРЫ ЯЗЫКА VBA 41
6.1. Оператор присваивания 41
6.2. Ввод и вывод данных 42
6.2.1. Функция InputBox 44
6.2.2. Функция MsgBox 46
6.3. Условная инструкция (If ... Then ... Else) 48
6.4. Оператор выбора варианта (Select Case) 51
6.5. Программирование циклов 53
6.5.1. Оператор цикла с предусловием (While … Wend) 54
6.5.2. Операторы цикла с условиями (Do … Loop) 55
6.5.3. Оператор цикла с параметром (For ... Next) 55
6.5.4. Использование операторов цикла 58
7. ОСНОВНЫЕ ОБЪЕКТЫ MS EXCEL 62
7.1. Объект Application. Свойства, методы и события 62
7.1.1. Свойства объекта Application 62
7.1.2. Основные методы объекта Application 62
7.1.3. События объекта Application 62
7.2. Основные свойства, методы и события семейства WorkBooks 63
7.2.1. Основные свойства объектов семейства Workbooks 63
7.2.2. Основные методы объектов семейства Workbooks 63
7.2.3. События объектов семейства Workbooks 63

7.3. Основные свойства и методы объектов семейства Worksheets. 64
7.3.1. Свойства объектов семейства Worksheets 64
7.3.2. Методы объектов семейства Worksheets 64
7.3.3. События объекта Worksheet 65
7.4. Объект Range 65
7.4.1. Адресация ячеек в Excel 65
7.4.2. Основные свойства объекта Range 66
7.4.3. Основные методы объекта Range 68
7.4.4. Методы объекта Range (команды Excel) 68
7.5. Инструкция With 69
8. МАССИВЫ 69
8.1. Одномерные массивы 70
8.1.1. Изменение нижней границы индексов (Option Base) 71
8.1.2. Изменение границ массивов (ReDim) 72
8.1.3. Ввод – вывод элементов массива 72
8.2. Многомерные массивы 75
8.3. Примеры использования массивов 76
9. ОБРАБОТКА СИМВОЛОВ И СТРОК 80
9.1. Пример поиска первого слова, начинающегося заданной буквой 80
9.2. Пример преобразования целого в символьное: «Сумма прописью» 81
10. ПОДПРОГРАММЫ 87
10.1. Процедуры без параметров 89
10.2. Процедуры с параметрами 90
10.2.1. Параметры-значения 90
10.2.2. Параметры-переменные 93
10.3. Синтаксис процедур 94
10.4. Вызов процедур 96
10.5. Правила соответствия между формальными и фактическими параметрами 97
10.6. Принцип локализации 97
11. ФУНКЦИИ 98
11.1. Описание функций 100
11.2. Вызов функции 102
11.3. Побочный эффект функции 103

11.4. Рекурсивные функции 105
11.5. Пример использования функций 107
11.6. Создание процедур (функций) пользователя 110
12. VBA В MS WORD 113
РАЗДЕЛ 2. ЗАДАЧИ ЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ 116
1. ЗАДАЧА О РАСПРЕДЕЛЕНИИ РЕСУРСОВ 116
1.1. Общая постановка задачи 116
1.2. Ввод условий задачи о распределении ресурсов 118
2. ЗАДАЧА О РАЦИОНЕ 126
2.1. Общая постановка задачи 126
2.2. Ввод условий задачи о рационе 129
2.3. Анализ оптимального решения 131
2.3.1. Отчет по результатам 131
2.3.2. Отчет по устойчивости 133
2.3.3. Отчет по пределам 134
3. ТРАНСПОРТНАЯ ЗАДАЧА 135
3.1. Общая постановка задачи 135
3.2. Решение транспортной задачи 137
3.3. Анализ оптимального решения 139
4. ЗАДАЧА О НАЗНАЧЕНИЯХ 141
4.1. Общая постановка задачи 141
4.2. Ввод условий задачи 143
5. ЗАДАЧА О РАСКРОЕ 145
5.1. Общая постановка задачи 145
5.2. Решение задачи о раскрое 147
5.3. Построение матрицы вариантов раскроя 150
ПРИЛОЖЕНИЯ 156
Приложение 1 156
Приложение 2 157
РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА 161

Учебное издание

Г а р к у ш а Олег Васильевич

ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА MS Office

Учебное пособие

Подписано в печать 17.02.2022. Выход в свет 25.03.2022.
Печать цифровая. Формат 60×84 $^{1}/_{16}$. Уч.-изд. л. 10,1.
Тираж 500 экз. Заказ № 4815.

Кубанский государственный университет
350040, г. Краснодар, ул. Ставропольская, 149.

Издательско-полиграфический центр
Кубанского государственного университета
350040, г. Краснодар, ул. Ставропольская, 149.