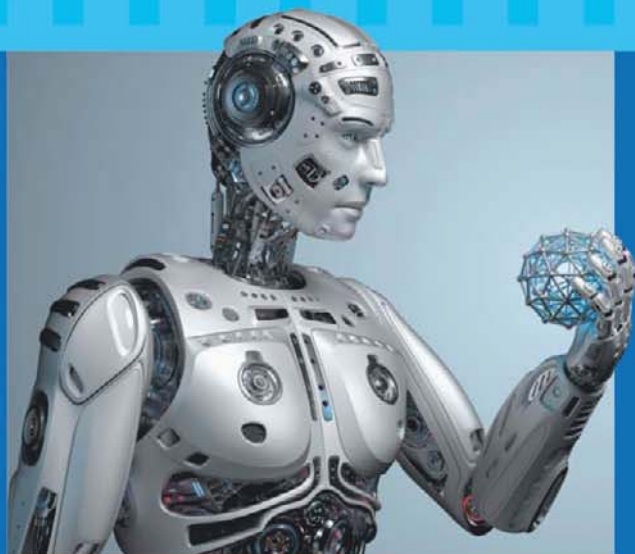




Л. Л. Босова
А. Ю. Босова

ИНФОРМАТИКА

9

к л а с с

БАЗОВЫЙ УРОВЕНЬ

УДК 373.167.1:004+004(075.3)
ББК 32.81я721
Б85

Учебник допущен к использованию при реализации имеющих государственную аккредитацию образовательных программ начального общего, основного общего, среднего общего образования организациями, осуществляющими образовательную деятельность, в соответствии с Приказом Министерства просвещения Российской Федерации № 858 от 21.09.2022 г.

Издание выходит в pdf-формате.

Босова, Людмила Леонидовна.

Б85 Информатика : 9-й класс : базовый уровень : учебник : издание в pdf-формате / Л. Л. Босова, А. Ю. Босова. — 5-е изд., перераб. — Москва : Просвещение, 2023. — 272 с. : ил.

ISBN 978-5-09-110059-4 (электр. изд.). — Текст: электронный.

ISBN 978-5-09-102544-6 (печ. изд.).

Учебник входит в состав УМК по информатике для 7–9 классов, включающего авторскую программу, учебники, электронные приложения, методическое пособие, рабочие тетради, сборники задач и другие компоненты. УМК может использоваться после вводного курса информатики в 5–6 классах или полностью самостоятельно обеспечивать освоение обязательного курса информатики в 7–9 классах, поддерживая базовую (1 ч/нед.) модель изучения предмета на уровне основного общего образования. Содержание учебника структурировано по темам «Разработка алгоритмов и программ», «Управление» (тематический раздел «Алгоритмы и программирование»), «Моделирование как метод познания» (тематический раздел «Теоретические основы информатики»), «Электронные таблицы» (тематический раздел «Информационные технологии»), «Информационно-коммуникационные технологии в современном обществе» (тематический раздел «Цифровая грамотность»). Теоретический материал поддержан развёрнутым аппаратом организации усвоения изучаемого материала, направленным на достижение обучающимися личностных, метапредметных и предметных образовательных результатов, обеспечивающим подготовку школьников к государственной итоговой аттестации по информатике в форме основного государственного экзамена (ОГЭ). Выдержан принцип инвариантности к конкретным моделям компьютеров и версиям программного обеспечения. Соответствует ФГОС ООО и Примерной рабочей программе по информатике для основного общего образования.

УДК 373.167.1:004+004(075.3)
ББК 32.81я721

ISBN 978-5-09-110059-4 (электр. изд.)
ISBN 978-5-09-102544-6 (печ. изд.)

© АО «Издательство «Просвещение»,
2020, 2023
© Художественное оформление.
АО «Издательство «Просвещение»,
2020, 2023
Все права защищены

ОГЛАВЛЕНИЕ

Введение	3
Глава 1. Алгоритмы и программирование	5
§ 1.1. Конструирование алгоритмов	5
§ 1.2. Запись вспомогательных алгоритмов на языке Паскаль	24
§ 1.3. Запись вспомогательных алгоритмов на языке Python	30
§ 1.4. Обработка одномерных массивов целых чисел на языке Паскаль	39
§ 1.5. Обработка одномерных массивов целых чисел на языке Python	55
§ 1.6. Алгоритмы управления	72
Тестовые задания для самоконтроля	79
Глава 2. Моделирование	82
§ 2.1. Моделирование как метод познания	82
§ 2.2. Знаковые модели	92
§ 2.3. Графические информационные модели	105
§ 2.4. Табличные информационные модели	118
§ 2.5. База данных как модель предметной области	131
Тестовые задания для самоконтроля	145
Глава 3. Обработка числовой информации в электронных таблицах	152
§ 3.1. Электронные таблицы	152
§ 3.2. Организация вычислений в электронных таблицах ...	162
§ 3.3. Средства анализа и визуализации данных	178
Задания для практических работ	194
Тестовые задания для самоконтроля	200

Глава 4. Информационно-коммуникационные технологии в современном обществе	206
§ 4.1. Локальные и глобальные компьютерные сети	206
§ 4.2. Информационные ресурсы и сервисы Интернета	219
§ 4.3. Деятельность в сети Интернет	236
§ 4.4. Информационное общество: возможности и проблемы	248
Тестовые задания для самоконтроля	261
Ключи к тестовым заданиям для самоконтроля	266
Ответы к вопросам и заданиям для самостоятельной подготовки	268

Глава 1

АЛГОРИТМЫ И ПРОГРАММИРОВАНИЕ

§ 1.1

Конструирование алгоритмов

Ключевые слова:

- пошаговая детализация
- вспомогательный алгоритм
- рекурсивный алгоритм
- формальные параметры
- фактические параметры

1.1.1. Методы построения алгоритмов

В 8 классе вы познакомились с базовыми алгоритмическими конструкциями (следованием, ветвлением, циклом), научились записывать несложные алгоритмы для формальных исполнителей и на одном из языков программирования.

При конструировании (разработке) алгоритма решения сложной задачи её принято разбивать на более простые подзадачи. При этом, как правило, используется один из следующих методов:

- 1) метод разработки «сверху вниз»;
- 2) метод разработки «снизу вверх».

Метод разработки «сверху вниз» иначе называется методом последовательного уточнения алгоритма или методом **пошаговой детализации**.

Процесс последовательного уточнения алгоритма выглядит следующим образом.

На первом шаге мы считаем, что перед нами совершенный исполнитель, который «всё знает и всё умеет». Поэтому достаточно определить исходные данные и результаты алгоритма, а сам

алгоритм представить в виде единого предписания — постановки задачи (рис. 1.1).

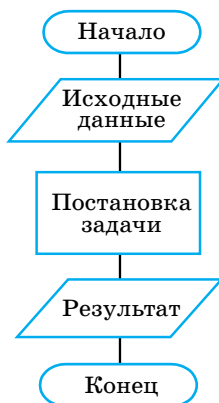


Рис. 1.1. Линейный алгоритм, являющийся результатом первого шага последовательного уточнения алгоритма

Если исполнитель не обучен исполнять заданное предписание, то необходимо представить это предписание в виде совокупности более простых предписаний (команд). Для этого:

- 1) задачу разбивают на несколько частей, каждая из которых проще всей задачи;
- 2) решение каждой части задачи формулируют в отдельной команде, которая также может выходить за рамки системы команд исполнителя;
- 3) при наличии в алгоритме предписаний, выходящих за пределы возможностей исполнителя, такие предписания вновь представляются в виде совокупности ещё более простых предписаний.

Процесс продолжается до тех пор, пока все предписания не будут понятны исполнителю.

Объединяя полученные предписания в единую совокупность выполняемых в определённой последовательности команд, получаем требуемый алгоритм решения исходной задачи.

При использовании метода разработки «снизу вверх» сначала разрабатывается библиотека — множество отдельных небольших алгоритмов, которые могут понадобиться для решения некоторого класса задач. После этого разрабатывается основной алгоритм, содержащий обращения к алгоритмам из библиотеки.

1.1.2. Разработка алгоритма методом пошаговой детализации для исполнителя Робот

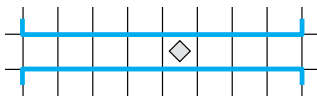
Вы уже знакомы с исполнителем Робот. Он действует на клетчатом поле, между клетками которого могут быть стены.

Система команд исполнителя Робот:

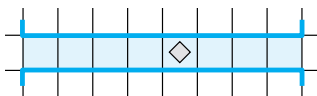
Команда	Описание команды
Вверх	Робот перемещается в соседнюю клетку в указанном направлении. Если в этом направлении между клетками стоит стена, то Робот разрушается
Вниз	
Вправо	
Влево	
Закрасить	Робот закрашивает ту клетку, в которой находится
Сверху свободно	Проверка истинности условия отсутствия стены у соответствующей стороны той клетки, где находится Робот: стены нет — «истина», иначе «ложь»
Снизу свободно	
Слева свободно	
Справа свободно	
Сверху стена	Проверка истинности условия наличия стены у соответствующей стороны той клетки, где находится Робот: стена есть — «истина», иначе «ложь»
Снизу стена	
Слева стена	
Справа стена	
Клетка закрашена	Проверка истинности условия: клетка закрашена — «истина», иначе «ложь»
если <условие> то <команда 1> иначе <команда 2> все	Организация ветвления: если <условие> истинно, то выполняется <команда 1>; если <условие> ложно, то выполняется <команда 2>
нц пока <условие> <последовательность команд> кц	Организация цикла: пока <условие> истинно, выполняется <последовательность команд>

В одном условии можно использовать несколько команд, применяя логические операции **и**, **или**, **не**.

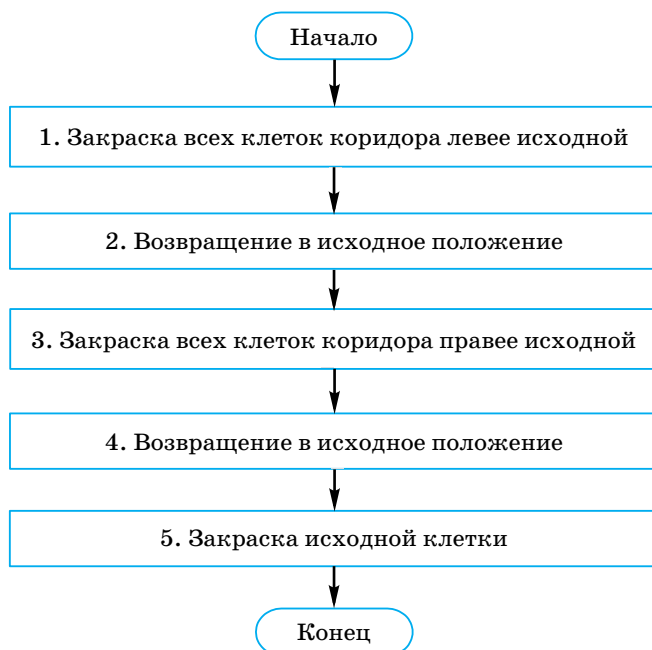
Известно, что Робот находится где-то в горизонтальном коридоре. Ни одна из клеток коридора не закрашена.



Составим алгоритм, под управлением которого Робот закрасит все клетки этого коридора и вернётся в исходное положение.



Представим план действий Робота следующими укрупнёнными шагами (модулями):



Детализируем каждый из пяти модулей.

1. Чтобы закрасить все клетки коридора, находящиеся левее Робота, прикажем Роботу шагнуть влево и выполнить цикл-ПОКА:

влево

нц пока сверху стена **и** снизу стена
закрасить; влево

кц

Под управлением этого алгоритма Робот закрасит все клетки коридора, находящиеся левее от него, и окажется в клетке левее коридора:



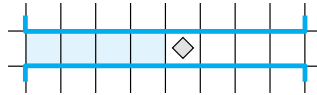
- Командой **вправо** вернём Робота в коридор. Вернём Робота в исходную клетку. Эта клетка — первая незакрашенная клетка, находящаяся правее Робота. Поэтому пока занимаемая Роботом клетка оказывается закрашенной, будем перемещать его **вправо**.

```

вправо
нц пока клетка закрашена
    вправо
кц

```

Под управлением этого алгоритма Робот окажется в исходной клетке:

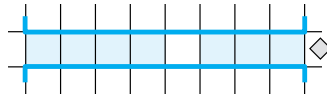


- Выполнив команду **вправо**, Робот пройдёт исходную клетку и займёт клетку правее исходной. Теперь можно закрашивать клетки коридора, расположенные правее исходной.

```

вправо
нц пока сверху стена и снизу стена
    закрасить; вправо
кц

```

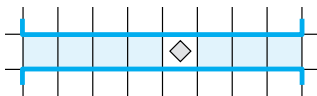


- Так как, выполнив предыдущий алгоритм, Робот оказался правее коридора, командой **влево** вернём его в коридор. Возвращение в исходную клетку обеспечивается алгоритмом:

```

влево
нц пока клетка закрашена
    влево
кц

```



5. По команде закрасить Робот закрашивает исходную клетку.

Полностью программа управления Роботом выглядит так:

использовать Робот

алг

нач

влево

нц пока сверху стена **и** снизу стена
закрасить; влево

кц

вправо

нц пока клетка закрашена
вправо

кц

вправо

нц пока сверху стена **и** снизу стена
закрасить; вправо

кц

влево

нц пока клетка закрашена
влево

кц

закрасить

кон



Выполните приведённый выше алгоритм в среде *КуМир*, предварительно создав в ней стартовую обстановку — горизонтальный коридор.

Разработайте алгоритм решения исходной задачи при условии, что в системе команд Робота отсутствуют следующие команды:

сверху стена

снизу стена

слева стена

справа стена

Модифицируйте алгоритм таким образом, чтобы с его помощью можно было бы решить аналогичную задачу в вертикальном коридоре. Убедитесь в работоспособности разработанного алгоритма.

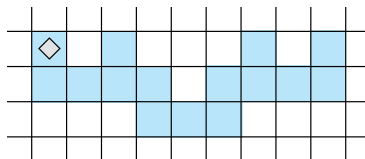
1.1.3. Вспомогательные алгоритмы

Во многих случаях при разработке алгоритмов решения сложных задач используют вспомогательные алгоритмы. Например, это удобно, когда в разных местах алгоритма необходимо выполнение одной и той же последовательности действий. Такую последовательность действий оформляют отдельным алгоритмом, который вызывают в нужных местах основного алгоритма. Один и тот же вспомогательный алгоритм может использоваться в разных программах. Вспомогательные алгоритмы делают более понятной структуру основного алгоритма.

Вспомогательный алгоритм — алгоритм, целиком используемый в составе другого алгоритма для решения некоторой подзадачи основной задачи.

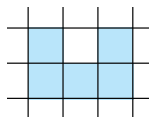
Пример 1

В среде *КуМир* составим алгоритм для исполнителя Робот, под управлением которого будет нарисован узор:



Начальное положение Робота отмечено ромбом; конечное положение значения не имеет.

Можно написать одну длинную программу, выполнив которую Робот закрасит все клетки, образующие этот узор. Но можно поступить иначе, обратив внимание на то, что в узоре трижды повторяется изображение одной и той же фигуры:



Разработав вспомогательный алгоритм для изображения этой фигуры, его можно вызвать нужное количество раз для построения требуемого узора.

```

использовать Робот
алг узор
нач
    фигура
    вправо; вниз
    фигура
    вправо; вверх
    фигура
кон
алг фигура
нач
    закрасить; вниз
    закрасить; вправо; закрасить; вправо
    закрасить; вверх; закрасить
кон

```



Выполните приведённый выше алгоритм в среде *КуМир*. Модифицируйте алгоритм таким образом, чтобы с его помощью можно было бы продолжить узор, дополнив его по горизонтали несколькими повторяющимися фигурами. Убедитесь в работоспособности разработанного алгоритма.

Каждый вспомогательный алгоритм получает уникальное имя, после которого указываются параметры алгоритма (при наличии) — имена переменных, обозначающих исходные данные (аргументы) и результаты вспомогательного алгоритма.

При представлении алгоритмов с помощью блок-схем для обозначения команды вызова вспомогательного алгоритма используется блок «предопределённый процесс» (рис. 1.2), внутри которого записывается название (имя) вспомогательного алгоритма, после которого в скобках перечисляются параметры — аргументы и результаты.

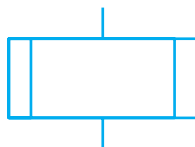


Рис. 1.2. Блок «предопределённый процесс»

Имя рассмотренного выше вспомогательного алгоритма — фигура; это вспомогательный алгоритм без параметров.

Пример 2

Исполнитель Черепаха перемещается на экране компьютера в квадратном окне со стороной 500 пикселей и может рисовать кончиком хвоста, оставляя след в виде линии. Исходное положение: Черепаха находится в центре окна, хвост опущен, направление — строго вверх.

Система команд исполнителя Черепаха:

Команда	Описание команды
поднять хвост	Последующие перемещения не оставляют следа в окне на экране
опустить хвост	Последующие перемещения оставляют след в окне на экране
вперед (n) назад (n)	Перемещение Черепахи на n шагов в направлении движения (в противоположном направлении); n — вещественное число
вправо (m) влево (m)	Изменение направления движения Черепахи на m градусов по часовой стрелке (против часовой стрелки); m — вещественное число
нц k раз <последовательность команд> кц	Черепаха повторяет k раз последовательность команд в скобках

Выясним, что получится в результате исполнения Черепахой следующего алгоритма:

```

нач
  нц 8 раз
    вперед (50)
    нц 3 раз
      вперед (50)
      вправо (120)
    кц
    назад (50)
    вправо (45)
  кц
кон
  
```

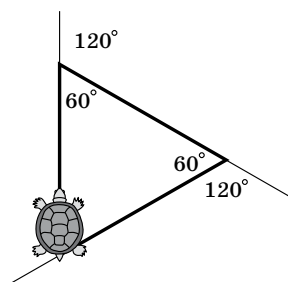
В этой программе два цикла, один из которых находится внутри другого. Вспомните, как называется такая конструкция.



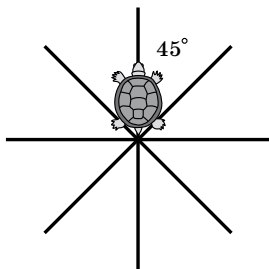


Мысленно разобьём задачу на две подзадачи, решаемые внутренним и внешним циклами соответственно.

Рассмотрим внутренний цикл. Результат его выполнения — равносторонний треугольник, нарисовав который Черепаха возвращается в исходное положение.



Результат работы внешнего цикла без учёта внутреннего цикла имеет вид:



Совместите мысленно результаты выполнения внешнего и внутреннего циклов. Опишите получившийся объект и дайте ему название.

Проверьте своё предположение, выполнив рассмотренный алгоритм в среде *КуМир*.

Оформите внутренний цикл в виде вспомогательного алгоритма без параметров **треугольник**. Убедитесь, что всё работает верно.

Можно модифицировать вспомогательный алгоритм **треугольник** во вспомогательный алгоритм с параметрами **многоугольник(k, n)**, где k — количество углов, а n — длина стороны правильного многоугольника:

```

алг многоугольник (арг цел  $k, n$ )
нач
    нц  $k$  раз
        вперед ( $n$ )
        вправо ( $360/k$ )
    кц
кон
  
```

Здесь k, n — **формальные параметры**, они используются при описании вспомогательного алгоритма. При конкретном обращении к вспомогательному алгоритму (вызове вспомогательного алгоритма) формальные параметры заменяются **фактическими параметрами**, т. е. именно теми величинами, для которых будет исполнен вспомогательный алгоритм. Типы, количество и порядок следования формальных и фактических параметров должны совпадать.

Основной алгоритм, вызывающий вспомогательный алгоритм многоугольник(k, n), может выглядеть так:

```
алг
нач
  нц 8 раз
    вперед (50)
    многоугольник (5, 15)
    назад (50)
    вправо (45)
  кц
кон
```

Пример 3

Как известно, натуральные числа, большие единицы, в зависимости от количества их делителей подразделяются на простые и составные:

- простое число — это натуральное число, большее единицы, которое делится только на единицу и само на себя;
- составное число — это натуральное число, большее единицы, которое делится не только на единицу и само на себя, но и ещё хотя бы на одно натуральное число.

На уроках математики вы неоднократно выполняли разложение натурального числа a на простые множители, скорее всего действуя так: находили p — первый простой делитель числа a (самое маленькое простое число, на которое данное число a делится без остатка) и делили на него число a ; находили первый простой делитель для полученного частного и делили частное на этот простой делитель и т. д., пока частное не становилось равным 1.

Нахождение первого простого делителя для некоторого числа — самостоятельная задача, которая может быть решена разными способами. Например, её можно решить подбором, воспользовавшись таблицей простых чисел и применив знание



таблицы умножения. В процессе разложения натурального числа на простые множители вы решали эту задачу неоднократно. Поэтому есть смысл представить её решение в виде самостоятельного алгоритма (рис. 1.3).

Здесь n — натуральное число, большее 1. Его первый простой делитель d находится перебором всех возможных делителей, начиная с 2 — наименьшего простого числа. Если само n окажется простым числом, то d будет присвоено значение n .

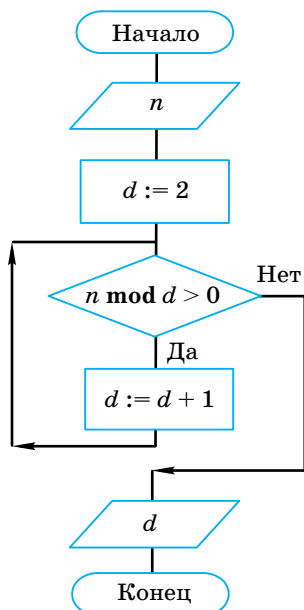


Рис. 1.3. Алгоритм нахождения первого простого делителя для некоторого числа

Попытайтесь самостоятельно записать алгоритм нахождения первого простого делителя, представленный на рис. 1.3, на известном вам языке программирования. Протестируйте свою программу на числах 121, 135 и 847. Если программа написана правильно, то результатами её выполнения будут числа 11, 5 и 7 соответственно.

Алгоритм нахождения первого простого делителя (см. рис. 1.3) можно рассматривать как вспомогательный при решении задачи разложения натурального числа на простые множители (рис. 1.4).

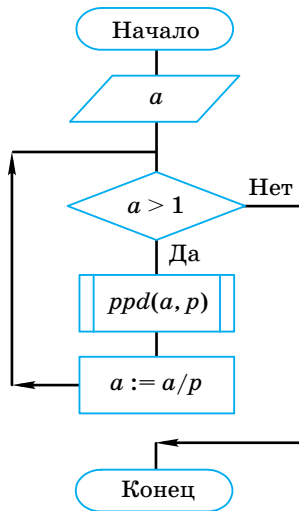


Рис. 1.4. Алгоритм разложения натурального числа на простые множители

Вспомогательный алгоритм нахождения первого простого делителя для некоторого числа мы назвали ppd . Его формальными параметрами являются переменные n, d ; фактическими — a, p .

В общем случае команда вызова вспомогательного алгоритма выполняется следующим образом (рис. 1.5):

- 1) формальные входные данные вспомогательного алгоритма заменяются значениями фактических входных данных, указанных в команде вызова вспомогательного алгоритма;
- 2) для заданных входных данных исполняются команды вспомогательного алгоритма;
- 3) полученные результаты присваиваются переменным с именами фактических результатов;
- 4) осуществляется переход к следующей команде основного алгоритма.

Имена и типы переменных, которые заявлены в качестве формальных параметров, объявляются внутри вспомогательного алгоритма по тем же правилам, что и для основного алгоритма. Такие переменные называются **локальными**. При необходимости работать с переменными и в основном алгоритме, и во вспомогательном их объявляют как **глобальные**.

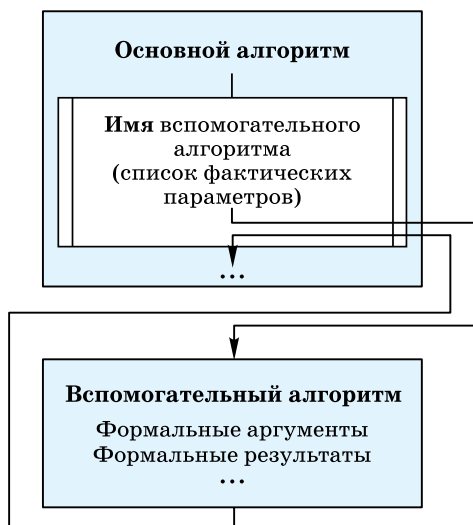


Рис. 1.5. Схема выполнения команды вызова вспомогательного алгоритма



Алгоритм, в котором прямо или косвенно содержится ссылка на него же как на вспомогательный алгоритм, называют **рекурсивным**.

Рассмотрим несколько примеров рекурсивных алгоритмов.

Пример 4

Алгоритм вычисления степени с натуральным показателем n для любого вещественного числа a можно представить в виде рекурсивного (рис. 1.6).

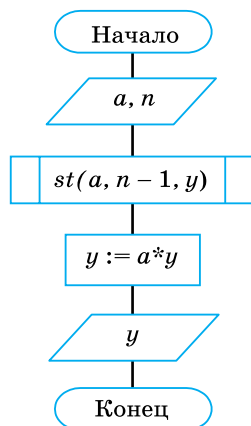


Рис. 1.6. Рекурсивный алгоритм возведения в степень

Число a в степени n (n -я степень числа a) есть не что иное, как произведение $a^{n-1} \cdot a$; в свою очередь, $a^{n-1} = a^{n-2} \cdot a$ и т. д.

Пример 5

Рекурсивный алгоритм положен в основу эффективного решения головоломки «Ханойская башня» (рис. 1.7).

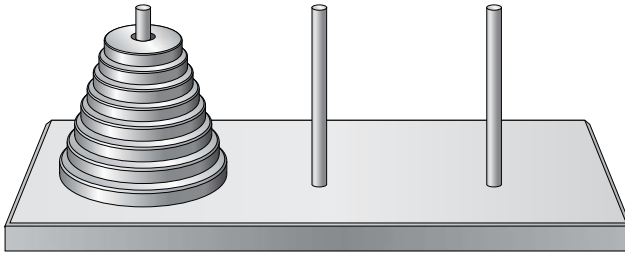


Рис. 1.7. Головоломка «Ханойская башня»

В сети Интернет размещено много разных онлайн-вариантов игры «Ханойская башня». Подберите какой-нибудь из них по своему вкусу и попытайтесь решить головоломку для восьми колец за минимально возможное количество ходов. При возникновении трудностей загляните в главу 3: подсказка содержится в задании 6 для практических работ.

www

Пример 6

Рассмотрим алгоритм построения геометрической фигуры, которая называется снежинкой Коха. Шаг процедуры построения состоит в замене средней трети каждого из имеющихся отрезков двумя новыми такой же длины, как показано на рис. 1.8.

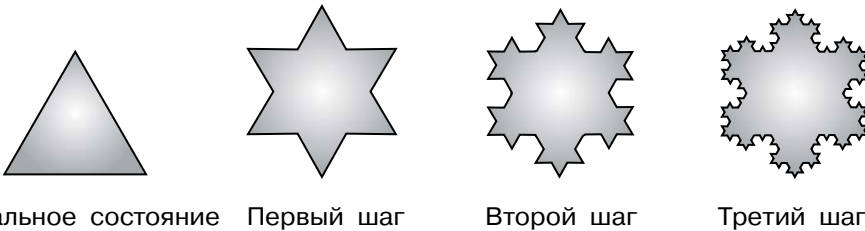


Рис. 1.8. Шаги алгоритма построения снежинки Коха

С каждым шагом фигура становится всё причудливее. Граница снежинки Коха — положение кривой после выполнения бесконечного числа шагов.



Попробуйте подсчитать, сколько рёбер в границе снежинки Коха после четвёртого шага; после пятого шага.

САМОЕ ГЛАВНОЕ

Один из основных методов конструирования алгоритмов решения сложных задач — метод пошаговой детализации. Его суть состоит в том, что исходная задача разбивается на несколько частей, каждая из которых проще всей задачи, и решение каждой части формулируется в отдельном предписании; если получаются предписания, выходящие за пределы возможностей исполнителя, то они представляются в виде совокупности ещё более простых предписаний. Процесс продолжается до тех пор, пока все предписания не будут понятны исполнителю.

Вспомогательный алгоритм — алгоритм, целиком используемый в составе другого алгоритма для решения некоторой подзадачи основной задачи.

Алгоритм, в котором прямо или косвенно содержится ссылка на него же как на вспомогательный алгоритм, называют рекурсивным.



Вопросы и задания

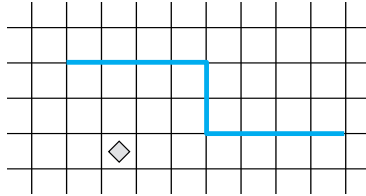


1. Почему при решении сложной задачи затруднительно сразу конкретизировать все необходимые действия? Обсудите этот вопрос в группе.
2. В чём заключается метод пошаговой детализации при построении алгоритма?
3. Какая связь между методом разработки алгоритма «сверху вниз» и такими процессами, как написание сочинения или подготовка к многодневному туристическому походу?
4. Известен рост каждого из n учеников 9 «А» класса и m учеников 9 «Б» класса. Опишите укрупнёнными блоками алгоритм сравнения среднего роста учеников этих классов.

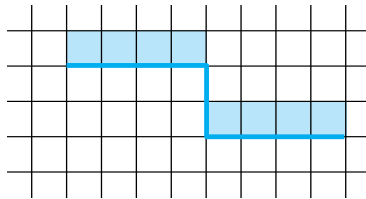


5. На бесконечном поле имеется стена, состоящая из двух горизонтальных и одного вертикального участков, примерная форма которой представлена на рисунке. Длины участков неизвестны. Робот находится в клетке под левой горизонтальной частью стены.

На рисунке указан один из возможных вариантов расположения стены и Робота.



Задача Робота — закрасить клетки, примыкающие сверху к горизонтальным участкам стены. Например, для приведённого выше рисунка Робот должен закрасить следующие клетки:



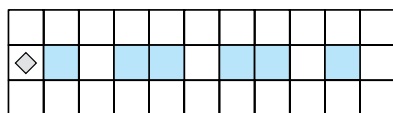
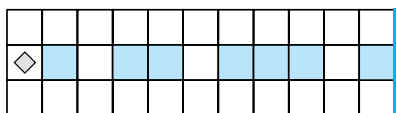
Конечное расположение Робота может быть произвольным. Задача должна быть решена для стены, горизонтальные и вертикальный участки которой могут иметь любые размеры.

Есть план действий Робота:

- 1) двигаться вверх до горизонтального участка стены;
- 2) двигаться влево под горизонтальным участком стены и сместиться на 1 клетку левее стены;
- 3) занять положение над горизонтальным участком стены;
- 4) закрасить клетки слева направо над горизонтальным участком стены;
- 5) спуститься вниз вдоль вертикального участка стены;
- 6) закрасить клетки слева направо над горизонтальным участком стены.

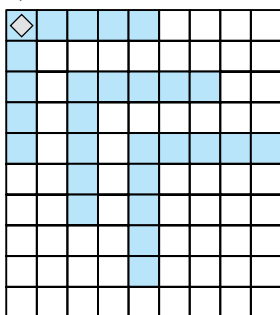
Реализуйте этот план, составив алгоритм для Робота в среде *КуМир*.

6. В ряду из десяти клеток правее Робота некоторые клетки закрашены. Последняя закрашенная клетка может примыкать к стене. Составьте алгоритм, который закрашивает клетки, примыкающие сверху и снизу к каждой закрашенной клетке. Проверьте работу алгоритма в следующих случаях:

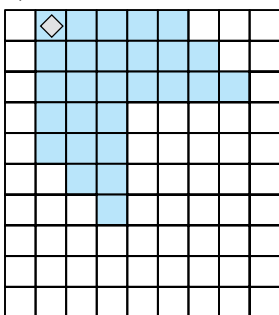


7. Для чего нужны вспомогательные алгоритмы? Опишите процесс выполнения команды вызова вспомогательного алгоритма в основном алгоритме.
8. Составьте алгоритмы, под управлением которых Робот закрасит указанные клетки. Используйте вспомогательный алгоритм.

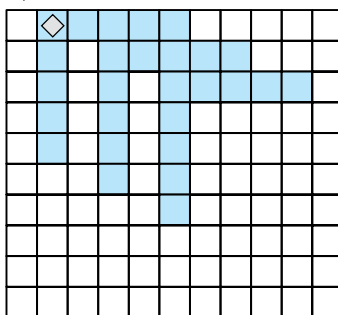
а)



б)

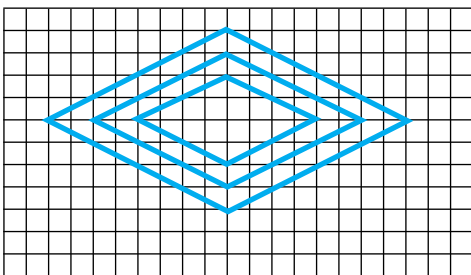


в)

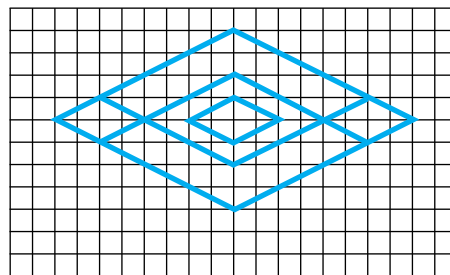


9. Сталкивались ли вы с идеей формальных и фактических параметров при изучении математики и физики? Приведите пример.
10. Для исполнителя Чертёжник в среде *КуМир* составьте алгоритмы рисования следующих фигур.

а)



б)



Воспользуйтесь вспомогательным алгоритмом с параметром ромб:

```
алг ромб (арг вещ  $x$ ,  $y$ ,  $a$ )  
нач  
    сместиться в точку ( $x$ ,  $y$ )  
    опустить перо  
    сместиться на вектор( $2*a$ ,  $a$ )  
    сместиться на вектор( $-2*a$ ,  $a$ )  
    сместиться на вектор( $-2*a$ ,  $-a$ )  
    сместиться на вектор( $2*a$ ,  $-a$ )  
    поднять перо  
кон
```

Придумайте собственные рисунки на основе вспомогательного алгоритма с параметром ромб.

11. Какие алгоритмы называют рекурсивными? Приведите пример рекурсии из жизни.
12. В среде *КуМир* исследуйте алгоритм для Черепахи, содержащий рекурсию:

```
использовать Черепаха  
алг чертеж (цел  $a$ )  
нач  
    если  $a < -150$   
        то стоп  
    все  
    нц 4 раз  
        вперед ( $a$ ); вправо ( $90$ )  
    кц  
    чертеж ( $a-5$ )  
кон
```

Проследите, что получится при изменении значений параметра a . Добавьте параметр k — количество углов основной фигуры. Предложите свои варианты рисунков для Черепахи, которые можно получить с помощью рекурсии.



§ 1.2

Запись вспомогательных алгоритмов на языке Паскаль

Ключевые слова:

- подпрограмма
- процедура
- функция
- рекурсивная функция

Запись вспомогательных алгоритмов в языках программирования осуществляется с помощью **подпрограмм**. В Паскале подпрограмма является частью основной программы. Её описание располагается между разделом **var** и программным блоком основной программы. Если подпрограмм несколько, то их описания располагаются в произвольном порядке одно за другим.

Структура описания подпрограммы аналогична структуре основной программы. Описание подпрограммы начинается с заголовка и заканчивается оператором **end**.

В Паскале различают два вида подпрограмм: процедуры и функции.

1.2.1. Процедуры



Процедура — подпрограмма, имеющая произвольное количество входных и выходных данных.

Описание процедуры имеет вид

```
procedure <имя_процедуры> (<описание параметров-значений>;  
                           var <описание параметров-переменных>);  
begin  
  <операторы>  
end;
```

В заголовке процедуры после её имени приводится перечень формальных параметров и их типы. Входные параметры, значения которых не изменяются в программе, должны быть параметрами-значениями. Выходные (результатирующие) параметры должны быть параметрами-переменными.

Для вызова процедуры достаточно указать её имя со списком фактических параметров. В качестве параметров-значений можно указывать имена переменных, константы и выражения.

Например, заголовок процедуры вычисления наибольшего общего делителя может быть описан так:

```
procedure nod (a, b: integer; var c: integer);
```

Возможны следующие варианты вызова этой процедуры:

nod (36, 15, z) — в качестве параметров-значений использованы константы;
 nod (x, y, z) — в качестве параметров-значений использованы имена переменных;
 nod (x + y, 15, z) — в качестве параметров-значений использованы выражение и константа.

В любом случае между фактическими и формальными параметрами должно быть полное соответствие по количеству, порядку следования и типу.

Пример 1

Хорошо известный вам алгоритм Евклида для нахождения наибольшего общего делителя (НОД) двух чисел в виде процедуры можно описать так:

```
procedure nod(a, b: integer; var c: integer);
begin
    while a <> b do
        if a > b then a := a - b else b := b - a;
        c := a
    end;
```

Задача поиска НОД для последовательности натуральных чисел, количество членов которой k вводится с клавиатуры, может быть решена с помощью следующей программы:

<pre>var i, k, x, y, z: integer;</pre>	Переменные основной программы
<pre>----- procedure nod(a, b: integer; var c: integer); begin while a <> b do if a > b then a := a - b else b := b - a; c := a end;</pre>	Подпрограмма
<pre>----- begin read(k); read(x); for i := 2 to k do begin read(y); nod(x, y, z); x := z end; writeln('НОД=', x) end.</pre>	Основная программа