Краснодарский университет МВД России

Е. В. Михайленко

ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ

Учебное пособие

Краснодар
2020

УДК 004.43
ББК 32.97
М69

Одобрено
редакционно-издательским советом
Краснодарского университета
МВД России

Рецензенты:

А. Н. Прокопенко, кандидат технических наук, доцент (Белгородский юридический институт МВД России имени И.Д. Путилина);

О. Ю. Бубнова, кандидат физико-математических наук (Нижегородская академия МВД России).

Михайленко Е. В.

М69 Введение в программирование : учебное пособие / Е. В. Михайленко. – Краснодар : Краснодарский университет МВД России, 2020. – 106 с.

ISBN 978-5-9266-1614-6

Характеризуются основные понятия программирования, алгоритмы, ветвление, циклические структуры, обработка массивов и файлов, организация функций и подпрограмм.

Для профессорско-преподавательского состава, курсантов и слушателей образовательных организаций МВД России.

УДК 004.43
ББК 32.97

ISBN 978-5-9266-1614-6

Раздел 1. Алгоритмы

1.1. Понятие алгоритма

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

Алгоритм включает 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.

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, инструкций), образующих прерывную (дискретную) структуру алгоритма. Только выполнив требования одного предписания, можно приступить к выполнению следующего.

Детерминированность (*определенность*). Будучи понятным, алгоритм не должен содержать предписаний, смысл которых может восприниматься неоднозначно. Четкое выполнение алгоритма приводит к получению одного и того же результата для одних и тех же данных, сколько бы мы раз его не выполняли.

Результативность (конечность). Выполнение любого алгоритма обязательно должно завершиться получением некоторого искомого результата либо сигнала о том, что данный алгоритм неприменим для решения поставленной задачи.

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

1.2. Формы представления алгоритмов

Алгоритмы можно представлять различным образом. Выбор формы определяется задачами, которые решаются с помощью алгоритмов, а также с учетом исполнителей для которых составлен данный алгоритм. Наиболее популярными являются записи алгоритмов в текстуальной форме, графическое представление алгоритмов, а также использование языков программирования.

Текстуальная форма представляет собой строгое пошаговое описание метода выполнения действий исполнителя с той или иной степенью дискретизации. При использовании текстуальной формы каждый шаг алгоритма записывают на обычном языке с использованием общеизвестных понятий, формул и определений. При этом глубина детализации зависит от степени готовности исполнителя алгоритма к пониманию представленных записей.

Составим, например, алгоритм нахождения наибольшего общего делителя (НОД) двух 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.

Формализуя данную задачу, введем переменные a и b, которым будет присвоено значение двух исследуемых чисел и разобьем действия исполнителя на следующие последовательные шаги.

1. Если $a = b$, то работа алгоритма закончена и наибольший делитель чисел a и b равен числу a.
2. Если $a > b$, то переменной a присваивается значение $a – b$, иначе переменной b присваивается значение $b – a$.
3. Выполняется пункт 1 данного алгоритма.

В зависимости от значений введенных чисел результат выполнения алгоритма будет различным. В таблице 1.1 представлен пример нахождения наибольшего общего делителя для чисел 64 и 40.

Таблица 1.1.

Выполнение алгоритма нахождения наибольшего общего делителя чисел 64 и 40

	Вводимые значения	1-й проход	2-й проход	3-й проход	4-й проход
a	64	64 – 40 = 24	24	24 – 16 = 8	8
b	40	40	40 – 24 =16	16	16 – 8 = 8

Из таблицы мы видим, что вводимые значения переменных *a* и *b* равны соответственно 64 и 40. В каждом из четырех проходов описанного алгоритма уменьшаются значения либо переменной *a* либо переменной *b* на величину наименьшего из двух этих значений. В результате после четвертого прохода значения переменных становятся равными, значит, согласно нашему алгоритму, вычислительный процесс следует завершить, а наибольший делитель чисел *64* и 40 равен восьми.

Для большей наглядности алгоритмы часто представляют в *графической форме*. В этом случае алгоритм описывают некоторой последовательностью графических объектов, блок-схем, которые соединены между собой направленными линиями потоков данных. Каждый пункт алгоритма отображается на схеме некоторой геометрической фигурой – *блоком* и дополняется элементами словесной записи. Основное направление потока данных идет сверху вниз и слева направо (стрелки могут не указываться), а в случае направлений снизу вверх и справа налево – стрелка обязательна.

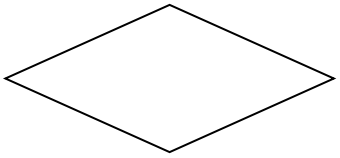
Процесс описания алгоритмов в графической форме регламентируется ГОСТом 19.701-90 (ИСО 5807-85) «Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения». Гост 19.701-90 распространяется на условные обозначения в схемах алгоритмов, программ, данных и систем и устанавливает правила выполнения схем, используемых для отображения различных видов задач обработки данных и средств их решения.

Однако при разработке программных приложений программисты часто отступают от данного ГОСТа используют общепринятые в среде разработчиков

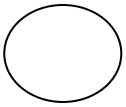
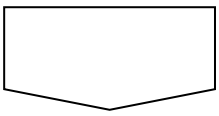
обозначения. В таблице 1.2 представлены наиболее используемые функциональные элементы блок-схем.

Таблица 1.2.

Наиболее используемые обозначения элементов графического представления алгоритмов

Наименование	Обозначение	Описание
Начало (конец) программы		Элемент отображает вход из внешней среды или выход из неё (наиболее частое применение − начало и конец программы). Внутри фигуры записывается соответствующее действие.
Блок ввода-вывода данных		Преобразование данных в форму, пригодную для обработки (ввод) или отображения результатов обработки (вывод). Данный элемент не определяет носителя данных (для указания типа носителя данных используются специфические символы).
Блок вычислений (операции)		Выполнение одной или нескольких операций, обработка данных любого вида (изменение значения данных, формы представления, расположения). Внутри фигуры записывают непосредственно сами операции в виде описаний или математических формул.
Логический блок (условие)		Отображает решение или функцию переключательного типа с одним входом и двумя или более альтернативными выходами, из

		которых только один может быть выбран после вычисления условий, определенных внутри этого элемента. Вход в элемент обозначается линией, входящей обычно в верхнюю вершину элемента. Если выходов два или три, то обычно каждый выход обозначается линией, выходящей из оставшихся вершин (боковых и нижней). Если выходов больше трех, то их следует показывать одной линией, выходящей из вершины (чаще нижней) элемента, которая затем разветвляется. Соответствующие результаты вычислений могут записываться рядом с линиями, отображающими эти пути.
Блок модификации (цикл с параметром)		Используется как заглавная часть цикл цикла с управляющей переменной (параметром), описания его структуры. В данном элементе устанавливаются начальное и конечное значения параметра цикла, шаг приращения параметра. Вход в элемент обозначается линией, входящей обычно в верхнюю часть элемента, выход из правого угла шестиугольника. Тело цикла размещается ниже блока модификации, вход в тело цикла – из середины нижней стороны фигуры, а соединение с выходом тела цикла в левом углу.

Предопределенный процесс (подпрограмма)		Элемент отображает выполнение процесса, состоящего из одной или нескольких операций, который определен в другом месте программы (в подпрограмме, программном модуле). Внутри символа записывается название процесса и передаваемые в него данные. В программировании – это, как правило, вызов процедуры или функции.
Соединитель		Указание связи прерванными линиями между потоками данных в пределах одного листа.
Межстраничные соединения		Указание связи между потоками данных на разных листах.

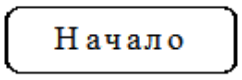
Составление блок-схем не является обязательным элементом при разработки программных приложений, однако в случае, когда алгоритм имеет сложную структуру, графический метод значительно упрощает восприятие структуры задачи, осмысливание постановки и реализации всей решаемой задачи и его частей.

При составлении алгоритмов в графической форме необходимо придерживаться следующих принципов:

1) блок-схема всегда должна начинаться элементом «Начало», а заканчиваться элементом «Конец»;
2) все элементы блок-схемы должны быть связаны между собой. Не должно быть «висячих», не связанных элементов.

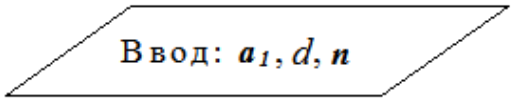
Пример 1.1. Составим графическое описание алгоритма нахождения значений n-ого члена и конечной суммы S_n арифметической прогрессии.

Первым элементом графического описания любого алгоритма будет блок «Начало»:



.

Для решения задачи нам потребуются значения первого члена арифметической прогрессии a_1, разность прогрессии d и количество членов в прогрессии n. Все эти значения можно определить в блоке ввода-вывода данных:



.

Значение n-ого элемента прогрессии можно получить по формуле:

$$a_n = a_1 + d \cdot (n-1),$$

а сумму первых n членов, как произведение среднего арифметического первого и n-ого членов прогрессии на n:

$$S_n = \frac{a_1 + a_n}{2} n .$$

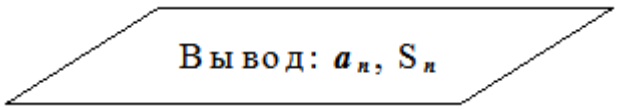
Все действия запишем в двух последовательных блоках вычислений:

$$a_n = a_1 + d \cdot (n-1)$$

$$S_n = \frac{a_1 + a_n}{2} n$$

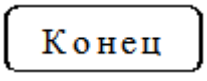
.

После вычислений результирующие значения a_n и S_n необходимо вывести с помощью блока ввода-вывода данных:



.

В завершении описания алгоритма будем использовать блок «Конец программы»:



.

Вышеописанные графические элементы расположим последовательно друг под другом и соединим блоки линией потока данных с указанием направления:

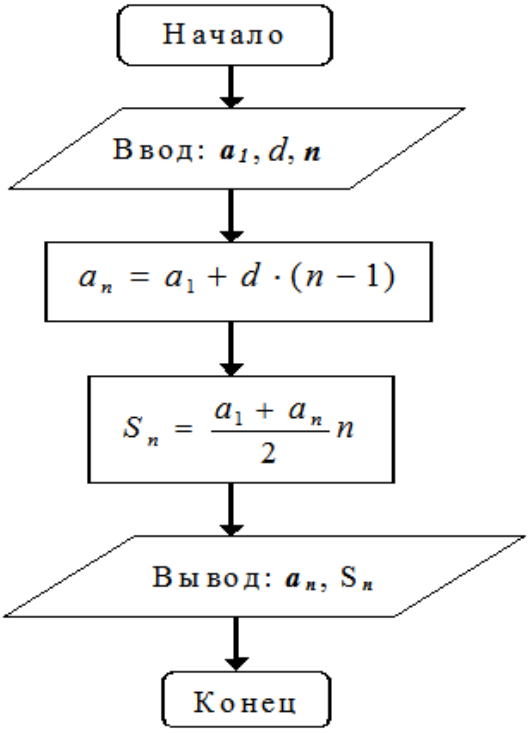


Рис. 1.1. Графическое представление алгоритма нахождения значений члена a_n и конечной суммы S_n арифметической прогрессии.

Еще одной формой описания алгоритмов является представление алгоритмов на *языках программирования*. Использование этой формы является необходимым условием для разработки любых компьютерных программ. Методы и технологии описания алгоритмов зависят от выбранного языка программирования и среды разработки программных приложений. Однако общие принципы разработки программных модулей везде одинаковы.

1.3. Базовые структуры программирования

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

К основным структурам программирования относятся следующие: линейные, разветвляющиеся и циклические.

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

Линейные алгоритмы понятны и просты в реализации. В большинстве своем алгоритмы программных продуктов являются нелинейными, однако они могут включать линейные фрагменты или, например, линейную последовательность модулей со сложной структурой. Рассмотренный ранее алгоритм вычисления значений n-ого члена и конечной суммы арифметической прогрессии являет линейным. Рассмотрим еще один пример.

Пример 1.2. Перевести угловые значения из градусов в радианы.

Для решения данной задачи необходимо ввести величину угла в градусной мере. Вспомним, что развернутый угол составляет 180 градусов или π радиан, следовательно, 180 градусов равны π радианам, а один градус – это $\frac{\pi}{180}$ радиан. Значит, для перевода из градусной меры в радианы количество градусов необходимо умножить на π и разделить на 180.

Рассмотрев математические основания задачи составим алгоритм решения. В начале программы введем значение угла в градусах $Grad$, затем рассчитываем значение угла в радианах по формуле $\frac{\pi}{180} \cdot Grad$ и, наконец, выведем полученное значение. В графической форме описанный алгоритм будет выглядеть следующим образом:

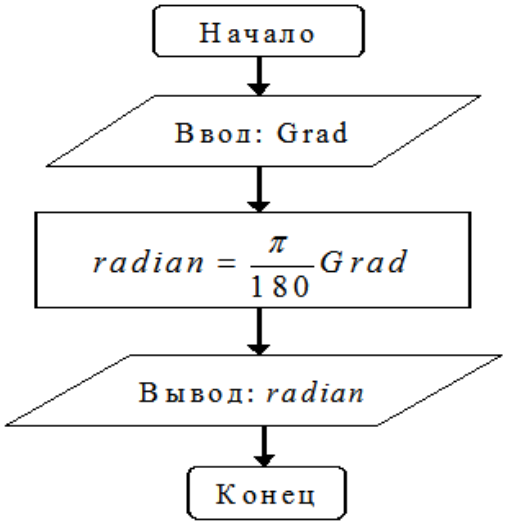


Рис. 1.2. Алгоритм перевода углового значения из градусов в радианы.

Частным случаем нелинейного алгоритма является ветвление.

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.

Данную структуру можно описать в виде графической структурной схемы (рис. 1.3) Как видим из схемы, в зависимости от результата проверки условия выполняется одно из двух действий.

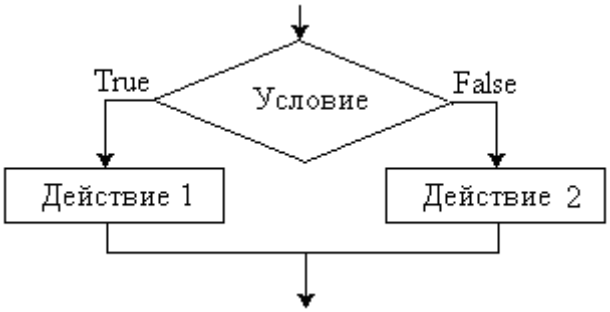


Рис. 1.3. Структурная схема ветвления.

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 рисунке 1.4. алгоритм.

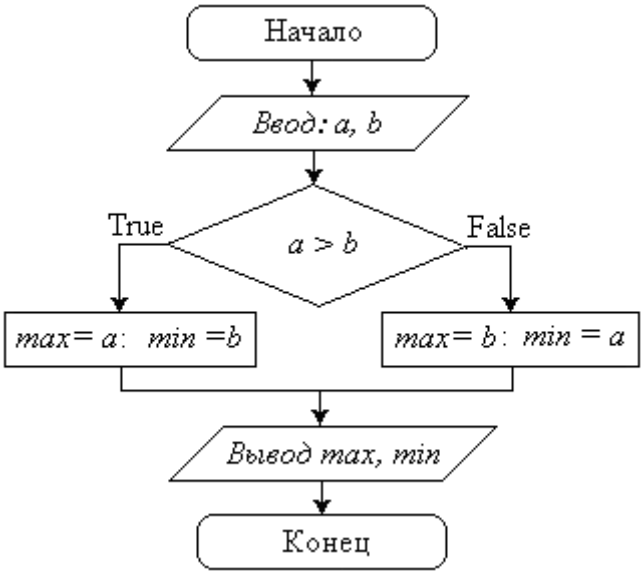


Рис. 1.4. Алгоритм нахождения максимального и минимального значений.

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*.

Альтернативная ветвь *False* в структуре *ветвление* может отсутствовать, если в ней нет необходимости. Обычно такую алгоритмическую структуру называют *обходом* (рис. 1.5).

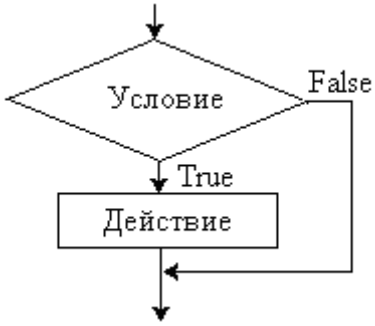
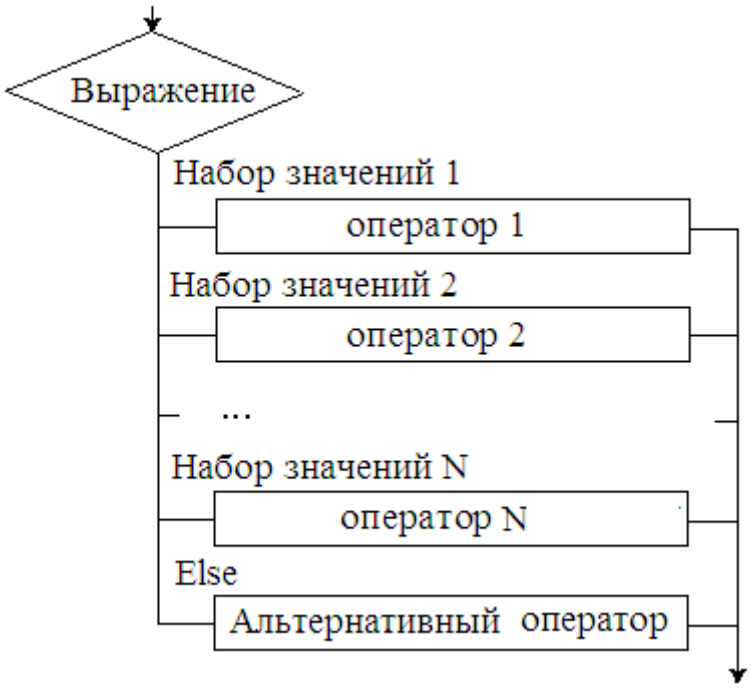


Рис. 1.5. Структурная схема инструкции ветвления без ветви *False.*

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



Структурная схема ветвления с множественным выбором.

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

Для определения дня недели нам потребуется определить номер недели, который можно получить автоматически из операционной системы. В зависи-

мости от номера дня недели, определяемого компьютером автоматически, необходимо присвоить переменной *t* значение дня недели, соответствующего определенному номеру, а затем день недели вывести, например, на экран монитора или использовать далее для последующих вычислений.

Данный алгоритм можно реализовать следующим образом (рис. 1.6):

Рис. 1.6. Структурная схема алгоритма вывода дня недели.

Пример. Составим 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:

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

Составление алгоритма необходимо начать с введения начальных данных. В нашем случае введем коэффициенты квадратного уравнения *a*, *b* и *c*. Следующий шаг алгоритма – вычисление дискриминанта. Затем, будем использовать структуру ветвления. В логическом блоке «Условие» установим неравенство: $D < 0$. В случае выполнения поставленного условия (ветвь True) выведем сообщение «Действительных корней нет!», в противном случае (ветвь False) по выше определенным формулам рассчитаем значения переменных x_1 и x_2, после чего выведем рассчитанные корни уравнения.

Приведем структурную блок-схему решения данной задачи (рис. 1.7.).

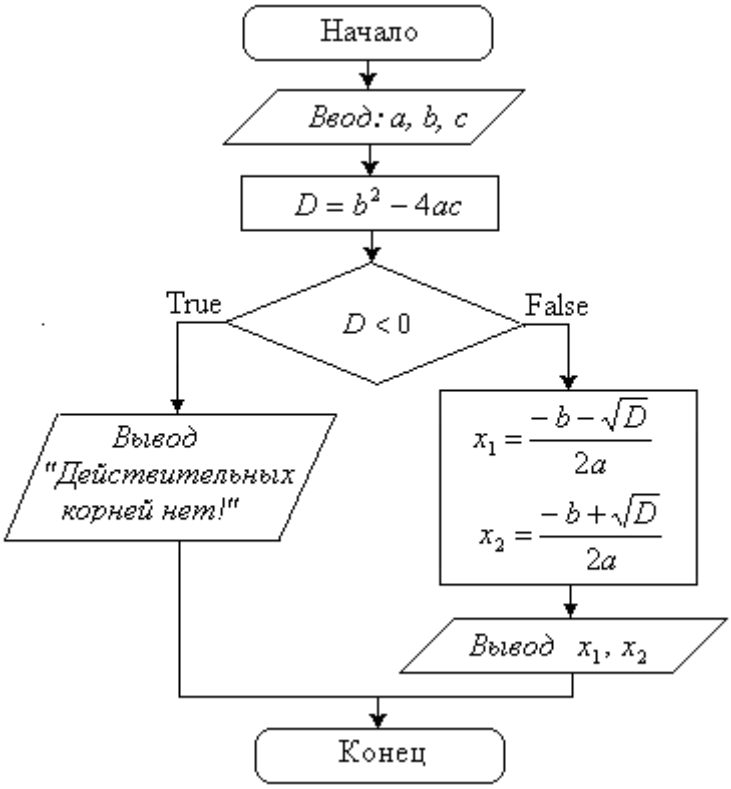


Рис. 1.7. Нахождение действительных корней квадратного уравнения.

Алгоритмическая структура, обеспечивающая организацию многократного исполнения определенного набора инструкций, называется *циклическим процессом* или просто *циклом.*

Последовательность действий, повторяющихся в цикле, называют *телом цикла*. Каждый проход цикла называется *итерацией*. Переменные, изменяющи-

еся в процессе выполнения цикла и влияющие на его окончание, называются *параметрами цикла*.

Пример. Данная алгоритмическая структура (рис. 1.8.) используется для увеличения значения натурального числа *n* числа, до тех пор, пока *n* не будет кратно *t*.

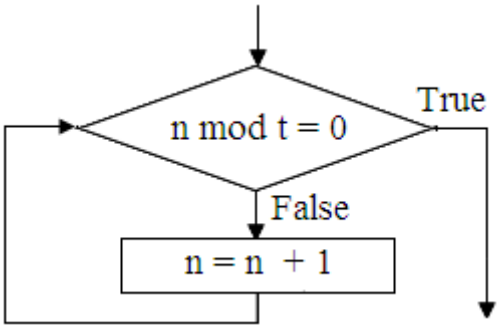


Рис. 1.8. Циклическая структура.

Если начальное значение параметра цикла *n* = 58, а значение переменной *t* = 7, то количество итераций 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) Логические выражения условия цикла должны содержать значения, определенные до первого выполнения тела цикла.

Алгоритм рассмотренного примера удовлетворяет обоим правилам. Приведем примеры циклов, в которых данные требования игнорируются (рис. 1.9).

Пример. Рассмотрим две алгоритмические структуры.

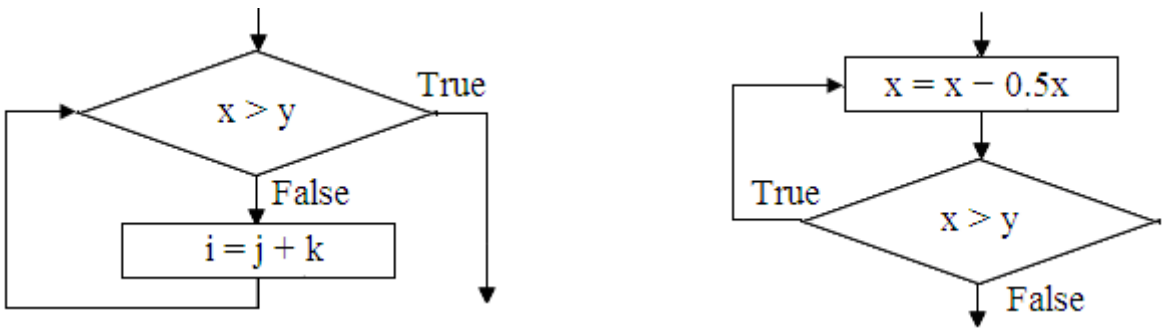


Рис. 1.9. Структурные схемы двух циклов.

В первой структуре тело цикла никак не влияет на условие. Если до выполнения цикла $x > y$, то тело цикла не выполнится ни разу. В противном случае тело цикла будет выполняться неограниченное количество раз. При использовании второй структуры, если переменная *x* до наступления цикла принимает отрицательное значения, то с каждой итерацией она будет увеличиваться и, при соответствующих значениях *y* тело цикла будет выполняться «бесконечно».

Циклические алгоритмы принято подразделять на циклы с предусловием, циклы с постусловием и циклы с параметром.

Циклом с предусловием называется цикл, в котором истинность условия проверяется каждый раз *перед* выполнением операторов тела цикла (рис. 1.10).

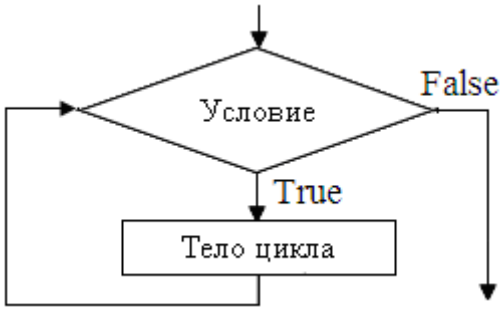


Рис. 1.10. Цикл с предусловием.

Данная структура работает следующим образом:

1. Проверяется условие. Если оно истинно (ветвь True), то выполняется тело цикла. В противном случае (ветвь False) цикл заканчивается, и управление передается оператору, следующему за телом цикла.

2. После выполнения тела цикла снова выполняется проверка условия.

Пример. Найти все делители заданного натурального числа *N*.

Рассмотрим алгоритм нахождения делителей (рис. 1.11). Согласно условию задачи, в начале программы введем значение натурального числа *N*. Будем проверять на делимость все натуральные числа от единицы до самого *N*. Для этого зададим переменной *d*, отвечающей за делители, начальное значение 1.

Для организации цикла с предусловием определим условие $d \le N$. В случае выполнения этого условия выполняем тело цикла, а если условие не выполняется, то происходит выход из цикла.

В теле цикла используем структуру обход. Если *d* является делителем числа *N*, то есть условие выполняется, число *d* записываем в список *t*, в противном случае никаких действий не чего не предпринимаем. Здесь же в теле цикла увеличиваем *d* на единицу.

Тело цикла будет выполняться до тех пор, пока будет выполняться условие $d \le N$. Как только значение переменной *d* станет больше *N*, произойдет выход из цикла, а затем вывод уже заполненного списка делителей *t*.

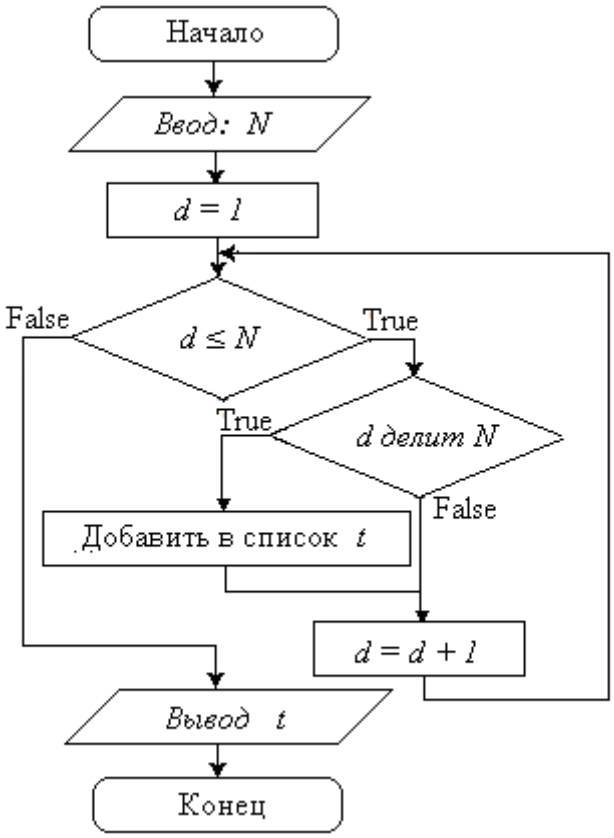


Рис. 1.1. Алгоритм нахождения делителей натурального числа N.

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

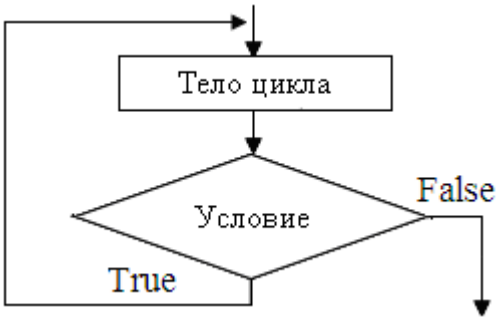


Рис. 1.12. Цикл с постусловием.

Цикл с постусловием тестирует условие после выполнения тела цикла. Если условие выполняется, то программа снова обращается к началу тела цикла. Следует отметить, что цикл с постусловием всегда будет выполнен хотя бы один раз.

Пример. Определить количество цифр целой части действительного числа *X*.

Для того чтобы подсчитать количество цифр в заданном числе, необходимо определить, сколько раз это число можно разделить на десять нацело. Алгоритм решения задачи будет основан на использовании цикла с постусловием.

Алгоритм подсчета цифр можно представить в виде структурной схемы (рис. 1.13). Во избежание потери значения переменной *X* ее значение присваивается промежуточной переменной *M*, и все действия совершаются с новой введенной переменной.

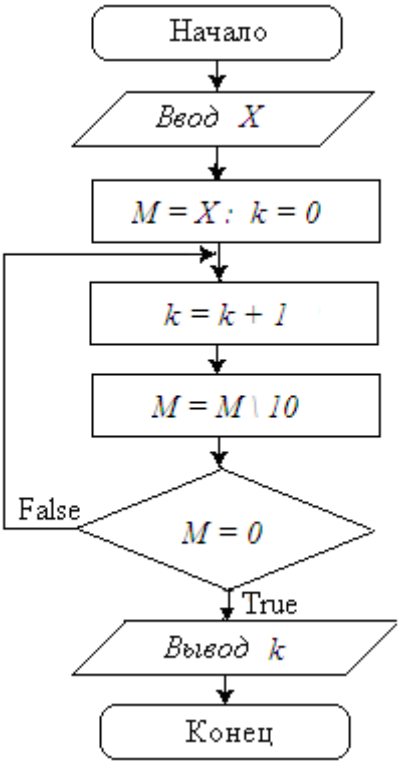


Рис. 1.13. Алгоритм нахождения количества цифр целой части числа.

В теле цикла на каждом проходе увеличивается значение счетчика *k*, определяющего количество итераций, а переменная *M* целочисленно делится на десять. Для целочисленного деления будем использовать символ «\».

Условие $M = 0$ проверяется после каждого прохода. В случае выполнения условия (ветвь *True*) происходит выход из цикла и вывод значения переменой *k*. В противном случае (ветвь *False*) возвращаемся к выполнению тела цикла.

Подсчитаем количество цифр у числа –72865,348. При выполнении описанного алгоритма тело цикла выполнится 5 раз, и, следовательно, количество цифр в числе –72865,348 будет равно 5. Результаты пошагового выполнения алгоритма приведены в таблице 1.3.

Таблица 1.3.

Результаты пошагового выполнения алгоритма

Шаг	X = –72865,348
1	–72865,348 \ 10 = – 7286
2	– 7286 \ 10 = – 728
3	– 728 \ 10 = – 72
4	-72 \ 10 = – 7
5	-7 \ 10 = 0

Конструкции циклов с предусловием и постусловием универсальны и могут использоваться для разработки любых циклических алгоритмов, но часто возникают более простые случаи, когда количество итераций в цикле заранее известно и тело цикла должно быть выполнено фиксированное количество раз. В таких случаях используется *цикл с параметром* (*управляющей переменной*).

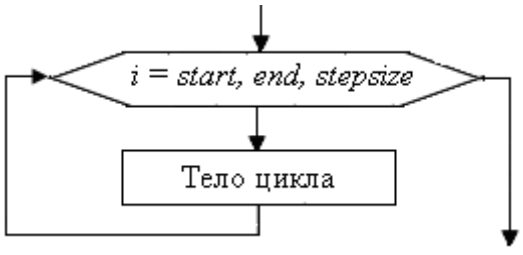


Рис. 1.14. Цикл с параметром.

Алгоритм работы цикла с параметром заключается в следующем.

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 пункту 2.

Рассмотрим пример использования цикла с параметром.

Пример 13. Вычислить факториал натурального числа *N.*

Напомним, что факториалом *N*! называется произведение всех натуральных чисел от единицы до *N*:

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

Например, $6! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 = 720$.

Следует заметить, что факториал нуля тоже существует и равен единице.

$$0! = 1.$$

Составим алгоритм нахождения факториала с помощью цикла с параметром (рис. 1.15). Введем значение переменной *N*. Начальное значение факториала Fact определим равным единице.

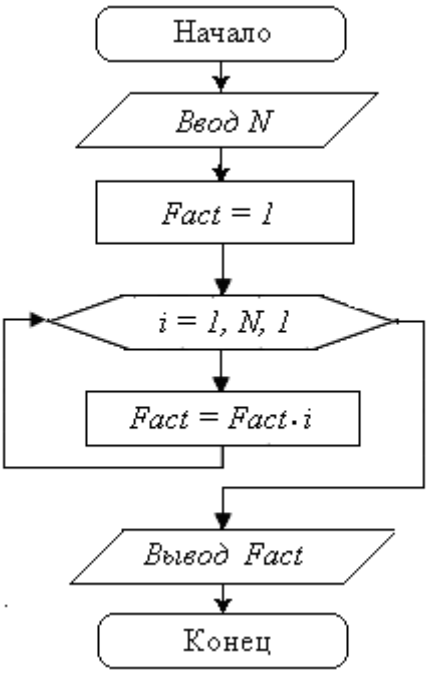


Рис. 1.15. Алгоритм нахождения факториала натурального числа *N.*

Очевидно, для расчета значения факториала *N*! необходимо выполнить ровно *N* итераций, а, значит, удобно использовать цикл с параметром, последовательно проходящим все значения от 1 до *N* с шагом 1.

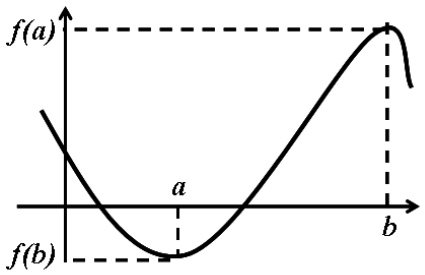
Тело цикла будет состоять из одного действия вычисления нового значения факториала – умножения факториала, полученного на предыдущей итерации, на текущее значение параметра цикла.

Выход из цикла произойдет после того, когда после N-ого прохода тела цикла параметр *i* примет значение N+1. Рассчитанное значение переменной Fact выводится и действие алгоритма завершается.

Рассмотрим еще один пример использования циклических структур.

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

Рассмотрим приближенный метод нахождения



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.

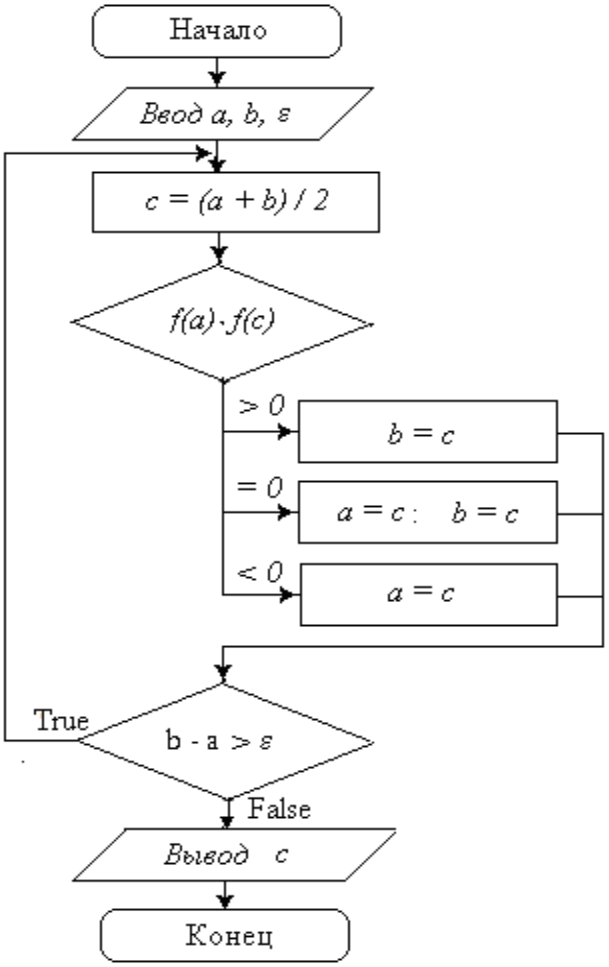
x	$-\infty$	-1	3/4	1	$+\infty$
Знак $f(x)$	+	-	-	-	+

Из таблицы видно, что уравнение имеет два действительных корня:

$$x_1 \in (-\infty,-1]; \quad x_{2\in}[1,+\infty).$$

Уменьшим промежутки, в которых находятся корни. Для этого выберем значения аргумента, в которых функция принимает положительные значения:

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



Алгоритм нахождения корня методом половинного деления.

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)·f(c) > 0*, тогда функция в данных точках одного знака и, следовательно, корня на промежутке [*a*, *c*] нет и мы уменьшаем отрезок [*a*, *b*], перенося левую его границу *a* в точку *c*..

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

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.

Раздел 2. Введение в языки программирования

Историю высокоуровневого 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.

1 мая 1964 года профессорами Дармутского колледжа (США) Джоном Кемени (John Kemeny) и Томасом Куртцем (Thomas Kurtz) была разработана первая версия языка программирования *Basic*. Свое имя язык получил от сокращения полного названия, данного учеными – *Beginner's All-purpose Symbolic Instruction Code,* которое дословно можно перевести, как многоцелевые символические инструкционные коды для начинающих. Кроме того, и слово *Basic* ассоциируется с понятием «базовый».

Первоначально язык *Basic* предназначался для обучения программированию, однако 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*, что потребовало большое количество программистов, владеющих Basic.

На популярность языка *Basic* повлияло использование его Полом Алленом и Уильямом (Биллом) Гейтсом в 1975 году для разработки интерпретатора первой персональной ЭВМ Altair. MS Basic стал первым продуктом, выпущенным компанией Microsoft.

В связи с массовым выпуском персональных ЭВМ другими производителями Basic стал использоваться практически на популярных моделях Apple, Commodore, Atari, HP, затем интерпретатор стал использоваться и для появившегося в 1981 году компьютера IBM PC. Язык Basic приобрел огромную популярность во всем мире в силу своей простоты и ориентации на диалоговый режим.

В 1985 году корпорацией Microsoft реализован язык QuickBasic, предназначенный для операционных систем DOS и MAC OS. Это компилируемый язык, с больший набором структурных конструкций, пользовательских типов данных, благодаря чему стало возможным использовать Basic наравне с классическими языками – Pascal или C.

В 1991 корпорацией Microsoft

В 1976 г разработан первый *стандарт минимального подмножества* языка Basic, в 1978 и 1987 годах появились стандарты ANSI (American national standards institute), а в 1984 и 1991 годах – два стандарта ISO (International Organization for Standardization).

Сегодня наибольшее распространение получил *Visual Basic*, и в первую очередь *Visual Basic for Applications* (*VBA*), обслуживающий все приложения *Microsoft Office*. Это мощный инструмент, включенный в линейку продуктов Microsoft Office, AutoCad, SolidWorks, CorelDRAW, ESRI ArcGIS.

VBA – интерпретируемый язык, однако является мощным средством программной обработки данных в большом разнообразии современных программных приложений, позволяет использовать доступные объекты операционной системы, легко решая многие прикладные задачи.

2.1. Типы данных VBA

Для обработки компьютером 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.

Иерархия типов в языке Basic:

Простые
- Числовые
  - Целые
  - Вещественные
- Логические
- Даты

Структурированные
- Строковые
- Массивы
- Variant

Объектные

Таблица 2.1.

Описание типов данных

Идентификатор	Длина (байт)	Диапазон значений
Целые типы		
byte	1	0 … 255
integer	2	-32 768 … 32 767
long	4	-2 147 483 648 … 2 147 483 647
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	$4.94 \times 10^{-324} .. -1.79 \times 10^{308}$
urrency	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
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
Variant (символьный)	22+длина строки	Любая последовательность символов

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

2.2. 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]
```

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

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

Пример.

a! = 5.64

b! = 44.76

x% = (a + b)/2

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.

Пример. Разработать программные модули для расчета значений *n*-ого члена и конечной суммы арифметической и геометрической прогрессий.

Структурные блок-схемы разрабатываемых модулей будут выглядеть следующим образом:

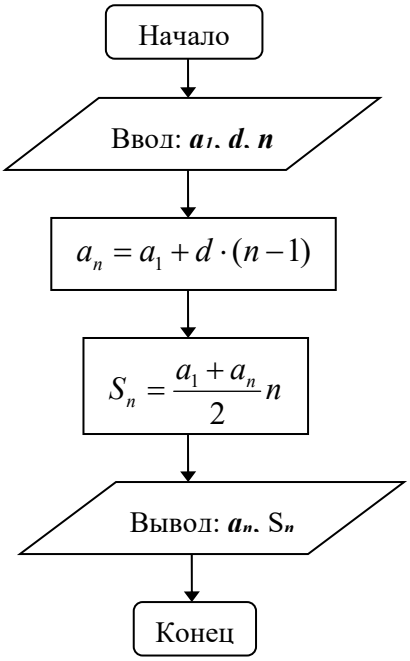


Рис. 2.1. Арифметическая прогрессия

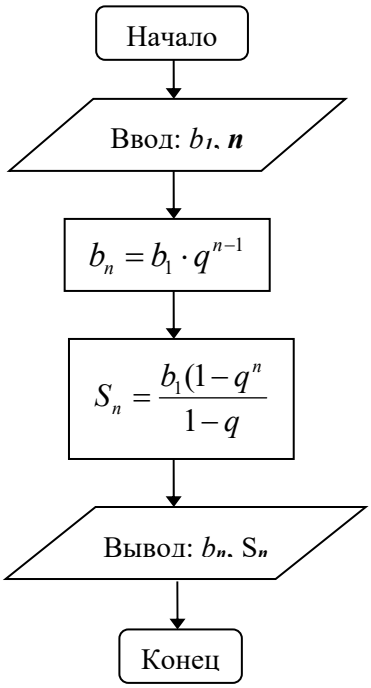


Рис. 2.2. Геометрическая прогрессия

Подготовим место для вводимых данных на листе MS Excel.

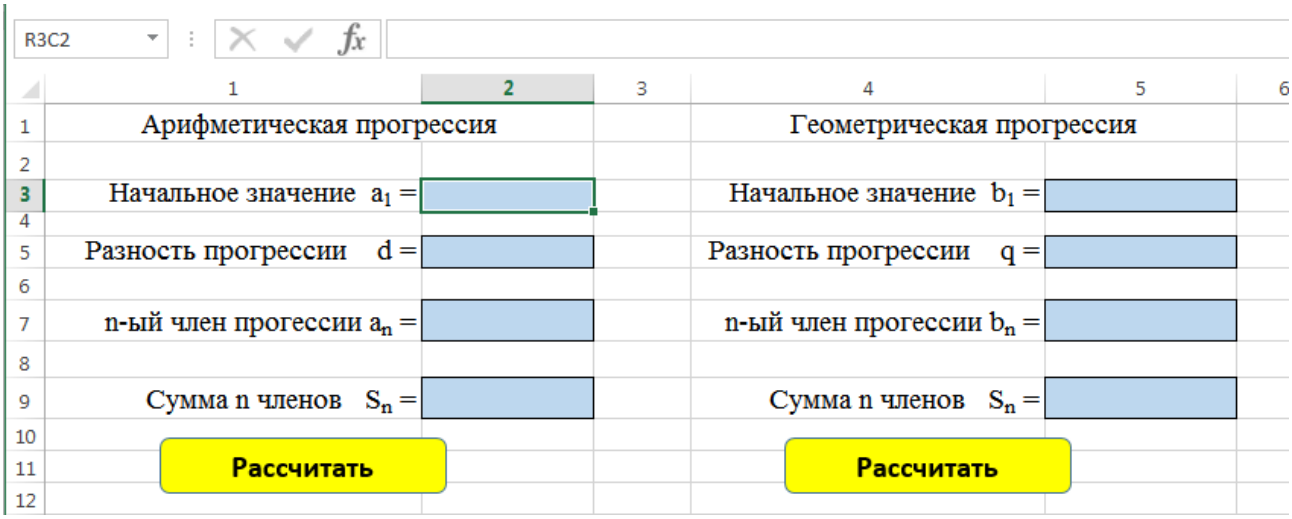


Рис. 2.3. Данные на листе Excel.

Для написания программы воспользуемся командой «Visual Basic» в закладки «Разработчик» и откроем редактор Visual Basic for Application. Выбрав команду «Module» из закладки «Insert» редактора, создадим новый модуль для помещения в него программного кода.

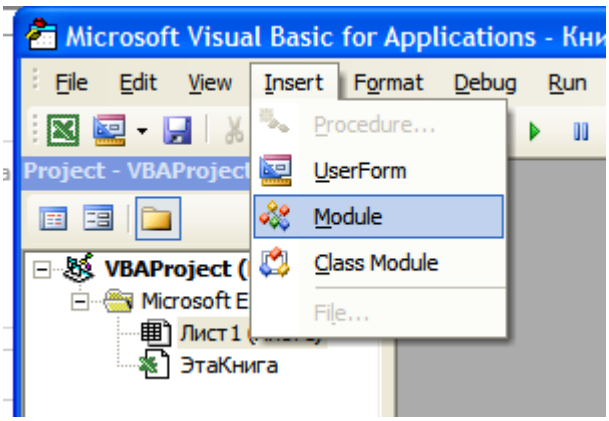


Рис.2.4. Закладка Insert.

Затем можно приступить к написанию программы.

```
Sub Арифметическая_прогрессия()
Dim a1 As Double, d As Double, an As Double, Sn As Double, n As Long
n = InputBox("Введите количество членов N", "Арифметическая прогрессия")
a1 = Cells(3, 2)
d = Cells(5, 2)
an = a1 + d * (n - 1)
Sn = (a1 + an) / 2 * n
Cells(7, 2) = an
Cells(9, 2) = Sn
End Sub
```

Рис. 2.5. Программные код макроса.

Началу программы соответствует инструкция Sub. Именем программы может быть любой идентификатор, содержащий латинские, русские буквы, цифры и некоторые другие символы. Вместо пробелов следует использовать знаки подчеркивания. Конец программы End Sub появится автоматически.

Инструкцией *Dim* объявим явно все переменные, используемые в алгоритме. Значения переменных *a*1и *d* возьмем из подготовленной формы листа MS Excel. Количество членов *n* введем через диалоговое окно.

Далее произведем расчеты значений *a*1 и Sn. Полученные значения запишем на лист MS Excel в ячейки (7, 2) и (9, 2).

Для удобного запуска программы назначим макросу «Арифметическая_прогрессия» объекту «Рассчитать».

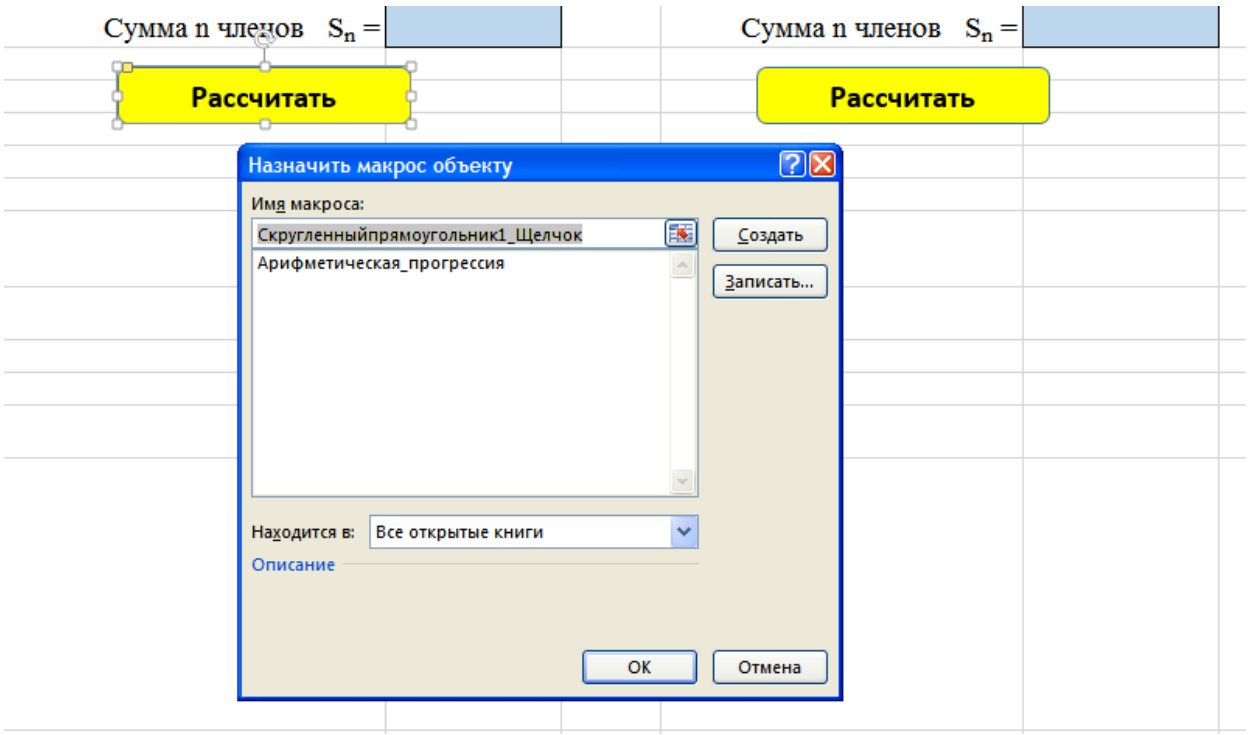


Рис. 2.6. Назначение макроса объекту.

После записи на лист начального члена и разности прогрессии программу можно запускать.

Аналогично можно записать и код второго макроса.

```
Sub Геометрическая_прогрессия()
Dim b1 As Double, q As Double, bn As Double, Sn As Double, n As Long
n = InputBox("Введите количество членов N", "Геометрическая прогрессия")
b1 = Cells(3, 5)
q = Cells(5, 5)
bn = b1 * q ^ (n - 1)
Sn = b1 ^ (1 - q) / (1 - q)
Cells(7, 5) = bn
Cells(9, 5) = Sn
End Sub
```

Результат выполнения программы при n=20 показан на рисунке.

	1	2	3	4	5
1	Арифметическая прогрессия			Геометрическая прогрессия	
2					
3	Начальное значение a_1 =	10		Начальное значение b_1 =	30
4					
5	Разность прогрессии d =	3		Разность прогрессии q =	0,5
6					
7	n-ый член прогессии a_n =	67		n-ый член прогессии b_n =	5,72205E-05
8					
9	Сумма n членов S_n =	770		Сумма n членов S_n =	10,95445115
10					
11	Рассчитать			Рассчитать	

Рис. 2.7. Результат выполнения программы.

2.3. 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.

В языке VBA используются операторы арифметические, логические, сравнения и присвоения.

Арифметических операторов в VBA всего 7:

сложение (+); вычитание (-);

умножение (*); деление (/);

возведение в степень (^); целочисленное деление (\);

деление по модулю (Mod).

Оператор присвоения в VBA — знак равенства. Можно записывать так:

Let nVar = 10

а можно еще проще:

nVar = 10

Во втором случае нельзя путать знак равенства с оператором равенства.

Выражение *nVar = 10* значит «присвоить переменной nVar значение 10», а если строка выглядит так: *If (nVar = 10),* то это значит «если значение переменной nVar равно 10». Если переменной нужно назначить объект, то делается это другими способами.

Операторов сравнения в VBA всего 8:

равенство (=), например, *If (nVar = 10)*;

больше, чем и меньше, чем (> и <), например, *If (nVar > 10)*;

больше или равно и меньше или равно (>= и <=), например, *If (nVar >= 10)*;

не равно (<>), например, *If(nVar<>10)*;

сравнение объектов (Is). Определяет, ссылаются объектные переменные на тот же объект или на разные, например, *If(obj1 is obj2)*;

подобие (Like). Сравнивает строковый объект с шаблоном и определяет, подходит ли шаблон.

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

Сравнение строковых значений:

- при сравнении строковых значений регистр учитывается;
- пробелы в строковых значениях также учитываются;
- при сравнении текстовых строк на больше/меньше по умолчанию сравниваются просто двоичные коды символов — какие больше или меньше. Если нужно использовать тот порядок, который идет в алфавите, то можно воспользоваться командой

Option Compare Text

Общий его синтаксис оператора Like выглядит как

Выражение1 Like Выражение2

При этом Выражение1 — любое текстовое выражение VBA, а Выражение2 — шаблон, который передается оператору Like. В этом шаблоне можно использовать специальные подстановочные символы

2.4. Функции для работы с числовыми значениями

Функций для работы с числовыми значениями в VBA очень много. Используются они реже, чем строковые функции, но во многих ситуациях без них не обойтись.

Ниже приведены только универсальные функции VBA для работы с числовыми значениями. Эти функции доступны из любых приложений VBA.

- ***ABS()*** — эта функция возвращает абсолютное значение переданного ей числа (читайте, то же число, но без знака). Например, ABS(3) и ABS(-3) вернут одно и то же значение 3. Обычно используется тогда, когда нам нужно определить разницу между двумя числами, но при этом мы не знаем, какое число — первое или второе — больше. Результат вычитания может быть и положительным и отрицательным. Чтобы он был только положительным, используется эта функция.
- ***Int()***, ***Fix()*** и ***Round()*** позволяют по разному округлять числа: Int возвращает ближайшее меньшее целое, Fix() отбрасывает дробную часть,

Round() округляет до указанного количества знаков после запятой. При этом Round() работает не совсем правильно, в чем легко убедиться:

MsgBox(Round(2.505, 2))

Поэтому на практике для округления лучше использовать Format():

MsgBox(Format(2.505, "#,##0.00"))

- *Rnd()* и команда *Randomize* используются для получения случайных значений (очень удобно для генерации имен файлов и в других ситуациях). Обычный синтаксис при применении Rnd выглядит так:

  случайное_число = Int(минимум + (Rnd() максимум))*

  *MsgBox(Int(1 + (Rnd() * 100)))*

  Настоятельно рекомендуется перед вызовом функции Rnd() выполнить команду Randomize для инициализации генератора случайных чисел.

- *Sgn()* — позволяет вернуть информацию о знаке числа. Возвращает 1, если число положительное, -1, если отрицательное и 0, если проверяемое число равно 0.

2.5. Функции для организации взаимодействия с пользователем

Во многих программах VBA необходимо обеспечить взаимодействие с пользователем — проинформировать его о чем-то и (возможно) получить от него ответную реакцию. В принципе, для пользователя можно просто вывести текст в окне приложения (например, в текущем документе Word) или воспользоваться формой и элементами управления. Как это делается — мы узнаем в соответствующих главах. В этой части мы рассмотрим только применение для этой цели встроенных функций VBA.

Самой простой способ вывести информацию пользователю — воспользоваться встроенной функцией VBA *MsgBox()*. Примеров применения этой функции в нашей книге уже было множество, а полный ее синтаксис выглядит так:

MsgBox(Текст[,кнопки] [,заголовок окна] [, файл справки, метка в файле справки])

Возможностей у MsgBox() достаточно много:

- можно отображать разное кол-во кнопок (*OK*, *Cancel*, *Abort*, *Retry*, *Ignore*, *Yes*, *No*),
- можно показывать символы *Critical, Warning, Question, Information*,
- можно выбирать кнопку по умолчанию,
- можно делать окно модальным или обычным.

В зависимости от того, на какую пользователь кнопку нажал, такое значение возвращается приложению (всего 7 вариантов). Подробнее — в справке по VBA. Пример возврата значения от MsgBox():

```
Dim nVar As Integer
nVar = MsgBox ("Будем делать?", 65, "Демонстрационное окно сообщения")
```

Если значение nVar равно 1, то пользователь нажал OK, если 2, то Cancel.

Иногда (например, при пакетной обработке данных) хотелось бы, чтобы окно сообщения через некоторое время закрывалось само собой. Это можно сделать при помощи метода Popup() объекта Wscript.Shell. Для этого в проект через меню **References** нужно добавить ссылку на Windows Script Host Object Model (файл C:\WINNT\system32\wshom.ocx), а после этого использовать следующий код:

```
  Dim oShell As New WshShell
oShell.Popup "Test", 5
```

В остальном функциональность получившего окна одинакова с MsgBox(). Код возврата, если пользователь не нажал ни на какую кнопку, равен -1.

Самый простой способ принять информацию от пользователя — воспользоваться функцией *InputBox()*. Все очень просто :

```
  Dim Input
Input = InputBox("Введите Ваше имя: ")
MsgBox (" Вы ввели: " & Input )
```

Для InputBox() можно указать текст приглашения, заголовок окна, значение по умолчанию, местонахождение окна и файл справки. Не забывайте, что все вводимое пользователем InputBox() автоматически переводит в тип данных String — может потребоваться выполнить преобразование.

2.6. Функции преобразования и проверки типов данных

В программах на VBA очень часто приходится преобразовывать значения из одного типа данных в другой. Несколько типичных ситуаций, когда этим приходится заниматься:

- преобразование из строкового значение в числовое при приеме значения от пользователя через InputBox();

- преобразование значения даты/времени в строковое, когда нам нужно отобразить дату или время единообразно вне зависимости от региональных настроек на компьютерах пользователей;

- преобразование значения из строкового в дату/время для применения специальных функций даты/времени.

Чаще всего для конвертации типов данных используются функции, имя которых выглядит как C (от слова Convert) + имя типа данных. Вот перечень этих функций: *CBool(), CByte(), CCur(), CDate(), CDbl(), CDec(), CInt(), CLng(), CSng(), CStr(), CVar(), CVDate(), CVErr()***.** Просмотреть, что в итоге получилось, можно при помощи функции TypeName(), например:

```
nVar1 = CInt(InputBox("Введите значение"))
MsgBox TypeName(nVar1)
```

Кроме того, еще несколько полезных для конвертации функций:

-*Str()* — позволяет перевести числовое значение в строковое. Делает почти то же самое, что и *CStr()*, но при этом вставляет пробел впереди для положительных чисел.

- *Val()* — "вытаскивает" из смеси цифр и букв только числовое значение. При этом эта функция читает данные слева направо и останавливается на первом нечисловом значении (допускается единственное нечисловое значение — точка, которая будет отделять целую часть от дробной). Очень удобно, когда у нас вперемежку с числовыми данными прописываются единицы измерения или валюта.

Чтобы не возникло ошибок при конвертации, можно вначале проверять значения на возможность конвертации при помощи функций *IsNumeric()* и

IsDate(). Для проверки на соответствие специальным значениям можно использовать функции *IsArray()*, *IsEmpty()*, *IsError()*, *IsMissing()*, *IsNull()* и *IsObject()*. Все эти функции возвращают True или False в зависимости от результатов проверки переданного им значения.

Для того, чтобы преобразовать десятичные данные в строковое представление шестнадцатеричных и восьмеричных значений, используются функции *Hex()* и *Oct()*. Для обратного преобразования специальных функций не предусмотрено, но вы можете указать компилятору VBA, что эти числа записаны в шестнадцатеричном или восьмеричном формате, записав их, например, как &O12 и &HA.

Раздел 3. Ветвление

Частным случаем нелинейного алгоритма является ветвление.

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 реализации ветвления в кодах программ используются инструкции условного перехода *If* и выбора *Select Case*. Кроме того, для изменения линейной последовательности выполнения алгоритма без проверки условия применяют инструкцию *Goto*.

3.1. Инструкция *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.

Инструкцию *If* можно представить в виде структурной схемы (рис. 3.1.): Как видим из схемы, в зависимости от результата проверки условия выполняется одно из двух действий.

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

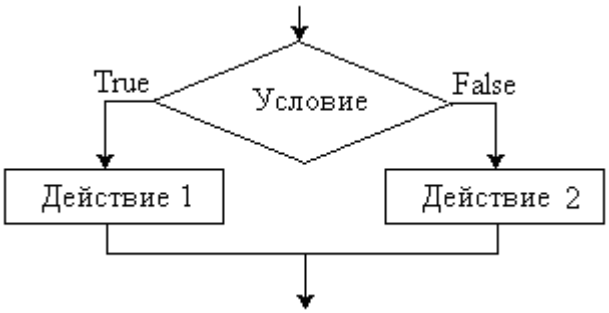


Рис. 3.1. Структурная схема инструкции *If.*

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 …
```

Пример. Для нахождения максимального и минимального значений из двух введенных пользователем чисел можно использовать представленный на рис. 2 алгоритм.

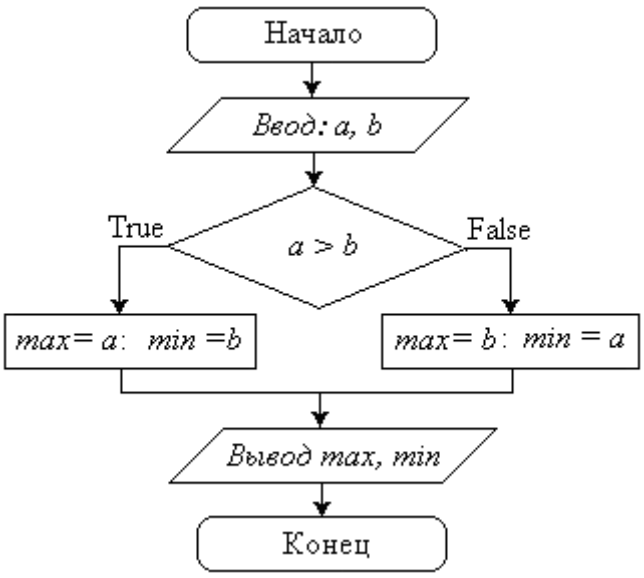


Рис. 3.2. Алгоритм нахождения максимального и минимального значений.

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 рис. 3.3 представлен результат выполнения программы при введенных значениях a = -7, b = -2.

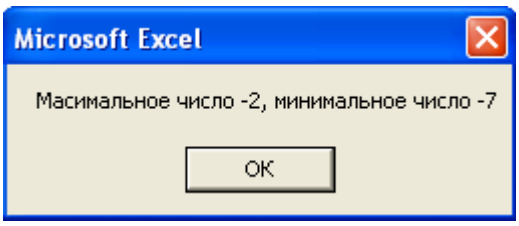


Рис. 3.3. Окно вывода максимального и минимального из двух чисел.

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

Для решения задачи по формуле

$$D = b^2 - 4ac \tag{3.1}$$

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

$$x_1 = \frac{-b - \sqrt{D}}{2a}, \qquad x_2 = \frac{-b + \sqrt{D}}{2a}. \tag{3.2}$$

Приведем структурную блок-схему решения данной задачи (рис. 3.4).

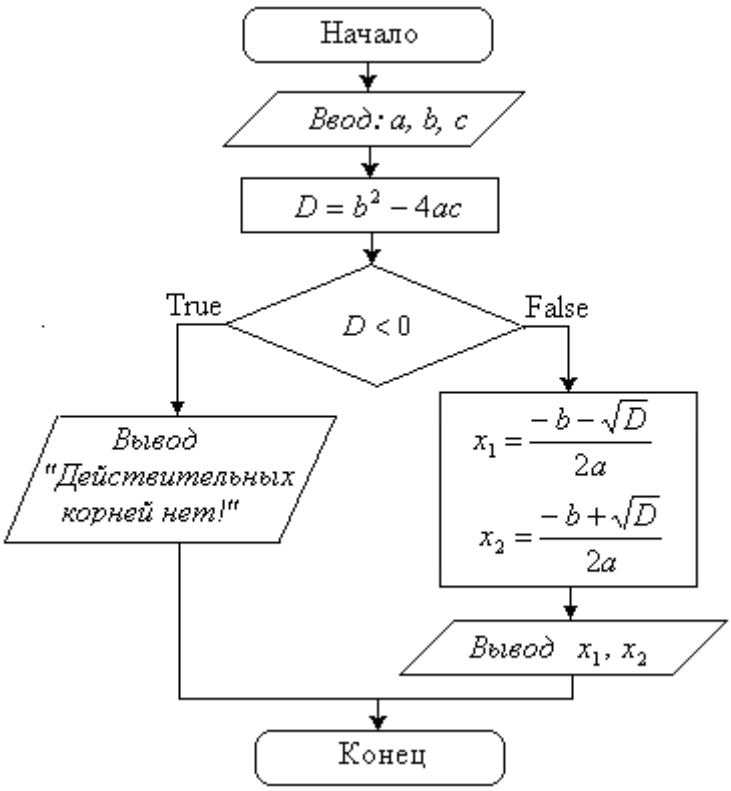


Рис. 3.4. Нахождение действительных корней квадратного уравнения.

Предложенному алгоритму соответствует следующий 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 рис. 5 представлен результат определения корней квадратного уравнения $-2x^2-10x+12=0$.

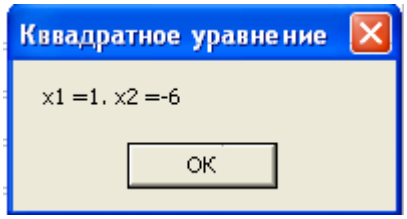


Рис. 3.5. Окно вывода решений квадратного уравнения.

Альтернативная ветвь *Else* в инструкции *If* может отсутствовать, если в ней нет необходимости. На рис. 6 представлена структурная схема ветвления без альтернативной ветви. Обычно такую алгоритмическую структуру называют *обходом*.

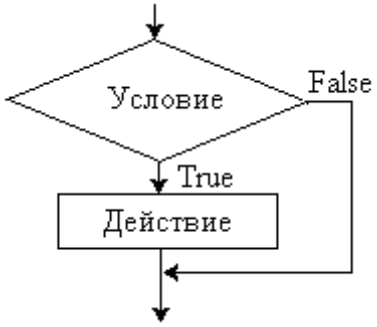


Рис. 3.6. Структурная схема инструкции *If* без ветви *Else*.

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

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. На рис. 7. представлены графические структурные схемы обоих вариантов.

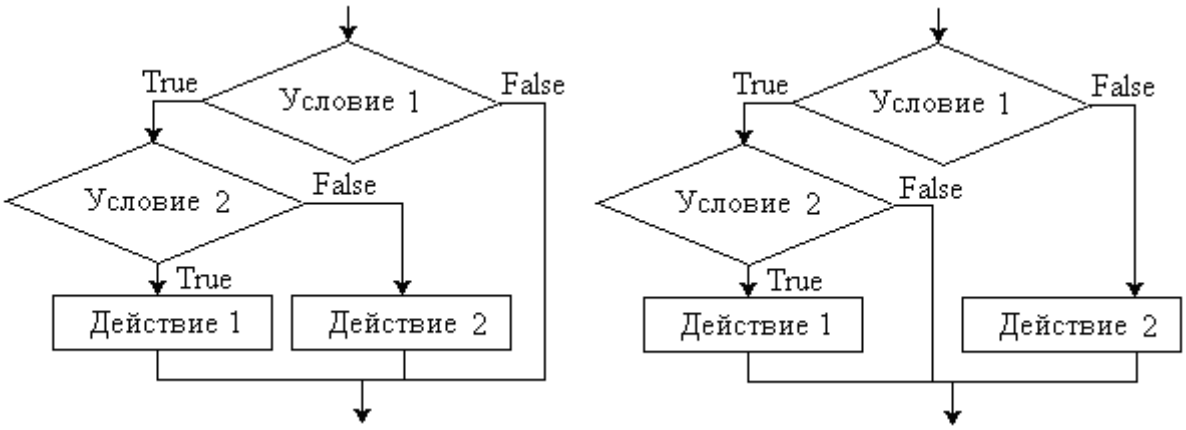


Рис. 3.7. Вложенные условные структуры.

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:

$$\frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1}. \qquad (3)$$

Умножим обе части уравнения (3) 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 $Ax + By + C = 0$

$$A = y_2 - y_1, \quad B = x_1 - x_2, \quad C = -x_1y_2 + x_2y_1. \qquad (4)$$

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 формулам (4) 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
```

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 Ox, 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 Abs(B) <> 1 And B <> 0 Then t = t + Str(B) + "y"
    If Abs(B) = -1 Then t = t + "-y"
    If Abs(B) = 1 Then t = t + "y"
  Else
    If A = -1 Then t = "-x"
    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 t, , "Общее уравнение прямой, проходящей через точки P1(" _
       & x1 & "; " & y1 & ") è P2(" & x2 & "; " & y2 & ")"
End Sub
```

На рис. 8 представлен результат вывода общего уравнения прямой, проходящей через точки $P_1(-1; -3)$ и $P_2(2; 8)$.

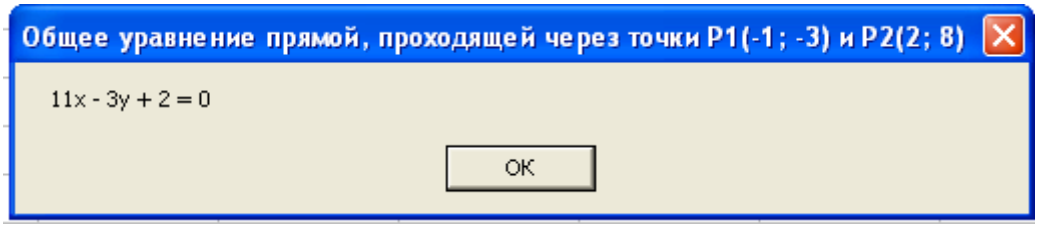


Рис. 3.8. Окно вывода общего уравнения прямой.

3.2. Инструкция *Select Case*

Инструкция выбора *Select Case* используется в тех случаях, когда в зависимости от значений исследуемой переменной 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
```

Инструкция *Select Case* выполняет следующие действия.

В случае если выражение принимает значение из набора значений 1, то выполняется оператор 1.

Если выражение принимает значение из набора значений 2, то выполняется оператор 2.

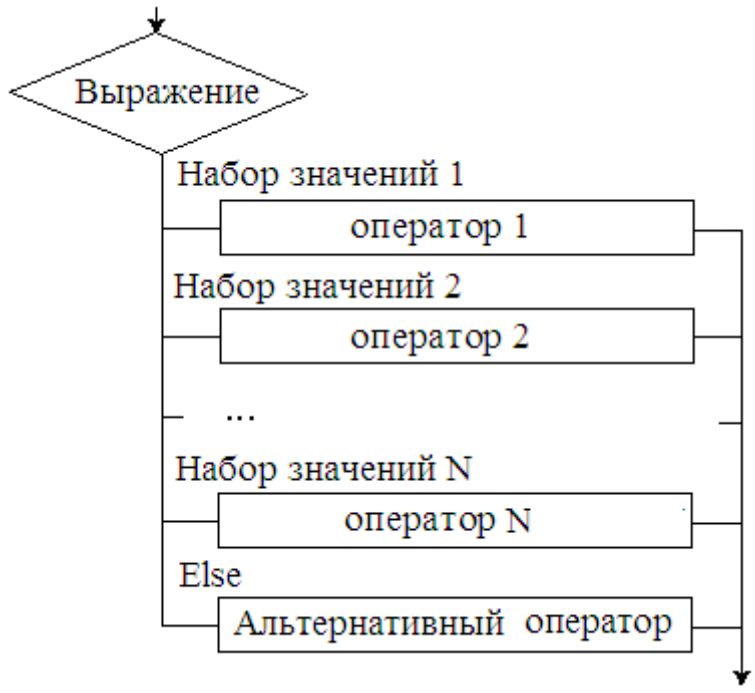


Рис. 3.9. Структурная схема инструкции *Select Case.*

Если выражение не принимает значений из имеющихся наборов, то выполняется альтернативный оператор, расположенный после ключевого слова *Case Else*. Альтернативный оператор в данной инструкции является необязательным и может быть опущен.

Работу инструкции *Select Case* можно описать структурой схемой (рис. 9).

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

Инструкция *Select Case* допускает использование составных операторов.

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

Рис. 3.10. Структурная схема программы вывода дня недели.

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

На рис. 3.10 представлена структурная схема решения данной задачи. Алгоритм заключается в выборе варианта для значения номера дня недели и присвоении переменной *t* соответствующего значения строковой записи.

Запишем представленный алгоритм в виде программного кода.

```
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 рис. 3.11 представлен результат вывода дня недели.

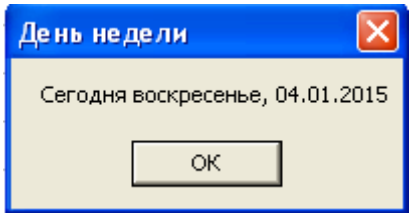


Рис. 3.11. Окно вывода дня недели и текущей даты.

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 (Таб. 3.1).

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.

Таблица 3.1.

ASCII коды 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	

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 рассматривались коды не всех выводимых символов, то для удобства пользователя необходимо использование альтернативного оператора, который заключается в присвоении переменной *t* значения «Символ не определен».

```
Sub Определение_символа()
  Dim s As String
  s = InputBox("Введите символ")
  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
End Sub
```

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 грани-

цами (-∞, *a*), (*a*, +∞), объединения различных промежутков. В таких случаях набор значений определяется с использованием операторов сравнения.

На рис. 3.12 представлен результат определения введенного с клавиатуры символа и его код.

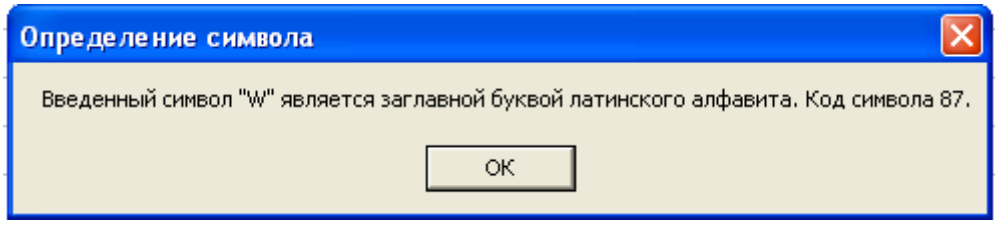


Рис. 3.12. Окно вывода результата определения символа.

Пример. Найти значение функции, заданной аналитически:

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

При написании программы заметим, что функция $y(x)$ имеет три промежутка, на которых функция определяется различными аналитическими выражениями. В коде программы удобно использовать инструкцию выбора 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*. В коде

программы второй набор значений аргумента $-3 < x \le 1$ задан выражением $Is \le 1$. На самом деле, второй набор будет использоваться только в том случае, если в первом наборе $Is \le -3$ значение переменной *x* не будет найдено.

На рис. 3.13 представлен результат нахождения значения функции *y*(-0,5).

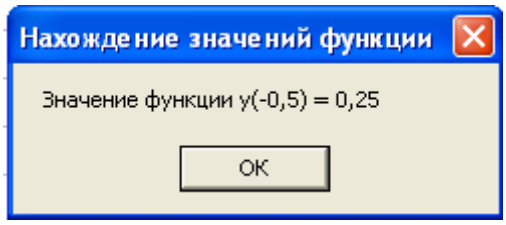


Рис. 3.13. Окно вывода значений функции *y*(*x*).

3.1.3. Инструкция *GoTo*

Помимо инструкций условного перехода *If* и *Select Case* в VBA используется инструкция безусловного перехода *GoTo*.

Инструкция *GoTo* имеет следующий синтаксис:

GoTo строка

Инструкция *GoTo* указывает на переход выполнения программы к строке программного кода, отмеченной номером или меткой. Номер или метка должны находиться в начале строки. Имя метки может быть любым идентификатором.

Использование безусловного перехода крайне ограничено. Обычно программисты используют инструкцию *GoTo* в процессе написания или отладки программ. Например, при необходимости игнорирования выполнения части программы перед началом неисполняемого фрагмента программного кода устанавливается инструкция *GoTo*, а после его конца– соответствующая метка.

```
…
GoTo Part_22
…
Неисполняемая часть программы
…
Part_22
…
```

Раздел 4. Циклы

При написании программ часто возникает необходимость повторения одних и тех же действий или операций.

Алгоритмическая структура, обеспечивающая организацию многократного исполнения определенного набора инструкций, называется *циклическим процессом* или просто *циклом.*

Последовательность действий, повторяющихся в цикле, называют *телом цикла*. Каждый проход цикла называется *итерацией*. Переменные, изменяющиеся в процессе выполнения цикла и влияющие на его окончание, называются *параметрами цикла*.

Пример. Данная алгоритмическая структура (рис. 14) используется для увеличения значения натурального числа *n* числа, до тех пор, пока *n* не будет кратно *t*.

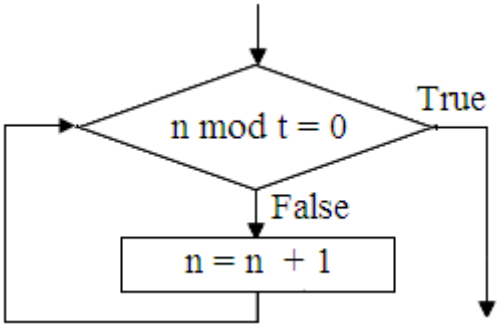


Рис. 14. Циклическая структура.

Если начальное значение параметра цикла $n = 58$, а значение переменной $t = 7$, то количество итераций 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) Логические выражения условия цикла должны содержать значения, определенные до первого выполнения тела цикла.

Алгоритм рассмотренного примера (рис. 14) удовлетворяет обоим правилам. Приведем примеры циклов, в которых данные требования игнорируются.

Пример 8. Рассмотрим две алгоритмические структуры (рис. 15, 16).

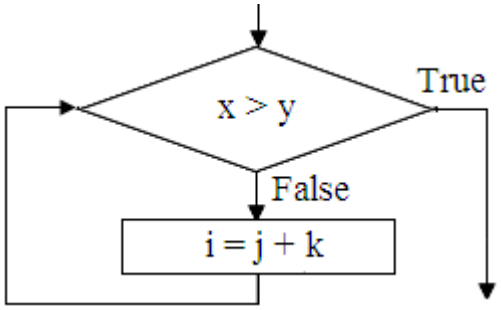


Рис. 4.1. Циклическая структура.

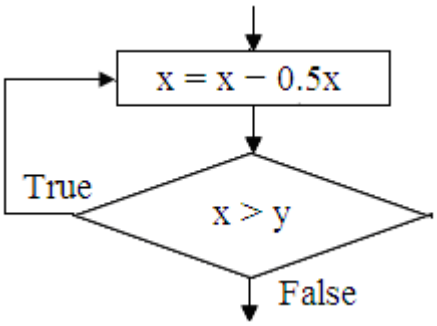


Рис. 4.2. Циклическая структура.

В первой структуре (рис. 4.1) тело цикла вообще никак не влияет на условие. Если до выполнения цикла $x > y$, то тело цикла не выполнится ни разу. В противном случае тело цикла будет выполняться неограниченное количество раз, что приведет к «зависанию» программы.

При использовании второй алгоритмической структуры (рис. 4.2.), если переменная *x* до наступления цикла принимает отрицательное значения, то с каждой итерацией она будет увеличиваться и, следовательно, при соответствующих значениях *y* тело цикла будет выполняться «бесконечно».

4.1. Инструкция *While ... Wend*

Инструкция *While ... Wend* используется для организации циклов с предусловием. Она имеет следующий синтаксис:

```
While условие
  тело цикла
Wend
```

Здесь *While .. Wend* – зарезервированные слова VBA,

условие – логическое выражение,

тело цикла – выполняемые в цикле инструкции.

Работу инструкции *While .. Wend* можно описать структурой схемой (рис. 4.3).

Оператор While ... Wend работает следующим образом:

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

2. После выполнения тела цикла программа снова выполняет проверку условия.

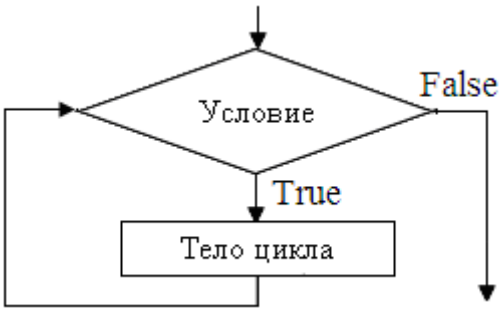


Рис. 4.3. Структурная схема выполнения инструкции *While .. Wend.*

Пример. Вывести все делители заданного натурального числа *N*:

Рассмотрим алгоритм нахождения делителей (рис. 4.4.). После ввода числа N в цикле переменной *d* присваиваем подряд все натуральные значения от 1 до *N*. В теле цикла проверяем условие. Если *d* является делителем числа *N*, то число *d* записываем в список *t*, и после увеличиваем *d* на единицу.

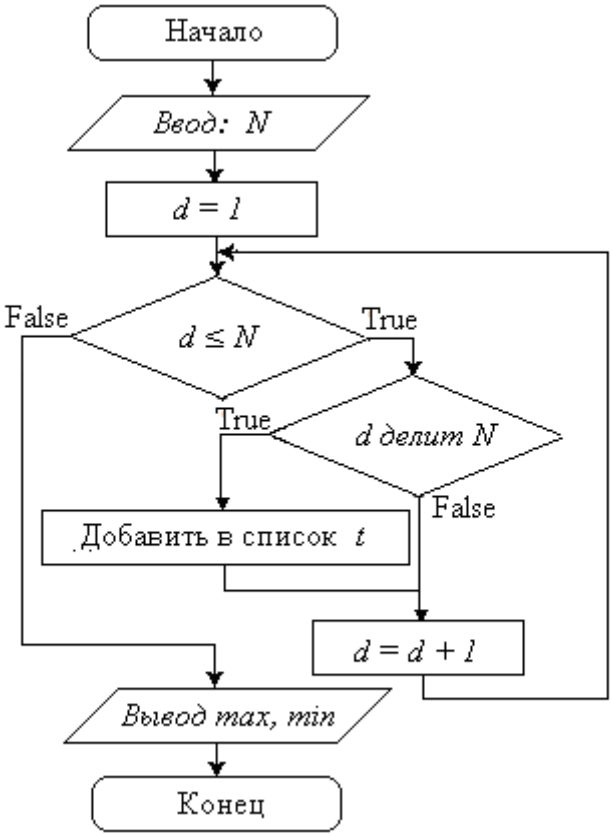


Рис. 4.4. Структурная схема нахождения делителей натурального числа N.

Данный алгоритм можно реализовать следующим программным кодом.

```
Sub Делители_While_Wend()
'Выводится список всех делителей числа N / инструкция 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) + ";"
    d = d + 1
  Wend
  t = " : " + Left(t, Len(t) - 1) + "."
  MsgBox "Делители числа " & N & t, , "Делители_While_Wend"
End Sub
```

В представленном листинге программы условие завершения цикла представлено строкой `While d <= N`, а условие делимости числа N на d – логическим выражением `N Mod d = 0`.

На рис. 4.5 представлен результат нахождения всех натуральных делителей числа 30000.

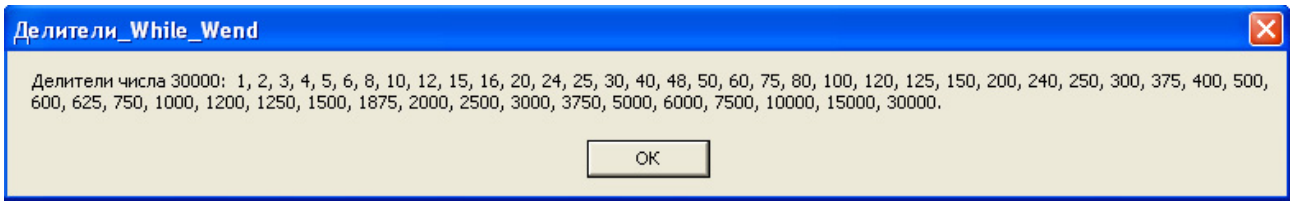


Рис. 4.5. Окно вывода делителей.

4.2. Инструкция *Do*

В языке Visual Basic for Application инструкция *Do* реализована конструкцией со следующим синтаксисом:

```
Do [{While | Until} условие]
  Тело цикла
Loop [{While | Until} условие]
```

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

Категория 1. Циклы с предусловием:

Do While .. Loop;

Do Until .. Loop.

Категория 2. Циклы с постусловием:

Do .. Loop While;

Do .. Loop Until.

4.2.1. *Do While .. Loop*

Jодной из конструкций *Do* является инструкция *Do While .. Loop*. Она является точным аналогом инструкции While ... Wend.

Синтаксис инструкции *Do While .. Loop* следующий:

```
Do While условие
  Тело цикла
Loop
```

Как и в случае инструкции *Do While .. Loop*, данная инструкция описывает цикл с предусловием, так как она тестирует условие до выполнения тела цикла (рис. 4.6).

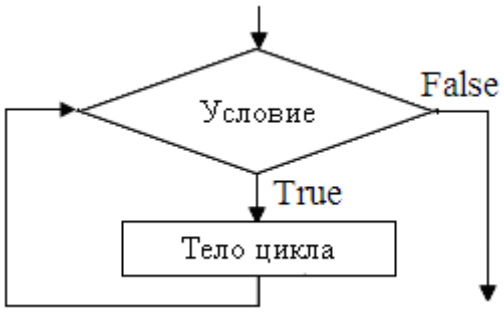


Рис. 4.6. Структурная схема выполнения инструкции *Do While .. Loop.*

Пример. Найти наибольший общий делитель (НОД) и наименьшее общее кратное (НОК) двух чисел.

Для решения первой части задачи воспользуемся алгоритмом Евклида: Согласно данному методу будем уменьшать каждый раз большее из чисел на величину меньшего до тех пор, пока оба значения не станут равными. Например, для чисел $x_1 = 64$ и $x_2 = 40$ процесс вычисления наибольшего общего делителя (x_1, x_2) решение будет выглядеть следующим образом.

	1-й шаг	2-й шаг	3-й шаг	4-й шаг	5-й шаг
X_1	64	64 – 40 = 24	24	24 – 16 = 8	8
X_2	40	40	40 – 24 =16	16	16 – 8 = 8

Нахождение наименьшего общего кратного [x_1, x_2] можно найти как частное от произведения $x_1 \cdot x_2$ двух чисел на их наибольший общий делитель (x_1, x_2). В нашем случае значение наименьшего общего кратного $[64,40] = 64 \cdot 40 / 8 = 320.$

Для реализации алгоритма будем использовать цикл с предусловием (рис. 21).

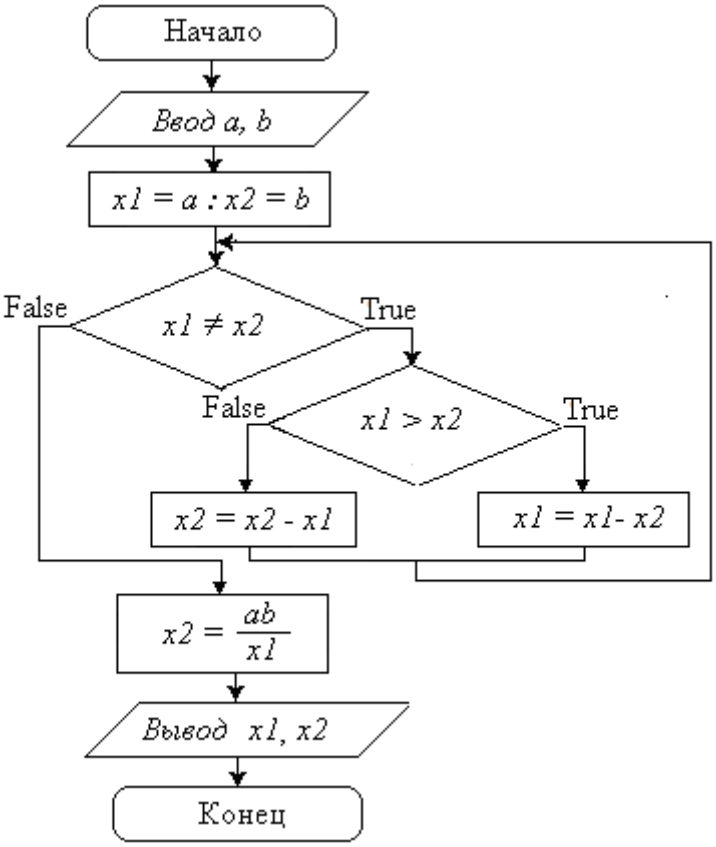


Рис. 4.7. Структурная схема нахождения НОД и НОК двух чисел.

В предложенной схеме во избежание потери вводимых данных значения *a* и *b* присваиваются переменных x_1 и x_2 и в ходе алгоритма Евклида изменяются значения новых переменных. Для вывода наибольшего общего делителя (*a*, *b*) возьмем значение переменной x_1, а переменную x_2 используем для расчета наименьшего общего кратного [*a*, *b*].

Представленный алгоритм (рис. 4.7) реализуем с использованием инструкции Do While .. Loop.

```
Sub Нахождение_НОК_и_НОД_двух_чисел()
' Определение наибольшего общего делителя и наименьшего
'общего кратного двух чисел / Инструкция Do While .. Loop
  Dim a As Long, b As Long
  a = InputBox("Введите первое число a", "Ввод начальных значений")
  b = InputBox("Введите второе число b", "Ввод начальных значений")
  x1 = a: x2 = b
  Do While x1 <> x2
    If x1 > x2 Then x1 = x1 - x2 Else x2 = x2 - x1
  Loop
  x2 = a * b / x1
  MsgBox "Наименьший общий делитель (a, b) = " & x1 & _
         ", наименьшее общее кратное [a, b] = " & x2 & ".", , _
         "Нахождение НОК и НОД двух чисел"
End Sub
```

На рис. 4.8. представлено окно вывода результата выполнения программы, при начальных значениях $a = 64$, $b = 40$.

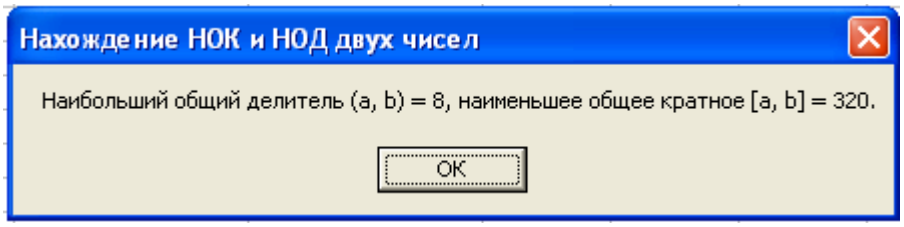


Рис. 4.8. Результат выполнения программы.

4.2.2. *Do Until .. Loop*

Еще одной конструкцией Do с предусловием является *Do Until ... Loop*. Синтаксис инструкции следующий:

```
Do Until условие
  Тело цикла
Loop
```

Инструкция *Do Until... Loop* также используется для организации 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 11. 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 $x_1 + x_2 > N$ (рис 23), 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.

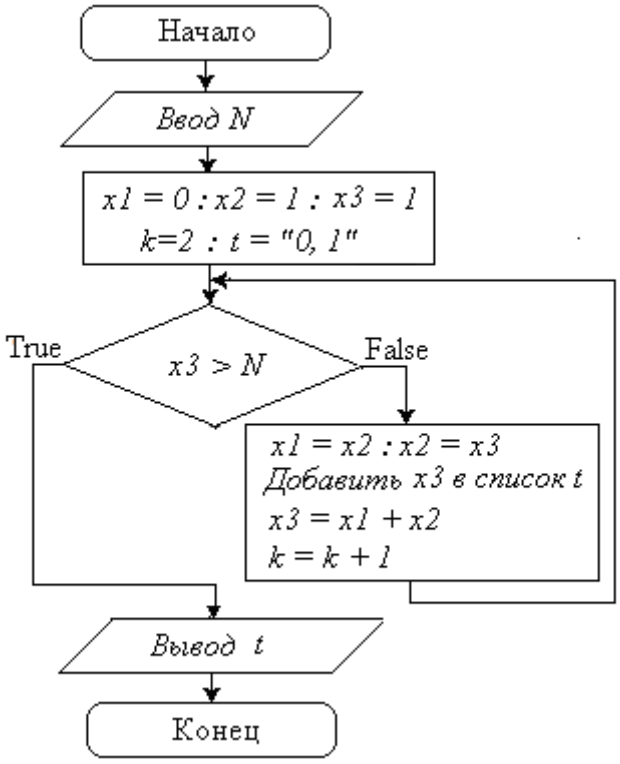


Рис. 4.9. Алгоритм вычисления чисел Фибоначчи.

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
```

На рис. 4.10 представлено окно вывода результата выполнения программы, при введенном значении *N*=100 000.

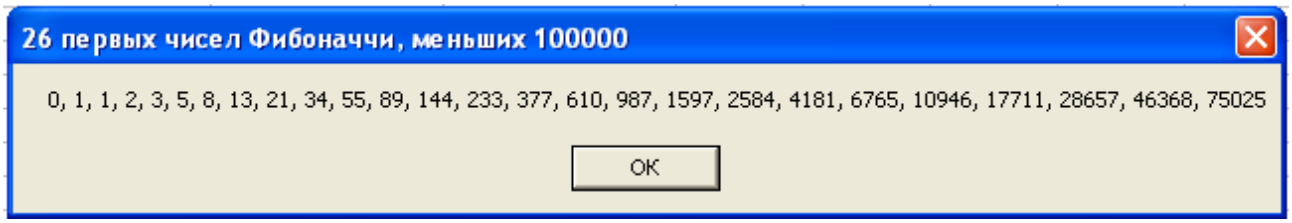


Рис. 4.10. Результат выполнения программы Числа_Фибоначчи.

4.2.3. *Do .. Loop While*

Одной из форм реализации цикла *Do* с постусловием является конструкция *Do .. Loop While*.

Ее синтаксис следующий:

```
Do
  Тело цикла
Loop While условие
```

Инструкция *Do... Loop While* тестирует условие после выполнения тела цикла. Если условие выполняется, то программа снова обращается к началу тела цикла. Следует отметить, что цикл с постусловием всегда будет выполнен хотя бы один раз.

Пример 12. Найти корень уравнения y=f(x) на интервале [a, b].

Задача численного нахождения корней нелинейных уравнений состоит из двух этапов:

3) *Отделение корней* – это нахождение числовых промежутков, в которых содержится только один корень.
4) *Уточнение корня* – это. вычисление отделенных корней с заданной точностью на выбранных интервалах.

Исследуя функцию $y = f(x)$ (рис. 4.11) 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.

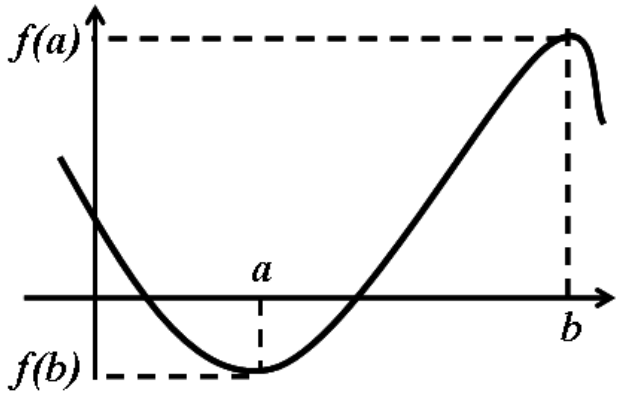


Рис. 4.11. Отделение корней уравнения $y = f(x)$.

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:

$$x_1 \in (-\infty,-1]; \quad x_{2\in}[1,+\infty).$$

Уменьшим промежутки, в которых находятся корни. Для этого выберем значения аргумента, в которых функция принимает положительные значения:

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

Рассмотрим алгоритм уточнения корня (рис. 4.12). Выберем один из интервалов с границами a и b, содержащий корень уравнения.

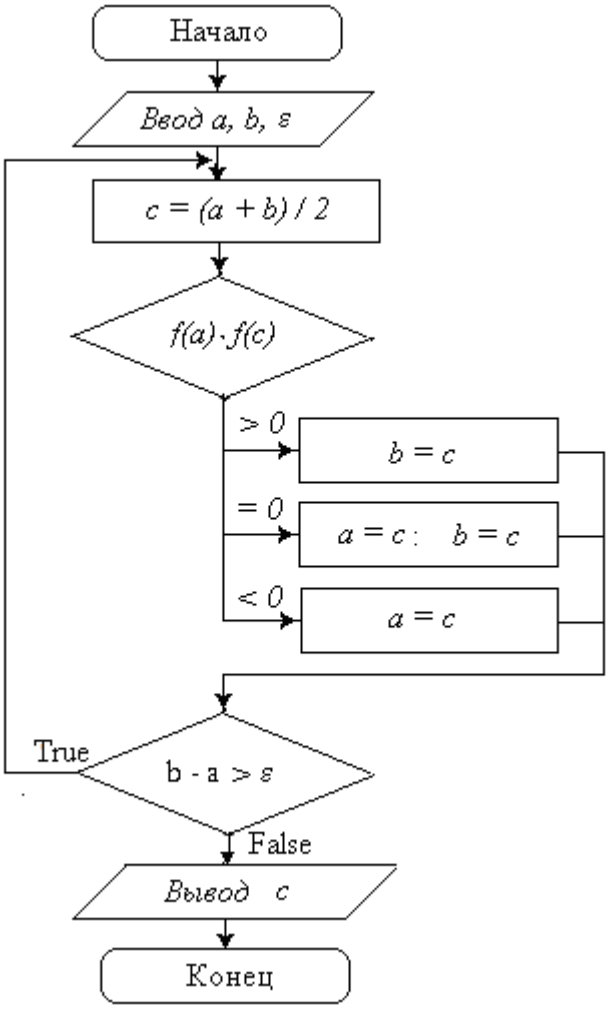


Рис. 4.12. Алгоритм нахождения корня методом половинного деления.

Разделим выбранный интервал пополам точкой $c = \frac{a+b}{2}$. Теперь возможны три случая.

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

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

3) Если *f(a)·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.

Перед написанием кода программы 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
```

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
```

На рис. 4.13 представлено окно вывода результата выполнения программы, при *a* = -10, b = -1, epsilon = 0,001.

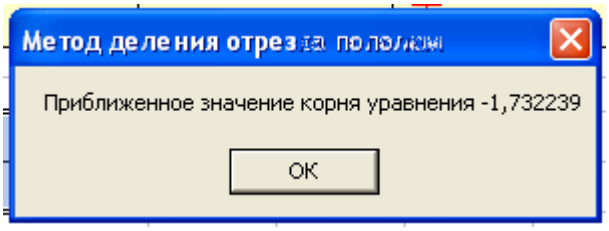


Рис. 4.13. Результат выполнения программы Метод_половинного_деления..

4.2.4. *Do .. Loop Until*

Еще одной конструкцией *Do* с постусловием является *Do .. Loop Until*. Ее синтаксис следующий:

```
Do
  Тело цикла
Loop Until условие
```

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

Пример. Определить количество цифр целой части действительного числа *X*.

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

Алгоритм подсчета цифр можно представить в виде структурной схемы (рис. 4.14). Во избежание потери значения переменной *X* ее значение присваивается переменной *M*, и все действия совершаются с новой переменной.

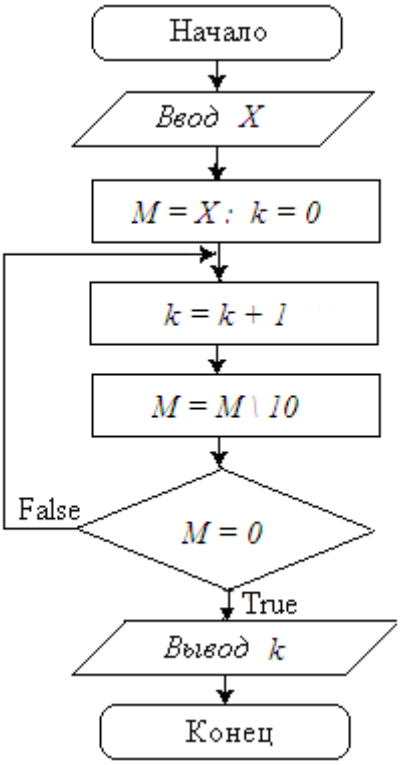


Рис. 4.14. Алгоритм нахождения количества цифр целой части числа.

Например, пусть X = -72865,1004384 тогда количество цифр k = 5. Результаты вычислений приведены ниже.

k	X = -72865,1004384
1	-72865,1004384 \ 10 = --7286
2	-7286 \ 10 = --728
3	-728 \ 10 = --72
4	-72 \ 10 = --7
5	-7 \ 10 = 0

Представленный алгоритм можно реализовать с использованием инструкции *Do... Loop Until*.

```
Sub Количество_цифр_целой_части_числа()
' Определение количества цифр целой части числа X.
' / Инструкция Do .. Loop Until
  Dim X As Single
  X = InputBox("Введите действительное число X", _
               "Определение количества цифр целой части числа")
  M = X
  Do
    k = k + 1
```

```
    M = M \ 10
  Loop Until M = 0
  MsgBox "Количество цифр целой части числа " & X & " равно " & k & ".", , _
         "Определение количества цифр числа"
End Sub
```

На рис. 4.15 представлено окно вывода результата выполнения программы, при введенном числе $X = 1078{,}6027384$.

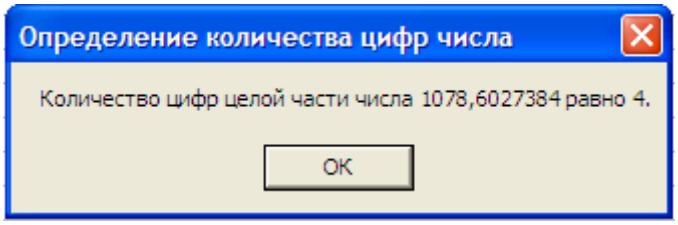


Рис. 4.15. Результат выполнения программы Количество_цифр_целой_части_числа.

4.3. Инструкция *For ... Next*

Инструкции циклов с предусловием и постусловием достаточно универсальны, но часто возникают более простые случаи, когда количество итераций в цикле заранее известно и тело цикла должно быть выполнено фиксированное количество раз. В таких случаях используется цикл с параметром *For ... Next.*

Синтаксис оператора следующий:

```
For counter = start To end [Step stepsize]
  [Тело цикла]
Next [counter]
```

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

counter – управляющая переменная (параметр) цикла;

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

Алгоритм работы инструкции *For ... Next* (рис. 30) заключается в следующих пунктах.

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 пункту 2.

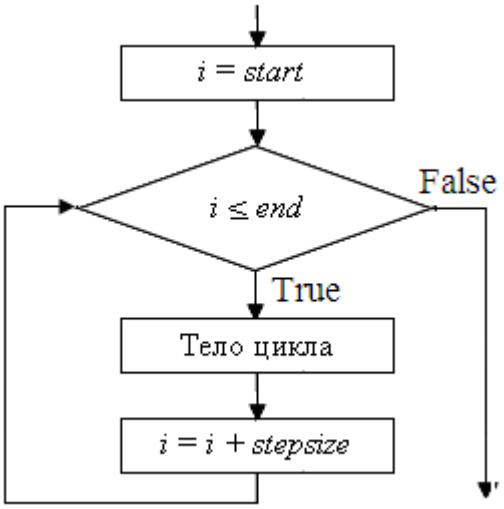


Рис. 4.16. Алгоритм работы инструкции *For ... Next.*

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

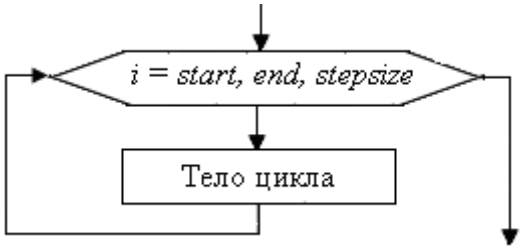


Рис. 4.17. Компактная схема алгоритма цикла с параметром *For ... Next.*

Алгоритмическую схему для цикла с параметром можно записать компактнее, с помощью специального блока (рис. 4.17).

Пример. Вычислить факториал *N*!

Напомним, что факториалом *N*! называется произведение натуральных чисел от единицы до *N*:

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

Например, $6! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 = 720$.

Следует помнить, что факториал нуля 0! = 1.

Очевидно, для расчета значения факториала *N*! необходимо выполнить равно *N* итераций, а, значит, удобно использовать цикл с параметром, изменяющимся от 1 до *N* .с шагом 1. Тело цикла будет состоять из одного действия вычисления нового значения факториала – умножения факториала, полученного на предыдущей итерации, на текущее значение параметра цикла (рис. 32).

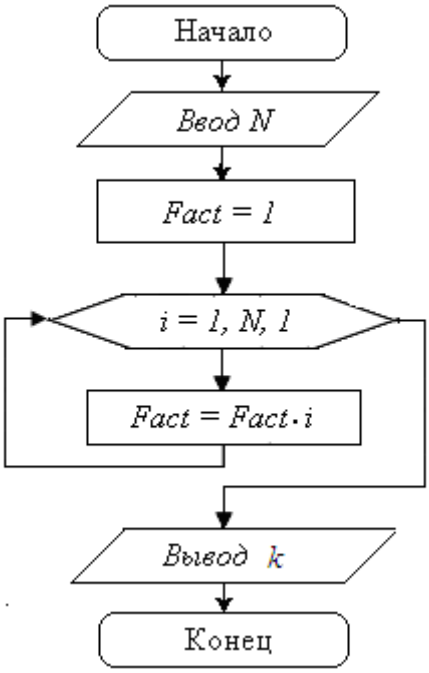


Рис. 4.18. Алгоритм нахождения факториала *N*!/

Программный код описанного алгоритма легко реализуется с помощью инструкции *For ... Next.* Заметим, что шаг приращения, равный единице, в строке *For i* = 1 to *N step* 1 можно не записывать. Тогда данная строка будет выглядеть так: *For i* = 1 to *N*.

```
Sub Факториал()
' Вычисление факториала N!.
' / Инструкция For .. Next (шаг равен единице)
  Dim N As Integer
  N = InputBox("Введите N", "Вычисление факториала N! ")
  Fact = 1
  For i = 1 To N
    Fact = Fact * i
  Next i
  MsgBox "Значение факториала " & N & "!  = " & Fact, , _
         "Вычисление факториала"
End Sub
```

На рис. 4.19 представлен результат выполнения программы при *N* = 12.

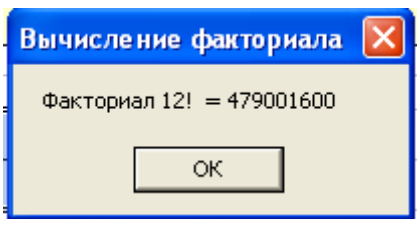


Рис. 4.19. Окно вывода результата вычисления факториала.

2.4. 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.

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

Алгоритм программы понятен (рис. 4.20). 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.

В приведенном алгоритме в цикл с параметром *r*, принимающем все целые значения от *m* до *n*, вложен цикл с двойным постусловием d *делит* r *или* $d \geq \sqrt{r}$.

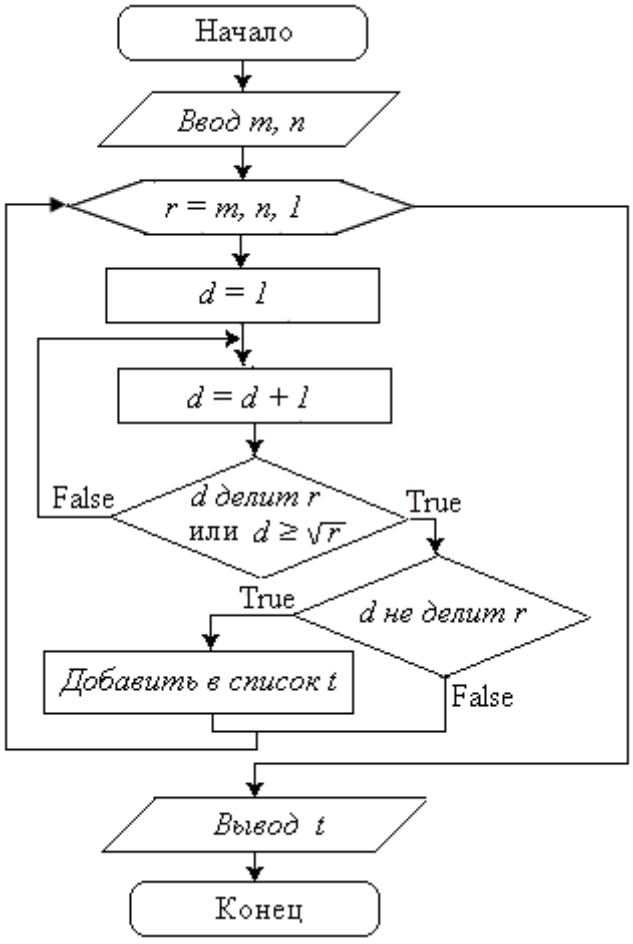


Рис. 4.20 Алгоритм нахождения простых чисел в заданном промежутке.

Во внутреннем цикле число *r* проверяем на делители *d* от 2 до $\sqrt{r}$. Если такой делитель найден (d *делит* r) или значение *d* равно или превышает *r* ($d \geq \sqrt{r}$), то вложенный цикл заканчивает свою работу, и в случае отсутствия делителя в список *t* помещается текущее значение *r*.

При написании программы 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
```

На рис. 4.21 представлен результат нахождения простых чисел в промежутке от 100 до 200.

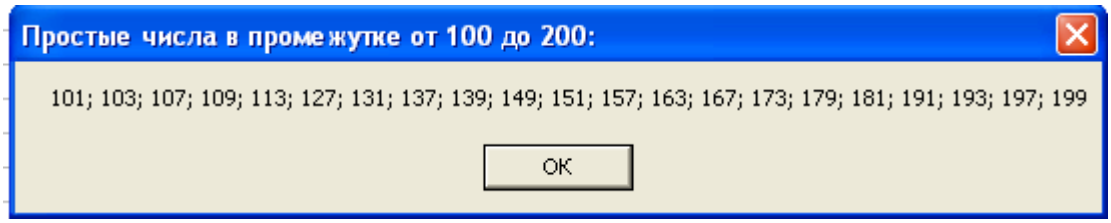


Рис. 4.21. Окно вывода результата выполнения программы Простые_числа.

Рассмотрим еще один пример вложенных циклов.

Пример. Найти приближенное значение интеграла $\int_a^b f(x)dx$.

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 $a = x_0 < x_1 < x_2 < \ldots < x_n = b$ (Рис. 36). 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.

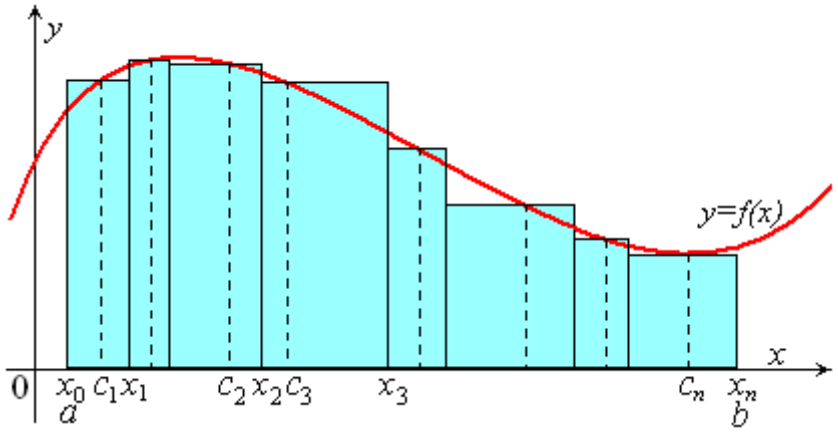


Рис. 4.22. Графическое представление определенного интеграла.

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

$$S_n = \sum_{i=1}^{n} f(c_i)(x_i - x_{i-1}) = \sum_{i=1}^{n} f(c_i)\Delta x_i \qquad (4.1)$$

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

Пусть $\max_i \Delta x_i = \lambda$. Тогда конечный предел интегральной суммы (5) при $n \to \infty$ и $\lambda n \to 0$ называется *определенным интегралом* от функции *f(x)* на отрезке [*a; b*] и обозначается

$$\lim_{\substack{n\to\infty \\ \lambda_n \to 0}} S_n = \int_a^b f(x)dx . \qquad (4.2)$$

Если количество разбиений *n* отрезка интегрирования [*a*; *b*] на элементарные отрезки конечно, но достаточно велико, то, суммируя *n* площадей прямоугольников, получим приближенное значение интеграла $\int_a^b f(x)dx$.

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

Для определенности точки c_i берут либо на левых границах элементарных отрезков $(c_i = x_{i-1})$, на правых границах отрезков $(c_i = x_i)$ либо в серединах отрезков $\left(c_i = \dfrac{x_i + x_{i-1}}{2}\right)$. В зависимости от выбора точек описанные способы приближенного интегрирования подразделяются на методы левых, правых и средних прямоугольников.

Возьмем, например, метод правых прямоугольников. Задача нахождения определенного интеграла, таким образом, сводится к нахождению конечной суммы площадей прямоугольников

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} f(x_i)\frac{b-a}{n}. \qquad (7)$$

Для организации автоматизированного вычисления выражения (7) следует организовать цикл с параметром. Ниже приведены алгоритм (рис. 37) и соответствующая ему часть кода программы.

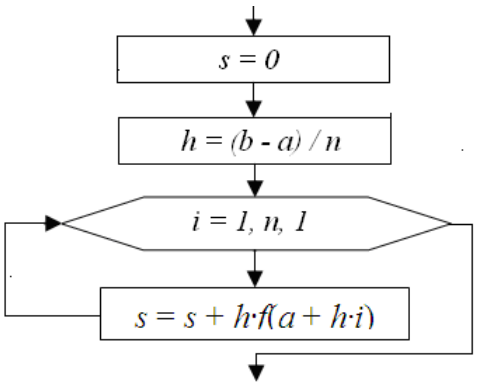


Рис. 4.23. Алгоритм реализации метода правых прямоугольников.

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

Важная проблема, возникающая при создании численных алгоритмов нахождения определенных интегралов заключается в расчете количества разбиений n. На практике можно использовать следующий способ.

Задается начальное значение количества разбиений n отрезка $[a, b]$, например, $n = 10$. Затем по изложенному выше алгоритму находится приближенное значение интеграла S_0 для данного разбиения. Увеличиваем количества разбиений n в два раза. Вследствие этого шаг интегрирования h в два раза уменьшится, следовательно, точность вычислений будет больше. Рассчитаем новое значение интеграла S и сравним его с S_0, полученным ранее. Если абсолютное значение разности $|S - S_0|$ будет меньше заданной погрешности $\varepsilon>0$, то приближенное значение найдено, в противном случае снова увеличиваем количество разбиений n в два раза.

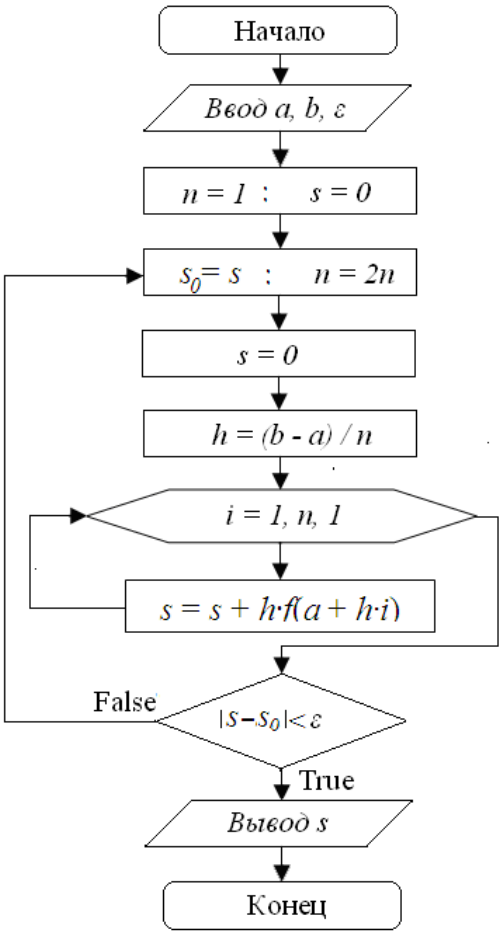


Рис. 4.24. Алгоритм метода правых прямоугольников с определением количества разбиений.

Алгоритм нахождения приближенного значения определенного интеграла методом правых прямоугольников с определением количества разбиений описан в виде структурной схемы (рис. 4.24).

Предложенный алгоритм можно реализовать следующим программным кодом.

```
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
```

На рис. 4.25 представлен результат нахождения приближенного значения определенного интеграла $\int_{-2}^{3} \frac{4x^3 \sin(5x-3)}{x^2-10} dx$ методом правых прямоугольников с автоматическим определением количества разбиений и заданной погрешностью ε = 0,001.

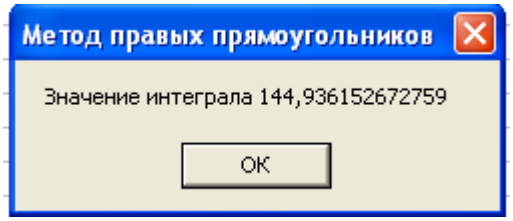


Рис. 4.25. Окно вывода результата выполнения программы Метод_правых_прямоугольников.

Раздел 5. Обработка массивов

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.

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

Массив, представленный на рисунке 5.1, имеет 6 элементов, каждый элемент сохраняет число вещественного типа, элементы в массиве пронумерованы от 1 до 6. Такого рода массив, представляющий собой просто список данных одного типа, называют *простым*, или *одномерным массивом*.

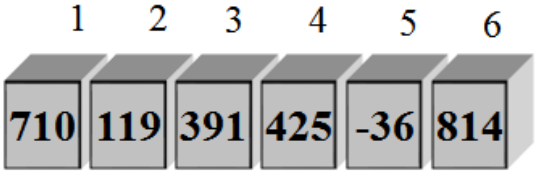


Рис.5.1. Одномерный массив.

Одномерный массив можно представить в виде таблицы, состоящей из одной строки.

№ элемента	1	2	3	4	5	6
Значение	710	119	391	425	-36	814

Рис. 5.2. Табличное представление одномерного массива.

Для доступа к данным, хранящимся в определённом элементе массива, необходимо указать *имя массива* и порядковый номер этого элемента, называемый *индексом*. Пусть имя представленного массива будет *a*, тогда значение элемента массива a_2 равно 119, а значение элемента a_6 равно 814.

Если возникает необходимость хранения данных в виде таблиц, в формате строк и столбцов, то необходимо использовать *двумерные массивы*. На рисунке 5.2. представлен двумерный массив, состоящий из тридцати элементов, каждый из которых определяется двумя индексами.

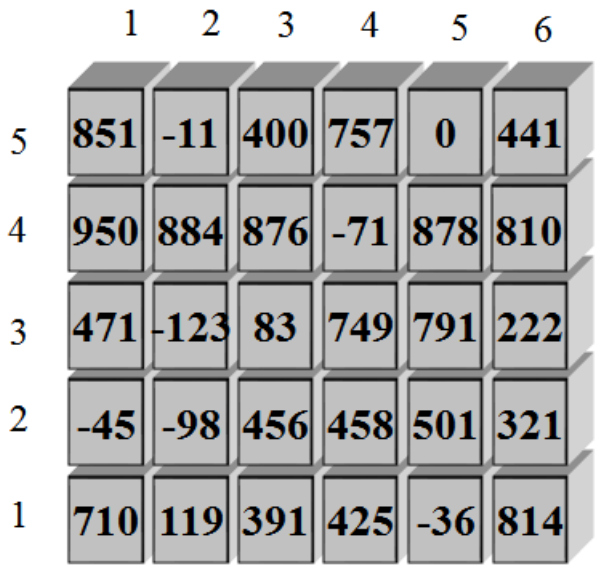


Рис.5.2. Двумерный массив.

Представленный массив можно описать в виде таблицы, состоящей из пяти строк и шести столбцов.

индексы	1	2	3	4	5	6
1	710	119	391	425	-36	814
2	-45	-98	456	458	501	321
3	471	-123	83	749	791	222
4	950	884	876	-71	878	810
5	851	-11	400	757	0	441

Строки в массиве 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. Например, если представляемый на рис 5.2. массив имеет имя *b*, то значение элемента b_{11} будет равено 710, а значение элемента b_{43} равно 876.

Можно хранить данные и в *трехмерном* массиве (рис 5.3). Слои в таком массиве можно считать первым измерением, строки – вторым, а столбцы – третьим. Для доступа к данным, хранящимся в элементе массива, необходимо указать имя массива и три индекса: номер слоя, номер строки и номер столбца. Например, если представляемый на рис 5.3. массив имеет имя *c*, то значение элемента c_{111}, находящегося в первом слое, в первой строке, в первом столбце, будет равно 710, значение элемента со второго слоя c_{253} равно 97, а элемент третьего слоя c_{326} будет равен 20.

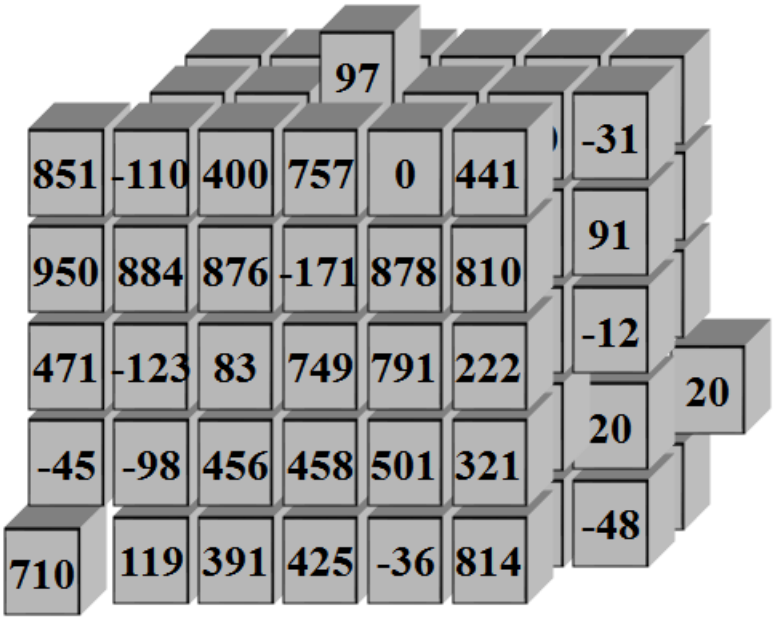


Рис.5.3. Трехмерный массив.

Такие массивы можно представлять в табличном виде, объединяя двумерные представления для каждого слоя.

индексы		1	2	3	4	5	6
1	1	710	119	391	425	-36	814
	2	-45	-98	456	458	501	321
	3	471	-123	83	749	791	222
	4	950	884	876	-71	878	810
	5	851	-11	400	757	0	441
2	1	80	19	31	24	61	-48
	2	-34	18	45	35	51	20
	3	46	-11	27	34	77	-12
	4	75	7	32	-31	88	91
	5	47	-92	97	66	100	-31
3	1	0	9	3	2	16	-11
	2	-3	8	14	5	1	20
	3	4	-9	25	31	7	-2
	4	2	6	12	-3	8	1
	5	4	-9	7	6	0	-1

При создании программных приложений можно использовать массивы и большей размерности.

5.2. Объявление массивов

Для объявления массива служит служебное слово **Dim**.

Формат оператора **Dim** следующий

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.

Примеры.

Dim a(1 To 100) **As** Integer

Dim b(0 To 50, 0 To 20) **As** Double

Dim c(1 To 5, 1 To 7, 1 To 12) **As** Byte

Dim d(1 To 5, 1 To 7, 1 To 12) **As** long

Dim e(1 To 3, 1 To 3, 1 To 3) **As** String

Dim f(0 To 4, 0 To 6, 1 To 12, 0 To 5)

Dim a(100) **As** Integer

Dim b(50, 20) **As** Double

Dim c(5, 7, 12) **As** Byte

Dim d(5, 7, 12) **As** long

Dim e(3, 3, 3) **As** String

Dim f(4, 6, 12, 5)

Пример. Нахождение наибольшего *Max* и наименьшего *Min* из элементов числового массива и их номеров *NoMax*, *NoMin.*

Алгоритм решения задачи следующий.

Объявим переменную *N* (количество обрабатываемых элементов массива) и целочисленный массив *x.*

```
Dim n As Byte, x(255) As Integer
```

Инструкцией *InputBox* введем количество элементов массива, а в цикле с параметром *i* сгенерируем элементы массива.

```
n = InputBox("Введите количество элементов массива")
Randomize
For i = 1 To n
  x(i) = Rnd(Timer) * 200 - 100
  t = t + Str(x(i)) + "; "
Next i
```

Предположим, что первый элемент массива является максимальным, и запишем его в переменную *Max*, а в *NoMax* его номер (число 1).

Аналогично, первый элемент может быть и минимальным запишем его в переменную *Min*, а в *NoMin* его номер (число 1).

```
NoMax = 1
NoMin = 1
Max = x(1)
Min = x(1)
```

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 *NoMax* текущее значение индекса i.

Таким же образом, если текущий элемент массива оказывается меньше минимального, то записываем его в переменную *Min*, а в переменную *NoMin* текущее значение индекса i.

```
For i = 2 To n
  If x(i) > Max Then
    NoMax = i
    Max = x(i)
  End If
  If x(i) < Min Then
    NoMin = i
    Min = x(i)
  End If
Next i
```

Инструкцией *MsgBox* выводим весь массив, максимальное и минимальное значения элементов массива.

```
MsgBox (t & vbCr & "Максимальный элемент x(" & NoMax & ") = " & Max
        & vbCr & "Минимальный элемент x(" & NoMin & ") = " & Min)
```

Ниже приведено окно вывода результатов программы при *N*= 30.

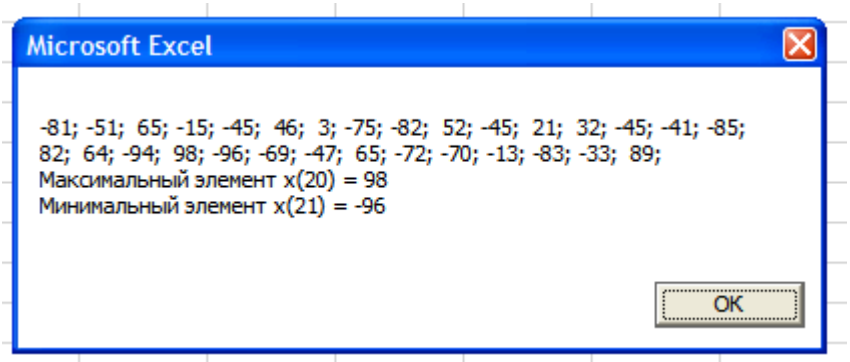


Рис. 5.4. Окно вывода результатов программы при *N*= 30

Пример. Вывод всех положительных, отрицательных, четных, нечетных элементов массива.

Алгоритм решения задачи следующий.

Объявим переменную *N* (количество обрабатываемых элементов массива) и целочисленный массив *x*. Затем, как и в прошлой задаче введем количество элементов массива, сгенерируем целочисленный массив и запишем его элементы в строковую переменную *t*.

```
Sub Нахождение_четных_нечентных_положит_отрицат_элементов()
Dim n As Byte, x(255) As Integer
n = InputBox("Введите количество элементов массива")
Randomize
For i = 1 To n
  x(i) = Rnd(Timer) * 200 - 100
  t = t + Str(x(i)) + "; "
Next i
```

Затем в цикле с параметром с помощью структур *if* будем анализировать каждый элемент массива.

Если x(i) Mod 2 = 0, тогда элемент массива четный;

если x(i) Mod 2 = 1, тогда элемент массива нечетный;

при выполнении условия x(i) > 0, элемент массива положительный,

а при x(i) < 0, элемент массива отрицательный.

```
For i = 1 To n
  If x(i) Mod 2 = 0 Then t1 = t1 + Str(x(i)) + "; "
  If x(i) Mod 2 = 1 Then t2 = t2 + Str(x(i)) + "; "
  If x(i) > 0 Then t3 = t3 + Str(x(i)) + "; "
  If x(i) < 0 Then t4 = t4 + Str(x(i)) + "; "
Next i
```

В каждом из случаев помещаем значения массивов с соответствующие переменные *t1, t2, t3, t4*.

Затем инструкцией *MsgBox* выводим весь массив, четные, нечетные, положительные и отрицательные элементы массива.

```
MsgBox ("Весь массив: " & t & vbCr & _
        "Четные элементы:" & t1 & vbCr & _
        "Нечетные элементы:" & t2 & vbCr & _
        "Положительные элементы:" & t3 & vbCr & _
        "Отрицательные элементы:" & t4)
```

Ниже приведено окно вывода результатов программы при *N*= 20.

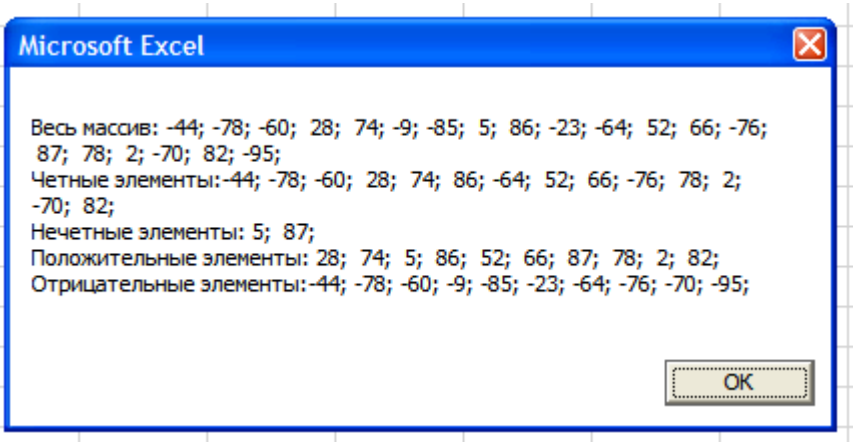


Рис. 5.5. Окно вывода результатов программы.

5.3. 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
```

В первом случае значения массива вводятся вручную с помощью инструкции InputBox. Второй фрагмент программы вводит элементы массива с листа MS EXCEL.

```
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
```

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

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
```

Пример. Вставка в числовой массив нового элемента ***b*** на позицию ***k*** со сдвигом элементов.

Алгоритм решения задачи следующий.

Объявим переменную *N* (количество обрабатываемых элементов массива) и целочисленный массив *x*. Затем введем количество элементов массива, сгенерируем целочисленный массив и запишем его элементы в строковую переменную *t*. Выведем массив на лист MS EXCEL

```
Sub вставка_нового_элемента()
Dim n As Byte, k As Byte, x(255) As Integer
n = InputBox("Введите количество элементов массива")
Randomize
For i = 1 To n
  x(i) = Rnd(Timer) * 200 - 100
  t = t + Str(x(i)) + "; "
Next i
Cells(1, 1) = t
```

Введем значения нового элемента ***b*** и позицию вставки ***k***.

```
b = InputBox("Введите вставляемый элемент")
k = InputBox("Введите позицию вставляемого элемента")
```

В цикле с параметром переместим все элементы, начиная с позиции ***k*** вправо. В позицию ***k*** поместим число ***b***. Затем увеличим на единицу количество просматриваемых элементов массива.

```
For i = n To k Step -1
  x(i + 1) = x(i)
Next i
x(k) = b
n = n + 1
```

Массив выведем также на на лист MS EXCEL, но строкой ниже.

```
 t = ""
For i = 1 To n
  t = t + Str(x(i)) + "; "
Next i
Cells(2, 1) = t
```

Ниже приведен вывод результатов выполнения программы при N=20, b=5, k=10.

	A	B	C	D	E	F	G	H
1	-6; -29; 56; 33; -57; 78; -81; -49; -74; -25; -56; -24; 2; -58; -92; -61; -6; -27; 40; -82;							
2	-6; -29; 56; 33; -57; 78; -81; -49; -74; 55; -25; -56; -24; 2; -58; -92; -61; -6; -27; 40; -82;							
3								

Рис. 5.6. Результаты выполнения программы.

Пример. Удаление из массива элементов с номерами от *m1* до *m2*.

Алгоритм решения задачи следующий.

Объявим переменную *N* (количество обрабатываемых элементов массива) и целочисленный массив *x*. Затем, как и в прошлой задаче, введем количество элементов массива, сгенерируем целочисленный массив и запишем его элементы в строковую переменную *t*. Выведем массив на лист MS EXCEL.

```
Sub удаление_элементов()
Dim n As Byte, k As Byte, x(255) As Integer
n = InputBox("Введите количество элементов массива")
Randomize
For i = 1 To n
  x(i) = Rnd(Timer) * 200 - 100
  t = t + Str(x(i)) + "; "
Next i
Cells(1, 1) = t
```

Введем значения интервала удаляемых элементов.

```
m1 = InputBox("Введите номер перваго из удаляемых элементов")
m2 = InputBox("Введите номер последнего из удаляемых элементов")
```

Рассчитаем сдвиг элементов, стоящих после удаляемого интервала и в цикле с параметром перенесем данные влево на ***k*** позиций. Затем уменьшим размер просматриваемой области массива на ***k***.

```
k = m2 - m1 + 1
For i = 0 To k - 1
  x(m1 + i) = x(m1 + i + k)
Next i
n = n - k
```

В конце программы выведем все значения на лист MS EXCEL.

```
t = ""
For i = 1 To n
  t = t + Str(x(i)) + "; "
Next i
Cells(2, 1) = t
```

Ниже приведено окно вывода результатов программы при *N*= 12, m1=4, m2=8.

	A	B	C	D	E
1	-17; -11; 40; -30; 70; 86; -69; 2; -89; 8; 81; 50;				
2	-17; -11; 40; -89; 8; 81; 50;				

Рис. 5.7. Вывод результатов программы.

5.4. Сортировка элементов одномерного массива

*Сортировка п*редставляет 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:

$x_1 \le x_2 \le \ldots \le x_n$.

При сортировке по убыванию после перестановки элементы с меньшими индексами должны быть не меньше элементов с большими индексами:

$$x_1 \ge x_2 \ge \ldots \ge x_n$$

5.4.1. Сортировка простыми обменами

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.

Таблица 5.1.

Результаты выполнения алгоритма пузырьковой сортировки

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

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

5.4.2. 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.

Таблица 5.2.

Результаты выполнения алгоритма шейкерной сортировки

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		

В нашем примере сортировка массива осуществлена не в **5**, а в **3** прохода. С каждым новым проходом количество обрабатываемых элементов уменьшается.

Пример. Сортировка одномерного массива пузырьковым методом. (по возрастанию, по убыванию).

Алгоритм решения задачи следующий.

Разместим номера элементов одномерного массива и их значения в верхнем левом углу листа MS Excel.

	1	2	3	4	5	6	7	8	9	10	11	12
1	№ элемента	1	2	3	4	5	6	7	8	9	10	
2	Исходный массив	8	15	-2	0	40	10	-3	-5	20	-20	
3												

Рис. 5.7. Размещение номеров и элементов массива.

Объявим одномерный массив *a* для помещения в него сортируемых значений. Определим количество сортируемых элементов, например, 10. Затем в цикле с параметром *i* последовательно присвоим элементам массива *a* соответствующие значения ячеек второй строки листа.

```
Sub Пузырьковый_метод_сортировки()
Dim a(100) As Integer
                        'количество элементов в массиве
n = 10
                        'ввод массива
For i = 1 To n
  a(i) = Cells(2, i + 1)
Next i
```

Согласно «пузырьковому методу» необходимо осуществить $n - 1$ проход, в каждом из которых последовательно, слева-направо, сравниваются попарно все соседние элементы массива. Если порядок в паре неверный, то есть левый элемент в паре больше правого, то эта пара элементов меняется местами.

Данный алгоритм можно реализовать следующим фрагментом программы.

```
                    'Сортировка n-1 проходов
For i = 1 To n - 1
  For j = 1 To n - 1
    If a(j) > a(j + 1) Then
      p = a(j): a(j) = a(j + 1): a(j + 1) = p
    End If
  Next j
```

Следует обратить внимание на то, что обмен значений двух элементов осуществляется с помощью дополнительной переменной p, которая используется как буфер. Можно провести аналогию.

Если необходимо поменять содержимое двух емкостей, например, разных жидкостей в двух стаканах, то можно воспользоваться третьим пустым стаканом: перелить в него жидкость из первого стакана, в пустой первый стакан поместить содержимое второго, и, наконец, из третьего стакана перелить жидкость во второй. Этот алгоритм и называется «методом трех стаканов».

В нашем случае роль первого стакана выполняет элемент массива a_j, второй стакан – это a_{j+1}, а буфером является переменная p.

После сортировки массива выведем в строку $i + 2$ номер прохода, а затем в цикле с параметром j все значения массива в измененном порядке.

```
                    'вывод массива на каждом проходе
    Cells(i + 2, 1) = Str(i) + "-й проход"
    For j = 1 To n
      Cells(i + 2, j + 1) = a(j)
    Next j

Next i
```

Ниже представлен лист MS TXCEL с выведенным на него массивом после каждого прохода.

	1	2	3	4	5	6	7	8	9	10	11	12
1	№ элемента	1	2	3	4	5	6	7	8	9	10	
2	**Исходный массив**	**8**	**15**	**-2**	**0**	**40**	**10**	**-3**	**-5**	**20**	**-20**	
3	1-й проход	8	-2	0	15	10	-3	-5	20	-20	40	
4	2-й проход	-2	0	8	10	-3	-5	15	-20	20	40	
5	3-й проход	-2	0	8	-3	-5	10	-20	15	20	40	
6	4-й проход	-2	0	-3	-5	8	-20	10	15	20	40	
7	5-й проход	-2	-3	-5	0	-20	8	10	15	20	40	
8	6-й проход	-3	-5	-2	-20	0	8	10	15	20	40	
9	7-й проход	-5	-3	-20	-2	0	8	10	15	20	40	
10	8-й проход	-5	-20	-3	-2	0	8	10	15	20	40	
11	9-й проход	-20	-5	-3	-2	0	8	10	15	20	40	
12												

Рис. 5.8. Вывод результатов после каждого из 9-ти проходов.

5.4.3. 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 двум.

Рассмотрим алгоритм сортировки выбором на примере сортировки по возрастанию.

Таблица 5.3.

Результаты выполнения алгоритма сортировки выбором

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				

Как мы видим, количество проходов в данном методе равно $n - 1$.

5.4.4. Сортировка вставками

Одним из видов сортировки является сортировка вставками. Алгоритм метода следующий.

Представим, что массив состоит только из одного элемента, следовательно, он уже отсортирован.

Берем второй элемент массива и сравниваем его с первым. Если он меньше первого, тогда этот элемент становится первым, а бывший первый сдвигается на второе место.

Затем берем третий элемент массива и последовательно сравниваем его с первыми двумя. Если этот элемент меньше первого или второго, то вставляем его в нужное место, а оставшуюся часть массива передвигаем на одну позицию вправо.

Таким образом мы продолжаем вставлять все элементы до последнего включительно. Важно обратить внимание на следующие правила:

1. Вставка в массив осуществляется от второго до последнего элемента.
2. Количество проходов равно $n - 1$.
3. Просматриваемая часть массива на i-том проходе содержит $i - 1$ элементов.
4. После вставки соответствующего элемента сравнивать его с последующими элементами не имеет смысла.

Рассмотрим алгоритм сортировки вставками на нашем примере.

Таблица 5.4.

Результаты выполнения алгоритма сортировки вставками

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

В нашем примере сортировка массива осуществлена в 5 проходов, однако в алгоритме происходит экономия как на объеме просматриваемых элементов, так и на количестве обменов.

Пример. Сортировка одномерного массива методом выбора (по возрастанию, по убыванию).

Алгоритм решения задачи следующий.

Разместим номера элементов одномерного массива и их значения в верхнем левом углу листа MS Excel.

	1	2	3	4	5	6	7	8	9	10	11	12
1	№ элемента	1	2	3	4	5	6	7	8	9	10	
2	Исходный массив	8	15	-2	0	40	10	-3	-5	20	-20	
3												

Рис. 5.9. Размещение номеров и элементов массива.

Объявим одномерный массив *a* для помещения в него сортируемых значений. Определим количество сортируемых элементов, например, 10. Затем в цикле с параметром *i* последовательно присвоим элементам массива *a* соответствующие значения ячеек второй строки листа.

```
Sub Пузырьковый_метод_сортировки()
Dim a(100) As Integer
                    'количество элементов в массиве
n = 10
                    'ввод массива
For i = 1 To n
  a(i) = Cells(2, i + 1)
Next i
```

Согласно методу выбора необходимо осуществить $n - 1$ проход, , в каждом из которых последовательно, слева-направо, сравниваются попарно все соседние элементы массива. Если порядок в паре неверный, то есть левый элемент в паре больше правого, то эта пара элементов меняется местами.

Данный алгоритм можно реализовать следующим фрагментом программы.

```
                    'Сортировка n-1 проходов
For i = 1 To n - 1
  For j = 1 To n - 1
    If a(j) > a(j + 1) Then
      p = a(j): a(j) = a(j + 1): a(j + 1) = p
    End If
  Next j
```

Следует обратить внимание на то, что обмен значений двух элементов осуществляется с помощью дополнительной переменной p, которая используется как буфер. Можно провести аналогию. Если необходимо поменять содержимое двух емкостей, например, разных жидкостей в двух стаканах, то можно воспользоваться третьим пустым стаканом: перелить в него жидкость из первого стакана, в пустой первый стакан поместить содержимое второго, и, наконец, из третьего стакана перелить жидкость во второй. Этот алгоритм и называется «методом трех стаканов».

В нашем случае роль первого стакана выполняет элемент массива a_j, второй стакан – это a_{j+1}, а буфером является переменная p.

После сортировки массива выведем в строку $i + 2$ номер прохода, а затем в цикле с параметром j все значения массива в измененном порядке.

```
                    'вывод массива на каждом проходе
  Cells(i + 2, 1) = Str(i) + "-й проход"
  For j = 1 To n
    Cells(i + 2, j + 1) = a(j)
  Next j

Next i
```

Ниже представлен лист MS TXCEL с выведенным на него массивом после каждого прохода.

	1	2	3	4	5	6	7	8	9	10	11	12
1	№ элемента	1	2	3	4	5	6	7	8	9	10	
2	**Исходный массив**	**8**	**15**	**-2**	**0**	**40**	**10**	**-3**	**-5**	**20**	**-20**	
3	1-й проход	8	-2	0	15	10	-3	-5	20	-20	40	
4	2-й проход	-2	0	8	10	-3	-5	15	-20	20	40	
5	3-й проход	-2	0	8	-3	-5	10	-20	15	20	40	
6	4-й проход	-2	0	-3	-5	8	-20	10	15	20	40	
7	5-й проход	-2	-3	-5	0	-20	8	10	15	20	40	
8	6-й проход	-3	-5	-2	-20	0	8	10	15	20	40	
9	7-й проход	-5	-3	-20	-2	0	8	10	15	20	40	
10	8-й проход	-5	-20	-3	-2	0	8	10	15	20	40	
11	9-й проход	-20	-5	-3	-2	0	8	10	15	20	40	
12												

Рис.5.10. Вывод результатов после 9-ти итераций.

5.5. Решение типовых задач

Рассмотрим несколько пример решения типовых задач.

Пример. Найти произведение двух сопряженных матриц заданных порядков.

Алгоритм решения задачи следующий.

Объявим три массива: массивы *a* и *b* будут использоваться для помещения элементов сопряженных перемножаемых матриц, а массив *c* включать результат матричного умножения.

```
Sub произведение_матриц()
Dim a(100, 100) As Integer
Dim b(100, 100) As Integer
Dim c(100, 100) As Integer
```

Зададим порядки m и n первой матрицы и, используя вложенные циклы, генерируем и выводим значения элементов первого массива.

```
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
```

Для второй матрицы количество строк равно количеству столбцов первой матрицы, поэтому введем только второй порядок матрицы *B*. Генерация эле-

ментов происходит аналогично. Следует отметить, что элементы второго массива будут выведены на лист MS EXCEL через n+1 столбцов.

```
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
```

Расчет и вывод произведения матриц осуществим с помощью трех вложенных друг в друга циклов с параметрами, здесь используется рассмотренная ранее формула $c_{ij} = \sum_{k} a_{ik} \cdot b_{kj}$.

```
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
```

Ниже приведен вывод результатов выполнения программы при N=20, b=5, k=10.

	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													

Рис. 5.11. Результаты выполнения программы.

Пример. Нахождение определителей *n*-го порядка.

Алгоритм решения задачи следующий. Будем вычислять определитель матрицы используя следующую теорему.

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 $\boldsymbol{\pi}$ 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:

$$\pi = \begin{pmatrix} 1 & 2 & 3 & \dots & n \\ i_1 & i_2 & i_3 & \dots & i_n \end{pmatrix}.$$

Рассмотрим алгоритм нахождения определителя четвертого порядка.

Объявим двумерный массив *A* для помещения в него элементов матрицы и вспомогательный одномерный массив *m*. Ввод массива произведем с листа MS EXCEL

```
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
```

Вычислительный процесс сводится к выбору всевозможных произведений элементов массива, взятых по одному из каждой строки и проверки условия несовпадения содержащих элементы столбцов. Данный алгоритм реализуется четырьмя вложенными друг в друга циклов с параметрами *i, j, k, l* и условиями их неравенства. Кроме того, для определения количества инверсий под-

становки $\pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ i & j & k & l \end{pmatrix}$, будем использовать еще два вложенных цикла с параметрами *x* и *y*.

```
                    '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
```

Ниже приведено окно вывода результатов программы при *N*= 12, m1=4, m2=8.

	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							

Рис. 5.12. Окно вывода результатов программы

Пример. Решение систем линейных алгебраических уравнений методом Гаусса.

Алгоритм решения задачи следующий.

Разместим значения элементов расширенной матрицы в верхнем левом углу листа MS Excel.

	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						

Рис. 5.13. Размещение исходных данных.

Объявим массив для помещения в него коэффициентов СЛАУ, а также переменные *M* и *N*, определяющие количеств уравнений и количество переменных соответственно.

```
Dim a(100, 100) As Double
Dim M As Byte, N As Byte
```

Определим значения переменных *M* и *N*, например, так:

```
M = 5: N = 5                    'Количество уравнений и переменных
```

Во вложенных циклах с параметрами *i* и *j* считаем коэффициенты расширенной матрицы:

```
For i = 1 To M                  'Считывание элементов
  For j = 1 To N + 1
    a(i, j) = Cells(i, j)
  Next j
Next i
```

Установим позицию печати для выводимых на каждом шаге значений преобразованной матрицы.

```
s = M + 3                        'Установкв позиции печати
```

Для упрощения текста программы определим подпрограмму для обмена значениями двух переменных. Эту подпрограмму будем вызывать достаточно часто в основном модуле.

```
Sub swap(x, y)
  p = x
  x = y
  y = p
End Sub
```

Прямой ход метода Гаусса начинается с проверки текущих диагональных элементов a_{ii} на каждом *i*-том шаге. Если диагональный элемент равен нулю, то последовательно рассматривая ниже стоящие строки выбираем такую из них, что элемент находящийся в столбце *nso* будет отличаться от нуля. Затем в цикле с параметром *j* поменяем местами *i*-тую строку и строку с номером *nso*.

```
                                '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
```

Далее выполняем *i*-тый шаг прямого хода. Здесь определяем коэффициент *b*, как частное от деления a_{ij} на a_{ii}, а затем в цикле с параметром k рассчитываем значения *j*-той строки.

```
                                '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
```

На каждом шаге прямого хода выводим полученные коэффициенты СЛАУ на лист, начиная с позиции *s*. Из закрываем цикл по *i*.

```
                               '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
```

Ниже представлен вывод последнего шага прямого хода метода Гаусса. Здесь матрица имеет верхний треугольный вид.

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

Рис. 5.14. Вывод последнего шага прямого хода метода Гаусса.

Для проведения обратного хода метода Гаусса произведем все вышеперечисленные операции в обратном порядке, и, начиная с предпоследней строки до первой, обнулим все значения, находящиеся выше главной диагонали. После выполнения каждой итерации в цикле с параметром *i*, будем выводить полученные значения преобразованной расширенной матрицы.

```
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
```

Ниже представлен вывод последнего шага обратного хода метода Гаусса. Матрица стала диагональной.

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

Рис. 5.16. Вывод последнего шага обратного хода метода Гаусса

Ненулевые элементы, находящиеся вне главной диагонали, означают погрешности компьютерных вычислений. На самом деле, порядок этих значений 10^{-16}, такими неточностями можно пренебречь.

И, наконец, для получения окончательного решения разделим элементы матрицы на диагональные элементы и выведем решение.

```
                '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
```

Результат выполнения этой части программы будет выглядеть так:

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						

Рис. 5.16. Вывод решения СЛАУ методом Гаусса.

Литература

1. Алексеев Е.Р. Free Pascal и Lazarus: Учебник по программированию / Е.Р. Алексеев, О. В. Чеснокова, Т. В. Кучер. – М.: ALT Linux; Издательский дом ДМК-пресс, 2010.
2. Вирт Н. Алгоритмы и структуры данных / Н. Вирт. – М.: Мир, 1989.
3. Бакнелл Дж.. Фундаментальные алгоритмы и структуры данных в Delphi. – СПб.: ДиаСофтЮП, 2003 г.
4. Зенков А.В., Численные методы: учеб. пособие / А. В. Зенков. – Екатеринбург: Изд-во Урал. ун-та, 2016.
5. Зыков, С. В. Введение в теорию программирования / С. В. Зыков. – 2-е изд. – М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016.
6. Кнут Д.Э. Искусство программирования, т.1. "Основные алгоритмы", М.: "Мир", 1976.
7. Кузьменко В.Г. VBA. – М.: ООО «Бином-Пресс», 2012.
8. Михайленко Е.В. Информатика и ЭВМ в психологии: курс лекций. / Е.В. Михайленко, И.Н. Старостенко. – Краснодар: Краснодар. ун-т МВД России, 2009.
9. Михайленко Е.В. Комбинаторный анализ. – Краснодар: Краснодар. ун-т МВД России, 2015.
10. Михайленко Е.В. Математический анализ и дифференциальные уравнения: учеб. пособие / Е.В. Михайленко, М.А. Швецова. – Краснодар: Краснодар. ун-т МВД России, 2010.
11. Михайленко Е.В. Методы оптимизации: сборник задач. / Е.В. Михайленко, А.А. Хромых. – Краснодар: Краснодар. ун-т МВД России, 2015.
12. Михайленко Е.В. Приближенные методы вычисления интегралов. – Краснодар: Краснодар. ун-т МВД России, 2015.
13. Михайленко Е.В. Прикладная математика: курс лекций. / Е.В. Михайленко, А.А. Хромых. – Краснодар: Краснодар. ун-т МВД России, 2014.
14. Михайленко, Е.В. Организация ветвления и циклов: лекция / Е.В. Михайленко. – Краснодар: Краснодарский университет МВД России, 2015.
15. Михайленко, Е.В. Технологии программирования: учеб. пособие / Е.В. Михайленко, А.К. Назаров. // Труды сотрудников КрУ МВД России (Каф. ИиМ). – Краснодар: КрУ МВД России, 2018.
16. Пирумов У.Г., Численные методы: теория и практика: учеб. пособие для бакалавров / У.Г. Пирумов и [др.]. – 5-е изд., перераб. И доп. –М.: Издательство Юрайт, 2012. – 421 с.
17. Старостенко, И.Н. Аппаратное и программное обеспечение компьютерных систем: учеб. пособие / И.Н. Старостенко, Ю.Н. Сопильняк. – Краснодар: Краснодар: Краснодарский университет МВД России, 2011
18. Старостенко, И.Н. Языки программирования [Текст]: лекция / И.Н. Старостенко. – Краснодар: Краснодарский университет МВД России, 2008

ОГЛАВЛЕНИЕ

Учебное издание

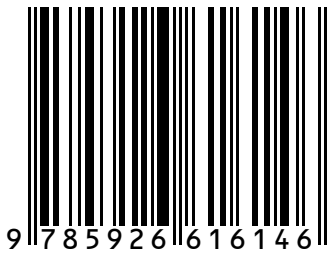
Михайленко Евгений Владимирович

ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ

Учебное пособие

В авторской редакции
Компьютерная верстка *Г. А. Артемовой*

ISBN 978-5-9266-1614-6



Подписано в печать 28.02.2020. Формат 60x84 1/16.
Усл. печ. л. 6,2. Тираж 100 экз. Заказ 16.

Краснодарский университет МВД России.
350005, Краснодар, ул. Ярославская, 128.