Краснодарский университет МВД России

Е. В. Михайленко
А. К. Назаров

ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

Учебное пособие

Краснодар
2019

УДК 004.42
ББК 32.973
М69

Одобрено
редакционно-издательским советом
Краснодарского университета
МВД России

Рецензенты:
П. Н. Жукова, доктор физико-математических наук, доцент (Белгородский юридический институт МВД России имени И.Д. Путилина);
С. В. Крыгин (Нижегородская академия МВД России);
К. А. Стаценко (Главное управление МВД России по Краснодарскому краю).

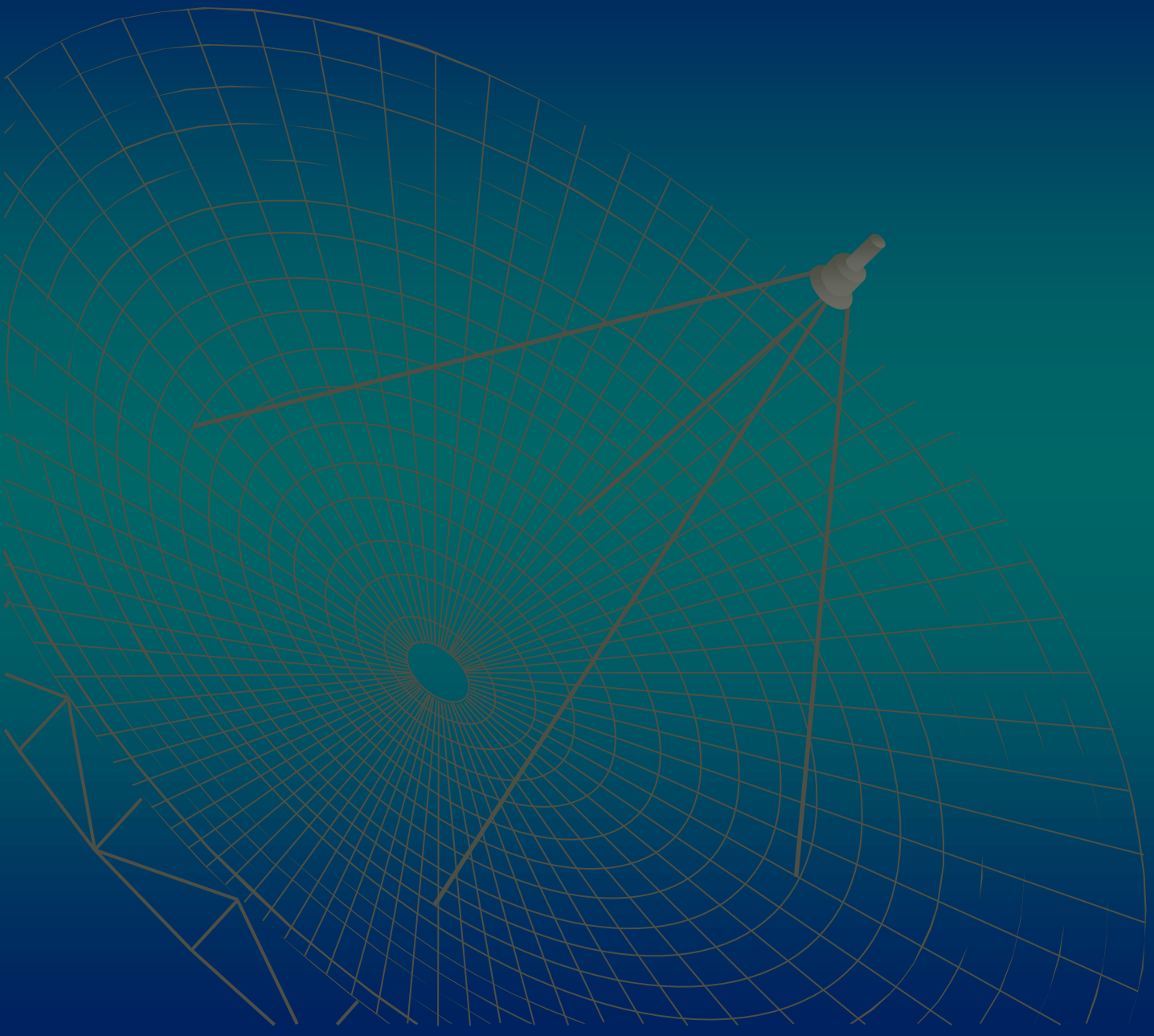
Михайленко Е. В.
М69 Технологии программирования [Электронный ресурс] : учебное пособие / Е. В. Михайленко, А. К. Назаров. – Электрон. дан. – Краснодар : Краснодарский университет МВД России, 2019. – 1 электрон. опт. диск (CD-ROM).

ISBN 978-5-9266-1505-7

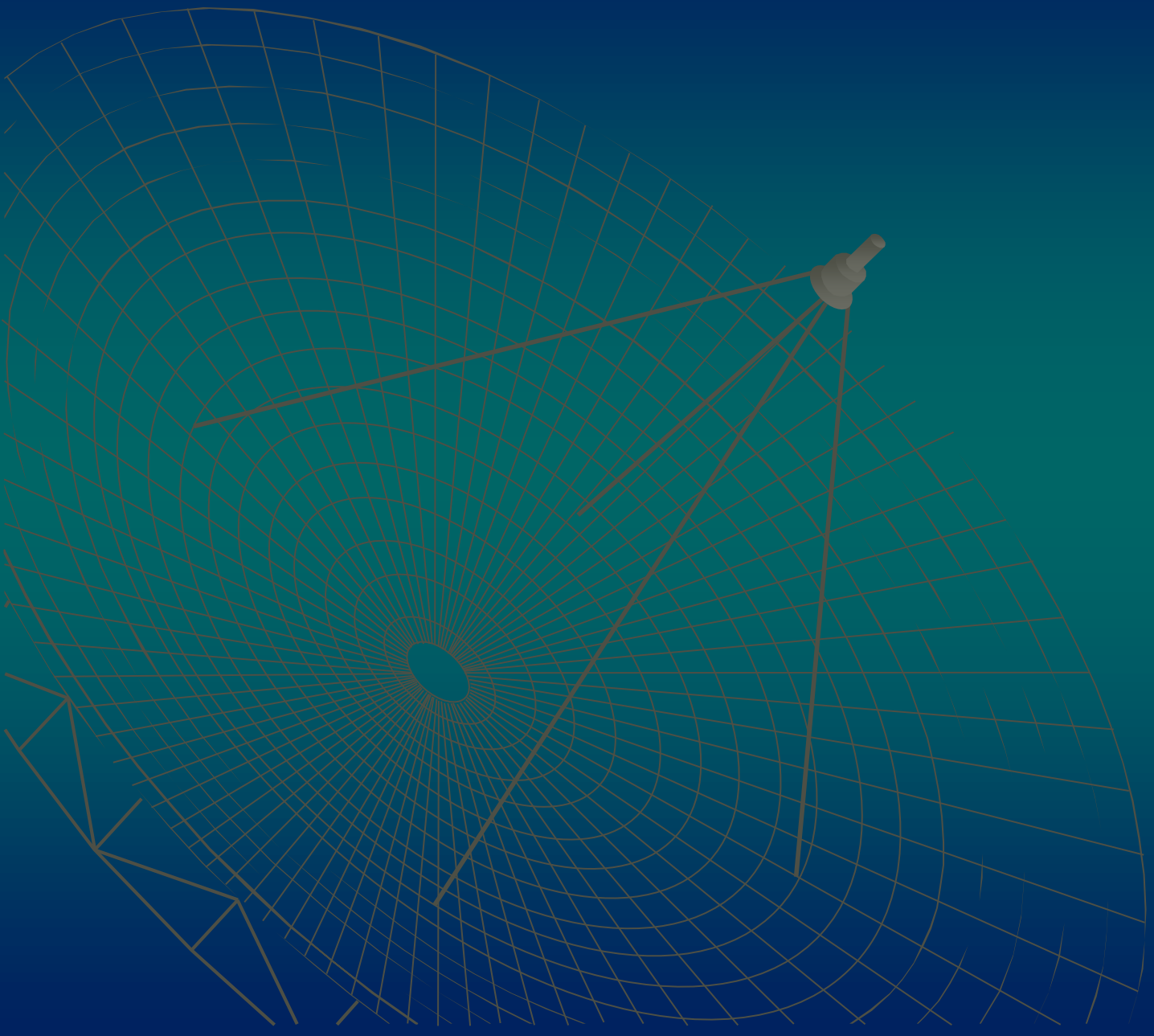
Содержится теоретический материал, описывающий основные понятия программирования, алгоритмы, организацию ветвления программ, циклы и обработку массивов. Представлены основные методы и технологии объектно-ориентированного программирования.
Для профессорско-преподавательского состава, адъюнктов, курсантов, слушателей образовательных организаций МВД России и сотрудников органов внутренних дел Российской Федерации.

УДК 004.42
ББК 32.973

ISBN 978-5-9266-1505-7



Содержание



Глава 1. Алгоритмы

1.1 Основные понятия программирования

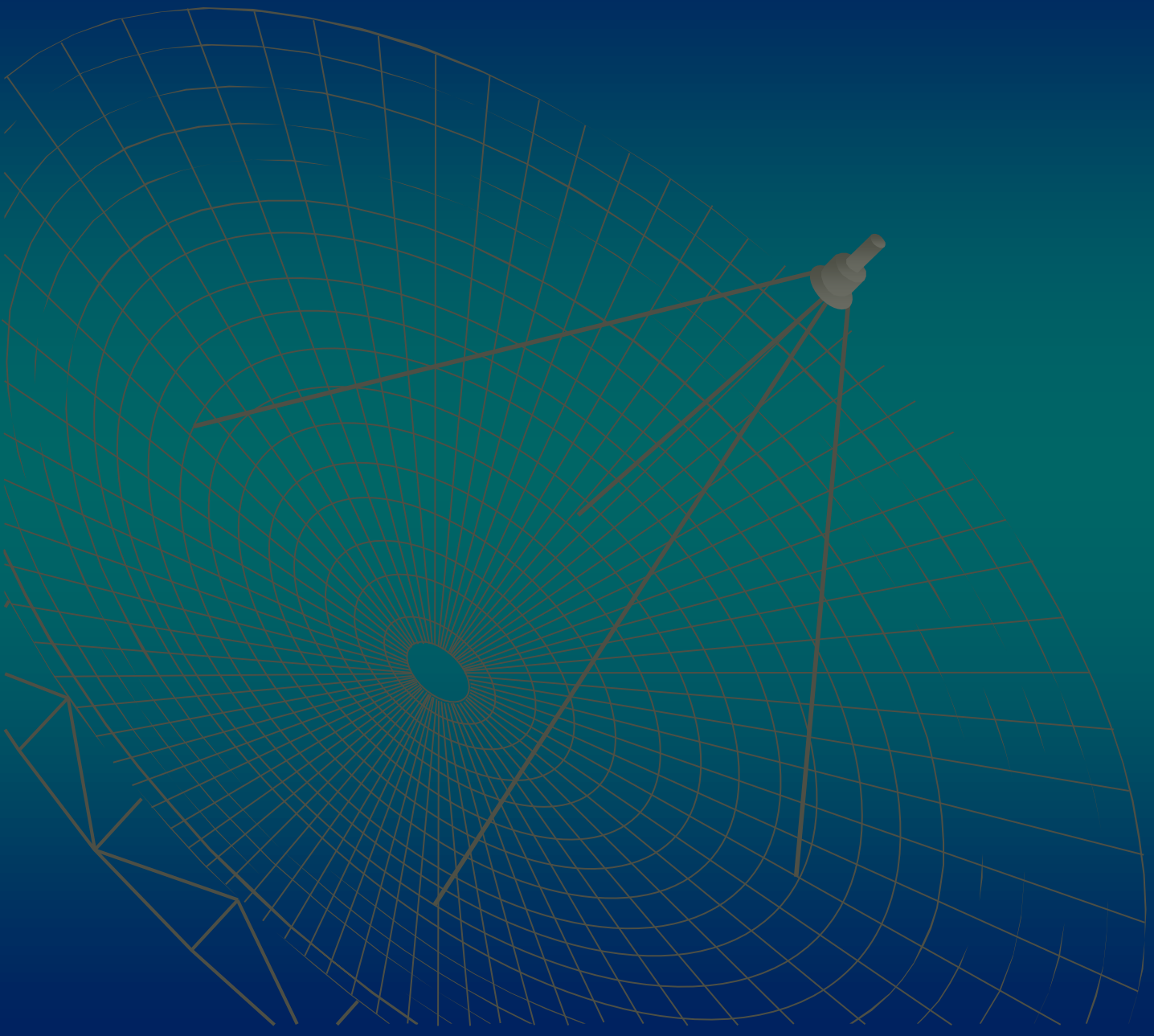
1.2 Понятие алгоритма.

1.3 Основные свойства алгоритмов.

1.4 Способы представления алгоритмов.

1.5 Базовые структуры алгоритмов.

1.6 Решение типовых задач.



1.1 Основные понятия программирования
1.1.1. Понятия информации и данных.

Информация (лат. informatio – разъяснение) – сведения об объектах, явлениях, их параметрах, свойствах, состоянии, которые уменьшают имеющуюся о них степень неопределенности, неполноты знаний. (Н.В. Макарова)

Информация – это обозначение содержания, полученного из внешнего мира в процессе нашего приспособления к нему и приспособления к нему наших чувств. (Н. Винер)

Информация характеризует не сообщение, а соотношения между сообщением и его потребителем. Без наличия потребителя (хотя бы потенциального) говорить об информации бессмысленно.

Данные – хранимые, но не используемые сведения об объектах, явлениях, их параметрах, свойствах, состоянии.

Классификация информации

Информацию можно разделить на виды по различным критериям, например,

по способу восприятия:

визуальная — воспринимаемая органами зрения,

звуковая — воспринимаемая органами слуха,

тактильная — воспринимаемая тактильными рецепторами,

обонятельная — воспринимаемая обонятельными рецепторами,

вкусовая — воспринимаемая вкусовыми рецепторами;

по форме представления:

текстовая — передаваемая в виде символов, предназначенных обозначать лексемы языка,

числовая — в виде цифр и знаков, обозначающих математические действия,

графическая — в виде изображений, предметов, графиков,

звуковая — устная или в виде записи и передачи лексем языка аудиальным путём,

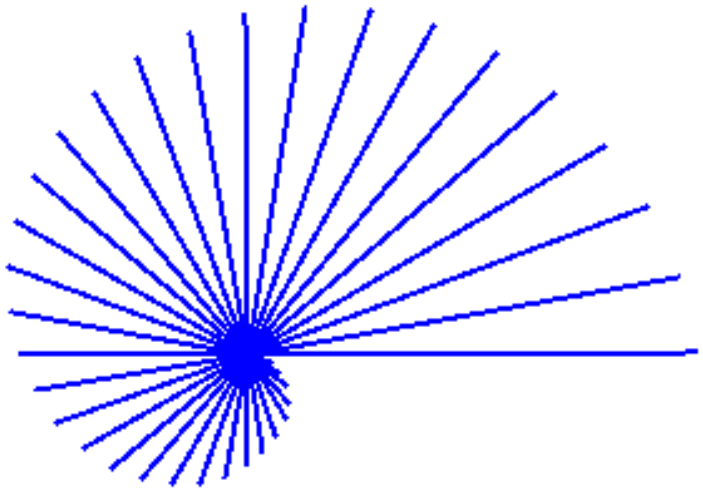
видеоинформация — передаваемая в виде видеозаписи.

1.1.2. Понятие исполнителя

Исполнитель — это человек или автомат (в частности, им может быть процессор ЭВМ), умеющий выполнять некоторый, вполне определенный конечный набор действий. Приказ на выполнение действия из указанного набора, выраженный каким-либо заранее оговоренным способом, называется предписанием, а вся совокупность допустимых приказов — системой предписаний исполнителя.

Давая задание исполнителю на выполнение некоторой работы, мы обычно выдаем ему не одно предписание, а некоторую конечную последовательность предписаний, задавая также порядок, в котором эти предписания должны быть выполнены. Такая последовательность предписаний с указанием порядка их выполнения называется программой.

```
1  использовать Рисователь
2  алг
3  нач
4  ▪ цел i, x, y, r
5  ▪ вещ n
6  ▪ перо(2, "синий")
7  ▪ нц для i от 0 до 360 шаг 10
8  ▪ ▪ n:=i*3.14/180
9  ▪ ▪ r:=int(i*0.5)
10 ▪ ▪ x:=int(r*cos(n))
11 ▪ ▪ y:=int(r*sin(n))
12 ▪ ▪ линия(400, 150, x+400, y+150)
13 ▪ кц
14 кон
```



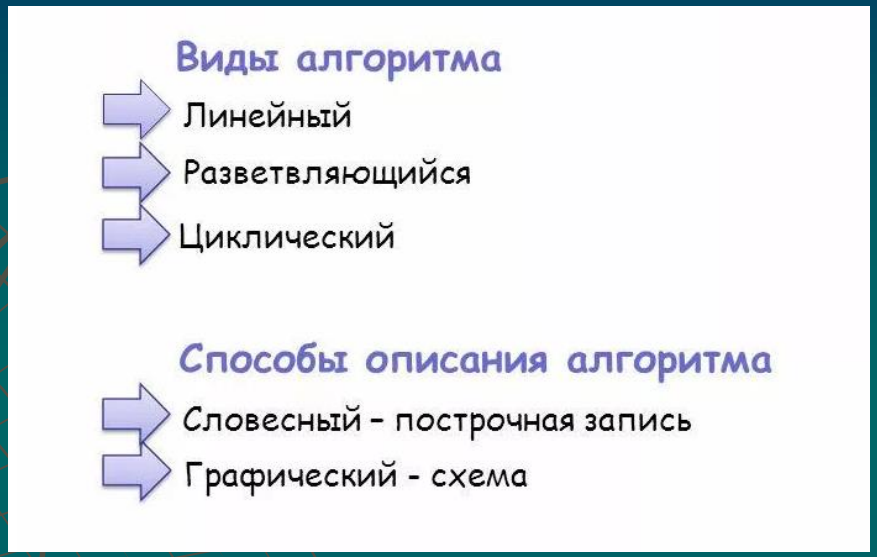
1.1.3. Программирование

Программирование в своей самой исходной форме является порождением команд и наблюдением, что они выполняются. Это смесь математики и лингвистики: определяется множество вычислений и нужно приказать машинам выполнить их, задавая порядок с помощью правильной грамматики. Грамматика в программировании является синтаксисом, т.е. способом соединения слов в словосочетания и предложения, и существенно отличается в различных языках.

```
'время года
TextWindow.Write("Напиши название месяца с маленькой буквы: ")
m = TextWindow.Read()
If m = "декабрь" Or m = "январь" Or m = "февраль" Then
  v = "ЗИМА"
Else
  If m = "март" Or m = "апрель" Or m = "май" Then
    v = "ВЕСНА"
  Else
    If m = "июнь" Or m = "июль" Or m = "август" Then
      v = "ЛЕТО"
    Else
      If m = "сентябрь" Or m = "октябрь" Or m = "ноябрь" Then
        v = "ОСЕНЬ"
      Else
        v = "НЕ СУЩЕСТВУЕТ"
      EndIf
    EndIf
  EndIf
EndIf
TextWindow.WriteLine("Время года - " + v)
```

Точный набор инструкций, которые необходимо выполнить исполнителю для решения поставленной задачи за конечное число шагов, называется алгоритмом (от имени ученого Аль-Хорезми).

Элементарное предписание, предусматривающее выполнение какого-либо действия, называется командой, а сами действия, предписываемые командой, называются операторами.



Данные, их описание и алгоритм, записанный на языке программирования, называется программой, а процесс создания программ – программированием. При этом, язык программирования – это формализованный язык для написания программ. Все языки программирования являются искусственными, в них синтаксис и семантика строго определены.

Классификация программ

Прикладное программное обеспечение (ПО) –программы для конечных пользователей (не программистов). Например, бухгалтерские программы, системы управления предприятиями, игры, обучающие системы и т.д.

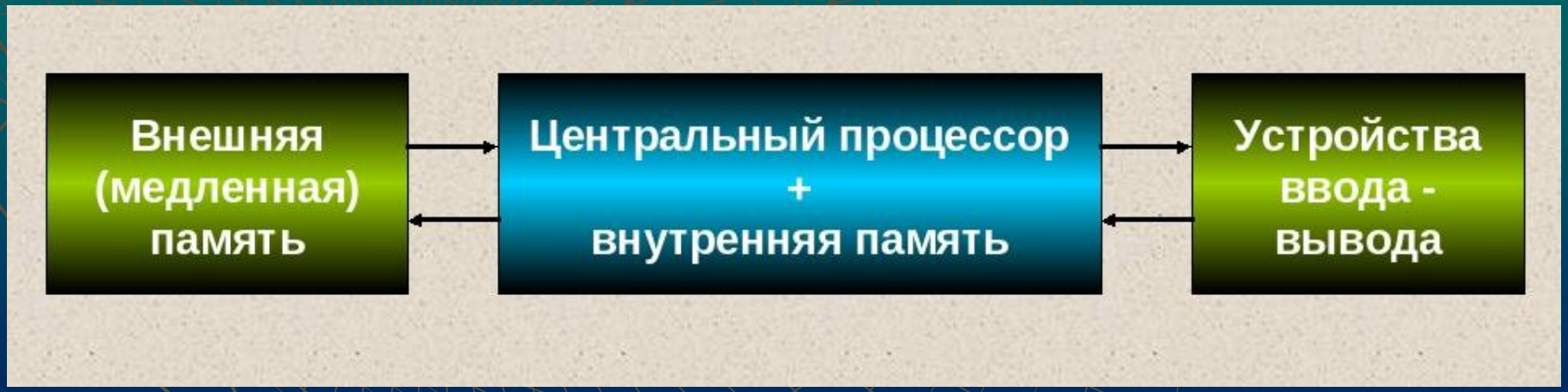
Системное ПО – программы для программистов. К примеру, операционные системы (ОС) и офисные пакеты, системы управления базами данных (СУБД) и т.д.

Научно-исследовательское ПО – прототипы прикладного, системного ПО для проверки новых алгоритмов и методов обработки данных.

1.1.4. Принципы фон Неймана

В 1946 году Д. фон Нейман, Г. Голдстайн и А. Беркс в своей совместной статье изложили новые принципы построения и функционирования ЭВМ. В последствие на основе этих принципов производились первые два поколения компьютеров. В более поздних поколениях происходили некоторые изменения, хотя принципы Неймана актуальны и сегодня.

По сути, Нейману удалось обобщить научные разработки и открытия многих других ученых и сформулировать на их основе принципиально новое.



Принципы фон Неймана

Принцип двоичного кодирования. Для представления данных и команд используется двоичная система счисления.

Принцип однородности памяти. Как программы (команды), так и данные хранятся в одной и той же памяти (и кодируются в одной и той же системе счисления — чаще всего двоичной). Над командами можно выполнять такие же действия, как и над данными.

Принцип адресуемости памяти. Структурно основная память состоит из пронумерованных ячеек; процессору в произвольный момент времени доступна любая ячейка.

Возможность условного перехода в процессе выполнения программы. Не смотря на то, что команды выполняются последовательно, в программах можно реализовать возможность перехода к любому участку кода.

Самым главным следствием этих принципов можно назвать то, что теперь программа уже не была постоянной частью машины (как например, у калькулятора). Программу стало возможно легко изменить.

1.2 Понятие алгоритма.

Понятие алгоритма является одним из основных понятий современной математики. Еще на самых ранних ступенях развития математики (Древний Египет, Вавилон, Греция) в ней стали возникать различные вычислительные процессы чисто механического характера. С их помощью искомые величины ряда задач вычислялись последовательно из исходных величин по определенным правилам и инструкциям. Со временем все такие процессы в математике получили название алгоритмов (алгорифмов).

Аль – Хорезми
Мухаммед Бен – Мусса

Термин ***алгоритм*** происходит от имени средневекового узбекского математика Аль-Хорезми, который еще в IX в. (825 г.) дал правила выполнения четырех арифметических действий в десятичной системе счисления. Процесс выполнения арифметических действий был назван ***алгоризмом***.

С 1747 г. Вместо слова *алгоризм* стали употреблять *алгорисмус*, смысл которого состоял в комбинировании четырех операций арифметического исчисления – сложения, вычитания, умножения, деления.

К 1950 г. *алгорисмус* стал *алгорифмом*. Смысл алгорифма чаще всего связывался с алгорифмами Евклида – процессами нахождения наибольшего общего делителя двух натуральных чисел, наибольшей общей меры двух отрезков и т.п.

Алгоритмом называется система формальных правил, четко и однозначно определяющая процесс выполнения заданной работы в виде конечной последовательности действий или операций.

Алгоритм – это понятные и точные предписания исполнителю совершить конечное число шагов, направленных на решение поставленной задачи.

Исполнителем алгоритма может быть человек, группа людей, робот, станок, компьютер, язык программирования и т.д. Важнейшим свойством, характеризующим любого из этих исполнителей, является то, что исполнитель умеет выполнять некоторые команды.

Алгоритм должен быть составлен таким образом, чтобы исполнитель, в расчете на которого он создается, мог ***однозначно*** и ***точно*** следовать командам алгоритма и эффективно получать определенный результат. Это накладывает на записи алгоритмов целый ряд обязательных требований.

1.3 Основные свойства алгоритмов.

1. Дискретность

Описываемый процесс должен быть разбит на последовательность отдельных шагов. Возникающая в результате такого разбиения запись представляет собой упорядоченную совокупность четко разделенных друг от друга предписаний (директив, команд, операторов), образующих прерывную (дискретную) структуру алгоритма. Только выполнив требования одного предписания, можно приступить к выполнению следующего.

2. Понятность

Используемые на практике алгоритмы составляются с ориентацией на конкретного исполнителя. Поэтому, составляя алгоритм, можно использовать лишь те команды, которые входят в систему команд исполнителя.

3. Определенность (детерминированность)

Будучи понятным, алгоритм не должен содержать предписаний, смысл которых может восприниматься неоднозначно. Это означает, что алгоритм выдает один и тот же результат для одних и тех же данных.

4. Результативность (конечность)

Обязательное получение некоторого искомого результата либо сигнала о том, что данный алгоритм неприменим для решения поставленной задачи;

5. Массовость

Алгоритм должен быть применим не к одной исключительной задаче, а некоторому классу задач данного типа.

1.4 Способы (формы) представления алгоритмов

– **содержательная** (текстуальная форма);

– **графическая форма** (в виде блок-схемы);

– представление алгоритмов на **языках программирования**.

Содержательная форма представления алгоритма:

Пример.

1. Если *a=b*, то работа алгоритма закончена, иначе выполняется пункт 2.

2. Если a>b, то переменной a присваивается значение *a-b*, иначе переменной *b* присваивается значение *b-a*.

3. Выполняется пункт 1 данного алгоритма.

Пусть *a=56*, *b=21*, тогда в результате работы данного алгоритма переменная *a* примет значение …

a=7

Пример. Составить алгоритм в текстуальной форме для расчета факториала натурального числа *n*.

Пусть *fakt=n!*, тогда

1. Полагаем *fakt=1* и переходим к пункту 2.

2. Полагаем *i =1* и переходим к пункту 3.

3. Полагаем *fakt=fakt·i* и переходим к пункту 4.

4. Проверяем, равно ли *i* числу *n*. Если *i=n*, то вычисления прекращаем. Если *i<n*, то увеличиваем *i* на единицу и переходим к пункту 3.

Пусть *n=4*, тогда в результате работы данного алгоритма значение факториала будет равно …

fakt=24

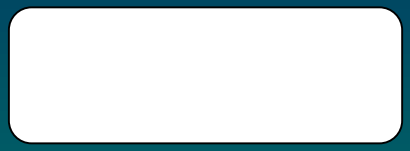
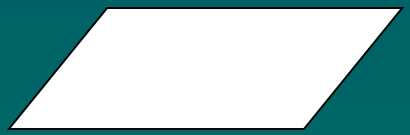
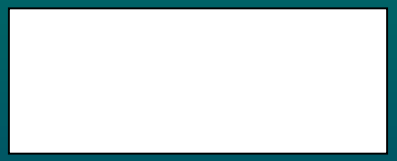
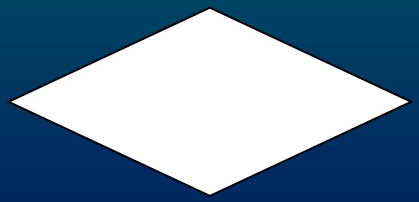
Запись алгоритмов в виде блок-схем

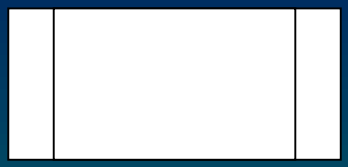
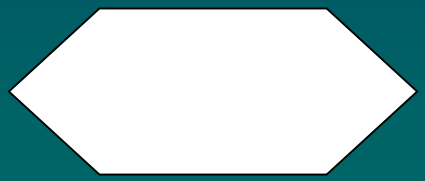
Схема алгоритма – графическое представление алгоритма. Каждый пункт алгоритма отображается на схеме некоторой геометрической фигурой – ***блоком*** – и дополняется элементами словесной записи.

Блоки на схемах соединяются ***линиями потоков информации***. Основное направление потока информации идет сверху вниз и слева направо (стрелки могут не указываться), снизу вверх и справа налево – стрелка обязательна.

Количество входящих линий для блока не ограничено. Выходящая линия должна быть одна (исключение составляет логический блок).

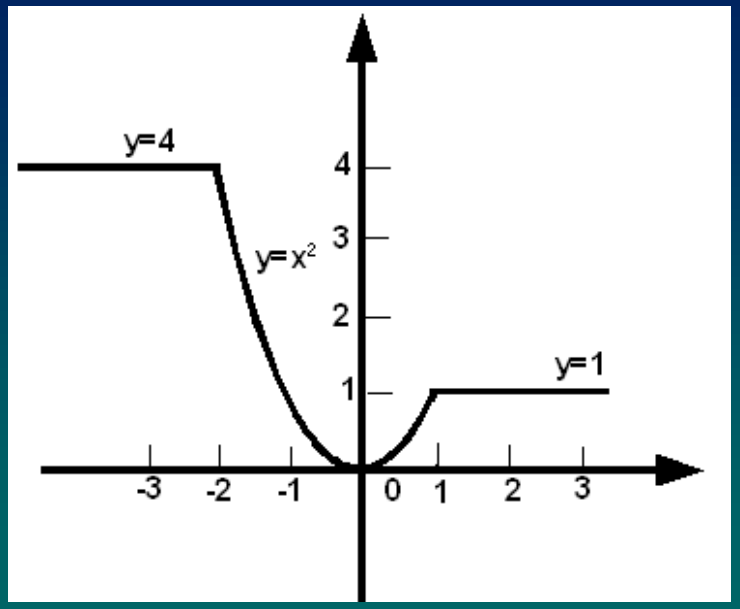
Основные элементы блок-схем

Фигура	Наименование	Содержание
	Начало (конец)	Начало или конец алгоритма, вход или выход в программу
	Блок ввода-вывода данных	Общие обозначения ввода (вывода) данных
	Блок вычислений	Вычислительные действия или последовательность действий
	Логический блок	Выбор направления выполнения алгоритма в зависимости от некоторого условия

Фигура	Наименование	Содержание
	Процесс пользователя (подпрограмма)	Обращение к программе или подпрограмме
	Блок модификации	Заголовок цикла с параметром
	Соединитель	Указание связи прерванными линиями между потоками информации в пределах одного листа
	Межстраничные соединения	Указание связи между информацией на разных листах

Пример. Найти значение функции, заданной аналитически:

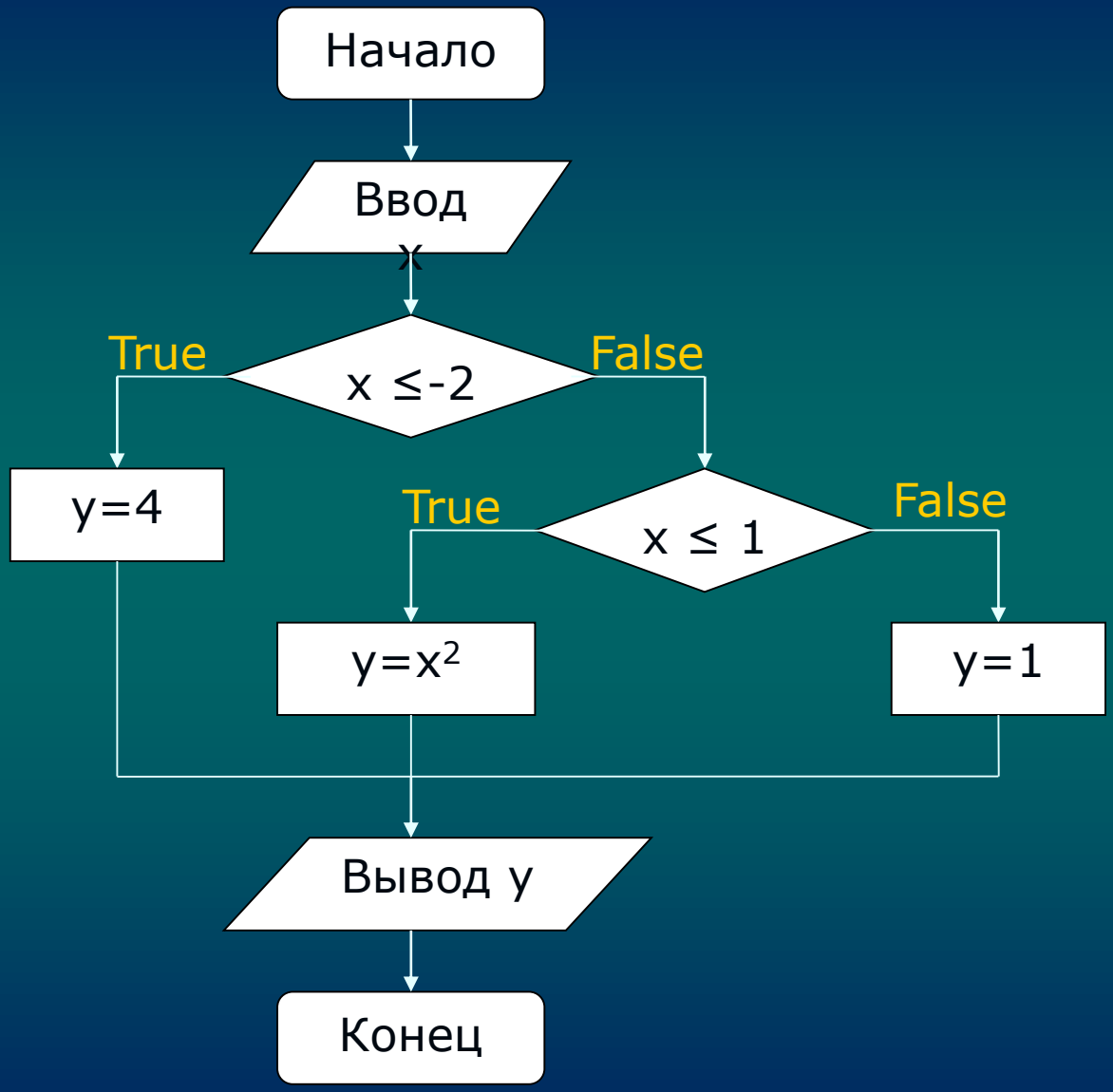
$$y(x)=\begin{cases} y=4, & x\le -2; \\ y=x^2, & -2<x\le 1; \\ y=1, & x>1. \end{cases}$$



Составим словесный алгоритм решения этой задачи:

1. Начало алгоритма.
2. Ввод числа ***x*** (аргумент функции).
3. Если значение ***x*** ≤ -2, то переход к п.4, иначе переход к п.5.
4. Вычисление значения функции: y=4, переход к п.8.
5. Если значение ***x*** ≤ 1, то переход к п.6, иначе переход к п.7.
6. Вычисление значения функции: y=1, переход к п.8.
7. Вычисление значения функции: $y=x^2$.
8. Вывод значений аргумента x и функции y.
9. Конец алгоритма.

Запишем этот алгоритм в виде блок-схемы:



1.5 Базовые структуры алгоритмов

Базовые структуры алгоритмов – это определенный набор блоков и стандартных способов их соединения для выполнения типичных последовательностей действий.

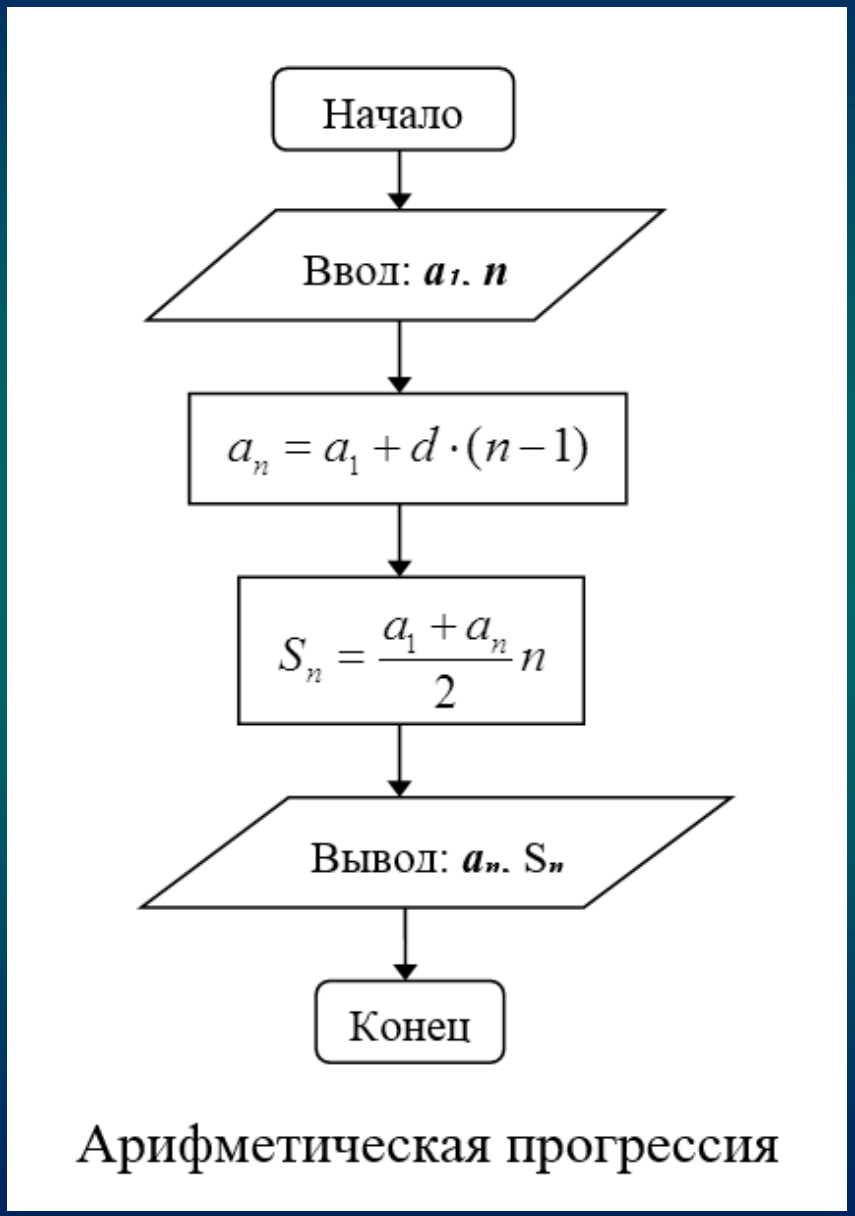
К основным структурам относятся следующие:

1. Линейные
2. Разветвляющиеся
3. Циклические

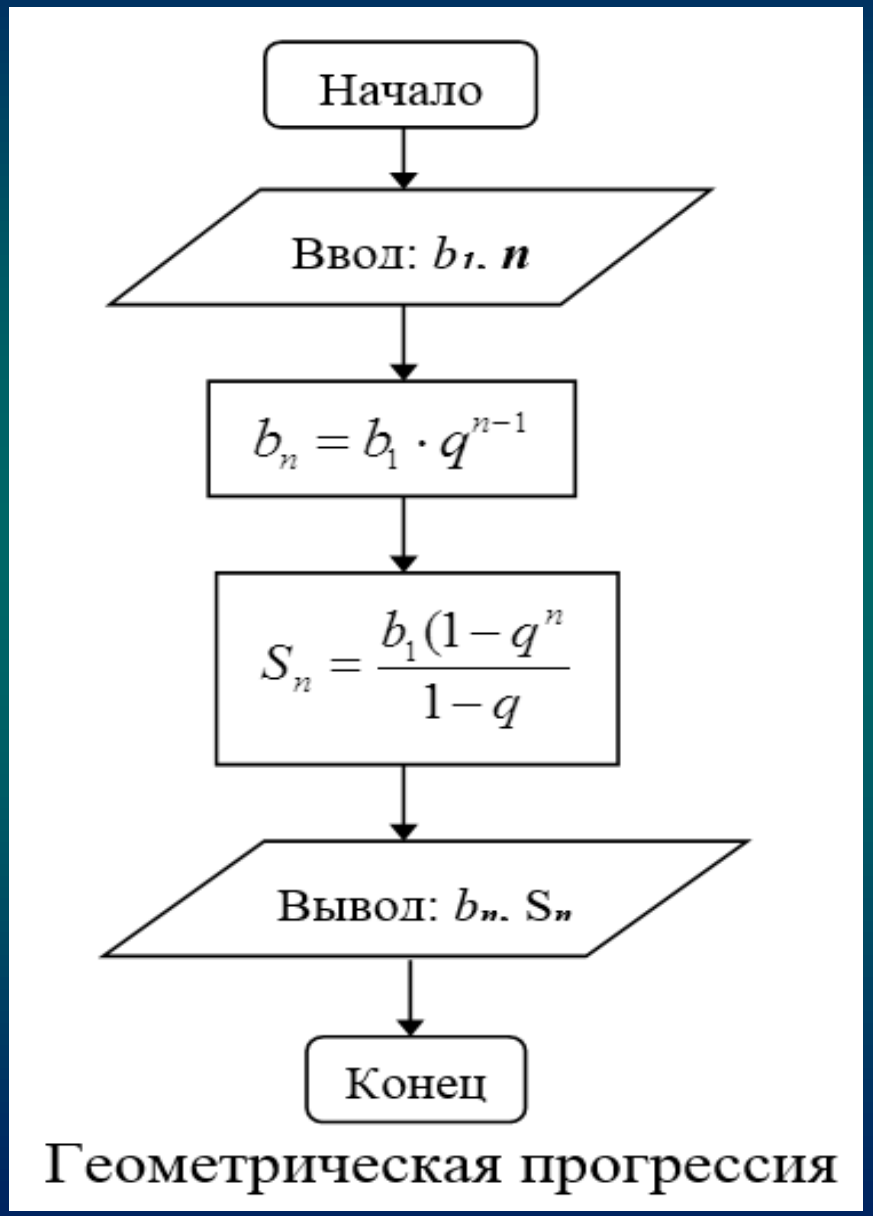
1.5.1 Линейная структура

Линейными называются алгоритмы, в которых действия осуществляются последовательно друг за другом.

Пример. Вычислить конечные суммы и значения членов арифметической и геометрической прогрессий.

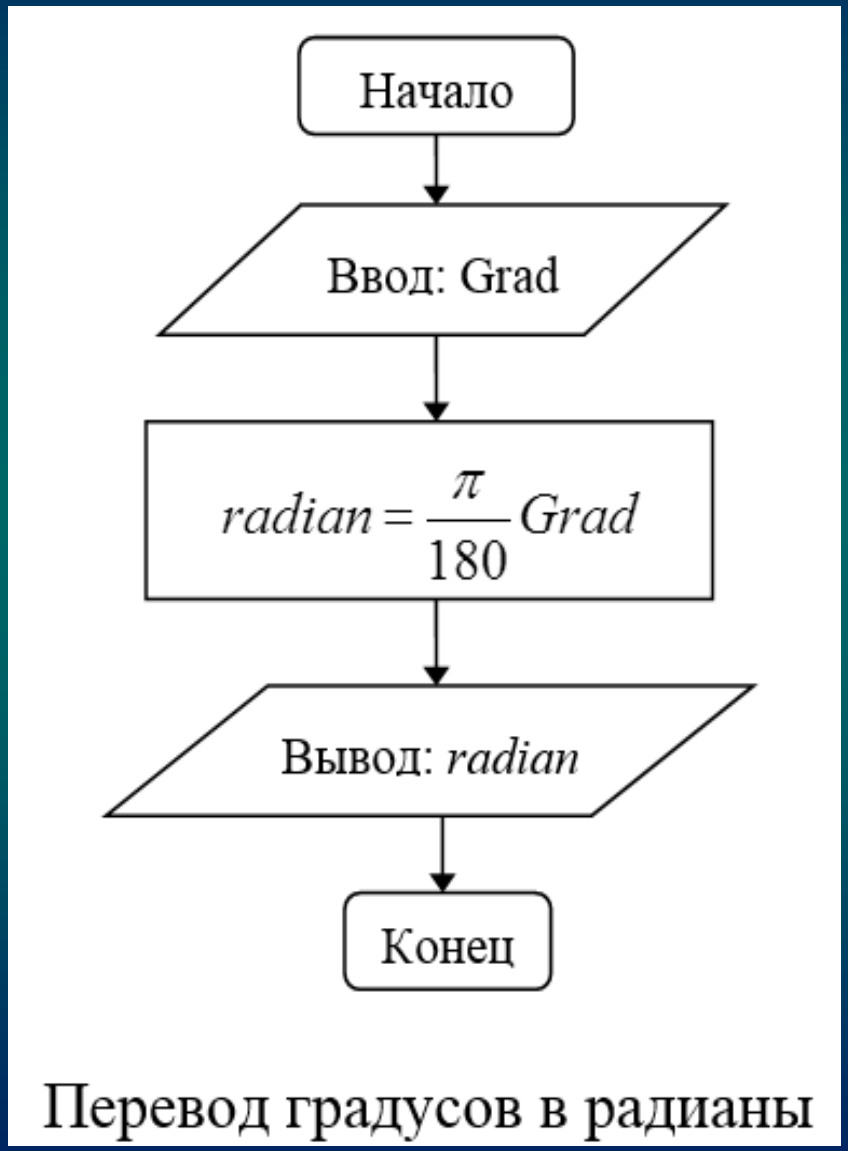


Арифметическая прогрессия

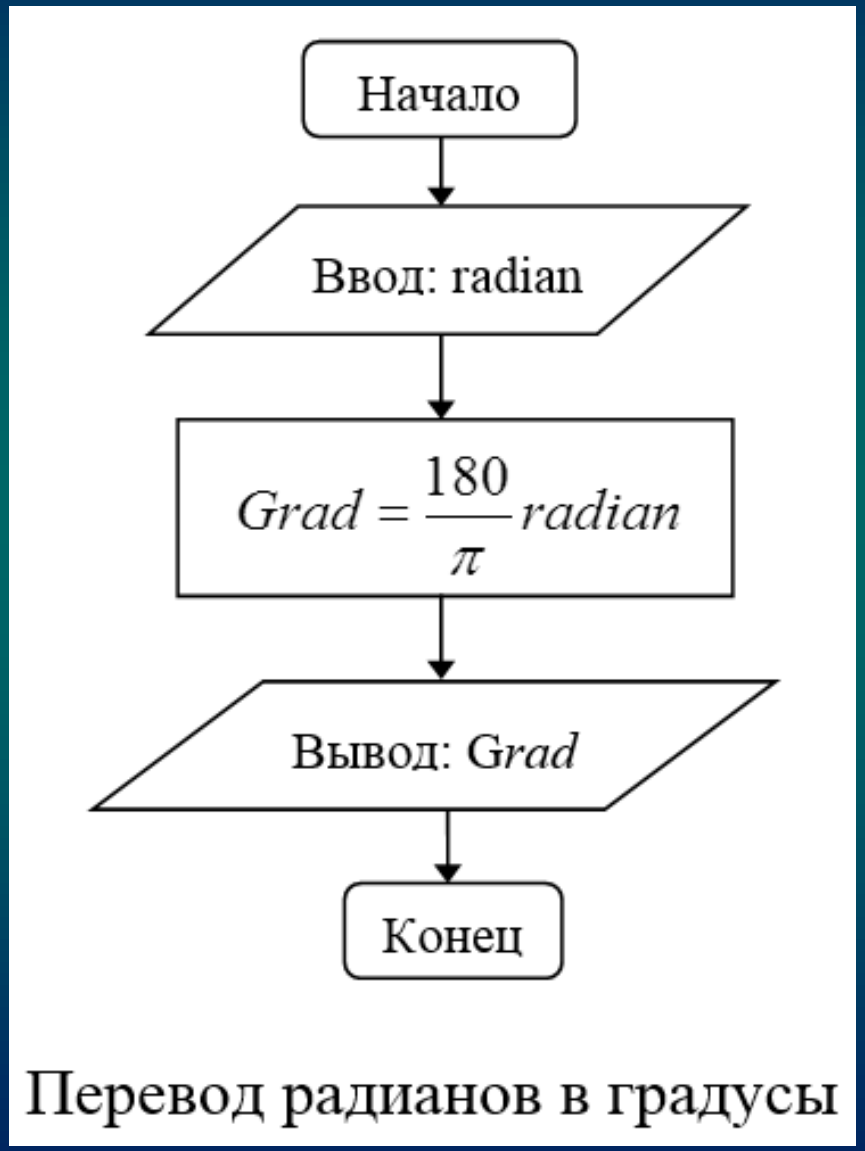


Геометрическая прогрессия

Пример. Перевести угловые значения из градусов в радианы, из радианов в градусы.

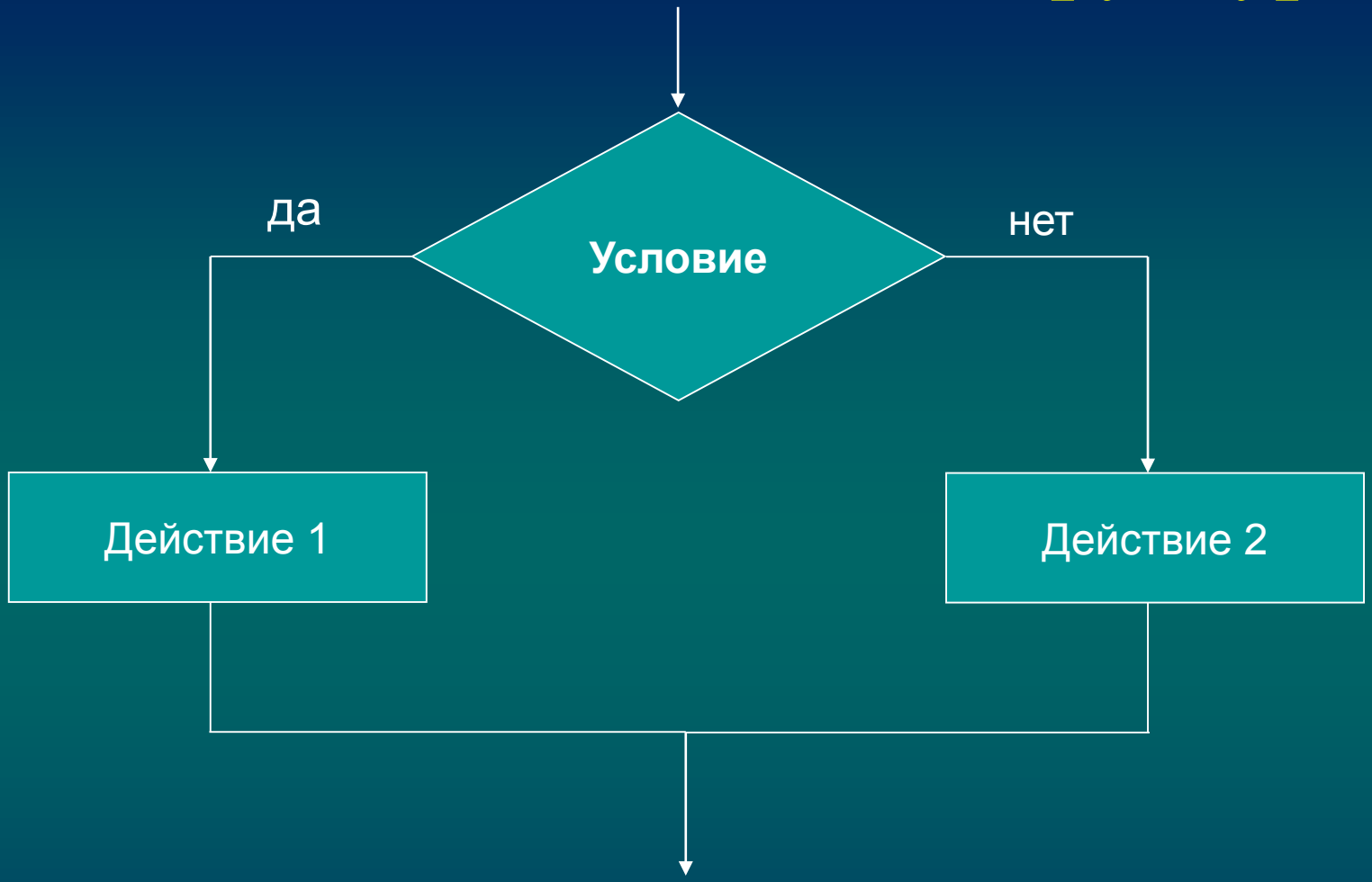


Перевод градусов в радианы



Перевод радианов в градусы

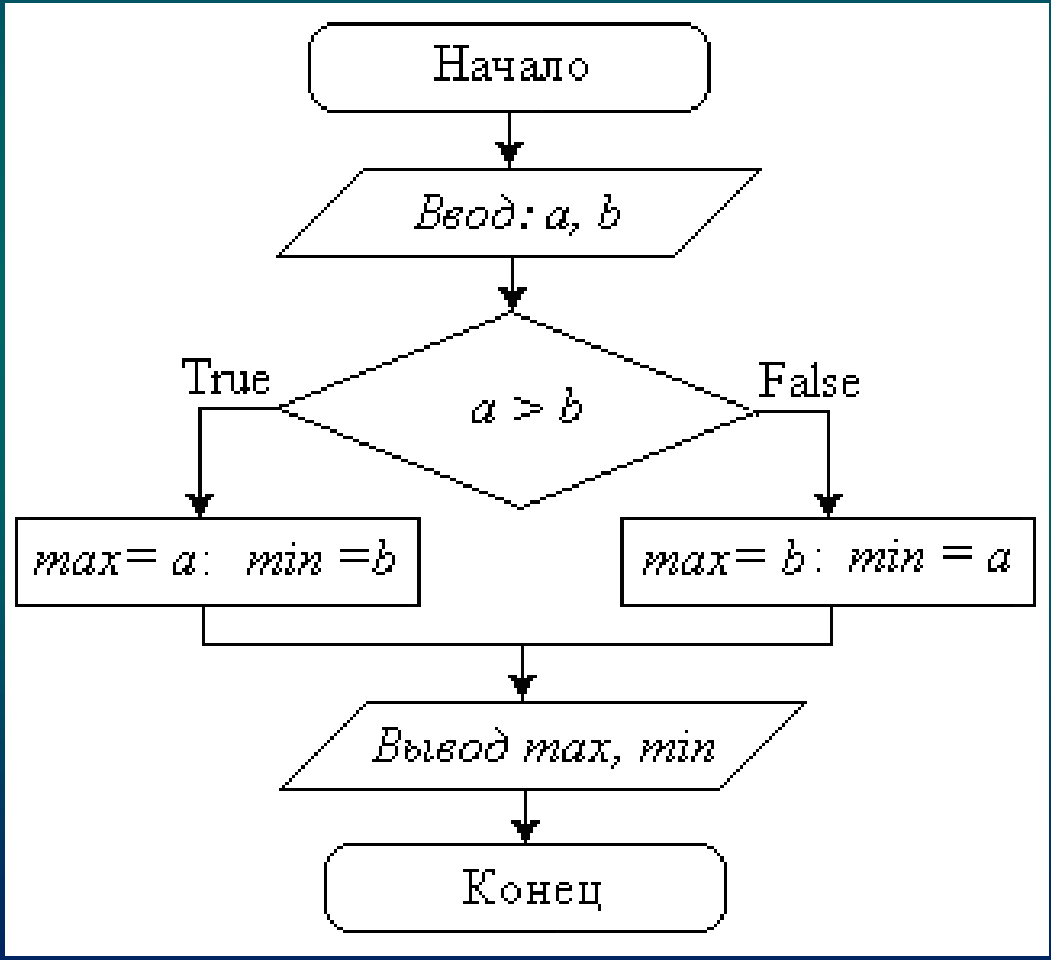
1.5.2 Разветвляющиеся структуры



Под ***ветвлением*** будем понимать алгоритмическую структуру, в которой в зависимости от результата проверки поставленного условия осуществляется выбор только одной из двух или нескольких альтернативных ветвей алгоритма.

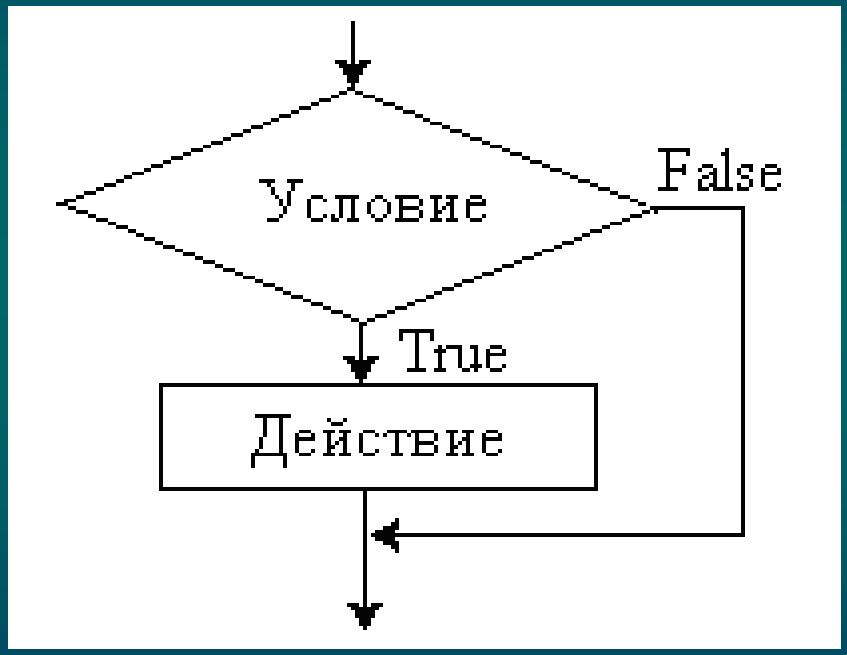
Для сравнения переменных в условных выражениях используют *операции отношения*: =, <, >, <=, >=, <>.

Пример. Для нахождения максимального и минимального значений из двух введенных пользователем чисел можно использовать представленный на алгоритм.



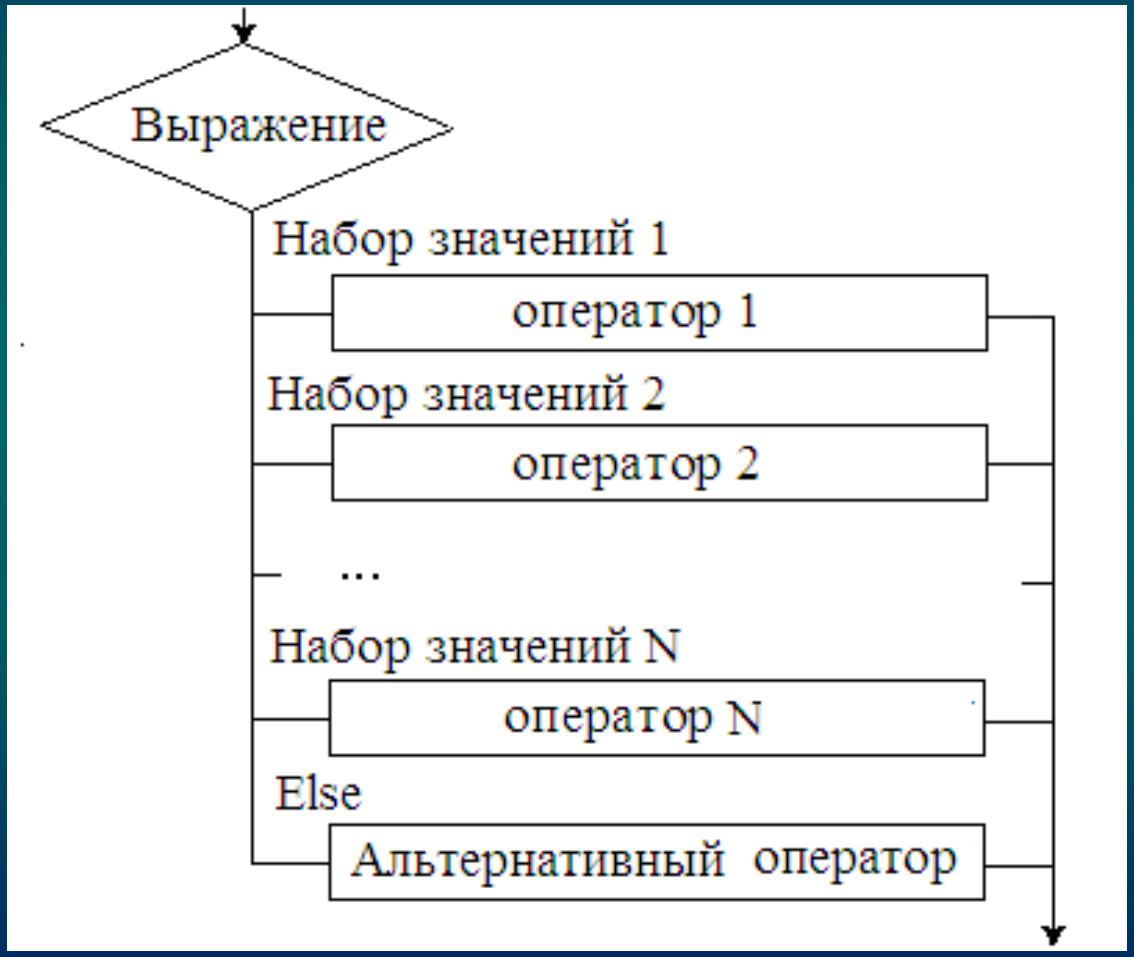
Обход

Альтернативная ветвь ***Else*** в структуре ***ветвление*** может отсутствовать, если в ней нет необходимости. Обычно такую алгоритмическую структуру называют ***обходом***.

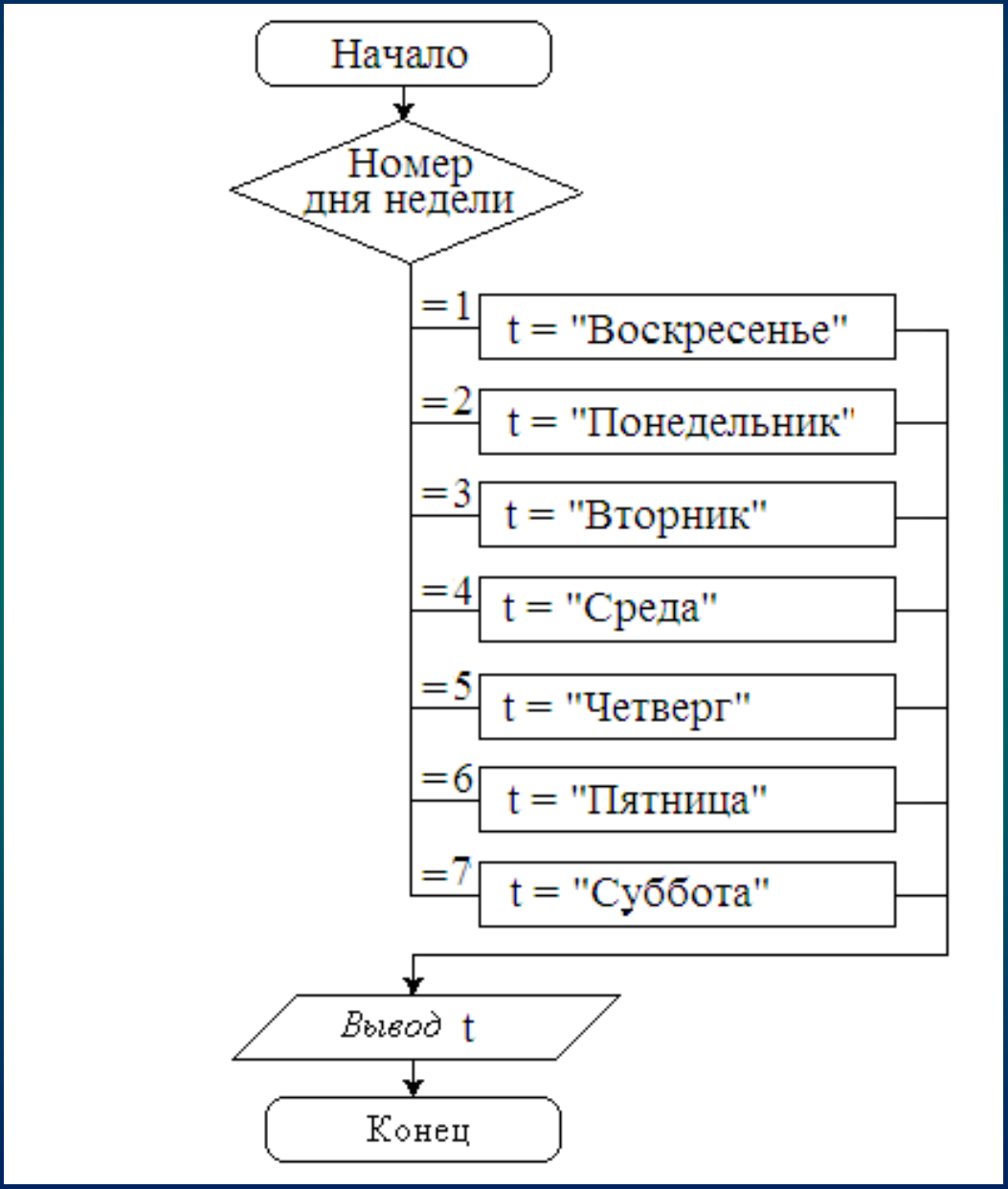


Множественный выбор

В случае если структура имеет не две, а три или более ветвей, то ее называют *множественным выбором* и описывают следующей блок-схемой.



Пример. Вывести на экран название дня недели, соответствующее текущей дате. Данный алгоритм можно реализовать следующим образом:



Пример. Составить алгоритм нахождения действительных корней квадратного уравнения $ax^2 + bx + c = 0$.

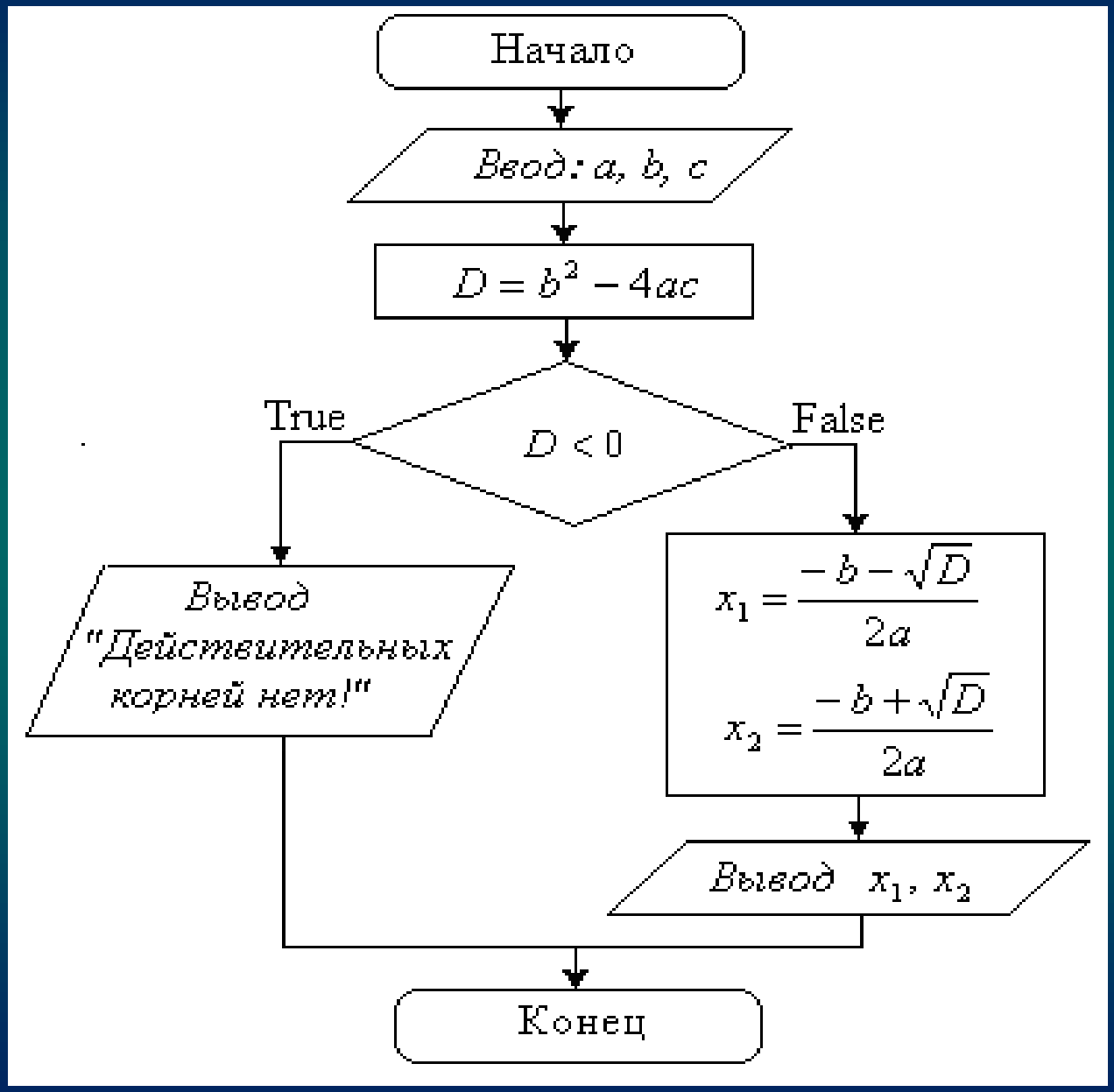
Для решения задачи по формуле $D = b^2 - 4ac$ найдем дискриминант уравнения. Если полученное значение $D<0$, то это означает, что действительных корней нет, в противном случае корни уравнения определяются по формулам:

$$x_1 = \frac{-b - \sqrt{D}}{2a} \qquad x_2 = \frac{-b + \sqrt{D}}{2a}$$

Составим словесный алгоритм решения этой задачи.

1. Начало алгоритма.
2. Ввод числовых значений переменных a, b и c.
3. Вычисление значения дискриминанта d по формуле $d = b^2 - 4ac$.
4. Если $d < 0$, то переход к п.5, иначе переход к п.6.
5. Вывод сообщения «Вещественных корней нет» и переход к п.8.
6. Вычисление корней x_1 и x_2 по выше указанным формулам.
7. Вывод значений x_1 и x_2.
8. Конец алгоритма.

Приведем структурную блок-схему решения данной задачи.



1.5.3 Циклические структуры

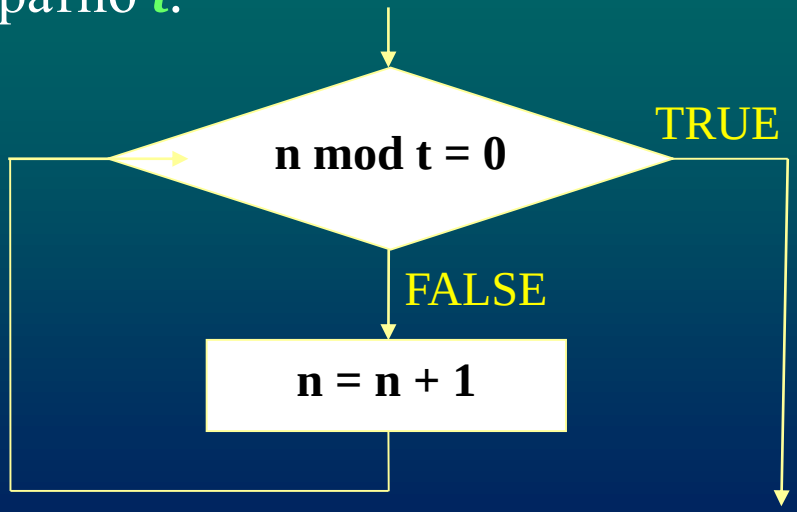
Циклический процесс или ***цикл*** – это повторение одних и тех же действий и операций в ходе выполнения программы.

Последовательность действий, которые повторяются в цикле, называют ***телом цикла***.

Один проход цикла называют ***итерацией***.

Переменные, которые изменяются внутри цикла и влияют на его окончание, называются ***параметрами цикла***.

Пример. Данная алгоритмическая структура используется для увеличения значения натурального числа ***n***, до тех пор, пока ***n*** не будет кратно ***t***.

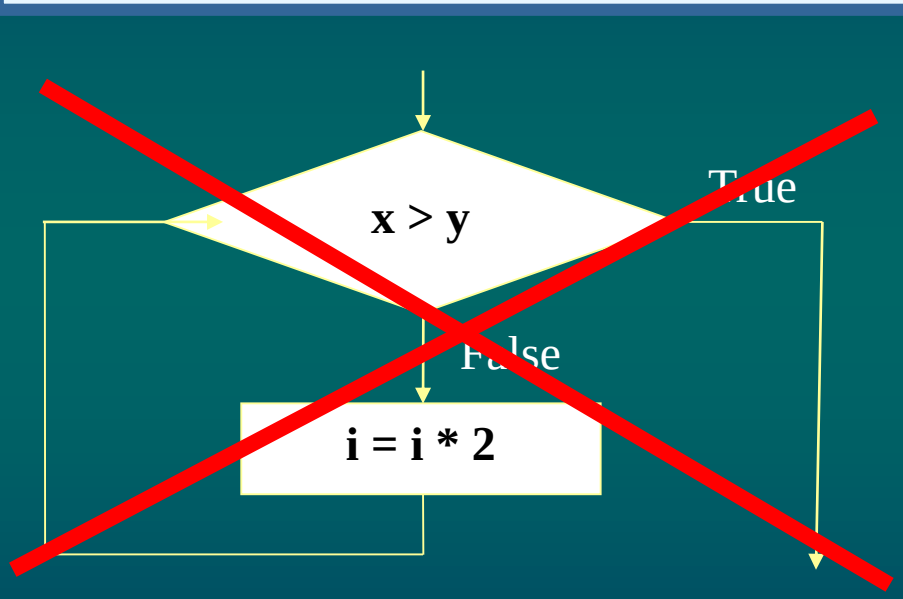


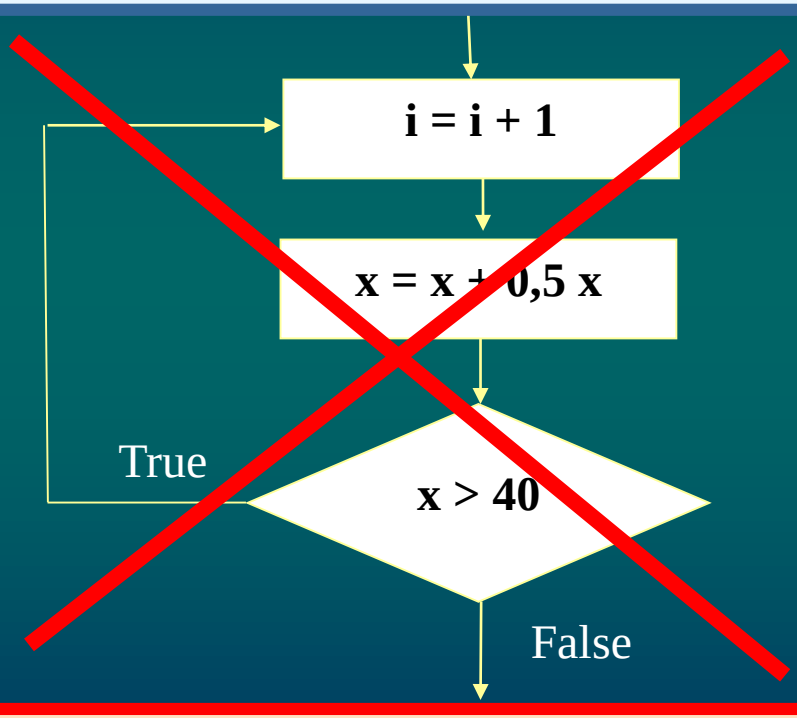
При начальном значении параметра цикла ***n***=58, а ***t***=7 количество итераций этого цикла равно 5.

При написании циклических алгоритмов необходимо придерживаться следующих правил:

1) Чтобы цикл мог когда-нибудь закончиться, **содержимое тела цикла должно обязательно влиять на условие цикла**.

2) Условие должно состоять из **корректных выражений и значений, определенных еще до первого выполнения тела цикла**.





В первом алгоритме вычисления в теле цикла не влияют на условие цикла.

Второй цикл никогда не закончится при $x<2$. (Сумма членов геометрической прогрессии со знаменателем $q<1$).

Циклическим называется алгоритм, в котором некоторая часть операций (тело цикла – последовательность команд) выполняется многократно.

Многократно – не значит до бесконечности! Организация циклов, никогда не приводящая к остановке в выполнении алгоритма, является нарушением требования его результативности – получения результата за конечное число шагов.

Перед операцией цикла осуществляются операции ***присвоения начальных значений*** тем объектам, которые используются в теле цикла.

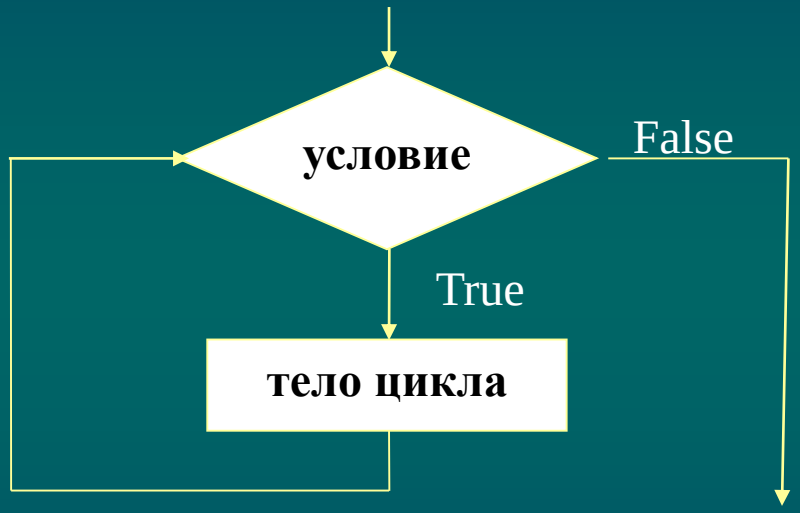
В цикл входят в качестве базовых следующие структуры: ***блок проверки условия*** и блок, называемый ***телом цикла***.

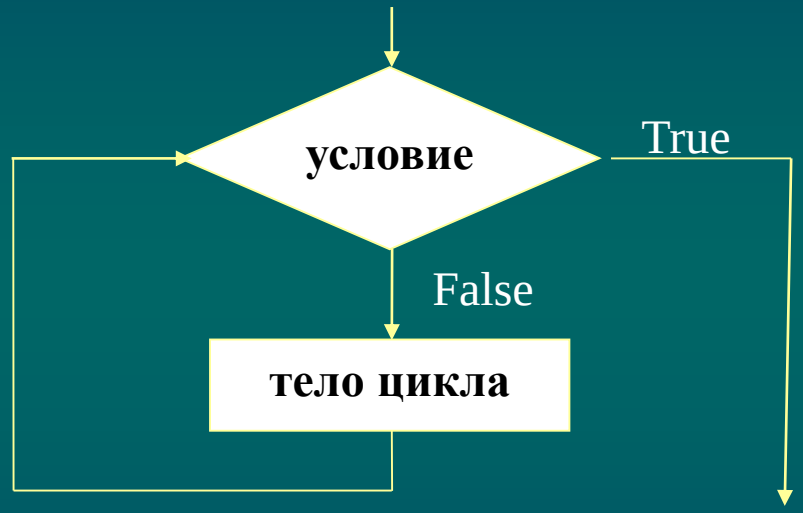
К основным циклическим структурам относятся следующие:

- *циклы с предусловием* (циклы «*пока*»)
- *циклы с постусловием* (циклы «*до*»)
- *циклы с параметром* (циклы «*для*»)

Циклы с предусловием

Циклом с предусловием называется цикл, в котором истинность условия проверяется каждый раз ***перед*** выполнением операторов тела цикла.



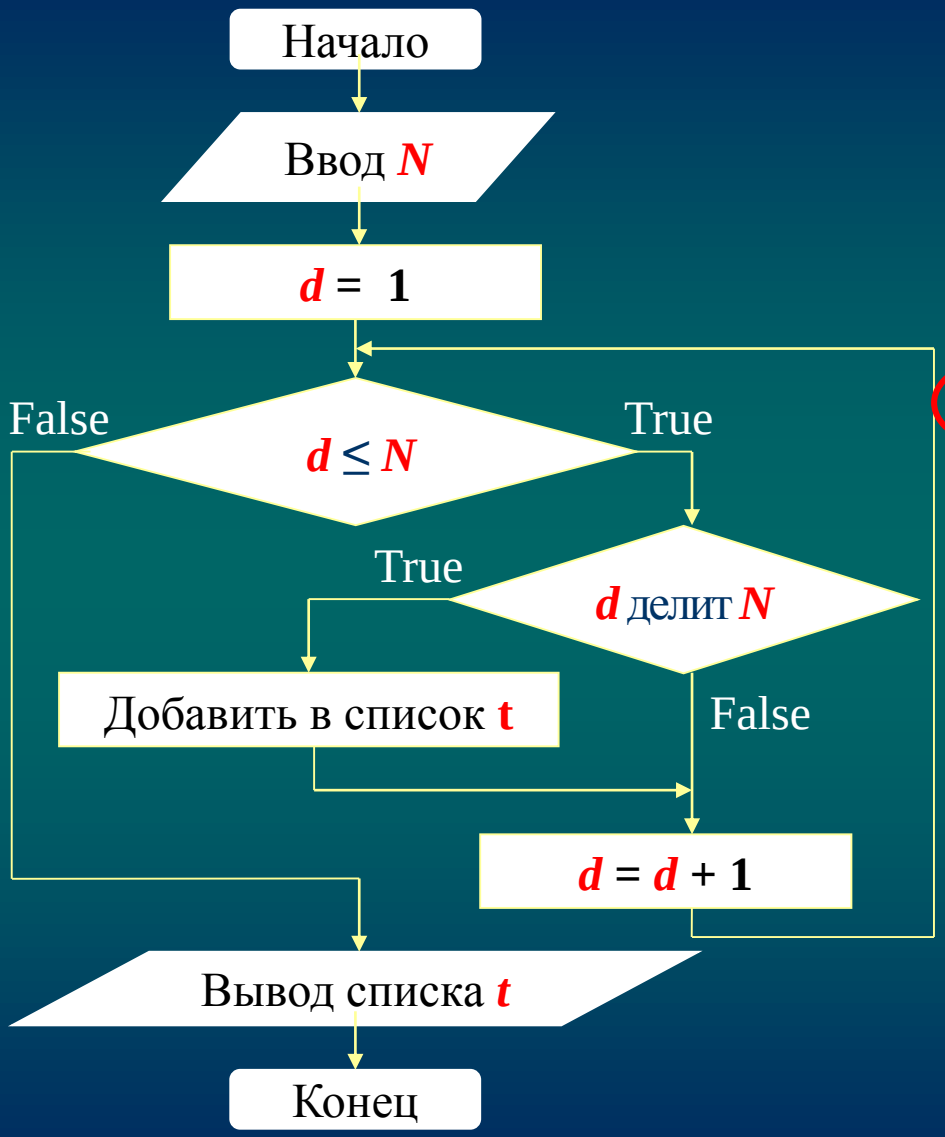


Цикл с предусловием работает следующим образом:

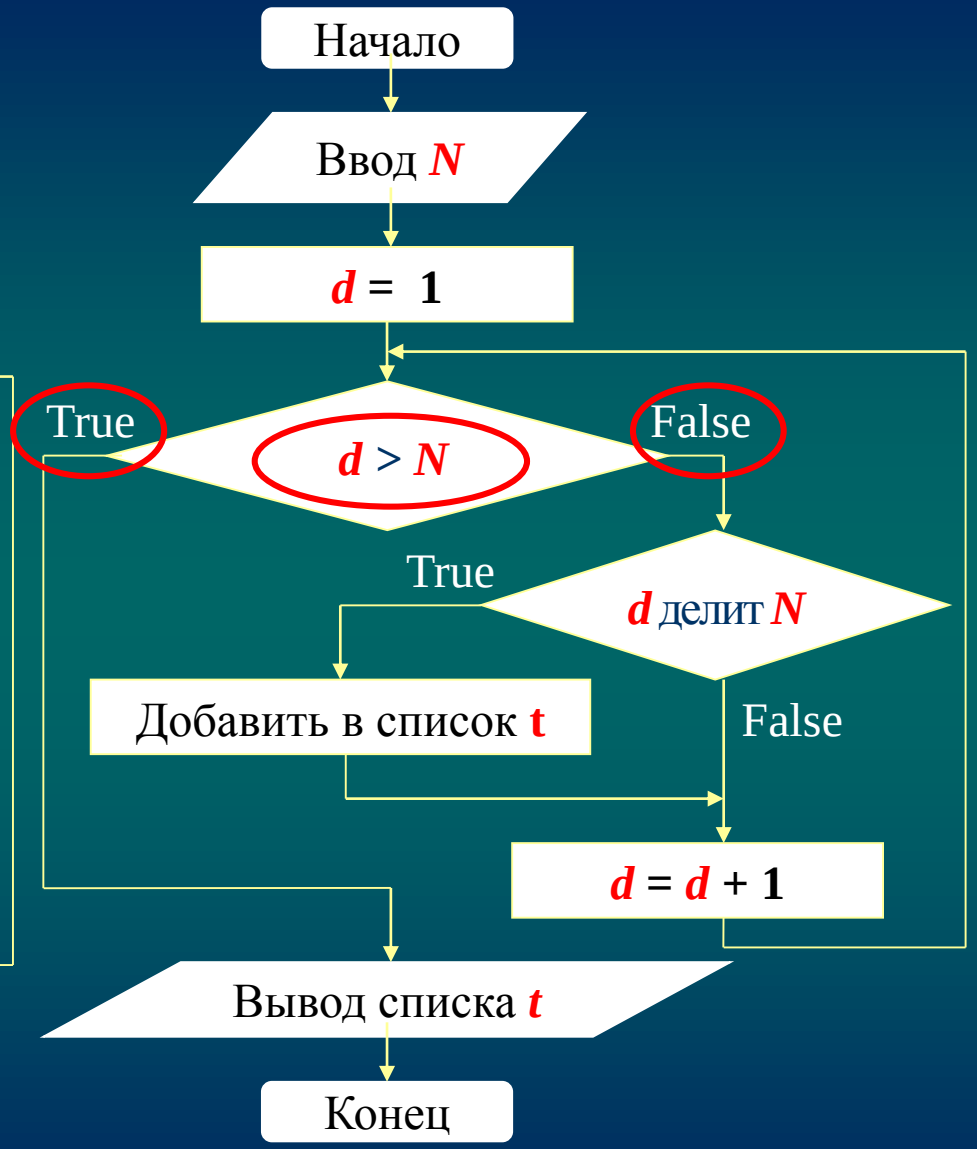
1. Проверяется условие. Если оно ***истинно*** (***ложно***), то осуществляется переход на выполнение тела цикла. В противном случае цикл заканчивается, и происходит выход из цикла.

2. Выполняется тело цикла и программа снова переходит на проверку условия.

Пример. Используя циклы с предусловием, найдем все делители натурального числа N:



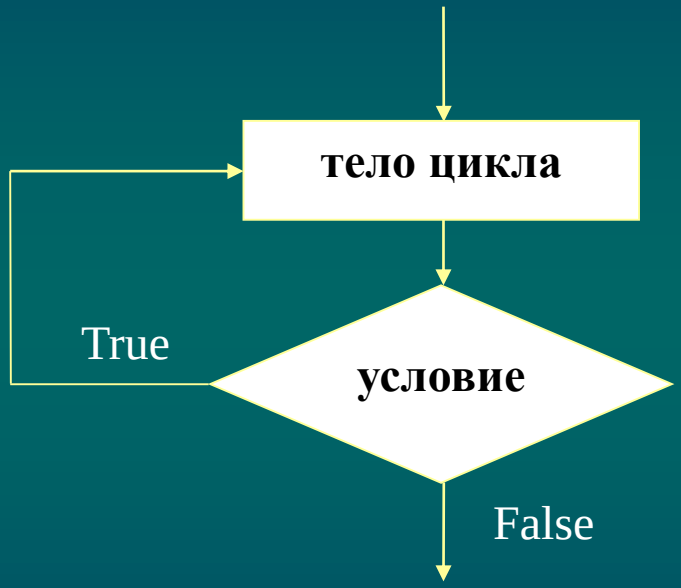
Цикл выполняется в случае истинного условия

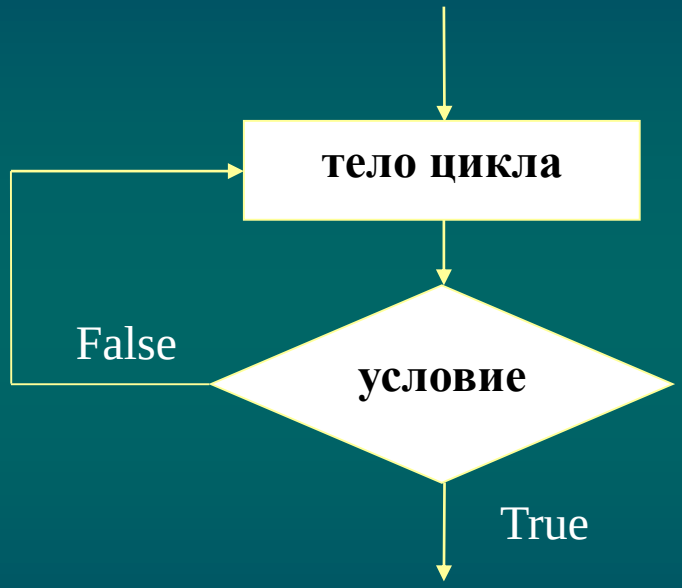


Цикл выполняется в случае ложного условия

Циклы с постусловием

Циклом с постусловием называется цикл, в котором истинность условия проверяется каждый раз ***после*** выполнения операторов тела цикла.

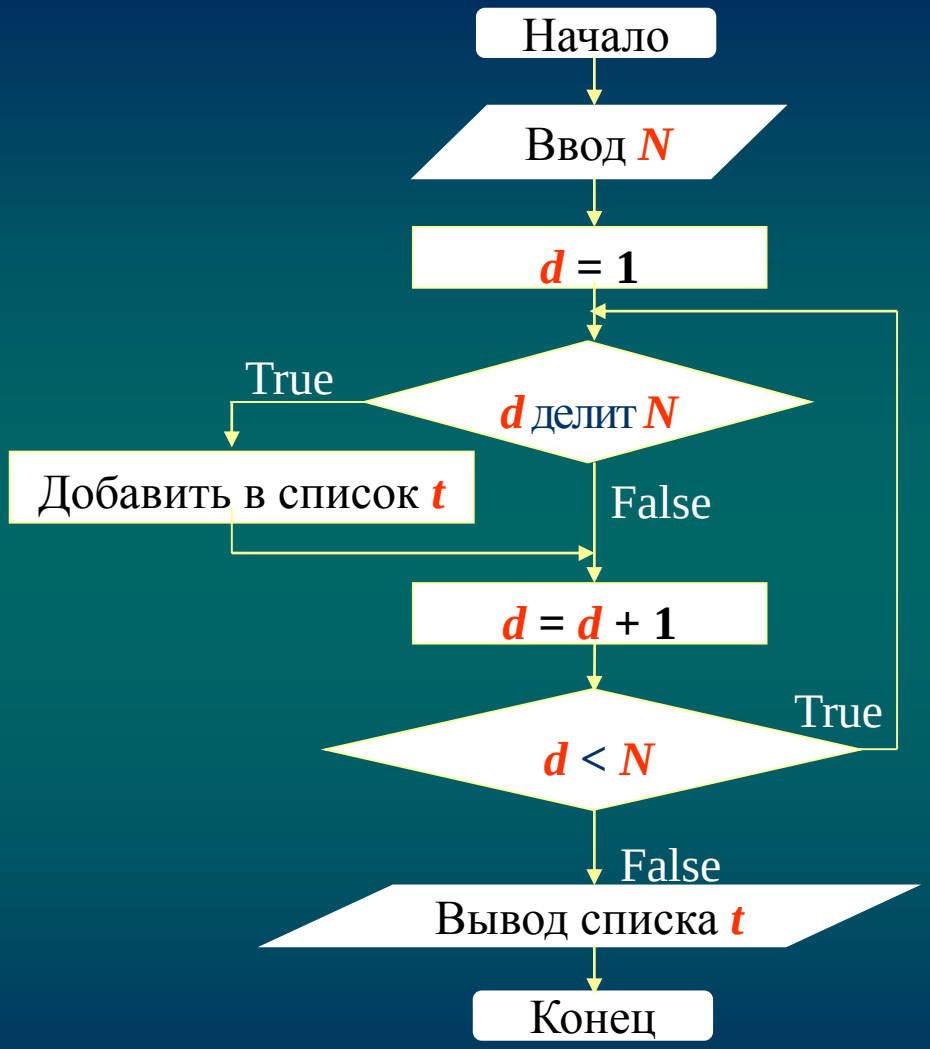




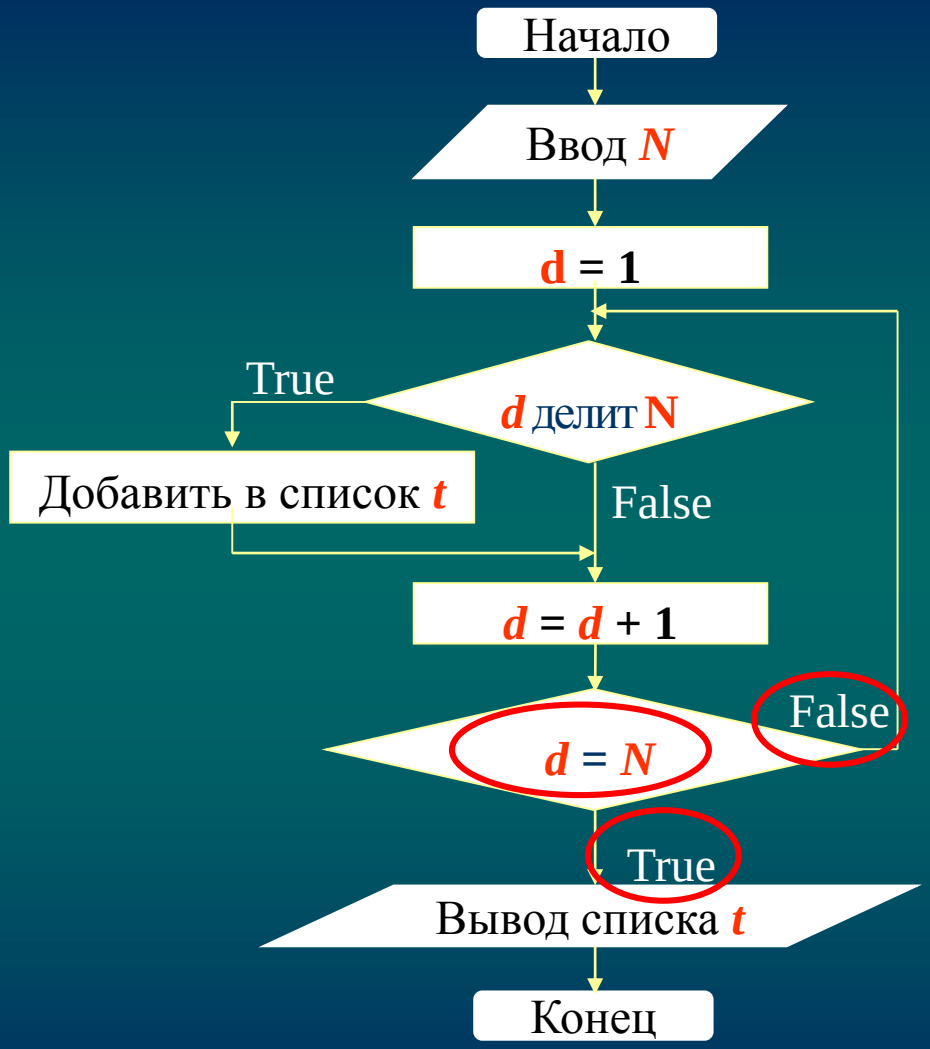
В структуре «***Цикл с постусловием***» условие проверяется ***после*** выполнения тела цикла. Если условие ***истинно*** (***ложно***), то программа снова обращается к началу тела цикла, иначе происходит выход из цикла.

Цикл с постусловием всегда будет выполнен хотя бы один раз!

Пример. Решим задачу нахождения всех делителей натурального числа N используя циклы с постусловием.



Цикл выполняется в случае истинного условия

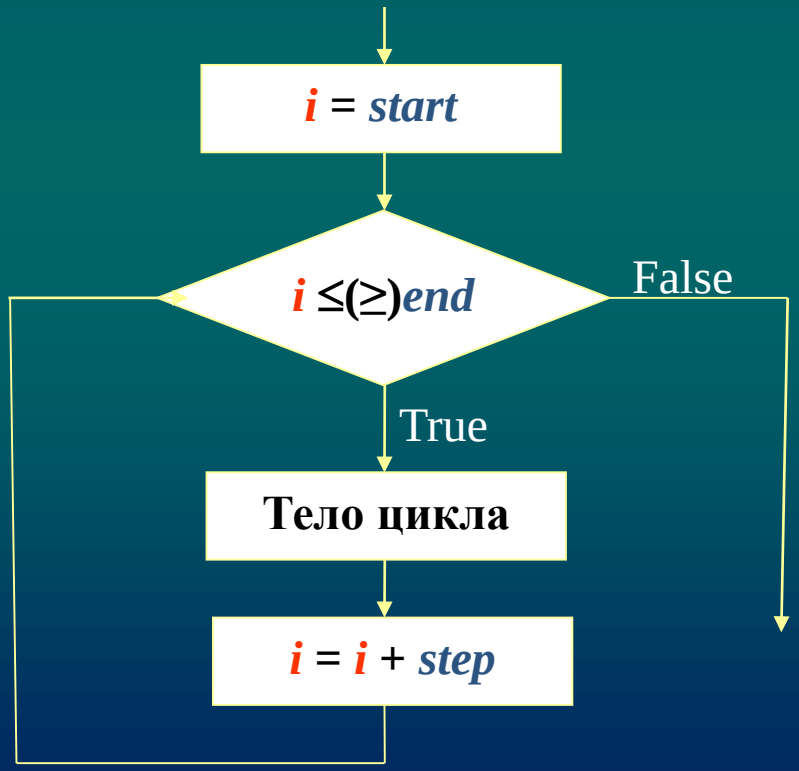


Цикл выполняется в случае ложного условия

Циклы с параметром

Циклы с предусловием и постусловием обладают значительной гибкостью, но не удобны для организации «***строгих***» циклов, которые должны быть выполнены ***фиксированное число раз***.

Циклы с параметром (***циклы с управляющей переменной***) используются именно в таких случаях.



Алгоритм работы цикла с параметром.

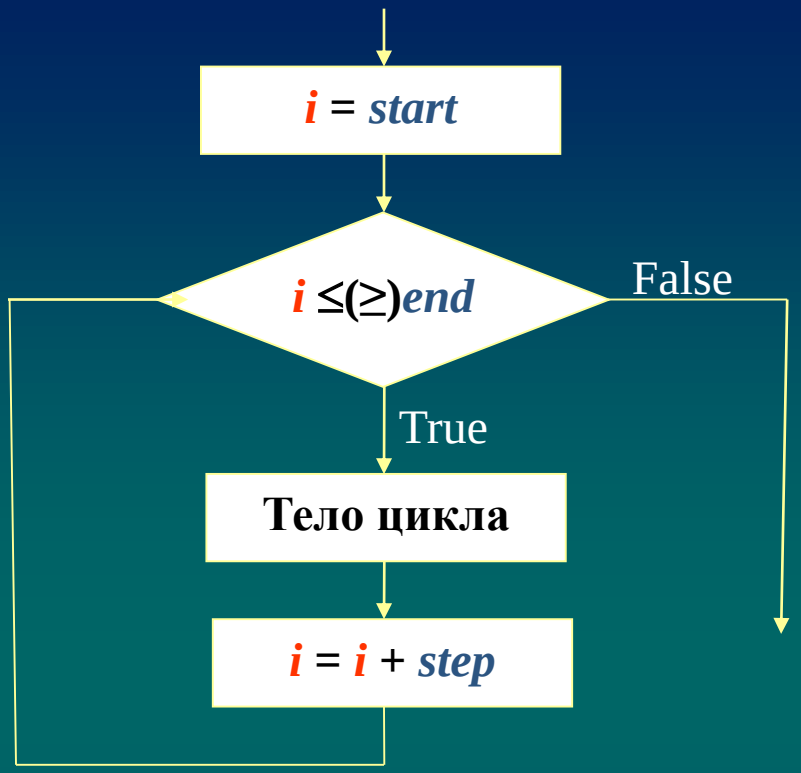
Параметру цикла ***i*** присваивается начальное значение ***start***.

2. Если ***i*** > (<) конечного значения ***end***, то цикл завершает свою работу.

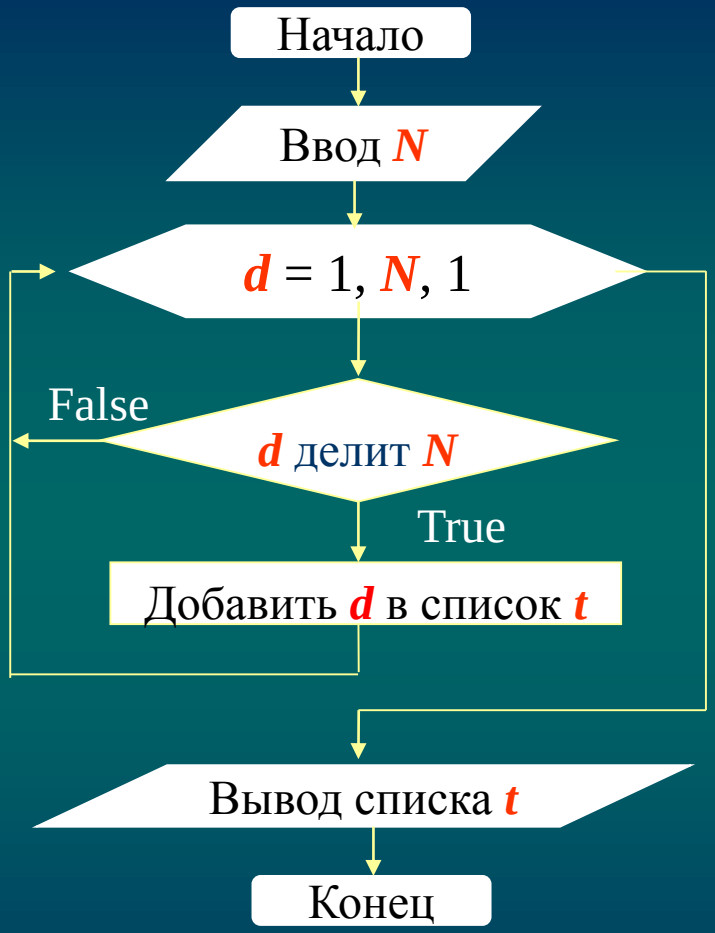
3. Выполняется тело цикла.

4. Значение ***i*** изменяется на шаг ***step*** и осуществляется переход к ***п.2***.

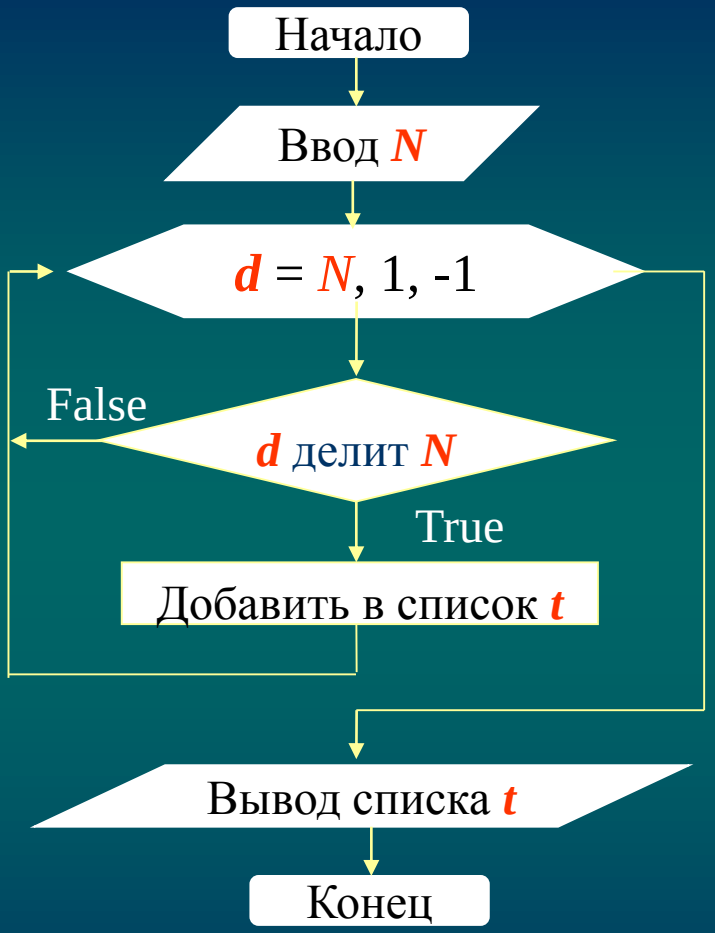
При шаге ***step > 0*** параметр ***i*** с каждым проходом ***возрастает***, а при ***step < 0*** параметр ***i убывает***.



Пример. Построим алгоритм нахождения делителей N используя циклы с возрастающим и убывающим параметрами.



Цикл с возрастающим параметром



Цикл с убывающим параметром

1.6 Решение типовых задач

1.6.1 Алгоритм Евклида

Пример. Найти наибольший общий делитель чисел A и B.

Для решения задачи воспользуемся алгоритмом Евклида: будем уменьшать каждый раз большее из чисел на величину меньшего до тех пор, пока оба значения не станут равными.

Используя алгоритм Евклида, найдем наибольший общий делитель чисел 64 и 40.

A	**64**	**64 – 40 = 24**	**24**	**24 – 16 = 8**	**8**
B	**40**	**40**	**40 – 24 =16**	**16**	**16 – 8 = 8**

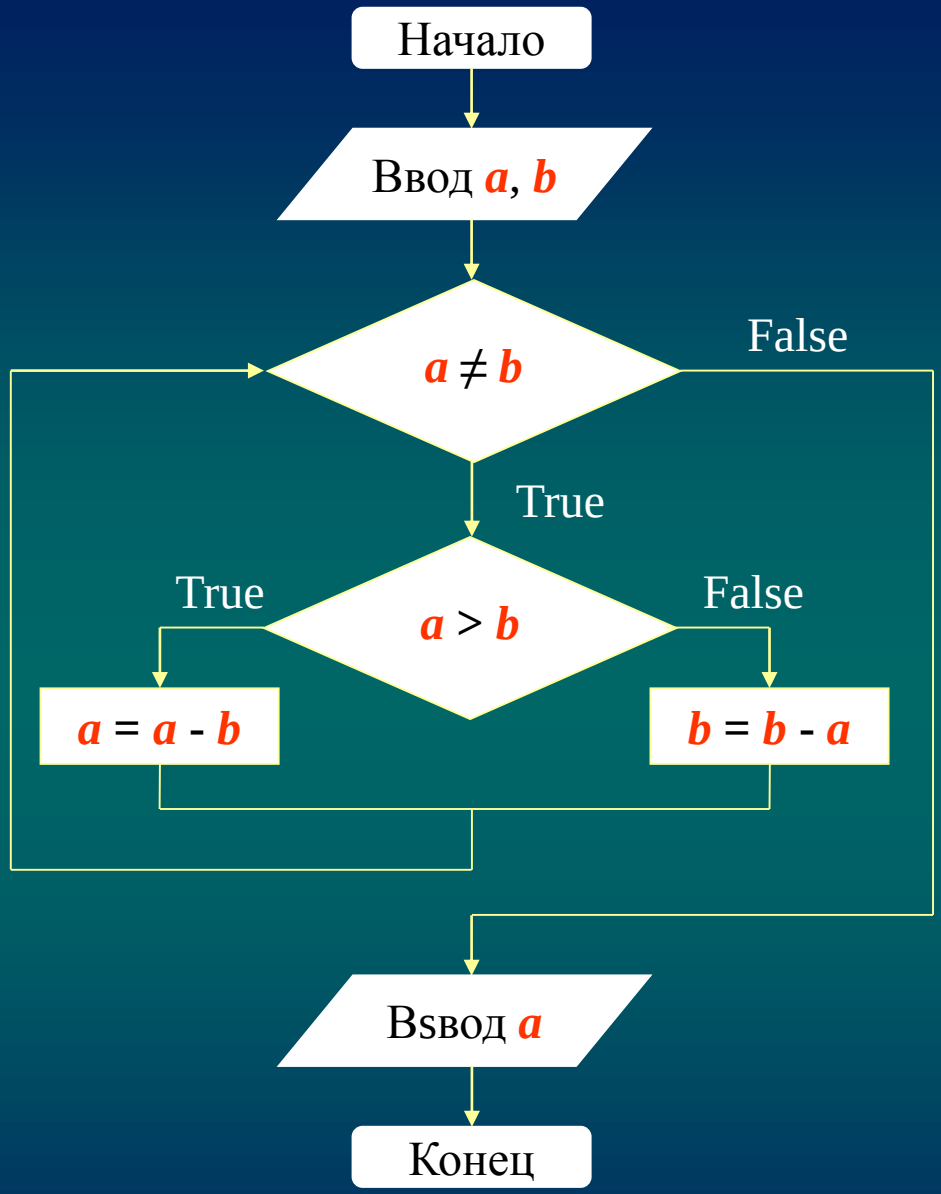


Схема нахождения НОД по алгоритму Евклида

В блок–схеме решения задачи будем использовать цикл с предусловием.

Тогда тело цикла, выполняющее уменьшение чисел A = A – B либо B = B - A будет повторяется до тех пор, пока A не равно B.

1.6.2 Вычисление факториала

Пример. Вычислить факториал числа **N**

(N!=1 · 2 · 3 · … ·N)

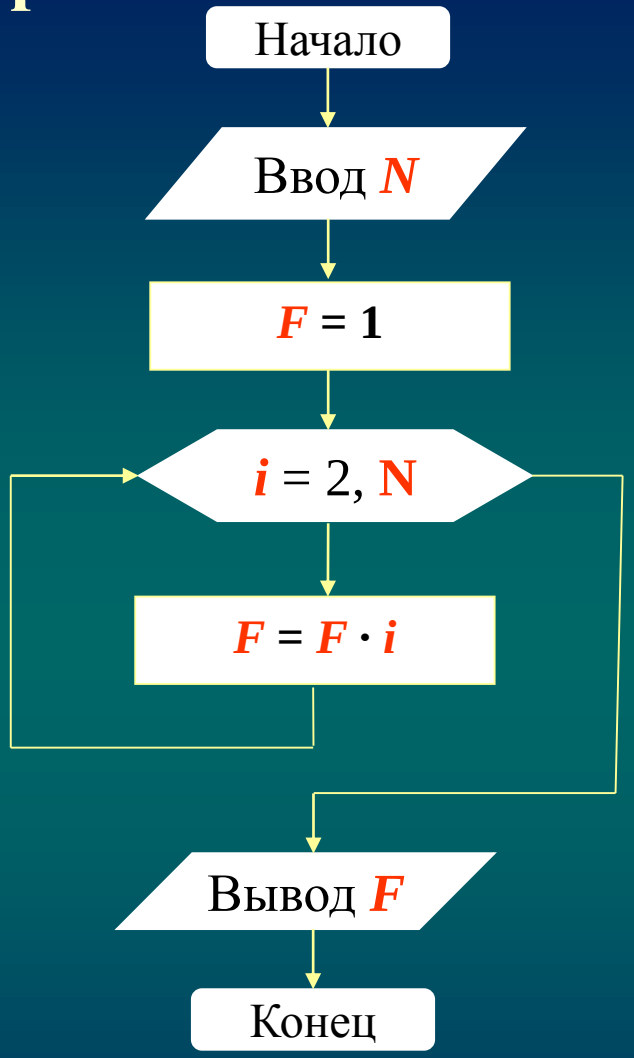
Вводится число **N**.

Переменной ***F*** присваивается начальное значение, равное единице.

Затем организуется цикл, параметром которого выступает ***i***.

Если ***i*** ≤ ***N***, то выполняется оператор тела цикла, в котором значение ***F*** умножается на текущее значение параметра цикла ***i***, а результат присваивается ***F***.

Когда параметр ***i*** становится больше ***N***, цикл заканчивается, и выводится значение переменой ***F***.



Алгоритм нахождения факториала

1.6.3 Подсчет цифр числа

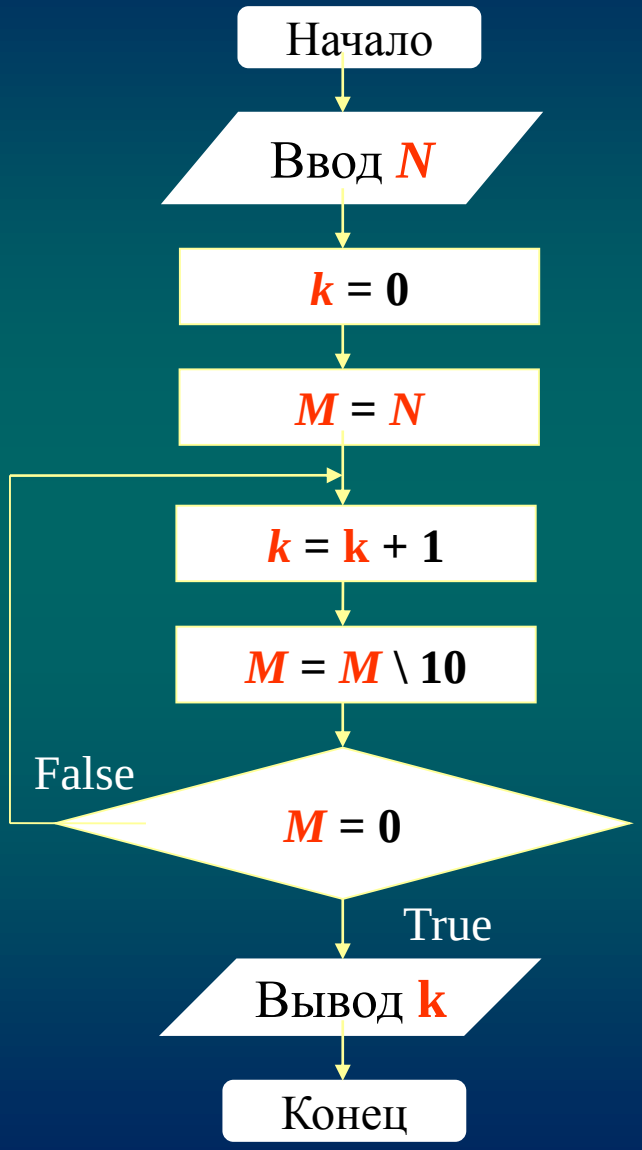
Пример. Определить количество цифр в натуральном числе **N**.

Для того чтобы подсчитать количество цифр в числе, необходимо определить, сколько раз заданное число можно разделить на десять нацело.

Например, пусть N=72865, тогда N делится на 10 ровно 5 раз.

Результаты вычислений сведены в таблицу.

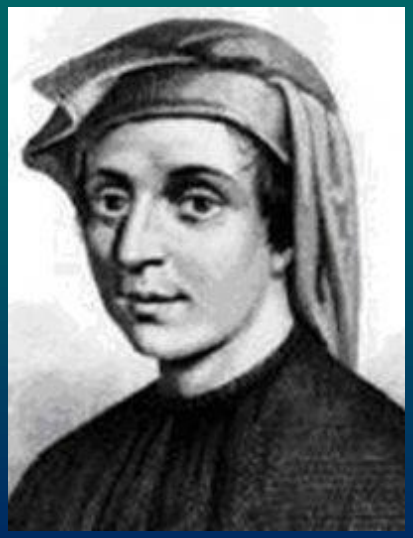
k	N = 72865
1	72865 \ 10 = 7286
2	7286 \ 10 = 728
3	728 \ 10 = 72
4	72 \ 10 = 7
5	7 \ 10 = 0



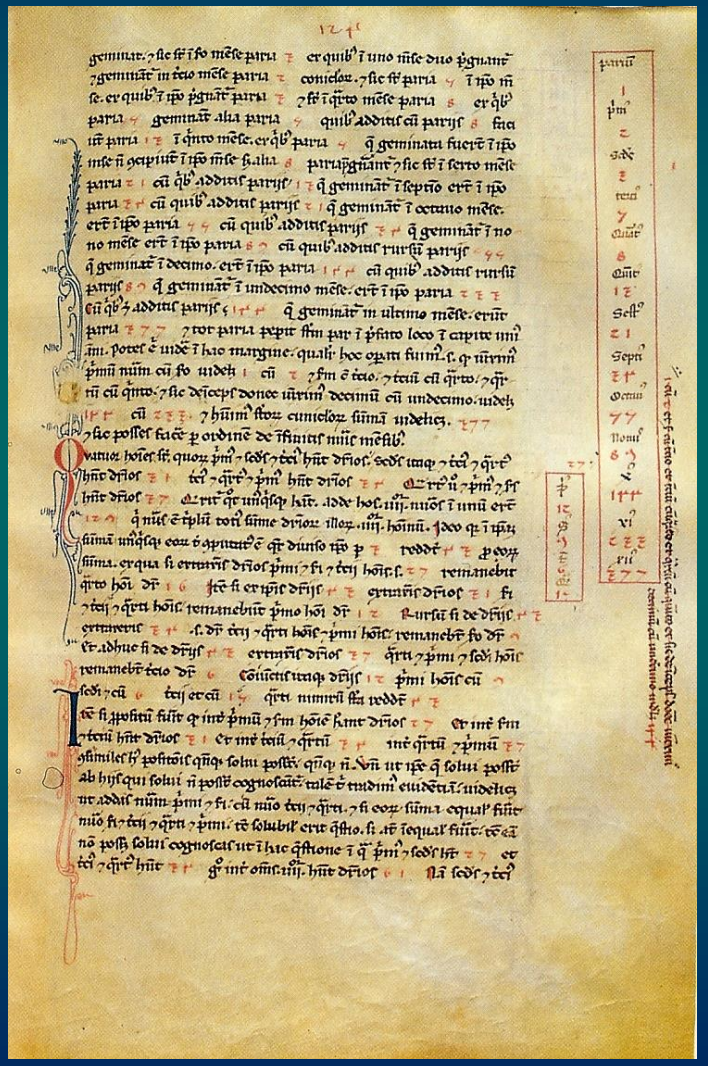
1.6.4 Расчет чисел Фибоначчи.

Пример. Выписать числа Фибоначчи, не превышающие число N.

Числа Фибоначчи – это бесконечная последовательность чисел x_1, x_2, x_3, …, x_{n-2}, x_{n-1}, x_n, …, определенных по правилу: первые два числа определены, как $x_1 = 0$, $x_2 = 1$, а все последующие числа получаются, как сумма двух предыдущих чисел данной последовательности.



Числа Фибоначчи были известны в Древней Индии и в Древнем Египте, но в Европе первым их описал Леонардо Пизанский (Фибоначчи, 1170-1250) в работе «Книга абака».



Согласно определению задаем первые два числа $x_1 = 0$, $x_2 = 1$ и помещаем их в список t.

Затем открываем цикл с предусловием $x_1 + x_2 > N$.

В теле цикла происходит перемещение значений

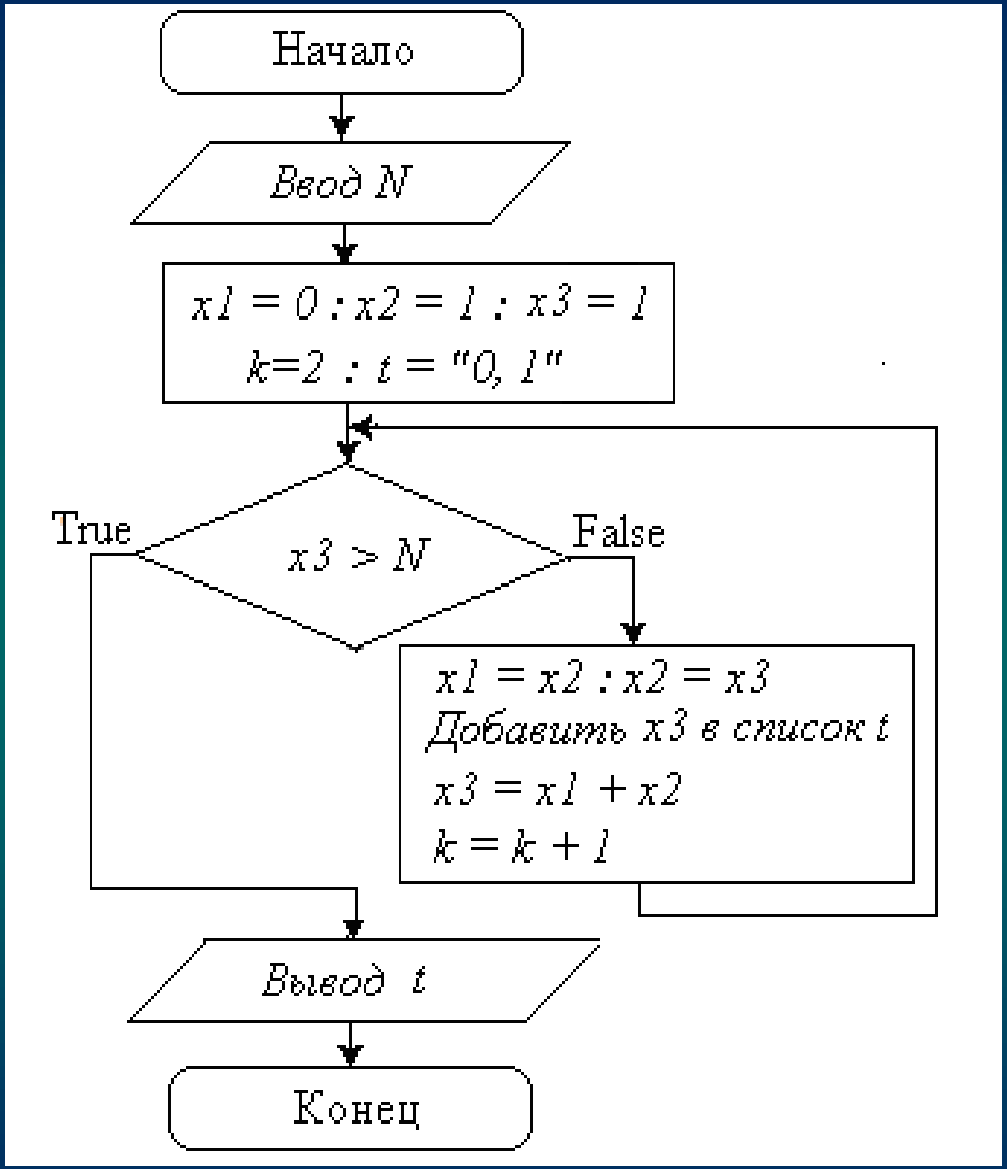
$x_1 = x_2$ и $x_2 = x_3$,

а затем расчет нового значения переменной $x_3 = x_1 + x_2$ и добавление x_3 в список t.

Циклический процесс заканчивается, когда сумма чисел

$x_1 + x_2$

станет больше введенного N.



Алгоритм вычисления чисел Фибоначчи

Глава 2. Введение в языки программирования

2.1 Общее описание языка VBA
2.2 Структура языка
2.3 Создание и запуск программы
2.4 Синтаксис и семантика языка
2.5 Типы данных
2.6. Переменные и константы
2.7. Операторы VBA
2.8. Встроенные функции

2.1. Общее описание VBA

Историю программирования традиционно ведут от языка *Fortran* (*FORmula TRANslator*), ставшего в 1957 г. первым языком программирования высокого уровня, не имевшим жесткой привязки к физическим адресам внутренних и периферийных устройств компьютера.

В **1963** г. профессорами **Дармутского** колледжа **Д.Кемени** и **Р.Куртцем** были разработаны первые 14 команд языка ***Basic***.

Basic (***Beginner's All-purpose Symbolic Instruction Code***) первоначально предназначался для обучения программированию.

В конце 60-х - начале 70-х годов язык получил мощную поддержку фирм *General Electric*, *HP* и *DEC*, и позднее практически все мини- и микрокомпьютеры снабжались *Basic-системами*.

Благодаря президенту *Microsoft Б.Гейтсу* Basic стал первым языком программирования для персональных компьютеров. А в 1976 г разработан *стандарт минимального подмножества* языка Basic, язык приобрел огромную популярность во всем мире в силу своей простоты и ориентации на диалоговый режим.

Сегодня наибольшее распространение получил *Visual Basic*, и в первую очередь *Visual Basic for Applications* (*VBA*), обслуживающий все приложения *Microsoft Office*.

Как и все языки высокого уровня, *Basic* строится в соответствии с концепцией *процедурного программирования*.

Процедурный подход основан на *алгоритмической декомпозиции* решаемой проблемы и реализуется посредством решения очевидных формализуемых задач.

Правило «*разделяй и властвуй*» ориентирует на представление проблемы набором самостоятельных блоков данных и процедур таким образом, чтобы, выполнив каждую из них, можно было прийти к решению всей проблемы. Благодаря процедурам лучше прослеживается структура больших и сложных программ; они обеспечивают логическую сегментацию всей задачи и облегчают отладку.

Достоинствами VBA являются:

1) ***Простой синтаксис*** языка. Небольшое число базовых понятий. Программы на VBA легко читаемы.

2) Достаточно ***низкие аппаратные и системные требования***.

3) ***Универсальность*** языка. Язык VBA применим для решения практически всех задач программирования.

4) Поддержка ***процедурного***, ***структурного*** и ***объектно-ориентированного*** программирования.

2.2. Структура языка

Состояние и поведение программных объектов описывается на *алгоритмическом языке* операциями над данными.

Данные в операциях представляются *операндами*, которые соединяются *знаками операций* - специальными лексемами - *операторами*, зарегистрированными в словаре языка.

В первую очередь операциями описываются процессы инициализации переменных и констант, то есть присваивания им определенных значений. Операции описывают, также, логические, математические и прочие процессы.

Кроме операторов, лексемами являются *стандартные константы* и имена *стандартных функций* языка программирования. Обращение к стандартной функции называют вызовом. Передаваемые при вызове данные для функции именуются значениями *аргументов*, а результат их обработки - *возвращаемым значением*.

Стандартная функция обрабатывает передаваемые ей значения аргументов по заданному алгоритму, возвращая результат в операцию, из которой она была вызвана.

Операции группируют в *выражения* - аналоги предложений естественных языков, строящиеся в соответствии с синтаксисом языка.

В дополнение к выражениям, текст программы часто содержит *комментарии*, помогающие понять смысл выражений - их семантику. Благодаря особому синтаксису, комментарии не влияют на ход выполнения программы.

Процедуры с выражениями и комментариями заключаются в *файлы исходных текстов*, из которых собираются *программы*.

Группы файлов обычно объединяются *проектом*. Через проект поддерживается отношение исходных текстов программы с *операционной системой* и *библиотеками*.

Проект включает в себя *объекты приложения*, в котором он разрабатывается и выполняется.

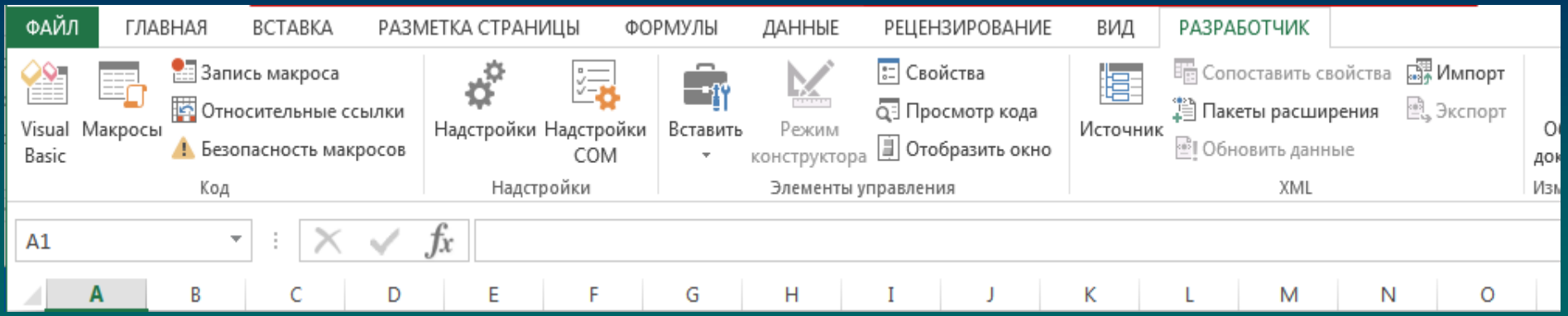
В случае *Word* или *Excel* это:

- *модули ThisDocument* или *ThisWorksheet* шаблона *Normal* и всех открытых документов;
- *дополнительные модули* с размещенными в них программными фрагментами;
- *модули форм* для ведения диалога с пользователем, описывающие их поведение;
- *модули классов*, характеризующие состав, свойства и подпрограммы авторских объектов разработчика.

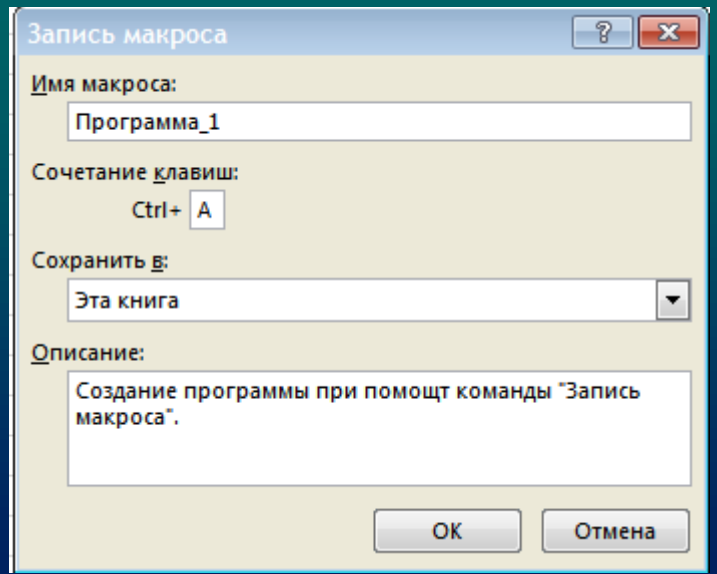
Проект хранится в одном файле с документом. В среде *Microsoft Office* можно обрабатывать сразу несколько документов, представленных своими шаблонами.

2.3. Создание и запуск программы

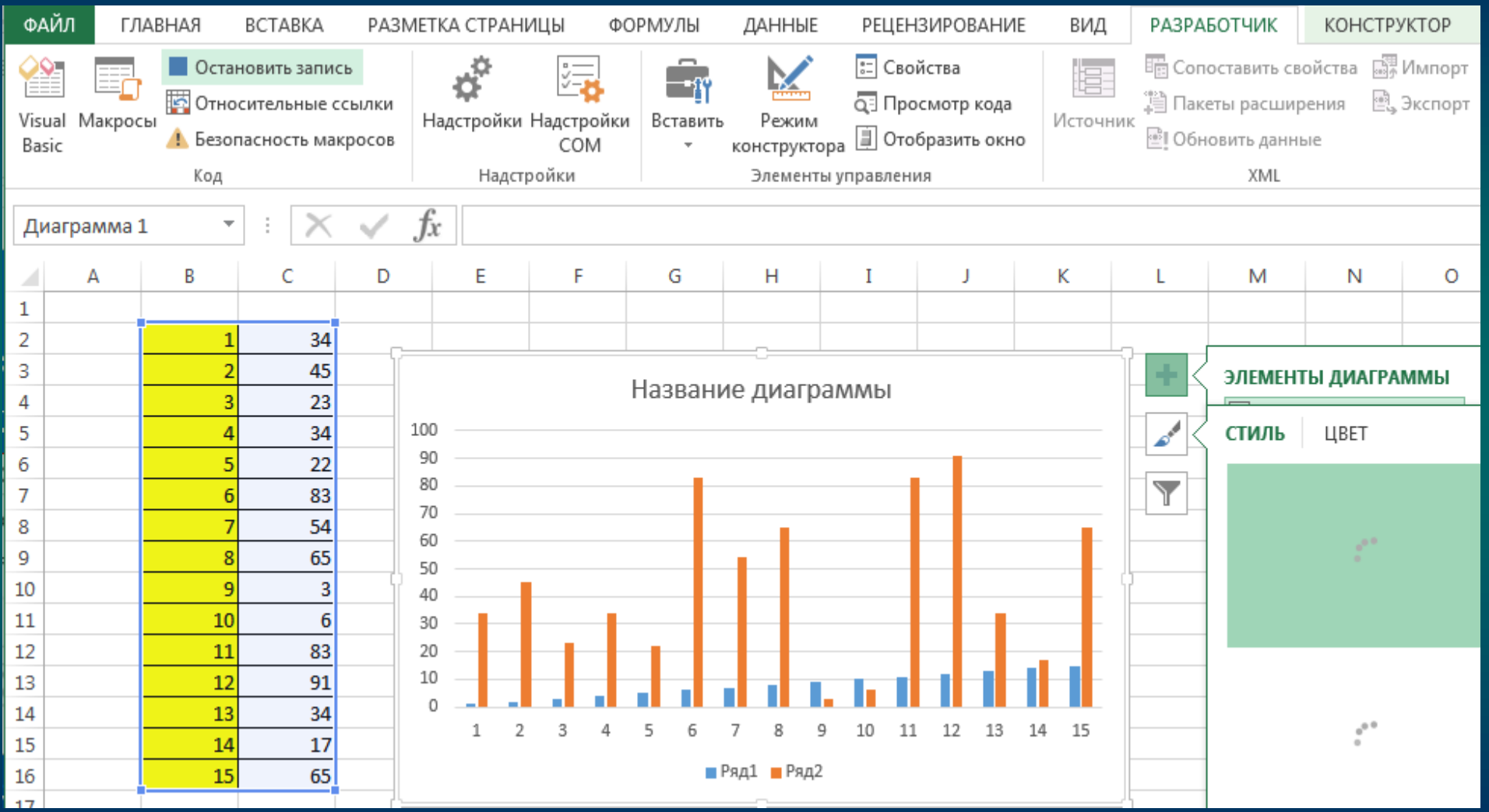
В приложениях *Microsoft Office* создание и разработка программ осуществляется с помощью закладки «*Разработчик*».



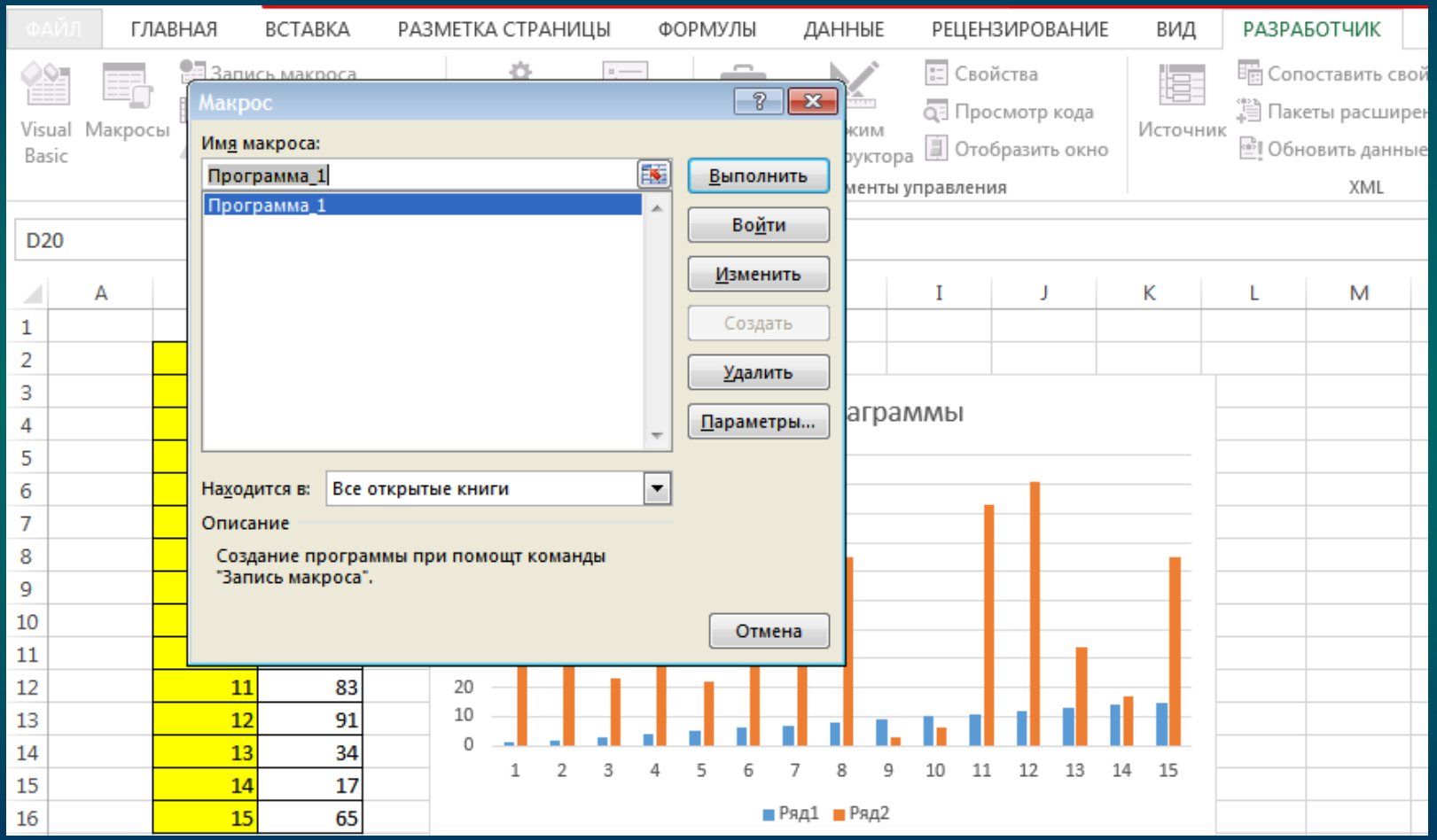
В простейшем случае программу можно написать с помощью пиктограммы «*Запись макроса*». Тогда все действия программиста с открытым приложением последовательно будут записаны в виде программного кода на языке *VBA*.



Для завершения записи необходимо остановить процесс командой «*Остановить запись*».

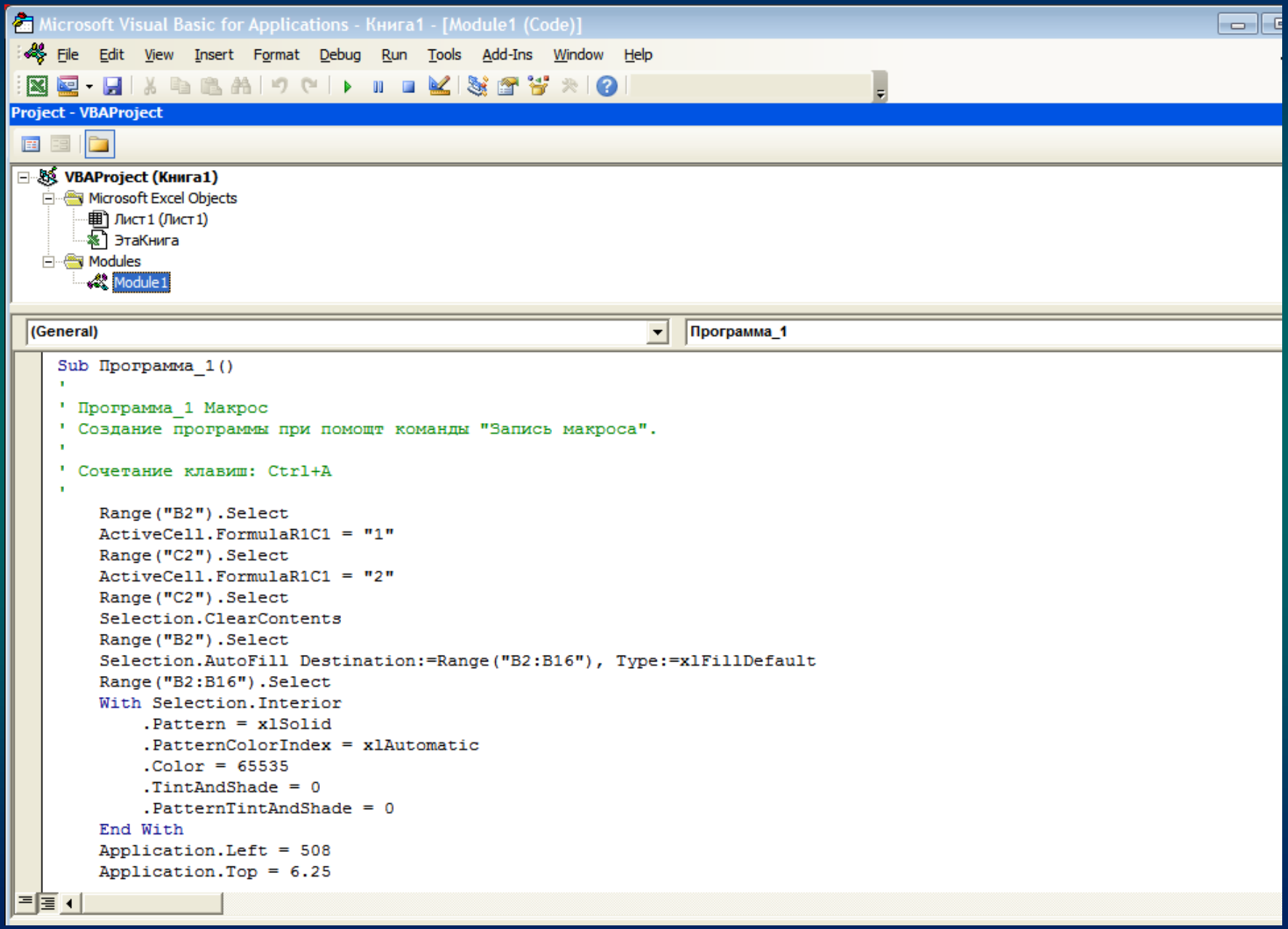


Для работы с созданной программой достаточно в закладке «Разработчик» выбрать команду «*Макросы*», а затем, отметив соответствующее название программы, совершить с ней действия, указанные в появившемся окне.

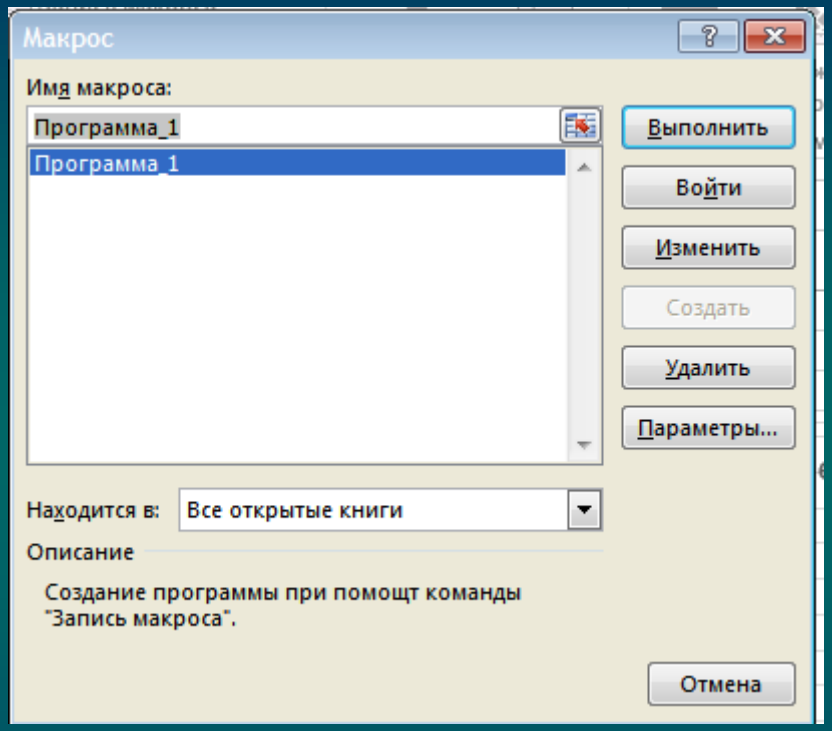


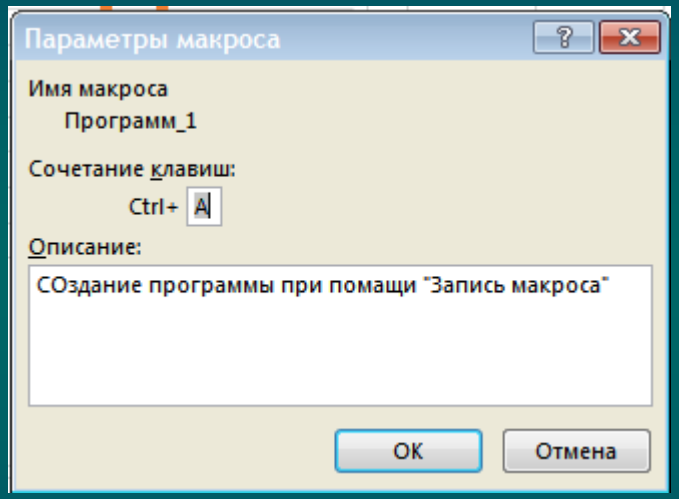
Кнопка «*Выполнить*» позволит выполнить все записанные ранее действия в том же порядке.

Кнопки «*Войти*» и «*Изменить*» запустят редактор *VBA* для дальнейшей работы с программным кодом.

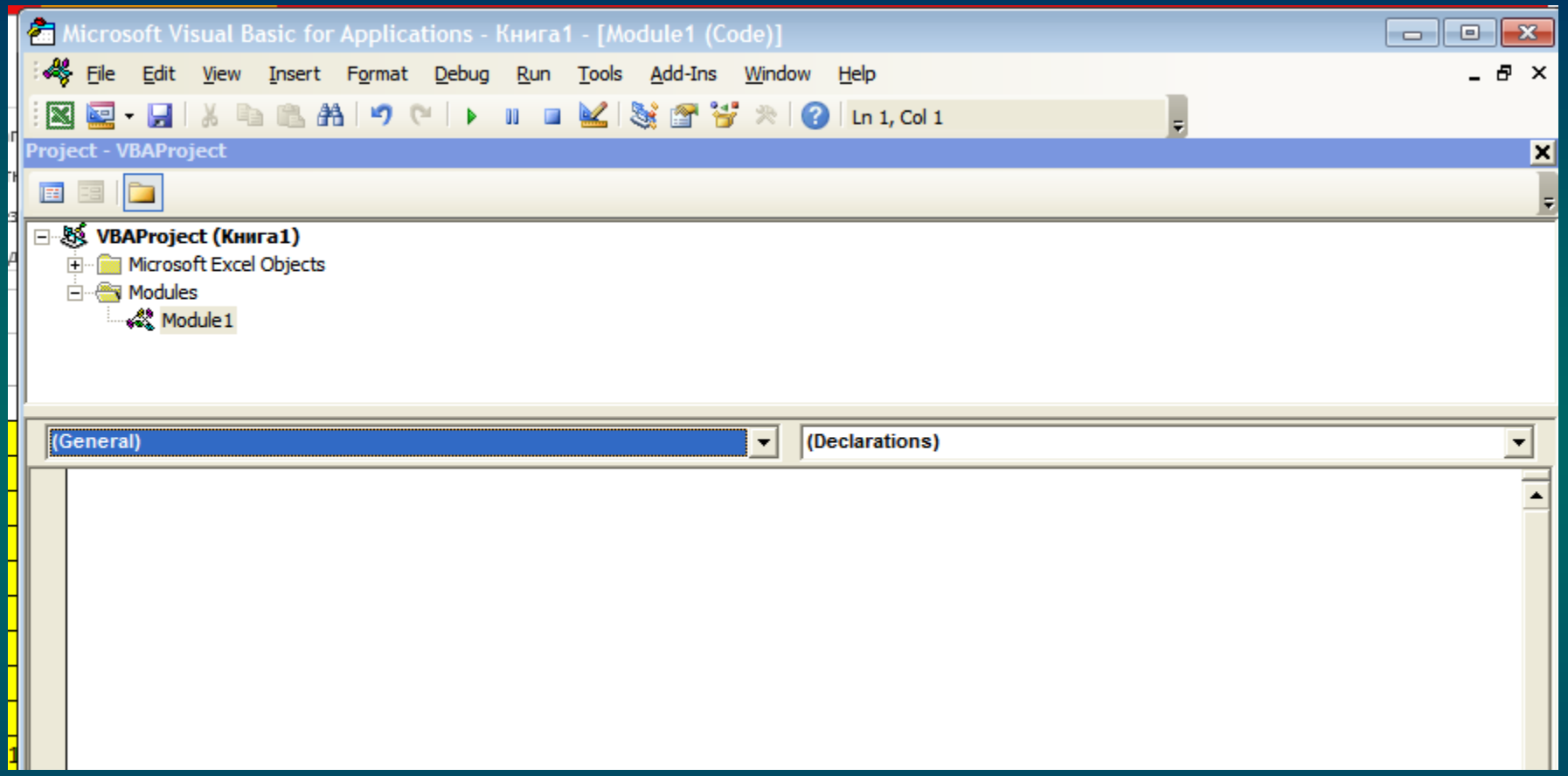


Кнопки «*Удалить*» и «*Параметры*» соответственно удалят выбранный макрос или выведут основные параметры макроса.



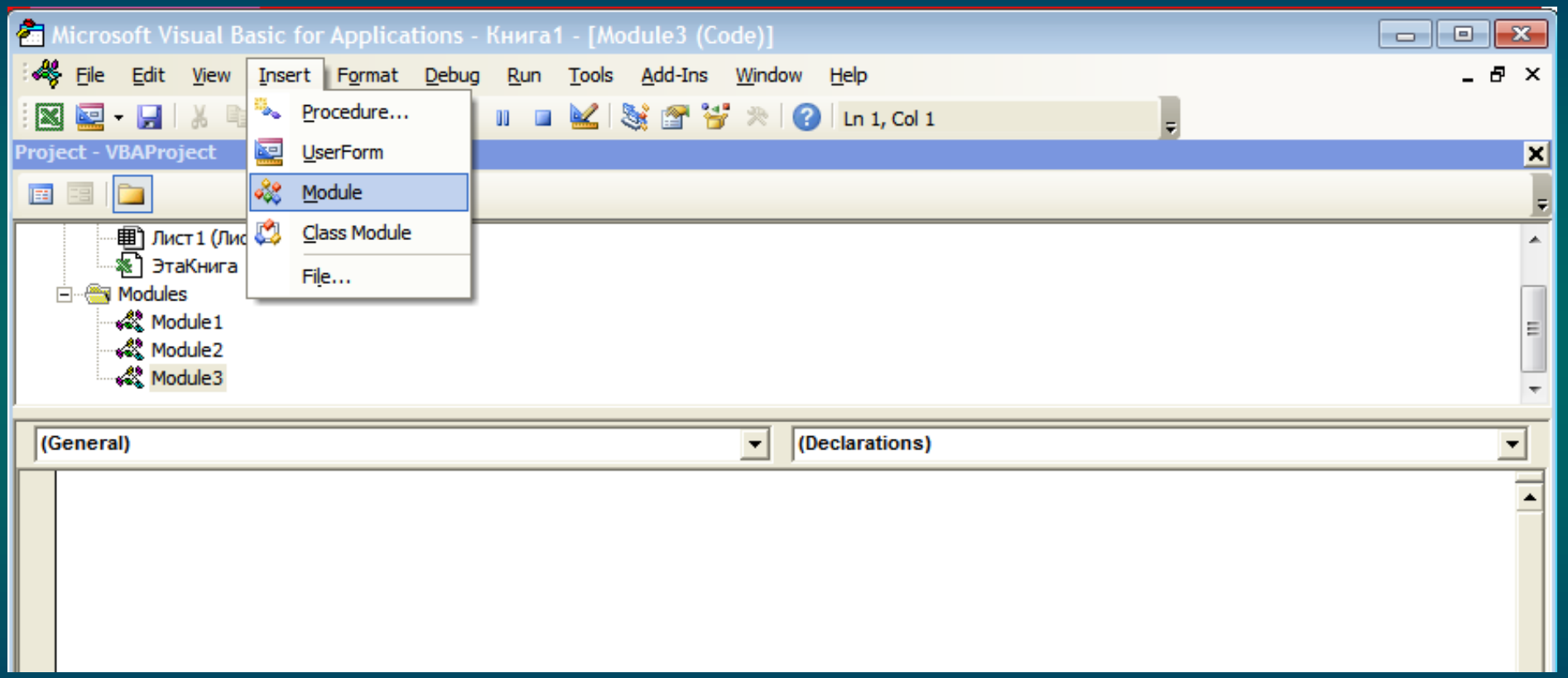


Программисты редко пользуются командой команды «*Запись макроса*» и часто пишут программы вручную. Для этого можно использовать пиктограмму «*Visual Basic*».



Автоматически создается «*пустой*» модуль для записи кода программы. В одном модуле можно разместить одну ил несколько программ, созданные программистом функции иди процедуры.

Для проектов, содержащих большое количество программ, процедур и функций, обычно пользуются несколькими модулями. Для создания нового модуля используется команда «*Module*» в закладке «*Insert*».

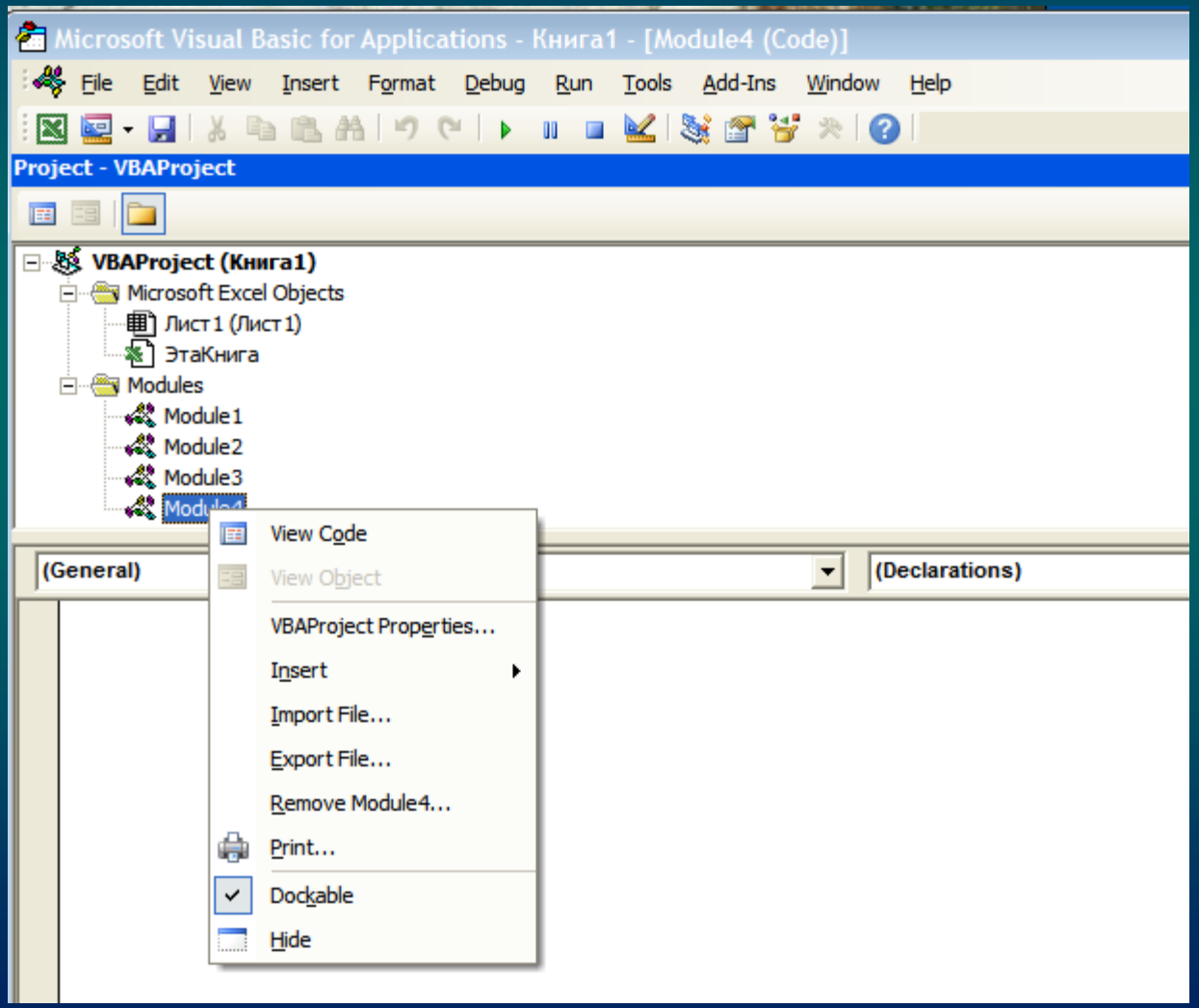


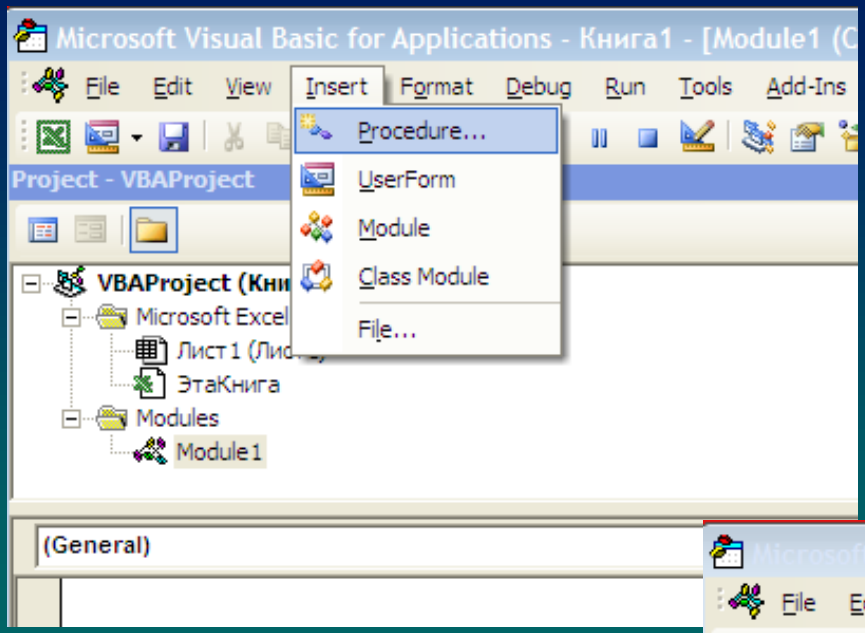
Для каждого модуля допустимы ряд опций.

«*VBAProject Properties…*» позволяет изменить основные свойства проекта, а также защитить проект от изменений или просмотра;

«*Remove*» позволяет перенести или удалить текущий модуль;

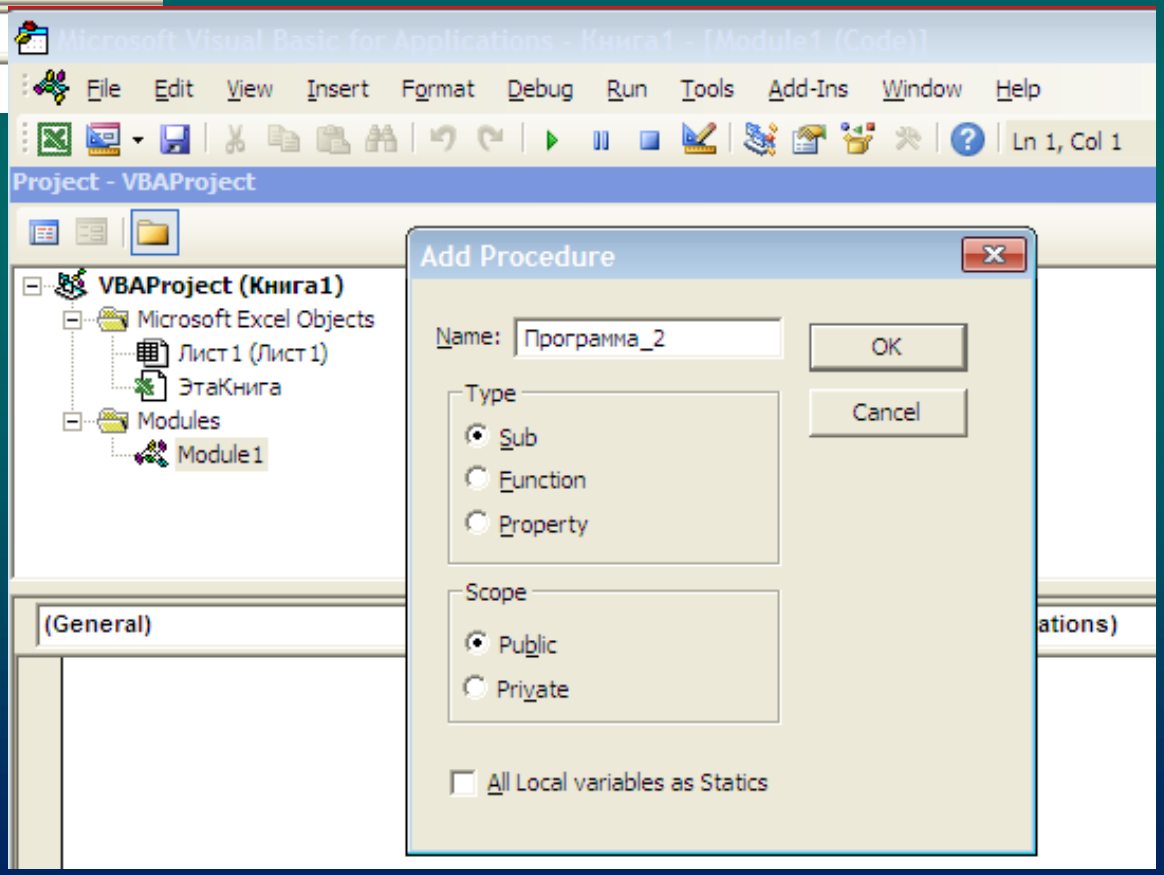
«*Print*» – распечатать модуль.



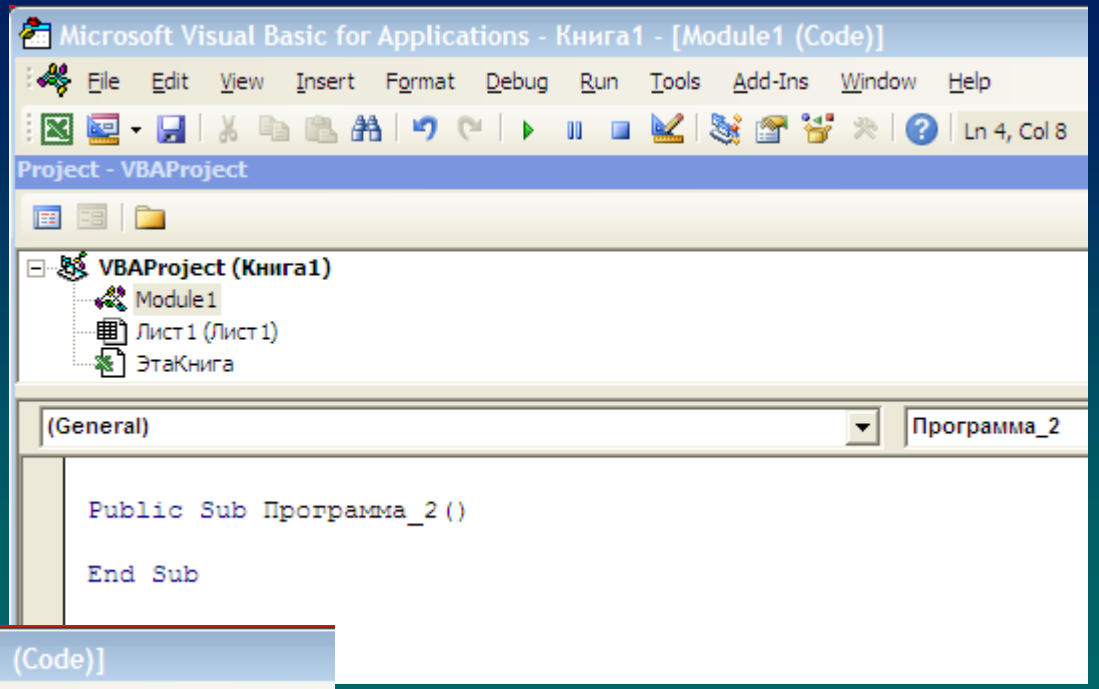


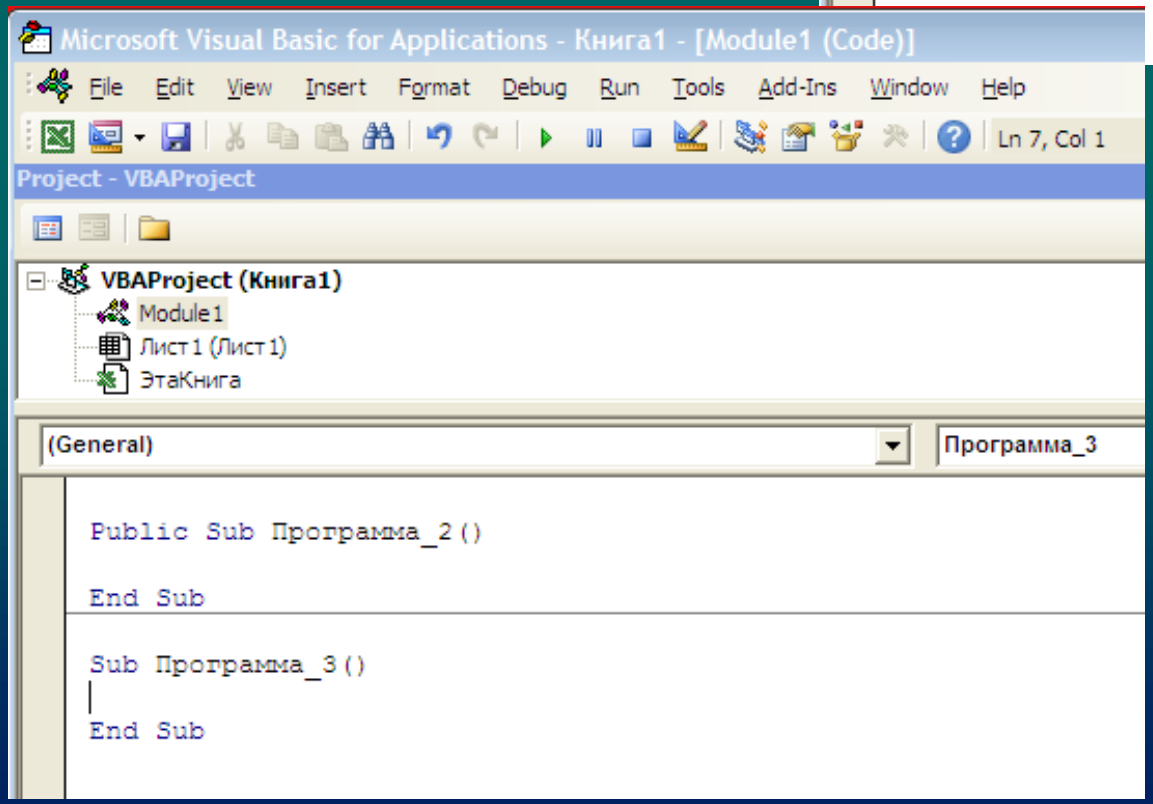
Для создания новой программы достаточно выбрать команду «*Procedure*» из закладки «*Insert*».

В появившемся окне необходимо заполнить поле «*Name*» (имя) и выбрать опцию «*Sub*» (программа).



В окне редактирования автоматически появится начало и конец новой программы. Весь код основной части программы можно записать между этими двумя строками





Создать программу можно, набрав в новой строке:

«Sub имя_программы»

При этом слово «Private» не является обязательным.

Строка «End Sub» появится автоматически.

2.4. Синтаксис и семантика языка

Описание каждого элемента языка задается его СИНТАКСИСОМ и СЕМАНТИКОЙ.

Синтаксические определения устанавливают правила построения элементов языка.

Семантика определяет смысл и правила использования тех элементов языка, для которых были даны синтаксические определения.

4.1. Алфавит языка

Алфавит - это совокупность допустимых в языке символов. Алфавит включает следующий набор основных символов:

строчные и прописные латинские буквы:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

a b c d e f g h i j k l m n o p q r s t u v w x y z п

пробел, подчеркивание

арабские цифры: 0 1 2 3 4 5 6 7 8 9

знаки операций: + - * / = <> < > <= > = := @

ограничители: . , ' ( ) [ ] { } : ;

спецификаторы: ^ # $

2.4.2. Элементарные конструкции

Элементарные конструкции языка Basic включают в себя имена, числа и строки.

Имена (идентификаторы) называют элементы языка - константы, метки, типы, переменные, процедуры, функции, модули, объекты.

Идентификатор в Basic может включать в себя:

буквы алфавита, цифры, символ подчеркивания.

Строчные и прописные буквы не различаются (например, NM21, Nm21 и nm21 будет означать одно и то же).

Длина идентификатора не может превышать 255 символа.

В качестве имен не допускается использовать служебные слова.

Цифра и символ подчеркивания не могут стоять на первом месте в идентификаторе.

Например,

x, x1, y_,, a_x, det21, det2x2, car, group_323 являются допустимыми идентификаторами;

2det, My friend, do, pr-k, - недопустимые названия.

Для отделения друг от друга идентификаторов, чисел, зарезервированных слов используются разделители. В качестве них можно использовать:

-пробел и табуляцию; - перевод строки; комментарий.

Перед комментариями в тексте программы ставится апостроф.

Числа обычно записываются в десятичной системе счисления. Они могут быть целыми и действительными. Положительный знак числа может быть опущен. Целые числа записываются в форме без десятичной точки, например:

217 -45 8954 +483

Действительные числа записываются в форме с десятичной точкой:

28.6 0.65 -0.018 4.0

Возможна также запись с использованием десятичного порядка, который изображается буквой E:

5E12 -1.72E9 73.1E-16

В "переводе" такую запись следует понимать соответственно как:

$5x10^{12}$ $-1.72x10^{9}$ $73.1x10^{-16}$

Допускается запись целых чисел и фрагментов действительных чисел в форме с порядком в шестнадцатеричной системе счисления:

$7F $40 $ABC0

Строки - это последовательность символов, записанная между кавычками. Если в строке в качестве содержательного символа необходимо употребить сами кавычки, то следует записать их три раза. Примеры строк:

” (Действительных решений нет!)” ” ” ” ” ”

” Значение переменной N” ” 5x2 - 6y + 8” ” ”

2.5. Типы данных

Для обработки ЭВМ данные представляются в виде величин и их совокупностей. С понятием величины связаны такая важная характеристика, как ее тип.

Тип определяет:

- возможные значения переменных, констант, функций, выражений, принадлежащих к данному типу;
- внутреннюю форму представления данных в ЭВМ;
- операции и функции, которые могут выполняться над величинами, принадлежащими к данному типу.

В языке VBA задание типа величины является необязательным!!!.

Тип переменных и констант может быть задан в любой части программы до их использования.

2.5.1. Иерархия типов в языке Basic:

- Простые
  - Числовые
    - Целые
    - Вещественные
  - Логические
  - Даты
- Структурированные
  - Строковые
  - Массивы
  - Variant
- Объектные

2.5.2. Описание типов данных

Идентификатор	Длина (байт)	Диапазон значений	Операции
Целые типы			
byte	1	0 … 255	+, -, /, *, Div, Mod, >=, <=, =, <>, <, >
integer	2	-32 768 … 32 767	+, -, /, *, Div, Mod, >=, <=, =, <>, <, >
long	4	-2 147 483 648 … 2 147 483 647	+, -, /, *, Div, Mod, >=, <=, =, <>, <, >
Вещественные типы			
single	4	$-3.402 \times 10^{38} .. -1.401 \times 10^{-45}$ $1.401 \times 10^{-45} .. 3.402 \times 10^{38}$	+, -, /, *, >=, <=, =, <>, <, >
double	8	$4.94 \times 10^{-324} .. -1.79 \times 10^{308}$	+, -, /, *, >=, <=, =, <>, <, >
currency	8	922 337 203 685 477.5808 … 922 337 203 685 477.5807	+, -, /, *, >=, <=, =, <>, <, >
decimal	14	-79 228 162 514 264 337 593 543 950 335 … 79 228 162 514 264 337 593 543 950 335	+, -, /, *, >=, <=, =, <>, <, >
Логический тип			
boolean	1	true, false	Not, And, Or, Xor, >=, <=, =, <>, <, >

Идентификатор	Длина (байт)	Диапазон значений	Операции
Строковый тип			
String Произв. длина	10 + длина строки	все символы кода ASCII	+, >=, <=, =, <>, <, >
String Фиксир. длина	длина строки	все символы кода ASCII	+, >=, <=, =, <>, <, >
Дата			
Date	8	01.01.0100 … 31.12.9999	+, >=, <=, =, <>, <, >
Тип Variant			
Variant (числовой)	16	Любые числа, не превышающие double	+, -, /, *, Div, Mod, >=, <=, =, <>, <, >
Variant (символьный)	22 + длина строки	Любая последова-тельность символов	+, >=, <=, =, <>, <, >
Объектный тип			
Object	4	Адреса объектов	=

Если не указан тип переменной или константы, то по умолчанию используется тип Variant. Тип такой переменной изменяется в зависимости от последнего присвоения.

2.6. Переменные и константы

Переменной называют элемент программы, который предназначен для хранения, коррекции и передачи данных внутри программы.

Самым простым способом создания переменной является *объявление переменной неявно*, используя ее идентификатор в любом операторе VBA. При этом автоматически создается переменная и резервируется память для ячейки памяти переменной.

Сохранение значений данных в переменной называется *присваиванием переменной*.

Присваивание производится с помощью *оператора присваивания*, представленного знаком равенства.

Пример.

```
a = 2.5
```

Этот оператор сохраняет численное значение 2,5 в ячейке памяти, заданной именем переменной a.

Все переменные, которые VBA создает неявным образом («*на лету*», «*on the fly*»), имеют тип данных Variant.

Неявное объявление переменных удобно, но имеет некоторые недостатки. Например, при неверном написании идентификатора в тексте программы автоматически создается новая переменная и нарушается логическая последовательность программы.

Для явного объявления переменных используется VBA-оператор *Dim* со следующим синтаксисом:

Dim var_1 [As type_1], var_2 [As type_2], ...,var_n [As type_n]

Пример.

```
dim a As Integer, b As Integer, x As Single
```

Если при явном объявлении переменных не заданы типы переменных, то переменным автоматически присваивается тип variant.

Пример.

```
dim x1, y1, z1, x2, y2, z2, x3, y3, z3
```

При неявном объявлении переменных также можно задать ее тип данных. Для этого к концу идентификатора переменной добавляются символы определения типов:

Символ	Тип данных
!	Single
@	Currency
#	Double
%	Integer
&	Long
$	String

Пример.

```
a! = 5.64
b! = 44.76
x% = (a + b)/2
```

Для явного объявления *строковых переменных фиксированной длины* используется VBA-оператор *Dim* со следующим синтаксисом:

```
Dim str_name As String * N
```

Пример.

```
dim text_1 As String * 20
```

В примере объявляется строковая переменная text_1 с заданной длиной в 20 символов.

Для обнаружения ошибок, связанных с неявным объявлением переменных, VBA предоставляет инструкцию *Option Explicit*, использование которой требует объявления всех переменных перед их использованием.

Пример.

```
Option Explicit
Dim a As integer, b As integer, c As integer
Dim d, x1, x2
```

Команда Option Explicit действует только на работу программ в модуле, в котором она находится!

Константа - это идентификатор, обозначающий некоторую *неизменную величину* определенного типа.

Преимущества использования констант:

- код становится лучше читаемым (убираются потенциальные ошибки);
- возможность многократного использования в программном коде одного и того же значения, введенного один раз.

В VBA константы определяются в любом месте программы при помощи ключевого слова Const:

Пример:

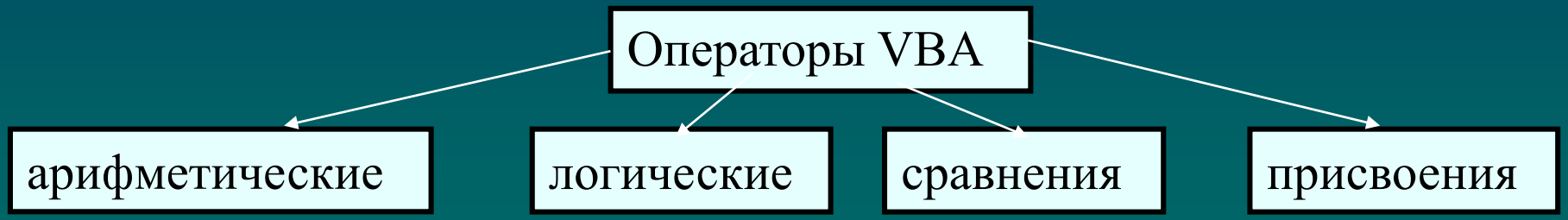
```
Const Np As String = ”Краснодарский университет МВД России ”
Const k_pro As Double = 2.59483672905
```

Константы удобны при работе с группами именованных элементов (дни недели, месяцы, цвета, клавиши, типы окон и т.п.). Они позволяют использовать в коде программы легко читаемые обозначения вместо труднозапоминаемых числовых кодов.

При попытке в теле процедуры изменить значение константы будет выдано сообщение об ошибке.

2.7. Операторы VBA

Оператор — это наименьшая способная выполняться единица кода VBA. Оператор может объявлять или определять переменную, устанавливать параметр компилятора VBA или выполнять какое-либо действие в программе.



7.1. Арифметические операторы		
Сложение	+	2+3
Вычитание	-	5-14.6
Умножение	*	7*8
Деление	/	15/8
Возведение в степень	^	2^9
Целочисленное деление	\	19\3
Деление по модулю	mod	23 mod 5

2.7.2. Операторы сравнения

Равно	=	a = b
Больше	>	X(2) >-3
Меньше	<	z3 < 6
Больше или равно	>=	t >= ” ”
Меньше или равно	<=	Kol <= 20
Не равно	<>	Norma <> 0
Сравнение объектов	Is	obj1 is obj2
Подобие	Like	stroka like shablon

Операторы сравнения всегда возвращают true, если утверждение истинно, и false, если ложно.

Правила сравнений строковых значений:

при сравнении строковых значений регистр учитывается;

пробелы в строковых значениях также учитываются;

при сравнении строк на больше/меньше по умолчанию сравниваются двоичные коды символов.

При проверке нескольких условий используются логические операторы

2.7.3. Логические операторы		
Логическое И	And	X mod 3 = 0 and x <= 20
Логическое ИЛИ	Or	M = ”хорошо” or M= ”отлично”
Логическое отрицание	Not	Not (x > 10*y or x > z^2)
Логическое исключение	Xor	t \ 10 = t Xor t > 0
Эквивалентность	Eqv	
Импликация	Imp	

And возвращает TRUE, если истинны оба условия;
Or возвращает TRUE, если истинно хотя бы одно из условий;
Not возвращает TRUE, если условие ложно;
Xor возвращает TRUE, если только первое выражение истинно или только второе выражение истинно;
Eqv возвращает TRUE, если выражения имеют одинаковое значение;
Imp возвращает TRUE во всех случаях, кроме если первое выражение истинно, а второе ложно.

2.7.4. Оператор присвоения

Оператор присвоения *Let* служит для изменения значений переменных.

Синтаксис

Let var = выражение

Слово let в операторе присваивания можно опускать.

Примеры:

```
Let x = 10
k = k + 1
D = b^2 – 4*a*c
NameObject = No + ” : ” + index & + FirstName
NewDate = Today + Cout mod 365
```

Важно помнить, что следует соблюдать соответствие типов переменной, стоящей слева от знака равенства, и выражения стоящего справа от знака равенства, в операторе присваивания!

2.8. Встроенные функции

В языке программирования VBA предусмотрено несколько десятков *встроенных функций*. Они доступны в любой программе на языке VBA, при этом безразлично, в среде какого программного продукта программа будет выполняться - Excel, Word, Access или, например, AutoCAD.

Основные категории функций

- Математические функции
- Функции обработки массивов
- Функции обработки строк
- Тригонометрические функции
- Функции преобразования типа данных
- Функции загрузки данных
- Функции работы с файлами
- Функции обработки системных параметров
- Функции обработки цвета
- Функции работы с датами
- Функции преобразования чисел в разные системы счисления
- Функции работы с объектами
- Финансовые функции
- Функции форматирования
- Функции работы с указателями

2.8.1. Математические функции

2.8.1.1. Функция Abs

Abs(Number)

Служит для вычисления абсолютного значения(модуля) числа.

Возвращаемое значение

В результате действия функции Abs возвращается значение, тип которого совпадает с типом переданного аргумента и равняется абсолютному значению указанного в аргументе числа

Параметры

Number

Обязательный аргумент может представлять любое допустимое числовое выражение. Если аргумент имеет значение Null, то возвращается также Null. Если аргумент - не инициализированная переменная, то возвращается нулевое значение

Пример

```
Abs(50.3)     ' возвращается 50.3
Abs(-38.4)    ' возвращается 38.4
```

2.8.1.2. Функция Log

Log (Number)

Функция вычисляет значения натурального логарифма *ln(x)*. Применяется в математических и статистических расчетах.

Для вычисления логарифма числа *x* по основанию *n* следует разделить натуральный логарифм числа *x* на натуральный логарифм числа *n*:

Log(*x*) / Log(*n*)

Возвращаемое значение

Возвращает значение типа *Double*, содержащее натуральный логарифм числа.

Параметры

Number

Обязательный параметр является любым действительным числовым выражением, значение которого больше нуля. Если аргумент равен нулю или отрицателен, то генерируется ошибка Invalid procedure call or argument.

Пример

```
Log(8)/Log(2)          ' Возвращает число 3
Log(Exp(7))            ' Возвращает число 7
```

2.8.1.3. Функция Exp

Exp(Number)

Функция Exp используется для вычисления степени числа *e* (*e*≈2.7182818...). Функция Exp является обратной к функции Log (логарифм по основанию e), поэтому ее иногда называют антилогарифмом.

Параметры

Number

Обязательный аргумент представляет собой значение типа *Double*. Аргумент не может превышать число 709 782 712 893. Если аргумент не может быть оценен как число, то генерируется ошибка *Type mismatch*.

Пример

```
Exp(2)          ' возвращается 7.38905609893065
```

2.8.1.4. Функция Fix

Fix(Number)

Функция Fix(Fixed) отбрасывает дробную часть числа. Для отрицательного значения аргумента число функция возвращает ближайшее отрицательное целое число, большее либо равное указанному.

Возвращаемое значение

Функция возвращает значение типа, совпадающего с типом аргумента, которое содержит целую часть числа

Параметры

Number

Обязательный аргумент Number может представлять любое допустимое числовое выражение. Если значение аргумента не попадает в диапазон допустимых значений *Double*, то генерируется ошибка стадии выполнения Overflow. Если аргумент имеет другой тип данных, то генерируется ошибка Type mismastch.

Пример

```
Fix (9.81)      ' Возвращает 9
Fix (-5.458)    ' Возвращает -5
```

2.8.1.5. Функция Int

Int(Number)

Функция Int(Integer) возвращает целую часть числа. Для отрицательного значения аргумента функция Int возвращает ближайшее отрицательное целое число, меньшее либо равное указанному

Возвращаемое значение

Функция возвращает значение типа, совпадающего с типом аргумента, которое содержит целую часть числа

Параметры

Number

Обязательный аргумент Number может представлять любое допустимое числовое выражение. Если значение аргумента не попадает в диапазон допустимых значений *Double*, то генерируется ошибка *Overflow.* Если аргумент имеет другой тип данных, то генерируется ошибка *Type mismatch*.

Пример

```
Int(9.81)       ' Возвращает 9
Int(-5.458)     ' Возвращает -6
```

2.8.1.6. Функция IsNumeric

IsNumeric(Expression)

Функция IsNumeric проверяет, является ли значение данного выражения числом. Функция способна обрабатывать данные любого типа без генерации ошибки.

Возвращаемое значение

Возвращает значение типа *Boolean*, показывающее, имеет ли выражение числовое значение. Функция IsNumeric возвращает *True*, если выражение имеет числовое значение; в противном случае возвращается *False*.

Параметры

Expression

Обязательный аргумент представляет выражение типа *Variant*, содержащее числовое выражение или строковое выражение.

Пример

```
a = 5/3
b = ”Список функций ”
IsNumeric(a)          ' Возвращает True
IsNumeric(b)          ' Возвращает False
```

2.8.1.7. Функция Rnd

Rnd(Number)

Функция Rnd [(Random)] служит для генерации случайных чисел.

Возвращаемое значение

Функция Rnd возвращает значение в диапазоне от 0 до 1 типа *Single*, содержащее случайное число (причем 1 не входит в этот диапазон, а 0 входит). При каждом запуске программы, функция генерирует одну и ту же последовательность псевдослучайных чисел для одного и того же аргумента.

Во избежание этого явления используется инструкция Randomize.

Для получения значения случайного числа в интервале [*a; b)* используется формула:

(*b* - *a*) * Rnd + a.

Параметры

Number

Необязательный аргумент представляет любое допустимое числовое выражение.

Пример

```
Rnd(-2)                 ' Возвращает число от 0 до 1
Int(21 * Rnd(i) - 10)   ' Возвращает целое число от -10 до 10
```

2.8.1.8. Функция Round

Round (Number [,NumDigitAfterDecimal])

Функция Round служит для округления чисел до заданной точности.

Возвращаемое значение

В результате действия функции возвращается округленное число, тип которого совпадает с типом переданного аргумента.

Параметры

Number

Обязательный аргумент может представлять любое допустимое числовое выражение.

NumDigitAfterDecimal

Необязательный аргумент, представляющий собой целое положительное число, указывающее, сколько знаков следует оставить после запятой. Если аргумент опущен, то округление происходит до целых.

Пример

```
Round(2.895648546) ' Возвращает число 3
Round(2.895648546, 4)        ' Возвращает число 2.8956
```

2.8.1.9. Функция Sgn

Sgn (Number)

Функция Sgn определяет знак аргумента.

Возвращаемое значение

Функция возвращает величину с типом данных *Variant(Integer)* следующим образом:

Положительный аргумент - возвращается 1;
Аргумент равен нулю - возвращается 0;
Отрицательный аргумент - возвращается -1.

Параметры

Number

Обязательный аргумент представляет собой любое выражение, оцениваемое как число. Если аргумент не может быть оценен как число, генерируется ошибка Type mismatch.

Пример

```
Sgn (95.64)          ' Возвращает число 1
Sgn (-5)             ' Возвращает число -1
Sgn (0)              ' Возвращает число 0
```

2.8.1.10. Функция Sqr

Sqr (Number)

Функция Sqr вычисляет значения элементарной математической функции квадратного корня.

Возвращаемое значение

Возвращает значение типа *Double*, содержащее квадратный корень указанного числа.

Параметры

Number

Обязательный аргумент представляет значение любого допустимого неотрицательного числового выражения. Если аргумент равен нулю или отрицателен, то генерируется ошибка Invalid procedure call or argument

Пример

```
Sqr (625)          ' Возвращает число 25
Sqr (2)            ' Возвращает число 1,4142135623731
```

2.8.2. Тригонометрические функции

Наименование функции	*Возвращаемое значение*	*Тип*	*Параметр*
Atn (x)	Величина угла в радианах	Double	Допустимое числовое значение
Sin (φ)	Синус угла в радианах	Double	Допустимое числовое значение
Cos (φ)	Косинус угла в радианах	Double	Допустимое числовое значение
Tan (φ)	Тангенс угла в радианах	Double	Допустимое числовое значение

Если аргумент не может быть оценен как число, генерируется ошибка Type mismatch.

Пример

```
Atn (1)      ' Возвращает число 0.785398163397448
Sin (2)      ' Возвращает число 0.909297426825682
Cos (2)      ' Возвращает число -0.416146836547142
Tan (2)      ' Возвращает число -2.18503986326152
```

2.8.3. Функции обработки строк

2.8.3.1. Функция Asc

Asc (String), AscB (String), AscW (String)

Функция Asc (ASCII code) преобразует символ в числовое значение

Возвращаемое значение

В результате действия функции Asc возвращается Asc-код ANSI первого символа строки типа *Integer;*

функции AscB - первый байт строки, вне зависимости от кодировки;

функции AscW - Unicode-код для первого символа строки, если система поддерживает Unicode. В остальных случаях аналогична Asc.

Параметры

String

Обязательный аргумент может представлять любое строковое выражение. Если строка не содержит символов, возникает ошибка выполнения.

Пример

```
Asc("Аргумент функции")     ' Возвращает число 192
AscB("Аргумент функции")    ' Возвращает число 16
AscW("Аргумент функции")    'Возвращает число 1040
```

2.8.3.2. Функция Chr

Chr(CharCode); ChrB(CharCode); ChrW(CharCode)

Chr$(CharCode); ChrB$(CharCode); ChrW$(CharCode)

Функция Chr(Character)позволяет получить символ по значению его числового кода ANSI или Unicode

Возвращаемое значение

Chr, ChrB, ChrW возвращают значение субтипа *String* типа *Variant*.

Chr$, ChrB$, ChrW$ возвращают значение типа *String*.

Chr и Chr$ возвращают символ по его кодировке в стандарте ANSI.

ChrB и ChrB$ возвращают однобайтовую строку.

ChrW и ChrW$ возвращает символ Unicode, однако в системах, не поддерживающих Unicode, ее поведение аналогично Chr.

Параметры

CharCode

Обязательный аргумент является значением типа *Long*. Коды 0-31 соответствуют стандартным управляющим символам ASCII.

Пример

```
Chr (136)          ' Возвращает символ «€»
ChrB$ (163)        ' Возвращает символ « £»
ChrW (174)         ' Возвращает символ « ®»
```

2.8.3.3. Функция Val

Val(String)

Функция Val(Value) служит для преобразования аргумента в числовой тип данных. Функция Val распознает в качестве разделителя целой и дробной части только точку (.). Пробелы игнорируются. Все остальные нечисловые символы не распознаются. Функция Val прекращает чтение строки на первом символе, который она не может распознать в качестве части числа.

Возвращаемое значение

Возвращает числовое представление аргумента с подходящим типом данных

Параметры

String

Обязательный аргумент является любым допустимым строковым выражением.

Пример

```
Val(”Всего: $456.78”)          'Возвращает число 0
Val(” 34 567 890. 79 ”)        'Возвращает число 34567890.79
Val(” 23 руб. 45 коп.”)        'Возвращает число 23
```

2.8.3.4. Функция Str

Str(Number)

Функция Str(String) используется для приведения числового выражения типа *Long* в строку(тип *String*)

Возвращаемое значение

Функция Str(Number) возвращает значение Number, преобразованное в текстовый тип данных *String*.

При преобразовании в начале строки возвращаемого значения резервируется место для знака числа. Если число положительно, то в этом месте будет пробел, если число отрицательно, то выводится знак минус. В качестве десятичного разделителя дроби функция Str воспринимает только точку.

Параметры

Number

Любое выражение, которое можно записать в виде строки

Пример

```
x=-29.78
Str(x)                  'Возвращает строку ”-29.78”
Str(2*x+100)            'Возвращает строку ” 40,44”
```

2.8.3.5. Функция String

String(Number, Character)

Функция String используется для создания строки из одинаковых символов.

Возвращаемое значение

Возвращает значение типа *Variant (String)*, содержащее строку повторяющихся символов указанной длины.

Параметры

Number

Обязательный аргумент - число типа *Long*, определяющее длину возвращаемой строки.

Character

Обязательный аргумент - значение типа *Variant*. Код символа или строковое выражение, первый символ которого используется при создании возвращаемой строки.

Пример

```
String(10, "X")        'Возвращает строку "XXXXXXXXXX"
String(30, 136)        'Возвращает строку "‹‹‹‹‹‹‹‹‹‹‹‹‹‹‹‹‹‹‹‹‹‹‹‹‹‹"
```

2.8.3.6. Функция Len

Len(String / Varname)

Функция Len вычисляет число символов в строке или размер заданной переменной. Из двух возможных аргументов должен быть указан только один. Для определяемых пользователем типов Len возвращает размер, который требуется для записи переменной в файл.

Возвращаемое значение

Возвращает значение типа *Long*, содержащее число символов в строке или число байт, необходимое для размещения переменной.

Параметры

String / Varname

Обязательный аргумент - любое допустимое строковое выражение.

Если аргумент имеет значение типа *Variant*, то функция Len обрабатывает его так же, как и значение типа *String*.

Пример

```
x = ”Значение типа ”: y =  ”Long”
Len(x & ” ” ” ” & y & ” ” ” ”)          'Возвращает число 20
```

2.8.3.7. Функция Left

Left(String, Length)

Функция Left служит для усечения исходной строки до заданной длины слева.

Возвращаемое значение

Возвращает строку фиксированной длины.

Параметры

Функция содержит именованные аргументы:

String

Обязательный аргумент - строковое выражение, из которого извлекаются символы.

Length

Обязательный аргумент - значение типа *Variant(Long)*. Числовое выражение, указывающее число возвращаемых символов.

Пример

```
x = ”Функция Left служит для усечения исходной строки.”
Left (x, 12)          'Возвращает строку ”Функция Left”
Left (x, 7)           'Возвращает строку ”Функция
```

2.8.3.8. Функция Right

Right(String, Length)

Функция Right служит для усечения исходной строки заданной длины справа.

Возвращаемое значение

Возвращает значение типа *Variant (String)*, содержащее указанное число последних символов.

Параметры

String

Обязательный аргумент - строковое выражение, из которого извлекаются символы.

Length

Обязательный аргумент - значение типа *Variant (Long)*. Числовое выражение, указывающее число возвращаемых символов.

Пример

```
x = ”Функция Right служит для усечения исходной строки.”
Right (x, 36)  'Возвращает строку ”для усечения исходной строки.”
Right (x, 7)          'Возвращает строку ”строки.”
```

2.8.3.9. Функция Mid

Mid(String, Start, [Length])

Функция Mid(Middle) используется для считывания заданного числа символов подряд от заданной позиции в строке слева направо.

Возвращаемое значение

Возвращает значение типа *Variant (String)*, содержащее указанное число символов строки

Параметры

String Обязательный аргумент - строка, из которой извлекаются символы.

Start Обязательный аргумент - значение типа *Long*. Позиция символа в строке String, с которого начинается нужная подстрока.

Length Необязательный аргумент - значение типа *Variant Long)*. Число возвращаемых символов. Если этот аргумент опущен или превышает число символов, расположенных справа от позиции Start, то возвращаются все символы от позиции Start до конца строки.

Пример

```
x = ”Функция Mid служит для усечения исходной строки.”
Mid (x, 8, 10)          'Возвращает строку ”Mid служит”
```

2.8.4. Функции дат и времени

2.8.4.1. Функция Date

Date; Date$

Функция Date позволяет получить текущую системную дату по системному календарю компьютера.

Возвращаемое значение

Date возвращает значение типа *Variant(Date)*,

Date$ возвращает данные типа *String*, содержащее текущую системную дату. Формат даты, возвращаемый функцией, определяется национальными системными установками.

Параметры

Функция не имеет аргументов

Пример

```
Cells(1, 1) = ”Сегодня ” & Date$      'Возвращает ”Сегодня 01-31-2014”
Cells(2, 1) = ”Завтра ”  & Date + 1   'Возвращает ”Завтра 01.02.2014”
```

2.8.4.2. Функция Now

Now

Функция Now позволяет узнать текущую дату и время по системному календарю и часам компьютера.

Примечание: Пользователь может самостоятельно устанавливать системное время и дату на своем компьютере, поэтому эти значения могут не иметь ничего общего с реальной датой и временем!

Возвращаемое значение

Возвращает значение типа *Variant(Date)*, содержащее текущую дату и время.

Параметры

Функция не имеет аргументов

Пример

```
Cells(3, 1) = ”Сейчас ” & Now
                'Возвращает ”Сейчас 31.01.2014 10:09:47”
Cells(4, 1) = ”Через 1000 дней ” & Now + 1000
                'Возвращает ” Через 1000 дней 27.10.2016 10:09:47”
```

2.8.4.3. Функция Year

Year(Date)

Функция используется для получения года из заданной даты.

Возвращаемое значение

Возвращает значение типа *Variant(Integer)*, содержащее целое число, представляющее год.

Параметры

Date

Обязательный параметр является любым значением типа *Variant*, числовым выражением, строковым выражением или любой комбинацией, с помощью которой может быть отражена дата.

Пример

```
Cells(4, 1) = "Текущий год " & Year(Now)
                                    'Возвращает ”Текущий год 2014”
i = "21-02-16"
    Cells(5, 1) = "Расчетный год " & Year(i) +17
                                    'Возвращает ” Расчетный год 2033”
```

2.8.4.4. Функция Month

Month(Date)

Функция Month получает номер месяца для заданной даты

Возвращаемое значение

Возвращает значение типа *Variant(Integer)*, содержащее целое число (от 1 до 12 включительно), которое представляет месяц в значении даты.

Параметры

Date

Обязательный аргумент может быть любым значением типа *Variant*, числовым выражением, строковым выражением или любым их сочетанием, которое представляет дату.

Пример

```
Cells(6, 1) = "Сейчас номер месяца " & Month(Now)
                    'Возвращает ”Сейчас номер месяца 1”
Cells(7, 1) = "Через 10000 дней номер месяца " & Month(Now + 10000)
                    'Возвращает ” Через 10000 дней номер месяца 6”
```

2.8.4.5. Функция Day

Day(Date)

Функция Day используется для получения дня месяца из даты. Отчет дат начинается с 31 декабря 1899 года, поэтому нулевым днем будет 30 декабря, а при отрицательных значениях будет показана предыдущая дата

Возвращаемое значение

Возвращает значение типа *Variant(Integer)* от 1 до 31, которое представляет день месяца.

Параметры

Date

Обязательный аргумент, представляющий собой любое выражение, оцениваемую как дату.

Пример

```
Day("мар 14 2014")          'Возвращает 14
Day("14 марта 2014")        'Возвращает 14
Day(Now)                    'Возвращает 31
Day(2094)                   'Возвращает 24
```

2.8.4.6. Функция Weekday

Weekday(Date,[FirstDayOfWeek])

Функция используется для получения номера дня (между 1 и 7) из указанной даты.

Возвращаемое значение

Возвращает значение типа *Variant(Integer)*, содержащее целое число (между 1 и 7), представляющее день недели.

Параметры

Date

Обязательный. Значение типа *Variant*, числовое выражение, строковое выражение или любая комбинация, позволяющая отобразить дату.

Firstdayofweek

Необязательный. Константа, указывающая первый день недели. Если этот аргумент опущен, считается, что неделя начинается с воскресенья.

Пример

```
Weekday ("18 марта 1848")        'Возвращает 7
Weekday («31/12/14")       'Возвращает 3
Weekday(Now, 2)            'Возвращает 5
```

2.8.4.7. Функция Timer

Timer

Функция Timer используется для получения числа секунд, прошедших с полуночи до текущего момента дня.

Возвращаемое значение

Возвращает значение типа *Single*, представляющее число секунд, прошедших после полуночи. Значение, возвращаемое таймером, ограничено числом секунд в сутках и обнуляется ровно в полночь

Параметры

Не содержит аргументов

Пример

```
t0 = Timer 'Возвращает 41675.17
                ' (Вызвано в 11:56)
```

2.8.4.8. Функция Time

Time

Функция Time используется для получения текущего времени по системным часам компьютера.

Возвращаемое значение

Возвращает значение типа *Variant(Date)*, содержащее текущее время по системным часам компьютера. Формат возвращаемого значения определяется национальными системными настройками.

Параметры

Не содержит аргументов

Пример

```
tv = Timer 'Возвращает 11:42:08
```

2.8.4.9. Функция Hour

Hour(Time)

Функция Hour используется для получения значения часов из выражения времени. Вне зависимости от формата выражения времени аргумента и системных установок функция Hour всегда возвращает время в 24-часовом формате.

Возвращаемое значение

Возвращает значение типа *Variant(Integer)* от 0 до 23, которое представляет часы в значении времени.

Параметры

Time

Обязательный аргумент - любое значение типа *Variant*, числовым выражением, строковым выражением или любым их сочетанием, которое представляет значение времени.

Пример

```
tv = Timer                    'Возвращает 11:42:08
h1 = Hour(tv)                 'Возвращает 11
h2 = Hour("10:18:44 PM") 'Возвращает 22
```

2.8.4.10. Функция Minute

Minute(Time)

Функция Minute используется для получения значения минут часа из выражения времени.

Возвращаемое значение

Возвращает значение типа *Variant(Integer)* от 0 до 59, содержащее число минут неполного часа.

Параметры

Time

Обязательный аргумент - любое значение типа *Variant*, числовым выражением, строковым выражением или любым их сочетанием, которое представляет значение времени.

Пример

```
td = now                          'Возвращает 11:43:28
m1 = Mimute (td)                  'Возвращает 43
m2 = Mimute ("10:18:44 PM")       'Возвращает 18
```

2.8.4.11. Функция Second

Second(Time)

Функция Second используется для получения значения секунд минуты из выражения времени.

Возвращаемое значение

Возвращает значение типа *Variant(Integer)* от 0 до 59, содержащее число секунд неполной минуты.

Параметры

Time

Обязательный аргумент - любое значение типа *Variant*, числовым выражением, строковым выражением или любым их сочетанием, которое представляет значение времени.

Пример

```
td = now                          'Возвращает 12:15:14
s1 = Second (td)                  'Возвращает 14
s2 = Second ("10:18:44 PM")       'Возвращает 44
```

2.8.5. Функции вывода и вывода информации

2.8.5.1. Функция InputBox

InputBox (*строка* [, *заголовок*] [, *по умолчанию*] [, *Xpos*] [, *Ypos*] [, *файл справки*, *контекст*])

Отображает приглашение в диалоговом окне, ожидает ввода текста или нажатия кнопки пользователем.

Возвращаемое значение

В результате действия функции InputBox возвращается строка с содержимым текстового поля, введенным пользователем.

Параметры

строка - обязательный аргумент, содержит строковое выражение, отображаемое в виде сообщения в диалоговом окне, максимальная длина - 1024 символа. Если аргумент состоит из нескольких строк, можно разделить строки с помощью знака возврата каретки Chr(13), символ перевода строки Chr(10) или константы vbcr.

заголовок - необязательный аргумент. Строковое выражение, отображаемое в строке заголовка окна. При отсутствии *заголовка* в строке заголовка помещается имя приложения.

по умолчанию - необязательный аргумент. Строковое выражение, отображаемое в текстовом поле в качестве ответа по умолчанию, если не было введено. Если параметр по умолчанию не введен, текстовое поле отображается пустым.

Xpos - необязательный аргумент. Числовое выражение, которое определяет в твипах расстояние по горизонтали от левого края диалогового окна до левой границы экрана. Если параметр не введен, диалоговое окно будет размещено по горизонтали по центру.

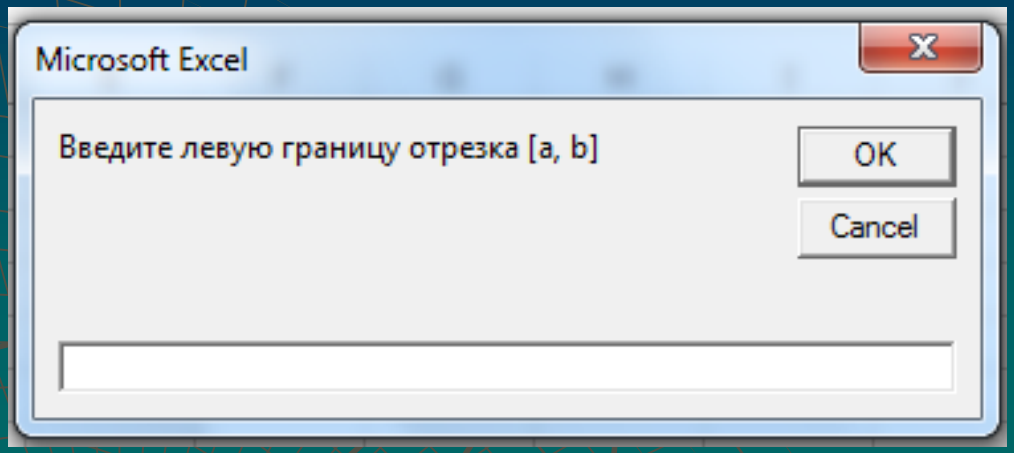
Ypos - необязательный аргумент. Числовое выражение, которое определяет в твипах расстояние по вертикали от верхнего края диалогового окна до верхней границы части экрана. Если параметр не введен, диалоговое окно будет размещено с отступом одной трети экрана по вертикали.

файл справки - необязательный аргумент. Строковое выражение, которое определяет файл справки, используемый для предоставления контекстной справки для диалогового окна. Если этот параметр указан *helpfileконтекста* также должен быть предоставлен.

контекст - необязательный аргумент. Числовое выражение, номер контекста справки, назначенных соответствующему разделу справки автора справки.

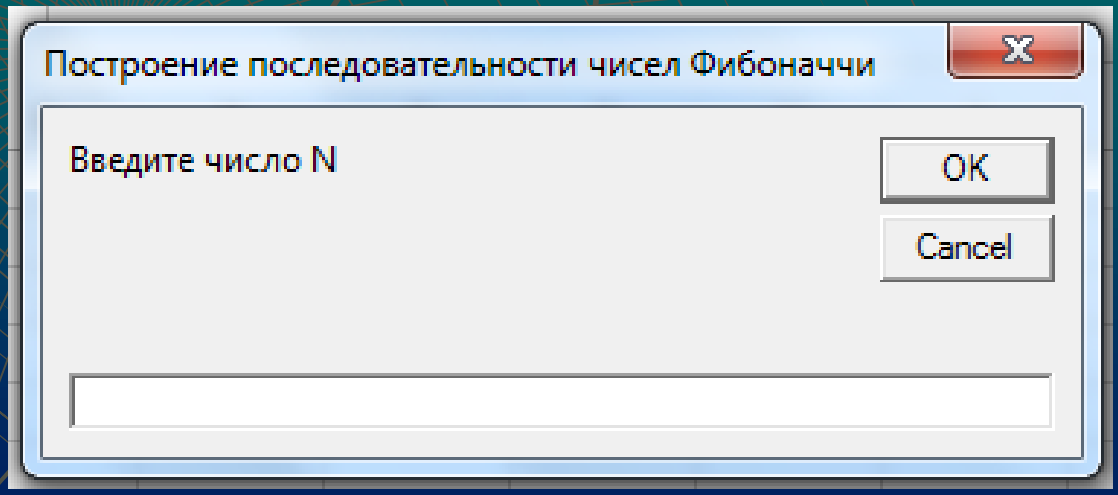
Пример

```
Dim a As Single, b As Single, epsilon As Single
a = InputBox("Введите левую границу отрезка [a, b]")
b = InputBox("Введите правую границу отрезка [a, b]")
epsilon = InputBox("Введите погрешность")
```



Пример

```
  Dim N As Long
  N = InputBox("Введите число N ", _
"Построение последовательности чисел Фибоначчи, не превышающих число N")
```



2.8.5.2. Функция MsgBox

MsgBox (*строка* [, *кнопки*] [, *заголовок*] [, *файл справки*, *контекст*])

Отображает сообщение в диалоговом окне, ожидает нажатия кнопки. и

Возвращаемое значение

В результате действия функции MsgBox возвращается целое число, указывающее, какая кнопка была нажата.

Параметры

строка - обязательный аргумент, содержит строковое выражение, отображаемое в виде сообщения в диалоговом окне, максимальная длина - 1024 символа. Если аргумент состоит из нескольких строк, можно разделить строки с помощью знака возврата каретки Chr(13), символ перевода строки Chr(10) или константы vbcr.

кнопки - необязательный аргумент, сожержит числовое выражение, которое представляет собой сумму значений, задающих номер и тип кнопок для отображения, стиль значка для использования, кнопки по умолчанию и модальность окна сообщения. Если этот параметр опущен, значение по умолчанию для *кнопок* равно 0.

заголовок - необязательный аргумент. Строковое выражение, отображаемое в строке заголовка окна. При отсутствии *заголовка* в строке заголовка помещается имя приложения.

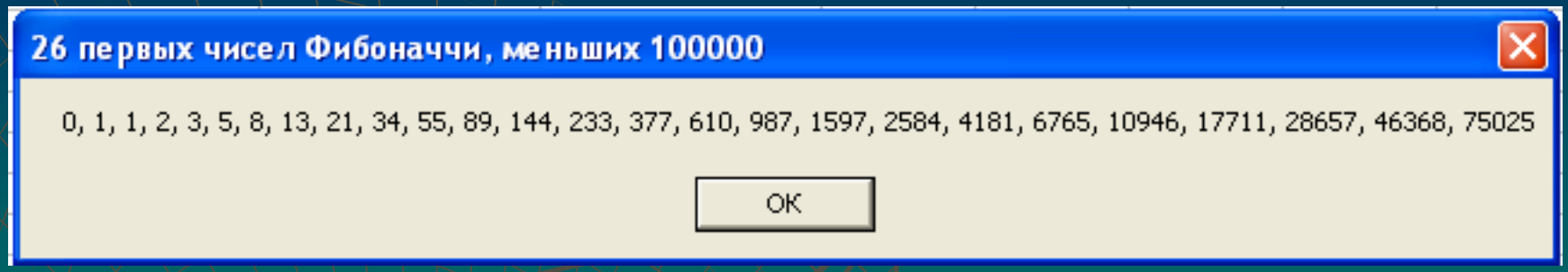
файл справки - необязательный аргумент. Строковое выражение, которое определяет файл справки, используемый для предоставления контекстной справки для диалогового окна. Если этот параметр указан *helpfileконтекста* также должен быть предоставлен.

контекст - необязательный аргумент. Содержит числовое выражение, номер контекста справки, назначенных соответствующему разделу справки автора справки.

Константы	Значение	Описание
vbOKOnly	0	Отображение только кнопки ОК
vbOKCancel	1	Отображение кнопки ОК и Отмена
vbAbortRetryIgnore	2	Отображаются кнопки Прервать, Повторить и Пропустить
vbYesNoCancel	3	Отображаются кнопки Да, Нети Отмена
vbYesNo	4	Отображаются кнопки Да и Нет
vbRetryCancel	5	Отображаются кнопки Повторить и Отмена
vbExclamation	48	Отображение Сообщение с предупреждением

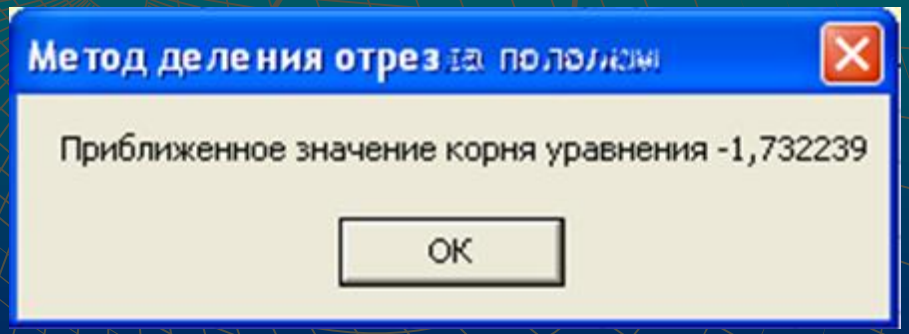
Пример

```
MsgBox Left(t, Len(t) - 1), , k & " первых чисел Фибоначчи, меньших" & N
```



Пример

```
MsgBox "Приближенное значение корня уравнения " & c, , _
       "Метод деления отрезка пополам"
```



Глава 3. Организация ветвления программ

3.1. Организация ветвлений в VBA.

3.2. Инструкция If.

3.2.1. Определение инструкции If.

3.2.2. Использование инструкции If.

3.2.3. Обход.

3.2.4. Вложенные условные структуры.

3.3. Инструкция Select Case.

3.3.1. Определение инструкции Select Case.

3.3.2. Примеры использования Select Case.

3.3.3. Использование операторов сравнения.

3.4. Инструкция GOTO.

3.5. Использование функции MSGBox для организации ветвления.

3.1. Организация ветвлений в VBA

В программных приложениях, имеющих практическую значимость, в подавляющем большинстве случаев используются ***нелинейные алгоритмы***.

К таким алгоритмам относятся ***структуры ветвления***, порядок действий или вычислений в которых зависит от определенных условий, например, от исходных данных, промежуточных результатов, полученных на предыдущих шагах выполнения программы.

Под ***ветвлением*** будем понимать алгоритмическую структуру, в которой в зависимости от результата проверки поставленного условия осуществляется выбор только одной из двух или нескольких альтернативных ветвей алгоритма.

Для организации ветвления в среде Visual Basic for Application (VBA) предусмотрены две алгоритмические конструкции ***If*** и ***Select Case***, а для осуществления безусловного перехода инструкция ***GoTo***.

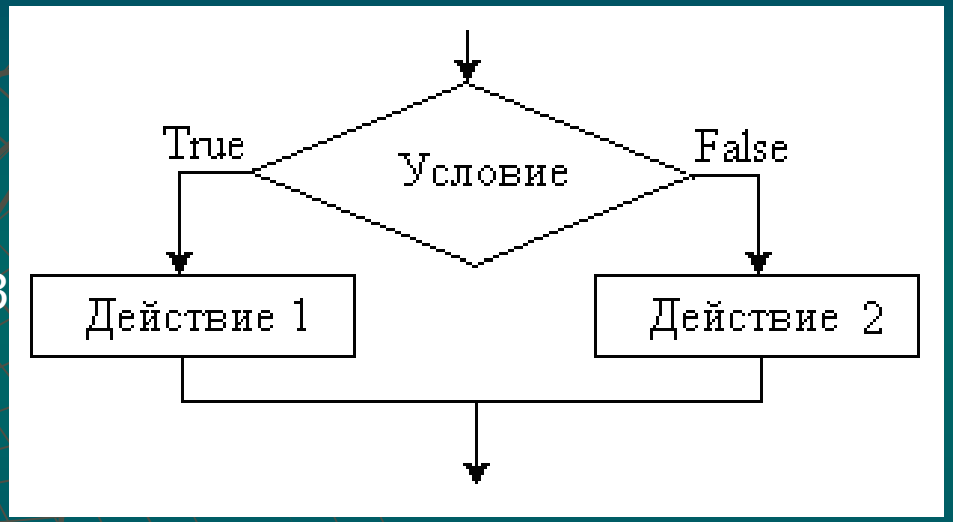
3.2. Инструкция *If*

3.2.1. Определение инструкции *If*

Инструкция *If* используется для организации ветвления с выбором одного из двух альтернативных путей продолжения выполнения алгоритма.

Инструкция *If* имеет следующий синтаксис:

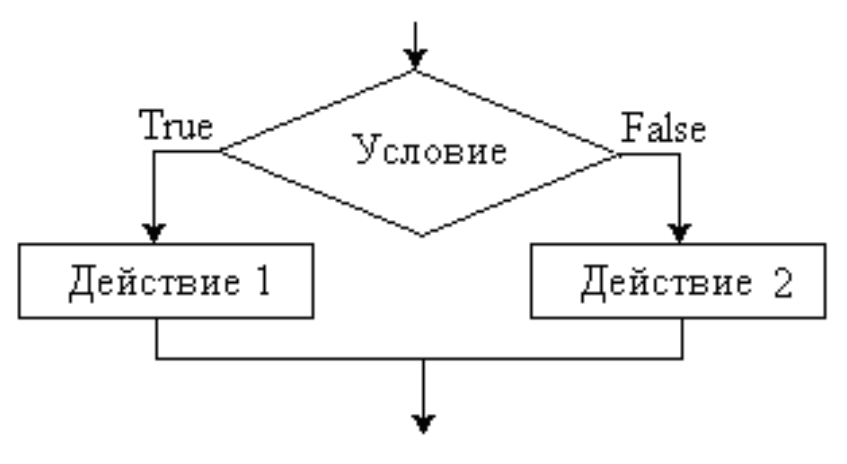
```
If условие Then
   действие 1
  Else
   действие 2
End If
```



В данной записи *If*, *Then*, *Else*, *End If* являются служебными словами. Условие может содержать любое логическое выражение, в результате вычислений принимающее значения *True* или *False*.

Действие 1 и Действие 2 могут быть представлены отдельными инструкциями или сложными алгоритмическими структурами.

Для сравнения переменных в условных выражениях используют *операции отношения*: =, <, >, <=, >=, <>.



Условные выражения составляют с применением логических операций ***not*** (логическое отрицание), ***and*** (логическое «и», конъюнкция), ***or*** (логическое «или», дизъюнкция), ***xor*** (логическое исключение), ***imp*** (импликация), ***eqv*** (эквиваленция).

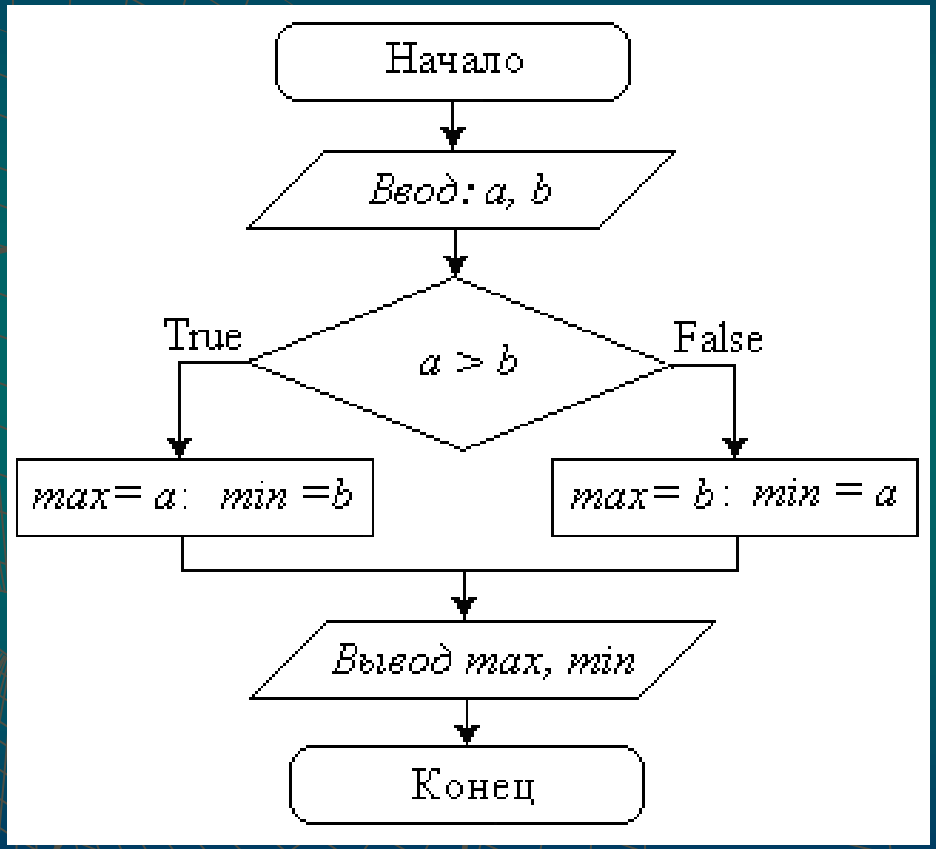
В языке VBA ***приоритет операций отношения больше, чем у логических операций***, поэтому составные части сложного логического выражения в скобки не заключаются

Допустим, нужно проверить, принадлежит ли переменная ***x*** полуинтервалу (***a***, ***b***]? Тогда инструкция *If* будет иметь вид:

```
if x > a and x <= b then …
```

3.2.2. Использование инструкции *If*

Пример. Для нахождения максимального и минимального значений из двух введенных пользователем чисел можно использовать следующий алгоритм.



Здесь мы видим, что если ***a*** будет больше, чем ***b***, то переменной ***max*** присваивается значение переменной ***a***, а переменной ***min*** – значение переменной ***b***. В противном случае *max* = ***b***, *min* = ***a***.

Данный алгоритм может быть реализован следующим программным кодом.

```
Sub Выбор_максимального_и_минимального_значений()
  Dim a As Single, b As Single
  a = InputBox("Введите число a", "Ввод начальных значений", "0")
  b = InputBox("Введите число b", "Ввод начальных значений", "0")
  If a > b Then
      Max = a
      Min = b
    Else
      Max = b
      Min = a
  End If
  MsgBox "Максимальное значение " & Max & ", минимальное " & Min
End Sub
```

В приведенном программном коде после объявления переменных ***a*** и ***b*** и ввода их значений используется структура: *If* ... *Then*

...

Else

...

End If

Однако структуру *If* в VBA можно сделать компактней, объединив все инструкции ветвей *Then* и *Else* в составные операторы. Перечисляемые в составном операторе инструкции разделяются двоеточием.

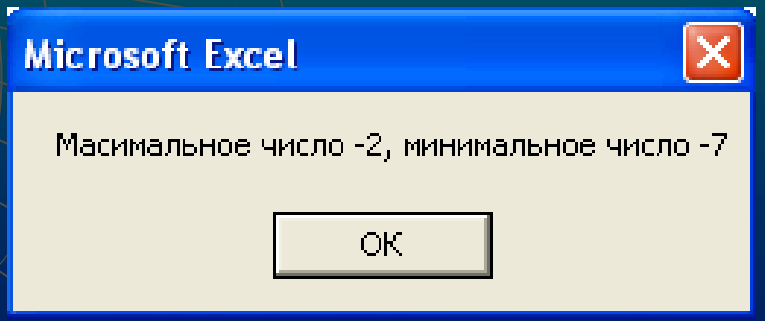
```
If a > b Then
    Max = a: Min = b
  Else
    Max = b: Min = a
End If
```

Кроме того, структуру *If* в нашем примере можно записать в одну строку.

```
If a > b Then Max = a: Min = b Else Max = b: Min = a
```

Таким образом, в коде программы мы экономим целых шесть строк. В последней записи обозначение конца инструкции *If* не требуется.

На рисунке представлен результат выполнения программы при введенных значениях **a** = -7, **b** = -2.



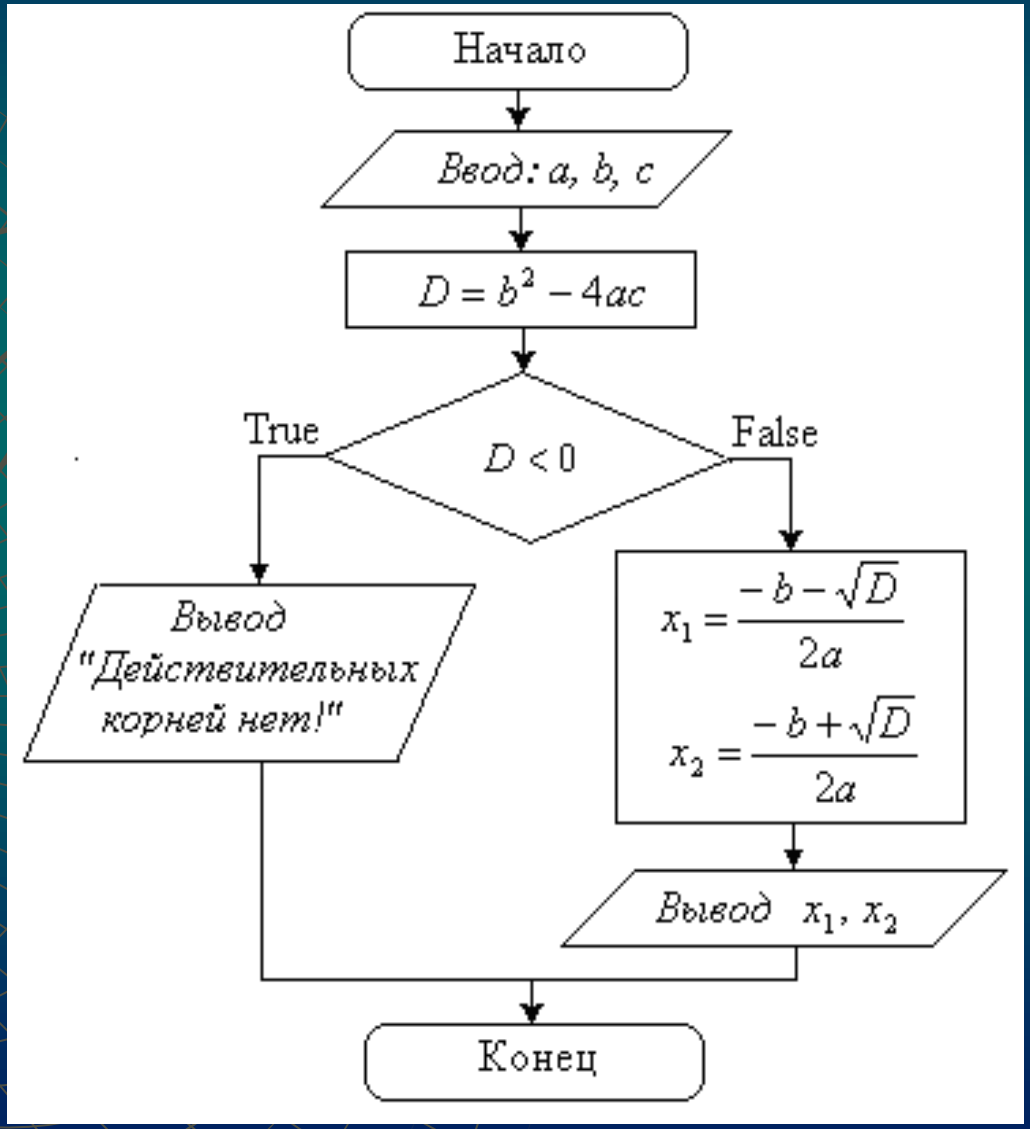
Пример. Написать программу нахождения действительных корней квадратного уравнения $ax^2 + bx + c = 0$.

Для решения задачи по формуле $D = b^2 - 4ac$ найдем дискриминант.

Если полученное значение $D < 0$, то это означает, что действительных корней нет, в противном случае корни уравнения определяются по формулам:

$$x_1 = \frac{-b - \sqrt{D}}{2a} \qquad x_2 = \frac{-b + \sqrt{D}}{2a}$$

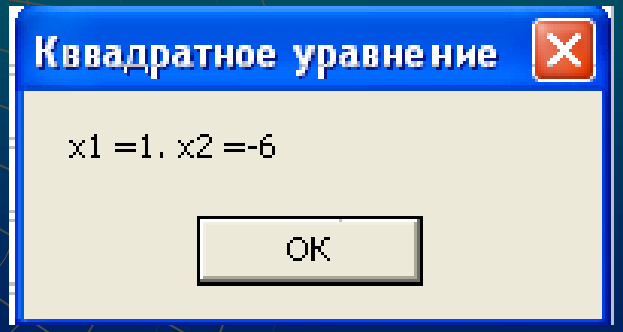
Приведем структурную блок-схему решения данной задачи::



Предложенному алгоритму соответствует программный код:

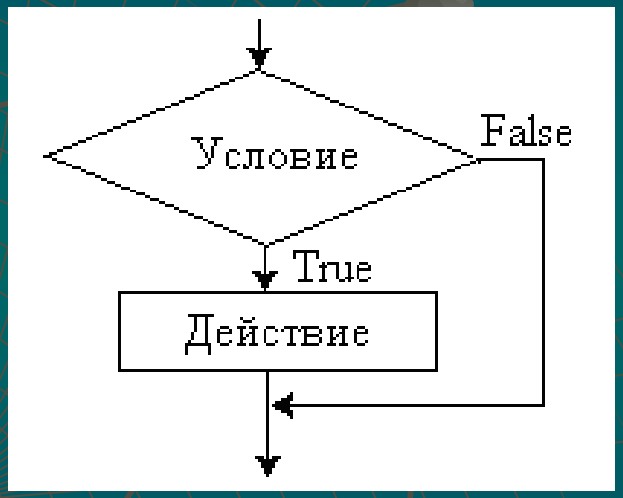
```
Sub Квадратное_уравнение()
'Вычисление корней квадратного уравнения
  Dim a As Double, b As Double, c As Double
  Dim d As Double, x1 As Double, x2 As Double
  a = InputBox("Введите коэффициент a")
  b = InputBox(" Введите коэффициент b")
  c = InputBox(" Введите коэффициент c")
  d = b ^ 2 - 4 * a * c      'Вычисление дискриминанта
  If d < 0 Then
      MsgBox "Действительных корней нет"
    Else
      x1 = (-b - Sqr(d)) / (2 * a)
      x2 = (-b + Sqr(d)) / (2 * a)
      MsgBox "x1 =" & x1 & ". x2 =" & x2
  End If
End Sub
```

На рисунке представлен результат определения корней квадратного уравнения $-2x^2 - 10x + 12 = 0$.



3.2.3. Обход

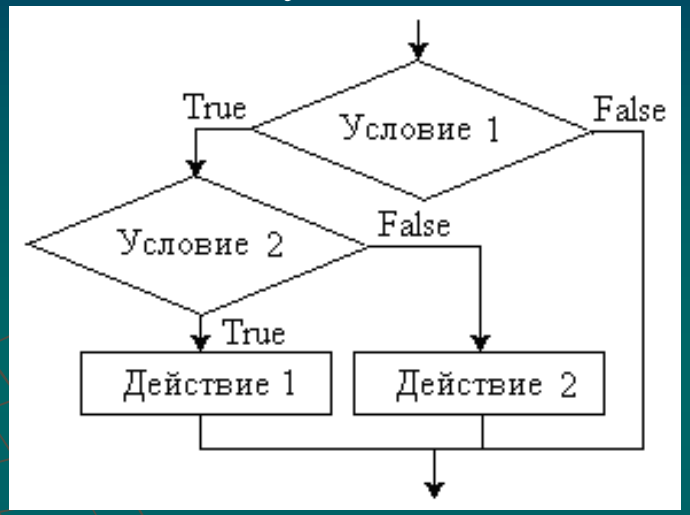
Альтернативная ветвь *Else* в инструкции *If* может отсутствовать, если в ней нет необходимости. На рисунке представлена структурная схема ветвления без альтернативной ветви. Обычно такую алгоритмическую структуру называют ***обходом***.



В случае ***невыполнения логического условия*** управление передается оператору, стоящему в программе после инструкции *if*.

3.2.4. Вложенные условные структуры

Условные структуры могут быть вложены друг в друга. При вложениях условных структур всегда действует правило: альтернатива *Else* считается принадлежащей ближайшей инструкции *If*, имеющей ветвь *Else*. Кроме того, можно ориентироваться и по служебным словам *End IF*, обозначающим конец инструкции.

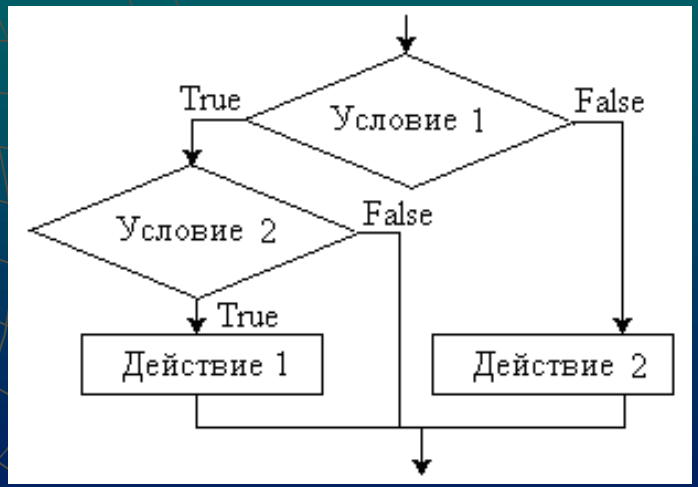


Например, в записи

If условие 1 *Then*
If условие 2 *Then*
действие 1
Else действие 2
End If
End If

действие 2 относится к инструкции *If* с условием 2, а в конструкции

If условие 1 *Then*
If условие 2 *Then*
действие 1
End If
Else действие 2
End If



действие 2 принадлежит оператору *If* с условием 1.

Пример. Вывести общее уравнение прямой на плоскости, проходящей через точки $P_1(x_1, y_1)$ и $P_2(x_2, y_2)$.

Из курса математики мы знаем каноническое уравнение прямой, проходящей через две точки:

$$\frac{x-x_1}{x_2-x_1}=\frac{y-y_1}{y_2-y_1}$$

Умножим обе части уравнения на произведение $(x_2-x_1)(y_2-y_1)$

$$(x-x_1)(y_2-y_1)=(y-y_1)(x_2-x_1)$$

Раскроем скобки и перенесем полученные выражения из правой части уравнения в левую его часть

$$xy_2-xy_1-x_1y_2+x_1y_1-yx_2+yx_1+y_1x_2-y_1x_1=0$$

После приведения подобных слагаемых получим

$$(y_2-y_1)x+(x_1-x_2)y-x_1y_2+x_2y_1=0$$

Из полученного уравнения можно выписать коэффициенты общего уравнения прямой $Ax + By + C = 0$:

$$A=y_2-y_1 \qquad B=x_1-x_2 \qquad C=-x_1y_2+x_2y_1$$

В случае вертикальных прямых, если $x_1 = x_2$, коэффициенты будут равны:

$$A=1 \qquad B=0 \qquad C=-x_1$$

Составим программный код полученного алгоритма. Объявим переменные и введем значения координат точек $P_1(x_1, y_1)$ и $P_2(x_2, y_2)$.

```
Sub Общее_уравнение_прямой_проходящей_через_две_точки()
  Dim x1 As Single, y1 As Single, x2 As Single, y2 As Single
  Dim A As Single, B As Single, C  As Single
                        'Ввод координат точек P1 и P2
  x1 = Cells(2, 2)
  y1 = Cells(2, 4)
  x2 = Cells(3, 2)
  y2 = Cells(3, 4)
```

Согласно полученным формулам рассчитаем коэффициенты A, B и C общего уравнения прямой.

```
                    'Определение коэффициентов уравнения Ax + By +C = 0
A = y2 - y1
B = x1 - x2
C = -x1 * y2 + x2 * y1
If x1 = x2 Then A = 1: B = 0: C = -x1
```

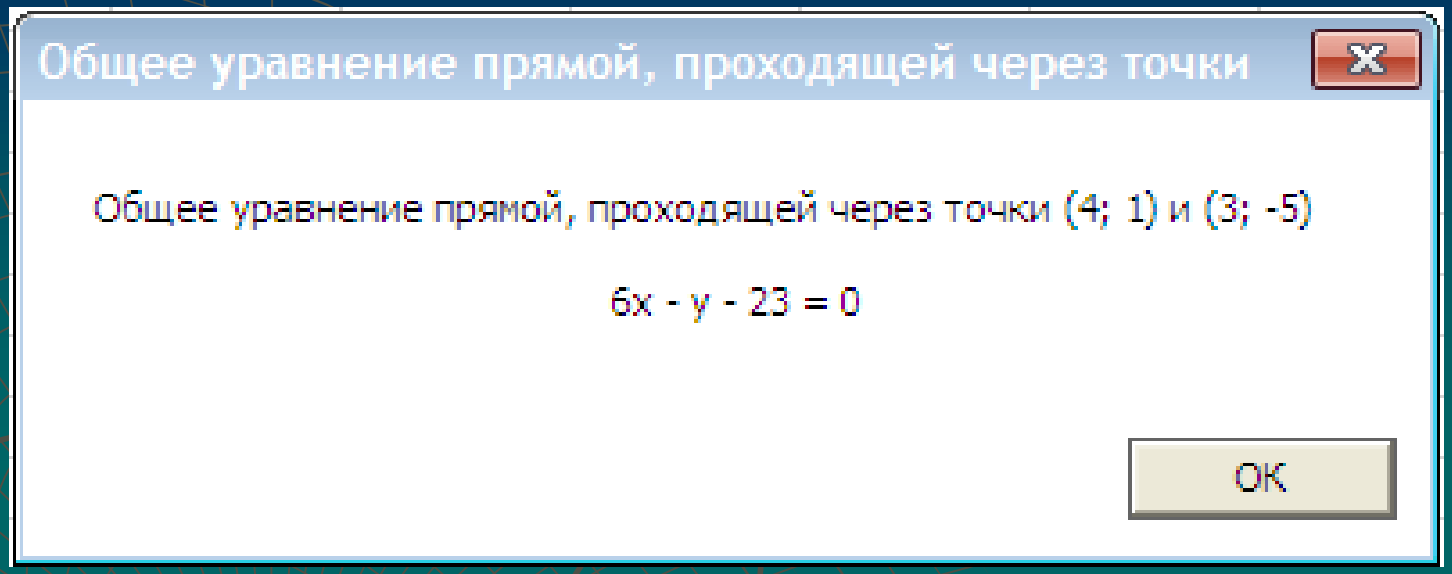
Теперь основная задача заключается в создании записи общего уравнения прямой $Ax + By + C = 0$.

Если выписать коэффициенты перед переменными, не исследуя их значений, то можно, например, для прямой, совпадающей с осью абсцисс, вместо уравнения $y = 0$, получить выражение $0x + 1y + 0 = 0$.

Для проверки всех вариантов значений коэффициентов *A*, *B* и *C* будем использовать вложенные инструкции *If*, а также *If* без ветвей *Else*.

```
                    'Запись общего уравнения Ax + By +C = 0
  If A = 0 Then
      If B < 0 Then B = -B: C = -C
      If B = 1 Then t = t + "y"
      If B <> 1 And B <> 0 Then t = t + Str(B) + "y"
    Else
      If A < 0 Then A = -A: B = -B: C = -C
      If A = 1 Then t = "x"
      If Abs(A) <> 1 Then t = Str(A) + "x"
      If B < 0 Then t = t + " -"
      If B > 0 Then t = t + " +"
      If Abs(B) = 1 Then t = t + " y"
      If Abs(B) <> 1 And B <> 0 Then t = t + Str(Abs(B)) + "y"
  End If
  If C < 0 Then t = t + " -" + Str(Abs(C))
  If C > 0 Then t = t + " +" + Str(Abs(C))
  t = t + " = 0"
  MsgBox "   Общее уравнение прямой, проходящей через точки " & _
         "(" & x1 & "; " & y1 & ") и (" & x2 & "; " & y2 & ")" & vbCr & vbCr & _
         String(50, " ") & t, , "Общее уравнение прямой, проходящей через точки"
End Sub
```

На рисунке представлен результат вывода общего уравнения прямой, проходящей через точки $P_1(4; 1)$ и $P_2(3; -5)$.



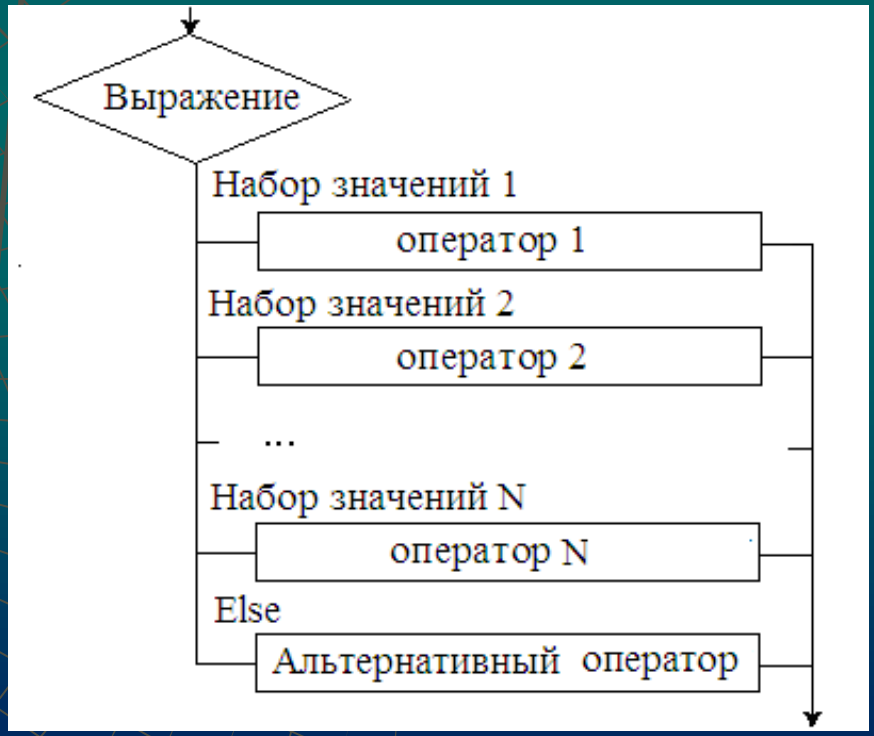
3.3. Инструкция *Select Case*

3.3.1. Определение инструкции Select Case

Инструкция выбора *Select Case* используется в тех случаях, когда в зависимости от ***значений исследуемого выражения*** необходимо выполнить те или иные операторы.

Инструкция *Select Case* имеет следующий синтаксис:

```
Select Case выражение
  Case набор значений 1
    оператор 1
  Case набор значений 2
    оператор 2
…
  Case набор значений N
    оператор N
  Case else
    альтернативный оператор
End Select
```



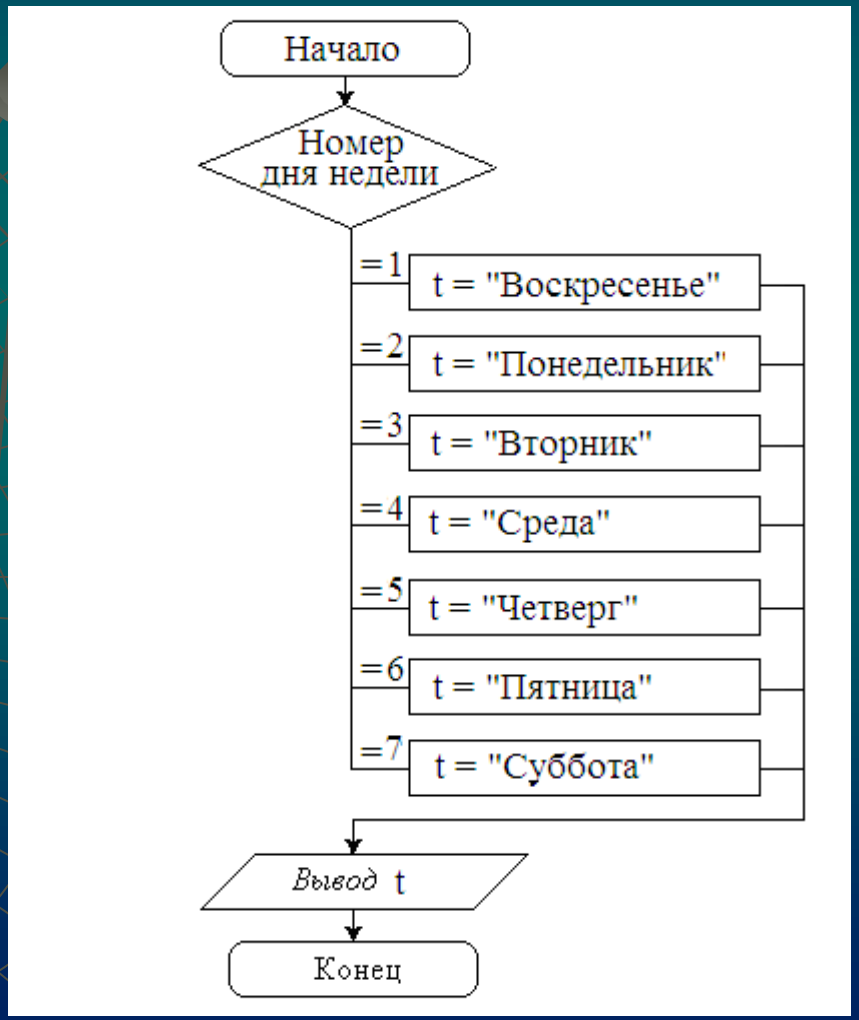
Наборами значений могут служить перечисляемые множества чисел, символов, дат, а также выражения, определяющие интервалы значений заданных типов. Значения в каждом наборе должны быть уникальны, то есть они не должны пересекаться в разных наборах.

Инструкция *Select Case* допускает использование составных операторов.

3.2. Примеры использования *Select Case*

Пример. Вывести на экран название дня недели, соответствующее текущей дате.

На рисунке представлена структурная схема решения данной задачи. Алгоритм заключается в выборе варианта для значения номера дня недели и присвоении переменной ***t*** соответствующего значения строковой записи.

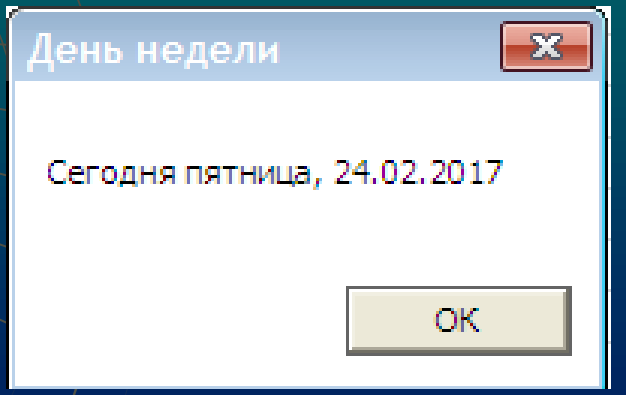


Для решения задачи воспользуемся функцией *Weekday*(*Date*), позволяющей вычислить номер дня недели текущей даты, с условием, что первый день недели – воскресенье.

```
Sub День_недели()
'Программа выводит день недели на русском языке
  Select Case Weekday(Date)
    Case 1: t = "воскресенье"
    Case 2: t = "понедельник"
    Case 3: t = "вторник"
    Case 4: t = "среда"
    Case 5: t = "четверг"
    Case 6: t = "пятница"
    Case 7: t = "суббота"
  End Select
  MsgBox "Сегодня " & t & ", " & Date, , "День недели"
End Sub
```

В предложенной записи инструкции *Select Case* отсутствует ветвь *Case Else*. Это объясняется тем, что выражение *Weekday*(*Date*) может принимать только одно из значений 1, 2, 3, 4, 5, 6 или 7.

На рисунке представлен результат вывода дня недели.



Пример. По введенному пользователем символу определить, является ли этот символ цифрой, знаком арифметической операции, знаком препинания, заглавной, строчной буквой латинского или русского алфавитов.

Воспользуемся ***ASCII*** кодами (*American Standard Code for Information Interchange*) основных вводимых с клавиатуры символов:

	30	40	50	60	70	80	90	100	110	120		190	200	210	220	230	240	250
0		(	2	<	F	P	Z	d	n	x			И	Т	Ь	ж	р	ъ
1		)	3	=	G	Q	[	e	o	y			Й	У	Э	з	с	ы
2		*	4	>	H	R	\	f	p	z		А	К	Ф	Ю	и	т	ь
3	!	+	5	?	I	S	]	g	q	{		Б	Л	Х	Я	й	у	э
4	"	,	6	@	J	T	^	h	r	|		В	М	Ц	а	к	ф	ю
5	#	-	7	A	K	U	_	i	s	}		Г	Н	Ч	б	л	х	я
6	$	.	8	B	L	V	`	j	t			Д	О	Ш	в	м	ц	
7	%	/	9	C	M	W	a	k	u			Е	П	Щ	г	н	ч	
8	&	0	:	D	N	X	b	l	v			Ж	Р	Ъ	д	о	ш	
9		1	;	E	O	Y	c	m	w			З	С	Ы	е	п	щ	

В таблице каждому символу сопоставлен уникальный код. Таблица ***ACSII*** колов приведена не полностью. В ней отсутствуют управляющие символы (коды от нуля до 32) и символы с кодами от 126 до 191, которые в подавляющем большинстве отсутствуют на клавиатуре.

Для определения категории вводимого с клавиатуры символа ***s*** воспользуемся структурой *Select Case*, в которой будем анализировать выражение *ASC*(*s*).

В случае нажатия пользователем на клавишу, соответствующую цифре, функция *ASC*(*s*) возвратит значения кодов от 48 до 57.

Коды 42, 43, 45, 47, 92, 94 соответствуют символам «*», «+», «-», «/», «\»,«^» соответственно.

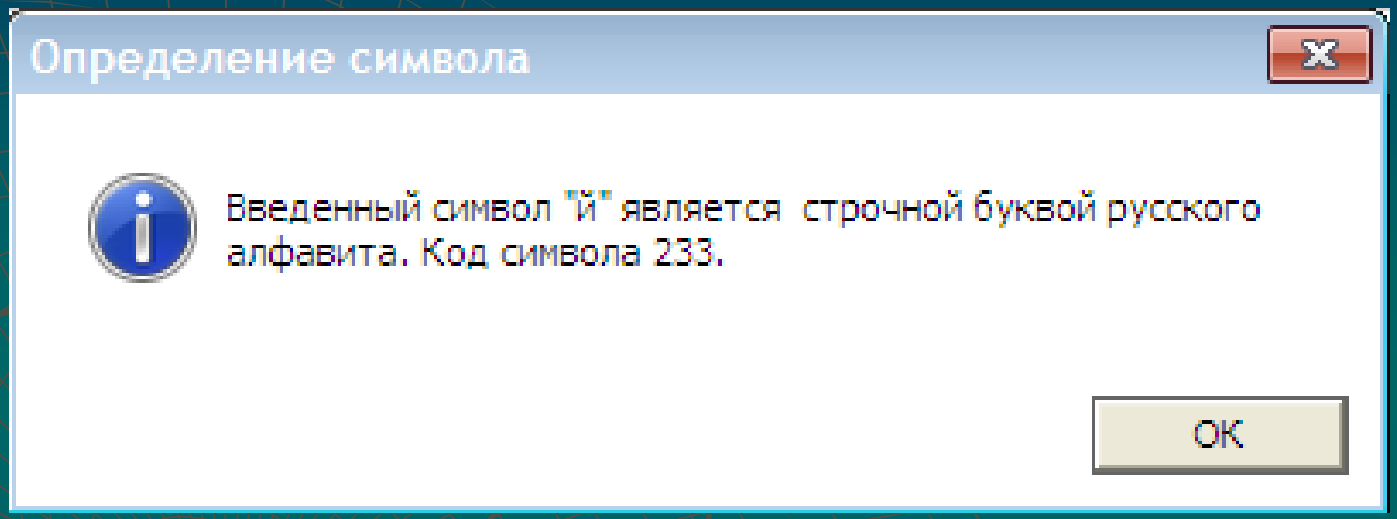
Если возвращено значение 33, 34, 39, 44, 46, 58, 59 или 63, то такой символ определим как знак пунктуации.

Аналогично рассматриваем интервалы 65 – 90, 97 – 122, 192 – 223 и 224 –255 для определения заглавных латинских, строчных латинских, заглавных русских и строчных русских букв.

Так как в программе анализировались коды не всех выводимых символов, то для удобства пользователя необходимо использование альтернативного оператора, который заключается в выводе сообщения «Символ не определен».

```
Sub Определение_символа()
  Dim s As String
  s = InputBox("Введите символ", "Ввод символа")
  t = "Введенный символ " + """ + s + """ + " является "
  Select Case Asc(s)
    Case 48 To 57
      t = t + "цифрой."
    Case 42, 43, 45, 47, 92, 94
      t = t + "знаком арифметической операции."
    Case 33, 34, 39, 44, 46, 58, 59, 63
      t = t + "знаком пунктуации."
    Case 65 To 90
      t = t + "заглавной буквой латинского алфавита."
    Case 97 To 122
      t = t + "строчной буквой латинского алфавита."
    Case 192 To 223
      t = t + "заглавной буквой русского алфавита."
    Case 224 To 255
      t = t + " строчной буквой русского алфавита."
    Case Else
      t = "Символ не определен."
  End Select
  t = t + " Код символа" + Str(Asc(s)) + "."
  MsgBox t, vbInformation, "Определение символа"
End Sub
```

На рисунке представлен результат определения введенного с клавиатуры символа и его код.



3.3.3. Использование операторов сравнения

В приведенных примерах наборы значений содержали конечные дискретные множества. Здесь мы использовали перечисление значений, как, например, в записи

```
Case 42, 43, 45, 47, 92, 94
```

либо конечные сегменты, включающие концы промежутков

```
Case 48 To 57
```

На практике программист часто сталкивается со случаем, когда для проверяемых выражений в инструкции *Select Case* необходимо рассматривать интервалы (***a***, ***b***), полуинтервалы [***a***, ***b***) и (***a***, ***b***], интервалы с бесконечными границами (-∞, ***a***), (***a***, +∞), а также объединения различных промежутков.

В таких случаях набор значений определяется с использованием операторов сравнения >, <, >=, <=.

Пример. Найти значение функции, заданной аналитически:

$$y(x)=\begin{cases} y=9, & x\le -3; \\ y=x^2, & -3<x\le 1; \\ y=x, & x>1. \end{cases}$$

Заметим, что функция ***y*(*x*)** имеет три промежутка, на которых функция определяется различными аналитическими выражениями. В коде программы удобно использовать инструкцию *Select Case* с тремя наборами значений аргумента ***x***.

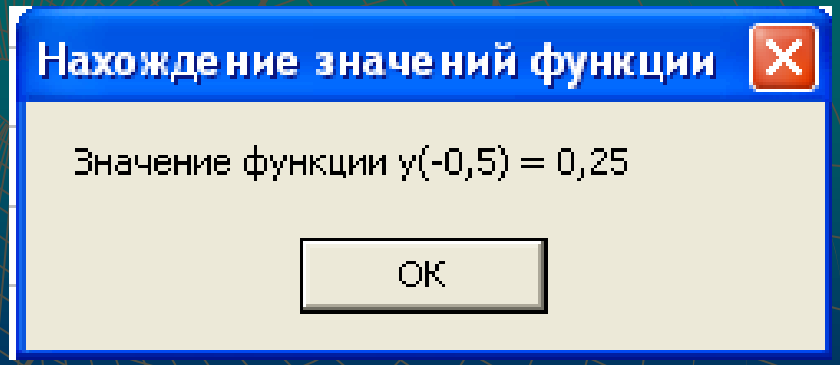
```
Sub Нахождене_значений_функции()
  Dim x As Single
  x = InputBox("Введите значение аргумента x", "Ввод данных")
  Select Case x
    Case Is <= -3
      y = 9
    Case Is <= 1
      y = x ^ 2
    Case Is > 1
      y = x
  End Select
  MsgBox "Значение функции y(" & x & ") = " & y, , _
         "Нахождение значений функции"
End Sub
```

Следует отметить, что при использовании операторов сравнения >, <, >=, <=, редактор VBA автоматически вставляет перед ними оператор *Is*.

```
Select Case x
  Case Is <= -3
    y = 9
  Case Is <= 1
    y = x ^ 2
  Case Is > 1
    y = x
End Select
```

В коде программы второй набор значений аргумента (-3; 1] задан выражением *Is* <= 1. На самом деле, второй набор будет использоваться только в том случае, если в первом наборе (--∞; ***-3***] значение переменной ***x*** не будет найдено.

Ниже представлено окно вывода значения функции ***y***(-0,5).



3.4. Инструкция *GOTO*

Помимо операторов условного перехода существует также оператор безусловного перехода goto.

Формат оператора: **goto** *метка*

Оператор goto переходит при выполнении программы к определенной строке программы, отмеченной ***номером*** или ***меткой***.

Номер строки или м*етка* должны находиться в начале строки.

Имя метки может быть любым идентификатором.

```
...
7: Оператор 1
...
if условие Then goto 7 else goto m12;
...
m12: Оператор 2
...
```

Само понятие структурного программирования и общепринятый стиль программирования на структурных языках **НЕ ПРИВЕТСТВУЕТ** применение меток и операторов перехода в программах. Это затрудняет понимание программы, кроме того, применение меток отрицательно сказывается на эффективности генерируемого кода.

3.5. Использование функции *MSGBox* для организации ветвления.

Функция ***MSGBox*** ***о***тображает сообщение в диалоговом окне, ожидает нажатия кнопки и определяет нажатую кнопку.

Var = ***MSGBox*** (*Prompt*, *Buttons*, *Title*)

Prompt. Выражение типа *String*, отображаемое в диалоговом окне в виде сообщения. Максимальная длина параметра *Prompt* составляет примерно 1024 знака. Если *Prompt* состоит из нескольких строк, то можно разделить строки с помощью знака VbCr.

Buttons Числовое выражение, являющееся суммой значений, задающих номер и тип отображаемых кнопок, стиль используемого значка, тип кнопки по умолчанию и признак модальности окна сообщения. Если параметр *Buttons* опущен, то по умолчанию используется нулевое значение.

Title Выражение типа *String*, отображаемое в строке заголовка диалогового окна. Если аргумент *Title* опущен, то в строку заголовка помещается имя приложения.

Числовой Код информационного значка + Числовой Код определенной кнопки определяется с помощью таблицы значений кодов:

ЧитКод1	Пиктограм	ЧитКод2	Набор Кнопок
16		0	ОК
32		1	ОК Отмена
48		2	Стоп Повтор Пропустить
64		3	Да Нет Отмена
		4	Да Нет
		5	Повтор Отмена

```
MsgBox("проверка" , 64 + 4,  "использование MsgBox при организации ветвления")
```

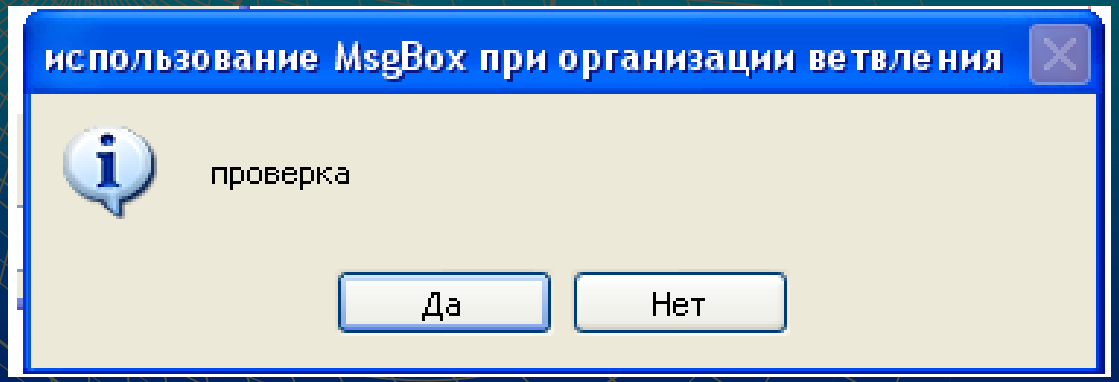


Таблица кодов кнопок:

Кнопка	Ее Номер
OK	1
Отмена	2
Стоп	3
Повтор	4
Пропустить	5
Да	6
Нет	7

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click

    Dim kod As Short

    kod = MsgBox("проверка", 64 + 4, "использование MsgBox при организации
    ветвления")    ' переменной kod присваивается код нажатой кнопки в окне
    сообщения

    If kod = 6 Then  MsgBox("Нажата кнопка 'Да'", , "результат выбора") :
    Label1.Text = "Нажата кнопка 'Да' "

    If kod = 7 Then  MsgBox("Нажата кнопка 'Нет'", , "результат выбора") :
    Label1.Text = "Нажата кнопка 'Нет' "

End Sub
```

Глава 4. Циклы

4.1. Инструкция While ... Wend
4.2. Оператор цикла Do
4.3. Цикл с параметром For ... Next
4.4. Использование циклов для решения задач
4.5. Вложенные циклы
4.6. Работа с таблицами
4.7. Ввод и обработка списков

4. Циклы

Циклический процесс или ***цикл*** – это повторение одних и тех же действий и операций в ходе выполнения программы.

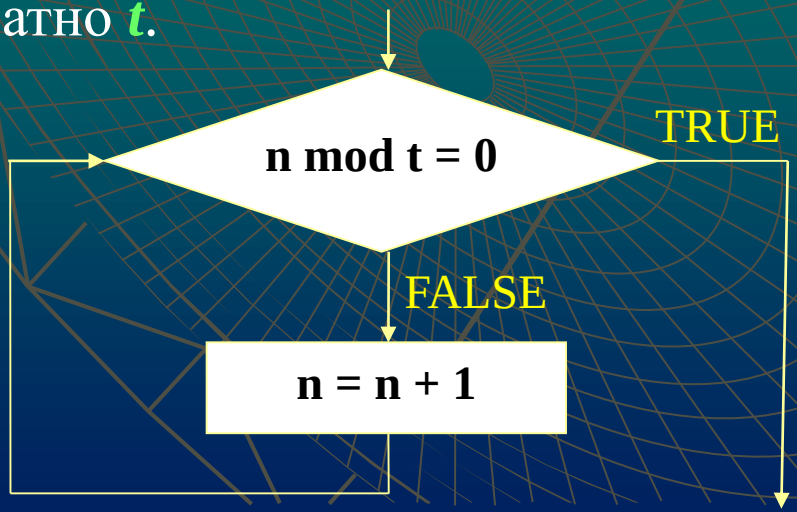
Последовательность действий, которые повторяются в цикле, называют ***телом цикла***.

Один проход цикла называют ***итерацией***.

Переменные, которые изменяются внутри цикла и влияют на его окончание, называются ***параметрами цикла***.

Пример.

Данная алгоритмическая структура используется для увеличения значения натурального числа ***n*** числа, до тех пор, пока ***n*** не будет кратно ***t***.

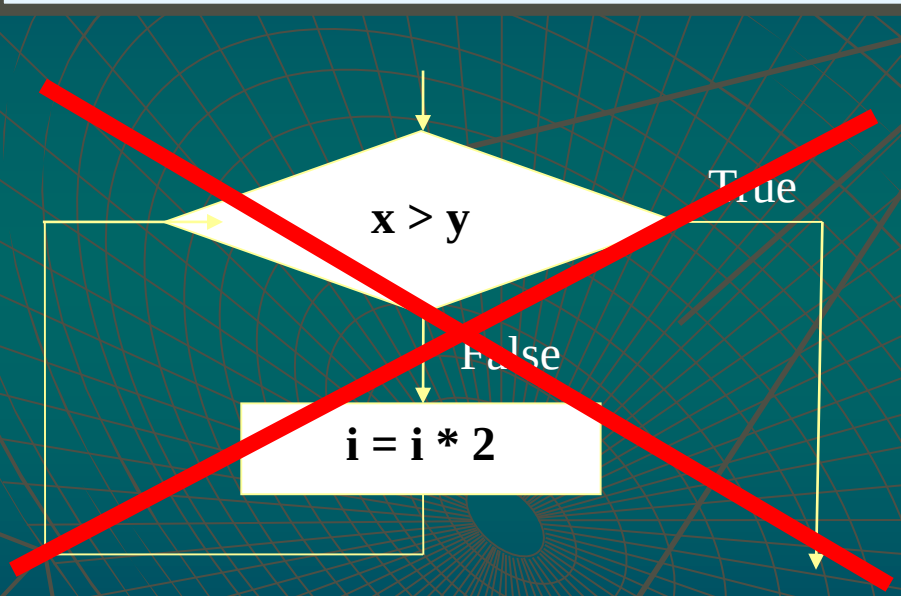


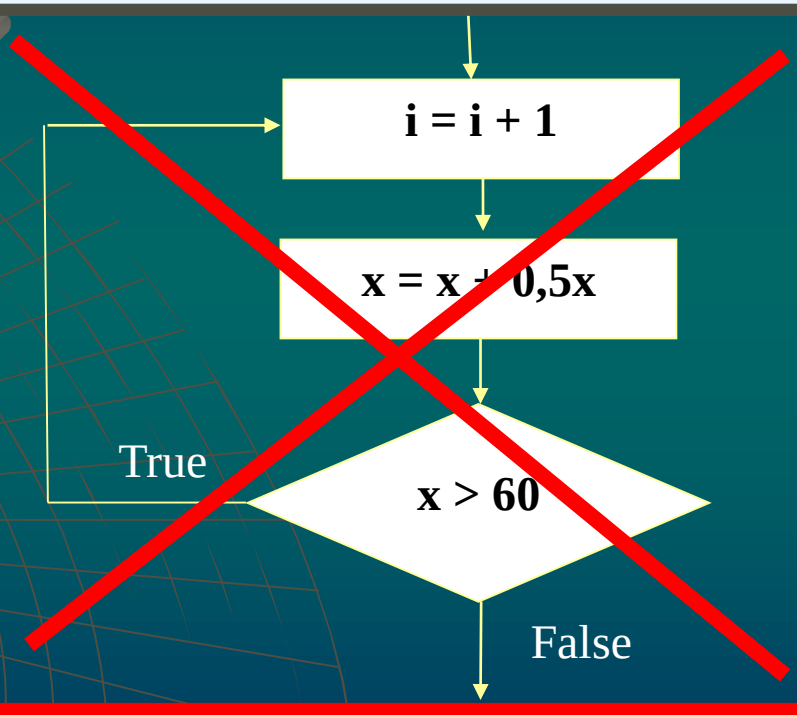
При начальном значении параметра цикла ***n***=58, а ***t***=7 количество итераций этого цикла равно 5.

При написании циклических алгоритмов необходимо придерживаться следующих правил:

1) Чтобы цикл мог когда-нибудь закончиться, **содержимое тела цикла должно обязательно влиять на условие цикла**.

2) Условие должно состоять из **корректных выражений и значений, определенных еще до первого выполнения тела цикла**.





В первом алгоритме вычисления в теле цикла не влияют на условие цикла. Второй цикл никогда не закончится при достаточно малых значениях ***x***.

4.1. Инструкция While ... Wend

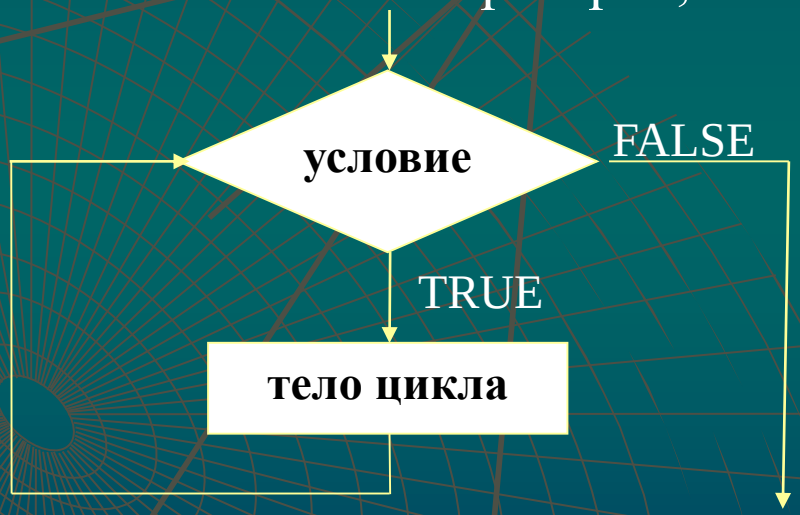
Оператор с предусловием While ... Wend имеет синтаксис:

While condition
statements
Wend

Здесь While .. Wend – зарезервированные слова VBA,

condition –логическое выражение,

statements – один или несколько операторов, составляющие тело цикла.

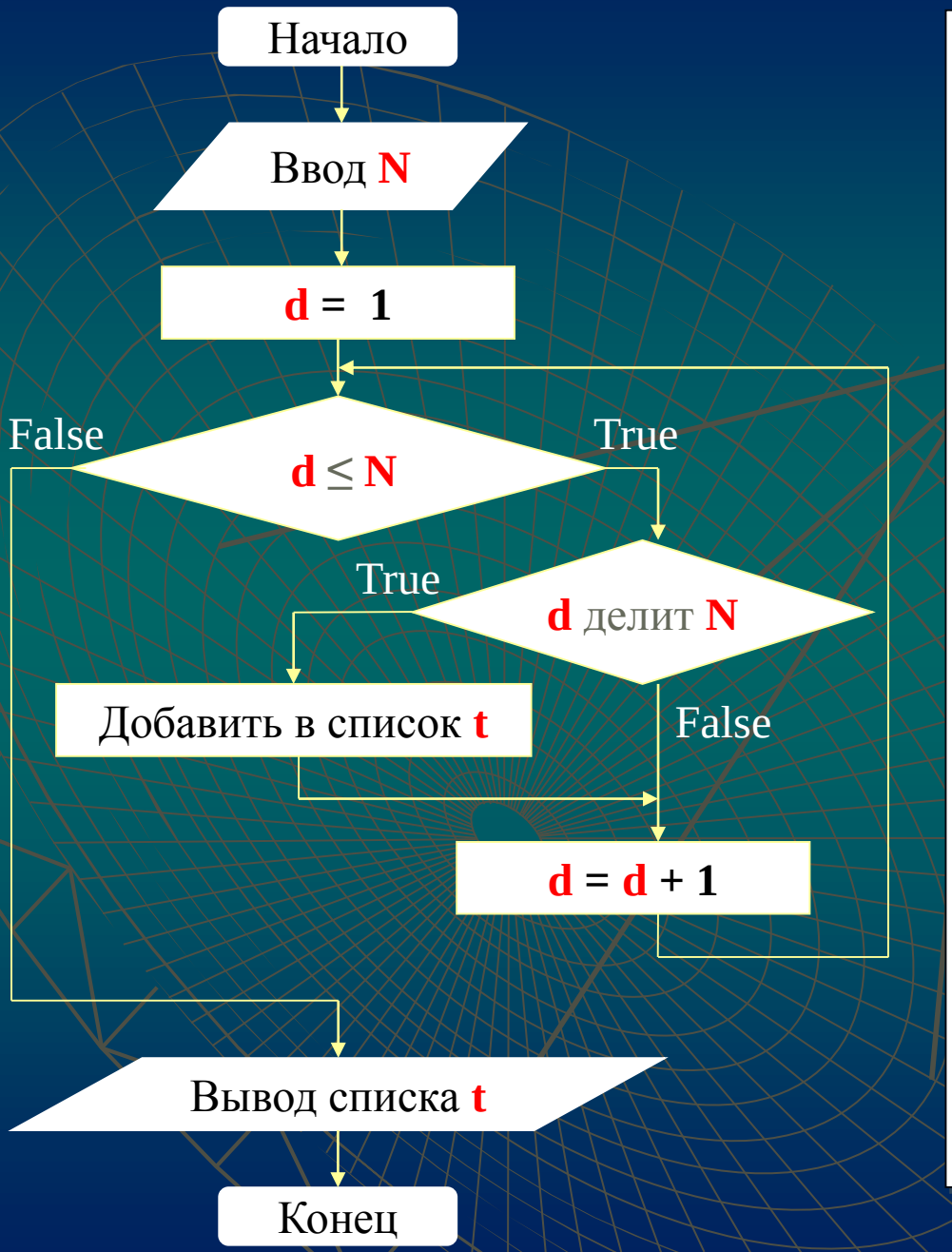


Оператор While ... Wend работает следующим образом:

1. Проверяется условие. Если оно истинно (True), то переход на выполнение тела цикла. В противном случае (False) цикл заканчивается, и управление передается оператору, следующему за телом цикла.

2. Выполняется тело цикла и программа переходит на проверку условия.

Пример. Вывести все делители натурального числа N:




```
Sub While_Wend_Делители()
'Выводится список всех делителей
'числа N  оператор While ... Wend

Dim N As Long
N = InputBox("Введите число N")
d = 1

While d <= N
   If N Mod d = 0 Then
      t = t + Str(d) + ", "
   End If
      d = d + 1
Wend

t = " : " + Left(t, Len(t) - 2) + "."
MsgBox "Делители числа " & N & t

End Sub
```

4.2. Оператор цикла Do

В языке **VBA** цикл Do реализован конструкцией

Do [{**While** | **Until**} *condition*]
    [*statements*]
  [**Exit Do**]
    [*statements*]
Loop [{**While** | **Until**} *condition*]

Оператор **Do** имеет ряд опций и является настолько гибким, что в действительности, он представляет четыре различных конструкции цикла в двух базовых категориях:

Операторы с предусловием

Do While .. Loop;
Do Until .. Loop;

Операторы с постусловием

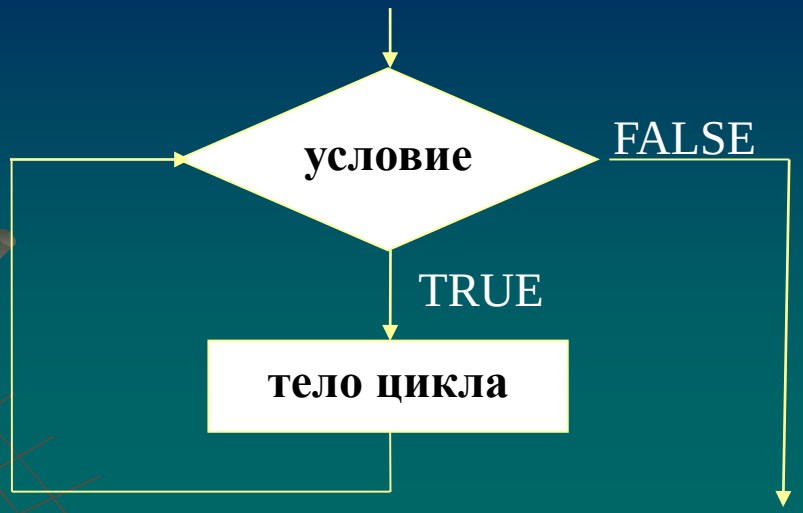
Do .. Loop While;
Do .. Loop Until.

4.2.1. Конструкция Do While .. Loop

Наиболее простой конструкцией является **Do While .. Loop**.

Ее синтаксис следующий:

Do While *condition*
statements
Loop



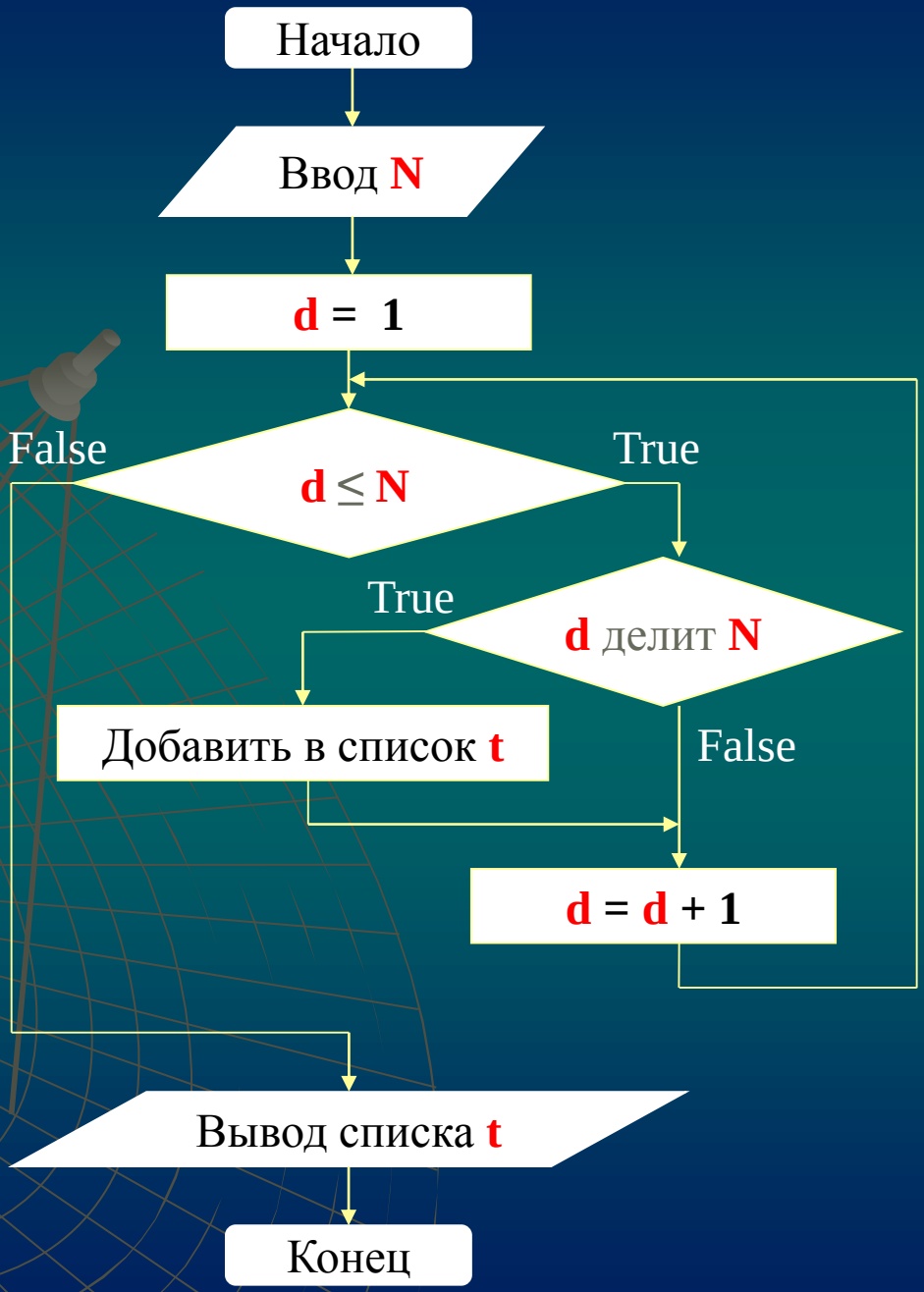
Здесь **Do While .. Loop** – зарезервированные слова VBA,
condition –логическое выражение;
statements –операторы, составляющие тело цикла.

Оператор Do While... Loop является аналогом оператора While ... Wend.

Как и в случае оператора While ... Wend, данный оператор является оператором цикла с предусловием, так как он тестирует условие до выполнения тела цикла.

Пример. Используя оператор Do While... Loop, найдем все делители натурального числа N:

Алгоритм решения задачи оставим **без изменения**:



Найдите три отличия в представленных программах.

```
Sub While_Wend_Делители()
'Выводится список всех делителей
'числа N  оператор While ... Wend
Dim N As Long
N = InputBox("Введите число N")
d = 1

While d <= N
   If N Mod d = 0 Then
      t = t + Str(d) + ", "
   End If
      d = d + 1
Wend

t = " : " + Left(t, Len(t) - 2) + "."
MsgBox "Делители числа " & N & t

End Sub
```

```
Sub Do_While_Loop_Делители()
'Выводится список всех делителей
'числа N  оператор Do While ... Loop
Dim N As Long
N = InputBox("Введите число N")
d = 1

Do While d <= N
   If N Mod d = 0 Then
      t = t + Str(d) + ", "
   End If
      d = d + 1
Loop

t = " : " + Left(t, Len(t) - 2) + "."
MsgBox "Делители числа " & N & t

End Sub
```

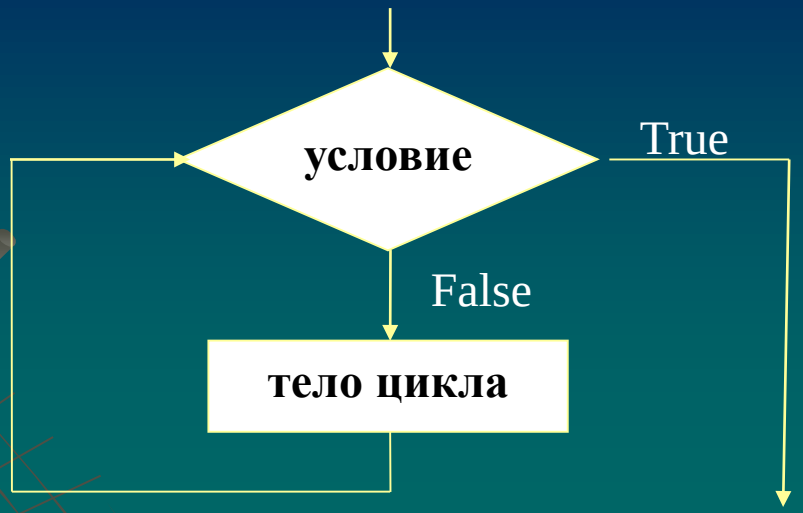
Вторая программа получена из первой заменой: While → Do While, Wend → Loop.

4.2.2. Конструкция Do Until ... Loop

Еще одной конструкцией с предусловием является Do Until ... Loop.

Ее синтаксис следующий:

Do Until *condition*
statements
Loop

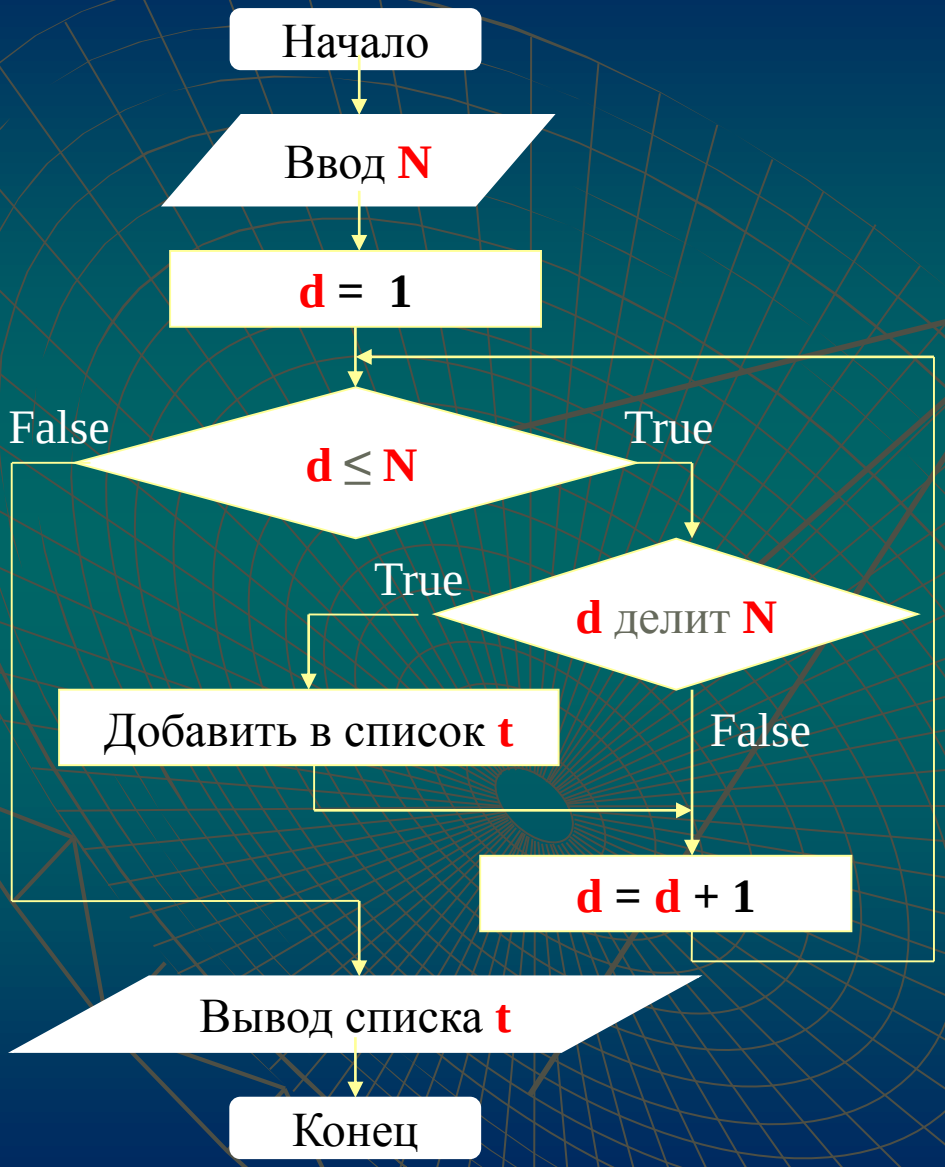


Здесь **Do Until .. Loop** – зарезервированные слова VBA,
condition –логическое выражение;
statements –операторы, составляющие тело цикла.

Do Until... Loop также является оператором цикла с предусловием, так как он тестирует условие до выполнения тела цикла.

В отличие от оператора Do While... Loop, в операторе Do Until ... Loop тело цикла выполняется в случае невыполнения условия.

Пример. Используя оператор Do Until ... Loop, найдем все делители натурального числа N:



Do While... Loop

Do Until ... Loop

Отличие приведенных программ минимально:

```
Sub  Do_While_Loop_Делители()
'Выводится список всех делителей
'числа N  оператор Do While ... Loop
Dim N As Long
N = InputBox("Введите число N")
d = 1

Do While d <= N
   If N Mod d = 0 Then
      t = t + Str(d) + ", "
   End If
      d = d + 1
Loop

t = " : " + Left(t, Len(t) - 2) + "."
MsgBox "Делители числа " & N & t

End Sub
```

```
Sub  Do_Until_Loop_Делители()
'Выводится список всех делителей
'числа N  оператор Do Until ... Loop
Dim N As Long
N = InputBox("Введите число N")
d = 1

Do Until d > N
   If N Mod d = 0 Then
      t = t + Str(d) + ", "
   End If
      d = d + 1
Loop

t = " : " + Left(t, Len(t) - 2) + "."
MsgBox "Делители числа " & N & t

End Sub
```

4.2.3. Конструкция Do .. Loop While

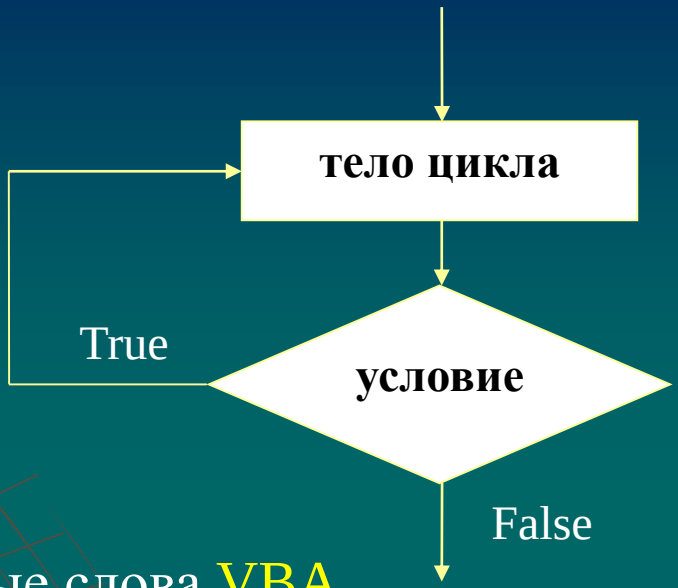
Одной из конструкций цикла с постусловием является **Do .. Loop While**.

Ее синтаксис следующий:

Do

statements

Loop While *condition*



Здесь **Do .. Loop While** – зарезервированные слова VBA,

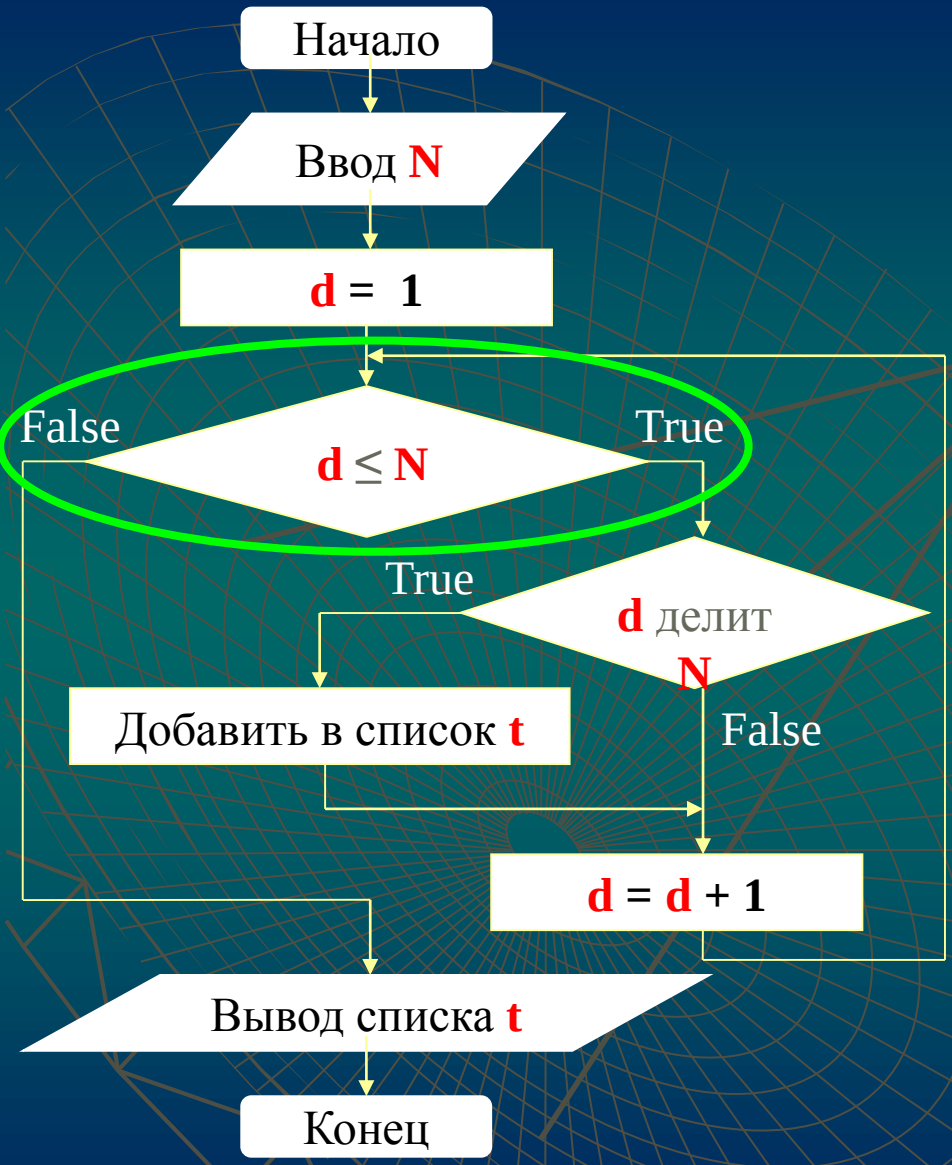
condition –логическое выражение;

statements –операторы, составляющие тело цикла.

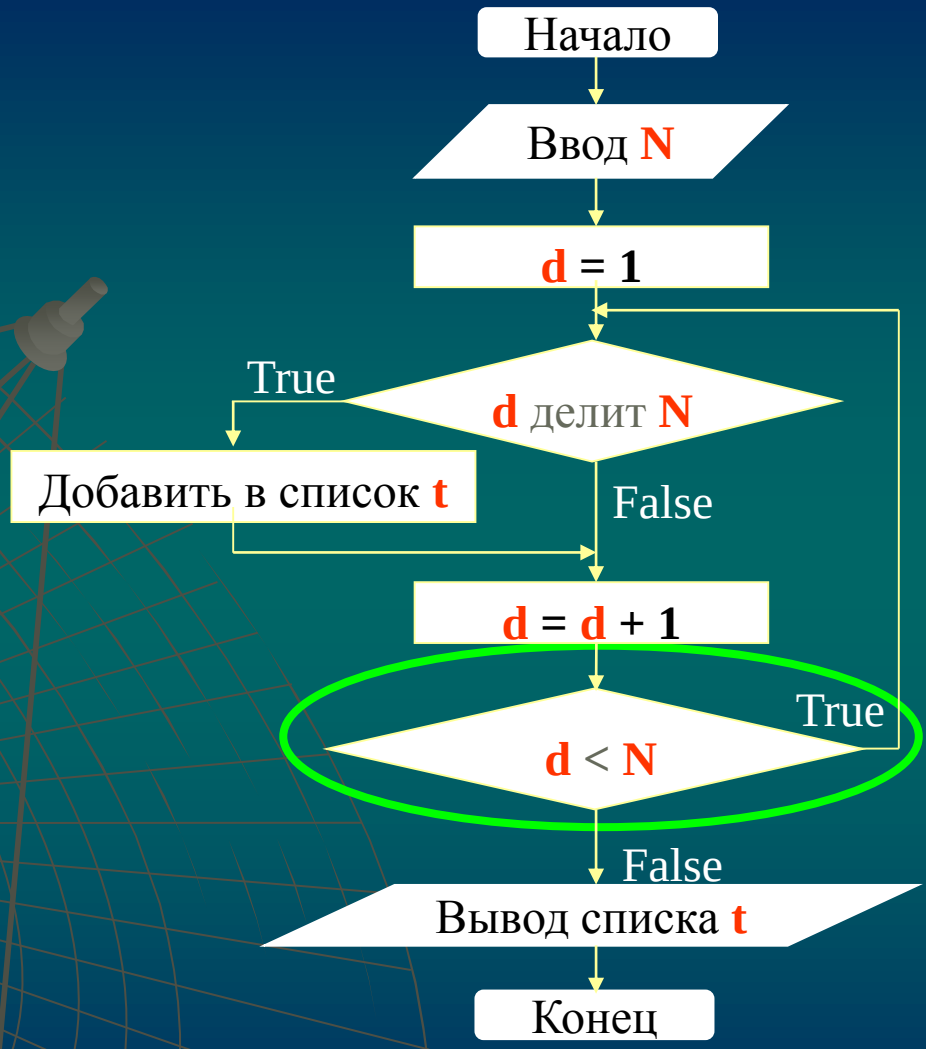
Оператор Do... Loop While тестирует условие после выполнения тела цикла. Если условие выполняется, то программа снова обращается к началу тела цикла.

Тело цикла с постусловием всегда будет выполнен хотя бы один раз!

Пример. Решим нашу задачу, используя оператор Do. .. Loop While.



Do While ... Loop



Do. .. Loop While

Сравните представленные программы.

```
Sub  Do_While_Loop_Делители()
'Выводится список всех делителей
'числа N  оператор Do While ... Loop
Dim N As Long
N = InputBox("Введите число N")
d = 1

Do While d <= N
   If N Mod d = 0 Then
      t = t + Str(d) + ", "
   End If
      d = d + 1
Loop

t = " : " + Left(t, Len(t) - 2) + "."
MsgBox "Делители числа " & N & t

End Sub
```

```
Sub  Do_Loop_While_Делители()
'Выводится список всех делителей
'числа N  оператор Do ... Loop While
Dim N As Long
N = InputBox("Введите число N")
d = 1

Do
   If N Mod d = 0 Then
      t = t + Str(d) + ", "
   End If
      d = d + 1
Loop While d < N

t = " : " + Left(t, Len(t) - 2) + "."
MsgBox "Делители числа " & N & t

End Sub
```

Вторая программа получена из первой переносом оператора While в конец цикла и изменением логического выражения.

4.2.4. Конструкция Do .. Loop Until

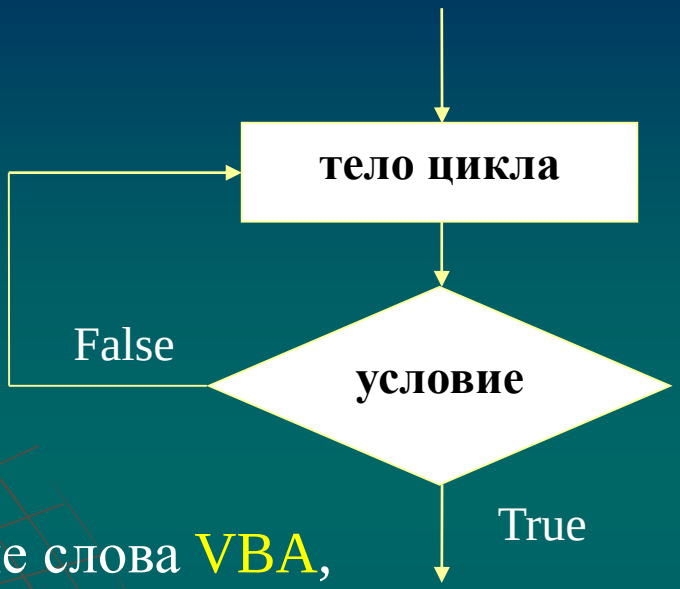
Еще одной из конструкций **Do** с постусловием является Do .. Loop Until.

Ее синтаксис следующий:

Do

statements

Loop Until *condition*



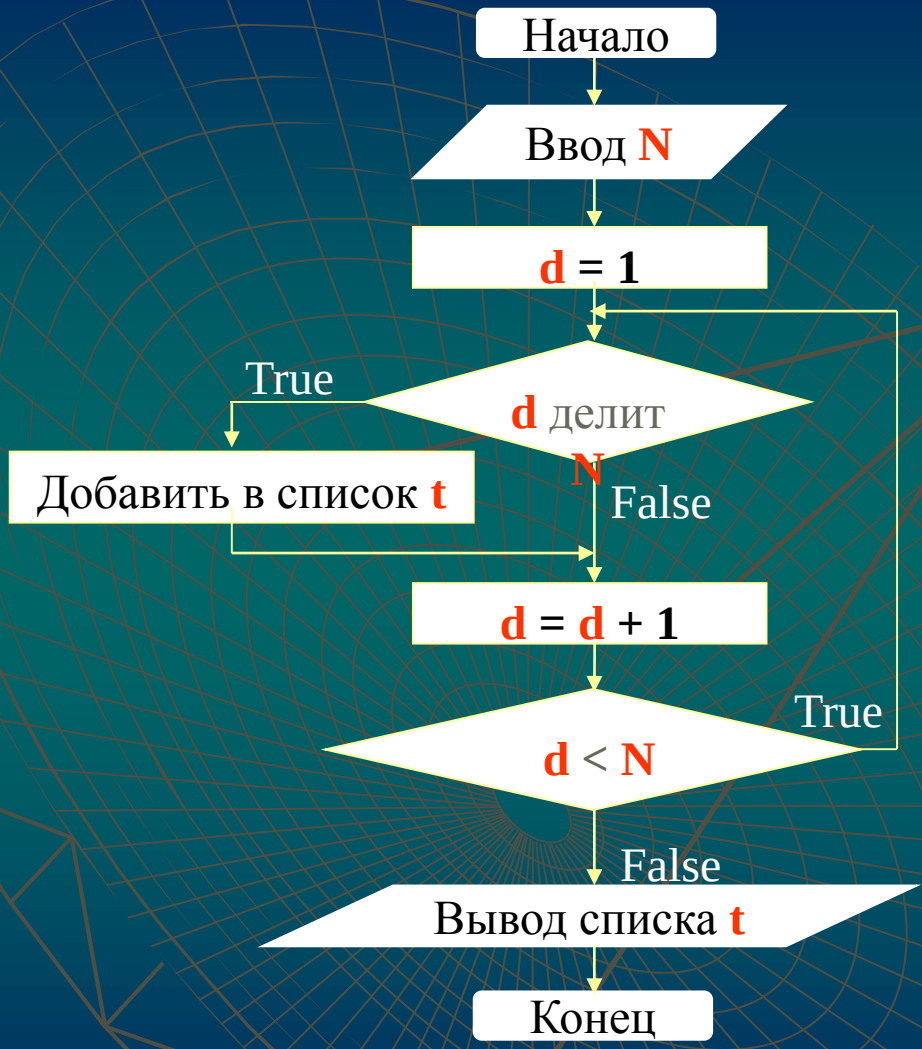
Здесь **Do .. Loop Until** – зарезервированные слова VBA,

condition – любое логическое выражение;

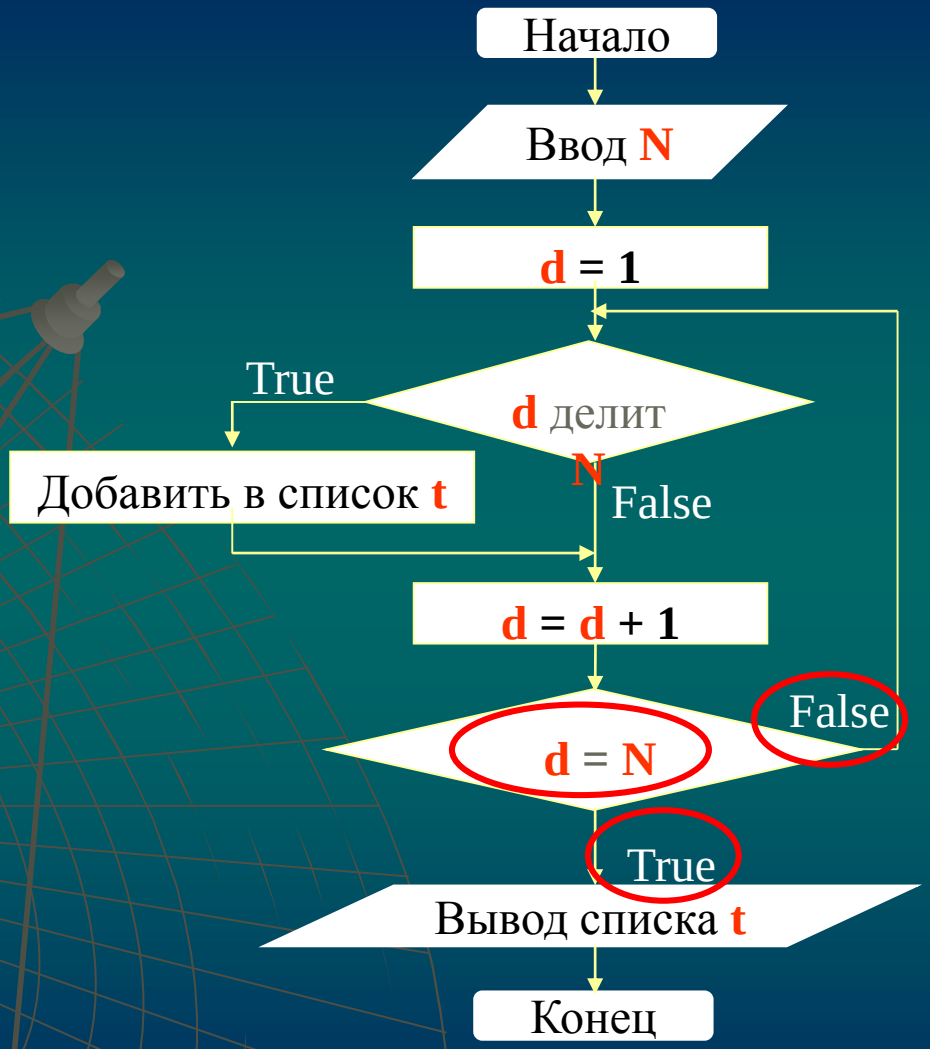
statements –операторы, составляющие тело цикла.

Оператор Do... Loop Until тестирует условие после выполнения тела цикла. Повторное выполнение тела цикла будет в случае выполнения заданного условия.

Пример. Решим задачу нахождения всех делителей натурального числа N используя оператор Do. .. Loop Until.



Do. .. Loop While



Do. .. Loop Until

В представленных программах небольшие отличия.

```
Sub  Do_Loop_While_Делители()
'Выводится список всех делителей
'числа N  оператор Do ... Loop While
Dim N As Long
N = InputBox("Введите число N")
d = 1

Do
   If N Mod d = 0 Then
      t = t + Str(d) + ", "
   End If
      d = d + 1
Loop While d < N

t = " : " + Left(t, Len(t) - 2) + "."
MsgBox "Делители числа " & N & t

End Sub
```

```
Sub  Do_Loop_Until_Делители()
'Выводится список всех делителей
'числа N  оператор Do ... Loop Untile
Dim N As Long
N = InputBox("Введите число N")
d = 1

Do
   If N Mod d = 0 Then
      t = t + Str(d) + ", "
   End If
      d = d + 1
Loop Until d = N

t = " : " + Left(t, Len(t) - 2) + "."
MsgBox "Делители числа " & N & t

End Sub
```

Вторая программа получена из первой заменой:

Loop While d < N на Loop Until d = N

4.3. Цикл с параметром For ... Next

Операторы цикла с условием обладают значительной гибкостью, но не удобны для организации «строгих» циклов, которые должны быть выполнены фиксированное число раз.

Оператор цикла for используется именно в таких случаях.

Синтаксис оператора следующий:

For ***counter*** = ***start*** **To** ***end*** [**Step** ***stepsize***]
[***statements***]
Next [***counter***]

Здесь **For, To, Step, Next** – зарезервированные слова VBA,

counter – управляющая переменная (параметр) цикла;

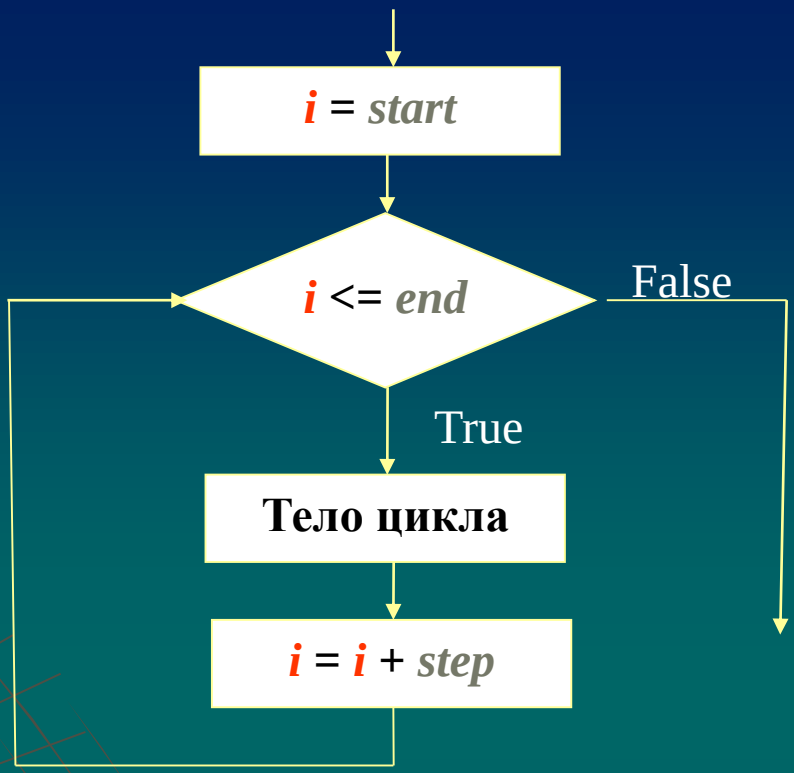
start, end, stepsize – начальное, конечное значения, шаг приращения параметра;

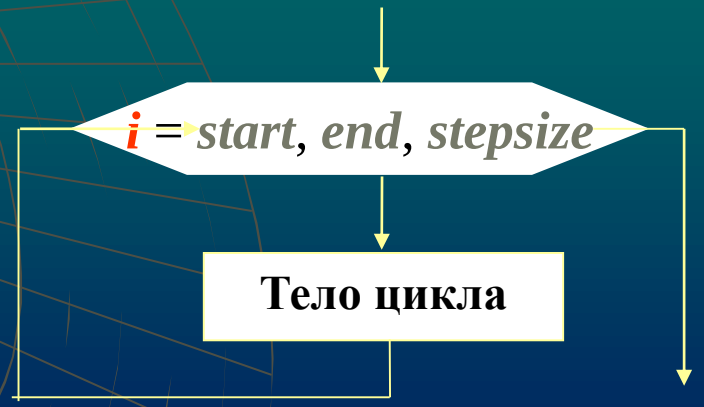
statements –операторы, составляющие тело цикла.

Алгоритм работы цикла For ... Next

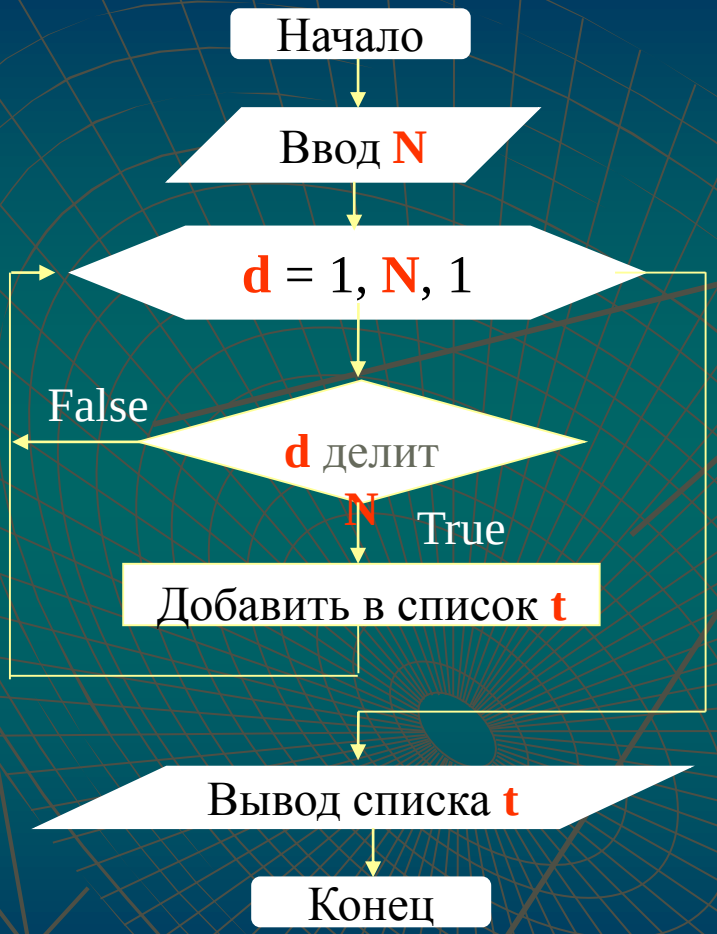
1. Параметру цикла ***i*** присваивается начальное значение ***start***.
2. Если ***i*** > ***end***, то цикл завершает свою работу.
3. Выполняется тело цикла.
4. Значение ***i*** изменяется на шаг ***stepsize*** и переход к п.2.

При шаге ***stepsize*** > 0 параметр ***i*** с каждым проходом возрастает, а при ***stepsize*** < 0 параметр ***i*** убывает.



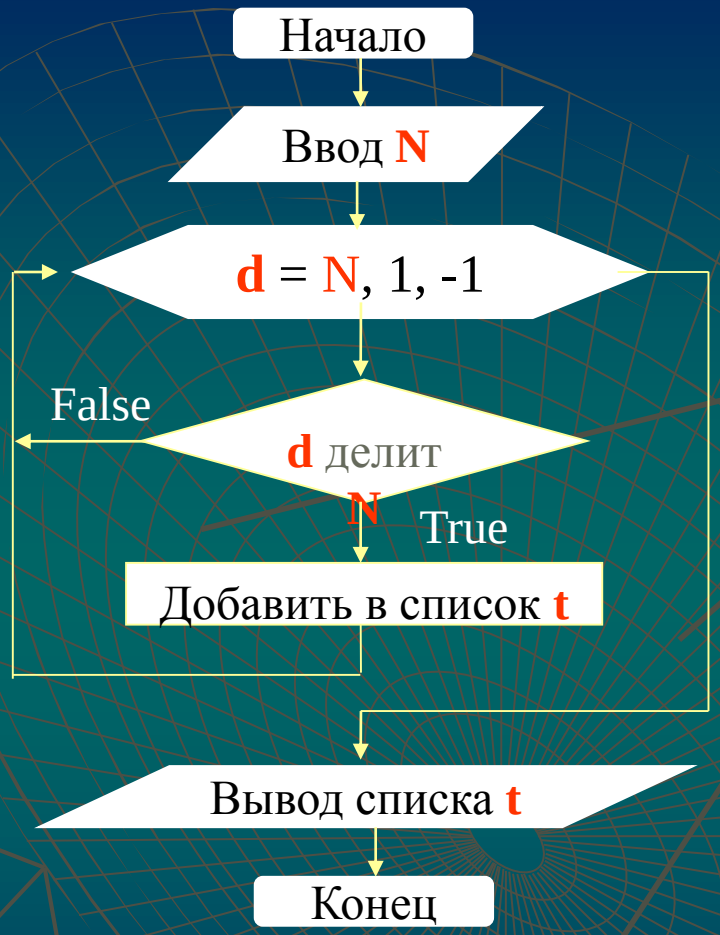


Пример. Решим задачу нахождения делителей N используя оператор For ... Next с возрастающим счетчиком и с убывающим счетчиком.



For ... Next
с возрастающим счетчиком

```
Sub For_Next_Возр_Делители
' Программа выводит все делители
числа N оператор For ... Next

Dim N As Long
N = InputBox("Введите число N")

For d = 1 To N Step 1
   If N Mod d = 0 Then
      t = t + Str(d) + ", "
   End If
Next d

t = " : " + Left(t, Len(t) - 2) + "."
MsgBox "Делители числа " & N & t

End Sub
```



For ... Next

с убывающим счетчиком

```
Sub For_Next_Убыв_Делители
' Программа выводит все делители
числа N оператор For ... Next

Dim N As Long
N = InputBox("Введите число N")

For d = N To 1 Step -1
   If N Mod d = 0 Then
      t = t + Str(d) + ", "
   End If
Next d

t = " : " + Left(t, Len(t) - 2) + "."
MsgBox "Делители числа " & N & t

End Sub
```

Пример. Найти наибольший общий делитель чисел A и B.

Для решения задачи воспользуемся алгоритмом Евклида: будем уменьшать каждый раз большее из чисел на величину меньшего до тех пор, пока оба значения не станут равными.

Используя алгоритм Евклида, найдем наибольший общий делитель чисел 64 и 40.

A	**64**	**64 – 40 = 24**	**24**	**24 – 16 = 8**	**8**
B	**40**	**40**	**40 – 24 =16**	**16**	**16 – 8 = 8**

В блок–схеме решения задачи будем использовать цикл с предусловием.

Тогда тело цикла, выполняющее уменьшение чисел A = A – B либо B = B - A будет повторяется до тех пор, пока A не равно B.

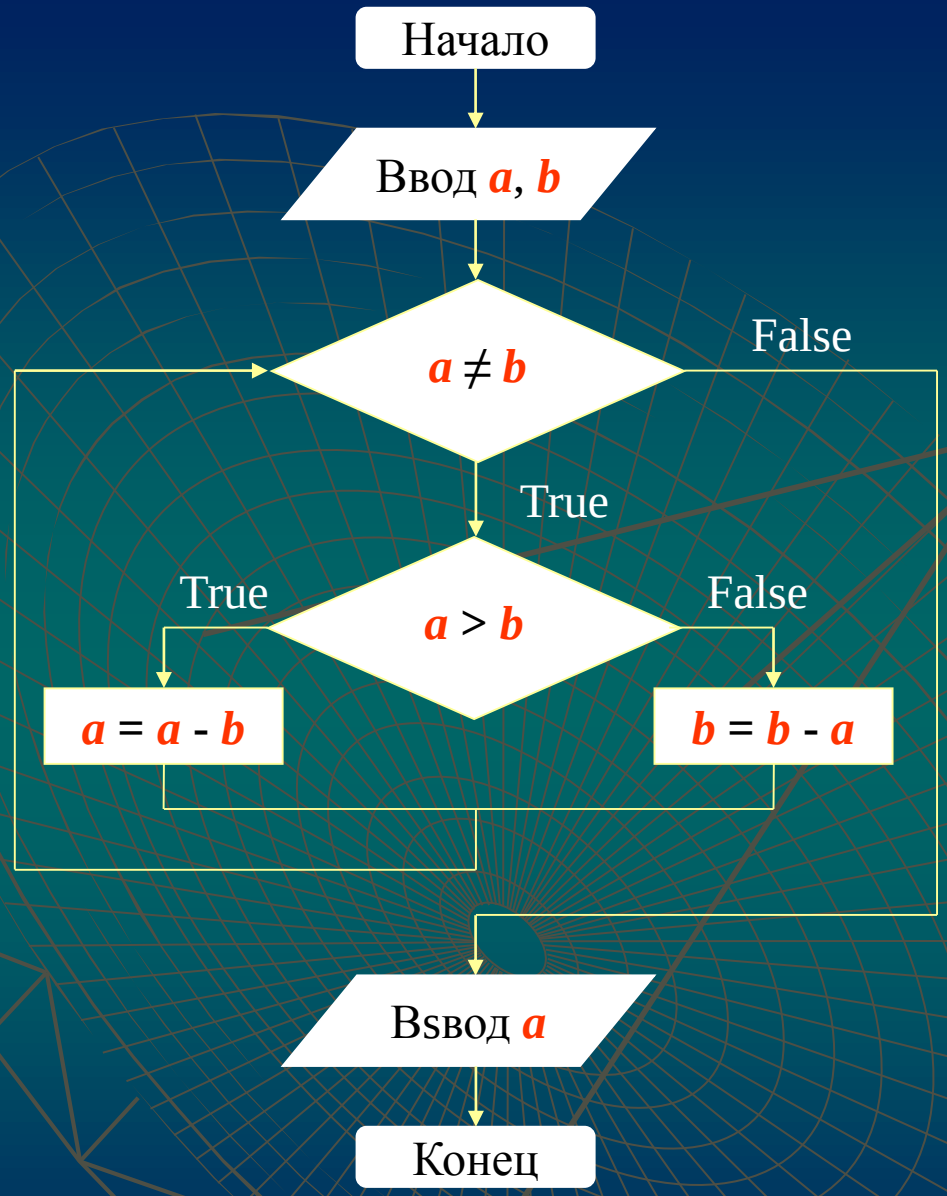


Схема нахождения НОД по алгоритму Евклида

При создании программы воспользуемся циклом While … Wend.

```
Sub НОД_Евклида
' Программа выводит наибольший
' общий делитель чисел a и b.

Dim a As Long, b As Long
a = InputBox("Введите число a")
b = InputBox("Введите число b")

While a <> b
    If a > b Then a = a – b Else b = b - a
Wend

MsgBox "НОД чисел a и b = "  & a

End Sub
```

Пример. Вычислить факториал числа **N**

(N!=1 · 2 · 3 · … ·N)

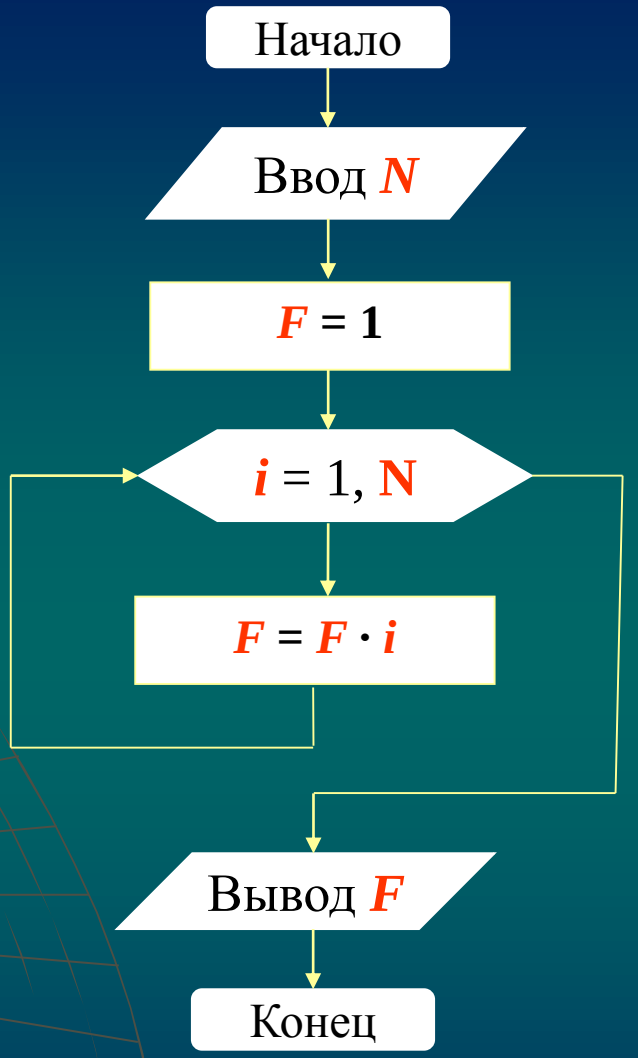
Вводится число **N**.

Переменной ***F*** присваивается начальное значение, равное единице.

Затем организуется цикл, параметром которого выступает ***i***.

Если $i \le N$, то выполняется оператор тела цикла, в котором значение ***F*** умножается на текущее значение параметра цикла ***i***, а результат присваивается ***F***.

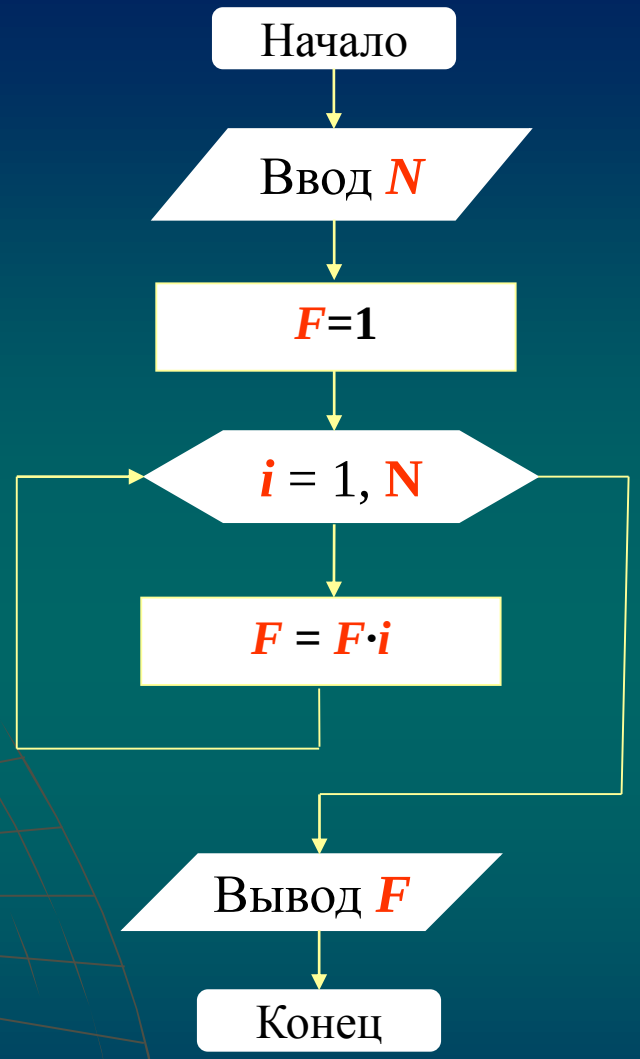
Когда параметр ***i*** становится больше ***N***, цикл заканчивается, и выводится значение переменой ***F***.



Алгоритм нахождения факториала

При создании программы воспользуемся циклом For … Next.

```
Sub Факториал
' Программа вычисляет
' факториал натурального числа N.

Dim N As Integer, F As Double
N = InputBox("Введите число N")
F =1
For i = 1 To N
   F = F * i
Next i

MsgBox "Факториал" & N & "!  = "  & F

End Sub
```



Алгоритм нахождения факториала

4.4. Использование циклов для решения задач
4.4.1. Алгоритм Евклида

Пример. Найти наибольший общий делитель чисел A и B.

Для решения задачи воспользуемся алгоритмом Евклида: будем уменьшать каждый раз большее из чисел на величину меньшего до тех пор, пока оба значения не станут равными.

Используя алгоритм Евклида, найдем наибольший общий делитель чисел 64 и 40.

A	**64**	**64 – 40 = 24**	**24**	**24 – 16 = 8**	**8**
B	**40**	**40**	**40 – 24 =16**	**16**	**16 – 8 = 8**

В блок–схеме решения задачи будем использовать цикл с предусловием.

Тогда тело цикла, выполняющее уменьшение чисел A = A – B либо B = B - A будет повторяется до тех пор, пока A не равно B.

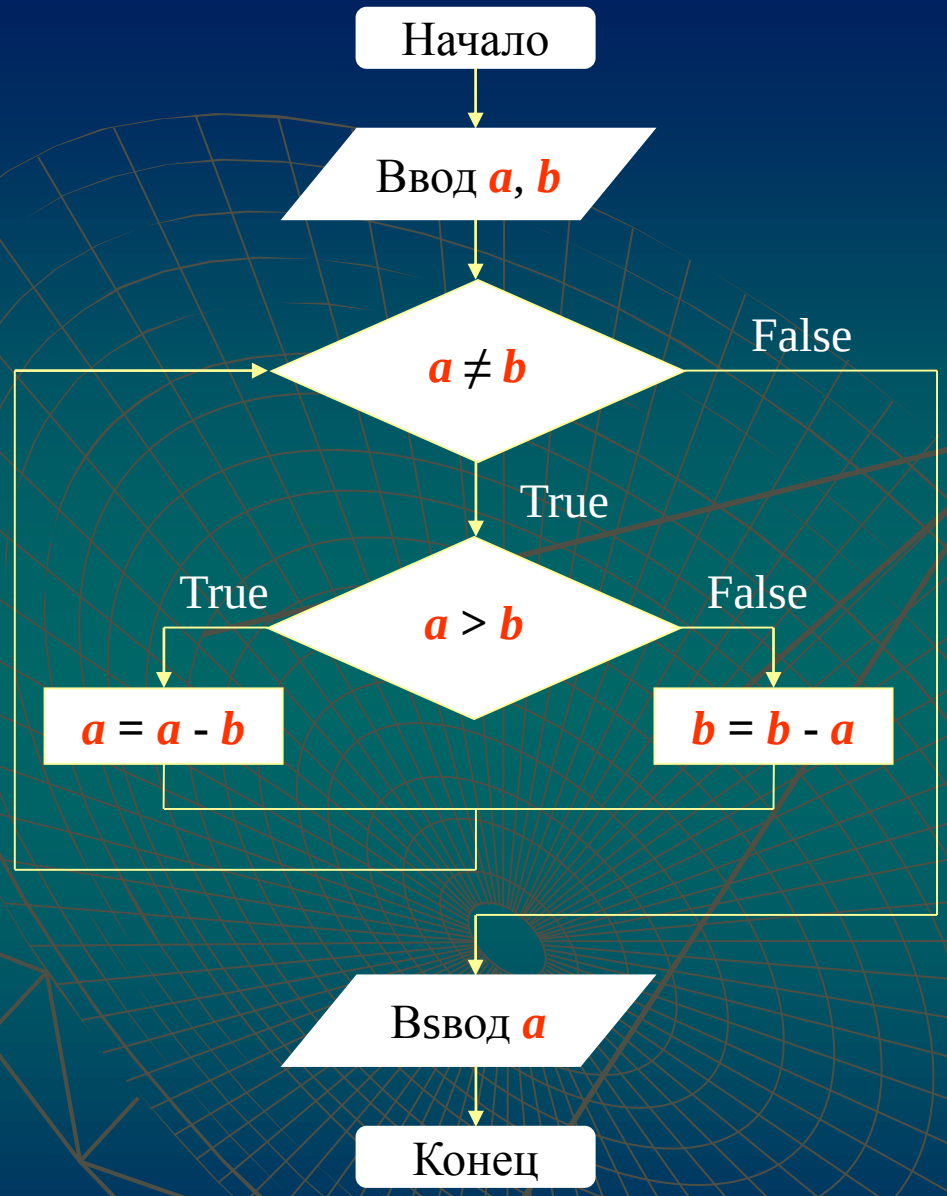


Схема нахождения НОД по алгоритму Евклида

При создании программы воспользуемся циклом While … Wend.

```
Sub НОД_Евклида
' Программа выводит наибольший
' общий делитель чисел a и b.

Dim a As Long, b As Long
a = InputBox("Введите число a")
b = InputBox("Введите число b")

While a <> b
    If a > b Then a = a – b Else b = b - a
Wend

MsgBox "НОД чисел a и b = "  & a

End Sub
```

4.4.2. Вычисление факториала

Пример. Вычислить факториал числа **N**

$(N!=1 \cdot 2 \cdot 3 \cdot \ldots \cdot N)$

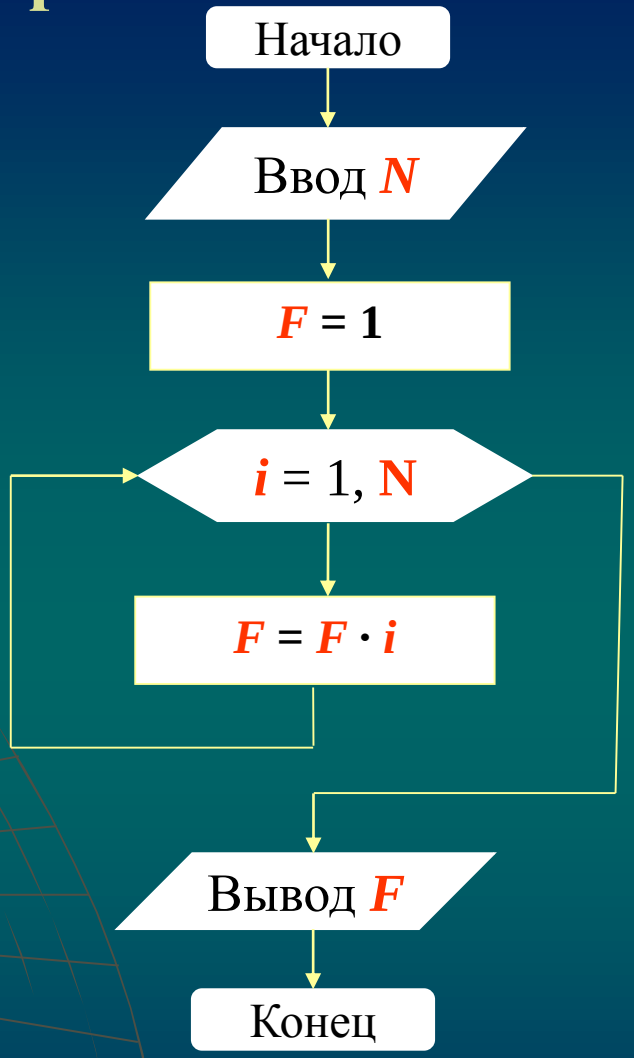
Вводится число **N**.

Переменной ***F*** присваивается начальное значение, равное единице.

Затем организуется цикл, параметром которого выступает ***i***.

Если $i \le N$, то выполняется оператор тела цикла, в котором значение ***F*** умножается на текущее значение параметра цикла ***i***, а результат присваивается ***F***.

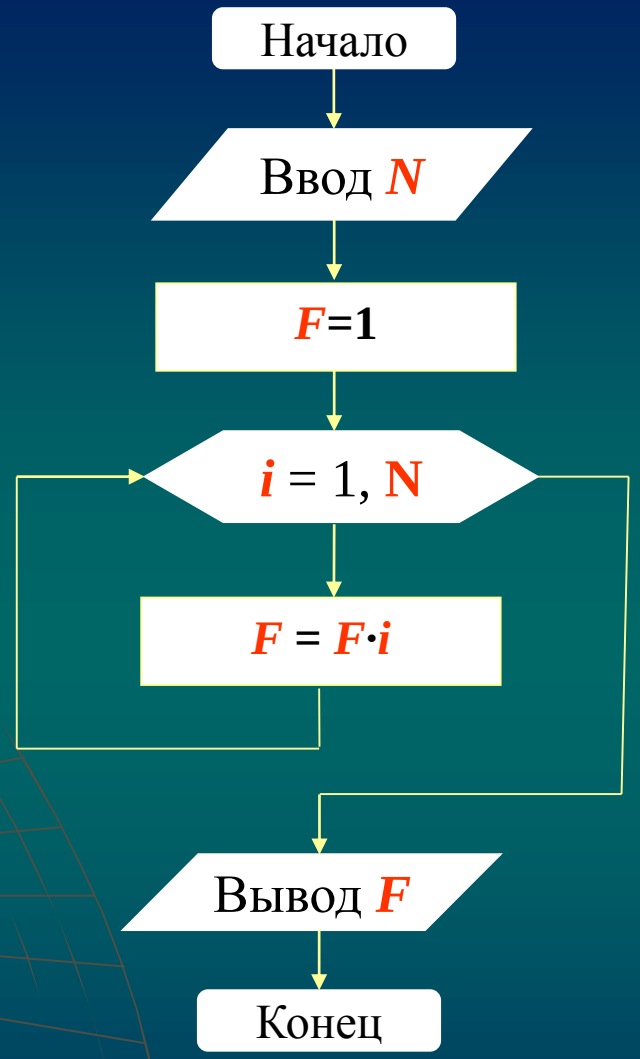
Когда параметр ***i*** становится больше ***N***, цикл заканчивается, и выводится значение переменой ***F***.



Алгоритм нахождения факториала

При создании программы воспользуемся циклом For … Next.

```
Sub Факториал
' Программа вычисляет
' факториал натурального числа N.

Dim N As Integer, F As Double
N = InputBox("Введите число N")
F =1
For i = 1 To N
   F = F * i
Next i

MsgBox "Факториал" & N & "!  = "  & F

End Sub
```



Алгоритм нахождения факториала

4.4.3. Подсчет цифр числа

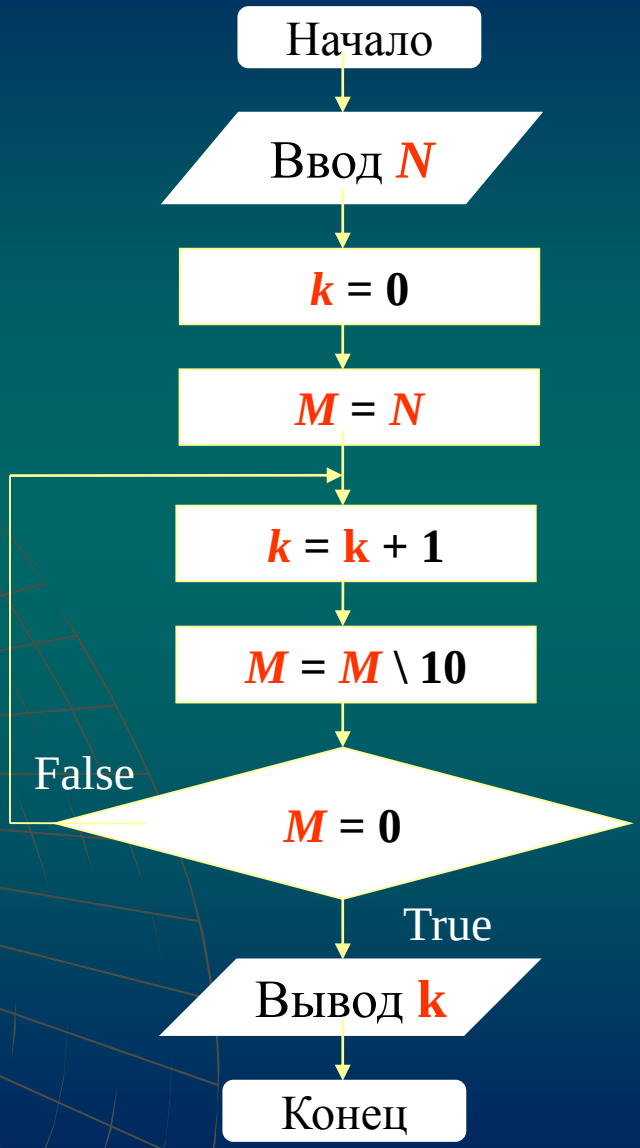
Пример. Определить количество цифр в натуральном числе **N**.

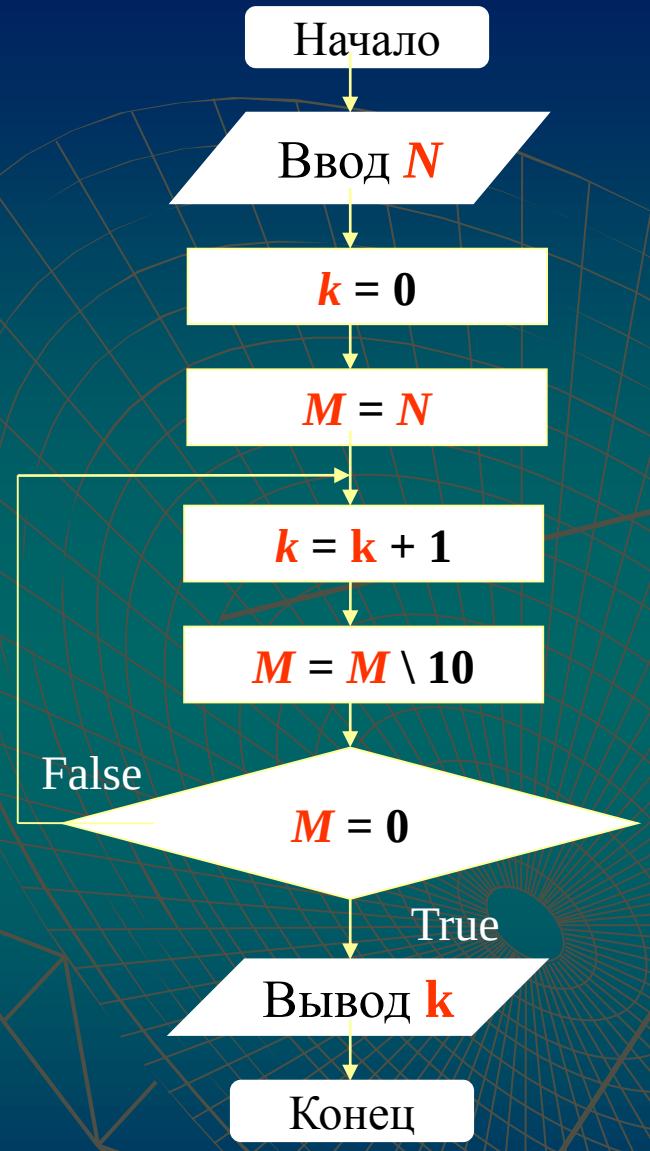
Для того чтобы подсчитать количество цифр в числе, необходимо определить, сколько раз заданное число можно разделить на десять нацело.

Например, пусть N=72865, тогда N делится на 10 ровно 5 раз.

Результаты вычислений сведены в таблицу.

k	N = 72865
1	72865 \ 10 = 7286
2	7286 \ 10 = 728
3	728 \ 10 = 72
4	72 \ 10 = 7
5	7 \ 10 = 0



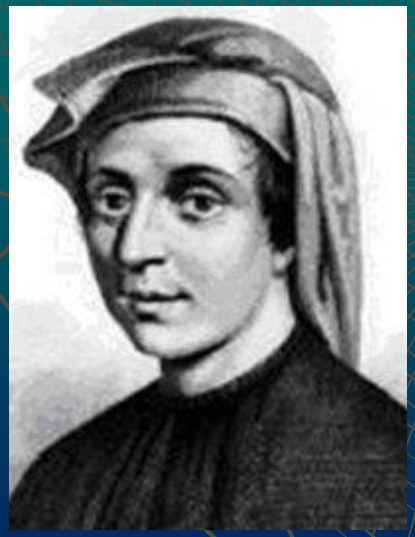



```
Sub Количество_цифр()
' Программа определяет количество
' цифр натурального числа N.

Dim N As Long
N = InputBox("Введите число N")
M = N

Do
   k = k + 1
   M = M \ 10
Loop Until M = 0

MsgBox " Число цифр числа" & N & _
"  равно " & k

End Sub
```

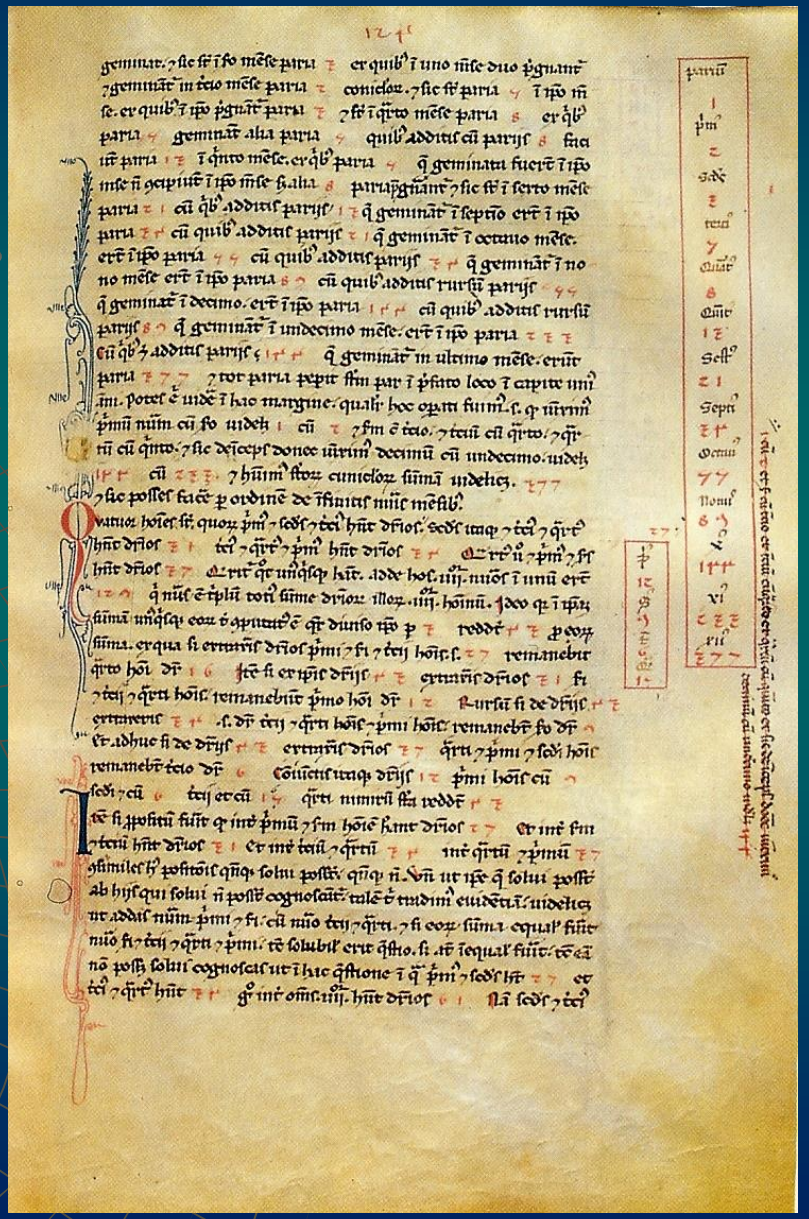
4.4.4. Расчет чисел Фибоначчи

Пример. Выписать числа Фибоначчи, не превышающие заданное число ***N***.

Числа Фибоначчи – это бесконечная последовательность чисел x_1, x_2, x_3, …, x_{n-2}, x_{n-1}, x_n, …, определенных по правилу: первые два числа определены, как $x_1 = 0$, $x_2 = 1$, а все последующие числа получаются, как сумма двух предыдущих чисел данной последовательности.



Числа Фибоначчи были известны в Древней Индии и в Древнем Египте, но в Европе первым их описал Леонардо Пизанский (Фибоначчи, 1170-1250) в работе «Книга абака».



Согласно определению задаем первые три числа $x_1 = 0$, $x_2 = 1$, $x_3 = 1$ и помещаем первые два в список t.

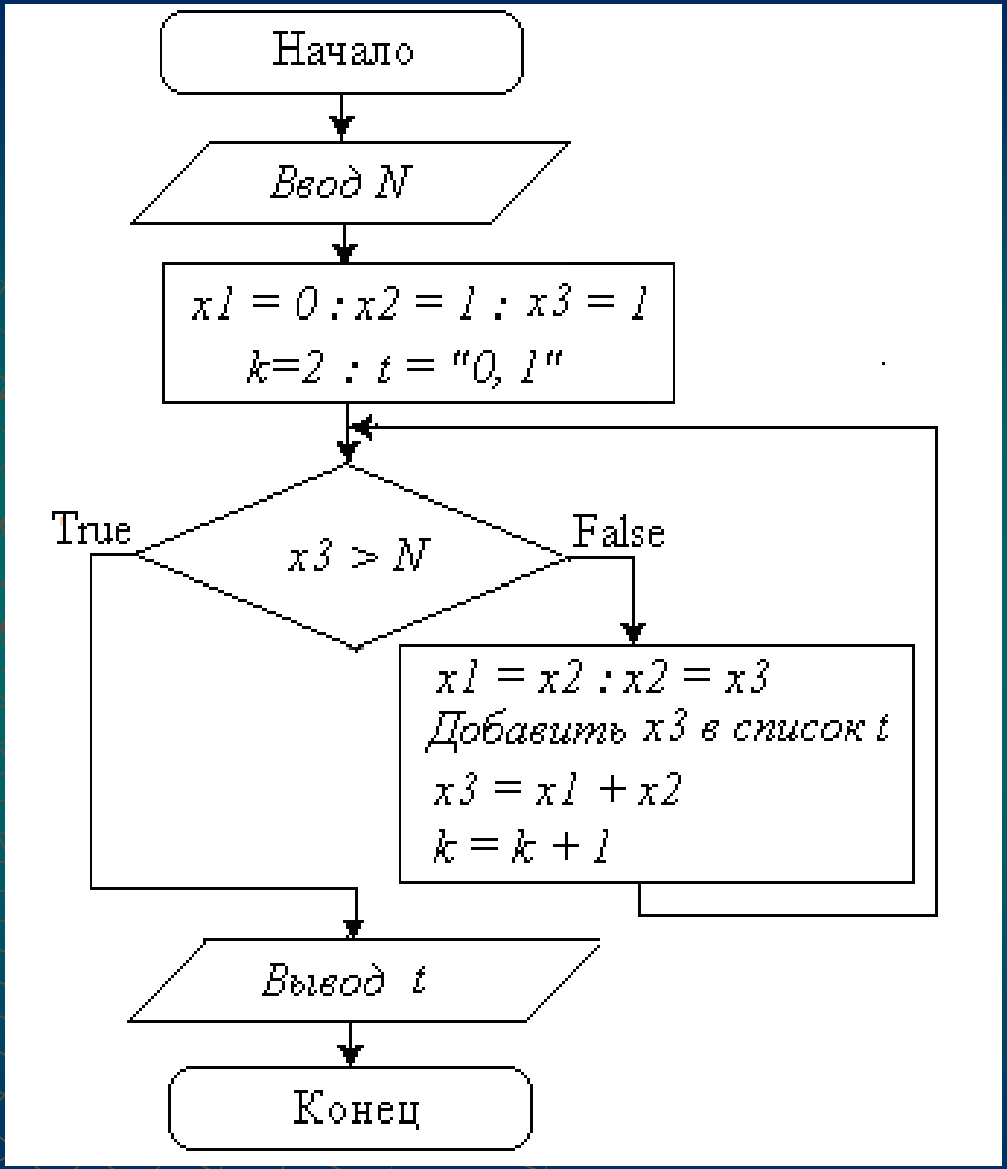
Затем открываем цикл с предусловием $x_3 > N$.

В теле цикла происходит перемещение значений

$x_1 = x_2$ и $x_2 = x_3$,

а затем расчет нового значения переменной $x_3 = x_1 + x_2$ и добавление x_3 в список t.

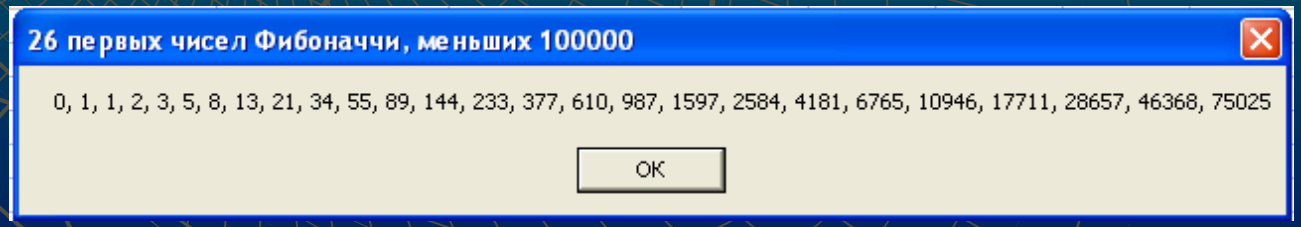
Циклический процесс заканчивается, когда значение x_3 станет больше введенного N.



Алгоритм вычисления чисел Фибоначчи

Представленная структурная схема реализована с помощью инструкции *Do Until... Loop*

```
Sub Числа_фибоначчи()
' Построение последовательности чисел фибоначчи, не превышающих число N
' Инструкция Do Until .. Loop
  Dim N As Long
  N = InputBox("Введите число N ", _
"Построение последовательности чисел фибоначчи, не превышающих число N")
  x1 = 0: x2 = 1: x3 = 1
  k = 2
  t = " 0, 1,"
  Do Until x1 + x2 > N
    x1 = x2
    x2 = x3
    t = t + Str(x3) + ","
    x3 = x1 + x2
    k = k + 1
  Loop
  MsgBox Left(t, Len(t) - 1), , k & " первых чисел фибоначчи, меньших" & N
End Sub
```



4.4.5. Нахождение корней нелинейных уравнений

Пример. Найти корень уравнения $y=f(x)$ на интервале $[a; b]$.

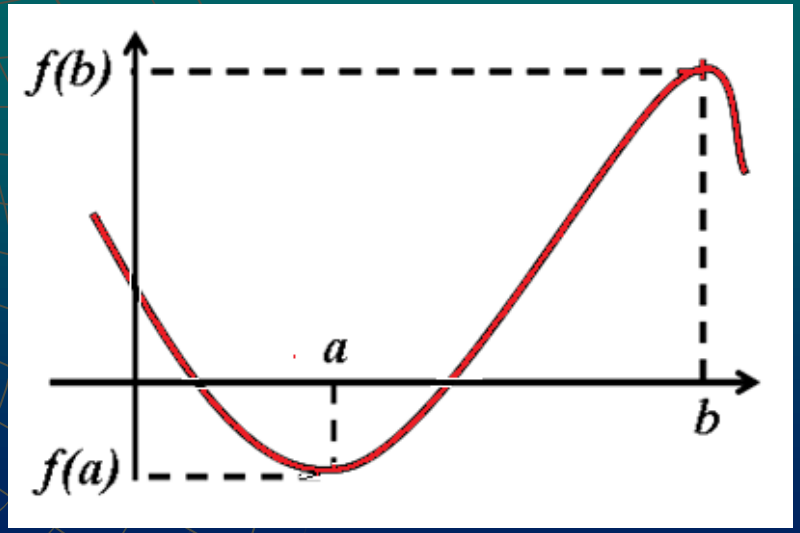
Задача численного нахождения корней нелинейных уравнений состоит из двух этапов:

1) *Отделение корней* – это нахождение числовых промежутков, в которых содержится только один корень.

2) *Уточнение корня* – это вычисление отделенных корней с заданной точностью на выбранных интервалах.

Исследуя функцию $y = f(x)$ с помощью первой производной, можно найти все ее локальные максимумы и минимумы, а затем рассчитать значения функции $f(a)$ и $f(b)$ в точках a и b полученных экстремумов.

Если знаки функции в соседних экстремальных точках различны, а функция непрерывна, тогда она будет непрерывно возрастать или убывать на данном отрезке $[a;b]$, и, следовательно, иметь на этом отрезке ровно один корень.



Отделим, например, корни уравнения $f(x)=x^4-x^3-2x^2+3x-3.$

Найдем производную функции $f'(x)=4x^3-3x^2-4x+3.$

Определим корни производной

$$4x^3-3x^2-4x+3=0;$$
$$4x(x^2-1)-3(x^2-1)=0;$$
$$(x^2-1)(4x-3)=0;$$
$$x_1=-1;\quad x_2=3/4;\quad x_3=1.$$

Составим таблицу знаков функции $f(x)$ в экстремальных точках и при стремлении аргумента к бесконечности.

x	-∞	-1	3/4	1	+∞
Знак f(x)	+	-	-	-	+

Из таблицы видно, что уравнение имеет два действительных корня:

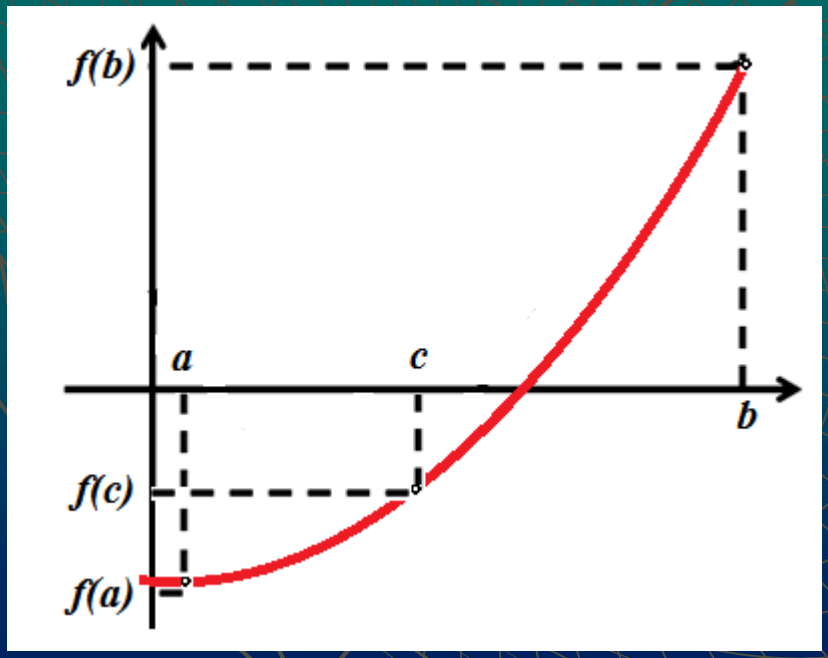
$$x_1\in(-10,-1];\quad x_2\in[1,10).$$

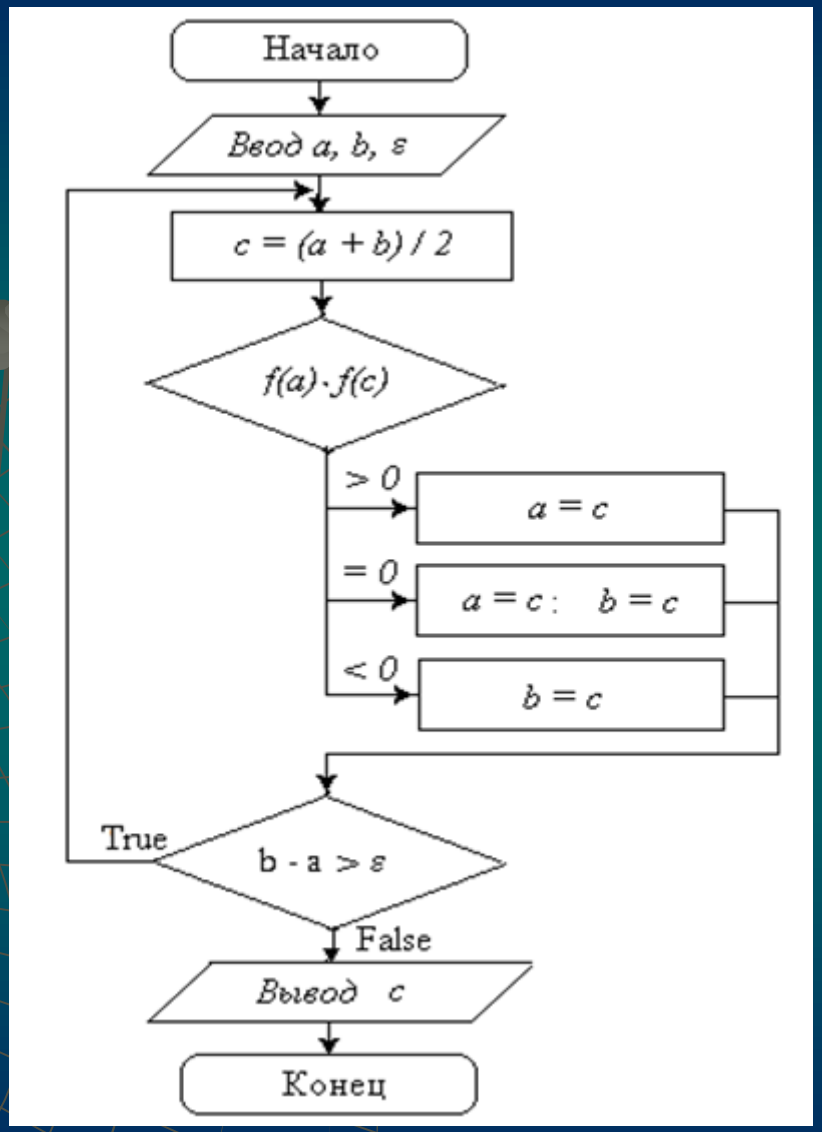
Рассмотрим алгоритм уточнения корня. Выберем один из интервалов с границами a и b, содержащий корень уравнения.

Разделим выбранный интервал пополам точкой $c = \frac{a+b}{2}$

Теперь возможны три случая.

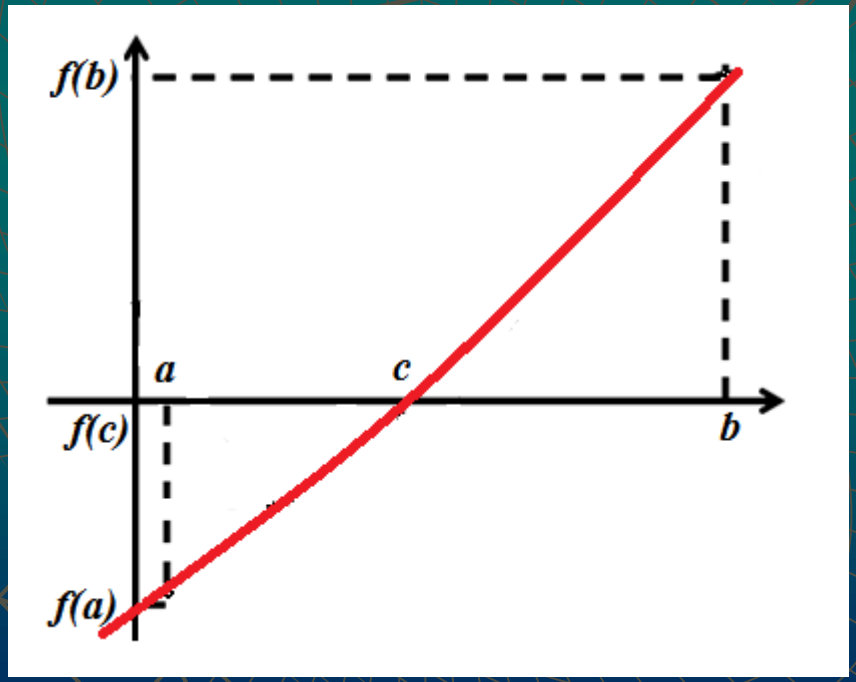
1) Если $f(a) \cdot f(c) > 0$, тогда функция в данных точках одного знака и корня на промежутке $[a;c]$ нет, и мы уменьшаем отрезок $[a;b]$, перенося левую его границу a в точку c.

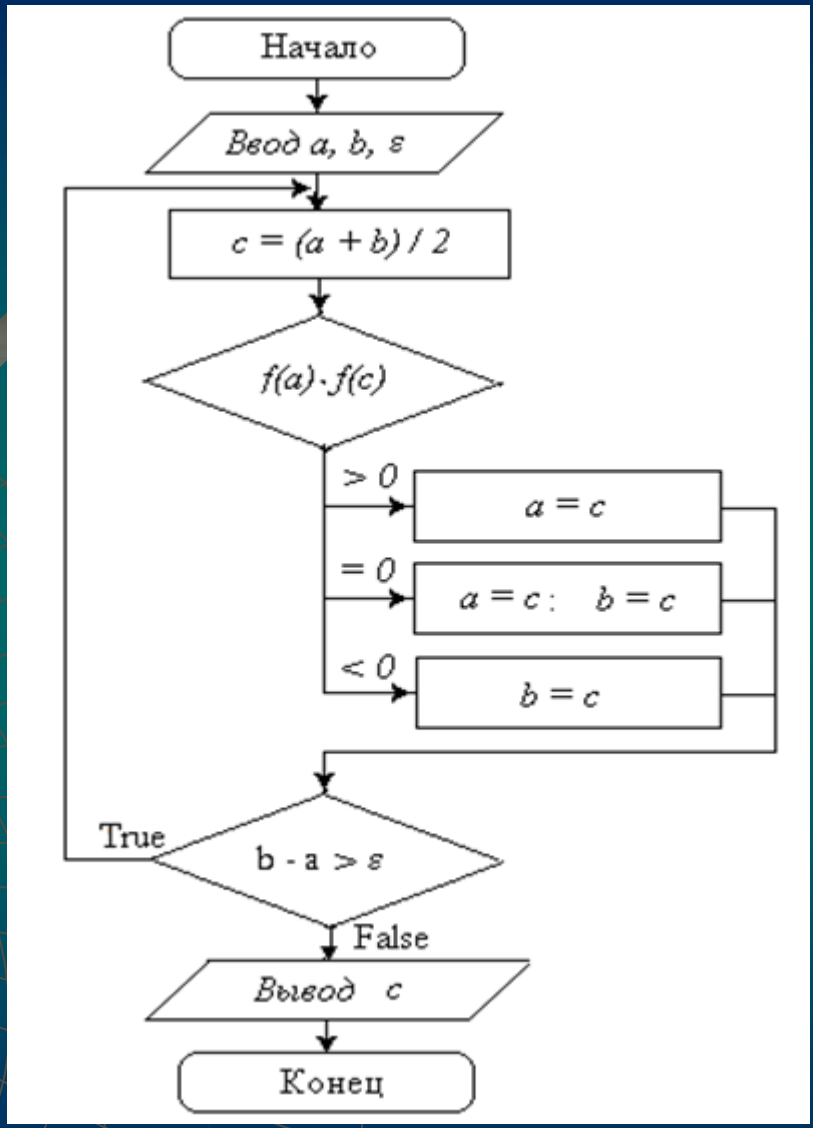




Алгоритм нахождения корня методом половинного деления

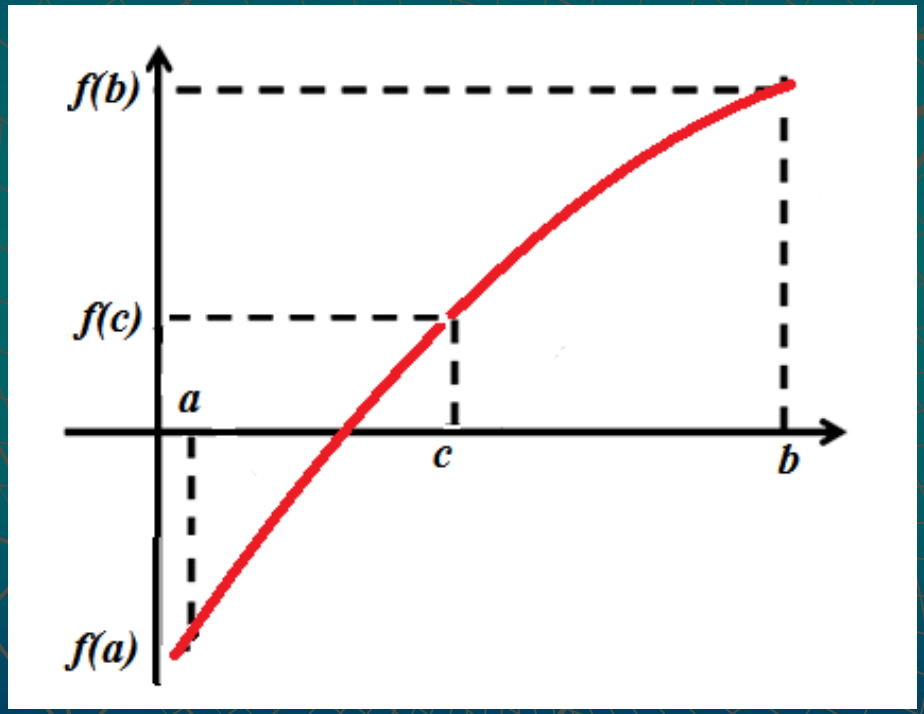
2) Если $f(a) \cdot f(c) = 0$, то решение уже получено и c является корнем уравнения. Тогда мы уменьшаем до нуля отрезок $[a;b]$, перенося в точку c левую и правую его границы.

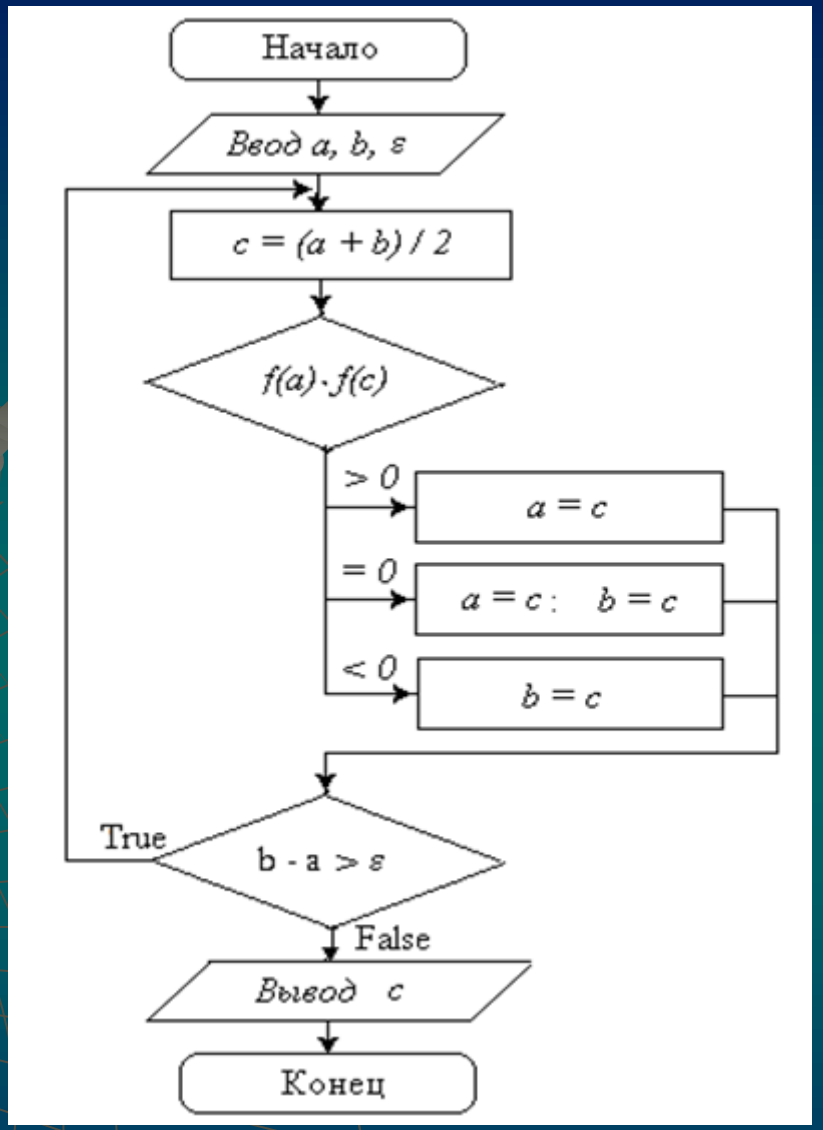




Алгоритм нахождения корня методом половинного деления

3) Если $f(a)\cdot f(c) < 0$, тогда функция в точках a и c принимает значения разных знаков и, следовательно, корень находится на этом промежутке $[a;c]$ и мы уменьшаем отрезок $[a;b]$, перенося правую его границу b в точку c.





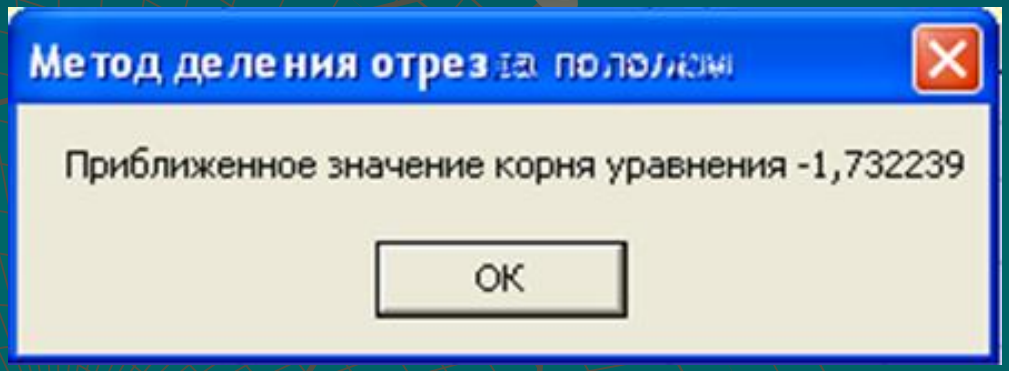
Циклический процесс деления отрезка пополам будем продолжать до тех пор, пока длина отрезка $[a;b]$ не станет меньше заданной величины погрешности ε.

В коде программы будем использовать инструкцию *Do .. Loop While*.

```
Sub Метод_половинного_деления()
' Нахождение корня нелинейного уравнения y=f(x) на отрезке [a, b]
' / Инструкция Do .. Loop While
  Dim a As Single, b As Single, epsilon As Single
  a = InputBox("Введите левую границу отрезка [a, b]")
  b = InputBox("Введите правую границу отрезка [a, b]")
  epsilon = InputBox("Введите погрешность")
  Do
    c = (a + b) / 2
    Select Case f(a) * f(c)
      Case Is < 0
        b = c
      Case 0
        a = c: b = c
      Case Is > 0
        a = c
    End Select
  Loop While b - a > epsilon
  MsgBox "Приближенное значение корня уравнения " & c, , _
         "Метод деления отрезка пополам"
End Sub
```

Создадим функцию f(x), которая неоднократно будет вызываться из основного модуля.

```
Function f(x)
  f = x ^ 4 - x ^ 3 - 2 * x ^ 2 + 3 * x - 3
End Function
```



Окно вывода результата выполнения программы, при *a* = -10, b = -1, epsilon = 0,001.

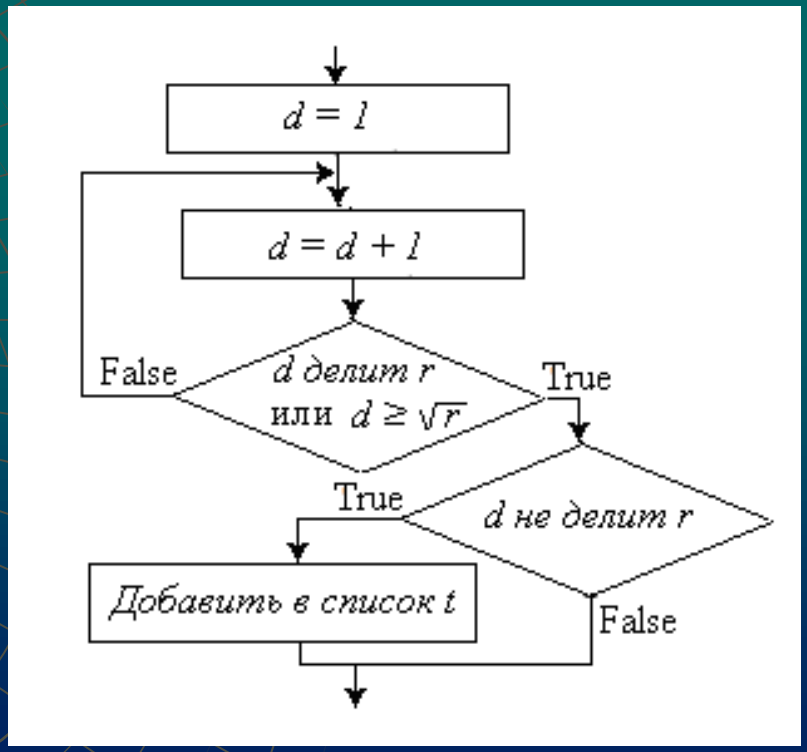
4.5. Вложенные циклы

На практике часто приходится иметь дело со сложными алгоритмами. К таким относятся, например, структуры ***вложенных циклов***. Эта алгоритмическая структура получается, если в теле рассматриваемого цикла находится один или несколько других циклов. Тип и конфигурация участвующих в сложной структуре циклов не важны, а количество циклических вложений в кодах программ может быть любым.

2.1. Вывод простых чисел

Пример. Вывести все простые числа в промежутке от *m* до *n*.

Необходимо перебрать все целые числа от *m* до *n*, при этом определить, имеют ли они нетривиальные делители. Если в результате проверки окажутся простые числа, то их выводим.



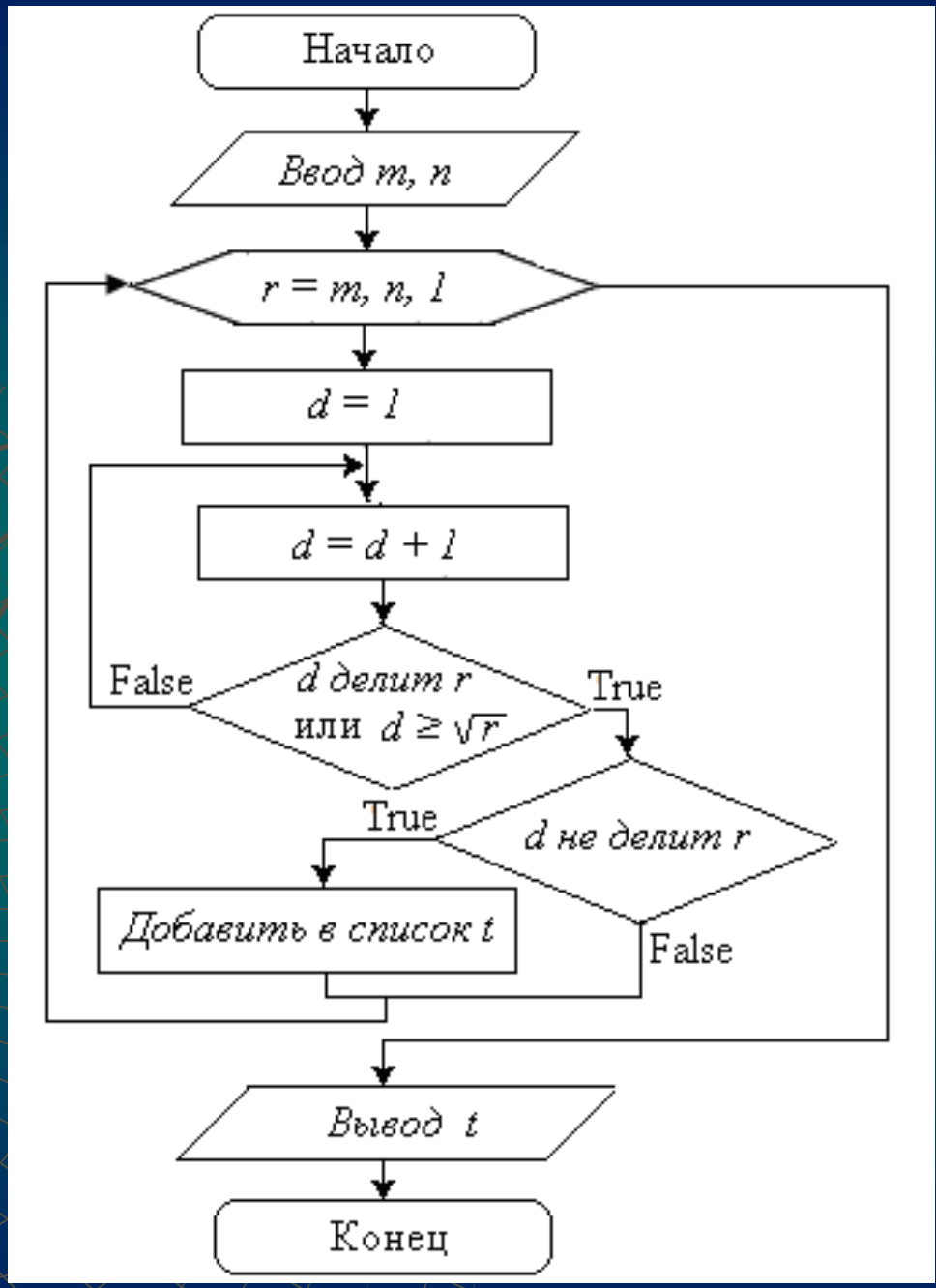
Алгоритм определения простоты числа ***r***

В приведенном алгоритме в цикл с параметром ***r***, принимающем все целые значения от ***m*** до ***n***, вложен цикл с двойным постусловием

$$d \text{ делит } r \text{ или } d \geq \sqrt{r}$$

Во внутреннем цикле число ***r*** проверяем на делители ***d*** от **2** до $r^{0,5}$.

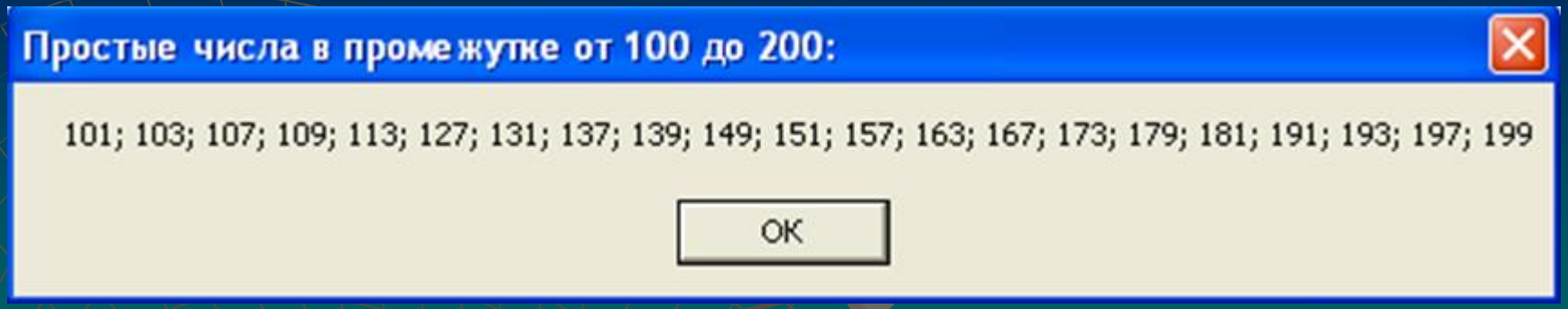
Если такой делитель найден или значение ***d*** равно или превышает ***r*** , то вложенный цикл заканчивает свою работу, и в случае отсутствия делителя в список ***t*** помещается текущее значение ***r***.



Внешний цикл удобно реализовать с помощью инструкции ***For .. Next***, так как мы уже знаем начальное ***m*** и конечное ***n*** значения параметра ***r*** цикла, а также его приращение ***stepsize*** = **1**. Для организации вложенного цикла с постусловием применим структуру ***Do .. Loop Until***.

```
Sub Простые_числа()
'Программы выводит все простые числа от m до n
' Вложенные числа
  Dim m As Long, n As Long
  m = InputBox("Введите начало промежутка")
  n = InputBox("Введите конец промежутка")
  For r = m To n
    d = 1
    Do
      d = d + 1
    Loop Until d >= Sqr(r) Or r Mod d = 0
    If r Mod d <> 0 Then t = t + Str(r) + ";"
  Next r
  MsgBox Left(t, Len(t) - 1), , _
        "Простые числа в промежутке от " & m & " до " & n & ":"
End Sub
```

Результат нахождения простых чисел в промежутке от 100 до 200



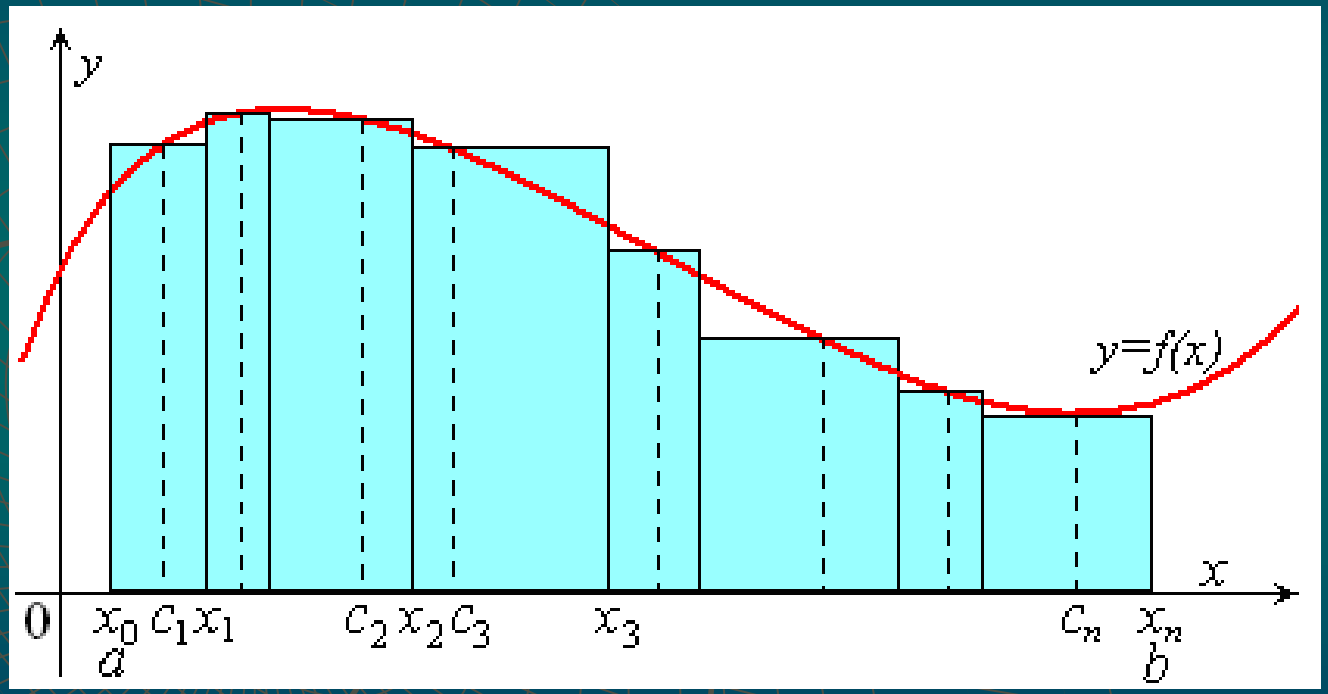
Окно вывода результата выполнения программы Простые_числа.

4.5.2. Нахождение определенных интегралов

$$\int_a^b f(x)dx$$

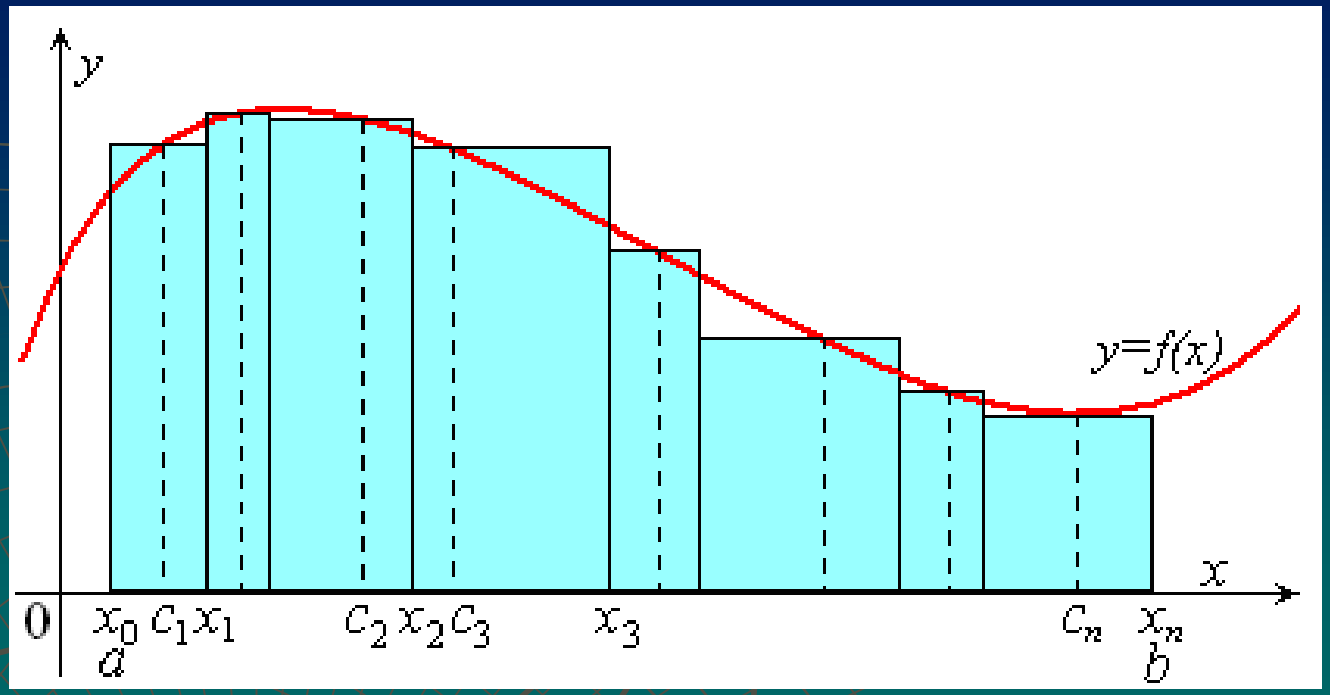
Пример. Найти приближенное значение интеграла.

Решение задачи следует из графического представления определенного интеграла. Пусть функция $f(x)$ определена и ограничена на отрезке $[a; b]$, где $a < b$.



Разобьем данный отрезок на n элементарных промежутков, таким образом, что $a = x_0 < x_1 < x_2 < \ldots < x_n = b$.

Обозначим $\Delta x_i = x_i - x_{i-1}$ длины получившихся отрезков, а на каждом из обозначенных отрезков $[x_{i-1}; x_i]$ выберем произвольным образом точки c_i.



Тогда сумму площадей n прямоугольников с длинами сторон Δx_i и $f(c_i)$.

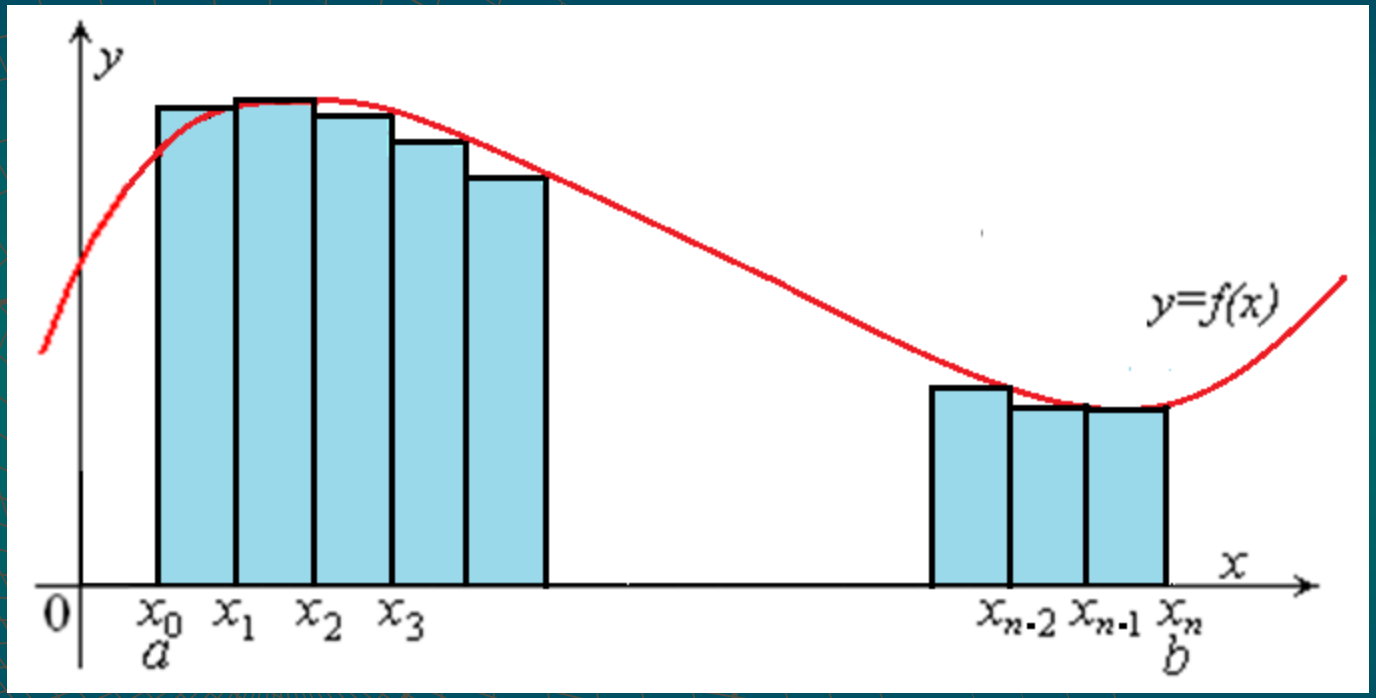
$$S_n = \sum_{i=1}^{n} f(c_i)(x_i - x_{i-1}) = \sum_{i=1}^{n} f(c_i)\Delta x_i$$

назовем ***интегральной суммой*** функции $f(x)$ на отрезке $[a; b]$.

Если количество разбиений n отрезка интегрирования $[a; b]$ на элементарные отрезки конечно, но достаточно велико, то, суммируя n площадей прямоугольников, получим приближенное значение интеграла.

Для простоты вычислений узлы интегрирования $a=x_0, x_1, x_2, \ldots, x_n=b$ можно разместить равномерно. В этом случае шаг интегрирования будет постоянен

$$h = x_i - x_{i-1} = \frac{b-a}{n}$$

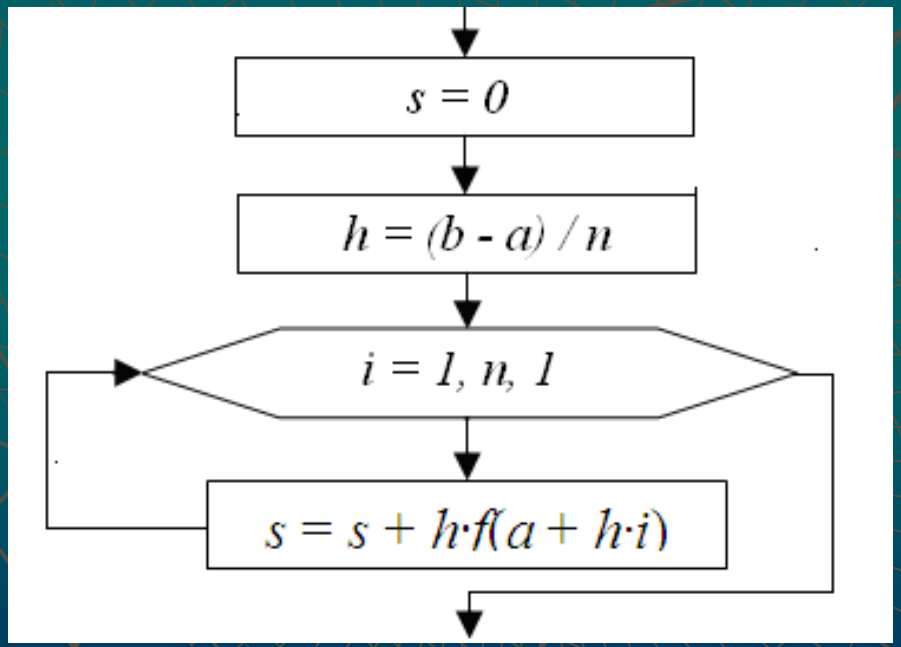


Для определенности точки c берут либо на левых границах элементарных отрезков, либо на правых границах, либо в середине отрезков. В зависимости от выбора точек способы приближенного интегрирования подразделяются на методы левых, правых и средних прямоугольников.

Возьмем метод правых прямоугольников. Задача нахождения интеграла сводится к нахождению суммы площадей прямоугольников

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} f(x_i)\frac{b-a}{n}.$$

Для организации автоматизированного вычисления следует организовать цикл с параметром.



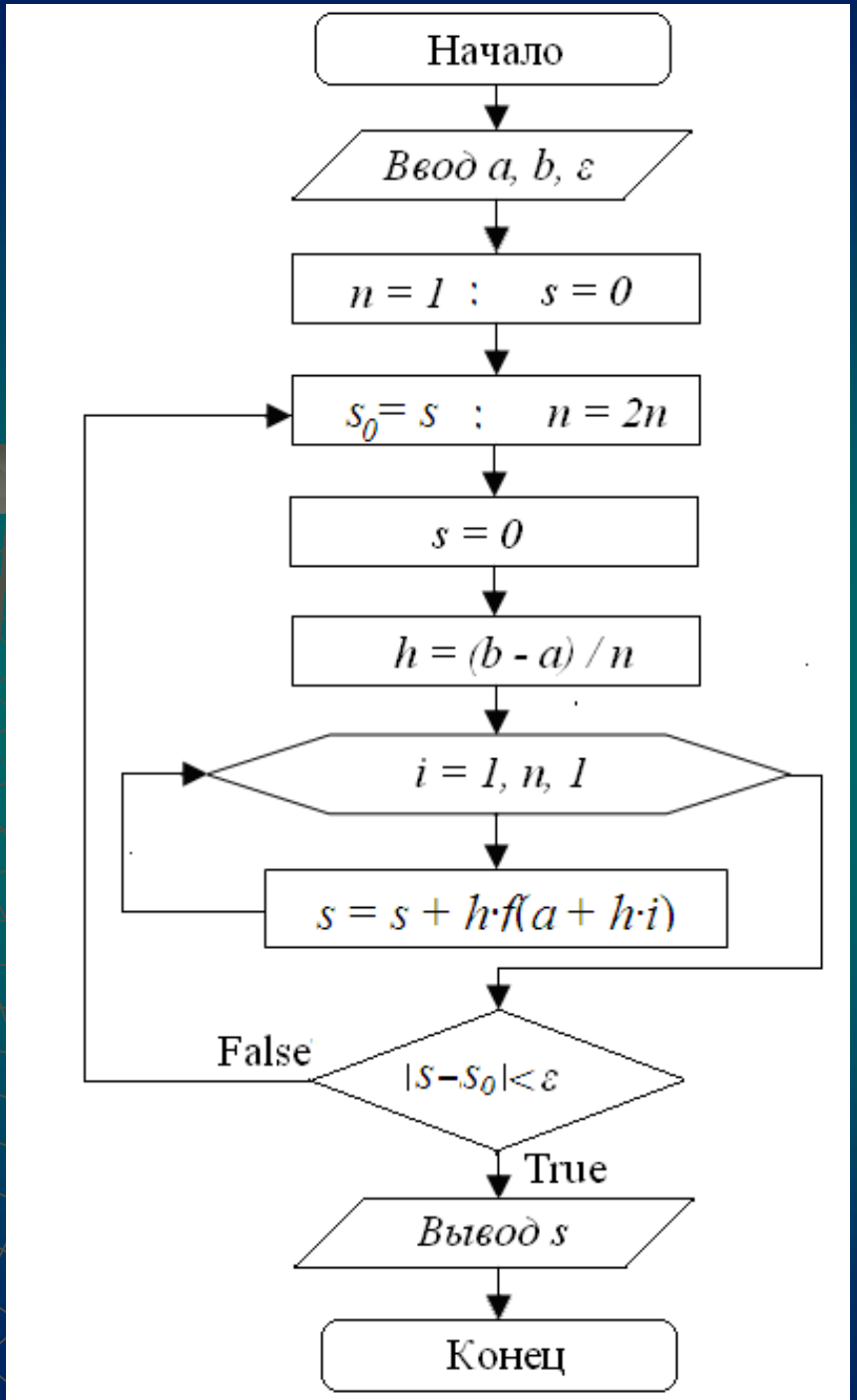

```
s = 0
h = (b - a) / n
For i = 1 To n
   s = s + h * f(a + h * i)
Next i
```

Задается начальное значение количества разбиений n отрезка $[a; b]$, например, $n = 10$.

Затем находим приближенное значение интеграла S_0 для n.

Увеличиваем количества разбиений n в два раза. Точность вычислений нового значения S будет больше.

Если абсолютное значение разности $|S-S_0|$ будет меньше заданной погрешности $\varepsilon>0$, то решение найдено, иначе снова увеличиваем количество разбиений n в два раза.



Код программы нахождения приближенного значения определенного интеграла методом правых прямоугольников с определением количества разбиений

```
Sub Метод_правых_прямоугольников()
' Вычисление приближенного значения определенного интеграла
' методом правых прямоугольников с определением количества разбиений
  a = InputBox("Введите левый предел интегрирования")
  b = InputBox("Введите правый предел интегрирования")
  epsilon = InputBox("Введите погрешность")
  s = 0: n = 1
  Do
    s0 = s
    n = n * 2
    s = 0
    h = (b - a) / n
    For i = 1 To n
      s = s + h * f(a + h * i)
    Next i
  Loop Until Abs(s - s0) < epsilon
  MsgBox "Значение интеграла " & s, , "Метод правых прямоугольников"
End Sub
```

4.6. Работа с таблицами

Пример. Вывести таблицу квадратов чисел от *0* до *99*.

Наша таблица будет содержать *100* чисел, поэтому для ее создания можно использовать *10* строк и *10* столбцов листа таблицы *Excel*.

Рассчитаем и выведем значения, использую два вложенных цикла с параметрами ***i*** и ***j***.

```
For i = 0 To 9
  For j = 0 To 9
    Cells(i + 1, j + 1) = (i + 10 * j) ^ 2
  Next j
Next i
```

В данном фрагменте кода расчетные значения определяются как квадрат выражения ***i*** + 10***j***., где ***i*** – это количество единиц выводимых чисел, а ***j*** – количество их десятков.

	1	2	3	4	5	6	7	8	9	10
1	0	100	400	900	1600	2 500	3600	4900	6400	8100
2	1	121	441	961	1681	2 601	3721	5041	6561	8281
3	4	144	484	1024	1764	2 704	3844	5184	6724	8464
4	9	169	529	1089	1849	2 809	3969	5329	6889	8649
5	16	196	576	1156	1936	2 916	4096	5476	7056	8836
6	25	225	625	1225	2025	3 025	4225	5625	7225	9025
7	36	256	676	1296	2116	3 136	4356	5776	7396	9216
8	49	289	729	1369	2209	3 249	4489	5929	7569	9409
9	64	324	784	1444	2304	3 364	4624	6084	7744	9604
10	81	361	841	1521	2401	3 481	4761	6241	7921	9801

Однако, на выводимом листе непонятно, каким образом формируется таблица. Добавим к выводимой таблице квадратов боковик и заголовок.

Для вывода боковика в цикле по ***i*** добавим строку, выводящую единицы:

```
For i = 0 To 9
  Cells(i + 2, 1) = i
  For j = 0 To 9
```

А для вывода заголовка в цикле по ***j*** добавим строку, выводящую десятки:

```
If i = 0 Then Cells(1, j + 2) = j * 10
```

Из–за условия ***i*** = *0* значения десятков будут выводиться только один раз.

Кроме того, выводимую таблицу следует передвинуть на одну строку вправо и на один столбец влево, поэтому изменились и координаты вывода на лист.

```
For j = 0 To 9
  If i = 0 Then Cells(1, j + 2) = j * 10
  Cells(i + 2, j + 2) = (i + 10 * j) ^ 2
Next j
```

Полный код программы «***Квадраты чисел***»

```
Sub Квадраты чисел()
'11 Вывести таблицу квадратов чисел от 0 до 99.

For i = 0 To 9
  Cells(i + 2, 1) = i
  For j = 0 To 9
    If i = 0 Then Cells(1, j + 2) = j * 10
    Cells(i + 2, j + 2) = (i + 10 * j) ^ 2
  Next j
Next i

End Sub
```

Результат выполнения программы «***Квадраты чисел***»

	1	2	3	4	5	6	7	8	9	10	11
1		0	10	20	30	40	50	60	70	80	90
2	0	0	100	400	900	1 600	2500	3600	4900	6400	8100
3	1	1	121	441	961	1 681	2601	3721	5041	6561	8281
4	2	4	144	484	1024	1 764	2704	3844	5184	6724	8464
5	3	9	169	529	1089	1 849	2809	3969	5329	6889	8649
6	4	16	196	576	1156	1 936	2916	4096	5476	7056	8836
7	5	25	225	625	1225	2 025	3025	4225	5625	7225	9025
8	6	36	256	676	1296	2 116	3136	4356	5776	7396	9216
9	7	49	289	729	1369	2 209	3249	4489	5929	7569	9409
10	8	64	324	784	1444	2 304	3364	4624	6084	7744	9604
11	9	81	361	841	1521	2 401	3481	4761	6241	7921	9801

Пример. Найти первые **n** чисел Ферма и определить их простоту.

Простое число называется ***числом Ферма***, если оно имеет вид $2^m + 1$.

Решение задачи сводится к следующему алгоритму:

В цикле с параметром выводим в таблицу порядковый номер числа Ферма, вычисляем и выводим значение числа Фермам с данным номером, а затем определяем его простоту и также выводим результат.

В начале программы определяем количество выводимых чисел Ферма.

```
Dim N As Byte
N = InputBox("Введите количество чисел Ферма")
```

Выводим заголовки для столбцов таблицы.

```
Cells(1, 1) = "№"
Cells(1, 2) = "Число Ферма"
```

Затем в цикле с параметром ***i*** выводим в первый столбец порядковый номер числа Ферма, а во второй столбец само значение рассчитанного числа.

```
For i = 1 To N
  Cells(i + 1, 1) = i
  F = 2 ^ i + 1
  Cells(i + 1, 2) = F

Next i
```

Для определения простоты рассчитанного числа ***F*** вложим в вышеприведенный цикл следующую структуру:

```
k = 0
For j = 2 To Sqr(F)
   f0 = F / j
   Do While f0 > 2000000000
      f0 = f0 - 2000000000
   Loop
   If f0 = Int(f0) Then k = k + 1
Next j
If k = 0 Then Cells(i + 1, 3) = "Простое" Else Cells(i + 1, 3) = "Составное"
```

Здесь в цикле с параметром ***j*** мы перебираем все возможные нетривиальные делители от двух до корня из ***F***. В случае отсутствия делителей (*k=0*) выводим в третий столбец «*Простое*» иначе «*Составное*».

Вложенный цикл *Do While .. Loop* используется из тех соображений, что функция *Int* не будет работать со значениями, превышающими максимальное значение типа ***Long***, и поэтому в теле цикла *Do While .. Loop* каждый раз происходит уменьшение значения f_0 на 2 000 000 000 в случае такого превышения.

Полный код программы
«*Нахождение чисел Ферма и определение их простоты*»

```
'10.1. Нахождение чисел Ферма, определение их простоты
Sub чисда_Ферма()

Dim N As Byte
N = InputBox("Введите количество чисел Ферма")

Cells(1, 1) = "№"
Cells(1, 2) = "Число Ферма"
For i = 1 To N
  Cells(i + 1, 1) = i
  F = 2 ^ i + 1
  Cells(i + 1, 2) = F
  k = 0
  For j = 2 To Sqr(F)
     f0 = F / j
     Do While f0 > 2000000000
        f0 = f0 - 2000000000
     Loop
     If f0 = Int(f0) Then k = k + 1
  Next j
  If k = 0 Then Cells(i + 1, 3) = "Простое" Else Cells(i + 1, 3) = "Составное"
Next i

End Sub
```

Результат выполнения программы «***Нахождение чисел Ферма и определение их простоты***».
(верхняя часть таблицы, содержащая первые 27 значений)

	1	2	3
1	№	Число Ферма	
2	1	3	Простое
3	2	5	Простое
4	3	9	Составное
5	4	17	Простое
6	5	33	Составное
7	6	65	Составное
8	7	129	Составное
9	8	257	Простое
10	9	513	Составное
11	10	1 025	Составное
12	11	2 049	Составное
13	12	4 097	Составное
14	13	8 193	Составное
15	14	16 385	Составное
16	15	32 769	Составное
17	16	65 537	Простое
18	17	131 073	Составное
19	18	262 145	Составное
20	19	524 289	Составное
21	20	1 048 577	Составное
22	21	2 097 153	Составное
23	22	4 194 305	Составное
24	23	8 388 609	Составное
25	24	16 777 217	Составное
26	25	33 554 433	Составное
27	26	67 108 865	Составное
28	27	134 217 729	Составное

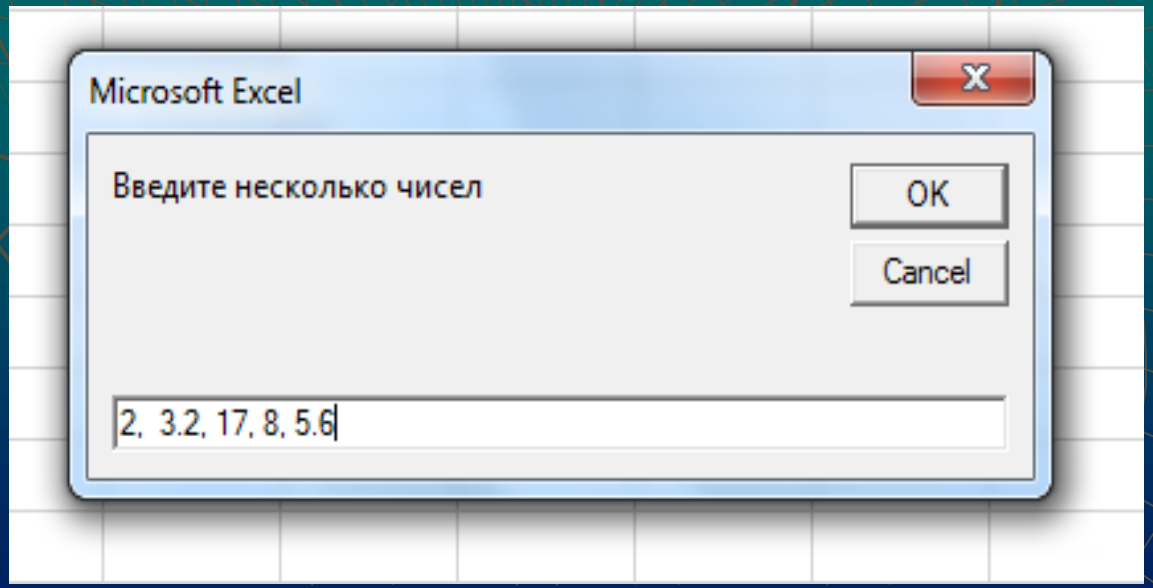
4.7. Ввод и обработка списков

Пример. Выполнить преобразование списка числовых данных, рассчитать сумму чисел и среднее арифметическое значение.

При обработке списков числовых данных, например, введенных в виде строки с помощью функции ***InputBox***, существенной проблемой является распознавание чисел, находящихся в строке.

Условимся, что пользователь будет вводить данные в определенном понятном ему порядке, используя вместо разделителей запятые или точки с запятой.

Тогда процесс ввода может выглядеть следующим образом:

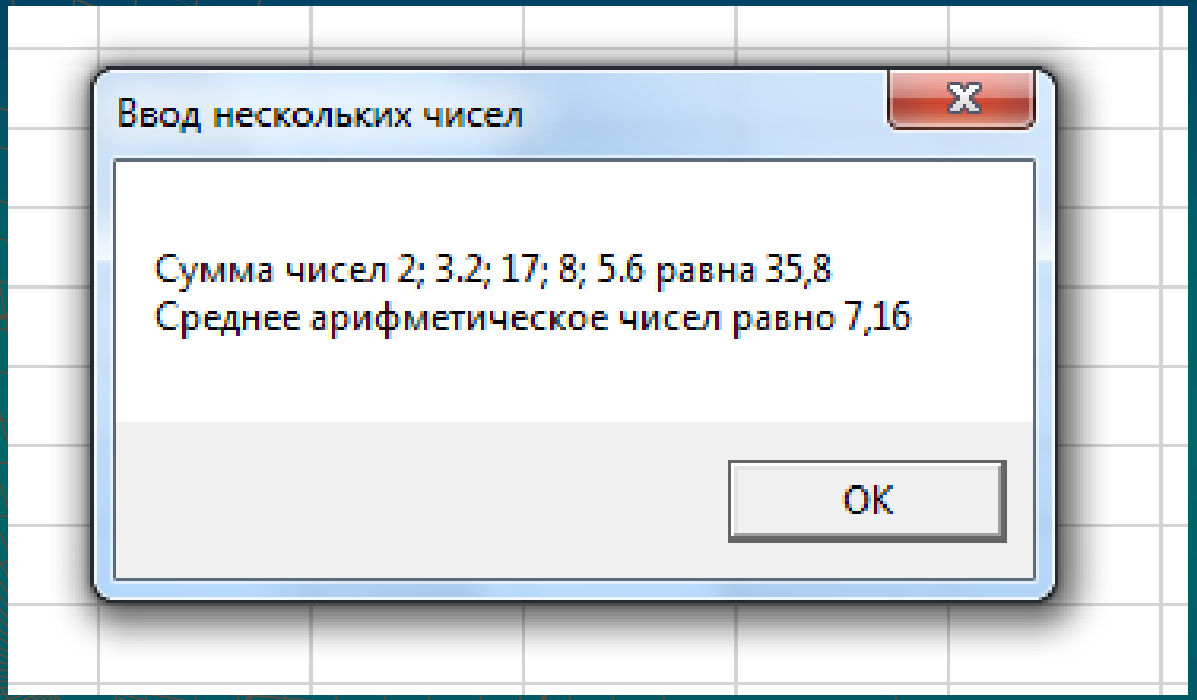


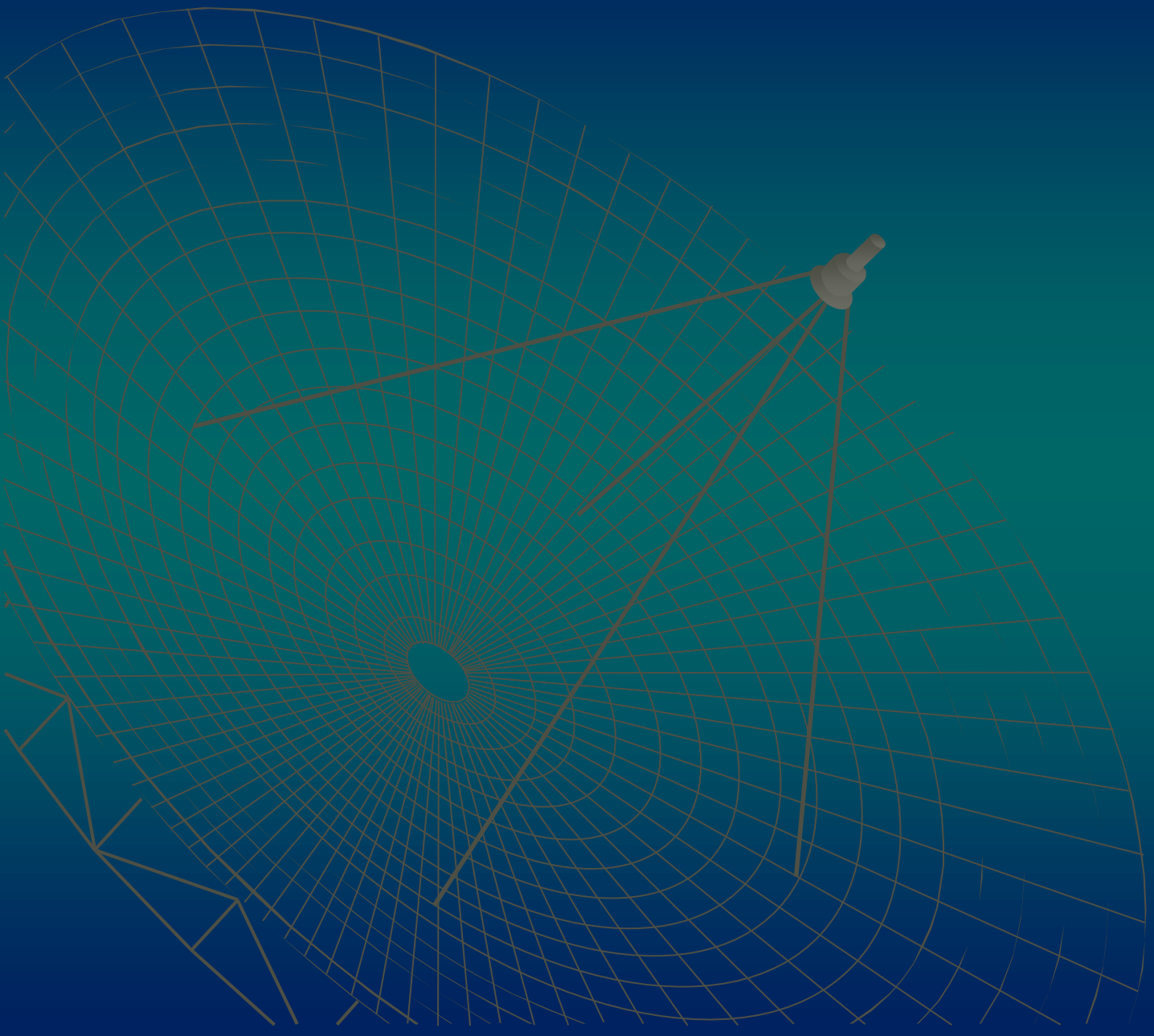
Необходимо отделить в строковой записи каждое из введенных чисел, преобразовать их в числовой формат, а затем использовать полученные числовые значения для вычислений.

Алгоритм извлечения чисел можно реализовать в двух вложенных числах: внутренний цикл будет использоваться для подсчета количества символов в каждом числе, а внешний - для отделения чисел, их преобразования в числовой формат и выполнения расчетов.

```
Sub Числа_из_списка()

t = InputBox("Введите несколько чисел")
r = "Сумма чисел"
Do Until t = ""
  i = 0
  Do
    i = i + 1
  Loop Until Mid(t, i, 1) = ";" Or _
             Mid(t, i, 1) = "," Or i = Len(t)
  a = Val(t)
  k = k + 1
  s = s + a
  r = r + Str(a) + ";"
  t = Mid(t, i + 1)
Loop
sa = s / k
r = Mid(r, 1, Len(r) - 1)
y = MsgBox(r & " равна " & s & vbCr & _
    "Среднее арифметическое чисел равно " & s / k _
    , , "Ввод нескольких чисел")

End Sub
```

Результат выполнения программы «***Числа из списка***».

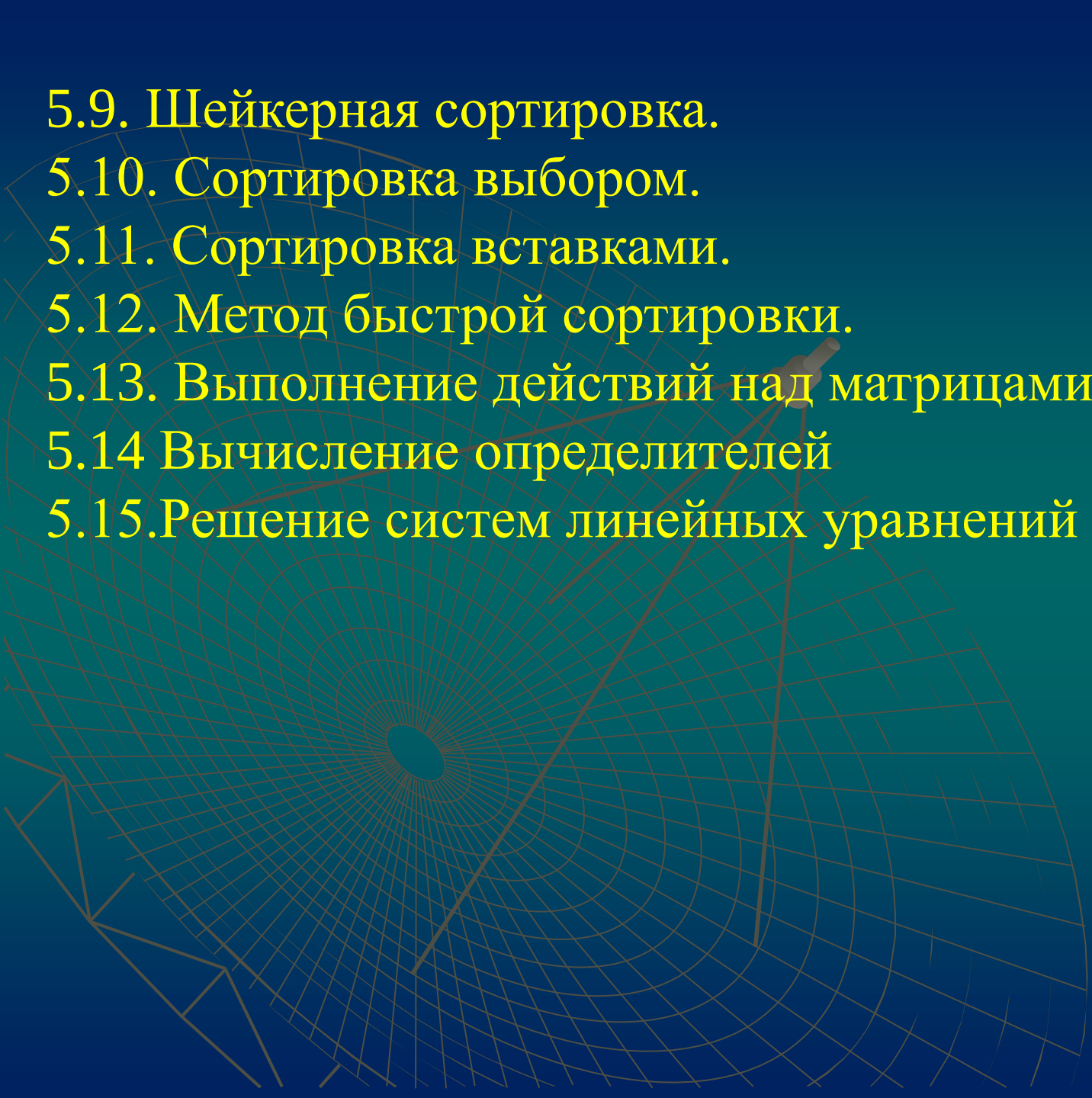


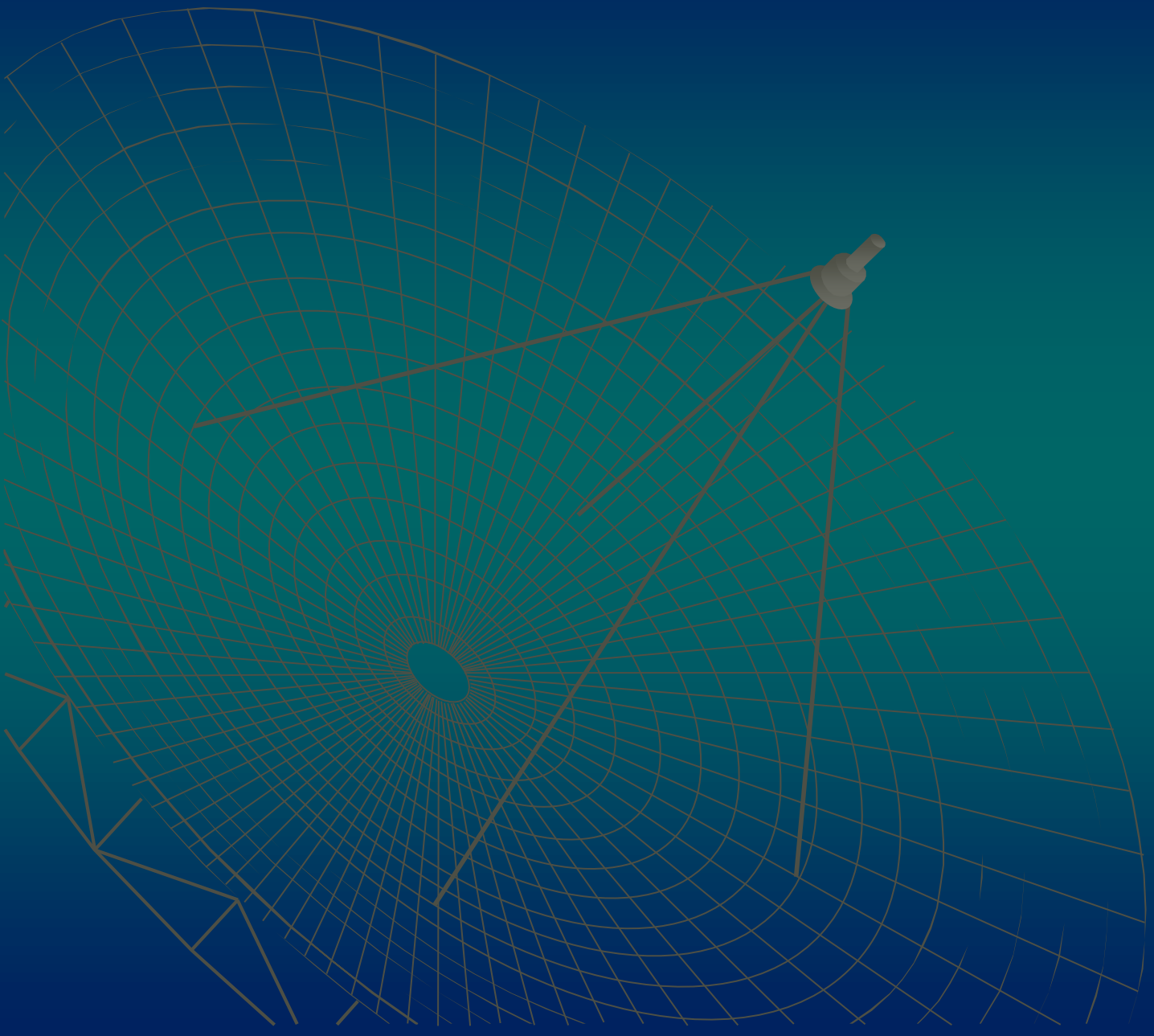


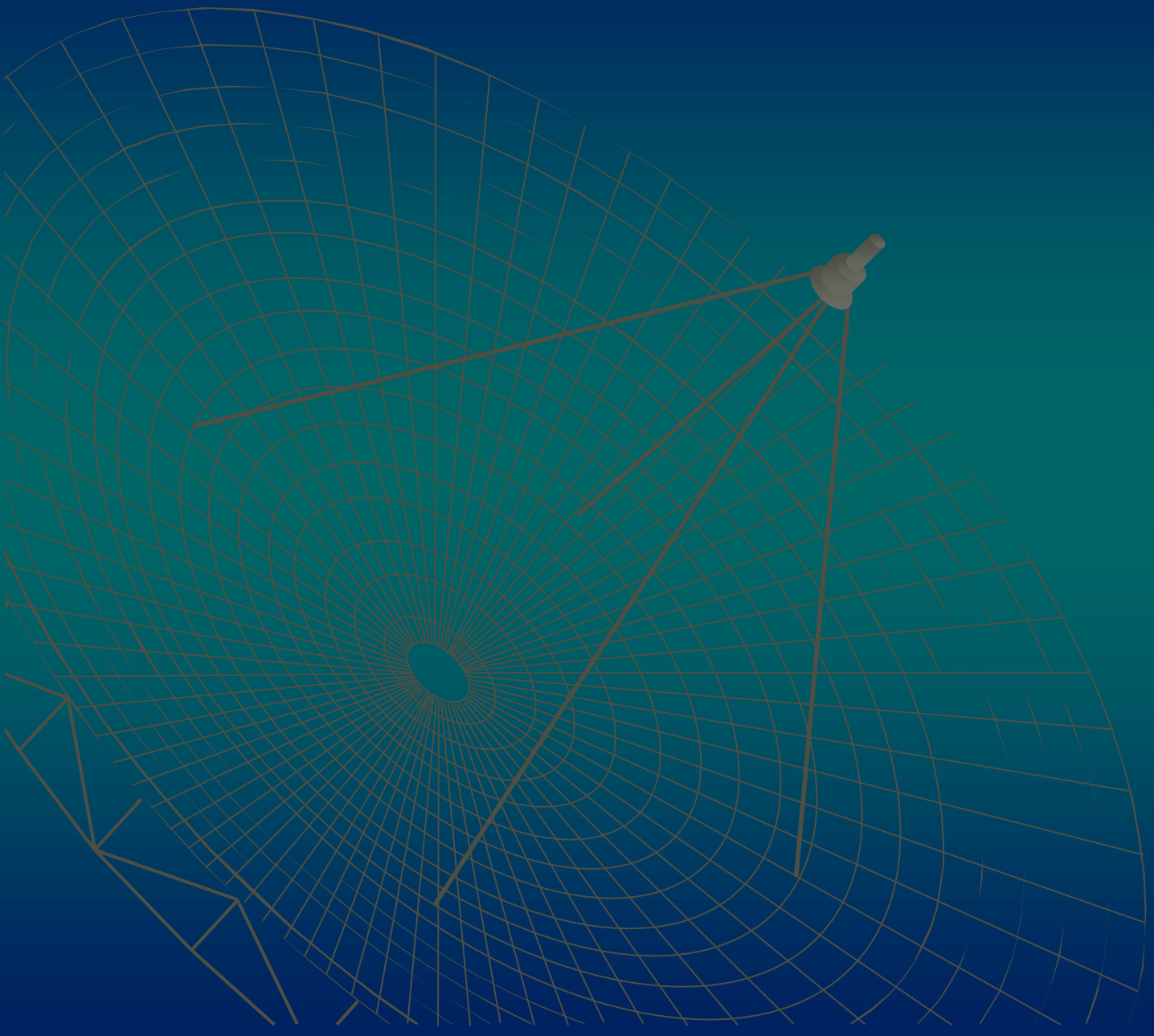
Глава 5. Обработка массивов

5.1. Общие сведения о массивах
5.2. Описание массивов
5.3. Операции над массивами
5.4. Ввод-вывод элементов массива
5.5. Вычисление суммы и произведения элементов массива
5.6. Поиск максимального элемента в массиве и его номера
5.7. Сортировка элементов.
5.8. Сортировка простыми обменами.

5.9. Шейкерная сортировка.
5.10. Сортировка выбором.
5.11. Сортировка вставками.
5.12. Метод быстрой сортировки.
5.13. Выполнение действий над матрицами
5.14 Вычисление определителей
5.15.Решение систем линейных уравнений







5.1. Общие сведения о массивах

При обработке однотипных данных (числовых значений, строк, дат, ….) оказывается удобным использовать массивы. Вместо создания множества ***n*** переменных для хранения каждого значения можно использовать одну структуру с уникальным именем, где каждому значению будет соответствовать его порядковый номер.

№ элемента	1	2	3	4	5	6	7
Значение	**56,68**	**23,32**	**- 0,38**	**25,75**	**9,38**	**- 5,76**	**2,78**

Массив **(*array*)** - это структурированный тип данных, состоящий из фиксированного числа элементов одного типа.

Массив, представленный в таблице, имеет 7 элементов, каждый элемент сохраняет число вещественного типа, элементы в массиве пронумерованы от 1 до 7.

Такого рода массив, представляющий собой просто список данных одного и того же типа, называют простым, или **одномерным массивом**. Для доступа к данным, хранящимся в определённом элементе массива, необходимо указать **имя массива** и **порядковый номер** этого элемента, называемый ***индексом***.

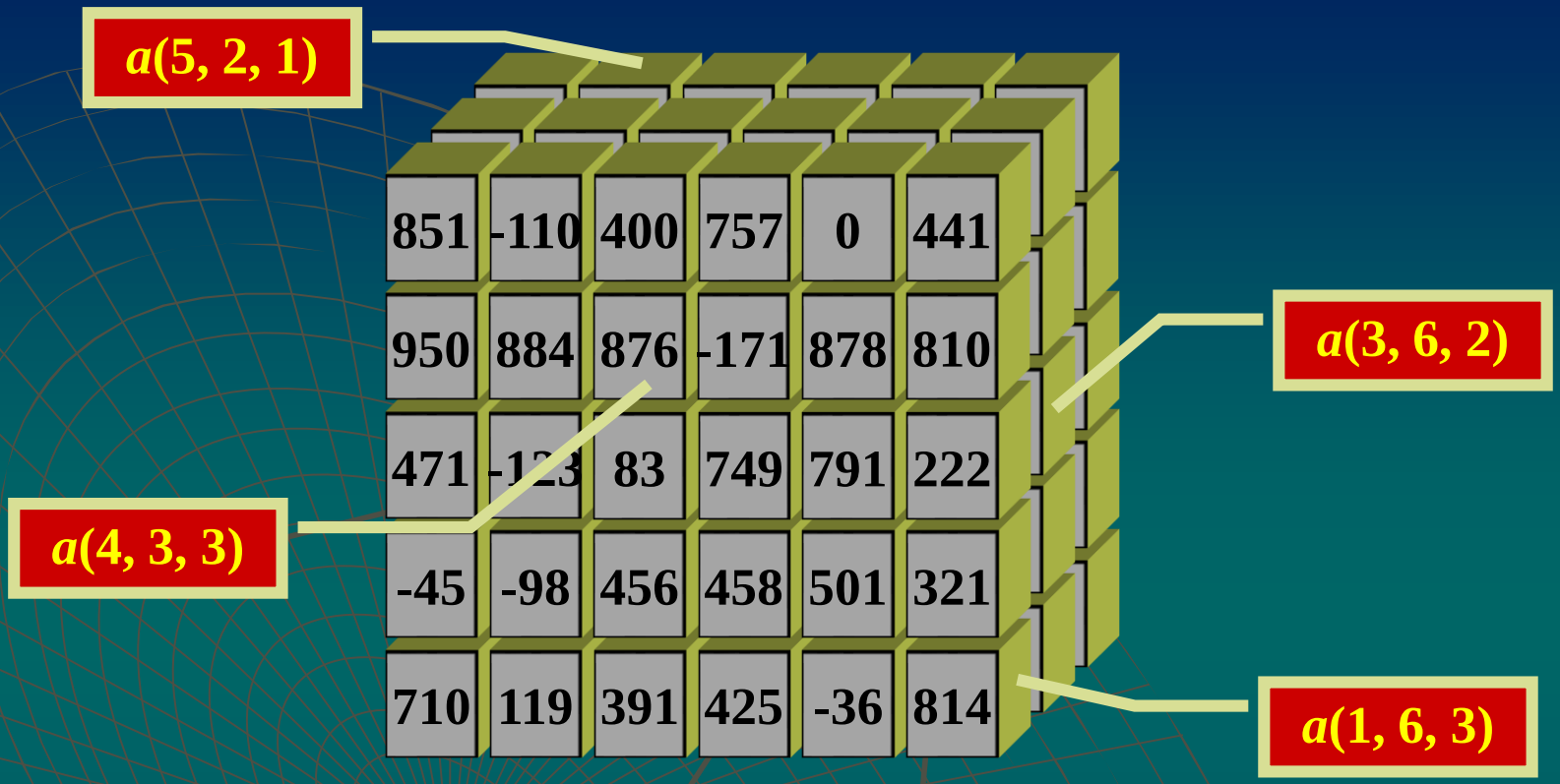
Если возникает необходимость хранения данных в виде таблиц, в формате строк и столбцов, то необходимо использовать **двумерные массивы**.

	1	2	3	4	5
1	**- 65**	**73**	**42**	**- 4**	**37**
2	**17**	**- 20**	**0**	**43**	**36**
3	**3**	**13**	**27**	**- 46**	**2**
4	**51**	**41**	**1**	**12**	**31**
5	**5**	**15**	**- 15**	**6**	**21**
6	**- 47**	**- 9**	**66**	**- 11**	**6**
7	**55**	**5**	**- 5**	**53**	**14**

В таблице приведён пример массива, состоящего из 7 строк и 5 столбцов. Это ***двумерный*** массив. Строки в нём можно считать первым измерением, а столбцы вторым.

Для доступа к данным, хранящимся в этом массиве, необходимо указать имя массива и два индекса: первый должен соответствовать номеру строки, а второй номеру столбца, в которых хранится необходимый элемент.

Можно хранить данные и в **трехмерном** массиве,



На экране приведён пример массива, состоящего из 5 строк, 6 столбцов и 3 слоев. Это **трехмерный** массив. Строки в нём можно считать первым измерением, столбцы - вторым, а слои - третьим. Для доступа к данным, хранящимся в элементе массива, необходимо указать имя массива и три индекса: номер строки, номер столбца и номеру слоя.

Можно представить массив и большей размерности!!!

5.2. Объявление массивов

Для объявления массива служит инструкция **Dim**.

Структура оператора **Dim**:

Dim *varname*[(([*subscripts*])] [***As*** *type*]

В этой записи

Dim, As – служебные слова;

Varname – имя массива, любой уникальный идентификатор;

Subscripts – индексы для указания размерности массива;

Type – тип данных, содержащихся в массиве.

Пример.

```
Dim a(1000) As Integer
Dim b(10 To 19) As Double
Dim x1(5, 7) As Byte, x2(3, 12) As Byte
Dim Q(1 To 5, 3 To 7, 101 To 120) As Long
Dim  FName(100, 100, 100, 100) As String
```

Объявленные в примере массивы называются ***статическими***, количество элементов в них изменить нельзя.

Иногда необходимо в процессе работы изменять размер массива. Для этого используют ***динамические массивы***.

Пример. Рассмотрим фрагмент программного кода.

```
Dim x()

n1 = 20
ReDim x(n1)
For i = 0 To n1
   x(i) = i ^ 2
Next i

n2 = 40
ReDim Preserve x(n2)
For i = 0 To n2
   x(i) = x(i) - x(n2 - i)
Next i

ReDim x(40, 20)

ReDim Preserve x(40, 50)
```

Сперва инструкцией ***Dim*** массив объявляется без указания размерностей, а затем в в коде программы с помощью инструкции ***ReDin*** можно ***неоднократно*** устанавливать новую размерность массива (как увеличивать ее, так и уменьшать).

Команда ***ReDim*** не только изменяет размер массива, но и удаляет из него все старые значения. Чтобы старые значения сохранить, используется ключевое слово ***Preserve*.**

Для сохранения данных массива ***можно изменять только значение последнего измерения***, в противном случае при запуске программы отобоазитя ошибка «***Subscript out of range***»/

5.3. Операции над массивами

Для работы с массивом как с единым целым необходимо использовать имя массива (без указания индексов в скобках).

При создании массива можно сразу заполнить его начальными значениями. Для этого используется функция ***Array***.

Пример.

```
Dim x, y
x = Array(1.2, 2.4, 0.3, 1.4)
```

После объявления ***x*** с помощью функции ***Array*** переменная ***x*** преобразуется в одномерный массив, содержащий 4 элемента, с начальными значениями: $x_0 = 1{,}2$; $x_1 = 2{,}4$; $x_2 = 0{,}3$; $x_3 = 1{,}4$.

Очистить массив можно командой ***Erase***.

```
Erase x
```

При этом ***для статических массивов значения элементов массива очищаются***, а ***динамический массив разинициализируется***, то есть его размерность станет нулевой и он превратится в переменную.

Для определения количества элементов массива в каком-либо измерении используется функция ***Ubound***.

UBound (имяМассива [, измерение])

Если массив одномерный, то измерение передавать не надо.

Для доступа к элементу массива необходимо указать имя массива и в скобках порядковый номер элемента массива.

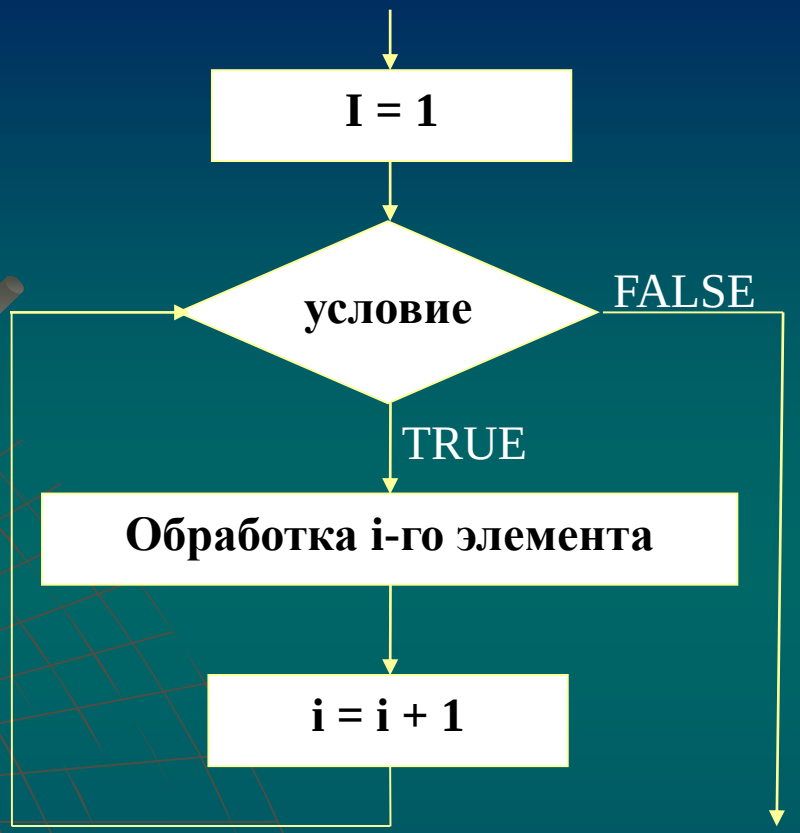
Пример

```
a(3) = f(x)
b(i) = i ^ 2
x1(i, j) = x2(i, j) * (-1) ^ (i + j)
Q(i, j, k) = Q(i, j, k) * p1(k) + p2(2) + p3(k)
```

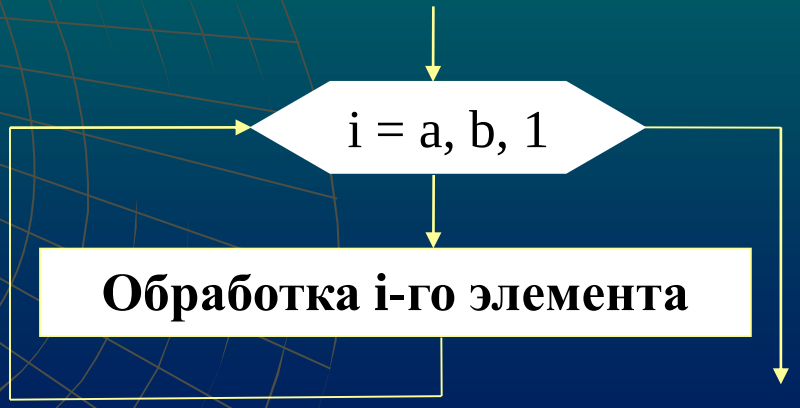
Выполнение большинства операций над массивом можно организовывать поэлементно, для чего необходимо организовать цикл, в котором последовательно обрабатывать элементы массива; сначала обрабатываем 1-й элемент массива, затем 2-й, 3-й, ... , n-й.

Для обработки элементов массива удобно использовать циклы с предусловием, либо цикл с управляющей переменной.

Блок-схема обработки элементов массива с использованием цикла с предусловием



Алгоритм обработки массива с использованием цикла с управляющей переменной



5.4. Ввод-вывод элементов массива

Язык VBA не имеет специальных средств ввода-вывода всего массива, поэтому данную операцию следует организовывать поэлементно.

При вводе массива необходимо последовательно вводить 1-й, 2-й, 3-й и т.д. элементы массива, аналогичным образом поступить и при выводе.

Следовательно, как для ввода (вывода) необходимо организовать стандартный цикл обработки массива. Для обращения к элементу массива необходимо указать имя массива и в скобках номер элемента.

Пример. Введем одномерный массив a.

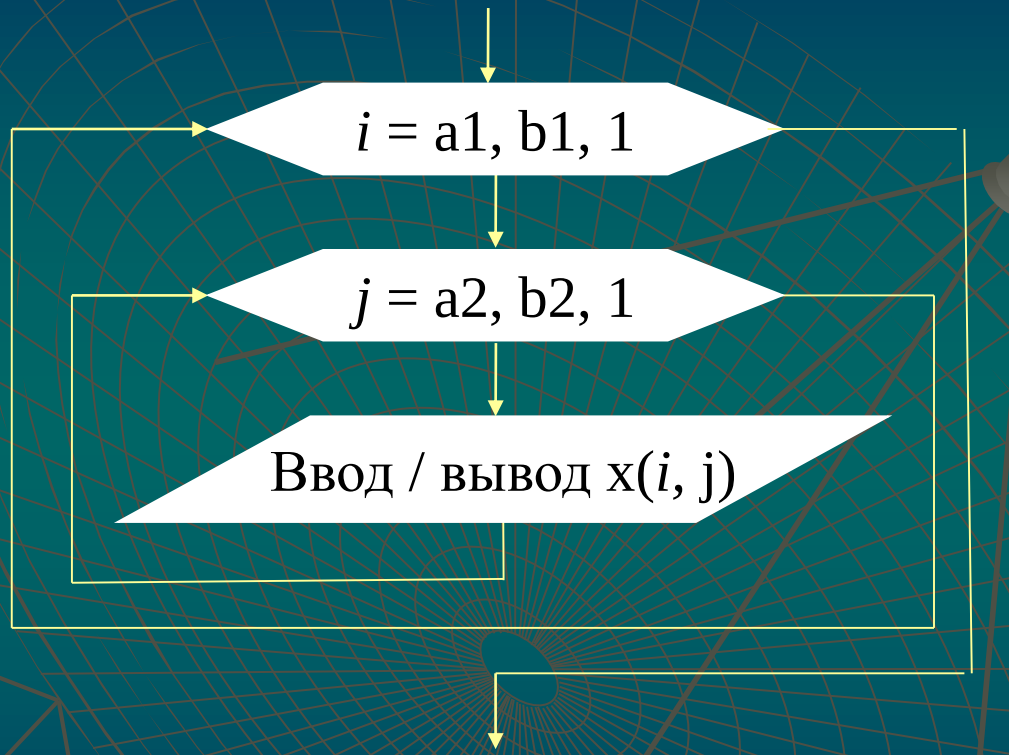
```
Sub Ввод_одномерного_массива()
Dim a(1 To 20) As Integer

For i = 1 To 5
  a(i) = InputBox("введите элемент")
Next i

End Sub
```

```
Sub Ввод_одномерного_массива()
  Dim a(1 To 20) As Integer

  For i = 1 To 20
     a(i) = Cells(2, i + 1)
  Next i

End Sub
```

Вывод массива организуется аналогично вводу, только вместо блока ввода элемента массива будет блок вывода.

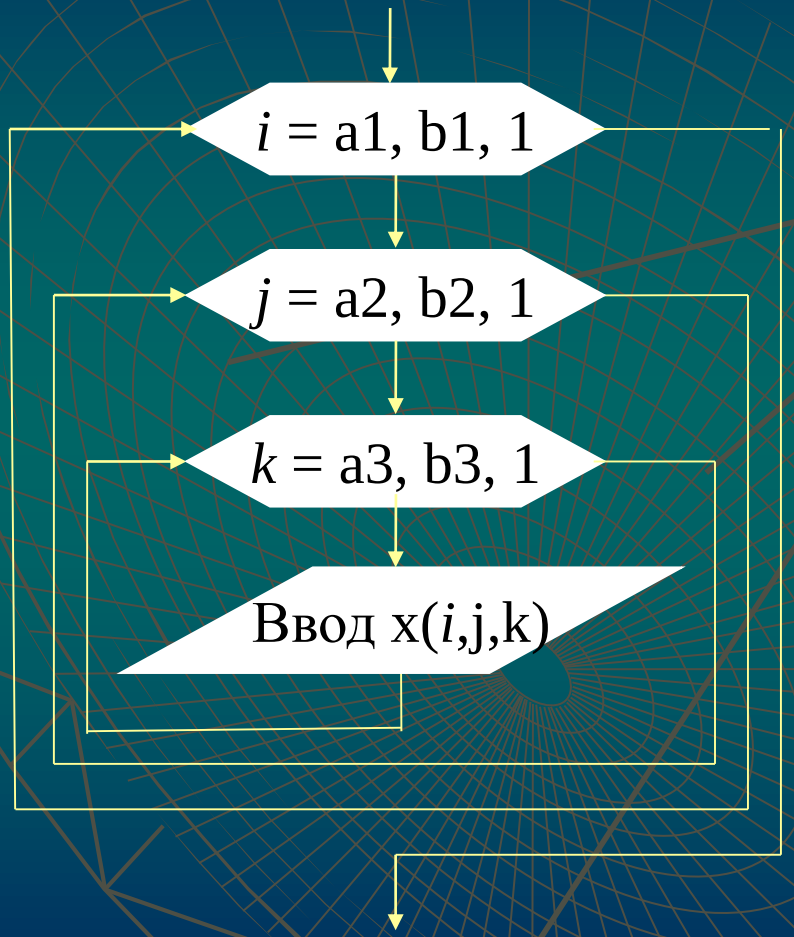
```
Sub Вывод_одномерного_массива()
Dim a(1 To 20) As Integer

For i = 1 To 20
  a(i) = Rnd(i) * 200 - 100
Next i

For i = 1 To 20
  Cells(1, i) = a(i)
Next i

End Sub
```

Для организации двумерного массива требуются два вложенных друг в друга циклов.




```
Sub Двумерный_массив()
Dim x(10, 8) As Integer

For i = 1 To 10
  For j = 1 To 8
    x(i, j) = Rnd(i) * 200 - 100
Next j, i

For i = 1 To 10
  For j = 1 To 8
    x(i, j) = Abs(x(i, j))
Next j, i

For i = 1 To 10
  For j = 1 To 8
    Cells(i + 2, j) = x(i, j)
Next j, i

End Sub
```

Аналогично выполняется действия и с трехмерным массивом.

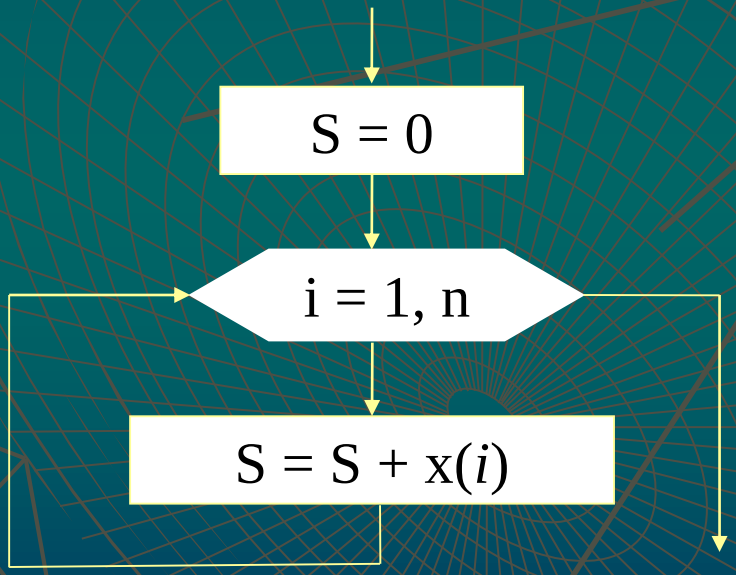



```
Sub Трехмерный_массив()
Dim x(10, 8, 9) As String

For i = 1 To 4
  For j = 1 To 6
    For k = 1 To 9
      x(i, j, k) = Chr(Rnd(i * j * k) * 254 + 1)
Next k, j, i

For i = 1 To 4
  For j = 1 To 6
    For k = 1 To 9
      Cells(j + (i - 1) * 7 + 1, k) = x(i, j, k)
Next k, j, i

End Sub
```

5.5. Вычисление суммы и произведения элементов массива

Дан массив X, состоящий из n элементов. Найти сумму элементов этого массива.

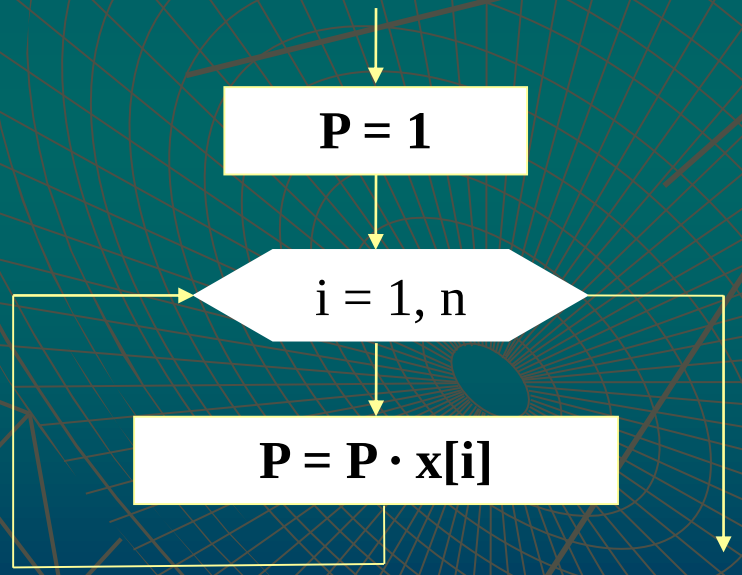
Переменной S присваивается значение, равное нулю, затем последовательно суммируются элементы массива X.



Соответствующий алгоритму фрагмент программы будет иметь вид:

```
Sub сумма_элементов()

Dim x(12) As Integer
n = InputBox("Введите N")
S = 0

For i = 1 To n
  x(i) = Rnd(i) * 2000 - 1000
  Cells(1, i) = x(i)
Next

For i = 1 To n
  S = S + x(i)
Next

Cells(2, 1) = "S = " + Str(S)"
End Sub
```

Найдём произведение элементов массива X. Решение задачи сводится к тому, что значение переменной P, в которую предварительно была записана единица, последовательно умножается на значение *i*-го элемента массива.



Соответствующий фрагмент программы будет иметь вид

```
Sub Произведение_элементов()
'
Dim x(12) As Integer

n = InputBox("Введите M")
P = 1

For i = 1 To n
  x(i) = Rnd(i) * 20 - 10
  Cells(4, i) = x(i)
Next

For i = 1 To n
  P = P * x(i)
Next

Cells(5, 1) = "P =" + Str(P)
End Sub
```

5.6.Поиск максимального элемента в массиве и его номера

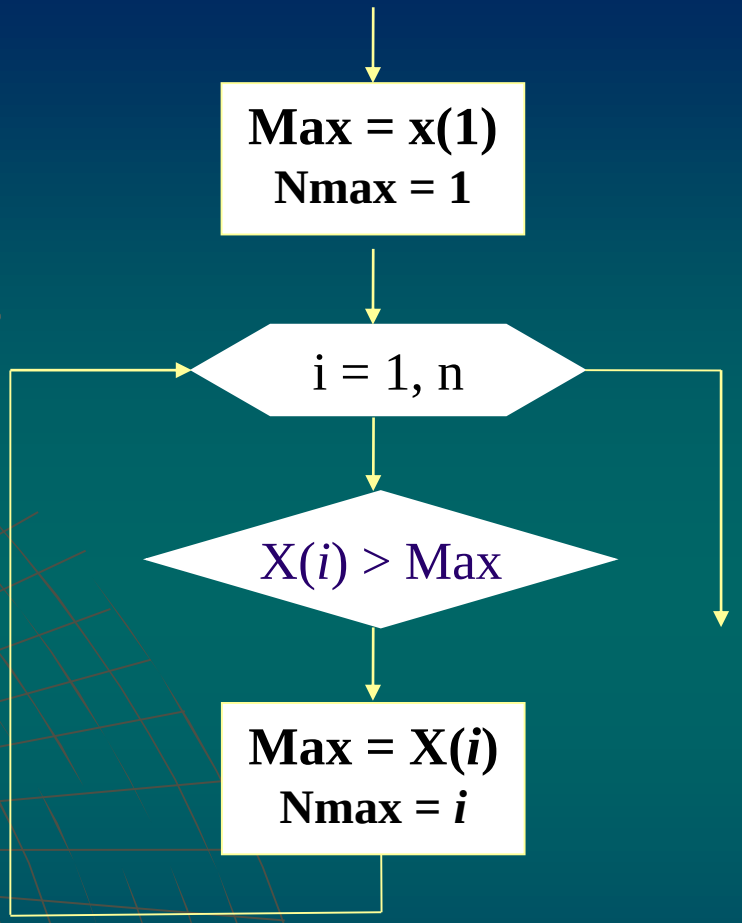
Рассмотрим задачу поиска максимального элемента (Max) и его номера (Nmax) в массиве X, состоящем из n элементов.

Алгоритм решения задачи следующий.

Предположим, что первый элемент массива является максимальным, и запишем его в переменную Max, а в Nmax его номер (число ***i***).

Затем все элементы, начиная со второго, сравниваем в цикле с максимальным.

Если текущий элемент массива оказывается больше максимального, то записываем его в переменную Max, а в переменную Nmax текущее значение индекса i.



Алгоритм поиска максимального элемента массива и его номера

```
Sub Максимальный_элемент()
'
Dim x(200) As Integer

n = InputBox("Введите количество чисел")
For i = 1 To n
  x(i) = Rnd(i) * 20 - 10
  Cells(7, i) = x(i)
Next

Max = x(i)
Nmax = 1
For i = 2 To n
  If x(i) > Max Then Max = x(i): Nmax = i
Next

Cells(8, 1) = "Максимальный элемент " + Str(Max)
Cells(9, 1) = "Номер максимального элемента " + Str(Nmax)

End Sub
```

5.7. Сортировка элементов

Сортировка представляет собой процесс упорядочения элементов в массиве в порядке возрастания или убывания их значений.

При сортировке **по возрастанию** после перестановки элементы с меньшими индексами должны быть не больше элементов с большими индексами т.е. $x_1 \le x_2 \le \ldots \le x_n$.

Пример. Отсортируем по возрастанию элементы массива:

Номер элемента	1	2	3	4	5	6
Исходный массив	12	-3	7	2	-16	8
После сортировки	-16	-3	2	7	8	12

При сортировке **по убыванию** после перестановки элементы с меньшими индексами должны быть не меньше элементов с большими индексами т.е. $x_1 \ge x_2 \ge \ldots \ge x_n$.

Пример. Отсортируем по убыванию элементы массива:

Номер элемента	1	2	3	4	5	6
Исходный массив	12	3	7	2	16	8
После сортировки	12	8	7	2	-3	-16

5.8. Пузырьковый метод

Наиболее известным методом сортировки массивов является **сортировка простыми обменами** (пузырьковым методом, англ. ***bubble sort***). Её популярность объясняется запоминающимся названием и простотой алгоритма.

Сортировка методом «пузырька» основана на выполнении в цикле операций сравнения и при необходимости обмена соседних элементов.

Алгоритм состоит из повторяющихся N-1 проходов «слева направо» по сортируемому массиву.

За каждый проход последовательно все соседние элементы попарно сравниваются и, если порядок в паре неверный, выполняется обмен элементов местами.

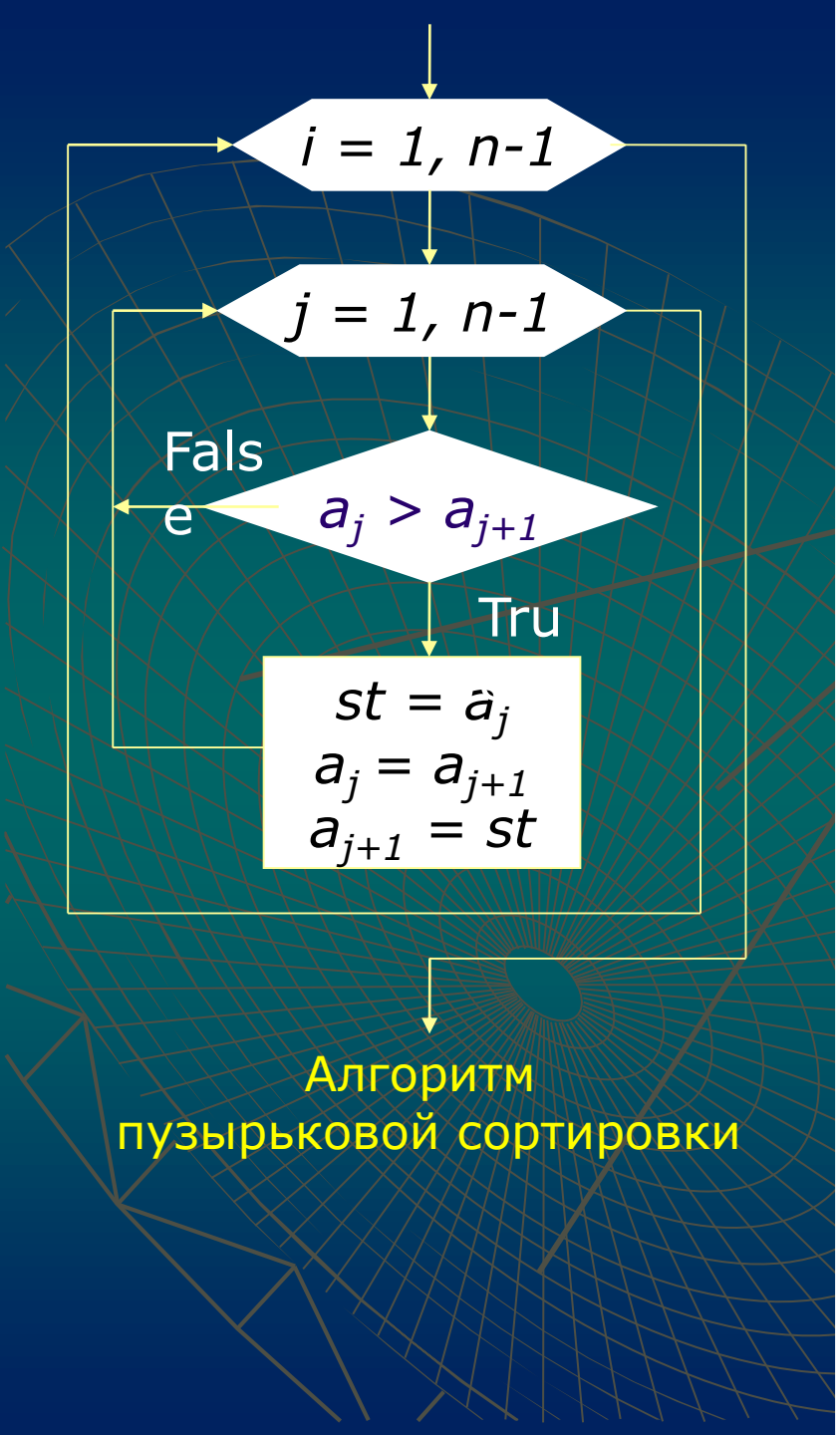
Очередной наибольший элемент оставшейся части массива ставится на своё место в конце массива, а все остальные элементы переместятся на одну позицию к началу массива.

Рассмотрим алгоритм пузырьковой сортировки на примере сортировки по возрастанию.

Номер элемента	1	2	3	4	5	6
Исходный массив	12	-3	7	2	-16	8
1-й проход	-3	7	2	-16	8	12
2-й проход	-3	2	-16	7	8	12
3-й проход	-3	-16	2	7	8	12
4-й проход	-16	-3	2	7	8	12
5-й проход	-16	-3	2	7	8	12

Для обмена двух элементов в массиве необходимо использовать буферную переменную, в которой временно хранится значение элемента, подлежащего замене.

Алгоритм считается учебным и практически не применяется вне учебной литературы, в то же время метод сортировки обменами лежит в основе некоторых более совершенных алгоритмов.



Алгоритм пузырьковой сортировки

```
Sub Пузырьковая_сортировка()
Dim a(100) As Integer
Randomize
n = InputBox("Введите кол-во элементов")
For i = 1 To n                    'Ввод элементов
    a(i) = Rnd(Timer) * 200 - 100
    Cells(1, i) = a(i)
Next i
For i = 1 To n - 1
          'Сортировка
  For j = 1 To n - 1
    If a(j) > a(j + 1) Then
      st = a(j)
      a(j) = a(j + 1)
      a(j + 1) = st
    End If
  Next j
  For j = 1 To n
    Cells(i + 1, j) = a(j)
  Next i
```

5.9. Шейкерная сортировка

Шейкерная сортировка (перемешивание, Cocktail sort) является модификацией пузырькового метода.

В методе пузырьковой сортировки отметим два обстоятельства.

1. Если при движении по части массива перестановки не происходят, то эта часть массива уже отсортирована и ее **можно не сортировать**.

2. При движении от начала массива к концу **минимальный элемент** становится на первую позицию, а **максимальный элемент** сдвигается только на одну позицию вправо.

Учитывая это, будем использовать 2 правила:

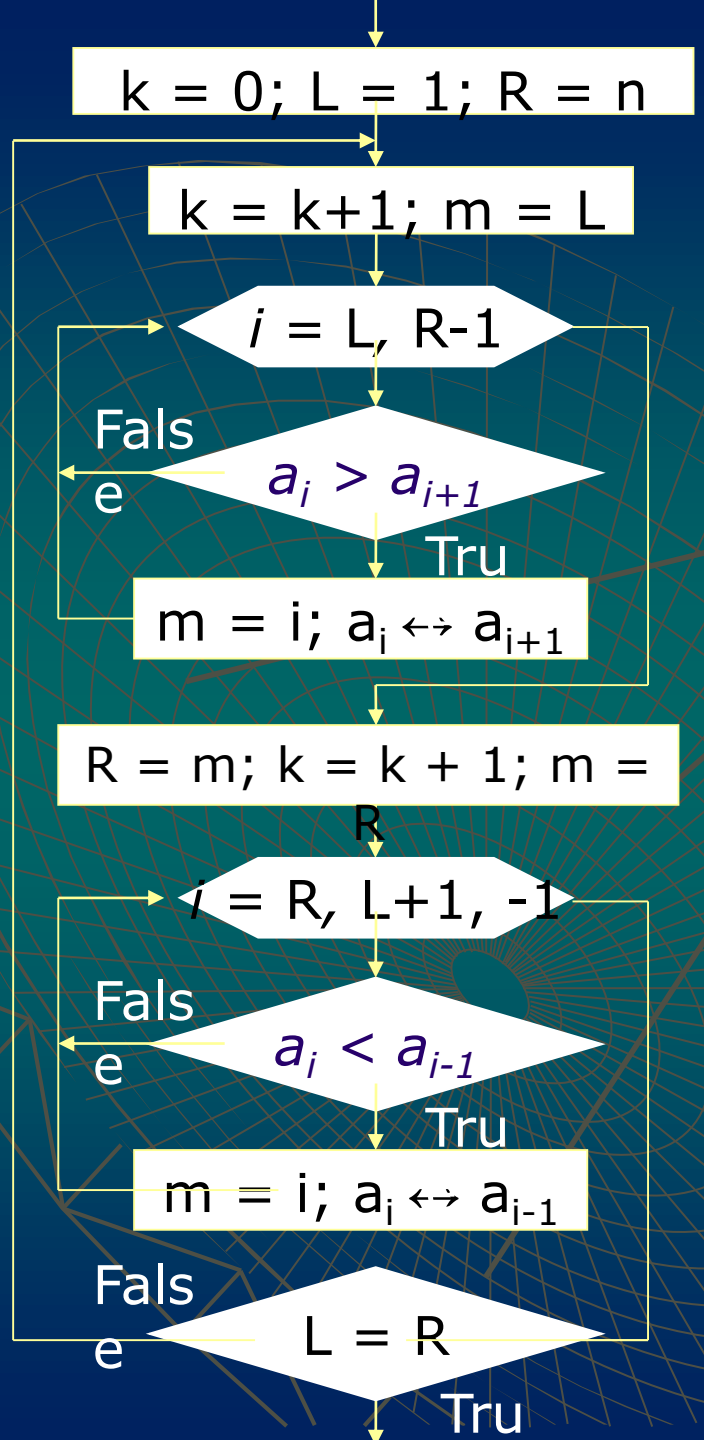
1. Границы рабочей части массива (т.е. части массива, где происходит движение) устанавливаются **в месте последнего обмена** по каждому проходу.

2. Массив просматривается поочередно **слева направо** и **справа налево**.

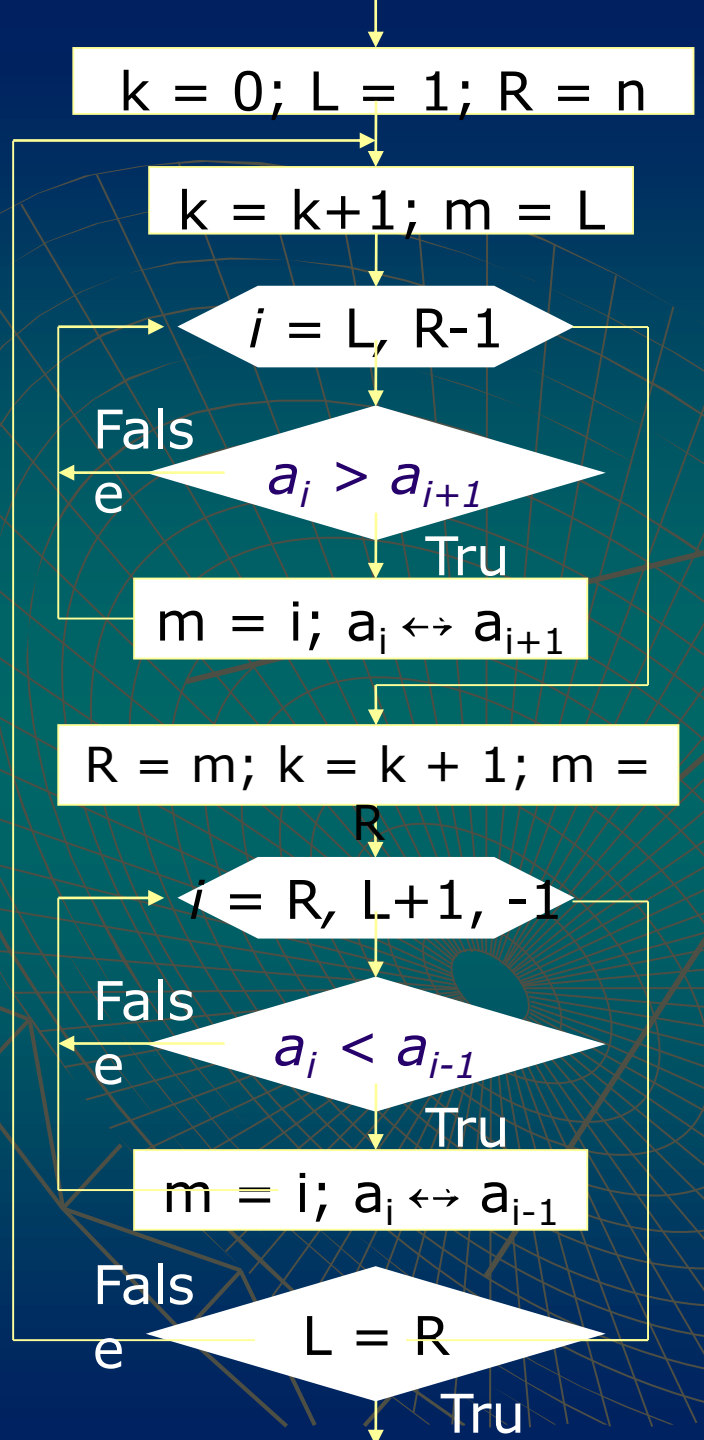
Рассмотрим алгоритм шейкерной сортировки на нашем примере.

Номер элемента	1	2	3	4	5	6
Исходный массив	12	-3	7	2	-16	8
1-й проход	-3	7	2	-16	8	12
2-й проход	-16	-3	7	2	8	
3-й проход		-3	2	7		

Шейкерная сортировка массива осуществлена не в 5, а в 3 прохода.
С каждым новым проходом количество обрабатываемых элементов уменьшается.




```
Sub Шейкерная_сортировка
…
k = 0                ‘Количество
проходов
L = 1
R = n
Do
  k = k + 1          ‘Проход слева
на право
  m = L
  For i = L To R - 1
    If a(i) > a(i + 1) Then
      m = i
      p = a(i): a(i) = a(i + 1): a(i
+ 1) = p
    End If
  Next i
  For i = L To r     ‘Вывод
массива
    Cells(k + 1, i) = a(i)
```




```
  k = k + 1          ' Проход
справа на лево
  m = R
  For i = R To L + 1 Step -1
    If a(i) < a(i - 1) Then
      m = i
      p = a(i): a(i) = a(i - 1): a(i -
1) = p
    End If
  Next i
  For i = L To r       'Вывод
массива
    Cells(k + 1, i) = a(i)
  Next i
  L = m
Loop Until L = r

End sub
```

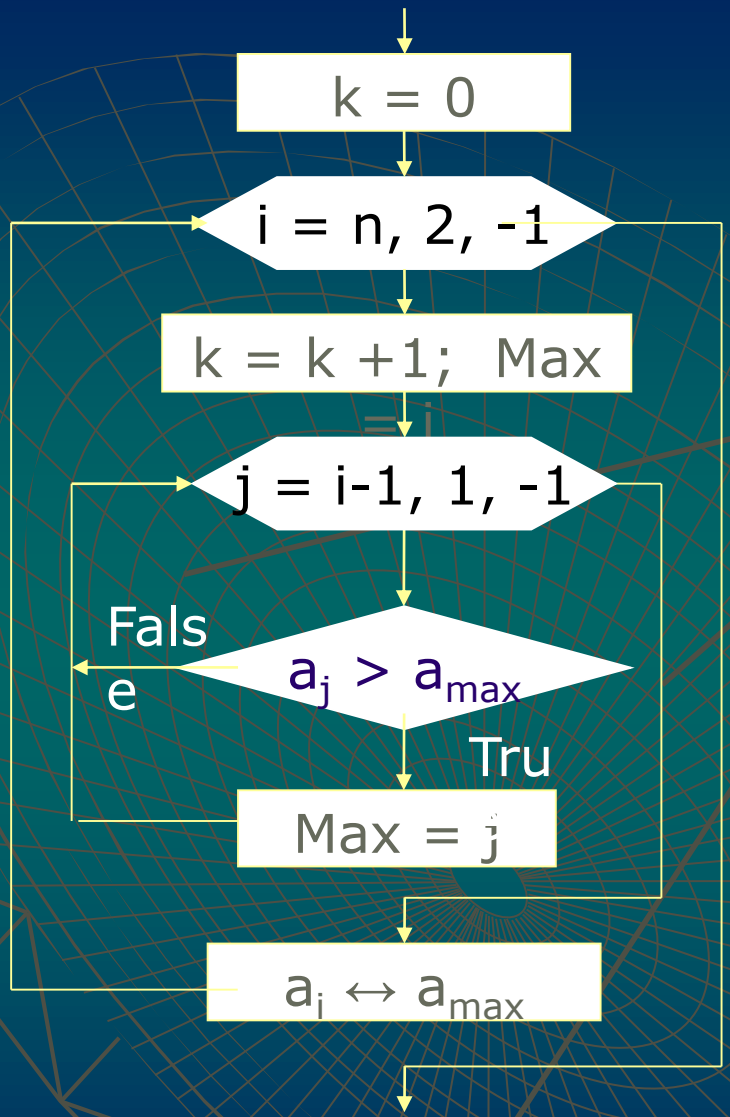
5.10. Сортировка выбором

Для сортировки элементов массива можно воспользоваться алгоритмом сортировки **выбора максимального** (**минимального**) **элемента**.

Найдём в массиве **самый большой элемент** и поменяем его местами с **последним элементом**.

Затем надо повторять эти действия, уменьшая количество просматриваемых элементов на единицу до тех пор, пока количество рассматриваемых элементов в массиве не станет равным одному.

Номер элемента	1	2	3	4	5	6
Исходный массив	12	-3	7	2	-16	8
1-й проход	8	-3	7	2	-16	12
2-й проход	-16	-3	7	2	8	
3-й проход	-16	-3	2	7		
4-й проход	-16	-3	2			
5-й проход	-16	-3				



Алгоритм Сортировки выбором

```
Sub сортировка_выбором()
…
k = 0            'Номер прохода
For i = n To 2 Step -1
  k = k + 1
  Max = i        'Номер макс. элемента
  For j = i - 1 To 1 Step -1
    If a(j) > a(Max) Then Max = j
  Next j
    p = a(i): a(i) = a(Max): a(Max) = p
  For j = 1 To i       'Вывод массива
    Cells(k + 1, j) = a(j)
  Next j
Next i
```

5.11. Сортировка вставками

Алгоритм метода:

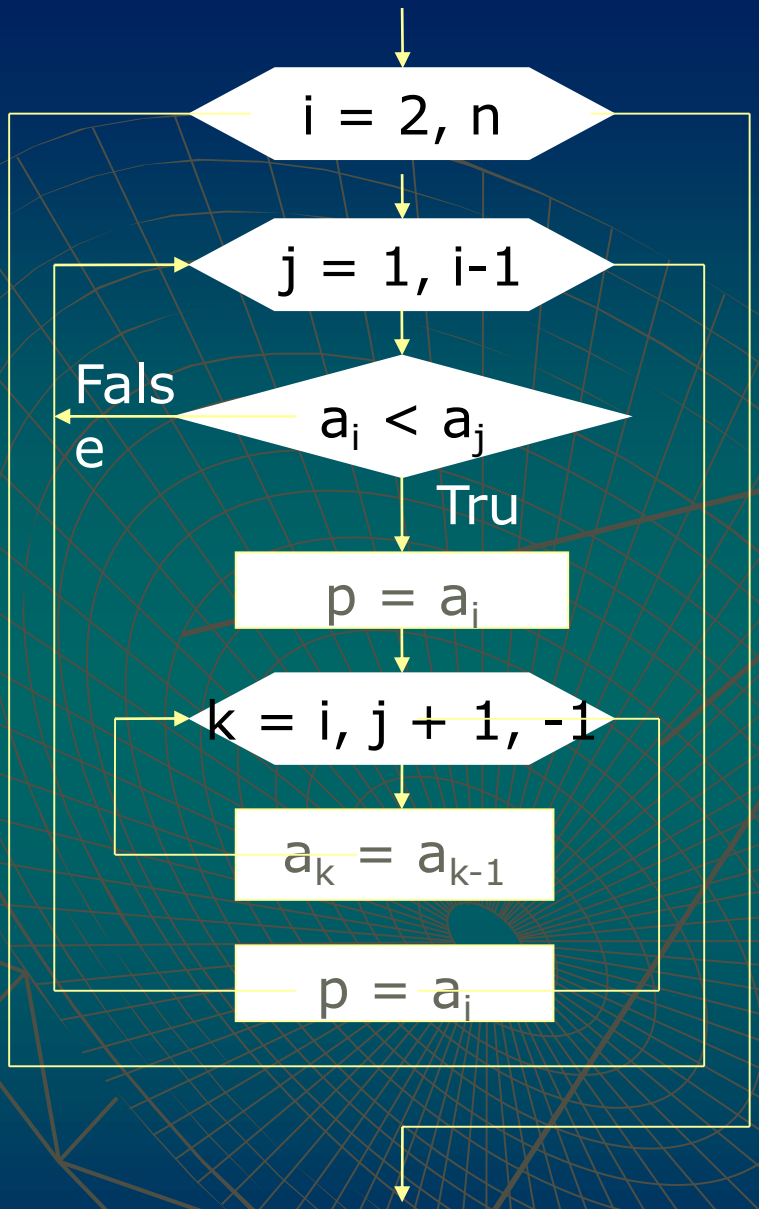
Берем первые два элемента и располагаем их в порядке возрастания.

Затем берем следующий элемент и вставляем его в нужное место среди тех, что мы уже обработали.

На каждом шаге новый элемент вставляется посредством передвижения больших элемента на одну позицию вправо.

Процесс завершится когда последний n-й элемент будет помещен в подходящее для него место.

Номер элемента	1	2	3	4	5	6
Исходный массив	12	-3	7	2	-16	8
1-й проход	-3	12				
2-й проход	-3	7	12			
3-й проход	-3	2	7	12		
4-й проход	-16	-3	2	7	12	
5-й проход	-16	-3	2	7	8	12



Алгоритм
сортировки вставками

```
Sub Сортировка_ вставками
…
For i = 2 To n
  For j = 1 To i - 1
    If a(i) < a(j) Then
      p = a(i)
      For k = i To j + 1 Step -
1
        a(k) = a(k - 1)
      Next k
      a(j) = p
    End If
  Next j

  For j = 1 To i      'Вывод
массива
    Cells(i + 1, j) = a(j)
  Next j
Next i

End Sub
```

5.12. Метод быстрой сортировки

Алгоритм быстрой сортировки разработан английским математиком **Чарльзом Хоаром** (Hoare) в 1960 году в МГУ. Быструю сортировку называют **сортировкой с разделением** или **алгоритмом сортировки Хоара**.

Метод заключается в следующем:

Выбирается произвольный элемент массива, называемый базовым или опорным. Этот элемент делит массив на две части: в левую помещаются все элементы меньшие базового элемента, в правую элементы, большие базового элемента.

Полученные два списка сортируются таким же образом, т.е. вновь выбираются базовые элементы (уже внутри подсписков), подсписки делятся на два, в левую помещаются элементы меньшие базового элемента, в правую элементы, большие базового элемента и т.д.

При реализации алгоритма процедура, выполняющая основные действия по сортировке вызывает саму себя (рекурсия).

Обычно в качестве базового элемента берут элемент **R** средний по месту нахождения элемента в списке. Вводятся два указателя i и j. В начале i =1, j = n, где n -число сортируемых записей.

На каждом шаге выполняется разбиение записей на две подгруппы:

Левая подгруппа	Правая подгруппа
$R_i < R$	$R < R_j$

Имеются два внутренних цикла.

Первый цикл делает просмотр элементов правой подгруппы справа. С базовым элементом R сравниваются R_n, R_{n-1}, R_{n-2}, ... до тех пор, пока не встретится элемент $R_j < R$.

Затем второй внутренний цикл выполняет просмотр элементов левой подгруппы слева. Здесь с базовым элементом R сравниваются R_1, R_2, R_3, ... до тех пор, пока не встретится элемент $R_i > R$.

Теперь возможны два случая:

1. $i < j$, это означает, что два элемента, на которые указывают индексы расположены в неправильном порядке. Меняем местами элементы и продолжаем выполнение внутренних циклов.

2. $i \geq j$, это означает, что текущая подгруппа успешно разделена. Выполнение внутренних циклов завершается.

Внешний цикл начинает свою работу вновь с определения базового элемента, теперь уже для текущей подгруппы.

5.13. Выполнение действий над матрицами

Матрицей называется прямоугольная таблица чисел, содержащая m строк и n столбцов.

Числа m и n называются порядками матрицы.

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{pmatrix}$$

Пример. Матрица **A** порядков 3x5.

$$\grave{A} = \begin{pmatrix} -1 & 0 & 3{,}5 & 9 & 9 \\ 0 & \frac{2}{3} & 65 & 400 & \sqrt{8} \\ \cos 3 & 17 & 0{,}8 & \pi & \log_2 5 \end{pmatrix}$$

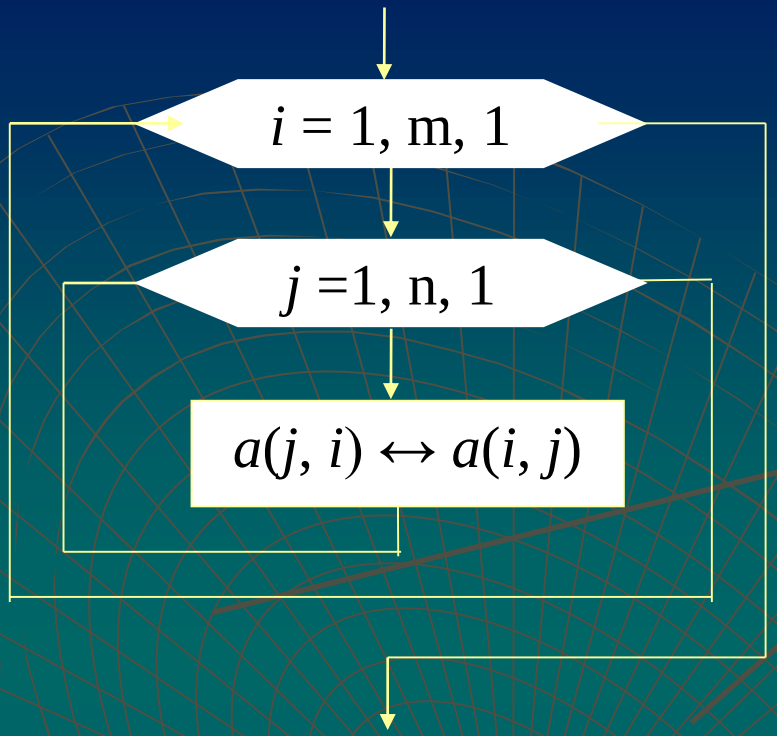
5.13.1. Транспонирование матриц

Преобразование матрицы, заключающееся в замене строк этой матрицы на ее столбцы с сохранением их порядка, называется **транспонированием матрицы**.

$$A=\begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{pmatrix} \qquad A^T=\begin{pmatrix} a_{11} & a_{21} & a_{31} & \dots & a_{n1} \\ a_{12} & a_{22} & a_{32} & \dots & a_{n2} \\ a_{13} & a_{23} & a_{33} & \dots & a_{n3} \\ \dots & \dots & \dots & \dots & \dots \\ a_{1m} & a_{2m} & a_{3m} & \dots & a_{nm} \end{pmatrix}$$

Пример. Транспонируем матрицы порядков 3x3 и 2x4.

$$\begin{pmatrix} -1 & 8 & 7 \\ 3 & 2 & 0 \\ 4 & -2 & -4 \end{pmatrix}^T = \begin{pmatrix} -1 & 3 & 4 \\ 8 & 2 & -2 \\ 7 & 0 & -4 \end{pmatrix}; \qquad \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 9 & 16 \end{pmatrix}^T = \begin{pmatrix} 1 & 1 \\ 2 & 4 \\ 3 & 9 \\ 4 & 16 \end{pmatrix}.$$



Блок-схема
транспонирования матрицы

```
Sub Транспонирование()
Dim a(100, 100) As Integer
Dim at(100, 100) As Integer
m = InputBox("Введите кол-во строк m")
n = InputBox("Введите кол-во столбцов n")

For i = 1 To m
  For j = 1 To n
    a(i, j) = Rnd(Timer) * 200 - 100
    Cells(i + 1, j) = a(i, j)
Next j, i

For i = 1 To m
  For j = 1 To n
    at(j, i) = a(i, j)
Next j, i

For i = 1 To n
  For j = 1 To m
    Cells(i + 1, j + n + 1) = at(i, j)
Next j, i
End Sub
```

	1	2	3	4	5	6	7	8	9
1									
2	63	42	-91	-17	73		63	58	90
3	58	-25	92	74	-89		42	-25	-27
4	90	-27	5	53	-89		-91	92	5
5							-17	74	53
6							73	-89	-89
7									

5.13.2. Матричное сложение

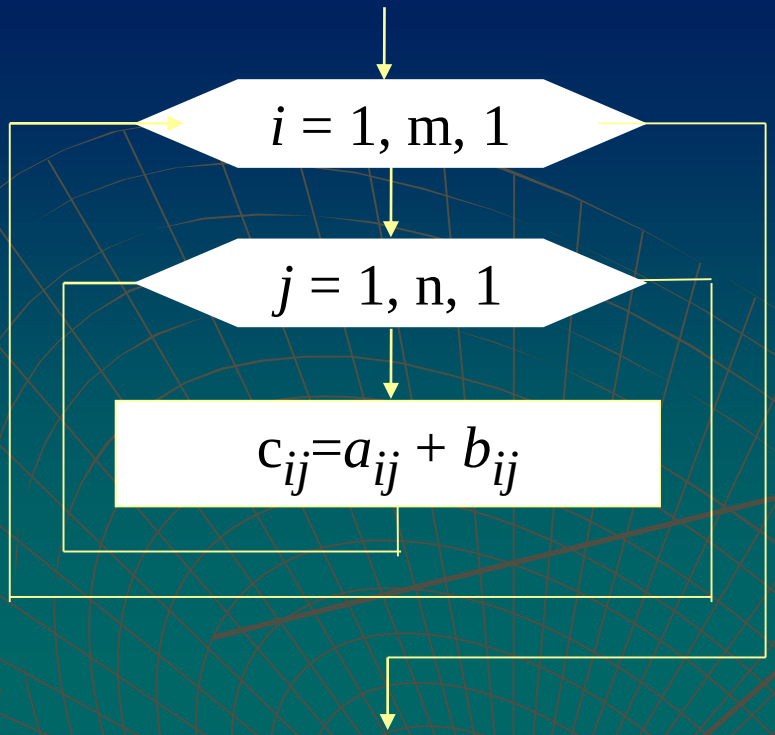
Суммой двух матриц **A** и **B** одних и тех же порядков ***m*** и ***n*** называется матрица **C** тех же порядков ***m*** и ***n***, элементы которой равны

$$c_{ij} = a_{ij} + b_{ij}$$

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{pmatrix} = \begin{pmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \dots & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{22} & \dots & a_{2n}+b_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1}+b_{n1} & a_{n2}+b_{n2} & \dots & a_{nn}+b_{nn} \end{pmatrix}$$

Пример. Сложим две матрицы порядков 3x2.

$$A + B = \begin{pmatrix} 11 & 3 & -2 \\ 4 & -12 & 5 \end{pmatrix} + \begin{pmatrix} -6 & 20 & 8 \\ 5 & 2 & 14 \end{pmatrix} = \begin{pmatrix} 11+(-6) & 3+20 & -2+8 \\ 4+5 & -12+2 & 5+14 \end{pmatrix} = \begin{pmatrix} 5 & 23 & 6 \\ 9 & -10 & 19 \end{pmatrix}$$



Блок-схема
сложения матриц
C = A + B

```
Sub сложение_матриц()
Dim a(100, 100) As Integer
Dim b(100, 100) As Integer
Dim c(100, 100) As Integer
m = InputBox("Введите кол-во строк n")
n = InputBox("Введите кол-во столбцов m")

For i = 1 To m
  For j = 1 To n
    a(i, j) = Rnd(Timer) * 200 - 100
    Cells(i + 1, j) = a(i, j)
    b(i, j) = Rnd(Timer) * 200 - 100
    Cells(i + 1, j + n + 1) = b(i, j)
Next j, i

For i = 1 To m
  For j = 1 To n
    c(i, j) = a(i, j) + b(i, j)
    Cells(i + 1, j + 2 * n + 2) = c(i, j)
Next j, i
End Sub
```

	1	2	3	4	5	6	7	8	9	10	11
1											
2	41	16	-40		7	-42	55		48	-26	15
3	-97	63	-91		52	42	-17		-45	105	-108
4	73	-25	74		58	92	-89		131	67	-15
5	90	5	-89		-27	53	18		63	58	-71
6	-6	25	-47		-40	30	-44		-46	55	-91
7											

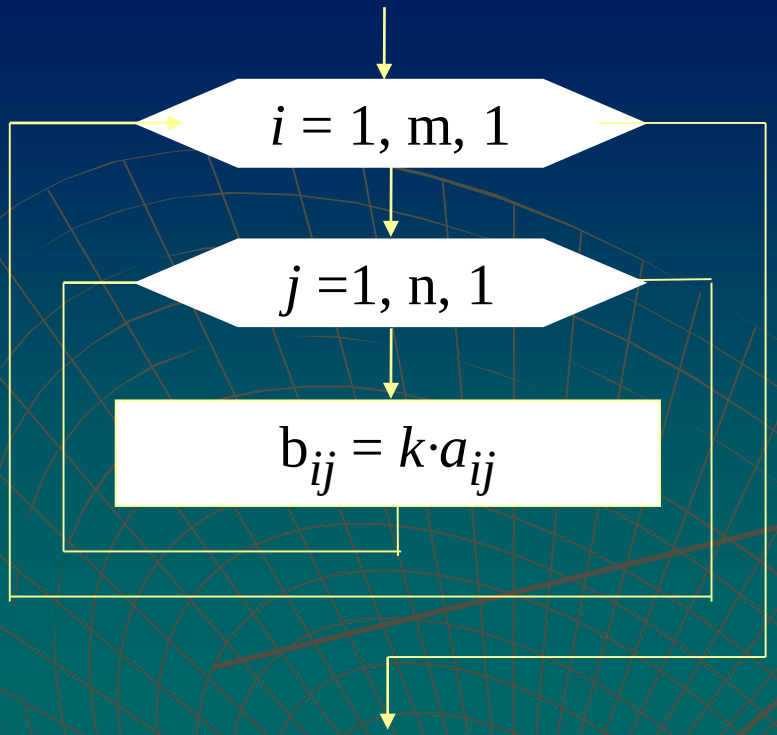
5.13.3. Умножение матрицы на число

Произведением матрицы A на вещественное число k называется матрица B, элементы которой равны: $b_{ij} = k \cdot a_{ij}$

$$k\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} ka_{11} & ka_{12} & \dots & ka_{1n} \\ ka_{21} & ka_{22} & \dots & ka_{2n} \\ \dots & \dots & \dots & \dots \\ ka_{n1} & ka_{n2} & \dots & ka_{nn} \end{pmatrix}$$

Пример. Умножим матрицу порядков 4x2 на 3.

$$3\cdot\begin{pmatrix} 2 & 1 \\ 4 & -2 \\ 8 & 3 \\ -1 & 3 \end{pmatrix} = \begin{pmatrix} 3\cdot 2 & 3\cdot 1 \\ 3\cdot 4 & 3\cdot(-2) \\ 3\cdot 8 & 3\cdot 3 \\ 3\cdot(-1) & 3\cdot 3 \end{pmatrix} = \begin{pmatrix} 6 & 3 \\ 12 & -6 \\ 24 & 9 \\ -3 & 9 \end{pmatrix}.$$



Блок-схема
Умножения матрицы на число
B = k·A

```
Sub умножение_матрицы_на_число()
Dim a(100, 100) As Integer
Dim b(100, 100) As Integer
m = InputBox(" Введите кол-во строк n ")
n = InputBox(" Введите кол-во столбцов m")

For i = 1 To m
  For j = 1 To n
    a(i, j) = Rnd(Timer) * 200 - 100
    Cells(i + 1, j) = a(i, j)
Next j, i

k = InputBox(" Введите множитель k")

For i = 1 To m
  For j = 1 To n
    b(i, j) = k * a(i, j)
    Cells(i + 1, j + n + 1) = b(i, j)
Next j, i
End Sub
```

	1	2	3	4	5	6	7	8	9	10	11	12	13
1													
2	41	7	16	-42	-40	55		-82	-14	-32	84	80	-110
3	-97	52	63	42	-91	-17		194	-104	-126	-84	182	34
4	73	58	-25	92	74	-89		-146	-116	50	-184	-148	178
5	90	-27	5	53	-89	18		-180	54	-10	-106	178	-36
6													

5.13.4. Произведение матриц

Произведением матрицы **A**, имеющей порядки, соответственно равные ***m*** и ***n***, на матрицу **B**, имеющую порядки, соответственно равные ***n*** и ***p***, называется матрица ***C***, имеющая порядки, соответственно равные ***m*** и ***p***, и элементы c_{ij} определяемые формулой:

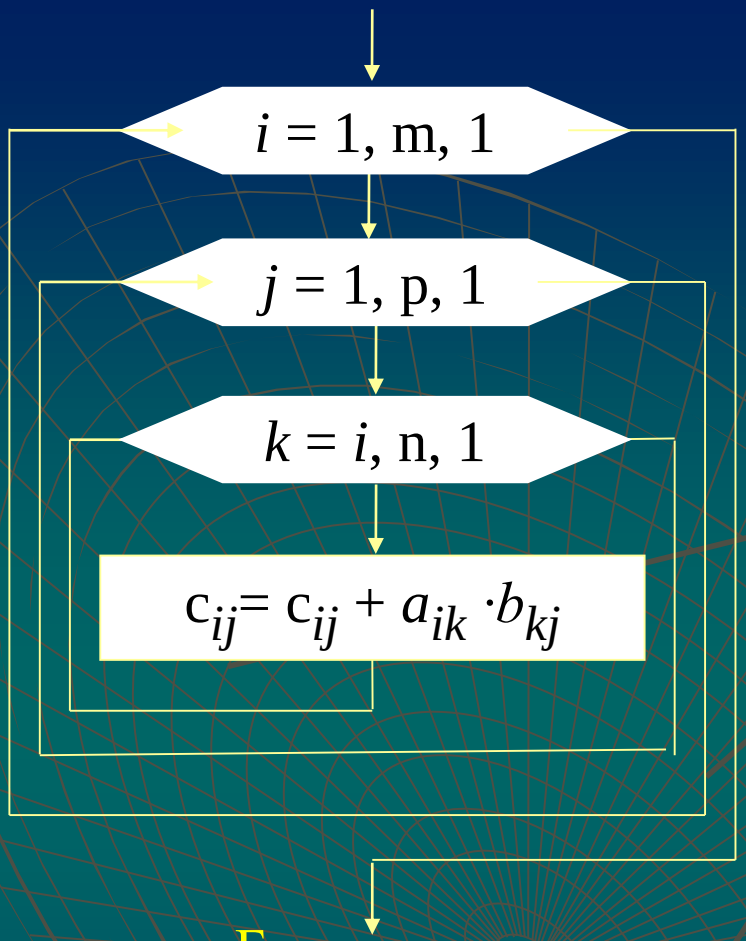
$$c_{ij} = \sum_k a_{ik} \cdot b_{kj}$$

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

Пример. Перемножим матрицы порядков 2x3 и 3 x3.

$$A = \begin{pmatrix} 2 & 0 & 1 \\ 0 & -1 & 2 \end{pmatrix}; \qquad B = \begin{pmatrix} 4 & 1 & 2 \\ 0 & 3 & 1 \\ 0 & 1 & 0 \end{pmatrix}.$$

$$AB = \begin{pmatrix} 2\cdot4+0\cdot0+1\cdot0 & 2\cdot1+0\cdot3+1\cdot1 & 2\cdot2+0\cdot1+1\cdot0 \\ 0\cdot4+(-1)\cdot0+2\cdot0 & 0\cdot1+(-1)\cdot3+2\cdot1 & 0\cdot2+(-1)\cdot1+2\cdot0 \end{pmatrix} = \begin{pmatrix} 8 & 3 & 4 \\ 0 & -1 & -1 \end{pmatrix}$$



Блок-схема
Перемножения матриц
$C = A \cdot B$

```
Sub произведение_матриц()
Dim a(100, 100) As Integer
Dim b(100, 100) As Integer
Dim c(100, 100) As Integer

m = InputBox("Введите кол-во строк 1 матрицы")
n = InputBox("Введите кол-во столбцов 1 матрицы")
For i = 1 To m  'Генерация и вывод 1 матрицы
  For j = 1 To n
    a(i, j) = Rnd(Timer) * 200 - 100
    Cells(i + 1, j) = a(i, j)
Next j, i

p = InputBox("Кол-во столбцов 2 матрицы")
For i = 1 To n  'Генерация и вывод 2 матрицы
  For j = 1 To p
    b(i, j) = Rnd(Timer) * 200 - 100
    Cells(i + 1, j + n + 1) = b(i, j)
Next j, i

For i = 1 To m  'Расчет и вывод произведения
  For j = 1 To p
    For k = 1 To n
      c(i, j) = c(i, j) + a(i, k) * b(k, j)
    Next k
    Cells(i + 1, j + n + p + 2) = c(i, j)
Next j, i
End Sub
```

	1	2	3	4	5	6	7	8	9	10	11	12	13
1													
2	-24	-40	90		-63	17	-84	-8		-5508	9072	2076	3492
3	96	-20	-44		81	-48	57	-24		-5820	-1104	-10348	-1432
4	-68	-67	29		-42	84	26	26		-2361	4496	2647	2906
5	-18	-17	43							-2049	4122	1661	1670
6	-35	27	-58							6828	-6763	2971	-1876
7													

5.14. Вычисление определителей

С каждой такой матрицей свяжем вполне определенную численную характеристику, называемую **определителем**, соответствующим этой матрице. В отличие от матрицы для обозначения определителя употребляют не круглые, а прямые скобки, а перед буквенным выражением указывается слово *det*. Определитель матрицы *A* обозначается:

$$\det A = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}$$

Пример. Определители порядков 1, 2, 3 и 4.

$$\det(-3{,}5) = -3{,}5;$$

$$\det\begin{pmatrix} -5 & 0 \\ 3 & 4 \end{pmatrix} = -20;$$

$$\begin{vmatrix} 2 & 4 & -5 \\ 3 & 0 & 1 \\ -1 & -2 & 0 \end{vmatrix} = -36;$$

$$\begin{vmatrix} 1 & 2 & 3 & 4 \\ 3 & 0 & 0 & 1 \\ 8 & 1 & 2 & 6 \\ 3 & 9 & 8 & 0 \end{vmatrix} = 3.$$

Теорема. Определитель матрицы A численно равен:

$$\det A = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} = \sum_{n} (-1)^{Z(\pi)} a_{1i_1} a_{2i_2} a_{3i_3} \cdot \dots \cdot a_{ni_n},$$

причем суммы должны быть распространены на все подстановки π набора чисел 1, 2, 3, ..., n.

Таким образом, из элементов матрицы **A** сначала составляются всевозможные произведения $a_{1i_1} a_{2i_2} a_{3i_3} \cdot \dots \cdot a_{ni_n}$ из n сомножителей каждое, содержащие по одному элементу их каждой строки и одному элементу из каждого столбца, а затем эти произведения складываются, причем знак $(-1)^{Z(\pi)}$ каждого слагаемого определяется числом инверсий соответствующей подстановки:

$$\pi = \begin{pmatrix} 1 & 2 & 3 & \dots & n \\ i_1 & i_2 & i_3 & \dots & i_n \end{pmatrix}.$$

Пример. Вычислить определитель $\det(A) = \begin{vmatrix} 2 & 4 & 1 \\ 3 & 5 & 1 \\ -1 & -2 & 6 \end{vmatrix}$.

$$\det A = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} = \sum_{n} (-1)^{Z(\pi)} a_{1i_1} a_{2i_2} a_{3i_3} \cdot \dots \cdot a_{ni_n},$$

$\det(A) = (-1)^{Z(1,2,3)} \cdot 2 \cdot 5 \cdot 6 + (-1)^{Z(1,3,2)} \cdot 2 \cdot 1 \cdot (-2) +$

$+ (-1)^{Z(2,1,3)} \cdot 4 \cdot 3 \cdot 6 + (-1)^{Z(2,3,1)} \cdot 4 \cdot 1 \cdot (-1) +$

$+ (-1)^{Z(3,1,2)} \cdot 1 \cdot 3 \cdot (-2) + (-1)^{Z(3,2,1)} \cdot 1 \cdot 5 \cdot (-1) =$

$= (-1)^0 \cdot 60 + (-1)^1 \cdot (-4) + (-1)^1 \cdot 72 + (-1)^2 \cdot (-4) +$

$+ (-1)^2 \cdot (-6) + (-1)^3 (-5) = 60 + 4 - 72 - 4 - 6 + 5 =$ **-13**

```
Sub Определитель_4_порядка()
Dim A(4, 4), m(4)
                        'Ввод матрицы
For i = 1 To 4
  For j = 1 To 4
    A(i, j) = Cells(i, j)
Next j, i
                        'Вычисление определителя
For i = 1 To 4
  For j = 1 To 4
    For k = 1 To 4
      For l = 1 To 4
        inv = 0
        m(1) = i: m(2) = j: m(3) = k: m(4) = l
        For x = 1 To 3
          For y = x + 1 To 4
            If m(x) > m(y) Then inv = inv + 1
        Next y, x
        If i <> j And i <> k And i <> l And j <> k And j <> l And k <> l Then
          D = D + ((-1) ^ inv) * A(1, i) * A(2, j) * A(3, k) * A(4, l)
        End If
Next l, k, j, i

Cells(1, 6) = D      'Вывод значения определителя
End Sub
```

	1	2	3	4	5	6	7
1	6	5	4	3		56	
2	2	3	4	5			
3	3	2	4	1			
4	4	2	5	2			
5							

5.15. Решение систем линейных уравнений

Метод Гаусса – метод последовательного исключения переменных – заключается в том, что с помощью элементарных преобразований система уравнений приводится к равносильной системе ступенчатого (или треугольного) вида, из которой последовательно, начиная с последних (по номеру) переменных, находятся все остальные переменные.

К элементарным преобразованиям относятся:

1) перестановка уравнений в системе;

2) добавление или вычеркивание тривиального уравнения;

3) умножение уравнения на отличное от нуля число;

4) прибавление к обеим частям уравнения любых чисел и соответственных частей другого уравнения системы.

$$\left(\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} & b_2 \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} & b_3 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} & b_n \end{array}\right) \Leftrightarrow \dots$$

$$\dots \Leftrightarrow \left(\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \dots & a_{1n} & b_1' \\ 0 & a_{22}' & a_{22}' & \dots & a_{2n}' & b_2' \\ 0 & 0 & a_{33}' & \dots & a_{3n}' & b_3' \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & a_{nn}' & b_n' \end{array}\right) \Leftrightarrow \dots$$

$$\dots \Leftrightarrow \left(\begin{array}{ccccc|c} a_{11}'' & 0 & 0 & \dots & 0 & b_1'' \\ 0 & a_{22}'' & 0 & \dots & 0 & b_2'' \\ 0 & 0 & a_{33}'' & \dots & 0 & b_3'' \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & a_{nn}'' & b_n'' \end{array}\right)$$

	A	B	C	D	E	F
1	0	-2	5	3	-2	-8
2	0	-5	-4	3	-4	-66
3	2	-5	2	0	6	-20
4	2	7	2	0	6	100
5	1	-5	-4	3	-4	-68
6						

```
Sub Метод_ГАУССА()

Dim a(100, 100) As Double
Dim M As Byte, N As Byte

M = 5: N = 5                    'Количество уравнений и переменных

For i = 1 To M                  'Считывание элементов
  For j = 1 To N + 1
    a(i, j) = Cells(i, j)
  Next j
Next i

s = M + 3                       'Установкв позиции печати
```

```
Sub swap(x, y)
  p = x
  x = y
  y = p
End Sub
```

```
                                    'Прямой ход метода Гаусса
Cells(s - 1, N) = "Прямой ход метода Гаусса"
For i = 1 To M - 1
  If a(i, i) = 0 Then               'Проверка диагонального элемента
    nso = i
    Do
      nso = nso + 1
    Loop Until a(nso, i) <> 0 Or nso > M
    If ns > N Then MsgBox ("Система не имеет решений!"): End
    For j = 1 To N + 1
      Call swap(a(i, j), a(nso, j))
    Next j
  End If
                                    'Шаг прямого хода
  For j = i + 1 To M
    b = a(j, i) / a(i, i)
    For k = 1 To N + 1
        a(j, k) = a(j, k) - b * a(i, k)
    Next k
  Next j
                                  'Печать матрицы после i-го шага
  For j = 1 To M
    For k = 1 To N + 1
        Cells(s + j, k) = a(j, k)
      Next k
  Next j
  s = s + M + 1
Next i
```

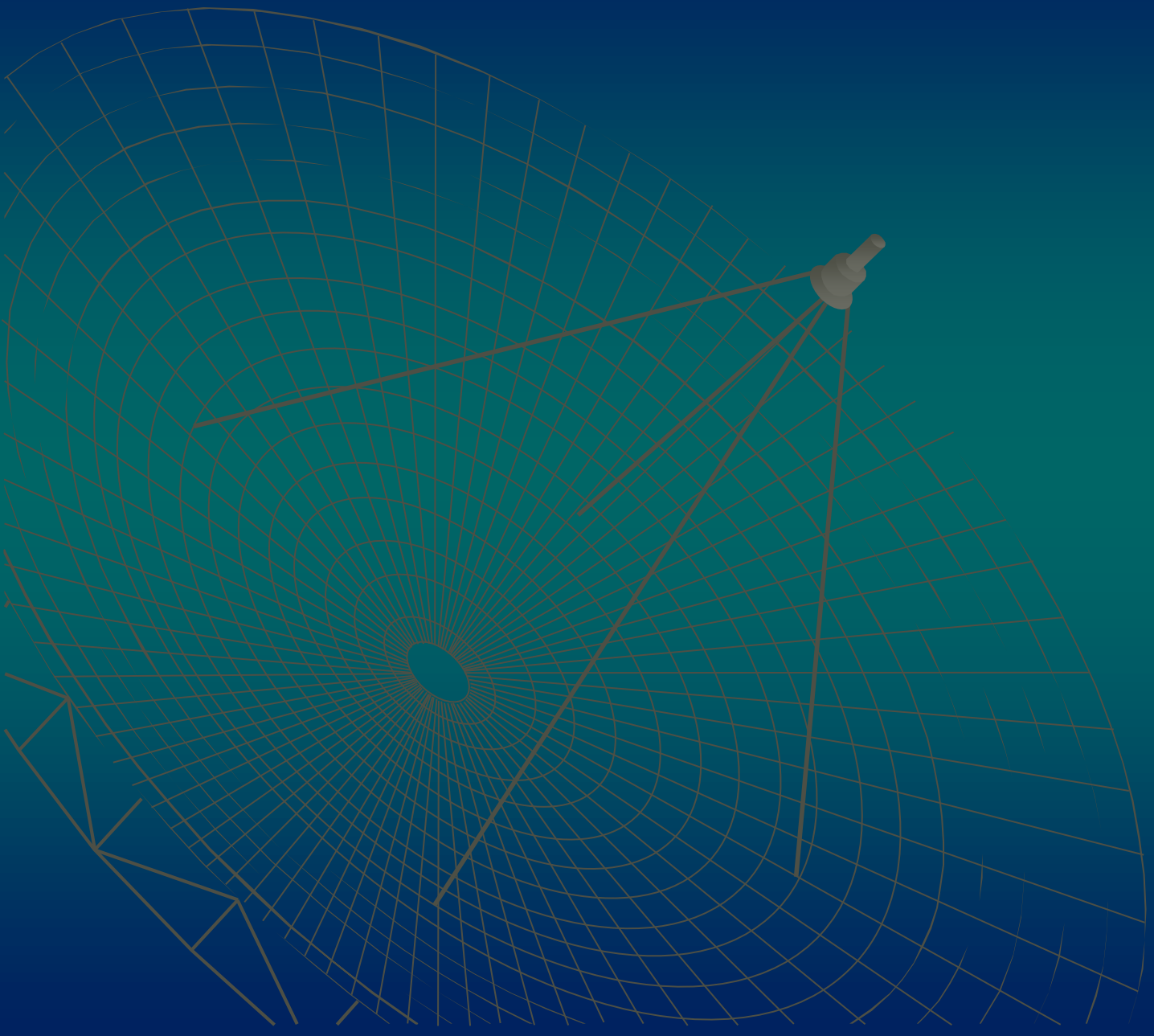
	A	B	C	D	E	F
1	0	-2	5	3	-2	-8
2	0	-5	-4	3	-4	-66
3	2	-5	2	0	6	-20
4	2	7	2	0	6	100
5	1	-5	-4	3	-4	-68
6						
7			Прямой ход мход метода Гаусса			
8						
9	2	-5	2	0	6	-20
10	0	-5	-4	3	-4	-66
11	0	-2	5	3	-2	-8
12	0	12	0	0	0	120
13	0	-2,5	-5	3	-7	-58
14						
15	2	-5	2	0	6	-20
16	0	-5	-4	3	-4	-66
17	0	0	6,6	1,8	-0,4	18,4
18	0	0	-9,6	7,2	-9,6	-38,4
19	0	0	-3	1,5	-5	-25
20						
21	2	-5	2	0	6	-20
22	0	-5	-4	3	-4	-66
23	0	0	6,6	1,8	-0,4	18,4
24	0	0	0	9,818182	-10,1818	-11,6364
25	0	0	0	2,318182	-5,18182	-16,6364
26						
27	2	-5	2	0	6	-20
28	0	-5	-4	3	-4	-66
29	0	0	6,6	1,8	-0,4	18,4
30	0	0	0	9,818182	-10,1818	-11,6364
31	0	0	0	0	-2,77778	-13,8889

```
s = s + 2
                                    'Обратный ход метода Гаусса
Cells(s - 1, N) = "Обратный ход метода Гаусса"
For i = M To 2 Step -1
    For j = i - 1 To 1 Step -1
        b = a(j, i) / a(i, i)
        For k = 1 To N + 1
            a(j, k) = a(j, k) - b * a(i, k)
        Next k
    Next j
                'Ппечать матрицы после i-го шага
    For j = 1 To M
        For k = 1 To N + 1
            Cells(s + j, k) = a(j, k)
        Next k
    Next j
    s = s + M + 1
Next i
```

32						
33				Обратный ход метода Гаусса		
34						
35	2	-5	2	0	-8,9E-16	-50
36	0	-5	-4	3	0	-46
37	0	0	6,6	1,8	0	20,4
38	0	0	0	9,818182	0	39,27273
39	0	0	0	0	-2,77778	-13,8889
40						
41	2	-5	2	0	-8,9E-16	-50
42	0	-5	-4	4,44E-16	0	-58
43	0	0	6,6	0	0	13,2
44	0	0	0	9,818182	0	39,27273
45	0	0	0	0	-2,77778	-13,8889
46						
47	2	-5	0	0	-8,9E-16	-54
48	0	-5	0	4,44E-16	0	-50
49	0	0	6,6	0	0	13,2
50	0	0	0	9,818182	0	39,27273
51	0	0	0	0	-2,77778	-13,8889
52						
53	2	0	0	-4,4E-16	-8,9E-16	-4
54	0	-5	0	4,44E-16	0	-50
55	0	0	6,6	0	0	13,2
56	0	0	0	9,818182	0	39,27273
57	0	0	0	0	-2,77778	-13,8889
58						

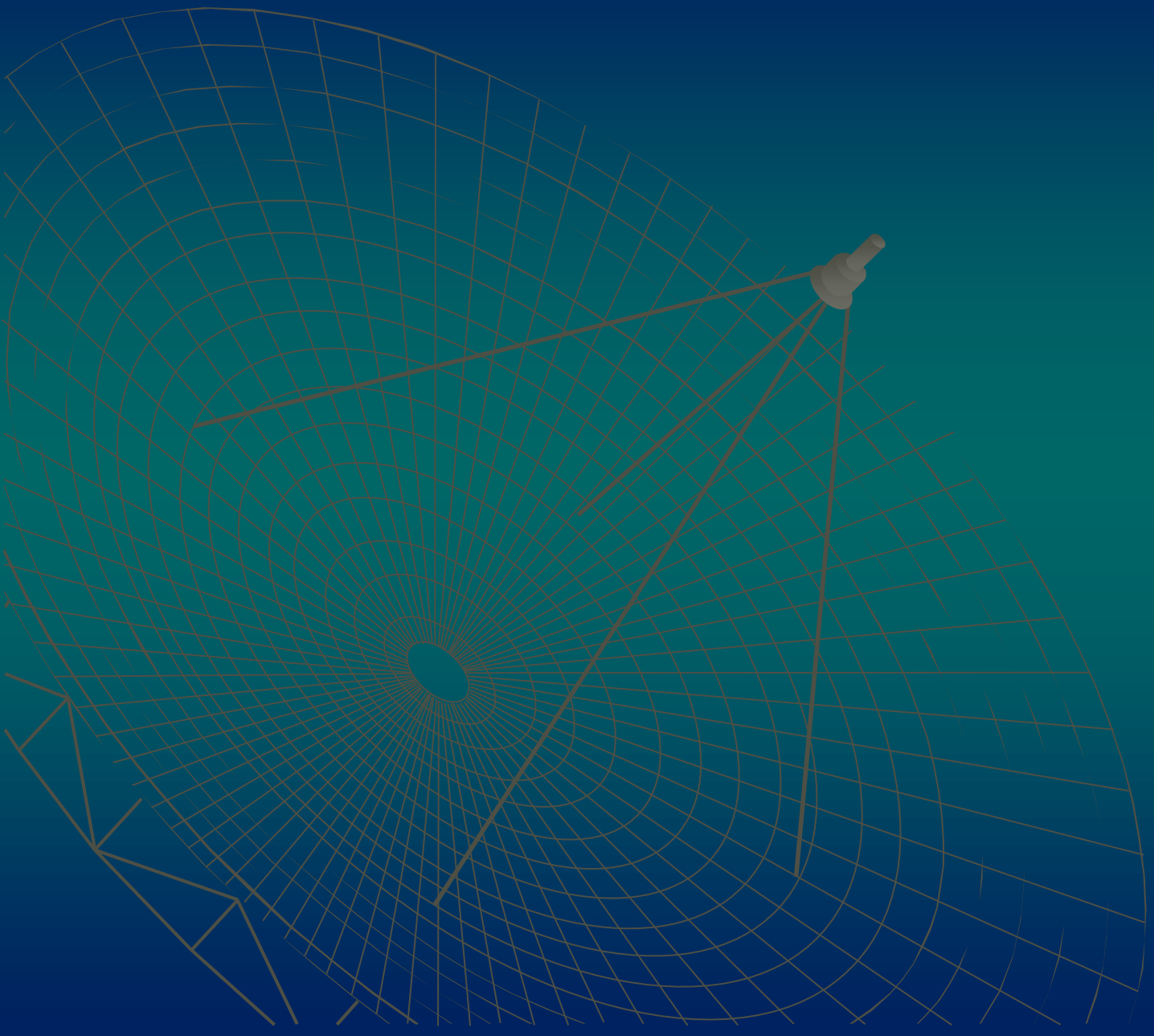
```
                    'деление на a(i,i)
s = s + 2
Cells(s - 1, N) = "РЕШЕНИЕ"
For i = 1 To M
    b = a(i, i)
    For j = 1 To N + 1
        a(i, j) = a(i, j) / b
        Cells(s + i, j) = a(i, j)
    Next j
Next i

End Sub
```

58						
59				РЕШЕНИЕ		
60						
61	1	0	0	-2,2E-16	-4,4E-16	-2
62	0	1	0	-8,9E-17	0	10
63	0	0	1	0	0	2
64	0	0	0	1	0	4
65	0	0	0	0	1	5
66						



Глава 6. Подпрограммы, процедуры и функции

6.1. Общие сведения о подпрограммах
6.2. Процедуры
6.3. Функции



6.1. Общие сведения о подпрограммах

6.1.1. Понятие подпрограммы

В практике программирования часто складываются ситуации, когда одну и ту же группу операторов, реализующих определённую цель, требуется повторить без изменений в нескольких местах программы. Для избавления от столь нерациональной траты времени была предложена концепция подпрограммы.

Подпрограмма – это именованная, логически законченная группа операторов языка, которую можно вызвать для выполнения любое количество раз из различных мест программы.

В языке Visual Basic for Application существуют два основных вида подпрограмм: **процедуры** и **функции**.

Процедура - это универсальная подпрограмма для выполнения каких-либо действий.

Функция - это специальная подпрограмма, которая **возвращает результат**.

Макрос в VBA — это просто процедура типа Sub, не имеющая параметров. Только макросы можно вызывать по имени из редактора VBA или приложения Office.

Все другие процедуры нужно вызывать либо из других процедур, либо специальными способами

Вызов функции является выражением, и может использоваться в других выражениях или в правой части оператора присваивания

Информация, передаваемая в подпрограмму для обработки, называется **параметрами**, а результат вычислений – **значениями функции**.

Обращение к подпрограмме называют **вызовом**.

6.1.2. Переменные и константы

Константы, переменные, типы данных могут быть объявлены как в основной программе, так и в подпрограммах различной степени вложенности.

Переменные и константы, объявленные явно или неявно в основной программе до определения подпрограмм, называются **глобальными**, они доступны вызываемым функциям и процедурам.

Переменные и константы, описанные в какой-либо подпрограмме, доступны только в ней и называются **локальными**.

Обмен информацией между вызываемой и вызывающей функциями осуществляется с помощью механизма передачи параметров.

Переменные, указанные в списке в заголовке функции, называются **формальными параметрами**, или просто параметрами подпрограммы. Все переменные из этого списка могут использоваться внутри подпрограммы. Список переменных в операторе вызова подпрограммы - это **фактические параметры**, или аргументы.

Формальные параметры процедуры можно разделить на два класса: параметры-значения и параметры-переменные

При передаче данных через **параметры-значения** в подпрограмму передаются значения фактических параметров, и доступа к самим фактическим параметрам из подпрограммы нет.

При передаче данных **параметры-переменные** заменяют формальные параметры, и, следовательно, в подпрограмме есть доступ к значениям фактических параметров.

Любое изменение параметров-переменных в подпрограмме приводит к изменению соответствующих им формальных параметров.

Следовательно, входные данные следует передавать через параметры-значения, для передачи изменяемых в результате работы подпрограммы данных следует использовать параметры-переменные.

6.2. Процедуры

6.2.1. Объявление процедуры

Описание процедуры имеет вид:

```
Sub имя_процедуры (список_формальных_параметров)
... ... ... ... ... ... ... ... ... ... ... ... ... ...
Тело процедуры
... ... ... ... ... ... ... ... ... ... ... ... ... ...
End Sub
```

Описание начинается с заголовка процедуры, где

Sub - ключевое слово,

имя_процедуры любой допустимый идентификатор,

список_формальных_параметров - имена формальных параметров и их типы, разделённые запятыми.

Объявить процедуру можно вручную, впечатав в коде строку, тогда редактор кода автоматически добавит строку *End Sub* и разделитель, либо на графическом экране, воспользовавшись меню **Insert -> Procedure**.

6.2.2. Область видимости процедур

По умолчанию все процедуры VBA определяются как *открытые* (Public). Это значит, что их можно вызвать из любой части программы — из того же модуля, из другого модуля, из другого проекта.

Объявить открытую процедуру можно с помощью слова *Public*.

```
Public Sub процедура_1(k As Integer, b As Boolean)
```

или не использовать никаких ключевых слов

```
Sub процедура_1(k As Integer, b As Boolean)
```

С помощью слова ***Private*** можно объявить процедуру локальной:

```
Private Sub процедура_2()
```

В этом случае эту процедуру можно будет вызвать только из того же модуля, в котором она расположена. Такое решение предотвращает ошибки, связанные с вызовом процедур, не предназначенных для этого, из других модулей.

Для ограничения области видимости открытых процедур, определенных как *Public* достаточно в разделе объявлений этого модуля вписать строку

```
Option Private Module
```

Если при объявлении процедуры использовать ключевое слово *Static*, то все переменные в этой процедуре автоматически станут статическими и будут сохранять свои значения и после завершения работы процедуры

```
Private Static Sub процедура_3()
```

6.2.3. Передача параметров

Чтобы процедура имела возможность принимать параметры, ее вначале нужно объявить с параметрами.

Пример: Напишем процедуру, позволяющую складывать два числа

```
Sub Sum(a, b, c)
    c = a + b
End Sub
```

Вызов такой процедуры может быть осуществлен с любой части программы с помощью оператора *Call*.

```
Sub решение()
Dim x As Integer, y As Integer
x = InputBox("Введите первое число")
y = InputBox("Введите второе число")
Call Sum(x, y, z)
MsgBox ("Суммв равна " & z)
End Sub
```

В данном случае все три параметра объявлены как обязательные, и поэтому попытка вызвать функцию без передачи ей какого-либо параметра, например, Sum(4, 5), приведет к ошибке «**Argument not optional**» (Параметр не является необязательным).

Чтобы можно было пропускать какие-либо параметры, эти параметры можно сделать необязательными. Для этой цели используется ключевое слово *Optional*.

```
Sub процедура_3(m As Integer, Optional n As Byte)
```

Для проверки того, был ли передан необязательный параметр, используется либо функция *IsMissing* (если для этого параметра был использован тип *Variant*), либо его значение сравнивается со значениями переменных по умолчанию (ноль для числовых данных, пустая строка для строковых и т.п.)

Передача параметров может происходить ***по позиции***, то есть при вызове первое значение присваивается первому параметру, а второе - второму.

В нашем примере для процедуры *Sum*

```
Sub Sum(a, b, c)
```

передача параметров осуществлялась по позиции

```
Call Sum(x, y, z)
```

Однако параметры можно передавать и ***по имени***

```
Call Sum(a:=x, c:=z, b:=y)
```

Несмотря на то, что здесь выполняется операция присвоения значений, оператор использует ***двоеточие со знаком равенства***.

При использовании знака равенства возникнет ошибка

Параметры могут передаваться *по ссылке* или *по значению*. В зависимости от этого процедуры могут изменять значения переменных или не изменять их.

При передаче параметров ***по ссылке*** в вызываемую процедуру передается ссылка на эту переменную в оперативной памяти. Если эту переменную в вызываемой процедуре изменить, то значение ***изменится*** и в вызывающей функции.

Для передачи ***по ссылке*** используется ключевое слово *ByRef*. Передача ***по ссылке*** принята в VBA по умолчанию.

Если параметры передаются ***по значению***, то в оперативной памяти создается копия этой переменной и вызываемой процедуре передается эта копия. Изменение переменной в вызываемой процедуре ***не влияет*** на исходную переменную.

Для передачи ***по значению*** используется ключевое слово *ByVal*.

6.2.4. Завершение выполнения процедуры

Для завершения выполнения процедуры в VBA предусмотрены конструкции *End* и *Exit*.

Синтаксис операторов:

Exit Sub
End Sub

Оператор *End* используется для завершения работы процедуры после выполнения всех операторов процедуры. После оператора *End* код процедуры заканчивается;

Оператор *Exit* используется для немедленного завершения работы процедуры в ходе ее работы. Он обычно помещается в блок операторов условного перехода *if* или *select case*, чтобы произвести выход, как только выяснилось, что функции по каким-то причинам дальше выполняться не будет (например, дальше идет код обработчика ошибок).

Пример. Отсортировать список фамилий

Для решения задачи будем использовать метод «Британского музея».

Реализовывать программу будем поэтапно.

1) Введем элементы массива

```
Sub Метод_Британского_музея()
  Dim a(64000)
  Do Until Cells(n + 1, 1) = ""
    n = n + 1
    a(n) = Cells(n, 1)
  Loop
```

2) Отсортируем его

```
For i = 1 To n - 1
  For j = i + 1 To n
    If a(i) > a(j) Then Call swap(a(i), a(j))
Next j, i
```

3) Осуществим вывод полученного массива

```
  For i = 1 To n
    Cells(i, 2) = a(i)
  Next i
End Sub
```

4) Создадим процедуру swap, которая меняет местами две переменные

```
Sub swap(x, y)
  p = x
  x = y
  y = p
End Sub
```

Результат выполнения программы

	1	2	3
1	Иванов	Абрамов	
2	Абрамов	Абрикосов	
3	Яблочкин	Баранов	
4	Абрикосов	Готглрьев	
5	Наганов	Иванов	
6	Петров	Наганов	
7	Баранов	Петров	
8	Готглрьев	Яблочкин	
9			
10			
11			
12			
13			
14			

6.3. Функции

6.3.1. Объявление функции

Описание функции также состоит из заголовка и тела:

Function имя_функции(список_формальных_параметров) As Type

… … … … … … … … … … … … … … …

Тело функции

… … … … … … … … … … … … … … …

End Function

Function – служебное слово;

имя_функции - любой допустимый идентификатор;

список_формальных_параметров - имена формальных параметров и их типы, разделённые запятой;

Type - тип возвращаемого функцией значения;

Как и процедуры функции являются самостоятельными подпрограммами и могут принимать параметры, выполнять ряд операторов и изменять значения своих параметров.

В отличие от процедуры, имя функции ***возвращает значение в вызывающую процедуру***.

6.3.2. Использование функций

Существуют три различия между подпрограммами типов *Sub* и *Function*:

Возвращаемое функцией значение присваивается самому имени функции. В теле функции всегда должен быть хотя бы один оператор, присваивающий значение имени функции

Как и переменные функции имеют тип, который определяет тип возвращаемого значения. В отсутствие ключевого слова ***As*** в операторе определения процедуры ей назначается по умолчанию тип ***variant***.

Обращение к функции осуществляется по имени с указанием списка фактических параметров Возвращаемое функцией значение можно использовать в выражениях в программе.

Пример

```
y = fun_1(q )
z = sr_arifm(a, b) - sr_arifm(c, d)
x = fun_1 ( q ) / sr_arifm(fun_1 (p), fun_1 (s))
```

Пример. Вычислить количество перестановок P_n ,сочетаний C_n^m , размещений A_n^m , сочетаний с повторениями $\overline{C}_n^m$, и размещений с повторениями $\overline{A}_n^m$.

Для решения задачи напишем функцию *fact*, которая будет возвращать значения факториала натурального числа и использоваться в программе неоднократно.

```
Function fact(n) As Double
  fact = 1
  For i = 1 To n
    fact = fact * i
  Next
End Function
```

Затем, составим коды функций *soch* и *razm*, уже с использованием функции *fact*.

```
Function soch(m, n)
  soch = fact(n) / (fact(n - m) * fact(m))
End Function
```

```
Function razm(m, n)
  razm = fact(n) / fact(n - m)
End Function
```

И, наконец, пишем код основной программы, позволяющий использовать все составленные подпрограммы.

```
Sub Комбинаторные_конфигурации()
    m = Cells(3, 2)
    n = Cells(5, 2)
    Cells(3, 5) = fact(n) 'Вывод перестановок из n
    Cells(3, 7) = soch(m, n) 'Вывод сочетаний из n  по m
    Cells(3, 9) = razm(m, n) 'Вывод размещений из n  по m
    Cells(6, 7) = soch(m, n + m - 1) 'Вывод сочетаний из n  по m
    Cells(6, 9) = n ^ m 'Вывод размещений из n  по m
End Sub
```

Результат выполнения программы приведен ниже

	1	2	3	4	5	6	7	8	9
1	Комбинаторные конфигурации								
2									
3	$m=$	7		$P_n =$	479 001 600	$C_n^m =$	792	$A_n^m =$	3 991 680
4									
5	$n=$	12							
6						$\overline{C}_n^m =$	31 824	$\overline{A}_n^m =$	35 831 808
7									

Пример. Найти все простые числа из промежутка [a, b]

Код программы будет проще, если мы напишем функцию *simple*, позволяющую определить, является ли заданное число простым.

```
Function simple(n) As Boolean
  d = 1
  k = Sqr(n)
  Do
    d = d + 1
  Loop Until n Mod d = 0 Or d > k
  If d > k Then simple = True
End Function
```

Затем, перебирая все подряд все натуральные числа из заданного промежутка, с помощью созданной функции определяем и выписываем те из них, значение функции *simple* в которых равно *True*.

```
Sub Нахождение_простых_чисел()
  Dim x1 As Long, x2 As Long
  x1 = InputBox("Задайте начало интервала [a, b]")
  x2 = InputBox("Задайте конец интервала [a, b]")
  For p = x1 To x2
    If simple(p) = True Then t = t + Str(p) + "; "
  Next p
  MsgBox (Left(t, Len(t) - 2))
End Sub
```

6.3.3. Рекурсивные функции

Под **рекурсией** в программировании понимают подпрограмму, которая вызывает сама себя.

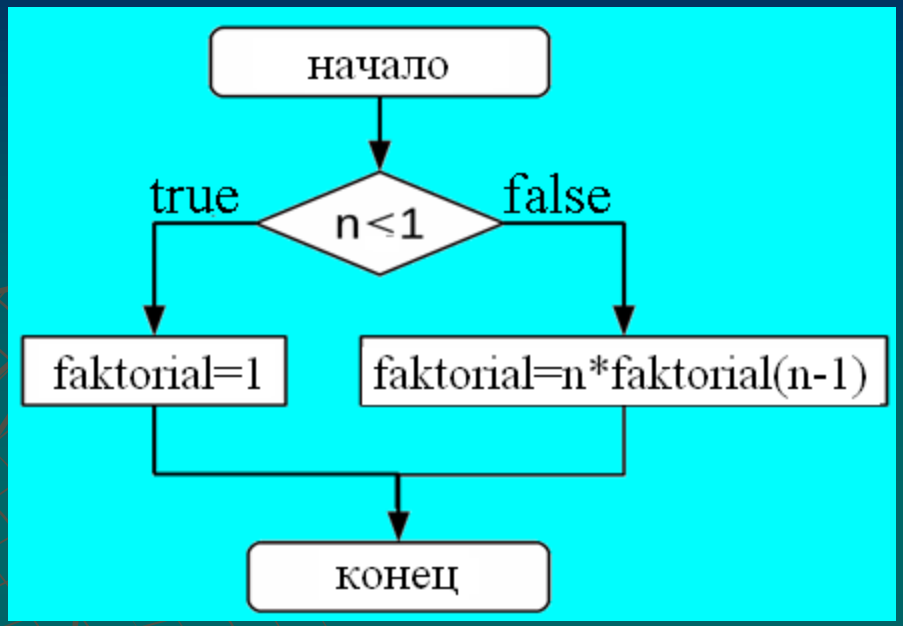
Рекурсивные функции используют реализации рекурсивных алгоритмов. Достоинством рекурсии является компактная запись, а недостатком расход памяти на повторные вызовы функций и передачу параметров. Кроме того, существует опасность переполнения памяти.

В рекурсивной функции необходимо обязательно **предусмотреть завершение рекурсивного вызова**!!!

Иначе функция **никогда** не завершит свою работу!

Пример. Вычислить факториал числа **n**.

Создадим функцию fact, алгоритм которой представлен на рисунке.

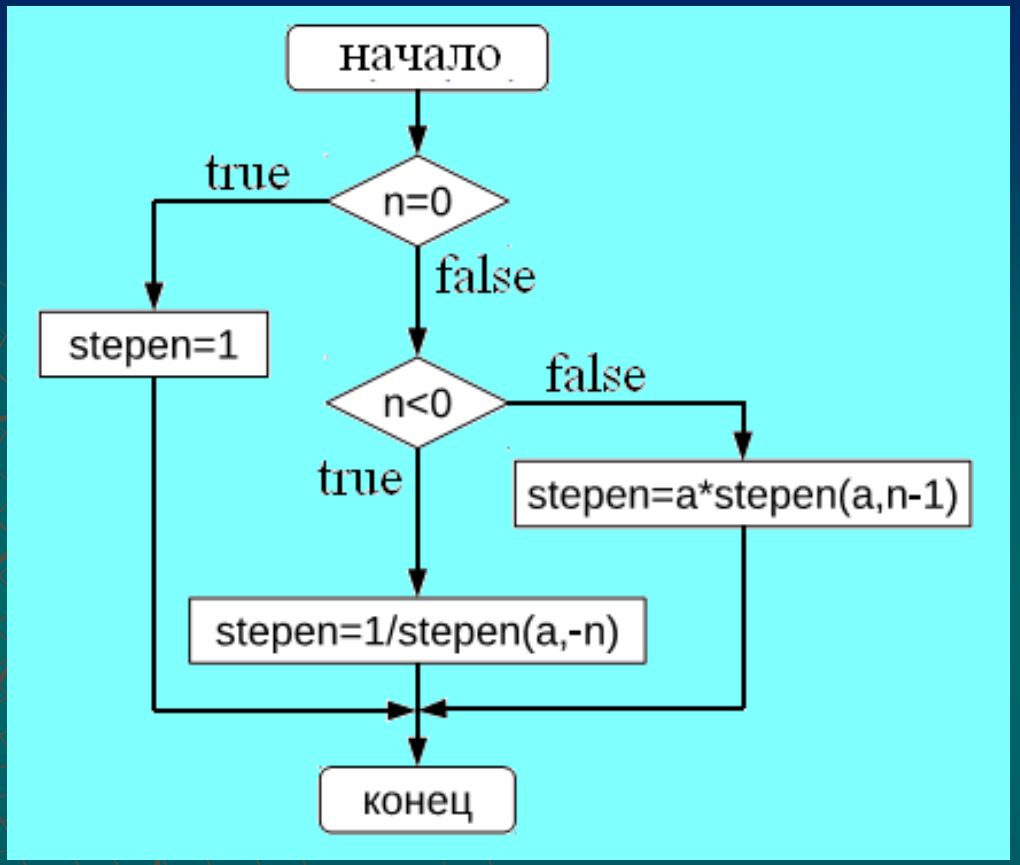


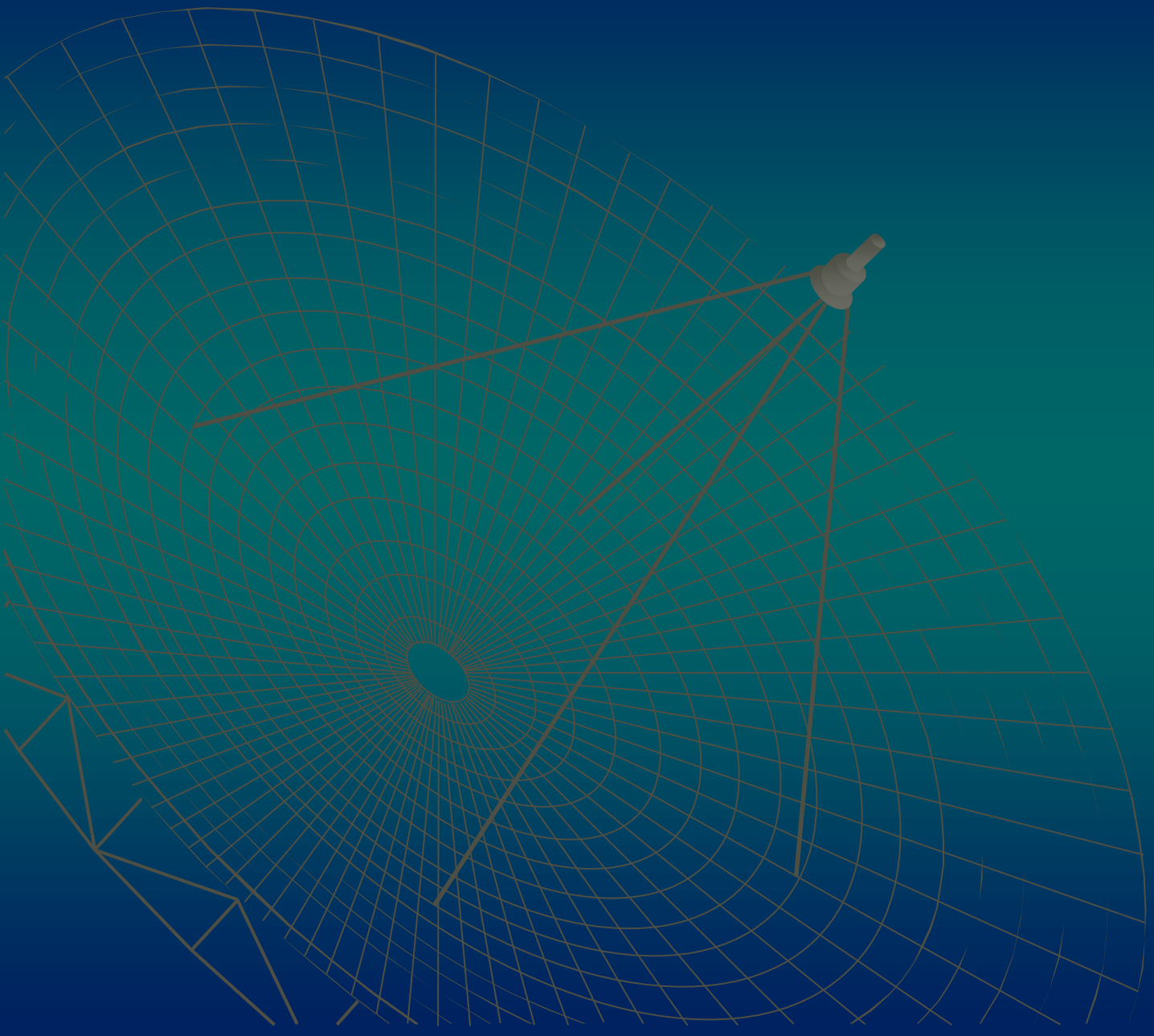
Реализация функции fact

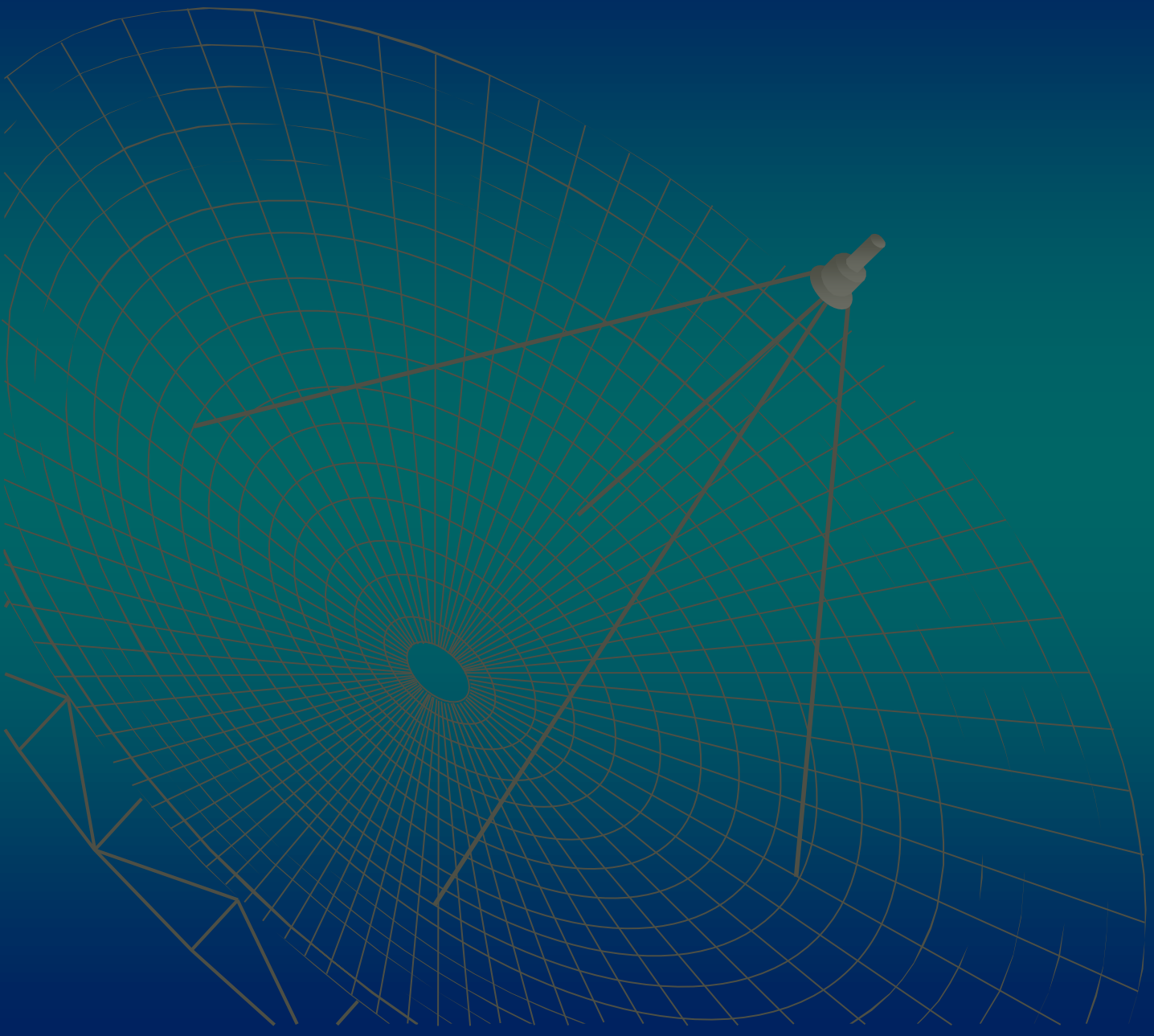
```
Function fact(n) As Long
  If n < 1 Then
      fact = 1
    Else
      fact = fact(n - 1) * n
  End If
End Function
```

Пример. Вычислить n-ю степень числа a (n - целое число)..

Создадим функцию stepen, алгоритм которой представлен на рисунке.

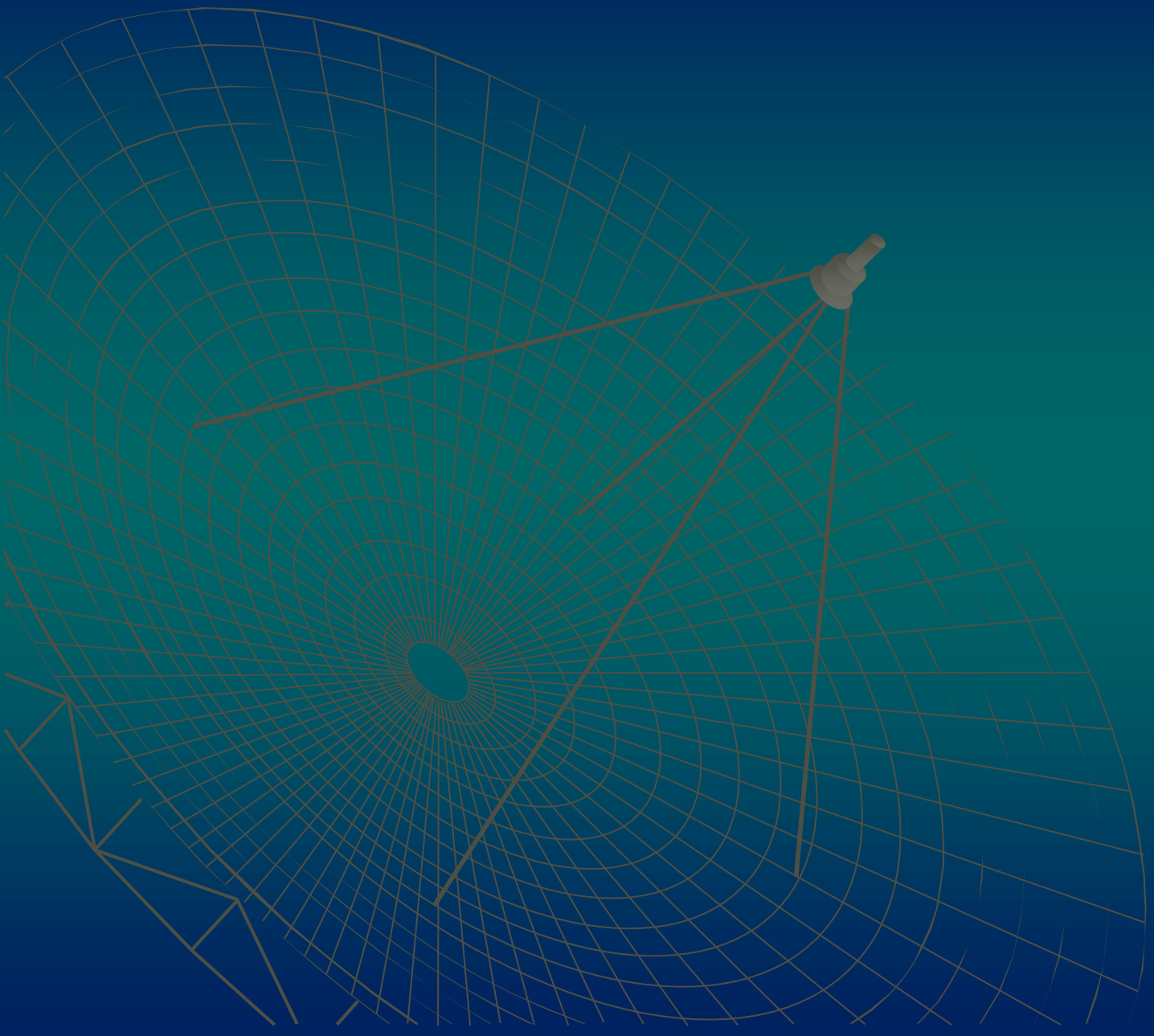






Глава 7. Обработка файлов

7.1.Управление файлами
7.2. Работа с дисками и каталогами
7.3. Атрибуты файлов
7.4. Поиск файлов
7.5. Навигация файлов
7.6. Получение информации о файлах



7.1. Понятие управления файлами

Visual Basic for Applications предоставляет различные операторы, функции и методы для выполнения общих задач управления файлами.

Управление файлами (*file management*) - термин, используемый для описания действий, которые выполняются с файлами, сохраненными на дисковых носителях.

Управление файлами включает следующие действия:

- ✓копирование файлов;
- ✓переименование файлов;
- ✓удаление неиспользуемых файлов;
- ✓перемещение файлов;
- ✓создание или удаление каталогов диска;
- ✓просмотр списка файлов;
- ✓определение размера файла, даты, времени модификации, атрибутов файлов или каталогов.

7.2. Работа с дисками и каталогами

7.2.1. Получение спецификации каталога

Информацию о пути текущего каталога можно получить, используя функцию CurDir (*current directory*). Функция CurDir возвращает строку, которая содержит полный путь текущей папки, включая буквенную метку диска.

Функция CurDir имеет следующий синтаксис:

CurDir[(drive)]

Здесь *drive* представляет любое выражение типа String и указывает функции CurDir, текущая папка какого диска необходима пользователю. Если аргумент *drive* опущен, CurDir возвращает текущую папку текущего диска.

Пример. Вывести текущую папку текущего диска, диска D:.

```
Sub Вывод_текущей_папки()
  MsgBox "Текущая папка " & CurDir
  MsgBox "Текущая папка диска D: " & CurDir("D:")
End Sub
```

7.2.2. Изменение текущего диска

Для изменения текущего диска необходимо использовать оператор ChDrive (*change drive*).

Оператор ChDrive имеет следующий синтаксис:

ChDrive *drive*

Аргумент *drive* — это любое выражение типа String, представляющее буквенную метку диска.

Если задается символ, не являющийся одной из букв от A до Z, VBA отображает runtime-ошибку. VBA отображает также runtime-ошибку, если задается буквенная метка для диска, которого фактически нет в данной компьютерной системе.

Пример. Изменить текущий диск на диск E:.

```
Sub Изменение_текущего_диска ()
  MsgBox "Текущая папка " & CurDir
  ChDrive "e"
  MsgBox "Текущая папка " & CurDir
End Sub
```

7.2.3. Изменение текущего каталога

Для изменения текущей папки на какую-либо другую используется оператор ChDir (*change directory*).

Оператор ChDir имеет следующий синтаксис:

ChDir *path*

Аргумент *path* представляет любое выражение типа String, имеющее результатом допустимый путь папки; *path* может содержать буквенную метку диска, однако ChDir изменяет текущую папку диска на указанную в *path* без изменения текущего диска.

Пример. Изменить текущую папку на корневой каталог, а затем на каталог «\Методические материалы\Программирование».

```
Sub Изменение_текущей_папки ()
  MsgBox "Текущая папка " & CurDir
  ChDir "\"
  MsgBox "Текущая папка " & CurDir
  ChDir "\ Методические материалы\Программирование "
  MsgBox "Текущая папка " & CurDir
End Sub
```

7.2.4. Создание каталогов

Для создания дисковой папки используется оператор MkDir (*make directory*).

Оператор MkDir имеет следующий синтаксис:

MkDir *path*

Аргумент *path* представляет любое выражение типа String, которое имеет результатом допустимый путь папки.

Если указано только имя новой папки, MkDir создает новую папку в текущем каталоге.

Path может содержать буквенную метку диска. Если буквенная метка диска не включена в аргумент *path*, MkDir создает новую папку на текущем диске. MkDir не изменяет текущего диска или папки.

Пример использования MkDir.

```
Sub создание_каталогов()
  MsgBox "Текущая папка " & CurDir
  t = InputBox("Введите имя новой папки"):  MkDir t
  MkDir "\c:\Примеры"
  MkDir "\Методические материалы\Программирование\Примеры"
End Sub
```

7.2.5. Удаление каталогов

Для удаления дисковой папки используется оператор RmDir. *(remove directory).*

Оператор RmDir имеет следующий синтаксис:

RmDir *path*

Аргумент *path* представляет любое выражение String, имеющее результатом допустимый путь папки.

Path может содержать буквенную метку диска. Если буквенная метка не включена в аргумент *path*, RmDir удаляет папку на текущем диске.

Если указано только имя удаляемой папки, то RmDir удаляет папку а текущем каталоге.

Пример использования RmDir.

```
Sub удаление_каталогов()
  MsgBox "Будет удалена папка c:\Примеры"
  RmDir "c:\Примеры"
  MsgBox "Текущая папка " & CurDir
  t = InputBox("Введите имя удаляемой папки")
  RmDir t
End Sub
```

7.3. Атрибуты файлов
7.3.1. Определение атрибутов

Каждый файл может иметь один или несколько атрибутов:

Archive. Атрибут *Archive* указывает, изменялся ли файл со времени его последнего резервирования.

Directory. Файл имеющий атрибут *Directory*, в действительности, является каталогом.

Hidden. Windows «скрывает» файл, имеющий атрибут *Hidden*, не показывая его в большинстве случаев при отображении папки.

Normal. Атрибут файла *Normal* является признаком отсутствия каких-либо специальных атрибутов.

Read-Only. Атрибут *Read-Only* означает, что операционная система препятствует изменению или удалению этого файла.

System. Атрибут указывает, что файл является частью операционной системы компьютера.

Volume Label. Атрибут *Volume Label* информирует о том, что файл является меткой тома диска.

Alias. Атрибут *Alias* определяет, что имя файла является алиасным и файл доступен только в Macintosh.

7.3.2. Константы атрибутов файла

Windows представляет каждый атрибут файла уникальным числом и сохраняет это число с информацией об имени и размере файла.

Если файл имеет несколько атрибутов, кодовые числа для каждого атрибута складываются и записываются вместе с именем файла и его длиной.

Константа	Код	Атрибут
vbNormal	0	Normal
vbReadOnly	1	Read-Only
vbHidden	2	Hidden
vbSystem	4	System
vbVolume	8	Volume Label
vbDirectory	16	Directory
vbArchive	32	Archive
vbAlias	64	Alias

7.3.3. Получение атрибутов файла

Для определения атрибутов файла используется функция **GetAttr**. Синтаксис:

GetAttr(pathname)

pathname - любое строковое выражение VBA, представляющее допустимое имя файла.

Pathname может включать буквенную метку диска и полный путь папки. Если буквенная метка не включается, **GetAttr** ищет указанный файл на текущем драйвере диска; если не включается путь папки, функция **GetAttr** выполняет поиск в текущей папке.

Пример. Найдем сумму кодовых чисел атрибутов текущего каталога.

```
Sub атрибуты_текущего_каталога()
  t = GetAttr(".")
  MsgBox "Сумма кодовых чисел атрибутов текущего каталога " & t
End Sub
```

В результате выполнения программы получим числовое значение суммы кодов., например **17**, которое однозначно является суммой кодов 16 (Directory) и 1 (Read-Only).

7.3.4. Изменение атрибутов

Для изменения атрибутов файла предназначен оператор **SetAttr**, который выполняет ту же самую задачу, что и команда Файл|Свойства. Оператор **SetAttr** имеет следующий синтаксис:

SetAttr pathname, attributes

pathname - любое строковое выражение, содержащее допустимую спецификацию файла;

attributes - числовая сумма кодов.

Числовая сумма кодов для *attributes* должно быть числом между **1** и **255** и состоять из одного из кодовых чисел атрибута файла или суммы кодовых чисел атрибутов.

Пример. Изменим сумму кодов атрибутов текущего каталога.

```
Sub изменение атрибутов_текущего_каталога()
  t = SetAttr ". " , 16
End Sub
```

В результате выполнения программы сумма кодов текущего каталога станет **16**, которое однозначно является суммой кодов 16 (Directory) и 1 (Read-Only).

7.4. Поиск файлов
7.4.1. Функция Dir

Для поиска файлов, находящихся в дисковом пространстве используется функция **Dir**.

Функция Dir имеет следующий общий синтаксис:

Dir(pathname[, attributes])

Аргумент ***pathname*** представляет выражение типа String, имеющее результатом допустимое имя файла.

Имя файла может содержать буквенную метку драйвера и полный путь папки. Имя папки может также включать символы «*» и «?».

Аргумент ***attributes*** является числом, представляющим атрибуты тех файлов, которые необходимо найти.

При использовании *attributes* функция Dir выполняет поиск файлов, имеющих эти атрибуты. Если аргумент *attributes* отсутствует, функция Dir выполняет поиск всех файлов, кроме имеющих атрибуты Hidden, Volume Label, Directory или System.

При вызове функции Dir с аргументом *pathname* она возвращает строку, содержащую имя ***первого*** найденного ею файла, совпадающее с именем файла в аргументе *pathname*.

Если имя файла содержит символы универсального сопоставления * или **?**, то для того, **чтобы найти все файлы** в каталоге, совпадающие с этим именем файла, необходимо использовать функцию Dir в два этапа:

1) Вызывать **Dir** с аргументом *pathname* для получения первого совпадающего файла.

2)Вызывать **Dir** неоднократно без аргументов до тех пор, пока **Dir** не возвратит пустую строку.

Как только **Dir** возвращает пустую строку, то больше не остается файлов, совпадающих с этим именем файла.

Пример. Следующая программа осуществляет вывод всех файлов, кроме скрытых и системных из текущей папки «\Файловые примеры»

```
Sub Поиск_файлов_Dir()
  ChDrive ("k")
  ChDir "\Файловые примеры"
  Cells(1, 1) = "Все файлы текущей папки " & CurDir
  Do
    k = k + 1
    If k = 1 Then t = Dir("*.*") Else t = Dir
    If t <> "" Then
      Cells(k + 1, 1) = Str(k) + ")"
      Cells(k + 1, 2) = t
    End If
  Loop Until t = ""
End Sub
```

Пример. Если водить маски поиска с помощью оператора Inputbox, то программа выведет все файлы, соответствующие заданному шаблону.

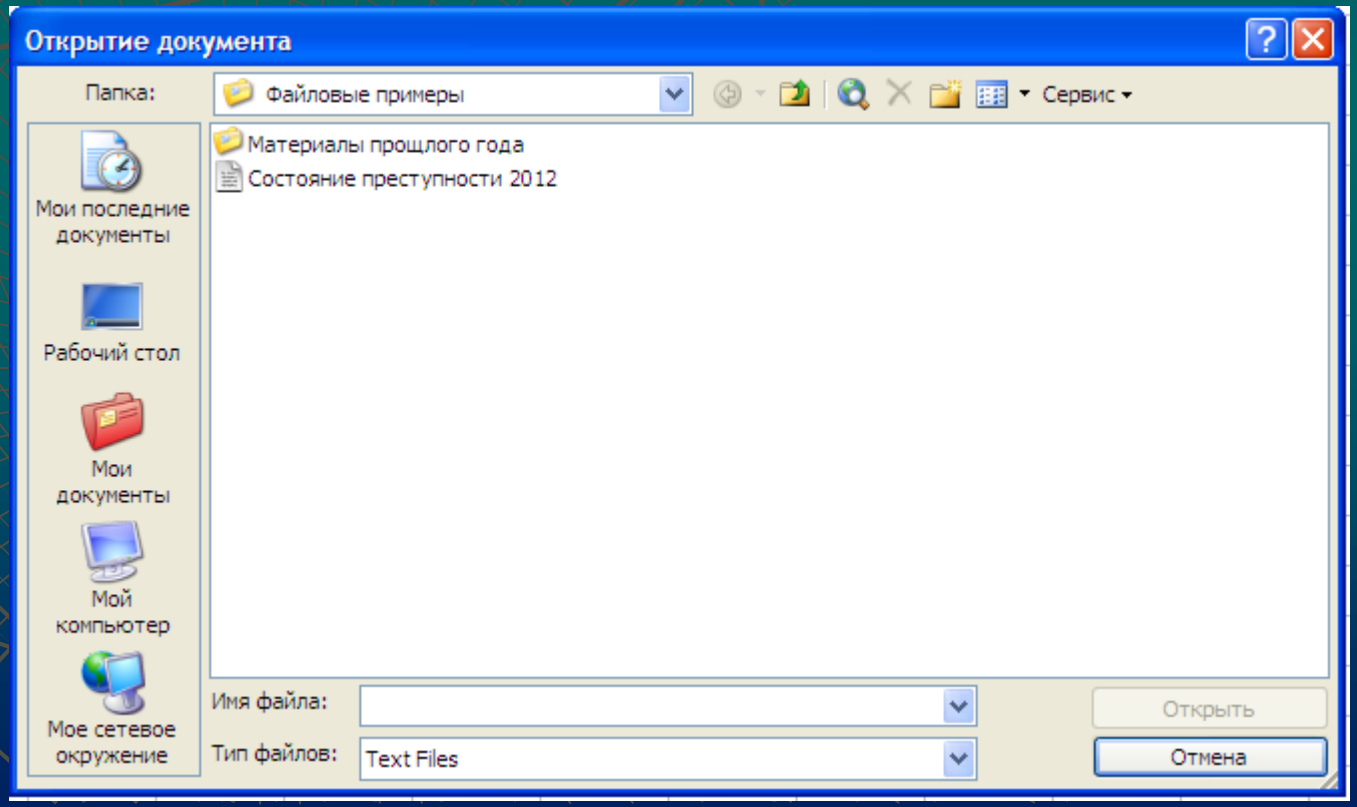
```
Sub Поиск_файлов_по_шаблону()
  Cells(1, 1) = "Все файлы текущей папки " & CurDir
  m = InputBox("Введите маску поиска")
  Do
    k = k + 1
    If k = 1 Then t = Dir(m) Else t = Dir
    If t <> "" Then
      Cells(k + 1, 1) = Str(k) + ")"
      Cells(k + 1, 2) = t
    End If
  Loop Until t = ""
End Sub
```

7.4.2. Метод GetOpenFilename

Метод GetOpenFilename используется для отображения диалогового окна, которое выглядит и работает так же, как диалоговое окно, отображаемое Excel при выборе команды File|Open.

Метод возвращает строку, которая содержит имя файла, выбранного пользователем, включая буквенную метку диска и полны путь папки.

Если пользователь отменяет диалоговое окно, GetOpenFilename возвращает значение False в качестве результата.



Метод GetOpenFilename имеет следующий синтаксис:

object.GetOpenFilename(fileFilter, filterIndex, title, buttonText, multiSelect)

Здесь *object* - обязательная ссылка на Excel-объект Application.

Аргумент *fileFilter* представляет любое допустимое выражение String, специально форматированное для задания фильтров файла, перечисленных в окне раскрывающегося списка Files of type в диалоговом окне Open. Если аргумент *fileFilter* опущен, фильтр файла для раскрывающегося списка Files o type — это All Files (*.*).

Аргумент *filterIndex* представляет любое численное выражение и указывает, какой фильтр файла должен использовать VBA как фильтр по умолчанию для раскрывающегося списка Files of type.

Аргумент *title* представляет любое выражение типа String и является заголовком, отображаемым VBA в диалоговом окне Open.

Аргумент *buttonText* — необязательный параметр, используемый только в Macintosh.

Аргумент *multiSelect* представляет любое выражение или значение Boolean. Если *multiSelect* равно True, GetOpenFilename возвращает массив, содержащий имена всех выбранных файлов.

Пример. Используя **метод** GetOpenFilename, осуществим поиск файлов по маске "*.xls".

```
Sub Просмотр_файлов_в_диалоговом_окне ()
t = Application.GetOpenFilename(«Электронные таблицы (*.xls), *.xls")
Cells(1, 1) = t
End Sub
```

7.5. Навигация файлов
7.5.1. Копирование файлов

Для копирования файла используется оператор FileCopy.

Оператор FileCopy имеет следующий синтаксис:

FileCopy source, destination

Как *source*, так и *destination* являются выражениями типа String, имеющие результатом допустимые имена файла.

Они могут включать полный путь папки и буквенную метку диска. При попытке копировать файл в самого себя VBA выдает runtime-ошибку. VBA также выдает runtime-ошибку при попытке копировать файл, когда нет достаточного дискового пространства для сохранения копируемого файла.

Пример. Использование оператора FileCopy для копирования файла.

```
Sub копирование_файлов()
FileCopy "Докладная записка.doc", "Докладная записка_2.doc "
End Sub
```

Пример. Использование оператора FileCopy для копирования группы файлов, заданных шаблоном.

```
Sub Копирование_файлов_по_шаблону()
  m = InputBox(" Введите маску поиска ")
  Do
    k = k + 1
    If k = 1 Then t = Dir(m) Else t = Dir
    If t <> "" Then
      FileCopy t, " Материалы прошлого года\Копия " + t
    End If
  Loop Until t = ""
End Sub
```

7.5.2. Удаление файла

Для удаления файла используется оператор Kill.

Оператор Kill имеет следующий синтаксис:

Kill pathname

Аргумент *pathname* - это любое выражение типа String, имеющее результатом допустимую спецификацию имени файла;

pathname может включать буквенную метку диска, полный путь папки и символы универсального сопоставления (* и ?).

Если *pathname* включает символы универсального сопоставления, Kill удаляет все файлы, совпадающие со спецификацией в *pathname*.

Пример. Использование оператора Kill для удаления группы файлов, заданных шаблоном.

```
Sub удаление_файлов()
Kill "Материалы прошлого года\*.*"
End Sub
```

Необходимо проверять атрибуты файла перед попыткой удалить его. При попытке удалить файл, имеющий какой-либо из атрибутов Hidden, System или Read-Only VBA выдает runtime-ошибку.

7.5.3. Переименование и перемещение файлов

Для переименования файла или перемещения его в другую папку используется оператор Name.

Оператор Name имеет следующий синтаксис:

Name oldpathname As newpathname

Аргументы *oldpathname* и *newpathname* — это выражения типа String, имеющие результатом допустимые имена файлов. Оба могут содержать полный путь папки, включая буквенную метку диска. Если *oldpathname* и *newpathname* ссылаются на разные папки, VBA перемещает файл в новую папку и изменяет его имя.

Пример. Использование оператора Name для переименования и перемещения файлов.

```
Sub Переименование_и_перемещение_файлов()
Name "Докладная записка_2.doc " As "Докладная записка 20012.doc "
Name "Состояние преступности 2012.txt" As "Материалы прошлого
года\Состояние преступности 2012.txt"
End Sub
```

7.6. Получение информации о файлах

7.6.1. Получение времени и даты

Всякий раз, при создании и изменении дискового файла, файловая система Windows записывает дату и время (в соответствии с часами данной компьютерной системы).

Чтобы сделать эту информацию доступной в процедурах VBA, используется функция FileDateTime. Эта функция возвращает информацию о дате и времени из файла как VBA-значение типа Date.

Функция FileDateTime имеет следующий синтаксис:

FiledateTime(*pathname*)

Аргумент *pathname* - это любое выражение типа String, имеющее результатом допустимую спецификацию файла; *pathname* может включать буквенную метку диска и полный путь папки, но не может содержать символы универсального сопоставления (* или ?).

Пример. Использование оператора FileDateTime.

```
Sub Получение_времени_и_даты()
t = "Отчеты за 2013.doc"
MsgBox t & "   " & FileDateTime(t)
End Sub
```

7.6.2. Получение длины файла

Для того чтобы определить длину файла, можно использовать функцию FileLen . Функция возвращает длину файла в байтах.

Функция FileLen имеет следующий синтаксис:

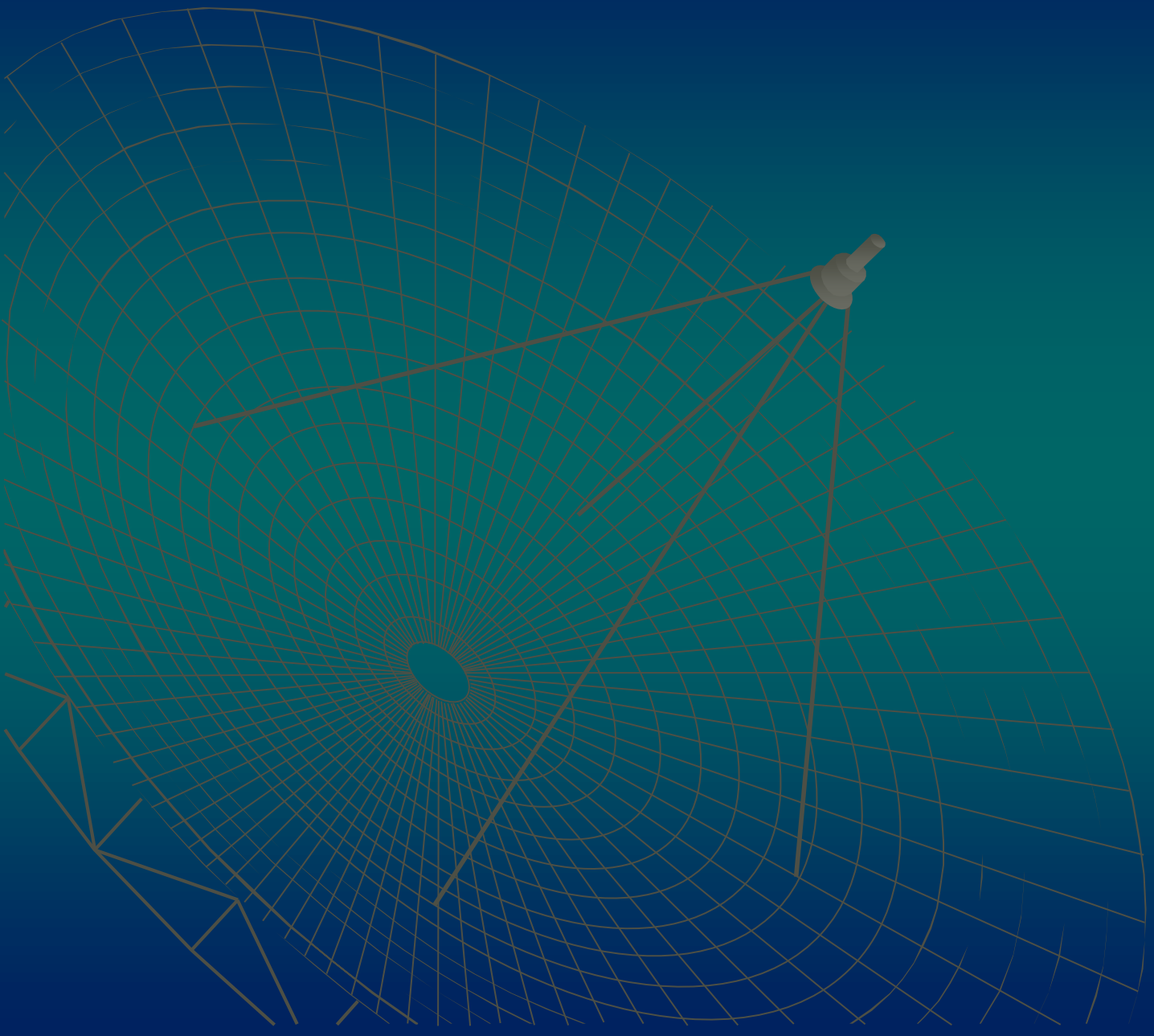
FileLen(*pathname*)

Аргумент *pathname* - выражение типа String, имеющее результатом допустимую спецификацию имени файла; *pathname* может включать буквенную метку диска и полный путь папки, но не может включать символы универсального сопоставления.

Если *pathname* определяет открытый файл, FileLen возвращает размер файла, каким он был, когда файл был сохранен на диске последний раз.

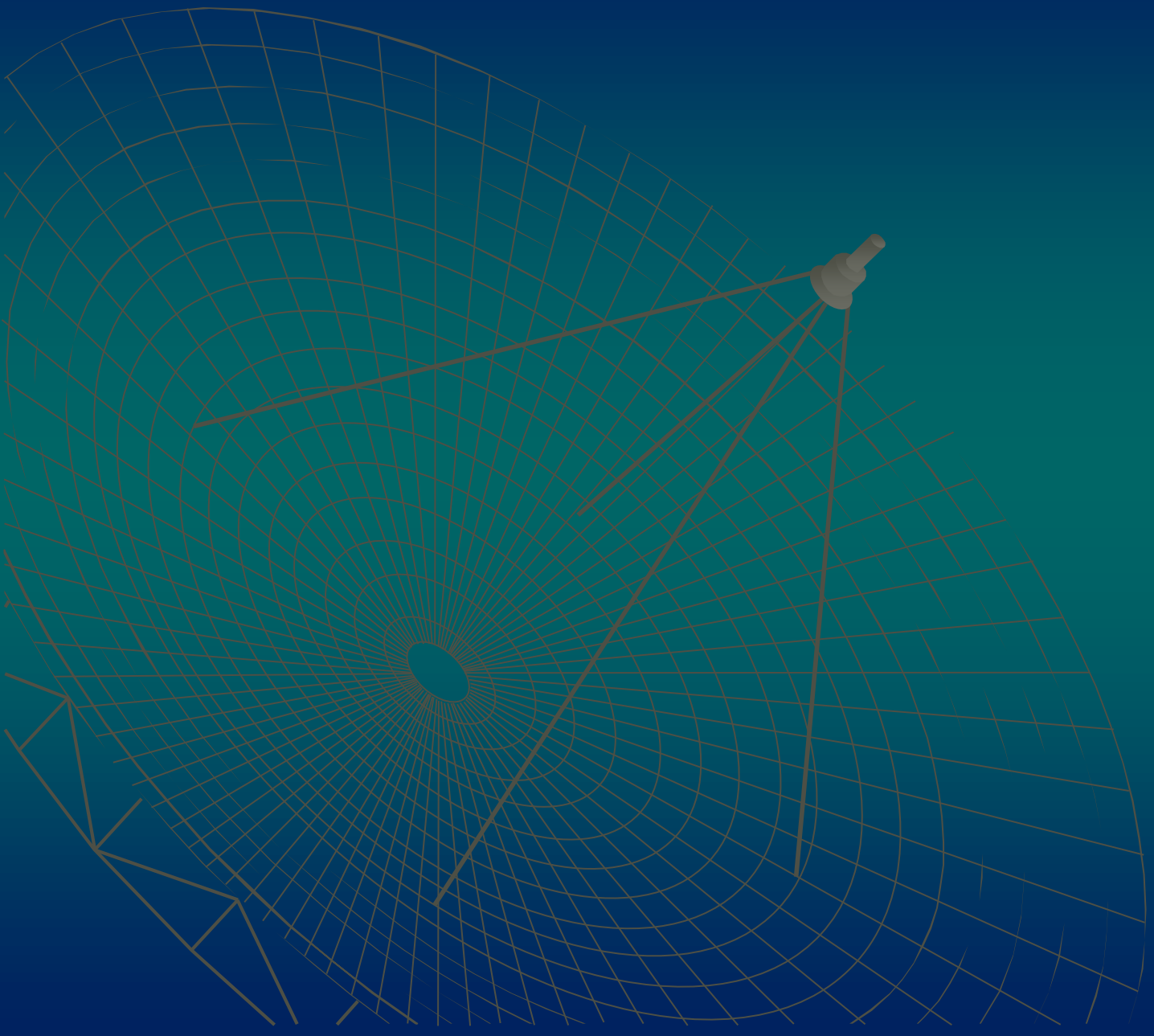
Пример. Использование оператора FileDateTime.

```
Sub Получение_длины_файла()
t = "Отчеты за 2013.doc"
MsgBox t & "   " & FileLen(t)
End Sub
```



Глава 8. Объектно-ориентированное программирование

8.1. Основные понятия объектно-ориентированного программирования
8.2. Использование объектов
8.3. Использование свойств и методов объектов
8.4. Ссылка на объекты с помощью With...End With
8.5. Формы
8.6. Элементы управления



8.1. Основные понятия объектно-ориентированного программирования

8.1.1. Объект

В середине 80-х годов была разработана новая концепция в программировании, известная как объектно-ориентированное программирование (ООП) (object-oriented programming, OOP).

Идея объектно-ориентированного программирования заключается в том, что **программное приложение** (как и реальный мир вокруг нас) **должно состоять из особых объектов**, каждый из которых имеет собственные специфические качества и поведение.

Выделение отдельных объектов при решении задачи из какой-либо предметной области, исходя из особенностей задачи, называется объектной **декомпозицией**.

Объекты состоят из данных, описывающих свойства этих объектов и процедур, обрабатывающих эти данные.

Объединение данных и обрабатывающих их функций и процедур в виде отдельных объектов называется **инкапсуляцией**.

Одни объекты могут состоять из других объектов.

Пример. Объект приложения Excel рабочая книга, состоит из объектов листов, которые, в свою очередь, состоят из ячеек или графических объектов.

Создание новых объектов путем использования уже существующих называется ***наследованием***.

Наследование дает программисту ряд преимуществ:

- не нужно повторно разрабатывать новый код, весь код для существующих объектов может быть использован для новых объектов;
- вероятность ошибок резко снижается. Если код для уже существующих объектов был уже отлажен и проверен;
- код программы становится более понятным, для того чтобы понять как работают объекты, созданные путем наследования достаточно понять как работают добавленные данные и функции.

Полиморфизмом называется возможность одних и тех же функций или процедур по разному обрабатывать данные, принадлежащие разным объектам.

8.1.2. Свойства объекта

Как и объекты реального мира, объекты программирования имеют различные присущие им качества или ***свойства (properties)***.

Объекты в Word и Excel имеют свойства, определяющие их вид и поведение:

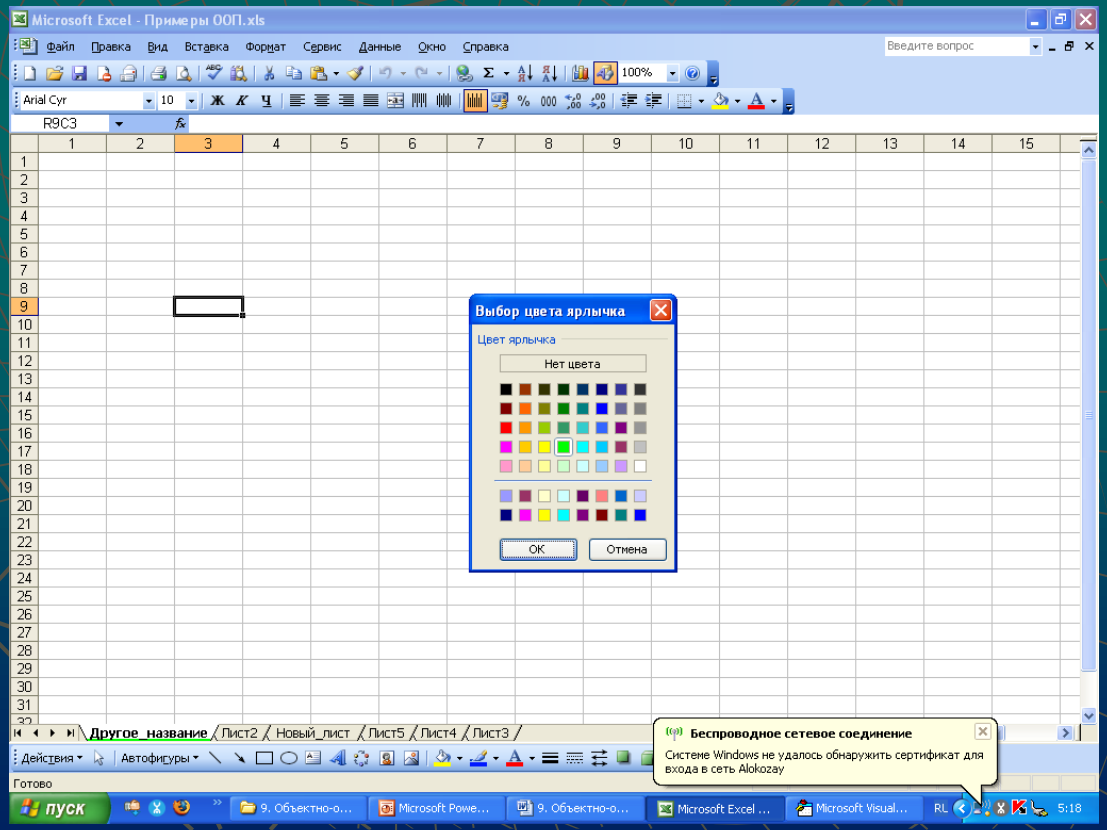
- текст в документах и таблицах может иметь соответствующий цвет, шрифт, начертание, размер, атрибуты отображения, эффекты;
- рабочий лист Excel имеет заголовок, цвет ярлычка, видимость;
- строки в рабочем листе или таблице, имеют свойство высоты;
- столбцы имеют свойство ширины и т.д.

Свойства регулируют вид и поведение объекта. Чтобы изменить вид и поведение объекта, необходимо изменить его свойства.

В Excel можно изменить поведение рабочего листа, изменяя свойство вычисления с автоматического на ручное, или можно изменить его вид, задав новый цвет для текста или графики в листе.

Чтобы узнать о текущем виде и поведении объекта, необходимо узнать о его свойствах. Можно определить диск, папку и имя рабочей книги Excel, рассмотрев свойство **FullName** этой рабочей книги.

Некоторые объекты имеют свойства с одними и теми же или подобными именами: объекты **Application**, **Workbook** и **Worksheet** в Excel – все имеют свойство **Name**. Каждый объект рабочего листа сохраняет данные для своих свойств внутри самого рабочего листа вместе с пользовательскими данными.



8.1.3. Методы объекта

Объекты реального мира почти всегда имеют тот тип присущего им поведения или действия, который они могут выполнить. Объекты VBA также имеют поведение и возможности, называемые *методами (methods)*.

Методы изменяют значения свойств объектов;

Методы выполняют также действия с данными (или над данными), сохраняемыми объектом.

Методы во многом похожи на процедуры VBA, но связаны с объектом; к методам объекта можно обратиться, только используя объект.

Для вызова метода объекта (функции обработки данных объекта) указывается не только наименование метода, но и объект (имя), которому принадлежит метод.

Несмотря на то, что объекты одного и того же типа совместно используют код для своих методов, метод считается частью объекта; когда вызывается определенный метод для конкретного объекта, метод воздействует только на объект, посредством которого происходит обращение к этому методу.

8.1.4. События

Важнейшая концепция VBA - обработка событий.

События (*event*) – это любые отслеживаемые изменения, происходящие с компонентами программы.

К событиям относятся щелчки мышью, нажатия на клавиши, открытие и закрытие окон или форм, перемещение окон на экране, изменение значений полей.

VBA построен таким образом, чтобы создавать на нем программы, управляемые событиями (*event-driven*). Такие методы построения программ противопоставляются процедурному программированию.

Наиболее употребляемые события

Событие	Описание
Initialize	происходит при подготовке формы к открытию
Click	реакция на одиночный щелчок мыши
DblClick	реакция на двойной щелчок мыши

8.1.5. Классы объекта

Программные объекты группируются в иерархические ***классы*** или коллекции объектов, как объекты реального мира вокруг нас.

Все объекты в одном и том же классе имеют одни и те же (или подобные) свойства и методы.

Каждый класс объектов может содержать один или несколько подклассов. Объекты, принадлежащие к определенному классу объектов, называются ***членами (members)*** этого класса.

VBA дает возможность пользователю создавать собственные классы объектов добавлением модуля класса в проект. Новый модуль класса должен содержать все определения свойств и методов для объекта.

После описания нового класса можно создавать его экземпляры (объекты этого типа).

8.2. Использование объектов

Операторы программ VBA, использующие объекты, обычно выполняют одно или несколько из следующих действий:

- определяют текущее состояние или статус объекта путем выборки значения, сохраняемого в определенном свойстве;
- изменяют состояние или статус объекта установкой значения; сохраненного в определенном свойстве;
- используют один из методов объекта, обеспечивая выполнение объектом одной из его встроенных задач.

Например, можно определить имя активных в данный момент рабочей книги или рабочего листа в Excel;

можно изменить их имена;

можно добавить рабочего листа в рабочую книгу.

8.2.1. Синтаксис определения

В операторах VBA нужно использовать следующий общий синтаксис для определения свойства или метода объекта:

Object.identifier

Object – любая допустимая ссылка на объект.

Объектные ссылки создаются заданием переменной для ссылки на объект или использованием методов или свойств объектов, возвращающих объектную ссылку.

Identifier – любое допустимое имя свойства или метода;

(VBA отображает сообщение о runtime-ошибке при попытке использовать свойства или методы, которые не являются частью указанного объекта.)

Точка отделяет объектную ссылку от имени свойства или метода.

Эта *точка-разделитель* также соединяет объектную ссылку с идентификатором свойства или метода.

При обращении к свойству или методу посредством объекта, необходимо определять объектную ссылку и идентификатор свойства или метода вместе.

8.2.2. Примеры использования

Пример. Определить имя активных в данный момент рабочей книги или рабочего листа в Excel.

```
Sub program_1
    t1 = ActiveWorkbook.Name
    t2 = ActiveSheet.Name
    MsgBox "Активный документ:  " & t1
    MsgBox "Активный лист:  " & t2
End Sub
```

В диалоговом окне функции MsgBox, используя свойство Name для объектов ActiveWorkbook и ActiveSheet будут отображены последовательно наименование активного документа и наименование активного листа.

Пример. Изменить имя активного рабочего листа в Excel с текущего на «Другое название», изменить цвет ярлычка листа на красный.

```
Sub program_2()
    ActiveSheet.Name = "Другое название"
    ActiveSheet.Tab.ColorIndex = 3
End Sub
```

Свойству Name для объекта ActiveSheet присваивается имя «Другое название», а затем свойство ColorIndex объекта ActiveSheet.Tab изменяется на «3», что соответствует красному цвету.

Пример. Вставить новый лист и разместить его на третью позицию.

```
Sub program_3
    Sheets.Add
    ActiveSheet.Name = " Новый_лист"
    Sheets(" Новый_лист ").Move After:=Sheets(3)
End Sub
```

Используем метод Add для вставки нового объекта Sheets (листа). Свойству Name для объекта ActiveSheet присваивается имя «Новый_лист», а затем, используя метод Move для объекта Sheets("Новый_лист") передвигаем лист на позицию третьего листа.

8.2.3. Общие объекты Excel

Объект	Описание
Application	Само приложение Excel (host-приложение)
Chart	Диаграмма в рабочей книге
Font	Этот объект содержит атрибуты шрифта и стиля для текста, отображаемого в рабочем листе
Name	Заданное имя для диапазона ячеек рабочего листа
Range	Диапазон ячеек (одна или более) или именованный диапазон в рабочем листе
Window	Любое окно в Excel; окна используются для отображения рабочих листов, диаграмм и т.д.
Worksheet	Рабочая таблица в книге
Workbook	Открытая рабочая книга

8.2.4. Общие объекты Word

Объект	Описание
Application	Само приложение Word (host-приложение).
Bookmark	Отдельная закладка в документе.
Document	Открытый документ.
Font	Содержит атрибуты шрифта и стиля для текста, отображаемого в объекте.
Paragraph	Отдельный параграф в документе.
Range	Непрерывная область в документе.
Style	Встроенный или определенный пользователем форматирующий стиль в документе.
Table	Отдельная таблица в документе.
TableOfContents	Отдельная таблица содержания документа.
Template	Шаблон документа.
Window	Любое окно в Windows. '

8.3. Использование свойств и методов объектов

Свойства объектов можно использовать только двумя способами: получать значение свойства или устанавливать его.

Не все свойства объекта изменяемы:

- свойства объектов, которые нельзя изменять, называют свойствами, доступными только на чтение (read-only);

- свойства, которые можно устанавливать, называют свойствами, доступными на чтение/запись (read-write).

Свойства обычно содержат численные, строковые, значения типа **Boolean**, хотя некоторые свойства могут возвращать значения типа **Object** или другие типы данных.

Обращение к свойству объекта имеет следующий синтаксис:

Object.Property

Object — допустимая объектная ссылка, тогда как *Property* представляет любое допустимое имя свойства для объекта, на который выполняется ссылка.

8.3.1. Синтаксис присваивания

Свойства используются в выражениях так же, как любое другое значение переменной или константы.

Можно присваивать значение свойства переменной, использовать свойства объектов в выражениях как аргументы к функциям и процедурам или как аргументы для методов какого-либо объекта.

Чтобы присвоить некоторой переменной значение свойства объекта, используйте следующий синтаксис:

Variable = Object.Property

Variable – любая допустимая переменная, имеющая совместимый со свойством объекта тип;

Object любая допустимая ссылка на объект;

Property – любое допустимое имя свойства для объекта, на который выполняется ссылка.

Можно также использовать свойство объекта непосредственно в каком-либо выражении или в качестве аргумента функции или процедуры'

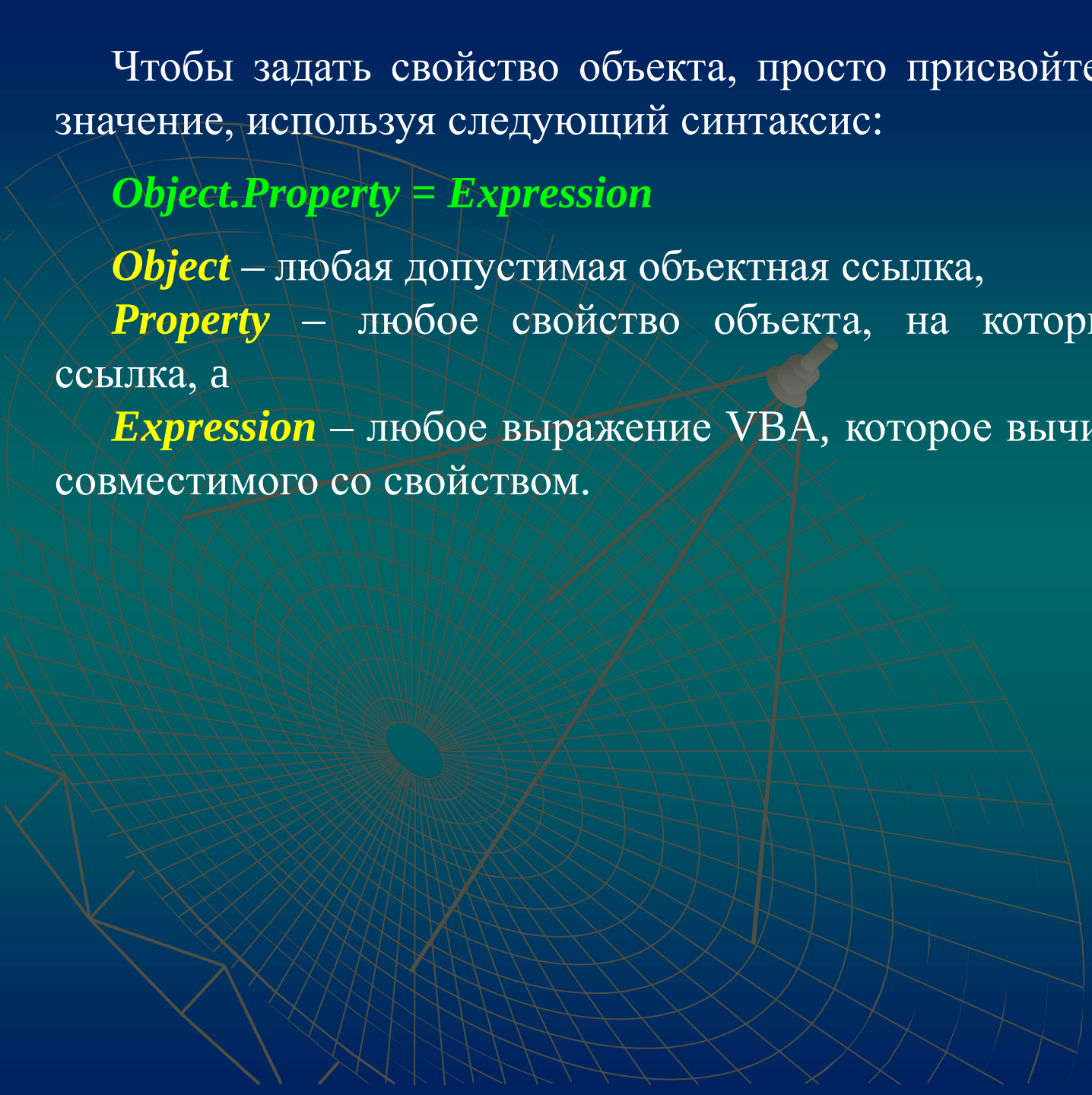
Чтобы задать свойство объекта, просто присвойте свойству новое значение, используя следующий синтаксис:

Object.Property = Expression

Object – любая допустимая объектная ссылка,

Property – любое свойство объекта, на который выполняется ссылка, а

Expression – любое выражение VBA, которое вычисляется до типа, совместимого со свойством.



Пример. Вывести на лист полное имя файла электронной таблицы. Отобразить в строке состояния запись «Краснодарский университет МВД России».

```
Sub program_4()
    Cells(1, 1) = ActiveWorkbook.FullName
    Application.StatusBar = "Краснодарский университет МВД России"
End Sub
```

Ячейке (1, 1) присвоим значение свойства FullName объекта ActiveWorkbook (активной рабочей книги). Свойству StatusBar для объекта Application (приложение) присваивается значение записи «Краснодарский университет МВД России».

Пример. Данный программный код закрашивает ячейки первого столбца текущего листа в разные цвета.

```
Sub program_5()
    For i = 0 To 10
        For j = 0 To 10
            For k = 0 To 10
                s= 100 * i + 10 * j + k + 1
                Cells(s, 1).Interior.Color = RGB(i * 25, j * 25, k * 25)
            Next k
        Next j
    Next i
End Sub
```

В циклах вызывается свойство Color объекта Cells(s, 1).Interior (активной рабочей книги). Свойству StatusBar для объекта Application (приложение) присваивается значение записи «Краснодарский университет МВД России».

8.3.3. Некоторые свойства объектов в Excel

Свойство	Тип/Что означает	Объекты
Path	String: драйвер и каталог, в котором сохранен объект	Application, Workbook
Selection	Object: текущий выделенный фрагмент	Application, Window
Sheets	Коллекция листов рабочей книги	Application, Workbook
StatusBar	String: сообщение в статусной строке	Application
ThisWorkBook	Object: рабочая книга, из которой выполняется текущая процедура	Application
Type	Integer: число, указывающее тип объекта	Window, Worksheet, Chart
Value	Действительное значение, отображаемое в ячейке	Range

8.3.4. Общеупотребительные методы объектов Excel

Метод	Назначение	Имеется в объектах
Activate	Активизирует объект	Window, Worksheet, Range, др.
Clear	Удаляет данные, сохраненные в указанном объекте	Range
Close	Закрывает указанный объект	Window, Workbook
Justify	Выравнивает текст, сохраненный в указанном объекте	Range
Save	Сохраняет файл рабочей книги	Application, Workbook
SaveAs	Сохраняет указанный объект в другом файле	Workbook, Worksheet
Select	Выбирает указанный объект	Range, Sheets, Worksheets
Send Keys	Пересылает нажатия клавиши в диалоговые окна в host-приложении	Application
Volatile	Регистрирует функцию как *изменяющуюся*	Application

8.4. Ссылка на объекты с помощью With...End With

Процедуры могут ссылаться на один и тот же объект часто с помощью нескольких операторов в одной строке.

VBA предоставляет особую структуру – **With .. End With**, позволяющую ссылаться на свойства или методы, которые принадлежат одному и тому же объекту, без задания всей объектной ссылки каждый раз.

Структура **With...End With** имеет следующий синтаксис:

```
With Object
        'операторы, использующие свойства и методы Object ;
 End With
```

Object — это любая допустимая объектная ссылка.

Пример. Данный программный код устанавливает параметры некоторых свойств области ("B3:D5«) активного рабочего листа.

```
Sub program_6()
   With Range("B3:D5")
      .Font.Name = "Times New Roman"
      .Font.Size = 14
      .Font.ColorIndex = 5
      .HorizontalAlignment = xlCenter
      .Orientation = 90
   End With
End Sub
```

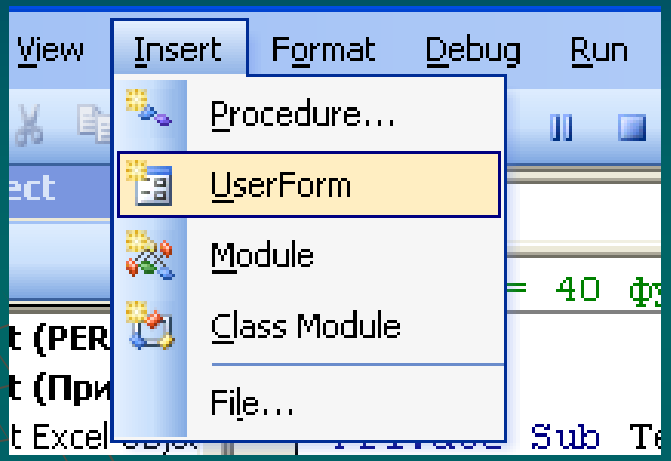
Свойства .Font.Name (название шрифта), .Font.Size (размер шрифта), .Font.ColorIndex (цвет шрифта), HorizontalAlignment (горизонтальное выравнивание), Orientation (ориентация) присваиваются всему объекту Range("B3:D5") (области ячеек "B3:D5").

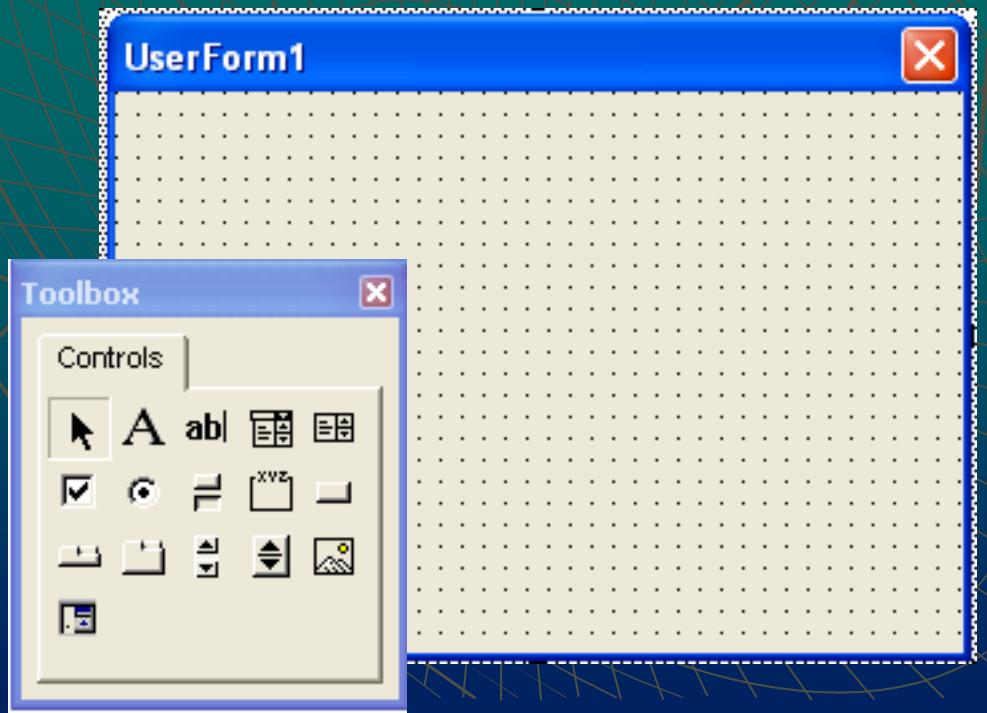
8.5. Формы

Применение встроенных функций InputBox и MsgBox для ввода или вывода данных ограничено. Для предоставления пользователю графического интерфейса часто используются формы VBA.

8.5.1. Создание формы

Для создания новой формы достаточно в редакторе Visual Basic для текущего проекта выбрать **Insert -> User Form**.

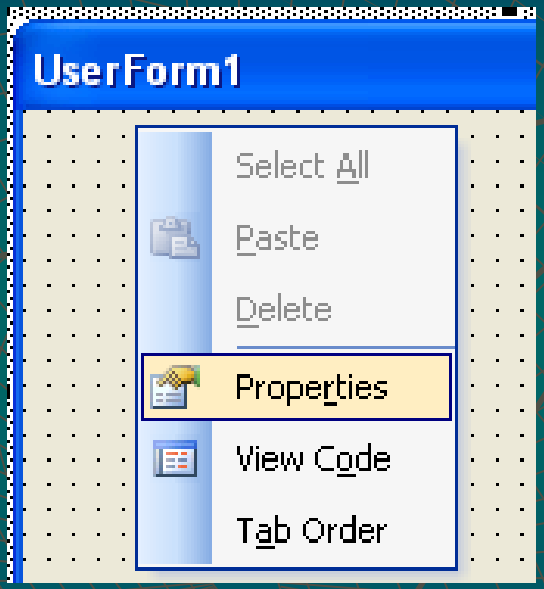




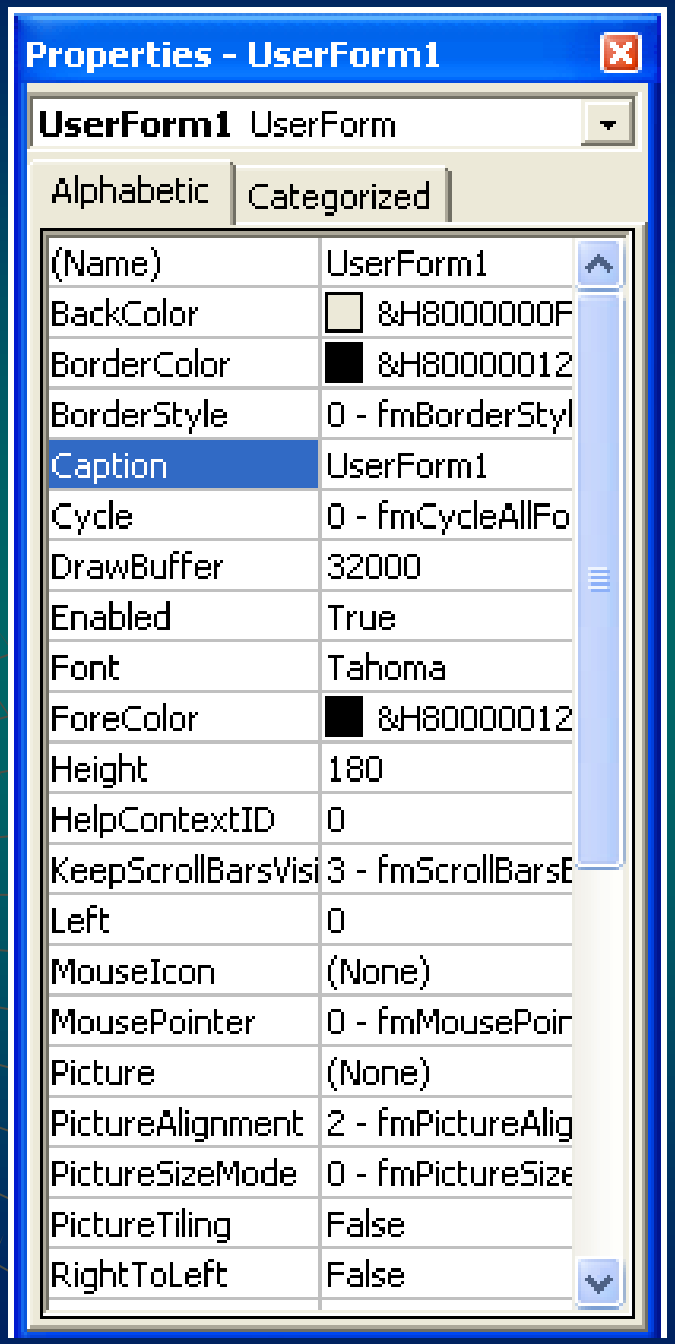
В результате откроется окно дизайнера форм (Form designer), в котором будет представлено пустое серое окно формы (UserForm1) и панель с набором элементов управления **Toolbox**

8.5.2. Свойства формы

Показ окна свойств формы включается выбором в контекстном меню команды Properties или клавишей <F4>.



Для формы и элементов управления многие свойства можно настраивать при помощи графического интерфейса окна свойств. При этом резко уменьшается программный код, записанный вручную.



Наиболее употребляемые свойства форм

Свойство	Описание
Name	Определяет имя формы
Caption	Определяет заголовок формы
Enabled	Возможность пользователя работать с формой
ShowModal	Установление модальности
Left	Отступ формы от верхнего края экрана (пикс.)
Top	Отступ формы от левого края экрана (пикс.)
Width	Размер формы по горизонтали (пикс.)
Height	Размер формы по вертикали (пикс.)
Font	Шрифт, начертание, размер, видоизменение
ForeColor	Цвет шрифта

Пример. Изменение заголовка формы «Форма_1» в программе может выглядеть так:

```
Форма_1.Caption = "Расчет параметров"
```

8.5.3. Методы формы

Методы формы используются для совершения каких-либо действий над формами.

Наиболее употребляемые методы форм

Метод	Описание
Show	Запуск формы из программного кода
Hide	Делает форму невидимой (прячет форму)
Unload	Выгрузка (удаление) формы из памяти

Остальные методы относятся либо к обмену данными через буфер обмена (Copy, Cut, Paste, …), либо к служебным возможностям формы (PrintForm, Repaint, Scroll, …).

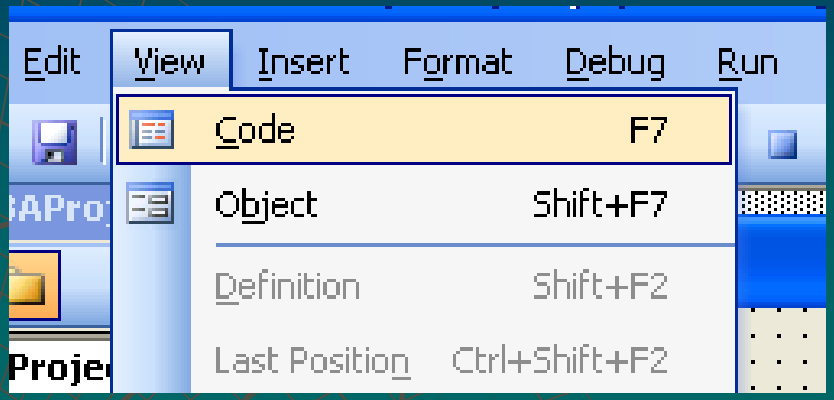
Пример. Вывод формы «Форма_1» на экран в программном приложении

```
Форма_1.Show
```

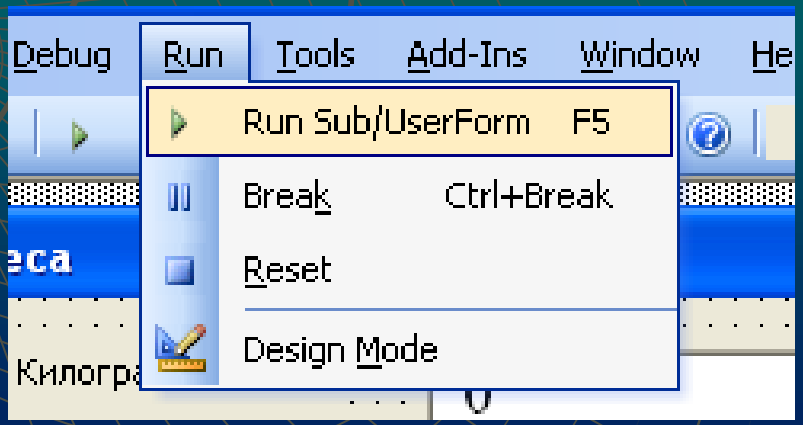
Если форма больше не потребуется, то ее можно удалить из памяти при помощи команды Unload :

```
Unload Форма_1
```

Для перехода на редактор кода для выбранной формы осуществляется выбором View -> Code или клавишей <F7>, возврат обратно в окно дизайнера форм - View -> Object или нажатием <Shift>+<F7>.



В процессе редактирования формы из окна редактора Visual Basic для запуска формы можно выбрать Run -> Sub/UserForm ил нажатием клавиши <F5>.



8.5.4. События

Важнейшая концепция VBA - обработка событий.

События (*event*) – это любые отслеживаемые изменения, происходящие с компонентами программы.

К событиям относятся щелчки мышью, нажатия на клавиши, открытие и закрытие форм, перемещение формы по экрану, изменение значений полей и т.п.

VBA построен таким образом, чтобы создавать на нем программы, управляемые событиями (*event-driven*). Такие методы построения программ противопоставляются процедурному программированию.

Поскольку форма – это контейнер для хранения элементов управления, главное ее событие – **Initialize** (происходит при подготовке формы к открытию). Все остальные события обычно используются не для формы, а для расположенных на ней элементов управления.

Наиболее употребляемые события форм

Свойство	Описание
Initialize	происходит при подготовке формы к открытию
Click	реакция на одиночный щелчок мыши
DblClick	реакция на двойной щелчок мыши
Error	используется при возникновении ошибки в форме
Terminate	используется при нормальном завершении работы формы

- Остальные события формы связаны с изменением размера окон, нажатиями клавиш, активизацией (получением фокуса) или деактивизацией (потерей фокуса) формы.

Если все элементы управления не помещаются на одной форме, то можно воспользоваться двумя формами (переходя между ними при помощи методов **Show** и **Hide** или воспользоваться несколькими вкладками для формы. Для этой цели используется специальный элемент управления **Multipage**.

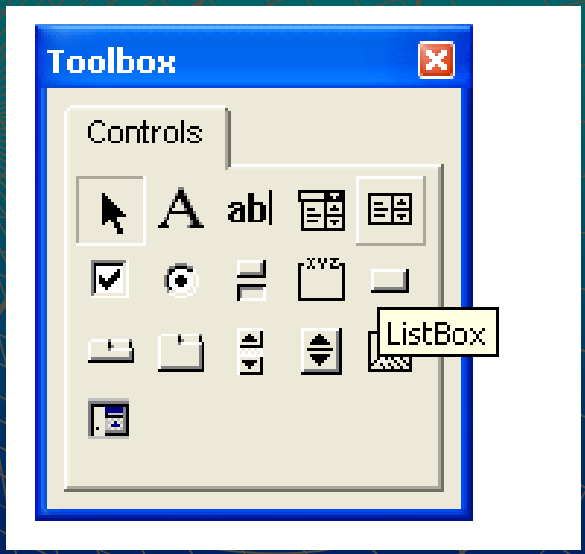
8.6. Элементы управления

Элементы управления – это специальные объекты, которые можно размещать на формах **VBA**, используемые для организации взаимодействия с пользователем.

Элементы управления реагируют на события, которые генерирует пользователь или программа (нажатие на кнопку, ввод, изменение значения, перемещение ползунка и т.п.)

8.6.1. Добавление элементов управления

Добавление элементов управления на форму чаще всего производится из дизайнера форм при помощи **Toolbox**.



8.6.2. Элемент управления *Label* (надпись)

Надпись – это область формы для помещения текста.

Работа с надписью сводится к получению и изменению ее свойств и иногда использования ее методов (Click, Error …).

Наиболее употребляемые свойства *Label*

Свойство	Описание
Name	Определяет имя надписи
Caption	Определяет текст надписи
Visible	Установление видимости
Left	Отступ от верхнего края формы (пикс.)
Top	Отступ от левого края формы (пикс.)
Width	Размер по горизонтали (пикс.)
Height	Размер по вертикали (пикс.)
Font	Шрифт, начертание, размер, видоизменение
ForeColor	Цвет шрифта

8.6.3. Элемент управления *TextBox* (текстовое поле)

Текстовое поле используется:

– для получения текстовых данных, вводимых пользователем;

– для вывода пользователю текстовых данных с возможностью их редактирования или без изменения.

Наиболее употребляемые свойства *TextBox*

Свойство	Описание
Name	Определяет имя надписи
Value / Text	Определяет содержание текстового поля
Visible	Установление видимости
ControlTipText	Текст всплывающей подсказки
Enabled	Отключение текстового поля для операций
Locked	Запрет на изменение данных
MaxLength	Максимальная длина вводимого значения
MultiLine	Возможность использования нескольких строк

Наиболее употребляемые свойства *TextBox (продолжение)*

Свойство	Описание
ScrollBars	Показ горизонтальных и вертикальных прокруток
WordWrap	автоматический переход на новую строку
Left	Отступ от верхнего края формы (пикс.)
Top	Отступ от левого края формы (пикс.)
Width	Размер по горизонтали (пикс.)
Height	Размер по вертикали (пикс.)
Font	Шрифт, начертание, размер, видоизменение
ForeColor	Цвет шрифта

Главное событие для текстового поля — это событие ***Change*** (изменение содержания поля). Обычно на это событие привязывается получение вводимых пользователем значений или синхронизация введенного значения с другими элементами управления.

8.6.4. Элемент управления *CommandButton* (кнопка)

CommandButton (кнопка) – самый распространенный элемент управления в формах. Кнопка используется для вызова процедуры.

Наиболее употребляемые свойства *CommandButton*

Свойство	Описание
Name	Определяет имя надписи
Caption	Определяет надпись на кнопке
Visible	Установление видимости
ControlTipText	Текст всплывающей подсказки
Enabled	Отключение кнопки
Default	Кнопка реагирует на нажатие клавиши <Enter>
Cancel	Кнопка реагирует на нажатие клавиши <Esc>
Picture	Назначение кнопке рисунка

Главное событие для кнопки – ***Click***. Как правило, к этому событию привязывается тот программный код, для которого создавалась кнопка.

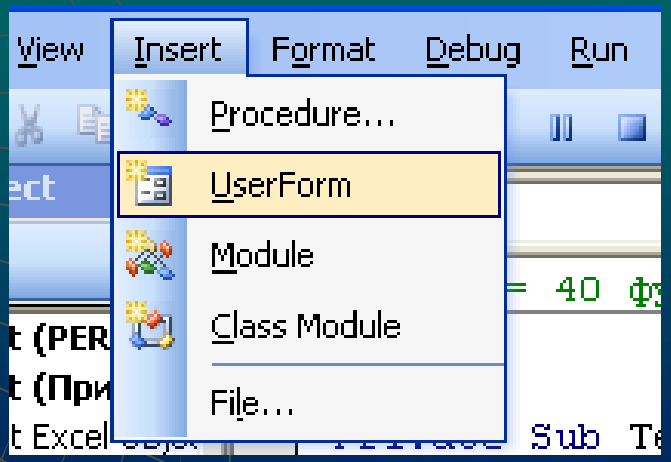
Пример. Решить квадратное уравнение $ax^2 + bx + c = 0$.

Разработка программы сводится к следующим действиям:

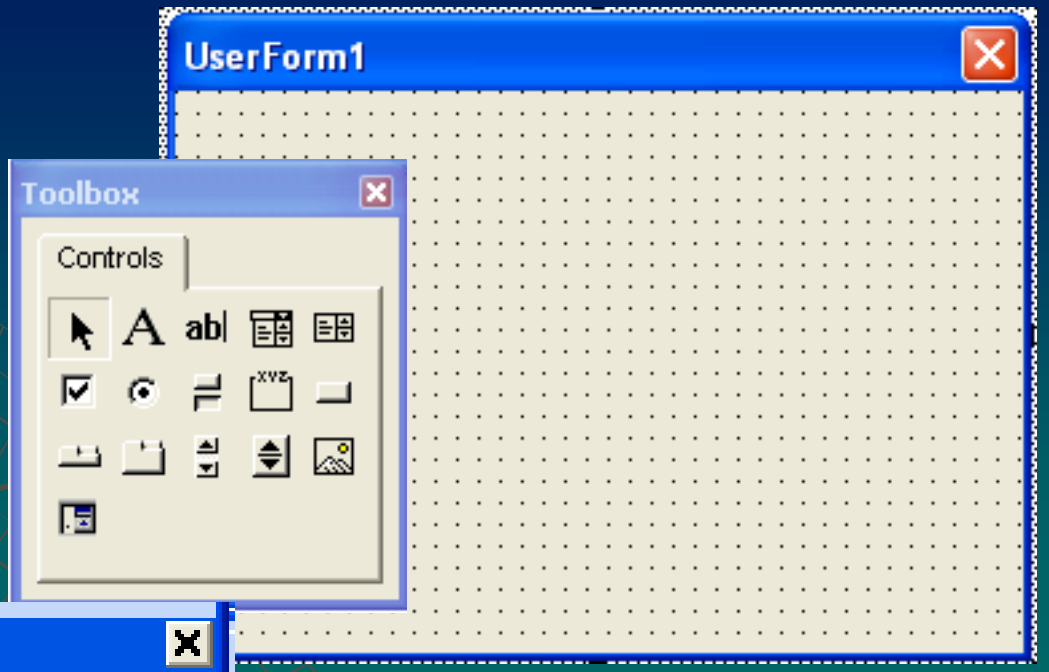
1) Создание новой формы;
2) Помещение на форму необходимых элементов управления;
3) Написание программного кода для обработки события;
4) Создание макроса для вызова формы.

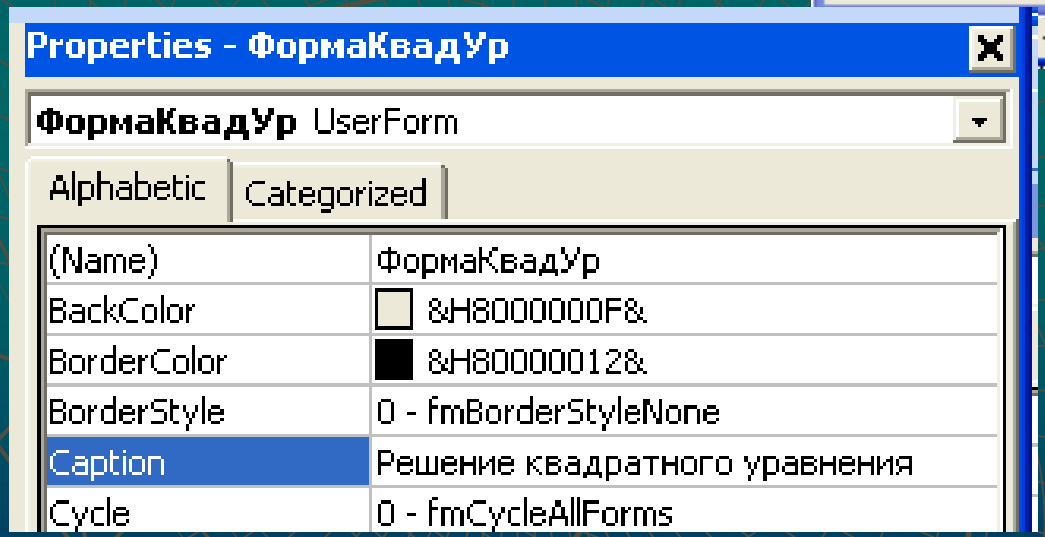
1-е действие. Создание новой формы.

Для создания новой формы выберем **Insert -> User Form**.



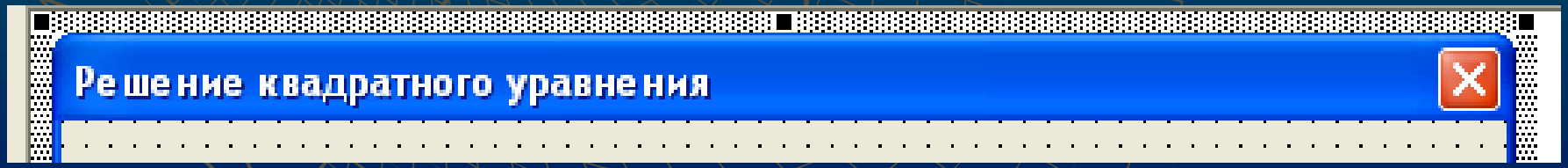
В результате откроется окно дизайнера форм (Form designer), в котором будет представлено пустое серое окно формы (UserForm1) и панель с набором элементов управления **Toolbox**





Изменим свойства Name и Caption созданной формы, заполнив их поля значениями «ФормаКвадУр» и «Решение квадратного уравнения».

Изменим размеры окна.



2 – действие. Помещение на форму необходимых элементов управления.

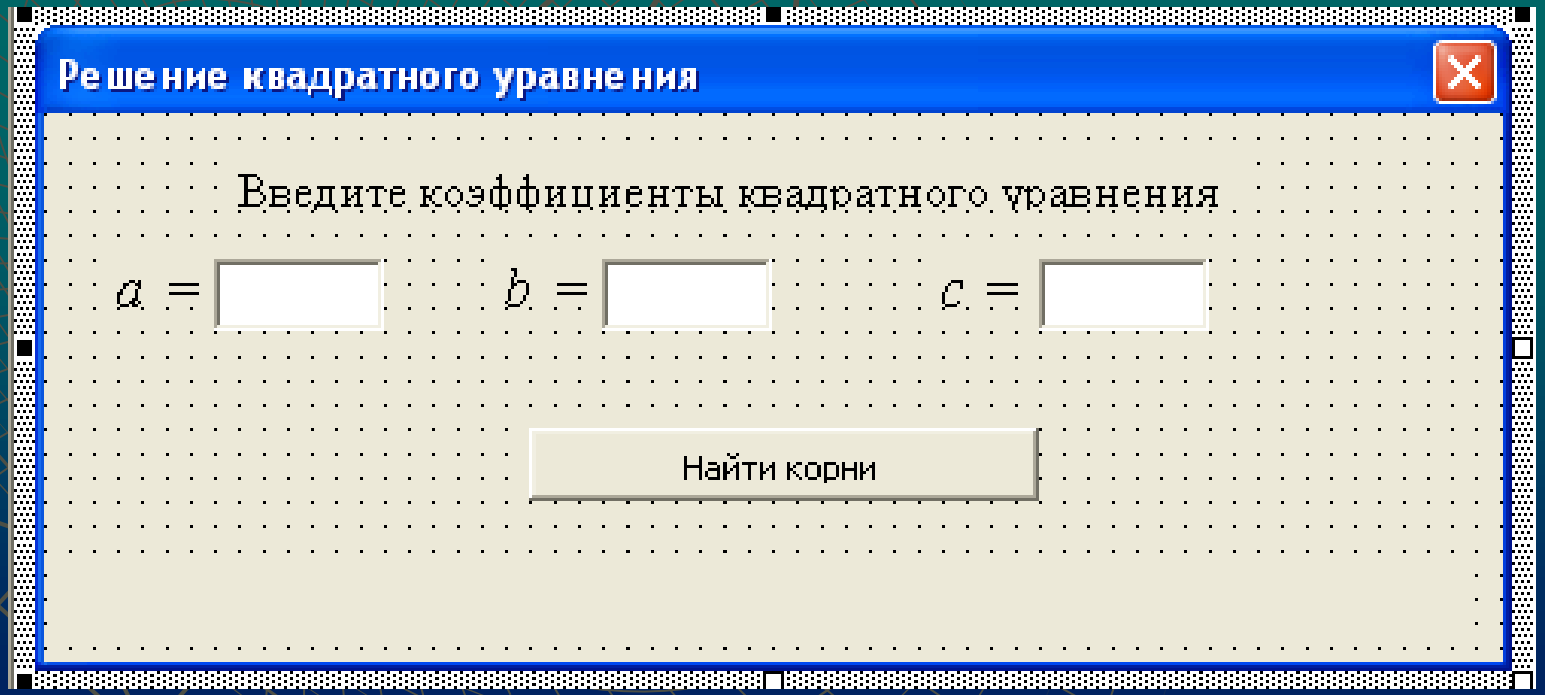
Для создания удобного пользовательского графического интерфейса разместим на форме с помощью Toolbox надписи:

Label1: «Введите коэффициенты квадратного уравнения»,

Label2: «*a* = », Label3: «*b* = », Label4: «*c* = »

и пустую надпись Label5: « » для помещения в нее результата.

Затем разместим кнопку CommandButton1: «Найти корни» для связывания ее с выполнением расчетов и текстовые поля TextBox1, TextBox2, TextBox3, не содержащие начальных значений для заполнения их пользователем.



3-е действие. Написание программного кода для обработки события.

Двойным щелчком по изображению кнопки CommandButton1: «Найти корни» создадим пустую процедуру, выполняющуюся при нажатии на кнопку CommandButton1_Click.

```
Private Sub CommandButton1_Click()

End Sub
```

Присвоим переменным a, b и c значения текстовых полей TextBox1, TextBox2, TextBox3 соответственно.

```
    a = Val(TextBox1.Value)
    b = Val(TextBox2.Value)
    c = Val(TextBox3.Value)
```

Затем анализируем переменную a.

Если a <> 0, тогда уравнение является квадратным и в зависимости от значения дискриминанта получим один из трех вариантов: «два корня», «один корень» и «действительных корней нет».

В противном случае (a = 0), уравнение не будет квадратным.

Полученные решения поместим на форму с помощью свойства Caption надписи Label5.

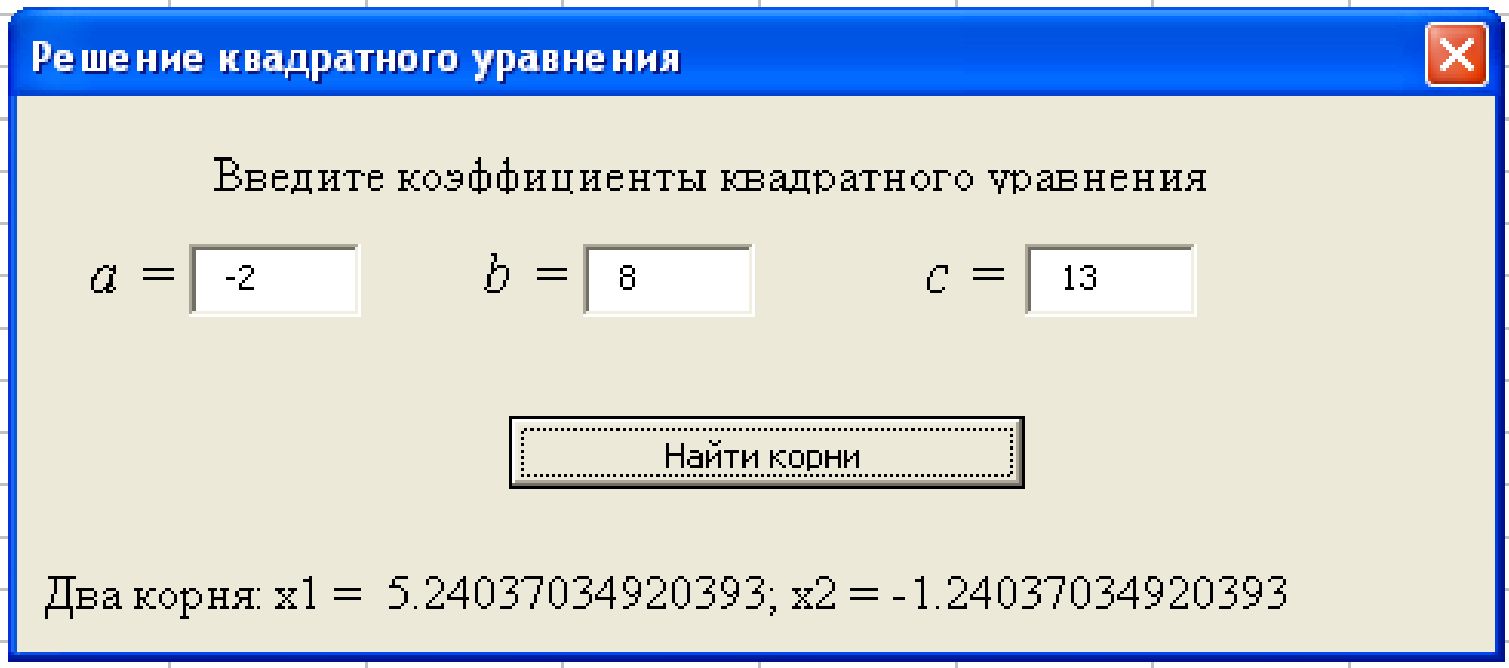
```
If a <> 0 Then
    d = b ^ 2 - 4 * a * c
    Select Case d
      Case Is > 0
        x1 = (-b - Sqr(d)) / (2 * a)
        x2 = (-b + Sqr(d)) / (2 * a)
        Label5.Caption = "Два корня: x1 = " + Str(x1) + "; x2 = " + Str(x2)
      Case 0
        x1 = (-b - Sqr(d)) / (2 * a)
        Label5.Caption = "Один корень: x = " + Str(x1)
      Case Is < 0
        Label5.Caption = "Действительных корней нет"
    End Select
  Else
      Label5.Caption = "Уравнение не является квадратным"""
End If
```

4-е действие. Создание макроса для вызова формы.

Для вызова формы напишем следующий программный код.

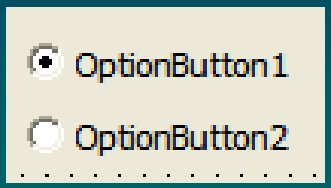
```
Sub рвбота_с_формой()
   ФормаКвадУр.Show
End Sub
```

После запуска макроса «работа_с_формой» и введения значений в поля TextBox1, TextBox2, TextBox3 программа выдаст следующий результат:



8.6.5. Элемент *OptionButton* (кнопка-переключатель)

Кнопки-переключатели ***OptionButton*** используются для выбора взаимоисключающих вариантов. Элемент выглядит как кнопка, которая при нажатии становится активной, а все другие кнопки заданной группы отключаются. ***OptionButton*** воспринимается пользователем как установление режима работы программы.



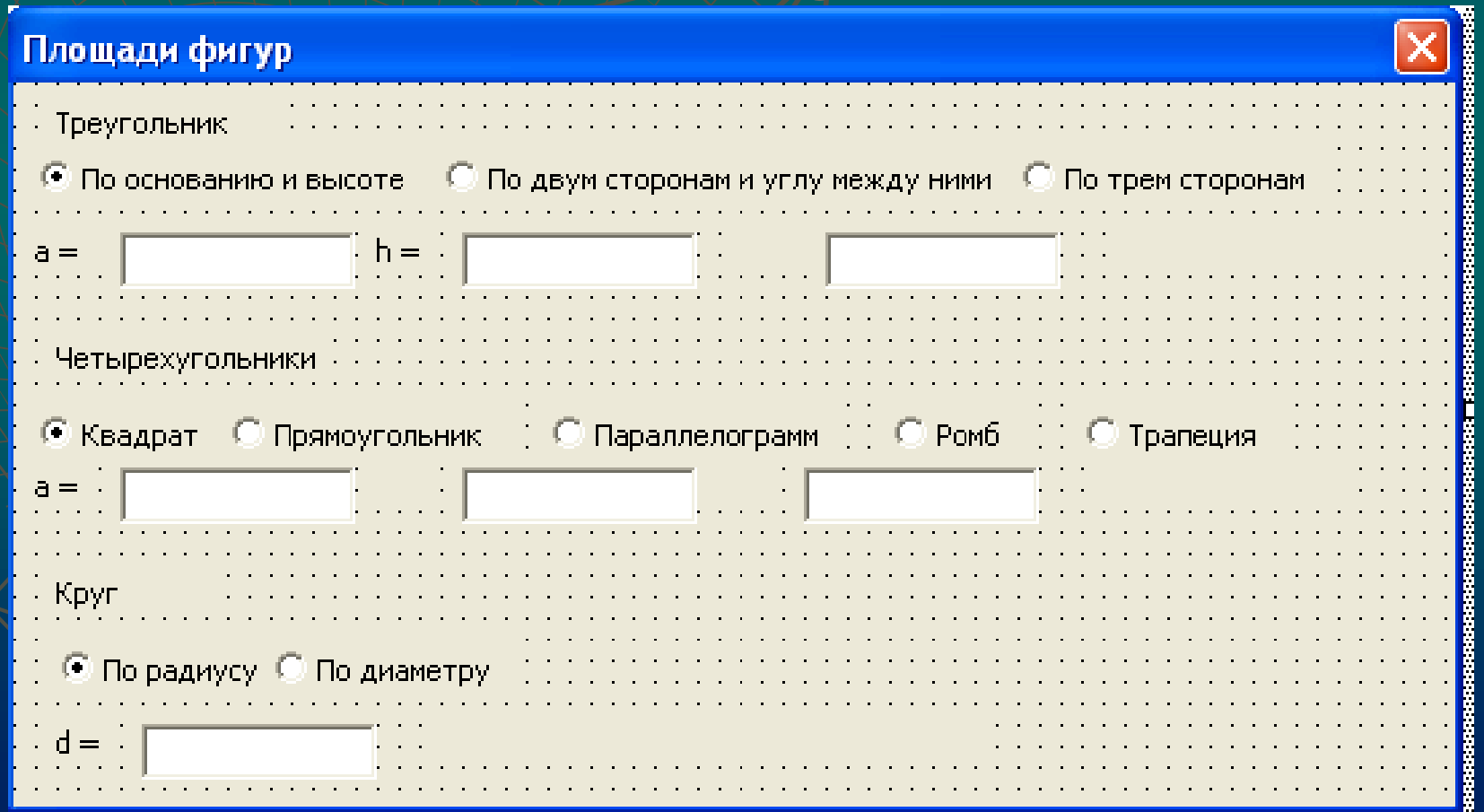
Наиболее употребляемые свойства *OptionButton*

Свойство	Описание
Name	Имя кнопки
Value	Значение (True – вкл., **False** – выкл.)
Caption	Надпись кнопки
TripleState	Режим использования состояния **Null** - серый
GroupName	Имя группы в которую входит кнопка
ControlTipText	Текст всплывающей подсказки
Enabled	Отключение действия кнопки
Locked	Запрет на переключение

Главное событие для кнопки-переключателя – ***Change***.
К переключению кнопки привязывается программный код.

Пример. Разработать форму, в которой пользователь может рассчитать площади различных геометрических фигур: треугольников, четырехугольников, круга.

Создадим новую форму, структурно состоящую из трех частей:

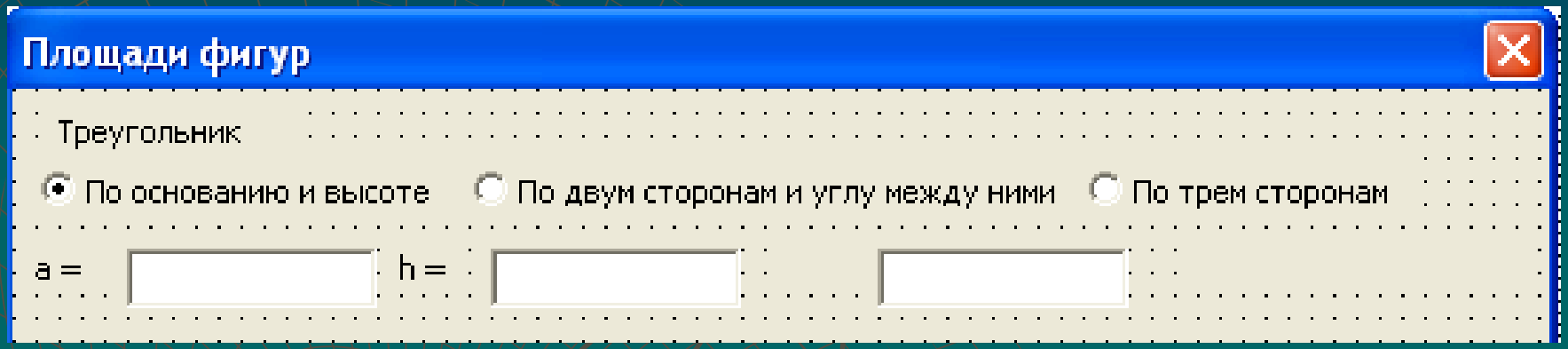


Соответственно, и работа программиста будет выполняться поэтапно:

1) «Треугольник»; 2) «Четырехугольники»; 3) «Круг».

Выполним 1-ю часть формы «Треугольник».

Для решения задачи создадим новую форму и назовем ее SForm. Заголовку формы назначим значение «Площади фигур».



На форме поместим:

- пять надписей LabelTr0, LabelTr1, LabelTr2 с заголовками «Треугольник», «a = », «h =» и пустые LabelTr3, LabelTr4;
- три пустых текстовых поля TextBoxTr1, TextBoxTr2, TextBoxTr3;
- три кнопки-перключателя ButtonTr1, ButtonTr2, ButtonTr3, с заголовками «По основанию и высоте», «По двум сторонам и углу между ними», «По трем сторонам».

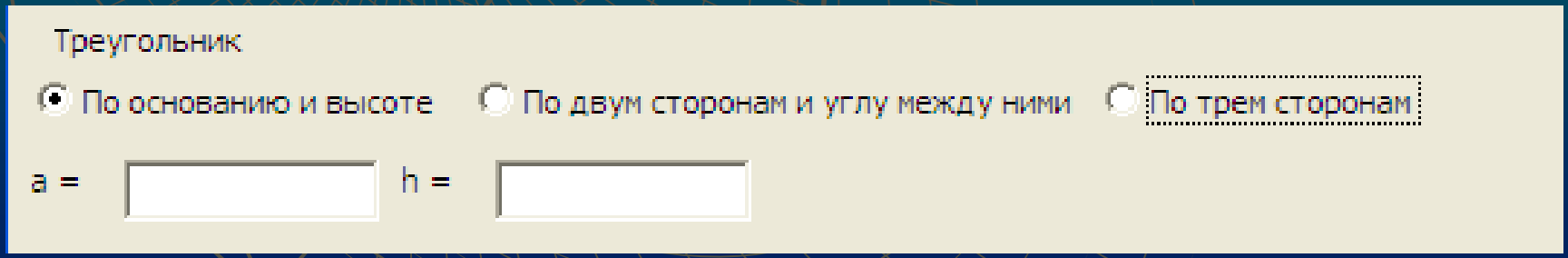
Все кнопки включим в группу «Треугольник» (свойство *GroupName*). Активной назначим первую кнопку.

Для установленных кнопок создадим процедуры, которые будут выполняться при событии нажатия на выбранную кнопку.

Для первой кнопки ButtonTr1 процедура будет следующей:

```
Private Sub ButtonTr1_Change()
  With SForm
     .LabelTr1.Caption = "a ="
     .LabelTr2.Caption = "h ="
     .LabelTr3.Caption = ""
     .TextBoxTr3.Visible = False
     .TextBoxTr1.Value = ""
     .TextBoxTr2.Value = ""
  End With
End Sub
```

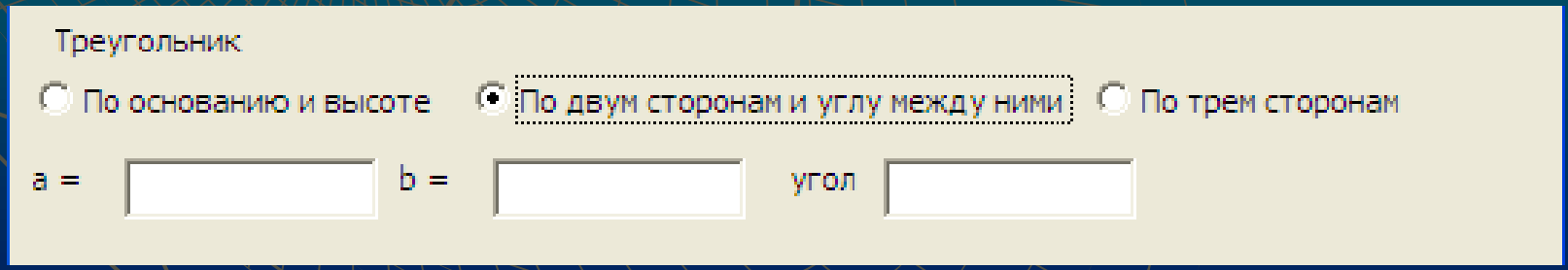
Действие процедуры заключается в установлении значений надписей LabelTr1, LabelTr2 и LabelTr3, скрытия поля TextBoxTr3, а также очистке текстовых полей TextBoxTr1, TextBoxTr2.



Процедура для второй кнопки ButtonTr2 аналогична:

```
Private Sub ButtonTr2_Change()
  With SForm
    .LabelTr1.Caption = "a ="
    .LabelTr2.Caption = "b ="
    .LabelTr3.Caption = "угол"
    .TextBoxTr3.Visible = True
    .TextBoxTr1.Value = ""
    .TextBoxTr2.Value = ""
    .TextBoxTr3.Value = ""
  End With
End Sub
```

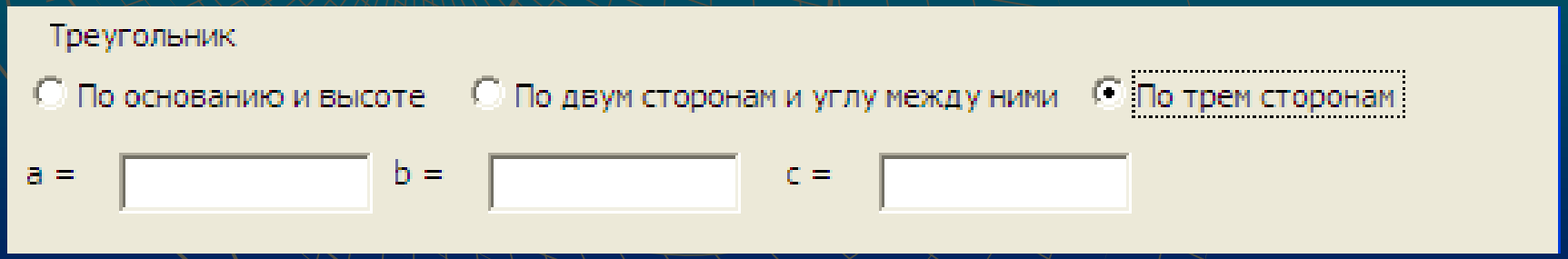
Действие процедуры заключается в установлении значений надписей LabelTr1, LabelTr2 и LabelTr3, установления видимости поля TextBoxTr3, а также очистке текстовых полей TextBoxTr1, TextBoxTr2 и TextBoxTr3.



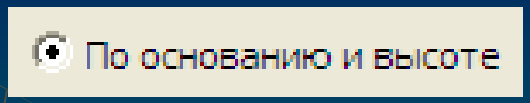
Процедуру для третьей кнопки ButtonTr3 запишем следующим образом:

```
Private Sub ButtonTr3_Change()
  With SForm
     .LabelTr1.Caption = "a ="
     .LabelTr2.Caption = "b ="
     .LabelTr3.Caption = "c ="
     .TextBoxTr3.Visible = True
     .TextBoxTr1.Value = ""
     .TextBoxTr2.Value = ""
     .TextBoxTr3.Value = ""
  End With
End Sub
```

Действие процедуры заключается в установлении значений надписей LabelTr1, LabelTr2 и LabelTr3, установления видимости поля TextBoxTr3, а также очистке текстовых полей TextBoxTr1, TextBoxTr2 и TextBoxTr3.

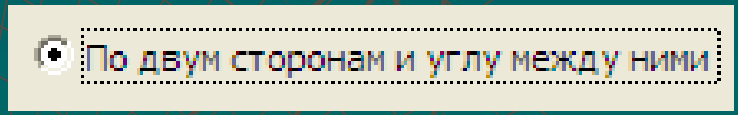


Условимся, что расчеты площадей будут выполняться при любом изменении в текстовых полях в зависимости от значений кнопок группы «Треугольник».



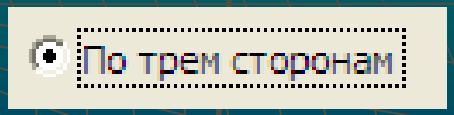
1) Если активна кнопка ButtonTr1, площадь треугольника находится как половина произведения основания треугольника на высоту:

$$S = \frac{a \cdot h}{2}.$$



2) При нажатой кнопке ButtonTr2 площадь определяется по формуле:

$$S = \frac{a \cdot b \cdot \sin \alpha}{2}.$$



3) В случае известных трех сторон треугольника (активна ButtonTr3) площадь вычисляется по формуле Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)}, \qquad p = \frac{a+b+c}{2}.$$

Для использования данных формул необходимо написать процедуры для каждого из текстовых полей TextBoxTr1, TextBoxTr2 и TextBoxTr3.

Рассмотрим процедуру, привязанная к изменению поля TextBoxTr1.

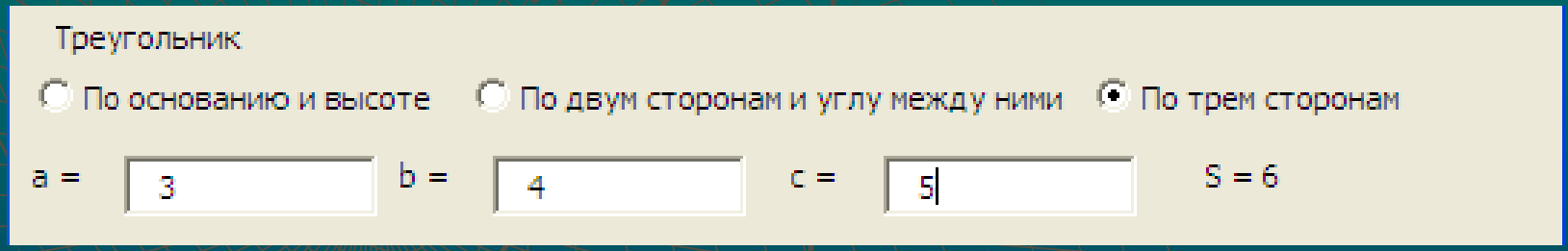
```
Private Sub TextBoxTr1_Change()
  With SForm
    a = Val(.TextBoxTr1.Value)
    b = Val(.TextBoxTr2.Value): h = b
    c = Val(.TextBoxTr3.Value)
    alfa = c * 3.14 / 180
    p = (a + b + c) / 2
    If .ButtonTr1.Value = True Then S = a * h / 2
    If .ButtonTr2.Value = True Then S = a * b * Sin(alfa) / 2
    If .ButtonTr3.Value = True Then
      S = p * (p - a) * (p - b) * (p - c)
      If S > 0 Then S = Sqr(S)
    End If
    If S > 0 Then S = "S = " & Round(S, 4) Else S = "S = ?"
    .LabelTr4.Caption = S
  End With
End Sub
```

В процедуре считываются значения всех трех полей (переменные a, b, c, h), рассчитываются параметры alpha, p, в зависимости от активности кнопок по формулам определяется и выводится площадь S.

Процедуры TextBoxTr2_Change и TextBoxTr3_Change, привязанные к изменениям полей TextBoxTr2 и TextBoxTr3 вызывают уже написанную процедуру TextBoxTr1_Change.

```
Private Sub TextBoxTr2_Change()
  TextBoxTr1_Change
End Sub

Private Sub TextBoxTr3_Change()
  TextBoxTr1_Change
End Sub
```

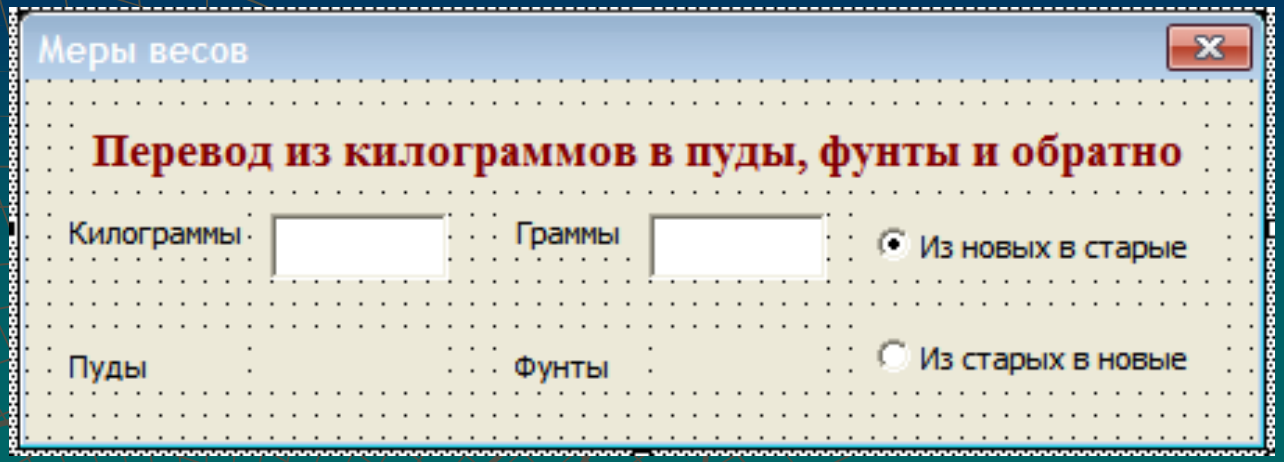
Результат работы с формой, при использовании формулы Герона:



Части формы «Четырехугольники» и «Круг» выполняются аналогично.

Пример. Разработать форму для перевода из старых мер весов (пудов и фунтов) в новые (килограммы и граммы) и обратно.

Создадим новую форму:



На форме поместим:

- пять надписей и определим им значения свойства caption: Label1 «Перевод из килограммов в пуды, фунты и обратно», Label2 «Килограммы», Label3 «Граммы», Label4 «Пуды», Label5 «Фунты» ;
- две надписи Вес3, Вес4 с пустыми значениями свойства caption;
- два текстовых поля Вес3, Вес4;
- две кнопки-переключателя НовыеСтарые и СтарыеНовые с заголовками «Из новых в старые», «Из старых в новые». Активной назначим первую кнопку.

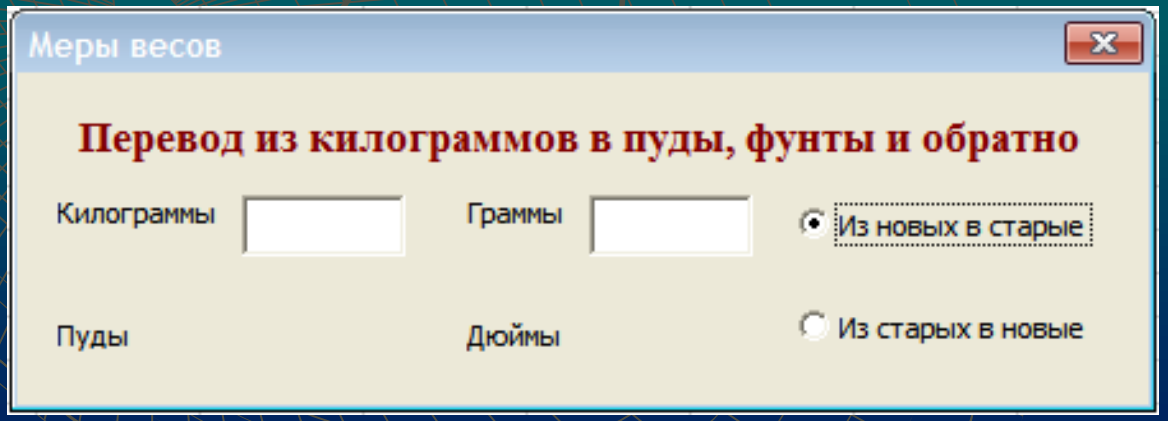
Для установленных кнопок создадим процедуры, которые будут выполняться при событии нажатия на выбранную кнопку.

Для первой кнопки НовыеСтарые процедура будет следующей:

```
Private Sub НовыеСтарые_Change()
  МЕРЫ.Label2.Caption = "Килограммы"
  МЕРЫ.Label3.Caption = "Граммы"
  МЕРЫ.Label4.Caption = "Пуды"
  МЕРЫ.Label5.Caption = "Дюймы"
  МЕРЫ.Bec1.Value = ""
  МЕРЫ.Bec2.Value = ""
End Sub
```

Действие процедуры заключается в установлении значений надписей Label2, Label3, Label4, Label5, и очистке текстовых полей Bec1 и Bec2.

В результате форма примет вид:

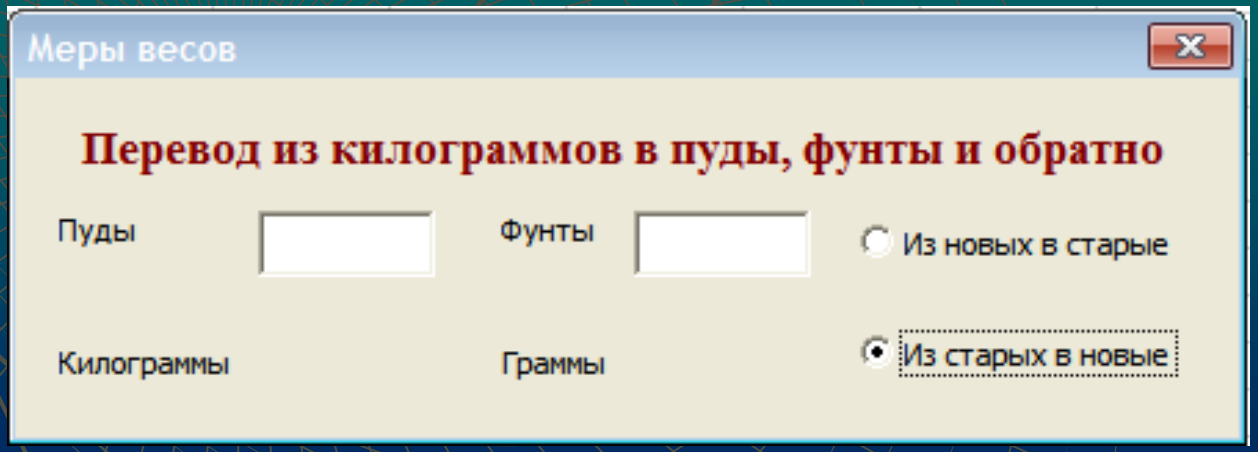


Для второй кнопки СтарыеНовые напишем аналогичную процедуру:

```
Private Sub СтарыеНовые_Change()
  МЕРЫ.Label2.Caption = "Пуды"
  МЕРЫ.Label3.Caption = "Фунты"
  МЕРЫ.Label4.Caption = "Килограммы"
  МЕРЫ.Label5.Caption = "Граммы"
  МЕРЫ.Вес1.Value = ""
  МЕРЫ.Вес2.Value = ""
End Sub
```

На форме изменяют значения те же надписи Label2, Label3, Label4, Label5, и снова очищаются текстовые поля Вес1 и Вес2.

В результате форма для пользователя примет вид:



Напишем процедуру Вес1_Change, которая будет запускаться при каждом изменении значения в поле Вес1:

```
Private Sub Вес1_Change()
Dim g, p, f, k
  If НовыеСтарые.Value = True Then
      g = Val(МЕРЫ.Вес1.Value) * 1000 + Val(МЕРЫ.Вес2.Value)
      p = Int(g / 1000 / 16.38)
      f = (g / 1000 / 16.38 - p) * 40
      МЕРЫ.Вес3.Caption = p
      МЕРЫ.Вес4.Caption = Round(f, 2)
    Else
      p = Val(МЕРЫ.Вес1.Value) + Val(МЕРЫ.Вес2.Value) / 40
      k = p * 16.38
      МЕРЫ.Вес3.Caption = Int(k)
      МЕРЫ.Вес4.Caption = Round((k - Int(k)) * 1000)
  End If
End Sub
```

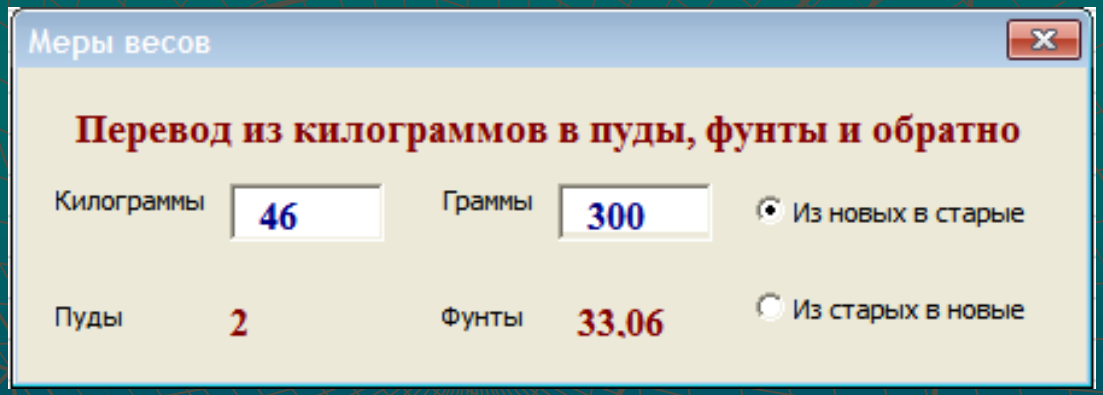
Если кнопка НовыеСтарые имеет значение True, тогда программа будет работать в режиме «из килограммов и граммов в пуды и фунты». Значения полей Вес1 и Вес2 используются для расчета веса в граммах g, переводим граммы в целое число пудов p, а остаток - в фунты f. Затем записываем полученные значения p и f в надписи Вес3 и Вес4.

В противном случае рассчитываем пуды p, переводим их в целое число килограммов k, Килограммы и остаток в граммах помещаем в надписи Вес3 и Вес4.

Процедура Bec2_Change, за пускающаяся при каждом изменении значения в поле Bec2 ничем не будет отличаться от процедуры Bec1_Change. Для того чтобы не дублировать полностью код можно просто вызвать процедуру Bec1_Change.

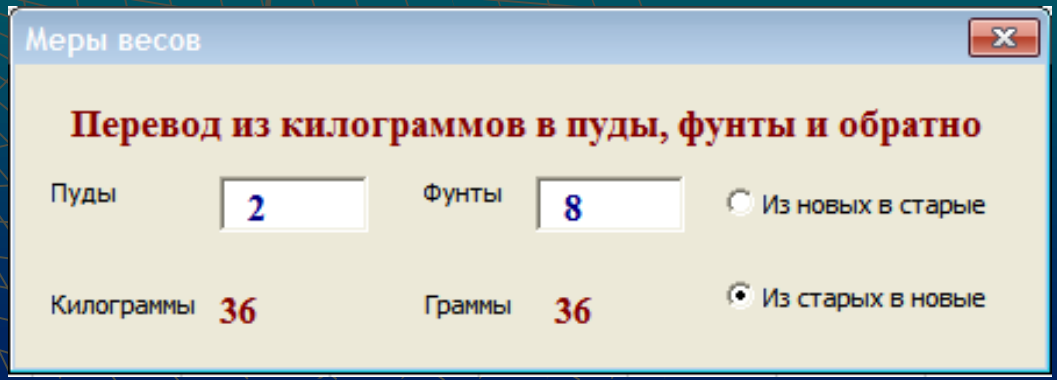
```
Private Sub Bec2_Change()
  Bec1_Change
End Sub
```

Результат выполнения процедур Bec1_Change и Bec2_Change будет выглядеть следующим образом:



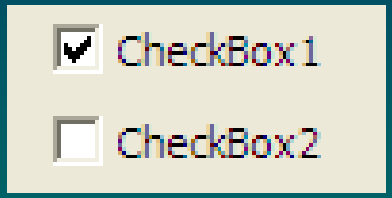
Режим
«Из новых в старые»

Режим
«Из старых в новые»



8.6.6. Элемент управления *CheckBox* (флажок)

Элемент управления ***CheckBox*** **(флажок)** используется для выбора невзаимоисключающих вариантов. Нажатие на флажок изменяет значение свойства *Value* данного флажка и не влияет на статус других флажков. Обычно ***CheckBox*** воспринимается пользователем как установление каких-либо опций работы программы.



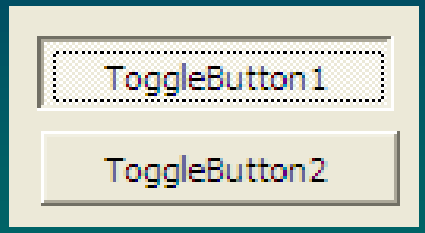
Наиболее употребляемые свойства *CheckBox*

Свойство	Описание
Name	Имя кнопки
Value	Значение (**True** – вкл., **False** – выкл.)
Caption	Надпись кнопки
TripleState	Режим использования состояния **Null** - серый
ControlTipText	Текст всплывающей подсказки
Enabled	Отключение действия кнопки
Locked	Запрет на переключение

Главное событие для кнопки-переключателя – ***Change***.

8.6.7. Элемент управления *ToggleButton* (кнопка с фиксацией)

Аналогично флажку ***CheckBox*** работает элемент управления ***ToggleButton***. Он также используется для выбора невзаимоисключающих вариантов. Нажатие на кнопку изменяет значение свойства *Value* данного флажка и не влияет на статус других кнопок.



Наиболее употребляемые свойства *ToggleButton*

Свойство	Описание
Name	Имя кнопки
Value	Значение (**True** – вкл., **False** – выкл.)
Caption	Надпись кнопки
TripleState	Режим использования состояния **Null** - серый
ControlTipText	Текст всплывающей подсказки
Enabled	Отключение действия кнопки
Locked	Запрет на переключение

Главное событие для кнопки-переключателя – ***Change***.

Пример. Составить программу вывода ряда распределения дискретной случайной величины ***X*** по биномиальному закону. Опциями могут быть вывод математического ожидания, дисперсии и среднего квадратического отклонения.

Решение.

Биномиальный закон распределения представляет собой закон распределения числа $X=m$ наступлений события A в n независимых испытаниях, в каждом из которых оно может произойти с одной и той же вероятностью p.

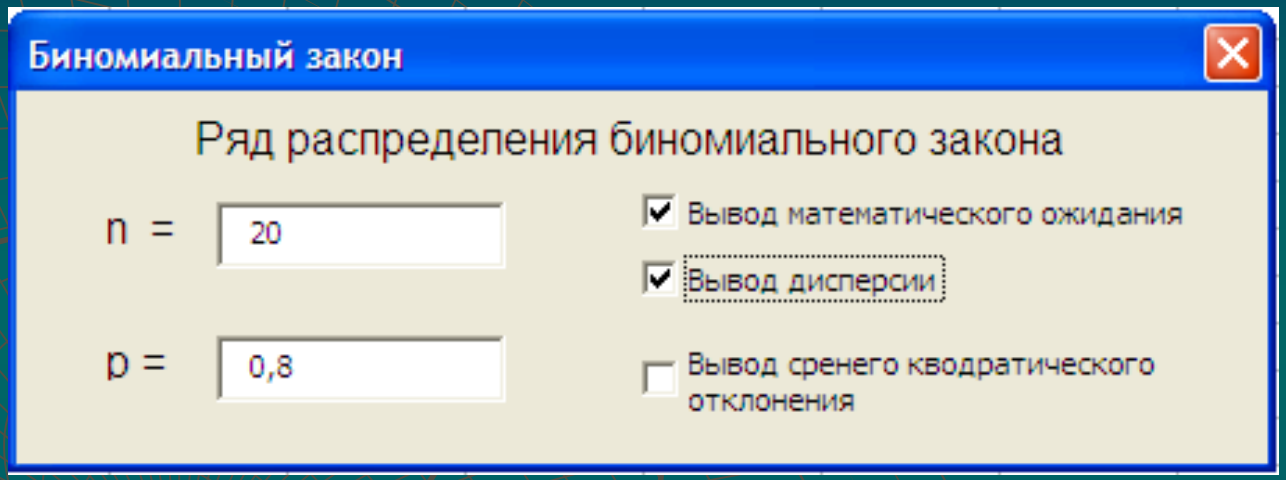
Ряд распределения биномиального закона имеет вид:

X_i	0	1	2	...	m	...	n
p_i	q^n	$C_n^1 p q^{n-1}$	$C_n^2 p^2 q^{n-2}$	...	$C_n^m p^m q^{n-m}$	...	p^n

Математическое ожидание, дисперсия и среднее квадратическое отклонение случайной величины X, определенной по биномиальному закону вычисляются по формулам:

$$M(X) = np \qquad D(X) = npq \qquad \sigma(X) = \sqrt{D(X)}$$

Для взаимодействия с пользователем будем использовать форму на которой поместим надписи и поля для внесения значений ***n*** и ***p***, а также флажки для активизации опций «Вывод математического ожидания», «Вывод дисперсии», «Вывод среднего квадратического отклонения».



Вывод таблицы и характеристик случайной величины X удобно производить на лист MS Excel.

Все расчеты на листе необходимо производить с учетом флажков при изменении любого из текстовых полей.

Для решения задачи напишем функцию *fact*, которая будет возвращать значения факториала натурального числа и использоваться в программе неоднократно.

```
Function fact(n) As Double
  fact = 1
  For i = 1 To n
    fact = fact * i
  Next
End Function
```

Затем, составим код функции *soch* с использованием функции *fact*.

```
Function soch(m, n)
  soch = fact(n) / (fact(n - m) * fact(m))
End Function
```

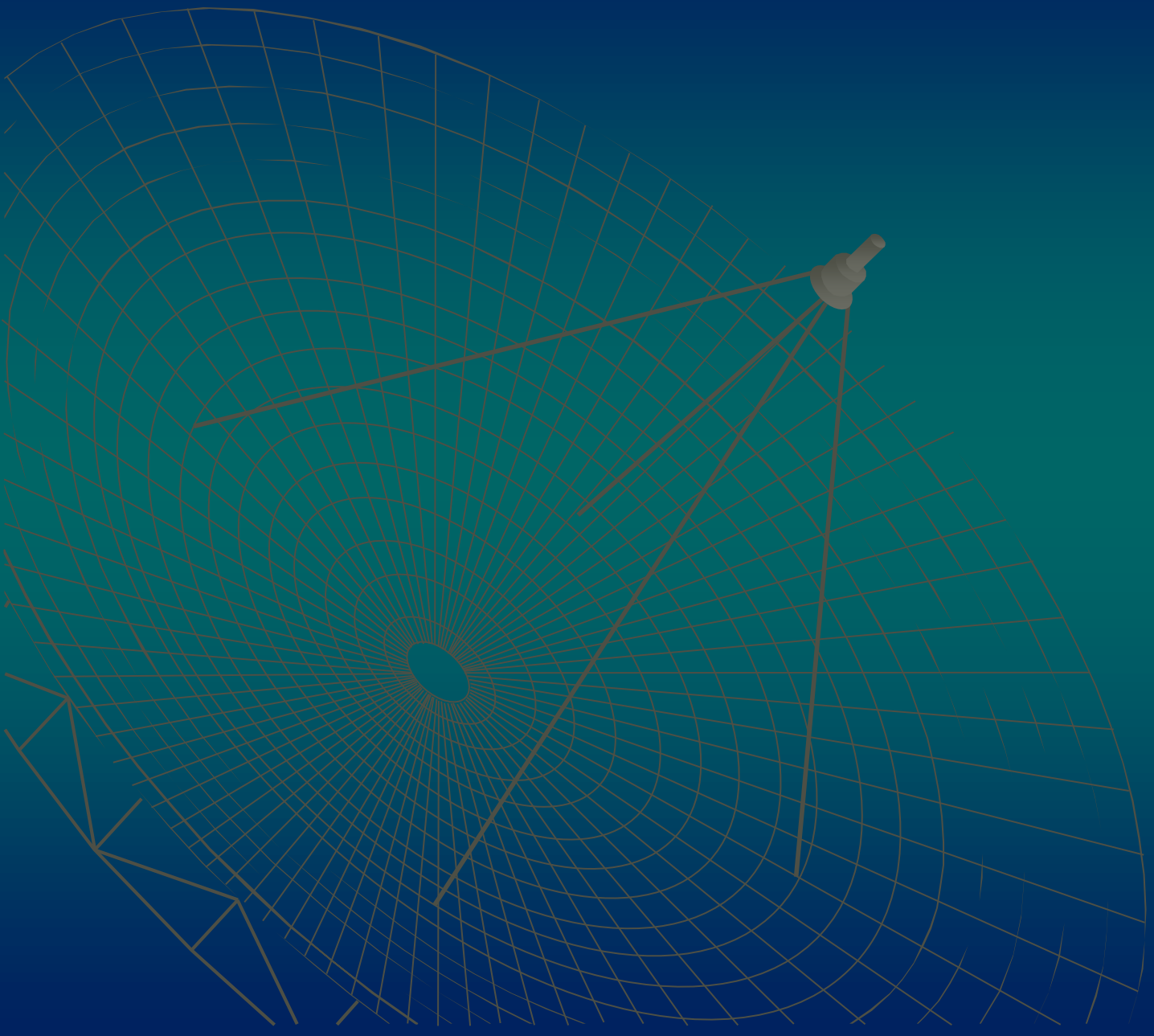
Функцию *soch* можно будет использовать для расчета значений вероятностей, определяемых по формуле Бернулли

$$C_n^m p^m q^{n-m}.$$

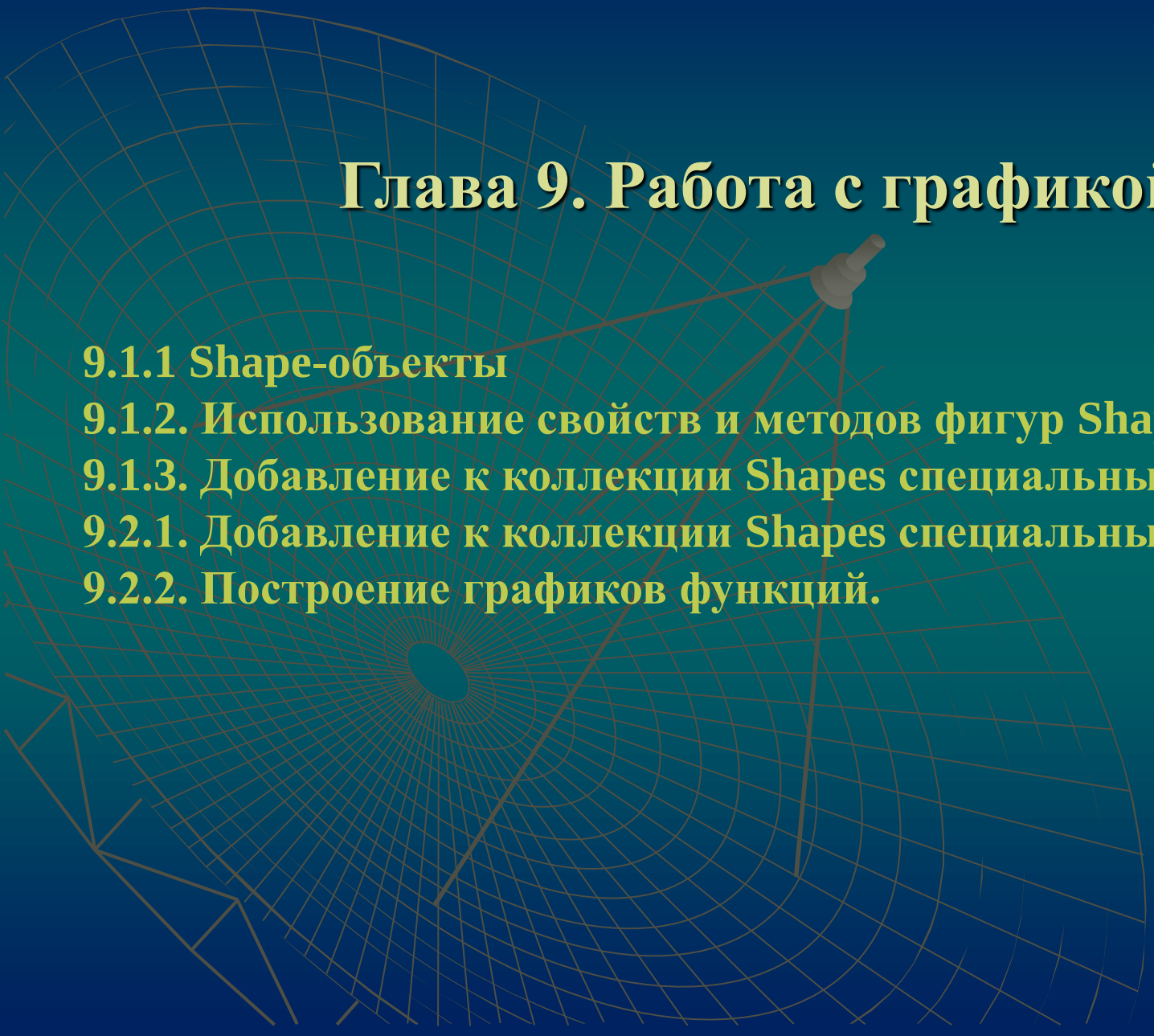
Задача

Реализовать работу с формами

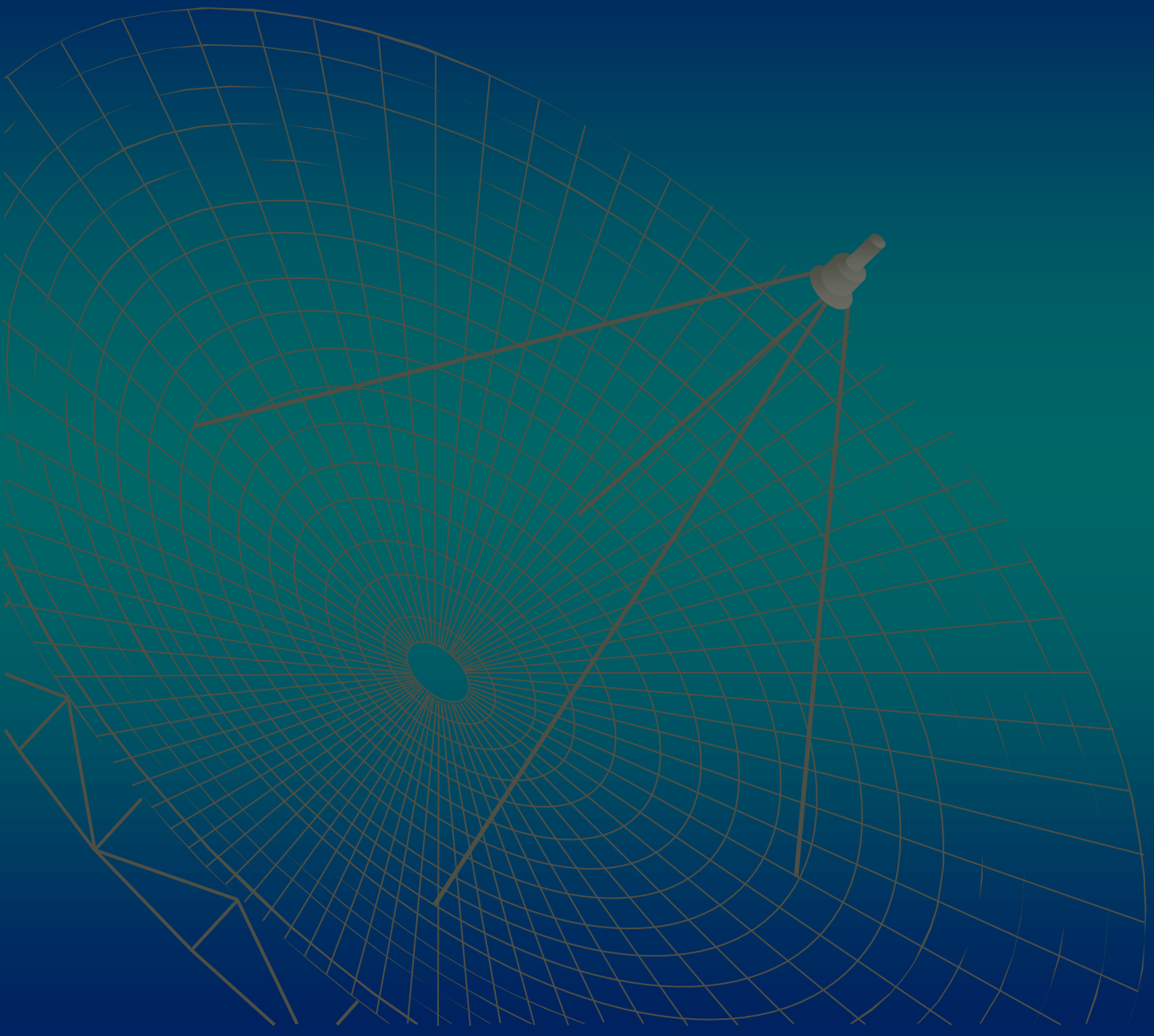
1. Решение квадратных уравнений $ax^2 + bx + c = 0$.
2. Расчет площадей геометрических фигур: треугольников, четырехугольников, круга.
3. Перевод из старых мер весов (пудов и фунтов) в новые (килограммы и граммы) и обратно.
4. Вывод ряда распределения дискретной случайной величины X по биномиальному закону.



Глава 9. Работа с графикой



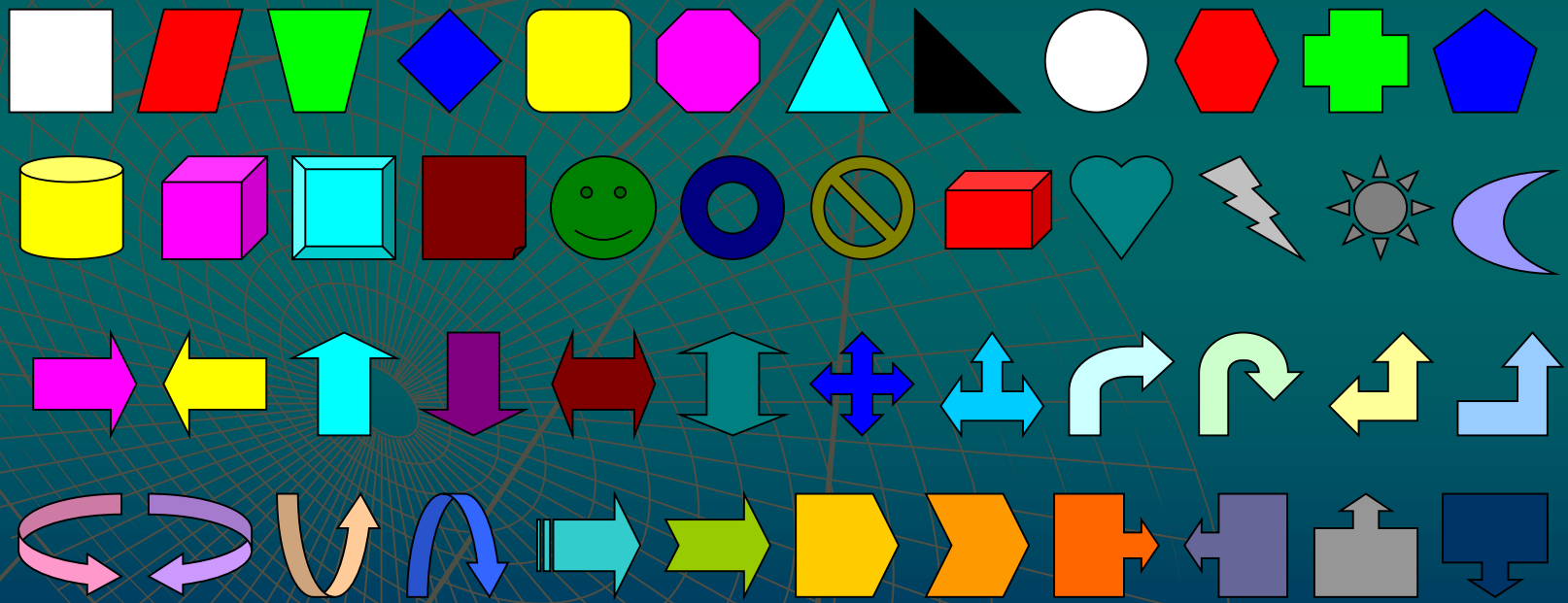
9.1.1 Shape-объекты
9.1.2. Использование свойств и методов фигур Shapes
9.1.3. Добавление к коллекции Shapes специальных фигур
9.2.1. Добавление к коллекции Shapes специальных фигур
9.2.2. Построение графиков функций.



9.1.1. Shape-объекты

VBA поддерживает общую модель объектов слоя векторной графики в Microsoft Word, Microsoft Excel и Microsoft PowerPoint.

Верхним уровнем этой модели является коллекция **Shapes,** содержащая все графические объекты (Shape-объекты), которые включаются в слой векторной графики.



Кроме коллекции **Shapes,** можно использовать коллекцию **ShapeRange.**

9.1.1.1. Добавление Shape-объекта

Для добавления в документ Shape-объекта используя ее метод **AddShapes**.

Синтаксис метода

expression.***AddShape***(*Type*, *Left*, *Top*, *Width*, *Height*)

expression - любое **Shapes**-выражение;

Type - константа (тип ***Long***), определяющая тип фигуры и принимающая одно из **msoAutoShapeType**-значений;

Left – (тип ***Single***) - горизонтальная координата левых углов фигуры;

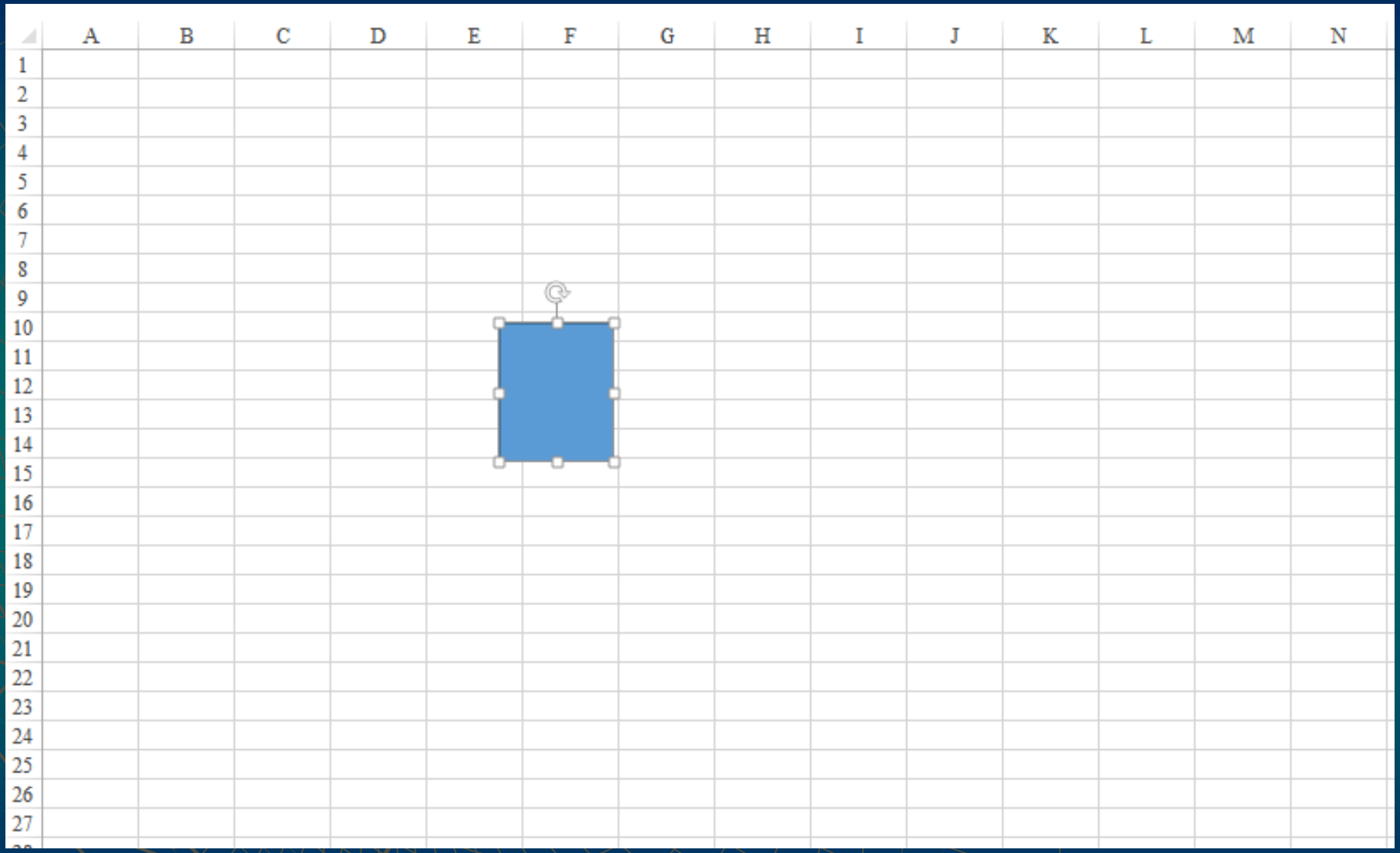
Top – (тип ***Single***) - вертикальная координата верхних углов фигуры;

Width - (тип ***Single***) - ширина фигуры;

Height - (тип ***Single***) - высота фигуры.

Пример. Добавление к коллекции активного листа **Shapes** прямоугольника.

```
Sub Добавленеие_прямоугольника()
    ActiveSheet.Shapes.AddShape(msoShapeRectangle, 200, 120, 50, 60).Select
End Sub
```



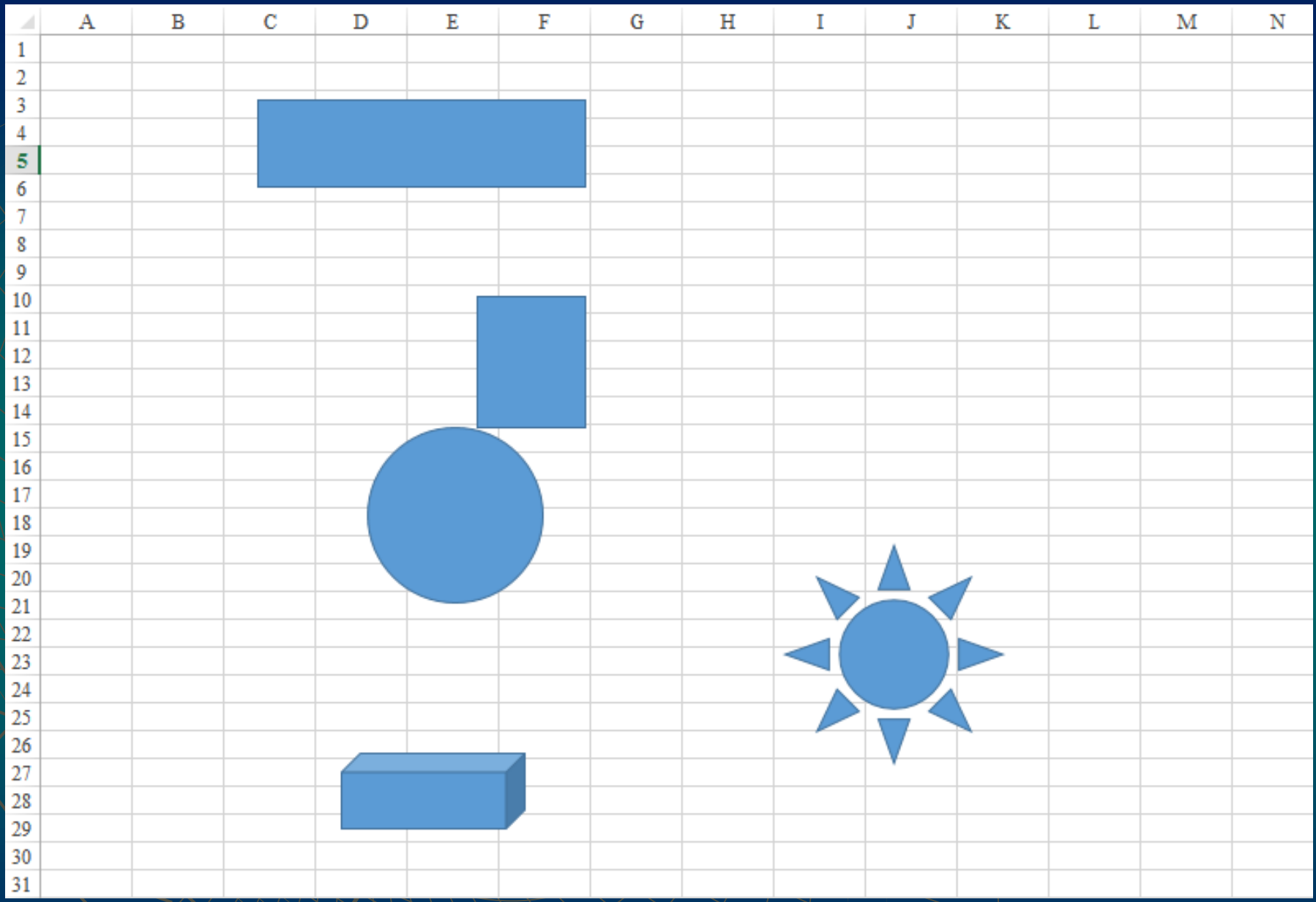
Добавление к коллекции активного листа **Shapes** прямоугольника.

При добавлении в коллекцию **Shapes** фигура автоматически получает имя (свойство **Name)** по умолчанию, но его можно указать, используя следующий синтаксис:
expression.AddShape(*Type*, *Left*, *Top*, *Width*, *Height*).Name=<имя фигуры>

Идентификацию фигур необходимо делать для дальнейшей работы с добавленными объектами.

Пример. Добавление к коллекции **Shapes** активного листа поименованных фигур.

```
Sub Добавление_поименованых_фигур()
  ActiveSheet.Shapes.AddShape(msoShapeRectangle, 100, 30, 150, 40) _
    .Name = "прямоугольник"
  ActiveSheet.Shapes.AddShape(msoShapeOval, 150, 180, 80, 80) _
    .Name = "Овал"
  ActiveSheet.Shapes.AddShape(msoShapeCube, 138, 329, 84, 35) _
    .Name = "Параллелепипед"
  ActiveSheet.Shapes.AddShape(msoShapeSun, 341, 234, 100, 100) _
    .Name = "Солнышко"
End Sub
```



Добавление поименованных фигур.

9.1.1.2. Использование индексов

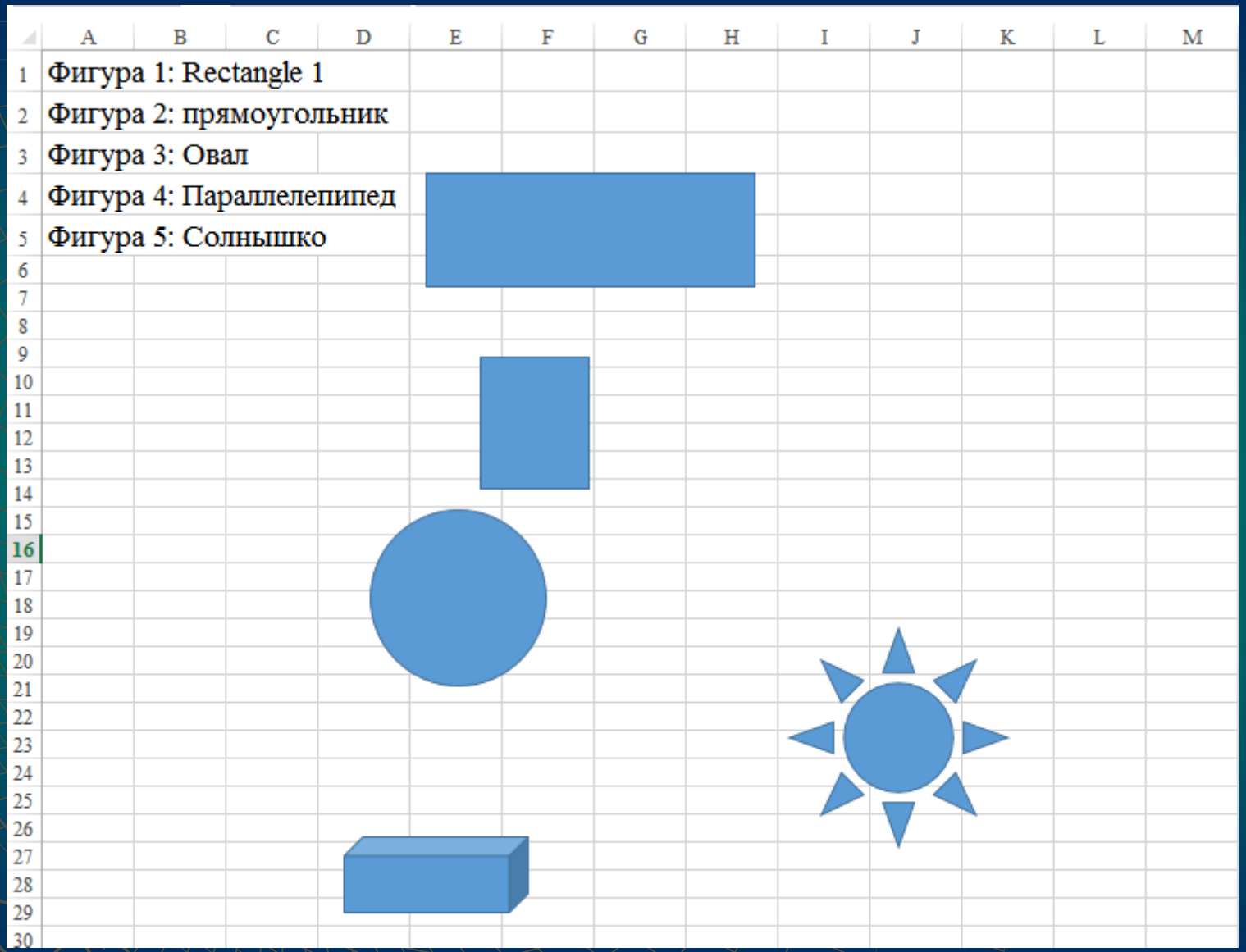
Для получения индекса фигуры или коллекции **ShapeRange,** являющуюся подмножеством коллекции **Shapes,** используется следующий синтаксис:

Shapes.Range(*index*)

index - индекс фигуры или массив индексов фигур.

Пример. Программа выводит на текущий лист имена находящихся на нем пять первых фигур.

```
Sub Использование_индекса_фигуры()
Dim Nf(1 To 100)
For i = 1 To 5
  Nf(i) = ActiveSheet.Shapes.Range(i).Name
  Cells(i, 1) = "Фигура" + Str(i) + ": " + Nf(i)
Next i
End Sub
```



Вывод на текущий лист имен находящихся на нем фигур

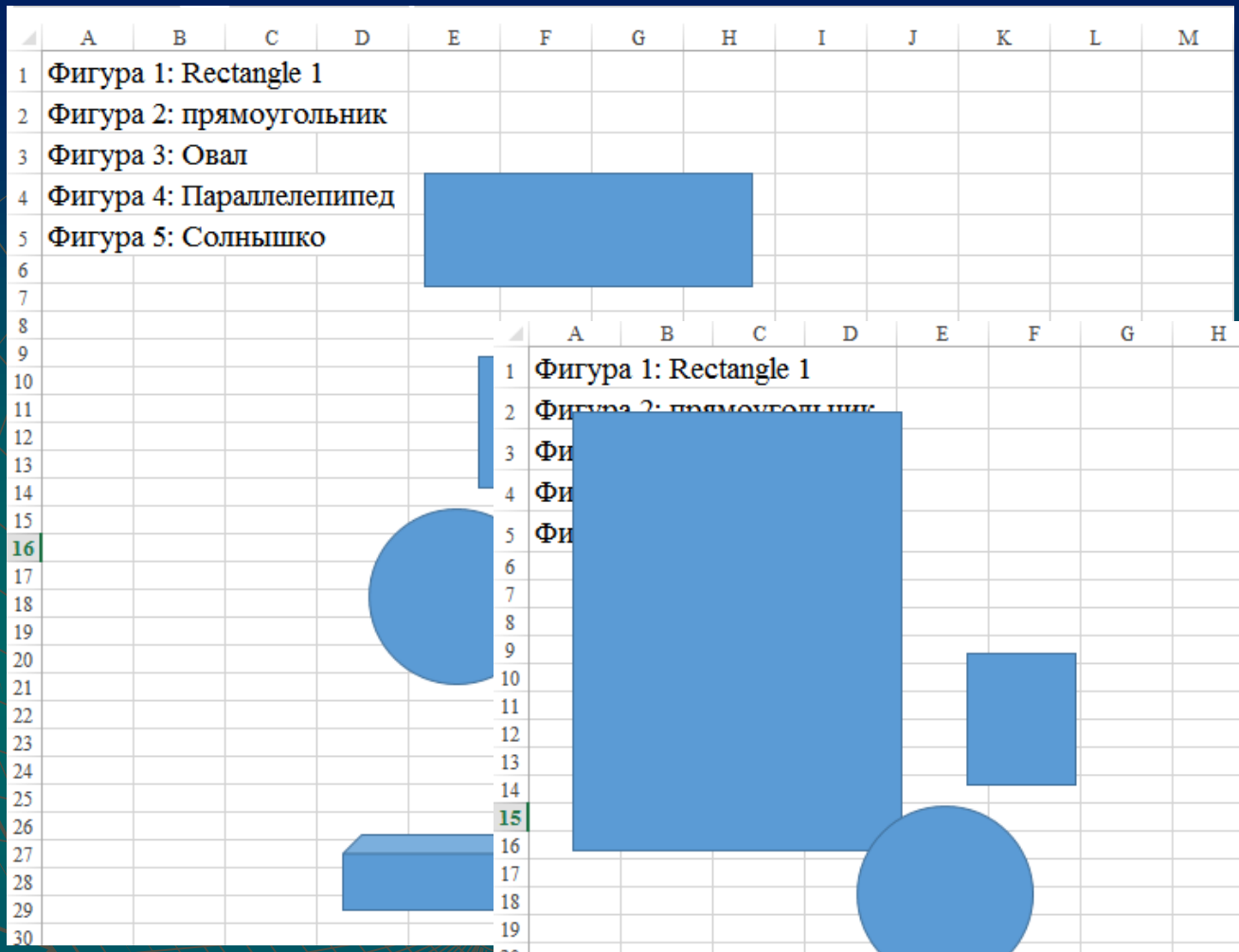
9.1.2. Использование свойств и методов фигур Shapes

9.1.2.1. Свойства фигур Shapes

Редактирование фигур осуществляется изменением их свойств. Объекты коллекции **Shapes** обладают как общими **(Left, Top, Height** и **Width),** так и специфическими (зависящими от типа объекта) свойствами.

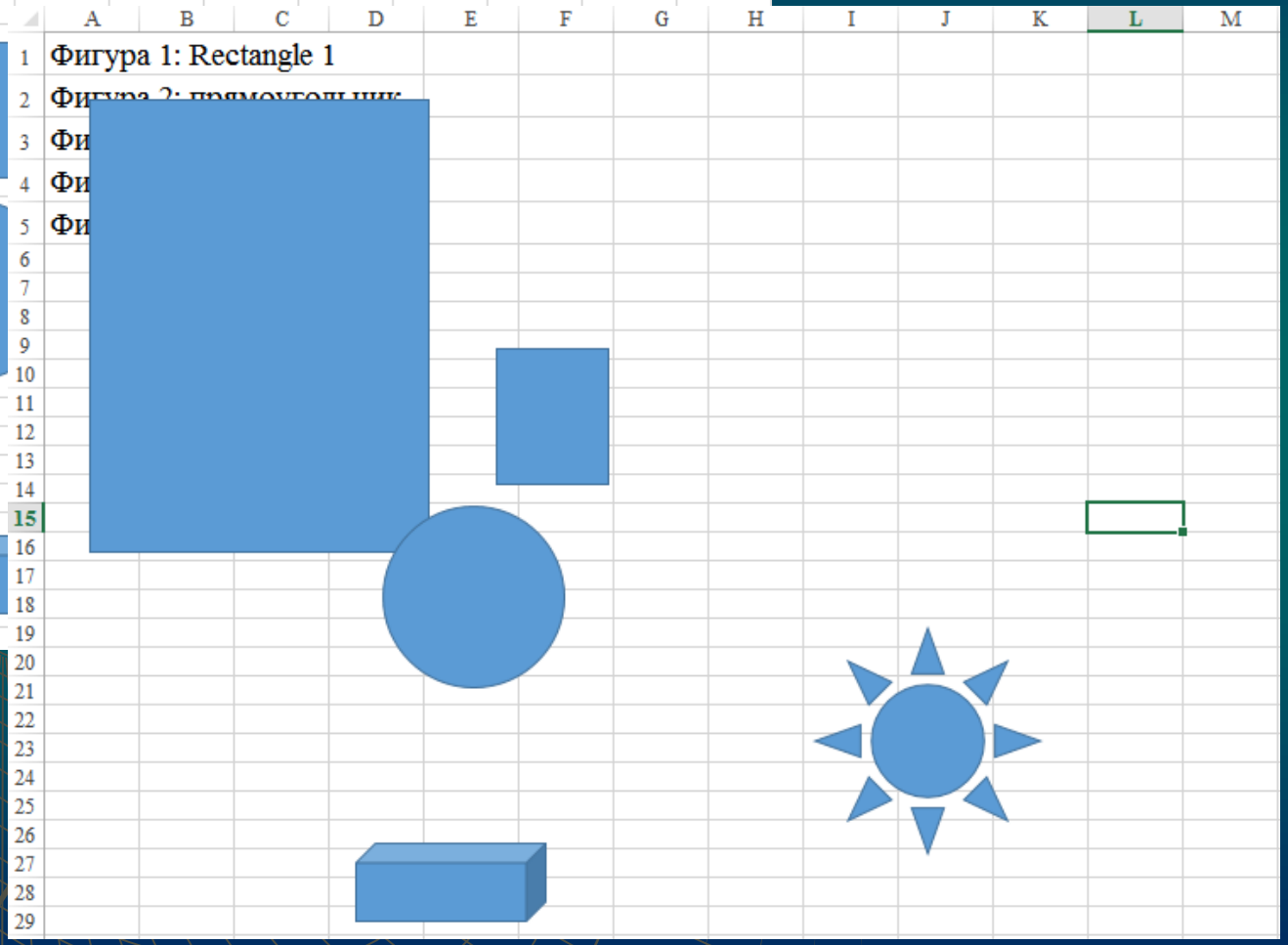
Пример. Программа изменяет некоторые общие свойства графического объекта "прямоугольник".

```
Sub Изменение_общих_свойств_прямоугольника()
  With ActiveSheet.Shapes("прямоугольник")
    .Top = 30
    .Left = 20
    .Height = 200
    .Width = 150
  End With
End Sub
```



Перед запуском программы

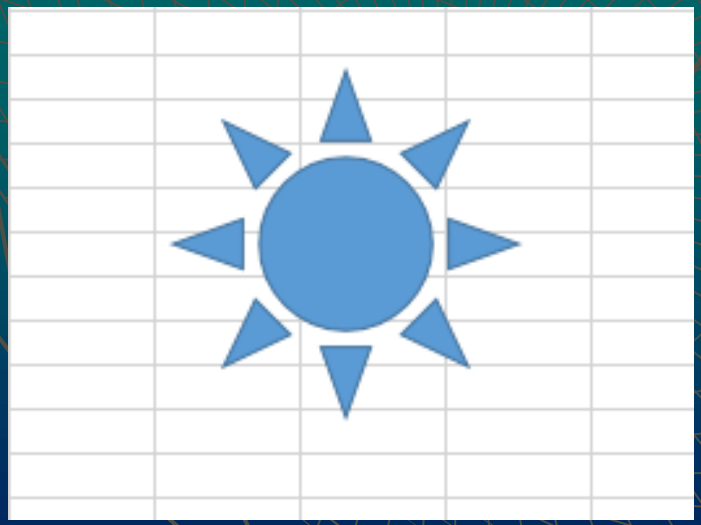
После запуска программы

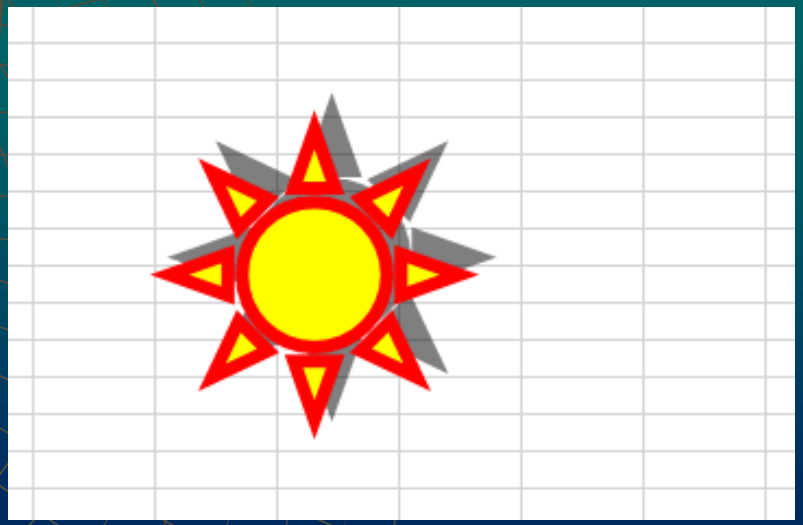


Изменение некоторых общих свойств объекта "прямоугольник"

Пример. Программа изменяет цвет заливки, цвет линий, толщину линий, стиль линий, тип тени графического объекта «солнышко».

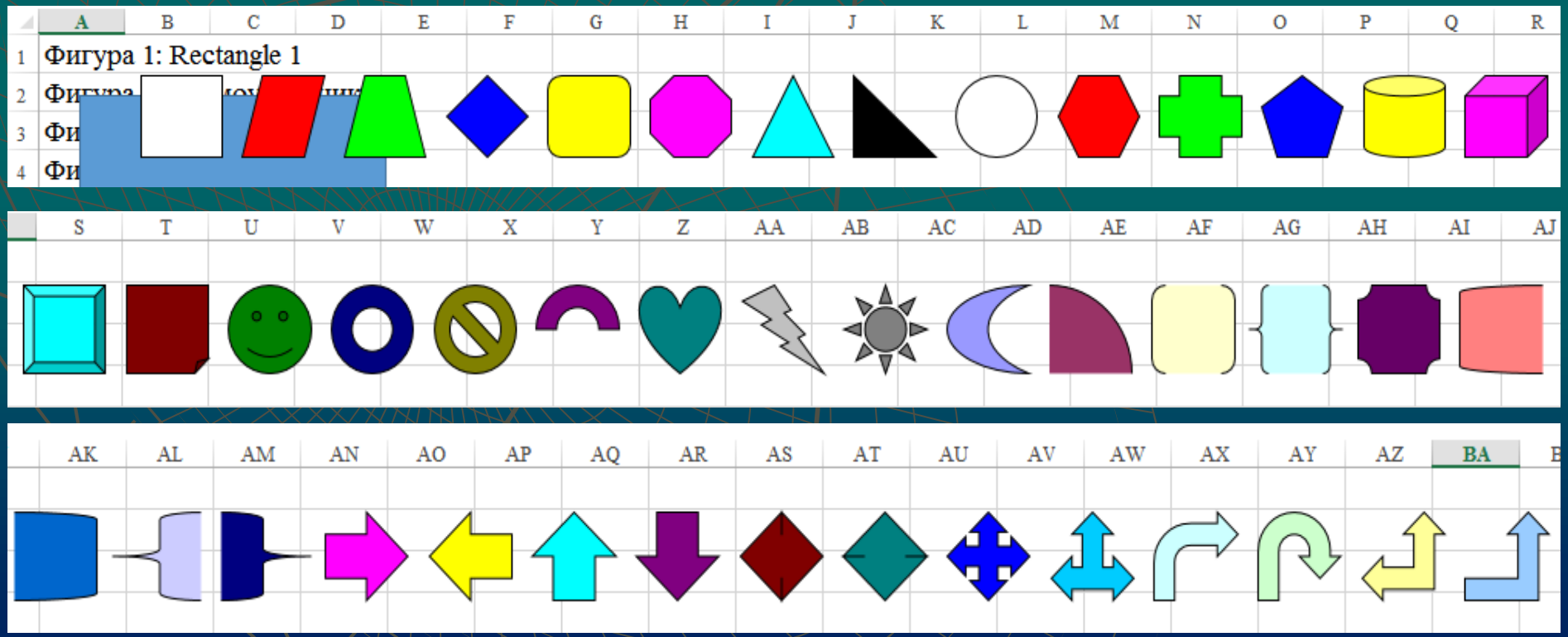
```
Sub Изменение_свойств_солнышка()
    With ActiveSheet.Shapes("солнышко")
        .Fill.ForeColor.SchemeColor = 13
        .Line.ForeColor.SchemeColor = 10
        .Line.Weight = 4.5
        .Line.Style = msoLineSingle
        .Shadow.Type = msoShadow2
    End With
End Sub
```

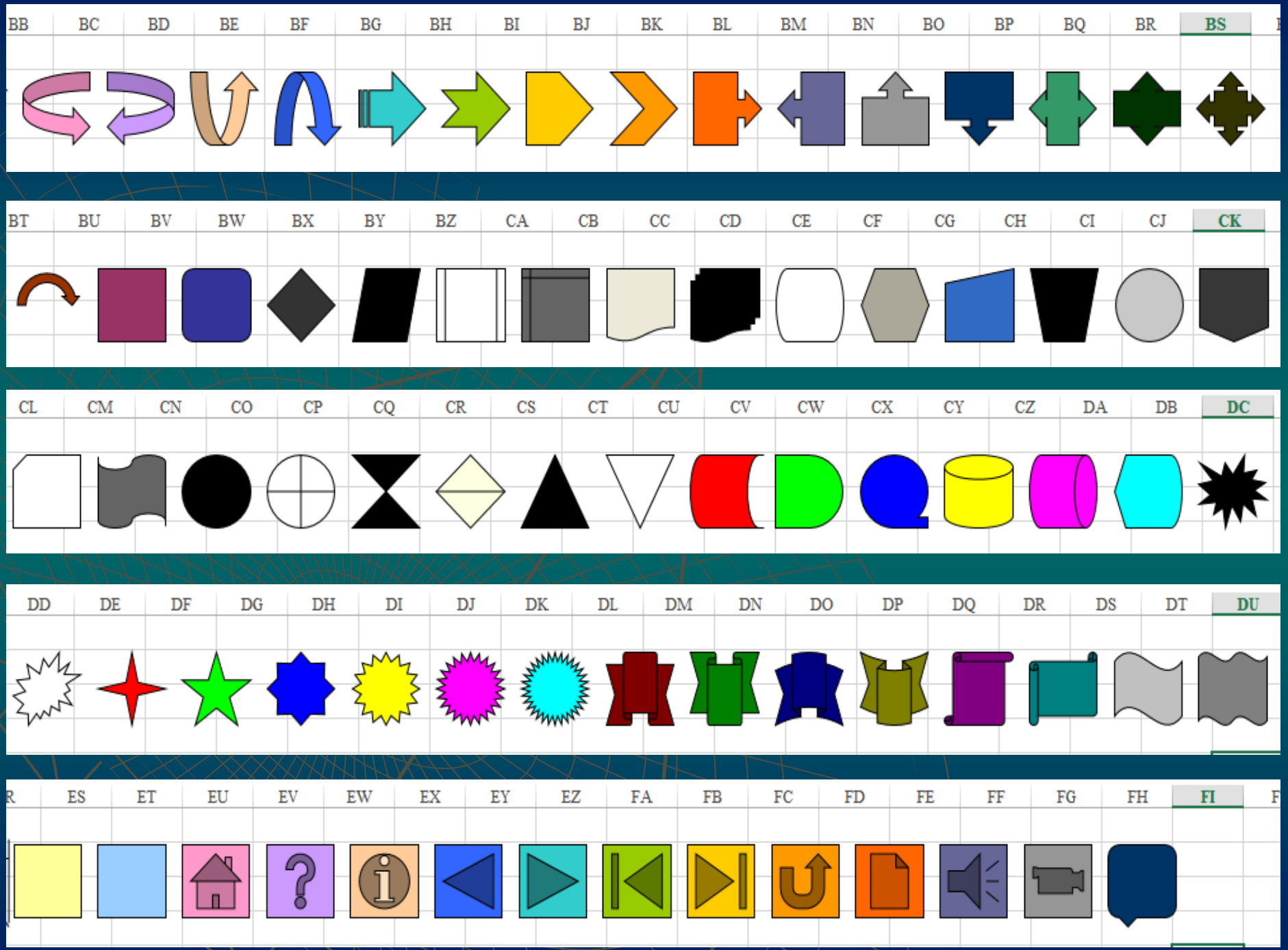




Пример. Программа выводит автофигуры по номерам их типов, изменяет цвет заливки, цвет линий добавленных объектов.

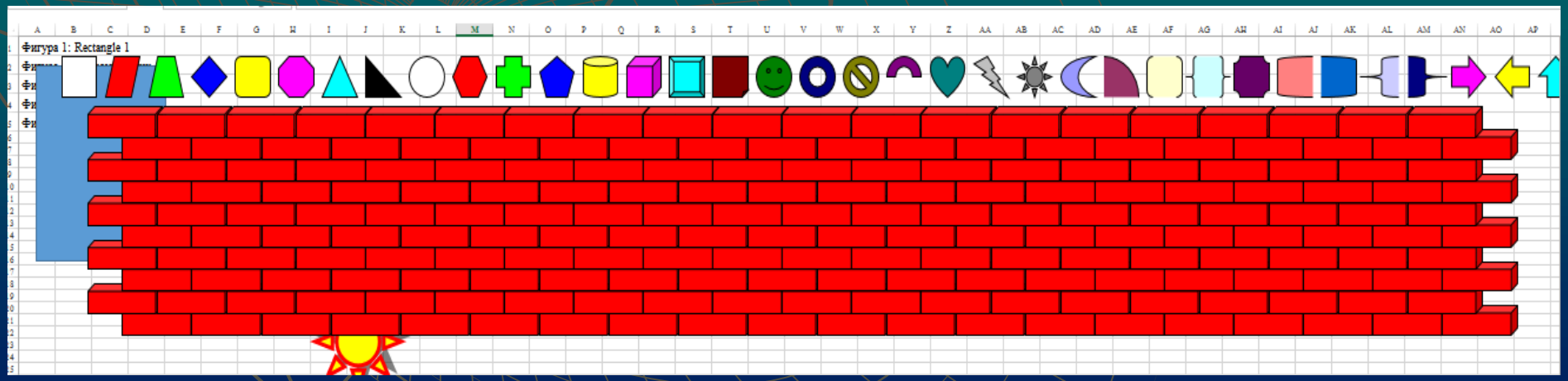
```
Sub Вывод_всех_автофигур()
For i = 1 To 137
  ActiveSheet.Shapes.AddShape(i, i * 50, 50, 40, 40).Select
    With Selection.ShapeRange
      .Fill.ForeColor.SchemeColor = i Mod 81
      .Line.ForeColor.SchemeColor = 8
    End With
Next i
End Sub
```




BB BC BD BE BF BG BH BI BJ BK BL BM BN BO BP BQ BR BS
BT BU BV BW BX BY BZ CA CB CC CD CE CF CG CH CI CJ CK
CL CM CN CO CP CQ CR CS CT CU CV CW CX CY CZ DA DB DC
DD DE DF DG DH DI DJ DK DL DM DN DO DP DQ DR DS DT DU
R ES ET EU EV EW EX EY EZ FA FB FC FD FE FF FG FH FI

Пример. Программа выводит на экран автофигуры «параллелепипед» по заранее рассчитанным координатам и изменяет цвет заливки, цвет линий объектов.

```
Sub Кирпичная_кладка()
For i = 1 To 10
  For j = 1 To 20
    x = j * 80 + 40 * (i Mod 2)
    y = 300 - i * 23
    ActiveSheet.Shapes.AddShape(14, x, y, 86, 30).Select
    With Selection.ShapeRange
        .Fill.ForeColor.SchemeColor = 2
        .Line.ForeColor.SchemeColor = 8
    End With
  Next j
Next i
End Sub
```



9.1.2.2. Методы фигур Shapes

Фигуры коллекции **Shapes** после их добавления в коллекцию можно перемещать, удалять, двигать, поворачивать, изменять внешний вид, добавлять в них текст и т.д.

Метод **SelectAll**

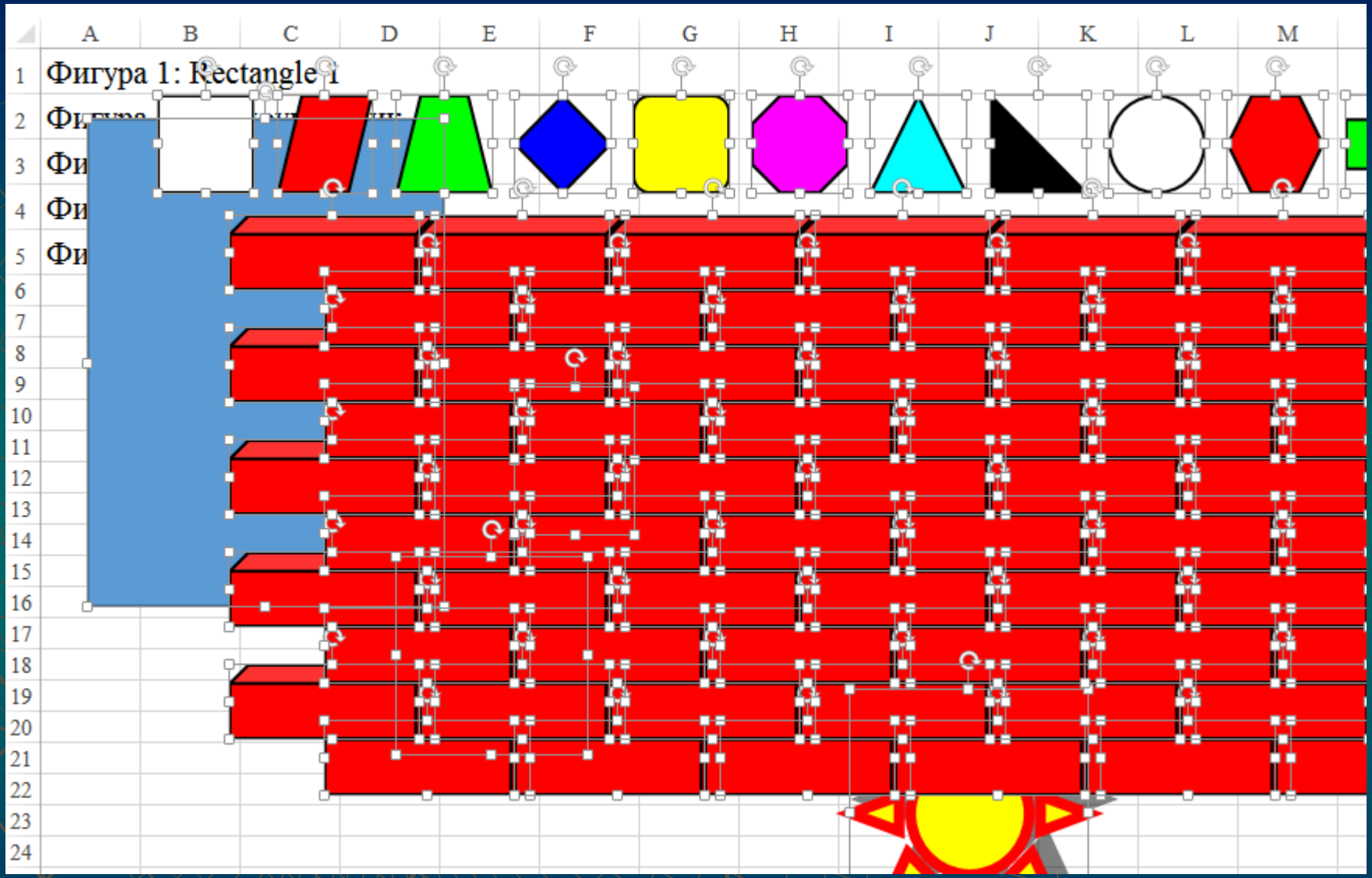
Для получения ссылки на всю коллекцию объектов в слое векторной графики необходимо использовать метод **SelectAll**.

Синтаксис

mDocument.Shapes.SelectAll

Пример. Для выделения всех объектов коллекции **Shapes** активного листа используется метод **SelectAll**.

```
Sub Выделение_всех_объектов()
    ActiveSheet.Shapes.SelectAll
End Sub
```



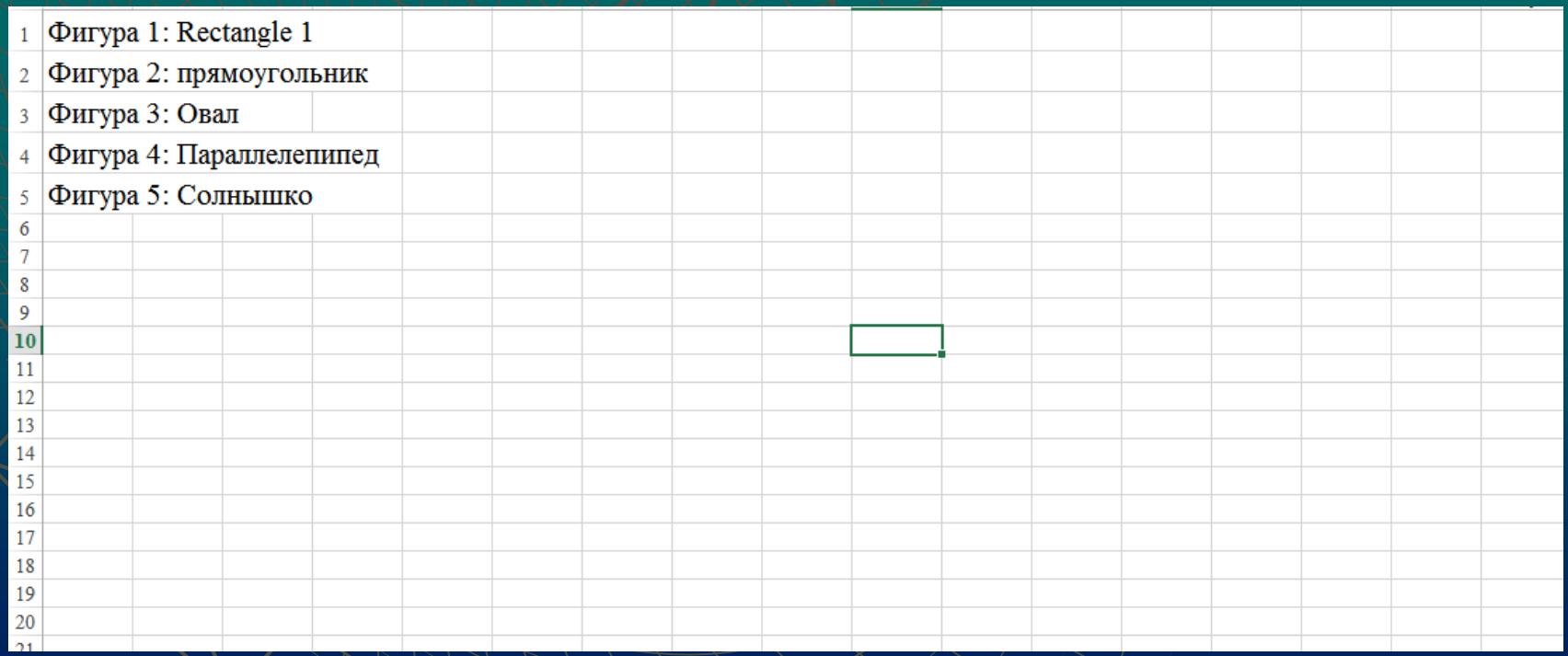
Выделение всех объектов коллекции **Shapes** активного листа

Метод Delete

Для удаления графических объектов необходимо использовать метод **Delete**.

Пример. Программа удаляет все объекты коллекции **Shapes** активного листа.

```
Sub Удаление_объектов()
    ActiveSheet.Shapes.SelectAll
    Selection.Delete
End Sub
```



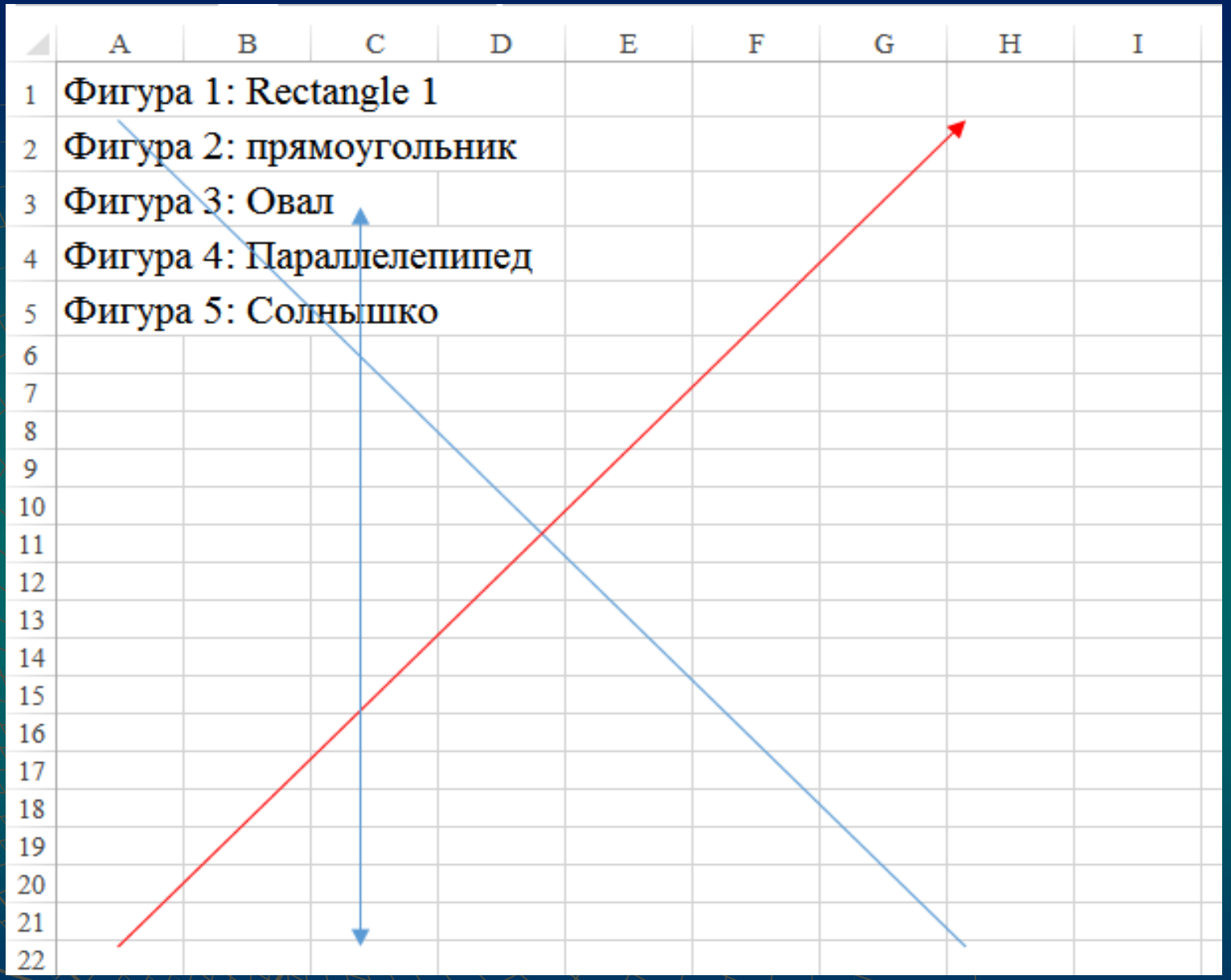
9.1.3. Добавление к коллекции Shapes специальных фигур

AddCallout	AddCallout(*Type* As MsoCalloutType, *Left* As Single, *Top* As Single, *Width* As Single, *Height* As Single) As Shape	Добавляет в документ не имеющую границ выноску. Возвращает **Shape**-объект, представляющий эту выноску.
AddConnector	AddConnector(*Type* As MsoConnectorType, *BeginX* As* Single, *BeginY* As Single, *EndX* As Single, *EndY* As Single) As Shape	Создает соединительную линию. Возвращает **Shape**-объект, который представляет новую соединительную линию. Когда добавляется соединительная линия, она не соединена ни с чем.
AddCurve	AddCurve*(SafeArrayOfPoints)* As Shape	Когда применяется к объекту коллекции **Shapes,** возвращает **Shape**-объект, который представляет кривую Безье в электронной таблице.
AddDiagram	AddDiagram(*Type* As MsoDiagramType, *Left* As Single, Гор As Single, *Width* As Single, *Height* As Single) As Shape	Создает диаграмму. Возвращает **Shape**-объект, который представляет новую диаграмму.

AddForm	AddFormControl(*Type* As XlFormControl, Left As Long, *Top* As Long, *Width* As Long, Height As Long) As Shape	Создает элемент управления Microsoft Excel. Возвращает **Shape**-объект, который представляет новый элемент управления.
AddLabel	AddLabel(Orientation As MsoTextOrientation, Left As Single, Гор As Single, *Width* As Single, *Height* As. Single) As Shape	Добавляет в документ надпись в виде прямоугольникас невидимыми границами и без заливки. Возвращает Shape-объект, который представляет эту новуюнадпись.
AddLine	AddLine*(BeginX* As Single, *BeginY* As Single, *EndX* As Single, *EndY* As Single) As Shape	Когда применяется, к объекту коллекции **Shapes,** возвращает **Shape**-объект, который представляет новую линию в электронной таблице.

Пример. Программа выводит и изменяет свойства трех линий: «Простая линия», «Линия со стрелкой в конце», «Линия с двумя стрелками».

```
Sub Линии()
  ActiveSheet.Shapes.AddLine(20, 20, 300, 300).Name = "Простая линия"
  ActiveSheet.Shapes.AddLine(20, 300, 300, 20).Name = "Линия со стрелкой в конце"
  ActiveSheet.Shapes.AddLine(100, 50, 100, 300).Name = "Линия с двумя стрелками"

  With ActiveSheet.Shapes("Линия со стрелкой в конце")
       .Line.ForeColor.SchemeColor = 10
       .Line.EndArrowheadStyle = msoArrowheadTriangle
       .Line.EndArrowheadLength = msoArrowheadLengthMedium
       .Line.EndArrowheadWidth = msoArrowheadWidthMedium
  End With
  With ActiveSheet.Shapes("Линия с двумя стрелками")
       .Line.BeginArrowheadStyle = msoArrowheadTriangle
       .Line.BeginArrowheadLength = msoArrowheadLengthMedium
       .Line.BeginArrowheadWidth = msoArrowheadWidthMedium
       .Line.EndArrowheadStyle = msoArrowheadTriangle
       .Line.EndArrowheadLength = msoArrowheadLengthMedium
       .Line.EndArrowheadWidth = msoArrowheadWidthMedium
  End With
End Sub
```



Вывод и изменение свойств трех линий

9.2.1. Добавление специальных фигур

9.2.1.1. Процедура AddLine

Для добавления линий используется процедура **AddLine**.

Синтаксис:

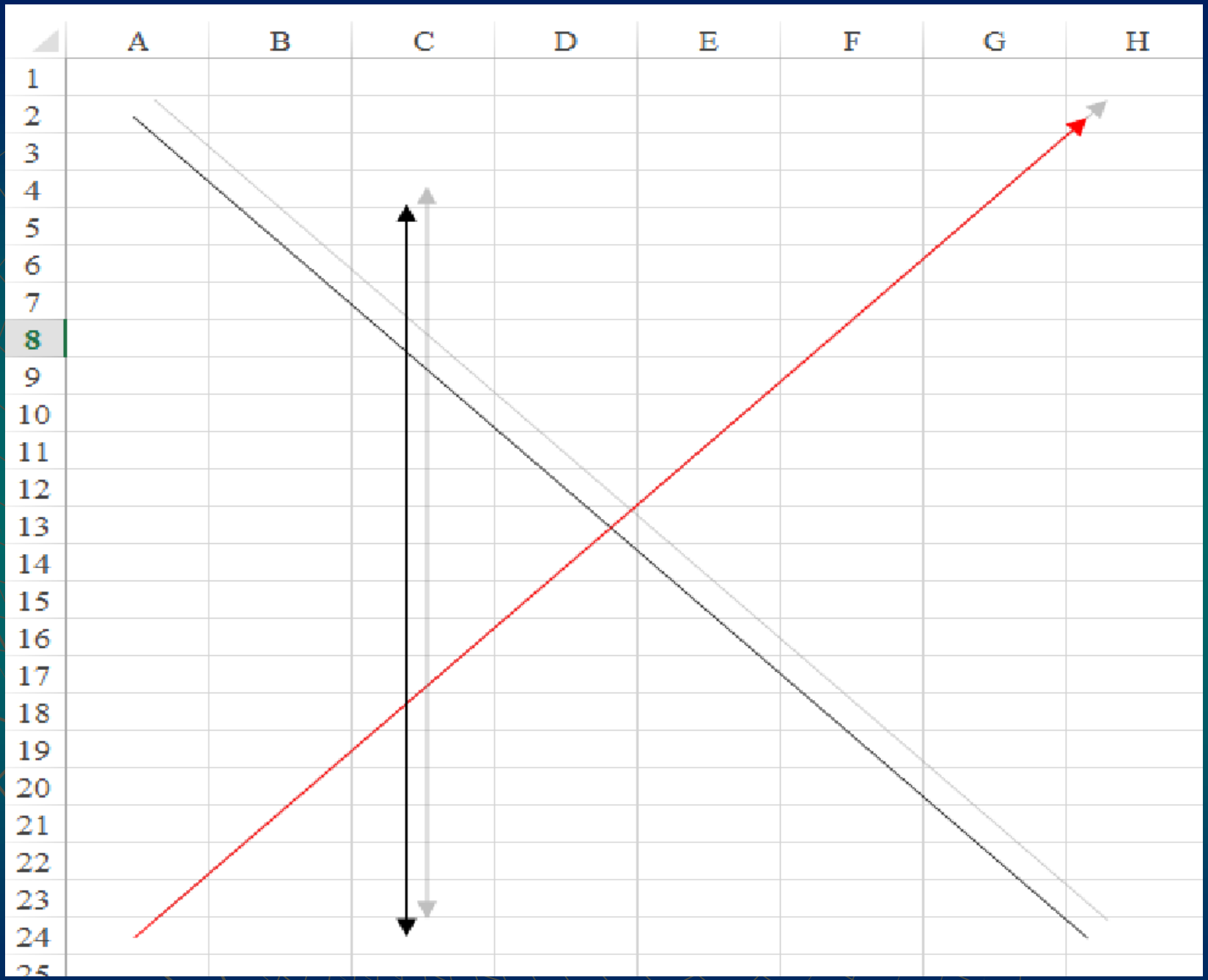
AddLine(X1, Y1, X2, Y2) As Shape

X1 As Single – горизонтальная координата начала линии;
Y1 As Single – вертикальная координата начала линии;
X2 As Single – горизонтальная координата конца линии;
Y2 As Single – вертикальная координата конца линии;.

Процедура возвращает **Shape**-объект, который представляет собой новую линию в документе.

Пример. Вывести на лист три линии: обычную, со стрелкой, с двумя стрелками.

```
Sub Линии()
  ActiveSheet.Shapes.AddLine(20, 20, 300, 300).Name = "Простая линия"
  ActiveSheet.Shapes.AddLine(20, 300, 300, 20).Name = "Линия со стрелкой в конце"
  ActiveSheet.Shapes.AddLine(100, 50, 100, 300).Name = "Линия с двумя стрелками"

  With ActiveSheet.Shapes("Линия со стрелкой в конце")
       .Line.ForeColor.SchemeColor = 10
       .Line.EndArrowheadStyle = msoArrowheadTriangle
       .Line.EndArrowheadLength = msoArrowheadLengthMedium
       .Line.EndArrowheadWidth = msoArrowheadWidthMedium
  End With
  With ActiveSheet.Shapes("Линия с двумя стрелками")
       .Line.BeginArrowheadStyle = msoArrowheadTriangle
       .Line.BeginArrowheadLength = msoArrowheadLengthMedium
       .Line.BeginArrowheadWidth = msoArrowheadWidthMedium
       .Line.EndArrowheadStyle = msoArrowheadTriangle
       .Line.EndArrowheadLength = msoArrowheadLengthMedium
       .Line.EndArrowheadWidth = msoArrowheadWidthMedium
  End With
End Sub
```



Результат выполнения программы

9.2.1.1. Процедура AddCallout

Для добавления выноски используется процедура **AddCallout**.

Синтаксис:

AddCallout(Type, Left, Top, Width, Height) As Shape

Type As MsoCalloutType – тип выноски;
Left As Single – координаты левых углов по горизонтали;
Top As Single - координаты верхних углов по вертикали;
Width As Single – ширина объекта;
Height As Single – высота объекта.

Процедура возвращает **Shape-**объект, который представляет собой новую не имеющую границ выноску.

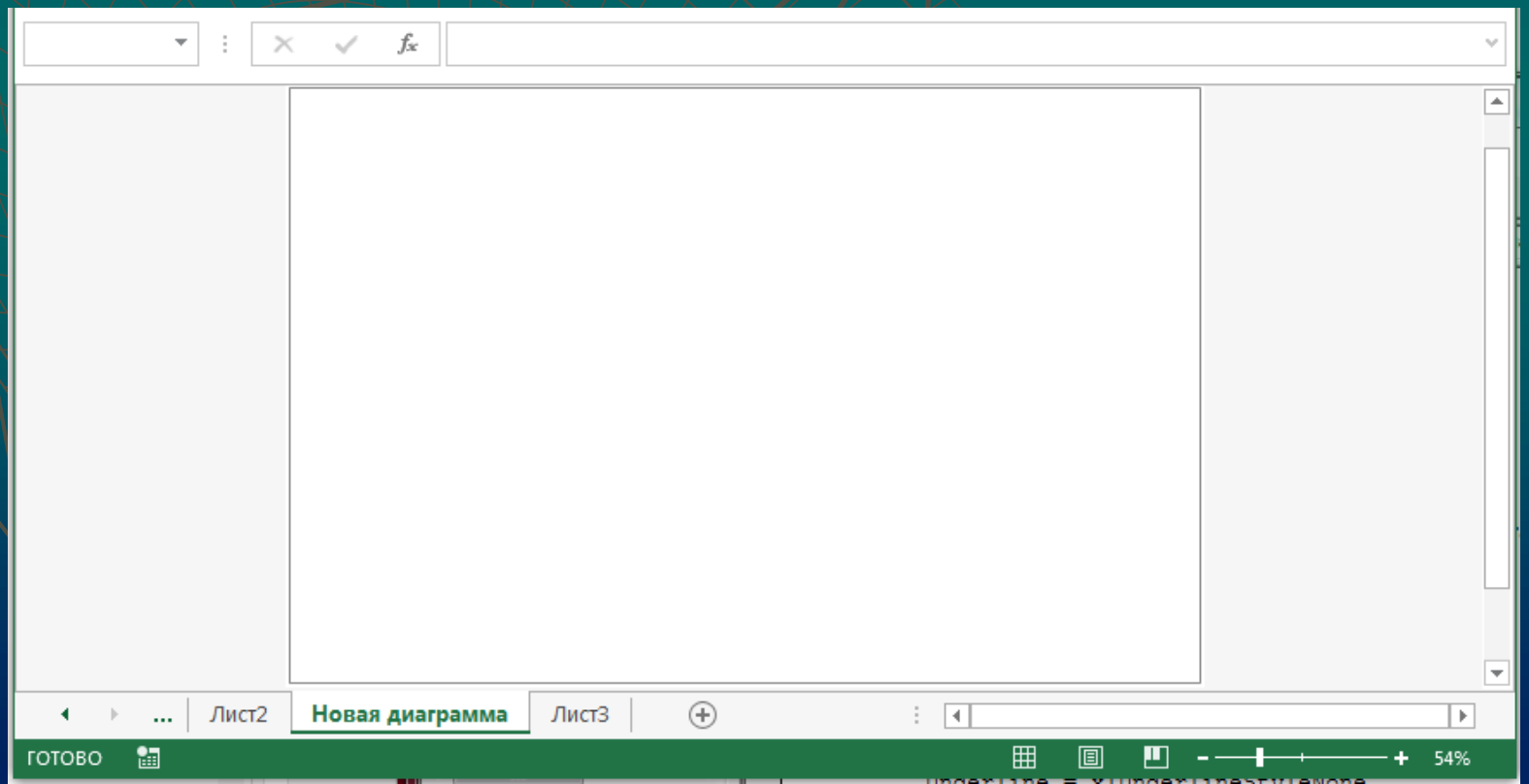
9.2.1.3. Добавление диаграмм

Для работы с диаграммами используется объект ***Chart***.

Создание диаграммы осуществляется при помощи метода Add коллекции Charts: Charts.Add

Пример. Следующий код добавляет новый лист диаграммы в рабочую книгу.

```
ActiveWorkbook.Charts.Add(, ActiveSheet).Name = "Новая диаграмма"
```



Единственное **обязательное действие** с диаграммой - определить источник данных. Для этого предназначен метод **SetSourceData**.

В качестве источника может выступать объект **Range**.

Второй параметр необязательный. Он определяет, в каком порядке считывать данные - сначала по столбцам или сначала по строкам.

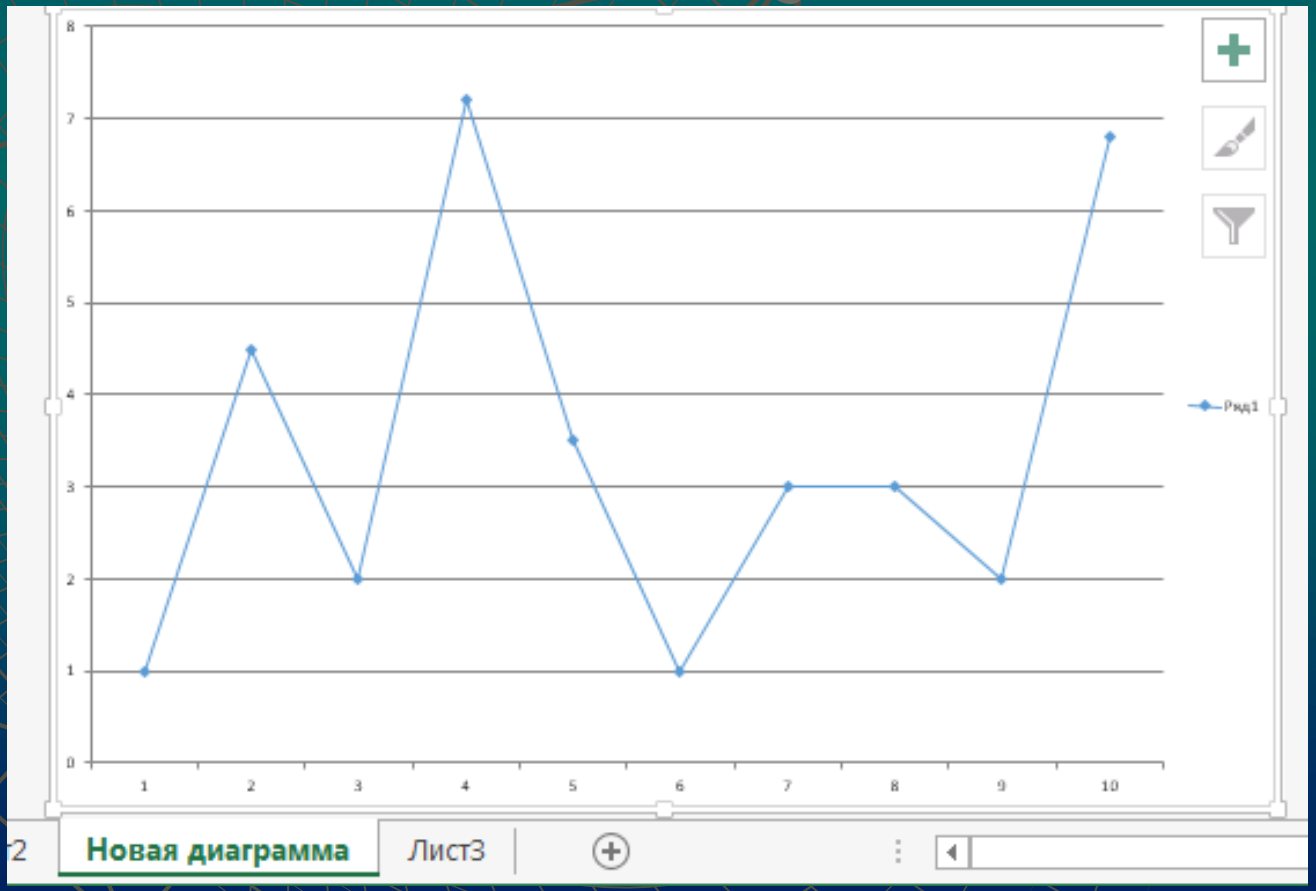
Пример. Возьмем в качестве источника ячейки **A1:A10** второго листа

```
With ActiveWorkbook.Charts("Новая диаграмма")
   .SetSourceData (Sheets("Лист2").Range("A1:A10"))
```

На практике необходимо определить тип диаграммы (по умолчанию «обычная гистограмма». Для этого используется свойство **ChartType** (всего 73 значения).

Пример. В качестве типа диаграммы используем xlLineMarkers.

```
.ChartType = xlLineMarkers
```

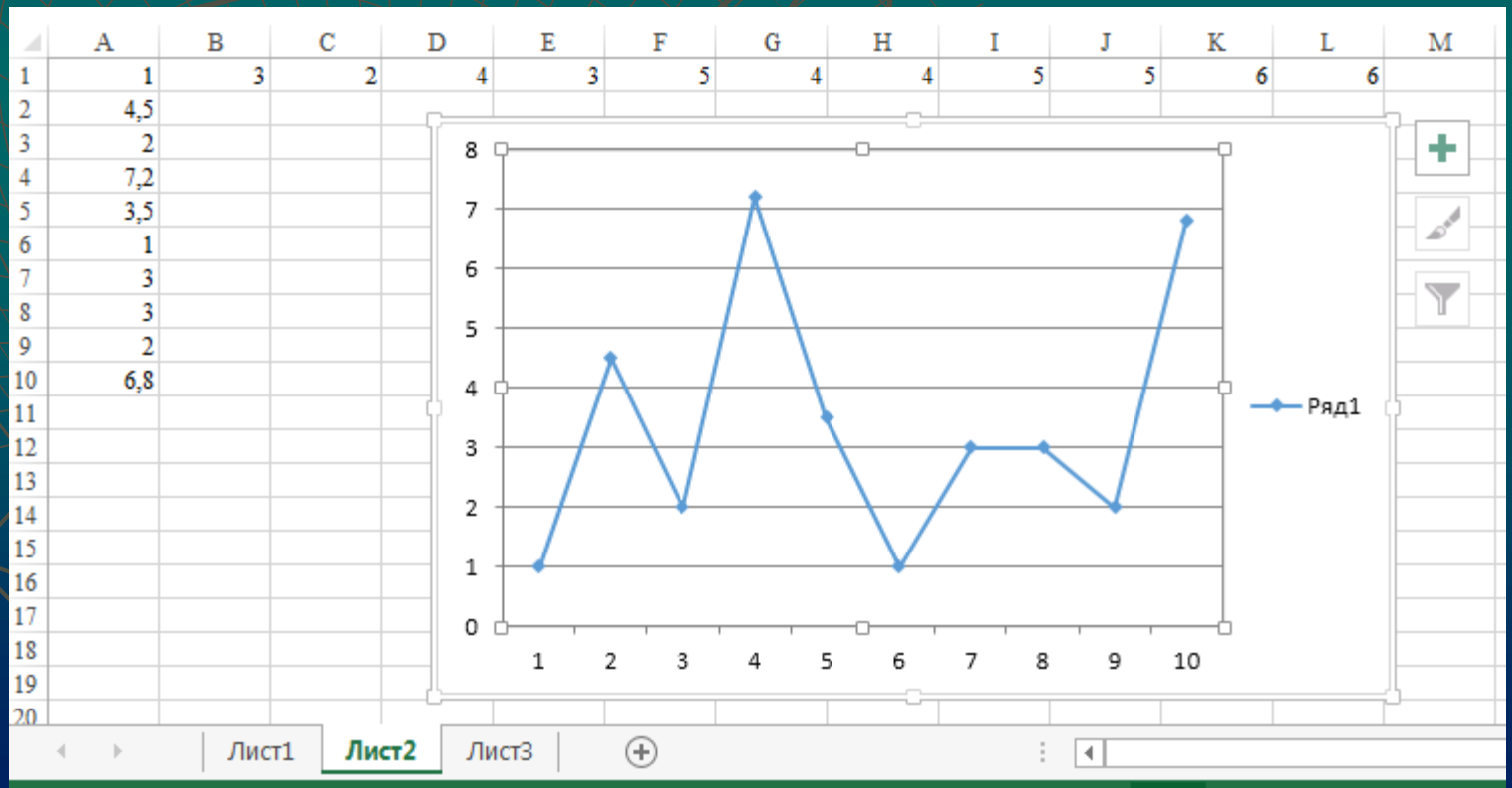


Если необходимо переместить диаграмму на уже существующий лист, то это можно выполнить при помощи метода **Location**.

Отдельный лист, созданный для диаграммы, при этом автоматически исчезнет.

Пример. Поместим диаграмму на лист «Лист2».

```
.Location xlLocationAsObject, "Лист2"
```



Некоторые свойства диаграмм

ChartArea - свойство возвращает одноименный объект ChartArea, который представляет собой область, занимаемую диаграммой и используется для настройки внешнего вида диаграммы (свойства Font, Interior и т.п.).

ChartTitle - свойство возвращает одноименный объект, при помощи которого можно настроить заголовок диаграммы (с такими свойствами, как Text, Font, Border и т.п.).

HasDataTable - добавление таблицы с числами, на основе которых была создана диаграмма.

SizeWithWindow — если поставить значение этого свойства в True (по умолчанию False), то размер диаграммы будет подогнан таким образом, чтобы точно соответствовать размеру листа.

Visible — возможность спрятать диаграмму без ее удаления.

Некоторые методы диаграмм

метод *Activate()* Он позволяет сделать диаграмму активной;

метод *ApplyDataLabels()* позволяет поместить на диаграмму метки для размещенных на ней данных. Этот метод принимает множество параметров, которые позволяют настроить отображение данных меток (показывать или не показывать значения и т.п.);

метод *Axes()* возвращает объект, представляющий оси диаграммы. Затем этот объект можно использовать для настройки данных осей;

Refresh() — возможность обновить диаграмму, если изменились данные, на основе которых она строилась;

Delete() — просто удаляет диаграмму.

9.2.2. Построение графиков функций.

9.2.2.1. Общая схема построения графиков функций

Алгоритм построения графика непрерывной функции $y = f(x)$ на отрезке $[a; b]$ состоит в следующем:

1-й шаг. Ввод параметров графика функции $y = f(x)$: начало и конец отрезка $[a; b]$, масштаб вывода M, расчет количества разбиений отрезка, шаг приращения аргумента.

2-й шаг. Разобьем отрезок $[a; b]$ на n равных частей и сформируем массивы X значений аргумента и Y значений функции $y = f(x)$ в точках $a = x_0, x_1, x_2, \ldots, x_n = b$.

3-й шаг. Преобразуем массивы X и Y в новую систему координат листа Excel.

4-й шаг. Построим отрезки $[(x_{i-1}, f(x_{i-1})); (xi, f(x_i))]$ в системе координат листа Excel.

5-й шаг. Построим оси системы координат xOy, произведем разметку осей и цифровых надписей разметки;

9.2.2.2. Ввод параметров графика

Для построения графика функции ***y = f (x)*** определим функцию ***Fx***, представляющую любое допустимое выражение, например:

```
Function fx(x)
   fx = 3 * Sin(2 * x - 4)
End Function
```

Объявим массивы ***X*** и ***Y***, в которых будем размещать соответственно реальные координаты функции, а также массивы ***H*** и ***V***, предназначенные для размещения преобразованных экранных координат.

```
Sub График_функции()
'Вывод на лист графика функции в масштабе M:1
Dim X(10000) As Double, Y(10000) As Double
Dim H(10000) As Double, V(10000) As Double
```

Очистим лист Excel от всех графических объектов.

```
ActiveSheet.Shapes.SelectAll
Selection.Delete
```

Введем начало и конец отрезка **[*a*; *b*]**, масштаб вывода ***M***.

```
a = -5                          'Левая граница графика
b = 7                           'Правая граница графика
M = 40                          'Масштаб
```

Масштаб вывода ***M*** означает, что реальная единица графика соответствует ***40*** графическим единицам (пикселям) выводимого на лист изображения. Чем больше значение ***M***, тем крупнее будет изображение.

Для удобства пользователя все параметры можно вывести с помощью специально подготовленной формы.

Рассчитаем количество разбиений отрезка **[*a*; *b*]**. Для этого умножим длину отрезка на масштаб ***M***.

```
N = (b - a) * M                 'Количество разбиений
```

Тогда шаг приращения значений аргумента функции x:

$$a = x_0, x_1, x_2, \ldots, x_n = b.$$

будет определяться по формуле:

```
s = 1 / M                       'Шаг приращений
```

9.2.2.3. Формирование массивов значений X и Y

Для заполнения массива ***X*** значений $a = x_0, x_1, x_2, \ldots, x_n = b$ аргумента и массива значений $f(a) = f(x_0), f(x_1), f(x_2), \ldots, f(x_n) = f(b)$ функции ***Y*** воспользуемся циклом с параметром.

Здесь же определим максимальное ***Ymax*** и минимальное ***Ymin*** значения функции.

```
For i = 0 To N                          'массив реальных координат
  X(i) = a + i * s                      'Реальная координата x
  Y(i) = fx(X(i))                       'Реальная координата y
  If Y(i) > Ymax Then Ymax = Y(i)       'Максимальное значение y
  If Y(i) < Ymin Then Ymin = Y(i)       'Минимальное значение y
Next i
```

9.2.2.4. Преобразование координат

Для преобразования реальных значений аргумента x_i в экранные горизонтальные координаты графика h_i будем учитывать начало отрезка **[*a*; *b*]** и масштаб вывода ***M***:

$$h_i = (x_i - a) \cdot M + 10$$

Для преобразования реальных значений функции y_i в экранные вертикальные координаты графика v_i будем учитывать максимальное значение функции y_{max} и масштаб вывода ***M***:

$$v_i = (-y_i + y_{max}) \cdot M + 10$$

Программный код этой части программы может выглядеть следующим образом:

```
For i = 0 To N                              'преобразование координат
  H(i) = (X(i) - a) * M + 10                'Вертикальная координата
  V(i) = (-Y(i) + Ymax) * M + 10            'Горизонтальная координата
Next i
```

9.2.2.5. Построение графика функции

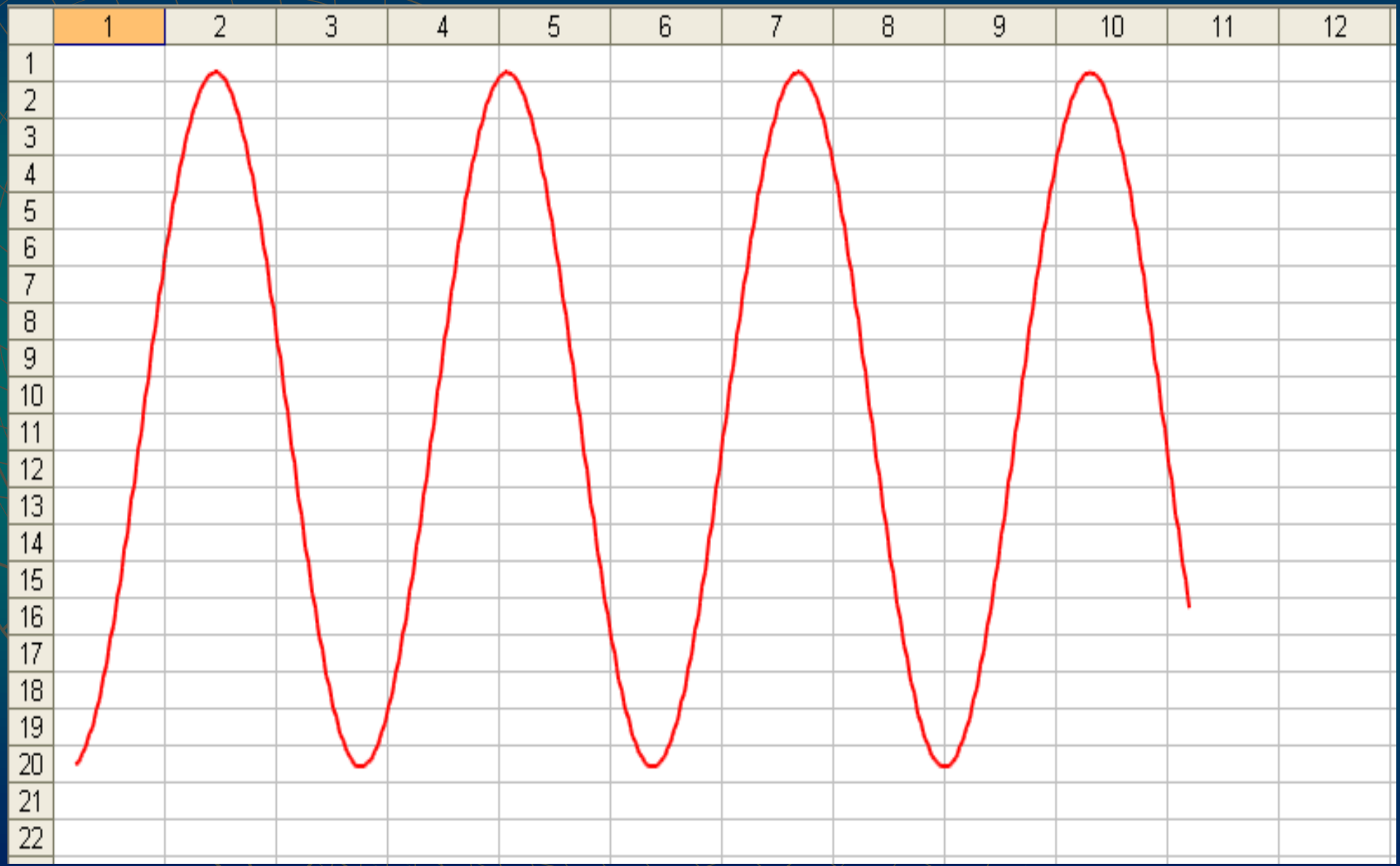
График функции представим в виде ***N*** отрезков с преобразованными графическими координатами

$[(h_{i-1}, v_{i-1})); (h_i, vi)]$.

Всем выводимым отрезкам дадим имена: L1, L2, L3, …, LN. Окрасим в се выводимые линии в цвет 10 (красный) и укажем толщину линий **1.5**.

```
For i = 1 To N                                  'построение графика функции
  ActiveSheet.Shapes.AddLine(H(i - 1), V(i - 1), H(i), V(i)) _
    .Name = "L" + Str(i)
  With ActiveSheet.Shapes("L" + Str(i))
    .Line.ForeColor.SchemeColor = 10
    .Line.Weight = 1.5
  End With
Next i
```

На рисунке представлен результат выполнения написанной части программы.



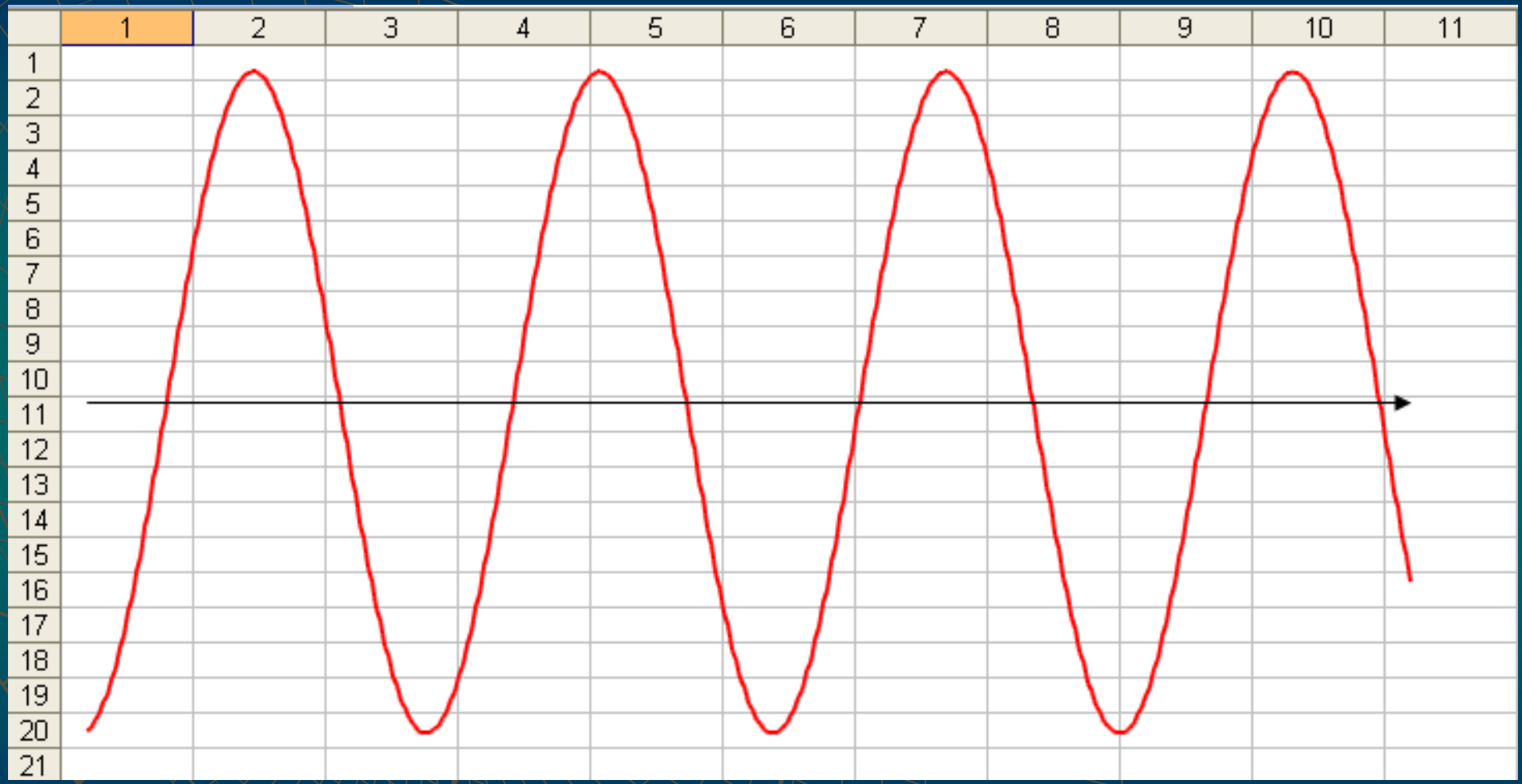
9.2.2.6. Рисуем оси координат

Используя формулы преобразования реальных координат в координаты листа Excel

$$h = (x - a) \cdot M + 10 \qquad v = (-y + y_{max}) \cdot M + 10$$

вычислим координаты начала и конца оси ***Ox***. Затем выведем ось на лист и определим стрелку на конце выводимой оси.

```
h1 = (a - a) * M + 10                          'Рисуем ось Ox
v1 = (0 + Ymax) * M + 10
h2 = (b - a) * M + 10
v2 = (0 + Ymax) * M + 10
ActiveSheet.Shapes.AddLine(h1, v1, h2, v2).Name = "Ось Ox"
With ActiveSheet.Shapes("Ось Ox")
   .Line.EndArrowheadStyle = msoArrowheadTriangle
   .Line.EndArrowheadLength = msoArrowheadLengthMedium
End With
```

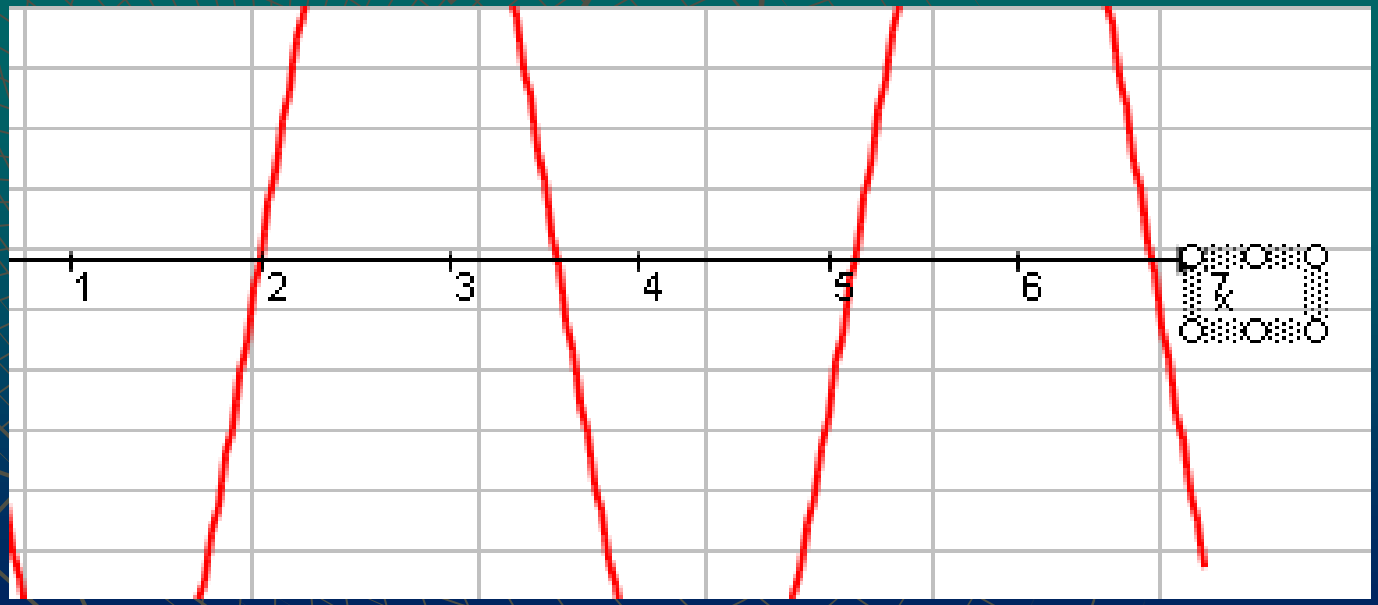

1 2 3 4 5 6 7 8 9 10 11
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

На выведенной оси выведем координатные единичные деления и укажем значения для всех делений

```
For i = Fix(a) To Fix(b)                    'Рисуем деления на оси Ox
  h1 = (i - a) * M + 10
  v1 = -2 + Ymax * M + 10
  h2 = (i - a) * M + 10
  v2 = 2 + Ymax * M + 10
  ActiveSheet.Shapes.AddLine(h1, v1, h2, v2).Name = "x" + Str(i)

  h1 = (i - a) * M + 7                      'Подписи оси Ox
  v1 = 2 + Ymax * M + 8
  ActiveSheet.Shapes.AddTextbox(msoTextOrientationHorizontal, _
                                h1, v1, 20, 10).Select
    With Selection
      .Characters.Text = Str(i)
      .Characters.Font.Size = 8
      .ShapeRange.Line.Visible = msoFalse
      .ShapeRange.Fill.Visible = msoFalse
  End With
Next i
```

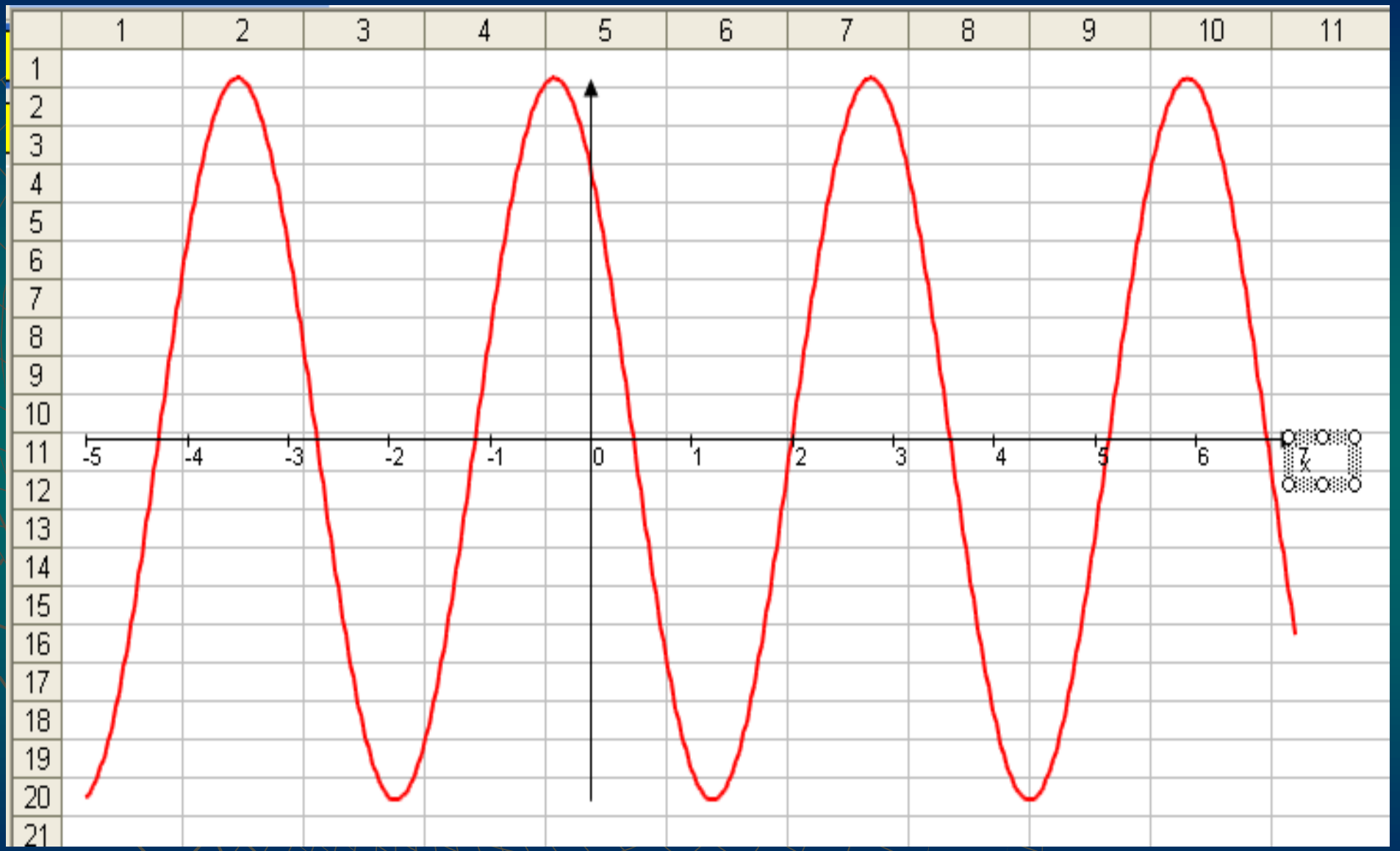
Для окончательного оформления оси Ox выведем в близи от стрелки наименование оси.

```
ActiveSheet.Shapes.AddTextbox(msoTextOrientationHorizontal, _
    (b - a) * M + 10, 2 + Ymax * M + 10, 20, 10).Select
    With Selection
        .Characters.Text = "x"
        .Characters.Font.Size = 8
        .ShapeRange.Line.Visible = msoFalse
        .ShapeRange.Fill.Visible = msoFalse
   End With
```



Аналогично выводим ось ***Oy***. Для этого рассчитываем координаты начала и конца оси, на конце определяем стрелку.

```
h1 = (0 - a) * M + 10                    'Рисуем ось Oy
v1 = (-Ymin + Ymax) * M + 10
h2 = (0 - a) * M + 10
v2 = (-Ymax + Ymax) * M + 10
ActiveSheet.Shapes.AddLine(h1, v1, h2, v2).Name = "Ось Oy"
With ActiveSheet.Shapes("Ось Oy")
   .Line.EndArrowheadStyle = msoArrowheadTriangle
   .Line.EndArrowheadLength = msoArrowheadLengthMedium
End With
```


1 2 3 4 5 6 7 8 9 10 11
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
-5 -4 -3 -2 -1 0 1 2 3 4 5 6
7
k

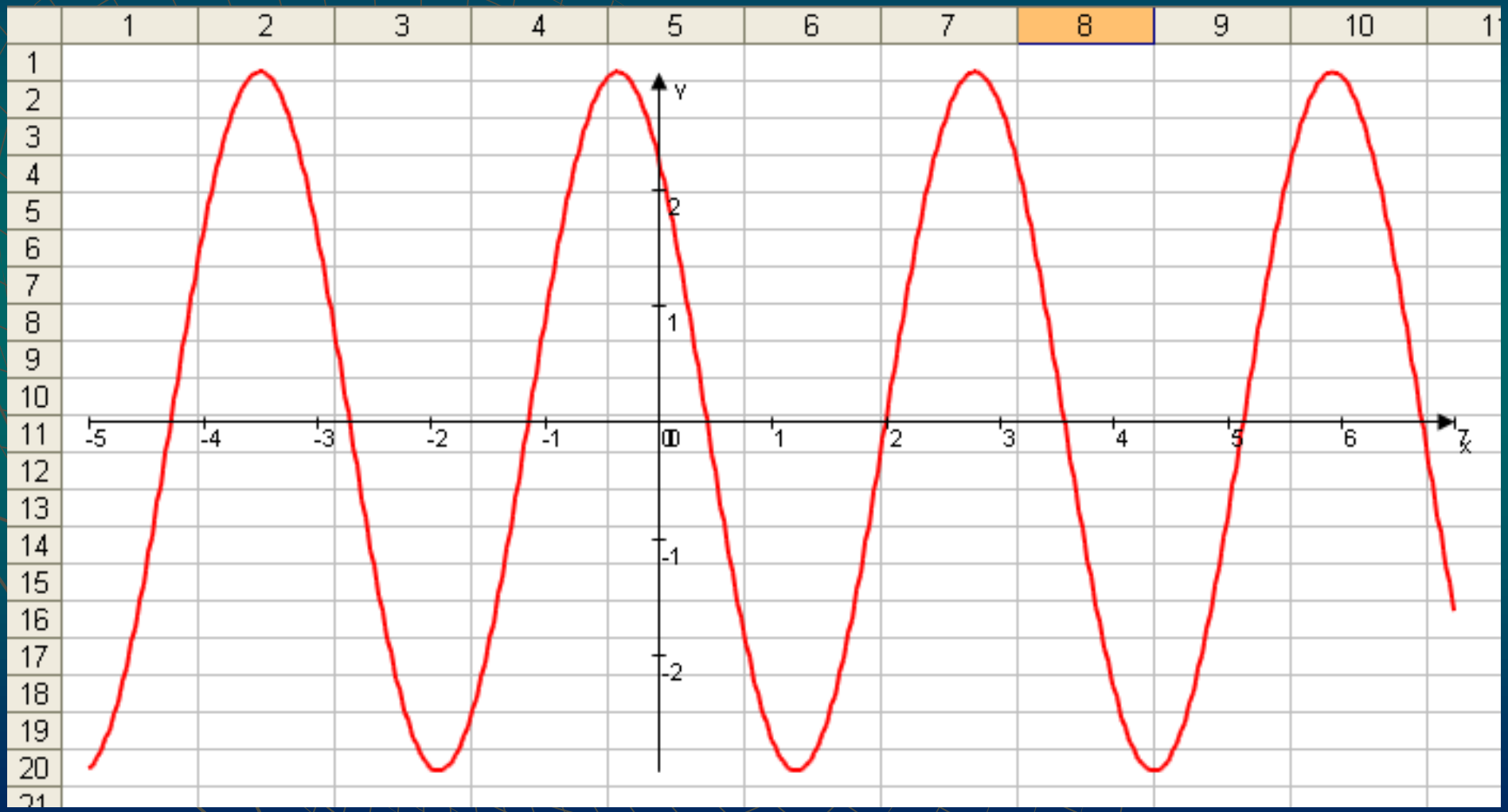
На оси рисуем деления и выводи подписи делений.

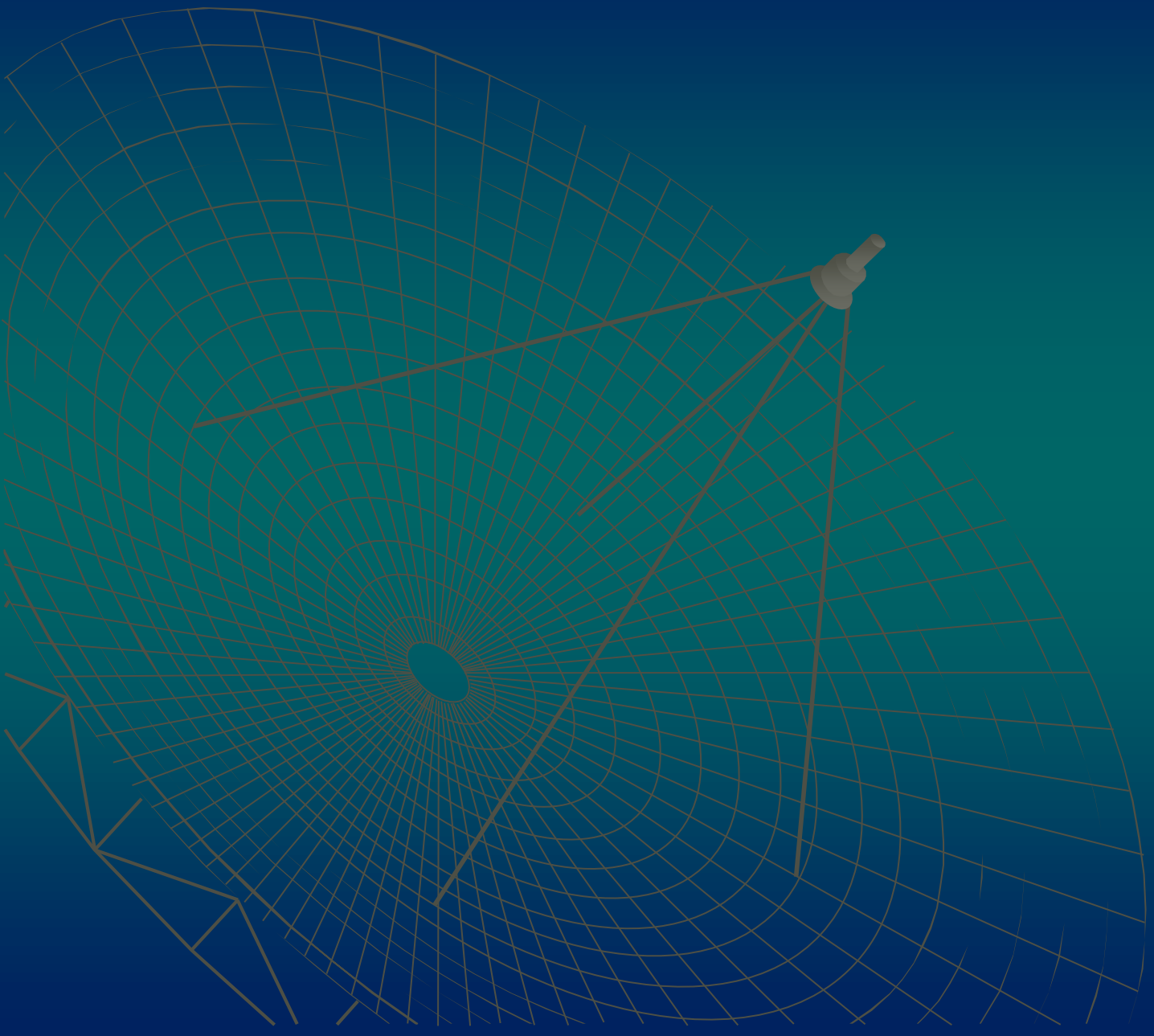
```
For i = Fix(Ymin) To Fix(Ymax)   'Рисуем деления на оси Oy
  h1 = -2 + (-a) * M + 10
  v1 = (-i + Ymax) * M + 10
  h2 = 2 + (-a) * M + 10
  v2 = (-i + Ymax) * M + 10
  ActiveSheet.Shapes.AddLine(h1, v1, h2, v2).Name = "y" + Str(i)

  h1 = (0 - a) * M + 9                'Подписи оси Oy
  v1 = (-i + Ymax) * M + 10
  ActiveSheet.Shapes.AddTextbox(msoTextOrientationHorizontal, _
                                h1, v1, 20, 10).Select
    With Selection
      .Characters.Text = Str(i)
      .Characters.Font.Size = 8
      .ShapeRange.Line.Visible = msoFalse
      .ShapeRange.Fill.Visible = msoFalse
  End With
Next i
```

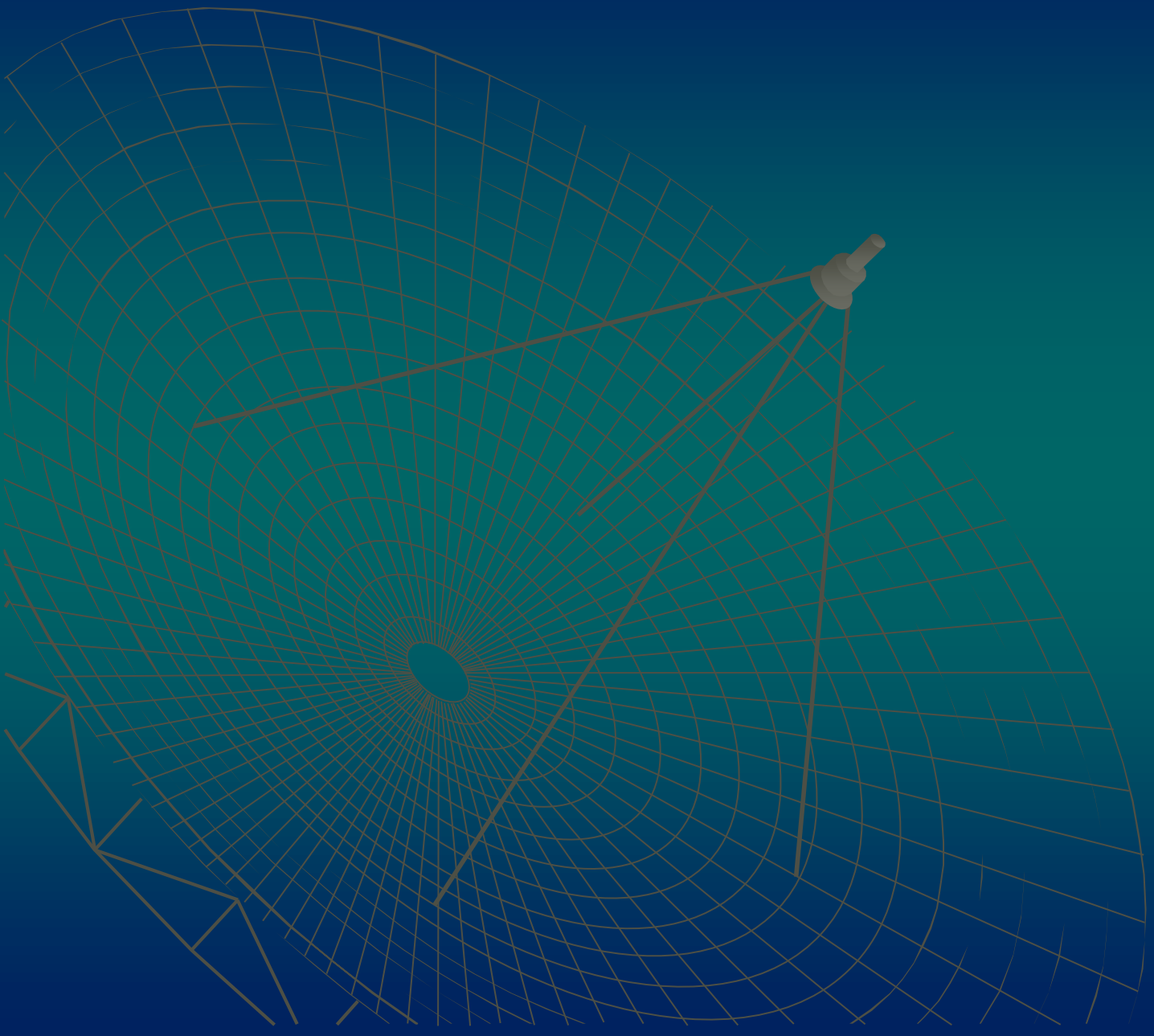
Выводим подпись оси Oy.

```
ActiveSheet.Shapes.AddTextbox(msoTextOrientationHorizontal, _
    (0 - a) * M + 14, 10, 20, 10).Select
    With Selection
        .Characters.Text = "y"
        .Characters.Font.Size = 8
        .ShapeRange.Line.Visible = msoFalse
        .ShapeRange.Fill.Visible = msoFalse
   End With
End Sub
```

Окончательный график функции

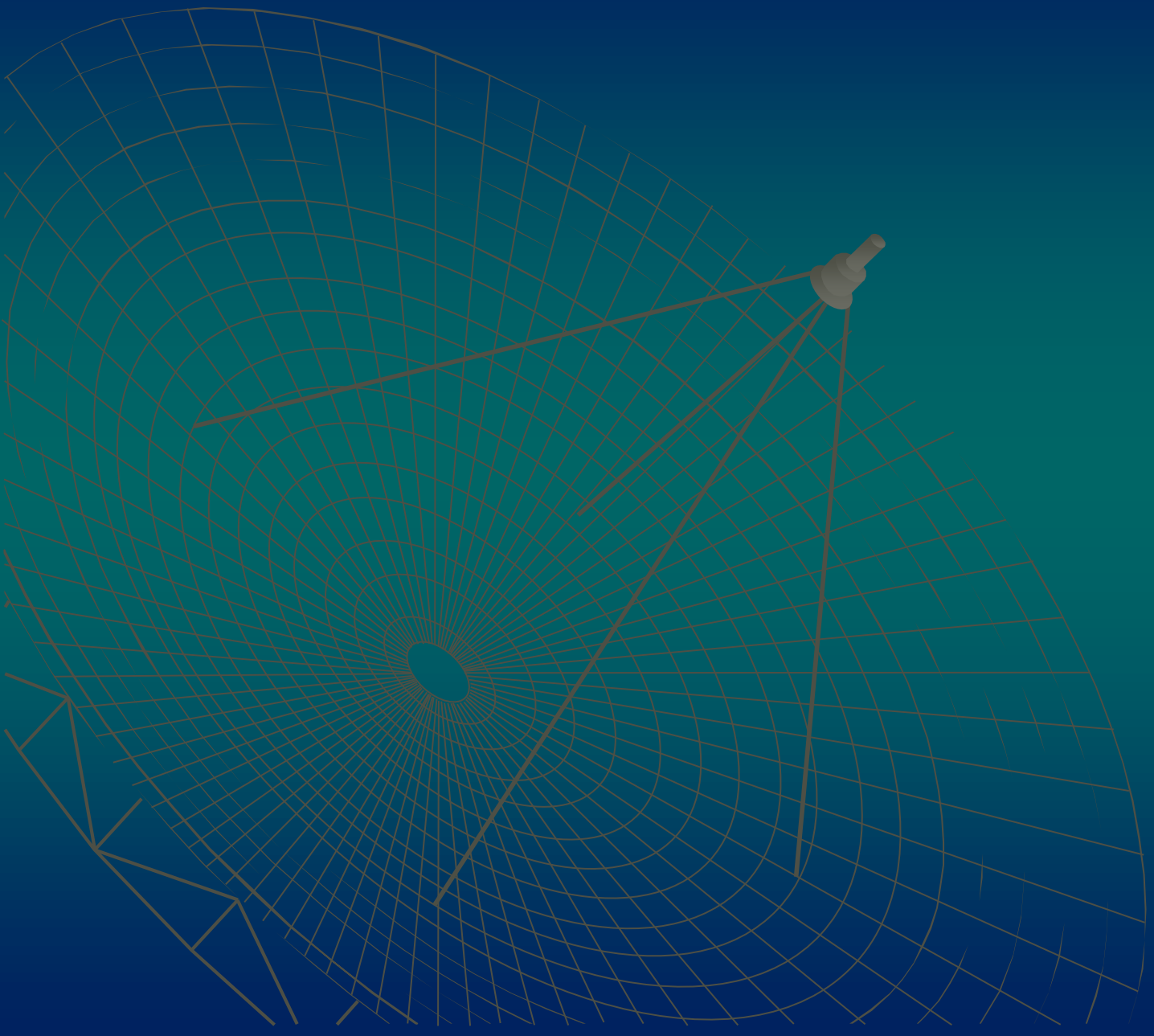






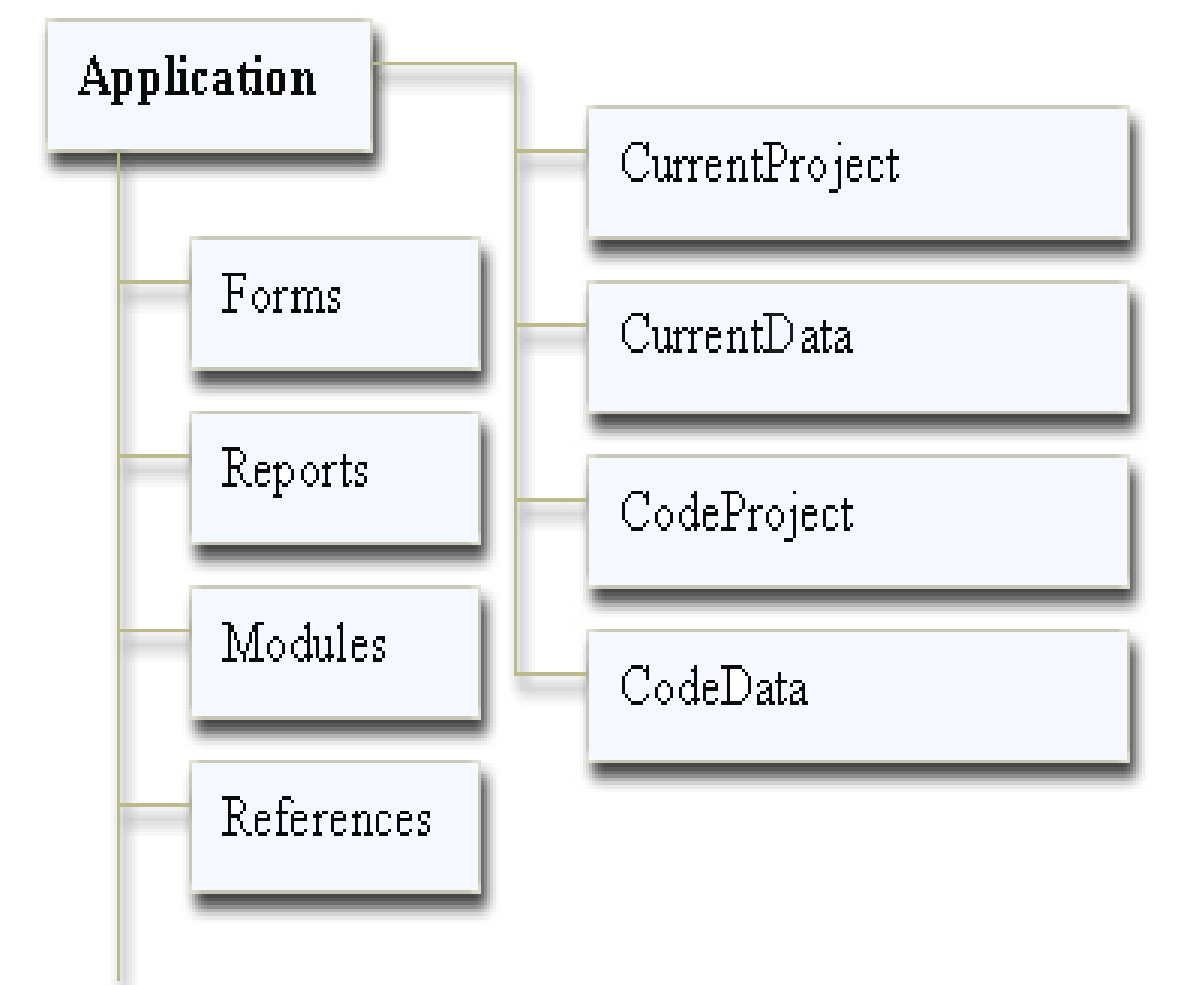
Глава 10. Базы данных

10.1. Понятие базы данных
10.2. Основные объекты MS Access
10.3. Объект Application, его свойства и методы
10.4. Управление базами данных. Библиотека ADO



10.1. Понятие базы данных

База данных представляет собой набор сведений, связанных с определенной темой или функцией, хранящийся в одном или нескольких файлах.



10.2. Основные объекты MS Access

Основными объектами базы данных MS Access являются:

1. Таблицы - это списки записей, относящиеся к конкретной области.
2. Формы - это объекты, с помощью которых пользователи могут добавлять, редактировать и отображать данные, хранящиеся в базе данных.
3. Отчеты - это объекты, позволяющие просматривать, форматировать и группировать информацию в базе данных.
4. Запросы - это объекты, предназначенные для нахождения информации в таблицах, фильтрации данных согласно критериям поиска (запросы на выборку), а также для редактирования данных (запросы на изменение).
5. Макросы - это инструмент, позволяющий автоматизировать задачи и добавлять функции в формы, отчеты и элементы управления.
6. Модули -

10.3. Объект Application, его свойства и методы

10.3.1 Основные свойства объекта Application

1. CodeContextObject — это свойство, которое позволяет определить, из какого объекта базы данных (формы, отчета и т.д.) был запущен модуль или макрос.
2. CodeData —свойство, позволяющее получить доступ к коллекциям AllDatabaseDiagrams, AllFunctions, AllQueries, AllStoredProcedures, AllTables и т.д., т.е. к диаграммам, функциям, запросам, процедурам, таблицам и базы данных.

Пример использования свойства CodeData для получения информации о всех таблицах в базе данных может выглядеть так:

```
For Each oTable In CodeData.AllTables
        Debug.Print oTable.Name
Next
```

3. CodeProject — используется для тех же целей, что и CodeData, но предоставляет доступ к коллекциям программных модулей AllForms, AllReports, AllMacros, AllModules и AllDataAccessPages
4. CurrentData — действует аналогично CodeData и CodeProject. Это свойство позволяет получить доступ к тем же коллекциям, включая полученные с внешнего источника данных (SQL Server). Аналогично работает свойство CurrentProject.

10.3.2 Основные методы объекта Application

1. AccessError() —метод для обработки ошибок. Он позволяет получить описание ошибок библиотек Access.
2. CloseCurrentDatabase() — закрывает текущую базу данных без закрытия Access. Обычно применяется для того, чтобы затем открыть другую базу данных без запуска нового экземпляра Access.

3. CreateControl() — позволяет программным образом создать элемент управления на форме.
4. DAvg(), DSum(), DCount(), DMax(), DMin() и т. п. — позволяют применить агрегатные функции к столбцу (или набору записей) в таблице или представлении;

10.3.3 Объект DoCmd

DoCmd – это объект, позволяющий программным образом выполнять макрокоманды Access — те действия (actions), которые можно просмотреть в окне конструктора макрокоманд.

Пример использования DoCmd, выполняющий запрос на языке SQL:
Application.DoCmd.RunSQL "Delete from table1"

Пример 1. Получить информацию о всех формах в базе данных Access.

```
Dim oA As AccessObject
For Each oA In CurrentProject.AllForms
        Debug.Print oA.Name
Next
```

Пример 2. Проверить открыта ли форма и вывести соответствующее сообщение на экран.

```
Debug.Print CurrentProject.AllForms("Форма1").IsLoaded
```

10.4. Управление базами данных. Библиотека ADO

Библиотека ADO (ActivX Data Object) – это специальный инструмент для работы с данными реализует доступ к объектам баз данных, электронным таблицам, мультимедийным данным и т.д.

10.4.1 Объект Connection

Объект Connection является объектом верхнего уровня, который представляет собой открытое соединение с источником данных. Этот объект содержит все остальные объекты и семейства.

Основные методы объекта Connection

Close – закрывает соединение и все активные наборы записей для данного подключения

Open – открывает сеанс подключения к источнику данных

Execute – выполняет запрос, инструкцию либо процедуру, доступную провайдеру

Свойства объекта Connection

ConnetionString – определяет параметры, используемые при подключении к источнику данных (имя провайдера, имя файла с информацией о подключении, путь к файлу сервера)

CursorLocation – определяет расположение курсора или область, где выполняется работа с данными

Mode – определяет режим доступа для изменения данных (только для чтения, только для записи и т.д.)

Provider – определяет имя провайдера для данного подключения

Пример подключения к источнику данных

```
Public Sub Work_Connection()
Dim conn As New ADODB.Connection
Dim ConnString As String
Dim cmd As New ADODB.Command
Dim rs As ADODB.Recordset
'задаем строку подключения
ConnString = "Provider=Microsoft.Jet.OLEDB.4.0;" _
          & "Data Source=H:\Work\S1.MDB;" _
          & "Persist Security Info=False"
'подключение к источнику данных
conn.Open ConnString
'ссылка на подключение
Set cmd.ActiveConnection = conn
cmd.CommandText = "SELECT * from Товары"
'свойства набора записей
rs.CursorLocation = adUseClient
```

```
'открываем набор записей
rs.Open cmd, , adOpenStatic, adLockBatchOptimistic
'закрываем набор данных
rs.Close
'закрываем подключение
conn.Close
End Sub
```

10.4.2 Работа с набором записей

Для работы с набором записей предназначен объект Recordset. Набор записей, представляемых данным объектом состоит из записей и полей.

Методы объекта Recordset

Move – передвигает позицию текущей записи
Open – открывает набор записей

Requery – обновляет данные записей путем повторного выполнения запроса

Resync –обновляет данные в текущем наборе записей из основной базы данных

Save – сохраняет набор записей в файл

Seek – выполняет поиск индекса набора записей для быстрого поиска строки, в которой находится искомое значение

Delete – удаляет текущую запись или группу записей

Update – сохраняет любые изменения текущей записи

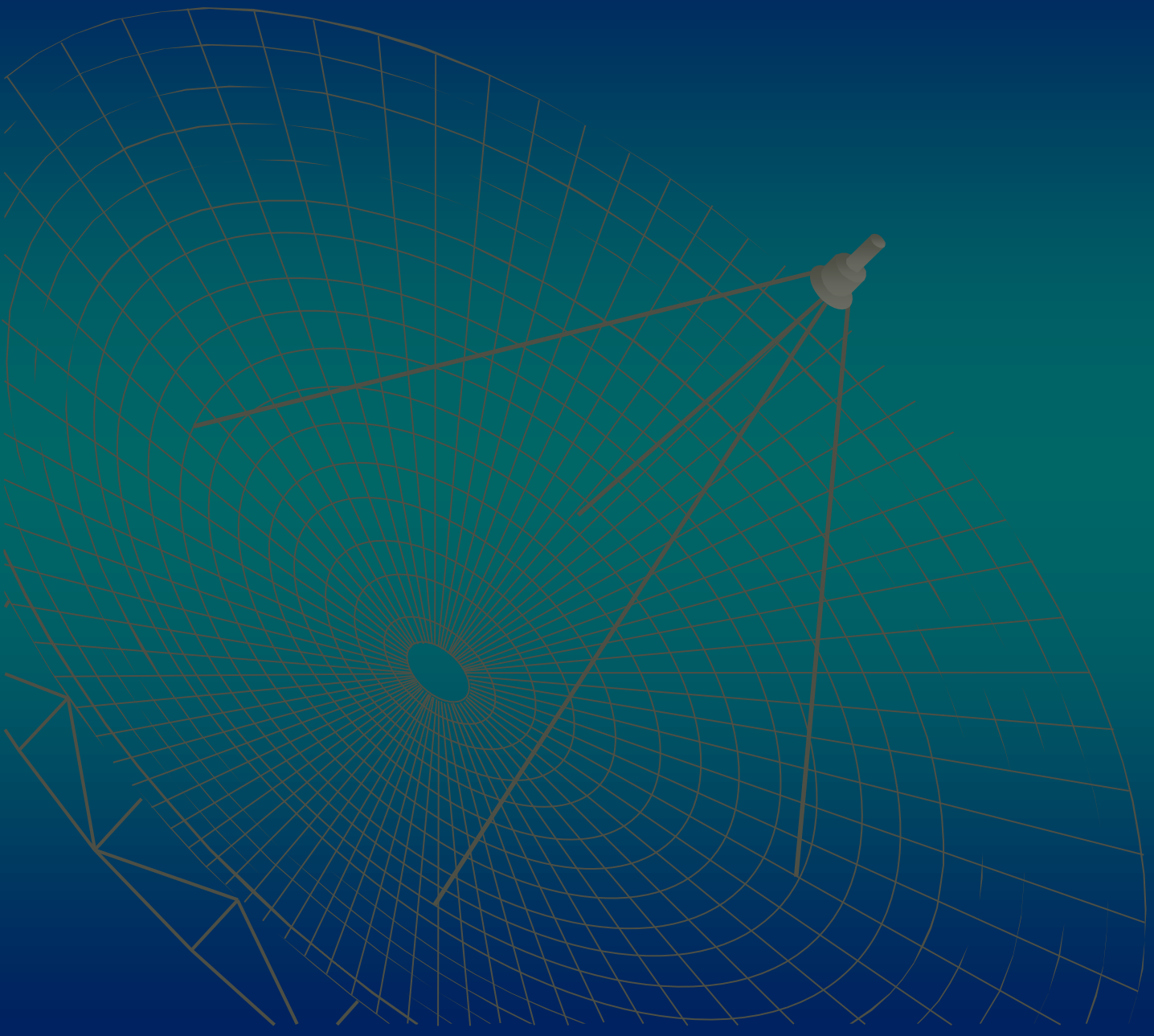
Свойства объекта Recordset

BOF – имеет значение true, если текущая запись находится перед первой записью

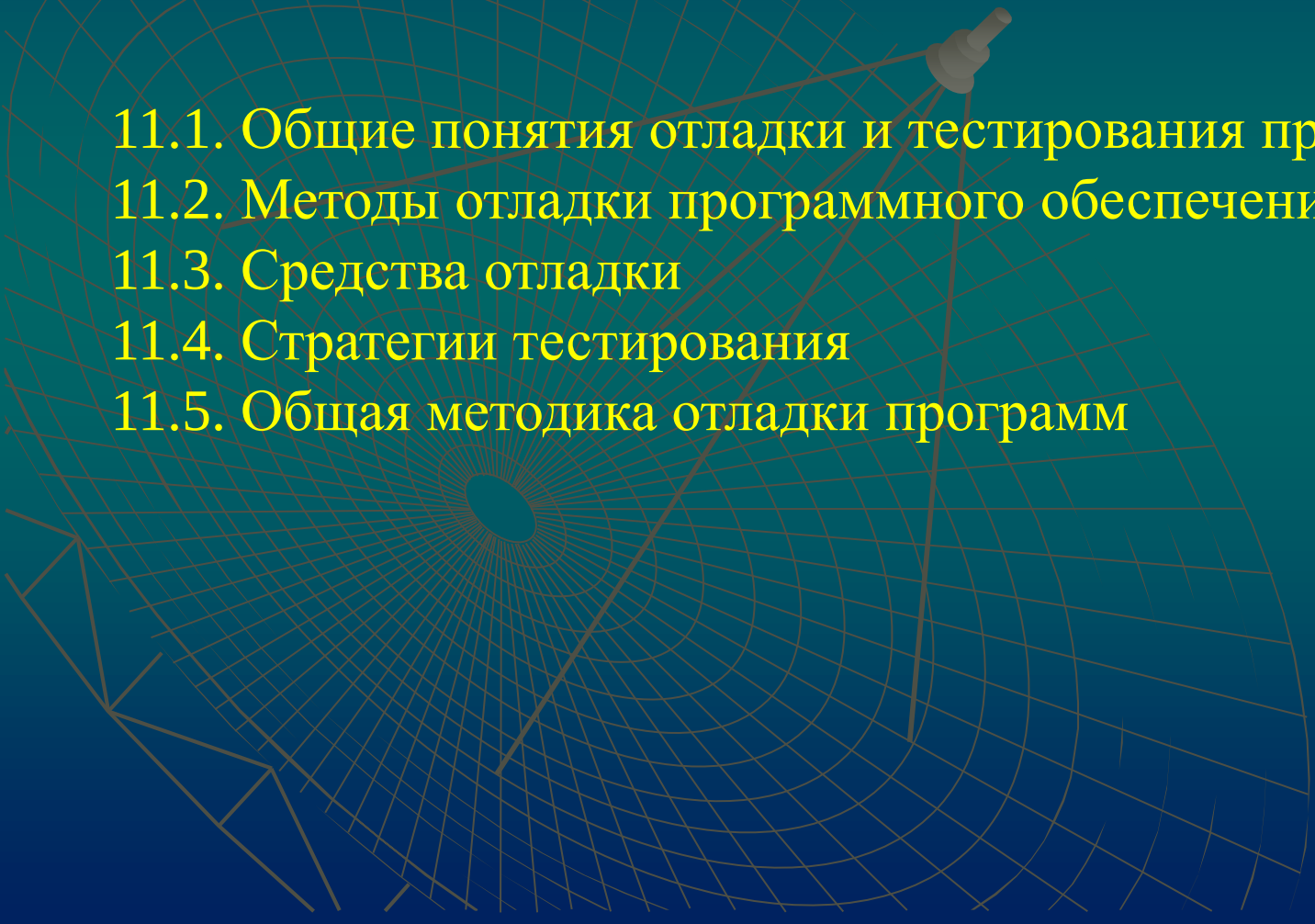
EOF – имеет значение true, если текущая запись находится после последней записи

RecordCount – определяет количество записей

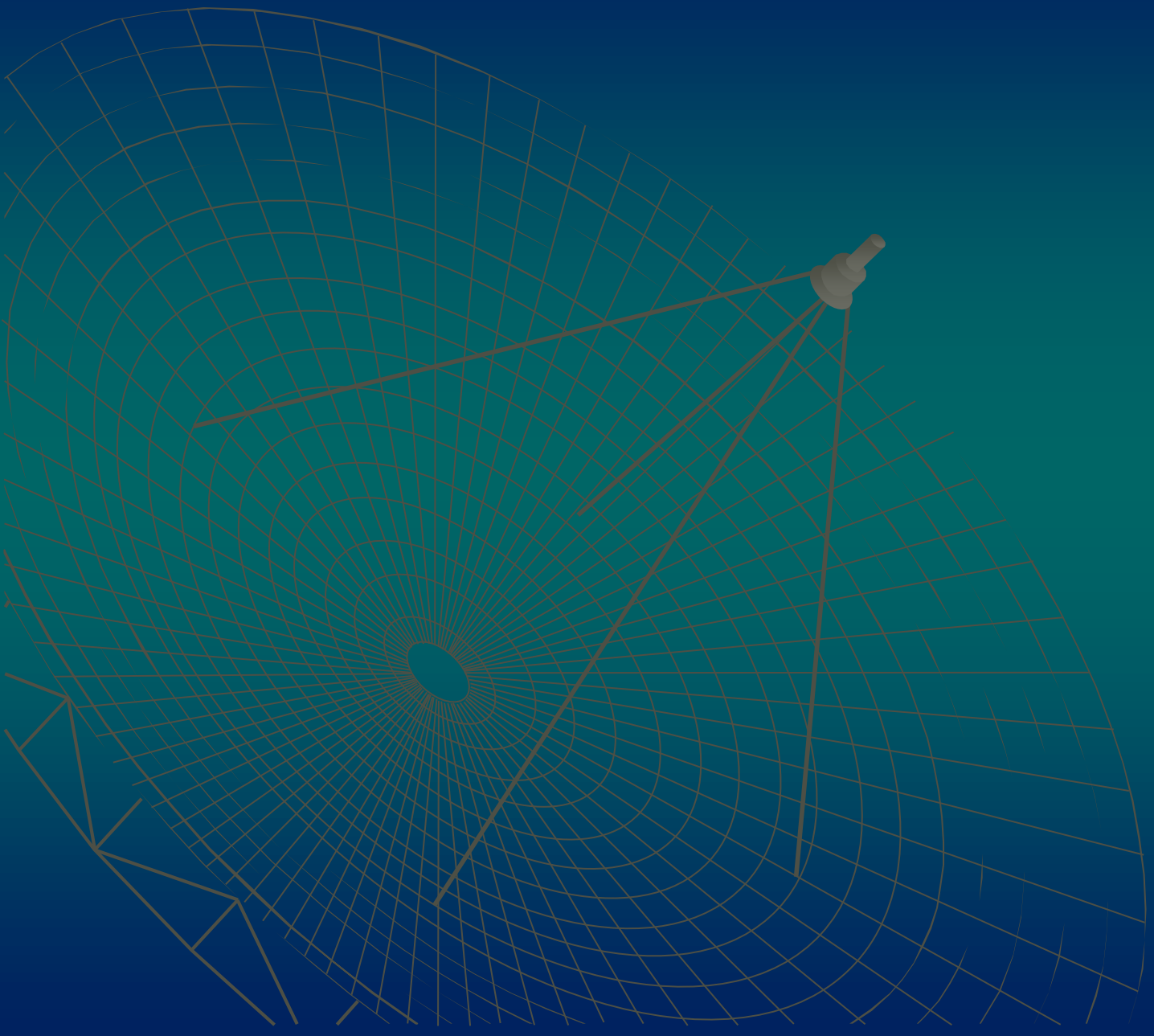
Source – определяет источник данных для записей



Глава 11. Методы отладки программ



11.1. Общие понятия отладки и тестирования программ
11.2. Методы отладки программного обеспечения
11.3. Средства отладки
11.4. Стратегии тестирования
11.5. Общая методика отладки программ



11.1. Общие понятия отладки и тестирования программ

Отладка программ - это процесс локализации и исправления ошибок, обнаруженных при тестировании программного обеспечения.

Для исправления ошибки необходимо определить ее причину, т.е. определить оператор программы или фрагмент, содержащие ошибку.

Локализацией называют процесс определения оператора программы, выполнение которого вызвало нарушение нормального вычислительного процесса.

Сложность отладки обусловлена следующими причинами:

- отладка требует знаний специфики управления используемыми техническими средствами, операционной системы, среды и языка программирования, реализуемых процессов, специфики различных ошибок, методик отладки и соответствующих программных средств;
- отладка психологически дискомфортна, так как необходимо искать собственные ошибки в условиях ограниченного времени;
- возможное взаимное влияние ошибок в разных частях программы;
- отсутствие четко сформулированных методик отладки.

Все ошибки можно разделить на три большие группы:

Классификация ошибок по этапу обработки программы

- ***синтаксические ошибки*** - ошибки, фиксируемые компилятором (транслятором, интерпретатором) при выполнении синтаксического и частично семантического анализа программы;
- ***ошибки компоновки*** - ошибки, обнаруженные компоновщиком (редактором связей) при объединении модулей программы;
- ***ошибки выполнения*** - ошибки, обнаруженные операционной системой, аппаратными средствами или пользователем при выполнении программы.

11.1.1. Синтаксические ошибки

Синтаксические ошибки относят к группе самых простых, так как синтаксис языка, как правило, строго формализован, и ошибки сопровождаются развернутым комментарием с указанием ее местоположения.

Чем лучше формализованы правила синтаксиса языка, тем больше ошибок может обнаруживать компилятор и, соответственно, меньше ошибок будет обнаруживаться на следующих этапах. В связи с этим говорят о языках программирования с защищенным синтаксисом и с незащищенным синтаксисом.

Многие синтаксические ошибки «отлавливаются» редактором кода VBA еще в процессе ввода кода. Об обнаружении других сообщается в ходе компиляции и запуска программы. При этом компилятор VBA выдает информацию о том, в какой строке кода обнаружена ошибка, и в чем она заключается. Рекомендуется проверить данную строку по справке VBA.

11.1.2. Ошибки компоновки

Ошибки компоновки связаны с проблемами, обнаруженными при разрешении внутренних и внешних ссылок.

Ошибки вычислений: некорректные действия над неарифметическими переменными, некорректное использование целочисленной арифметики, некорректное преобразование типов, незнание приоритетов выполнения операций для арифметических и логических выражений;

Логические ошибки: неправильная реализация логики программы при кодировании, игнорирование особенностей или ограничений конкретного языка программирования;

Ошибки межмодульного интерфейса: игнорирование системных соглашений, нарушение типов и последовательности при передаче параметров, несоблюдение единства единиц измерения формальных и фактических параметров, нарушение области действия локальных и глобальных переменных.

11.1.3. Ошибки выполнения

К самой непредсказуемой группе относятся ошибки выполнения. Они могут иметь разную природу, и по-разному проявляться. Часть ошибок обнаруживается и документируется операционной системой или системой программирования.

Выделяют четыре способа проявления таких ошибок:

1) Появление сообщения об ошибке, зафиксированной системой контроля выполнения команд, например, переполнении разрядной сетки, ситуации «деление на ноль», нарушении адресации и т.п.;
2) Появление сообщения об ошибке, обнаруженной операционной системой, например, нарушении защиты памяти, попытке записи на устройства, защищенные от записи, отсутствии файла с заданным именем и т.п.;
3) «Зависание» компьютера, как «простое», когда удается завершить программу без перезагрузки операционной системы, так и «тяжелое», когда для продолжения работы необходима перезагрузка;
4) Несоответствие полученных результатов с ожидаемыми.

11.1.4. Общая характеристика ошибок

Большинство ошибок обнаруживается вообще без запуска программы - просто внимательным просматриванием текста.

Если отладка зашла в тупик и обнаружить ошибку не удается, лучше отложить программу. Когда глаз "замылен", эффективность работы упорно стремится к нулю.

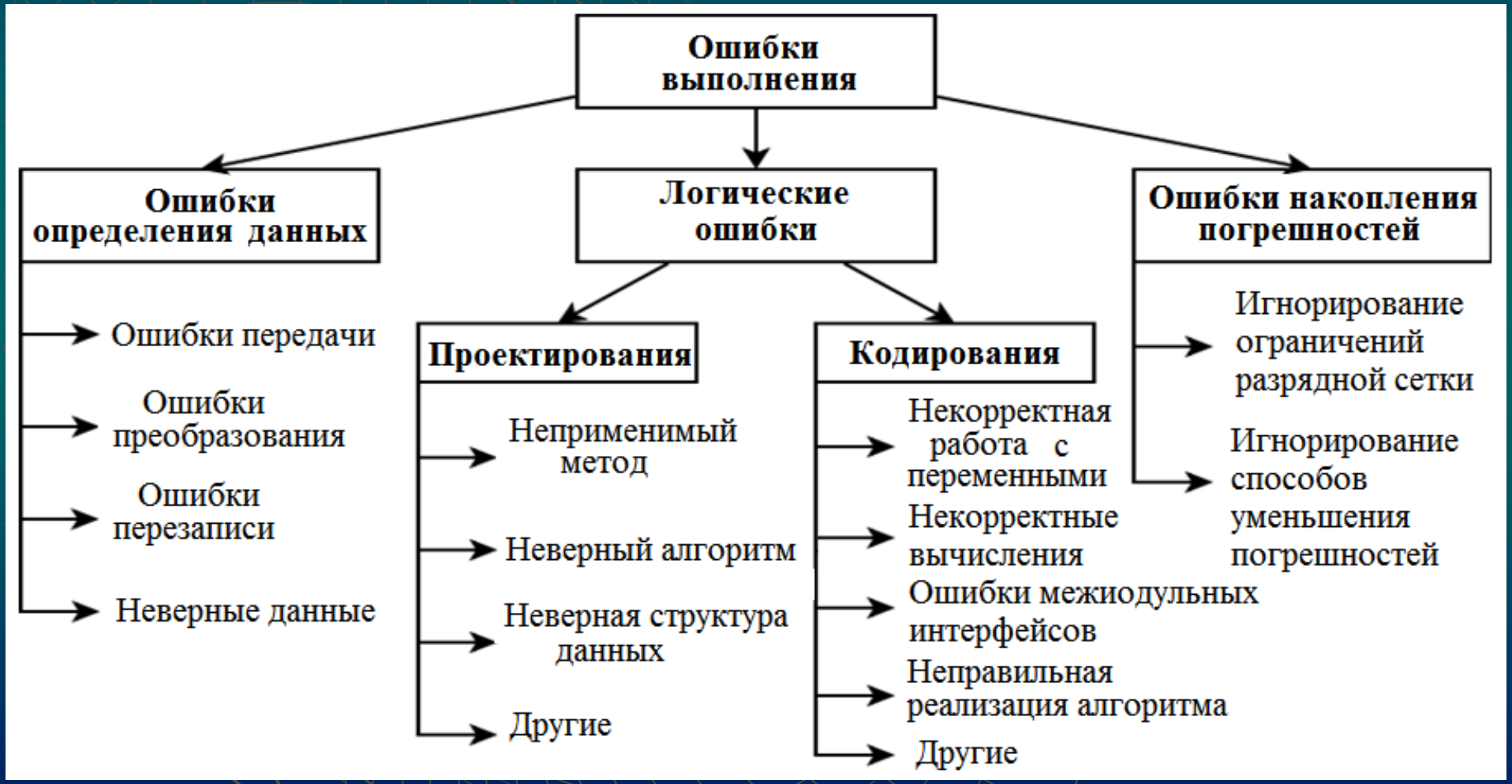
Чрезвычайно удобные вспомогательные средства - это отладочные механизмы среды разработки: трассировка, промежуточный контроль значений.

Экспериментирования типа «что будет, если изменить плюс на минус» нужно избегать всеми силами. Обычно это не дает результатов, а только больше запутывает процесс отладки, да еще и добавляет новые ошибки.

Главный прием обеспечения безошибочной работы программы - это ***ее тестирование***. При создании крупных программных продуктов на их тестирование часто уходит не меньше времени, чем на их создание.

Возможные причины ошибок:

- неверное определение исходных данных;
- логические ошибки;
- накопление погрешностей результатов вычислений.



Неверное определение исходных данных происходит, если возникают любые ошибки при выполнении операций ввода-вывода: ошибки передачи, ошибки преобразования, ошибки перезаписи и ошибки данных.

Использование специальных технических средств и программирование с защитой от ошибок позволяет обнаружить и предотвратить только часть этих ошибок.

Логические ошибки

1) могут следовать из ошибок, допущенных при проектировании (при выборе методов, разработке алгоритмов или определении структуры классов);
2) могут быть непосредственно внесены при кодировании модуля.

К ***ошибкам некорректного использования переменных*** относят:

- неудачный выбор типов данных;
- использование переменных до их инициализации;
- индексы, выходящие за границы определения массивов;
- нарушения соответствия типов данных переменных, массивов, объектов;
- неверное использование динамической памяти, адресной арифметики и т.п.

Накопление погрешностей результатов числовых вычислений возникает при некорректном отбрасывании дробных цифр чисел, некорректном использовании приближенных методов вычислений, игнорировании ограничения разрядной сетки представления чисел различных типов.

Сложность отладки увеличивается также вследствие влияния следующих факторов:

- опосредованного проявления ошибок;
- возможности взаимовлияния ошибок;
- получения внешне одинаковых проявлений разных ошибок;
- отсутствия повторяемости проявлений некоторых ошибок от запуска запуску (стохастические ошибки);
- возможности устранения внешних проявлений ошибок в исследуемой ситуации при внесении некоторых изменений в программу, например, при включении в программу диагностических фрагментов может аннулироваться или измениться внешнее проявление ошибок;
- написания отдельных частей программы разными программистами.

11.2.Методы отладки программного обеспечения

Отладка программы - это специальный этап в разработке программы, состоящий в выявлении и устранении программных ошибок, факт существования которых уже установлен.

Отладка программы предполагает обдумывание и логическое осмысление всей имеющейся информации об ошибке.

Большинство ошибок можно обнаружить по косвенным признакам посредством тщательного анализа текстов программ и результатов тестирования без получения дополнительной информации.

Для отладки используют различные методы:

1. Метод ручного тестирования;

2. Метод индукции;

3. Метод дедукции;

4. Метод обратного прослеживания;

5. Отладочный вывод.

11.2.1. Метод ручного тестирования

Это - самый простой и естественный способ данной группы. При обнаружении ошибки необходимо выполнить тестируемую программу вручную, используя различные тестовый наборы входных данных.

Рекомендации при тестировании

1) попытайтесь запустить программу при работе с большим количеством документов или когда не открыты документы;
2) проверьте, как работает программа, когда выделены разные элементы или группы элементов;
3) если предусматривается ввод информации, попробуйте специально передать программе неверные значения.
4) попробуйте прервать работу программы в самый неподходящий момент и потом вновь запустить ее;
5) проверьте, как ведет себя программа, когда пропадает сеть, заканчивается свободное место на диске, бумага в принтере;
6) проверьте работу программы под разными версиями Office и операционных систем;
7) попробуйте до запуска программы и во время работы переставлять системную дату и время, устанавливая различные значения.

11.2.2. Метод индукции

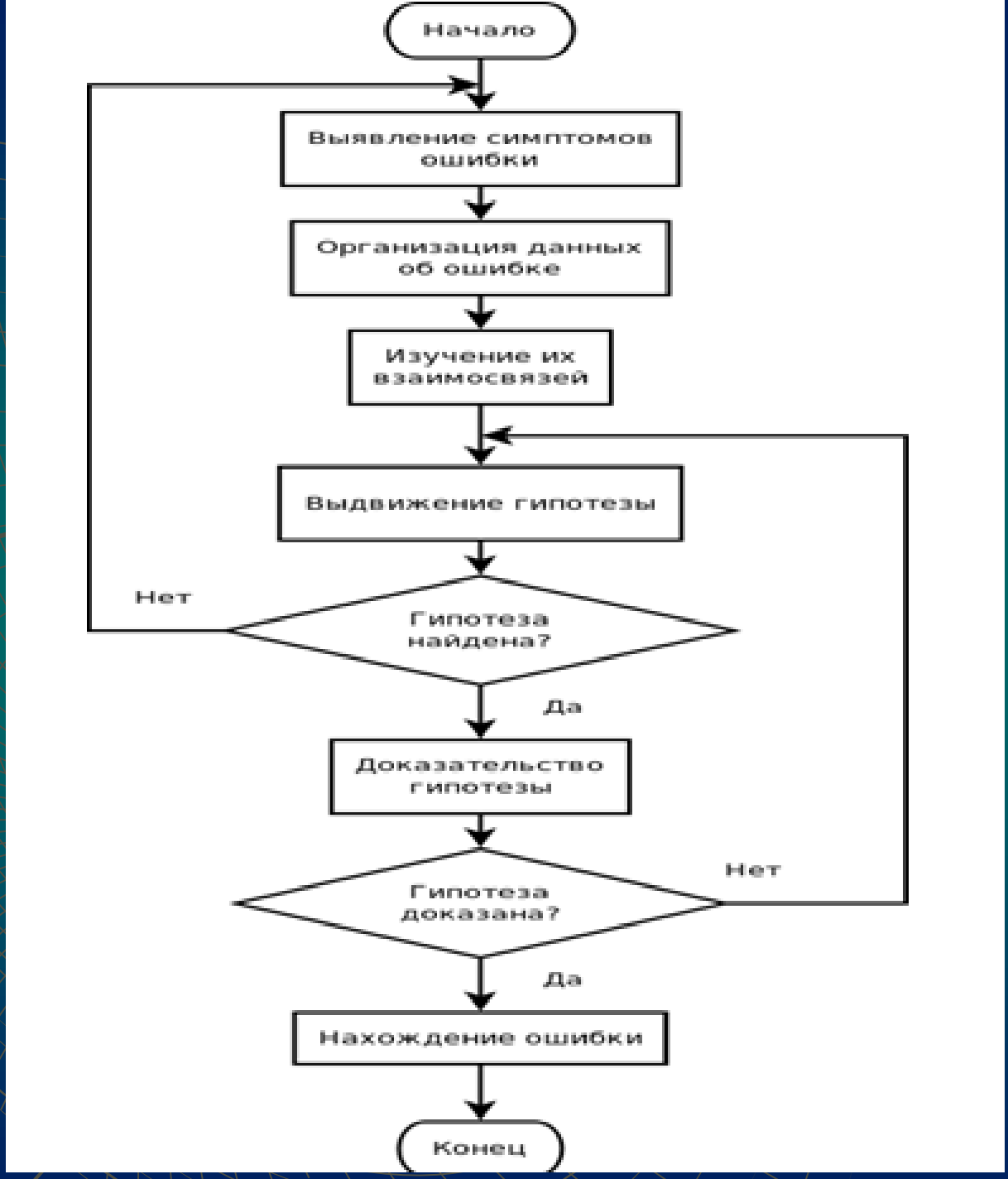
Метод основан на тщательном анализе симптомов ошибки, которые могут проявляться как неверные результаты вычислений или как сообщение об ошибке.

Если компьютер просто «***зависает***», то фрагмент проявления ошибки вычисляют, исходя из последних полученных результатов и действий пользователя.

Полученную таким образом информацию организуют и тщательно изучают, просматривая соответствующий фрагмент программы.

В результате этих действий выдвигают гипотезы об ошибках, каждую из которых проверяют.

Если гипотеза верна, то детализируют информацию об ошибке, иначе - выдвигают другую гипотезу.

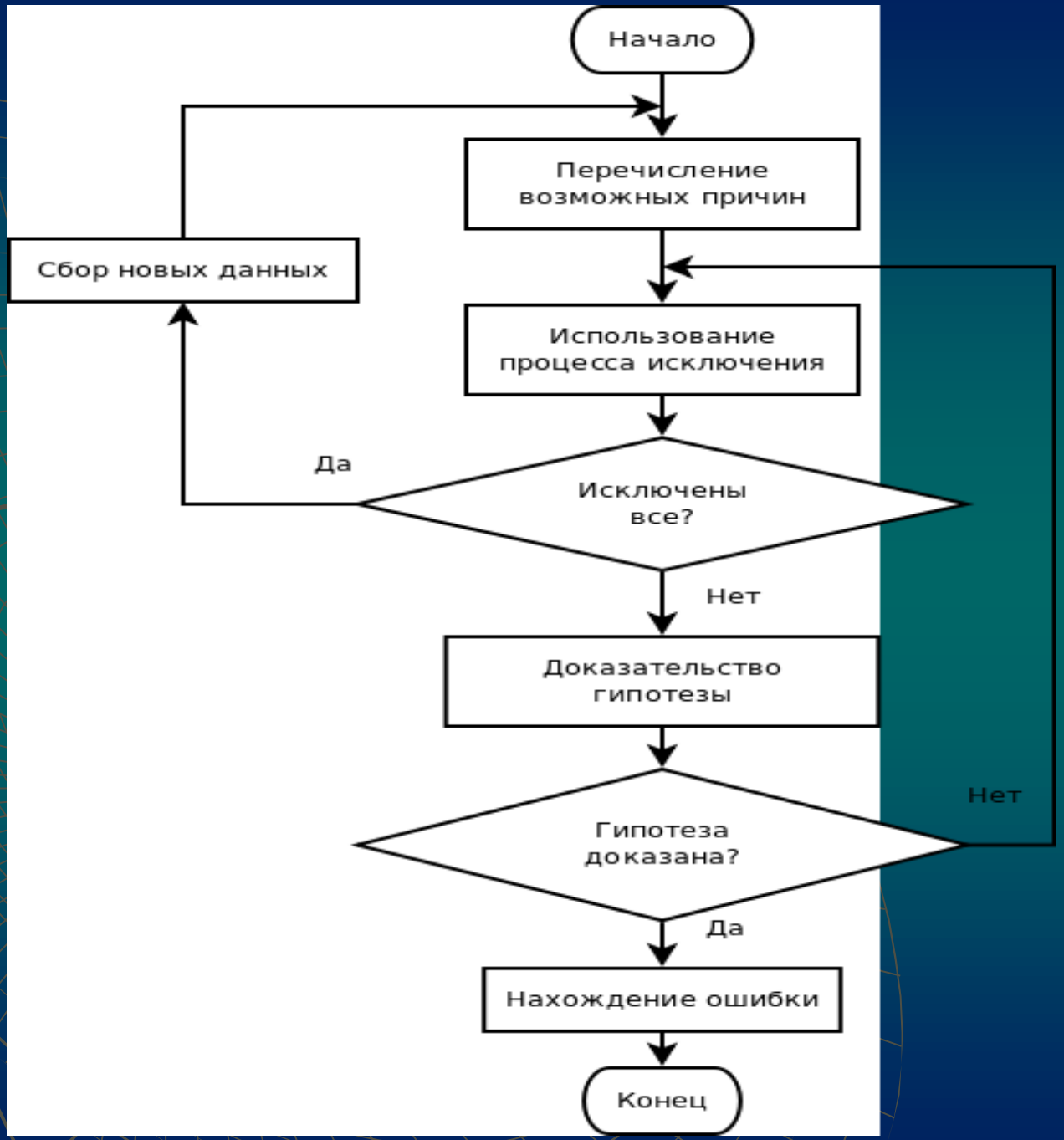

Начало
Выявление симптомов ошибки
Организация данных об ошибке
Изучение их взаимосвязей
Выдвижение гипотезы
Нет
Гипотеза найдена?
Да
Доказательство гипотезы
Нет
Гипотеза доказана?
Да
Нахождение ошибки
Конец

11.2.3. Метод дедукции

По методу дедукции вначале формируют множество причин, которые могли бы вызвать данное проявление ошибки. Затем анализируя причины, исключают те, которые противоречат имеющимся данным.

Если все причины исключены, то следует выполнить дополнительное тестирование исследуемого фрагмента. В противном случае наиболее вероятную гипотезу пытаются доказать.

Если гипотеза объясняет полученные признаки ошибки, то ошибка найдена, иначе - проверяют следующую причину.


Начало
Перечисление возможных причин
Использование процесса исключения
Исключены все?
Да
Нет
Сбор новых данных
Доказательство гипотезы
Гипотеза доказана?
Нет
Да
Нахождение ошибки
Конец

11.2.4. Метод обратного прослеживания

Для небольших программ эффективно применение метода обратного прослеживания.

Начинают с точки вывода неправильного результата. Для этой точки строится гипотеза о значениях основных переменных, которые могли бы привести к получению имеющегося результата.

Далее, исходя из этой гипотезы, делают предложения о значениях переменных в предыдущей точке. Процесс продолжают, пока не обнаружат причину ошибки.

Отладка начинается там, где впервые встретился неправильный результат. Затем работа программы прослеживается (мысленно или при помощи тестов) в обратном порядке, пока не будет обнаружено место возможной ошибки.

11.2.5. Отладочный вывод

Метод требует включения в программу дополнительного отладочного вывода в *узловых* точках. Узловыми считают точки алгоритма, в которых основные переменные программы меняют свои значения.

Например, отладочный вывод следует предусмотреть до и после завершения цикла изменения некоторого массива значений.

Если отладочный вывод предусмотреть в цикле, то будет выведено слишком много значений, в которых, как правило, сложно разбираться.

При этом предполагается, что, выполнив анализ выведенных значений, программист уточнит момент, когда были получены неправильные значения, и сможет сделать вывод о причине ошибки.

11.3. Средства отладки

Большинство современных сред программирования включают ***средства отладки***, которые обеспечивают максимально эффективную отладку. Они позволяют:

- ***выполнять программу по шагам***, причем как с заходом в подпрограммы, так и выполняя их целиком;
- ***предусматривать точки останова***;
- ***выполнять программу до оператора***, указанного курсором;
- ***отображать содержимое переменных*** при пошаговом выполнении;
- ***отслеживать поток сообщений*** и т.п.

11.3.1. Остановка программы

Один из основных приемов отладки программы - **возможность вовремя остановиться** в ходе выполнения для просмотра значения переменных, анализа значений переменных и функций, изменения хода выполнения программы.

Для этого предлагаются следующие возможности:

1) Запустить программу в ***режиме пошагового выполнения*** из меню ***Debug*** -> ***Step Into*** (<**F8**>). В этом случае программа будет переходить в режим паузы после выполнения каждого оператора;

2) Обозначить в программе ***точку останова*** (***breakpoint***), установив указатель в нужной строке и в меню ***Debug*** выбрав ***Toggle Breakpoint*** (<**F9**>) или щелкнув мышью по рамке слева от строки.

При запуске программа будет автоматически останавливаться последовательно на всех точках останова. Снятие точки останова - выполнить те же действия еще раз.

3) Использовать инструкции ***Stop*** или ***End*** в нужной строке программного кода.

4) Воспользоваться командой ***Break*** из меню ***Run*** или просто нажать на клавиши <**Ctrl**>+<**Break**>.

```
For i = 0 To N                              'массив реальных координат
  x(i) = a + i * s                          'Реальная координата x
  Y(i) = fx(x(i))                           'Реальная координата y
  If Y(i) > Ymax Then Ymax = Y(i)           'Максимальное значение y
  If Y(i) < Ymin Then Ymin = Y(i)           'Минимальное значение y
Next i

For i = 0 To N                              'преобразование координат
  H(i) = (x(i) - a) * M + 10                'Вертикальная координата
  V(i) = (-Y(i) + Ymax) * M + 10            'Горизонтальная координата
Next i

For i = 1 To N                              'построение графика функции
  ActiveSheet.Shapes.AddLine(H(i - 1), V(i - 1), H(i), V(i)) _
    .Name = "L" + Str(i)
  With ActiveSheet.Shapes("L" + Str(i))
    .Line.ForeColor.SchemeColor = 10
    .Line.Weight = 1.5
  End With
Next i

Stop

x1 = (a - a) * M + 10                       'Рисуем ось Ox
y1 = (0 + Ymax) * M + 10
x2 = (b - a) * M + 10
y2 = (0 + Ymax) * M + 10
ActiveSheet.Shapes.AddLine(x1, y1, x2, y2).Name = "Ось Ox"
With ActiveSheet.Shapes("Ось Ox")
  .Line.EndArrowheadStyle = msoArrowheadTriangle
  .Line.EndArrowheadLength = msoArrowheadLengthMedium
End With
```

Выполнение программы остановлено.

11.3.2. Действия в режиме паузы

В режим паузы обычно входят для того, чтобы предпринять какие-либо действия:

1) продолжить выполнение в пошаговом режиме (команда **Debug -> Step Into** или <F8>);

2) если в строке программы происходит вызов процедуры, которая уже отлажена и проблем вызвать не должна, то выполнить ее можно без остановок и перейти к следующему оператору (команда **Debug -> Step Over** или <Shift>+<F8>);

3) при заходе в процедуру и выполнения каких-либо действий можно срочно довести ее выполнение до конца (команда **Debug -> Step Out** или <Ctrl>+<Shift>+<F8>);

4) при желании исполнить код не пошагово, а участками (например, чтобы не проводить пошаговое выполнение больших циклов), можно щелкнуть правой кнопкой мыши по нужному участку кода и в контекстном меню выбрать **Run to Cursor** (команда **Debug -> Run to Cursor** или <Ctrl>+<F8>). Программа выполнится до выбранного фрагмента и остановится в месте курсора;

5) если нужно пропустить («перепрыгнуть») через участок кода, вызывающий ошибку, то можно выполнить команду **Debug -> Set Next Statement** (<Ctrl>+<F9>), а затем перетащить желтую отметку по левой границе вниз (или вверх);

6) альтернативный способ пропуска фрагмента кода - выделить часть кода и при помощи панели инструментов **Edit** его закомментировать (***после придется снимать комментарии!***);

7) если в процессе просмотра необходимо вернуться к месту остановки без долгих розысков, то в распоряжении программиста команда **Show Next Statement**;

8) для продолжения выполнения кода после остановки, можно воспользоваться командой **Run -> Continue** (<F5>);

9) прекратить выполнение программы можно при помощи команды **Run -> Reset** (<Alt>+<F4>) или одноименной кнопкой на панели инструментов **Standard**.

11.3.3. Просмотр текущих значений

В режиме паузы **для получения текущих значений** переменных можно воспользоваться **подсказками** в коде программы.

Для того, чтобы получить информацию о текущем значении переменной, достаточно навести на нее указатель мыши (если включен параметр по умолчанию **Tools** -> **Options** -> **Auto Data Tips**).

```
ActiveSheet.Shapes.AddTextbox(msoTextOrientationHorizontal, _
    (b - a) * M + 10, 2 + Ymax * M + 10, 20, 10).Select
   With Sel M = 40 n
       .Characters.Text = "x"
       .Characters.Font.Size = 8
       .ShapeRange.Line.Visible = msoFalse
       .ShapeRange.Fill.Visible = msoFalse
  End With

x1 = (0 - a) * M + 10                    'Рисуем ось Oy
y1 = (-Ymin + Ymax) * M + 10
x2 = (0 - a) * M + 10
y2 = (-Ymax + Ymax) * M + 10
ActiveSheet.Shapes.AddLine(x1, y1, x2, y2).Name = "Ось Oy"
With ActiveSheet.Shapes("Ось Oy")
  .Line.EndArrowheadStyle = msoArrowheadTriangle
  .Line.EndArrowheadLength = msoArrowheadLengthMedium
End With
```

Для просмотра информации об области видимости и типе данных переменной, необходимо установить на нее курсор вода текста и выбрать команду **Edit** -> **Quick Info** (<Ctrl>+<I>).

```
ActiveSheet.Shapes.AddTextbox(msoTextOrientationHorizontal, _
    (b - a) * M + 10, 2 + Ymax * M + 10, 20, 10).Select
          Implicit a As Variant
    With Selection
        .Characters.Text = "x"
        .Characters.Font.Size = 8
        .ShapeRange.Line.Visible = msoFalse
        .ShapeRange.Fill.Visible = msoFalse
    End With
```

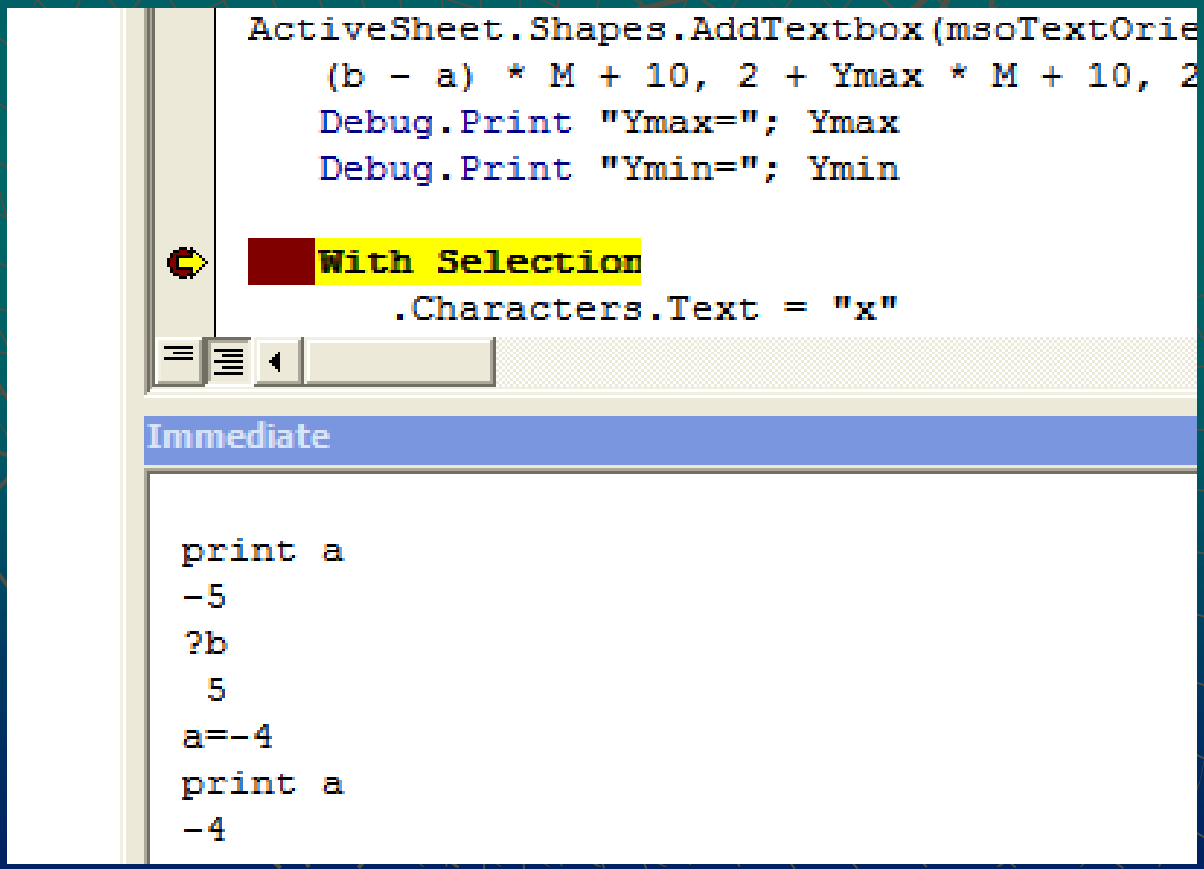
Если выяснено, что переменной (свойству объекта) присвоено неверное значение, то можно исправить соответствующую строку, вернуться назад при помощи команды **Set Next Statement** (<Ctrl>+<F9>) и продолжить выполнение программы с исправленным значением.

11.3.4. Окно Immediate

В окне **Immediate** можно просматривать или изменять значения переменных и свойств объекта.

Вызывается окно с помощью меню **View** или <Ctrl>+<G>.

Для просмотра используется инструкция ***print*** или просто вопросительный знак, а для изменения значения оператор присваивания.



Чтобы не печатать в окне **Immediate** выражения и имена переменных, фрагменты кода можно перетащить в окно **Immediate** из окна редактора кода с нажатой клавишей <Ctrl> .

Вывод в окно **Immediate** также можно производить при помощи метода Print объекта **Debug** из кода программы.

```
ActiveSheet.Shapes.AddTextbox(msoTextOrientatio
    (b - a) * M + 10, 2 + Ymax * M + 10, 20, 10)
    Debug.Print "Ymax="; Ymax
    Debug.Print "Ymin="; Ymin

    With Selection
        .Characters.Text = "x"
```

Immediate

```
Ymax= 2,99997061965211
Ymin=-2,99997621991776
```

В окне **Immediate** можно также вызывать любые операторы, процедуры и функции или методы объектов.

```
With Selection
    .Characters.Text = "x"
```

Immediate

```
?2*3
 6
?3*sin(5)/3
-0,958924274663138
?(b-a)/n
 0,0225
```

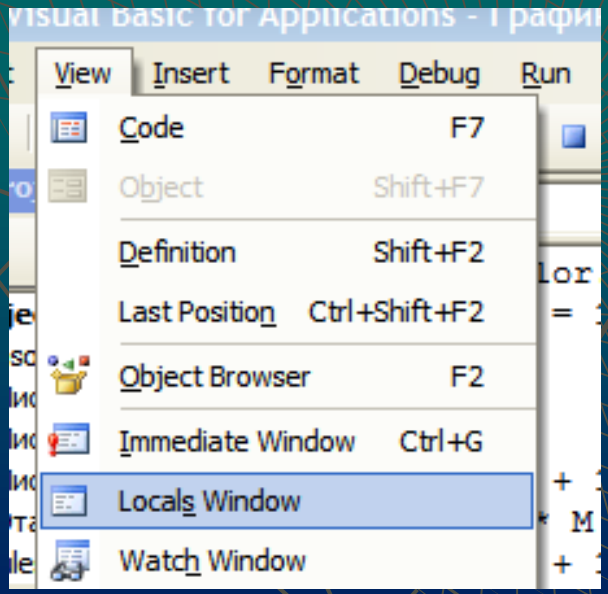
11.3.5. Окно Locals

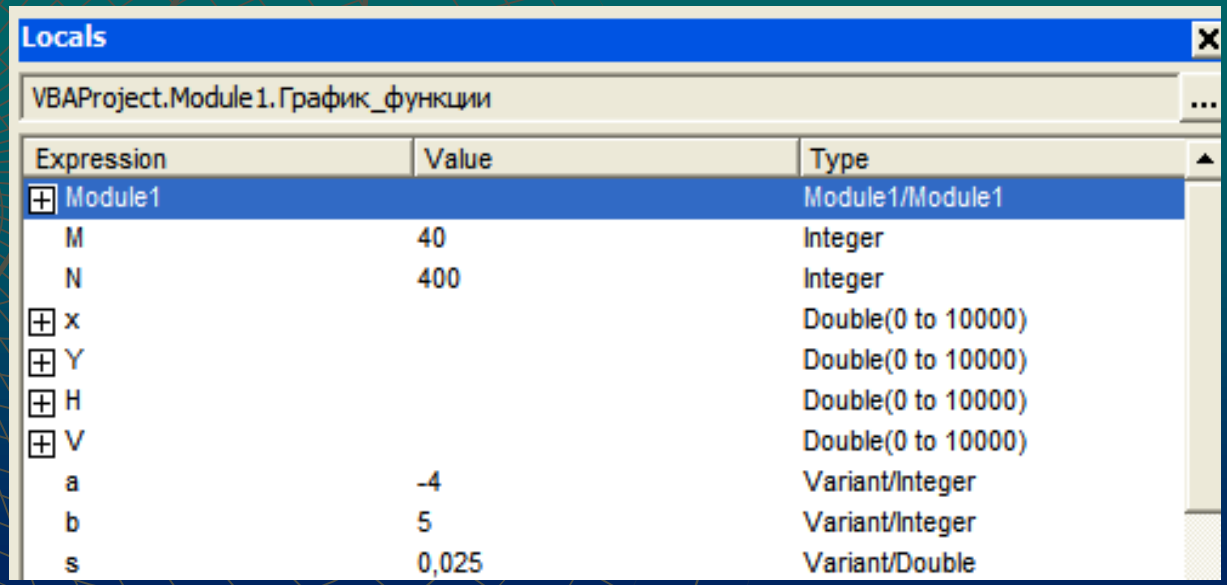
В ходе тестирования программы часто необходимо просмотреть и значения всех переменных и свойств объектов для последующего анализа данных и их редактирования.

В этом случае отображать каждое свойство или значение переменной в окне **Immediate** неэффективно. Гораздо удобнее в этой ситуации использовать окно **Locals**.

В окне **Locals** выводятся значения ***всех переменных и свойств объектов***, доступных в настоящий момент.

Вызывается окно с помощью меню **View**.

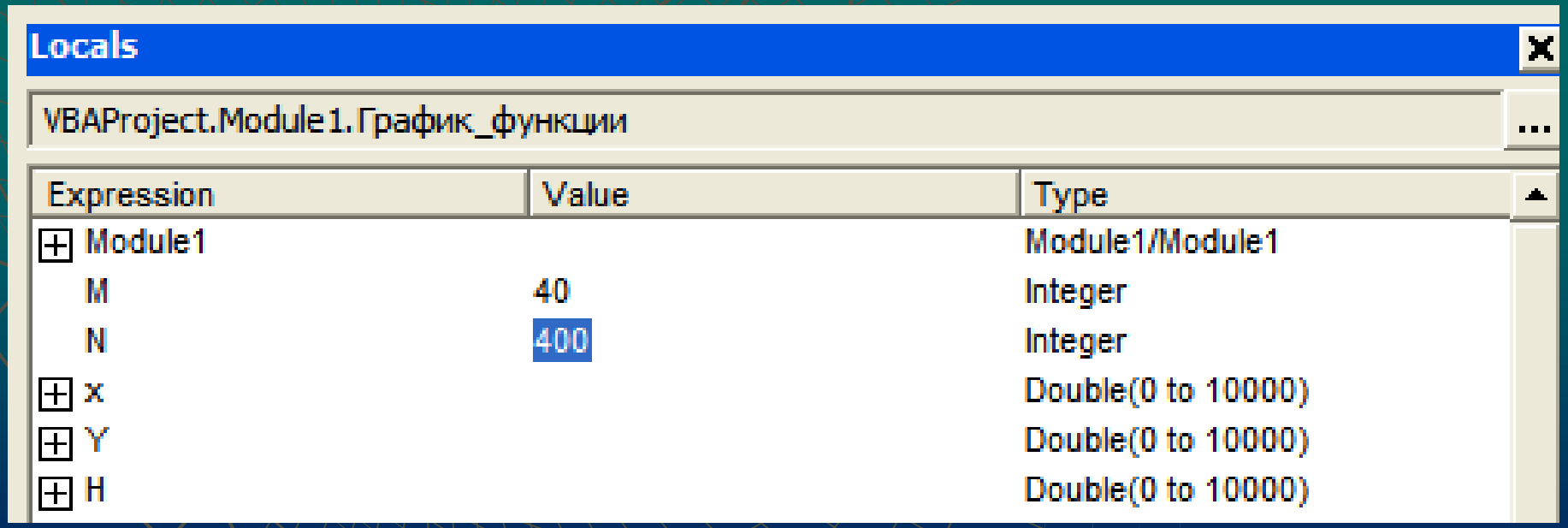




Чтобы поменять значение переменной или свойство объекта, необходимо выделить нужную строку в окне **Locals**, а потом в этой строке мышью выделить в столбце **Value** значение.

Затем можно впечатать поверх старого новое значение. Если значение должно быть строковым, то при необходимо заключить его в кавычки (как в коде), а если это значение даты - то в символы **#**.

Через окно **Locals** можно также менять значения для элементов массивов и коллекций.

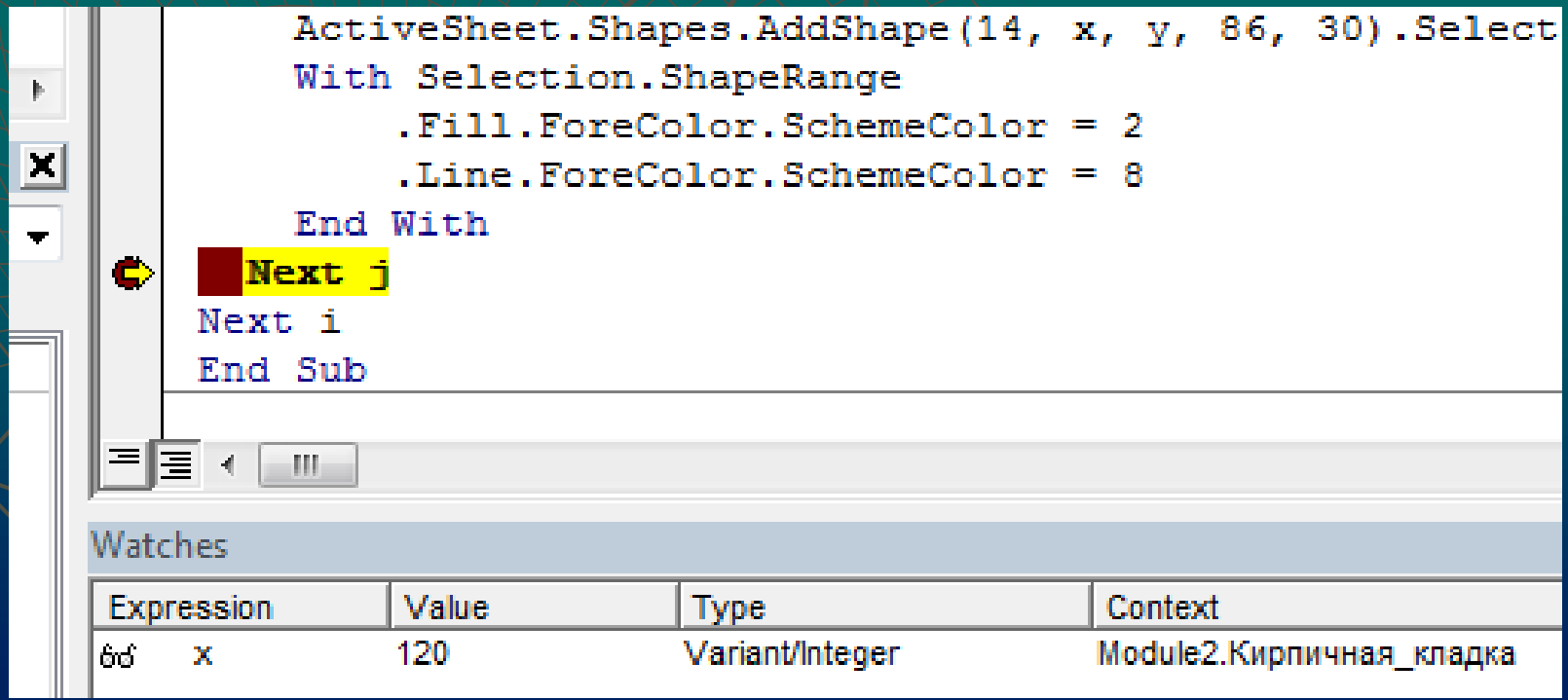


11.3.6. Окно Watches

Окно **Watches** позволяет контролировать ход выполнения программы, выполняя «наблюдение» в соответствии с заданными условиями.

При работе с окном **Watches** вначале потребуется определить значение, которое послужит сигналом к тому, чтобы вмешаться в ход выполнения программы. Это значение называется ***контролируемым выражением***.

Вызывается окно с помощью меню **View**.



Для создания контролируемого выражения можно:

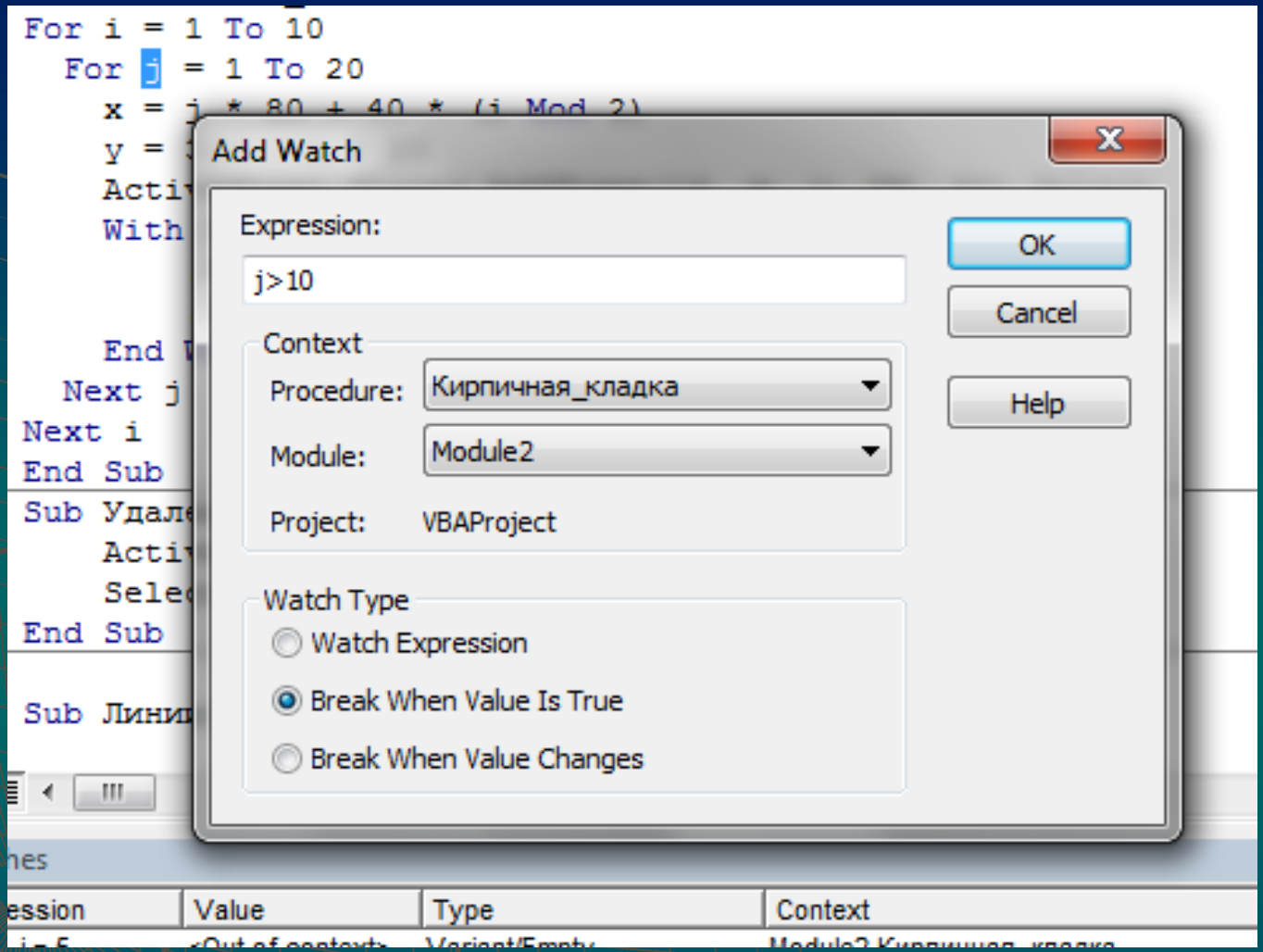
1) щелкнуть по переменному/свойству/выражению в окне редактора кода правой кнопкой мыши и в контекстном меню выбрать **Add Watch**. В диалоговое окно **Add Watch** будет автоматически помещено данное выражение;

2) воспользоваться командой **Add Watch** из меню **Debug**;

3) воспользоваться командой **Quick Watch** из меню **Debug**. В этом случае в окно **Watch** будет помещено готовое выражение. В качестве условия для "срабатывания" в него будет помещено текущее значение переменной/свойства;

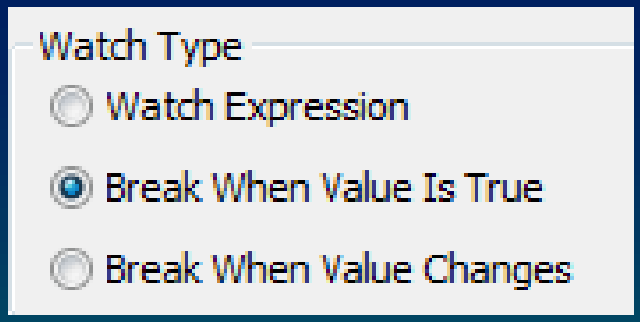
4) перетащить выражение из кода в окно **Watches** (его можно, как и все остальные, открыть через меню **View**).

В результате появится окно **Add Watch**.



В нем можно написать контролируемое выражение, чтобы оно возвращало ***true*** или ***false*** как в конструкции ***IF***, выбрать «область действия» выражения (в виде процедуры или модуля), а также принять главное решение: что делать в ходе наблюдения.

Для принятия решения всего три варианта:



1) Возможность изменения значения в столбце **Value** в окне **Watches**;

2) Перевод программы в режим паузы, если контролируемое выражение приняло значение true);

3) Перевод программы в режим паузы, если значение контролируемого выражения изменилось.

```
    x = j * 80 + 40 * (i Mod 2)
    y = 300 - i * 23
    ActiveSheet.Shapes.AddShape(14, x, y, 86, 30).Select
    With Selection.ShapeRange
        .Fill.ForeColor.SchemeColor = 2
```

Watches

Expression	Value	Type	Context
i > 5	False	Variant/Boolean	Module2.Кирпичная_кладка
x	120	Variant/Integer	Module2.Кирпичная_кладка
y	277	Variant/Integer	Module2.Кирпичная_кладка

11.4. Стратегии тестирования

1.) Тестирование программы как "черного ящика".

Мы знаем только о том, что делает программа, но даже не задумываемся о ее внутренней структуре. Задаем набор входных данных, получаем результаты, сверяем с эталонным. При этом обнаружить все ошибки мы можем только если составили тесты для всех возможных наборов данных.

"Черным ящиком" удобно тестировать небольшие подпрограммы.

2.) Тестирование программы как "белого ящика".

Здесь перед составлением теста мы изучаем логику программы, ее внутреннюю структуру. Тестирование будет считаться удачным, если проверяет программу по всем направлениям. Однако, как мы уже говорили, это требует огромного количества тестов.

На практике мы, как всегда, совместно используем оба принципа.

3.) Тестирование программ модульной структуры.

Мы снова возвращаемся к вопросу о структурном программировании. Структурированную программу мы будем тестировать частями. При этом нам нужно:

- строить набор тестов;
- комбинировать модули для тестирования.

Такое комбинирование может строиться двумя способами: Пошаговое тестирование - тестируем каждый модуль, присоединяя его к уже оттестированным. При этом можем соединять части программы сверху вниз (нисходящий способ) или снизу вверх (восходящий).

4.) Монолитное тестирование - каждый модуль тестируется отдельно, а затем из них формируется готовая рабочая программа и тестируется уже целиком.

Чтобы протестировать отдельный модуль, нужен модуль-драйвер (всегда один) и модули-заглушки (этих может быть несколько).

11.5. Общая методика отладки программ

Суммируя все сказанное выше, можно предложить следующую методику отладки программного обеспечения, написанного на универсальных языках программирования:

1 этап - изучение проявления ошибки - если выдано какое-либо сообщение или выданы неправильные или неполные результаты, то необходимо их изучить и попытаться понять, какая ошибка могла так проявиться. При этом используют индуктивные и дедуктивные методы отладки. В результате выдвигают версии о характере ошибки, которые необходимо проверить. Если ошибка не найдена или система просто "зависла", переходят ко второму этапу.

2 этап - локализация ошибки - определение конкретного фрагмента, при выполнении которого произошло отклонение от предполагаемого вычислительного процесса. Локализация может выполняться:

- путем отсечения частей программы, причем, если при отсечении некоторой части программы ошибка пропадает, то это может означать как то, что ошибка связана с этой частью, так и то, что внесенное изменение изменило проявление ошибки;

- с использованием отладочных средств, позволяющих выполнить интересующий нас фрагмент программы в пошаговом режиме и получить дополнительную информацию о месте проявления и характере ошибки, например, уточнить содержимое указанных переменных.

При этом если были получены неправильные результаты, то в пошаговом режиме проверяют ключевые точки процесса формирования данного результата. Как подчеркивалось выше, ошибка не обязательно допущена в том месте, где она проявилась. Если в конкретном случае это так, то переходят к следующему этапу.

3 этап - определение причины ошибки - изучение результатов второго этапа и формирование версий возможных причин ошибки.

4 этап - исправление ошибки - внесение соответствующих изменений во все операторы, совместное выполнение которых привело к ошибке.

5 этап - повторное тестирование - повторение всех тестов с начала, так как при исправлении обнаруженных ошибок часто вносят в программу новые.

Следует иметь в виду, что процесс отладки можно существенно упростить, если следовать основным рекомендациям структурного подхода к программированию:

- программу наращивать "сверху-вниз", от интерфейса к обрабатывающим подпрограммам, тестируя ее по ходу добавления подпрограмм;
- выводить пользователю вводимые им данные для контроля и проверять их на допустимость сразу после ввода;
- предусматривать вывод основных данных во всех узловых точках алгоритма (ветвлениях, вызовах подпрограмм).

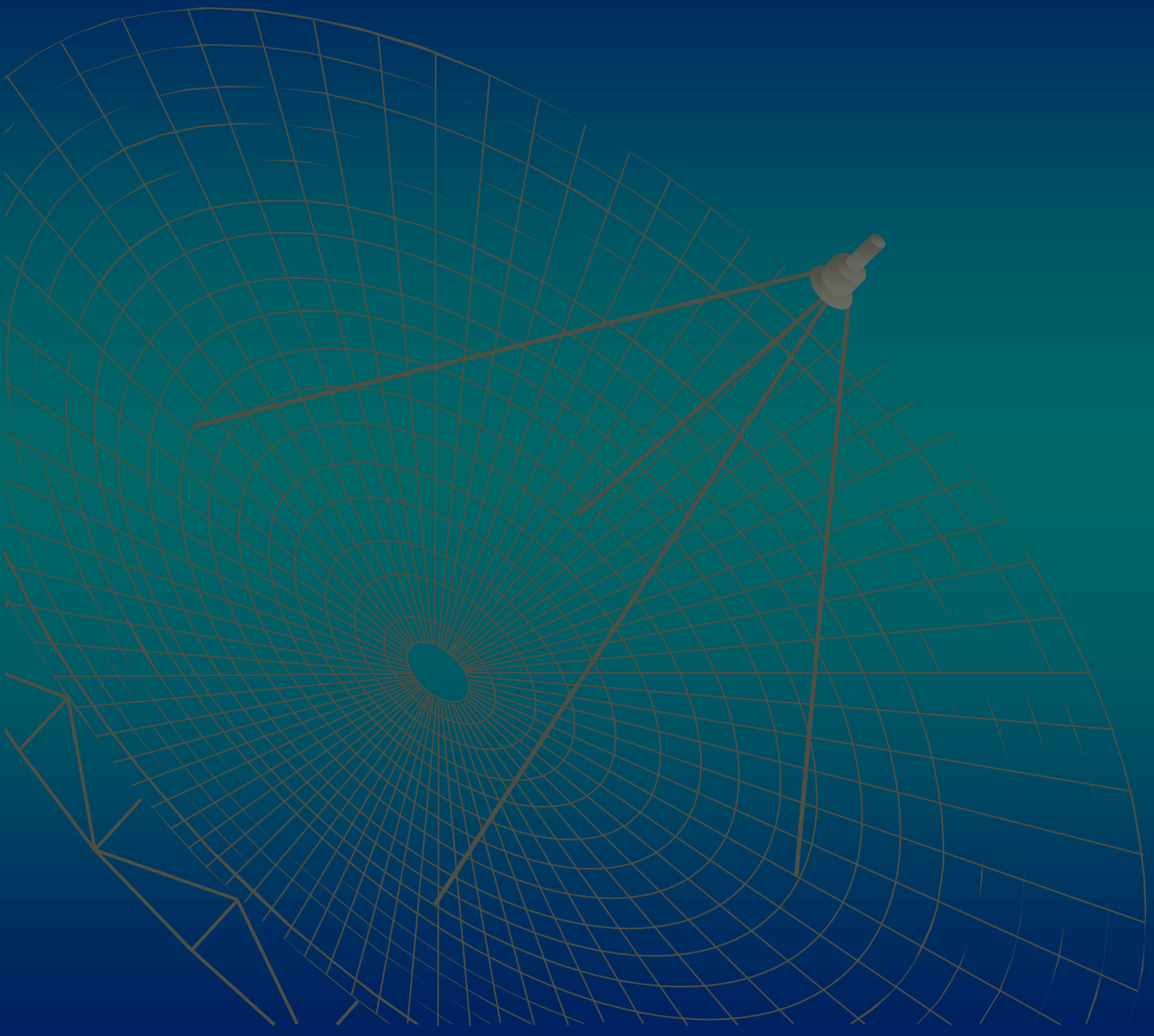
Кроме того, следует более тщательно проверять фрагменты программного обеспечения, где уже были обнаружены ошибки, так как вероятность ошибок в этих местах по статистике выше. Это вызвано следующими причинами.

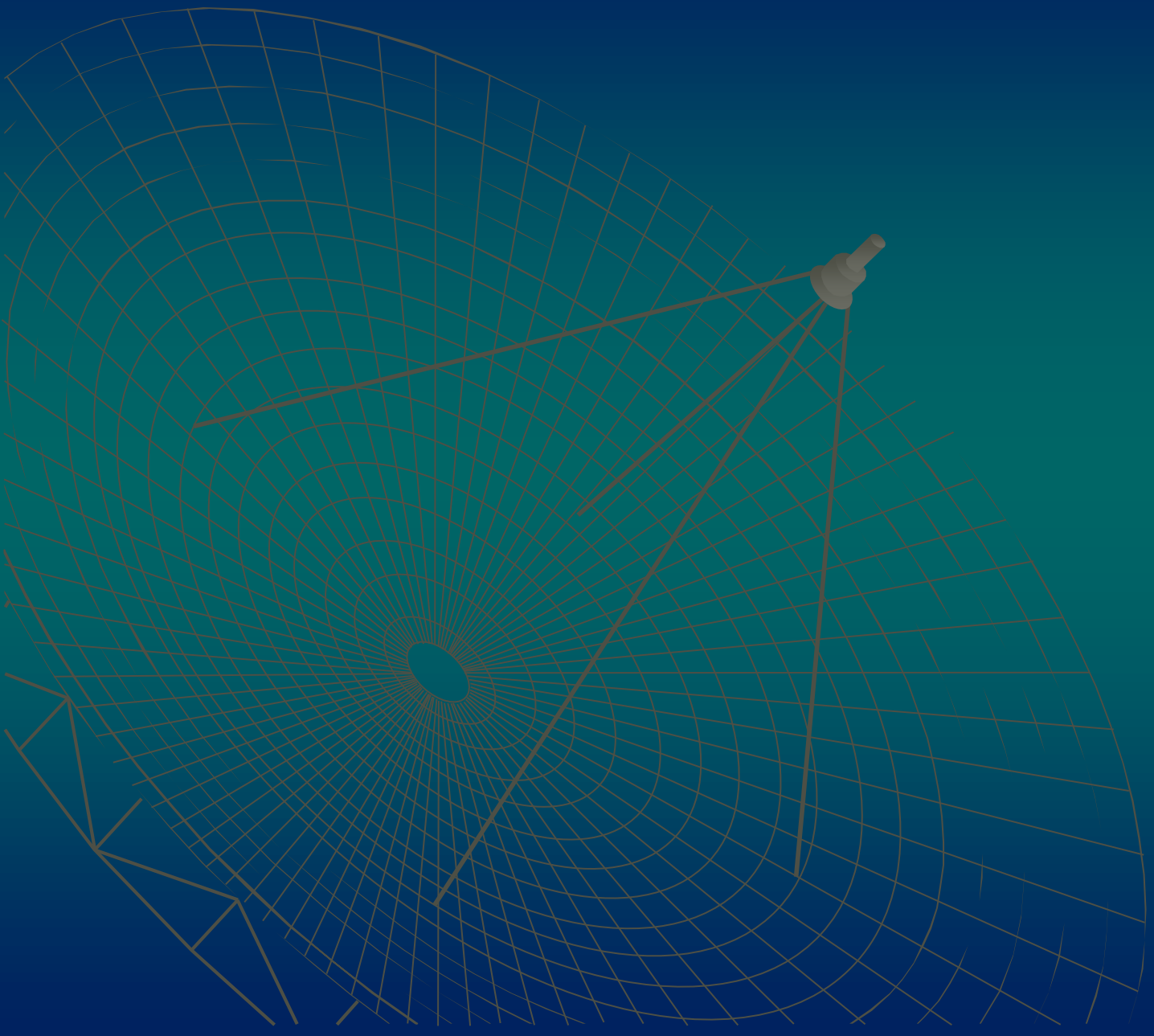
Во-первых, ошибки чаще допускают в сложных местах или в тех случаях, если спецификации на реализуемые операции недостаточно проработаны.

Необходимо отметить, что проще всего обычно искать ошибки определения данных и ошибки накопления погрешностей: их причины локализованы в месте проявления. Логические ошибки искать существенно сложнее. Ошибки кодирования бывают как простые, например, использование неинициализированной переменной, так и очень сложные, например, ошибки, связанные с затиранием памяти.

Затиранием памяти называют ошибки, приводящие к тому, что в результате записи некоторой информации не на свое место в оперативной памяти, затираются фрагменты данных или даже команд программы. Ошибки подобного рода обычно вызывают появление сообщения об ошибке.

Именно в этом случае часто прибегают к удалению из программы частей, хотя это и не обеспечивает однозначного ответа на вопрос, в какой из частей программы находится ошибка. Эффективнее попытаться определить операторы, которые записывают данные в память не по имени, а по адресу, и последовательно их проверить. Особое внимание при этом следует обращать на корректное распределение памяти под данные.





Список литературы

1. Астафьев Е.В., Информатика и математика: курс лекций. В 2 ч. / Е.Р. Астафьев, Е.В. Михайленко. – Краснодар: Краснодарский юридический институт МВД России, 2001. – Ч.2: Математическое обеспечение решения оперативно-служебных задач. –90 с.

2. Астафьев, Е.Р. Информатика и математика. Часть 1. Математика: учеб. пособие / Е.Р. Астафьев, П.В. Арбузов, С.В. Гуде, Е.В. Михайленко. – Краснодар: Краснодарский университет МВД России, 2006. –300 с.

3. Астафьев, Е.Р. Системное программное обеспечение: учеб. пособие / Е.Р. Астафьев, Е.В, Михайленко. - Краснодар: Краснодарская академия МВД России, 2004. – 64 с.

4. Михайленко, Е.В. Информатика и математика: учеб. пособие / под общ. ред. Е.В. Михайленко / Е.В. Михайленко, С.В. Гуде, И.Н. Старостенко, Ю.Н. Сопильняк, П.В. Арбузов. – Краснодар: Краснодарский университет МВД России, 2007. Ч. 2: Информатика. – 330 с.

5. Михайленко, Е.В. Информатика и ЭВМ в психологии: курс лекций / Е.В. Михайленко, И.Н. Старостенко. – Краснодар: Краснодарский университет МВД России, 2009. – 200 с.

6. Михайленко, Е.В. Информатика: учеб. пособие / Е.В. Михайленко, И.Н. Старостенко, Ю.Н. Сопильняк; под общ. ред. Е.В. Михайленко. –- Краснодар: Краснодарский университет МВД России, 2010. – 336 с.

7. Михайленко, Е.В. Информационная безопасность: курс лекций / Е.В. Михайленко, И.Н. Старостенко. - Краснодар: Краснодарская академия МВД России, 2005. – 102 с.

8. Михайленко, Е.В. Математика и информатика: курс лекций / Е.В. Михайленко, И.Н. Старостенко. – Краснодар: Краснодарский университет МВД России, 2009. – 420 с.

9. Михайленко, Е.В. Организация ветвления и циклов: лекция / Е.В. Михайленко. – Краснодар: Краснодарский университет МВД России, 2015. – 44 с.

Учебное издание

Михайленко Евгений Владимирович
Назаров Артур Карапетович

ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

Учебное пособие

В авторской редакции

ISBN 978-5-9266-1505-7

Подписано в печать 25.07.2019.
Объем: 303 Мб. Заказ 812.

Краснодарский университет МВД России.
350005, г. Краснодар, ул. Ярославская, 128.