Тема 1. ЭВМ — универсальная машина для обработки информации (4 ч)

Информация и информационные процессы. Информация в истории общества. Двоичное кодирование информации. Единицы измерения информации. Обработка информации.

Краткая история вычислительной техники. Основные компоненты ЭВМ. Потоки информации при работе школьной ЭВМ. Обработка информации на ЭВМ. Программирование как вид человеческой деятельности. Языки программирования.

Тема 2. Исполнители (4 ч)

Школьный алгоритмический язык. Исполнитель Робот. Общий вид алгоритма. Комментарии в алгоритмическом языке. Вызов команды исполнителя. Ошибки в алгоритмах.

Особенности записи чисел в информатике. Исполнитель Чертежник. Последовательное выполнение алгоритмов.

Тема 3. Вспомогательные алгоритмы. Выражения (6 ч)

Понятия основного и вспомогательного алгоритмов. Вызов вспомогательного алгоритма. Метод последовательного уточнения. Разделение труда между ЭВМ и исполнителями. Алгоритмы с аргументами. Выполнение вспомогательного алгоритма с аргументами. Модель памяти ЭВМ.

Правила записи арифметических выражений в алгоритмическом языке. Знаки операций и стандартных функций алгоритмического языка.

Тема 4. Команды повторения (8 ч)

Простые и составные команды. Цикл n **раз**. Команды "обратной связи". Цикл **пока**. Диалог ЭВМ — Робот при выполнении цикла **пока**. Основные свойства цикла **пока**

Тема 5. Команды ветвления и контроля (6 ч)

Общий вид команды **если**. Графическая схема выполнения команды **если**. Условия в алгоритмическом языке. Команда **выбор**. Графическая схема выполнения команды **выбор**. Команды контроля.

Тема 6. Величины в алгоритмическом языке (6 ч)

Необходимость работы с величинами в процессе выполнения алгоритма. Имя, значение и тип величины. Модель памяти ЭВМ. Описание величин. Команда присваивания. Виды величин в алгоритмическом языке. Общие правила выполнения команды вызова вспомогательного алгоритма. Алгоритмы-функции.

ИЗУЧЕНИЕ ОСНОВ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

ПОСОБИЕ ДЛЯ УЧИТЕЛЯ

Рекомендовано Главным учебно-методическим управлением общего среднего образования Государственного комитета СССР по народному образованию

МОСКВА «ПРОСВЕЩЕНИЕ» 1992

ББК 74.262
И 39

Авторы: А.В.Авербух, В.Б.Гисин, Я.Н.Зайдельман, Г.В.Лебедев

Рецензенты:
учитель информатики Томилинской средней школы № 18 М.В.Алексеева; доцент кафедры информатики и вычислительной техники МГЗПИ А.Х.Ин

И39 **Изучение** основ информатики и вычислительной техники: Пособие для учителя / А.В.Авербух, В.Б.Гисин, Я.Н.Зайдельман, Г.В.Лебедев. — М.: Просвещение,1992.— 302 с.: ил. — ISBN 5-09-002845-1.

Содержание пособия составляют методические рекомендации для преподавателей курса "Основы информатики и вычислительной техники", работающих по пробному учебнику А.Г.Кушниренко, Г.В.Лебедева, Р.А.Свореня (М.: Просвещение,1990). В пособии анализируется содержание учебника, предлагаются примерные сценарии уроков, разбираются упражнения и задачи учебника, приводятся дополнительные задания для учащихся.

И $\frac{4306010000-168}{103(03)-92}$ инф. письмо — 92, № 150 ББК 74.262

ISBN 5-09-002845-1

ПРЕДИСЛОВИЕ

Это пособие предназначено учителям, работающим в X — XI классах по пробному учебнику А.Г. Кушниренко, Г.В. Лебедева и Р.А. Свореня "Основы информатики и вычислительной техники" (М., 1990). В пособии даны рекомендации по изучению курса, подготовленные в соответствии с научно-методической концепцией авторов учебника. Предлагаемая методика была апробирована в школах Ленинградского района Москвы и Переславля-Залесского.

Важная особенность учебника в том, что он ориентирован на изучение базовых понятий и методов информатики и теории алгоритмов на сравнительно простом материале. Такая ориентация может вызвать у учителя определенные затруднения в преподавании курса, особенно на начальном его этапе. Одна из целей, которые ставили перед собой авторы пособия, — помочь учителю выявить фундаментальную направленность курса и раскрыть его развивающий потенциал.

Пособие разбито на две части (условно), соответствующие изучению курса в X и XI классах. С учетом сказанного в пособии наиболее подробно разобран материал первой части. Здесь раскрыты основные методические идеи авторов учебника и показаны основанные на них подходы к изучению курса. Параграфы первой части пособия в точности соответствуют параграфам учебника. Каждый параграф состоит из пяти рубрик.

Краткая аннотация. Коротко формулируется основное содержание параграфа, указывается место изучаемого материала в курсе. Раскрываются основные образовательные, развивающие и воспитательные цели изучения данной части курса. Перечисляются конкретные требования к результатам обучения, указывается обязательный для учащихся объем усвоения материала.

Основные понятия. Рубрика посвящена анализу теоретического содержания параграфа. Понятия рассматриваются глубже,

чем в учебнике, разъясняются трудные места. Рубрика может быть полезна как для опытных, так и для начинающих учителей.

Общие указания. Рубрика носит чисто методический характер. Упор делается не на содержании темы, а на трудностях, которые могут встретиться при ее изучении. Даются методические рекомендации по изучению параграфа. Рубрика ориентирована на начинающего учителя информатики (независимо от его квалификации в качестве программиста).

Рекомендации к ведению уроков. В рубрике предлагается примерное содержание уроков по теме параграфа, содержатся дополнительные задачи и вопросы, методические приемы, которые можно применить на уроке. В целом рубрика рассчитана на начинающего учителя, однако и опытный педагог может найти здесь материал для пополнения своего методического арсенала.

Указания к упражнениям. Приведен анализ всех задач и упражнений учебника, рассмотрены возможные усложнения, раскрыты межпредметные связи (большей частью с курсом математики).

Во второй части пособия стиль изложения и структура параграфов иные. В разделе, посвященном алгоритмическому языку, стиль изложения близок к стилю изложения в первой части. Однако число и объем рубрик сокращены, так, рубрика "Указания к упражнениям" отличается только тем, что не разбираются некоторые задачи, имеющие стандартные решения. Материал по второй и третьей главам учебника изложен в форме общих обзоров, в которых рассматриваются основные понятия, анализируется содержание учебного материала и даются общие методические указания по его изучению и решению задач.

Подготовка материала распределялась между авторами следующим образом. X класс: рекомендации к ведению уроков подготовлены А.В. Авербухом, указания к упражнениям — В.Б. Гисиным, содержание остальных рубрик и введения — результат совместной работы. XI класс: § 13—15 гл. 1 подготовлены В.Б. Гисиным, гл. 2 — А.В. Авербухом, гл. 3 — Я.Н. Зайдельманом. Тематическое планирование разработано А.В. Авербухом. Г.В. Лебедевым решен ряд задач и упражнений из разных разделов и написаны методические указания по их решению.

Авторы выражают искреннюю благодарность Л.Е.Самовольновой за постоянную поддержку и помощь в работе над пособием.

ВВЕДЕНИЕ

ОСНОВНЫЕ ПОНЯТИЯ

В школьном курсе информатики можно выделить пять фундаментальных понятий:

1) исполнитель;
2) ЭВМ как универсальное средство обработки информации;
3) алгоритм;
4) информация;
5) структура.

Понятия расположены в порядке возрастания их общности и сложности. Формирование общего представления об этих понятиях и умения применять их на практике — такова цель обучения. Более простые понятия, стоящие в начале списка, изучаются подробнее.

Перечислим основные требования к учащимся, вытекающие из необходимости овладеть базовыми понятиями.

1. Исполнитель. Иметь представление о свойствах исполнителя (система команд, среда, элементарное действие); понимать смысл формального исполнения команд; представлять себе процесс управления с обратной связью.

2. ЭВМ как универсальное средство обработки информации. Знать назначение и общие принципы работы основных компонент ЭВМ, применение, роль и возможности ЭВМ в различных отраслях деятельности человека; понимать формальную сущность работы ЭВМ; уметь оценивать целесообразность использования ЭВМ при решении конкретной задачи; иметь практические навыки работы на клавиатуре; уметь пользоваться прикладными программами (редакторами, базами данных, электронными таблицами, экспертными системами и т.д.).

3. Алгоритм. Знать понятие алгоритма,уметь исполнять готовые алгоритмы, находить в них ошибки, оценивать области применения; уметь составлять алгоритмы и записывать их на школьном алгоритмическом языке; иметь представление о других способах записи алгоритмов; уметь планировать свои действия.

4. Информация. Иметь интуитивное представление об информации, понимать сущность процессов ее накопления, кодирования, передачи и обработки; уметь оценивать информационный объем сообщения и соотносить его с ресурсами ЭВМ; получить общие сведения о двоичном кодировании и представлении информации в ЭВМ.

5.Структура. Понимать общность информационных связей между элементами различных структур (устройство ЭВМ, составные команды алгоритмического языка, иерархия основных и вспомогательных алгоритмов, схема автоматического управления исполнителями); осознавать наличие структуры в объектах различной природы (научная теория, механизм, молекула); понимать возможность формального описания различных структур (например, с помощью графических схем).

ЦЕЛИ

Центральное понятие курса (по месту в списке и по содержанию) — алгоритм. ЭВМ и исполнитель играют в первой главе учебника вспомогательную роль, с их помощью вводится и раскрывается понятие алгоритма.

Основное содержание учебной деятельности при изучении первой главы — составление и анализ алгоритмов.

Существует тесная связь между алгоритмизацией и программированием. Идея алгоритма — ведущая идея программирования на современных процедурных языках. Программирование рассматривается в учебнике как главный способ алгоритмизации. И алгоритмический язык — это весьма простой, но достаточно мощный язык программирования.

Однако алгоритмизация не сводится к программированию. Цель обучения — не подготовка будущих программистов, а привитие школьникам навыков алгоритмического мышления, выходящего далеко за рамки программирования.

Логический подход, принятый в современном курсе школьной математики, устанавливает взаимосвязь фактов. В отличие от него алгоритмический подход рассматривает взаимосвязь действий, в картине мира появляется динамика.

Алгоритмический подход позволяет передавать от человека к человеку не только знания, но и умения.

Основные компоненты алгоритмического мышления — структурный анализ задачи, разбиение большой задачи на малые, сведение нерешенной задачи к решенным, планирование возможных ситуаций и реакций на них, понимание и использование формальных способов записи решения — носят универсальный характер и имеют применения практически во всех сферах человеческой деятельности.

СРЕДСТВА

Школьный алгоритмический язык — не только основной способ записи алгоритмов, но и средство обучения школьников алгоритмизации. Рассмотрим некоторые характеристики алгоритмического языка, обеспечивающие его обучающую функцию.

1. Национальная лексика. Служебные слова языка должны быть понятны школьнику. Иностранные слова создают дополнительную сложность, не имеющую никакого отношения к сути предмета. Более того, заучивание иностранных слов часто затуманивает основную проблему — механизм работы и использование алгоритмических конструкций.

2. Структурность. Внутренняя структурная единица алгоритмического языка — составная команда — обеспечивает единство структуры алгоритма и его записи. (Структура программы на Бейсике — строки, связанные операторами перехода, — затрудняет восприятие алгоритмических конструкций.) Алгоритмический язык (в отличие от Бейсика) имеет еще и внешнюю структуру — иерархию алгоритмов.

3. Независимость от ЭВМ. В алгоритмическом языке нет деталей, связанных с устройством машины, можно вообще не знать, что такое бит, байт, адрес и т.д. Это позволяет сосредоточить внимание на алгоритмической сути решаемых задач.

МЕТОДЫ

Традиционная методика обучения программированию построена на математических объектах и алгоритмах (вычисление НОД, игра Баше, решение уравнений). В курсе школьной информатики такая методика оказалась неудачной.

В о - п е р в ы х, подготовка профессиональных программистов не входит в задачи курса.

В о - в т о р ы х, большинство школьников не имеет достаточной математической культуры.

Отсюда вытекает первая методическая особенность учебника. Изучение основных алгоритмических понятий происходит с минимальным привлечением математики на примерах управления автоматическими устройствами.

В то же время учитель имеет возможность варьировать математическую сложность курса в широких пределах. В настоящем пособии это показано в рекомендациях к упражнениям.

По традиционной методике изучения языков программирования сначала рассматривается очередная алгоритмическая конструкция, а затем демонстрируется ее применение при решении задач. Такой подход (от теории к практике) оправдан только при наличии устойчивой мотивации, интереса к предмету.

Отсюда следует вторая особенность учебника — проблемный подход (от практики к теории). Алгоритмические конструкции вводятся по мере необходимости при решении конкретных задач, а затем — в общем виде.

Формы и методы обучения существенно зависят от возможности доступа учеников к ЭВМ. Можно выделить три вида преподавания.

1. Машинный курс. Уроки информатики проходят непосредственно в компьютерном классе. Машины включаются на уроке по мере необходимости.

2. Полумашинный курс. Имеется регулярный доступ к ЭВМ за пределами школы. В этом случае уроки теории и практики отделены друг от друга.

3. Безмашинный курс. Доступ к ЭВМ эпизодический или вообще отсутствует.

Учебник может быть успешно использован во всех этих случаях. Общие методические рекомендации и поурочные разработки применимы при любом виде преподавания.

Для ускорения трудоемких вычислений при безмашинном выполнении алгоритмов учащиеся могут пользоваться различными калькуляторами, в том числе программируемыми. Однако нельзя считать калькулятор техническим средством обучения какому бы то ни было предмету, в частности информатике. Навыки работы на калькуляторе обеспечивают компьютерную грамотность в той же мере, в какой мастерство машиниста тепловоза обеспечивается умением ездить на велосипеде.

Для повышения интереса школьников к информатике можно широко использовать внеурочные формы занятий: кружки, факультативы, экскурсии, олимпиады, тематические вечера, викторины и т.п. При этом следует учитывать, что компьютерные игры весьма редко способствуют повышению интереса к предмету.

§ 1. ИНФОРМАЦИЯ (1 ч)

> Величайшее литературное произведение — в принципе не что иное, как разбросанный в беспорядке алфавит.
>
> *Жан Кокто*

Параграф посвящен понятию информации. Особое внимание уделяется двоичному кодированию и единицам измерения информации.

Основные цели. Дать учащимся интуитивное представление об информации, научить выделять информационную сторону различных процессов.

Требования к знаниям и умениям. Учащиеся должны уметь выделять информационную составляющую различных процессов; знать принцип двоичного кодирования и единицы измерения информационного объема.

ОСНОВНЫЕ ПОНЯТИЯ

Понятие *информации* столь же важно для описания картины окружающего мира, как более привычные понятия вещества и энергии.

Дать определение информации трудно. Более того, в это слово вкладывается различный смысл в технике, в науке и в обиходе. Тем не менее информация объективно существует и ее можно измерить.

Один из способов измерения количества информации предлагает теория информации, разработанная в 40-х годах Клодом Шенноном. Теория связывает количество информации с математическим понятием вероятности и физическим понятием энтропии. Шеннон связал количество информации в сообщении с вероятностью его появления и ввел единицу измерения количества информации — *бит*.

В технике используется та же единица. Однако сама информация при этом имеет несколько иной смысл. Например, если некоторая информация записана дважды, то в шенноновском понимании ее количество не всегда изменяется, а с технической точки зрения — удваивается. В школьном учебнике информатики понятие информации рассмотрено с технической точки зрения.

Информация кодируется с помощью последовательности сигналов. Такая последовательность сигналов называется *сообщением*.

ОБЩИЕ УКАЗАНИЯ

Рассматриваемый параграф носит ознакомительный характер, поэтому не следует требовать от учащихся знания строгих определений.

Раскрывая идею двоичного кодирования и его универсальный характер, полезно показать, как информация различной природы может быть представлена в виде двоичных кодов. Принципы кодирования текстов и изображений рассмотрены в п.1.4 учебника.

Дополнительно можно продемонстрировать учащимся представление чисел в двоичной форме, не требуя от них умения выполнять перевод чисел из десятичной системы счисления в двоичную.

РЕКОМЕНДАЦИИ К ВЕДЕНИЮ УРОКОВ

Первый урок можно провести в форме беседы о предмете информатики и связанных с ним понятиях. При этом не надо требовать заучивания наизусть определений. Достаточно, если в ходе беседы школьники приоткроют для себя информационную сторону явлений и получат общие представления о способах кодирования информации, о единицах измерения информационного объема.

На трех "китах" покоится мир — вещество, энергия, информация. Первые два "кита" хорошо знакомы учащимся (из физики и химии). Третий "кит" — предмет изучения информатики. Непривычно для школьника выглядит окружающий мир с информационной точки зрения (начиная с того, что человек оказывается бессмертным — осуществленная мечта Герострата).

Можно начать с записи определения информатики в тетрадь. Эта привычная для школы процедура с точки зрения информатики включает несколько этапов:

1. *Кодирование.* Учитель выразил хранящееся в своей памяти определение информатики с помощью слов.

2. *Передача.* С помощью колебаний воздуха (речь) и световых волн (мимика и жесты) он передал сообщение.

3. *Накопление.* Ученики записали его слова в тетрадь,...

4. *Обработка.* ...расставив необходимые знаки препинания.

Информация не может существовать без *физического носителя.* В нашем примере были использованы такие носители: мозг (не будем углубляться в особенности его работы), звуковые волны, световые волны, бумага...

З а д а н и е . Приведите примеры других носителей информации.

Рассмотрим теперь каждый этап. Кодирование информации — это представление ее с помощью какой-нибудь последовательности сигналов. Любая работа с информацией требует ее кодирования. Одну и ту же информацию можно кодировать поразному. Например, сообщение о постигшем бедствии:

• • • — — — • • •

"S O S"

"C O C"

З а д а н и е . Имеется сообщение "я тебя люблю". Закодируйте эту информацию другими способами. Свою готовность дать ответ закодируйте поднятием руки (см. упражнение 3,б).

Один из наиболее надежных и простых способов записи сообщений — ***двоичное кодирование.*** При этом используется всего два сигнала. Например: "0" — "1", "нет" — "да", "выключено" — "включено", "вниз" —"вверх". Таким способом можно записать любое сообщение.

Для примера предложим ученикам задумать любое число от 0 до 15. Учитель берется угадать число, задав ровно четыре вопроса. Ученик может отвечать "да" или "нет". Протокол диалога кодируется на доске цифрами 1 ("да") и 0 ("нет").

Предположим, задумано число 13. Диалог представлен в виде таблицы 1.

Таблица 1

Учитель	Ученик	На доске
1. Задуманное число больше семи?	да	1
2. Задуманное число больше одиннадцати?	да	11
3. Задуманное число больше тринадцати?	нет	110
4. Задуманное число больше двенадцати?	да	1101

На доске получен двоичный код числа 13. Очевидно, что стратегия учителя заключалась в делении интервала пополам, а вопросы строились так: верно ли, что задуманное число находится в половине, содержащей бóльшие числа?

Цель описанного методического приема — показать возможность закодировать любое число в двоичной форме.

При двоичном кодировании каждая цифра (разряд) называется *бит.* Один бит имеет два различных состояния (0 и 1). С помощью двух бит кодируются четыре различных состояния: 00, 01, 10, 11. Три бита — восемь состояний. В общем случае: n бит — 2^n состояний (в нашем примере числа от 0 до 15 требуют четырехразрядного двоичного числа).

На ЭВМ можно обрабатывать не только числа, но и тексты. При этом требуется закодировать около 200 различных символов. В двоичном коде для этого нужно не менее восьми разрядов ($2^8 = 256$). Восемь двоичных разрядов составляют *байт*:

1 байт = 8 бит.

Бит и байт — единицы измерения количества информации (информационного объема сообщения). Из них образуют производные единицы (см. учебник).

В о п р о с . Каков информационный объем такого сообщения: "люблю грозу в начале мая"?

О т в е т . 24 байта, или 192 бита. В сообщении 24 символа, считая пробелы.

Примечание. Объем информации понимается здесь в техническом (не шенноновском) смысле как объем памяти ЭВМ, необходимый для хранения информации в двоичном коде. С точки зрения теории информации рассматриваемая цитата дает неверную информацию, так как по современному календарю тютчевское "начало мая" приходится на середину мая (пример *старения* информации).

Полученное в результате кодирования информации сообщение *источник* информации может передать *получателю*. Происходит это по такой схеме:

Вопрос. Супруга царя Салтана родила Гвидона и хочет обрадовать мужа. Назовите источник информации, получателя, канал связи и помехи.

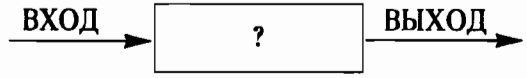
Ответ. Источник — царица;
получатель — Салтан;
канал связи — гонец;
помехи — ткачиха, повариха, Бабариха.

Получатель сообщения может сохранить его, записав на каком-либо носителе.

Задание. Назовите известные вам устройства для записи и хранения информации.

Получатель информации может внести в нее изменения — обработать. Примеры обработки информации: выполнение арифметических действий над числами, исправление ошибок в сочинении, оформление результатов лабораторного опыта.

Задача. Имеется устройство для обработки информации "черный ящик":



Определите по таблице входов и выходов (табл. 2), в чем заключается обработка информации этим устройством.

Ответ. Устройство вычисляет сумму наибольшей и наименьшей цифр поданного на вход числа.

Задание. Придумайте аналогичные устройства, запишите их входы и выходы, предложите вашему товарищу отгадать правила обработки информации.

Таблица 2

Входы	Выходы
48	12
1991	10
183	9
25431	6

Примечание. Обычно на перемене начинается игра в "черные ящики". Правила игры см.: Информатика и образование.—№4.—1988.

УКАЗАНИЯ К УПРАЖНЕНИЯМ

1. Одни и те же процессы можно отнести к различным видам работы с информацией в зависимости от точки зрения, выделения того или иного аспекта. Как правило, в любом процессе присутствуют все три вида работы с информацией. Важно не просто выделить главный, но и дать обоснование такого решения. Любые решения здесь достаточно условны. Главная цель выполнения этого упражнения — заставить учащихся задуматься над происходящими вокруг информационными процессами.

2. Задача разобрана в рекомендациях к уроку. Существует $2^4 = 16$ различных последовательностей из 4 нулей и единиц. Можно перечислить их в порядке возрастания двоичных чисел.

3. Коды всех используемых символов содержатся в п. 1.4 учебника. В задачах б)—д) пробел должен быть закодирован.

а) Слово "РИМ" кодируется последовательностью из 24 двоичных цифр:

11110010 11101001 11101101

(пробелы использованы для удобства чтения);

б) I LOVE YOU:

01001001	00100000	01001100	01001111	01010110	01000101
I		L	O	V	E

00100000	01011001	01001111	01010101;
	Y	O	U

в) TO BE OR NOT TO BE:

01010100	01001111	00100000	01000010	01000101	00100000
T	O		B	E	

```
01001111  01010010  00100000  01001110  01001111  01010100
 O         R                   N         O         T

00100000  01010100  01001111  00100000  01000010  01000101;
          T         O                   B         E
```

г) 1990:

```
00110001  00111001  00111001  00110000;
 1         9         9         0
```

д) 2X+Y=0:

```
00110010  01011000  00101011  01011001  00111101  00110000.
 2         X         +         Y         =         0
```

4. Будем разбивать последовательности бит (двоичных цифр 0, 1) на байты (последовательности из восьми бит) и находить по таблице из п. 1.4 соответствующие им символы:

а) МИМ; б) 161; в) AX+B=0; г) COMPUTER.

5, 8, 9, 10. Для выполнения заданий учащиеся должны самостоятельно получить недостающие данные. Разберем подробно решение задачи 5,б).

Оценим приблизительное число знаков на странице газеты "Комсомольская правда". Размер страницы (в сантиметрах) 54×40. Подсчитаем примерное число знаков на квадратном сантиметре страницы. Выделим на странице газеты несколько участков одинакового размера 10×10 и подсчитаем на каждом из них число знаков. В среднем получим по 2000 знаков на одном участке. Значит, на квадратном сантиметре в среднем около 20 знаков. Страница содержит около 54×40×20=43200 знаков. Один номер газеты на четырех страницах содержит приблизительно 173000 знаков. Следовательно, информационный объем газеты составляет около 170 килобайт. За год выходит 300 номеров газеты. Следовательно, годовая подшивка содержит примерно 170 × 300=51000 килобайт информации, или примерно 51 мегабайт. Ясно, что более точный подсчет очень сложен и вряд ли нужен.

6. В задаче речь идет, конечно, о минимальном числе бит, необходимых для кодирования оценки. Рассмотрим 4 оценки: 2, 3, 4, 5. Для их кодирования достаточно 2 бит (так же, как и для кодирования 4 оттенков цвета точек на фотографиях в п. 1.4). Например, 2 — 11 (черный), 3 — 10 (темно-серый), 4 — 01 (светло-серый), 5 — 00 (белый).

7. Последовательно увеличивая длину кодов n, можно подсчитать их количество при каждом n. Так, кодов длины 2 — 4. Для подсчета кодов длины 3 заметим, что кодов, начинающихся с единицы, — 4; столько же кодов, начинающихся с нуля. Следовательно, всего 2×4=8 кодов. Аналогично число кодов длины 4 равно удвоенному числу кодов длины 3, т.е. 2×8=16, и т.д. Число кодов длины 5 равно 2×16=32; длины 6 — 64; длины 7 — 128.

Этого числа кодов достаточно для кодирования 100 градаций яркости. Значит, для кодирования достаточно использовать 7 бит.

11. Важно в каждом случае четко зафиксировать, какая информация нужна для получения очередного числа (или буквы). В некоторых случаях а), б), в), е), з) каждый элемент однозначно определяется предшествующим. В других (г) для определения очередного элемента нужно знать два предшествующих. Ситуация может быть и более сложной. В д), ж) невозможно указать заранее число предшествующих элементов, по которым определяется очередной.

Правила в случаях а), б) угадываются легко. В случае в) каждое следующее число получается из предыдущего удвоением. В случае г) правило сложнее: здесь каждое число, начиная с третьего, — сумма двух предыдущих. В случае д) выписываются цифры идущих подряд натуральных чисел. В этом ряду очередной элемент зависит от всех предшествующих ему. Например, после цифр 1,9,9 может стоять цифра 2, если речь идет о числах 199, 200, или цифра 7, если речь идет о числе 1997. Правило в случае е): отбрасывается левая буква. В случае ж) выписаны первые буквы названий натуральных чисел: о — один, д — два, т — три и т.д. В случае з) правило довольно сложное. Очередное число описывает предыдущее. Указывается его цифровой состав. Вместо группы идущих подряд одинаковых цифр указывается количество цифр в группе и сама цифра. Например, запись числа 111221 состоит из трех единиц (31), двух двоек (22) и одной единицы (11). Поэтому следом за числом 111221 идет число 312211.

12. а) В предлагаемых текстах каждая буква заменяется следующей за ней по алфавиту. Буква "я" заменяется буквой "а".

б) Для каждого текста указывается число знаков, которые он содержит.

в) В формуле a + b указывается число согласных (a) и число гласных (b) букв в исходном тексте.

г) Первая буква текста переставляется в конец.

Можно провести анализ того, какая дополнительная информация (кроме текста) используется для выполнения преобразования. В а) — это алфавит (упорядоченный список букв). В б) и г) дополнительная информация не требуется. В в) ответ на вопрос о дополнительной информации зависит от трактовки задуманного правила. Если второе слагаемое — число гласных букв, а первое — число всех остальных, то дополнительная информация — список гласных букв. Рассмотрим правило, при котором первое слагаемое — число согласных букв, а второе — число гласных. Чтобы преобразовать текст в соответствии с этим правилом, требуется список гласных (10 букв) и список согласных (21 буква). Впрочем, если считать, что текстовая информация не содержит цифр и других знаков, не входящих в алфавит, то можно посту-

пить более экономно. Достаточно указать список гласных и список, содержащий всего две буквы: ъ и ь.

13. а) Число уменьшается на единицу : $n \to n-1$.

б) Для заданного числа указывается наибольшее непревосходящее его число, кратное трем: $n \to 3[n/3]$, где $[n]$ — целая часть числа.

в) Для заданного числа указывается число десятичных цифр в его записи: $n \to 1 + [\lg n]$.

г) Указывается число "дырочек" в составляющих число цифрах. Цифра 8 имеет две дырочки, 6, 9 и 0 — по одной.

14. Правило можно описать в виде последовательности операций над текстом (алгоритма преобразования текста). Описание удобно делать по шагам. Например, так.

1) Если первый знак — заглавная буква, заменить ее той же буквой , но строчной.

2) Если последний знак текста — вопросительный знак, заменить его восклицательным.

3) К полученному тексту приписать слева "Это ".

Несколько примеров: "?" → "Это !"; "ЭВМ" → "Это эВМ"; " " → "Это ".

Впрочем, правило можно описать и по-другому. Например, 2-й шаг заменить шагами: 2(а) Если последний знак текста — вопросительный знак, стереть его. 2(б) К полученному тексту приписать справа восклицательный знак. Тогда будем иметь: "?" → "Это!"; "ЭВМ"→ "Это эВМ!"; " " → "Это!".

§ 2. ЭЛЕКТРОННЫЕ ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ (1 ч)

Машина должна работать, человек — думать.

Принцип IBM

Параграф посвящен функциональному описанию ЭВМ. Вводится понятие ЭВМ как универсального информационного устройства.

Основные цели. Дать общее представление об основных компонентах ЭВМ и их назначении.

Требования к знаниям и умениям. Учащиеся должны иметь представление об ЭВМ как универсальной машине для работы с информацией, о составе и некоторых технических характеристиках ЭВМ; уметь решать оценочные задачи (не делая точных вычислений).

ОСНОВНЫЕ ПОНЯТИЯ

Важнейшее свойство ЭВМ — *универсальность*. Современные ЭВМ обрабатывают информацию различной формы (числа,

тексты, изображения, звуки и т.д.) и содержания. Универсальность ЭВМ основывается на том, что любые явления и процессы окружающего мира могут быть описаны с информационной точки зрения, а различные формы информации могут быть представлены с помощью двоичного кода.

Универсальность, однако, не означает одинаковости. Различные модели компьютеров существенно отличаются по своим возможностям. Это легко понять: всякий автомобиль есть средство передвижения, но характеристики, а следовательно, возможности и применения самосвала и "Жигулей" отличаются достаточно заметно.

Возможности ЭВМ определяются ее комплектностью, т.е. набором подключенных устройств, и техническими характеристиками.

Рассмотрим указанные в учебнике характеристики с точки зрения их влияния на функциональные возможности ЭВМ.

Например, разрядность процессора практически несущественна для пользователя, она влияет на сложность программирования тех или иных задач, но почти не сказывается на выборе круга задач, для решения которых пригоден конкретный компьютер.

Быстродействие процессора почти несущественно при решении задач в постоянном диалоге компьютера с человеком, например, при работе с текстами. В этом случае скорость работы определяется человеком, а ЭВМ большую часть времени находится в ожидании. А вот при выполнении инженерных или научных расчетов, при обработке больших объемов информации быстродействие может оказаться важнейшей характеристикой, определяющей выбор компьютера. Например, моделирование сложных процессов (космические полеты, атмосферные и природные явления) стало возможным только после появления суперкомпьютеров с быстродействием около миллиарда операций в секунду.

Важнейшая характеристика компьютера — объем ОЗУ (оперативного запоминающего устройства). В памяти хранится выполняемая программа и обрабатываемая информация, поэтому недостаток памяти может существенно ограничить сложность решаемых задач. С ростом памяти количество переходит в качество — возможности ЭВМ принципиально меняются: компьютер с 16 Кбайтами — не более чем дорогая игрушка, малопригодная для серьезной работы, 64 Кбайтами — приемлемый рабочий инструмент для простейших применений, 640 Кбайтами — инструмент профессионала в самых разных сферах деятельности.

Из внешних устройств ЭВМ остановимся на внешней памяти. Машина без внешней памяти практически бесполезна: нельзя сохранить ни программу, ни результаты ее работы. Важная ха-

рактеристика машины — объем одновременно подключаемой внешней памяти. Например, для обработки текста может хватить и сотни килобайт, а работа с большими банками данных требует многомегабайтных дисков.

Иногда оперативная и внешняя память заменяют друг друга: недостаток одного вида памяти компенсируется избытком другого. Использование части ОЗУ как внешней памяти называется *виртуальным диском*, а использование внешней памяти как расширения ОЗУ — *виртуальной памятью*.

ОБЩИЕ УКАЗАНИЯ

При машинном варианте содержание параграфа привязывается к конкретной конфигурации имеющихся ЭВМ. Для демонстрации возможностей ЭВМ используется соответствующее программное обеспечение. Целесообразно адаптировать упражнения в соответствии с техническими характеристиками имеющихся ЭВМ.

При безмашинном варианте технической стороне уделяется меньше внимания. Основной упор следует сделать на информационной сущности ЭВМ, их применении в различных сферах человеческой деятельности.

Предлагаемые упражнения нацелены на формирование представлений об информационных объемах текстов и изображений, о некоторых количественных характеристиках компьютеров.

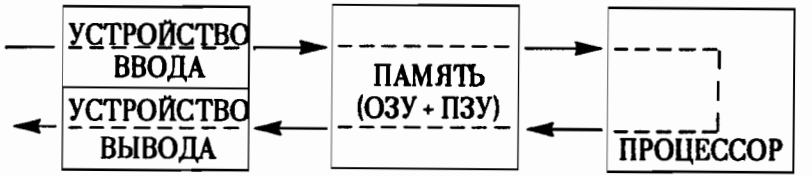
РЕКОМЕНДАЦИИ К ВЕДЕНИЮ УРОКОВ

На прошлом уроке были рассмотрены устройства для обработки информации — " черные ящики ". ЭВМ предназначены для той же цели:

Однако каждый из " черных ящиков " обрабатывал информацию фиксированной формы по жестко заданному правилу. В отличие от них, ЭВМ может обрабатывать информацию *разного вида* (числа, буквы, знаки, чертежи, звуки и т.д.) по *различным* правилам. То есть ЭВМ — *универсальная* машина для работы с информацией.

Важная особенность ЭВМ по сравнению с обычными исполнителями: "программа управления" действиями ЭВМ является частью ее среды.

После краткого изложения истории возникновения и развития вычислительной техники рассмотрим упрощенную схему ЭВМ:



По этой схеме учитель рассказывает о составе персональной ЭВМ, названиях и назначении ее отдельных частей, их взаимосвязи. При этом необходимо дать пояснения к терминам "разрядность" и "видеопамять" из таблицы на с. 15 учебника. Следует по возможности избегать применения специальной терминологии (т.е. слов типа " адаптер ", " контроллер ", " интерфейс " и др.).

После ознакомления с составом персональной ЭВМ и рассмотрения информационных связей между компонентами локальной сети в школьном кабинете учащиеся осваивают диалоговую демонстрационную учебную программу из имеющегося обеспечения. Для приобретения навыков работы на клавиатуре ЭВМ рекомендуется использовать игровые клавиатурные тренажеры.

В безмашинном варианте конец урока можно посвятить выполнению упражнений, помещенных после параграфа.

УКАЗАНИЯ К УПРАЖНЕНИЯМ

1,2. Упражнения служат для общего развития, заставляют задуматься о роли и месте ЭВМ в современном мире.

3. Воспользуемся данными из п. 1.5 учебника. На каждой странице примерно 50 строк по 60 символов. Итого, на одной странице около 3000 символов, т. е. информационный объем одной страницы составляет приблизительно 3 Кбайта. Значит, в оперативную память ЭВМ "Корвет" поместится примерно 20 страниц учебника.

4. Чтобы запомнить изображение, нужно сохранить информацию о каждой точке — черная она или белая. Для кодирования информации о точке достаточно одного бита. Например, коду 0 отвечает белая точка, коду 1 — черная. Следовательно, всего для кодирования изображения потребуется 256×512 бит. Так как $256 = 2^8$ и $512 = 2^9$, то требуется 2^{17} бит, или 2^{14} байт, или 2^4 килобайт. Таким образом, требуется 16 Кбайт. Этот пример показывает, насколько информационно емкими являются изображения. Для хранения изображения размером с экран монитора

требуется примерно такой же объем памяти, как для хранения 5 книжных страниц.

5. В этой задаче для кодирования информации о точке потребуется 3 бита: имеется ровно 8 последовательностей из трех двоичных цифр. Следовательно, по сравнению с запоминанием черно - белого изображения информационный объем возрастает втрое и составляет 48 Кбайт.

6. Примерный объем страницы 3 Кбайта (см. упражнение 3). На дискете объемом 720 Кбайт умещается порядка 240 страниц текста. Значит, для хранения всего учебника достаточно одной дискеты. В упражнении 5 вычислен информационный объем цветного изображения — 48 Кбайт. Следовательно, на дискете можно сохранить 15 цветных изображений. Информационный объем одного номера газеты "Правда" (6 газетных страниц) составляет примерно 250 Кбайт (см. упражнение 5 из § 1). Следовательно, на дискету поместится 2—3 номера газеты.

7. Плата за электроэнергию в расчете на одно рабочее место составляет 1.5×4 / 12 = 0.5 к. в час. При оценочном расчете этой суммой можно пренебречь. Тогда расходы на содержание класса складываются из его стоимости и расходов на ремонт и эксплуатацию. Расходы на ремонт и эксплуатацию за 10 лет — 12000 р. Общие расходы на содержание класса составляют 25000 + + 12000 = 37000 р. Расходы на одно ученическое место составляют 37000 / 12 ≈ 3000 р. В течение года класс работает примерно 2000 ч, в течение 10 лет 20000 ч. Стоимость работы на одной ученической ЭВМ в течение часа 3000 /20000 = 15 к. Задача носит абстрактный характер, так как безотказная работа ЭВМ "Корвет" в течение 10 лет маловероятна.

8. Объем учебника — примерно 600 тыс. символов (см. п. 1.5 учебника). Значит, чтобы его напечатать, потребуется около 600 000 / 80 = 7500 с, или немногим более 2 ч.

9. Объем памяти ЭВМ "Корвет" — 64 Кбайта (см. упражнение 3). Скорость передачи информации 9600 бит/с, или 1200 байт/с. Таким образом, время, необходимое для пересылки, равно 64×1024/1200 с, т.е. примерно 55 с. Цветное изображение занимает 3×256×512 бит памяти (см. упражнение 5). Для его пересылки потребуется 3×256×512/9600 с, т.е. примерно 41 с.

10. В этой задаче не хватает данных — в условиях не указан информационный объем диктанта. Сделаем его оценку. Будем считать, что текст диктанта содержит примерно 600 знаков, т.е. его информационный объем составляет 600 байт. На пересылку одного диктанта потребуется около 0.5 с (см. упражнение 9). Значит, на пересылку двенадцати диктантов уйдет примерно 6 с. Общий объем всех диктантов составляет 12×600 = 7200 байт, или примерно 7 Кбайт. Объем дискеты — 720 Кбайт, так что все диктанты, конечно, можно разместить на одной дискете.

После выполнения упражнений 4 — 6, 9, 10 полезно сопоставить объемы памяти, необходимые для хранения графической и текстовой информации.

11. Чтобы убедиться в совпадении двух диктантов, нужно попарно сравнить 600 символов. На это нужно около 6000 операций. Так как в секунду выполняется 600 тыс. операций, то для сравнения двух диктантов потребуется примерно 0.01 с.

§ 3. ОБРАБОТКА ИНФОРМАЦИИ НА ЭВМ (2 ч)

Практически невозможно научить хорошо программировать студентов, ориентированных первоначально на Бейсик: как потенциальные программисты они умственно оболванены без надежды на исцеление.

Эдсгер Дейкстра

В параграфе раскрывается информационная сущность ЭВМ как программируемой машины. Необходимость изучения программирования обосновывается его ролью в отделенном от материального информационном производстве.

Основные цели. Дать общее представление о сущности и необходимости программирования; сформировать первичные представления об алгоритме.

Требования к знаниям и умениям. Учащиеся должны знать различия материального и информационного производства, уметь решать простейшие алгоритмические задачи.

ОСНОВНЫЕ ПОНЯТИЯ

В теоретической части параграфа очень важны пункты 3.1 и 3.4. Здесь рассматриваются понятия *информационного производства* и *программы* как продукта этого производства.

Компьютеризация стимулировала отделение информационного производства от материального, что способствовало осознанию самостоятельной ценности любых инженерных (конструкторских, технологических) и научных разработок. Это подтверждается тем, что торговля технологиями ("ноу-хау") в современном мире играет все большую роль по сравнению с торговлей готовыми изделиями.

Есть, однако, существенное различие между программированием и инженерными разработками.

Инженер-конструктор разрабатывает проект, результат его труда — чертежи нового изделия. Для испытаний и доработок чертежей недостаточно, необходимо создать образец изделия "в металле". В отличие от конструктора программист создает непосредственно готовый продукт, который может быть немедленно испытан и использован.

Отсюда вытекает ряд психологических особенностей программирования. Вот что пишет об этом американский ученый Дж. Вейценбаум: "Инженер безоговорочно погружен в реальный мир. Его творчество ограничено законами этого мира; он... может делать только то, что соответствует этим законам. ...Программист вычислительных машин — творец миров, в которых он сам является единственным законодателем".

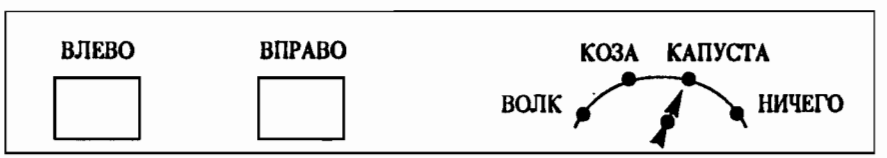
ОБЩИЕ УКАЗАНИЯ

В основном тексте параграфа почти не говорится об алгоритмах. Представление об алгоритмах формируется при решении упражнений. Эти упражнения достаточно разнообразны по содержанию и уровню сложности. Их объединяет то, что результат решения задачи — не число или формула, а описание некоторого процесса. Полезно по возможности вводить элементы формализации: это облегчает решение и готовит учащихся к изучению понятия исполнителя и алгоритмического языка.

РЕКОМЕНДАЦИИ К ВЕДЕНИЮ УРОКОВ

Абсолютному большинству учащихся хорошо знакомо понятие алгоритма. Многие из них применяют алгоритмы при решении задач на уроках математики, физики, химии. Однако только на уроках информатики алгоритм выступает как *результат* решения задачи. Все задачи, помещенные после § 3, требуют составления алгоритмов.

Рассмотрим задачу 1, с. 22. Прежде чем записать ее решение (алгоритм !), договоримся о форме записи. Представим себе, что перевозкой занимается не крестьянин, а некоторое электронное устройство. Попытаемся спроектировать пульт управления этим устройством. После обсуждения, во время которого предлагается множество лишних кнопок ("погрузи", "выгрузи", " греби" и т.д.), ученики, как правило, приходят к пульту, показанному ниже:



Пульт состоит из двух кнопок ("влево" и "вправо") и переключателя, имеющего четыре положения: "волк", "коза", " капуста", "ничего". Предположим, все на левом берегу. Установим переключатель в положение "коза" и нажмем кнопку " вправо". Электронный "крестьянин" совершит такие действия: погрузит в лодку козу, перевезет ее на правый берег, выгрузит ее

там и остановится в ожидании дальнейших указаний. Запишем алгоритм нажатий на кнопки:

вправо (коза)
влево (ничего)
вправо (волк)
влево (коза)
вправо (капуста)
влево (ничего)
вправо (коза)

Видно, что нажатия на кнопки "влево" и "вправо" чередуются. Если "объяснить" это "крестьянину", то на пульте управления достаточно ограничиться четырьмя кнопками: "волк ", "коза", "капуста", "ничего". Тогда алгоритм упростится:

коза; ничего; волк; коза; капуста; ничего; коза

Проанализируем ход наших мыслей. Почему оказались лишними предложенные ребятами кнопки "греби", "погрузи" и т.д.? Приходим к выводу: "крестьянин" "знает" порядок своих действий при перевозке, ему лишь надо сообщить, какой груз взять. Первая команда нашего алгоритма — "коза" — запускает встроенный в "крестьянина" алгоритм "перевозка":

погрузи (...); греби обратно; выгрузи; жди

Команда "греби обратно" приводит к чередованию направлений влево — вправо. По команде "коза" "крестьянин" подставляет слово "коза" вместо многоточия (в качестве *аргумента*) и исполняет алгоритм "перевозка". При этом первая же его команда запускает алгоритм "погрузи", тот вызывает к действию свои алгоритмы и т.д.

Приходим к выводу, что чем меньше "знает" исполнитель алгоритма, тем более подробно следует описывать алгоритм. Или: *степень детализации алгоритма зависит от системы команд исполнителя* (точную формулировку этого правила дадим позже, после изучения понятия исполнителя).

Таким образом, анализ этой простой алгоритмической задачи позволил ввести (пока в качестве пропедевтики) важнейшие понятия информатики — *исполнитель, система команд, элементарное действие, вспомогательные алгоритмы.*

Видно, что любая из предложенных после параграфа задача позволяет проиллюстрировать важные понятия алгоритмизации.

Приведем еще один пример анализа задачи на уроке. Выполним задачу 4 на с. 24. После ее самостоятельного решения школьники приходят обычно к одному из трех вариантов:

1-й вариант:	2-й вариант:	3-й вариант:
долить 5 л	долить 5 л	долить 5 л
долить 5 л	слить 3 л	слить 3 л
долить 5 л	долить 5 л	долить 5 л
слить 3 л	слить 3 л	слить 3 л
слить 3 л	долить 5 л	слить 3 л
слить 3 л	слить 3 л	долить 5 л
слить 3 л	слить 3 л	слить 3 л

В о п р о с . Каков минимальный объем ванны, при котором возможно решение задачи?

О т в е т . Семь литров.

В о п р о с . Какой алгоритм кажется вам лучшим?

О т в е т . Третий, потому что годится для большего числа различных ванн.

Делаем вывод: один из критериев качества алгоритма —его *область применения.*

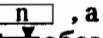
Ответим на вопрос: почему в курсе информатики мы рассматриваем задачи на составление алгоритмов? Потому что в основе работы ЭВМ лежит *программный принцип,* а универсальность ЭВМ (принципиальное отличие их от прочих устройств для обработки информации) основана на сменяемости программ.

Как известно, ЭВМ лучше всего понимает информацию, записанную в двоичном коде. Однако для человека затруднительно изъясняться на таком языке. Обратный вариант — научить ЭВМ человеческому языку — тоже не годится, потому что язык человека полон двусмысленностей и недомолвок. Необходим компромисс: общение человека с ЭВМ должно происходить на языке, промежуточном между машинным и человеческим. Таким средством служат *языки программирования.*

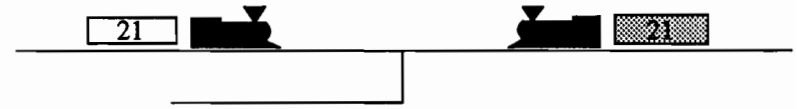
Далее учитель дает краткий обзор языков программирования, показывает информационные связи между отдельными компонентами ЭВМ, раскрывает особенности информационного производства, его отличия от материального производства.

УКАЗАНИЯ К УПРАЖНЕНИЯМ

1. Задача разобрана в рекомендациях к урокам.

2. Будем обозначать паровозы знаками  и  в зависимости от направления движения. Сцепку из n вагонов, идущих слева, будем обозначать через  , а идущих справа — через  . Например, обозначает сцепку, в которой к паровозу, идущему слева, прицеплены два вагона, идущих справа, а к ним еще три вагона, идущих слева. Решение задачи дается на рисунке 1.

Исходное положение:



Требуется получить конечное положение:

Решение:

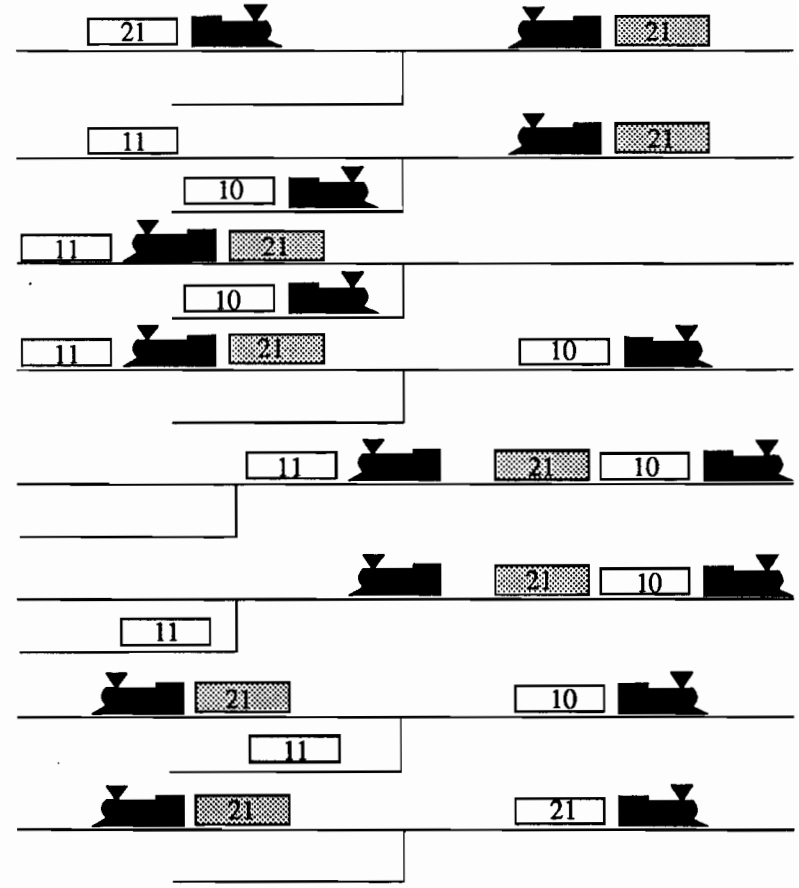


Рис. 1

3. Будем нумеровать кольца числами так, что большему кольцу присваивается больший номер. Размещение колец будем описывать тремя колонками чисел, в которых перечислены номера колец, надетых на соответствующий стержень. Пусть n —число колец. Тогда в общем виде задачу можно сформулировать так: требуется из исходного положения а) прийти в заключительное положение б) (рис. 2).

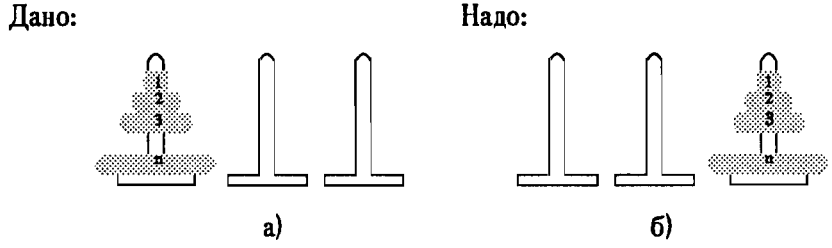


Рис. 2

Разрешается из любого столбца переписать верхнее число в любой другой столбец, соблюдая условие: числа в столбцах должны быть расположены сверху вниз в порядке возрастания.

Решение при n=2:

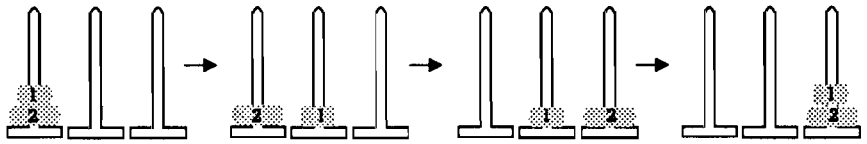


Рис. 3

Опишем теперь план решения для n=3. Сначала, так же как и при n=2, перекладываем кольца 1 и 2 с левого стержня на средний. Затем перекладываем кольцо 3 с левого стержня на правый. Затем снова, как и при n=2, перекладываем кольца 1 и 2 на правый стержень. Введем обозначения для процедуры перекладывания колец. Перекладывание колец 1, 2 с левого стержня на правый будем обозначать через 1—2 (1)→(3), на средний — через 1—2 (1)→(2) и аналогично для других случаев. Тогда план решения можно записать так:

1—2	(1)→(2)
3	(1)→(3)
1—2	(2)→(3).

Запишем теперь план решения для n=4:

1—3	(1)→(2)
4	(1)→(3)
1—3	(2)→(3).

Аналогичные планы получаем для n=5, 6 и т.д. В общем случае, если известно, как переложить n—1 кольцо, то без труда можно получить план перекладывания n колец:

1—(n—1)	(1)→(2)
n	(1)→(3)
1—(n—1)	(2)→(3).

Заметим, что такой же план был применен и в случае n=2.

Подсчитаем теперь число перекладываний. Обозначим через f(k) число перекладываний для переноса k колец с одного стержня на другой. Очевидно, f(1)=1, f(2)=3. В общем случае получаем:

1—(n—1)	(1)→(2)	f(n—1) перекладываний,
n	(1)→(3)	одно перекладывание,
1—(n—1)	(2)→(3)	f(n—1) перекладываний.

Отсюда $f(n) = 2f(n-1) + 1$.

Получаем: f(3)=7; f(4)=15; f(5)=31. Можно заметить закономерность: $f(3)=2^3-1$; $f(4)=2^4-1$; $f(5)=2^5-1$. В общем случае $f(n)=2^n-1$. Действительно, $f(6)=2\times2^5-1=2^6-1$ и т.д. Чтобы перенести 64 кольца, нужно сделать $2^{64}-1$ перекладываний.

Оценим величину этого числа. Как нетрудно подсчитать, в году 31 536 000 секунд. Считая, что в секунду делается 10 перекладываний, получаем, что за год можно выполнить примерно 32×10^7 перекладываний. Далее, $2^{10}=1024$, так что $2^{64}-1 = 2^4(2^{10})^6-1 = 16\times1024^6-1 > 16(10^3)^6 = 16\times10^{18}$. Следовательно, трудясь безостановочно, монахи будут перекладывать кольца более чем $(16\times10^{18})/(32\times10^7)$ лет, т. е. более 50 млрд. лет.

4. Предположим, что кнопка "долить 5 л" нажата x раз, а кнопка "слить 3 л" — y раз (предполагаем, что эта кнопка нажимается только когда в ванне не менее 3л воды). Тогда если в ванне 3 л воды, то 5x—3y=3. Отсюда 5x=3(y+1). Так как 5x делится на 3, то и x делится на 3. Различные неотрицательные решения этого уравнения получаем, когда x пробегает значения 3, 6, 9... При x=3 имеем y=4; при x=6 имеем y=9 и т.д. Впустую пропадает 3y л воды. Значит, в лучшем случае пропадает 12 л воды. Различные планы решения рассмотрены в рекомендациях к уроку.

5. Нужный эффект от решения задачи будет достигнут, если выработать формализованный язык для описания действий. Рассмотрим один из возможных вариантов. В качестве основного элементарного действия выберем переправу. Переправа описывается тремя параметрами: направлением (слева направо или спра-

ва налево) и пассажирами. Условимся обозначать мальчика буквой М, солдата — буквой С. Пассажиры перечисляются внутри пары скобок — квадратной и угловой. Угловая скобка указывает направление переправы. Например, запись [М М> означает переправу, при которой два мальчика переправляются с левого берега на правый.

Опишем последовательность переправ, после исполнения которых два солдата будут переправлены с левого берега на правый. Для удобства слева и справа от описания переправ указываются солдаты и мальчики, находящиеся на левом и правом берегу. Мы считаем, что в начальный момент лодка причаливает к левому берегу (солдаты подзывают мальчиков).

СС	[ММ>	
СС	<М]	М
СМ	[С>	М
СМ	<М]	С
С	[ММ>	С
С	<М]	МС
М	[С>	МС
М	<М]	СС

Заметим, что дважды выполняется последовательность из четырех переправ. Эти 4 переправы позволяют перевезти одного солдата с левого берега на правый, причем лодка с мальчиком оказывается снова у левого берега. Повторяя эту последовательность, можно перевезти с левого берега на правый любое число солдат.

6. При каждом своем ходе Петя берет столько спичек, чтобы оставшееся число спичек было кратно 4. Это возможно при первом ходе — Петя должен взять 3 спички, останется 12 спичек. В дальнейшем, если Коля берет k спичек, то Петя берет 4—k спичек. Так что после того, как Коля и Петя возьмут спички, число спичек уменьшится на 4, значит, снова число оставшихся спичек будет кратно 4. После ходов Пети будет оставаться 12, 8, 4, 0 спичек — это дает Пете победу.

Такая стратегия годится для любого числа спичек, не кратного 4, и обеспечивает победу тому, кто ходит первым. Если же исходное число спичек кратно 4, то та же стратегия обеспечивает победу тому, кто ходит вторым.

К решению задачи можно подойти индуктивно, последовательно рассматривая игру с числом спичек равным 1, 2, 3, 4, 5 и т.д. Легко понять, что проигрывает тот, кому приходится брать спичку из кучки с 0, 4, 8, 12 спичками.

7. Эта задача аналогична задаче 4. Чтобы получить решение, нужно представить число 7 в виде целочисленной комбинации

чисел 3 и 8. Например, 7 = 2×8 — 3×3. Исходя из этого, получаем решение.

Устанавливаем одновременно 8- и 3-минутные часы. По истечении 3 мин переворачиваем 3-минутные часы, затем переворачиваем их еще раз по истечении 6 мин. По истечении 8 мин переворачиваем 8-минутные часы. После того как кончится песок в 3-минутных часах (это произойдет по истечении 9 мин), начинаем приготовление эликсира. Его приготовление заканчивается в тот момент, когда кончится песок в 8-минутных часах.

Заметим, что в общей сложности затрачено 16 мин и выполнено 5 установок песочных часов. Увеличив число установок до 6, можно выиграть одну минуту: 7 = 5×3 — 8. При таком способе 3-минутные часы запускаются 5 раз. Отсчет времени для приготовления эликсира начинается после того, как кончится песок в 8-минутных часах.

8. Будем условно обозначать кнопки x+1 и 2x.

а) Самое простое решение — 5 раз нажать x+1. Число нажатий можно сократить до четырех:

x+1	2x	2x	x+1
1	2	4	5

(в первой строчке указаны нажимаемые кнопки, под ними — числа, которые будут появляться на экране).

б,в) Применим поиск решения "с конца" для числа 99. Получаем:

x+1	2x	x+1	2x	2x	2x	2x	x+1	2x	x+1
1	2	3	6	12	24	48	49	98	99

Строчки выстраиваются справа налево. Если очередное число нечетное, вычитаем единицу (это число получается после нажатия кнопки x+1); если число четное — делим на два (и считаем, что это число получено после нажатия кнопки 2x).

9. Разобьем монеты на 50 пар. За одно взвешивание в каждой паре выделяем более тяжелую монету. Все такие монеты складываем в одну кучку, все остальные — во вторую. Ясно, что самая тяжелая монета в первой кучке, самая легкая — во второй. Таким образом, после 50 взвешиваний задача свелась к тому, чтобы в одной кучке из 50 монет найти самую тяжелую, а в другой — самую легкую.

Выбираем в "тяжелой" кучке любые две монеты, сравниваем их и откладываем более тяжелую. Затем каждую из оставшихся 48 монет поочередно сравниваем с отложенной. Если отложенная монета оказывается более легкой, то заменяем ее выбранной, в противном случае отложенной считается прежняя монета. На любом шаге отложенная монета самая тяжелая из проверенных.

Следовательно, после того как будут проверены все монеты, отложенной окажется самая тяжелая. При этом будет сделано 49 взвешиваний.

Аналогичным образом за 49 взвешиваний выделяем в "легкой" кучке самую легкую монету. Всего будет сделано 50 + 49 + + 49 = 148 взвешиваний.

10,11. Рассмотрим ситуацию, когда монет всего три. Сравним произвольную пару монет. Если они имеют одинаковый вес, то третья монета фальшивая, в противном случае фальшивой является более легкая монета.

Выбрать фальшивую монету из 9 можно за два взвешивания: разбиваем 9 монет на три кучки по 3 монеты; за одно взвешивание определяем кучку с фальшивой монетой; еще за одно взвешивание определяем в этой кучке фальшивую монету. Аналогично можно за три взвешивания найти фальшивую монету среди 27 монет, за n взвешиваний — среди 3^n монет.

Из 4, 5, 6 монет выбрать фальшивую можно за два взвешивания. Например, так. Раскладываем монеты на три кучки: 2 + 2 + 0, 2 + 2 + 1 или 2 + 2 + 2. Выбираем произвольно две кучки с одинаковым числом монет. Взвешиваем их. Если кучки имеют разный вес , то фальшивая в более легкой. При одинаковом весе фальшивая монета в оставшейся кучке. Теперь фальшивую монету нужно найти среди оставшихся 1 или 2 монет.

Имея 1000 монет, поступим следующим образом. Возьмем две кучки по 334 монеты и взвесим их. Если они имеют одинаковый вес, то фальшивая монета среди оставшихся 332 монет, в противном случае фальшивая монета в более легкой кучке. Кучку с фальшивой монетой дополним другими монетами так, чтобы в ней оказалось $3^6 = 729$ монет. Среди 729 монет фальшивую можно найти за 6 взвешиваний. Таким образом, фальшивая монета будет найдена за 7 взвешиваний.

Покажем теперь, что не существует способа, который гарантирует нахождение фальшивой монеты среди 1000 монет за 6 взвешиваний. Обозначим через f(n) максимальное число монет, среди которых фальшивая может быть обнаружена за n взвешиваний.

Рассмотрим первое взвешивание. Пусть на чашках весов лежит по k монет. Тогда возможны два случая: 1) весы в равновесии, и, значит, фальшивая монета среди f(n)—2k монет; 2) одна из взвешиваемых кучек монет легче, и, значит, в ней фальшивая монета. Оставшихся n—1 взвешиваний должно хватить для нахождения фальшивой монеты в любом из этих случаев. Следовательно, $k \le f(n-1)$ и $f(n)-2k \le f(n-1)$.

Отсюда получаем:

$$f(n) \le 2k+f(n-1) \le 2f(n-1)+f(n-1) = 3f(n-1).$$

Таким образом, $f(n) \le 3f(n-1)$.

Ясно, что $f(0)=1$ — без взвешиваний фальшивая монета может быть найдена только в том случае, когда она единственная. Тогда $f(1) \le 3f(0) = 3$; $f(2) \le 3f(1) \le 3^2$; $f(3) \le 3f(2) \le 3^3$ и т. д. В частности, $f(6) \le 3^6 = 729$. Так что за 6 взвешиваний фальшивая монета может быть гарантированно выделена не более чем из 729 монет.

Ранее мы указали способ, позволяющий найти фальшивую монету среди 3^n монет. Учитывая это, получаем $f(n)=3^n$.

12. Приведем без доказательства способ, позволяющий решить задачу не более чем за 100 n взвешиваний в среднем. (В худшем случае требуется $2n^2+n$ взвешиваний.)

Представим себе, что монеты расположены в порядке возрастания массы и пронумерованы от 1 до 2n+1. Требуется найти монету с номером n+1.

Возьмем произвольную монету и определим ее условный номер, сравнив ее с остальными (на это требуется 2n взвешиваний). Пусть k монет оказались легче выбранной, а 2n—k — тяжелее. Это значит, что выбранная монета имеет номер k+1. Если k=n, задача решена. При k<n искомая монета находится среди тяжелых, при k>n — среди легких.

Возьмем кучку, содержащую искомую монету. Взяв произвольную монету и определив ее номер тем же способом, получим новую кучку (меньшую предыдущей), содержащую искомую монету. Поскольку размер кучки на каждом шаге уменьшается, процесс конечен.

Решение задачи, приведенной в учебнике (не более чем 100 n взвешиваний в любом случае) можно найти в книге Д.Кнута "Искусство программирования", т. 2.

ГЛАВА 1. АЛГОРИТМИЧЕСКИЙ ЯЗЫК

§ 4. ИСПОЛНИТЕЛЬ РОБОТ. ПОНЯТИЕ АЛГОРИТМА (2 ч)

Щелкни кобылу в нос — она махнет хвостом.

Козьма Прутков

В этом параграфе на примере Робота формируется понятие исполнителя. Элементарные действия Робота просты, наглядны, легко могут быть выполнены в тетради.

Основные цели. Познакомить учащихся с исполнителем Робот, выработать представления о свойствах исполнителей, о записи алгоритмов на алгоритмическом языке.

Требования к знаниям и умениям. Учащиеся должны понимать сущность автоматического управления исполнителем; уметь записывать и исполнять простейшие алгоритмы управления Роботом.

ОСНОВНЫЕ ПОНЯТИЯ

Учебник содержит сравнительно немного теоретического материала, многие термины даже не упоминаются, но учитель должен ясно представлять, на овладение какими понятиями нацелены внешне очень простые примеры.

Исполнитель. Точное определение исполнителя дать очень трудно, да в этом и нет необходимости. Важно понять основные характеристики исполнителя: среда, элементарные действия, система команд, отказы.

Среда или обстановка — это "место обитания" исполнителя. Например, среда Робота — бесконечное клетчатое поле. Стены и закрашенные клетки —тоже часть среды, их расположение и положение самого Робота задают конкретное состояние.

Система команд. Каждый исполнитель может выполнять команды только из некоторого строго заданного списка — системы команд исполнителя. Для каждой команды должны быть заданы условия применимости (в каких состояниях среды может быть выполнена команда) и описаны результаты выполнения команды. Например, команда Робота "вверх" может быть выполнена, если выше Робота нет стены, ее результат — смещение Робота на одну клетку вверх.

Исполнителя можно представить в виде устройства с кнопочным управлением. Каждая кнопка соответствует одной команде исполнителя, ее нажатие означает *вызов* этой команды. После вызова команды исполнитель совершает соответствующее *элементарное действие*.

Важно отметить, что нас интересует результат, а не механизм выполнения команды. Мы не знаем, как Робот красит клетку, на которой стоит: подпрыгивает или поочередно поднимает свои опоры, но для составления алгоритмов знать это совершенно не нужно, достаточно того, что в результате выполнения команды "закрасить" клетка оказывается закрашенной.

Отказы. Отказы исполнителя возникают при вызове команды в недопустимом для данной команды состоянии среды. Ситуации, в которых возникает отказ, различны для различных команд исполнителя. При выполнении некоторых команд отказ никогда не возникает. Пример такой команды — "закрасить". Таким образом, описание отказов исполнителя входит в описание его команд. Отказы могут происходить и при работе ЭВМ (например, при делении на нуль).

Формальное исполнение. Исполнитель ничего не знает о цели алгоритма. Он выполняет все полученные команды, не задавая вопросов "почему?" и "зачем?".

Управление исполнителями заключается в последовательном вызове команд (нажатии на кнопки). В простейшем случае можно считать, что это делает человек. Тогда схема взаимодействия выглядит так:

Человек отдает команду исполнителю, анализирует результат, отдает следующую команду и т.д. Такая схема вполне возможна, она часто применяется на практике (управление автомобилем, бытовыми электроприборами и т.д.).

Однако во многих случаях такая схема оказывается неудовлетворительной. Человек может ошибиться при наборе длинной последовательности команд, неверно проанализировать обстановку, не успеть в критической по времени ситуации.

Возникает идея посредника — управляющего устройства, которое, получив от человека необходимые инструкции, самостоятельно вызывает команды исполнителя. В роли такого посредника может выступать ЭВМ. В дальнейшем предполагается, что управление исполнителями всегда производится с помощью ЭВМ.

Управление в этом случае распадается на несколько этапов:

1. Составление алгоритма. Человек анализирует поставленную задачу и составляет последовательность команд, ведущих к цели. Эта последовательность образует *алгоритм,* который человек передает ЭВМ. Обратная связь с ЭВМ на этом этапе отсутствует. (Здесь не рассматривается процесс отладки, во время которого ЭВМ используется как инструмент разработки, а не исполнения алгоритма.)

2. Исполнение алгоритма. ЭВМ, выполняя алгоритм, управляет исполнителем. При этом возникает обратная связь ЭВМ с исполнителем (об этом подробнее говорится в § 9). Часть времени ЭВМ расходуется на обработку информации без обращения к исполнителю (связь "ЭВМ—ЭВМ"). Важнейшая особенность этого этапа — *отсутствие связи с человеком.* Впрочем, при выполнении диалоговых программ ЭВМ может связываться с человеком и во время исполнения (ситуация рассмотрена в § 13).

3. Получение результатов. По окончании исполнения алгоритма его результатами пользуется человек. Этими результатами может быть как работа, выполненная исполнителем, так и информация, полученная от ЭВМ.

На ЭВМ полностью распространяется *принцип формального исполнения*: ЭВМ не анализирует содержание и смысл алгоритма, она последовательно, команда за командой, исполняет его. Отсюда следует необходимость точной, не допускающей двусмысленности формы записи алгоритма. Такую форму обеспечивают *формальные языки,* один из которых — *школьный алгоритмический язык* — описан в учебнике.

Одна и та же последовательность действий может быть описана разными способами как на одном алгоритмическом языке, так и на разных алгоритмических языках.

Ошибки составления алгоритмов. Из принципа формального исполнения следует, что ни исполнитель, ни ЭВМ не могут совершать ошибок. Все ошибки, связанные с управлением исполнителями, совершает человек. Можно выделить 3 вида ошибок.

1. Синтаксические ошибки. Вызов команды, не входящей в систему команд исполнителя. *Эти ошибки обнаруживает*

ЭВМ. Она не передает исполнителю синтаксически неверные команды и сообщает человеку о допущенной им ошибке.

2. Семантические ошибки. Вызов команды в ситуации, когда эта команда не может быть исполнена. *Эти ошибки приводят к отказу исполнителя.*

3. Логические ошибки. ЭВМ выполнила алгоритм, исполнитель выполнил все команды, но цель (известная только человеку) не достигнута. *Эти ошибки не фиксируются ни ЭВМ, ни исполнителем,* поэтому важно понимать, что "безотказное" выполнение алгоритма еще не означает его правильности.

В заключение рассмотрим общий вид алгоритма. Здесь есть одно существенное отличие от старого учебника — появились строки **дано** и **надо**. В старой версии алгоритмического языка на этом месте стояли описания аргументов и результатов, в новом варианте эти описания включаются в список параметров (§ 6, 12), а их место заняли *область применения* (**дано**) и *цель исполнения* (**надо**) алгоритма.

Это изменение вытекает из общей направленности учебника — замены абстрактных величин наглядными геометрическими образами. Аргументы и результаты — это входная и выходная информация алгоритма, представленная в виде величин. Дано и надо — это входная и выходная информация, представленная в виде описания состояний среды. В этих строках могут появляться и величины, причем не просто перечень аргументов и результатов, но и какие-то ограничения или соотношения между ними.

Отметим, что в **дано** и **надо**, даже если они представлены комментариями, рекомендуется писать утверждения, описывающие состояние среды, а не описания действий. Например, вместо записи в "командном стиле"

надо | закрасить помеченные клетки

лучше записать

надо | помеченные клетки закрашены

Дано и **надо** позволяют ввести полный заголовок уже в самом первом алгоритме, не дожидаясь введения величин. Заголовок алгоритма полностью описывает его применение. Зная заголовок, можно пользоваться алгоритмом, не интересуясь входящими в него командами (если, конечно, алгоритм составлен правильно). Необходимо обеспечить соблюдение условия **дано**, и тогда после исполнения алгоритма будет выполнено условие **надо**.

Можно провести аналогию между выполнением алгоритма и выполнением одной команды исполнителя. В обоих случаях есть ограничения на применение, есть результат, можно не знать

механизма выполнения действий. Фактически, с точки зрения пользователя, управляющего исполнителем с помощью ЭВМ, готовые алгоритмы ничем не отличаются от команд исполнителя. Эта мысль будет подробно развита при изучении вспомогательных алгоритмов, но показать ученикам сходство команд и алгоритмов, обратив внимание на роль заголовка, полезно в самом начале изучения алгоритмического языка.

ОБЩИЕ УКАЗАНИЯ

Не следует требовать заучивания определений алгоритма, исполнителя, алгоритмического языка. Эти определения могут показаться чересчур общими и абстрактными.

В п. 4.2 дается описание исполнителя Робот. Здесь вводятся 5 команд Робота из 17: команды закраски и перемещения. Оставшиеся 12 команд (команды обратной связи) вводятся позднее, в § 9.

Для запоминания первых пяти команд достаточно разобрать один-два примера. При этом нужно обратить внимание учащихся на то, что алгоритмы составляются для решения некоторых точно поставленных задач. Задачи ставятся в форме, близкой к форме постановки задач в курсах математики и физики:

дано — **надо** (требуется получить).

Отметив это сходство и рассматривая алгоритм как решение задачи, важно подчеркнуть специфические черты такого решения.

Алгоритм дает решение не одной, а целого набора задач, входящих в область применения алгоритма. Подробное рассмотрение этого свойства можно отложить до изучения конструкций повторения.

Для решения одной задачи могут быть использованы разные алгоритмы. Это можно показать на примере из п. 4.3, рассмотрев несколько решений поставленной задачи:

1) вправо; вправо; вниз;
2) вниз; вправо; вправо;
3) вправо; вниз; вправо;
4) вправо; вправо; вниз; вниз; вверх;

Можно сделать вывод, что задача имеет бесконечно много решений. Если потребовать, чтобы Робот переходил из А в Б за минимальное число шагов, то верное решение будут давать только первые три последовательности команд. Доказательство этого факта (с изменением направлений) дано в решении упражнения 7.

При разборе примера отказа в п. 4.7 следует подчеркнуть, что изменилась постановка задачи (наличие стены на поле Робота). Выполнение серии команд из 1) приводит к отказу, в то время как серии команд из 2), 3) и 4) позволяют получить решение и новой задачи.

РЕКОМЕНДАЦИИ К ВЕДЕНИЮ УРОКОВ

Первое и основное понятие урока — ***исполнитель.***

Учитель может сделать краткое отступление о том, что исполнители бывают ***формальные*** и ***неформальные.*** Формальный исполнитель имеет жестко заданный список выполняемых команд и при вызове каждой из них непременно совершает ***элементарное действие.*** (Если исполнение невозможно, возникает отказ.)

В отличие от него неформальный исполнитель в ответ на поданную команду может сказать "неохота" и не выполнить ее. Далее будут рассматриваться только формальные исполнители.

Все команды, которые "понимает" исполнитель, образуют ***систему команд исполнителя*** (СКИ). Для неформального исполнителя понятие системы команд расплывчато. Для формального исполнителя систему команд можно записать однозначно. Вспомним автоматическое устройство из упражнения 8 на с. 24. Его СКИ включает две команды:

1) удвоить;
2) увеличить на единицу.

Система команд исполнителя может включать большое число команд (исполнитель Самолет), а может состоять всего из одной команды (исполнитель Мина).

Кроме системы команд, у каждого исполнителя есть некоторая ***среда,*** в которой он обитает.

У п р а ж н е н и е . Составьте описание системы команд и среды исполнителя Крестьянин из упражнения 1 на с. 22.

Р е ш е н и е .

СКИ	СРЕДА
1. влево (коза)	1. река
2. влево (волк)	2. левый берег
3. влево (капуста)	3. правый берег
4. влево (ничего)	4. лодка
5. вправо (коза)	5. волк
6. вправо (волк)	6. коза
7. вправо (капуста)	7. капуста
8. вправо (ничего)	

Рассмотрим исполнителя, которого зовут Робот. Коротко его можно охарактеризовать так: ***дискретная тележка без носа.*** "Дискретная" — значит, передвигается отдельными одинаковыми шагами по разделенному на клетки полю. "Без носа" означает, что Роботу все равно, в каком направлении двигаться.

В отличие от простой тележки, Робот умеет закрашивать клетку, на которой стоит. Это позволяет ему помечать некоторые клетки, например, чтобы запомнить свой путь (как это делал мальчик с пальчик, бросая на дорогу камушки).

Среда Робота (п. 4.2): бесконечное клетчатое поле, на котором могут находиться стены и закрашенные клетки.

Система команд Робота (п. 4.2) состоит из семнадцати команд, из которых пока изучим пять:

1) вверх;
2) вниз;
3) влево;
4) вправо;
5) закрасить.

Рассмотрим задачу 5,а) на с. 29. В каком порядке следует подавать команды Роботу? Например, так:

вправо
вниз
вниз

В о п р о с . Сколько всего вариантов решений?

О т в е т . Бесконечно много.

В о п р о с . Можно ли решить задачу, не используя команду "вниз"?

О т в е т . Нельзя.

Если серию некоторых действий Робот должен повторить один-два раза, то управлять им может человек. Но если требуется сотня повторений, то управление Роботом лучше поручить ЭВМ. В этом случае следует заранее придумать, записать и передать ЭВМ алгоритм управления, т.е. порядок подачи команд Роботу.

Затем рассмотрим общий вид записи алгоритма на ***алгоритмическом языке.***

После **алг** непременно указывается ***имя алгоритма.*** Оно должно быть коротким и выражать суть алгоритма или задачи, которую алгоритм решает. Примеры хороших имен: "вниз до стены", "парабола", "квадратное уравнение".

Пример плохого имени: "прохождение исполнителя Робот по траектории, напоминающей перемещение шахматного коня" (слишком длинно). Еще худшее имя, хоть и короткое: "Васька-дурак".

Последовательность команд алгоритма заключается между служебными словами **нач** и **кон**. Для отделения команд друг

от друга существует два способа: помещать их на разных строках или разделять точкой с запятой.

Составим алгоритм для задачи 5,б):

```
алг сквозь стену
нач
| вниз
| вправо
| вверх
кон
```

Вернемся к схеме автоматического управления Роботом. Мы видим три "действующих лица": человек, ЭВМ, Робот. Человек составляет алгоритм, записывает его в память ЭВМ и дает команду на исполнение алгоритма. Проследим теперь за действиями ЭВМ и Робота на примере алгоритма "сквозь стену" (табл. 3).

Таблица 3

ЭВМ	Робот
1. Находит в памяти алгоритм "сквозь стену".	
2. Находит в алгоритме слово **нач.**	
3. Дает команду "вниз".	
	1. Делает шаг вниз.
4. Дает команду "вправо".	
	2. Делает шаг вправо.
5. Дает команду "вверх".	
	3. Делает шаг вверх.
6. Встречает слово **кон.**	
7. Останавливается и ждет дальнейших указаний.	

Исполнение ролей ЭВМ и Робота во время урока можно поручить двум ученикам. Тогда приведенная выше таблица "оживает".

Затем вводятся служебные слова **дано** и **надо**. Их применение в алгоритме не обязательно, однако зачастую возникает необходимость проверить соответствие исходных данных некоторым требованиям (например: **дано** стоимость > 0). Кроме того, есть возможность наложить ограничения на результаты алгоритма (например: **надо** площадь ≥ 0).

Тогда общий вид алгоритма:

```
алг < имя >
    дано < условие 1 >      } заголовок алгоритма
    надо < условие 2 >
нач
|
|  < серия команд >         } тело алгоритма
|
кон
```

Выполняя алгоритм, ЭВМ совершает действия в таком порядке:

1. Находит алгоритм в своей памяти.
2. Находит слово **дано** и проверяет *условие 1*. Если оно ложно, то сообщает об этом и переходит непосредственно к действию 7.
3. Находит слово **нач**.
4. Выполняет серию команд.
5. Встречает слово **кон**.
6. Находит слово **надо** и проверяет *условие 2*. Если оно ложно, то сообщает об этом.
7. Останавливается и ждет дальнейших указаний.

В таблице действия 2 и 6 не рассматривались.

Алгоритм пишется один раз, а читается и проверяется многократно. Поэтому хорошо оформленный алгоритм должен читаться, как литературное сочинение.

Для этого в алгоритмическом языке предусмотрена возможность писать ***комментарии***, отделяя их от основного текста вертикальной чертой. Если ЭВМ при исполнении алгоритма встречает знак комментария, то с этого места до конца строки она информацию не читает, сразу переходит к следующей строке. Комментарии можно писать как в заголовке, так и внутри тела алгоритма. Последнее особенно полезно в длинных алгоритмах.

Посмотрим, насколько комментарии облегчают понимание работы алгоритма на примере задачи о волке, козе и капусте (упражнение 1 на с. 22).

```
алг коза и другие
   дано  |  все на левом берегу
   надо  |  все целы и на правом берегу
нач
|                    |              берега:
|                    |  левый                  правый
|  вправо (коза)     | волк и капуста   ↔   человек и коза
|  влево (ничего)    | человек,волк,капуста  ↔  коза
|  вправо (волк)     | капуста   ↔   человек,волк,коза
|  влево (коза)      | человек,коза,капуста  ↔  волк
|  вправо (капуста)  | коза   ↔   человек,волк,капуста
|  влево (ничего)    | человек,коза   ↔   волк,капуста
|  вправо (коза)     | задача решена
кон
```


В учебнике говорится об одной неприятной ситуации, названной отказом. Это случай, когда попытка выполнить команду приводит к аварии. Предположим, что первой командой алгоритма "коза и другие" будет "влево (коза)". "Крестьянин" находится на левом берегу и не в состоянии выполнить команду. Возникает ***отказ***.

Вспомним, что в автоматическом управлении участвуют три "действующих лица" и отказ может возникнуть на каждом из трех уровней. Независимо от уровня возникновения отказа, виновник аварии — автор ошибочного алгоритма. Поэтому имя автора надо указывать в комментариях алгоритмов.

Вопрос об ошибках в алгоритмах целесообразно рассмотреть на уроке несколько подробнее. В соответствии с тремя уровнями возникновения отказа различают три типа ошибок. Представим их в форме таблицы (табл. 4). Для примера опять возьмем задачу о перевозке.

Таблица 4

	Смысл ошибки	Название	Пример
1.	Цель не достигнута	Логическая	Волк съел козу
2.	Команда не входит в СКИ	Синтаксическая	Дана команда "суши весла!"
3.	Авария при попытке исполнить команду	Семантическая	Коза на левом берегу и дана команда: "влево (коза)"

Логическую ошибку обнаруживает человек, синтаксическую — ЭВМ, семантическую — исполнитель (без учета ошибок типа деления на нуль).

У п р а ж н е н и е . Составьте три ошибочных алгоритма для упражнения 5,б), с. 30 учебника так, чтобы каждый содержал ошибки одного вида.

В о з м о ж н о е р е ш е н и е (приводится только серия команд):

а) логическая ошибка
   вниз
   влево
   вверх
б) синтаксическая ошибка
   книзу
   направо
   кверху
в) семантическая ошибка
   вправо

Каждый ученик в конце тетради может вести ***словарь служебных слов***. При повторении и закреплении материала запишем:

алг — заголовок алгоритма;
дано — условие, проверяемое перед исполнением алгоритма.;
надо — условие, проверяемое после исполнения алгоритма;

нач — начало тела алгоритма;
кон — конец алгоритма;
| — комментарий;
; — разделитель команд в строке.

На рассматриваемых уроках отрабатываются и закрепляются практические навыки работы на ЭВМ. Алгоритмы, составленные при выполнении упражнений, следует ввести в ЭВМ и отладить (при машинном варианте преподавания).

Часть времени урока придется потратить на освоение клавиатуры. Важно добиться прочных навыков быстрого редактирования текста алгоритма.

Хорошие результаты дает методический прием классификации ошибок и способов их устранения (табл. 5).

Таблица 5

Вид ошибки	Способ устранения
Неверный символ	Подвести курсор на место неверного символа и сразу ввести правильный символ (не стирая предварительно неверный).
Лишний символ	Подвести курсор на место лишнего символа и нажать клавишу сжатия строки.
Пропущенный символ	Подвести курсор на место пропущенного символа, раздвинуть строку и ввести пропущенный символ.

При обучении работе с клавиатурой следует непременно записать назначение клавиш в тетрадях. Не нужно терять время на изучение специальных клавиш, не работающих в программной поддержке курса.

УКАЗАНИЯ К УПРАЖНЕНИЯМ

1. Исполняя алгоритм а), Робот закрашивает соседние клетки, имеющие с исходной общую сторону, и возвращается в исходную клетку. Этот алгоритм можно назвать "закраска смежных клеток".

Исполняя алгоритм б), Робот закрашивает соседние клетки, имеющие с исходной только общую вершину, и возвращается в исходное положение. Этот алгоритм может быть назван "закраска касающихся клеток".

После слова **дано** в обоих алгоритмах может быть записано:

дано | Робот в клетке А, стен на поле нет

После слова **надо** в случае а):

надо | Робот в клетке А, закрашены смежные клетки

в случае б):

надо | Робот в клетке А, закрашены касающиеся клетки

2. В случае а) достаточно убрать из алгоритма 1,б) все команды закраски. В случае б) целесообразно рассмотреть два решения. Можно вставить команды закраски перед каждым шагом Робота, тогда клетка, расположенная сверху от исходной, будет закрашена дважды. Чтобы избежать этого, достаточно не ставить команду закраски перед последним шагом (вниз).

3. Робот проходит клетки по одному разу, двигаясь вверх—вправо в а) и вправо в б). С учетом этого число закрашенных клеток должно совпадать с числом групп идущих подряд команд закраски. В а) закрашено 4 клетки, в б) — 3.

4. На рисунке 4 сплошной линией изображен маршрут Робота при исполнении алгоритма 3,а) и заштрихованы закрашиваемые клетки. По условиям задачи неизвестно, на каком шаге возникает отказ. Следовательно, чтобы наверняка избежать отказа, нужно указать маршрут, который не будет иметь ни одного общего шага с маршрутом из 3,а). Например:

закрасить
вверх
вправо
закрасить
вверх
вправо
закрасить
вверх
вправо
вниз
вниз
влево
закрасить
вправо
вверх

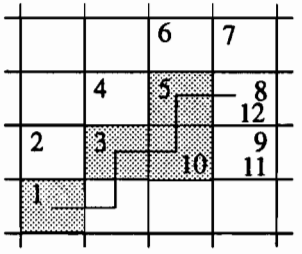


Рис. 4

Маршрут, соответствующий приведенной последовательности команд, изображен на рисунке 4 цифрами.

5. В случаях а) — г) нужно просто описать маршрут Робота. Таких маршрутов существует бесконечно много (например, Робот может "потоптаться" в двух соседних клетках). Можно дополнительно потребовать, чтобы маршрут был кратчайшим, т.е. чтобы Робот делал минимальное число шагов. В случаях б) и в)

существует ровно два кратчайших маршрута из А в Б. Сильным ученикам можно предложить доказать это утверждение. Заметим, что в случае в) можно опереться на доказательство для случая б).

Составление алгоритмов выхода из лабиринта в д)— и) служит подготовкой к изучению темы "Вспомогательные алгоритмы" и команд повторения. При выполнении упражнений нужно выделять на рисунке периодически повторяющиеся участки, составлять алгоритмы перемещения по этим участкам и затем из них составлять полный алгоритм. Образцом может служить алгоритм А2 из п. 4.8.

При выполнении е) может быть предложено дополнительное задание, связанное с тем, что лабиринт в случае е) получается из лабиринта д) поворотом на 90° против часовой стрелки. Поэтому, чтобы получить алгоритм для лабиринта е), достаточно в алгоритме для лабиринта д) сделать следующую замену команд: вверх → влево; влево → вниз; вниз → вправо; вправо → вверх.

6,7. Чтобы попасть из клетки А в клетку Б, Робот должен сделать вниз на два шага больше,чем вверх, и вправо — на один больше,чем влево, т.е.

$$n_н = n_в + 2,$$
$$n_п = n_л + 1.$$

Отсюда, обозначая через n общее число шагов, получаем

$$n = n_в + n_п + n_н + n_л = 2n_в + 2n_л + 3.$$

Маршрут из трех шагов получаем при условии

$$n_в = n_л = 0, \quad n_п = 1, \quad n_н = 2.$$

Приведем пример алгоритма, состоящего из трех команд:

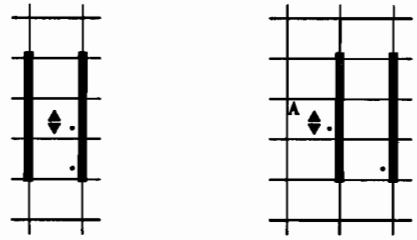
```
алг переход из А в Б
   дано | стен на поле нет, Робот в клетке А
   надо | Робот в клетке Б
нач
| вправо
| вниз
| вниз
кон
```

Еще два алгоритма получаются перестановками шагов: вниз, вниз, вправо; вниз, вправо, вниз.

Так как $n = 2(n_в + n_л + 1) + 1$ — нечетное число, то алгоритмы, при исполнении которых Робот делает 2, 4, 1990 шагов, невозможны. Доказательство этого факта можно сделать более наглядным, если раскрасить клетки поля в шахматном порядке. Робот за один шаг переходит с клетки одного цвета на клетку другого цвета. Таким образом, сделав четное число шагов, Робот будет находиться в клетке того же цвета, что и исходная, в то время как клетки А и Б имеют разный цвет. Алгоритм, содержащий любое нечетное число шагов, большее или равное 3, возможен. Для получения такого алгоритма к алгоритму из трех шагов можно приписать любой набор шагов, выполняя которые, Робот возвращается в исходную клетку. Например, алгоритм при исполнении которого Робот делает 7 шагов, получается, если к алгоритму из трех команд добавить в произвольном порядке по две команды вправо и влево.

8. Алгоритмы определяются условиями задачи неоднозначно, возможны различные решения. Число решений сокращается, если потребовать, чтобы маршрут был кратчайшим и клетки не закрашивались дважды. При соблюдении этих требований алгоритмы в случаях а), б) определяются однозначно; в случаях в) и д) существует четыре алгоритма, различающихся направлением обхода (по или против часовой стрелки) и тем, ставится ли команда закраски исходной клетки первой или последней. В случае е) имеется несколько кратчайших маршрутов (содержащих по 15 шагов), которые несложно указать.

Рассмотрим задание ж). Выделим повторяющийся участок (рис. 5). Заметим, что на таком участке указывается начальное положение Робота.



а) б)

Рис. 5

Исполнив команды:

закрасить; вниз
закрасить; вниз
вправо; вверх; вверх

Робот закрасит отмеченные клетки и перейдет в начальную точку следующего фрагмента (рис. 5,а). Так могут быть закрашены

клетки между стен. Клетки левее первой стены и правее последней требуют при этом специальной обработки:

закрасить; вниз; вниз; вправо; вверх; вверх
закрасить; вниз; закрасить;вверх

Получаем алгоритм:

```
алг из А в Б
  дано | Робот в клетке А
  надо | Робот в клетке Б
нач
  закрасить; вниз; вниз; вправо; вверх; вверх
  закрасить; вниз; закрасить; вниз; вправо; вверх; вверх
  закрасить; вниз; закрасить; вниз; вправо; вверх; вверх
  закрасить; вниз; закрасить; вниз; вправо; вверх; вверх
  закрасить; вниз; закрасить; вниз; вправо; вверх; вверх
  закрасить; вниз; закрасить; вверх
кон
```

Выделять повторяющиеся участки можно по-разному. Соответственно по-разному будут выглядеть и алгоритмы. Например, выделим участок, представленный на рисунке 5,б. Получаем алгоритм:

```
алг из А в Б
  дано | Робот в клетке А
  надо | Робот в клетке Б
нач
  закрасить; вниз; вниз; вправо; вверх; закрасить; вверх
  закрасить; вниз; вниз; вправо; вверх; закрасить; вверх
  закрасить; вниз; вниз; вправо; вверх; закрасить; вверх
  закрасить; вниз; вниз; вправо; вверх; закрасить; вверх
  закрасить; вниз; вниз; вправо; вверх; закрасить; вверх
  закрасить
кон
```

Для решения упражнений 9—11 можно воспользоваться следующими соображениями. Пусть $n_в$, $n_н$, $n_п$ и $n_л$ — число шагов, сделанных Роботом в соответствующих направлениях. Тогда Робот возвращается в исходное положение при условии, что

$$n_в = n_н \quad и \quad n_п = n_л. \qquad (*)$$

9. Так как в оставшихся командах два шага делаются влево и один вправо, то пропущенная команда — "вправо".

10. Если стерта одна из команд перемещения, то нарушается одно из равенств (*) и, следовательно, Робот не будет возвращаться в исходное положение. Таким образом, стертая команда — "закрасить".

11. Так как при исполнении алгоритма, составленного Петей, Робот возвращается в исходное положение, то выполняются равенства (*). При любой перестановке команд эти равенства остаются верными, значит, и после исполнения Колиного алгоритма Робот вернется в исходное положение. Условие отсутствия стен существенно, в противном случае может возникнуть отказ. Нужно обратить внимание, что результаты работы алгоритмов Коли и Пети не идентичны: могут, например, закрашиваться разные клетки.

12, 13. Задачу 12 можно решить непосредственным перебором различных пар команд. Ниже рассматривается общий подход.

Для удобства будем с помощью квадратных скобок обозначать число клеток, закрашиваемых Роботом при исполнении серии команд: [U] — число клеток, закрашиваемых Роботом при исполнении серии команд U. Если U и V — произвольные серии команд, то, очевидно, выполняются неравенства

$$[U], [V] \le [UV] \le [U] + [V].$$

Здесь и далее UV обозначает серию, которая получается приписыванием к серии команд U серии команд V.

Обозначим через A серию вызовов команд Робота в составленном Петей алгоритме. Обозначим через a и b команды, которые меняются местами, через X, Y, Z — серии вызовов команд Робота соответственно до команды a, между командами a и b и после команды b. Тогда можно записать A = XaYbZ. (В задаче 12 серия Y не содержит команд, так что в этом случае A = XabZ.) После перестановки команд получится серия A′= XbYaZ. Выясним, на сколько может измениться число закрашиваемых клеток при переходе от A к A′.

Серии aYb и bYa содержат одни и те же команды смещения. Значит, клетка, в которой окажется Робот после исполнения любой из этих серий, зависит только от начальной клетки и не зависит от самой серии (хотя маршруты Робота могут оказаться и разными).

Сравним теперь, что происходит, когда Робот исполняет команды из серии A = XaYbZ и из серии A′= XbYaZ. Исполнение команд из серии X в обоих случаях приводит к одному и тому же результату. После исполнения серии XaYb Робот будет находиться в той же клетке, что и после исполнения серии XbYa. Значит, исполнение команд серии Z также происходит одинаково в обоих случаях. Различия могут возникнуть лишь при исполнении серий aYb и bYa. Предположим, что при исполнении серии aYb закрашиваются только те клетки, которые закрашиваются при исполнении серии X или Z, а при исполнении серии bYa

не закрашивается ни одна из этих клеток. Тогда, очевидно, [A′] − [A] = [bYa]. Это дает нам наибольшее возможное увеличение числа закрашиваемых клеток. В общем случае

$$[A'] - [A] \le [bYa] \le [b] + [Y] + [a].$$

Так как a — это одна команда, то [a] = 0, если a — это команда смещения, и [a] = 1, если a — это команда закраски. То же верно и для команды b. Если обе команды a и b — это команды "закрасить", то серии A и A′ совпадают, и, значит, [A′] = [A], или [A′] − − [A] = 0. Во всех остальных случаях [a] + [b] ≤ 1. Следовательно,

$$[A'] - [A] \le [Y] + 1.$$

Аналогично,

$$[A] - [A'] \le [Y] + 1.$$

Если Y — пустая серия (не содержит команд), то [Y] = 0 и [A′] − [A] ≤ 1. Таким образом, при перестановке соседних команд число закрашиваемых клеток изменяется не более чем на одну. Если при исполнении серии команд A Робот закрашивает 5 клеток, т. е. [A] = 5, то при исполнении A′ Робот не может закрашивать 3, 6, 7 клеток, так как 4 ≤ [A′] ≤ 6. Значения 4, 5, 6 для [A′] возможны (ниже в примерах курсивом выделены переставляемые команды).

[A] = 5, [A′] = 4.
A: вправо; закрасить; вправо; закрасить; вправо; закрасить вправо; закрасить; ***вправо; закрасить***
A′: вправо; закрасить; вправо; закрасить; вправо; закрасить вправо; закрасить; ***закрасить; вправо***

[A] = 5, [A′] = 5.
A: вправо; закрасить; вправо; закрасить; вправо; закрасить вправо; закрасить; вправо; ***закрасить; закрасить***
A′: вправо; закрасить; вправо; закрасить; вправо; закрасить вправо; закрасить; вправо; ***закрасить; закрасить***

[A] = 5, [A′] = 6.
A: вправо; закрасить; вправо; закрасить; вправо; закрасить вправо; закрасить; вправо; закрасить; ***закрасить; вправо***

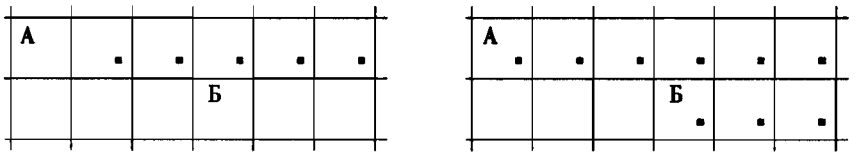
A′: вправо; закрасить; вправо; закрасить; вправо; закрасить вправо; закрасить; вправо; закрасить; ***вправо; закрасить***

Теперь рассмотрим задачу 11. Так как [A] − [A′] ≤ [Y] + 1 и [Y] ≤ [A], то [A′] ≤ 2[A] + 1. Аналогично, [A] ≤ 2[A′] + 1. Это показывает, что при [A] = 5 случаи [A′] = 0, [A′] = 1 и [A′] = 100

невозможны. Случай [A′] = 5 возможен даже при перестановке соседних команд. Случай [A′] = 7 тоже возможен:

A (рис. 6,а):
вправо; закрасить; вправо; закрасить; вправо; закрасить вправо; закрасить; вправо; закрасить
влево; закрасить; влево; закрасить; ***вниз***

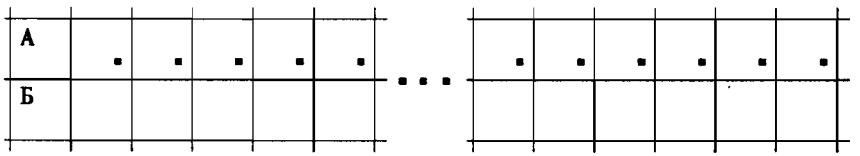
A′ (рис. 6,б):
вправо; закрасить; вправо; закрасить; вправо; закрасить вправо; закрасить; вправо; закрасить
вниз; закрасить; влево; закрасить; ***влево***



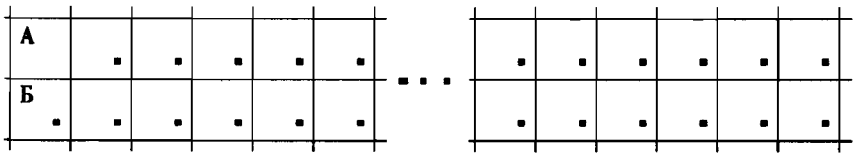
а) б)

Рис. 6

Завершим наше небольшое исследование. Для любого натурального числа n можно указать такую серию команд A, что [A] = n и при перестановке двух команд получается серия A′, для которой [A′] = 2[A] + 1. Приведем примеры:



а)



б)

Рис. 7

A (рис. 7,а):
вправо; закрасить; ... вправо; закрасить (n раз)
закрасить
закрасить; влево; ... закрасить; влево (n раз)
вниз

A′ (рис. 7,б):
вправо; закрасить; ... вправо; закрасить (n раз)
вниз
закрасить; влево; ... закрасить; влево (n раз)
закрасить

Аналогично для любых натуральных чисел s и t, таких, что $s \le t \le 2s + 1$, нетрудно указать алгоритмы A и A′, что [A] = s и [A′] = t.

Окончательно получаем:

алгоритмы A и A′, получающиеся друг из друга переменой местами двух команд, и такие, что [A] = s и [A′] = t можно указать в том и только в том случае, когда

$$s \le t \le 2s + 1 \text{ и } t \le s \le 2t + 1,$$

или

$$(s - 1)/2 \le t \le 2s + 1.$$

В частности, если при исполнении алгоритма закрашивается 5 клеток, т. е. s = 5, то после перестановки пары команд получаются алгоритмы, при исполнении которых закрашивается от $(5 - 1)/2 = 2$ до $2 \times 5 + 1 = 11$ клеток.

14. Достаточно переписать последовательность команд в обратном порядке, а затем в каждой команде перемещения изменить направление шага на противоположное. Решение задачи нужно пояснить примерами.

§ 5. ИСПОЛНИТЕЛЬ ЧЕРТЕЖНИК И РАБОТА С НИМ (2ч)

Я с детства не любил овал!
Я с детства угол рисовал!

Павел Коган

В § 5 изучается новый исполнитель — Чертежник. Изучение Чертежника требует от учащихся умения свободно оперировать декартовыми координатами точек на плоскости, выполнять и записывать в координатах сложение векторов.

Основные цели. Познакомить учащихся с Чертежником, расширить их представления об исполнителях. Подготовить учащихся к изучению вспомогательных алгоритмов (§ 6).

Требования к знаниям и умениям. Учащиеся должны знать команды Чертежника; уметь составлять и записывать алгоритмы управления Чертежником, в частности, алгоритмы рисования букв, цифр и других несложных изображений; по записи алгоритма на алгоритмическом языке воспроизводить действия Чертежника.

ОСНОВНЫЕ ПОНЯТИЯ

Параграф 5 значительно менее, чем предыдущий, насыщен новыми понятиями. Здесь развиваются заложенные в § 4 идеи, демонстрируется внешнее разнообразие исполнителей и единство алгоритмического подхода к управлению ими.

Важнейшее новое понятие этого параграфа — ***команды с аргументами.***

При вызове некоторых команд недостаточно сообщить исполнителю, что он должен сделать. Необходимо передать ему некоторую *дополнительную информацию.* Эта информация передается с помощью аргументов.

В памяти ЭВМ аргументы задаются в виде ***величин.*** Каждая команда требует аргументов определенного ***типа***: например, для команд Чертежника "сместиться в точку" и "сместиться на вектор" аргументами должны быть вещественные числа. Понятия величины и типа подробно рассмотрены в § 11.

ОБЩИЕ УКАЗАНИЯ

Элементарные действия Чертежника достаточно просты и наглядны. В то же время освоение этих действий опирается на определенную математическую подготовку учащихся. Поэтому изучение Чертежника целесообразно сочетать с повторением темы "Векторы на плоскости", уделив особое внимание сложению векторов.

Материал позволяет в широких пределах варьировать математическую трудность заданий в зависимости от уровня подготовки учащихся.

Команды Чертежника в одном отношении существенно отличаются от команд Робота — они имеют аргументы. В § 5 изучение команд с аргументами облегчается тем, что во всех алгоритмах аргументы указываются в виде чисел.

Наибольшую трудность для изучения представляет команда Чертежника "сместиться на вектор (a, b)". Используя эту команду, можно с помощью одного и того же алгоритма получать заданное изображение в разных местах плоскости, меняя лишь начальную точку.

При изучении § 5 можно изменить порядок изложения и после описания команд Чертежника (п. 5.2) рассмотреть пример алгоритма управления Чертежником (п. 5.4), в котором команда смещения на вектор не используется. После этого, когда учащиеся освоятся с новым исполнителем, можно перейти к изучению команды смещения на вектор.

При изучении этой команды нужно напомнить операцию сложения векторов и связать ее с выполнением команды: если перо находится в точке A, то после исполнения команды "сместиться

на вектор (a, b)" перо сместится в точку B такую, что OB = OA + + (a,b) (см. рис. 10 учебника).

В § 5 вводится запись вещественных чисел с точкой вместо запятой. Выработку соответствующих навыков можно отложить до § 7. В примерах алгоритмов в § 5 и 6 используются только целые числа.

РЕКОМЕНДАЦИИ К ВЕДЕНИЮ УРОКОВ

Мы знакомимся с новым исполнителем — Чертежником. Во многом он напоминает реально существующее устройство ЭВМ — графопостроитель. Методически более оправдано ввести команды Чертежника в таком порядке: сначала команду абсолютного смещения (в точку), а затем, после выполнения нескольких упражнений, команду относительного смещения (на вектор).

Выполним несложное задание, например упражнение 4,а), приведенное на с. 37 учебника. При этом будем считать, что в исходном положении перо Чертежника поднято и расположено в начале координат.

алг отрезок
  дано | перо поднято
  надо | нарисован отрезок (рис. 8)
нач
| сместиться в точку (1,2)
| опустить перо
| сместиться в точку (−1,1)
| поднять перо
кон

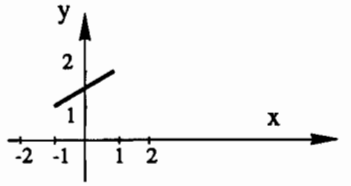


Рис. 8

В составленном алгоритме последняя команда кажется лишней. Однако в ней есть смысл: после работы с Чертежником нужно поднять перо, чтобы при следующем обращении к Чертежнику не нарисовать лишних линий. (В одной столовой висел призыв: "Покушали — убирайтесь!")

Подобного рода правила носят характер рекомендаций, их соблюдение помогает уменьшить количество отказов. В то же время отсутствие последней команды в приведенном алгоритме не может считаться ошибкой.

В о п р о с . Что будет, если выполнить команды алгоритма "отрезок" в обратном порядке?

О т в е т . Будет нарисован тот же самый отрезок, только перо останется опущенным и будет находиться в точке (1, 2).

Вновь вспомним задачу о волке, козе и капусте. Команда переправить козу на правый берег выглядела так: вправо (коза). В командах такого типа есть ***аргумент*** (в данном случае —

коза). В общем виде систему команд Крестьянина можно коротко представить так:

влево (**арг** x);
вправо (**арг** x), где x = волк, коза, капуста, ничего.

Как видим, команда Чертежника тоже имеет аргументы. В общем виде она выглядит так:

сместиться в точку (**арг вещ** x,y).

Служебное слово **арг** показывает ***вид*** переменных x и y. Это аргументы, т.е. переменные, значения которых должны быть известны ***перед*** выполнением команды.

В описании СКИ Крестьянина возможные значения аргумента были заданы с помощью ***перечисления***. Область значений аргументов в команде Чертежника задается с помощью указания ***типа*** переменных, принимающих действительные (в информатике — вещественные) значения.

В о п р о с . Что будет нарисовано при исполнении алгоритма "отрезок", если поменять знаки всех аргументов на противоположные?

О т в е т . Получится отрезок, симметричный относительно начала координат, т.е. точки (0,0).

Затем можно предложить ученикам для тренировки составить алгоритм получения какого-нибудь простого рисунка, например, домика (рис. 9).

алг домик 1
  дано | перо поднято
  надо | нарисован домик (рис. 9)
нач
| сместиться в точку (1,3)
| опустить перо
| сместиться в точку (4,3)
| сместиться в точку (4,1)
| сместиться в точку (1,1)
| сместиться в точку (1,3)
| сместиться в точку (2,4)
| сместиться в точку (4,3)
| поднять перо
кон

(крыша в т.(2,4))

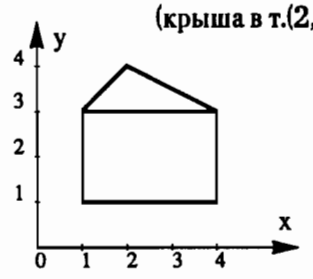


Рис. 9

Чтобы обосновать необходимость введения команды относительного перемещения пера, предложим учащимся составить алгоритм рисования точно такого же домика, но со смещением вправо по оси x (рис. 10).

Какие изменения следует внести в составленный алгоритм? Понятно, что придется в каждой команде с аргументами увеличить значение координаты x на единицу. Получается, что для рисования двух одинаковых фигур в разных местах координатной плоскости требуется писать два различных алгоритма. И все из-за того, что команда "сместиться в точку" привязана к началу координат. Чтобы избежать этого неудобства, в СКИ Чертежника введена команда:

сместиться на вектор (**арг вещ** a,b).

алг домик 2
  дано | перо поднято
  надо | нарисован домик (рис. 10)
нач
  сместиться в точку (2,3)
  опустить перо
  сместиться на вектор (3,0)
  сместиться на вектор (0,−2)
  сместиться на вектор (−3,0)
  сместиться на вектор (0,2)
  сместиться на вектор (1,1)
  сместиться на вектор (2,−1)
  поднять перо
  сместиться в точку (0,0)
кон

(крыша в т.(3,4))

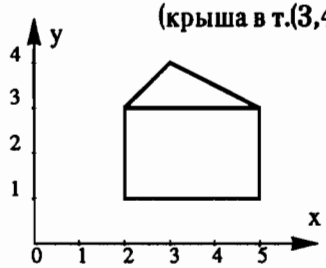


Рис. 10

Для смещения домика по координатной плоскости достаточно в алгоритме "домик 2" изменить аргументы первой команды.

Для объяснения особенности записи десятичных дробей в информатике напомним ученикам, что в нашем домике темно. Предложим им вырезать в передней стене квадратное окошко (рис. 11).

После рисования домика перо Чертежника находится в исходном положении. Чтобы попасть в левый нижний угол окна, необходимо сместиться вправо по x на 2 и вверх по y на 1.5. Если мы запишем команду "сместиться на вектор (2,1,5)", то она может быть понята по-разному (как известная фраза "казнить нельзя помиловать"). Чертежник может сместить перо на 2,1 по x и на 5 по y, и

(крыша в т.(2,4))

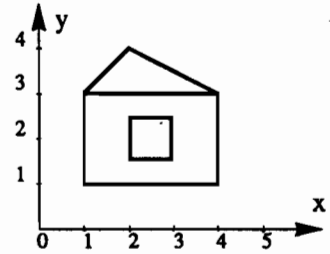


Рис. 11

окошко повиснет над домиком. Чтобы не было путаницы, договоримся всегда целую и дробную часть числа разделять точкой. Алгоритм рисования домика с окном будет таким:

алг домик с окном
  дано | перо поднято
  надо | нарисован домик с окном (рис. 11)
нач
  сместиться в точку (1,3)
  опустить перо
  сместиться на вектор (3,0)
  сместиться на вектор (0,−2)
  сместиться на вектор (−3,0)
  сместиться на вектор (0,2)
  сместиться на вектор (1,1)
  сместиться на вектор (2,−1)
  поднять перо
  сместиться в точку (0,0)
  сместиться на вектор (2,1.5)
  опустить перо
  сместиться на вектор (1,0)
  сместиться на вектор (0,1)
  сместиться на вектор (−1,0)
  сместиться на вектор (0,−1)
  поднять перо
  сместиться в точку (0,0)
кон

команды алгоритма домик 2

новые команды

С помощью Чертежника можно не только рисовать, но и писать. При этом проще строить буквы и цифры из отрезков прямых. Такая форма стала для нас привычной: она встречается на электронных часах, различных табло, почтовых конвертах и т.д.

З а д а н и е . Составьте алгоритм рисования буквы "А" (рис. 12).

алг буква А
  дано | перо в левом нижнем углу будущей буквы и поднято
  надо | нарисована буква А, перо в начале следующей буквы
  | (рис. 12)
нач
  опустить перо
  сместиться на вектор (0,3)
  сместиться на вектор (1,1)
  сместиться на вектор (1,0)
  сместиться на вектор (0,−4)
  сместиться на вектор (0,1)
  сместиться на вектор (−2,0)
  поднять перо
  сместиться на вектор (3,−1)
кон

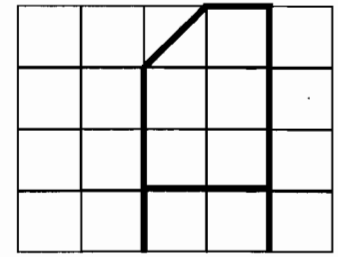


Рис. 12

Запишем в словарь:
арг — объявление величин-аргументов;
вещ — описание величины вещественного типа.

УКАЗАНИЯ К УПРАЖНЕНИЯМ

1. а) Запись 3,5,7 может быть прочитана двумя способами: 3.5,7 и 3, 5.7.

б) В записи, которую сделал Петя, нужно несколько запятых заменить точками. Точка может стоять только внутри числа, а числа разделяются запятой. Поэтому должно соблюдаться условие: между любыми двумя точками должна быть хотя бы одна запятая.

Если поставлено n точек, то в записи должно встретиться по крайней мере n−1 запятых. Всего точек и запятых 4 штуки, следовательно, n+n−1 ≤ 4, т. е. 2n ≤ 5. Значит, n может принимать значения 0, 1, 2.

При n=0 запись прочитывается одним способом: 7,3,5,0,1.

При n=1 имеются 4 способа: 7.3,5,0,1; 7,3.5,0,1; 7,3,5.0,1; 7,3,5,0.1.

При n=2 имеются 3 способа: 7.3,5.0,1; 7.3,5,0.1; 7,3.5,0.1.

Всего 8 способов.

3. Это упражнение подготавливает учащихся к использованию вспомогательных алгоритмов.

Составим серию команд для рисования окошка. Это можно сделать, например, так (мы считаем,что перо в левом нижнем углу домика):

```
поднять перо
сместиться на вектор (1,1)
опустить перо
сместиться на вектор ( 2, 0 )
сместиться на вектор ( 0, 2 )
сместиться на вектор (−2, 0 )
сместиться на вектор ( 0,−2 )
сместиться на вектор (0,1)
сместиться на вектор (2,0)
сместиться на вектор (−1,0)
сместиться на вектор (0,−1)
поднять перо
сместиться на вектор (−2,−1)
```

Затем эта серия команд приписывается в конец алгоритма "домик".

4. Задания упражнения 4 можно рассматривать в двух планах. Непосредственное выполнение заданий направлено на выработку навыков применения команд Чертежника. При таком

подходе можно ограничиться в случаях г) — е) рисованием какой-нибудь одной из требуемых фигур. Далее можно последовательно увеличивать сложность заданий, предлагая нарисовать несколько фигур, исследовать и описать множество возможных решений. Здесь возникают определенные математические трудности. В таком виде задания могут быть предложены сильным ученикам.

а) Наиболее простое решение получается, если составить алгоритм из трех команд:

```
сместиться в точку (1,2)
опустить перо
сместиться в точку (−1,1)
```

Можно рассмотреть решение той же задачи с использованием команды "сместиться на вектор". Для этого нужно на рисунке определить вектор смещения пера — (−2,−1). Полезно дать формулировку в терминах направлений: перо смещается на две единицы влево и на одну единицу вниз. Таким образом, в приведенной серии команд последняя команда может быть заменена командой "сместиться на вектор (−2,−1)".

б) Для рисования квадрата подходит фрагмент алгоритма "домик":

```
опустить перо
сместиться на вектор ( 4, 0 )
сместиться на вектор ( 0, 4 )
сместиться на вектор (−4, 0 )
сместиться на вектор ( 0,−4 )
поднять перо
```

в) Алгоритм получается из предыдущего заменой 4 на 6.

г) Наиболее простые решения получаются, если рисовать отрезок, параллельный одной из координатных осей с концом в точке (2,2), например:

```
сместиться в точку (2,2)
опустить перо
сместиться в точку (5,2)
```

Можно усложнить задание и предложить описать все отрезки, параллельные одной из осей и удовлетворяющие требованиям задачи. Такие отрезки имеют концы (a,2) и (a+3,2) или (2,a) и (2,a+3), где −1 ≤ a ≤ 2.

Разбор различных решений позволяет показать специфику команды смещения на вектор. Рассмотрим серию команд:

сместиться в точку (2,2)
опустить перо
сместиться на вектор (3,0)

В результате исполнения этой серии будет нарисован тот же отрезок, что и в результате исполнения серии команд, приведенной ранее. Если заменить первую команду командой вида

сместиться в точку (a,2)

где a — любое число, то в результате исполнения полученной серии будет нарисован отрезок с концами (a,2), (a+3,2). Если $-1 \le a \le 2$, то этот отрезок проходит через точку (2,2).

д) Нетрудно нарисовать требуемый квадрат со сторонами, параллельными осям координат:

сместиться в точку (1,1)
опустить перо
сместиться в точку (−1,1)
сместиться в точку (−1,−1)
сместиться в точку (1,−1)
сместиться в точку (1,1)

Применяя команду "сместиться на вектор", можно записать такую серию команд:

сместиться в точку (1,1)
опустить перо
сместиться на вектор (−2,0)
сместиться на вектор (0,−2)
сместиться на вектор (2,0)
сместиться на вектор (0,2)

При исполнении обеих серий перо будет обходить квадрат против часовой стрелки, начиная с вершины (1,1).

После составления последнего алгоритма можно предложить дополнительные вопросы.

Как изменить алгоритм, чтобы рисование квадрата начиналось с вершины (−1,1)? (Изменить первую команду и переставить третью команду в конец.)

Как изменить алгоритм, чтобы перо обходило квадрат по часовой стрелке? (Нужно записать команды смещения на вектор в обратном порядке и заменить сами векторы на противоположные, т.е. заменить на противоположные знаки аргументов).

Как изменить алгоритм, чтобы после его исполнения был нарисован квадрат с центром в точке (3,3)? (Первую команду заменить командой "сместиться в точку (4,4)".)

е) Задание близко к заданию в). Нетрудно нарисовать прямоугольники, удовлетворяющие условиям задачи. Например, имеются два прямоугольника со сторонами, параллельными осям координат, у которых правая верхняя вершина находится в точке (1,1), — "высокий" и "широкий". Эти прямоугольники будут нарисованы после исполнения следующих серий команд:

сместиться в точку (1,1)	сместиться в точку (1,1)
опустить перо	опустить перо
сместиться на вектор (−3,0)	сместиться на вектор (−4,0)
сместиться на вектор (0,−4)	сместиться на вектор (0,−3)
сместиться на вектор (3,0)	сместиться на вектор (4,0)
сместиться на вектор (0,4)	сместиться на вектор (0,3)

Можно поставить вопрос: каким условиям должны удовлетворять числа a и b, чтобы прямоугольник со сторонами, параллельными осям координат, и правой верхней вершиной в точке (a,b) содержал внутри себя начало координат? При $0<a<3$ и $0<b<4$ подойдет "высокий" прямоугольник, при $0<a<4$ и $0<b<3$ — "широкий".

Далее можно предложить серию заданий типа: как составить алгоритм для рисования прямоугольника с правой верхней вершиной в точке (a,b) при различных a и b. Требуемые алгоритмы имеют вид: первая команда — "сместиться в точку (a,b)", затем — после опускания пера — серия команд для рисования "высокого" или "широкого" прямоугольника в зависимости от того, каким условиям удовлетворяют числа a и b. Например, для точки (2,2) подходит любой из этих двух прямоугольников, для точки (3.8,1) — "широкий" и т.д.

ж) В качестве одного из возможных решений могут быть рассмотрены алгоритмы в) и г). Рассматривая общий случай, можно более основательно освоить команду смещения на вектор. С этой целью полезно разобрать алгоритм, записанный в общем виде:

сместиться в точку (a,b)
опустить перо
сместиться на вектор (x,y)
сместиться на вектор (u,v)
сместиться на вектор (−x,−y)
сместиться на вектор (−u,−v)

Здесь a, b, x, y, u, v — произвольные числа, удовлетворяющие лишь одному требованию: векторы (x,y) и (u,v) не должны быть коллинеарны (параллельны), т.е. $xv \ne yu$.

В заданиях г) — е) могут быть получены отрезки, квадраты и прямоугольники, наклоненные относительно осей. Нахождение

таких фигур связано с трудностями математического характера. Например, чтобы нарисовать отрезок длины 3, нужно сместиться на вектор (x,y), координаты которого удовлетворяют соотношению

$$x^2+y^2=3^2$$

и записываются конечными десятичными дробями. Такие уравнения рассматриваются в теории чисел. Очевидные решения x=0, y=3 или x=3, y=0 дают отрезки, параллельные осям. Не приводя общего анализа, укажем некоторые дополнительные решения.

г) Длину 3 имеют векторы с координатами (1.8,2.4), (0.74,2.98).

д) Требованиям задачи удовлетворяет квадрат с вершинами (0.2,1.4), (1.4,−0.2), (−0.2,−1.4), (−1.4,0.2).

е) Векторы (2.4,3.2), (2.4,−1.8) перпендикулярны и имеют длины соответственно 4 и 3. Используя эти векторы, можно указать, например, такую серию команд для рисования прямоугольника

сместиться в точку (−1,0)
опустить перо
сместиться на вектор (2.4,3.2)
сместиться на вектор (2.4,−1.8)
сместиться на вектор (−2.4,−3.2)
сместиться на вектор (−2.4,1.8)

5,8. Задания в этих упражнениях носят стандартный характер и могут быть использованы для закрепления навыков управления Чертежником. Упражнения подготавливают введение вспомогательных алгоритмов, поэтому в алгоритмах рисования букв и цифр следует использовать команды смещения на вектор. Полезно обратить внимание на случаи повторения серий команд для получения одинаковых фрагментов изображения.

Сделаем одно замечание к упражнению 5,г). Зеркальные отражения букв "И" и "Р" относительно горизонтальной оси — это буквы "N" и "Ь" (независимо от того, как проходит эта ось: выше, ниже букв или пересекает их). Симметрия относительно горизонтальной оси получается заменой знака второй координаты на противоположный.

6. В упражнении 6 трудность определяется в основном математическим содержанием задач.

а) Нужно обратить внимание на следующее: если при движении по замкнутой ломаной линии перо возвращается в исходную точку, то сумма всех векторов смещения равна нулевому вектору. В частности, для произвольного треугольника ABC имеем CA= =−(AB+BC). Таким образом, задав пару неколлинеарных векторов (x,y), (u,v) и точку (a,b), получаем серию команд для рисования треугольника:

сместиться в точку (a,b)
опустить перо
сместиться на вектор (x,y)
сместиться на вектор (u,v)
сместиться на вектор (−(x+u),−(y+v))

Чтобы одна из сторон была больше 1, достаточно, например, обеспечить условие $x^2 + y^2 > 1$.

б) Для простоты вначале можно рассмотреть алгоритм рисования конкретного треугольника с вершинами A(0,0), B(0,2), C(2,0). Середины сторон этого треугольника — точки M(1,1), N(1,0), P(0,1). Получаем алгоритм:

```
алг нарисовать треугольник с тремя медианами
  дано | перо поднято
  надо | нарисован треугольник с тремя медианами,
       | перо поднято и находится
       | в начальном положении
нач
| опустить перо
| сместиться в точку (0,2)  | AB
| сместиться в точку (2,0)  | BC
| сместиться в точку (0,0)  | CA
| сместиться в точку (1,1)  | AM
| сместиться в точку (2,0)  | MC
| сместиться в точку (0,1)  | CP
| сместиться в точку (0,2)  | PB
| сместиться в точку (1,0)  | BN
| сместиться в точку (0,0)  | NA
| поднять перо
кон
```

В комментариях записаны обозначения рисуемых отрезков.

Теперь можно рассмотреть общий случай. Пусть произвольным образом заданы точки A, B, C, не лежащие на одной прямой. Отрезки в комментариях по существу определяют алгоритм: достаточно соответствующим образом изменить параметры в командах перемещения. Координаты точек M,N,P вычисляются по координатам вершин. Например, при заданных A(x,y) и B(u,v) точка P (середина AB) имеет координаты ((x+u)/2,(y+v)/2).

в) В качестве одного из простых решений можно составить алгоритм рисования треугольника с вершинами A(0,0), B(1,1), C(1,−1). Рассмотрим более общую ситуацию. Векторы с координатами (x,y) и (u,v) перпендикулярны, если их скалярное произведение равно нулю, т.е. xu+yv=0. Выбрав произвольную пару ненулевых перпендикулярных векторов, не параллельных осям коор-

динат, можно построить прямоугольный треугольник так, как это сделано в упражнении 6,а).

г) Требуемое множество точек — квадрат с вершинами (1,0), (0,1), (−1,0), (0,−1).

7. При выполнении заданий б) и в) можно наряду с указанными там графиками рассмотреть график функции y=x−2. Составление алгоритмов для рисования пар графиков

$$y=x, \quad y=x-2 \quad \text{и} \quad y=|x|, \quad y=|x|-2$$

позволяет еще раз подвести учащихся к идее вспомогательного алгоритма. Алгоритм вычерчивания графика функции y=f(x)+a получается из алгоритма вычерчивания графика функции y=f(x), если во всех командах смещения в точку ко второй координате добавить a. Например, в результате исполнения серии команд

сместиться в точку (−4,4)
опустить перо
сместиться в точку (0,0)
сместиться в точку (4,4)

нарисован график функции y=|x| на отрезке −4,4. Если теперь добавить к ординатам всех точек −2, то получится серия команд для рисования графика y=|x|−2:

сместиться в точку (−4,2)
опустить перо
сместиться в точку (0,−2)
сместиться в точку (4,2)

Графики функций из упражнения 7 изображены на рисунке 13. Построение графиков удобно производить по порядку: каждый следующий получается из предыдущего смещением на вектор (0,−2) или зеркальным отражением относительно оси абсцисс части графика, лежащей под осью.

В качестве дополнительного упражнения можно рассмотреть построение еще нескольких графиков подобного рода. Например, f(x)=||x|−2|−1, f(x)=|||x|−2|−1|.

Рассмотрим построение последнего графика. Соответствующая ломаная состоит из четырех V-образных элементов. Таким образом, нужный график будет нарисован после исполнения серии команд

сместиться в точку (−4,1)
опустить перо
сместиться на вектор (1,−1) | рисование V-образного элемента
сместиться на вектор (1,1) | (повторить 4 раза)

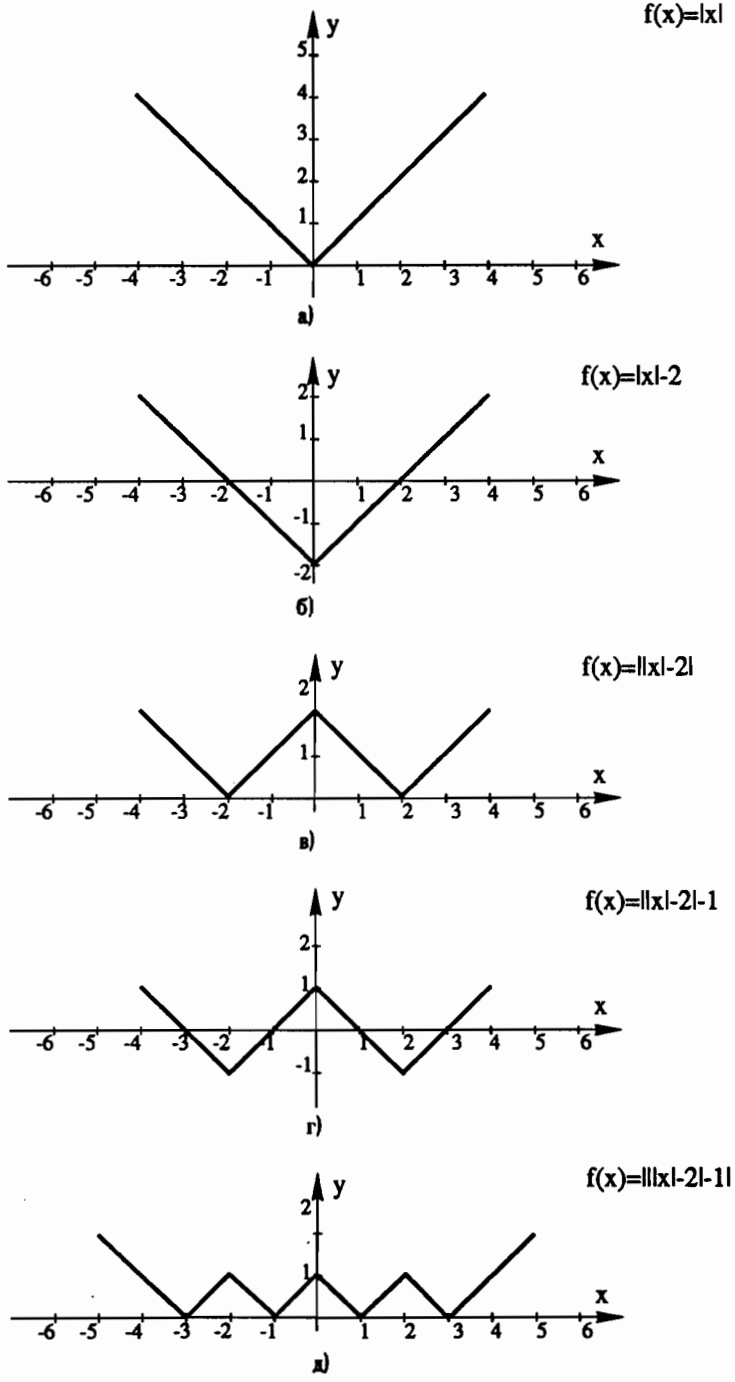


Рис. 13

9. Приведем сначала одно общее рассуждение. Рассмотрим произвольную ломаную, которую можно нарисовать, не отрывая перо от бумаги и не проводя ни одну линию дважды (такие линии называют уникурсальными). Пусть A — произвольная вершина этой ломаной, в которой сходятся три линии (рис. 14).

Рис. 14

Тогда если рисование линии начиналось не с вершины A, то в ходе рисования перо пришло в A (по линии 1), затем вышло из A (по линии 2) и снова вернулось в A (по линии 3). Таким образом, если рисование начиналось не с вершины A, то оно должно в ней закончиться. Аналогичное рассуждение справедливо для любой вершины, в которой сходится нечетное число линий (про такие вершины говорят, что они имеют нечетный порядок). Следовательно, если на ломаной имеются две вершины нечетного порядка, то рисовать ее нужно, начиная с одной из них, тогда рисование закончится в другой. Заметим, что на уникурсальной линии не может быть более двух вершин нечетного порядка.

а) Возможный путь обхода первой фигуры — ABCDBEDAE (рис. 15,а).

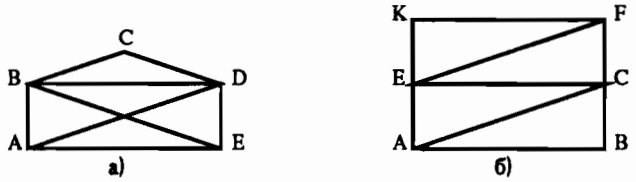


Рис. 15

Для составления алгоритма теперь достаточно найти координаты соответствующих векторов.Пусть вектор AB имеет координаты (0,y), а вектор AE — координаты (x,0). Тогда AD=AE+AB и AD имеет координаты (x,y). Далее, BE=AE−AB, так что координаты вектора BE равны (x,−y). Заметим, наконец, что BC= =(1/2)AD, CD=(1/2)BE. Положим x=6, y=4. Тогда Чертежник нарисует нужную фигуру после исполнения следующих команд:

опустить перо	
сместиться на вектор (0,4)	AB
сместиться на вектор (3,2)	BC
сместиться на вектор (3,−2)	CD
сместиться на вектор (−3,0)	DB
сместиться на вектор (3,−2)	BE

сместиться на вектор (0,2)	ED
сместиться на вектор (−3,−2)	DA
сместиться на вектор (3,0)	AE

б) При безмашинном варианте эту задачу лучше опустить. На компьютере задача решается подбором подходящих смещений.

в) Возможный путь обхода (рис. 15, б) — ABCAECFKEF.

10. а) В алгоритме рисования букв (см. п. 5.5) достаточно заменить последнюю команду командой смещения на вектор (2,0).

б) В командах алгоритмов рисования букв нужно увеличить в два раза все аргументы.

в), г) В алгоритмах рисования букв нужно изменить последнюю команду: в результате ее исполнения перо должно сместиться влево на ширину буквы и вниз на высоту буквы плюс расстояние между строками. Таким образом, в алгоритмах рисования букв (см. п. 5.5) последняя команда заменяется командой

сместиться на вектор (−2,−5).

Для букв удвоенного размера все аргументы увеличиваются в два раза.

11. Фигура, которая получается в результате исполнения алгоритма —лесенка из трех ступенек (рис. 16).

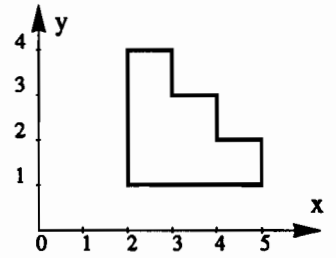


Рис. 16

Для ответа на вопросы из задания а) заметим, что две последние команды в алгоритме — "сместиться в точку (2,1)" и "поднять перо". Таким образом, после исполнения алгоритма перо будет поднято и расположено в точке (2,1).

Чтобы алгоритм рисовал вдвое большую фигуру, нужно увеличить вдвое все векторы в командах "сместиться на вектор". Для выполнения задания г) нужно вначале установить, как связаны координаты симметричных точек и векторов (рис. 17).

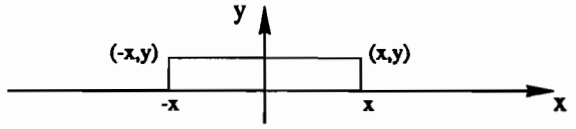


Рис. 17

После этого можно сделать общий вывод: если в командах алгоритма изменить у всех аргументов знаки первой координаты на противоположные, то при исполнении полученного алгорит-

ма будет нарисована фигура, симметричная первоначальной относительно вертикальной оси.

В пункте д) при изменении знаков обеих координат аргументов на противоположные получится фигура, симметричная исходной относительно начала координат.

12. Задание фиксирует внимание на связи операции сложения векторов с командами Чертежника, поэтому здесь имеет смысл рассмотреть эту связь более подробно. Нужно показать, что последовательное выполнение команд

сместиться на вектор (x,y)
сместиться на вектор (u,v)

приводит к такому же перемещению, как и команда

сместиться на вектор (x+u,y+v)

Отсюда можно вывести обобщение: в результате выполнения нескольких команд "сместиться на вектор" перо сместится на вектор, равный сумме всех векторов из указанных команд. Таким образом, в результате исполнения всех команд заданного алгоритма перо сместится на вектор (7,0). Так как перед исполнением алгоритма перо находилось в начале координат, то после исполнения алгоритма оно окажется в точке (7,0).

13, 14. Упражнения носят творческий характер. Автоматическое устройство (исполнитель) задается системой команд: необходимо указать названия команд и соответствующие им элементарные действия.

§6. ВСПОМОГАТЕЛЬНЫЕ АЛГОРИТМЫ. АЛГОРИТМЫ С АРГУМЕНТАМИ (4 ч)

Не в совокупности ищи единства, но более — в единообразии разделения.

Козьма Прутков

В § 6 вводится понятие вспомогательного алгоритма и рассматриваются методы построения алгоритмов с использованием вспомогательных алгоритмов. Материал параграфа лежит в основе общего подхода к построению алгоритмов методом последовательного уточнения.

Во вспомогательных алгоритмах с аргументами существенно используются величины — понятие, рассматриваемое в § 11. Одновременно с материалом § 6 будет целесообразно изучить пп. 11.1—11.5.

Основные цели. Познакомить учащихся с понятием вспомогательного алгоритма, дать навыки использования готовых алгоритмов в качестве вспомогательных, с построением алгоритмов методом последовательного уточнения. Ввести основные понятия, связанные с величинами в алгоритмическом языке. Познакомить учащихся с правилами записи и исполнения вспомогательных алгоритмов с аргументами.

Требования к знаниям и умениям. Учащиеся должны уметь исполнять, составлять и записывать несложные алгоритмы с использованием вспомогательных алгоритмов; уметь сводить составление алгоритмов решения несложных задач к составлению вспомогательных алгоритмов и конструированию из них основного алгоритма; владеть понятиями имени, значения и типа величины.

ОСНОВНЫЕ ПОНЯТИЯ

Ключевое понятие параграфа — ***вспомогательный алгоритм.*** Это не какой-то особый вид алгоритма, термин "вспомогательный" относится только к *использованию* алгоритма. Это справедливо и для термина "основной алгоритм". Вспомогательными и основными алгоритмы являются не сами по себе, а *по отношению друг к другу.* Эта мысль прослеживается в пп. 6.2 и 6.3 учебника.

Вызов вспомогательного алгоритма записывается очень просто. В основной алгоритм в качестве команды включается имя вспомогательного алгоритма. Таким образом, *по форме вызов вспомогательного алгоритма ничем не отличается от вызова команды исполнителя.*

Употребление общего термина "вызов" для команд исполнителя (п. 4.6) и вспомогательных алгоритмов (п. 6.2) не случайно. Работу ЭВМ можно упрощенно представить так. В памяти машины есть таблица, в которой хранятся имена команд всех исполнителей, подключенных к ЭВМ. Если в память ЭВМ записывается новый алгоритм, его имя заносится *в ту же таблицу* с пометкой "алгоритм" (табл. 6).

При исполнении алгоритма ЭВМ анализирует поочередно каждую команду. Анализ заключается в поиске соответствующего имени в таблице. Когда имя найдено, ЭВМ проверяет, является ли оно именем команды или алгоритма. Если это команда, она передается соответствующему исполнителю. Если же это вспомогательный алгоритм, выполняется этот алгоритм. Если имени в таблице нет, ЭВМ сообщает о *синтаксической ошибке.*

Исполнение вспомогательного алгоритма происходит так. Встретив команду вызова, ЭВМ запоминает место этой команды, находит в памяти вспомогательный алгоритм (место хранения самого алгоритма обычно указывается в таблице имен) и выполняет его. Закончив исполнение, ЭВМ вспоминает место команды вызова и переходит к исполнению следующей за ней команды.

Таблица 6

буква И	алгоритм
буква М	алгоритм
буква Р	алгоритм
вверх	команда Робота
влево	команда Робота
вниз	команда Робота
вправо	команда Робота
закрасить	команда Робота
МИР	алгоритм
опустить перо	команда Чертежника
поднять перо	команда Чертежника
сместиться в точку	команда Чертежника
сместиться на вектор	команда Чертежника
ход конем	алгоритм

Вспомогательный алгоритм может вызывать другие вспомогательные алгоритмы, длина такой цепочки вызовов теоретически не ограничена.

Применение вспомогательных алгоритмов позволяет разбить задачу на части, структурировать ее.

Важную роль в формировании алгоритмического мышления играет построение алгоритмов с использованием вспомогательных.

Можно выделить две стратегии такого построения, соответствующие двум технологиям программирования и двум компонентам алгоритмического мышления (табл. 7).

Таблица 7

Технология программирования	Построение алгоритмов	Алгоритмическое мышление
Снизу вверх	Использование готовых вспомогательных алгоритмов	Сведение нерешенной задачи к уже решенным
Сверху вниз	Проектирование вспомогательных алгоритмов	Разбиение большой задачи на малые

При технологии "снизу вверх" используются ранее составленные вспомогательные алгоритмы (пп. 6.1, 6.3 учебника). Для этого задача (основной алгоритм) разбивается на такие подзадачи, алгоритм решения которых уже известен, т.е. нерешенная задача сводится к решенным. При этом совершенно не обязательно знать полный текст вспомогательного алгоритма, достаточно знать его заголовок. Обеспечив условие дано и правильно обратившись к алгоритму, получаем результат, описанный в надо. В этом смысле механизм выполнения вспомогательного алгоритма можно считать несущественным, как и механизм выполнения простой команды.

При технологии "сверху вниз" задача разбивается на подзадачи. Затем эти подзадачи поочередно решаются (возможно, с использованием дальнейших разбиений). Решение всех подзадач автоматически приводит к решению главной задачи (п. 6.4 учебника). При составлении алгоритма это означает, что можно записать вызов еще не написанного вспомогательного алгоритма. Записывая такой вызов, необходимо определить имя алгоритма и требования к нему (условия дано и надо), т.е. заголовок алгоритма. В дальнейшем этот заголовок можно рассматривать как задание на разработку алгоритма. Заметим, что и в этом случае основной алгоритм пишется без знания текста вспомогательного.

Важное понятие параграфа — ***алгоритмы с аргументами***. Полное понимание механизма аргументов возможно только после изучения понятия величины (§ 11). Алгоритмы с аргументами аналогичны командам с аргументами (типа "сместиться в точку"). В обоих случаях кроме действия (алгоритма) задается некоторая дополнительная информация (аргументы), уточняющая эти действия.

ОБЩИЕ УКАЗАНИЯ

Значение понятия вспомогательного алгоритма выходит за рамки курса информатики. По существу, умение выделять вспомогательные алгоритмы или, в более общем контексте, выделять в задаче различные подзадачи является одним из фундаментальных.

Курс информатики вносит свой вклад в формирование этих общих умений, вводя точные определения и явные описания. Структуризация задачи — одна из основных линий курса информатики. Выработка соответствующих умений и стиля мышления — одна из конечных целей курса. На данном этапе важно накопить эмпирический материал и овладеть необходимой техникой алгоритмизации.

На примерах из пп. 6.1 — 6.5 учащимся нужно показать, что свойство быть вспомогательным алгоритмом определяется ро-

лью алгоритма в решении задачи, а не его собственными качествами, так же, например, как свойство быть слагаемым определяется ролью числа в выражении, а не его величиной.

С потребностью введения вспомогательных алгоритмов учащиеся впервые сталкиваются при изучении Чертежника в пп. 6.1, 6.3. Набор вспомогательных алгоритмов рисования букв применяется для решения серии однотипных задач рисования слов. Вспомогательные алгоритмы служат деталями, из которых могут конструироваться основные алгоритмы.

В курсе информатики учащиеся впервые встречаются с систематическим применением метода последовательного уточнения. В п. 6.4 алгоритм строится методом последовательного уточнения, а в п. 6.5 дается краткая характеристика этого метода.

Применение метода последовательного уточнения даже в несложных задачах содержит элементы исследовательской деятельности. Этот метод — важная компонента алгоритмической культуры, овладение им имеет существенное общеобразовательное значение. Использование вспомогательных алгоритмов позволяет учителю начать целенаправленную работу в этом направлении. В методическом плане эта работа связана в первую очередь с решением и разбором разнообразных задач.

Рассмотрим алгоритмы:

```
алг курица
нач
| яйцо
кон
```

```
алг яйцо
нач
| курица
кон
```

Здесь мы имеем пару алгоритмов, каждый из которых использует другой в качестве вспомогательного, и в результате ни один из них не может выступать в качестве основного. В п. 16.11 ("Рекурсия") показано, как осмысление и преодоление подобного рода парадоксов приводит к важному методу разработки алгоритмов.

Чтобы прояснить специфику использования вспомогательных алгоритмов с аргументами, имеет смысл ввести понятия имени, значения и типа величины.

Встретив в заголовке вспомогательного алгоритма слова "арг вещ а", ЭВМ отводит в памяти место для хранения вещественной величины а. При вызове вспомогательного алгоритма указывается конкретное значение, которое записывается в отведенное место. При выполнении вспомогательного алгоритма вместо величины а ЭВМ подставляет ее значение.

Исполнители принимают команды лишь с конкретными значениями аргументов: Чертежник не может исполнить команду "сместиться на вектор (а,0)". Формирование команд исполнителя с конкретными значениями берет на себя ЭВМ.

Основной текст параграфа не содержит примеров использования вспомогательных алгоритмов с аргументами, поэтому при объяснении нового материала нужно подобрать соответствующие примеры. Этой цели могут служить алгоритмы из упражнения 18.

РЕКОМЕНДАЦИИ К ВЕДЕНИЮ УРОКОВ

Составим алгоритм, по которому Чертежник напишет слово "МАМА". Вспомним, что уже составлены алгоритмы рисования буквы "М" (с. 35 учебника) и буквы "А" (с. 55 настоящего пособия). Алгоритмический язык позволяет в новых алгоритмах использовать ранее составленные:

```
алг слово мама
  дано | перо в исходном положении
  надо | написано слово "МАМА", перо
       | в начале следующей буквы
нач
| буква М
| буква А
| буква М
| буква А
кон
```

Рассмотрим работу ЭВМ, которой дана команда исполнить алгоритм "слово мама":

1. Находит алгоритм "слово мама" в памяти.
2. Проверяет условие дано (там комментарий).
3. Находит нач.
4. Фразу "буква М" распознает как *вызов вспомогательного алгоритма.*
5. Запоминает место вызова в алгоритме "слово мама" и переходит к исполнению вспомогательного алгоритма.
   - 5.1. Находит алгоритм "буква М" в памяти.
   - 5.2. Проверяет условие дано (там комментарий).
   - 5.3. Находит нач.
   - 5.4. Встречает команду "опустить перо" и передает ее Чертежнику.
   - 5.5 — 5.10. Передает Чертежнику следующие шесть команд.
   - 5.11. Встречает слово кон.
   - 5.12. Проверяет условие надо (там комментарий).
   - 5.13. Возвращается в алгоритм "слово мама" на запомненное место.
6. Встречает фразу "буква А".

7. Запоминает место вызова в алгоритме "слово мама" и переходит к исполнению вспомогательного алгоритма.
   7.1. Находит алгоритм "буква А" в памяти.
   7.2. Проверяет условие **дано** и т.д.

Ту же задачу можно решить с помощью более сложной структуры вспомогательных алгоритмов:

```
алг новое слово мама
  дано | перо в исходном положении
  надо | написано слово "МАМА", перо
       | в начале следующей буквы
нач
| слог МА
| слог МА
кон

алг слог МА
  дано | перо в начале первой буквы
  надо | написано "МА", перо
       | в начале следующей буквы
нач
| буква М
| буква А
кон
```

При таком решении в алгоритме "новое слово мама" дважды вызывается вспомогательный алгоритм "слог МА", который, в свою очередь, каждый раз вызывает вспомогательные алгоритмы "буква М" и "буква А". Получается *иерархия алгоритмов*:

При использовании вспомогательных алгоритмов *нас не интересует, каким образом устроен вспомогательный алгоритм.* Важно лишь, чтобы он получил правильные исходные данные и выдал требуемый результат.

Пусть составлены алгоритмы с такими заголовками:

```
алг буква М
  дано | перо в начале буквы
  надо | написана буква "М", перо в начале следующей буквы

алг буква И
  дано | перо в начале буквы
  надо | написана буква "И", перо в начале следующей буквы

алг буква Р
  дано | перо в начале буквы
  надо | написана буква "Р", перо в начале следующей буквы

алг буква У
  дано | перо в начале буквы
  надо | написана буква "У", перо в начале следующей буквы
```

Не зная, что именно заключено в этих алгоритмах между **нач** и **кон**, мы можем их использовать для составления алгоритмов рисования различных слов и фраз: РАМА, МУРА, АМУР, МУАР, АУРУМ, МИРУ МИР, РИМУ РИМ, УМРИ МИРРА и т.д.

Таким образом, готовые алгоритмы можно использовать при составлении более сложных алгоритмов. Часто используемые алгоритмы записывают в память ЭВМ, и полученную совокупность называют *библиотекой алгоритмов.*

Понятие вспомогательного алгоритма известно человеку и ЭВМ. Оно не касается исполнителя. Независимо от сложности иерархической структуры вспомогательных алгоритмов, исполнитель получает команды из своей СКИ и ему нет дела, откуда и как эти команды взялись.

При изучении вспомогательных алгоритмов в классе с хорошей логико-математической подготовкой можно рассмотреть сложный пример использования вспомогательного алгоритма при управлении Роботом:

```
алг путешествие
  дано | стен на поле нет
  надо | ?
нач
| влево
| путешествие
кон
```

Необычность этого алгоритма в том, что он использует в качестве вспомогательного самого себя. Это так называемый *рекурсивный* алгоритм. Что будет происходить в результате его исполнения? На этот вопрос существует два ответа — абстрактный и реальный.

Абстрактный ответ: в результате исполнения алгоритма "путешествие" Робот будет бесконечно двигаться влево (его поле бесконечно).

Реальный ответ: вспомним, как действует ЭВМ, встретив вызов вспомогательного алгоритма. Она запоминает место вызова, чтобы после выполнения вспомогательного алгоритма вернуться в основной. В алгоритме "путешествие" ЭВМ встречает вызов "путешествие", запоминает место вызова и приступает к повторному выполнению того же самого алгоритма, но уже в качестве вспомогательного. При этом она вновь встречает вызов "путешествие", запоминает второе место вызова и т.д.

В конце концов вся часть памяти, отведенная для запоминания мест вызова, окажется заполненной и ЭВМ выдаст сообщение об отказе.

Итак, Робот будет двигаться влево, пока не переполнится часть памяти ЭВМ, отведенная для запоминания мест вызова.

Особое внимание при изучении рассматриваемой темы следует обратить на ***метод последовательного уточнения*** (пошаговой декомпозиции, проектирования "сверху-вниз") как на общий способ рассуждений при разработке сложных систем (не только в программировании).

При этом можно начать с простого и близкого школьникам примера. Представим себе, что существует школьник Петя, который каждое утро планирует свой день. В его записной книжке мы встретим, например, такую запись (приводится фрагмент):

СРЕДА:
- 1. Завтрак
- 2. Уроки в школе
- 3. Обед
- 4. Секция
- 5. Домашнее задание на четверг (если успею)
- 6. В кино с Колей

ЗАВТРАК:
- 1. Поджарить яичницу
- 2. Сварить кофе
- 3. Принять указанную пищу

УРОКИ В ШКОЛЕ:
- 1. Химия
- 2. Математика
- 3. Литература
- 4. Физика
- 5. Биология
- 6. Английский
  - ХИМИЯ: списать математику
  - ЛИТЕРАТУРА: списать физику
  - БИОЛОГИЯ: списать английский...

Как же рассуждает Петя, планируя свой день? Очень просто: он разбивает день на логически завершенные части, затем каждую часть делит на более мелкие, затем — на еще более мелкие и т.д. То есть Петя ***последовательно уточняет свои действия.***

Попробуем рассуждать тем же способом при составлении алгоритма. Решим такую простую задачу: с помощью Робота закрасить в виде шахматной доски часть поля размером 8×8 клеток (рис. 18).

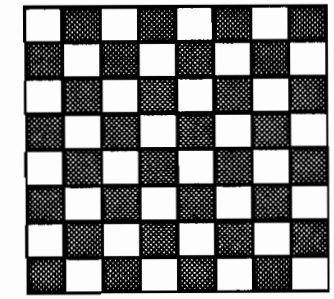


Рис.18

Разобьем нашу задачу на части. Выделим на поле четыре одинаковых полосы шириной две клетки (рис. 19).

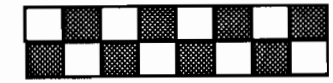


Рис. 19

Тогда основной алгоритм может быть таким:

```
алг шахматная доска
  дано | Робот в левой верхней клетке
  надо | поле закрашено в виде
       | шахматной доски
нач
| закрасить полосу
| вернуться
| закрасить полосу
| вернуться
| закрасить полосу
| вернуться
| закрасить полосу
кон
```

В свою очередь, задачу закрашивания полосы тоже разделим на части, выделив четыре одинаковых квадрата 2×2 клетки:

алг закрасить полосу
дано | Робот в левой верхней клетке полосы
надо | полоса закрашена в шахматном порядке
нач
| закрасить квадрат
| закрасить квадрат
| закрасить квадрат
| закрасить квадрат
кон

Теперь запишем алгоритмы "закрасить квадрат" и "вернуться":

алг закрасить квадрат
дано | Робот в левой верхней клетке квадрата
надо | закрашен квадрат, Робот в начале следующего
| квадрата
нач
| вниз
| закрасить
| вправо
| вверх
| закрасить
| вправо
кон

алг вернуться
дано | Робот правее полосы
надо | Робот в начале следующей полосы
нач
| вниз; вниз
| влево; влево
| влево; влево
| влево; влево
| влево; влево
кон

Вопрос: Что произойдет, если "шахматную доску" оградить стенами?

Ответ: Возникает отказ при выполнении последней команды алгоритма "закрасить квадрат" при его последнем вызове в алгоритме "закрасить полосу".

Применим метод последовательного уточнения для другой задачи. Пусть требуется изобразить домик на краю елового леса (рис. 20).

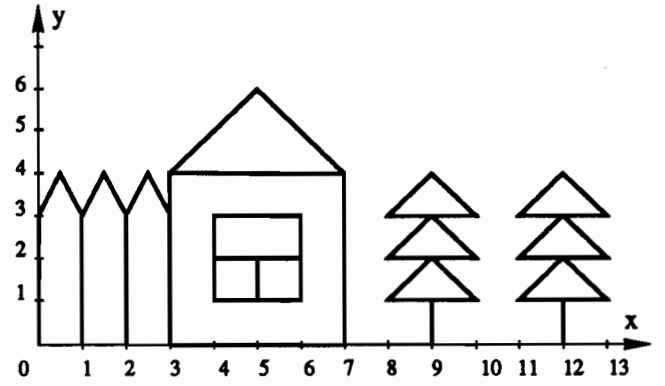


Рис. 20

алг домик возле леса
дано | перо в исходном положении
надо | сделан рисунок (рис. 20)
нач
| нарисовать забор
| сместиться в точку (3,0)
| домик с окошком
| сместиться в точку (9,0)
| нарисовать елку
| сместиться в точку (12,0)
| нарисовать елку
кон

Алгоритм "домик с окошком" был составлен при выполнении упражнения 3 на с. 37 учебника. Остается написать алгоритмы "нарисовать забор" и "нарисовать елку".

Для экономии времени урока класс можно разделить на бригады, каждая из которых разрабатывает один из вспомогательных алгоритмов.

алг нарисовать забор
дано | перо в исходном положении
надо | сделан рисунок забора
нач
| нарисовать доску
| нарисовать доску
| нарисовать доску
кон

```
алг нарисовать доску
  дано | перо в начале доски
  надо | сделан рисунок доски, перо в начале
       | следующей доски
нач
| опустить перо
| сместиться на вектор (0,3)
| сместиться на вектор (0.5,1)
| сместиться на вектор (0.5,−1)
| сместиться на вектор (0,−3)
| поднять перо
кон

алг нарисовать елку
  дано | перо в начале елки
  надо | сделан рисунок елки
нач
| опустить перо
| сместиться на вектор (0,1)
| нарисовать треугольник
| нарисовать треугольник
| нарисовать треугольник
| поднять перо
кон

алг нарисовать треугольник
  дано | перо в начале треугольника
  надо | сделан рисунок треугольника, перо в начале
       | следующего треугольника
нач
| опустить перо
| сместиться на вектор (−1,0)
| сместиться на вектор (1,1)
| сместиться на вектор (1,−1)
| сместиться на вектор (−1,0)
| поднять перо
| сместиться на вектор (0,1)
кон
```

Важнейшим понятием темы являются алгоритмы с аргументами. Начать изучение этого понятия с учащимися можно с такой задачи.

Пусть с помощью Чертежника необходимо нарисовать узор, изображенный на рисунке 21.

Будем решать эту задачу методом последовательного уточнения. Видно, что узор имеет четыре одинаковых фрагмента:

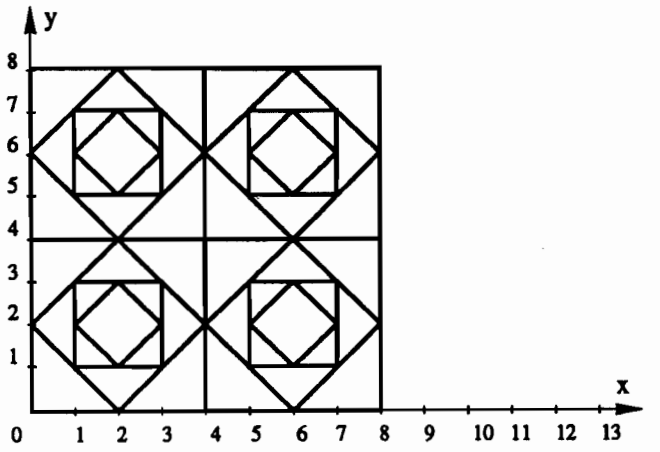


Рис. 21

```
алг нарисовать узор
  дано | перо в исходном положении
  надо | нарисован узор (рис. 21)
нач
| нарисовать фрагмент
| сместиться в точку (4,0)
| нарисовать фрагмент
| сместиться в точку (0,4)
| нарисовать фрагмент
| сместиться в точку (4,4)
| нарисовать фрагмент
кон
```

Каждый фрагмент состоит из квадратов двух видов: со сторонами, параллельными осям (1 и 3) и повернутыми на 45 ° (2 и 4), рисунок 22.

```
алг фрагмент
  дано | перо в левом нижнем углу
  надо | нарисован фрагмент (рис. 22)
нач
| квадрат 1
| сместиться на вектор (2,0)
| квадрат 2
| сместиться на вектор (−1,−1)
| квадрат 3
| сместиться на вектор (1,0)
| квадрат 4
кон
```

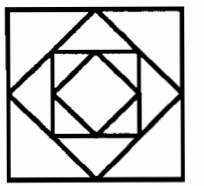


Рис. 22

Алгоритм рисования квадрата 1 был получен при решении упражнения 4,б) на с. 37 учебника. Чем будет отличаться от него алгоритм рисования квадрата 3? Только значением величины смещения по осям: вместо числа 4 надо везде подставить 2. Такую работу нельзя отнести к творчеству, а следовательно, выполнение этих рутинных операций желательно поручить ЭВМ.

Обозначив величину смещения буквой "а", мы придем к алгоритму рисования произвольного квадрата со сторонами, параллельными осям (А16, п. 6.7 учебника). По аналогии легко составить алгоритм рисования произвольного квадрата второго вида (квадраты 2 и 4):

```
алг повернутый квадрат (арг вещ в)
  дано | перо в нижнем углу
  надо | нарисован квадрат
       | перо в нижнем углу
нач
| опустить перо
| сместиться на вектор (в,в)
| сместиться на вектор (–в,в)
| сместиться на вектор (–в,–в)
| сместиться на вектор (в,–в)
| поднять перо
кон
```

Теперь легко записать алгоритм рисования фрагмента:

```
алг нарисовать фрагмент
  дано | перо в левом нижнем углу
  надо | нарисован фрагмент (рис. 22)
нач
| квадрат (4)
| сместиться на вектор (2,0)
| повернутый квадрат (2)
| сместиться на вектор (–1,–1)
| квадрат (2)
| сместиться на вектор (1,0)
| повернутый квадрат (1)
кон
```

Поскольку учащиеся впервые встретились с алгоритмами, имеющими аргументы, необходимо подробно рассмотреть, что делает ЭВМ при исполнении алгоритма "нарисовать узор".

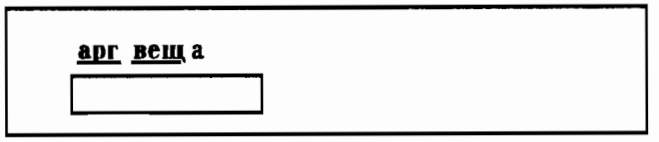
1. Находит в памяти алгоритм "нарисовать узор".
2. После **дано** – комментарий.
3. Находит **нач**.

4. Встречает вызов "нарисовать фрагмент".
5. Запоминает место вызова.
   - 5.1. Находит в памяти алгоритм "нарисовать фрагмент".
   - 5.2. После **дано** – комментарий.
   - 5.3. Находит **нач**.
   - 5.4. Встречает вызов "квадрат (4)".
   - 5.5. Запоминает место вызова.
     - 5.5.1. Находит алгоритм "квадрат (**арг вещ** а)".
     - 5.5.2. Выделяет в памяти место для записи значений переменных алгоритма "квадрат":

алг квадрат

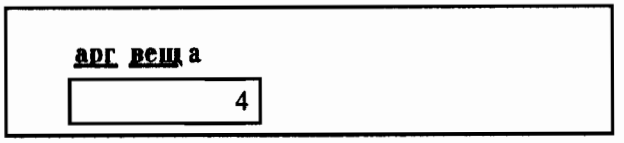
     - 5.5.3. Встречает *описание величины* "**арг вещ** а" и выделяет для нее ячейку:

алг квадрат



     - 5.5.4. Заносит в ячейку значение 4, заданное при вызове "квадрат (4)":

алг квадрат



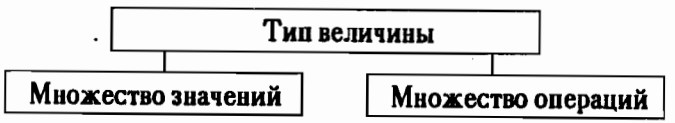
     - 5.5.5. После **дано** – комментарий.
     - 5.5.6. Находит **нач**.
     - 5.5.7. Дает команду Чертежнику "опустить перо".
     - 5.5.8. В команде "сместиться на вектор (а,0)" вместо имени величины подставляет ее значение и дает Чертежнику команду "сместиться на вектор (4,0)".
     - 5.5.9 – 5.5.11. Аналогично преобразуя следующие три команды, передает их Чертежнику.

5.5.12. Дает команду "поднять перо".
5.5.13. Встречает слово **кон**.
5.5.14. После **надо** — комментарий.
5.5.15. Стирает информацию в части памяти, выделенной для алгоритма "квадрат"
5.5.16. Возвращается в алгоритм "нарисовать фрагмент" на запомненное место.
5.6. Дает команду "сместиться на вектор (2,0)".
5.7. Встречает вызов "повернутый квадрат (2)".
5.8. Запоминает место вызова.
5.8.1. Затем находит алгоритм "повернутый квадрат (**арг вещ** а)".
5.8.2. Выделяет в памяти место для записи значений переменных алгоритма "повернутый квадрат".
... и т. д.

Для закрепления знаний и проверки усвоения материала можно предложить учащимся поочередно продолжить описание действий ЭВМ при исполнении алгоритма "нарисовать узор".

Обобщим сведения о величине, полученные при составлении и анализе алгоритма "нарисовать узор". Во вспомогательном алгоритме "квадрат (**арг вещ** а)" буквой "а" обозначены сразу три понятия — имя аргумента, его текущее значение и область памяти, в которой это значение хранится:

Перед именем величины в заголовке алгоритма стоят два служебных слова: **арг** и **вещ**. С их помощью *указаны вид и тип величины*. В данном случае "а" — это входная величина (**арг**), значением которой может быть любое вещественное число (**вещ**). Тип величины определяет, какие значения она может принимать и какие операции над этой величиной разрешены:



В алгоритмическом языке имеется несколько типов величин. Есть, например, величины, принимающие только целочисленные значения (типа **цел**).

Таким образом, понятие величины в информатике достаточно сложное, емкое и разностороннее. Иллюстрацией этого понятия может служить несколько сокращенный отрывок из "Алисы в Зазеркалье" Л.Кэрролла (когда трудно придумать пример для объяснения какого-нибудь понятия, обращаются к произведениям Л.Кэрролла):

"— Ты загрустила? — огорчился Рыцарь. — Давай я спою тебе в утешение песенку. *Заглавие* этой песенки называется "Пуговки для сюртуков".

— Вы хотите сказать — *песня* так называется? — спросила Алиса.

— Нет, ты не понимаешь, — ответил нетерпеливо Рыцарь. — Это *заглавие* так называется. А *песня* называется "Древний старичок".

— Это у песни такое заглавие?

— Да нет же ! Это *название*. А *заглавие* совсем другое: "С горем пополам".

— Так какая же это песня? —спросила Алиса в полной растерянности.

— Я как раз собирался тебе об этом сказать. "Сидящий на стене". Вот *какая* это песня!"

Понятия величины в математике и в информатике различны. Приведенный отрывок показывает, как трудно бывает преодолеть "косность мышления", привыкнуть к новому смыслу, казалось бы, знакомого понятия. К подробному изучению понятия величины мы вновь вернемся в §11.

Для того, кто хорошо усвоил понятие исполнителя, может быть полезным такое сравнение. В состав ЭВМ входит исполнитель Память, который умеет записывать информацию, передавать ее другим устройствам и стирать ее. Информация при этом бывает двух видов: алгоритмы и величины. При анализе работы ЭВМ один из учеников может исполнить роль Памяти, записывая на доске и стирая прямоугольники .

Запишем в словарь:

цел — описание величины целого типа.

В качестве дополнительного домашнего задания можно предложить учащимся составить алгоритм рисования фрагмента узо-

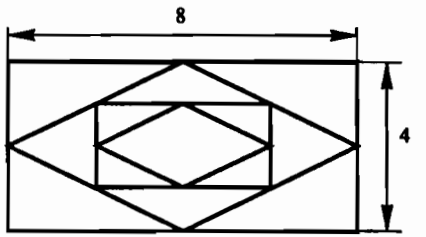


Рис. 23

ра размером 8×4 (рис. 23).

Вариант решения задачи:

```
алг нарисовать фрагмент 2
  дано | перо в левом нижнем углу
  надо | нарисован фрагмент (рис. 23)
нач
| прямоугольник (8,4)
| сместиться на вектор (—4,0)
| ромб (4,2)
| сместиться на вектор (—2,1)
| прямоугольник (4,2)
| сместиться на вектор (—2,0)
| ромб (2,1)
кон
```

Алгоритм "прямоугольник" составлен при выполнении упражнения 14 на с.51 учебника. Второй вспомогательный алгоритм:

```
алг ромб (арг вещ c,d)
  дано | перо в нижнем углу
  надо | нарисован ромб,
       | перо в нижнем углу
нач
| опустить перо
| сместиться на вектор (c,d)
| сместиться на вектор ( c,—d)
| сместиться на вектор (—c,—d)
| сместиться на вектор (c,—d)
| поднять перо
кон
```

УКАЗАНИЯ К УПРАЖНЕНИЯМ

1. Алгоритмы практически дословно совпадают с алгоритмом А11.

2,4. В результате исполнения алгоритма "домик" Чертежник рисует домик. После исполнения алгоритма перо будет находиться в левом нижнем углу. Полезно заметить, что с использованием алгоритма "квадрат" алгоритм "домик" можно сократить: вместо первых шести команд можно написать вызов "квадрат (4)".

Результатом исполнения алгоритма "улица из трех домиков" является рисунок, на котором в ряд по горизонтали выстроены три домика и промежутки между ними имеют ширину 2 (а не 6).

Для прояснения взаимосвязи основного и вспомогательного алгоритмов полезно рассмотреть несколько модификаций упражнения. Из трех компонентов упражнения — рисунок, основной алгоритм, вспомогательный алгоритм — указываются два, а третий требуется определить. Таким образом, мы получаем своеобразные "уравнения". В упражнении 2 требуется по основному и вспомогательному алгоритмам определить рисунок. Рассмотрим другие варианты.

Пусть рисунок остается прежним (улица из трех домиков), а из алгоритма "домик" удалена последняя команда (см. упражнение 4). Требуется внести соответствующие изменения в основной алгоритм. При такой постановке задачи нужно команду "сместиться на вектор (6,0)" в основном алгоритме из учебника заменить командой "сместиться на вектор (2,—4)"

Заметим, что вектор смещения в этих командах можно определить либо непосредственно, либо как сумму соответствующих векторов (—4,—4) и (6,0). Соответственно вместо команды "сместиться на вектор (2,—4)" может быть записана пара команд

```
сместиться на вектор (—4,—4)
сместиться на вектор (6,  0)
```

При таком варианте последняя команда вспомогательного алгоритма оказывается как бы выведенной из него в основной алгоритм, и должна повторяться каждый раз при вызове алгоритма "домик".

Пусть теперь рисунок остается без изменения, а основной алгоритм состоит из трех вызовов "домик" (команды смещения удалены). Требуется указать, как должен быть изменен алгоритм "домик". Нетрудно заметить, что достаточно последнюю команду в алгоритме "домик" из учебника заменить командой

```
сместиться на вектор (2,—4)
```

Заметим, что результат исполнения так полученного основного алгоритма отличается от результата исполнения алгоритма из учебника расположением пера: в случае "старого" алгоритма перо расположено в левом нижнем углу последнего, третьего домика, в случае "нового" алгоритма — на две единицы правее правого нижнего угла.

В качестве дополнительного задания можно предложить учащимся составить алгоритм рисования домика, содержащий как можно меньше команд. Ясно, что любой алгоритм рисования домика должен содержать не менее шести команд смещения — изображение домика содержит шесть отрезков. Нарисовать домик с помощью шести команд можно, если начать рисование с верхнего левого (или правого) угла (см. упражнение 9 из § 5):

```
опустить перо
сместиться на вектор (2, 2)
сместиться на вектор (2, −2)
сместиться на вектор (−4, 0)
сместиться на вектор (0, −4)
сместиться на вектор (4, 0)
сместиться на вектор (0, 4)
поднять перо
```

3. Возможно решение, аналогичное тому, которое дается в упражнении 1: записать шесть вызовов алгоритма "домик" и вставить между вызовами команды "сместиться на вектор (6,0)". Возможно и более рациональное решение, если воспользоваться алгоритмом "улица из трех домиков" как вспомогательным. Улица из шести домиков будет нарисована после исполнения такого алгоритма:

```
алг улица из шести домиков
нач
| улица из трех домиков
| сместиться на вектор (6,0)
| улица из трех домиков
кон
```

Полезно рассмотреть такую же задачу для девяти домиков. Для рисования улицы из девяти домиков можно трижды вызвать вспомогательный алгоритм "улица из трех домиков", а можно составить алгоритм и так:

```
алг улица из девяти домиков
нач
| улица из шести домиков
| сместиться на вектор (6,0)
| улица из трех домиков
кон
```

На примере алгоритма из этого упражнения хорошо видно, как один и тот же алгоритм может выступать и в роли основного и в роли вспомогательного.

Здесь, впрочем, нужно быть достаточно осторожными (смотри парадокс курицы и яйца в общих указаниях). К обсуждению вопроса об основных и вспомогательных алгоритмах целесообразно обратиться и при изучении алгоритмов с аргументами.

5. Рассмотрим алгоритм для выполнения первого из рисунков. Выделим два вспомогательных алгоритма: рисование квадрата (алгоритм А16 учебника) и рисование одной горизонтальной черты.


```
алг заштрихованный квадрат
  дано | перо Чертежника в левом нижнем углу А будущего
       | заштрихованного квадрата и поднято
  надо | нарисован заштрихованный квадрат с длиной стороны 4
       | и горизонтальной штриховкой с шагом 0.4, перо
       | Чертежника в точке А и поднято
нач
| квадрат (4)
| горизонтальная черта (4, 0.4)
| горизонтальная черта (4, 0.4)
| горизонтальная черта (4, 0.4)
| горизонтальная черта (4, 0.4)
| горизонтальная черта (4, 0.4)
| горизонтальная черта (4, 0.4)
| горизонтальная черта (4, 0.4)
| горизонтальная черта (4, 0.4)
| горизонтальная черта (4, 0.4)
| сместиться на вектор (0,−3.6)
кон
```

```
алг горизонтальная черта (арг вещ a, b)
  дано | перо Чертежника на b ниже левого конца черты и
       | поднято
  надо | проведена горизонтальная черта длины a на b выше
       | исходного положения пера, перо в левом конце
       | проведенной черты и поднято
нач
| сместиться на вектор (0,b)
| опустить перо
| сместиться на вектор (a,0)
| поднять перо
| сместиться на вектор (−a,0)
кон
```

Предложенный алгоритм не является оптимальным по числу команд (можно, например, сократить команды "поднять перо" и "опустить перо"). Приведенное решение в определенном смысле иллюстрирует дисциплину применения вспомогательных алгоритмов.

Основные алгоритмы в этом упражнении и в двух других состоят из большого числа повторяющихся команд, поэтому можно наметить их построение, не приводя полной записи.

К упражнению имеет смысл вернуться при изучении команд повторения.

6. Основной алгоритм практически дословно совпадает с алгоритмом А11 из п. 6.1. На это нужно специально обратить внимание. Преимущества применения вспомогательных алгоритмов

проявляются здесь особенно рельефно, так как вспомогательные алгоритмы отличаются не только в деталях, но и исполняются разными исполнителями.

```
алг СССР
  дано | Робот в левом нижнем углу будущего слова, стен на
       | поле нет
  надо | закрашены буквы СССР
нач
| буква С
| буква С
| буква С
| буква Р
кон

алг буква С
  дано | Робот в левом нижнем углу будущей буквы С, стен
       | на поле нет
  надо | закрашена буква С, Робот в левом нижнем углу
       | следующей буквы
нач
| закрасить; вверх; закрасить; вверх
| закрасить; вверх; закрасить; вверх
| закрасить; вправо; закрасить; вправо; закрасить
| влево; вниз; вниз; вниз; вниз
| закрасить; вправо; закрасить
| вправо; вправо
кон

алг буква Р
  дано | Робот в левом нижнем углу будущей буквы Р, стен
       | на поле нет
  надо | закрашена буква Р, Робот в левом нижнем углу
       | следующей буквы
нач
| закрасить; вверх; закрасить; вверх
| закрасить; вверх; закрасить; вверх
| закрасить; вправо; закрасить; вправо; закрасить
| вниз; закрасить; вниз; закрасить
| влево; закрасить
| вправо; вправо; вправо; вниз; вниз
кон
```

Алгоритмы в остальных четырех заданиях различаются только вспомогательными алгоритмами. Основной алгоритм во всех случаях один и тот же и совпадает с алгоритмом А13 из п. 6.4 учебника. Заметим, что достаточно составить два вспомогательных алгоритма "закрашивание блока" и три вспомогательных алгоритма "обход стены": для рисунков 21,б и 22,а совпадают алгоритмы закраски; для рисунков 21,а и 22,в — алгоритмы обхода; для рисунков 22,б и 22,в — алгоритмы закраски. (Рисунки учебника.)

Это упражнение хорошо иллюстрирует эффективность применения вспомогательных алгоритмов. Один и тот же основной алгоритм пригоден для решения серии однотипных задач; с другой стороны — один и тот же вспомогательный алгоритм может быть использован в нескольких задачах.

7, 12. Исполняя алгоритм "фрагмент", Робот закрашивает периметр квадрата 4×4, начиная с верхнего левого угла, и приходит в правый нижний угол квадрата. В результате исполнения алгоритма "фигура" будут закрашены по периметру 5 квадратов. Левая верхняя клетка каждого квадрата, начиная со второго, совпадает с правой нижней клеткой предыдущего квадрата.

Легко видеть, что угловые клетки "внутренних" квадратов, т.е. второго, третьего и четвертого, закрашиваются дважды. Таким образом, дважды будут закрашены 4 клетки. В алгоритме "фрагмент" выдается 12 команд закраски. Этот алгоритм вызывается в основном алгоритме 5 раз. Значит, в общей сложности Роботу будет выдано 60 команд закраски.

Эти выводы можно соотнести с общим числом закрашенных клеток. После первого вызова алгоритма "фрагмент" закрашиваются 12 клеток. Каждый следующий вызов добавляет еще по 11 клеток. Таким образом, всего оказывается $12 + 4 \times 11 = 56$ закрашенных клеток. То же число получается, если из общего числа команд закраски вычесть число тех команд, которые выдаются в тот момент, когда Робот находится в закрашенной клетке (т.е. число дважды закрашенных клеток): $56 = 60 - 4$.

8. Изменения касаются двух вещей. Во-первых, сторона квадрата увеличивается на одну клетку. Во-вторых, меняется относительное расположение клетки, в которой заканчивается исполнение алгоритма "фрагмент". Внесем изменения, соответствующие рисунку 23.

```
алг фрагмент
нач
| закрасить; вправо; закрасить; вправо; закрасить; вправо
| закрасить; вправо
| закрасить; вниз; закрасить; вниз; закрасить; вниз
| закрасить; вниз
| закрасить; влево; закрасить; влево; закрасить; влево
| закрасить; влево
| закрасить; вверх; закрасить; вверх; закрасить; вверх
| закрасить
| вправо; вправо; вниз
кон
```

В оставшихся случаях изменения коснутся только предпоследней строчки :

вправо — для рисунка 24, а, и
вправо; вправо — для рисунка 24, б.

9. Квадрат размером 20×20 разбивается на 25 квадратов размером 4×4: 5 рядов по 5 квадратов.Одно из возможных решений — использовать 25 раз вспомогательный алгоритм. Перед вызовом алгоритма выводить Робота в левый верхний угол очередного квадрата. Можно для удобства ввести дополнительные вспомогательные алгоритмы. Например:

```
алг к следующему ряду
нач
| вниз; вниз; вниз; вниз
| влево; влево; влево; влево; влево
| влево; влево; влево; влево; влево
| влево; влево; влево; влево; влево
кон
```

Еще более четкой структура алгоритма станет после введения вспомогательного алгоритма "ряд":

```
алг ряд
нач
| фрагмент; вправо; вправо; вправо; вправо
| фрагмент; вправо; вправо; вправо; вправо
| фрагмент; вправо; вправо; вправо; вправо
| фрагмент; вправо; вправо; вправо; вправо
| фрагмент
кон
```

С использованием этих вспомогательных алгоритмов основной алгоритм запишется так:

```
алг картина
нач
| ряд; к следующему ряду
| ряд; к следующему ряду
| ряд; к следующему ряду
| ряд; к следующему ряду
| ряд
кон
```

10. Составим план. Робот проходит, закрашивая клетки, по горизонтальному ряду клеток слева направо, переходит на следующий ряд (вниз) и проходит, закрашивая клетки, по этому ряду справо налево, затем переходит на следующий ряд. Эти действия повторяются 5 раз. Теперь можно перейти к составлению алгоритмов. Сначала составим основной алгоритм:

```
алг шахматное поле
  дано | стен на поле нет
  надо | закрашен в шахматном порядке квадрат
       | 10×10, левая верхняя вершина совпадает
       | с начальным положением Робота; после
       | исполнения алгоритма Робот на одну
       | клетку ниже левой нижней вершины квадрата
нач
| закрасить ряд 1; вниз; закрасить ряд 2; вниз
| закрасить ряд 1; вниз; закрасить ряд 2; вниз
| закрасить ряд 1; вниз; закрасить ряд 2; вниз
| закрасить ряд 1; вниз; закрасить ряд 2; вниз
| закрасить ряд 1; вниз; закрасить ряд 2; вниз
кон
```

Теперь перейдем к составлению вспомогательных алгоритмов:

```
алг закрасить ряд 1
  дано | стен на поле нет
  надо | закрашен в шахматном порядке ряд из 10 клеток
       | правее Робота, исходная клетка закрашена,
       | Робот находится в правой клетке ряда
нач
| закрасить; вправо; вправо; закрасить; вправо; вправо
| закрасить; вправо; вправо; закрасить; вправо; вправо
| закрасить; вправо
кон
```

```
алг закрасить ряд 2
  дано | стен на поле нет
  надо | закрашен в шахматном порядке ряд из 10 клеток
       | левее Робота, исходная клетка закрашена,
       | Робот находится в левой клетке ряда
нач
| закрасить; влево; влево; закрасить; влево; влево
| закрасить; влево; влево; закрасить; влево; влево
| закрасить; влево
кон
```

Можно поставить дополнительные вопросы. Например, что нужно записать в поле **надо** основного алгоритма и что полу-

чится после его исполнения, если в нем переставить местами вызовы "закрасить ряд 1" и "закрасить ряд 2" ? Как изменить вспомогательные алгоритмы, чтобы верхняя левая клетка квадрата оставалась незакрашенной ?

11. Составим алгоритм для рисунка 27,а учебника. Заметим, что орнамент составлен из рядов трех видов: в первом закрашено 6 клеток, во втором — 12, в третьем закрашенных клеток нет. Составим план. Робот закрашивает ряд и переходит в левую клетку следующего ряда. Эти действия повторяются, пока не будет получен нужный орнамент.

```
алг орнамент
  дано | стен на поле нет, Робот в клетке А
  надо | закрашены отмеченные клетки, Робот в клетке Б
нач
| закрасить ряд 1; закрасить ряд 2; закрасить ряд 3
| закрасить ряд 2; закрасить ряд 1; закрасить ряд 1
| закрасить ряд 2; закрасить ряд 3
| закрасить ряд 2; закрасить ряд 1
кон
```

Заметим, что в алгоритме "закрасить ряд 3" вызывается всего лишь одна команда "вниз". Можно упростить структуру основного алгоритма, введя вспомогательные алгоритмы, при исполнении которых закрашиваются полоски из двух рядов.

```
алг орнамент
  дано | стен на поле нет, Робот в клетке А
  надо | закрашены отмеченные клетки, Робот в клетке Б
нач
| закрасить полосу 1
| закрасить ряд 3
| закрасить полосу 2; закрасить полосу 1
| закрасить ряд 3
| закрасить полосу 2
кон
```

Здесь алгоритм " закрасить полосу 1" содержит вызовы:

закрасить ряд 1; закрасить ряд 2

а алгоритм "закрасить полосу 2" — вызовы:

закрасить ряд 2; закрасить ряд 1

13. Возможность вызова алгоритма "квадрат" с нулевым и с отрицательным значениями может оказаться для учащихся

достаточно неожиданной. Чтобы сделать результат понятным, нужно выписать тело алгоритма "квадрат", затем формально подставить вместо аргумента а указанные значения и исполнить получившуюся серию команд. Вызов "квадрат (0)" дает серию команд:

```
опустить перо
сместиться на вектор (0,0)
сместиться на вектор (0,0)
сместиться на вектор (0,0)
сместиться на вектор (0,0)
поднять перо
```

Таким образом, в результате исполнения алгоритма будет просто отмечена исходная точка.

Вызов "квадрат (—1)" приводит к серии команд:

```
опустить перо
сместиться на вектор (—1,0)
сместиться на вектор (0,—1)
сместиться на вектор (1,0)
сместиться на вектор (0,1)
поднять перо
```

В результате будет нарисован квадрат со стороной 1, правый верхний угол которого совпадает с исходной точкой. Заметим, что перо по-прежнему обходит квадрат против часовой стрелки.

14. Составим алгоритм рисования прямоугольника по аналогии с алгоритмом рисования квадрата.

```
алг прямоугольник (арг вещ x,y,a,b)
  дано | перо Чертежника поднято
  надо | нарисован прямоугольник со сторонами длины a и b,
       | параллельными осям, с левым нижним
       | углом в точке (x,y); перо Чертежника в левом
       | нижнем углу и поднято
нач
| сместиться в точку (x,y)
| опустить перо
| сместиться на вектор (a,0)
| сместиться на вектор (0,b)
| сместиться на вектор (—a,0)
| сместиться на вектор (0,—b)
| поднять перо
кон
```

Можно задать вопросы, аналогичные тем, которые заданы в упражнении 4. Какую последовательность команд исполнит Чертежник и что будет нарисовано при вызовах:

а) прямоугольник (x,y,0,0) — точка;
б) прямоугольник (x,y,1,0) — горизонтальный отрезок длины 1;
в) прямоугольник (x,y,0,1) — вертикальный отрезок длины 1;
г) прямоугольник (x,y,1,2) — прямоугольник со сторонами 1 и 2;
д) прямоугольник (x,y, 1,—2) — прямоугольник со сторонами 1 и 2, у которого левый верхний угол находится в точке (x,y);
е) прямоугольник (x,y, —1,—2) — прямоугольник со сторонами 1 и 2, у которого правый верхний угол находится в точке (x,y)?

Результаты исполнения алгоритма представлены на рисунке 24. В случаях г) и е), т. е. при ab>0, перо обходит прямоугольник против часовой стрелки, а в случае д), т.е. при ab<0, — по часовой стрелке. Заметим, что вызов "прямоугольник (x,y, a,a)" приводит к точно такой же последовательности команд, что и вызов "квадрат (a)" (с точностью до команды "сместиться в точку (x,y)").

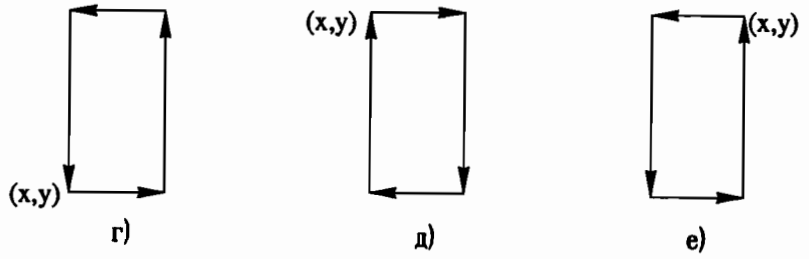


Рис. 24

Можно дать дополнительное задание с математическим содержанием. Например, составить алгоритм, в результате исполнения которого будет нарисован прямоугольник со сторонами 1 и 2, полученный поворотом на 90° около точки (x,y) против часовой стрелки из прямоугольника г) на рисунке 24:

прямоугольник (x,y,—2,1).

Здесь соответствующее геометрическое преобразование применено к аргументам алгоритма (a,b). Аналогичные вопросы можно поставить и для других преобразований.

Заметим, что в алгоритме "прямоугольник" роль аргумента играет по существу вектор-диагональ (a,b).

15. Поскольку точные координаты на картинках не заданы, то это упражнение разбивается на две части: сначала надо "придумать координаты" — например, перерисовать эти картинки на листочке в клеточку , а потом составить соответствующий алгоритм, в котором будет столько вызовов вспомогательного алгоритма "прямоугольник", сколько прямоугольников в соответствующей картинке.


```
алг рисунок Робота
  надо | нарисован Робот
нач
| прямоугольник (1,0,7,6)     | голова
| прямоугольник (0,1,1,3)     | левое ухо
| прямоугольник (8,1,1,3)     | правое ухо
| прямоугольник (2,3,2,2)     | левый глаз
| прямоугольник (5,3,2,2)     | правый глаз
| прямоугольник (3,1,3,1)     | рот
кон
```

```
алг рисунок собачки
  надо | нарисована собачка
нач
| прямоугольник (0,1,12,2)    | туловище
| прямоугольник (0,3, 1,1)    | хвостик
| прямоугольник (1,0, 1,1)    | левая задняя лапа
| прямоугольник (2.5,0,1,1)   | правая задняя лапа
| прямоугольник (8.5,0, 1,1)  | левая передняя лапа
| прямоугольник (10,0,1,1 )   | правая передняя лапа
| прямоугольник (12,2, 2,3)   | голова
| прямоугольник (14,2, 2,2)   | пасть
| прямоугольник (12.4,5,0.4,1) | маленькое ухо
| прямоугольник (13.2,5,0.4,2) | большое ухо
кон
```

В этом месте учитель может предложить ученикам подсчитать, сколько строк было бы в алгоритмах, если попытаться записать их без использования вспомогательных алгоритмов, провести дискуссию о том, что мы выигрываем (компактность, ясность, наше время, затраченное на составление алгоритма, возможность его изменить) и что теряем (некоторые линии рисуются дважды, перо для данной конкретной картинки движется не самым оптимальным образом) от использования вспомогательных алгоритмов.

В качестве дополнительного задания можно предложить учащимся нарисовать Робота, используя при этом соображения симметрии.

16. Эта задача очень похожа на предыдущую — прежде всего схемы надо перерисовать на бумагу в клеточку и только потом рисовать. Заметим, что в схемах кроме прямоугольников используются отрезки, поэтому при рисовании разумно использовать два вспомогательных алгоритма: "прямоугольник (арг вещ x,y,a,b)"(из упражнения 14) и "отрезок (арг вещ x1,y1, x2,y2)", который должен рисовать отрезок от точки (x1,y1) до точки (x2,y2):

```
алг схема А
  надо | нарисована схема, изображенная на рисунке
       | учебника
нач
| отрезок ( 0,3,2,3 )
| отрезок ( 2,1,2,5 )
| отрезок (2,1,4,1)
| прямоугольник (4,0,3,2)
| отрезок (7,1,9,1)
| отрезок (2,5,4,5)
| прямоугольник (4,4,3,2); отрезок (7,5,9,5)
| отрезок (9,1,9,5); отрезок (9,3,11,3)
кон

алг отрезок ( арг вещ x1,y1, x2,y2 )
  надо | нарисован отрезок от точки (x1,y1) до точки (x2,y2)
нач
| сместиться в точку ( x1, y1 )
| опустить перо
| сместиться в точку ( x2, y2 )
| поднять перо
кон
```

Внимание! Ученики могут придумать решение, в котором используются простейшие выражения вида "x+2", "y—1" и др., составив, например, такой вспомогательный алгоритм:

```
алг сопротивление (арг вещ x,y )
  надо | нарисовано сопротивление, начиная с точки (x,y)
нач
| отрезок ( x,y, x+2,y )
| прямоугольник ( x+2,y—1, 3,2 )
| отрезок ( x+5,y, x+7,y )
кон
```

Такое решение следует только приветствовать и использовать для перехода к следующему материалу — к записи выражений в алгоритмическом языке. С использованием вспомогательного алгоритма "сопротивление" решение упражнения 16,б можно записать так:

```
алг схема Б
  надо | нарисована схема Б
нач
| сопротивление ( 0,1 )
| сопротивление ( 7,1 )
кон
```



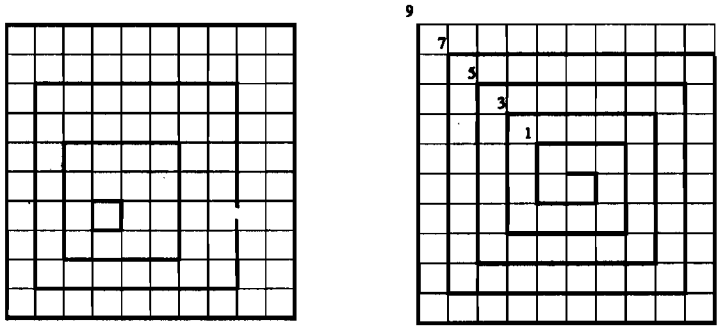
```
алг схема В
  надо | нарисована схема В
нач
| отрезок (0,7,2,7); прямоугольник ( 2,6,3,2)
| отрезок (5,7,9,7); прямоугольник (9,6,3,2)
| отрезок (12,7,16,7); прямоугольник (16,6,3,2)
| отрезок (19,7,21,7)
|       | нарисован верхний ряд сопротивлений
| отрезок ( 7,0,7,2); прямоугольник (6,2,3,2)
| отрезок (7,5,7,7); отрезок (14,0,14,2)
| прямоугольник (13,2,3,2); отрезок (14,5,14,7)
| отрезок (21,0,21,2); прямоугольник (20,2,3,2)
| отрезок (21,5,21,7)
|       | нарисованы вертикальные сопротивления
| отрезок (0,0,21,0)
кон
```

18. а)Результат исполнения алгоритма "тоннель" изображен на рисунке 25,а.

В качестве дополнительного задания можно предложить учащимся определить длину стороны наибольшего квадрата, если тоннель состоит не из четырех (как в заданном алгоритме), а из n квадратов. Здесь нужно заметить, что длина стороны наименьшего квадрата равна 1, длина стороны каждого следующего квадрата увеличивается на 3. Таким образом, длины сторон квадратов образуют арифметическую прогрессию с начальным членом 1 и разностью 3. Следовательно, длина стороны n-го квадрата равна 3n—2.

б) К решению предложенной задачи можно подойти двумя способами.

1-й способ. Достаточно непосредственно записать серию вызываемых команд Чертежника и затем определить результат их исполнения (рис. 25,б).



а) б)

Рис. 25

Отмечены точки, в которых перо Чертежника оказывается после вызова алгоритма "виток".

2-й способ. Нужно разобрать исполнение алгоритма "виток (a)" (рис. 26,а).

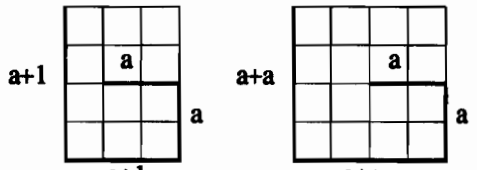


Рис. 26

Перо Чертежника смещается относительно своего исходного положения (перед исполнением команд из тела цикла) на вектор (−1,1). Последний результат можно получить не только непосредственно, прослеживая результаты исполнения команд, но и "теоретическим" путем, сложив все векторы смещения:

(a,0)+(0,−a)+(−a−1,0)+(0,a+1)=(−1,1).

19. Найдем вектор смещения при выполнении алгоритма "виток (a)": (a,0) + (0,−a) + (−a−a,0) + (0,a+a) = (−a, a), рисунок 26,б. Здесь вектор смещения после исполнения алгоритма зависит от величины аргумента. Так что спираль, получающаяся при исполнении основного алгоритма, раскручивается с ускорением.

20. Ясно, что "зеркальная" спираль будет раскручиваться против часовой стрелки. Чтобы получить такую спираль, выберем ось и отобразим исходную спираль симметрично относительно этой оси. Например, возьмем в качестве такой оси вертикальную прямую, проходящую через начало спирали. Тогда в командах алгоритма "виток" нужно поменять на противоположные знаки первых аргументов во всех командах смещения на вектор. Если выбрана горизонтальная ось, то знаки меняются у вторых аргументов. В качестве дополнительного задания можно предложить учащимся рассмотреть и другие варианты.

21.
```
алг спираль
    надо | нарисована спираль ( рис. 29 учебника).
нач
| опустить перо
| полувиток (1); полувиток (−2)
| полувиток (3); полувиток (−4)
| полувиток (5);
| поднять перо
кон
```



```
алг полувиток (арг вещ a)
нач
| сместиться на вектор (a, a);
| сместиться на вектор (a,−a)
кон
```

Учитель может облегчить эту задачу, пририсовав еще два звена спирали, тогда слабые ученики смогут ввести вспомогательный алгоритм "виток" по образцу упражнения 18,б и не использовать знаки для задания направления полувитка:

```
алг спираль
    надо | нарисована спираль (рис. 29 учебника)
         | плюс еще 2 звена
нач
| опустить перо
| виток (1)
| виток (3)
| виток (5)
| поднять перо
кон

алг виток (арг вещ a)
нач
| сместиться на вектор ( a, a)
| сместиться на вектор ( a,−a)
| сместиться на вектор (−a−1,−a−1)
| сместиться на вектор (−a−1, a+1)
кон
```

Здесь также полезно рассмотреть вопрос о перемещении пера при рисовании витка: перо смещается на вектор (−2,0).

22.
```
алг спираль
    надо | нарисована спираль, с растущим расстоянием
         | между витками
нач
| опустить перо
| полувиток (1); полувиток (−2)
| полувиток (4); полувиток (−8)
| полувиток (16)
| поднять перо
кон
```

В случае второго решения (с полными витками) надо менять как сам алгоритм "спираль", так и алгоритм "виток" (по образцу упражнения 19):

```
алг спираль
  надо | нарисована спираль с растущим расстоянием
       | между витками
нач
| опустить перо
| виток (1); виток (4); виток (16)
| поднять перо
кон

алг виток (арг вещ a)
нач
| сместиться на вектор (a, a)
| сместиться на вектор (a, −a)
| сместиться на вектор (−a−a,−a−a)
| сместиться на вектор (−a−a,a+a)
кон
```

24. а)
```
алг орнамент
нач
| ряд; ряд; ряд
кон
```

б)
```
алг ряд
нач
| фрагмент; фрагмент
| фрагмент; фрагмент
| сместиться на вектор (−16,−4)
кон
```

в)
```
алг фрагмент
нач
| сместиться на вектор (2, 0)
| опустить перо
| сместиться на вектор (2,−2)
| сместиться на вектор (−2,−2)
| сместиться на вектор (0, 1)
| сместиться на вектор (−2, 0)
| сместиться на вектор (0, 2)
| сместиться на вектор (2, 0)
| сместиться на вектор (0, 1)
| поднять перо
| сместиться на вектор (2, 0)
кон
```

В остальных пунктах алгоритмы "орнамент" и "ряд" мы будем записывать кратко в следующем виде:

а) 3 ряда по 4 фрагмента 4×4; вектор (−16,−4).

Пункты б)—д) записываются аналогично:

б) 4 ряда по 8 фрагментов 2×2; вектор (−16,−2).

```
алг фрагмент
нач
| сместиться на вектор (1, 0)
| опустить перо; сместиться на вектор (0,−2)
| поднять перо
| сместиться на вектор (−1, 1)
| опустить перо; сместиться на вектор (2, 0)
| поднять перо
| сместиться на вектор (0,−1)
кон
```

в) 2 ряда по 4 фрагмента 4×4; вектор (−16,−4).

```
алг фрагмент
нач
| опустить перо
| сместиться на вектор (1,−1); сместиться на вектор (2, 0)
| поднять перо; сместиться на вектор (−2, 0); опустить перо
| сместиться на вектор (0,−2); сместиться на вектор (−1,−1)
| поднять перо; сместиться на вектор (1, 1); опустить перо
| сместиться на вектор (2, 0); сместиться на вектор (1,−1)
| поднять перо; сместиться на вектор (−1, 1); опустить перо
| сместиться на вектор (0, 2); сместиться на вектор (1, 1)
| поднять перо
кон
```

г) 6 рядов по 6 фрагментов 2×2; вектор (−12,−2).

```
алг фрагмент
нач
| опустить перо
| сместиться на вектор (2,−2)
| поднять перо
| сместиться на вектор (−2, 0)
| опустить перо
| сместиться на вектор (2, 2)
| поднять перо
кон
```

д) 3 ряда по 3 фрагмента 4×4; вектор (−12,−4).

```
алг фрагмент
нач
| опустить перо
| сместиться на вектор (2,−1)
| сместиться на вектор (0,−2)
| сместиться на вектор (−2,−1)
| поднять перо; сместиться на вектор (4, 0); опустить перо
| сместиться на вектор (−2, 1)
| поднять перо; сместиться на вектор (0, 2); опустить перо
| сместиться на вектор (2, 1); поднять перо
кон
```

Пункт е) содержит существенное усложнение — используемые в нем фрагменты частично наложены друг на друга. Сами фрагменты не отличаются от использованных в пункте в). Для наложения их друг на друга в конец алгоритма "фрагмент" добавлена команда смещения пера в начало следующего фрагмента:

е) 3 ряда по 5 фрагментов 4×4; вектор (−10,−4).

```
алг фрагмент
нач
| опустить перо
| сместиться на вектор (1,−1); сместиться на вектор (2, 0)
| поднять перо; сместиться на вектор (−2, 0); опустить перо
| сместиться на вектор (0,−2); сместиться на вектор (−1,−1)
| поднять перо; сместиться на вектор (1, 1); опустить перо
| сместиться на вектор (2, 0); сместиться на вектор (1,−1)
| поднять перо; сместиться на вектор (−1, 1); опустить перо
| сместиться на вектор (0, 2); сместиться на вектор (1, 1)
| поднять перо
| сместиться на вектор (−2,0)
кон
```

Пункт ж) содержит другое усложнение: фрагмент состоит из нескольких "колец" уменьшающегося размера, поэтому в алгоритме "фрагмент" разумно использовать вспомогательный алгоритм "кольцо (арг вещ а)", рисующий кольцо "размера" а:

ж) 2 ряда по 2 фрагмента 8×8; вектор (−12, −4).

```
алг фрагмент
нач
| кольцо (6); кольцо (4); кольцо(2); кольцо (0)
| сместиться на вектор (4, 4)
кон
```



```
алг кольцо (арг вещ а)
нач
| сместиться на вектор (1, 0)
| опустить перо
| сместиться на вектор (а, 0); сместиться на вектор (1,−1)
| сместиться на вектор (0,−а); сместиться на вектор (−1,−1)
| сместиться на вектор (−а, 0); сместиться на вектор (−1, 1)
| сместиться на вектор (0, а); сместиться на вектор (1, 1)
| поднять перо
| сместиться на вектор (0,−1)
кон
```

Пункт з) содержит оба усложнения: фрагмент содержит "кольца" уменьшающегося размера, фрагменты частично накладываются друг на друга как по горизонтали, так и по вертикали:

з) 2 ряда по 3 фрагмента 7×7; вектор (−12,−4).

```
алг фрагмент
нач
| сместиться на вектор (4, 0)
| кольцо (3); кольцо (2); кольцо (1); кольцо (0)
| сместиться на вектор (0, 4)
кон
```

```
алг кольцо (арг вещ а)
нач
| опустить перо
| сместиться на вектор (а, −а); сместиться на вектор (0,−1)
| сместиться на вектор (−а,−а); сместиться на вектор (−1,0)
| сместиться на вектор (−а, а); сместиться на вектор (0,−1)
| сместиться на вектор (а, а); сместиться на вектор (1, 0)
| поднять перо
| сместиться на вектор (0,−1)
кон
```

§ 7. АРИФМЕТИЧЕСКИЕ ВЫРАЖЕНИЯ И ПРАВИЛА ИХ ЗАПИСИ (1 ч.)

Проекты, предлагающие программирование на естественном языке, гибельны по своей сути.

Эдсгер Дейкстра

В § 7 вводятся правила записи арифметических выражений в алгоритмическом языке и использования арифметических выражений в качестве аргументов.

Основные цели. Привить навыки взаимного преобразования линейной и традиционной записи арифметических выражений.

Требования к знаниям и умениям. Учащиеся должны уметь свободно переходить от линейной формы записи арифметических выражений к обычной и обратно; вычислять значения арифметических выражений непосредственно по их линейной записи.

ОСНОВНЫЕ ПОНЯТИЯ

Способность вычислять — неотъемлемое свойство электронной ***вычислительной*** машины. Поскольку ЭВМ выполняет действия формально, необходим формальный способ записи выражений. В алгоритмическом языке в качестве такого способа принята линейная запись. Кроме формализации записи и вычисления выражений, линейная запись обеспечивает возможность ввода арифметических выражений с клавиатуры отечественных ЭВМ.

В арифметические выражения могут входить вызовы ***функций***. О составлении и использовании алгоритмов-функций говорится в § 12. В § 7 вводятся ***стандартные функции*** алгоритмического языка. Обозначения и правила вычисления этих функций известны ЭВМ без дополнительных описаний.

При вычислении значений выражений ЭВМ действует по *формальным правилам*:

1. Заменить все входящие в выражение величины их значениями.
2. Вычислить значение выражений, заключенных в скобки. (Порядок действий внутри скобок определяется пунктами 2—6.)
3. Вычислить значения функций.
4. Выполнить *справа налево* возведения в степень.
5. Выполнить слева направо умножения и деления.
6. Выполнить слева направо сложения и вычитания.

ОБЩИЕ УКАЗАНИЯ

В алгоритмическом языке (в отличие от математики) запись арифметического выражения может содержать не только знаки арифметических операций, но и обращения к функциям.

Не нужно стремиться дать формальное определение арифметического выражения. Учащиеся должны овладеть практическими навыками работы с арифметическими выражениями в алгоритмическом языке. Материал параграфа может быть использован также для закрепления навыков записи чисел с десятичной точкой.

Изучая использование арифметических выражений в алгоритмах, нужно обратить внимание на роль ЭВМ (п. 7.2). Возможность записывать арифметические выражения в командах Чертежника не означает, что Чертежник может исполнять какие-то дополнительные команды. Чертежник воспринимает в качестве аргументов только конкретные числа. Все вычисления берет на себя ЭВМ.

Заметим, что таблица операций и стандартных функций содержит функции, которые еще не изучались в курсе математики. Таблица играет справочную роль, к ней можно обращаться в дальнейшем по мере необходимости.

Все упражнения носят стандартный характер. Можно порекомендовать учащимся при возникновении затруднений обращаться к примерам из п.7.5 и действовать по образцу. В качестве дополнительных упражнений можно предложить вычислить значения выражений, записанных в линейной форме, указав значения входящих в них величин. Например, найти значение выражения (a + b) * (c + d), при a=—3, b=5, c=2 и d=1.2.

РЕКОМЕНДАЦИИ К ВЕДЕНИЮ УРОКОВ

На одном из предыдущих уроков был составлен алгоритм рисования фрагмента узора размером 8×4 (с. 83 пособия). Усложним задачу: пусть требуется нарисовать фрагмент произвольного размера m×n (рис. 27). Вспомогательные алгоритмы "прямоугольник" и "ромб" имеют произвольные аргументы и могут быть использованы при решении поставленной задачи. А вот алгоритм "нарисовать фрагмент 2" (с. 84 пособия) должен быть изменен.

```
алг фрагмент размером (арг вещ m, n)
  дано | перо в нижнем левом углу фрагмента
  надо | нарисован фрагмент размером m × n (рис. 27)
нач
| прямоугольник (m, n)
| сместиться на вектор (—m/2, 0)
| ромб (m/2, n/2)
| сместиться на вектор (—m/4, n/4)
| прямоугольник (m/2, n/2)
| сместиться на вектор (—m/4, 0)
| ромб (m/4, n/4)
кон
```

Легко заметить, что при вызове "фрагмент размером (8,4)" будет нарисовано то же, что и при исполнении алгоритма "нарисовать фрагмент 2". Но алгоритм "фрагмент размером" более сложный и требует, по крайней мере, двух подробных пояснений.

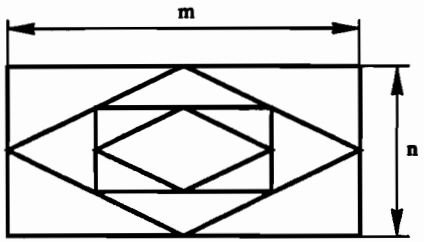


Рис. 27

Пояснение 1. В алгоритме встречаются вызовы, в которых появился знак деления (m/2, n/4, −m/4 и т.д.). Эти формулы называются ***арифметическими выражениями***. Они образуются при помощи знаков ***арифметических операций***. Чтобы арифметические выражения можно было вводить с клавиатуры, в алгоритмическом языке принята ***линейная запись*** выражений. Знаки арифметических операций представлены в верхней части таблицы на с. 56 учебника.

В отличие от математики в выражениях разрешается использовать только круглые скобки:

$$\frac{A^{x+1}+B}{C} \quad — \text{ это } \quad (A ** (x+1)+B)/C.$$

ЭВМ обрабатывает выражения формально, поэтому при их записи надо быть аккуратным и точным, ясно представлять действия ЭВМ при вычислении выражения. Например,

$$\frac{ab}{cd} \quad — \text{ это не } \quad a*b/c*d.$$

Правильная запись: a * b / (c * d) или a * b / c / d.

В арифметических выражениях можно использовать ***стандартные (встроенные) функции*** (приведены в нижней части таблицы на с. 56 учебника). При наличии в формуле тригонометрических или алгебраических функций выражение все равно принято называть арифметическим. Аргументы функций обязательно заключаются в круглые скобки:

$$\sqrt{\sin|x|} \quad — \text{ это } \quad sqrt(sin(abs(x))).$$

В алгоритмическом языке нельзя пропускать знак умножения. В выражении "ab/c" сочетание "ab" будет воспринято как имя величины, по ошибке не описанной в алгоритме. А выражения "a(b+c)" или "5a + 3b" будут расценены как синтаксические ошибки.

В выражениях разрешаются лишние скобки. Можно писать так: "(a*(x**2))/sqrt((b**2)/3)".

Пояснение 2. Оно необходимо потому, что впервые и основной, и вспомогательный алгоритмы имеют аргументы.

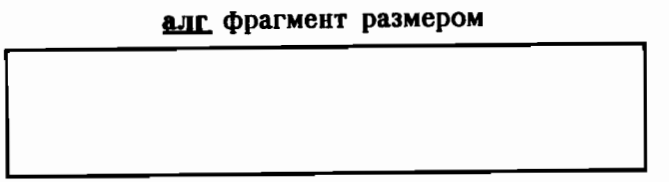
Как показывает опыт преподавания, учащиеся легко усваивают особенности записи формул на алгоритмическом языке, без труда переводят выражения из традиционной формы в линейную и обратно. Сильным ученикам становится скучно. Работа алгоритмов с аргументами — существенно более сложный материал, так что второе пояснение — для сильных.

Кроме того, учеба — это труд. Поэтому задача учителя — сделать все возможное, чтобы учиться было трудно (из школьного фольклора).

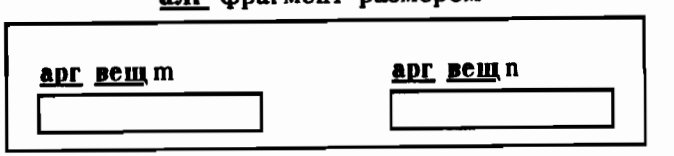
В этой части урока можно вовлечь в активную работу сразу пятерых. Первый, второй и третий выполняют алгоритмы, четвертый изображает на доске Память, пятый — играет роль Чертежника.

Рассмотрим работу ЭВМ при вызове "фрагмент размером (10,6)". ЭВМ выполняет следующие действия:

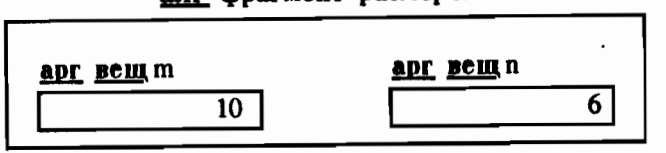
1. Находит в памяти алгоритм "фрагмент размером".
2. Выделяет в памяти место для записи значений переменных алгоритма "фрагмент размером":



3. Выделяет ячейки для записи величин:

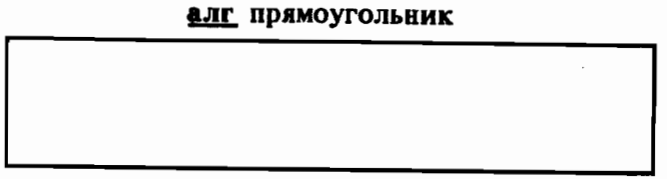


4. Заносит в ячейки значения, заданные при вызове:

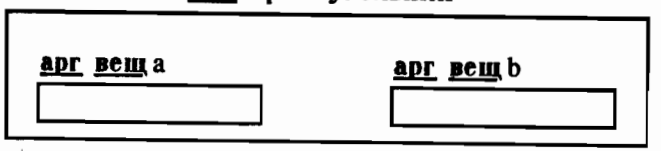


5. Проверяет **дано** (там комментарий).
6. Находит **нач**.

7. Встречает вызов "прямоугольник (m,n)".
8. Запоминает место вызова.
   8.1. Находит в памяти алгоритм "прямоугольник".
   8.2. Выделяет в памяти место:



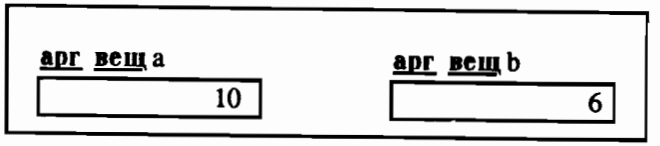
   8.3. Выделяет ячейки для записи величин:



   8.4. Значение величины m заносит в ячейку a, значение n — в ячейку b по такой схеме:

вызов: прямоугольник (m, n)

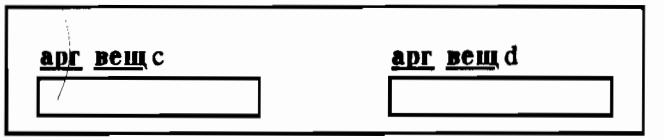
алгоритм: прямоугольник (арг вещ (a, b))



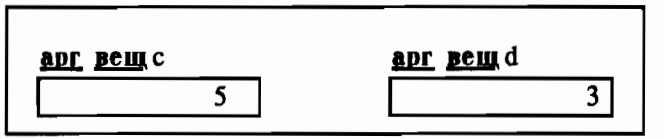
   8.5. Проверяет дано (там комментарий).
   8.6. Находит нач.
   8.7—8.12. Подставив вместо имен величин их значения, дает команды Чертежнику.
   8.13. Встречает кон.
   8.14. Проверяет надо (там комментарий).
   8.15. *Стирает в памяти часть, выделенную для алгоритма "прямоугольник".*
   8.16. Возвращается в алгоритм "фрагмент размером".
9. В выражение "сместиться на вектор (—m/2,0)" подставляет вместо имени m значение 10 и вычисляет значение выражения —10/2. Получает число —5.
10. Дает Чертежнику команду "сместиться на вектор (—5,0)".
11. Встречает вызов "ромб (m/2,n/2)".

12. Вычисляет значения m/2 и n/2, получая соответственно числа 5 и 3.
13. Запоминает место вызова.
   13.1. Находит в памяти алгоритм "ромб".
   13.2. Выделяет в памяти место:

   13.3. Выделяет ячейки для записи величин:



   13.4. Заносит в ячейки значения величин, вычисленные на шаге 12:



   13.5. Проверяет дано (там комментарий).
   13.6. Находит нач.
   13.7—13.12. Подставив вместо имен величин их значения, дает команды Чертежнику.
   13.13. Встречает кон.
   13.14. Проверяет надо (там комментарий).
   13.15. *Стирает в памяти часть, выделенную для алгоритма "ромб".*
   13.16. После этого возвращается в алгоритм "фрагмент размером".

И так далее.

Для слабых учащихся во время коллективного "прокручивания" алгоритма "фрагмент размером" можно предложить выполнить самостоятельно упражнения на с. 57 или составить алгоритм рисования елочки произвольного размера, изображенной на рисунке 28.

Вариант решения:

```
алг произвольная елочка(арг вещ a)
  дано | перо в нижней точке ствола
  надо | нарисована елочка
нач
| опустить перо
| сместиться на вектор (0, a)
| поднять перо
| сместиться на вектор (−3*a, 0)
| треугольник высотой (3*a)
| сместиться на вектор (a, 3*a)
| треугольник высотой (2*a)
| сместиться на вектор (a, 2*a)
| треугольник высотой (a)
кон

алг треугольник высотой (арг вещ c)
  дано | перо в нижней левой точке
  надо | нарисован треугольник высотой c и основанием 2*c
нач
| опустить перо
| сместиться на вектор (2*c, 0)
| сместиться на вектор (−c, c)
| сместиться на вектор (−c, −c)
| поднять перо
кон
```

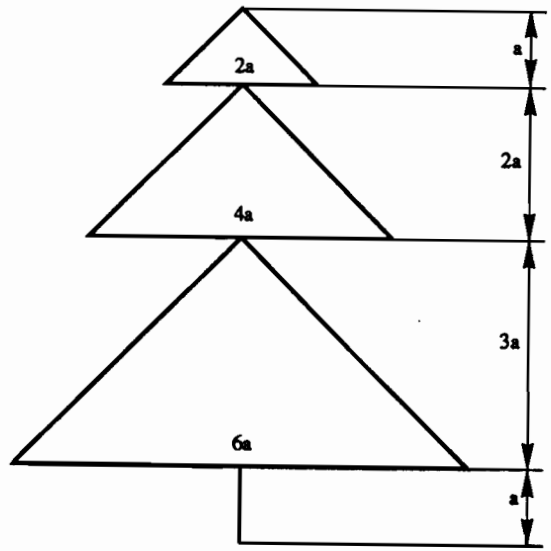


Рис. 28

В конце изучения темы учащимся может быть предложена проверочная работа, проводимая в зависимости от наличия и типа вычислительной техники в различных формах (зачет, письменная контрольная работа, практическая работа по отладке решений на ЭВМ и т.д.). Сложность и количество заданий выбирает учитель. Приведем примерные варианты заданий проверочной работы.

Задание 1

Нарисуйте результат работы алгоритма "путь" и впишите комментарии после **дано** и **надо**.

Вариант 1

```
алг путь 1
  дано
  надо
нач
| угол 1
| угол 1
| угол 1
| угол 1
кон

алг угол 1
  дано
  надо
нач
| закрасить
| вправо
| закрасить
| вниз
| закрасить
| вправо
кон
```

Вариант 2

```
алг путь 2
  дано
  надо
нач
| угол 2
| угол 2
| угол 2
кон

алг угол 2
  дано
  надо
нач
| закрасить
| вниз
| закрасить
| вправо
| закрасить
| вверх
| вверх
кон
```

Задание 2

Составьте алгоритм рисования узора:

```
алг узор
  дано | Робот в клетке А
  надо | Робот в клетке Б, нужные клетки закрашены
```

Вариант 1. Рисунок 29.
Вариант 2. Рисунок 30.

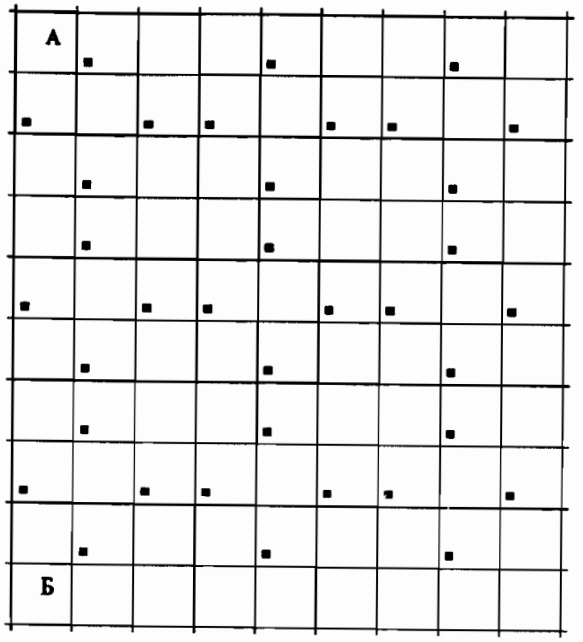


Рис. 29

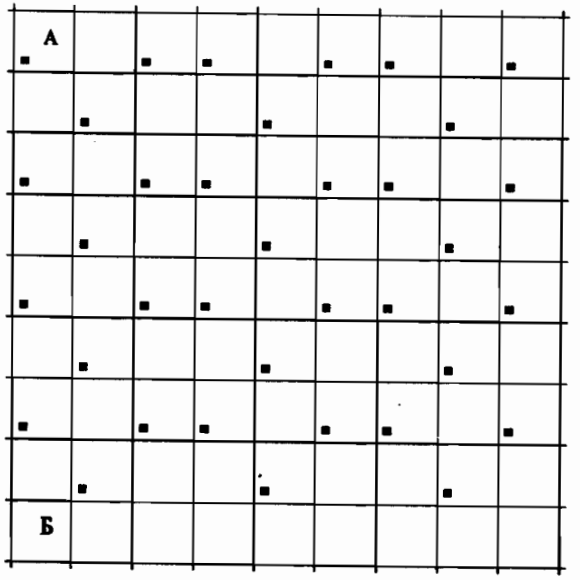


Рис. 30

Задание 3

Составьте алгоритм:

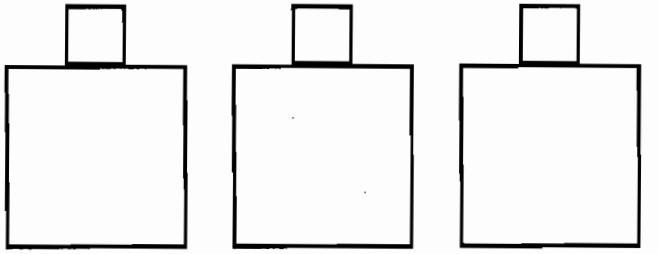
алг мишени
дано | перо Чертежника в начале координат и поднято
надо | нарисованы мишени для стрельбы

Усложненный вариант задания 3 (размеры рисунков для усложненного варианта указаны в скобках).

Составьте алгоритм:

алг мишени (**арг вещ** a)
дано | перо Чертежника в начале координат и поднято
надо | нарисованы мишени для стрельбы

Вариант 1



Размеры мишеней: "голова" 1 × 1 (a × a),
"туловище" 3 × 3 (3a × 3a).
Расстояние между мишенями равно 1 (a).

Вариант 2

Размеры мишеней: внутренний квадрат 1 × 1 (a × a),
средний квадрат 2 × 2 (2a × 2a),
внешний квадрат 3 × 3 (3a × 3a).
Расстояние между мишенями равно 1 (a).

§ 8. КОМАНДЫ АЛГОРИТМИЧЕСКОГО ЯЗЫКА. ЦИКЛ n раз

Если часы пробили тринадцать раз, то это не только означает, что тринадцатый удар был неверным. Он порождает сомнения в верности каждого из первых двенадцати ударов.

правило Мастерса

В § 8 и следующем за ним § 9 изучаются команды повторения. В § 8 вводится простейшая из команд повторения — команда n раз, на примерах раскрываются возможности ее использования в алгоритмах.

Основные цели. Раскрыть сущность команды повторения n раз как составной команды. Показать на примерах форму записи команды и специфику ее применения в алгоритмах.

Требования к знаниям и умениям. Учащиеся должны знать назначение команды повторения n раз, ее общий вид; уметь записывать, использовать и исполнять алгоритмы с этой командой.

ОСНОВНЫЕ ПОНЯТИЯ

Составные команды (повторения и ветвления) играют в алгоритмическом языке особую роль. Они не задают конкретных действий исполнителя, как, например, команда вызова. Выполняя эти команды, ЭВМ определяет последовательность вызова входящих в них простых команд. Таким образом обеспечивается возможность многократного безошибочного повторения серий команд, выбора нужной команды в зависимости от условий. Именно эти возможности лежат в основе массовых применений ЭВМ.

ОБЩИЕ УКАЗАНИЯ

Помимо непосредственных целей освоения команды n раз, § 8 несет дополнительную нагрузку. До § 8 изучались только команды-вызовы. Здесь же начинается изучение команд нового вида — составных команд алгоритмического языка. Освоение команды n раз должно помочь учащимся понять специфику составных команд и общие черты команд повторения.

Ранее учащимся неоднократно приходилось записывать в алгоритмах серии из одинаковых команд. С этой точки зрения команда n раз представляет собой естественное средство компактной и удобной записи алгоритмов.

Кроме того, существует целый класс задач, при решении которых без команды повторения не обойтись (п. 8.4).

В § 8 на примерах показаны различные возможности применения команды. Важно раскрыть учащимся специфику команды n раз как составной команды. Исполнение простой команды зависит от состояния внешней среды и от значений числовых аргументов. Для составных команд ситуация существенно сложнее. Результат исполнения команды n раз зависит не только от числа повторений, но и от тела цикла. Тем самым в роли как бы аргумента команды выступает серия команд.

Команда n раз позволяет показать учащимся повторение в "чистом виде". Нужно обратить внимание на важное отличие команды повторения от команд вызова. В алгоритме, составленном из команд вызова, по каждой команде можно точно и однозначно указать, какая команда должна исполняться следом за ней. Чтобы определить очередной вызов в алгоритме с командой повторения, нужно знать, закончилось ли ее выполнение. Например, в алгоритме из п. 8.6 после исполнения команды "вправо" следующим будет либо вызов "влево на (m)", либо "закрасить" в зависимости от того, закончено ли исполнение цикла.

РЕКОМЕНДАЦИИ К ВЕДЕНИЮ УРОКОВ

Команда повторения часто воспринимается учащимися лишь как способ сокращения записи алгоритма, содержащего повторяющиеся действия. Однако истинная необходимость в ней возникает в том случае, ***если надо указать число повторений в общем виде.*** Для иллюстрации этого применим такой методический прием: решим простую задачу, а затем постараемся усовершенствовать полученный алгоритм путем ***расширения области его применения.***

С помощью Робота требуется закрасить полосу длиной 4 клетки и шириной в одну клетку. Полоса тянется сверху вниз.

```
алг полоса длиной 4
  дано | стен на поле нет
  надо | закрашена полоса длиной 4 клетки
нач
| закрасить; вниз
| закрасить; вниз
| закрасить; вниз
| закрасить; вниз
кон
```

Недостаток написанного алгоритма в том, что он годится только для четырехклеточной полосы, т.е. имеет узкую область применения. Например, для полосы длиной 5 клеток придется писать новый алгоритм.

Вопрос. Как расширить область применения этого алгоритма?

Ответ. Опыт решения прошлых задач подсказывает, что надо использовать алгоритм с переменным аргументом вместо конкретных чисел.

Но в алгоритме "полоса длиной 4" вообще нет чисел! Как же быть? Заметим, что наш алгоритм состоит из повторяющейся пары команд: "закрасить" и "вниз". Для полосы длиной 5 клеток эту пару следует написать 5 раз. Для полосы длиной k клеток эту пару надо повторить k раз. Вот как это запишется на алгоритмическом языке:

```
алг полоса длиной (арг цел k)
  дано k > 0 | стен на поле нет
  надо | закрашена полоса длиной k клеток
нач
| нц k раз
| | закрасить; вниз
| кц
кон
```

В алгоритме "полоса длиной" мы впервые применили ***команду повторения (цикл)***. Нам встретились новые служебные слова: **нц**, **раз**, **кц**. Внутри цикла стоят две команды Робота: "закрасить" и "вниз".

В общем случае внутри цикла может быть сколько угодно команд. Обычно говорят, что цикл может включать ***серию команд***. В этой серии могут быть также вызовы вспомогательных алгоритмов. Команда, содержащая другие команды, называется составной. Цикл — ***составная команда***.

Вопросы:

1. Что будет исполнено при вызовах:

a) "полоса длиной (4)";

b) "полоса длиной (1)";

c) "полоса длиной (0)";

d) "полоса длиной (2*2)";

e) "полоса длиной (a)", где a — определенная величина целого типа;

f) "полоса длиной (sqrt(2))"?

2. В чем преимущества алгоритма "полоса длиной" по сравнению с "полоса длиной 4"?

Ответы:

1. a) будет закрашена полоса длиной 4 клетки;

b) будет закрашена 1 клетка;

c) произойдет отказ из-за нарушения условия дано (k>0);

d) будет закрашена полоса длиной 4 клетки;

e) при целом a>0 будет закрашена полоса длиной a клеток, иначе — отказ;

f) будет отказ ЭВМ. Аргумент алгоритма "полоса длиной" может быть только целым числом.

2. Во-первых, алгоритм "полоса длиной" имеет более широкую область применения, а во-вторых, его запись оказалась короче и понятнее.

Некоторые понятия, касающиеся команды повторения, рассмотрим на примере:

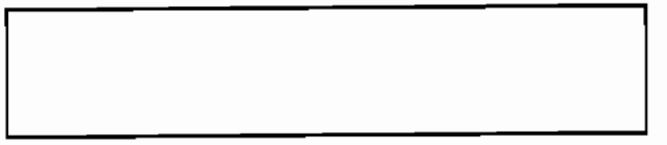
```
алг кройка
  дано | кусок ткани
  надо | отрезана нужная часть
нач
| нц 7 раз
| | отмерь
| кц
| отрежь
кон
```

Строка "**нц** 7 **раз**" называется ***заголовком цикла***. После этой строки следует ***серия команд*** (в данном случае — вызов "отмерь"). Серия команд внутри цикла называется ***телом цикла***.

Чтобы окончательно понять работу цикла, рассмотрим действия ЭВМ при вызове "полоса длиной (2)". ЭВМ выполняет следующие действия:

1. Находит алгоритм "полоса длиной".
2. Выделяет область памяти:

алг полоса длиной

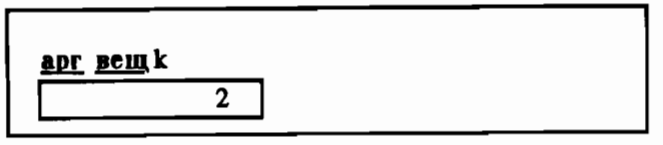


3. Выделяет ячейку:

алг полоса длиной

4. Записывает значение:

алг полоса длиной



5. Проверяет условие "k > 0", оно выполняется.
6. Находит **нач**.
7. Встречает заголовок цикла.
8. Запоминает число повторений (k).
9. Проверяет условие "число повторений > 0". Серия команд в теле цикла будет выполнена, если это условие соблюдается.
10. Выполняет команды до слова **кц** (приказывает Роботу закрасить клетку и сделать шаг вниз).
11. Из числа повторений вычитает единицу (2—1=1) и возвращается к заголовку цикла.
12. Проверяет условие "число повторений > 0", оно соблюдено.
13. Выполняет команды до слова **кц** (приказывает Роботу закрасить клетку и сделать шаг вниз).
14. Из числа повторений вычитает единицу (1—1=0) и возвращается к заголовку цикла.
15. Проверяет условие "число повторений > 0". Оно не соблюдается.
16. Пропускает все до слова **кц**.
17. Встречает **кон**.
18. Проверяет условие **надо** (там комментарий).
19. Заканчивает выполнение алгоритма и ждет указаний.

Команду повторения часто применяют в случаях, когда без нее можно было бы обойтись, но она делает запись алгоритма короче и понятнее. Для подтверждения этого вернемся к задаче 5,в на с. 46 учебника.

Используем цикл. Видно, что большой квадрат размером 1 × 1 состоит из ста маленьких 0.1 × 0.1. Воспользовавшись готовым алгоритмом "квадрат", получаем:

```
алг заштриховать квадрат
  дано | перо в левом нижнем углу
  надо | нарисован заштрихованный квадрат
нач
| нц 10 раз
| | нц 10 раз
| | | квадрат (0.1); сместиться на вектор (0.1,0)
| | кц
| | сместиться на вектор(—1, 0.1)
| кц
кон
```

Безусловно, задача допускает и другие решения. Приведенный вариант интересен тем, что содержит в явном виде конструкцию "цикл в цикле".

Вопрос. Сколько раз будет выполнен вызов каждой команды алгоритма "заштриховать квадрат"?

Ответ. Вызов "квадрат (0.1)" — 100 раз;
сместиться на вектор (0.1,0) — 100 раз;
сместиться на вектор (—1,0.1) — 10 раз.

Задание. Составьте алгоритм рисования заштрихованного квадрата произвольного размера, оставив прежним шаг штриховки.

Вариант решения.

```
алг заштриховать квадрат со стороной (арг цел c)
  дано c>0 | перо в левом нижнем углу
  надо | нарисован заштрихованный квадрат
       | с заданной стороной
нач
| нц 10*c раз
| | нц 10*c раз
| | | квадрат (0.1)
| | | сместиться на вектор (0.1,0)
| | кц
| | сместиться на вектор(—c, 0.1)
| кц
кон
```

Задание. Найдите ошибку в алгоритме.

```
алг многоточие (арг вещ k)
  дано k>0 | перо в исходном положении
  надо | нарисованы k точек
нач
| нц k раз
| | опустить перо
| | поднять перо
| | сместиться на вектор (0.1, 0)
| кц
кон
```

Ответ. В заголовке цикла не может использоваться величина вещественного типа.

Запишем в словарь:

```
нц n раз
| <серия команд>
кц
```

— составная команда цикла с известным числом повторений.

Для закрепления материала предложим несколько несложных задач. Например, требуется использовать Робота в зоопарке, чтобы временно заменить заболевшего пони, т.е. составить алгоритм с таким заголовком:

```
алг катание на Роботе (арг цел с)
  дано | с ≥ 10 | количество заплаченных копеек,
       |        | стоимость катания по кругу 10 коп.
  надо | Робот пробежал с≤10 кругов
```

При этом учтем, что кругом в зоопарке называется прямоугольник 5 × 15 клеток.

Возможное решение.

```
нач
| нц int (с≤10) раз
| | пройти один круг
| кц
кон
```

```
алг пройти один круг
  дано | Робот в исходном положении
  надо | пройден один круг
нач
| нц 15 раз
| | вправо
| кц
| нц 5 раз
| | вниз
| кц
| нц 15 раз
| | влево
| кц
| нц 5 раз
| | вверх
| кц
кон
```

Заметим, что в алгоритме "катание на Роботе" без цикла не обойтись, а в алгоритме "пройти один круг" он используется только для сокращения и упрощения записи.

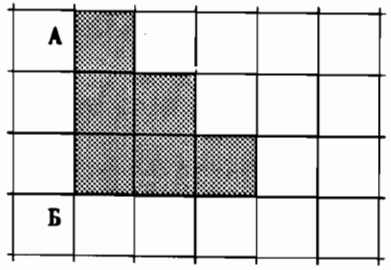


Рис. 31

Задача. Требуется разметить часть поля Робота в виде лестницы из трех ступеней (рис. 31). При этом можно воспользоваться алгоритмом "закрасить ряд" (А29) на с. 60 учебника как вспомогательным.

При проверке алгоритмов следует проконтролировать понимание учащимися не только работы цикла с заданным числом повторений, но и процесса вызова алгоритма с аргументами.

УКАЗАНИЯ К УПРАЖНЕНИЯМ

1. В упражнении используются вспомогательные алгоритмы "закрасить прямоугольник" , "закрасить ряд", "вверх на" (А30, А29, А28) и "влево на". Ответ на вопрос о числе закрашенных клеток можно получить по рисунку 33 из учебника. Для обоснования полученного ответа и подсчета общего числа команд требуется более детальный анализ работы алгоритма.

а) Запишем команды, которые будут выполняться после подстановки указанных значений параметров (мы записываем те команды, которые запрашиваются после вызова алгоритма "прямоугольник").

```
нач
| нц 1 раз
| | закрасить ряд (1)
| | вниз
| кц
| вверх на (1)
кон
```

Более подробно:

закрасить
вправо
влево
вниз
вверх

Таким образом закрашивается одна клетка. Робот исполняет в общей сложности 5 команд.

б) Вызов "закрасить прямоугольник (0,11)" сводится к исполнению таких команд:

```
нач
| нц 11 раз
| | закрасить ряд (0)
| | вниз
| кц
| вверх на (11)
кон
```

Вызов "закрасить ряд (0)" приводит к исполнению двух циклов с нулевым числом повторений. Следовательно, Робот не получит никаких команд и не сдвинется с места. Таким образом, при исполнении цикла в основном алгоритме ЭВМ выдает Роботу 11 раз команду "вниз". При вызове алгоритма "вверх на" Робот 11 раз исполняет команду "вверх".

Всего ЭВМ выдает Роботу 22 команды. Не будет выдано ни одной команды закраски, так что и закрашенные клетки не появятся.

в) При вызове "закрасить прямоугольник (9,0)" получаем:

```
нач
| нц 0 раз
| | закрасить ряд (9)
| | вниз
| кц
| вверх на (0)
кон
```

В этом случае ЭВМ не выдает Роботу ни одной команды и, естественно, закрашенные клетки не появятся.

В результате вызова "закрасить прямоугольник (9,11)" будет закрашен прямоугольник размером 9×11. Всего будет закрашено 99 клеток.

Подсчет числа команд удобно провести в общем случае для произвольных положительных m и n. Обозначим через x число команд в теле цикла, тогда nx — число команд при исполнении цикла. При исполнении алгоритма "вверх на (n)" Роботу выдается n команд. Обозначим через N общее число команд при исполнении алгоритма. Тогда N=nx+n. Найдем теперь x. При закрашивании ряда, содержащего m клеток, ЭВМ выдает Роботу 3m команд (2m команд в цикле и m команд в алгоритме "влево на (m)"). Следовательно, x=3m+1. Отсюда N=n(3m+1)+n = (3m+2)n. Для заданных m=9 и n=11 получаем N=11(39 + 2)=319.

2. а) При третьем исполнении цикла возникнет следующая ситуация. Робот закрасит ряд, примыкающий к нижней стене, и получит команду "вниз". Это приведет к отказу.

б) В этом случае отказ произойдет уже при первом исполнении цикла. Здесь нужно разобрать выполнение алгоритма "закрасить ряд (m)". Робот закрашивает m клеток, но при этом смещается на одну клетку правее закрашиваемого ряда. Значит, при третьем исполнении цикла в алгоритме "закрасить ряд", когда Роботу будет выдана команда "вправо", произойдет отказ.

3. Нужно внести такие изменения, чтобы Робот во время исполнения алгоритма не выходил за пределы закрашиваемого прямоугольника. Это можно сделать, например, так:


```
алг закрасить ряд (арг цел m)
  дано | Робот в клетке A
  надо | закрашено m клеток вправо от исходного
       | положения, Робот в клетке A
нач
| нц m−1 раз
| | закрасить; вправо
| кц
| закрасить
| влево на (m−1)
кон

алг закрасить прямоугольник (арг цел m,n)
  дано | Робот в клетке A
  надо | закрашен прямоугольник размером m×n,
       | Робот в клетке A (левый верхний угол)
нач
| нц n−1 раз
| | закрасить ряд (m)
| | вниз
| кц
| закрасить ряд (m)
| вверх на (n−1)
кон
```

4. Сначала нужно выделить фрагменты, из которых состоит ломаная — зубцы. Зубец можно нарисовать с помощью двух команд:

сместиться на вектор (a,a)
сместиться на вектор (a,−a)

Получаем:

```
алг горизонтальная ломаная (арг цел n, вещ a)
  дано | n — число зубцов, a — высота зубца, перо в начале
       | ломаной
  надо | нарисована ломаная, перо в конце ломаной и поднято
нач
| опустить перо
| нц n раз
| | сместиться на вектор (a,a)
| | сместиться на вектор (a,−a)
| кц
| поднять перо
кон
```

Теперь можно выяснить, что получается при a<0: ломаная будет нарисована справа налево и зубцы обращены вниз.

5. Зубец, повернутый на 90° по часовой стрелке, получится в результате исполнения команд

сместиться на вектор (a,—a)
сместиться на вектор (—a,—a)

Это можно установить непосредственно по чертежу. Полезно получить этот результат, используя преобразования координат при повороте на 90° по часовой стрелке: вектор с координатами (x,y) переходит в вектор с координатами (y,—x). Следовательно, вектор с координатами (a,a) заменяется вектором с координатами (a,—a). Теперь алгоритм "вертикальная ломаная" получается так же, как в предыдущем упражнении:

```
алг вертикальная ломаная (арг цел n, арг вещ a)
  дано | n — число зубцов, a — высота зубца, перо в начале
       | ломаной
  надо | нарисована ломаная, перо в конце ломаной
       | и поднято
нач
| опустить перо
| нц n раз
| | сместиться на вектор (a,—a)
| | сместиться на вектор (—a,—a)
| кц
| поднять перо
кон
```

6. Трудность этой задачи связана с тем, что после рисования ломаной перо Чертежника нужно переместить к началу следующей ломаной. В случае горизонтальной ломаной перо смещается на 2na вправо, следовательно, после рисования очередной ломаной его нужно сместить на 2na влево и на b вниз, т.е. на вектор (—2na,—b). В случае б) перо просто смещается на b вниз, т.е. на вектор (0,—b). Получаем следующие алгоритмы:

```
алг серия горизонтальных ломаных (арг цел m, n, вещ a, b)
  дано | перо в начале первой ломаной
  надо | нарисовано m горизонтальных ломаных
       | на расстоянии b одна под другой
нач
| нц m раз
| | горизонтальная ломаная (n,a)
| | сместиться на вектор (—2*n*a,—b)
| кц
кон
```



```
алг серия вертикальных ломаных (арг цел m, n, вещ a, b)
  дано | перо в начале первой ломаной
  надо | нарисовано m вертикальных ломаных
       | на расстоянии b одна под другой
нач
| нц m раз
| | вертикальная ломаная (n,a); сместиться на вектор (0,—b)
| кц
кон
```

7. Анализ исполнения алгоритма "горизонтальная ломаная" с отрицательным аргументом a (см. упражнение 4) облегчает выполнение задания. По аналогии с горизонтальной ломаной можно сделать вывод, что при отрицательном аргументе a вертикальная ломаная рисуется снизу вверх. Опираясь на это, нетрудно получить нужный рисунок.

8. Задачу можно решить для заданных конкретных размеров квадрата, но мы рассмотрим более общий случай.

Обратим внимание на сходство с задачей закрашивания прямоугольника. Требуемый алгоритм получается из алгоритма "закрасить прямоугольник" чисто редакционными изменениями. При составлении основного алгоритма воспользуемся вспомогательным алгоритмом "разрисовать полосу (m)":

```
алг разрисовать прямоугольник (арг цел m,n)
  дано | перо в левом нижнем углу прямоугольника
  надо | прямоугольник m × n заполнен картинками
нач
| нц n раз
| | разрисовать полосу (m); сместиться на вектор (0,1)
| кц
| нц n раз
| | сместиться на вектор (0,—1)
| кц
кон

алг разрисовать полосу (арг цел m)
  дано | перо в левом нижнем углу полосы
  надо | полоса m × 1 заполнена картинками
нач
| нц m раз
| | картинка; сместиться на вектор (1,0)
| кц
| нц m раз
| | сместиться на вектор (—1,0)
| кц
кон
```

Полученные нами алгоритмы можно упростить: например, заменить последний цикл на одну команду "сместиться на вектор (—m,0)".

Чтобы нарисовать 100 экземпляров картинки, сделаем вызов "разрисовать квадрат (10,10)". Получаем нужный алгоритм:

```
алг разрисовать квадрат 10×10
  дано | перо в левом нижнем углу квадрата
  надо | квадрат 10×10 заполнен картинками
нач
| разрисовать прямоугольник (10,10)
кон
```

9. В решении используются вложенные циклы. Подобная конструкция рассматривалась при построении алгоритма закраски прямоугольника. Как и в том случае, можно воспользоваться вспомогательными алгоритмами. Общая идея построения обоих алгоритмов: выделить повторяющиеся фрагменты изображения, составить основные алгоритмы, используя вызовы вспомогательных алгоритмов для построения выделенных фрагментов, и затем составить вспомогательные алгоритмы.

а) На изображении периодически повторяется пара соседних строк. В соответствии с технологией разработки алгоритма "сверху вниз" сначала составляется основной алгоритм и дается описание вспомогательного, затем составляется вспомогательный алгоритм. Таким образом, можно дать следующее решение:

```
алг шахматная доска (арг цел m, n)
  дано | Робот в левом верхнем углу поля размером 2m×2n
  надо | клетки закрашены в шахматном порядке
нач
| нц n раз
| | закрасить пару рядов (m)
| кц
кон
```

В результате исполнения вспомогательного алгоритма "закрасить пару рядов" должны закрашиваться клетки двух смежных рядов и Робот должен переходить в первую клетку следующего ряда:

```
алг закрасить пару рядов (арг цел m)
  дано | пара рядов длины 2m, Робот в левой клетке
       | верхнего ряда
  надо | клетки рядов закрашены в шахматном порядке,
       | Робот в левой клетке следующего ряда
нач
| закрасить ряд 1 (m)
| закрасить ряд 2 (m)
кон
```


Здесь мы снова действуем в соответствии с технологией разработки "сверху вниз". Приведем алгоритмы закраски рядов.

```
алг закрасить ряд 1 (арг цел m)
  дано | Робот в левой клетке ряда длины 2m
  надо | клетки ряда закрашены в шахматном порядке, начиная
       | с черной, Робот в правой клетке следующего ряда
нач
| нц m раз
| | закрасить
| | вправо
| | вправо
| кц
| вниз
| влево
кон
```

```
алг закрасить ряд 2 (арг цел m)
  дано | Робот в правой клетке ряда длины 2m
  надо | клетки ряда закрашены в шахматном порядке, начиная
       | с черной, Робот в левой клетке следующего ряда
нач
| нц m раз
| | закрасить
| | влево
| | влево
| кц
| вниз; вправо
кон
```

Аналогичным образом можно получить решение, закрашивая пары столбцов.

Заметим, что возможны и существенно иные решения. Например, можно построить алгоритм так, чтобы Робот закрашивал ряд, начинающийся с черной клетки, возвращался назад, закрашивал ряд, начинающийся с белой клетки, и т. д. Опуская подробности, приведем соответствующие алгоритмы.

```
алг шахматная доска (арг цел m, n)
нач
| нц n раз
| | закрасить ряд Ч (m)
| | возврат (m)
| | закрасить ряд Б (m)
| | возврат (m)
| кц
кон
```

```
алг закрасить ряд Ч (арг цел m)
нач
| нц m раз
| | закрасить
| | вправо
| | вправо
| кц
кон

алг закрасить ряд Б (арг цел m)
нач
| нц m раз
| | вправо
| | закрасить
| | вправо
| кц
кон

алг возврат (арг цел m)
нач
| нц 2*m раз
| | влево
| кц
| вниз
кон
```

б) Здесь закрашено m смежных столбцов, имеющих высоту по n клеток. Каждый следующий столбец смещен на одну клетку вверх и вправо по сравнению с предыдущим. Выход в начало следующего столбца мы включаем во вспомогательный алгоритм. Запишем основной алгоритм:

```
алг наклонная полоса (арг цел m, n)
  дано | Робот в нижней левой клетке полосы
  надо | закрашена наклонная полоса,
       | ширина снизу вверх — n клеток,
       | протяженность слева направо — m клеток
нач
| нц m раз
| | закрасить столбец (n)
| кц
кон
```

В результате исполнения вспомогательного алгоритма "закрасить столбец" Робот должен закрасить столбец высотой n клеток и перейти в нижнюю клетку соседнего столбца.


```
алг закрасить столбец (арг цел n)
  дано | Робот в нижней клетке столбца
  надо | закрашен столбец высотой n клеток, Робот в
       | нижней клетке следующего столбца
нач
| нц n раз
| | закрасить
| | вверх
| кц
| нц n—1 раз
| | вниз
| кц
| вправо
кон
```

§ 9. АЛГОРИТМЫ С "ОБРАТНОЙ СВЯЗЬЮ". КОМАНДА пока

Сама история подтвердила малую эффективность административно-командного управления.

из газет

В § 9 изучается команда повторения **пока**. При изучении этой команды рассматривается важная идея обратной связи, передачи информации от исполнителя к ЭВМ. Использование обратной связи позволяет составлять гибкие алгоритмы управления исполнителями. Идея обратной связи раскрывается на примере Робота (к пяти ранее изученным командам добавляется еще 12 команд для реализации связи с ЭВМ). В команде **пока** применяется операция проверки условий. Смысл условий и их запись на алгоритмическом языке систематически изучаются в § 10. В примерах § 9 (за исключением п. 9.14 и некоторых упражнений) используются только простые условия. Тем не менее целесообразно при изучении команды **пока** рассмотреть и составные условия (п.10.6 из §10 учебника).

Основные цели. Ввести команду повторения **пока**; на примерах показать ее применение. Ввести понятия обратной связи, простых и составных условий, показать способы применения условий в команде **пока**.

Требования к знаниям и умениям. Учащиеся должны знать общий вид команды повторения **пока**, ее свойства и схему исполнения. Учащиеся должны уметь записывать команду повторения **пока** с командами Робота; составлять алгоритмы с командой повторения **пока**, подобные алгоритмам § 9. Учащиеся должны понимать смысл составных условий и способ их записи; уметь записывать и использовать составные условия, содержащие 1—2 простых условия.

ОСНОВНЫЕ ПОНЯТИЯ

Понятие ***обратной связи*** затрагивалось еще в § 4 при рассмотрении схемы взаимодействия человека, ЭВМ и исполнителя. Практически обратная связь исполнителя с ЭВМ реализуется с помощью команд специального вида.

Таким образом, различается два вида команд: ***команды-действия и команды обратной связи.***

Все изученные ранее команды относятся к командам-действиям. При их вызове ЭВМ передает исполнителю указания выполнить некоторые действия. При этом ЭВМ не получает от исполнителя никакой информации. (Последнее утверждение не совсем точно. Обычно исполнитель передает ЭВМ сигнал успешного выполнения команды или сообщение об отказе.)

При вызове команды обратной связи исполнитель выполняет соответствующее действие и передает ЭВМ некоторую информацию. Напомним, что нас интересует результат, а не механизм выполнения команды. *Результат выполнения команды обратной связи — информация, полученная ЭВМ.* Эта информация представлена в виде величины определенного типа.

Сформулируем некоторые свойства команды **пока**:

1) условие проверяется только *перед* исполнением тела цикла, но не проверяется в процессе выполнения (п. 9.9);

2) если условие не соблюдается с самого начала, тело цикла не выполняется ни разу (п. 9.7);

3) после завершения команды **пока**, независимо от числа повторений, условие не соблюдается;

4) исполнение команды **пока** может не завершиться (п. 9.8).

Остановимся на основном отличии команды **пока** от других команд повторения.

Число повторений в циклах n **раз** и **для** (цикл **для** будет рассмотрен в § 13) определяется по заголовку и, следовательно, точно известно в момент начала выполнения цикла. Число повторений в цикле **пока** по заголовку не определяется и становится известным только после выполнения цикла.

ОБЩИЕ УКАЗАНИЯ

При изучении параграфа учащиеся сталкиваются сразу с несколькими новыми объектами: новая команда, команды обратной связи, условия и их проверка.

Объединяющую роль играет идея обратной связи. Значимая сама по себе, она становится также методическим средством освоения сложной конструкции алгоритмического языка. В § 9 в командах повторения **пока** в качестве условий везде фигурируют вызовы команд Робота (только в п. 9.13 вызов команды сочетается с отношением между величинами). Это позволяет

анализировать команду повторения **пока** и ее свойства в форме естественного и наглядного диалога между ЭВМ и Роботом. Здесь удается четко разделить две функции: формирование ответа в зависимости от ситуации и реакция на ответ. Первую реализует исполнитель, в данном случае — Робот, вторую ЭВМ.

Форма описания команд обратной связи (с указанием типа) согласуется с формой описания алгоритмов-функций (см. пп. 12.8 —12.10).

Материал § 9 позволяет показать сходство всех команд повторения. Каждая из них содержит серию повторяемых команд, называемую телом цикла. В каждой из них в той или иной форме дается указание о числе повторений. После изучения команды присваивания и цикла **для** можно показать, что цикл **пока** является в определенном смысле наиболее общей из изученных команд повторения. Используя команду **пока**, сравнительно несложно реализовать команды повторения **для** и n **раз**.

В п. 9.14 впервые появляется составное условие. Имеет смысл сразу разобрать и другие составные условия (п. 10.6). Это обеспечит преемственность при переходе к командам ветвления и контроля в §10.

Трудность изучения связок **и**, **не**, **или** находится в прямой зависимости от их соответствия языковым эквивалентам. Так, смысл связки **и** в условии "А **и** Б" практически полностью совпадает с обычным употреблением союза "и".

В то же время смысл связки **или** в условии "А **или** Б" заметно отличается от употребительного смысла союза "или" в естественном языке. В алгоритмическом языке **или** имеет соединительный смысл: условие "А **или** Б" выполняется, если выполняется хотя бы одно из условий А **или** Б. В обиходном же смысле условия А и Б обычно являются взаимоисключающими ("Быть или не быть?").

При разборе алгоритма А34, использующего отношения величин (п. 9.13), полезно применить диалог, подобный диалогу из п. 9.9. Например, при a=5:

ЭВМ: уровень радиации?	***Робот:*** 3
ЭВМ: вниз	***Робот:*** смещается вниз
ЭВМ: уровень радиации?	***Робот:*** 6

Поскольку при последнем ответе условие "6 < a" не соблюдается, исполнение цикла, а вместе с ним и алгоритма заканчивается.

При построении алгоритмов с использованием команды **пока** могут быть введены элементы доказательных рассуждений. Это поможет реализовать развивающий потенциал курса, установить связи с математикой. В информатике как научной ди-

сциплине доказательства, в частности доказательства правильности алгоритмов, играют почти такую же важную роль, как и в математике. Систематическое применение доказательств в учебном курсе информатики связано с серьезными трудностями методического характера и не в полной мере согласуется с целями курса. В то же время введение элементов доказательных рассуждений оправдано и целесообразно. Некоторые примеры такого рода рассуждений даются при разборе упражнений.

Доказательный анализ алгоритма может проводиться как до, так и после непосредственного прослеживания его исполнения. Первое существенно труднее и требует от учащихся основательной математической культуры.

Доказательные рассуждения особенно необходимы при построении алгоритмов с командой **пока**. Это связано с тем, что алгоритмы с командой повторения **пока** во многих случаях обладают большим (иногда бесконечным) набором исходных ситуаций, в которых они применимы (например, исходная ситуация может определяться только тем, что где-то справа от Робота имеется стена и т.п.). Получение требуемого результата после исполнения алгоритма во всех этих ситуациях может оказаться неочевидным, а потому нуждается в обосновании. Для проведения доказательных рассуждений нужно использовать свойства команды повторения **пока**, перечисленные выше.

Уровни строгости и доказательности, естественно, должны варьироваться в зависимости от математической подготовки учащихся.

РЕКОМЕНДАЦИИ К ВЕДЕНИЮ УРОКОВ

Для объяснения необходимости цикла **пока** воспользуемся уже рассмотренным методическим приемом: составим алгоритм решения простейшей задачи, а затем будем постепенно расширять область применения алгоритма. Пусть Робот стоит в трех шагах левее стены. Нужно подвести его к стене (рис. 32).

```
алг три шага до стены
  дано | Робот в трех шагах
       | левее стены
  надо | Робот возле стены
нач
| вправо
| вправо
| вправо
кон
```

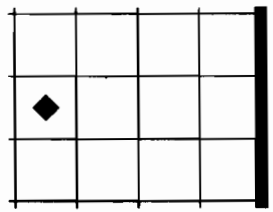


Рис. 32

Теперь расширим область применения алгоритма, считая, что расстояние до стены может меняться:


```
алг до стены (арг цел a)
  дано a≥0 | Робот левее стены на расстоянии a шагов
  надо | Робот возле стены
нач
| нц a раз
| | вправо
| кц
кон
```

Усложним задачу. Пусть расстояние до стены неизвестно (мы не видим поле Робота).

Чтобы управлять Роботом вслепую, необходимо каким-то способом *получать от него информацию о состоянии среды*. Например, Робот должен уметь подсчитывать и сообщать расстояние до стены. Или более простой (для изготовителя) вариант: "научить" Робота определять наличие стены рядом с ним. Создатели Робота выбрали второй путь. Значит, придется перед каждым шагом спрашивать Робота: "справа свободно?", и пока он будет отвечать утвердительно, двигать его вправо:

```
алг направо до стены
  дано | стена где-то справа
  надо | Робот возле стены
нач
| нц пока справа свободно
| | вправо
| кц
кон
```

Для получения информации от Робота надо установить с ним ***обратную связь***. Мы уже говорили о возможности обратной связи с исполнителем, но потребность в ней возникла только теперь.

"Справа свободно" — это *команда* Роботу проверить, нет ли рядом с ним стены (с правой стороны), и передать результат по обратной связи. Робот ответит либо **да**, либо **нет**.

В отличие от ранее рассмотренных команд, результат выполнения команды "справа свободно" не *действие* Робота, а *величина*. Она может принимать всего два значения, которые мы обозначаем служебными словами **да** и **нет**. Это пример величины *логического типа* (**лог**). ЭВМ подставляет полученное от Робота значение в алгоритм вместо команды "справа свободно".

В о п р о с . Какие *типы* величин мы знаем?

О т в е т . Мы знаем типы **вещ**, **цел** и теперь — **лог**.

В о п р о с . Какие *виды* величин мы прошли?

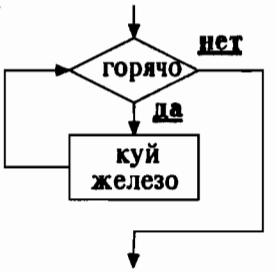
О т в е т . Аргументы и промежуточные величины.

Если мы вводим в алгоритм произвольную величину неизвестного ЭВМ типа и вида, то обязаны описать ее. Если же эта величина — результат стандартного действия, то описывать ее не надо. Поэтому результат команды "справа свободно" не описывается.

Робот имеет еще девять команд с результатом типа лог (п. 9.1 на с. 63 учебника). Итак, мы теперь знаем пятнадцать команд Робота (остались две неизученные).

Разберемся, как работает команда повторения пока. Разумеется, эта команда составная. Как обычно, рассмотрим простейший алгоритм. Пусть это будет алгоритм управления каким-то неизвестным исполнителем Гефест:

```
алг ковка железа
  дано | имеется болванка
  надо | получена заготовка
нач
| нц пока горячо
| | куй железо
| кц
кон
```



Выполнение цикла в этом алгоритме представляется схематически (схемы алгоритмов, аналогичные приведенной выше, изредка используются программистами как способ записи алгоритмов).

После слова пока в алгоритме "ковка железа" (и внутри ромба в схеме алгоритма) стоит команда "горячо". Ее результат — величина типа лог. Эту логическую величину можно назвать *условием входа в цикл*. Если условие соблюдается (т.е. значение величины "горячо" равно да), то вход в цикл разрешен. (Одновременно слово "горячо" служит командой исполнителю — проверить, горячо ли, и сообщить результат по обратной связи.) Как и в любой команде цикла, после слова кц происходит возврат к заголовку цикла (нц) и вновь проверяется условие входа в цикл.

В командах n раз число повторений известно *перед* выполнением цикла. В команде пока число повторений выясняется лишь после ее выполнения. Поэтому эту команду называют иногда ***циклом с неизвестным числом повторений.***

При выполнении команды цикла пока возможны три случая.

1. Условие входа в цикл не соблюдается (имеет значение нет) при первой же проверке. Тогда серия команд в теле цикла не выполнится ни разу (нельзя ковать холодное железо).

Выполняя алгоритм "направо до стены", в обстановке, показанной на рисунке 33, ЭВМ не даст Роботу ни одной команды.

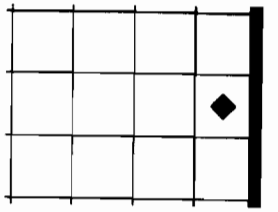


Рис. 33

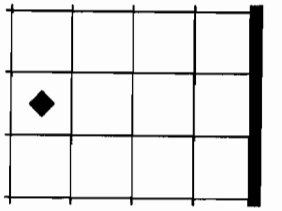


Рис. 34

2. Условие входа в цикл сначала соблюдается (имеет значение да), а в процессе выполнения серии команд цикла изменяется на обратное (железо остывает).

Выполняя алгоритм "направо до стены", в обстановке, показанной на рисунке 34, ЭВМ даст Роботу три команды "вправо".

3. При входе в цикл и при каждой следующей проверке условие соблюдается (имеет значение да). Для алгоритма "направо до стены" такой случай возникнет при отсутствии стен справа от Робота (это не соответствует условию задачи).

Произойдет "зацикливание", и Робот будет бесконечно двигаться вправо.

Еще пример:

```
нц пока волк в лес смотрит
| корми
кц
```

Известно, что кормление волка не может изменить направление его взгляда (в данном случае — лес). Такой цикл тоже будет повторяться вечно.

З а д а н и е. Сформулируйте задачу, требующую применить цикл пока при управлении Чертежником.

О т в е т. Этого сделать нельзя, потому что Чертежник не имеет команд обратной связи.

Рассмотрим задачу. Слева направо тянется бесконечная стена, имеющая один проход неизвестной ширины. Робот стоит левее прохода в клетке, прилегающей к стене сверху (частный случай

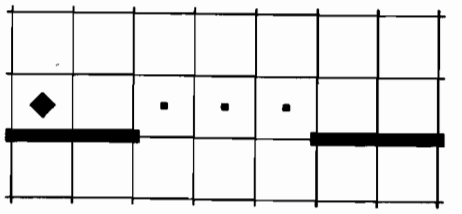


Рис. 35

изображен на рисунке 35). Нужно закрасить клетки с верхней стороны от прохода и освободить проход (отойти в сторону).

```
алг маркировка прохода
  дано | снизу стена, проход где-то правее
  надо | клетки, примыкающие к проходу
       | с верхней стороны, закрашены (рис. 35)
нач
| найти проход
| отметить проход
кон

алг найти проход
  дано | снизу стена, проход где-то правее
  надо | Робот с верхней стороны
       | возле прохода
нач
| нц пока снизу стена
| | вправо
| кц
кон

алг отметить проход
  дано | Робот с верхней стороны, возле прохода
  надо | клетки, примыкающие к проходу
       | с верхней стороны, закрашены
нач
| нц пока снизу свободно
| | закрасить
| | вправо
| кц
кон
```

При решении задачи можно было обойтись без последовательного уточнения. Но во-первых, этот метод позволил свести задачу к более легким и записать решение просто и понятно. Во-вторых, появилась возможность проследить, как выход (надо) одного алгоритма становится входом (дано) другого. А в-третьих, алгоритм "найти проход" можно будет использовать при решении задачи 11 на с. 74.

Можно придумать много различных применений Робота. Он может побывать на других планетах, демонстрировать ходы шахматных фигур, маркировать клетки для последующей работы каких-нибудь неизвестных исполнителей, заменять заболевшего пони.

Но, по-видимому, имеет смысл использовать Робота там, где без автоматических устройств не обойтись. Например, при работе в радиоактивной местности, на лютом холоде или сильной жаре. Примерно такие рассуждения привели создателей Робота к мысли о необходимости введения в его СКИ команд сообщения по обратной связи значений температуры и радиации.

Рассмотрим задачу:

Внутри прямоугольника неизвестного размера, огороженного стенами, произошла авария на атомной электростанции (рис. 36,клетка Б). Зона повышенной радиации (выше а) расположена вдали от стен прямоугольника и "выпукла" (радиация убывает по мере удаления от эпицентра взрыва). Робот находится в левом верхнем углу прямоугольника. Требуется закрасить все клетки вне радиоактивной зоны. При этом Робот не может заходить в глубь радиоактивной зоны более чем на один шаг, так как в районе эпицентра уровень радиации опасен даже для Робота.

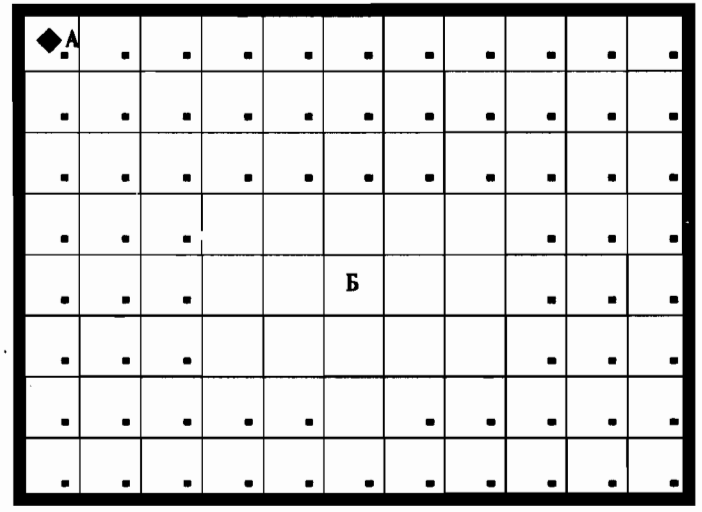


Рис. 36

Приведем одно из возможных решений. Будем перемещать Робота по периметру прямоугольника и в каждой клетке двигаться от стены до зоны (либо до противоположной стены).

```
алг закраска клеток вне зоны
  дано | Робот в левом верхнем углу прямоугольника
  надо | клетки вне зоны закрашены, Робот в исходном
       | положении (рис. 36)
нач
| обработка вправо; обработка вниз
| обработка влево; обработка вверх
кон
```

```
алг обработка вправо
  дано | Робот в левом верхнем углу прямоугольника
  надо | нужная часть клеток вне зоны закрашена,
       | Робот в правом верхнем углу
нач
| нц пока справа свободно
|  вправо; закраска вниз
| кц
кон

алг закраска вниз
  дано | Робот у верхней стены
  надо | все клетки между Роботом и зоной (или нижней
       | стеной) закрашены, Робот в исходном положении
нач
| нц пока снизу свободно и радиация ≤ a
|  закрасить; вниз
| кц
| вверх до стены
кон
```

Алгоритмы "обработка вниз", "обработка влево" и "обработка вверх" записываются аналогично.

Запишем в словарь:

```
нц пока <условие>        — цикл с неизвестным
| <серия команд>          числом повторений
кц
```

Приводим примерные варианты заданий проверочной работы №2.

Задание 1

Укажите конечное положение Робота и клетки, которые будут закрашены, после выполнения следующего алгоритма (рис. 37):

Вариант 1

```
нц пока радиация < 7
| вправо на (4); закрасить; влево на (2)
кц
```

Вариант 2

```
нц пока радиация < 7
| вправо на (4); закрасить; влево на (6)
кц
```


В клетках указаны значения радиации.

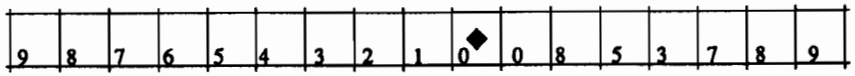


Рис. 37

Задание 2

Составьте алгоритм изображения "Такси" и "Забор" с помощью Чертежника заданного рисунка 38 переменных размеров.

Вариант 1 Вариант 2

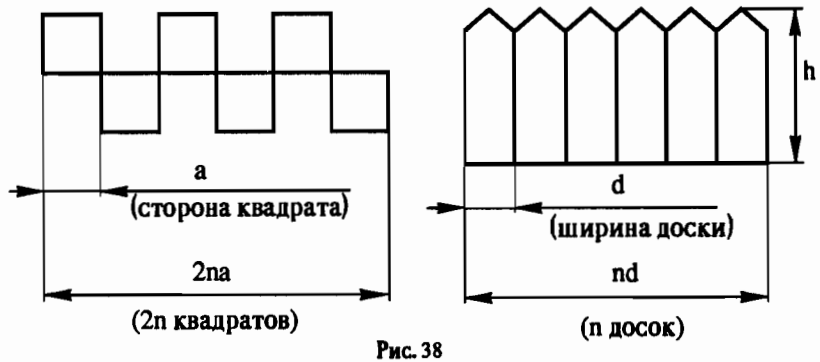


Рис. 38

Задание 3

Вариант 1

Робот находится где-то внутри прямоугольника неизвестного размера, образованного закрашенными клетками (в клетке А). Стен на поле нет. Требуется перевести Робота в левый верхний угол прямоугольника (клетку Б). Частный случай представлен на рисунке 39.

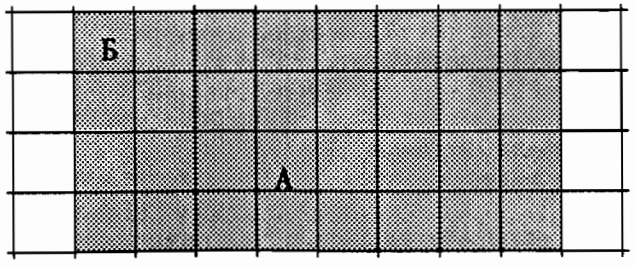


Рис. 39

Вариант 2

Робот в клетке А. Где-то справа от Робота имеется прямоугольное поле закрашенных клеток неизвестного размера. Требуется пройти поле и остановиться возле его правого края в клетке Б (частный случай показан на рисунке 40).

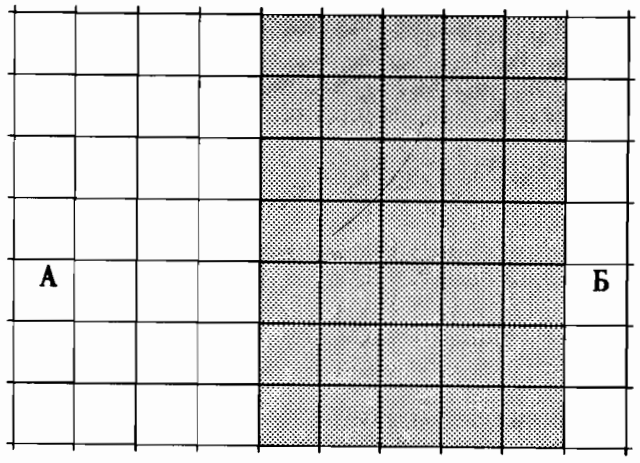


Рис. 40

УКАЗАНИЯ К УПРАЖНЕНИЯМ

1. При исполнении приведенной в упражнении команды повторения окажутся закрашенными все клетки левее конечного положения Робота; клетка, в которой будет расположен Робот, останется незакрашенной. Поэтому после команды повторения нужно поставить вызов команды закраски клетки. Получаем алгоритм:

```
алг закрасить ряд вправо и вернуться (А32)
  дано | Робот стоит у левой стены внутри огороженного с
       | четырех сторон прямоугольника (рис.41 учебника)
  надо | горизонтальный ряд, в левой клетке которого стоял Ро-
       | бот, полностью закрашен; Робот в исходном положении
нач
| закрасить
| нц пока справа свободно
| | закрасить; вправо
| кц
| закрасить | полоса закрашена, и Робот в ее правой клетке
| влево до стены
кон
```


2. План действий: Робот поочередно закрашивает ряды сверху вниз, пока не закрасит последний ряд, идущий вдоль стены. Затем Робот возвращается в исходное положение. Получаем:

```
алг закрасить прямоугольник
  дано | Робот стоит в левом верхнем углу внутри
       | огороженного с четырех сторон прямоугольника
  надо | весь прямоугольник закрашен,
       | Робот в исходном положении
нач
| закрасить ряд вправо и вернуться
| нц пока снизу свободно
| | вниз
| | закрасить ряд вправо и вернуться
| кц
| вверх до стены
кон
```

3. План решения: вначале переместить Робота в левый верхний угол, а затем исполнить тот же алгоритм, что и в предыдущем упражнении. Используя алгоритм "сместиться к углу" (А36) в качестве вспомогательного, получаем:

```
алг закрасить прямоугольник 1
  дано | Робот стоит внутри огороженного с четырех
       | сторон прямоугольника
  надо | весь прямоугольник закрашен,
       | Робот в левом верхнем углу
нач
| сместиться к углу;
| закрасить прямоугольник
кон
```

4. Предлагаемые упражнения удобно решать, используя диалог ЭВМ с Роботом.

а) В этом упражнении в диалог ЭВМ с Роботом нужно включить и действия ЭВМ. Рассмотрим случай, когда Робот стоит в закрашенной клетке (ситуация (1)):

ЭВМ: клетка закрашена? ***Робот:*** да
ЭВМ: меняет да на нет

Исполнение цикла прекращается, поскольку нарушено условие повторения.

ЭВМ: клетка закрашена? ***Робот:*** нет
ЭВМ: меняет нет на да
ЭВМ: закрасить ***Робот:*** закрашивает клетку
ЭВМ: клетка закрашена? ***Робот:*** да
ЭВМ: меняет да на нет

Исполнение цикла заканчивается, поскольку нарушено условие повторения. Заметим, что, после того как Робот закрашивает клетку, ситуация (2) превращается в ситуацию (1), поэтому последние две строчки диалога в ситуации (2) — это повторение диалога в ситуации (1). Теперь несложно дать ответ на вопрос, поставленный в упражнении. В ситуации (1) ЭВМ выдает Роботу одну команду-вопрос "клетка закрашена". В ситуации (2) — 3 команды:

клетка закрашена
закрасить
клетка закрашена

б) В ситуации (1) возникает диалог:

ЭВМ: клетка закрашена? ***Робот***: да
ЭВМ: закрасить ***Робот***: закрашивает клетку
ЭВМ: клетка закрашена? ***Робот***: да

И далее, поскольку условие повторения соблюдается, команды будут повторяться сколь угодно много раз, т. е. ЭВМ "зациклится".

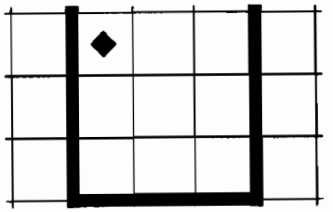
Полезно обратить внимание учащихся на то, что вопрос о "зацикливании" можно было бы решить без выписывания диалога, опираясь лишь на свойство (4), п. 9.11, команды пока: в случае (1) условие соблюдается и исполнение команд из тела цикла не приводит к его нарушению (более того, оно вообще не меняет ситуацию).

В случае (2) условие повторения не соблюдается, поэтому ЭВМ, выдав команду-вопрос и получив ответ Робота:

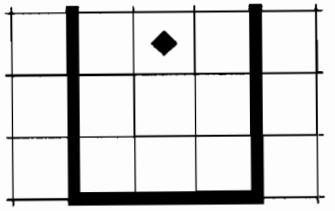
ЭВМ: клетка закрашена? ***Робот:*** нет, прекращает исполнение цикла.

5. В случае а) условие повторения не соблюдается, поэтому, выдав команду-вопрос и получив отрицательный ответ, ЭВМ прекращает исполнение цикла.

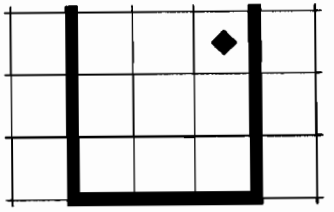
Случай б) можно сначала разобрать, опираясь на общие свойства команды повторения пока, а потом проверить полученный результат с помощью диалога. При движении Робота вправо условие "сверху свободно" будет соблюдаться. Следовательно, исполнение команды пока не завершится. Тогда возможны два исхода: либо ЭВМ зациклится, либо при исполнении цикла возникнет отказ. Первое возможно лишь в том случае, если, двигаясь вправо, Робот не наткнется на стену. В рассматриваемом случае это не так. Следовательно, исполнение команды пока завершится отказом. Рассмотрим диалог:



ЭВМ: сверху свободно? ***Робот:*** да
ЭВМ: вправо ***Робот:*** смещается вправо



ЭВМ: сверху свободно? ***Робот:*** да
ЭВМ: вправо ***Робот:*** смещается вправо

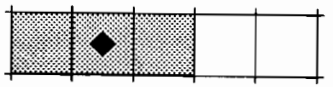


ЭВМ: сверху свободно? ***Робот:*** да
ЭВМ: вправо ***Робот:*** отказ

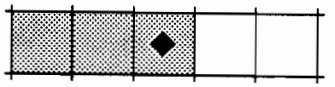
6. Сначала нужно наметить план решения. Например, можно принять такой план: переместить Робота вправо до правой границы закрашенного прямоугольника и затем вниз до нижней границы (полезно рассмотреть алгоритмы, полученные в результате реализации иных планов). Получается следующий алгоритм:

```
алг в правый нижний угол прямоугольника
  дано | Робот находится внутри закрашенного
       | прямоугольника, на поле Робота стен нет
  надо | Робот находится в правом нижнем углу
       | закрашенного прямоугольника
нач
| нц пока клетка закрашена
| | вправо
| кц
| влево
| нц пока клетка закрашена
| | вниз
| кц
| вверх
кон
```

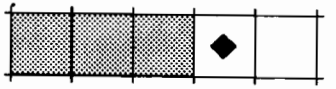
Здесь нуждаются в пояснении команды "влево" и "вверх", поставленные после циклов. После завершения первой команды **пока** Робот будет находиться в незакрашенной клетке.Это первая незакрашенная клетка, в которой Робот окажется при движении вправо. Чтобы вернуть Робота в закрашенный прямоугольник, нужно переместить его на одну клетку влево. Аналогично объясняется появление команды "вверх". Сказанное можно проиллюстрировать диалогом ЭВМ и Робота. Рассмотрим ситуацию, возникающую при движении Робота направо:



ЭВМ: клетка закрашена? ***Робот:*** да
ЭВМ: вправо ***Робот:*** смещается вправо



ЭВМ: клетка закрашена? ***Робот:*** да
ЭВМ: вправо ***Робот:*** смещается вправо



ЭВМ: клетка закрашена? ***Робот:*** нет

На этом исполнение цикла заканчивается.

Заметим, что при исполнении алгоритма Робот переместится в нижний правый угол прямоугольника независимо от своего исходного положения: важно только, чтобы он находился внутри закрашенного прямоугольника.

Информация о том, что Робот находится в левом верхнем углу, является избыточной.

7.а) Мы предполагаем, что в исходном положении Робот стоит в незакрашенной клетке.

```
алг закрасить до стены вправо и вернуться
  дано | где-то правее Робота есть стена
  надо | закрашен ряд клеток между Роботом и стеной
       | (рис.44 учебника), Робот в исходном положении
нач
| нц пока справа свободно
| | вправо
| | закрасить
| кц
|     | Робот у стены, клетки закрашены
| влево до незакрашенной клетки
кон

алг влево до незакрашенной клетки
  дано | слева от Робота есть незакрашенная клетка,
       | между Роботом и этой клеткой стен нет
  надо | Робот сместился влево, в первую незакрашенную
       | клетку; если Робот уже в незакрашенной клетке,
       | то положение Робота не изменилось
нач
| нц пока клетка закрашена
| | влево
| кц
кон
```

В этой задаче главное сообразить, что если Робот красит клетки по дороге "туда" (к стенке), то на "обратном" пути исходную клетку он может отличить от остальных — это первая (на обратном пути) незакрашенная клетка. Для возвращения Робота в исходное положение удобно ввести вспомогательный алгоритм "влево до незакрашенной клетки" — это делает алгоритм более понятным и упрощает решение остальных пунктов упражнения.

б)

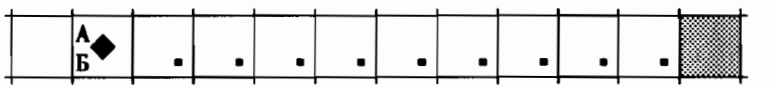


Рис. 41

```
алг закрасить до закрашенной клетки вправо и вернуться
  дано | где-то правее Робота есть закрашенная клетка
  надо | закрашен ряд клеток между Роботом и этой клеткой,
       | Робот в исходном положении (рис.41)
нач
| вправо
| нц пока клетка не закрашена
| | закрасить; вправо
| кц
|     | Робот в клетке, которая была закрашена с самого на-
|     | чала; клетки, отмеченные точками (рис. 41), закрашены
| влево до незакрашенной клетки
кон
```

Внимание! Типичная ошибка в этой задаче — использование цикла

```
нц пока клетка не закрашена
| вправо; закрасить
кц
```

Если ученики такой ошибки не допустят, их следует явно спросить, что произойдет при использовании такого цикла, каков будет диалог ЭВМ — Робот. В любом случае эту ошибку надо подробно разобрать, объяснив, что Робот закрасит только одну клетку.

в)

```
алг закрасить коридор без исходной клетки
  дано | Робот где-то в горизонтальном коридоре
  надо | закрашены все клетки коридора, кроме стартовой
       | (клетки А), Робот в исходном положении (рис. 42)
нач
| вправо
| нц пока снизу стена
| | закрасить; вправо
| кц
| влево
|     | Робот в правой клетке коридора; клетки коридора
|     | справа от исходной закрашены
| влево до незакрашенной клетки
|     | Робот в исходном положении
| нц пока снизу стена
| | закрасить; влево
| кц
| вправо
|     | Робот в левой клетке коридора; все клетки,
|     | отмеченные точками (рис. 42), закрашены
| вправо до незакрашенной клетки
кон
```


Рис. 42

Обратите внимание, что перед вызовом вспомогательного алгоритма "влево до незакрашенной клетки" мы заставляем Робота сделать шаг влево, чтобы он попал в закрашенную клетку. Вспомогательный алгоритм "вправо до незакрашенной клетки" аналогичен алгоритму "влево до незакрашенной клетки". В этой задаче следует проверять, что предлагаемые учениками решения правильно работают во всех ситуациях, например, когда исходная клетка является крайней клеткой коридора.

г)

Рис. 43

Эта задача отличается от в) только тем, что и исходную клетку тоже надо закрасить. Поэтому ее можно решить так:

```
алг закрасить коридор
  дано | Робот где-то в горизонтальном коридоре
  надо | закрашены все клетки коридора (рис. 43),
       | Робот в исходном положении (рис. 43)
нач
| закрасить коридор без исходной клетки; закрасить
кон
```

Следует обратить внимание, что переставить эти две команды местами нельзя, так как во вспомогательном алгоритме "закрасить коридор без исходной клетки" существенным является то, что исходная клетка не закрашена.

д)

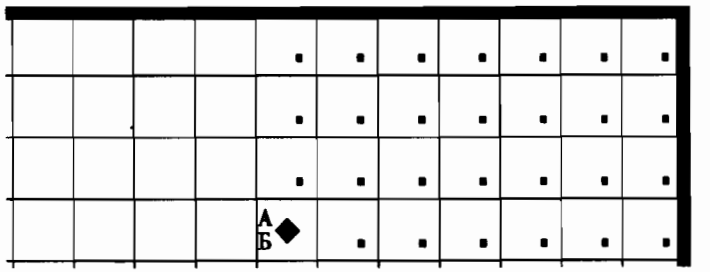


Рис. 44

```
алг закрасить угол
  дано | Робот внутри прямоугольника, огороженного
       | стенами
  надо | закрашены все клетки правее и выше стартовой,
       | Робот в исходном положении (рис. 44)
нач
| закрасить до стены вправо и вернуться
| нц пока сверху свободно
| | вверх; закрасить до стены вправо и вернуться
| | закрасить
| кц
| вниз до незакрашенной клетки
кон
```

Здесь в качестве вспомогательных используются алгоритмы "закрасить до стены вправо и вернуться" (пункт а) и "вниз до незакрашенной клетки" (он аналогичен алгоритму "влево до незакрашенной клетки").

Решения учеников следует проверить в ситуации, изображенной на рисунке 45.

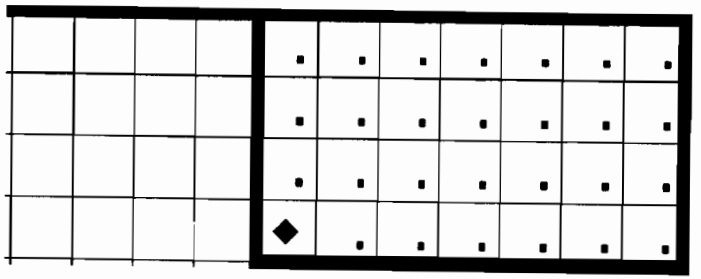


Рис. 45

8. При составлении алгоритмов полезно вначале давать их словесные описания, указывая проверяемые условия и выдаваемые Роботу команды. Например, в случае а) словесное описание алгоритма может выглядеть так: Робот движется вдоль стены вправо до тех пор, пока она не кончится; затем обходит ее, делая шаги вниз и влево. Далее полученное словесное описание формализуется. Слова "пока стена не кончится" на алгоритмическом языке переписываются в виде "пока снизу стена" и т. д. Приведем соответствующие алгоритмы.

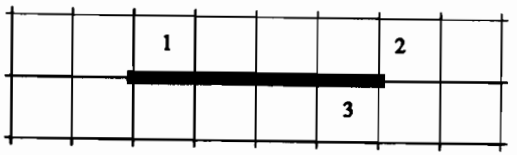


Рис. 46

а)
```
алг обход стены
нач
| нц пока снизу стена
| | вправо
| кц
| вниз; влево
кон
```

На рисунке 46 цифрами обозначены положения Робота: 1 — исходное, 2 — после исполнения цикла, 3 — конечное.

б)
```
алг закрасить полосу сверху от стены
нач
| нц пока снизу стена
| | влево
| кц
| вправо
| нц пока снизу стена
| | закрасить
| | вправо
| кц
кон
```

В этом алгоритме первый цикл переводит Робота к левому краю стены. При исполнении второго цикла Робот двинется вдоль стены вправо, закрашивая клетки. На рисунке 47 цифрами обозначены положения Робота: 1 — исходное, 2 — после исполнения первого цикла, 3 — перед исполнением второго цикла, 4 — после исполнения второго цикла (конечное положение). За-крашиваемые клетки отмечены.

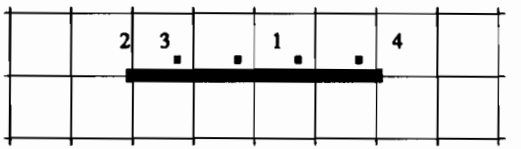


Рис. 47

Полезно разобрать ошибочное решение задачи, это поможет лучше понять свойства команды пока. Рассмотрим, что произойдет, если в предыдущем алгоритме во втором цикле переставить местами команды "закрасить" и "вправо". Закрашиваемые клетки отмечены на рисунке 48 точками. Ответ можно получить не только исполняя алгоритм, но и путем рассуждений.

При исполнении ошибочного алгоритма Робот перемещается точно так же, как и при исполнении правильного. Число закра-

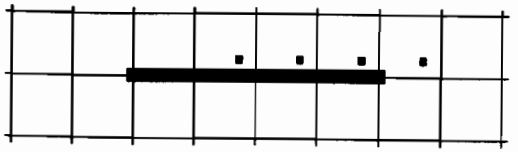


Рис. 48

шенных клеток равно числу повторений во втором цикле, т.е. длине стены. Последняя исполняемая команда алгоритма — "закрасить". Следовательно, клетка 4 на рисунке 48 будет закрашена. Таким образом, в ошибочном алгоритме закрашиваемая полоса смещена на одну клетку вправо по сравнению с полосой, закрашиваемой в правильном алгоритме.

в) Сначала нужно наметить план действий. Робот обходит стену (как в алгоритме а)) и затем, двигаясь вдоль стены, закрашивает клетки так, как это сделано в случае б). Используя в качестве вспомогательного алгоритм "обход стены", получаем:

```
алг закрасить полосу снизу от стены
  дано |
  надо |
нач
| обход стены
| нц пока сверху стена
| | закрасить
| | влево
| кц
кон
```

На рисунке 49 цифрами обозначены положения Робота: 1 — исходное, 2 — после исполнения алгоритма "обход стены" (сравните с рис. 48), 3 — конечное. Закрашиваемые клетки отмечены точками.

г) В этом задании алгоритм может быть составлен с использованием алгоритмов из предыдущих случаев в качестве вспомогательных. Достаточно сравнить рисунки 48, 49. Таким образом, получаем:

```
алг закрасить прилегающие к стене клетки
  дано |
  надо |
нач
| закрасить полосу снизу от стены
| вверх; вправо
| закрасить полосу сверху стены
кон
```

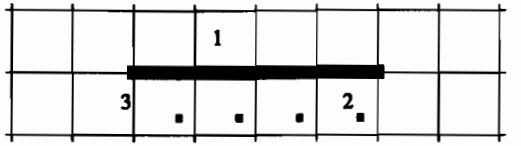



Рис. 49

В составленном алгоритме четыре вызова. На рисунке 50 цифрами обозначены положения Робота: 1 — исходное, 2 —после исполнения алгоритма "закрасить полосу снизу от стены", 3 и 4 — до и после исполнения алгоритма "закрасить полосу сверху от стены".

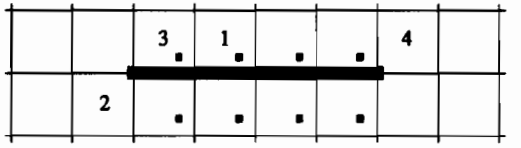


Рис. 50

Возможен другой план: Робот движется вдоль стены сверху , закрашивая клетки, обходит стену, движется вдоль стены снизу, снова обходит стену и движется вдоль стены сверху, пока не встретит закрашенную клетку. Приведем соответствующий алгоритм:

```
алг закрасить прилегающие к стене клетки
  дано | Робот находится рядом со стеной, выше
       | стены; стена идет горизонтально
  надо | закрашены прилегающие к стене клетки
нач
| нц пока снизу стена
| | закрасить; вправо
| кц
| вниз; влево
| нц пока сверху стена
| | закрасить; влево
| кц
| вверх; вправо
| нц пока клетка не закрашена
| | закрасить; вправо
| кц
кон
```

На рисунке 51 цифрами обозначены положения Робота: 1 — исходное, 2 — после исполнения первого цикла, 3 и 4 — до и

после исполнения второго цикла, 5 и 6 — до и после исполнения третьего цикла.

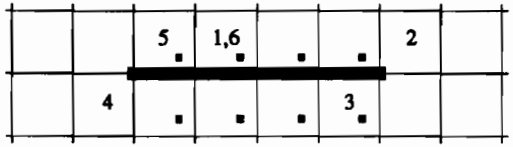


Рис. 51

Проиллюстрируем типичные методы оценки эффективности алгоритмов. Оценим эффективность числом шагов, которые делает Робот, исполняя алгоритм. Пусть n — протяженность стены. Рассмотрим рисунок 52. Пусть Робот в исходном положении находится в клетке с номером x. Подсчитаем, сколько шагов делает Робот, исполняя первый и второй алгоритмы.

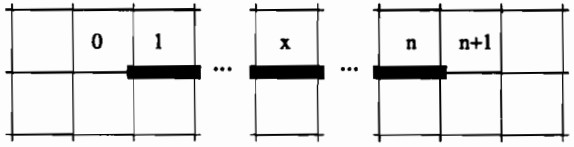


Рис. 52

При исполнении первого алгоритма Робот сначала переходит из клетки x в клетку 0, делая при этом x шагов, затем проходит вдоль стены вправо n+1 шаг, делает шаг вниз, проходит вдоль стены влево n+1 шаг. Всего $p = x + (n+1) + 1 + (n+1) = 2n+3+x$ шагов.

При исполнении второго алгоритма Робот проходит из клетки x в клетку n+1 через (n+1)—x шагов, делает шаг вниз, проходит вдоль стены влево через n+1 шаг, делает шаг вверх, проходит из клетки 0 в клетку x через x шагов. Всего $q = (n+1) - x + 1 + (n+1) + 1 + x = 2n+4$ шагов.

Следовательно, при исполнении первого алгоритма Робот делает на $p - q = (2n+3+x) - (2n+4) = x-1$ шагов больше. Если считать, что Робот находится около середины стены, т.е. $x \approx n/2$, то $p - q \approx n/2$. Таким образом, если алгоритм исполняется многократно и начальное расположение Робота носит случайный характер, так, что "в среднем" он расположен около середины стены, то применение второго алгоритма дает "в среднем" выигрыш в n/2 шагов при каждом его исполнении.

9. План алгоритма. Сначала Робот движется в одном направлении (например, вверх), пока не дойдет до стены; после этого Робот, перемещаясь вдоль стен, обходит и закрашивает клетки, прилегающие к стенам с их внутренней стороны. Приведем алгоритм:


```
алг закрасить прилегающие к стенам клетки
  дано | Робот находится внутри прямоугольника, огорожен-
       | ного стенами; внутри прямоугольника стен нет
  надо | закрашены прилегающие к стенам клетки
       | внутри прямоугольника
нач
       | 1
  вверх до стены
       | 2
  закрасить
  нц пока справа свободно
  | вправо; закрасить
  кц
       | 3
  нц пока снизу свободно
  | вниз; закрасить
  кц
       | 4
  нц пока слева свободно
  | влево; закрасить
  кц
       | 5
  нц пока сверху свободно
  | вверх; закрасить
  кц
       | 6
  нц пока клетка не закрашена
  | закрасить; вправо
  кц
       | 7
кон
```

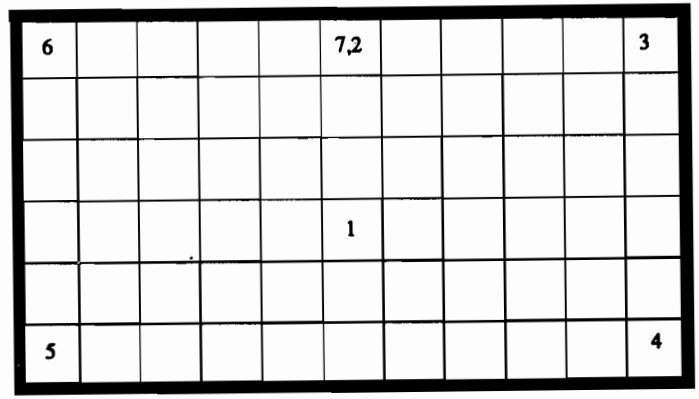


Рис. 53

Для удобства пояснений в алгоритме в комментариях расставлены цифры. На рисунке 53 цифрами обозначены соответствующие положения Робота.

Можно придерживаться и другого плана: сначала Робот перемещается в один из углов (например, в верхний левый) и затем обходит прямоугольник по периметру с внутренней стороны. Этот алгоритм менее эффективен, при его исполнении Робот делает большее число шагов (или равное, если он находится вначале у левой стены), чем при исполнении предыдущего. Однако по структуре этот алгоритм несколько проще и его удобнее составлять методом "сверху вниз".

```
алг закрасить прилегающие к стенам клетки
  дано | Робот находится внутри прямоугольника, огорожен-
       | ного стенами; внутри прямоугольника стен нет
  надо | закрашены прилегающие к стенам клетки
       | внутри прямоугольника
нач
| в левый верхний угол
| закрасить около стен
кон

алг в левый верхний угол
  дано | Робот находится внутри прямоугольника, огорожен-
       | ного стенами; внутри прямоугольника стен нет
  надо | Робот находится в левом верхнем углу
       | прямоугольника
нач
| вверх до стены; влево до стены
кон
```

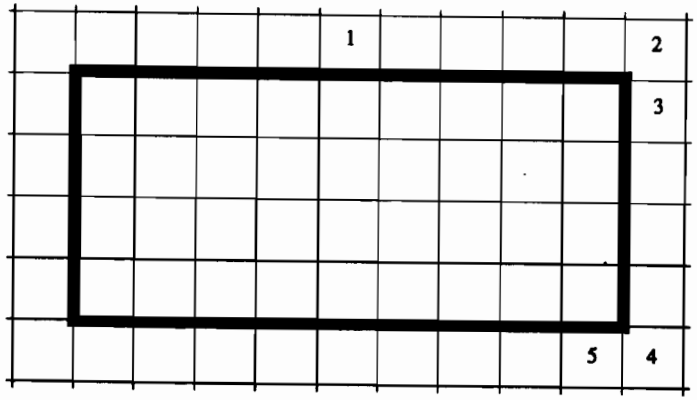


Рис. 54


```
алг закрасить около стен
  дано | Робот находится в левом верхнем углу
       | прямоугольника
  надо | закрашены прилегающие к стенам клетки
       | внутри прямоугольника
нач
| нц пока справа свободно
| | вправо
| | закрасить
| кц
| нц пока снизу свободно
| | вниз
| | закрасить
| кц
| нц пока слева свободно
| | влево
| | закрасить
| кц
| нц пока сверху свободно
| | вверх
| | закрасить
| кц
кон
```

Нужно обратить внимание учащихся на условия **дано** и **надо**.Вспомогательные алгоритмы вызываются один за другим. Соответственно **дано** в первом вспомогательном алгоритме и **надо** во втором совпадают с **дано** и **надо** в основном алгоритме; а **надо** в первом совпадает с **дано** во втором. Можно сказать, что первый вспомогательный алгоритм подготавливает условия для исполнения второго.

В общем случае при подобной "линейной декомпозиции" основного алгоритма **дано** каждого следующего вспомогательного алгоритма должно вытекать из **надо** предыдущего; **дано** первого вытекать из **дано** основного; **надо** основного — из **надо** последнего.

10. План алгоритма. Робот движется вдоль стены вправо, пока не кончится верхняя стена, затем движется вниз, пока не кончится правая стена, и затем заходит за нижнюю стену. До составления алгоритма можно сделать рисунок и отметить на нем последовательные положения Робота подобно тому, как это сделано в предыдущем упражнении (рис. 54). При составлении алгоритма соответствующие места можно указать цифрами в комментариях.

```
алг обход прямоугольника
  дано | Робот над верхней стороной прямоугольника, огоро-
       | женного стенами; снаружи прямоугольника стен нет
  надо | Робот под нижней стороной прямоугольника (рис. 54)
нач
     | 1
  нц пока снизу стена
  | вправо
  кц
     | 2
  вниз
     | 3
  нц пока слева стена
  | вниз
  кц
     | 4
  влево
     | 5
кон
```

11. Сначала составим план действий Робота. Дойти до перегородки, найти проход, дойти до нижней стены, переместиться в правый нижний угол. Используя вспомогательные алгоритмы, получаем (рис. 55):

```
алг в правый нижний угол через проход
  дано | Робот находится в левом верхнем углу прямоугольника,
       | огороженного стенами; прямоугольник перегорожен
       | стеной с проходом, идущей от левого до правого края;
       | проход не прилегает к стенам прямоугольника
  надо | Робот находится в правом нижнем углу (рис. 55)
нач
                              | A
  вниз до стены               | 1
  найти проход                | 2
  вниз до стены               | 3
  вправо до стены             | Б
кон
```

Вспомогательный алгоритм "найти проход" приведен в рекомендациях к уроку.

Полезно рассмотреть, как будет исполнен алгоритм, если в перегородке правее Робота прохода нет. В этом случае исполнение цикла в алгоритме "найти проход" приведет к отказу. Чтобы этого избежать, запишем указанный цикл так:

```
нц пока снизу стена и справа свободно
| вправо
кц
```

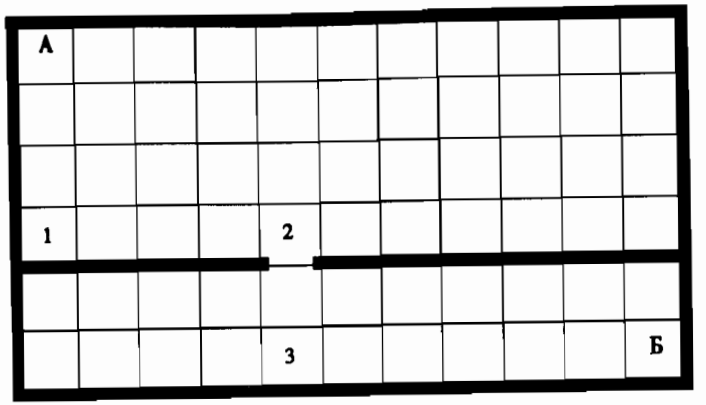



Рис. 55

12. Эта задача решается так же, как и задача 10. По существу разница лишь в том, что после (или перед) каждой командой смещения нужно ставить команду закраски клетки.

```
13. алг закрасить клетки вокруг стены
  дано | Робот в клетке А (рис. 52 учебника)
  надо | клетки, отмеченные точками, закрашены
       | Робот в исходном положении
нач
| закрасить; вправо
| нц пока снизу стена
| | закрасить
| | вправо
| кц
| закрасить; вниз; закрасить; влево
| нц пока слева свободно
| | закрасить; влево
| кц
| нц пока слева стена
| | закрасить; вниз
| кц
| закрасить; влево; закрасить; вверх
| нц пока сверху свободно
| | закрасить; вверх
| кц
| нц пока сверху стена
| | закрасить; влево
| кц
| закрасить; вверх
кон
```

14. Надо повторить рисунок 42 (с.70 учебника) и нарисовать на нем две дорожки, отмечая точками места проверки условия цикла.

§ 10. УСЛОВИЯ В АЛГОРИТМИЧЕСКОМ ЯЗЫКЕ. КОМАНДЫ ЕСЛИ И ВЫБОР. КОМАНДЫ КОНТРОЛЯ (6 ч)

Компромисс всегда обходится дороже, чем любая из альтернатив.

закон Джухэни

В этом параграфе изучаются команды ветвления **если** и **выбор**, дается общий вид этих команд, графическая схема, приводятся примеры их использования для составления алгоритмов. Вводится команда контроля **утв** и дается пример ее применения. Рассматриваются способы записи и смысл составных условий.

Основные цели. Ввести команды ветвления, показать их применение. Дать общее представление о составных условиях и командах контроля.

Требования к знаниям и умениям. Учащиеся должны знать способы записи составных условий на алгоритмическом языке и понимать их смысл; знать назначение, способ записи на алгоритмическом языке и схему команд ветвления **если** и **выбор**; уметь использовать их при составлении алгоритмов.

ОСНОВНЫЕ ПОНЯТИЯ

Простые и составные условия развивают введенное в § 7 понятие выражения.

Введение числовых величин (§ 6) потребовало научиться строить арифметические выражения.

В § 9 вводятся логические величины, представленные как команды-вопросы и отношения между величинами. При этом возникает необходимость строить логические выражения. Таким образом, условие есть не что иное, как ***логическое выражение***; его значением могут быть логические константы **да** или **нет**.

Для определения порядка действий в логическом выражении в учебнике рекомендуется использовать скобки. При отсутствии скобок первой выполняется операция **не**, второй — **и**, третьей — **или**. При этом все арифметические операции выполняются раньше логических.

Команды ветвления и выбора описаны в учебнике достаточно подробно. Отметим полную взаимозаменяемость этих

команд; всякое ветвление можно представить в форме выбора, и наоборот:

```
если условие
| то серия 1
| иначе серия 2
все
```

```
выбор
| при условие: серия 1
| иначе серия 2
все
```

```
если условие
| то серия
все
```

```
выбор
| при условие: серия
все
```

```
выбор
| при условие 1 : серия 1
| при условие 2 : серия 2
|        ...
| при условие N : серия N
| иначе серия N+1
все
```

```
если условие 1
| то серия 1
| иначе если условие 2
|        | то серия 2
|        |        ...
|        | иначе если условие N
|        |        | то серия N
|        |        | иначе серия N+1
|        |        все
|        |       ...
|        все
все
```

```
выбор
| при условие 1 : серия 1
| при условие 2 : серия 2
|        ...
| при условие N : серия N
все
```

```
если условие 1
| то серия 1
| иначе если условие 2
|        | то серия 2
|        |        ...
|        | иначе если условие N
|        |        | то серия N
|        |        | все
|        |        ...
|        все
все
```

Команда контроля играет важную роль в формировании алгоритмического мышления. На первый взгляд она может показаться бесполезной: никаких активных действий, проверка зачастую совершенно очевидных утверждений. Однако именно использование команды **утв** (наряду с **дано** и **надо**) позволяет четко сформулировать требования к алгоритму, круг решаемых им задач и возможные ограничения, выделить промежуточные цели, достигаемые в ходе исполнения алгоритма.

Используя утверждения, подобные применяемым в команде контроля, можно доказывать правильность алгоритмов путем

строгих математических рассуждений. Примеры такого рода доказательств приведены в п. 16.10 учебника.

ОБЩИЕ УКАЗАНИЯ

Команды ветвления вместе с командами повторения составляют полный набор средств конструирования алгоритмов. В целом команды ветвления проще команд повторения и сами по себе усваиваются легче. Более сложно научиться использовать их в сочетании с командами повторения. В параграфе команды ветвления используются вместе с командой повторения **пока**.

Когда цикл используется внутри одной из серий команды **если**, сложность возникающей алгоритмической структуры может быть уменьшена за счет применения вспомогательных алгоритмов. Так сделано, например, в п. 10.4 (А39). Вспомогательные алгоритмы позволяют здесь продемонстрировать команду **если** "в чистом виде".

Конструкция, в которой команда **если** находится внутри цикла (п. 10.5, А40), вообще говоря, более трудна для понимания.

Одна команда ветвления может использоваться внутри другой. Такие сочетания не запрещены алгоритмическим языком. Например, в п. 12.5 (А49) встречается конструкция вида

```
если ...
| то ...
| иначе ...
|   если ...
|   | то ...
|   | иначе ...
|   все
все
```

При первом знакомстве с командами ветвления не стоит останавливаться на подобных сложных случаях.

Графические схемы команд ветвления играют вспомогательную роль. Главная цель их использования — сделать более наглядной запись команды и более понятной ее структуру. Графические схемы команд ветвления удобно применять для иллюстрации отдельных сложных фрагментов алгоритмов, например вложенных команд ветвления.

В § 9 пособия уже говорилось о составных условиях. Сделаем еще несколько замечаний. Составные условия легче понимаются, если отрицание **не** относится только к простым условиям.

Следующие пары составных условий имеют одинаковый смысл:

не (условие 1 **и** условие 2)	(**не** условие 1) **или** (**не** условие 2)
не (условие 1 **или** условие 2)	(**не** условие 1) **и** (**не** условие 2)
не (**не** условие)	условие

Используя это, можно упрощать составные условия.

В отношениях между величинами можно вообще обойтись без отрицаний. Например, **не** (a=b) означает то же, что и a≠b, а **не** (a<b) — то же, что и a≥b.

Учитывая, что все команды обратной связи Робота типа **лог** имеют парные команды-отрицания, в алгоритмах можно вообще обходиться без **не**.

Рассмотрим команду контроля **утв**. Если после **утв** записано условие, то это условие проверяется и, если оно нарушено, выполнение алгоритма прекращается (п. 10.10). Команда **утв** может употребляться для записи утверждений-комментариев. В этом случае исполнение алгоритма не зависит от наличия или отсутствия команды **утв**. Применяется также комбинированная форма записи команды.

Следует обратить внимание учащихся на различие в записи условий в командах **утв**. Если условие проверяемое, то оно должно быть записано по правилам записи условий. Если же условие пишется в качестве комментария, то его можно записать неформально, описательно. К такой записи прибегают обычно, когда для проверки условия требуется выполнение специальных дополнительных алгоритмов. В учебнике команда **утв** употребляется, как правило, именно в таком виде.

Применение команды **утв** играет двойную роль. С одной стороны, как отмечается в учебнике, она облегчает понимание написанного алгоритма. С другой стороны, использование команды **утв** дисциплинирует учащихся при составлении алгоритма, нацеливает на точную фиксацию результатов исполнения отдельных фрагментов алгоритма.

РЕКОМЕНДАЦИИ К ВЕДЕНИЮ УРОКОВ

З а д а ч а (см. задачу 5а) на с. 83 учебника). Робот находится в каком-то углу огороженного стенами прямоугольника неизвестного размера. Внутри стен нет. Требуется перевести Робота в противоположный угол.

На первый взгляд кажется, что надо выполнить одно из четырех действий (табл. 8).

Таблица 8

Исходное положение	Конечное положение
Верхний левый угол	Нижний правый угол
Нижний левый угол	Верхний правый угол
Верхний правый угол	Нижний левый угол
Нижний правый угол	Верхний левый угол

Из таблицы видно, что каждая сторона меняется на противоположную. Значит, на первом шаге уточнения решение задачи можно разбить всего на два действия:

1) смена положения по горизонтали;
2) смена положения по вертикали.

```
алг переход в противоположный угол
  дано | Робот в некотором углу прямоугольника
  надо | Робот в противоположном углу
нач
| переход по горизонтали
| переход по вертикали
кон
```

Очевидно, что вспомогательные алгоритмы аналогичны. Составим первый из них. Возможны два случая (табл. 9).

Таблица 9

Исходное положение	Конечное положение
У левой стены	У правой стены
У правой стены	У левой стены

```
алг переход по горизонтали
  дано | Робот у какой-то вертикальной стены
       | и внутри прямоугольника
  надо | Робот у противоположной стены
нач
| если слева стена
|   то вправо до стены
|   иначе влево до стены
| все
кон
```

Аналогично будет выглядеть второй вспомогательный алгоритм, необходимый для решения исходной задачи:

```
алг переход по вертикали
  дано | Робот у какой-то горизонтальной стены
       | внутри прямоугольника
  надо | Робот у противоположной стены
нач
| если снизу стена
|   то вверх до стены
|   иначе вниз до стены
| все
кон
```


Команда ветвления имеет две формы записи, ***полную и сокращенную.*** Общий вид команды в полной форме:

```
если   <условие>
| то   <серия команд 1>
| иначе   <серия команд 2>
все
```

При работе этой команды возможны два случая:

1) однократно выполняется серия 1 (если условие соблюдается);
2) однократно выполняется серия 2 (если условие не соблюдается).

Слово "однократно" подчеркивает отличие ветвления от цикла.

Сокращенная форма ветвления:

```
если <условие>
| то <серия команд 1>
все
```

Здесь тоже может быть два случая:

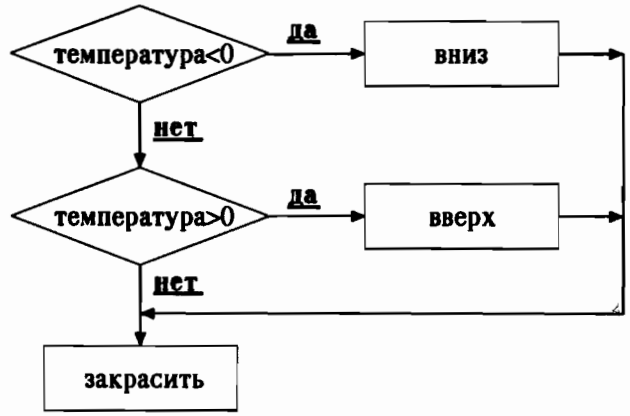
1) серия команд выполняется однократно (если условие соблюдается);
2) серия команд не выполняется ни разу (если условие не соблюдается).

Рассмотрим еще одну задачу. Робот должен передать информацию о знаке температуры в той клетке, где он находится. Считаем, что во время выполнения алгоритма мы видим поле Робота, но не имеем доступа к ЭВМ. Договоримся так. Если температура положительна, то он закрасит клетку выше своего начального положения. Если отрицательна — то ниже. Если температура равна нулю, то будет закрашена исходная клетка.

Вариант решения:

```
алг определение знака температуры
  дано | Робот в некоторой клетке поля
  надо | закрашена одна клетка
       | в зависимости от знака температуры
нач
| если температура < 0
|   то вниз
|   иначе если температура > 0
|           то вверх
|           все
| все
| закрасить
кон
```

Изобразим алгоритм решения графически:



Необходимость выбора из трех вариантов (меньше, больше, равно) заставила использовать две вложенные одна в другую команды **если**. Запись алгоритма, содержащая сочетание "**иначе если**", затрудняет понимание решения. На графической схеме алгоритм выглядит проще и понятнее.

В алгоритмическом языке предусмотрена возможность более наглядной записи выбора из трех и более вариантов с помощью команды **выбор**:

```
алг определение знака температуры
  дано | Робот в некоторой клетке поля
  надо | закрашена одна клетка
       | в зависимости от знака температуры
нач
| выбор
|   при температура < 0 : вниз
|   при температура > 0 : вверх
| все
| закрасить
кон
```

При составлении алгоритмов мы несколько раз использовали простые и составные условия. Рассмотрим их подробнее.

Отношения между величинами записываются с помощью шести ***знаков отношений:*** <, >, =, ≠, ≤, ≥. В информатике получаемые выражения называются ***условиями, утверждениями или логическими выражениями.***

Логическое выражение может принимать всего два значения — **да** либо **нет** (вспомним величины типа **лог**).

Упражнения.	**Ответы.**
Каковы значения выражений:	
а) 3 = 2	**нет**
б) 2*2 = 4	**да**
в) 3 ≥ 5	**нет**
г) sqrt(4) ≤ 3	**да**
д) x < 0	зависит от x

Видно, что в логических выражениях разрешается использовать знаки арифметических операций (см. б)) и стандартные функции (см. г)). Над самими логическими выражениями тоже можно проделывать операции. Таких операций три: **не**, **и**, **или**. Они называются ***логическими операциями.***

не да — это **нет**
не нет — это **да**
да и да — это **да**
да и нет — это **нет**
нет и нет — это **нет**
да или да — это **да**
да или нет — это **да**
нет или нет — это **нет**

Как и арифметические действия, логические операции имеют приоритеты. Первой выполняется операция **не**, после нее — **и**, последней — **или**. Порядок выполнения логических операций можно менять с помощью круглых скобок.

Упражнения.	**Ответы.**
Определите значения выражений:	
а) 5>8	**нет**
б) 5>8 **и** 2*2=4	**нет**
в) **не** 5>8 **и** 2*2=4	**да**
г) **не** (5>8 **и** 2*2=4)	**да**
д) **не** (5>8 **или** 2*2=4)	**нет**
е) **не** (x>8 **или** 2*2=4)	**нет** (от x не зависит)

Логические выражения, образованные с помощью знаков логических операций, называются составными. При этом в составное логическое выражение может входить только одно простое (например: **не** Y>0). Составные выражения можно образовать из ответов Робота на команды-вопросы. Например: слева свободно **и** снизу свободно.

Используя операцию **не**, можно исключить из СКИ Робота пять команд, потому что:

сверху стена = **не** сверху свободно
справа свободно = **не** справа стена
клетка закрашена = **не** клетка не закрашена и т.д.

Однако сказанное относится, конечно, только к командам-вопросам. Например, нельзя считать одинаковыми команду "вверх" и бессмысленное сочетание "не вниз" (хотя вспомогательный алгоритм с именем "не вниз" может существовать).

Все преобразования в логических выражениях делает ЭВМ, без участия исполнителей. Например, встретив выражение "не сверху свободно?", ЭВМ дает Роботу команду-вопрос "сверху свободно", получает от него ответ, а затем сама преобразует полученное значение в обратное.

Если в составное логическое выражение входит несколько простых, то полученное значение не всегда зависит от значения каждой из составляющих. Например, в случае е) последнего упражнения (с. 165) результат не зависит от значения выражения p>8, а значит, и от величины p. Сравните: значение b не влияет на значение арифметического выражения 0 * b + 1/2.

Этим обстоятельством пользуется ЭВМ при нахождении результата составного логического выражения. Получив значение **да** для левой части выражения, содержащего **или**, ЭВМ, как правило, правую часть даже не рассматривает.

Как видно из графической схемы (с. 80 учебника), перебор вариантов в команде **выбор** происходит в порядке их написания. Поэтому ***более строгие условия должны предшествовать менее строгим.*** Чтобы показать это, рассмотрим задачу: Даны стороны треугольника. Требуется определить его вид и вызвать соответствующий вспомогательный алгоритм.

```
алг определить вид треугольника (арг вещ a, b, c)
  дано | a+b>c и a+c>b и b+c>a
  надо | определен вид треугольника
нач
| выбор
| | при a = b и a = c : равносторонний
| | при a = b или a = c или b = c : равнобедренный
| | иначе произвольный
| все
кон
```

Если в приведенном алгоритме изменить порядок проверки условий, то равносторонний треугольник будет неверно определен как равнобедренный.

Иногда требуется во время выполнения алгоритма проверить соблюдение некоторого условия, чтобы избежать отказа. Например, в такой задаче: Робот в левом верхнем углу прямоугольника неизвестного размера, ограниченного стенами. Других стен на поле нет. Между верхней и нижней стенами нечетное число клеток. Закрасить клетки, в которых уровень радиации больше единицы.


```
алг маркировка опасных клеток
  дано | Робот в верхнем левом углу прямоугольника
  надо | закрашены опасные клетки
нач
| нц пока снизу свободно
| | маркировка вправо
| | вниз
| | маркировка влево
| | вниз
| кц
| маркировка вправо
кон
```

При составлении этого алгоритма предполагалось, что в соответствии с условием задачи между верхней и нижней стенами нечетное расстояние. Если кому-то придет в голову применить наш алгоритм для прямоугольника с четной высотой, то произойдет отказ, ведь условие "снизу свободно" при каждом выполнении цикла проверяется один раз, а команда "вниз" вызывается дважды.

Для подобных ситуаций алгоритмический язык содержит ***команды контроля.*** Две из них, **дано** и **надо**, позволяют проверить некоторые условия *до* и *после* выполнения алгоритма. Для проверки условий во время *выполнения* используется команда **утв**. Эта команда, как и команды **дано** и **надо**, может применяться для записи комментариев. Измененный алгоритм запишется так:

```
алг маркировка опасных клеток
  дано | Робот в верхнем левом углу прямоугольника
  надо | закрашены опасные клетки
нач
| нц пока снизу свободно
| | маркировка вправо
| | вниз
| | маркировка влево
| | утв снизу свободно
| | вниз
| кц
| маркировка вправо
| утв | Робот в нижнем правом углу
кон

алг маркировка вправо
  дано | Робот у левой стены
  надо | закрашены опасные клетки
       | горизонтальной полосы
```

```
нач
| нц пока справа свободно
|    если радиация > 1
|    | то закрасить
|    все
|  вправо
| кц
| если радиация > 1
| | то закрасить
| все
| утв | Робот у правой стены
кон
```

Аналогично записывается алгоритм "маркировка влево".

Запишем в словарь:

```
если <условие>                  } команда ветвления
| то <серия команд 1>           } (полная форма)
| иначе <серия команд 2>        }
все

если <условие>                  } команда ветвления
| то <серия команд 1>           } (сокращенная форма)
все                             }
```

не, и, или — логические операции (в порядке убывания приоритета)

```
выбор                                   }
| при <условие 1> : <серия 1>           }
| при <условие 2> : <серия 2>           } команда выбора
|              ...                      } из трех и большего
| при <условие N> : <серия N>           } числа вариантов
|    иначе <серия N + 1>                }
все                                     }
```

утв — команда контроля во время выполнения алгоритма

УКАЗАНИЯ К УПРАЖНЕНИЯМ

1.
```
алг разметка горячих клеток коридора
дано | Робот стоит в левой клетке горизонтального коридора
надо | Робот вышел из коридора вправо, клетки, в которых
     | температура выше 100 градусов, закрашены
```



```
нач
| нц пока снизу стена
|    если температура > 100
|    | то закрасить
|    все
|  вправо
| кц
кон
```

2. Задания а) — г) стандартны, здесь можно ограничиться ответами: а) $x \geq 0$; б) $y \leq 0$; в) $x \leq y$; г) $x \geq -1$ **и** $y \geq -1$.

Задание д) несколько сложнее. Можно сначала разобрать задание е). Внутренние точки круга находятся на расстоянии меньше 1 от центра, точки (0,0). Следовательно, эти точки удовлетворяют неравенству $x^2+y^2 \leq 1$. Соответственно точки вне круга удовлетворяют неравенству $x^2+y^2>1$. На алгоритмическом языке получаем запись: е) x*x + y*y > 1, или x**2 + y**2 > 1.

Задание д) можно выполнить несколькими способами. Целям изучаемого параграфа в наибольшей степени соответствует такой. Заштрихованный квадрат — пересечение четырех полуплоскостей. Поэтому нужно записать условия, соответствующие этим полуплоскостям, и соединить их связкой **и**. Уравнения граничных прямых: y=−x+1; y=x+1; y=−x−1; y=x−1 (начиная с первой четверти, против часовой стрелки). Соответственно полуплоскости определяются условиями: $y \leq -x+1$; $y \leq x+1$; $y \geq -x-1$; $y \geq x-1$. Окончательно получаем: $y \leq -x+1$ **и** $y \leq x+1$ **и** $y \geq -x-1$ **и** $y \geq x-1$.

Можно подойти к решению задачи несколько иначе, заметив, что граница квадрата определяется уравнением $|x|+|y|=1$. Тогда по аналогии с окружностью получаем для всех внутренних точек: д) abs(x) + abs(y) ≤ 1.

Эту аналогию можно пояснить. Предположим, что расстояние между точками на плоскости определяет устройство, которое может перемещаться только в четырех основных направлениях. Тогда, как легко видеть, расстояние от точки (0,0) до точки (x,y) равно сумме расстояний между точками (0,0), (x,0) и (x,0), (x,y), т. е. $|x|+|y|$, и граница квадрата —это "единичная окружность" при описанном способе измерения расстояния.

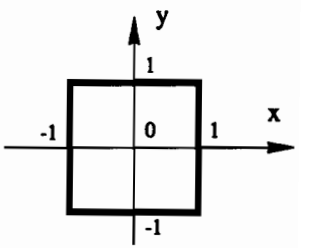


Рис. 56

В качестве дополнительного упражнения можно предложить задание: изобразить "единичную окружность", если расстояние от точки (0,0) до точки (x,y) определяется по формуле max(|x|,|y|). Здесь получается квадрат, изображенный на рисунке 56.

Все его внутренние точки описываются соотношением $\max(|x|,|y|)\le 1$. Можно то же условие записать так:

$x\le 1$ **и** $x\ge -1$ **и** $y\le 1$ **и** $y\ge -1$.

3. Переделка алгоритма сводится к замене **и** на **или**.

4. Перед выполнением отдельных заданий полезно рассмотреть и составить общую схему алгоритма: Робот движется вправо (10 шагов) и, встречая закрашенные клетки, предпринимает определенные действия. Общая схема алгоритма:

```
алг закрасить соседние клетки
  дано | на поле Робота стен нет
  надо | для 10 клеток правее Робота закрашены клетки, примы-
       | кающие к закрашенным в соответствии с требованиями
нач
  нц 10 раз
    вправо
      если клетка закрашена
        то
           [          ]
      все
  кц
кон
```

Теперь для случаев а) — в) нетрудно выписать содержимое рамки:

а)	б)	в)
вниз	вниз	влево
закрасить	закрасить	закрасить
вверх	вверх	вправо
	вверх	
	закрасить	
	вниз	

По аналогии с предыдущими можно записать для случая г): вправо; закрасить; влево.

Однако при этом может произойти неверная закраска. Исполнение такого алгоритма приводит к тому, что оказываются закрашенными все клетки, расположенные правее первой закрашенной клетки. На рисунке 57 изображена последовательность ситуаций, возникающих при повторениях тела цикла.

Обнаружив, что 8-я клетка закрашена, Робот закрашивает 9-ю; затем, так как 9-я закрашена, — 10-ю, далее 11-ю. Этот пример показывает, что для выполнения задания г) нужно изменить план действий.

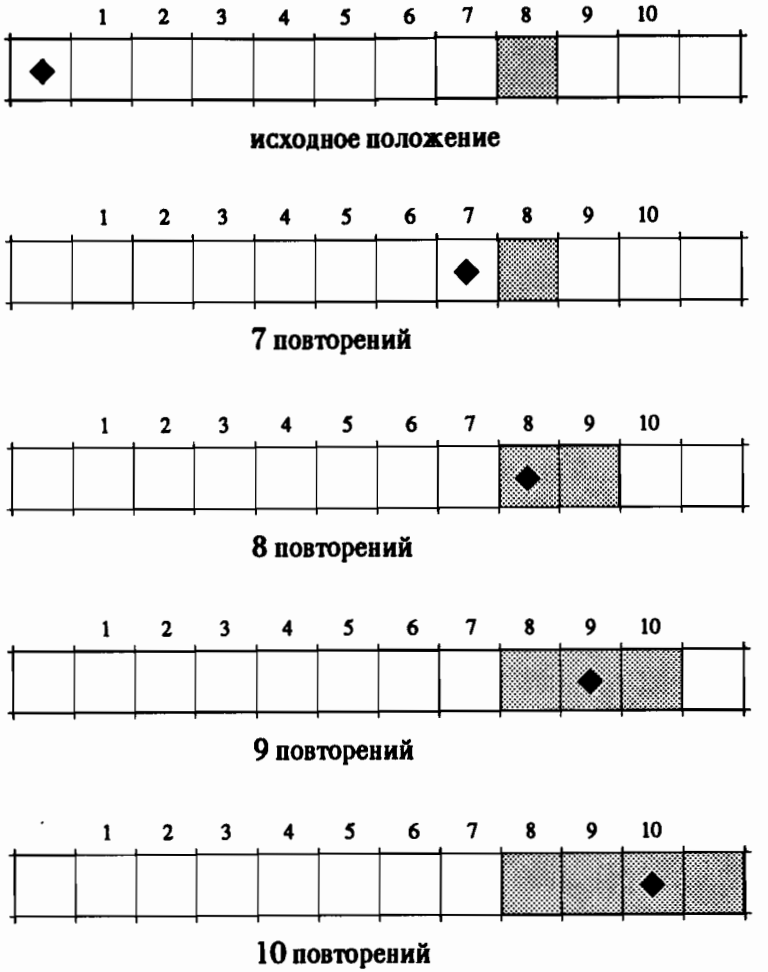


Рис. 57

Можно заметить, что задания в) и г) превращаются одно в другое, если всюду поменять между собой "право" и "лево". Получаем такой план: Робот проходит направо до конца полосы (11 шагов), затем движется налево и закрашивает клетки правее закрашенных. С сильными учащимися можно попытаться разобрать построение рекурсивного алгоритма.

```
алг закрасить соседние клетки Г
  дано | на поле Робота стен нет
  надо | для 10 клеток правее Робота закрашены
       | клетки, примыкающие к закрашенным справа
```

```
нач
| нц 11 раз
| | вправо
| кц
| нц 10 раз
| | влево
| | | если клетка закрашена
| | | то вправо; закрасить; влево
| | | все
| кц
кон
```

Алгоритм для д) можно получить, комбинируя алгоритмы в) и г):

```
алг закрасить соседние клетки
   дано | на поле Робота стен нет
   надо | для 10 клеток правее Робота
        | закрашены клетки, примыкающие
        | к закрашенным справа
нач
|      | 1
| нц 10 раз
| | вправо
| | | если клетка закрашена
| | | то влево; закрасить; вправо
| | | все
| кц
| вправо
|      | 2
| нц 10 раз
| | влево
| | | если клетка закрашена
| | | то вправо; закрасить; влево
| | | все
| кц
|      | 3
кон
```

На рисунке 58 приведен пример исполнения алгоритма.

При исполнении первого цикла Робот закрашивает клетки левее каждой из закрашенных в положении 1. При исполнении второго цикла Робот закрашивает клетки правее закрашенных в положении 2.

Проверим, не закрашивает ли Робот лишних клеток. Если клетка, закрашенная в положении 2, закрашена также и в положении 1, то Робот должен закрасить клетку левее нее по усло-

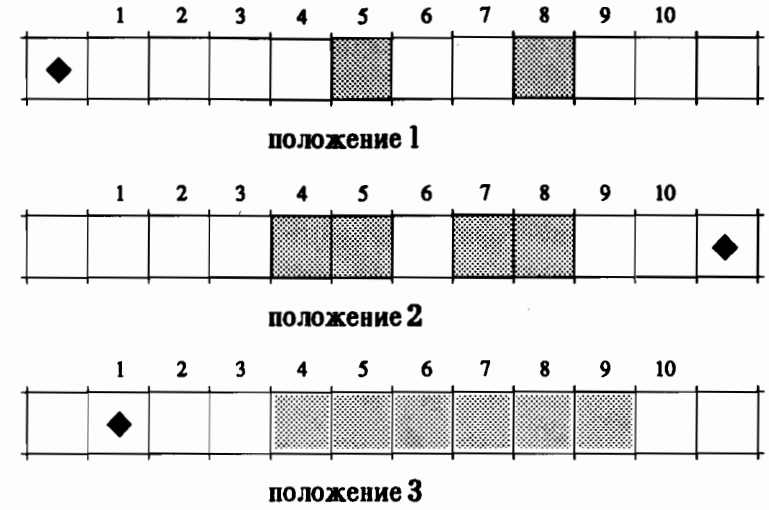


Рис. 58

вию задания. Если же эта клетка была закрашена при исполнении первого цикла, то левее нее находится клетка, закрашенная в исходном положении. В этом случае Робот закрашивает уже закрашенную клетку.

Таким образом, в результате исполнения алгоритма будут закрашены все клетки левее и правее тех, которые закрашены в исходном положении.

5.а)
```
алг переход в противоположный угол
   дано | Робот стоит в каком-то углу прямоугольника,
        | огороженного стенами; других стен нет
   надо | Робот в противоположном углу
нач
| если слева стена
| | то вправо до стены
| | иначе влево до стены
| все
| если сверху стена
| | то вниз до стены
| | иначе вверх до стены
| все
кон
```

При выполнении первой команды, **если** Робот переходит к противоположной стене, — по горизонтали, а при выполнении второй — по вертикали. В итоге Робот смещается по диагонали в противоположный угол.

б)
```
алг к противоположной стене
  дано | Робот стоит у стены (но не в углу) внутри прямо-
       | угольника, обнесенного со всех сторон стенами;
       | внутри прямоугольника других стен нет
  надо | Робот перешел к противоположной стене
нач
| выбор
| | при сверху стена :  вниз до стены
| | при снизу  стена :  вверх до стены
| | при справа стена :  влево до стены
| | при слева  стена :  вправо до стены
| все
кон
```

В этой задаче можно также попросить учащихся записать на алгоритмическом языке условие "Робот стоит у стены":

сверху стена **или** снизу стена **или** справа стена **или** слева стена

Условие "в углу":

(слева стена **и** сверху стена) **или**
(сверху стена **и** справа стена) **или**
(справа стена **и** снизу стена) **или**
(снизу стена **и** слева стена)

А также условие "Робот стоит у стены, но не в углу":

(сверху стена **или** снизу стена **или**
справа стена **или** слева стена)
и не
((слева стена **и** сверху стена) **или**
(сверху стена **и** справа стена) **или**
(справа стена **и** снизу стена) **или**
(снизу стена **и** слева стена))

Заметим, что это же условие можно записать по-другому, например так:

(сверху стена **и** справа свободно **и** слева свободно) **или**
(снизу стена **и** справа свободно **и** слева свободно) **или**
(справа стена **и** сверху свободно **и** снизу свободно) **или**
(слева стена **и** сверху свободно **и** снизу свободно)

или так:

(справа свободно **и** слева свободно **и**
(сверху стена **или** снизу стена))
или
(сверху свободно **и** снизу свободно **и**
(справа стена **или** слева стена))

6.
```
алг выход из коридора
  дано | Робот внутри коридора без боковых выходов
  надо | Робот вышел из коридора
нач
| если снизу стена
| | то | коридор горизонтальный
| |    нц пока снизу стена
| |    | вправо
| |    кц
| |  иначе | коридор вертикальный
| |    нц пока справа стена
| |    | вниз
| |    кц
| все
кон
```

Условия в **дано** и **надо** в этой задаче также можно попросить записать на алгоритмическом языке. Полезно также заметить, что команда **если** в этом алгоритме не нужна — приведенный выше алгоритм можно записать проще:

```
алг выход из коридора
  дано | Робот внутри коридора без боковых выходов
  надо | Робот вышел из коридора
нач
| нц пока снизу стена
| | вправо
| кц
| нц пока слева стена
| | вниз
| кц
кон
```

Последний алгоритм рекомендуется подробно разобрать, обратив внимание на замену условия "справа стена" на условие "слева стена" во втором цикле, так как без такой замены в следующей ситуации алгоритм будет работать неверно (рис. 59):

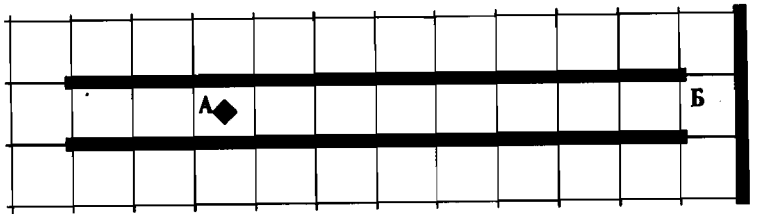


Рис. 59

Таким образом, в этом алгоритме вместо "слева" нельзя написать "справа", хотя по постановке задачи между ними нет никакой разницы. Подобного рода недостатки алгоритмов обычно называют "подводными камнями". От этого недостатка можно избавиться, если условие "в коридоре" отразить внутри алгоритма более полно:

```
алг выход из коридора
  дано | Робот внутри коридора без боковых выходов
  надо | Робот вышел из коридора
нач
| нц пока снизу стена и сверху стена
|  вправо
| кц
| нц пока слева стена и справа стена
|  вниз
| кц
кон
```

Теперь "вправо" можно менять на "влево", "вниз" — на "вверх", и алгоритм во всех случаях будет работать верно. Следует обратить на это внимание учащихся и рекомендовать им по возможности избегать "подводных камней" в алгоритмах.

7. Эту задачу рекомендуется давать только после разбора эталонного (последнего) решения упражнения 6. Нужно заметить, что если Робот вышел из тупика, то условие повторения нарушается во всех четырех командах пока.

```
алг выход из тупика
  дано | Робот внутри тупика
  надо | Робот вышел из тупика
нач
| нц пока снизу стена и сверху стена и справа свободно
|  вправо
| кц
| нц пока снизу стена и сверху стена и слева свободно
|  влево
| кц
| нц пока слева стена и справа стена и сверху свободно
|  вверх
| кц
| нц пока слева стена и справа стена и снизу свободно
|  вниз
| кц
кон
```


8.

```
алг на другой конец коридора
  дано | Робот на перекрестке, от которого отходят
       | один коридор и три тупика
  надо | Робот вышел из другого конца коридора
нач
| вправо до стены или выхода
| если снизу стена и сверху стена
|                  | т.е. справа тупик
|  то
|  влево до перекрестка; влево до стены или выхода
|    если снизу стена и сверху стена
|                  | т.е. слева тупик
|     то
|     вправо до перекрестка; вверх до стены или выхода
|       если слева стена и справа стена
|                  | т.е. сверху тупик
|        то
|        вниз до перекрестка;
|        вниз до стены или выхода
|       все
|    все
| все
кон
```

```
алг вправо до стены или выхода
  дано | Робот на перекрестке, вправо отходит
       | коридор или тупик
  надо | Робот либо вышел вправо из коридора,
       | либо уперся в правый конец тупика
нач
| вправо | вход в коридор или тупик
| нц пока снизу стена и сверху стена и справа свободно
|  вправо
| кц
кон
```

Алгоритмы "влево до стены или выхода", "вверх до стены или выхода", "вниз до стены или выхода" составляются аналогично.

```
алг влево до перекрестка
  дано | Робот в тупике с выходом влево
  надо | Робот вышел влево из тупика (на перекресток)
нач
| нц пока снизу стена и сверху стена и слева свободно
|  влево
| кц
кон
```

Заметим, что вместо "влево до перекрестка" можно было написать вызов "влево до стены или выхода", т.е. использовать 4 вспомогательных алгоритма вместо 8. Можно было вместо "влево до перекрестка", "вправо до перекрестка" и др. использовать и вызов "выход из тупика" (упражнение 7).

§ 11. ВЕЛИЧИНЫ В АЛГОРИТМИЧЕСКОМ ЯЗЫКЕ. КОМАНДА ПРИСВАИВАНИЯ (2 ч)

Преподавание программирования — дело почти безнадежное, а его изучение — непосильный труд.

Чарлз Уэзерелл

В параграфе вводится понятие величины, описывается модель работы ЭВМ с величинами; вводится команда присваивания и раскрываются возможности ее применения для построения алгоритмов.

Основные цели. Дать учащимся представление о величинах в алгоритмическом языке, о работе ЭВМ с величинами; показать использование величин и команды присваивания при составлении алгоритмов.

Требования к знаниям и умениям. Учащиеся должны знать атрибуты величин (тип, имя, значение); понимать и применять для составления алгоритмов команду присваивания; уметь схематично описывать изменение памяти ЭВМ при исполнении команды присваивания.

ОСНОВНЫЕ ПОНЯТИЯ

В § 2 уже говорилось об ЭВМ как об универсальной информационной машине. Информация в ЭВМ представляется в виде ***величин,*** поэтому понятие величины очень важно для понимания процессов обработки информации.

Рассмотрим основные характеристики величины: имя, тип, вид и значение.

С точки зрения человека, ***имя*** величины — это ее обозначение в алгоритме. Для ЭВМ имя означает также место в памяти, где хранится значение величины.

Очень важно, что память для величин выделяется как бы внутри памяти для алгоритма. Это означает, что ***в разных (в том числе вызывающих друг друга) алгоритмах можно использовать одинаковые имена, которые будут обозначать разные величины.*** Имена величин алгоритма относятся к

его внутренней структуре. Следовательно, при вызове вспомогательного алгоритма не требуется знать имена входящих в него величин, достаточно знать типы и виды величин, описанных в заголовке алгоритма.

В учебнике не сказано, какие имена величин допустимы в алгоритмическом языке. Можно считать, что соблюдается соглашение, принятое во многих языках программирования: именем может быть любая последовательность букв, цифр и знаков подчеркивания, начинающаяся с буквы.

Тип величины задает множество допустимых значений величины и множество применимых к ней операций.

Множество значений типа **цел** теоретически совпадает с множеством целых чисел, но ресурсы ЭВМ ограничены, поэтому реально допускаются целые числа из некоторого диапазона.

Множество значений типа **вещ** теоретически совпадает с множеством действительных чисел, но и здесь возникают ограничения. Выделением диапазона уже не обойтись, так как любой отрезок числовой оси содержит бесконечно много действительных чисел. Поэтому действительные числа представляются в памяти ЭВМ приближенно, с округлением.

Цел и **вещ** — числовые типы величин. Необходимость различать их связана в первую очередь с особенностями ЭВМ: целые числа представляются в памяти ЭВМ точно, а вещественные — приближенно. Операции над целыми и вещественными величинами выполняются по разным правилам: целые вычисления производятся точно, вещественные — приближенно.

Над числовыми величинами можно выполнять пять арифметических операций и шесть операций сравнения ($<, \le, >, \ge, =, \ne$). Результат операций сравнения имеет тип **лог.** Кроме того, для числовых величин определен ряд стандартных функций.

Множество значений типа **лог** включает всего два элемента: **да** и **нет**. Над ними можно производить три логические операции (**не**, **и**, **или**) и две операции сравнения ($=, \ne$). Результат всех операций имеет тип **лог**.

Вид величины (это понятие рассмотрено в § 12 учебника) характеризует ее использование в алгоритме, роль содержащейся в величине информации. В алгоритмическом языке имеются разные виды величин: **арг**, **рез**, **арг рез**, промежуточные и др.

Имя, тип и вид величины указываются в ее описании. Это статические характеристики величины, они не могут меняться в процессе работы алгоритма.

Значение — динамическая характеристика величины, оно может многократно меняться в ходе работы алгоритма.

Зная команду присваивания, можно по-новому взглянуть на уже известные понятия.

Покажем, например, что **пока** — более универсальная команда повторения, которая может заменить цикл n **раз**,

```
нц n раз
| серия
кц
```

```
R := n
нц пока R > 0
| серия
| R := R − 1
кц
```

Сколько раз будет выполняться цикл **нц** n **раз**, если в теле цикла есть команды, меняющие значение n? Число повторений вычисляется при входе в цикл и не зависит от хода выполнения цикла. В приведенном примере цикл исполнится N **раз**, где N — значение величины n до исполнения команды повторения.

ОБЩИЕ УКАЗАНИЯ

Понятие числовой величины знакомо учащимся из математики и физики. В курсе информатики понятие величины уточняется и наполняется новым содержанием. Изучение числовых величин должно создать предпосылки для последующего изучения величин других типов (табличные, символьные, литерные, § 14, 15).

Различия между целыми и вещественными величинами довольно условны и не столь контрастны, как, например, между числовыми и литерными величинами. Поэтому задачу окончательного формирования понятия типа величины имеет смысл отложить до изучения символьных и литерных величин.

Для закрепления понятий имени, типа и вида величин можно предложить учащимся заполнить таблицу величин для какого-нибудь алгоритма. Рассмотрим, например, величины из алгоритма "парабола"(п. 11.10, А45).

Имя	Тип	Вид
a	вещественный **вещ**	аргумент **арг**
b	вещественный **вещ**	аргумент **арг**
n	целый **цел**	аргумент **арг**
x	вещественный **вещ**	промежуточная
d	вещественный **вещ**	промежуточная

Присваивание — основной способ изменения значения величины. Тип величины в левой части присваивания и тип выражения в правой части должны совпадать. Исключение: допускается присваивание вида **вещ** := **цел**.

Тип выражения определяется по типам входящих в него величин с помощью набора индуктивных правил. Например, если выражения a, b целые, то a+b целое; если a целое, b вещественное, то a+b вещественное и т.п. Некоторые сложности возникают при использовании деления. Если a и b целые, то a / b обычно вещественное число. Для получения результата деления нацело нужно писать div(a,b). Подобное различие операций деления сложно для школьников, поэтому не следует акцентировать на нем внимание и требовать его строгого соблюдения.

Модель памяти ЭВМ позволяет проиллюстрировать исполнение команды присваивания. Особое внимание нужно обратить на случай, когда величина из левой части команды присваивания входит в выражение в правой части (см. упражнения 1—4).

Важно подчеркнуть, что исполнение команды присваивания меняет значение только одной величины — той, что стоит в левой части; значения остальных величин остаются без изменения. Например, команда "y:=2*x" оставляет без изменения значение величины x, а команда "x:=2*x" увеличивает значение x вдвое.

Рассмотрим некоторые типичные ошибки, возникающие при работе с величинами.

1. Величина не описана.

2. Величина описана дважды. Примеры:

а) **нач цел** p, **вещ** p;

б) **алг** A (**арг вещ** x) ... **нач вещ** x.

3. Выполняются операции, не соответствующие типу и значению величины. Пример:

```
нач цел m, n, вещ a
 ...
n:=1;  a:=5.4
m:=mod (n,a)
```

4. Величине присваивается значение, несовместимое с ее типом. Примеры:

а)
```
нач цел n
 n:=1.3
```

б)
```
нач цел m,n
 n:=3
 m:=n/2
```

в)
```
алг A(арг цел n)
 ...
алг Б
 ...
 A(1.3)
```

5. Во время исполнения алгоритма изменяется значение величины-аргумента. Примеры:

а)
```
алг A(арг вещ x)
 ...
 x:=2.3
```

б)
```
алг A(арг цел n)
 ...
нц для n от 1 до 5
 ...
```

6.Во время исполнения команды повторения **для** изменяется значение параметра цикла; после исполнения команды повторения **для** используется значение параметра цикла. Примеры:

```
нц для i от 1 до n          нц для i от 1 до n
| i:=a+1                    | ...
| ...                       кц
кц                          a:=i+1
```

(Ситуации, связанные с циклом **для**, рассматриваются при изучении § 13.)

Заметим, что перечисленные ошибки имеют разный статус. Ошибки типа 1, 2 — синтаксические и могут быть обнаружены до выполнения алгоритма.

Ошибки типа 3, 4 — тоже синтаксические. Иногда принимаются соглашения, которые позволяют исполнить алгоритм при наличии в нем таких ошибок. Для этого, например, действительное число округляется до целого или берется его целая часть и т.п.

Примеры типа 5, 6 могут формально и не считаться ошибками. При наличии конструкций типа 5, 6 алгоритмы трудны для понимания. В конкретных реализациях алгоритмического языка на ЭВМ конструкции из примеров 5, 6 могут быть разрешены. Тем не менее следует избегать их при составлении алгоритмов.

РЕКОМЕНДАЦИИ К ВЕДЕНИЮ УРОКОВ

Анализируя выполнение алгоритмов, мы использовали условное представление памяти ЭВМ в виде прямоугольников. Значение величины попадало в ячейку при вызове алгоритма (аргумент).

Приведем пример алгоритма, в котором кроме упомянутого случая встречается промежуточная величина — параметр цикла (§ 13), хранящаяся в ячейке памяти.

```
алг закрасить теплые клетки (арг вещ t)
  дано | Робот в левой клетке коридора
       | длиной 11 клеток
  надо | закрашены клетки внутри коридора,
       | в которых температура выше t
нач
| нц 11 раз
| |   если температура > t
| |   | то закрасить
| |   все
| | вправо
| кц
кон
```


Часто бывает необходимо задавать или изменять значение величины во время выполнения алгоритма. Для этого в алгоритмическом языке используется ***команда присваивания.***

Изменим условие последней задачи. Пусть требуется закрасить клетки, в которых температура выше, чем в исходной клетке. Новый алгоритм будет таким:

```
алг закрасить теплые клетки
  дано | Робот в левой клетке коридора
       | длиной 11 клеток
  надо | закрашены клетки внутри коридора,
       | в которых температура выше,
       | чем в исходной клетке
нач вещ t, цел i
| t := температура
| нц 10 раз
| | вправо
| |   если температура > t
| |   | то закрасить
| |   все
| кц
| вправо
кон
```

При выполнении команды "t := температура" ЭВМ дает Роботу команду сообщить значение температуры, получает по обратной связи вещественное число и записывает его в ячейку, выделенную для переменной t.

Рассмотрим действия ЭВМ на примере фрагмента алгоритма:

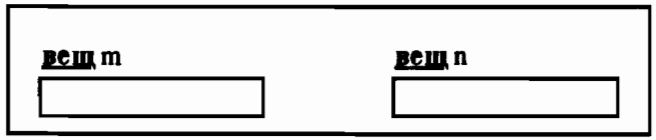
```
алг пример
  дано |
  надо |
нач вещ m,n
| m := 1
| n := m / 2 + 7
| ...
| ...
кон
```

1. Выделение области памяти:

алг пример

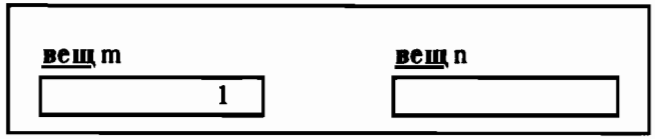
2. Выделение ячеек для величин:

алг пример



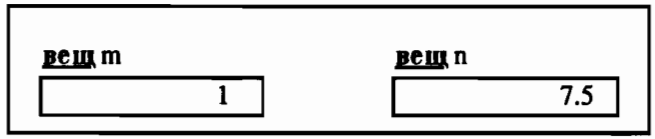
3. Выполнение первого присваивания:

алг пример



4. Выполнение второго присваивания:

алг пример



Команда присваивания применяется в трех вариантах.

Вариант 1. *переменная := константа*

Примеры:

```
F := 5
число := 7.5
l := да
m := нет
```

В о п р о с . Определите типы величин.

О т в е т . F — **цел**; число — **вещ**; l, m — **лог**.

Вариант 2. *переменная := переменная*

Примеры:

```
c := b
b := a
a := c
```

В о п р о с . Каков результат последовательного выполнения указанных команд?

О т в е т . Значения величин a и b меняются местами.

Вариант 3. *переменная := выражение*

Примеры:

```
s := a * t ** 2 / 2
цифра := mod(N, 10)
F := не снизу стена
R1 := x > 0
```

В о п р о с . Определите типы величин в левой части.

О т в е т . s — **вещ**; цифра — **цел**; F, R1 — **лог**.

При использовании команды присваивания встречается ряд характерных ошибок.

Упражнение: найдите ошибки в приведенных фрагментах:	**Ответы:**
1) **нач цел** d | d := 1 | d + 1 := 2 **кон**	выражение в левой части второй команды присваивания
2) **нач лог** n, **вещ** x | x := 3.1416 / 4 | n := sin(x) **кон**	несоответствие типов величины n и выражения в правой части
3) **нач вещ** b,y,z | y := 3 | z := 1.6 | b := 5 (y + z) **кон**	синтаксическая ошибка в правой части последней команды присваивания
4) **нач вещ** a,b | b := a / 2 | a := 0 ; b := 5 | b := b + 1 **кон**	значение величины a в правой части первой команды не определено
5) **нач лог** F,T | T := вниз | F := **не** снизу свободно **кон**	в правой части первого присваивания команда-действие
6) **нач цел** m | **если** m := 1 | | **то** m = 1 | **все** **кон**	перепутаны присваивание и сравнение

Запишем в словарь:

:= — команда присваивания

УКАЗАНИЯ К УПРАЖНЕНИЯМ

В упражнениях 1 — 4 полезно иллюстрировать решения изображением памяти ЭВМ до и после исполнения команд.

1. а) Значение величины x равно 5.

б) Значение величины x увеличивается на 5 и становится равно 8.

в) Значение величины x не изменяется и равно 3.

2. Обозначим через a значение величины x до исполнения команд. Тогда получаем:

а) $a + 5 = 3$; $a = -2$.

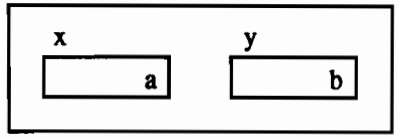
б) $-a = 3$; $a = -3$.

в) $a = 3$.

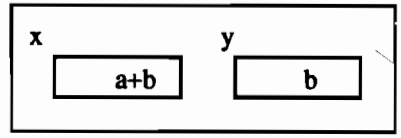
г) $a + 1 = 3$; $a = 2$.

д) $a = 3$.

3. Обозначим через a и b значения величин x и y до исполнения команды присваивания "x := x + y". Рассмотрим результат исполнения команды на модели памяти ЭВМ.



До исполнения команды

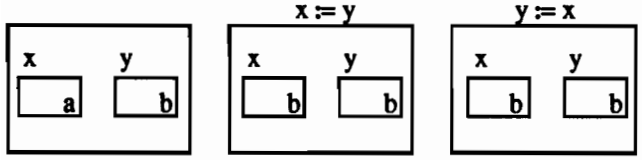


После исполнения команды

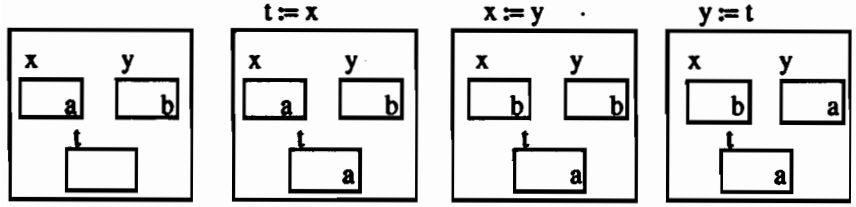
Отсюда $b = 5$ и $a + b = 3$. Значит, $a = -2$.

4. В задаче есть маленькая ловушка: если a = b, то после исполнения любой серии команд величины x и y меняются значениями. Разберем случай $a \neq b$. Для каждой серии команд изобразим последовательность состояний памяти ЭВМ.

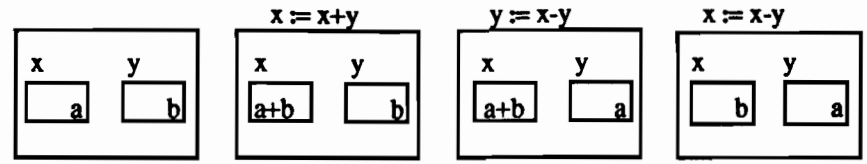
а)



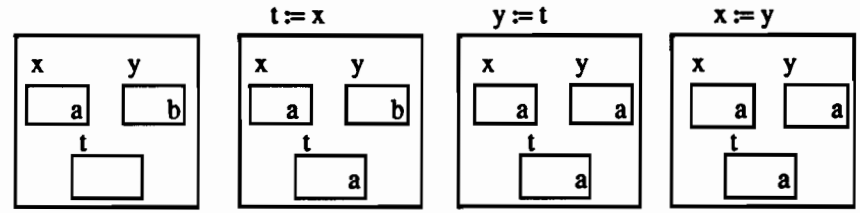
б)



в)



г)

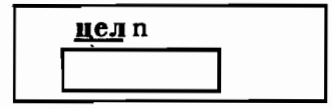


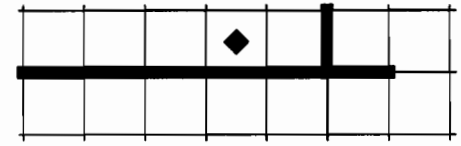
Значения величин x и y меняются в случаях б) и в).

5. Приведем схематичную запись. Начальные действия (до команды "n:=0") во всех случаях выглядят одинаково.

а) ЭВМ отводит место в памяти для алгоритма и приступает к его исполнению. Встретив описание **цел** n, отводит место для целой величины n.

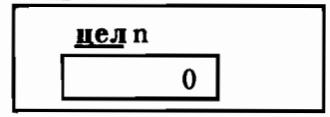
алг вниз сквозь стену

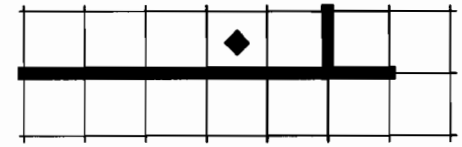




ЭВМ: n := 0

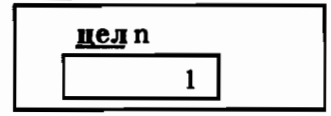
алг вниз сквозь стену

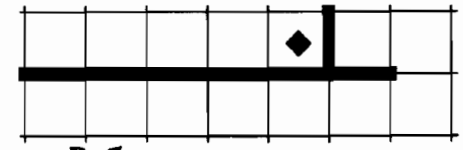




ЭВМ: снизу стена ?
Робот: да
ЭВМ: вправо
Робот: делает шаг вправо
ЭВМ: n := n + 1

алг вниз сквозь стену

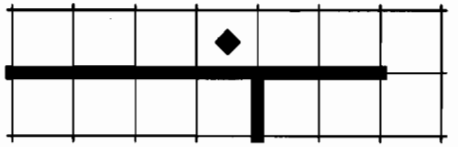




ЭВМ: снизу стена ?
Робот: да
ЭВМ: вправо
Робот: отказ

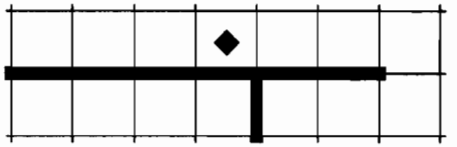
б) ЭВМ отводит место в памяти для алгоритма и приступает к его исполнению. Встретив описание **цел** n, отводит место для целой величины n.

алг вниз сквозь стену
цел n



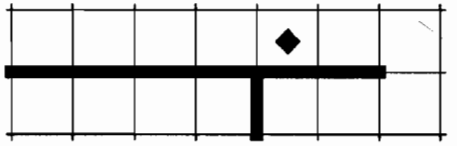
ЭВМ: n := 0

алг вниз сквозь стену
цел n
0



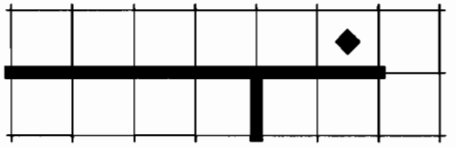
ЭВМ: снизу стена ?
Робот: да
ЭВМ: вправо
Робот: делает шаг вправо
ЭВМ: n := n + 1

алг вниз сквозь стену
цел n
1



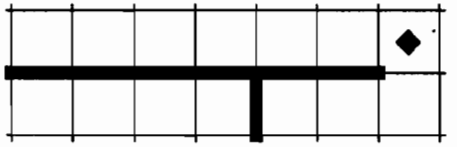
ЭВМ: снизу стена ?
Робот: да
ЭВМ: вправо
Робот: делает шаг вправо
ЭВМ: n := n + 1

алг вниз сквозь стену
цел n
2



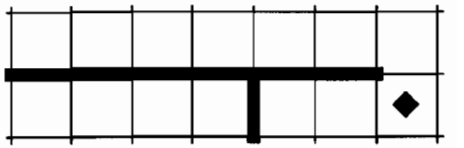
ЭВМ: снизу стена ?
Робот: да
ЭВМ: вправо
Робот: делает шаг вправо
ЭВМ: n := n + 1

алг вниз сквозь стену
цел n
3



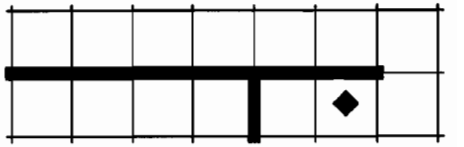
ЭВМ: снизу стена ?
Робот: нет
ЭВМ: вниз
Робот: делает шаг вниз

алг вниз сквозь стену
цел n
3



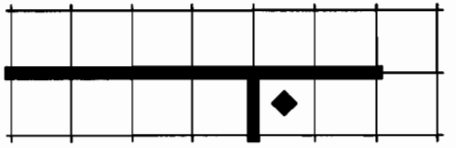
ЭВМ: влево
Робот: делает шаг влево

алг вниз сквозь стену
цел n
3



ЭВМ: влево
Робот: делает шаг влево

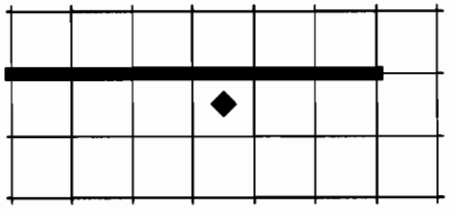
алг вниз сквозь стену
цел n
3



ЭВМ: влево
Робот: отказ

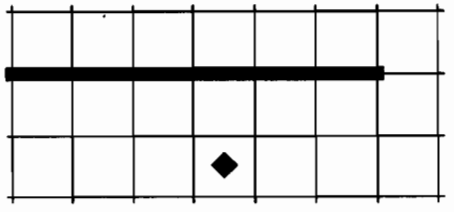
в, г) Так как снизу от Робота свободно, команды из тела цикла **пока** не выполняются. Робот делает шаг вниз. Так как n=0, команды из тела цикла n **раз** не выполняются и на этом исполнение алгоритма заканчивается. Приведем схему исполнения алгоритма для случая г).

алг вниз сквозь стену
цел n
0

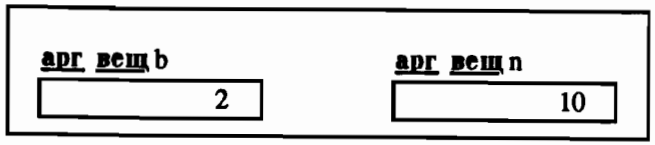


ЭВМ: снизу стена ?
Робот: нет
ЭВМ: вниз
Робот: делает шаг вниз

алг вниз сквозь стену
цел n
0



6. Разберем работу ЭВМ при исполнении вызова алгоритма "график (2, 10)". В момент вызова "график (2, 10)" память будет иметь следующий вид:



Встретив после слова **нач** описание величин x, d и v, ЭВМ отведет для них часть памяти. Роль шага по оси абсцисс играет величина d. Значение величины v — разность между ординатами соседних вершин ломаной.

Пусть x и y — координаты вершины ломаной; составим таблицу значений величин x, y, v (табл. 10).

Таблица 10

x	0	0.20	0.40	0.60	0.80	1.00	1.20	1.40	1.60	1.80	2.00
v	0	0.04	0.12	0.20	0.28	0.36	0.44	0.52	0.60	0.68	0.76
y	0	0.04	0.16	0.36	0.64	1.00	1.44	1.96	2.56	3.24	4.00

По этой таблице нетрудно определить зависимость между величинами x и y: $y = x^2$.

Тот же результат можно получить и теоретически. Заметим, что соотношение $y = x^2$ справедливо для начальной точки графика (0,0). Покажем теперь, что если перед исполнением команд из цикла n **раз** перо Чертежника находится в точке (x,y), удовлетворяющей соотношению $y = x^2$, то и после исполнения команд координаты пера Чертежника будут удовлетворять этому соотношению. Действительно, перо сместится на вектор (d, v) и окажется в точке (x+d, y+v). Так как $v = 2xd + d^2$ и по предположению $y = x^2$, то $y+v = x^2 + 2xd + d^2 = (x+d)^2$. Таким образом, все вершины ломаной лежат на параболе $y = x^2$. Подобный подход к составлению и анализу алгоритмов более подробно рассматривается в § 16.

7. Для составления алгоритма можно воспользоваться схемой, по которой составлен алгоритм рисования параболы. Это упражнение готовит учащихся к работе с алгоритмами-функциями.

```
алг график радиоактивности
  дано | Робот находится перед входом в тупик,
       | идущий вправо
  надо | нарисован график радиоактивности на промежутке от
       | 0 до x, где x — число клеток в тупике;
       | значение в нуле — уровень радиации перед входом
       | в тупик
```



```
нач вещ x,y,d
| d := 1
| x := 0; y := уровень радиации
| сместиться в точку (x,y)
| опустить перо
| нц пока справа свободно
| | вправо; x := x + d
| | y := уровень радиации
| | сместиться в точку (x,y)
| кц
| поднять перо
кон
```

Рассмотрим дополнительно еще две задачи. Пусть, как и в предыдущей задаче, Робот стоит перед входом в тупик, идущий вправо.

а) Требуется закрасить клетки, в которых уровень радиации составляет более 80% от максимального.

Составим план алгоритма. Робот проходит до конца тупика и определяет максимальный уровень радиации. Затем, двигаясь назад, Робот закрашивает клетки, уровень радиации в которых превосходит заданный. Задача типа определения максимального уровня радиации встречается часто, поэтому имеет смысл разобрать этот фрагмент алгоритма как самостоятельную задачу. Впоследствии, после изучения алгоритмов с результатами, этот фрагмент можно оформить в виде вспомогательного алгоритма.

Значение максимального уровня радиации в тупике будет храниться в памяти ЭВМ как вещественная величина с именем mx (напрашивается имя max, но это имя стандартной функции). Если Робот стоит перед входом в тупик, то значение величины mx будет найдено после исполнения следующей серии команд:

```
mx := 0
нц пока справа свободно
| вправо
| если уровень радиации > mx
| | то mx := уровень радиации
| все
кц
```

Нужно обратить внимание на команду присваивания, где в правой части стоит обращение к Роботу. В результате исполнения этой команды величина mx получает значение уровня радиации той клетки, где расположен Робот.

Перейдем к составлению полного алгоритма. Сделаем одно замечание о выходе из тупика. Можно организовать выход с помощью цикла **пока**:

```
нц пока снизу стена и сверху стена
| влево
кц
```

С учетом целей параграфа предпочтительнее несколько иной подход. При движении в тупик нужно подсчитать число шагов и выходить из тупика по счету шагов.

```
алг закраска радиоактивных клеток
  дано | Робот находится перед входом в тупик,
       | идущий вправо
  надо | закрашены все клетки тупика, в которых уровень
       | радиации составляет более 80% от максимального
нач вещ mx, цел n
| mx := 0; n := 0
| нц пока справа свободно
| | вправо; n := n + 1
| | если уровень радиации > mx
| | | то mx := уровень радиации
| | все
| кц
| утв | Робот находится в конце тупика;
|     | mx — максимальный уровень радиации в тупике;
|     | n — число клеток в тупике
| нц n раз
| | если уровень радиации > 0.8 * mx
| | | то закрасить
| | все
| | влево
| кц
кон
```

б) Требуется закрасить все клетки, в которых температура выше средней.

Алгоритм по структуре аналогичен алгоритму из предыдущего задания. Сначала нужно разобрать фрагмент алгоритма с подсчетом средней температуры. Для нахождения средней температуры нужно просуммировать температуру во всех клетках тупика и поделить сумму на число клеток в тупике. Пусть t —величина средней температуры, а n — число клеток в тупике. Тогда значение t может быть найдено после исполнения серии команд:

```
t := 0; n := 0
нц пока справа свободно
| вправо; n := n + 1; t := t + температура
кц
t := t / n
```


Нужно обратить внимание учащихся на прием, с помощью которого суммируется набор значений температур: к величине найденной суммы каждый раз добавляется очередное слагаемое.

Заметим, что выполнение предыдущей серии команд приведет к отказу, если тупика фактически нет, т.е. его длина равна 0 и n = 0. Чтобы избежать отказа в такой ситуации (и тем самым расширить область применения алгоритма), можно вместо команды присваивания "t := t / n" записать составную команду

```
если n > 0
| то t:=t/n
все
```

и считать, что если тупика нет, то средняя температура равна 0. Окончательно получаем алгоритм:

```
алг закраска горячих клеток
  дано | Робот находится перед входом в тупик,
       | идущий вправо
  надо | закрашены все клетки тупика, в которых
       | температура выше средней
нач вещ t, цел n
| t := 0; n := 0
| нц пока справа свободно
| | вправо
| | n := n + 1
| | t := t + температура
| кц
| если n > 0
| | то t := t / n
| все
| утв | Робот находится в конце тупика;
|     | t — средняя температура клеток в тупике;
|     | n — число клеток в тупике
| нц n раз
| | если температура > t
| | | то закрасить
| | все
| | влево
| кц
кон
```

8. Во всех заданиях алгоритмы составляются по одной схеме: на первом этапе Робот перемещается к стене (или в угол) с помощью команды повторения пока, при этом подсчитывается число шагов; на втором этапе Робот перемещается с учетом числа сделанных шагов.

а)
```
алг вниз до стены закрасить и вернуться
  дано | где-то ниже Робота есть стена
  надо | Робот дошел до этой стены, закрасил клетку
       | и вернулся в исходное положение
нач цел n
| n := 0
| нц пока снизу свободно
| | вниз; n:=n+1
| кц
| закрасить
| вверх на (n)
кон
```

б)
```
алг закрасить клетку в правом нижнем углу
  дано | Робот где-то внутри прямоугольника,
       | огороженного стенами, других стен нет
  надо | Робот закрасил клетку в правом нижнем углу пря-
       | моугольника и вернулся в исходное положение
нач цел m,n
| m := 0; n := 0
| нц пока снизу свободно
| | вниз; n:=n+1
| кц
| нц пока справа свободно
| | вправо; m:=m+1
| кц
| закрасить; влево на (m); вверх на (n)
кон
```

в)
```
алг отойти вдвое дальше от левой стены
  дано | где-то левее Робота есть стена, других стен нет
  надо | Робот отошел вправо от стены на расстояние,
       | вдвое большее, чем исходное
нач цел n
| n := 0
| нц пока слева свободно
| | влево; n:=n+1
| кц
| вправо на (2*n)
кон
```

г)
```
алг симметрия
  дано | где-то ниже Робота есть горизонтальная стена
       | длиной в одну клетку, других стен нет
  надо | Робот оказался в положении, симметричном
       | исходному относительно стены
```



```
нач цел n
| n:=0
| нц пока снизу свободно
| | вниз; n:=n+1
| кц
| влево; вниз; вправо; вниз на (n)
кон
```

д)
```
алг симметрия
  дано | где-то ниже Робота есть горизонтальная стена, ко-
       | торая и вправо и влево кончается, других стен нет
  надо | Робот оказался в положении, симметричном
       | исходному относительно стены
нач цел m,n
| n := 0
| нц пока снизу свободно
| | вниз; n:=n+1
| кц
| m := 0
| нц пока снизу стена
| | влево; m:=m+1
| кц
| вниз; вправо на (m); вниз на (n)
кон
```

е)
```
алг обойти прямоугольное препятствие
  дано | Робот над прямоугольником, огороженным
       | стенами, других стен нет
  надо | Робот под прямоугольником на той же вертикали
нач цел n
| вниз до стены; n:=0
| нц пока снизу стена
| | влево; n:=n+1
| кц
| вниз
| нц пока справа стена
| | вниз
| кц
| вправо на (n)
кон
```

ж)
```
алг вниз сквозь бесконечную стену
  дано | Робот над горизонтальной стеной, в одну сторону
       | уходящей в бесконечность, других стен нет
  надо | Робот под стеной на клетку ниже
       | исходного положения
```

```
нач цел n,m
| n:=1; вправо на (n); m:=n
| нц пока снизу стена
| | n:=n+1; влево на (n); m:=m—n
| | если снизу стена
| | | то n:=n+1; вправо на (n); m:=m+n
| | все
| кц
| вниз
| если m>0
| | то влево на (m)
| | иначе вправо на (—m)
| все
кон
```

Здесь величина m играет роль горизонтальной "координаты" положения Робота — в ней хранится число шагов по горизонтали от исходного положения вправо (m>0) или влево (m<0). В процессе выполнения алгоритма Робот "колеблется" около начального положения вправо-влево, с каждым разом отходя все дальше и дальше до тех пор, пока не обнаружит край стены (справа или слева). В этот момент Робот делает шаг вниз и, используя значение величины m, возвращается на нужную вертикаль — точно на клетку ниже исходного положения.

Алгоритм можно записать короче, если использовать вспомогательный алгоритм "сдвиг на (арг цел n)" и задавать направление движения Робота знаком аргумента:

```
алг вниз сквозь бесконечную стену
  дано | Робот над горизонтальной стеной, в одну сторону
       | уходящей в бесконечность, других стен нет
  надо | Робот под стеной на клетку ниже исходного
       | положения
нач цел m
| m:=1; сдвиг на (m)
| нц пока снизу стена
| | сдвиг на (—m) | возврат в исходное положение
| | m:=—m
| | если m>0
| | | то m:=m+1
| | все
| | сдвиг на (m)
| кц
| вниз
| сдвиг на (—m) | возврат на клетку ниже исходного положения
кон
```



```
алг сдвиг на (арг цел n)
нач
| если n>0
| | то вправо на (n)
| | иначе влево на (—n)
| все
кон
```

Команды "m:=—m; если m>0 то m:=m+1 все " заставляют величину m последовательно принимать значения 1,—1, 2,—2, 3,—3, 4,—4 и т.д. Эти команды обеспечивают "колебание" значения величины m, а команда "сдвиг на (m)" заставляет соответственно двигаться Робота.

9. Чтобы определить клетку с минимальным уровнем радиации, Робот должен пройти по всем клеткам прямоугольника, измеряя уровни радиации. При этом ЭВМ запоминает информацию о наименее радиоактивной клетке. Затем Робот перемещается в клетку с минимальным уровнем радиации.

Вид алгоритма существенно зависит от того, в какой форме запоминается информация о клетке с минимальным уровнем радиации. Разберем два варианта. В первом случае в памяти ЭВМ хранится величина минимального уровня радиации. Во втором — координаты клетки с минимальным уровнем радиации (относительно левого верхнего угла).

Рассмотрим алгоритмы. Одна из основных трудностей состоит в том, чтобы записать серию команд, исполняя которые Робот обойдет все клетки прямоугольника. Это можно сделать разными способами. Например:

```
нц пока справа свободно
| вправо
кц
нц пока слева свободно
| влево
кц
нц пока снизу свободно
| вниз
| нц пока справа свободно
| | вправо
| кц
| нц пока слева свободно
| | влево
| кц
кц
```

Этот алгоритм, далеко не самый эффективный, прост для понимания. Можно повысить его эффективность, применяя команду ветвления и не заставляя Робота возвращаться:

```
нц пока справа свободно
| вправо
кц
нц пока снизу свободно
| вниз
| если справа свободно
| | то
| |   нц пока справа свободно
| |   | вправо
| |   кц
| | иначе
| |   нц пока слева свободно
| |   | влево
| |   кц
| все
кц
```

Можно подойти к задаче по-другому. Сначала определить размеры прямоугольника и затем делать обход (m и n — целые величины):

```
n := 0
m := 0
нц пока справа свободно
| вправо
| n := n + 1
кц
нц пока снизу свободно
| вниз
| m := m + 1
кц
  утв | Робот в правом нижнем углу;
      | n+1 — ширина,
      | m+1 — высота прямоугольника
если справа свободно
| то
| нц n раз
| | вправо
| кц
| иначе
| нц n раз
| | влево
| кц
все
```



```
нц m раз
| вверх
| если справа свободно
| | то
| | нц n раз
| | | вправо
| | кц
| | иначе
| | нц n раз
| | | влево
| | кц
| все
кц
```

Перейдем теперь к составлению алгоритмов.

1) Пусть в памяти ЭВМ хранится величина mn — минимальный уровень радиации. В любой момент исполнения алгоритма значение этой величины — минимальный уровень радиации среди проверенных клеток.

```
алг в клетку с минимальным уровнем радиации
  дано | Робот находится в левом верхнем углу
       | прямоугольника, огороженного стенами
  надо | Робот находится в клетке с минимальным
       | уровнем радиации
нач вещ mn
| mn := радиация
| нц пока справа свободно
| | вправо
| | если радиация<mn то mn:=радиация все
| кц
| нц пока снизу свободно
| | вниз
| | если радиация<mn то mn:=радиация все
| | если справа свободно
| | | то
| | | нц пока справа свободно
| | | | вправо
| | | | если радиация<mn то mn:=радиация все
| | | кц
| | | иначе
| | | нц пока слева свободно
| | | | влево
| | | | если радиация<mn то mn:=радиация все
| | | кц
| | все
| кц
```

```
  утв | Робот находится в нижнем левом или нижнем
      | правом углу; значение величины mn — минимальный
      | уровень радиации
  если справа свободно
  | то
  | нц пока справа свободно и радиация>mn
  | | вправо
  | кц
  | иначе
  | нц пока слева свободно и радиация>mn
  | | влево
  | кц
  все
  нц пока сверху свободно и радиация>mn
  | вверх
  | если справа свободно
  | | то
  | | нц пока справа свободно и радиация>mn
  | | | вправо
  | | кц
  | | иначе
  | | нц пока слева свободно и радиация>mn
  | | | влево
  | | кц
  | все
  кц
кон
```

Вторая часть алгоритма аналогична первой. Исполняя ее, Робот начинает обход клеток прямоугольника (выходя из другого угла) и обходит их до тех пор, пока не встретит клетку с минимальным уровнем радиации. Это утверждение нетрудно обосновать. Действительно, команды перемещения выдаются Роботу только при соблюдении условия "радиация>mn". Таким образом, если Робот окажется в клетке с минимальным уровнем радиации, то ни в одной команде повторения **пока** условие не будет соблюдаться, поэтому Робот не сделает больше ни одного шага, т.е. останется в клетке с минимальным уровнем радиации.

2) В памяти ЭВМ хранятся координаты клетки с минимальным уровнем радиации. Пусть i, j — координаты текущей клетки, mn — минимальный уровень радиации, k, l — координаты клетки с минимальным уровнем радиации из числа просмотренных.

```
алг в клетку с минимальным уровнем радиации
  дано | Робот находится в верхнем левом углу
       | прямоугольника, огороженного стенами
  надо | Робот находится в клетке с минимальным
       | уровнем радиации
```



```
нач вещ mn, цел i,j,k,s
  mn := уровень радиации
  i := 0;  j := 0
  k := 0;  s := 0
  нц пока справа свободно
  | вправо
  | j := j + 1
  | если радиация<mn
  | | то mn := радиация
  | |      k := i
  | |      s := j
  | все
  кц
  нц пока снизу свободно
  | вниз; i := i + 1
  | если радиация < mn
  | | то mn := радиация
  | |      k := i;  s := j
  | все
  | если справа свободно
  | | то
  | | нц пока справа свободно
  | | | вправо; j := j + 1
  | | | если радиация < mn
  | | | | то mn := радиация
  | | | |      k := i;  s := j
  | | | все
  | | кц
  | | иначе
  | | нц пока слева свободно
  | | | влево; j := j — 1
  | | | если радиация < mn
  | | | | то mn := радиация
  | | | |      k := i;  s := j
  | | | все
  | | кц
  | все
  кц
  утв | Робот находится в правом нижнем или
      | левом нижнем углу; значение величины mn —
      | минимальный уровень радиации
  вверх до стены
  влево до стены
  утв | Робот находится в левом верхнем углу
  вниз на (k)
  вправо на (s)
кон
```

Алгоритмы получились громоздкие, с большим числом повторов. Избежать этого можно, используя вспомогательные алгоритмы с результатами. Это упражнение рекомендуется выполнить после изучения § 12.

§ 12. РЕЗУЛЬТАТЫ АЛГОРИТМОВ И АЛГОРИТМЫ-ФУНКЦИИ

> Никто не научится хорошо составлять большие программы, пока он не научится хорошо составлять малые.
>
> *Дэвид Грис*

В параграфе вводится понятие вида величины, рассматриваются алгоритмы с результатами и алгоритмы-функции, даются правила работы с такими алгоритмами, описывается работа ЭВМ при вызове вспомогательных алгоритмов.

Основные цели. Выработать у учащихся навыки применения алгоритмов с результатами и алгоритмов-функций, научить их правильно оформлять такие алгоритмы, понимать ход их выполнения на ЭВМ, дать представление о механизме передачи параметров (аргументов и результатов) вспомогательных алгоритмов.

Требования к знаниям и умениям. Учащиеся должны уметь определять вид величины; составлять и применять алгоритмы с результатами и алгоритмы-функции; исполнять алгоритмы и находить их результаты.

ОСНОВНЫЕ ПОНЯТИЯ

Результатом всех ранее рассмотренных алгоритмов были действия исполнителя. В § 12 вводятся алгоритмы, результатом которых является переданная в основной алгоритм информация в форме ***величин-результатов.***

Величины-аргументы и результаты называют ***параметрами*** алгоритма. Параметры обеспечивают явную информационную связь основных и вспомогательных алгоритмов, позволяют писать содержательные алгоритмы, не использующие исполнителей.

При вызове устанавливается связь между ***формальными*** параметрами, описанными в заголовке вспомогательного алгоритма, и ***фактическими,*** указанными в команде вызова в основном алгоритме.

Связь устанавливается строго по порядку следования: первый фактический соответствует первому формальному и т.д. Следовательно, количество формальных и фактических параметров

должно совпадать, т.е. параметры должны быть ***согласованы по числу.***

Кроме того, формальные и фактические параметры должны быть ***согласованы по типу.*** Фактический параметр должен быть выражением (для аргументов) или именем величины (для результатов) того же типа, что и соответствующий формальный параметр.

Передачу параметров можно представить в виде присваивания "формальный := фактический" для аргументов и "фактический := формальный" для результатов. Такое представление объясняет необходимость согласования типов и невозможность использования выражений в качестве фактических параметров-результатов.

С использованием параметров-результатов связан один сложный вопрос. Что произойдет, если двум разным формальным результатам поставить в соответствие одну и ту же фактическую величину? Например, так вызвать алгоритм А49 из п. 12.5: "квур (1, −6, 5, n, x, x)". Результаты такого вызова непредсказуемы и существенно зависят от того, как реализован в ЭВМ механизм передачи параметров. Мы будем считать, что подобные вызовы запрещены правилами алгоритмического языка.

Величина может быть одновременно аргументом и результатом, в этом случае она описывается в алгоритме как **арг рез** (см. упражнение 5, с. 114 учебника).

Алгоритмы-функции принципиально ничем не отличаются от обычных алгоритмов с результатами. Разница лишь в оформлении самого алгоритма и его вызова.

Вызов обычного алгоритма — это отдельная команда основного алгоритма. Вызов функции не требует специальной команды, он может использоваться в любом выражении в качестве величины, тип которой определяется типом результата функции.

Особенности оформления функции связаны с особенностями ее использования. Результат функции не включается в список формальных параметров (это нарушило бы согласование по числу). Тип результата указывается перед именем функции, что соответствует использованию функции как величины в выражениях.

Имя результата с точки зрения основного алгоритма совпадает с именем функции. Кажется естественным использование имени функции в качестве имени результата и в самой функции, но при этом становится невозможной рекурсия (см. п. 16.11). Обращение функции к самой себе будет восприниматься как использование текущего значения результата. Поэтому в качестве имени результата в алгоритме-функции используется специальное служебное слово **знач**.

Во всех примерах учебника **знач** — единственный результат функции. Вообще говоря, функция может иметь и результаты, описанные обычным образом, с помощью слова **рез**. Такие результаты называются *побочным эффектом функции*. В учебнике и пособии функции с побочным эффектом не используются.

При использовании готовых функций нас по-прежнему может не интересовать внутренний механизм работы функции. В этом смысле функции аналогичны командам обратной связи исполнителей.

ОБЩИЕ УКАЗАНИЯ

До сих пор изучались алгоритмы, результатами которых были изменения в среде исполнителя, их выполнение иллюстрировалось рисунками поля Робота или Чертежника. Пошаговое исполнение алгоритмов с величинами-результатами можно проследить только по схеме памяти ЭВМ.

Понятие алгоритма-функции можно сопоставить с понятием функции в курсе математики. Сходство достаточно очевидно, а вот вопрос об отличиях является более тонким. Может показаться несколько неожиданным пример алгоритма-функции без аргументов (упражнение 7, в,г, с.102 учебника). В школьной математике такое невозможно.

Алгоритм-функция с аргументами может при разных вызовах выдавать разные результаты для одних и тех же значений аргументов. Это объясняется тем, что результаты алгоритмов-функций зависят не только от значений аргументов, но и от состояния среды исполнителя. Чтобы застраховаться от ошибок, в информатике обычно придерживаются такого правила: после исполнения алгоритма-функции состояние памяти ЭВМ (исключая величину-результат) и среды исполнителя остаются прежними.

РЕКОМЕНДАЦИИ К ВЕДЕНИЮ УРОКОВ

Вызов любой команды исполнителя приводит к какому-нибудь результату. Результаты можно условно разделить на два вида: изменение среды исполнителя и величины.

Команда Робота "вниз" изменяет его положение, команда "закрасить" меняет обстановку на поле Робота (появляется закрашенная клетка). Это результаты первого вида. Команда "снизу свободно" дает в результате величину типа **лог**, команда "радиация" имеет результат типа **вещ**. Это результаты второго вида.

Мы уже говорили, что вызов команды исполнителя по существу ничем не отличается от вызова вспомогательного алгоритма. Поэтому все сказанное о командах в полной мере можно отнести к алгоритмам. Однако пока все составленные нами алгоритмы получали в качестве результата изменения среды исполнителя. В алгоритмическом языке могут использоваться вспомогательные алгоритмы, дающие результаты второго вида — величины.

Рассмотрим задачу:

```
алг маркировка теплых клеток
  дано | Робот перед входом в тупик,
       | идущий вправо
  надо | закрашены клетки, температура
       | в которых выше средней
```

Сначала ввиду простоты задачи постараемся обойтись без пошаговой детализации. Тело алгоритма будет таким:

```
нач вещ s, цел n
| s := 0 | суммарная температура
| n := 0 | число шагов в тупике
| нц пока справа свободно
| | вправо
| | n :=n + 1
| | s :=s + температура
| кц
| утв | n > 0
| | s := s / n    | s — средняя температура
| нц n раз
| | если температура > s
| | | то закрасить
| | все
| | влево
| кц
кон
```

Теперь решим ту же задачу методом последовательного уточнения. Конечно, может показаться, что в данном случае этот метод лишь усложнит решение. Но, разобравшись в процедуре *передачи параметров* на простом примере, мы сможем перейти к более содержательным задачам.

```
алг маркировка теплых клеток
  дано | Робот перед входом в тупик,
       | идущий вправо
  надо | закрашены клетки, температура
       | в которых выше средней
нач вещ s, цел n
| найти среднюю температуру (s, n)
| разметить тупик (s, n)
кон
```

Алгоритм "найти среднюю температуру" выдает в результате значение средней температуры (типа **вещ**) и число клеток в тупике (типа **цел**). Составим его:

```
алг найти среднюю температуру (рез вещ s, цел l)
  дано | Робот перед входом в тупик,
       | идущий вправо
  надо | s — средняя температура
       | l — длина тупика
нач
| s:=0;   l:=0
| нц пока справа свободно
| | вправо
| | l:=l+1
| | s:=s+температура
| кц
| утв l>0  |  s — суммарная температура
| s:=s/l   |  s — средняя температура
кон
```

Второй вспомогательный алгоритм:

```
алг разметить тупик (арг вещ s, цел l)
  дано | s — средняя температура
       | l — длина тупика
  надо | закрашены клетки, температура
       | в которых выше средней
нач
| нц l раз
| | если температура > s
| | то закрасить
| | все
| | влево
| кц
кон
```

Почему величины s и l описаны трижды (в основном алгоритме как промежуточные, а в двух вспомогательных как аргументы и результаты)? Дело в том, что первый и второй вспомогательные алгоритмы обладают самостоятельностью, взаимно независимы, и одинаковые имена переменных в них использованы только для облегчения понимания их совместной работы.

На практике совпадение имен вовсе не обязательно. Более того, эти алгоритмы могут быть разработаны разными алгоритмистами (программы пишут программисты, а алгоритмы — видимо, алгоритмисты).

В подтверждение сказанного изменим имена величин в основном и во втором вспомогательном алгоритмах:

```
алг маркировка теплых клеток
  дано | Робот перед входом в тупик,
       | идущий вправо
  надо | закрашены клетки, температура
       | в которых выше средней
нач вещ T, цел D
| найти среднюю температуру (T, D)
| разметить тупик (T, D)
кон

алг разметить тупик (арг вещ t, цел d)
  дано | t — средняя температура
       | d — длина тупика
  надо | закрашены клетки, температура
       | в которых выше средней
нач
| нц d раз
| | если температура > t
| | то закрасить
| | все
| | влево
| кц
кон
```

Как работает каждый из вспомогательных алгоритмов в отдельности, понятно. Важно разобраться в их взаимодействии, в том, как первый из них *передает* результаты второму. Процесс *передачи параметров* в данном случае происходит по такой схеме:

```
алг найти среднюю температуру (рез вещ s, цел l)
                                  ↓      ↓
        основной алгоритм:        T      D
                                  ↓      ↓
алг разметить тупик (арг вещ t, цел d)
```

На уроке можно "оживить" анализ алгоритма, вызвав к доске учеников (этот прием уже упоминался). В рассматриваемом примере потребуются четверо. Пусть для определенности алгоритм "маркировка теплых клеток" выполняет ученик *Авербух*, алгоритм "найти среднюю температуру" — ученик *Гисин*, алгоритм "разметить тупик" — ученик *Зайдельман*, наиболее ответственную роль Робота поручим отличнику *Лебедеву*.

Рассмотрим ход выполнения алгоритмов для тупика, изображенного на рисунке 60. В каждой клетке указано значение температуры.

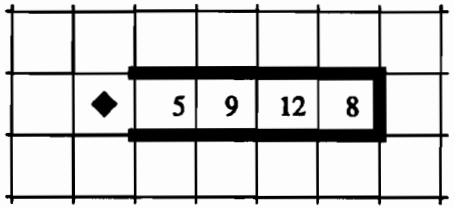


Рис. 60

Авербух рисует на доске область памяти для своего алгоритма, выделяет ячейки для величин:

алг маркировка теплых клеток

вещ T	**цел** D

Затем *Авербух* вызывает к доске *Гисина*.

Гисин выделяет на доске часть памяти для своего алгоритма. Затем выполняет первую строчку (команды присваивания):

алг найти среднюю температуру

вещ s	**цел** l
0	0

Гисин. Справа свободно?

Лебедев. Да.

Гисин. Вправо!

Лебедев перемещает Робота на один шаг вправо.

Гисин увеличивает значение l на единицу, после чего обращается к *Лебедеву:* "Температура?"

Лебедев. Пять.

Гисин заносит полученное число в ячейку s...

В этом месте мы пропустим часть диалога и обратимся сразу к моменту окончания выполнения первого вспомогательного алгоритма.

Гисин выполняет последнее присваивание (значение s в этот момент равно 34):

алг найти среднюю температуру

вещ s	**цел** l
8.5	4

Гисин сообщает значения величин *Авербуху:* "Первая величина равна восьми с половиной, вторая — четырем". После этого стирает с доски свою часть памяти и садится на место.

Авербух заносит полученные числа (в порядке их поступления) в соответствующие ячейки:

алг маркировка теплых клеток

вещ T	**цел** D
8.5	4

Авербух вызывает к доске *Зайдельмана*.

Зайдельман выделяет область памяти с ячейками для своего алгоритма, спрашивает у *Авербуха* значения величин.

Авербух. Первая величина равна восьми с половиной, вторая —четырем.

Зайдельман заносит значения величин в ячейки:

алг разметить тупик

вещ t	**цел** d
8.5	4

Зайдельман. Температура?
Лебедев. Восемь.
Зайдельман. Влево!
Лебедев перемещает Робота на один шаг влево.
Зайдельман. Температура?
Лебедев. Двенадцать.
Зайдельман. Закрасить!
Лебедев закрашивает клетку.
Зайдельман. Влево! ..
И так далее.

Закончив выполнение своего алгоритма, *Зайдельман* стирает с доски свою область памяти и возвращается на место. *Авербух* уходит на место, не стирая свою область памяти, но стерев ячейки с промежуточными величинами.

Таким образом, на доске остается чистая область памяти основного алгоритма и поле Робота, на котором вторая и третья клетки тупика закрашены.

В о п р о с . Какие результаты были выданы первым вспомогательным алгоритмом?

О т в е т . Значение средней температуры, число шагов и новое положение Робота (в восточной клетке тупика).

В о п р о с . Каковы результаты второго вспомогательного алгоритма?

О т в е т . Новая обстановка на поле Робота — он вернулся в исходное положение, нужные клетки закрашены.

В о п р о с . Каковы результаты основного алгоритма?

О т в е т . Они совпадают с результатами второго вспомогательного алгоритма.

Решим еще одну задачу. Робот в верхнем левом углу прямоугольника размером 10 × 6. Найти число закрашенных клеток внутри прямоугольника.

```
алг число закрашенных клеток (рез цел k)
  дано | Робот в верхнем левом углу
       | прямоугольника размером 10 × 6
  надо | k — число закрашенных клеток
нач
| k := 0
| нц 3 раз
| | подсчет вправо (k)
| | вниз
| | подсчет влево (k)
| | если снизу свободно
| | | то вниз
| | все
| кц
кон
```

Алгоритм "подсчет влево" аналогичен алгоритму "подсчет вправо", приведенному ниже.

```
алг подсчет вправо (арг рез цел m)
  дано | Робот в западной клетке полосы
  надо | найдено число закрашенных
       | клеток на полосе
```



```
нач
| нц 9 раз
| | если клетка закрашена
| | | то m := m + 1
| | все
| | вправо
| кц
| если клетка закрашена
| | то m :=m + 1
| все
кон
```

Рассмотренный пример интересен тем, что и основной, и вспомогательные алгоритмы занимаются подсчетом одного параметра, передавая его значение друг другу как эстафетную палочку.

Во вспомогательных алгоритмах встретился новый вид величины: аргумент-результат (арг рез).

Теперь мы знаем все четыре вида величин в алгоритмическом языке:

1) аргументы;
2) промежуточные;
3) результаты;
4) аргументы-результаты.

Познакомившись с результатами и аргументами-результатами, мы можем решать не только задачи управления автоматическими устройствами, но и многие задачи из математики, физики и т.д.

Попробуем, например, вычислить вероятность максимального выигрыша в спортлото 6 из 45. Для этого надо подсчитать число возможных комбинаций. Обратное ему число и будет искомой вероятностью. Расчет числа комбинаций сделаем по известной из математики формуле:

$$C_{45}^{6}=\frac{45!}{6!\,(45-6)!}.$$

Восклицательный знак вовсе не требует кричать при чтении записанного выражения. Это знак математической операции, называется он ***факториал*** и обозначает произведение всех натуральных чисел от единицы до указанного числа.

Например: $6! = 1\cdot2\cdot3\cdot4\cdot5\cdot6 = 720$.

```
алг спортлото 6 из 45 (рез цел c, рещ b)
  надо | b — вероятность выигрыша
       | c — число комбинаций
```

```
нач цел m1,m2,m3
| факториал(45, m1)
| факториал(6, m2)
| факториал(39, m3)
| c := m1 / (m2 * m3)
| b := 1 / c
кон

алг факториал (арг цел n, рез цел m)
  дано | n > 0
  надо | m = 1*2*3*4* ... *n
нач цел k
| m := 1; k := 2
| нц пока k ≤ n
| | m := k * m; k := k + 1
| кц
кон
```

Заметим в скобках, что приведенный алгоритм дает малую точность вычислений, и можно предложить более хороший вариант, упростив выражение.

Сравнив основной алгоритм с формулой вычисления, мы видим, что в алгоритме простота и наглядность потеряны. Дело в том, что выражения типа 45! в математике обозначают как операцию, так и ее результат. А в записи "факториал (45, m1)" на алгоритмическом языке имя ("факториал") и результат алгоритма (m1) не совпадают.

В алгоритмическом языке предусмотрена возможность назвать одинаково алгоритм и его результат. (Сравните: запись "температура" обозначает одновременно и команду Робота и ее результат.) Конечно, при этом результатов может быть не более одного.

Вычисление факториала можно записать так (см. также с. 106 учебника):

```
алг цел факт (арг цел n)
  дано | n > 0
  надо | знач = 1*2*3*4* ... *n
нач цел k
| знач := 1; k := 2
| нц пока k ≤ n
| | знач := k * знач; k := k + 1
| кц
кон
```

Записанный алгоритм вычисления факториала отличается тем, что перед его именем указан тип (цел), а внутри использовано новое служебное слово знач. Смысл этих изменений понятен, если учесть *совпадение имени алгоритма и имени его результата.* Такие алгоритмы называются ***алгоритмами-функциями.***

Исправим основной алгоритм, используя составленный алгоритм-функцию:

```
алг спортлото 6 из 45 (рез цел c, вещ b)
  дано |
  надо | c — число комбинаций
       | b — вероятность выигрыша
нач
| c := факт(45) / (факт(6) * факт(39))
| b := 1 / c
кон
```

Выполнение алгоритма дает такие результаты: c = 8145060; b = 1.227738E—7.

Запись значения величины b требует пояснений. Это так называемая ***показательная*** или ***экспоненциальная*** форма представления числа. (Сравните значения слов "показывать" и "экспонировать".) Для простоты можно считать, что буква "E" заменяет словосочетание "умножить на 10 в степени". Тогда

b = 1.227738 *умножить* на 10 в *степени* —7 = 1.227738×10^{-7}.

Поскольку уже существует, по крайней мере, два варианта спортлото и возможно появление новых, то составим алгоритм для любого случая игры:

```
алг спортлото (арг цел a, d, рез цел c, вещ b)
  дано a > d  | вариант угадывания: d из a
  надо | c — число комбинаций
       | b — вероятность выигрыша
нач
| c := факт(a) / (факт(d) * факт(a — d))
| b := 1 / c
кон
```

Заметим, что мы вновь воспользовались алгоритмом-функцией "факт". Записав ее в память ЭВМ, мы можем ее применять наряду со стандартными функциями (табл. на с. 56 учебника).

Вызов "спортлото (36, 5, сочетаний, вероятность)" даст такие результаты:

сочетаний = 376992;

вероятность = 2.652576E—6.

Некоторые выводы:

1. Не существует задач, которые нельзя решить без алгоритма-функции. Алгоритм-функция лишь позволяет сделать запись более наглядной.

2. Так как имя алгоритма-функции обозначает также ее результат, то в заголовке перед именем надо указать тип результата.

3. У алгоритма-функции может быть не более одного результата (хотя в процессе работы может измениться среда исполнителя как "побочный эффект").

4. В теле алгоритма-функции ее результат обозначается служебным словом **знач**.

5. Вероятность выигрыша в спортлото "5 из 36" больше, чем в игре "6 из 45". Впрочем, обе вероятности выигрыша очень близки к нулю.

Остается рассмотреть работу памяти ЭВМ при вызове алгоритма-функции. Сделаем это на примере из алгоритма "спортлото 6 из 45":

c := факт (45) / (факт (6) * факт (39))

Встретив вызов "факт (45)", ЭВМ выделяет в памяти область такого вида:

алг факт

цел знач	**арг цел** n
	45

В конце выполнения алгоритма-функции память примет такой вид (значение k при выходе из цикла стерлось):

алг факт

цел k	**цел знач**	**арг цел** n
	1.196222E+56	45

Затем полученное значение подставляется в основной алгоритм:

c := 1.196222E+56 / (факт (6) * факт (39))

а вся область памяти алгоритма-функции стирается.

После этого выподняется вызов "факт (6)". ЭВМ вновь выделяет в памяти место для алгоритма-функции "факт":

алг факт

цел знач	**арг цел** n
	6

Подстановка полученного значения в основной алгоритм после окончания работы алгоритма-функции приводит к такой записи:

c := 1.196222E+56 / (720 * факт(39))

Аналогично происходит третий вызов алгоритма-функции. В результате вызова строчка в основном алгоритме примет такой вид:

c := 1.196222E+56 / (720 * 2.039788E+46)

Потом выполняются вычисления:

c := 8145060

Наконец, выполняется команда присваивания, т.е. в ячейку "c" основного алгоритма записывается полученное значение.

Запишем в словарь:

рез — величина-результат;

арг рез — величина, одновременно аргумент и результат;

знач — результат алгоритма-функции.

Приведем варианты заданий для итоговой проверочной работы.

Задание 1

Впишите комментарии после **дано** и **надо**. Исполните алгоритм при заданном значении x. Изобразите в тетради схемы памяти ЭВМ в моменты исполнения строки, отмеченной звездочками.

Вариант 1

x = 365

```
алг сцч (арг цел x, рез цел y)
  дано x ≥ 0 |
  надо |
нач
| y := 0
| нц пока x > 0
|   y := y + mod(x, 10)
|   x := int(x/10)
|     | ***
| кц
кон
```

Вариант 2

```
x = 412

алг чзн (арг цел x, рез цел y)
  дано x ≥ 0 |
  надо |
нач
| y := 0
| нц пока x > 0
| | y := 10 * y + mod (x,10)
| | x := int (x/10)
| | | ***
| кц
кон
```

Задание 2

На поле Робота имеется тупик шириной в одну клетку, ограниченный стенами и имеющий неизвестную длину. Тупик протянулся слева направо. Вход в тупик слева. Некоторые клетки внутри тупика закрашены. Робот стоит возле входа в тупик.

Вариант 1

Составьте алгоритм нахождения средней температуры в закрашенных клетках тупика. Учтите, что закрашенных клеток может не оказаться.

Вариант 2

Составьте алгоритм нахождения максимального значения радиации в тех клетках тупика, которые не закрашены. Учтите, что минимальная возможная радиация равна нулю.

Задание 3

При выполнении задания разрешается использовать алгоритмы-функции, приведенные в учебнике.

Вариант 1

Даны стороны треугольника. Составьте алгоритм нахождения максимальной высоты.

Вариант 2

Даны три числа. Составьте алгоритм нахождения числа с наименьшей дробной частью.

УКАЗАНИЯ К УПРАЖНЕНИЯМ

1.а)
```
алг расстояние до стены вправо (рез цел n)
  дано | правее Робота есть стена
  надо | положение Робота не изменилось, n — число шагов,
       | которое надо сделать Роботу, чтобы подойти к стене
       | вплотную (т.е. число клеток между Роботом и стеной)
нач
| n:=0
| нц пока справа свободно
| | вправо; n:=n+1
| кц
| влево на (n)
кон
```

б)
```
алг расстояния вправо и влево (рез цел n1,n2)
  дано | правее и левее Робота есть стены
  надо | положение Робота не изменилось, n1 — расстояние
       | вправо до стены, n2 — расстояние влево до стены
нач
| расстояние до стены вправо (n1)
| расстояние до стены влево (n2)
кон
```

Вспомогательный алгоритм "расстояние до стены вправо" смотрите при решении упражнения 1,а; вспомогательный алгоритм "расстояние до стены влево" записывается аналогично.

в)
```
алг к ближайшей стене вправо или влево
  дано | правее и левее Робота есть стены
  надо | Робот подошел к ближайшей из этих стен
нач цел n1,n2
| расстояния вправо и влево (n1,n2)
| если n1<n2
| | то вправо до стены
| | иначе влево до стены
| все
кон
```

г) Составим план. Робот проходит до конца тупика, при этом вычисляются средняя температура и средний уровень радиации. Затем Робот выходит из тупика, и во время выхода подсчитывается число опасных клеток.

```
алг подсчет числа опасных клеток (рез цел n)
  дано | Робот на клетку левее тупика, уходящего вправо
  надо | положение Робота не изменилось, n — число клеток,
```

```
        | в которых и температура и уровень радиации
        | выше средних значений в тупике
нач вещ r,t, цел m
  m:=0;  r:=0;  t:=0
  нц пока справа свободно
    вправо;  m:=m+1 | m — число клеток
    r := r+радиация | r — сумма уровней радиаций
    t := t+температура | t — сумма температур
  кц
  r:=r/m | средний уровень радиации в тупике
  t:=t/m | среднее значение температуры в тупике
  n:=0
  нц пока снизу стена и сверху стена
    если радиация>r и температура>t
      то n: = n+1
    все
    влево
  кц
кон
```

```
д) алг биквур (арг вещ p,q, рез цел n, рез вещ x1,x2,x3,x4)
    дано | p и q — коэффициенты биквадратного уравнения
         | x**4+p*x**2+q=0
    надо | n—число разных корней, x1—x4 — корни (если есть)
  нач вещ y1,y2,d, цел m | используем обозначения y=x**2
                         | d:=p*p/4—q
    выбор
      при d<0 : y1:=—1;  y2:=—1
      при d=0 : y1:=—p/2
      при d>0 : y1:=—p/2+sqrt(d);  y2:=—p/2—sqrt(d)
    все
    утв | либо y1<0 и y2<0, либо y1>y2
    выбор
      при y1>0 и y2>0 : n:=4;  x1:=—sqrt(y1);  x2:=sqrt(y1)
                x3:=—sqrt(y2);  x4:=sqrt(y2)
      при y1>0 и y2=0 : n:=3;  x1:=—sqrt(y1);  x2:=sqrt(y1)
                x3:=0
      при y1>0 и y2<0  n:=2;  x1:=—sqrt(y1);  x2:=sqrt(y1)
      при y1=0 : n:=1;  x1:=0
      иначе n:=0
    все
  кон
```

В первом выборе специальное значение "—1" используется как признак отсутствия корней в соответствующем случае (так как y≥0).

2. Упражнение выполняется по аналогии с тем, как это сделано в п.12.11. ЭВМ выдает Чертежнику 13 команд:

```
сместиться в точку (0, 2 )
опустить перо
сместиться в точку (0.2, 1.8)
сместиться в точку (0.4, 1.6)
сместиться в точку (0.6, 1.4)
сместиться в точку (0.8, 1.2)
сместиться в точку (1.0, 1.0)
сместиться в точку (1.2, 1.44)
сместиться в точку (1.4, 1.96)
сместиться в точку (1.6, 2.56)
сместиться в точку (1.8, 3.24)
сместиться в точку (2.0, 4.0)
поднять перо
```

Заметим, что число команд, выданных Чертежнику, превышает на 3 значение величины n: в теле цикла выдается по одной команде при каждом проходе, две команды выдаются до цикла и одна — после.

3. а) Из курса физики известно, что (если пренебречь сопротивлением воздуха) максимальная высота подъема тела, брошенного вертикально вверх со скоростью v, определяется по фомуле $h = v^2 / (2g)$, где g — ускорение свободного падения. Эту формулу нетрудно вывести. Пусть t —время, прошедшее после броска. Скорость движущегося тела равна v—gt. Максимальная высота достигается в момент, когда тело прекращает подниматься и начинает падать, т. е. когда v—gt=0. Таким образом, максимальная высота достигается при t=v/g. Отсюда

$$h=gt^2/2=g(v/g)^2/2=v^2/(2g).$$

Примем g равным 9.8. Получаем алгоритм:

```
алг вещ максимальная высота подъема 1 (арг вещ v)
  дано v≥0 | тело брошено вертикально вверх со скоростью v
  надо | найдена максимальная высота подъема
нач
| знач := (v * v) / (2 * 9.8)
кон
```

б) Вертикальная составляющая скорости тела, брошенного под углом α к горизонту со скоростью v, равна в момент броска v*sin α. Максимальная высота подъема полностью определяется значением этой величины. Воспользуемся алгоритмом из предыдущего задания:

```
алг вещ максимальная высота подъема 2 (арг вещ v, α)
  дано v≥0 и α≥0 и α≤3.14/2 | тело брошено со скоростью v
       | под углом α к горизонту
  надо | найдена максимальная высота подъема
нач
| знач := максимальная высота подъема 1 (v * sin(α))
кон
```

Можно составить алгоритм и непосредственно:

```
нач
| знач := (v * sin(α)) ** 2 / (2 * 9.8)
кон
```

Заметим, что, используя алгоритм "максимальная высота подъема 2", можно получить максимальную высоту подъема тела, брошенного вертикально вверх, в результате вызова "максимальная высота подъема 2 (v, 3.14/2)".

При составлении алгоритмов в задачах в)—д) возможны различные подходы в зависимости от того, насколько существенно используется заданное количество чисел — три. В "переборном" варианте явно выписываются условия, при которых получается соответствующий результат. В "универсальном" варианте моделируются методы работы с табличными величинами, здесь по очереди обрабатываются аргументы a, b и c. Целесообразно провести сопоставление этих двух вариантов, например, на задаче в).

в) Составим "переборный" алгоритм:

```
алг цел количество максимальных (арг вещ a, b, c)
  дано |
  надо | знач = количество максимальных чисел
       | среди трех заданных
нач
| выбор
|   при a=b и b=c : знач:=3
|   при a=b и b>c или
|        a=c и c>b или
|        b=c и c>a : знач:=2
|   иначе      знач:=1
| все
кон
```

Этот алгоритм прост по структуре, но плохо поддается модификациям. Если, например, требуется найти количество максимальных чисел среди четырех, то алгоритм придется составлять фактически заново.

Рассмотрим другой, более универсальный алгоритм:

```
алг цел количество максимальных 2 (арг вещ a, b, c)
  дано |
  надо | знач = количество максимальных чисел
       | среди трех заданных
нач вещ x
| x := max (a,b)
| x := max (x,c)
| утв | x = max (a,b,c)
| знач:=0
| если a=x то знач:=знач+1 все
| если b=x то знач:=знач+1 все
| если c=x то знач:=знач+1 все
кон
```

Если нужно найти количество максимальных среди четырех чисел a, b, c, d, достаточно добавить к последнему алгоритму команды "x := max (x, d)" и

```
если d=x то знач:=знач+1 все
```

Вторая часть алгоритма (после команды утв) — это подсчет количества чисел, удовлетворяющих заданному условию, в нашем случае чисел, равных x. Если бы потребовалось найти количество чисел, обладающих каким-нибудь другим свойством, достаточно было бы заменить условие в командах ветвления. Например, для подсчета количества чисел, равных половине максимума, достаточно вместо a=x, b=x, c=x записать a=x/2, b=x/2, c=x/2.

Заметим, что в методическом плане составление алгоритма "количество максимальных 2" готовит учащихся к работе с табличными величинами. При подсчете числа максимальных элементов в таблице алгоритм был бы составлен из двух циклов: в первом определяется максимальный элемент, во втором — подсчитывается число элементов равных ему.

Приведем еще один алгоритм, также достаточно универсальный, но основанный на несколько иной идее. На каждом шаге подсчитывается количество максимальных чисел среди просмотренных. Требуемое значение получается после просмотра всех чисел.

```
алг цел количество максимальных 3 (арг вещ a, b, c)
  дано |
  надо | знач = количество максимальных чисел
       | среди трех заданных
```

```
нач вещ x
| x := a
| знач := 1
| если b ≥ x
| | то
| | выбор
| | | при b = x: знач := знач + 1
| | | при b > x: знач := 1; x := b
| | все
| все
| если c ≥ x
| | то
| | выбор
| | | при c = x: знач := знач + 1
| | | при c > x: знач := 1; x := c
| | все
| все
кон
```

Для сохранения единообразия обработки величин b и c в алгоритме имеется "лишняя" команда: можно обойтись без присваивания "x:=c".

г) "Переборный" алгоритм:

```
алг цел количество разных (арг вещ a, b, c)
  дано |
  надо | знач = количество разных чисел
       | среди трех заданных
нач
| выбор
| | при a = b и b = c: знач := 1
| | при a = b или b = c или c = a: знач := 2
| | иначе знач := 3
| все
кон
```

Здесь важен порядок условий в команде "выбор", например, если переставить местами первое и второе, то при a=b=c будет получено неверное значение.

д) Уточним постановку задачи. Обозначим через u минимальное, а через v максимальное из чисел a, b, c. Если все три числа различны, будем называть средним то число x, которое удовлетворяет неравенствам u<x<v. Если среди заданных чисел имеется два максимальных, то средним будем считать максимальное число, при наличии двух минимальных — минимальное. Для составления алгоритма воспользуемся алгоритмами из пп. в) и г).


```
алг вещ среднее из трех (арг вещ a, b, c)
  дано |
  надо | знач = среднее по величине число
       | среди трех заданных
нач вещ u,v
| u := min (min (a,b), c ); v := max (max (a,b), c)
| если количество разных (a, b, c) = 3
| | то
| | выбор
| | | при u<a и a<v : знач := a
| | | при u<b и b<v : знач := b
| | | при u<c и c<v : знач := c
| | все
| | иначе
| | если количество максимальных (a, b, c) = 2
| | | то     знач := u
| | | иначе знач := v
| | все
| все
кон
```

е)
```
алг вещ расстояние до отрезка (арг вещ x)
      дано |
      надо | знач = расстояние от точки на числовой оси с
           | координатой x до ближайшей точки отрезка 0,1
    нач
    | выбор
    | | при x < 0: знач := −x
    | | при x > 1: знач := x − 1
    | | иначе знач := 0
    | все
    кон
```

ж) Нетрудно доказать, что для произвольной точки плоскости M ближайшая к ней точка единичной окружности с центром в начале координат O — это точка окружности A, лежащая на луче OM. Так как $|OM|=\sqrt{x^2+y^2}$ и $|OA|=1$, то $|AM|=|\sqrt{x^2+y^2}-1|$, где x,y — координаты точки A. Получаем алгоритм:

```
алг вещ расстояние до окружности (арг вещ x, y)
  дано |
  надо | знач = расстояние от точки (x, y) на плоскости до
       | ближайшей точки единичной окружности с центром
       | в начале координат
нач
| знач := abs (sqrt(x * x + y * y) − 1)
кон
```

з) **алг вещ** расстояние до отрезка 2 (**арг вещ** x)

```
  дано |
  надо | знач = расстояние от точки плоскости с коорди-
       | натами (x,y) до ближайшей точки отрезка 0,1
нач
| выбор
| | при 0≤x≤1 : знач := abs(y)
| | при x>1 : знач := sqrt((x − 1)**2 + **2)
| | иначе знач := sqrt(x**2 + y**2)
| все
кон
```

и) Исследуем задачу. Уравнение $(x-1)(x^2+bx+c)=0$ имеет один корень в одном из двух случаев: если уравнение $x^2+bx+c=0$ не имеет корней, т.е. $b^2-4c<0$, или если 1 является кратным корнем уравнения $x^2+bx+c=0$, т.е. если $b=-2$ и $c=1$.

Если предыдущие условия не выполняются, то уравнение имеет два различных корня в одном из двух случаев: если уравнение $x^2+bx+c=0$ имеет кратный корень (отличный от 1), т. е. $b^2-4c=0$, или если 1 является корнем этого уравнения, т. е. если $1+b+c=0$.

Во всех остальных случаях уравнение имеет три различных корня.

```
алг цел число различных корней (арг вещ b, c)
  дано |
  надо | знач = число различных корней уравнения
       | (x − 1)(x^2 + bx + c) = 0
нач
| выбор
| | при (b = −2 и c = 1) или (b*b − 4 * c < 0): знач := 1
| | при (b * b − 4 * c = 0) или (1 + b + c = 0): знач := 2
| | иначе знач:=3
| все
кон
```

Можно составить алгоритм, используя в качестве вспомогательных алгоритмы "квур" из п. 12.5 (А48) и "количество разных" из упражнения 2г):

```
алг цел число различных корней (арг вещ b, c)
  дано |
  надо | знач = число различных корней уравнения
       | (x − 1)(x^2 + bx + c) = 0
```



```
нач вещ x1,x2, цел n
| квур (1, b, c, n, x1, x2)
| если n=0
| | то знач := 1
| | иначе знач := количество разных (1, x1, x2)
| все
кон
```

к) График функции $y=x^2+x+a$ — парабола, вершина которой имеет абсциссу $-1/2$. Парабола симметрична относительно вертикальной оси, определяемой уравнением $x=-1/2$. Следовательно, справа и слева от точки $-1/2$ имеется одинаковое число целых решений неравенства $n^2+n+1<0$. Учитывая это, получаем алгоритм:

```
алг цел число целых решений квнер (арг вещ a)
  дано |
  надо | знач = число целых решений неравенства n^2+n+1 <0
нач цел n
| n :=0
| нц пока n*n+n+a<0
| | n := n + 1
| кц
| утв | n=число неотрицательных целых решений
| | неравенства n^2+n+1 < 0
| знач := 2 * n
кон
```

Здесь нужно обратить внимание на роль величины n: это одновременно счетчик найденных решений и очередное проверяемое значение.

л) Воспользуемся функцией mod(x,y) — остаток от деления x на y. Число y является делителем целого числа x тогда и только тогда, когда mod(x,y) = 0. Получаем алгоритм:

```
алг цел наименьший делитель (арг цел n)
  дано n ≥ 2
  надо | знач = наименьший отличный от единицы
       | натуральный делитель числа n
нач
| знач := 2
| нц пока mod(n, знач) ≠0
| | знач := знач + 1
| кц
кон
```

м) Последняя цифра в десятичной записи натурального числа n — это остаток от деления числа n на 10, т.е. mod(n,10). Получаем алгоритм:

```
алг цел последняя цифра (арг цел n)
  дано n ≥ 0
  надо | знач = последняя цифра числа n
нач
| знач:=mod(n, 10)
кон
```

н) Десятичная запись натурального числа n содержит k цифр, если $10^{k-1} \le n < 10^k$. Следовательно, количество цифр в десятичной записи числа n — это наименьшее неотрицательное целое число k такое, что $n<10^k$.

```
алг цел количество цифр (арг цел n)
  дано n ≥ 0
  надо | знач = количество цифр в записи числа n
нач
| знач := 0
| нц пока n ≥ 10 ** знач
| | знач := знач + 1
| кц
кон
```

Если школьники знакомы с логарифмами, задача решается проще:

```
нач
| знач := int(lg(n)) + 1
кон
```

о) Если n содержит k цифр, то первая цифра — это частное при делении с остатком числа n на число 10^k. Получаем алгоритм:

```
алг цел старшая цифра (арг цел n)
  дано n ≥ 0
  надо | знач = старшая цифра числа n
нач цел k
| k:= int(lg(n)) + 1
| знач := div(n, 10**k)
кон
```

4. Приведенный алгоритм-функция вычисляет НОД чисел m0 и n0. Опишем работу ЭВМ при вызове "НОД (14, 21)" (приведем циклическую часть алгоритма):

алг НОД

арг цел m0	цел знач	арг цел n0
	14	21
	цел m	цел n

m:=m0; n:=n0

алг НОД

цел знач	арг цел m0	арг цел n0
	14	21
	цел m	цел n
	14	21

m ≠ n ? да
m > n ? нет
n := n − m

алг НОД

цел знач	арг цел m0	арг цел n0
	14	21
	цел m	цел n
	14	7

m ≠ n ? да
m > n ? да
m := m − n

алг НОД

цел знач	арг цел m0	арг цел n0
	14	21
	цел m	цел n
	7	7

5.а)
```
алг цел НОК (арг цел m0,n0)
  дано m0>0 и n0>0
  надо | знач = наименьшее общее кратное m0 и n0
нач
| знач := m0*n0 / НОД(m0,n0)
кон
```

б)
```
алг сокращение дроби (арг цел a,b  арг рез цел (a1,b1))
  дано a>0 и b>0 | числитель и знаменатель дроби
  надо | a1 и b1 — числитель и знаменатель
       | сокращенной дроби
нач
| a1:=a/НОД(a,b)
| b1:=b/НОД(a,b)
кон
```

6.а)
```
алг цел последовательность Фибоначчи (арг цел n)
  дано n > 0
  надо | знач = n-й член последовательности Фибоначчи
нач цел k,a,b
| если n ≤ 2
| | то знач := 1
| | иначе
| |   a := 1; b := 1
| |   знач := a + b
| |   нц для k от 4 до n
| |   |   a := b
| |   |   b := знач
| |   |   знач := a + b
| |   кц
| все
кон
```

Значения величин a и b, так же как и значения результата, последовательно изменяются в команде повторения. После исполнения команд из тела цикла с параметром k значения величин a и b — это соответственно (k—2)-й и (k—1)-й члены последовательности Фибоначчи; знач — k-й член последовательности Фибоначчи.

б) Для нахождения суммы первых n членов последовательности Фибоначчи можно воспользоваться алгоритмом а). Получаем:

```
алг цел сумма Фибоначчи (арг цел n)
  дано n > 0
  надо | знач = сумма первых n членов последовательности
нач цел k
| знач :=0
| нц для k от 1 до n
| | знач := знач + последовательность Фибоначчи (k)
| кц
кон
```


Здесь мы сталкиваемся с одной типичной ситуацией. Запоминая слагаемые, можно повысить эффективность алгоритма:

```
алг цел сумма Фибоначчи (арг цел n)
  дано n > 0
  надо | знач = сумма первых n членов
       | последовательности Фибоначчи
нач цел k,a,b,c
| если n ≤ 2
| | то знач:=n
| | иначе
| | b := 1; c := 1; знач := 2
| | нц для i от 3 до n
| | | a := b; b := c; c := a + b
| | | знач := знач + c
| | кц
| все
кон
```

7.а)
```
алг вещ площадь (арг вещ a,b,c)
  дано a≥0 и b≥0 и c≥0 |длины сторон треугольника
  надо | знач = площадь треугольника
нач вещ p
| p := (a+b+c)/2
| знач := sqrt( p*(p—a)*(p—b))*(p—c)) | формула Герона
кон
```

б)
```
алг цел число закрашенных клеток вверх (арг цел n)
  дано n≥0 | выше Робота стен нет
  надо | знач = число закрашенных клеток среди n клеток
       | выше Робота, Робот в исходном положении
нач
| знач:=0
| нц n раз
| | вверх
| | если клетка закрашена то знач:=знач+1 все
| кц
| вниз на (n)
кон
```

в)
```
алг цел расстояние до стены вниз
      дано | где-то ниже Робота есть стена
      надо | положение Робота не изменилось, знач =число
           | шагов, которое надо сделать Роботу, чтобы
           | подойти к стене вплотную
           | (т.е. число клеток между Роботом и стеной)
нач
| знач:=0
| нц пока снизу свободно
| | вниз; знач:=знач+1
| кц
| вверх на (знач)
кон
```

г)
```
алг цел число выходов из клетки
      надо | положение Робота не изменилось,
           | знач=число сторон клетки, не перегороженных
           | стенами
нач
| знач:=0
| если сверху свободно то знач:=знач+1 все
| если справа свободно то знач:=знач+1 все
| если снизу свободно то знач:=знач+1 все
| если слева свободно то знач:=знач+1 все
кон
```


XI класс

§ 13. КОМАНДЫ ВВОДА/ВЫВОДА ИНФОРМАЦИИ. ЦИКЛ ДЛЯ

Как только Вы используете все возможные способы решения и не найдете подходящего, тут же найдется решение простое и очевидное для всех других людей.

третий закон Снэйфу

ОБЩИЕ УКАЗАНИЯ

Команды ввода/вывода целесообразно связать с общим понятием исполнителя. Нужно показать, что команды ввода/вывода (**ввод**, **вывод**, **нс**) — это команды соответствующих исполнителей (устройств ввода и вывода информации). Полное формальное описание этих команд, условий их выполнимости и возможных отказов можно не давать. Смысл этих команд и способы применения в достаточной мере раскрываются примерами. Команды ввода/вывода, хотя и вписываются в общую концепцию исполнителя, имеют свою специфику. Ранее (например, при изучении команд ветвления) уже встречалась ситуация, когда исполнение алгоритма зависит от состояния среды, в которой действует исполнитель. То же имеет место и для команд ввода/вывода. Специфика в том, что здесь среда исполнителя (вводимые данные) формируется человеком в процессе исполнения алгоритма и исполнение алгоритма не предопределено начальным состоянием среды.

При первом знакомстве с командой повторения **для** полезно сопоставить ее с другими командами повторения. Команды n **раз** и **для** имеют одно общее (и важное) характерное свойство — количество повторений цикла известно *в момент начала его исполнения.*

Рассмотрим главное отличие этих циклов. В цикле n **раз** при каждом повторении исполняется одна и та же серия команд,

номер повторения при исполнении цикла неизвестен. В цикле **для** появляется дополнительная величина — ***параметр цикла***. Таким образом, серия команд может зависеть от номера повторения.

Это различие является ключевым при решении вопроса о применении того или иного вида цикла. Если повторяются одинаковые действия, можно применить цикл n **раз**, если действия необходимо варьировать, нужен цикл **для**.

Для команды повторения **пока** число повторений определяется в ходе выполнения этой команды. Величины, для которых проверяется условие повторения, могут меняться более сложно, чем параметр цикла в команде **для**.

Полезно в общем виде показать, как с помощью одних команд повторения моделируются другие. К одинаковым результатам приведет исполнение следующих команд:

```
нц n раз
| тело цикла
кц
```

```
нц для i от 1 до n
| тело цикла
кц
```

```
i:=1
нц пока i≤n
| тело цикла
| i:=i+1
кц
```

А также исполнение команд:

```
нц для i от i1 до i2
| тело цикла (i)
кц
```

```
i:=i1
нц пока i≤i2
| тело цикла (i)
| i:=i+1
кц
```

```
i:=i1
нц i2—i1+1 раз
| тело цикла (i)
| i:=i+1
кц
```

Таким образом, команды повторения n **раз** и **для** взаимозаменяемы и обе могут быть заменены командой повторения **пока**. Команда повторения **пока** в общем случае не может быть заменена командами повторения других типов (например, тогда, когда число повторений цикла неизвестно к началу его исполнения).

Заметим, что во всех примерах из основного текста § 13 учебника начальное значение параметра цикла равно 1 и тело цикла состоит из вызова одной команды (две команды в алгоритме "квадраты"). Поэтому разобрать общий вид команды повторения **для** целесообразно после выполнения упражнений, когда будет накоплен достаточный запас примеров ее использования.

В учебнике наложены довольно жесткие ограничения на цикл **для**: нет шага, параметр может быть только целой величиной. Эти ограничения не носят принципиального характера, они призваны облегчить усвоение материала учениками.

УКАЗАНИЯ К УПРАЖНЕНИЯМ

1.а) — е) достаточно просты и служат для освоения команд ввода/вывода. В качестве усложнения задания можно предложить учащимся предусмотреть в алгоритмах реакцию на ввод "недопустимых" значений. Например, рассмотрим упражнение 1,в):

```
алг сокращение дроби 1
нач цел a,b,a1,b1
| вывод "Введите числитель и знаменатель дроби"
| ввод a, b
| если a>0 и b>0
| | то
| |   сокращение дроби (a, b, a1, b1)
| |   вывод a1, b1
| | иначе
| |   вывод "Недопустимые значения a или b"
| все
кон
```

Заметим, что ограничения на a и b можно было бы записать и в **дано**, не используя команду ветвления.

В упражнении 1,ж) удобно воспользоваться параметрическим заданием окружности:

$$x = a + r\cos t,$$
$$y = b + r\sin t,$$

где t меняется в промежутке от 0 до 2π. Получаем алгоритм (см. также задачу 3,в)):

```
алг окружность (арг цел n)
  дано | (a, b) — центр, r — радиус окружности
  надо | нарисована окружность (приближенно)
нач вещ a,b,r, цел k
| вывод "Введите координаты центра и радиус окружности"
| ввод a, b, r
| сместиться в точку (a+r, b)
| опустить перо
| нц для 1 от 1 до n
| | t = 2*π*k/n
| | сместиться в точку (a+r*cos(t), a+r*sin(t))
| кц
| поднять перо
кон
```

На самом деле в результате исполнения этого алгоритма Чертежник рисует правильный n-угольник, вписанный в

окружность. Таким образом, одновременно получается решение задачи 6.

3. К решению задач данного упражнения можно подойти двумя способами.

1) Достаточно непосредственно записать серию вызываемых команд Чертежника и затем определить результат их исполнения. В случае а) Чертежнику будет выдано 14 команд, в случае б) — 18 команд (число команд складывается из двух команд, предшествующих циклу, и четырех команд в теле цикла, повторенных в случае а) 3 раза и в случае б) 4 раза). В задаче а) ЭВМ выдает Чертежнику следующие команды:

сместиться в точку (0,0)
опустить перо
сместиться на вектор (1,0)
сместиться на вектор (0,−1)
сместиться на вектор (−2,0)
сместиться на вектор (0,2)
сместиться на вектор (2,0)
сместиться на вектор (0,−2)
сместиться на вектор (−3,0)
сместиться на вектор (0,3)
сместиться на вектор (3,0)
сместиться на вектор (0,−3)
сместиться на вектор (−4,0)
сместиться на вектор (0,4)

Изображение, полученное в результате исполнения этих команд, дано на рисунке 61.

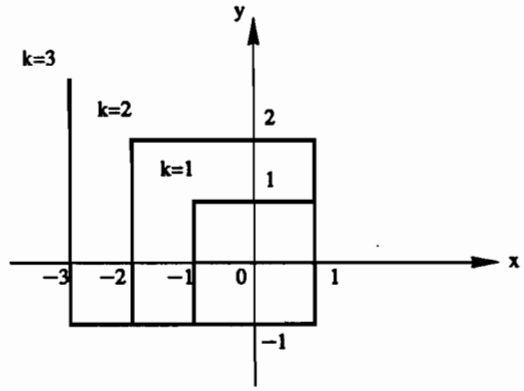


Рис. 61

Отмечены точки, в которых перо Чертежника оказывается после исполнения серии команд из тела цикла при соответствующем значении k. Аналогично получается рисунок и для задания б) (рис. 62).

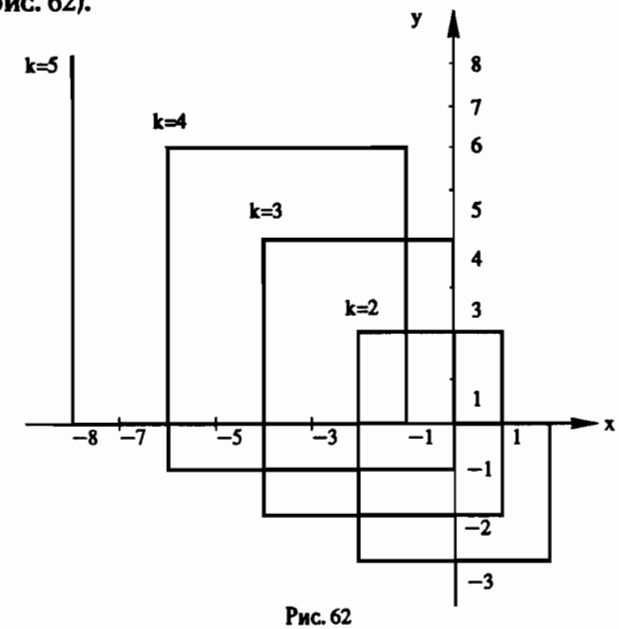


Рис. 62

2) Нужно разобрать однократное исполнение тела цикла.

а) При исполнении четырех команд тела цикла при заданном значении k будет изображен один "виток" спирали (рис. 63,а).

При этом перо Чертежника смещается относительно своего исходного положения (перед исполнением команд из тела цикла) на вектор (−1,1). Последний результат можно получить не только непосредственно, прослеживая результаты исполнения команд, но и "теоретическим" путем, сложив все векторы смещения: (k,0)+(0,−k)+(−p−1,0)+(0,k+1)=(−1,1). Это позволяет определить точку, в которой находится перо перед k-м исполнением серии команд из тела цикла. При k=1, т.е. перед первым исполнением, перо устанавливается в точку (0,0). Перед k-м исполнением при k>1 тело цикла повторяется k−1 раз, и, значит, перо k−1 раз

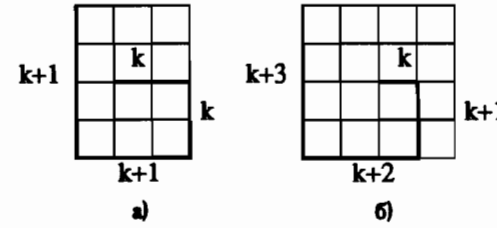


Рис. 63

смещается на вектор (−1,1), т.е. в конечном итоге на вектор (−k+1,k−1). Таким образом, известна начальная точка k-го витка −(−k+1,k−1) и четыре команды, с помощью которых он рисуется. Следовательно, можно изобразить каждый из витков, получающихся при k=1, 2, 3.

б) Так же как и в случае а), можно найти вектор, на который перо Чертежника смещается при однократном выполнении тела цикла. Здесь снова вектор смещения не зависит от номера повторения: так как (k,0)+(0,−k−1)+(−k−2,0)+(0,k+3)=(−2,2), то перо смещается на вектор (−2,2). Получающийся при этом виток изображен на рисунке 63,б. Перед исполнением тела цикла для k=2 перо устанавливается в точку (0,0), следовательно, перед исполнением тела цикла для k>2 перо k−2 раза сместится на вектор (−2,2) и будет находиться в точке (−2k+4, 2k−4). Таким образом, здесь тоже можно изобразить по отдельности все четыре витка для k=2, 3, 4, 5.

Задача 3,в) сходна с задачей построения графика функции. Для получения изображения можно составить таблицу, связывающую значения параметра цикла и координаты точки, в которую смещается перо. После этого строится ломаная линия, последовательно соединяющая найденные точки. При выполнении вычислений вручную можно ограничиться меньшим числом точек. Например, для i = −10, −5, 0, 5, 10. При этом получится приближение к нужному изображению: участки ломаной заменяются прямолинейными отрезками (рис. 64,а).

i	−10	−5	0	5	10
x	1	0.25	0	0.25	1
y	−1	−0.125	0	0.125	1

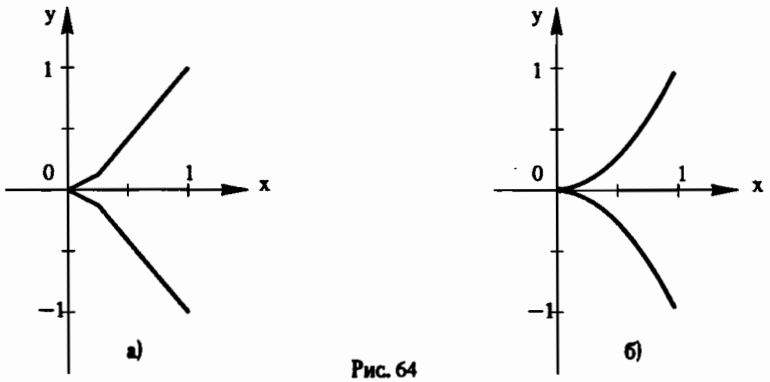


Рис. 64

Анализ полученного результата поможет сделать некоторые замечания, облегчающие построение изображения по указанной

21 точке. Изображение ломаной состоит из двух участков, симметричных относительно оси абсцисс. При этом первый участок рисуется от "конца" к точке (0,0), а второй — от точки (0,0) к "концу". При i=−10 перо Чертежника на самом деле не смещается, так как оно установлено перед исполнением цикла в точку (1,−1), а команда в теле цикла при i=−10 имеет вид "сместиться в точку(1,−1)".

Изображение, построенное по всем точкам для i от −10 до 10, изображено на рисунке 64,б.

Упражнение 3, в) имеет важное математическое содержание. Речь идет по существу о параметрическом задании кривой. Полученное изображение — это приближение с помощью ломаной линии кривой, задаваемой в параметрической форме уравнениями

$$x=t^2,\ y=t^3 \text{ при } -1 \le t \le 1.$$

В общем виде кривая задается в параметрической форме уравнениями

$$x=f(t),\ y=g(t).$$

Учитывая это, можно составить следующую схему алгоритма для рисования параметрически заданной кривой:

```
алг кривая (арг вещ a,b, арг цел n)
  дано y>0 | перо Чертежника поднято
  надо | нарисована кривая x=f(t), y=g(t) на участке
       | a ≤ t ≤ b в виде ломаной из n звеньев;
       | перо в точке (f(b), g(b)) и поднято
нач цел i
| сместиться в точку (f(a), g(a))
| опустить перо
| нц для i от 1 до n
|   сместиться в точку (f(a+i*(b−a)/n), g(a+i*(b−a)/n))
| кц
| поднять перо
кон
```

В частности, если f(t)=t, т.е. x=t, то y=g(x), и приведенная схема дает схему построения графика функции g(x).

В случае разбираемого упражнения имеем a=−1, b=1, n=20, $f(t)=t^2$, $g(t)=t^3$. Тогда

$$a+i*(b-a)/n = -1+i/10 = (i-10)/10.$$

Следовательно, соответствующая серия команд будет выглядеть так:

```
сместиться в точку (1,−1)
опустить перо
нц для i от 1 до 20
| сместиться в точку (((i−10)/10)**2, ((i−10)/10)**3)
кц
```

Обозначим теперь i−10 через k. Тогда, в то время как i пробегает значения от 1 до 20, k пробегает значения от −9 до 10. Таким образом, исполнение приводимой ниже серии команд дает тот же рисунок, что и исполнение предыдущей серии:

```
сместиться в точку (1,−1)
опустить перо
нц для k от −9 до 10
| сместиться в точку ((k/10)**2, (k/10)**3)
кц
```

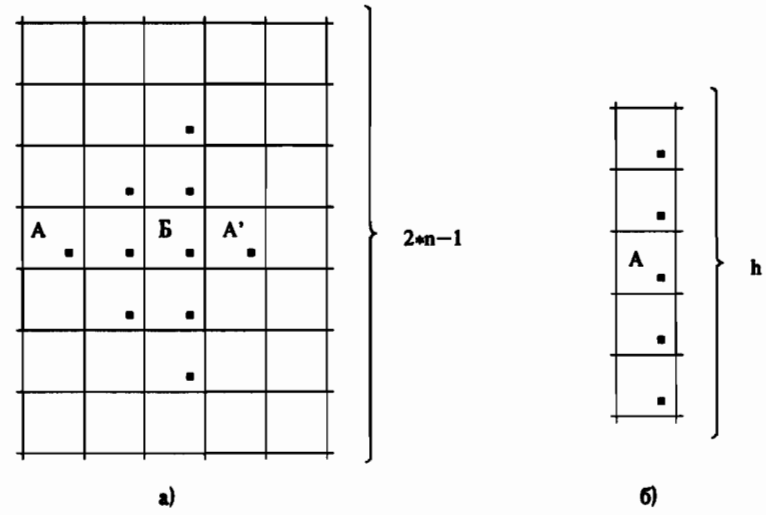
Последняя серия команд практически совпадает с приводимой в учебнике (если учесть, что при i=−10 перо остается в точке (1,−1) и исполнение команды из тела цикла при i=−10 фактически является лишним).

Используя параметрическую форму задания кривых, можно рисовать различные интересные линии. Подробнее о кривых, задаваемых в параметрической форме, можно прочитать практически в любом учебнике по математическому анализу.

4.а) В этой задаче Робот должен закрасить серию последовательно расположенных треугольников. Рассмотрим один треугольник (рис. 65,а).

При продвижении из клетки А в клетку Б высота закрашиваемых столбцов, начиная с 1, с каждым шагом увеличивается на 2. Так как n — число клеток на участке АБ, то высота h треугольника равна 1+2(n−1) = 2n−1. Таким образом, h должно быть нечетным положительным числом. Теперь несложно составить основной алгоритм:

```
алг закрасить серию треугольников (арг цел m, n)
  дано | n>0 и m>0
       | Робот в левой вершине первого
       | треугольника
  надо | закрашена серия треугольников
нач
| нц m раз
| | закрасить треугольник (n)
| кц
кон
```

а) б)

Рис. 65

В основном алгоритме использован вызов вспомогательного алгоритма "закрасить треугольник (n)". В результате исполнения этого алгоритма Робот должен закрашивать треугольник, изображенный на рисунке 65,а, и переходить в левую вершину следующего треугольника (А' на этом же рисунке). Снова применяя метод разработки алгоритма "сверху вниз", заметим, что закраска треугольника сводится к последовательной закраске столбцов. Получаем:

```
алг закрасить треугольник (арг цел n)
  дано | n>0
       | Робот в левой вершине треугольника
  надо | закрашен треугольник
нач цел k
| нц для k от 1 до n
| | закрасить столбец (2*k+1); вправо
| кц
кон
```

Отметим, что при вызове вспомогательного алгоритма "закрасить столбец (h)" Робот закрашивает столбец, изображенный на рисунке 65,б.

В исходном положении Робот находится в клетке А.

```
алг закрасить столбец (арг цел h)
  дано h>0 | h — нечетное число
       | Робот в середине столбца высоты h
  надо | столбец закрашен, Робот в исходной клетке
```

```
нач
| нц (h−1)/2 раз
| | вверх; закрасить
| кц
| нц (h−1)/2 раз
| | вниз
| кц
|
| нц (h−1)/2 раз
| | вниз; закрасить
| кц
| нц (h−1)/2 раз
| | вверх
| кц
| закрасить
кон
```

Задача 4, б) сложнее предыдущих. Во-первых, при составлении алгоритма нужно по существу использовать команду повторения для. Во-вторых, помимо однотипных повторяющихся фрагментов, изображение содержит "доперiодическую" часть — крайнюю нижнюю полосу.

При составлении алгоритма имеет смысл использовать вспомогательный алгоритм "закрасить полосу". Сначала полезно рассмотреть словесное описание алгоритма, затем можно перейти к его записи. Словесное описание выглядит примерно так: сначала закрашивается нижняя полоса из m клеток, затем закрашивается серия из n полос, содержащих по m−1 клетке, затем серия из n полос, содержащих по m−2 клетке, и т. д. Число клеток в закрашиваемых полосах с каждой новой серией уменьшается на 1, пока не получатся полосы из одной клетки. Следовательно, всего будет закрашена m−1 серия полос. Получаем следующий алгоритм:

```
алг закрасить лестницу (арг цел m, n)
  дано |
  надо |
нач цел k
| закрасить полосу (m); вверх; вправо
| нц для k от 1 до m−1
| | нц n раз
| | | закрасить полосу (m−k); вверх
| | кц
| | вправо
| кц
кон
```

На примере этого алгоритма можно пояснить учащимся конструкцию вложенного цикла.

5. а) Можно в цикле найти сумму n слагаемых:

```
алг цел сумма квадратов (арг цел n)
  дано | n≥0
  надо | знач = n² + (n+1)² +...+ (2n)²
нач цел k
| знач := 0
| нц для k от n до 2*n
| | знач := знач + k * k
| кц
кон
```

Можно воспользоваться формулой для суммы квадратов. Обозначим через S(k) сумму квадратов первых k натуральных чисел. Тогда S(k) = k(k+1)(2k+1)/6. Отсюда

$$n^2 + (n+1)^2 + \ldots + (2n)^2 = S(2n) - S(n-1) =$$
$$= (2n(2n+1)(4n+1) - (n-1)n(2n-1))/6 =$$
$$= n(n+1)(14n+1)/6.$$

Используя эту формулу, получаем алгоритм:

```
алг цел сумма квадратов (арг цел n)
  дано | n ≥ 0
  надо | знач = n² + (n+1)² +...+ (2n)²
нач
| знач := n * (n + 1) * (14 * n + 1) / 6
кон
```

На задачах б), в) можно проиллюстрировать один из распространенных приемов составления алгоритмов. Как и в других задачах, требуемая сумма вычисляется поочередным добавлением слагаемых. При этом в памяти ЭВМ хранится не только текущее значение суммы, но и значение последнего слагаемого. Это существенно упрощает вычисление следующего слагаемого и нового значения суммы.

Чтобы пояснить этот метод, начнем с более знакомой для учащихся геометрической прогрессии. Требуется найти сумму $p+pq+\ldots+pq^{n-1}$. Как известно из курса математики, $p+pq+\ldots+pq^{n-1}=p(q^n-1)/(q-1)$ (предполагается, что $q \neq 1$). С учетом этой формулы получаем простой алгоритм:

```
алг вещ сумма гп (арг вещ p, q, цел n)
  дано q ≠ 1 и n ≥ 0
  надо | знач = сумма первых n членов геометрической
       | прогрессии с первым членом p и знаменателем q
нач
| знач := p * (q ** n − 1) / (q − 1)
кон
```

Больший интерес представляет составление алгоритма с постепенным формированием суммы:

```
алг вещ сумма гп (арг вещ p, q, цел n)
  дано | n ≥ 0
  надо | знач = сумма первых n членов геометрической
       | прогрессии с первым членом p и знаменателем q
нач цел k
| знач := p
| нц для k от 1 до n−1
| | знач := знач + p * q ** k
| кц
кон
```

Рассмотрим этот алгоритм более подробно и найдем, сколько производится арифметических операций при его исполнении. Будем считать, что при возведении q в степень k происходит k − 1 умножений. Тогда всего выполняется n сложений и 1 + 2 +... ...+ n−1 = n(n−1)/2 умножений. Заметим теперь, что каждое очередное слагаемое получается из предыдущего умножением на q. Учитывая это, можно сократить число арифметических операций за счет запоминания дополнительной информации. Запишем тело алгоритма:

```
нач вещ a, цел k
| a := p
| знач := a
| нц для k от 1 до n−1
| | a := a * q
| | знач := знач + a
| кц
кон
```

Число операций сложения осталось прежним, а число операций умножения уменьшилось. В новом алгоритме выполняется n−1 умножений вместо n(n−1)/2 в старом. При больших n этот выигрыш в числе операций очень заметен.

Разобранный пример хорошо иллюстрирует, как за счет использования дополнительной информации может быть сокращено число операций. При этом, правда, увеличивается объем памяти ЭВМ, отводимой для алгоритма: требуется хранить дополнительную величину a. Это типичная ситуация — за повышение скорости решения задачи приходится платить увеличением объема памяти ЭВМ.

Теперь перейдем к задаче 5, б). Заметим, что k-е слагаемое получается из предыдущего умножением на k (k>1). Основываясь на этом, составляем алгоритм по аналогии с предыдущим:


```
алг вещ сумма факториалов (цел n)
  дано | n > 0
  надо | знач = сумма первых n факториалов
нач цел k,a
| знач := 0
| a := 1
| нц для k от 1 до n
| | a := a * k
| | знач := знач + a
| кц
кон
```

В задаче 5, в) k-е слагаемое получается из предыдущего умножением на 1/k. По аналогии с б) составляем алгоритм:

```
алг вещ сумма обратных факториалам (цел n)
  дано | n > 0
  надо | знач = сумма первых n чисел, обратных факториалам
нач вещ a, цел k
| знач := 0
| a := 1
| нц для k от 1 до n
| | a := a / k
| | знач := знач + a
| кц
кон
```

§ 14. ТАБЛИЧНЫЕ ВЕЛИЧИНЫ И РАБОТА С НИМИ

Когда исследователь перестает видеть за деревьями лес, он охотно склоняется к разрешению этой трудности путем перехода к изучению отдельных листьев.

"Ланцет", 1960 г.

ОБЩИЕ УКАЗАНИЯ

При изучении табличных величин учащиеся сталкиваются с принципиально новым типом величин. Необходимо специально обратить их внимание на это. Ранее работа велась с атомарными величинами. Табличные величины в отличие от них имеют вполне определенную внутреннюю структуру, представляют собой организованную совокупность величин (в § 14 — числовых). Такие величины, имеющие сложную внутреннюю структуру, называются в информатике структурными. По-прежнему определяющие атрибуты величины — имя, тип и значение. Рассмотрим более подробно атрибуты табличных величин.

Имя табличной величины входит в состав имен образующих ее атомарных величин. Имена этих атомарных величин имеют вид a[i], где a — имя табличной величины, а i (индекс) — натуральное число. Таким образом, если определена табличная величина с именем a, то a[1], a[2] и т.д. — это имена составляющих ее величин. Благодаря применению индексов можно работать с величинами, имена которых определяются в процессе исполнения алгоритмов. Например, в алгоритме "поиск" (п. 14.11) в таблице a ищется величина, имеющая значение x. Если такая величина содержится в таблице, то после исполнения алгоритма определено ее имя — a[i].

Роль табличных величин при составлении алгоритмов полезно проиллюстрировать анализом алгоритмов для одной и той же задачи, написанных с использованием и без использования табличных величин.

Рассмотрим алгоритм сумма1:

```
алг вещ сумма1 (арг цел n, вещ a1, a2, a3)
  дано | 0≤n≤3
  надо | знач=сумма первых n элементов из a1, a2, a3
нач
| если n=0
| | то знач:=0
| все
| если n=1
| | то знач:=a1
| все
| если n=2
| | то знач:=a1+a2
| все
| если n=3
| | то знач:=a1+a2+a3
| все
кон
```

При исполнении этого алгоритма вычисляется сумма элементов $a_1+...+a_n$, где $0 \le n \le 3$. Ясно, что можно было бы подобным образом записать алгоритм для вычисления суммы и большего числа слагаемых, если только число слагаемых не превосходит заранее фиксированного числа N (так как мы должны заранее зарезервировать имена a_1, a_2,... для всех слагаемых). При вызове алгоритма

сумма1 (n, a[1], a[2],..., a[n])

будет получаться тот же результат, что и при вызове сумма (n, a). Но только при условии, что $n \le N$. Очевидно, что применение табличных величин сокращает запись алгоритма и это сокращение становится принципиальным даже при не очень больших N.

Кроме того, расширяется область применимости алгоритма. Так, алгоритм "сумма1" применим только в тех случаях, когда число слагаемых не превосходит 3. Алгоритм "сумма" из учебника применим теоретически при любом числе слагаемых (на практике действуют ограничения, связанные с характеристиками вычислительной техники и реализацией алгоритмического языка).

Тип табличной величины однозначно определяет типы составляющих ее элементов и их взаимосвязь. При описании табличной величины указывается тип ее элементов (цел, вещ, а в дальнейшем также лог и сим) и размерность таблицы (т.е. линейная таблица или прямоугольная). Эти характеристики жестко связаны с табличной величиной и составляют описание ее типа. Размеры таблицы могут меняться при исполнении алгоритмов.

Заметим, что в алгоритмических языках вопросы о неизменности характеристик табличных величин могут решаться поразному. Например, не фиксируется жестко размерность таблицы и для таблицы a[1:m, 1:n] считается, что a[i,j] и a[(i−1)*n+j] —это имена одного и того же элемента таблицы a.

Значение табличной величины определено, когда определены значения всех составляющих ее элементов. Таким образом, значение, например, числовой табличной величины — соответствующая числовая таблица.

Иногда бывает удобной функциональная трактовка табличных величин. Линейную табличную величину можно рассматривать как функцию, область определения которой — натуральные числа 1, 2,...,n. Прямоугольная таблица может рассматриваться как функция двух переменных и т.д. Таким образом, табличная величина дает табличное задание функции. Функции в информатике задают также формулой (аналитически) или алгоритмом-функцией.

Предположим, что для табличной величины a описана зависимость значения элемента a[i] от индекса i, т. е. имеется алгоритм-функция f, при вызове которого f(i) выдает значение a[i] (например, значение a[i] есть i^2 для всех i). Ясно, что тогда в алгоритме, работающем с таблицей a, можно все обращения a[i] заменить на f(i), и результат при этом не изменится. В памяти ЭВМ вместо таблицы a будет храниться алгоритм вычисления функции f. Здесь мы снова сталкиваемся с традиционной ситуацией: за память приходится платить скоростью и наоборот. Значения a[i] выдаются быстрее, чем вычисляются значения f(i), но при этом таблица a может занимать в памяти больше места, чем алгоритм вычисления f. При разработке эффективных алгоритмов приходится отыскивать оптимальные решения подобных проблем.

Заметим, что близкое по времени изучение команды повторения для и табличных величин не случайно: команда повторения для — это идеальный (или почти идеальный) инструмент для обработки таблиц.

УКАЗАНИЯ К УПРАЖНЕНИЯМ

Упражнения **1 — 2** целесообразно проиллюстрировать рисунками типа тех, которые приведены в учебнике.

В упражнении **4** ответ (i — индекс последнего элемента, равного x) нужно разобрать на примерах. Можно предложить учащимся так изменить алгоритм "поиск", чтобы выдавался номер первого элемента, равного x. Приведем два варианта таких изменений. В первом — индекс i не меняется после того, как его значение становится отличным от нуля; во втором — просмотр таблицы ведется от конца к началу.

```
алг поиск 1 (арг цел n, вещ таб a[1:n], вещ x, рез цел i)
  надо | a[i]=x, если значение x встречается в таблице a,
       | i =0 в противном случае
нач цел j
| i:=0
| нц для j от 1 до n
| | если i=0 и a[j]=x
| | | то i:=j
| | все
| кц
кон
```

```
алгпоиск 2 (арг цел n, вещ таб a[1:n], вещ x, рез цел i)
  надо | a[i]=x, если значение x встречается в таблице a,
       | i =0 в противном случае
нач цел j
| i:=0
| нц для j от 1 до n
| | если a[n−j+1]=x
| | | то i:=j
| | все
| кц
кон
```

Алгоритмы в упражнениях **5** и **6**, а), б) получаются из соответствующих алгоритмов, приведенных в тексте параграфа, несложными модификациями.

Рассмотрим упражнения **6**, в), г). На примере этих упражнений можно проиллюстрировать связь между табличными величинами и функциями. Задачу 6, в) можно свести к 6, г), если сформировать предварительно таблицу x[1:n] такую, что

$$x[i] = a+(i-1)*((b-a)/(n-1)).$$

Полезно разобрать два решения задачи 6, в): с вычислением i-го значения аргумента на каждом шагу и с предварительным составлением таблицы.

Приведем алгоритм для задачи 6, г):

```
алг ломаная (арг цел n, вещ таб x[1:n],y[1:n])
  дано n≥2 | в таблицах x и y заданы координаты вершин
           | ломаной линии с n вершинами
  надо | Чертежник нарисовал эту ломаную линию
нач цел i
| сместиться в точку (x[1], y[1])
| опустить перо
| нц для i от 1 до n
| | сместиться в точку (x[i], y[i])
| кц
кон
```

(конечно, можно в качестве нижней границы изменения индекса указать 2).

Задача упорядочения элементов линейной таблицы (упражнение 6, е) — одна из наиболее популярных задач курсов по основам программирования. Составим не самый эффективный, но довольно простой алгоритм. Воспользуемся вспомогательными алгоритмами, для которых мы приводим только заголовки.

```
алг перестановка1 (арг цел i,j,n, арг рез вещ таба1:n )
  надо | в таблице переставлены местами
       | элементы с индексами i и j

алг цел индекс минимума1 (арг цел k,n, вещ таб a1:n)
  надо | знач = индекс минимального элемента в таблице a
       | среди элементов с индексами от k до n
```

Основной алгоритм:

```
алг упорядочение (арг цел n, арг рез цел таб a1:n)
  надо | элементы a упорядочены по неубыванию,
       | т.е. переставлены так, что a[1]≤a[2]≤...≤a[n]
нач цел k
| нц для k от 1 до n−1
| | j:=индекс минимума1 (k, n, a)
| | перестановка1 (k, j, n, a)
| кц
кон
```

Алгоритмы в упражнениях **7**, **8** по существу аналогичны алгоритмам, разобранным в основном тексте параграфа. В этих упражнениях естественным образом применяются вложенные циклы **для**. Эти упражнения можно опустить, предложив их учащимся, интересующимся программированием.

§ 15. ЛОГИЧЕСКИЕ, СИМВОЛЬНЫЕ И ЛИТЕРНЫЕ ВЕЛИЧИНЫ

Процесс составления программ для ЭВМ особенно привлекателен тем, что доставляет эстетические переживания, сходные с переживаниями, которые возникают при сочинении стихов или музыки.

Дональд Кнут

ОБЩИЕ УКАЗАНИЯ

В § 15 понятие величины наполняется новым содержанием. Во-первых, вводятся новые типы величин, во-вторых — новые операции.

Понятие логической величины довольно сложно. Некоторые логические присваивания выглядят для учащихся непривычно. Целесообразно разобрать несколько примеров типа тех, которые приведены в п. 15.2 и в упражнении **1**, например:

```
q:=0=1; q:=0=1 или 0<1
```

и т.п. Нужно обратить внимание учащихся на то, что выражение a=b в записи алгоритма не означает автоматически, что a и b равны.

Равенство в алгоритмическом языке — это операция, результатом которой является логическое значение.

Полезно сопоставить алгоритмы, записанные с использованием логических величин и без них. Например, алгоритм "перекресток" (п. 15.4) можно изменить так:

```
алг цел перекресток1
  надо | если из клетки можно уйти в любом направлении,
       | то знач = 1 , в противном случае знач = 0
нач
| если справа свободно и слева свободно и
|      сверху свободно и снизу свободно
|  то знач:=1
|  иначе знач:=0
| все
кон
```

Тогда основной алгоритм примет следующий вид:

```
алг лог вправо до перекрестка 1
  дано | Робот в горизонтальном коридоре,
       | левее перекрестка
  надо | Робот на перекрестке
```



```
нач
| нц пока перекресток1=0
|  вправо
| кц
кон
```

Сравнение приведенного алгоритма и алгоритма из учебника, конечно, в пользу последнего. Запись алгоритма с логическими величинами ближе к естественному языку и легче для понимания. Кроме того, команда **пока** работает с логическими величинами, поэтому при каждом исполнении цикла происходит вычисление логического выражения "перекресток=0" и только после этого проверка. В больших алгоритмах отмеченные выше преимущества использования логических величин играют важную роль.

В учебнике не говорится прямо о тех операциях, в которых могут участвовать логические величины. Однако из текста видно, что это логические операции **и**, **или**, **не**, которые были введены ранее. Сложнее вопрос о том, могут ли логические величины использоваться в операциях сравнения. Его рассмотрение (относящееся скорее к математической логике, чем собственно к основам информатики) лежит несколько в стороне от основной линии школьного курса. Поэтому не нужно на этом вопросе заострять внимание.

При составлении алгоритмов желательно избегать сложных конструкций, в которых применяется сравнение логических величин. Здесь могут появиться некоторые "странные" (на первый взгляд) записи типа q:=q=(q=q) и т.п. С сильными учащимися можно рассмотреть некоторые вопросы "алгебры высказываний". Например, можно предложить им проверить, что логическое выражение (p **и** q) **или** ((**не** p) **и** (**не** q)) принимает значение **да** тогда и только тогда, когда логические величины p и q имеют одинаковые значения.

Рассмотрим теперь символьные и литерные величины. Само по себе понятие символьной величины несложно. Понятие литерной величины вводится как результат естественного взаимодействия понятий символьной величины и линейной таблицы.

Нужно специально остановиться на вопросе, зачем понадобилось вводить тип величин **лит**, а не ограничиться, например, более естественным **таб сим**. Дело в том, что при обработке текстов широко применяются команды соединения, вырезки и присваивания вырезке. Эти операции характерны именно для литерных величин. Операции над литерными величинами достаточно просты и естественны. В дальнейшем (например, в § 23) будет показано, что с помощью этих операций могут составляться довольно сложные алгоритмы обработки текстов. При работе с числовыми таблицами подобные операции используются зна-

чительно реже, и поэтому в алгоритмическом языке соответствующие команды для числовых величин не предусмотрены.

Остановимся на некоторых особенностях работы с литерными величинами. *Литерные (или символьные) величины, составленные из цифр, не являются числовыми и не могут участвовать в арифметических операциях.* Например, "12"+"25" есть "1225", а не 37; запись "12"+25 вообще не имеет смысла и т.д. Нужно указать и на то, что в отличие от арифметического сложения для операции соединения не выполняется перестановочный закон: "1"+"2"="12", "2"+"1"="21".

Можно предложить учащимся решить некоторые несложные "уравнения" с литерными величинами. Приведем несколько примеров таких уравнений.

Определить, для каких символьных величин справедливо равенство x="x" (имя совпадает со значением). Здесь x— любой символ.

Пусть a — литерная величина. Найти литерную величину x, удовлетворяющую соотношению a+x=a. Здесь x — пустая строка, x="".

Описать алгоритм решения уравнения a+x=b, где a, b, x — литерные величины и a — начальная часть b.

Пусть x и y — литерные величины, причем длин (x)=1. Сформулировать условие, при котором x+y=y+x (если x="a", а y="bcd...", то, сравнивая величины "abcd..." и "bcd...a", получаем цепочку равенств a=b, b=c, ..., т.е. y=x+x+...).

УКАЗАНИЯ К УПРАЖНЕНИЯМ

3, 4. При попытке выполнить алгоритм А72 в обстановке, когда от коридора вверх отходит другой коридор, не завершится исполнение алгоритма А73 (условие повторения будет все время соблюдаться, пока Робот находится в вертикальном коридоре). Изменим алгоритм А73: теперь Робот необязательно проходит до конца тупика; убедившись, что вверх можно сделать три шага, он сообщает о наличии тупика (Робот "подхалтуривает").

```
алг лог сверху длинный тупик
  надо | знач = да , если вверх отходит ветка длины больше 3,
       | Робот в исходном положении
нач цел n
| n:=0
| нц пока сверху свободно и n≤3
| | вверх; n:=n+1
| кц
| знач := ( n=3 ) и сверху свободно
| вниз на (n)
кон
```


5. Приведем одно из возможных решений:

```
алг число проходов (рез цел n)
  дано | Робот в левом нижнем углу прямоугольника,
       | огороженного стенами. В нижней стене прямоугольника есть несколько проходов разной ширины
  надо | Робот в правом нижнем углу прямоугольника,
       | n — число проходов в нижней стене
нач
| n:=0
| если снизу свободно то вдоль прохода; n:=n+1 все
| нц пока справа свободно
| | если снизу свободно
| | | то вдоль прохода; n:=n+1
| | | иначе вдоль стены
| | все
| кц
кон

алг вдоль стены
  дано | Робот сверху от стены
  надо | Робот в клетке правее стены
нач
| нц пока снизу не свободно и справа свободно
| | вправо
| кц
кон

алг вдоль прохода
  дано | Робот сверху от прохода
  надо | Робот в клетке правее прохода
нач
| нц пока снизу свободно и справа свободно
| | вправо
| кц
кон
```

8. В задании а) достаточно перечислить точки, восклицательные и вопросительные знаки в строке x (предположение: эти знаки не встречаются внутри предложений). Задание б) — частный случай г). Рассмотрим возможное решение задачи в):

```
алг лит перевертыш ( арг лит a )
  надо | знач = "перевертыш", если строка a совпадает с
       | собой после переворачивания, и знач ="не перевертыш" в противном случае
```

```
нач цел n,i, лог q
| n:=длин (a)
| i:=1
| q:=да
| нц пока i<n и q
| | q:=a(i)=a(n−i+1)
| | i:=i+1
| кц
| если q
| | то знач:="перевертыш"
| | иначе знач:="не перевертыш"
| все
кон
```

В этом алгоритме показана возможность применения логической величины в алгоритме проверки выполнения некоторого условия. По сути достаточно сформулировать условие проверки. Остальная часть алгоритма стандартна. В данном случае проверка продолжается до первого несовпадения.

Заметим, что можно получить совсем короткое решение задачи 8, в), воспользовавшись алгоритмом из задачи 8, д) как вспомогательным. При составлении этого алгоритма может быть использован употребительный прием формирования строк:

```
алг переворачивание (арг лит a, рез лит b)
  надо | строку a записать в обратном порядке в строку b
нач цел i, лит b
| n:=длин (a)
| b:=""
| нц для i от 1 до n
| | b:=b+a(n−i+1)
| кц
кон
```

Задача из упражнения 8, г) содержит "подводные камни". Во-первых, нуждается в уточнении постановка задачи. Так, если a="nonon", x="non" (нет), y="oui" (да), то под результатом можно понимать одну из строк "ouion", "nooui", "ououi", "ouioui" в зависимости от принятых соглашений о замене. Далее, пусть a="cdcd", x="cd", y="dc". Снова нужно уточнить, что считается результатом: "dcdc" или (после повторной замены) "ddcc". Если a="cdd", то результатом замены можно считать "dcd", а можно и "ddc".

Рассмотрим несколько алгоритмов:

```
алг замена 1 (арг лит x,y, арг рез лит a)
  дано | длин (x)= длин (y)
  надо | всюду в строке a x заменено на y
```



```
нач цел m,n,i
| m:=длин (x); n:=длин (a); i:=1
| нц пока i ≤ n−m+1
| | если a[i:i+m−1]=x
| | | то
| | | | a[i:i+m−1] = y; i:=i+m
| | | иначе i:=i+1
| | все
| кц
кон
```

Если вхождения x не "перекрываются", то после исполнения этого алгоритма все вхождения x будут заменены на y:

```
алг замена 2 (арг лит x,y, арг рез лит a)
  дано | длин (x)= длин (y)
  надо | всюду в строке a x заменено на y
нач цел m,n,i
| m:=длин(x); n:=длин (a)
| нц для i от 1 до n−m+1
| | если a[i:i+m−1]=x
| | | то a[i:i+m−1] = y
| | все
| кц
кон
```

Этот алгоритм при x="cd", y="dc", a="cdd" дает результат "ddc"; при a="cdcd" — результат "dcdc".

```
алг замена 3 (арг лит x,y, арг рез лит a)
  дано | длин (x)= длин (y)
  надо | всюду в строке a x заменено на y
нач цел m,n,i, лог q
| q:=да
| нц пока q
| | q:=нет
| | m:=длин(x)
| | n:=длин(a)
| | нц для i от 1 до n−m+1
| | | если a[i:i+m−1]=x
| | | | то
| | | | | a[i:i+m−1] = y
| | | | | q:=да
| | | все
| | кц
| кц
кон
```

При исполнении этого алгоритма замены производятся до тех пор, пока в строке a хотя бы один раз встречается x. В некоторых ситуациях этот алгоритм может зациклиться (например, если x=y).

Можно составить и другие алгоритмы. По существу составление алгоритма можно считать уточнением задачи.Чтобы избежать отмеченных выше сложностей (их полезно разобрать с сильными учениками), можно сузить область применимости алгоритма. Тогда эти сложности снимаются.

В задачах **9**, а),б) удобно воспользоваться литерной переменной, содержащей циклическое продолжение алфавита. Например, в задаче 9, а) — это строка

w="абвгдежзийклмнопрстуфхцчшщъыьэюяа";

в задаче 9, б) — строка

w="абвгдежзийклмнопрстуфхцчшщъыьэюяабвгдежзий"

(впрочем, и в первой задаче можно использовать эту длинную строку).

Получаем алгоритм:

```
алг шифр (арг лит a, рез лит b)
  надо | b — зашифрованная строка a
нач лит w, цел i
| w="абвгдежзийклмнопрстуфхцчшщъыьэюяабвгдежзий"
| b:=""
| нц для i от 1 до длин (a)
| | b:=b + w (номер (a[i]) + 1)
| кц
кон
```

Здесь "номер" — вспомогательный алгоритм вычисления алфавитного номера буквы:

```
алг цел номер (арг сим s)
  дано | s — буква русского алфавита
  надо | знач:=алфавитный номер буквы s
нач лит w, цел i,n
| w="абвгдежзийклмнопрстуфхцчшщъыьэюя"
| знач:=0
| нц для i от 1 до длин (w)
| | если w[i]=s
| | | то знач:=i
| | все
| кц
кон
```


При составлении алгоритмов расшифровки алфавит удобно продолжить "в начало":

w="цчшщъыьэюяабвгдежзийклмнопрстуфхцчшщъыьэюя".

Алгоритмы шифровки в 9, в),г) составить несложно. Сложнее алгоритмы расшифровки. Их имеет смысл разобрать с сильными учениками.

§ 16. СОСТАВЛЕНИЕ ЦИКЛИЧЕСКИХ АЛГОРИТМОВ

ОБЩИЕ УКАЗАНИЯ

Этот раздел — один из самых трудных в учебнике. Он посвящен изложению методов алгоритмизации именно как методов. В ослабленном курсе (например, при преподавании в VII—VIII классах) этот материал можно опустить.

В п.16.1 авторы пытаются объяснить, что такое рекуррентная последовательность вообще. Для дальнейшего это особенной роли не играет и может быть пропущено, если вызывает трудности.

Изучение п. 16.2 разумно, например, начать с постановки вопроса: дана последовательность

2, 3, 5, 9, 17,...,

угадайте несколько следующих членов. Возможно, школьники заметят, что каждый следующий член получается из предыдущего умножением на 2 и вычитанием 1 — это хорошо. (Они могут также заметить, что члены последовательности — степени двойки, к которым прибавлена 1. Это хуже, поскольку в этом случае рекуррентное соотношение не нужно.)

Для объяснения построения алгоритма вычисления n-го члена этой последовательности авторы придумали такой методический ход. Сначала, в п. 16.2, они выписывают алгоритм, который заполняет таблицу, содержащую члены последовательности. Вроде бы этот алгоритм не должен вызвать трудностей в понимании у школьников (если они вообще понимают, что означает цикл **для** и что такое табличные величины). Затем, в п. 16.3, из этого алгоритма строчка за строчкой устраняются табличные величины, и получается эквивалентный алгоритм без табличных величин.

Преимущество такого способа объяснения состоит в том, что иначе легко запутаться и, например, написать цикл не от 2 до n, а от 1 до n. (Кстати, что в этом случае будет вычислять алгоритм ? Ответ: n+1-й член последовательности.)

В п. 16.4 тот же метод проиллюстрирован на примере задачи о вычислении сопротивлений. Тут главная трудность, вероятно,

состоит в том, что школьники давно уже забыли этот раздел физики, если когда-то и знали. С точки зрения программирования здесь показывается применение метода рекуррентных соотношений к задачам, формулировка которых никаких соотношений в явном виде не содержит.

В п. 16.5 строится программа для вычисления последовательности Фибоначчи. Здесь, как и раньше, легко составить алгоритм с использованием таблицы:

```
a[1] := 1;  a [2] := 1
нц для i от 3 до n
| a[i] := a[i—1] + a[i—2]
кц
```

а вся трудность связана с тем, чтобы не употреблять таблиц в этом алгоритме.

Если раньше вместо хранения таблицы достаточно было хранить один-единственный очередной элемент последовательности, то для последовательности Фибоначчи это не пройдет: нужно хранить два элемента: a[i—2] и a[i—1], которые оба нужны при вычислении a[i]. Они называются в книге "старое a" и "новое a".

Пункт 16.6 показывает, что алгоритм вычисления последовательности Фибоначчи может быть упрощен — можно не рассматривать отдельно первые два члена. Для этого, правда, нужно добавить к последовательности фиктивные "нулевой" и "минус первый" члены. Это — стандартный прием в программировании, обычно существенно упрощающий алгоритм.

Пункт 16.7 начинается с обсуждения рыбной ловли. Более строго и без использования аналогий алгоритм A83 можно объяснить следующим образом.

В ходе выполнения цикла для i от 2 до n в переменной v хранится максимум нескольких первых элементов таблицы. На каждом шаге мы сравниваем этот максимум со следующим элементом таблицы. Если этот новый элемент окажется меньше максимума уже просмотренных, то на максимум он не влияет. Если же он окажется больше, то он и будет максимумом на следующем шаге. Это можно было бы записать в виде команды выбора

```
если v > a[i]
| то v:= a[i]
все
```

Далее приводится аналогичный алгоритм определения максимального уровня радиации, в котором данные берутся не из

массива, а, так сказать, поступают "по ходу дела". В конце пункта дается определение однопроходного алгоритма.

В п. 16.8 дается алгоритм подсчета числа максимумов. Тут надо прежде всего понять, что значат слова "число максимумов". Если в таблице все элементы различны, то максимальный элемент у нее один. Однако может так случиться, что некоторые элементы одинаковы: например, среди чисел 1,5,1,5,5,2 три максимальных (равных 5). Наибольшее число максимальных элементов будет в случае, когда все числа равны — тогда все элементы максимальны.

Идею приводимого алгоритма можно объяснить так. Пусть мы имеем таблицу из 7 элементов, в ней 3 максимальных элемента, равных 17. Что будет, если добавить в нее еще один элемент x? Тут есть три возможности:

1) если этот новый элемент меньше 17, то он не будет максимальным и по-прежнему останутся три максимальных элемента, равных 17;

2) если новый элемент равен 17, то получатся 4 максимальных элемента, равных 17;

3) наконец, если x больше 17, то он будет единственным максимальным элементом.

Эти три варианта и соответствуют трем вариантам в команде выбора в алгоритме A85.

Приступая к изучению алгоритма в п. 16.9, надо иметь в виду:

а) слово — это необязательно осмысленное слово: любая последовательность идущих подряд букв называется словом;

б) строка символов — это таблица, в которой каждый элемент является либо буквой, либо пробелом.

Попробуйте, зная это, разобраться в алгоритме A86. Может быть, вам поможет такое замечание: q = да, если просмотренная на предыдущем шаге часть слова кончается пробелом (или пуста).

Пункт 16.10 посвящен наиболее важной и интересной теме — понятию инварианта цикла. Хороший программист отличается от плохого, в частности, тем, что он понимает, что такое инвариант цикла, и может применять это понятие на практике. Более подробно вопрос об инварианте цикла обсуждается в книгах Э.Дейкстры "Дисциплина программирования", Д.Гриса "Наука программирования", Н.Вирта "Систематическое программирование" и в учебнике "Программирование для математиков" А.Г.Кушниренко и Г.В.Лебедева (тех же самых, которые являются авторами школьного учебника).

Далее следует п. 16.11, который в результате опечатки назван 16.1 со звездочкой. Он посвящен рекурсивным алгоритмам. Это —отдельная большая тема, и приведенных двух примеров недостаточно, чтобы с ней познакомиться.

Приведенные примеры иллюстрируют лишь два аспекта рекурсии. Первый пример показывает, что, несмотря на простоту составления алгоритма, рекурсией надо пользоваться с осторожностью и отдавая себе ясный отчет, как именно будет выполняться алгоритм.

Второй пример (закраска области) показывает, что есть задачи, которые рекурсивно решить крайне сложно, но рекурсивное решение которых записывается просто и коротко.

Учителю следует иметь в виду, что в силу ряда особенностей человеческого мышления этот простой по сути материал тем не менее является достаточно сложным для восприятия. На преодоление этой сложности нацелены упражнения 24 и 25 на с. 141 учебника. В то же время, в отличие, например, от рекуррентных соотношений, используемых в § 26, рекурсия нигде далее в учебнике не используется и не применяется. Поэтому материал о рекурсии помечен звездочкой и по желанию учителя может быть опущен.

УКАЗАНИЯ К УПРАЖНЕНИЯМ

1. Изюминка предлагаемой задачи в том, что не нужно каждый раз заново вычислять величину x**k/k! для каждого следующего k: ее можно получить из предыдущей умножением на x и делением на k.

2. Аналогично 1, только надо умножать на x**2 и делить на (k−2)*(k−1).

3. Задача а) проверяет, понят ли аналогичный алгоритм в тексте. В задачах б) и в) не забудьте прочесть указание!

4. Здесь физические трудности возрастают еще более: надо помнить про эдс и внутреннее сопротивление. Напоминание: цепь с эдс E и внутренним сопротивлением R эквивалентна идеальной батарейке с эдс E без внутреннего сопротивления, включенной последовательно с сопротивлением R.

5. Все три случая этой задачи совершенно аналогичны, и смысл ее в том, что не нужно хранить элементы последовательности и вычислять сумму каждый раз заново. Вместо этого нужно хранить сумму обратных величин (сумму квадратов в пункте б)) всех уже найденных элементов и корректировать ее.

6. Описанный способ быстрого вычисления квадратного корня интересен с математической точки зрения; его объяснение можно найти, например, в (к сожалению, давно не переиздававшихся) популярных книгах Я.И.Перельмана.

7. С точки зрения построения алгоритма эта задача совершенно аналогична предыдущей.

8. В случае а) надо помнить три последних члена последовательности, в б) — два, в в) — k последних членов.

9. Упражнение решается по образцу п. 16.6 (с. 129 учебника).

10. Прежде всего продолжите последовательности вправо, найдя закон их построения (в первой посмотрите на отношение соседних членов, во второй — на разницу между каждым членом и удвоенным предыдущим, в третьей закон аналогичен закону последовательности Фибоначчи, только вместо сложения используется вычитание).

11. Эта задача совершенно бессмысленна, если не выполнять ее с помощью компьютера. Если же выполнить ее, то компьютер нарисует дугу окружности или что-то очень близкое. Пожалуй, самое простое объяснение этого факта состоит в использовании комплексных чисел: каждая следующая пара чисел, рассматриваемая как комплексное число, получается из предыдущей умножением на число 1 − 0.1i, модуль которого близок к 1. Впрочем, это объяснение вам, вероятно, ни к чему, так как школьники все равно не проходят комплексных чисел.

12. Это — замечательная трудная задача из книги Дейкстры "Дисциплина программирования", только надо сказать, что алгоритм не может использовать табличных величин и время его работы должно не превосходить константы, умноженной на двоичный логарифм числа y. (Решение смотри в этой же книге.) Если же таких ограничений не накладывать, то смысл задачи теряется.

13. а) Нельзя, это зависит от того, был ли максимальный элемент больше, меньше или равен 5. б) Один. Шесть. Пять. в) Этот алгоритм получается из приведенного в п. 16.8 заменой таблицы на коридор, подобной проделанной в п. 16.7.

14. На пути в тупик нужно вычислить максимальный уровень радиации (или среднюю температуру, вычислив сумму и разделив на подсчитанное число клеток), а на обратном пути — закрашивать.

15. а) Надо помнить число перемен знака и знак последней клетки. б), в) Аналогично.

16. Надо помнить число локальных максимумов, величину последнего просмотренного элемента и то, является ли он локальным максимумом в просмотренной части таблицы.

17. Решение можно прочесть в упомянутых книгах Дейкстры и Кушниренко-Лебедева.

18. Инвариант: n равно квадрату i. В конце i=1000, n=1000000.

19. Инвариант: d неотрицательно, a=c*d+b. После окончания работы d<b, поэтому c — частное, а d — остаток при делении a на b.

20. Это — алгоритм нахождения числа общих элементов в двух возрастающих таблицах вместе с доказательством его правильности.

21. Инвариант цикла: a*p+b*q = 2*a*b. Кроме того, НОД(a,b)*НОК(a,b)=a*b для любых целых a и b.

22. НОД(a,b)*m не меняется в ходе исполнения алгоритма.

23. Инвариант: таблица получена из исходной с помощью перестановки, все числа с номерами меньше i неотрицательны, все числа с номерами больше j отрицательны. В конце i не меньше j, и это значит, что все неотрицательные элементы стоят левее всех отрицательных.

23. (Это — вторая задача с номером **23**.) Вычисляется десятичное и двоичное значение строки цифр a.

24. При ручном выполнении алгоритма A89 рекомендуется рисовать состояние памяти ЭВМ (лучше на доске или карандашом, чтобы можно было стирать), как описано в п. 12.4 (с. 95) и п. 16.11 (с. 135) учебника.

25. Робот уходит на бесконечность вверх, закрашивая клетки по пути.

26. В формулировке учебника — это очень простая, обычная задача. Она становится сложной и интересной, если на поле Робота есть стены. В этом случае надо, начиная с k=1 и последовательно увеличивая его, вызывать рекурсивный алгоритм, выясняющий, можно ли попасть из данной клетки в закрашенную не более чем за k шагов, и в котором этот же алгоритм вызывается из доступных (не отгороженных стеной) соседних клеток с уменьшенным на 1 значением k.

УКАЗАНИЯ К УПРАЖНЕНИЯМ НА ПОВТОРЕНИЕ

В сущности, это не упражнения на повторение, а новые и довольно трудные задачи. История их появления такова: когда по просьбе А.П.Ершова срочно переписывался учебник информатики (первый, в двух книгах), надо было чем-то заполнить оставшееся место и было решено включить туда трудные задачи, чтобы школьники не думали, что они уже одолели все премудрости программирования. Эти трудные задачи были разбавлены простыми, и все вместе было названо упражнениями на повторение. Эта традиция продолжается и в настоящем учебнике.

Ограничиваемся краткими комментариями.

1. Задача а) интересна, если дополнительно потребовать, чтобы алгоритм не использовал переменные и рекурсию. Вот ее решение:

```
алг поиск прохода
  надо | Робот на той же горизонтали у прохода вниз,
       | если он есть
нач
| нц пока слева свободно
| | влево
| кц
| нц пока справа свободно и не снизу свободно
| | вправо
| кц
кон

алг к нижней стене
  надо | Робот у нижней стены
нач
| поиск прохода
| нц пока снизу свободно
| | вниз
| | поиск прохода
| кц
кон
```


После этого остается лишь пройти по нижней стене в левый угол.

Задача б) интересна, если можно действительно запустить программу на компьютере: забавно наблюдать, как Чертежник рисует копию лабиринта Робота.

2. Схема алгоритма (n — данное число):

```
k :=2
нц пока mod (n,k) ≠ 0
| k :=k+1
кц
```

3. Число простое, если его наименьший делитель, больший 1, равен самому числу.

4. Если у числа n есть делители, то хотя бы один из них не превосходит квадратного корня из n (делители группируются в пары, произведение элементов каждой пары равно n и один из элементов не превосходит квадратного корня из n).

5. Схема возможного алгоритма:

```
нц пока n ≠ 1
| k:= наименьший делитель (n)
| вывод k
| n := n/k
кц
```

Здесь исрользуется как вспомогательный алгоритм упражнение 2.

6. Это — алгоритм Евклида (смотрите об этом, например, в книге "Простое и сложное в программировании").

7. Печатать, начиная с младших разрядов, можно так:

```
нц пока n ≠ 0
| вывод mod (n,10)
| n := div (n,10)
кц
```

Печатать, начиная со старших разрядов, можно так: начав с младших разрядов, записывать цифры в массив, а потом напечатать его в обратном порядке.

8. Период появляется тогда, когда при делении столбиком остатки (а не цифры в частном) начинают повторяться. Полезно иметь в виду также, что длина предпериода не превосходит n.

10. Задачу можно рассматривать как задачу о слиянии двух упорядоченных таблиц, содержащих кратные чисел 13 и 17. (Сами таблицы, разумеется, хранить не надо.)

11. В однопроходном алгоритме достаточно помнить число различных элементов просмотренной части и последний элемент.

12. Квадратичный алгоритм (алгоритм, в котором число действий пропорционально квадрату длины таблицы) не интересен: каждый элемент надо сравнивать со всеми предыдущими. Существует алгоритм, в котором число действий пропорционально n, умноженному на log n: следует воспользоваться логарифмической сортировкой (смотрите, например, "Простое и сложное в программировании") и свести задачу к предыдущей.

13. Задача о слиянии двух упорядоченных таблиц (смотрите раздел о сортировке слиянием в "Простом и сложном...").

14. Задача из книги Д.Гриса "Наука программирования", названная там "Жулик на пособии". Идея решения: для каждой таблицы заводится целочисленная переменная, отмечающая границу между просмотренной и непросмотренной частью; инвариант: если общий элемент есть, то он есть в непросмотренных частях. Шаг цикла: если все три первых непросмотренных элемента в трех таблицах равны, то нужный элемент найден, если нет, то меньший из них можно перевести в просмотренные без нарушения инварианта.

15. Задача из той же книги Гриса, что и предыдущая. Идея решения: сравним x с элементом таблицы, стоящим в верхнем правом углу. Если он больше, то первую строку можно отбросить; если меньше, можно отбросить последний столбец.

16. Для каждого k от 1 до m можно индуктивно вычислять длину максимального начала последовательности b, которое может быть получено из a[1]...a[p] вычеркиванием.

17. Обозначив через f[k,l] максимальную длину общей подпоследовательности в a[1]...a[k] и b[1]...b[l], можно постепенно заполнять таблицу f[k,l].

18. Смотрите книгу Гриса "Наука программирования".

19. Смотрите книгу Дейкстры "Дисциплина программирования".

20. Там же.

21. Смотрите книгу "Простое и сложное...".

22. Пусть, например, мы знаем числа s[1]...s[k], где s[i] — это число способов уплатить i рублей купюрами в 1, 3 с 5 рублей. Тогда число t[i] способов уплатить i рублей купюрами в 1, 3, 5, 10 рублей может быть найдено по формуле

$$t[i] = s[i] + s[i-10] + s[i-20] + \ldots$$

(сумма продолжается, пока не появятся отрицательные числа). Применяя такие соображения, можно постепенно ввести в рассмотрение все купюры, заполняя соответствующие таблицы.

23. Смотрите книгу Вирта "Систематическое программирование". (Число, о котором идет речь в задаче, равно 1729.)

24. Смотрите книгу Ахо, Хопкрофта и Ульмана "Построение и анализ вычислительных алгоритмов" (алгоритм Дейкстры нахождения кратчайшего пути в графе).

25. Для каждой буквы слова b надо проверить, что она встречается в b не большее число раз, чем в a.

26. Надо вычислить a*d, b*c и (a+b)*(c+d), затем a*c+b*d получится как разность.

27. Каждый раз надо смещаться в ту сторону, куда это можно, но не назад.

28. Задачу можно упрстить задав размеры прямоугольника.

29. Видимо, в цикле нужно вычислять координаты и скорость шара сразу после очередного отражения.

30. Надо обходить забор, "держась за него правой рукой" и считая "число поворотов вокруг собственной оси" — оно равно +1 или — 1 в зависимости от того, внутри Робот или снаружи. Дополнительные трудности при обнаружении возврата в исходную точку связаны с тем, что два параллельных куска забора могут отстоять на одну клетку.

31. По формуле Грина площадь, ограниченная кривой, может быть вычислена как контурный интеграл дифференциальной формы x*dy (или y*dx).

32 — 34. Если у вас есть доступ к программе КуМир, то имеет смысл сравнить решения учеников с соответствующими исполнителями этой системы (фото 19—22 вклейки учебника).

ГЛАВА 2. УСТРОЙСТВО ЭВМ

1. ПРЕДИСЛОВИЕ

В отличие от 1-й главы, где делается попытка формирования структурного алгоритмического мышления довольно нетрадиционным путем, 2-я глава содержит более или менее знакомый материал об устройстве ЭВМ: физических и логических основах функционирования, машинных командах, техническом устройстве периферийного оборудования, стандартном программном обеспечении и т.д.

Вследствие этого по 2-й главе можно найти сравнительно много дополнительной литературы для школьников (см., например, рекомендуемую в учебнике на с. 224 книгу, названную в русском переводе "Знакомьтесь: компьютер" (М.: Мир, 1989).

Вторая глава имеет четкую логику построения. Преподаватель вместе с учениками как бы собирает ЭВМ из простейших элементов: рассматривается работа МОП-транзистора, затем даются схемы вентилей, из них строится процессор и анализируются основные принципы его работы, после этого изучается периферийное оборудование и программное обеспечение.

Авторы учебника, следуя указанной логике, удачно выбрали наиболее важные фрагменты для изучения. "Если нет возможности охватить весь лес целиком, то займемся изучением отдельных листьев" (см. эпиграф к § 14 пособия). Но если при этом не делается хотя бы краткого обобщения, то неизбежно теряется общая картина.

Здесь мы постараемся восполнить этот пробел. Кроме того, дадим более подробный разбор тех понятий, которые, по нашему мнению, способствуют дальнейшему развитию логического и алгоритмического мышления учеников.

2. ФИАСКО КОММУТАТИВНОГО ЗАКОНА

Приступая к преподаванию 2-й главы, многие учителя, особенно бывшие инженеры и настоящие физики, испытывают облегчение, как заблудившийся путник, выйдя на знакомую тропинку. Возникает иллюзия, будто методика преподавания учебных тем 2-й главы давно известна и проверена многолетней практикой.

Попробуем показать, что это впечатление обманчиво.

Методика обучения взрослых программистов в свое время была механически перенесена в ПТУ, техникумы, школы. Поэтому, во-первых, материал излагался с опорой на хорошее знание основ физики, математики, на элементарную (хотя бы) техническую грамотность. Во-вторых, предполагалась заинтересованность обучаемого в новых знаниях. Любой учитель знает: у современного школьника обычно нет ни того, что "во-первых", ни того, что "во-вторых".

Кроме того, время, необходимое для изучения устройства ЭВМ по старой методике, значительно превышает возможности курса в общеобразовательной школе.

Но главное, пожалуй, не это. Главное то, что в педагогике от перемены мест слагаемых сумма меняется. Когда-то изучение устройства ЭВМ стояло в самом начале курса информатики. В этом была своя логика: "Вот вам алгебра Буля и двоичная арифметика. А вот электромагнитное реле (триод, транзистор, кристалл кремния), с помощью которого теоретические построения обретают физический смысл. Теперь давайте соберем вентиль, из него — ячейку, затем блок, наконец, модель вычислительной машины. А теперь попробуем понять, зачем эта машина нам нужна". После этого излагалось программирование от ассемблера к языку высокого уровня.

В некоторых современных учебниках информатики глава об устройстве ЭВМ переместилась в середину (иногда даже в конец) курса. Однако сама глава при этом практически не претерпела изменений. Логика нарушалась: "Вот вам ЭВМ. Она понимает такие-то команды на языке высокого уровня. Видите, она вас слушается! А теперь давайте из вентилей соберем ЭВМ". Зачем ее собирать, если она уже есть?

Непросто убедить школьника в необходимости изучения устройства ЭВМ, если он уже почувствовал, что и без этих знаний можно вполне успешно ею пользоваться. А если и сам учитель считает эти знания лишними, то ситуация патовая.

Таким образом, при отсутствии мотивации глава об устройстве ЭВМ кажется многим ученикам второстепенной и скучной, а традиционная методика преподавания не срабатывает.

Поэтому важнейшая задача учителя — убедить ученика в необходимости изучения этого, на первый взгляд ненужного материала. Можно, например, опереться на тот незаслуженный пиетет, который имеет у нас всякий человек, понимающий машинный язык и знающий устройство этой загадочной ЭВМ. Парадоксально, но "нянька", обучающая ЭВМ выполнять простейшие действия и понимать несколько слов на человеческом языке, окружена ореолом кудесника и чудотворца.

Другой путь: сделать переход от алгоритмического языка ко 2-й главе максимально плавным и незаметным. Попробуем пойти по второму пути. Задача еще более усложняется из-за того, что в учебнике этот переход настолько незаметен, что приходится признать, что его нет вовсе.

3. АЛГЕБРА ЛОГИКИ, ДВОИЧНОЕ КОДИРОВАНИЕ, ВЫКЛЮЧАТЕЛИ И КИТАЙЦЫ

Начнем с отдельных параграфов 1-й главы, вернемся к логическим операциям, выражениям (с. 79) и величинам (с. 116). Затем напомним запись значений логической величины с помощью нулей и единиц, а в связи с этим повторим двоичное кодирование (с. 6), которое, кстати, должно оставаться на ознакомительном уровне, на манер китайской грамоты: известно, что существует китайский (читай: машинный) язык, что с помощью него можно закодировать любую информацию, что имеются автоматические переводчики с китайского и обратно (трансляторы, преобразователи), что в природе даже есть живые китайцы (системные программисты), владеющие китайским свободно. Но не следует требовать от учащихся зубрить иероглифы, т.е. тренироваться записывать информацию на машинном языке, переводить числа из одной системы счисления в другую.

Итак, двоичная система оказалась удобной в качестве языка логики, так как логические величины могут принимать всего два значения: **да** ("истина") или **нет** ("ложь"). Это поняли спустя почти сто лет после того, как английский математик-самоучка Джордж Буль сформулировал основные положения своей алгебры в работах "Математический анализ логики" (1847), "Исследование законов мышления" (1854) и др.

Булева алгебра была развита американским ученым Чарлзом Сендерсом Пирсом. Хотя Пирс и понимал, что логические выражения можно наполнять физическим смыслом с помощью, например, электрических схем, но развитие практического приложения алгебры логики связано с именем другого американца, основоположника ***теории информации*** Клода Шеннона, одного из разработчиков теоретических основ вычислительной техники. В своей докторской диссертации Шеннон, сочетавший в себе качества хорошего математика и отличного специалиста по электротехнике, раскрыл связи между двоичным способом кодирования информации, алгеброй логики и электрическими (в те времена — релейными) схемами. Затем на смену реле пришли электронные лампы, далее транзисторы, интегральные схемы и т.д.

Но независимо от элементной базы, важнейшие принципы работы ЭВМ одни и те же. В роли "элементарной частицы" всегда выступает разновидность выключателя. И если правильно соединить очень много простых бытовых выключателей и нанять очень много людей, которые будут ими в нужный момент щелкать, то получится вычислительная машина.

Покажем это на простейших схемах из кнопок и лампочек. Нажатие на кнопку означает, что некоторая логическая величина принимает значение **да**. Если логическое выражение тоже равно **да**, то лампочка загорается (рис. 66):

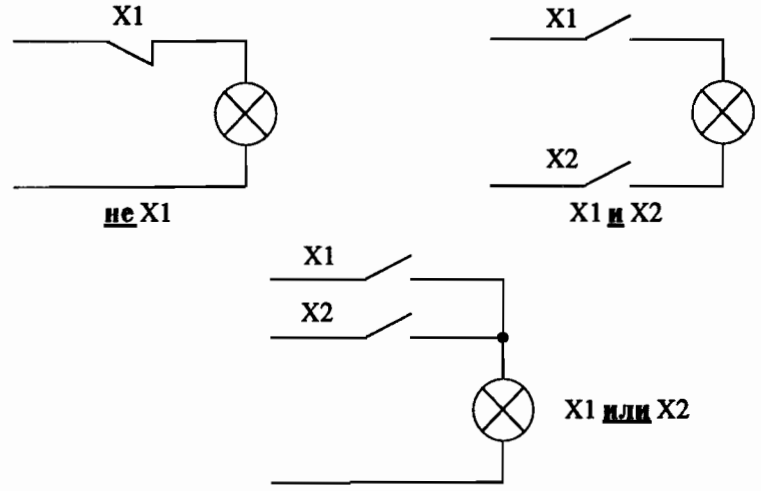


Рис. 66

Из простых выключателей мы собрали действующие модели ***вентилей***, т.е. схем, реализующих логические операции. Таким образом, опираясь на материал 1-й главы, мы плавно перешли к изучению ***логических принципов работы вычислительной техники.***

Однако в учебнике 2-я глава открывается изложением ***физических принципов.*** Поэтому на время приходится прерваться и изучить принцип действия одной из разновидностей выключателя — МОП-транзистора.

4. МОП - ТРАНЗИСТОРЫ

Основная цель заключается не столько в том, чтобы познакомить аудиторию с физическими основами работы p—n-перехода, сколько в том, чтобы снять с ЭВМ налет таинственности (особенно в нынешнее время проторенессанса мистической культуры в лице домовых и психотерапевтов).

Почему именно МОП-транзистор выбран для изучения ? Потому, что внутри современных ЭВМ четвертого поколения происхо-

дят те же физические процессы, только транзисторы стали очень маленькими. На 1 см^2 размещается более полумиллиона элементов (транзисторов, конденсаторов, резисторов). Эти элементы составляют ***интегральную микросхему*** (ИМС).

Интегральные микросхемы часто называют чипами от английского слова chips, обозначающего щепки или жареный картофель. Основа чипов — кремний, имеющий четыре валентных электрона. Чистый кремний не обладает электрической проводимостью. Но при внесении в него микропримеси пятивалентного элемента (фосфора) возникает лишний электрон, а при добавлении мельчайшей частицы трехвалентного элемента (обычно бора) возникает "дырка", и кремний приобретает соответственно электронную или дырочную проводимость.

Миниатюризация интегральных схем не беспредельна. Причина этого не только в возрастающей трудности проектирования "масок" для изготовления микросхемы (с. 153). Чем миниатюрнее изделие, тем больше оно подвержено внешним влияниям. Это порождает жесткие требования к технологическому процессу. В помещении цеха микропроцессорного производства запыленность не превышает 4 частиц на литр (в обычном цехе этот показатель равен миллиону!).

Кроме того, при плотности свыше 10 миллионов элементов на квадратный сантиметр снижается помехоустойчивость микросхемы. Непредсказуемое переключение микроскопического транзистора может произойти под влиянием диффузии атомов в кристалле кремния, при действии космического излучения, слабых полей.

Однако пора от физических и технологических проблем (т.е. от ***специальных*** вопросов) вернуться к общему вопросу курса —к логическим основам устройства ЭВМ. Этот вопрос имеет гораздо большее общеобразовательное значение, поэтому остановимся на нем подробнее. Но перед этим для обобщения (хотя бы минимального) перечислим основные микросхемы, входящие в состав ЭВМ: различные процессоры, постоянное запоминающее устройство (ПЗУ), перепрограммируемое запоминающее устройство (ППЗУ), оперативное запоминающее устройство (ОЗУ), генератор тактовых импульсов, преобразователи входных-выходных сигналов.

5. ЧТО УМЕЕТ ЭВМ ?

Мы знаем, что метод пошаговой детализации позволяет представить алгоритм как последовательность вспомогательных алгоритмов. Вспомогательные алгоритмы, как нам известно, состоят из команд. Но рассуждения можно продолжить: каждая команда состоит из нескольких операций, которые, в свою очередь, делятся на еще более простые...

Так что же "на самом деле" умеет ЭВМ ? Ответ на этот вопрос зависит от того, что именно подразумевается под термином "ЭВМ" и с какой точки зрения на нее смотрят (табл. 11).

Таблица 11

С точки зрения:	ЭВМ умеет:	ЭВМ состоит из:
пользователя	исполнять алгоритмы	щели для диска, дисплея ...
программиста	исполнять команды алгоритмического языка	процессора, ОЗУ, ПЗУ ...
системного программиста	исполнять машинные команды	регистров, ячеек памяти ...
проектировщика ЭВМ	выполнять логические операции	вентилей, микросхем ...
разработчика микросхем	ничего	кристалла кремния с примесями ...

Во время изучения 1-й главы мы рассматривали ЭВМ как универсальный "черный ящик" со сменяемыми алгоритмами обработки информации. Иными словами, мы пребывали в верхней строке таблицы и не опускались до изучения внутренностей машины.

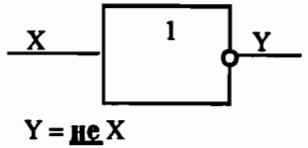
Теперь же мы начали с самой нижней строчки (см. "МОП-транзисторы") и кратко познакомились с устройством микросхем. Поднимемся вверх на одну строку и соберем из микросхем процессор.

6. ПРОЕКТИРУЕМ ПРОЦЕССОР

Независимо от принципа работы вентилей договоримся их обозначать в соответствии с принятым у нас стандартом (с. 149 учебника). Принятые обозначения можно объяснить. Значок "&" (английское and) используется для записи операции **и** в логических выражениях. Единица в вентиле "или" — это остаток от устаревшего обозначения "≥1" (легко убедиться, что этот вентиль действует именно так).

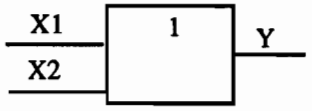
Для пояснения логики работы вентилей можно воспользоваться ***таблицами истинности***. Далее приведены таблицы истинности трех основных вентилей, реализующих три основные логические операции: **не** (инвертор), **или** (дизъюнктор), **и**

(конъюнктор). Заметим, что элемент "и" работает с сигналами "1" так же, как элемент "или" с "0".



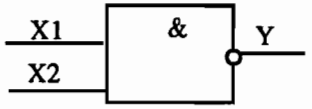
Y = **не** X

X	Y
0	1
1	0



Y = X1 **или** X2

X1	X2	Y
0	0	0
1	0	1
0	1	1
1	1	1



Y = X1 **и** X2

X1	X2	Y
0	0	0
1	0	0
0	1	0
1	1	1

При изучении 1-й главы мы убедились, что любые сложные логические выражения можно свести к комбинации трех логических операций. Следовательно, перечисленные три вентиля обладают *функциональной полнотой*, т.е. этих трех элементов достаточно для составления любых логических схем.

Так как на машинном языке величины логического типа кодируются нулевым или единичным битом, то ЭВМ "не замечает" разницу между логической переменной и одноразрядным двоичным числом. Следовательно, в процессоре допустимы арифметические операции над логическими величинами. Кроме того, их можно сравнивать.

Разработаем схему, реализующую уравнение:

$$Y = X1 > X2.$$

Начнем с составления таблицы истинности (табл. 12).

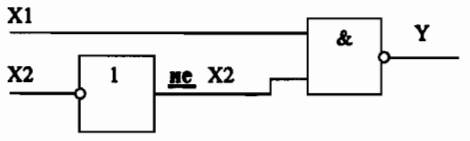


Рис. 67

Таблица 12

X1	X2	Y
0	0	0
1	0	1
0	1	0
1	1	0

Таблица истинности определяется уравнением:

$$Y = X1 \text{ и не } X2.$$

Теперь легко изобразить соответствующую схему (рис. 67). Каждая схема с логической точки зрения представляет собой выражение. Различные схемы, дающие при одинаковых входных сигналах одинаковые результаты, соответствуют равным логическим выражениям. Например, мы убедились, что

$$X1 > X2 \text{ равно } X1 \text{ и не } X2.$$

Равенство логических выражений определяется *основными законами алгебры логики* и следствиями из них.

Если учитель разделяет наше убеждение, что законы логики имеют некоторое общеобразовательное значение, то на работе вентилей и их комбинаций ему следует остановиться подробнее, чем это сделано в учебнике. Чтобы не вынуждать школьников зубрить логические формулы, можно подготовить плакат (табл. 13).

Таблица 13

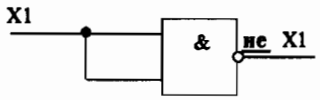
Основные законы алгебры логики		
Закон	**Для "или"**	**Для "и"**
переместительный (коммутативный)	x **или** y = y **или** x	x **и** y = y **и** x
сочетательный (ассоциативный)	x **или** (y **или** z) = (x **или** y) **или** z	(x **и** y) **и** z = x **и** (y **и** z)
распределительный (дистрибутивный)	(x **или** y) **и** z = = x **и** z **или** y **и** z	x **и** y **или** z = = (x **или** z) **и** (y **или** z)
инверсии (формулы де Моргана)	**не** (x **или** y) = = **не** x **и не** y	**не** (x **и** y) = = **не** x **или не** y

Для удобства разработчиков выпускаются микросхемы, реализующие сложные логические функции. В том числе "и-не" ("элемент Шеффера"), "или-не" ("элемент Пирса"), сумматоры, триггеры, счетчики, шифраторы и т.д. Логическое устройство

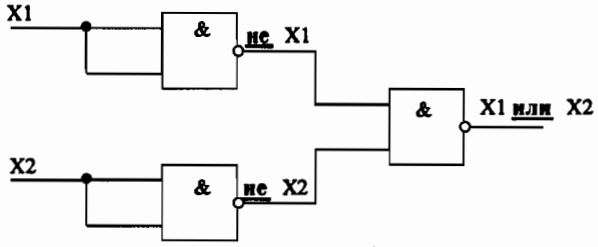
большинства из них мы рассмотрим чуть позже. Важно, что одно и то же логическое выражение может быть реализовано на различных схемах, т.е. набор микросхем обладает *функциональной избыточностью.*

В качестве иллюстрации в учебнике (с. 149) показана реализация логической операции с помощью вентиля "или-не" (рис. 78). Точно так же можно строить любые схемы только из вентилей "и-не". Из них легко собираются схема "не" и схема "или" (рис. 68). (На рис. 68, б реализована одна из формул де Моргана.)

Любое, на первый взгляд сложное, почти "разумное" действие ЭВМ можно разбить на элементарные "микродействия". Рассмотрим устройство некоторых логических элементов, выполняющих эти "микродействия".



а)



б)

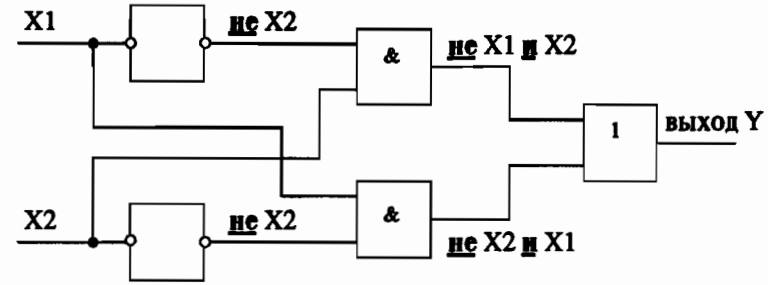
Рис. 68

Начнем с так называемой ***схемы сравнения*** (рис. 69), которая вырабатывает единичный сигнал на выходе, если входные сигналы не равны между собой, и выдает нулевой сигнал в обратном случае (упражнение 3, а на с. 154). Заметим в скобках, что иногда схемой сравнения называют инверсный вариант.

На рисунке 69,б приведено стандартное обозначение схемы сравнения, а на рисунке 69,в работа схемы сравнения иллюстрируется на примере регулирования потоков некоторой жидкости (один из наглядных способов иллюстрации) при различных входных сигналах.

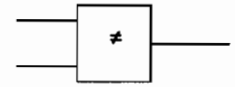
Заметим, что схема сравнения дает на выходе младший разряд суммы двоичных чисел, поданных на входы:

$$0+0=0$$
$$0+1=1$$
$$1+0=1$$
$$1+1=10$$

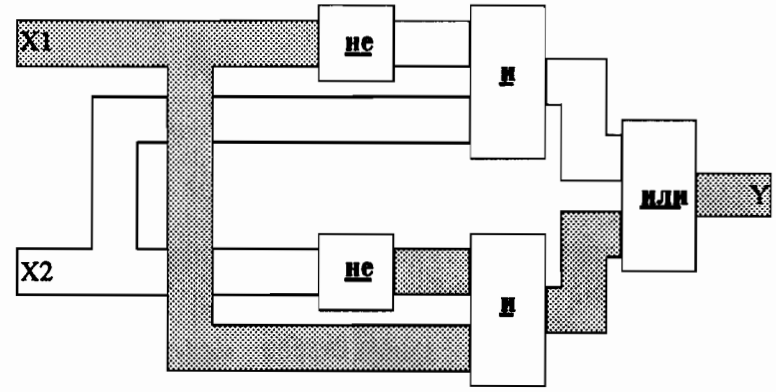


Y=не X1 и X2 или X2 и X1

а)

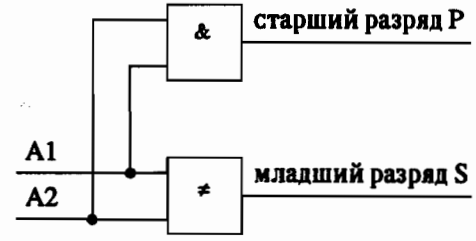


б)

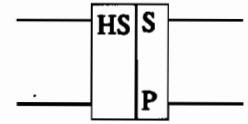


в)

Рис. 69



а)



б)

Рис. 70

Используя элемент сравнения, можно построить схему, определяющую сумму двух двоичных одноразрядных чисел A1 и A2 (рис. 70, а). Такая схема позволяет складывать два бита. Она называется ***полусумматором***. Обозначение полусумматора приведено на рисунке 70, б.

Современные ЭВМ оперируют восьми-, шестнадцати-, тридцатидвух- и шестидесятичетырехразрядными числами. Чтобы понять, как складываются в ЭВМ многоразрядные числа, попробуем сначала вручную (на доске, в тетрадях) сложить два восьмиразрядных числа, проделав это по шагам, начиная с младших разрядов. При этом воспользуемся привычным алгоритмом сложения в столбик. Только вместо "один в уме" будем писать: "перенос равен 1".

шаг 1
$$+\begin{array}{r}10010110\\01011011\\\hline\end{array}$$
1 (0 + 1 = 1, перенос равен 0)

шаг 2
$$+\begin{array}{r}10010110\\01011011\\\hline\end{array}$$
01 (1 + 1 + 0 = 10, перенос равен 1)

шаг 3
$$+\begin{array}{r}10010110\\01011011\\\hline\end{array}$$
001 (1 + 0 + 1 = 10, перенос равен 1)

шаг 4
$$+\begin{array}{r}10010110\\01011011\\\hline\end{array}$$
0001 (0 + 1 + 1 = 10, перенос равен 1)

шаг 5
$$+\begin{array}{r}10010110\\01011011\\\hline\end{array}$$
10001 (1 + 1 + 1 = 11, перенос равен 1)

шаг 6
$$+\begin{array}{r}10010110\\01011011\\\hline\end{array}$$
110001 (0 + 0 + 1 = 1, перенос равен 0)

шаг 7
$$+\begin{array}{r}10010110\\01011011\\\hline\end{array}$$
1110001 (0 + 1 + 0 = 1, перенос равен 0)

шаг 8
$$+\begin{array}{r}10010110\\01011011\\\hline\end{array}$$
11110001 (1 + 0 + 0 = 1, перенос равен 0)

Для построения схемы, складывающей многоразрядные числа, потребуется ***каскад***, составленный из ***полных сумматоров***. Каждый полный сумматор (рис. 71) делает один шаг "сложения в столбик", т.е. выполняет сложение двух очередных разрядов с учетом переноса. Таким образом, полный сумматор должен складывать три отдельных бита, поэтому в его состав войдут два полусумматора (отсюда префикс "полу-"):

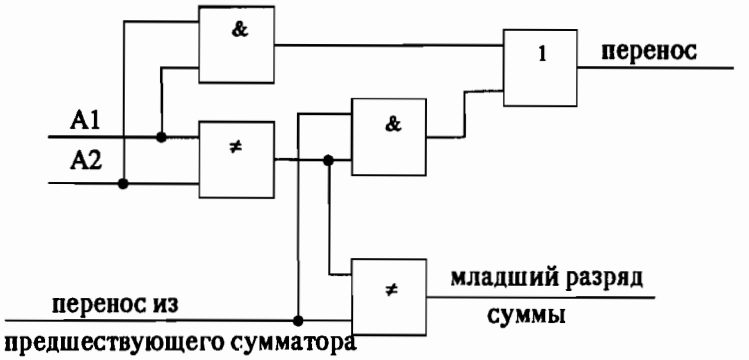


Рис. 71

Здесь предложим учащимся заполнить таблицу истинности для всех возможных комбинаций трех входных сигналов.

С помощью сумматора можно выполнять любые арифметические операции: сложение, вычитание (сводится к сложению уменьшаемого с числом, обратным вычитаемому), умножение (многократное сложение), деление (многократное вычитание) и т.д.

Для осуществления различных операций процессор обладает небольшой собственной памятью, каждая ячейка которой называется регистром. Чтобы понять принцип работы регистра, рассмотрим RS-триггер (об инверсном выходе P для упрощения умалчиваем). Схема RS-триггера, построенного на вентилях "или-не", приведена на рисунке 79 (с. 150 учебника). Можно также воспользоваться вентилями "и-не" (рис. 72). Попробуйте предложить учащимся составить такую схему самостоятельно.

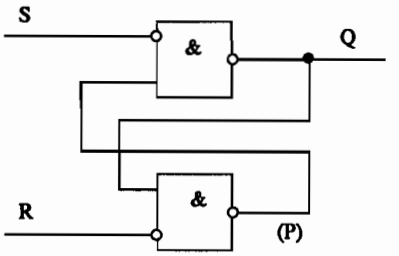


Рис. 72

Составляя таблицу истинности триггера, учтем, что его выход является входом в следующий момент времени. Поэтому введем такие обозначения:

Входные сигналы
- S — установка единицы (set);
- R — установка нуля (reset);
- $Q_{стар}$ — выход в предыдущий момент времени.

Выходной сигнал — Q.

Таблица 14

S	R	$Q_{стар}$	Q	Примечание
0 0	0 0	1 0	1 0	сохраняется предыдущее значение выхода
1 1	0 0	0 1	1 1	при установке единицы предыдущее состояние не играет роли
0 0	1 1	0 1	0 0	при установке нуля предыдущее состояние не играет роли
1 1	1 1	0 1	? ?	запрещенное состояние входов (S = R = 1)

Условное обозначение RS-триггера изображено на рисунке 73:

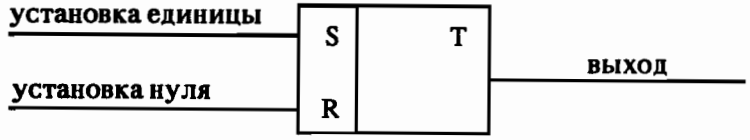


Рис. 73

В процессоре используются и другие типы триггеров, но они не слишком отличаются от RS-триггера, поэтому не будем их рассматривать.

Чтобы связать процессор с другими устройствами, потребуются различные преобразователи. Сделаем схемы некоторых из них.

Шифратор (рис. 74) преобразует входной сигнал, обозначающий десятичное число от 0 до 7 (например, нажата соответствующая кнопка), в трехразрядное двоичное число. Одновременное нажатие двух кнопок запрещено.

Для того чтобы понять устройство шифратора, разработаем схему шифратора с двумя выходами. Сколько входов будет иметь

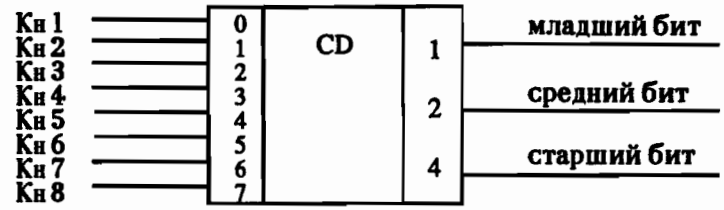


Рис. 74

схема? Понятно, что число входов равно 2^2, т.е. четырем. Обозначив входы через X_i, а выходы через Y_i, составим таблицу истинности (табл. 15).

Таблица 15

десятичное число	двоичное число
0	00
1	01
2	0
3	11

X1	X2	X3	X3	Y1	Y2
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

По таблице истинности легко составить принципиальную схему шифратора. Итак, связь с устройствами ввода установлена.

Рассмотрим теперь обозначение дешифратора (рис. 75), преобразующего трехзначное двоичное число в сигнал, соответствующий десятичному числу от 0 до 7. Затем составим схему дешифратора с двумя входами, используя уже известные вентили (решения не приводятся). Появилась связь с устройствами вывода.

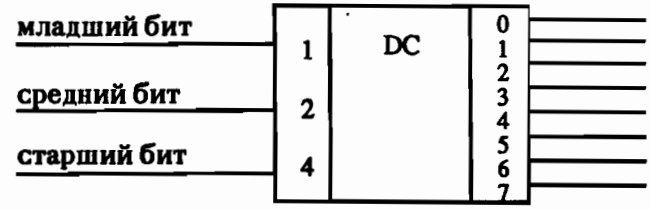


Рис. 75

Изученных элементов вполне достаточно, чтобы построить принципиальную схему процессора (мы фактически уже разработали фрагменты одноразрядного процессора).

Другая важнейшая часть ЭВМ — это память. Поэтому перейдем к изучению логических основ устройства машинной памяти.

7. РАЗРАБАТЫВАЕМ СТРУКТУРНЫЕ ПРИНЦИПЫ ОРГАНИЗАЦИИ ПАМЯТИ

Изготавливать всю память из триггеров дорого. К тому же, несмотря на высокое быстродействие, триггер имеет существенный недостаток — он "забывает" информацию при отключении питания. Поэтому при проектировании ОЗУ, ПЗУ и других видов памяти используются физические явления, отличные от тех, что происходят в транзисторе.

Как правило, чем больше объем запоминающего устройства, тем меньше его быстродействие, тем оно дешевле, тем ниже оно располагается в иерархической структуре памяти ЭВМ. Ниже на схеме (рис. 76) показан процесс ***чтения*** данных из разных видов памяти. При ***записи*** данных направления информационных потоков меняются на противоположные.

Рис. 76

На схеме видно, что запрос к внешней памяти выполняется ступенчато, как в известном детском стихотворении: "Король, его величество, просил ее величество, чтобы ее величество спросила у молочницы, нельзя ль доставить масло на завтрак королю. Придворная молочница пошла к своей корове..." (рис. 77)

Итак, в ЭВМ применяется много разновидностей запоминающих устройств. Эти устройства отличаются емкостью, быстродействием, работают на разных физических принципах. Однако все запоминающие устройства похожи *логически*, т.е. базируются на однотипных структурных принципах.

Любое запоминающее устройство строится из простейших элементов, каждый из которых хранит один бит информации.

Рис. 77

Для увеличения быстродействия ЭВМ ее различные части обмениваются информацией большего объема, от 8 до 64 бит у разных марок ЭВМ. Информационный объем сообщения, одновременно передаваемого внутри ЭВМ от одного блока к другому, называется ***машинным словом***.

Элементы памяти объединяются в ***ячейки***, каждая из которых содержит стандартное число бит, как правило, кратное машинному слову. Каждая ячейка имеет собственный номер — ***адрес***. Таким образом, понятия величины в ЭВМ, передачи параметров, исполнения команд присваивания, ввода и вывода наполняются материальным содержанием.

И в самом понятии величины открываются новые грани. Мы теперь можем взглянуть на величину "глазами ЭВМ". Как видим, ***имя*** величины есть лишь непонятное для ЭВМ, но удобное для человека сочетание букв. Встречая имя величины, ЭВМ *идентифицирует* номер ячейки, содержащей заданную величину. Поэтому имена в информатике часто называют ***идентификаторами***.

Значение величины для ЭВМ — это содержимое ячейки памяти с определенным адресом. ***Вид*** величины позволяет ЭВМ прогнозировать, каким образом и в какой момент последуют операции записи в память и извлечения из памяти значения величины. ***Тип*** величины информирует ЭВМ о ***формате записи значения в ячейке***. Например, для хранения величины типа лог достаточно одного бита.

При использовании языка высокого уровня человек может не знать, в каких ячейках хранятся значения величин алгоритма. Процессор сам следит за распределением памяти, устанавливает соответствия между именами и адресами величин. Но во многих языках программирования есть возможность прямого доступа к памяти, когда человек оперирует не именами, а адресами величин.

Теперь, повинуясь воле авторов учебника, мы должны на время прервать увлекательный процесс проектирования ЭВМ и дать исторический обзор развития вычислительной техники.

8. ПОКОЛЕНИЯ, СЕМЕЙСТВА И БРАК

Первый прообраз современной ЭВМ был построен на лампах в конце 30-х гг. С точки зрения сегодняшнего дня это был бракованный экземпляр, дающий сбои каждые семь минут. Впрочем, у него было одно неоспоримое достоинство — он служил прекрасным обогревателем помещения и зимой в машинном зале отопление не включали.

От этого обогревателя и произошли ЭВМ, историю которых принято, как у людей, делить на поколения. Правда, время жизни человеческого поколения примерно 80 лет, а ЭВМ — 10 лет.

Первая "настоящая" ЭВМ была создана в 1945 г. Она получила имя ENIAC (Electronic Numerical Integrator And Calculator), имела 20 ячеек памяти и выполняла 5 тыс. операций в секунду. ENIAC была представителем первого поколения.

Деление ЭВМ на поколения достаточно условно. Все зависит от критерия. В учебнике в качестве критерия взята элементная база. Существуют и другие критерии, например быстродействие. Деление на четыре поколения традиционно, но вряд ли необходимо для подробного изучения в общеобразовательной школе.

Более важно, на наш взгляд, выделить ***концептуальные*** этапы развития вычислительной техники (не только электронной). Таких этапов три:

1. Середина прошлого века. Чарлз Беббидж. Разделение арифметико-логического и запоминающего устройства. Появление терминов "адрес" и "код операции". Первые перфокарты. Изобретение команды условного перехода.
2. 1945 г. Джон фон Нейман. Современная архитектура вычислительной машины. Принцип хранимой программы.
3. 1984 г. Отход от традиционной фон-неймановской архитектуры. (Микропроцессор, занимающий площадь менее 1 $см^2$, на самом деле состоит из нескольких связанных между собой процессоров, работающих параллельно.)

Раз есть прародитель и поколения, то должны быть и семейства. Принято выделять четыре семейства (как и любая классификация в информатике, деление ЭВМ на семейства весьма условно и устарело в момент принятия).

Самые мощные — это суперЭВМ. Длина машинного слова в них составляет 64 бита и более. У них большой объем ОЗУ,

измеряемый десятками Мбайт. Второе семейство — большие ЭВМ. Обычно 32-разрядные и с меньшим объемом ОЗУ.

Затем миниЭВМ. Длина машинного слова 16 или 32 бита и сравнительно небольшое ОЗУ. Перечисленные три семейства объединяют ЭВМ, имеющие достаточно большие ресурсы и предназначенные для одновременной работы нескольких пользователей (мультипрограммный режим).

И наконец, микроЭВМ. Они, как правило, являются персональными, т.е. предназначены для одного пользователя. В настоящее время выпускается так много разновидностей ПЭВМ, что трудно обобщить их технические характеристики. Машинное слово содержит от 8 до 32 бит. Объем ОЗУ колеблется от десятков Кбайт до одного Мбайта и даже более того. (Объем ОЗУ связан с разрядностью процессора и размером шины адреса. Для 16-разрядной шины адреса максимальный объем ОЗУ равен 2^{16} байт = 2^6 Кбайт = 64 Кбайт. Однако имеется техническая возможность увеличить объем адресуемой памяти, что и делается в большинстве ПЭВМ.)

Пришло время вернуться к процессору и разобраться, зачем мы его собрали.

9. ПРОЦЕССОР ВЫПОЛНЯЕТ ПРОГРАММУ

Наш процессор умеет выполнять арифметические операции, операции сравнения и логические операции. Часть процессора, которую мы спроектировали, назовем ***арифметико-логическим устройством (АЛУ)***.

Процессор выполняет указанные операции не по собственной инициативе, а повинуясь написанной человеком программе. Она в память процессора, конечно, не поместится, поэтому будем считать, что машинная программа хранится в оперативной памяти (иногда и в ОЗУ вся программа не помещается).

Подобно программе, написанной на алгоритмическом языке, машинная программа представляет собой последовательность команд. Но в отличие от известных нам команд, машинная команда имеет стандартный формат и строго фиксированную длину, кратную машинному слову.

В машинной команде указывается операция, которую должен выполнить процессор, и адреса величин, над которыми должна быть произведена эта операция:

К О М А Н Д А	
Код операции	Адреса величин

В зависимости от количества величин, указанных в машинной команде, процессоры делятся на трехадресные, двухадресные, полутораадресные, одноадресные и безадресные (стековые). Ни-

же показаны форматы команд одноадресного и трехадресного процессоров:

К О М А Н Д А	
Код операции	Адрес аргумента или результата
Что делать?	Откуда взять или куда положить?

К О М А Н Д А			
Код операции	Адрес первого аргумента	Адрес второго аргумента	Адрес результата
Что делать?	Откуда взять?		Куда положить?

Поскольку во время выполнения программы необходимо часто обращаться к оперативной памяти, то возложим на процессор дополнительные функции: чтение команды из памяти; извлечение из памяти значений величин, указанных в команде; преобразование команды в последовательность команд АЛУ; запись результатов выполнения команды в память; вычисление порядкового номера следующей команды. Ответственную за эти функции часть процессора назовем ***устройством управления (УУ)***.

Устройство управления содержит два важных регистра — Счетчик Команд (СК) и Слово Состояния Процессора (ССП).

В специальной литературе по ЭВМ коллективного пользования (серии ЕС и т.п.) аббревиатура ССП иногда расшифровывается как Слово Состояния Программы. Это специальный регистр, предназначенный для обработки прерываний в мультипрограммном режиме. Как правило, этот регистр 64-разрядный, первая половина которого описывает текущее состояние процессора, а оставшиеся 32 бита — состояние программы в момент прерывания.

Нас же интересуют всего два бита регистра ССП, хранящие значение ***знака результата последней команды*** S (Signum). Для определенности можно считать, что соответствие содержимого двух битов ССП и значения S задается таблицей:

ССП		Значение S
0	0	+1
0	1	0
1	0	−1
1	1	переполнение

Теперь процессор готов к работе. Разберемся, как он это делает, исполняя готовые программы, а затем предложим учащимся самостоятельно составить простые машинные программы (с. 154—159 учебника).

В учебнике на с. 156 даны примеры команд процессора ЭВМ "Электроника УК НЦ" (микропроцессор К1801 ВМ2). Для любителей экзотики приведем мнемонические обозначения этих команд на языке ассемблера (табл. 16).

Таблица 16

Команда	Мнемокод
стоп	HALT
очистить x	CLR X
увеличить x на единицу	INC X
уменьшить x на единицу	DEC X
переслать из x в y	MOV X,Y
прибавить x к y	ADD X,Y
вычесть x из y	SUB X,Y
сравнить x с y	CMP X,Y
сравнить x с нулем	TST X
если меньше переход на d	BLT .D
если равно переход на d	BEQ .D
если больше переход на d	BGT .D
если не меньше переход на d	BGE .D
если не равно переход на d	BNE .D
если не больше переход на d	BLE .D
переход на d	BR .D

Итак, имеется процессор и различные запоминающие устройства. Нам осталось, во-первых, подключить устройства ввода-вывода и, во-вторых, научить процессор ими управлять. Начнем по порядку.

10. ПОДКЛЮЧАЕМ ПЕРИФЕРИЙНОЕ ОБОРУДОВАНИЕ

Прежде всего договоримся о двух вещах.

1. Мы не будем изготавливать внешние устройства (т.е. подробно разбираться в том, как они работают), а станем рассматривать их в качестве готовых исполнителей вроде Робота или Чертежника.

2. Мы изучим только то оборудование, которое имеется у нас "живьем" в дисплейном классе. О прочих типах внешних устройств будем лишь коротко упоминать.

Сначала остановимся на ***внешней памяти***, о которой мы уже говорили вкратце. К устройствам внешней памяти относится накопитель на гибких магнитных дисках (ГМД, дискетах,

флоппи-дисках). Еще его называют дисководом. Он имеется в классе, мы можем увидеть щель для дискет, можем потрогать дискету, увидеть прорези для магнитных головок.

Кроме дисковода, к устройствам внешней памяти относятся стриммер, накопитель на сменных пакетах, накопитель на магнитном барабане и, наконец, накопитель на жестких дисках, так называемый "винчестер".

Кстати, о "винчестере". Программисты любят придумывать и употреблять жаргонные словечки, часть из которых русского происхождения. Например, слово "обеспече́ние" с неправильным ударением на четвертом слоге (как "компа́с" у моряков или "до́быча" у шахтеров). Но большинство слов программистского слэнга имеет английское происхождение. Объясним часть из них (кроме тех, происхождение которых объяснено в других местах текста).

1. Файл. От английского FILE — папка, картотека.
2. Принтер. От английского PRINTER — печатающее устройство.
3. Сканнер. От английского SCANNER — блок развертки в телевизоре (SCAN — пристально разглядывать).
4. Джойстик. От английского JOYSTICK — рычаг управления.
5. Стриммер. От английского STREAMER — магнитофонная лента.
6. Плоттер. От английского PLOTTER — устройство для построения графиков, диаграмм, чертежей по точкам.
7. Диджитайзер. От английского DIGITIZER — устройство для кодирования информации (графической) в цифровом виде. (Вспомним BINARY DIGIT, BIT — двоичная цифра.)
8. Винчестер. Есть несколько версий происхождения этого термина. Приведем две из них. 1-я версия: вначале выпускался герметизированный дисковод с двумя дисками по 30 Мбайт, что обозначалось "30/30", подобно калибру охотничего ружья "WINCHESTER". 2-я версия: дисковод на жестких дисках был разработан в одном из английских филиалов фирмы IBM в городе WINCHESTER.

Не рассматривая подробно работу устройств внешней памяти, приведем список их технических характеристик (табл. 17).

Кроме устройств внешней памяти, к периферийному оборудованию относятся устройства ввода-вывода. Из огромного перечня этих устройств рассмотрим клавиатуру, монитор и принтер. Эти устройства есть у нас в дисплейном классе.

Клавиатура позволяет набирать отдельные символы, слова, фразы. С ее помощью мы как бы пишем процессору письмо. Для отправки письма служит специальная клавиша ввода. Впрочем, ученики уже прекрасно умеют обращаться с клавиатурой, не

Таблица 17

УСТРОЙСТВА ВНЕШНЕЙ ПАМЯТИ
НАКОПИТЕЛЬ НА СМЕННЫХ ДИСКАХ
тип носителя (пакет, дискета); емкость носителя, Мбайт или Кбайт; размер носителя, дюймы; число дисков; число рабочих поверхностей; число дорожек на поверхности; скорость передачи данных, Кбайт/с; плотность записи, бит/мм.
СТРИММЕР
тип ленты; емкость, Мбайт; скорость передачи данных, Кбайт/с; плотность записи, бит/мм.
НАКОПИТЕЛЬ НА ЖЕСТКИХ ДИСКАХ
тип носителя (барабан, винчестер); емкость, Мбайт; время доступа к записи, мс; скорость обмена информацией, Кбайт/с.

забывают нажимать клавишу ввода и не сидят перед дисплеем в ожидании ответа на неотправленное письмо.

Остается добавить, что, с точки зрения пользователя, клавиатуры отличаются друг от друга лишь расположением клавиш.

У отечественных ЭВМ верхние шесть клавиш образуют слово "ЙЦУКЕН", у зарубежных ЭВМ — "QWERTY". Существуют комбинированные клавиатуры, на которых латинские символы расположены в порядке "QWERTY", а русские — "ЙЦУКЕН".

Такое расположение клавиш традиционно. Так было принято на пишущих машинках для замедления (!) работы человека, чтобы механические детали не задевали друг друга.

Поскольку внутри клавиатуры ЭВМ нет механических деталей, то в настоящее время разрабатываются другие расположения клавиш, ускоряющие работу. Однако широкого распространения эти разработки пока не получили.

В таблице 18 приведен перечень главных технических характеристик монитора и принтера. (Некоторые характеристики монитора меняются в зависимости от типа преобразователей, связывающих ЭВМ с монитором.)

Кроме того, перечислим еще некоторые устройства ввода-вывода: "мышь", "джойстик", сканнер, диджитайзер, микрофон, световое перо, тактильный экран, плоттер, громкоговоритель.

Таблица 18

УСТРОЙСТВА ВЫВОДА
МОНИТОР
тип, число цветов (цветной, монохромный); размер экрана по диагонали, мм; формат экрана в текстовом режиме, a×b, знаков; емкость буфера, байт; разрешающая способность, т.е. число точек на экране в графическом режиме, a×b.
ПРИНТЕР (АЦПУ)
тип (матричный, термографический, струйный, литерный, строчный, лазерный и т.д.); число цветов; скорость печати, строк/мин или символов/мин; ширина строки, символов; способ подачи бумаги (ручной, автоматический); срок службы красящего элемента, часов; число одновременных закладок; количество шрифтов; возможность печати букв кириллицы и способ обеспечения этой возможности.

Подробнее мы с ними познакомимся тогда, когда они появятся у нас дома или хотя бы в школьном классе.

Наша ЭВМ почти готова. Имеется процессор, память, устройства ввода-вывода. Осталось научить процессор командовать всей этой аппаратурой. Но предварительно повторим пройденное. Воспользуемся при этом опорным конспектом (табл. 19).

Решим несколько задач, подобных тем, которые мы решали в самом начале курса. При этом обратим внимание на технические характеристики различных устройств наиболее распространенного сегодня стандарта для персональных ЭВМ.

Кроме того, познакомимся с удобным способом расчетов в информатике, сводящихся к операциям над степенями двойки. При расчетах удобно пользоваться справочной таблицей 20.

З а д а ч а 1. Информационная емкость человеческой яйцеклетки приблизительно равна 2^{33} бит. На скольких "винчестерах" (по 20 Мбайт) можно уместить генетическую информацию одного человека?

Р е ш е н и е. 1. Переведем биты в Мбайты:

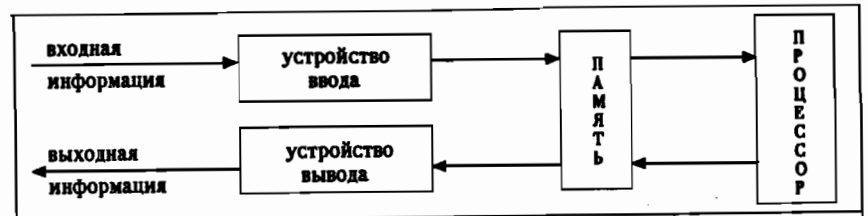
$$2^{33} \text{ бит} = 2^{30} \text{ байт} = 2^{10} \text{ Мбайт (1 Гигабайт).}$$

2. Найдем количество "винчестеров":

$$2^{10} \text{ Мбайт} / (2 * 10) \text{ Мбайт} = 2^9 / 10 = 51.2.$$

О т в е т: 52 "винчестера".

Таблица 19



Кодирование информации	Передача информации	Накопление информации	Обработка информации
виды информации: числа тексты алгоритмы графики рисунки звуки показания приборов нотные знаки химические уравнения	ввод: клавиатура "мышь" "джойстик" сканнер световое перо диджитайзер микрофон тактильный экран вывод: монитор принтер плоттер громкоговоритель	внутренняя память: ОЗУ СОЗУ ПЗУ ППЗУ внешняя память: флоппи-диск "винчестер" стриммер пакет дисков магнитный барабан	УУ, АЛУ

Таблица 20

n	2^n	Соотношения между единицами
0	1	1 байт = 2^3 бит
1	2	
2	4	1 Кбайт = 2^{10} байт
3	8	
4	16	1 Мбайт = 2^{10} Кбайт = 2^{20} байт
5	32	
6	64	
7	128	1 Кбит = 2^{10} бит
8	256	
9	512	1 Мбит = 2^{10} Кбит = 2^{20} бит
10	1024	

Задача 2. Емкость одного условного печатного листа равна приблизительно 32 Кбайта. Сколько чистого времени (без учета смены бумаги) потребуется для распечатки текста одной газеты (2 усл.п.л.): а) на матричном (64 символа в секунду) принтере; б) на лазерном (512 символов в секунду) принтере?

Решение. 1. Представим все числовые данные в виде степеней двойки:

$$32 = 2^5; 64 = 2^6; 512 = 2^9.$$

2. На матричном принтере:

$$2 * 2^5 * 2^{10} / 2^6 = 2^{10} = 1024 \text{ (с)} \approx 17 \text{ (мин)}.$$ **Ответ:** 17 мин.

3. На лазерном принтере:

$$2 * 2^{15} / 2^9 = 2^7 = 128 \text{ (с)} \approx 2 \text{ (мин)}.$$ **Ответ:** 2 мин.

Задача 3. Сколько таких газет (без учета иллюстраций) умещает один стриммер емкостью 64 Мбайта?

Решение. 1. Переведем Мбайты в Кбайты:

$$64 \text{ Мбайт} = 2^6 \text{ Мбайт} = 2^{16} \text{ Кбайт}.$$

2. Вычислим число газет (по 64 Кбайта каждая):

$$2^{16} / 2^6 = 2^{10} = 1024.$$ **Ответ:** 1024 газеты.

Задача 4. Оперативная память IBM PC AT имеет объем 640 Кбайт. Сколько страниц книги поместится в ней, если считать, что на странице 32 строки по 64 символа?

Решение. 1. Число символов на одной странице:

$$2^5 * 2^6 = 2^{11} \text{ (символов)}.$$

2. Информационный объем одной страницы (1 символ требует 1 байт):

$$2^{11} \text{ байт} = 2 \text{ Кбайт}.$$

3. Число страниц:

$$640 / 2 = 320.$$ **Ответ:** 320 страниц.

Задача 5. Монитор EGA имеет 320 точек по вертикали и 640 точек по горизонтали. Определите объем памяти, необходимый для хранения одного: а) "черно-белого" изображения; б) цветного изображения из 16 цветов.

Решение. 1. Количество точек на экране:

$$2^5 * 10 * 2^6 * 10 = 2^{11} * 100 \text{ (точек)}$$

2. "Черно-белое" изображение требует 1 бит на каждую точку. Вычислим необходимый объем памяти:

$$2^{11} * 100 \text{ бит} = 2 * 100 \text{ Кбит} = 2^{-2} * 100 \text{ Кбайт} = 25 \text{ Кбайт}.$$

Ответ: 25 Кбайт.

3. 16 цветов кодируются четырьмя битами. Поэтому объем памяти, необходимый для цветного изображения, вчетверо больше:

$$25 * 4 = 100 \text{ Кбайт}.$$ **Ответ:** 100 Кбайт.

Задача 6. Дисковод персональной ЭВМ IBM PS/2 предназначен для флоппи-дисков емкостью 720 Кбайт (3.5 дюйма, около 90 мм). Сколько цветных изображений можно уместить на одном флоппи-диске?

Решение. Так как одно изображение занимает 100 Кбайт, то уместится

$$720 / 100 = 7.2.$$ **Ответ:** 7 изображений.

Итак, мы почти собрали ЭВМ. "Почти", потому что ЭВМ без программ беспомощна, как новорожденный младенец. И подобно тому как человек умен пропорционально не только "качеству мозгов", но и своему опыту, "умственные способности" ЭВМ определяются не столько качеством аппаратуры, сколько качеством заложенных в нее программ. Если программист — дурак, то ЭВМ с его программами тоже безнадежно глупа. Вспомните, как дала сбой одна из первых бухгалтерских программ, встретив в ведомости фамилию Стоун-Джексон и принявшись вычитать одно из другого.

Займемся программным обеспечением.

11. ЭВМ = АППАРАТУРА+ ПРОГРАММЫ

Программное обеспечение — неотъемлемая часть любой ЭВМ, оно продается и поставляется вместе с аппаратурой. Начинающий пользователь, выбирая ЭВМ, руководствуется привычной логикой: чем мощнее, тем лучше. Он внимательно изучает технические показатели — быстродействие, разрядность, объем оперативной памяти. В результате ему часто приходится разрабатывать или заказывать необходимые прикладные программы, затрачивая порой суммы, в несколько раз превышающие стоимость аппаратуры.

Опытный пользователь прежде всего задается вопросом, насколько качественно, развито и доступно работающее на выбранной ЭВМ программное обеспечение. Можно, продав последнюю рубашку, приобрести для школы вычислительный комплекс "Эльбрус", и это будет бесполезная груда неплохой аппаратуры, так как нет подходящих учебных и обучающих программ. А можно выбрать КУВТ на базе БК-0010, технические характеристики которых таковы, что за согласие работать на этой технике надо бы приплачивать. И тем не менее благодаря героическим усилиям членов "Клуба пользователей БК" и от-

дельных граждан, создающих все новые и новые программы, пользы от БК в условиях школы будет больше, чем от "Эльбруса".

Наличием разнообразного программного обеспечения объясняется грандиозный успех персональных ЭВМ "Apple-2" на рубеже 70—80-х гг. Фирма не скрывала ни внутреннего устройства, ни операционной системы своей машины, и ежедневно появлялись все новые разработки программистов для "Apple-2".

В настоящее время качество программного обеспечения и распространенность соответствующих ЭВМ настолько взаимозависимы, что невозможно сказать, что важнее. С одной стороны, огромная популярность IBM-совместимых машин объясняется тем, что фирма IBM в специальных журналах регулярно публикует сведения о новых версиях операционной системы, о появлении новых инструментальных средств и приглашает всех желающих создавать новые программы.

С другой стороны, таких желающих становится все больше и больше, так как широкая распространенность стандарта IBM обеспечивает большой рынок сбыта программ. Например, электронные таблицы Lotus стоимостью менее 500 долларов принесли фирме Lotus Development Corporation доход около 1 миллиарда, потому что разошлись в количестве более 2 миллионов экземпляров.

Рассмотрим классификацию программного обеспечения:

Системное программное обеспечение включает в себя операционную систему; средства контроля, тестирования и диагностики ЭВМ; "надстройку" над операционной системой, облегчающую пользователю диалог с ЭВМ.

Прикладное программное обеспечение состоит из библиотеки стандартных процедур (тригонометрические, логарифмические и т.д.); пакетов прикладных программ (рассмотрены ниже); систем программирования (средства трансляции, редактирования, загрузки и отладки программ).

Главный инструментарий пользователя — пакеты прикладных программ. Для большинства ЭВМ имеются тысячи различных пользовательских пакетов. Но можно выделить пять основных разновидностей прикладных программ:

1) программы обслуживания сетей (обеспечивают связь с другими ЭВМ);
2) электронные таблицы;
3) графические пакеты;

4) системы управления базами данных;
5) системы подготовки текстов.

Как правило, современные прикладные программы не только совместимы с определенными ЭВМ, но и ***взаимно совместимы.*** Например, экономист готовит набор расчетных показателей с помощью электронных таблиц, затем, используя графический редактор, получает те же показатели в виде графиков или диаграмм (не вводя данные повторно), после чего снабжает графическую часть пояснениями с помощью текстового редактора. Нажимая определенную клавишу, он получает готовый материал, отпечатанный на бумаге.

Специальное программное обеспечение предназначено для решения уникальных задач (управление экспериментом, космической аппаратурой, технологическим процессом). Оно разрабатывается под конкретное применение и, как правило, не тиражируется и не продается. Иногда в состав специального программного обеспечения входит уникальная операционная система. Стоимость специального программного обеспечения может в десятки раз превосходить стоимость ЭВМ, для которой оно создано.

После такого общего обзора остановимся подробнее на системе программирования и операционной системе. В состав системы программирования входит три вида трансляторов: ассемблеры, компиляторы и интерпретаторы.

Ассемблер переводит мнемокоды команд в соответствующие коды операций, а имена переменных в адреса ячеек памяти. Понятно, что это очень простой способ перевода, поэтому ассемблер — это несложная программа, занимающая в памяти мало места.

Компилятор преобразует программу, написанную на языке программирования, в ***объектный модуль***, закодированный на машинном языке. Чтобы полученная программа могла работать, необходимо связать ее с другими модулями, преобразовать имеющиеся в ней обращения к стандартным процедурам в конкретные адреса, обеспечить соответствие типов. Поэтому объектный модуль обрабатывается ***редактором связей*** и превращается в ***загрузочный модуль.*** Загрузочный модуль можно запускать в работу. Таким образом, компилятор переводит всю программу целиком на язык машины.

Иначе действует интерпретатор. Он прочитывает первую команду алгоритма, переводит ее на язык ЭВМ и тут же дает приказ процессору исполнить эту команду. Затем он делает то же самое со второй командой алгоритма, потом с третьей и так продолжает до тех пор, пока не дойдет до конца программы. Иными словами, интерпретатор — специалист по синхронному переводу.

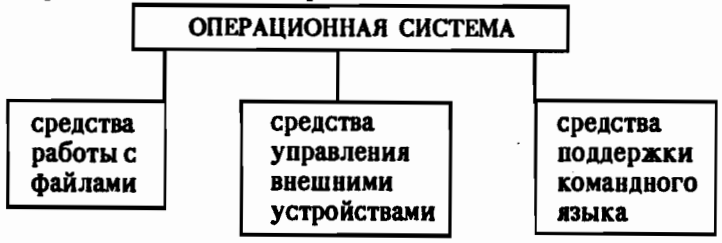
Самая "заметная" для начинающего пользователя часть программного обеспечения — операционная система (ОС). Наибольшее распространение на персональных ЭВМ получили три базовых семейства ОС: CP/M, MS-DOS и UNIX.

Система CP/M (control program for microcomputers) самая простая из названных. ***Ядро системы*** реализует лишь простейшие команды управления машиной и работы с файлами. Основная нагрузка при работе ЭВМ ложится на ***оболочку системы,*** т.е. прикладные программы, работающие под управлением ОС и обеспечивающие дружественный диалог с ЭВМ, выполнение различных сервисных операций (разметка дисков, тестирование, работа с сетью и т.д.). Система CP/M хороша для начинающего пользователя, но для опытного программиста часто бывает недостаточно ее возможностей. Для работы этой ОС требуется всего 16 Кбайт памяти, поэтому она используется на дешевых восьмиразрядных персональных ЭВМ.

Более сложна MS-DOS (MicroSoft-disk operating system). Она имеет достаточно развитые сервисные средства, древовидную иерархическую файловую структуру, более мощный командный язык. Система занимает приблизительно 60 Кбайт и, как правило, применяется на шестнадцатиразрядных ЭВМ.

Но наиболее удобны для профессионала системы семейства UNIX. В них сочетаются достоинства MS-DOS и большие дополнительные возможности. Преимущество семейства UNIX объясняется, в частности, наличием развитой библиотеки служебных программ. Система поддерживает многопользовательский режим работы. Командный язык UNIX во многом совпадает с языком C ("си"), кроме того, почти все модули системы написаны на этом языке. Однако вместе с библиотекой система требует около 5 Мбайт памяти.

Любую операционную систему, независимо от класса, можно условно разделить на стандартные составные части:



Итак, начав с МОП-транзистора, мы с учениками постепенно собрали электронную вычислительную машину. Конечно, специалисты могли бы это сделать и без нас, но не будем жалеть затраченных сил. Зато теперь мы убедились, что в "умных" машинах нет ничего сверхъестественного.

ГЛАВА 3. ПРИМЕНЕНИЯ ЭВМ

Эта глава учебника на первый взгляд традиционна. О применениях ЭВМ сегодня можно прочитать в десятках, если не сотнях, популярных книг, ориентированных на самый различный возраст. Без этой темы не обходится ни один курс информатики. Это и понятно, ведь умение грамотно применить ЭВМ — одна из основных целей курса, основа компьютерной грамоты.

Есть, однако, и существенное различие в изложении темы в учебнике и в других источниках. Учебник не только показывает, *что* можно делать с помощью ЭВМ, но и объясняет, *как* это делается.

Ничего подобного нет в многочисленных "Основах компьютерной грамотности": описание простейших операторов Бейсика или даже правил Пролога не объясняет работу больших информационных систем или простых редакторов текста. В результате в сознании ученика возникает разрыв между элементами программирования и приложениями ЭВМ, способность компьютера выполнять содержательную работу представляется чудом и никак не связывается с выполнением простейших учебных программ.

Авторы учебника сделали попытку избежать подобного раздвоения. Традиционный рассказ о применениях ЭВМ сопровождается фрагментами на алгоритмическом языке. Эти фрагменты демонстрируют принципиальную возможность указанных приложений, знакомят с идеями их реализации.

Показывая упрощенные фрагменты реальных систем, авторы учебника придерживаются ими же сформулированного "принципа корыта" : "рассказывая об океанских лайнерах, продемонстрируем корыто и объясним, почему оно плавает".

"Принцип корыта" применяется в школе не только на информатике: например, работа с простейшими электрическими цепями делает более понятной работу бытовой электросети.

Таким образом, первая цель изучения данной главы — знакомство с основными массовыми применениями современных ЭВМ. Вторая цель — понять методы, лежащие в основе этих применений. Ученик, хотя бы в общих чертах представляющий, как устроены сложные программы, не ждет от машины чуда, он

понимает, что ЭВМ — это техническое устройство, обрабатывающее информацию по известным ему принципам.

В зависимости от уровня и интересов учеников и имеющегося программного обеспечения изучение главы можно построить по-разному, сделав упор на практическом освоении компьютера или на усвоении важных теоретических понятий.

Основное средство достижения первой цели — практическая работа на компьютере с богатым программным обеспечением.

Средство достижения второй цели — построение и анализ информационных моделей и фрагментов алгоритмов.

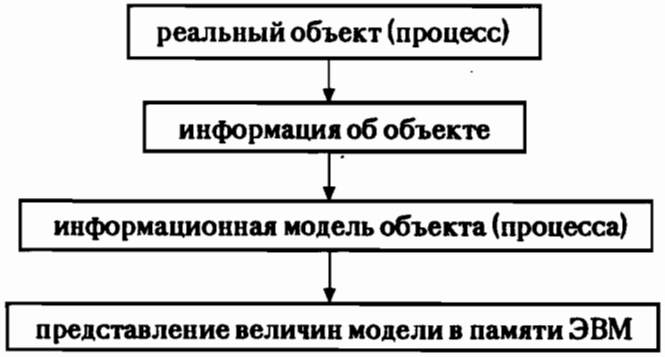
Ключевое понятие темы — информационная модель. Понятие модели, ее построение и оформление средствами алгоритмического языка описываются в § 21—22 учебника. Эти параграфы, как наиболее насыщенные теоретическим материалом, ниже рассмотрены подробнее.

Параграфы 23—27 описывают реальные применения ЭВМ, проиллюстрированные в соответствии с "принципом корыта" упрощенными информационными моделями и алгоритмами. Эти параграфы можно изучать в любом порядке, некоторые из них можно по усмотрению учителя опустить или заменить дополнительным материалом.

Параграф 28 — "гуманитарный". Это скорее не учебный материал, а затравка для дискуссии о морально-этических проблемах компьютеризации.

В § 21 вводится одно из базовых понятий курса — информационная модель. Перед разговором о моделях полезно повторить понятия информации и величины, необходимый для этого материал можно найти в § 1, 3, 11, 14, 15 учебника и пособия. Информационная модель понимается здесь как набор величин. Алгоритмы, работающие с этими величинами, в модель не входят. В учебнике используется выражение "в рамках модели". Как выглядят эти рамки, включающие в себя алгоритмы, показано в § 22.

Проанализировав и сопоставив понятия информации, модели, величины, кодирования, можно построить такую схему последовательных преобразований информации:



Очень интересен и важен первый этап этого преобразования. Отделение информации от реальных объектов — это первый шаг абстракции, начало любого научного знания.

На этом этапе происходит разделение существенной и несущественной информации: всякий реальный объект обладает столь многими свойствами, что учесть их все просто невозможно, а в конкретной задаче и не нужно. При этом информация, существенная для одной задачи, может оказаться несущественной для другой.

Ошибочное пренебрежение существенной информацией может привести к неверному решению, а учет несущественной — неоправданно усложнить решение.

Однако, вообще говоря, этот этап находится за рамками информатики. Выделение существенной информации — задача той науки или прикладной сферы, которая ставит задачу, например, все модели п.21.4 основаны на математических знаниях.

Информатика учит структурировать выделенную информацию и готовить ее к обработке на ЭВМ. Эти вопросы рассмотрены в пп.21.2—21.4.

Пункты 21.5 и 21.6 затрагивают довольно сложные вопросы, и в слабо подготовленном классе их можно не рассматривать.

Информационная модель Робота помогает разъяснить противоречие, которое, возможно, возникало еще в начале курса. В учебнике Робот представляется реальным, "железным" исполнителем, подключенным к ЭВМ, а на практических занятиях ученики видели, что и сам Робот живет только на экране ЭВМ. Модель М16 показывает, как можно имитировать Робота. Подобная модель, дополненная средствами отображения среды Робота на экране, использовалась вместо "железного" Робота в учебных целях.

В п.21.6 рассматривается *информационная модель алгоритма*. Этот пункт тесно связан с § 20. Фактически здесь показано, как ЭВМ исполняет алгоритмы, написанные не на машинном языке. Например, перевод алгоритма из текстовой формы в табличную в соответствии с моделью М18 — это фактически, компиляция. Исполнение закодированного таблицей алгоритма — это интерпретация, так что название алгоритма А92 не случайно.

Параграф 22 посвящен конструкции алгоритмического языка **исп-кон**. Эта конструкция позволяет объединить описания величин информационной модели и использующих эти величины алгоритмов. В отличие от **алг-кон**, конструкция **исп-кон** позволяет описать несколько самостоятельных алгоритмов с различными именами.

Таким образом, информационная модель исполнителя включает описание команд (действий) исполнителя в виде алгоритмов и описание его среды в виде общих величин этих алгоритмов.

Вспомним (§ 4), что, кроме среды и команд, есть еще одна важная характеристика исполнителя — *отказы*. Чтобы учесть их при построении модели исполнителя, надо описывать допустимые ситуации в строках **дано** алгоритмов-команд или пользоваться командой **утв**. Тогда при неверном использовании исполнителя условие в **дано** или **утв** нарушится, и ЭВМ сообщит об отказе.

Например, исполнитель маленький "Робот" (с.180) шагает вправо даже через стены. С учетом возможного отказа алгоритм "вправо" должен выглядеть так:

```
алг вправо
  дано | x < 32 и правее [x, y] = нет
нач
| x := x + 1
кон
```

С точки зрения программирования конструкция **исп-кон** нужна для введения ***глобальных величин***, т.е. величин, общих для нескольких алгоритмов.

Использование общих величин сокращает запись за счет отказа от описаний в заголовках и перечисления величин при вызове алгоритма.

Это, однако, не единственное и не главное преимущество. Использование общих величин вместо аргументов и результатов позволяет как бы спрятать стуктуры данных от вызывающего алгоритма. Раньше мы отмечали, что любой алгоритм можно переписать, не меняя его заголовок, это не потребует внесения изменений в вызывающие его алгоритмы. Использование общих величин вносит еще большую гибкость: можно изменить структуру данных информационной модели (см., например, упр.17 к § 21), при этом все изменения будут локализованы внутри **исп-кон**, никакие другие алгоритмы менять не придется.

Именно это свойство лежит в основе метода последовательного уточнения с использованием исполнителей. Более подробно об этом можно прочитать в книге А.Г.Кушниренко и Г.В.Лебедева "Программирование для математиков" (М.: Наука, 1988).

ПРИМЕРНОЕ ПЛАНИРОВАНИЕ МАШИННОГО ВАРИАНТА КУРСА

Машинный вариант курса предполагает работу на ЭВМ со школьным алгоритмическим языком, Роботом и Чертежником (системы Е-практикум, КуМир, Мега-Е и др.).

В приведенном ниже примере планирования до косой черты указано число уроков, проводимых без использования ЭВМ, после черты — число уроков, проводимых на ЭВМ.

Объем курса (всего)	102	68	34
Введение (всего)	3/2	3/1	2/1
1. Информация	1/1	1/—	1/—
2. ЭВМ	1/—	1/—	—/—
3. Обработка информации на ЭВМ	1/1	1/1	1/1
Глава 1. Алгоритмический язык (всего)	24/28	17/19	7/13
4. Исполнитель "Робот". Алгоритм	1/1	1/1	1/1
5. Исполнитель "Чертежник"	1/1	1/1	—/—
6. Вспомогательные алгоритмы. Аргументы	2/4	2/3	1/1
7. Арифметические выражения	1/—	1/—	1/—
8. Составные команды. Цикл n **раз**	1/1	1/1	—/1
9. Команды обратной связи. Цикл **пока**	2/4	2/4	1/3
10. Условия. **Если**, **выбор**, **утв**	2/2	1/1	1/1
11. Величины в АЯ. Команда присваивания	2/2	2/2	1/2
12. Результаты и алгоритмы-функции	2/2	2/2	—/1
13. Команды ввода/вывода. Цикл **для**	2/2	1/1	—/1
14. Табличные величины и работа с ними	2/2	2/2	1/1
15. Величины типов **лог**, **сим**, **лит**	2/2	1/1	—/1
16. Составление циклических алгоритмов	3/2	—/—	—/—
Упражнения на повторение	—/3	—/—	—/—
Глава 2. Устройство ЭВМ (всего)	7/5	4/4	3/1
17. Физические основы вычислительной техники	3/—	2/—	1/—
18. Команды и работа процессора ЭВМ	2/2	1/1	—/1
19. Устройства ввода/вывода информации	1/1	—/1	1/—
20. Работа ЭВМ в целом	1/2	1/2	1/—
Глава 3. Применения ЭВМ (всего)	18/16	9/11	4/3
21. Информационные модели	3/3	3/3	2/—
22. Исполнители в алгоритмическом языке	3/3	2/2	—/1

23.	Информационные системы	2/2	1/2	1/1
24.	Обработка текстовой информации	2/2	1/2	—/—
25.	Научно-технические расчеты на ЭВМ	3/2	—/—	—/—
26.	Моделирование и вычисления на ЭВМ	2/2	1/2	—/1
27.	ЭВМ в проектировании и производстве	2/2	—/—	—/—
28.	Заключение	1/—	1/—	1/—

ПРИМЕРНОЕ ПЛАНИРОВАНИЕ ПЕРВЫХ 24 УРОКОВ

Приведенное выше планирование подразумевает в основном традиционную организацию курса: учитель на уроке объясняет новый материал, разбирает алгоритмы, задает домашнее задание. На следующем уроке отвечает на вопросы, разбирает домашнее задание. Потом проводится один или несколько уроков по решению задач на ЭВМ.

Вот один из вариантов детального поурочного планирования при таком подходе для 102-часового курса (уроки 1—24):

1. Предмет информатики. Информация. Двоичное кодирование информации. Бит, байт. Понятие информации в информатике. Обработка информации как формальное преобразование последовательности символов (с. 3—11 учебника).

В классе: решение упражнений 8, 9 (с. 11 учебника).

На дом: 3,б, 4,в, 11,а—д, 12,а—г (с. 11—12), прочесть с.13—14:

дополнительно №: 10, 18 (с. 11—12).

2. (Работа на ЭВМ.) Техника безопасности. Работа с программами "Клавир" (освоение клавиатуры), "Черный ящик" (упр. типа 12—13, с.12), "Биты и байты" и др.

3. Составные части школьной ЭВМ, их назначение и характеристики. Потоки информации между частями ЭВМ (с.14—17).

В классе: упр. 2, 3 (с. 17).

На дом: 5, 6,в, 9 (с. 18); 14, 15 (с. 12); 8,а—в (с. 24).

4. Понятие исполнителя. Программирование, программа, язык программирования. Информационное производство, роль ЭВМ в развитии общества (с. 19—22).

В классе: упр. 1, 8,а—в (с. 23—24).

На дом: упр. 2, 4, 5, 7, 9; дополнительно №: 3, 11 (с. 23—24).

5. Проверка на ЭВМ решений задач урока 4 с использованием программ "Коза и капуста", "Встречные поезда", "БК-10", "Переправа" и др.

6. Исполнитель Робот. Запись алгоритмов на школьном алгоритмическом языке. Схема "Человек—ЭВМ—Исполнитель". Классификация ошибок (с. 25—28).

В классе: алг. А1, А2; упр. 1,б, 2,б, 3,а (с. 29).

На дом: 5,в, 5,д, 5 ,а, 7,а—г, 8,ж (с. 29—31);

дополнительно №: 4 (с.29), 10, 12 (с.32).

7. (Работа на ЭВМ.) Знакомство с системой программирования на основе школьного алгоритмического языка (Е-практикум, КуМир, Мега-Е и т.п.). Сдача на ЭВМ домашнего задания урока 5.

8. Исполнитель Чертежник. Аргументы команд (с. 32—36).

В классе: алг. А6, А7; упр. 2, 10, 11,а (с. 37—39).

На дом: 3, 4,б—в, 5,а—б, 9,а, 9,в (с. 37—38);

дополнительно №: 6,б—г, 11,г, 14 (с. 38—40).

9. Сдача на ЭВМ домашнего задания урока 8.

10. Основной и вспомогательный алгоритм. Вызов вспомогательного алгоритма. Метод последовательного уточнения (с. 40—43).

В классе: алг. А11, А12, А13, А14, А15; дополнительно №: А8, А17.

На дом: 4 (с. 45), 6,б (с. 46), 6,б (с. 47), 9 (с. 49), 12 (с. 51);

дополнительно №: 8, 10 (с. 48—49).

11. Сдача на ЭВМ домашнего задания урока 10. (Упр. 4 и 6 выполняются путем изменения подготовленных учителем алгоритмов А8, А17, А13—А15. При выполнении упр.9 используется подготовленный учителем алгоритм "фрагмент".)

12. Алгоритмы с аргументами. Модель памяти ЭВМ. Выполнение алгоритмов с аргументами (с. 43—45).

В классе: алг. А16, упр. 13, 18,а—б (с. 51—52).

На дом: 14, 15,б, 20, 21; дополнительно №: 22 (с. 51—52).

13. Сдача на ЭВМ домашнего задания урока 12.

На дом: 23, 24,а, 24,г, 24,ж;

дополнительно №: 24,е, 24,з (с. 52—54).

14. Сдача на ЭВМ домашнего задания урока 13 (путем изменения подготовленных учителем алгоритмов А23—А25).

На дом: подготовка к контрольной работе № 1.

15. Контрольная работа на ЭВМ по тематике уроков 10—14.

16. Арифметические выражения и их линейная запись. Кто вычисляет выражения? Стандартные функции и знаки операций алгоритмического языка (с. 54—57).

На дом: 1,в, 2,ж—и, 3,е, 3,и, 4,в, 4,н, 4,ф (с. 57—58).

17. Цикл "n раз" и его использование (с. 58—61).

В классе: алг. А27, А28, А29, А30; упр. 1, 2 (с. 61—62).

На дом: 3, 4, 6,а, 9; дополнительно №: 7, 8 (с. 61—62).

18. Сдача на ЭВМ домашнего задания урока 17.

19. Команды обратной связи. Управление Роботом вручную (роль Робота исполняет ученик). Цикл **пока**. Работа ЭВМ при выполнении цикла **пока**. Имитация диалога ЭВМ-Робот учениками. Примеры зацикливания и невыполнения тела цикла ни разу (с. 63—66).

В классе: алг. А31, А34, А32.

На дом: 4, 5, 6, 7,а, 1, 2; дополнительно №: 3 (с. 71—72).

20. Сдача на ЭВМ домашнего задания урока 19.

21. Проверка условия и выполнение тела цикла. Составление алгоритмов с циклом **пока**. Решение задач на составление алго-

ритмов с циклом пока (с. 66—71).

В классе: алг. А33, А35—А37, упр. 11 (с. 74).

На дом: 7,б, 8,а—б, 9; дополнительно №: 12, 13, 14 (с. 72—75).

22. Сдача на ЭВМ домашнего задания урока 21 (решение упр. 14 проверяется путем изменения подготовленных учителем алгоритмов А35—А36).

На дом: 7,в—г, 8,в—г, 10; дополнительно №: 7,д (с. 73—74).

23. Сдача на ЭВМ домашнего задания урока 22.

На дом: подготовка к контрольной работе № 2.

24. Контрольная работа на ЭВМ по тематике уроков 16—23.

ДРУГИЕ ФОРМЫ ПРОВЕДЕНИЯ КУРСА

Приведенный выше план уроков основан на примерном равенстве теоретических уроков и практических занятий на ЭВМ.

Возможны и другие способы преподавания курса, например основанные на проведении большинства уроков на ЭВМ.

Один из способов состоит в том, что на материал каждой темы, как правило, тратится два занятия. На первом учитель использует ЭВМ для иллюстрации нового материала, заранее прочитанного учениками, направляет деятельность учеников при решении упражнений в классе и задает домашнее задание. На втором ученики сдают домашнее задание на ЭВМ и получают задание прочесть к следующему уроку те или иные разделы учебника. Для сложных тем число уроков, на которых ученики сдают домашнее задание на ЭВМ, может быть увеличено.

Другой способ состоит в прохождении всего курса в режиме практикума. При этом на дом задается чтение соответствующего раздела учебника и решение упражнений; на уроке упражнения выполняются на ЭВМ; на дом задается чтение следующего раздела и решение упражнений; эти упражнения на следующем уроке выполняются на ЭВМ и т.д. Обычно в этом режиме дополнительно должны быть организованы консультации для учеников, чтобы они могли получить ответы на возникшие у них вопросы, уточнить неясности и т.п.

Естественно, в реальной жизни учитель может комбинировать те или иные способы прохождения курса в зависимости от доступности и надежности работы ЭВМ, наличия программного обеспечения, уровня подготовки учеников и пр. Он может, например, проходить одни темы в традиционном стиле, другие — в безмашинном варианте, третьи — в режиме практикума, изобретать свои способы преподавания, комбинировать разные стили в пределах одной темы и даже одного урока и т.п.

Ниже приведен фрагмент поурочного планирования 34-часового курса для такого комбинированного стиля, при котором около 80% уроков проводится в режиме практикума на ЭВМ.

1. Предмет информатики. Информация. Двоичное кодирование информации. Бит, байт. Понятие информации в информатике. Обработка информации как формальное преобразование последовательности символов (с. 3—11 учебника).

В классе: решение упражнений 8, 9 (с. 11 учебника).

На дом: 3,б, 4,в, 11,а—д, 12,а—г (с. 11—12).

Прочесть 2 (с. 13—17); дополнительно №: 10, 14, 18 (с. 11—12).

2. Работа на ЭВМ: составные части ЭВМ; техника безопасности; работа с программами "Клавир" (освоение клавиатуры), "Черный ящик" (упр. типа 12—13, с. 12), "Биты и байты" и др.

На дом: 1, 2, 4, 7, 8,а—в, прочесть 3 (с. 19—22);

дополнительно №: 3, 9, 11 (с. 23—24).

3. Работа на ЭВМ с программами "Коза и капуста", "Встречные поезда", "БК-10", "Переправа" и др.

Устный опрос по материалу 1—3.

4. Исполнитель Робот. Запись алгоритмов на школьном алгоритмическом языке. Схема "Человек—ЭВМ—Исполнитель". Классификация ошибок (с. 25—28).

В классе: алг. А1, А2; упр. 1,б, 2,б, 3,а (с. 29).

На дом: 5,в, 5,д, 8,ж (с. 29—31);

прочесть 5 (с. 32—36); 9,а, 9,в (с. 38); дополнительно №: 4, 7 (с. 29), 10,12 (с. 32).

5. Работа на ЭВМ: знакомство с системой программирования на школьном алгоритмическом языке (Е-практикум, КуМир, Мега-Е и т.п.); исполнение на ЭВМ подготовленных учителем алгоритмов А2, А8; сдача на ЭВМ домашнего задания урока 4.

На дом: прочесть 6 до п.6.7 (с. 40—43).

6. Выполнение на ЭВМ подготовленных учителем алгоритмов А11, А12, А13, А14, А15, А8, А17 с целью усвоения понятий основного и вспомогательного алгоритма, понятия вызова вспомогательного алгоритма и метода последовательного уточнения. Решение упр. 4 (с. 45) путем модификации алгоритмов А8, А17; упр. 6,б (с. 46) путем модификации А15; упр. 6,б (с. 47) путем модификации А14.

На дом: 4 (с. 45), 6,б (с. 46), 6,б (с. 47), 8,а, 9 (с. 49);

дополнительно №: 10, 11 (с. 49—50).

7. Сдача на ЭВМ домашнего задания урока 6.

На дом: 14, 15,б, 21, 24,а,24,ж; прочесть пп.6.7—6.9 6 (с. 43—45); дополнительно №: 22, 24,е, 24,з (с. 51—54).

8. Выполнение на ЭВМ подготовленных учителем алгоритмов А16, А20, А21, А22, А24, А25 с целью усвоения понятия вспомогательного алгоритма с аргументами. Сдача домашнего задания урока 7.

На дом: прочесть 7 (с. 54—57); упр. 1,в, 2,ж—и, 3,е, 3,и, 4,в, 4,н, 4,ф (с. 57—58).

9. Составные команды. Цикл n раз (с. 58—61).

В классе: алг. А27, А28, А29, А30; упр. 1, 2 (с. 61—62).

На дом: 3, 4, 6,а, 9;

дополнительно №: 7, 8 (с. 61—62).

ОГЛАВЛЕНИЕ

Учебное издание

Авербух Александр Владимирович
Гисин Владимир Борисович
Зейдельман Яков Наумович
Лебедев Геннадий Викторович

ИЗУЧЕНИЕ ОСНОВ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Зав. редакцией *Т.А.Бурмистрова*
Редактор *А.К.Компанец*
Художники *М.В.Лебедев, Б.Л.Николаев*
Художественный редактор *Ю.В.Пахомов*
Макет и исполнение иллюстраций *Е.Л.Лебедевой*
Корректор *Н.И.Новикова*

ИБ N 13530

Набор и верстка осуществлены на компьютерной технике с использованием редакционно-издательской системы Wave 4™ Bestinfo, Inc.
Подписано к печати 17.07.91. Формат 60×90$^{1}/_{16}$. Бумага офсетная. Гарнитура Таймс. Печать офсетная. Усл. печ. л. 19+0,25 форз. Усл. кр.-отт. 19,75. Уч.-изд. л. 15,42+0,42 форз. Цена 2 р. 10 к.

Ордена Трудового Красного Знамени издательство «Просвещение» Министерства печати и массовой информации РСФСР. 129846, Москва, 3-й проезд Марьиной рощи, 41.

Отпечатано при посредстве В/О «Внешторгиздат»
Отпечатано Графишер Гросбетриб Пёсиек ГмбХ · Эйн Мондрук-Бетриб
Gedruckt bei Graphischer Großbetrieb Pößneck GmbH · Ein Mohndruck-Betrieb

Тема 1 (7). Диалоговые системы. Табличные величины (12 ч)

Команды ввода и вывода информации. Диалоговые системы. Цикл для. Табличные величины. Линейные таблицы. Понятие о прямоугольных таблицах.

Тема 2(8). Логические, символьные и литерные величины (6 ч)

Логические величины, выражения и присваивания. Логические алгоритмы-функции. Использование логического алгоритма-функции в методе последовательного уточнения.

Символьные величины. Литерные величины. Длина литерной величины. Операция соединения. Вырезки. Команда присваивания вырезке.

Тема 3(9). Составление циклических алгоритмов (4 ч)

Рекуррентные соотношения. Рекуррентные вычисления. Метод рекуррентных соотношений. Однопроходные алгоритмы. Инвариант цикла. Рекурсия.

Тема 4(10). Устройство ЭВМ (12 ч)

Кодирование информации электрическими сигналами. Электронный ключ. Вентиль "не". Вентиль "или-не". Обозначения вентилей. Процессор. Элемент памяти (триггер). Память. Взаимодействие процессора и памяти. Поколения ЭВМ. Изготовление микросхем.

Память, процессор, программа. Основной алгоритм работы процессора. Команды процессора. Машинные программы.

Клавиатура. Монитор. Дисковод. Принтер. Взаимодействие основных частей ЭВМ. Магистраль. Устройства и исполнители. Исполнитель монитор (экран). Исполнитель клавиатура.

Алгоритмический язык и машинные коды. Компиляция. Интерпретация. Программа начальной загрузки. Операционная система.

Тема 5(11). Информационные модели (10 ч)

Информационные модели. Кодирование геометрической информации. Информационная модель исполнителя Робот. Исполнители в алгоритмическом языке. Общие (глобальные) величины в информационной модели исполнителя. Метод последовательного уточнения с использованием исполнителей.

Тема 6(12). Информационные системы (6 ч)

Информационные системы. Учебные информационные системы. Базы данных.

Тема 7(13). Прочие применения ЭВМ (12 ч)

Системы обработки текстов. Учебный редактор текстов. Приближенные вычисления на ЭВМ. Метод Монте-Карло. Моделирование и вычислительный эксперимент на ЭВМ. Метод дискретизации непрерывных процессов. Черчение на ЭВМ. Вычислительный эксперимент в компьютерном проектировании. Станки с числовым программным управлением (ЧПУ). Проектирование и производство — единый цикл.

Тема 8(14). От индустриального общества — к информационному (6 ч)

Проникновение ЭВМ во все сферы жизни. Ошибки в применениях ЭВМ.