Оглавление

Введение

Вы держите перед собой учебное пособие, которое поможет вам подготовиться к экзамену по информатике, научиться программировать, решать задачи на компьютере. В качестве языка программирования мы выбрали язык Паскаль. Этот язык создан специально для учебных целей, удобен в качестве первого изучаемого языка программирования, так как, с одной стороны, он не очень сложен, а с другой — на нем можно решать достаточно серьезные задачи.

Никаких специальных знаний для того, чтобы начать заниматься программированием, не требуется. Чтобы усвоить материал нашего пособия, достаточно уметь читать и немного — считать, поэтому можно начать занятия и в 11-м классе, и в 10-м, и даже раньше.

Главное место в учебном пособии занимают задачи с решениями. Даже необходимые теоретические сведения и определения подаются с помощью задач. Решая задачи, мы объясняем, зачем нужны новые конструкции языка, как их использовать, показываем способы разработки алгоритмов, приемы программирования. В тексте разобрано порядка полутора сотен задач, причем почти все решения снабжены подробными пояснениями и доведены до полной программы (ее можно «набить» на компьютере и выполнить — будет работать!).

Как «набить» программу? Для этого вам понадобится компьютер, умеющий работать с одной из версий языка Паскаль. В пособии подробно рассматривается, как работать в среде Borland Pascal 7.0 (Борланд Паскаль, Турбо Паскаль), — все приведенные программы писались и отлаживались именно в ней. Также вы найдете пояснения по работе со средами Free Паскаль и ABC-Паскаль. Приведенные в пособии программы, за некоторым исключением, будут без изменений правильно работать в любой среде, так как писались на стандартном Паскале. Все случаи, где применены какие-то особенности Турбо Паскаля, в тексте отмечены, и вы сможете работать с такой программой в своей версии языка (например, ABC-Паскаль), может быть, слегка изменив ее.

Конечно, подбирая задачи, мы учитывали, что большинству из вас наше пособие понадобится для подготовки к ЕГЭ. Однако здесь вы не найдете решений вариантов ЕГЭ разных лет (хотя, конечно же, все типовые задачи рассматриваются). Наша цель — не показать вам, как решаются задачи,

которые когда-то были на экзаменах, а помочь научиться думать, составлять новые алгоритмы, писать программы и выполнять их на компьютере.

Немного о содержании учебника. Глава 1 знакомит с основными, базовыми понятиями программирования. Уже в ней мы разбираем достаточно сложные задачи, правда, пока на уровне блок-схем. В главах 2–12 мы изучаем основные конструкции языка Паскаль, учимся решать задачи. В принципе, изучив этот материал, можно решить любую задачу по программированию из предлагавшихся на ЕГЭ. В главах 13–14 содержится дополнительный материал, он поможет решать «обычные» задачи лучше и быстрее, а также пригодится тем, кто собирается участвовать (и побеждать!) в олимпиадах. А глава 15, можно сказать, развлекательная. Материал, который в ней рассматривается, обычно не входит ни в ЕГЭ, ни в школьную программу. Прочитав эту главу, вы сможете научить свой компьютер рисовать, сочинять музыку, сможете сами писать простенькие компьютерные игры, а заодно повторите все, чему научились ранее. В главе 16 приводятся примеры достаточно сложных задач, для решения которых понадобятся знания из разных глав учебника.

В учебнике много задач (около 600). Часть из них вы найдете после объяснения очередной темы. Многие из таких задач очень похожи на примеры, разобранные в тексте главы. Так что, если непонятно, как решать задачу, попробуйте перечитать материал еще раз, поищите аналогичную задачу. Конечно, есть и более сложные задачи, над которыми придется подумать. Они помечены звездочкой. В конце каждой главы вы найдете задачи по всему материалу, рассмотренному в главе; в этом случае не всегда удастся найти в главе аналог задачи, надо самостоятельно продумать метод решения.

Правильность решения практически всех задач очень легко проверить — в этом поможет компьютер. Введите текст программы, входные данные, запустите ее и сравните полученный ответ с правильным. Откуда взять правильный ответ? Подсчитать вручную! Не беспокойтесь, это просто, никаких сложных вычислений нигде делать не надо. Только не ленитесь проверять программы для разных наборов входных данных. К сожалению, программа может быть правильной, но неэффективной (например, слишком громоздкой), — здесь уже компьютер не поможет. Мы постараемся научить вас писать хорошие программы: некоторые задачи имеют несколько вариантов решений, и мы подробно разбираем, какой из них лучше.

Автору в работе над этой книгой большую помощь оказали преподаватели подготовительных курсов факультета Вычислительной математики и кибернетики (ВМК) МГУ имени М. В. Ломоносова Родин В. И., Вовк Е. Т., Линев Н. Б., Дорофеева И. Д., Лапонина О. Р.

Глава 1
Основные понятия и определения

Программирование

Решение многих задач, возникающих в самых различных сферах человеческой деятельности, было бы невозможно без применения компьютеров. Причем компьютеры — это не только вычислительные устройства с дисплеем, которые стоят в классе информатики или дома на столе. Они окружают нас повсюду: «притаились» в плеере, мобильном телефоне, фотоаппарате, в турникетах в метро и в школе, и даже в домашних бытовых приборах.

Для каких бы целей ни было предназначено устройство: полет ракеты на Марс, расчет зарплаты, выдача денег в банкомате, пропуск учеников в школу — надо «научить» его решать поставленную задачу, т. е. написать инструкции, детально разъясняющие, как действовать во всех возможных ситуациях. Деятельность по написанию таких инструкций (команд) для вычислительных машин или управляющих чем-либо устройств называется программированием, а сами инструкции — программами.

Этапы решения задачи

Предположим, что перед нами стоит задача, для решения которой необходимо написать программу. Решение задачи разбивается на этапы.

1. **Постановка задачи**. Необходимо определить, каковы будут исходные данные (что подается «на вход») и каких результатов надо достичь (что получить «на выходе»). При решении учебных задач этот этап иногда может отсутствовать, так как исходные данные и конечные результаты определены в формулировке задания.
2. **Выбор метода решения и разработка алгоритма**. Это один из основных, главных этапов решения задачи. От правильности выбора метода и эффективности алгоритма зависят размер программы и ее быстродействие.
3. **Составление программы и ввод ее в память компьютера**. Часто именно этот этап неправильно называют программированием. На самом деле, составление программы — лишь некоторая часть решения задачи с помощью компьютера. Сейчас этот процесс принято называть **кодированием**.

4. **Отладка программы**. Созданная программа может содержать ошибки, допущенные как в процессе кодирования, так и на любом из предыдущих этапов (может быть, неправильно разработан алгоритм, неверно выбраны исходные данные и т. д.). Ошибки выявляются в процессе отладки и тестирования — выполнения программы с заранее подготовленными наборами исходных данных, для которых известен результат.
5. **Вычисление и обработка результатов**.

Остановимся подробнее на главном этапе — разработке алгоритма.

Что такое алгоритм?

Алгоритм — это система правил, набор инструкций, позволяющий решить некоторую задачу, детально разработанное описание методов ее решения.

С алгоритмами мы сталкиваемся в математике (алгоритм сложения многозначных чисел в столбик), физике, химии, при изучении языков (правила правописания); они подстерегают нас и в повседневной жизни: алгоритм перехода улицы, правила пользования лифтом и др. Поваренная книга также является сборником алгоритмов для приготовления блюд.

Задание. *Приведите еще примеры алгоритмов из математики, физики, химии, повседневной жизни.*

Любой алгоритм рассчитан на то, что его инструкции будет кто-то выполнять. Назовем этого кого-то «исполнителем» и будем иметь в виду, что наши инструкции должны быть понятны ему, он должен уметь выполнять действия, записанные в алгоритме.

Существуют разные способы представления алгоритмов: с помощью формул, схем, рисунков, также можно записать алгоритм в виде программы и, конечно же, просто словами на обычном (естественном) языке.

Словесная формулировка алгоритма

Часто алгоритмы записываются в виде списка инструкций. Сформулируем таким образом алгоритм пользования лифтом.

1. Нажмите кнопку вызова.
2. Когда автоматические двери откроются, войдите в лифт.
3. Встаньте так, чтобы не мешать закрытию дверей.
4. Нажмите кнопку нужного этажа.
5. Когда лифт приедет на нужный этаж, дождитесь открывания дверей.
6. Выйдите из лифта.

В получившемся алгоритме все инструкции выполняются ровно один раз, последовательно друг за другом. Такие алгоритмы называются **линейными**.

Все ли учтено в нашем списке инструкций? Вдруг лифт сломался и никогда не придет к тому, кто его вызвал, — не указано, что делать в этом случае. А если лифт остановился между этажами? Все знают, что в этом случае нужно вызывать диспетчера. Включим эти инструкции в алгоритм.

Пункт 1 теперь будет выглядеть так.

1. Нажмите кнопку вызова. Если никакой реакции нет, значит, лифт сломан и вам придется идти пешком, если же все в порядке — выполняйте пункт 2.

А пункт 5 так:

5. Дождитесь остановки лифта. Если двери открылись, то выйдите из лифта, иначе нажмите кнопку вызова диспетчера и выполняйте его указания.

Отметим, что здесь мы использовали прием разработки алгоритмов, называемый «сверху вниз». При этом способе задача сначала разбивается на несколько подзадач, а потом некоторые из них, в свою очередь, могут быть разбиты на отдельные части.

Получившийся алгоритм уже не является линейным, в нем есть **разветвления**. В случае поломки лифта пункты 2–6 выполняться не будут, пункт 5 на самом деле состоит из двух частей, и в каждом случае выполняется одна из них.

Блок-схема. Основные конструкции

Для алгоритма с разветвлениями словесная запись в виде перечня инструкций не всегда удобна, в этом случае часто применяется графическая запись алгоритма в виде блок-схемы: каждая инструкция помещается в блок (на чертеже — геометрическая фигура), блоки соединяются стрелочками, определяющими порядок выполнения инструкций. Этот способ представления алгоритмов достаточно популярен и общепринят, существует даже Государственный стандарт на изображение блок-схем. Мы не будем детально его изучать и слишком буквально ему следовать, используем некоторые конструкции, с помощью которых удобно графически представлять изучаемые нами алгоритмы.

Каждый алгоритм должен иметь начало (точка входа), для его обозначения будем использовать значок овал, а для обозначения конца (выхода) — значок овал с крестиком внутри. Каждый алгоритм имеет ровно один вход, а вот выходов может быть несколько (но никак не менее одного!), так как процесс решения задачи должен когда-то заканчиваться с некоторым результатом, причем в зависимости от условий результаты могут получаться разные.

Рассмотрим наиболее часто используемые типы блоков (рис. 1.1). Сразу заметим, что в любой из этих блоков должен быть хотя бы один вход.

Прямоугольник. В нем приводится описание некоторых действий, которые необходимо выполнить. Из прямоугольника всегда выходит только одна стрелочка (т. е. следующая инструкция всегда четко определена).

Параллелограмм. В нем записываются данные, которые необходимо ввести или вывести (исходные или выходные данные). Из параллелограмма выходит тоже только одна стрелочка.

Ромб. Используется для обозначения ветвления. В ромбе размещается вопрос (условие), на который возможен ответ «да» или «нет» (забегая вперед, скажем, что такая конструкция называется «логическое выражение»). Из ромба должны выходить две стрелочки. Одна помечается словом «да» (или «верно», «истинно»), другая — словом «нет» (или «неверно», «ложно»). В зависимости от значения истинности помещенного в ромбик условия далее выполняется переход по одной из стрелочек.

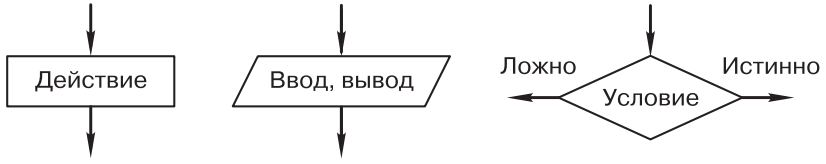

Рис. 1.1

Есть и некоторые другие блоки, о которых мы расскажем позже.

Изобразим алгоритм пользования лифтом в виде блок-схемы (рис. 1.2).

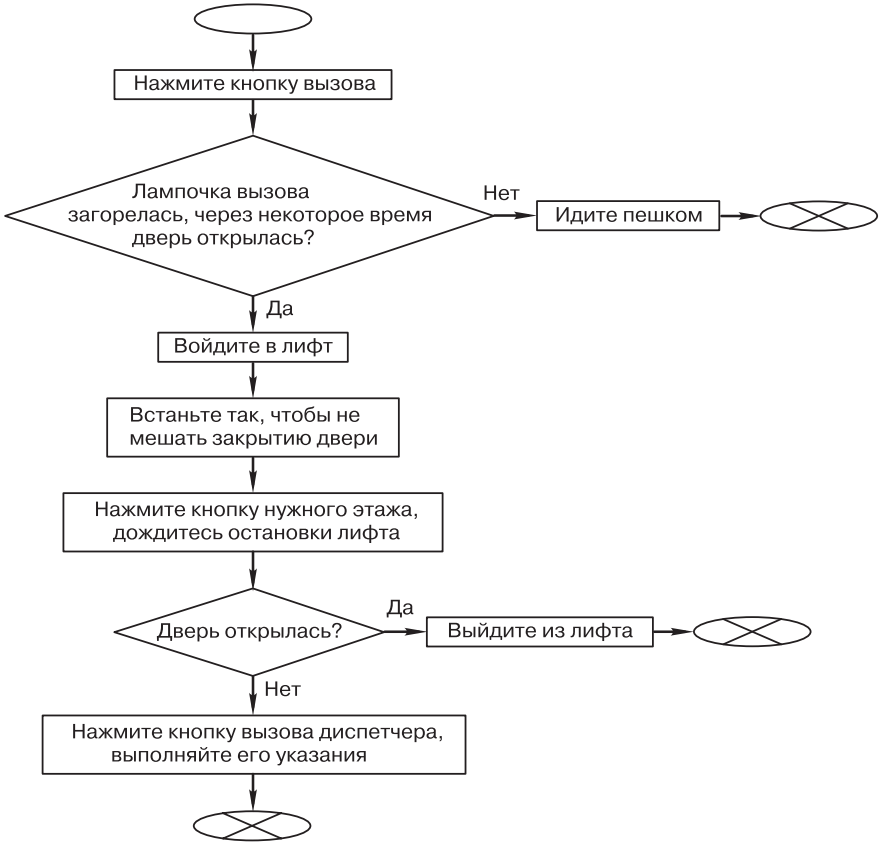

Рис. 1.2

Обратите внимание, в этой схеме мы не использовали блоки ввода и вывода. Так получилось потому, что это алгоритм не для компьютера, а «бытовой». В программах, которые мы будем писать для компьютера, обычно (но не обязательно) будет присутствовать ввод данных и обязательно — вывод результатов (если в результате выполнения программы ничего не будет выведено — как же понять, что она сделала).

Задание. *Понятно, что наш алгоритм неполон. Например, лифт может остановиться, не доехав до нужного этажа, если его кто-то вызвал. Продолжите разработку данного алгоритма. Подумайте, какие инструкции нуждаются в уточнении.*

Переменная. Присваивание

Составим алгоритм для решения следующей задачи. *Человек приобрел огород треугольной формы. Какой длины забор ему понадобится, чтобы огородить свой участок?*

Понятно, что надо измерить все стороны треугольника и найти его периметр, сложив полученные величины. Для записи формул в математике, физике обычно используют буквенные обозначения. Мы тоже ими воспользуемся, обозначив длины сторон треугольника a, b и c, а периметр — P. Блок-схема алгоритма изображена на рис. 1.3.

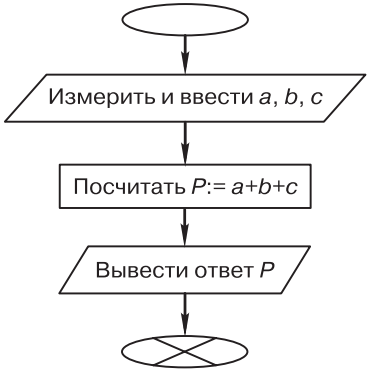

Рис. 1.3

Обозначенные буквами величины мы будем называть **переменными**. При разных начальных условиях (разных огородах) они будут принимать разные числовые значения, меняться.

Обратите внимание на запись формулы `P := a + b + c`. Здесь вместо принятого в математике знака «=» использован знак, состоящий из двух символов «:=». Будем называть его **присваиванием**.

В математике знак равенства используется в двух случаях: когда надо проверить, равна ли правая часть левой (например, «если $x = 0$, то…»),

и в формулах, когда надо подсчитать значение правой части и положить переменную в левой части равной этому значению (т. е. присвоить ей это значение). Чтобы избежать двойного толкования этого знака, будем в первом случае использовать знак равенства «=» и называть это действие сравнением, а во втором — знак присваивания «:=», а действие — присваиванием переменной нового значения.

Этот значок произошел от такой стрелочки: ⇦. В записи программ для первых компьютеров эта стрелочка показывала, что некоторое число пересылается в определенную ячейку, т. е. запись X⇦5 обозначала инструкцию: «число 5 переслать (положить) в ячейку, предназначенную для X». Таким образом, в левой части операции присваивания всегда пишется переменная, которая должна получить новое значение, а в правой — само значение (оно может вычисляться, т. е. записываться в виде выражения).

С точки зрения компьютера **переменная** — это фрагмент оперативной памяти, в котором хранится значение. Компьютер обращается к этому участку памяти и записывает туда новое значение или считывает то, что там находится. Для наглядности память компьютера можно представить себе как большой шкаф с ящиками. Каждый ящик — это место для переменной. Для того чтобы можно было найти нужный ящик, наклеиваем на него табличку с именем. Теперь при необходимости туда можно положить значение или посмотреть, что лежит в ящике. И еще одна важная особенность: при обращении к переменной мы забираем из ящика не само значение, а его копию, т. е. само значение остается в ящике без изменения.

При выполнении действия `P := a + b + c` сначала вычисляется правая часть, затем результат вычисления помещается в ящик с именем (табличкой) P. Это и есть операция присваивания переменной значения. Она всегда работает справа налево.

Получившийся алгоритм — линейный. Займемся теперь нелинейными алгоритмами.

Условие. Виды разветвлений

Простейшая задача с разветвлением, пожалуй, такая: из двух чисел выбрать наибольшее (договоримся, что, если числа равны, «наибольшим» будет любое из них). Блок-схема этого алгоритма представлена на рис. 1.4.

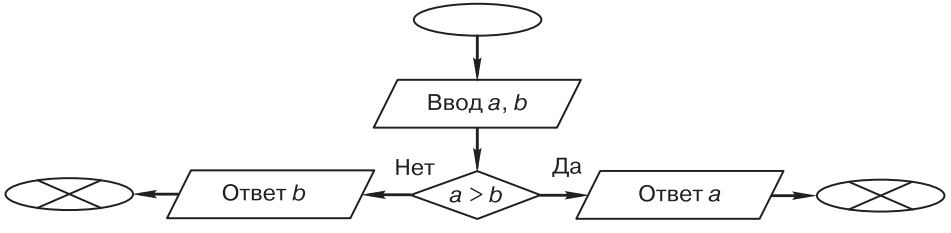

Рис. 1.4

Задание. *Посмотрите, как усложняется алгоритм для трех чисел (составьте соответствующую блок-схему).*

На рис. 1.5 вы видите, как выглядит блок проверки условия (разветвление) в общем случае. Официально он называется **полной условной конструкцией**.

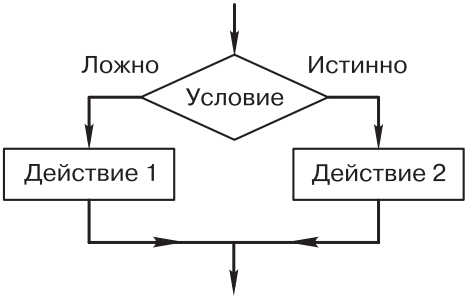

Рис. 1.5

В таком алгоритме выполняется одно из двух действий в зависимости от истинности условия. Однако часто бывают случаи, когда в подобной схеме присутствует только одно действие, которое надо реализовать при выполнении некоторого условия, а в противном случае делать ничего не надо. (Из жизни можно привести такой пример: «Если идет дождь, возьми зонт».)

На рис.1.6 изображен частный случай разветвления — **неполная условная конструкция**, которую удобно называть **обходом**.

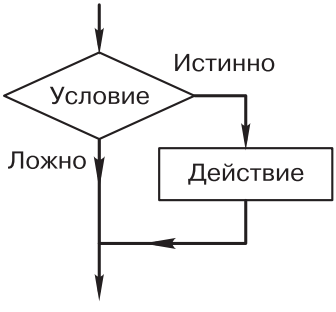

Рис. 1.6

Чтобы реализовать присваивание $X := |X|$, понадобится именно обход, так как значение X надо изменять только в том случае, когда значение X отрицательно (рис. 1.7):

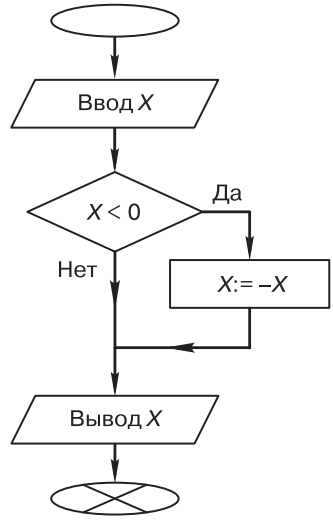

Рис. 1.7

Пример 1.1. *Компьютеру сообщается значение X, а он печатает значение Y, которое находит по формуле:*

$$Y = \begin{cases} X^2 & \text{при} \quad X \leqslant 0; \\ \sqrt{X} & \text{при} \quad X > 0. \end{cases}$$

Блок-схема решения представлена на рис. 1.8.

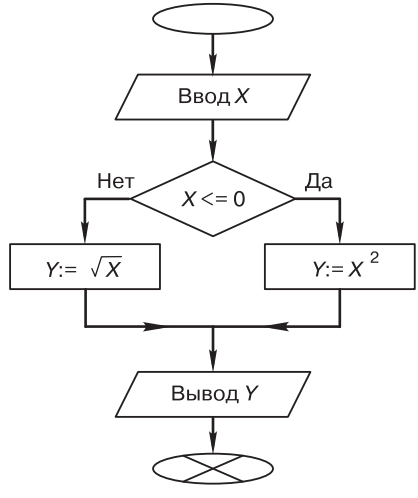

Рис. 1.8

Пример 1.2. *Заданы 3 числа. Определить, можно ли построить треугольник с такими сторонами.*

Чтобы определить, можно ли построить треугольник с заданными длинами сторон, существует несколько способов:

1. Наибольшая сторона должна быть меньше суммы двух других сторон.
2. Наименьшая сторона должна быть больше разности двух других сторон.
3. Каждая сторона должна быть меньше суммы двух других сторон.
4. Каждая сторона должна быть больше модуля разности двух других сторон.

Выбираем третий способ. Обратите внимание: если окажется, что какое-то неравенство не выполняется, то другие проверять уже не надо. Блок-схема решения задачи представлена на рис. 1.9.

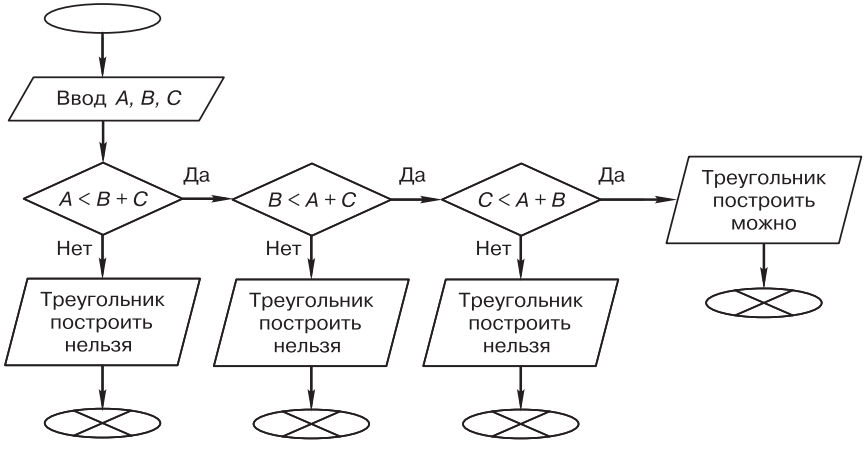

Рис. 1.9

В этой схеме у нас один и тот же блок — параллелограмм, в котором записано, что построение невозможно, — повторяется трижды. Можно ли сократить схему, нарисовав только один такой блок и подведя к нему три стрелочки? Этого делать не сто́ит: получившаяся схема уже не разделится на блоки, описанные выше (условные конструкции в ней нельзя будет считать ни полными, ни конструкциями обхода). Мы будем в наших схемах использовать только описанные выше блоки, так как потом научимся их «переводить» в программы на языке Паскаль.

Задание

1. *Решите задачу из примера 1.2 другими способами.*
2. *Дополните алгоритм. Пусть в случае, если треугольник построить можно, определяется его вид: является ли он равносторонним, равнобедренным, прямоугольным, тупоугольным, остроугольным.*

Цикл

Ранее мы успешно справились с алгоритмом определения периметра треугольника (в задаче про огород). Если бы вместо него в задаче был четырехугольник, наверное, мы ее решали бы так же. А если 10-угольник? К тому же, если сначала измерить все стороны и только после этого их суммировать, понадобится все измерения хранить в памяти. Удобнее измерять очередную сторону и прибавлять ее значение к уже накопленной сумме.

Пример 1.3. *Найдем предложенным на рис. 1.10 способом периметр четырехугольника.*

При таком решении задачи мы будем как бы накапливать значение периметра, постепенно добавляя к нему значения сторон. Значит, когда мы еще ничего не измерили (и не прибавили), периметр равен нулю.

Рис. 1.10

Обратим внимание на инструкции вида `P := P + X1`. Если бы мы не договорились использовать знак «:=», а записывали эти инструкции со знаком «=», получилось бы правильное с точки зрения математики выражение только при $X1 = 0$. В этом и состоит отличие операции присваивания от операции сравнения. Мы сначала вычисляем значение выражения в правой части (используя уже известное значение *P*), а потом значение *P* изменяем (она ведь у нас и называется «переменная»).

Задача решена совершенно правильно, но давайте посмотрим, нужно ли нам длину каждой стороны хранить в особой переменной — ведь она нам нужна только один раз в следующем операторе. Значит, можно длины всех сторон по очереди присваивать одной и той же переменной *X*. Перерисуйте блок-схему в соответствии с этой договоренностью и хорошенько посмотрите на нее. Все правильно, но как-то нехорошо, что практически одни и те же действия записаны 4 раза. А если бы пришлось искать периметр многоугольника, имеющего 100 сторон? Не хочется рисовать такой длинный алгоритм? В нашем алгоритме есть 4 повторяющихся фрагмента, которые различаются только номером измеряемой стороны. Давайте обозначим этот номер переменной *K*, а результат измерения длины стороны будем обозначать все время одинаково *X* (это можно сделать, так как значение длины стороны запоминать не надо, оно нужно только один раз, чтобы прибавить его к периметру). Получим фрагмент, представленный на рис. 1.11.

Рис. 1.11

Организуем работу так, чтобы этот фрагмент повторялся. При этом будем изменять значение *K* (рис. 1.12). Так как вначале нам нужна первая сторона, присвоим *K* единицу, а потом эту переменную будем увеличивать на 1.

После изменения значения *K*, если мы направим стрелочку к нашему повторяющемуся фрагменту, будет измеряться уже вторая сторона многоугольника. Так можно измерить и сложить все стороны. Но где же написать ответ? Где конец нашего алгоритма? Из прямоугольника может выходить только одна стрелочка, и мы направили ее вверх. Откуда взять вторую стрелочку — к ответу?

Дело в том, что после добавления длины очередной стороны к периметру надо не всегда переходить к измерению следующей стороны многоугольника (иначе если мы будем делать это всегда, то «зациклимся», т. е. будем вечно ходить по многоугольнику и измерять его стороны бесконечное

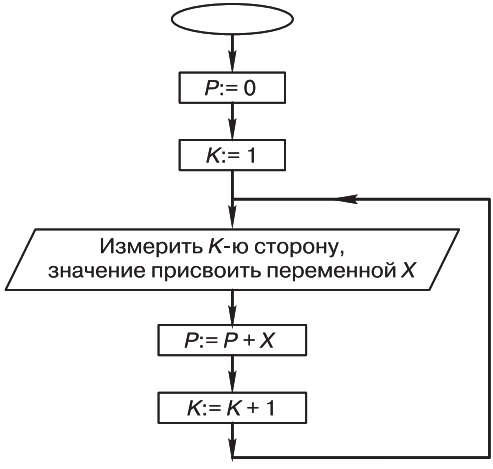

Рис. 1.12

количество раз). Как только все стороны многоугольника измерены, вычисления надо прекратить. Как узнать, что четырехугольник «закончился»? Очень просто — мы ведь знаем, что у него всего 4 стороны, поэтому, если *K* стало равно 5, вычисления следует прекратить, так как пятой стороны у четырехугольника нет. Правильный алгоритм представлен на рис. 1.13.

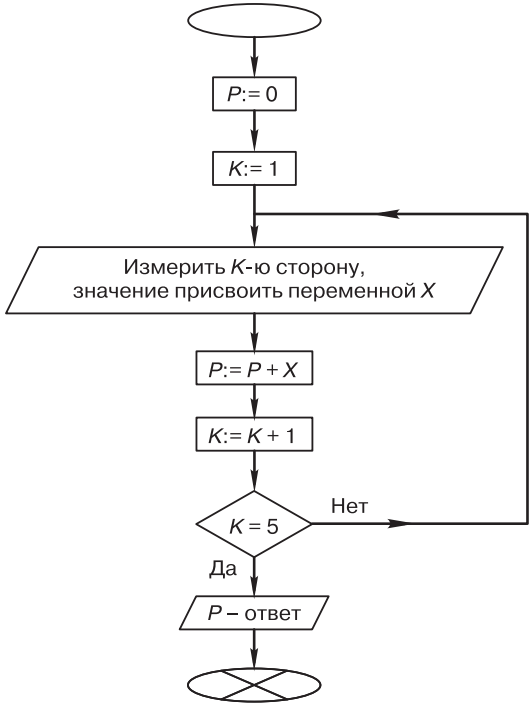

Рис. 1.13

ВМК МГУ – ШКОЛЕ

Развитие и широкое распространение компьютеров вызывают насущную потребность в высококвалифицированных специалистах в области прикладной математики, вычислительных методов и информатики. Сегодня наш факультет – один из основных факультетов Московского университета, ведущий учебный и научный центр России в области фундаментальных исследований и образования по прикладной математике, информатике и программированию.

Высокая квалификация преподавателей и сотрудников факультета, сочетание их глубокого теоретического и практического опыта являются залогом успешной работы наших выпускников в ведущих научных центрах, промышленных, коммерческих и других учреждениях.

Факультет не только учит студентов, но и ведет большую работу со школьниками и учителями:

- на факультете работают вечерняя математическая школа, подготовительные курсы и компьютерные курсы для школьников;
- для учителей есть курсы повышения квалификации и ежегодно проводятся летние школы по математике и информатике;
- сотрудники факультета и преподаватели других факультетов МГУ, работающие на подготовительных курсах факультета, готовят учебные и методические пособия по математике, информатике и физике как для школьников, так и для учителей.

Мы рады видеть новых студентов и приветствуем новых партнеров в научном сотрудничестве и инновационной деятельности.

Декан факультета вычислительной математики и кибернетики МГУ им. М. В. Ломоносова, академик РАН ***Е. И. Моисеев***

Сайт факультета ВМК МГУ:

http://www.cs.msu.ru

ISBN 978-5-00101-196-5