Александр Дуванов
Алексей Рудь
Виктор Семенко

АЗЫ ПРОГРАММИРОВАНИЯ

ФАКУЛЬТАТИВНЫЙ КУРС

КНИГА ДЛЯ УЧИТЕЛЯ

Санкт-Петербург
«БХВ-Петербург»
2005

УДК 681.3.06(075.3)
ББК 32.973я721
Д79

Дуванов А. А., Рудь А. В., Семенко В. П.

Д79 Азы программирования. Факультативный курс. Книга для учителя. — СПб.: БХВ-Петербург, 2005. — 496 с.: ил.

ISBN 5-94157-584-X

В книге обобщен уникальный многолетний опыт Роботландского сетевого университета по обучению детей программированию. Материал излагается в стиле курса «Азы информатики» и значительно расширяет и углубляет тему программирования, которая в базовом курсе нацелена на общеобразовательные задачи. Повышенное внимание уделяется основам формализации и построения алгоритмов, тестированию и отладке программ. Подробно рассмотрены алгоритмические конструкции: циклы двух видов, ветвление, применение процедур, рекурсия; целый раздел посвящен принципам построения трансляторов.

Изложена методика проведения занятий и турниров по программированию.

Прилагаемый компакт-диск содержит программные среды исполнителей и решения всех заданий учебника и задачника.

Для учителей средних образовательных учреждений

УДК 681.3.06(075.3)
ББК 32.973я721

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. гл. редактора	*Людмила Еремеевская*
Зав. редакцией	*Григорий Добин*
Редактор	*Елена Михальчук*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн обложки	*Инны Тачиной*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 27.06.05.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 39,99.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-584-X

Оглавление

Состав комплекта

Комплект «Азы программирования» содержит всё необходимое для построения факультатива, сопровождающего школьный курс «Азы информатики». Он включает в себя:

- книгу «Азы программирования. Магия для начинающих» — учебник;
- книгу «Азы программирования. Задачи роботландских турниров» — задачник;
- **книгу «Азы программирования» — пособие для учителя**;
- CD с программными средами и дополнительными материалами (сопровождает книгу учителя).

Обращение к читателю

Уважаемые коллеги!

Мы с вами любим информатику, а особенно, — что греха таить! — программирование.

Ведь программирование — это математика информатики: «ум в порядок приводит», и её музыка: доставляет изысканное наслаждение!

Мы с вами, конечно, вкусили, вкушаем и, надо думать, будем вкушать яблочки с этого дерева, пусть не самые крупные, но не менее сладкие и полезные, насыщенные витамином увлечённости и азарта.

К сожалению, в последнее время интерес молодежи к программированию угасает. Как жаль! Ведь программирование — не только замена пустому времяпрепровождению. Программирование — это солидный багаж для вступления в успешную жизнь. Спрос на программистов только растёт. И получают они за свою работу хорошие деньги. Гораздо большие, чем простые компьютерные пользователи.

А чем сейчас увлекается молодежь на компьютерах? Правильно: в основном игрушками. Кому нужны такие «спецы»?

В Роботландском университете есть несколько программистских курсов: Азы программирования (31), Буки программирования (32), Web-программирование (43). Каждый из этих курсов не просто интересен, но и уникален. Но в основе всех их лежат подходы, которые восходят к великому Дийкстре.

Классик Э. Дийкстра говорил, что самый главный язык, который должен знать программист, — это свой родной, на котором он изъясняется в повседневной жизни.

Иными словами, соль программирования не в языке программирования, а в умении чётко сформулировать задачу, выдвинуть идею решения, разработать алгоритм! И только потом перевести алгоритм в программу, записав несколько заклинаний на языке посвящённых!

Этот тезис особенно ярко проявляет свою суть в «Азах программирования». Здесь нет проблем с языком — он тривиален, а идей, алгоритмов — выше головы!

Книга обобщает многолетний опыт Роботландского университета, а ключевые особенности обучения можно конспективно изложить так:

- «Голая» алгоритмика, практически без структур данных, т. к. цель — хорошо прочувствовать алгоритмические конструкции: *циклы двух видов*, *ветвление*, *применение процедур*, *рекурсию*.
- Привычка максимально использовать то, что есть, думать, как обойтись меньшими ресурсами.
- Не отвлекаться на «машинные» особенности.
- Универсальный язык записи, с которого легко перейти на любой процедурный. Языка-то практически нет — одни алгоритмы!
- А вот элементы структурности есть, подобно Паскалю и Си.
- Вкус не к «наворотам», а именно к *алгоритмам*, к *мышлению*.
- Большой класс задач, тренирующих применение рекурсии. Не так уж просто, даже для старшеклассников и студентов.
- Целый раздел и серия задач «Транслятор — это просто!» Становятся понятными (не только учителю, но и 5–8-классникам) принципы построения анализаторов и интерпретаторов.

История Кукарачи

Исполнитель Кукарача (первое имя Таракан) родился на Дальнем Востоке в 1984 году. Роботландии тогда ещё не было, а был новосибирский Муравей, придуманный Г. А. Звенигородским вместе с группой исследователей (в неё входил и Ю. А. Первин, будущий директор Роботландии).

Второй участник будущей Роботландии, А. А. Дуванов в то время работал преподавателем в Благовещенском пединституте. Интерес благовещенцев

к пионерским работам новосибирцев был велик. Настолько, что Дальний Восток приехал в Сибирь пообщаться с умными людьми, увидеть уникальную работу школьников на компьютерах своими глазами. По трагическому стечению обстоятельств как раз в это время от гриппа умер Г. А. Звенигородский. Ему было 32 года. Он даже не увидел издание своей книги, которая вышла уже после его смерти (Г. А. Звенигородский. Первые уроки программирования. Библиотечка Квант. Выпуск 41. Москва. Наука. 1985).

Разговор состоялся с Юрием Абрамовичем Первиным. Был показан школьный урок в младших классах за компьютерами «Агат». Была подробная содержательная беседа о новосибирском опыте, в частности, об исполнителе Муравей.

В Благовещенске закипела работа. В группу энтузиастов, которую возглавил А. А. Дуванов, вошли: О. Г. Какаулин, В. В. Немилостива, Ю. В. Прашкович, О. Д. Десятириков.

Появилась идея: взяв за основу среду Муравья, придумать такую её модификацию и такой учебный язык, которые, с одной стороны, были бы предельно просты и доступны самым маленьким, а с другой — позволили бы представить все основные управляющие структуры.

Результат первых опытов в этом направлении — исполнитель Тараканчик, работающий в командном режиме. Олег Какаулин спустя 18 лет реконструировал среду Таранканчика. Вы можете посмотреть, как выглядел Тараканчик на ЭВМ ИСКРА-1256 (и даже поработать с ним) на следующей ссылке:

www.botik.ru/~robot/history/chick.htm

Затем появился Таракан с возможностью программного управления. Позже, после поездки А. А. Дуванова на Кубу, Таракан получил второе имя — Кукарача.

Тараканы и Муравьи начали свой победный марш по нашим широким просторам. В качестве примера можно указать самарскую работу: Г. Н. Гутман, О. М. Карпилова. Муравьиные сказки. Книга для учащихся. Москва. Просвещение. 1993.

Самарский Муравей больше походит на Кукарачу (хотя в книге ссылки на роботландского исполнителя нет), чем на своего новосибирского однофамильца. Однако кукарачинский язык в Самаре неоправданно усложнён введением переменных, перечисляемым циклом, подпрограммой с параметрами и другими дополнениями, которые, с одной стороны, сводят на нет первоначальную простоту, а с другой — плохо вписываются в примитивную клетчатую среду и систему команд исполнителя.

Кукарача вошёл в состав курса Роботландии с самого начала. Алгоритмические структуры по своей сути — вещь житейская, общеобразовательная и необходимая для изучения именно в младших классах, когда формируется

стиль мышления ребёнка. Именно поэтому, а не с целью подготовки юных программистов, Кукарача внедрился в школы. Среда, понятная ребёнку, предельно простая система команд и структурный процедурный язык (без всяких `GOTO`) хорошо соответствовали поставленной задаче.

Придумав исполнителя, мы даже и не подозревали о всех его необычных возможностях. Первотолчком, закрутившим Кукарачу на новых оборотах, стали рекурсивные задачи Е. П. Лилитко, которые он придумал для своей дочки и любезно передал нам. Потом было много других задач, неожиданно радостных находок. Особый успех Кукарача приобрёл на курсе 31 Роботландского университета (www.botik.ru/~robot/ru).

Турнирные задания на этом курсе всегда неожиданны, а эмоциональный всплеск конкурсов имеет в университете повышенный накал. Каждый год куратор придумывает задачу для конкурса в надежде, что никто из ребят её не решит. Но такого ещё не случалось. Школьники дошли до того, что спокойно пишут трансляторы и даже решили для Кукарачи задачу о Ханойской башне (Первушин Данил, Снежинск)! И всё это притом, что в языке нет ни одной переменной!

История Корректора

Корректор, как и Кукарача, переехал в Переславль-Залесский из Благовещенска (дальневосточный город, центр Амурской области, граница с Китаем). Этот исполнитель был придуман А. А. Дувановым двумя годами позже Кукарачи, в 1986 году. В этом же году была подготовлена рукопись книги «Введение в программирование. Корректор».

Однако эта рукопись так и не появилась в печати. На следующий год Дуванов переехал из Благовещенска в Переславль-Залесский и поступил на работу в Институт программных систем АН СССР в лабораторию школьной информатики, которую возглавил, приехав в Переславль из Новосибирска, Ю. А. Первин.

В Переславле закипела работа над новыми идеями на уровне творческих безумств и вдохновения, съедающего всё время. Рукопись сначала просто была положена в стол, а потом на время забыта.

Тем не менее, Корректор был одной из самых первых программ, реализованных на Ямахе и вошедших в самую раннюю версию Роботландии. В дальнейшем, при переносе Роботландии сначала на УКНЦ, а затем на PC, Корректор остался на старой квартире. Мы не стали включать Корректор в новые версии Роботландии, потому что его программирование выходило за рамки общеобразовательного курса информатики для младших школьников, носило факультативный характер и предназначалось для детей, которые хотели заниматься именно программированием. На пристройку к Роботландии

кружка юных программистов не было сил и времени: нужно было разрабатывать и опробовать в школьных классах ставшие теперь традиционными роботландские общеобразовательные темы.

Только в 1992 году, когда для издательства «Педагогика-Пресс» готовилась книга «Необычайные приключения Пети Кука в Роботландии» (авторы А. Дуванов, Ю. Первин), рукопись по Корректору была извлечена из картонной коробки и использована при подготовке четырёх глав приключений главного героя.

Роботландский университет дал новую жизнь Корректору. Рекурсивные опусы Кукарачи у Корректора получили логическое продолжение, более оправданное в его псевдотьюринговской среде, чем на клетчатом поле коллеги.

Вступление

Программирование в школьном курсе информатики

Введение в программирование обязательно должно быть составной частью школьного курса информатики. Это как формулы в математике. Можно много говорить о важности математического мышления, но пока школьник на практике не пройдёт через вычисление гипотенузы по формуле Пифагора, все слова будут красивы, но бесполезны.

Так и в информатике. Можно много говорить об алгоритмическом мышлении, приводить примеры с алгоритмами перехода улицы и заварки чая, но пока школьник не напишет код с ветвлениями, циклами и рекурсией для конкретного формального исполнителя, не пройдёт через этап тестирования и отладки, разговор об алгоритмическом мышлении будет красив, но бесполезен.

С другой стороны, не хочется впадать и в другую крайность, подменять алгоритмическую культуру техническим программированием, а алгоритмику — языком программирования. Ведь не все школьники в будущем станут программистами, но алгоритмические навыки нужны всем как элемент культуры и как элемент повседневной практики, профессиональной (в любой области) и бытовой.

В курсе «Азы информатики» (см. начало следующего пункта) этот важный вопрос решается при помощи компромисса между простотой учебных исполнителей и полнотой алгоритмических построений на базе простого языка программирования.

Есть исполнитель (Кукарача, Корректор) с минимальной системой команд, есть язык программирования с минимальным числом интуитивно понятных конструкций и есть наглядные среды, в которых выполнение программы представлено визуально (перемещение исполнителей, кубиков, изменение сред на экране), а не скрыто в железных недрах системных плат. Серьёзные алгоритмические построения выполняются на этой примитивной базе в «чистом» виде, без технических деталей «настоящих» языков программирования и изощрённой сложности «настоящих» исполнителей.

Большая проблема для новичка — понятие переменной. Переменных нет в роботландском языке управления исполнителями. Вся техническая сторона программирования предельно проста и наглядна. Зато алгоритмические структуры — настоящие: процедуры, циклы, развилки, рекурсии. Задачи в средах Кукарачи и Корректора тоже настоящие (вплоть до построения трансляторов!). Блеск и восторг! Авторы верят, что читатели смогут в этом убедиться самостоятельно, без излишней агитации! Недаром Кукарача и Корректор живут уже 20 лет и продолжают набирать обороты.

«Азы программирования» и «Азы информатики»

«Азы информатики» — это базовый курс информатики для начинающих. Он рассчитан на 5 лет школьного обучения, начиная с пятого класса общеобразовательной школы. Автор курса — А. А. Дуванов. Сохраняя методические идеи классической «Роботландии», новый курс предлагает школьнику и педагогу современные средства для реализации педагогической задачи, делает обучение более эффективным, увлекательным и контролируемым. Основной методический приём курса — формирование концептуальных основ информатики через практические задачи, решаемые на компьютере. Курс поддержан электронными лабораториями и бумажными учебниками издательства «БХВ-Петербург». Дополнительную информацию о курсе можно получить на сайте **www.botik.ru/~robot** или от автора с адреса **kurs@robotland.pereslavl.ru**.

Тема программирования, конечно, отражена в курсе «Азы информатики», но в небольшом, общеобразовательном объёме (как закрепление алгоритмики на уровне кодирования формального исполнителя).

Используя же «Азы программирования», учитель сможет поставить факультатив для тех школьников 5–8 классов, для которых программирование станет будущей успешной профессией. Книга поможет увлечённым и азартным ребятам раскрыть свои интересы, развить способности.

Эмоциональный фон книги увлечёт темой, а её содержание научит серьёзному программированию, очищенному от шелухи сложных языковых конструкций и многочисленных технических деталей производственных языков программирования.

Материал книги обкатан многолетней практикой курсов Роботландского университета, выверен педагогически и является уникальным (серьёзные задачи в примитивных средах).

Примерный план факультатива

По мнению авторов, на базе «Азов программирования» можно построить двухгодичный факультатив по программированию для учащихся 5–8 классов.

В таблицах 1 и 2 приводится примерный план, рассчитанный на 68 часов (2 года, по 1 часу в неделю). В графе «Тема» обозначено соответствие изучаемого материала разделам книг «Азы программирования. Магия для начинающих. Книга ученика» («Учебник»), «Азы программирования. Задачи роботландских турниров» («Задачник»).

Первый год обучения

Таблица 1

Номер	Тема	Часы
Учебник. Часть I. Кукарача		
1	Глава 1. Кукарача и его среда обитания (среда, управление при помощи команд)	1
2	Глава 2. Вася экономит свой труд (понятие программы)	1
3	Глава 3. Новые команды и их повторение (процедурное программирование, команда цикла `ПОВТОРИ`)	1
4	Глава 7. Задачи 14–19	1
5	Глава 4. Кукарача на распутье (ветвления)	2
6	Глава 7. Задачи 1–6	2
7	Глава 5. Другой тип повторения (цикл `ПОКА`)	2
8	Глава 7. Задачи 7–11, 20–27	3
9	Глава 6. Кукарача хочет укусить себя за хвост (рекурсивные программы)	2
10	Глава 7. Задачи 12–13, 28	2
11	Олимпиада	1
12	Разбор решений	1
Задачник (задачи Кукарачи)		
13	Глава 1. Олимпиада 1996/1997	1
14	Глава 2. Турнир 1997/1998	2
15	Глава 3. Турнир 1998/1999	2

Таблица 1 *(окончание)*

Номер	Тема	Часы
16	Глава 4. Турнир 1999/2000	2
17	Глава 5. Турнир 2000/2001	2
18	Глава 6. Турнир 2001/2002	2
19	Глава 7. Турнир 2002/2003	2
20	Глава 8. Турнир 2003/2004	2
21	Глава 9. Олипмиада 2004/2005	1
22	Олимпиада	1
23	Разбор решений	1
Итого:		34

Второй год обучения

Таблица 2

Номер	Тема	Часы
Учебник. Часть II. Корректор		
1	Глава 8. Знакомство с исполнителем	1
2	Глава 9. Язык программирования Корректора	1
3	Глава 15. Задачи 7–12	1
4	Глава 10. Отладка программ	1
5	Глава 15. Задачи 13–17	1
6	Глава 11. Приёмы программирования Корректора	1
7	Глава 15. Задачи 18–21	1
8	Глава 12. Арифметика чисел, палочек и символов	2
9	Глава 15. Задачи 1–3, 22–26	2
10	Глава 13. Преобразования, подсчеты, редактирование	2
11	Глава 15. Задачи 4–6	2
12	Глава 14. Трансляторы	2

Таблица 2 *(окончание)*

Номер	Тема	Часы
Учебник. Часть III. Транслятор?.. Это очень просто!		
13	Глава 16. Язык Бэкуса-Наура	1
14	Глава 17. Кукарача и лексический анализ выражений	1
15	Глава 18. Ох уж эта рекурсия!	1
16	Глава 19. Лексический анализатор Корректора	1
17	Глава 20. Примитивный транслятор	2
18	Олимпиада	1
19	Разбор решений	1
Задачник (задачи Корректора)		
20	Глава 2. Турнир 1997/1998	1
21	Глава 3. Турнир 1998/1999	1
22	Глава 4. Турнир 1999/2000	1
23	Глава 5. Турнир 2000/2001	1
24	Глава 6. Турнир 2001/2002	1
25	Глава 7. Турнир 2002/2003	1
26	Глава 8. Турнир 2003/2004	1
27	Олимпиада	1
28	Разбор решений	1
Итого:		34

Содержание книги

В этой книге вы найдёте дополнительные материалы по темам, излагаемым в книге ученика, методические рекомендации и решение всех задач (решения заданий задачника помещены на CD, который сопровождает эту книгу).

Задач очень много, решения не всегда короткие, поэтому на большей части этих страниц вы обнаружите программные коды, снабжённые описаниями алгоритмов и наборами данных для тестирования.

Решаясь на такую объёмную публикацию, мы сказали себе:

- рабочие коды в книге обязаны быть: они должны демонстрировать на практике те приёмы и те стили программирования, которые обсуждаются на теоретических страницах. Если в кодах, поставляемых на CD, авторы

из-за недостатка времени (а чаще просто из-за лени) позволяют себе определённую небрежность, то в бумажном варианте это категорически невозможно! Все коды должны иметь идеальный вид без малейших расхождений с рекомендациями на страницах учебника;

- коды, опубликованные в книге, должны легко читаться и пониматься без всякого запуска на компьютере, с ними можно будет работать прямо на диване, ну, может быть, вооружившись ручкой и блокнотом. Этот принцип, конечно, универсально относится к любому программированию, но для учебных кодов он особенно важен. Поэтому программы мы предваряли подробными алгоритмическими описаниями и не жалели строк для комментариев (эта конструкция языка программирования использовалась нами чаще других);
- программирование немыслимо без тщательно спланированного тестирования. Все программные решения мы будем снабжать наборами продуманных тестов и создавать тем самым мостик от общих рекомендаций по отладке к привитию практических навыков тестирования.

Ох, и задали мы себе работку, ох, и возложили на себя ответственность! Читателю предстоит проверить, насколько хорошо авторы сумели выполнить поставленные перед собой задачи.

Авторство задач и решений

Условие каждой задачи и каждое решение сопровождаются в книге ссылкой на автора этой задачи и этого решения. Если такой ссылки нет, то это означает, что автором задачи и (или) решения является А. А. Дуванов.

Авторство иллюстраций

Автором всех иллюстраций в учительской, ученической книгах и задачнике является А. А. Дуванов. Для построения рисунков были использованы художественные заготовки А. А. Русса (художник Роботландии).

Содержание диска

Диск содержит:

- программные среды исполнителей Кукарача и Корректор (автор продукта А. А. Дуванов);
- коды программ, разбираемых на страницах учебника;
- коды программ-решений заданий учебника;

- коды программ-решений заданий задачника;
- методические рекомендации и описание решений заданий задачника в двух видах:
  - для печати в виде doc-файлов с картинками высокого разрешения;
  - для просмотра с экрана в виде гипертекстового приложения.

Программные коды Кукарачи располагаются на CD по адресу:
`./azpr/kurs/coc97/task/`

Программные коды Корректора располагаются на CD по адресу:
`./azpr/kurs/kor97/task/`

Папки `task` имеют следующую структуру:

- `I` (Часть I. Кукарача)
  - `1` (Решения задач первой главы части I.)
  - `2` (Решения задач второй главы части I.)
  - `3` (Решения задач третьей главы части I.)
  - `4` (Решения задач четвёртой главы части I.)
  - `5` (Решения задач пятой главы части I.)
  - `6` (Решения задач шестой главы части I.)
  - `7` (Решения задач седьмой главы части I.)
  - `book` (Коды программ из текста учебника.)
- `II` (Часть II. Корректор)
- `III` (Часть III. Транслятор?.. Это очень просто!)
- `IV` (Задачник)

Имена программ построены по следующему правилу:

`nmmkkxx.coc` — программа для Кукарачи;

`nmmkkxx.kor` — программа для Корректора.

Здесь:

- `n` — номер части;
- `mm` — номер главы внутри части;
- `kk` — номер пункта внутри главы;
- `xx` — номер задачи внутри пункта.

Если задача имеет несколько вариантов решения, то к имени добавляется ещё и номер варианта.

Пример

Файл

`./azpr/kurs/coc97/task/I/2/10205031.coc`

является первым вариантом решения задачи 3 из пятого пункта второй главы первой части книги учебника.

Методические рекомендации и описание решений заданий задачника располагаются на CD по адресу:

- `./book/print/` — doc-файлы для печати;
- `./book/screen/` — гипертекстовое приложение, стартовый файл `index.htm`.

Вы можете время от времени заглядывать в сетевой каталог:

ftp://ftp.botik.ru/rented/robot/univer/3/book/

В эту папку мы будем помещать обновления и дополнения материалов, содержащихся на CD к этой книге.

Благодарности

Исполнители Кукарача и Корректор не увидели бы свет, как и эта книга, если бы не пионерские труды Г. А. Звенигородского, проложившие первые тропинки для школьной информатики в Новосибирском академгородке.

Трудно представить появление героев (а значит, и самой книги) без работы благовещенской группы, в которой особую роль играли Олег Какаулин и Валентина Немилостива. Дополнительно тёплые слова благовещенским художникам, создавшим портреты Кукарачи (Наташа Медведева) и Корректора (Алексей Морозов).

Е. П. Лилитко можно назвать духовным отцом Кукарачи (вдохнул в исполнителя рекурсивную душу).

Непосредственными участниками затеи являются авторы великолепных задач и решений. Их имена украшают книгу в соответствующих местах. Особая благодарность талантливому человеку, юному программисту из Снежинска — Даниле Первушину. Его остроумные находки не раз приводили обложечных авторов в восхищение!

Я. Н. Зайдельман совершил над рукописью колоссальный труд: он проверил тексты, задачи, решения. В результате рабочих дискуссий многие страницы были улучшены принципиально.

Наконец, спасибо всем участникам роботландских курсов «Азы программирования» (Магам, Магистрам и всем остальным). Материал книги вырос из вашего нескучного коллективного труда.

Исполнители и язык программирования

Далее приводится краткое описание Кукарачи, Корректора и языка программирования этих исполнителей.

Исполнитель Кукарача

Кукарача — исполнитель, работающий на клетчатом поле (рис. 1).

Рис. 1. Клетчатое поле с исполнителем и кубиками

Кукарача может переползать из клетки в клетку вверх, вниз, влево, вправо, но не по диагонали. Выход исполнителя за пределы поля запрещён (отказ «Не могу»). В каждой клетке может находиться кубик с нанесённым на его грань символом. Кукарача, перемещаясь по полю, может толкать один или несколько кубиков перед собой и «сбрасывать» кубики за пределы поля.

Некоторые кубики на поле могут иметь «скрытые» символы, обозначаемые знаком вопроса. Считается, что такие кубики расположены в клетке «символом вниз». Толкая скрытый кубик, Кукарача перемещает его в следующую клетку по ходу движения, переворачивает и видит надпись.

Исполнитель в состоянии прочитать символ на кубике независимо от того, была ли запись символа обычной или скрытой.

Система команд исполнителя (СКИ) включает пять команд (рис. 2 и табл. 3).

Рис. 2. Система команд Кукарачи

Таблица 3

Команда Кукарачи	Как выполняется
ВПРАВО	Сместиться на одну клетку вправо
ВЛЕВО	Сместиться на одну клетку влево
ВВЕРХ	Сместиться на одну клетку вверх
ВНИЗ	Сместиться на одну клетку вниз
СТОЯТЬ	Пустая команда — исполнитель не выполняет никаких действий

Любая команда, кроме команды **СТОЯТЬ**, может привести к отказу «Не могу», если её выполнение уводит Кукарачу за пределы поля.

Кукарача способен выполнять в своей среде следующие проверки (запись <символ> обозначает шаблон, на месте которого может быть любой символ) (табл. 4).

Таблица 4

Запись условия	Результат проверки
<символ>	*Истина*, если Кукарача толкнул кубик с указанным в условии символом, *ложь* в противном случае
НЕ <символ>	*Ложь*, если Кукарача толкнул кубик с указанным в условии символом, *истина* в противном случае
ПУСТО	*Истина*, если клетка, в которую сместился Кукарача, пуста, *ложь* в противном случае
НЕ ПУСТО	*Ложь*, если клетка, в которую сместился Кукарача, пуста, *истина* в противном случае
ЦИФРА	*Истина*, если Кукарача толкнул кубик с цифрой (один из символов 0, 1, 2, 3, 4, 5, 6, 7, 8, 9), *ложь* в противном случае
НЕ ЦИФРА	*Ложь*, если Кукарача толкнул кубик с цифрой (один из символов 0, 1, 2, 3, 4, 5, 6, 7, 8, 9), *истина* в противном случае

Исполнитель Корректор

Корректор работает с длинной лентой, которая разбита на клетки (ячейки).

В ячейку может быть записан один символ. Символ — это буква, цифра, другие знаки, составляющие фиксированный алфавит исполнителя.

Считается, что лента спрятана в непрозрачный для Корректора футляр. Исполнитель видит только одну ячейку через специальное окно в футляре и может записывать в «оконную» ячейку (окно ленты) любые символы из своего алфавита. Корректор может перемещать окно вправо и влево вдоль ленты и, таким образом, записывать символы в любые ячейки на ней.

В среде исполнителя есть ящик — специальная ячейка памяти. В ящик Корректор может снимать копию символа из окна и, наоборот, копировать символ из ящика в окно на ленту (рис. 3).

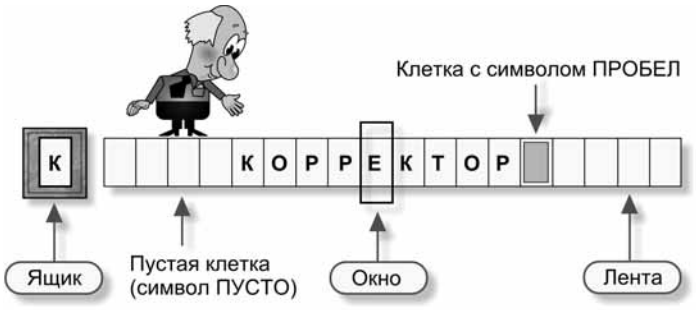

Рис. 3. Среда Корректора

На рис. 4 и в табл. 5 приводится полный набор команд Корректора и его алфавит.

Рис. 4. Система команд Корректора

Таблица 5

Команда Корректора	Как выполняется
ВПРАВО	Переместить окно на одну клетку вправо
ВЛЕВО	Переместить окно на одну клетку влево
ПИШИ <символ>	Записать указанный символ в клетку на ленте. В качестве символа можно указывать ключевые слова ПУСТО и ПРОБЕЛ
ЯЩИК+	Копировать символ с ленты в ящик
ЯЩИК-	Копировать символ из ящика на ленту
ОБМЕН	Поменять местами содержимое ящика и окна ленты
ПЛЮС	Заменить символ в окне символом, следующим по порядку в алфавите Корректора. Команда приводит к отказу (ситуация «Не могу»), когда в окне перед её выполнением записан последний символ алфавита
МИНУС	Заменить символ в окне символом, предыдущим по порядку в алфавите Корректора. Команда приводит к отказу (ситуация «Не могу»), когда в окне записан первый символ алфавита (специальный символ ПУСТО)
СТОЯТЬ	Пустая команда; её выполнение не вызывает никаких изменений в среде Корректора

Корректор способен выполнять в своей среде следующие проверки (табл. 6).

Таблица 6

Запись условия	Результат проверки
<символ>	*Истина*, если символ в окне совпадает с символом, указанным в условии, *ложь* в противном случае. В качестве символа можно указывать ключевые слова ПУСТО, ПРОБЕЛ и ЦИФРА
Я=Л	*Истина*, если символ в ящике совпадает с символом в окне, *ложь* в противном случае
Я#Л	*Истина*, если символ в ящике не совпадает с символом в окне, *ложь* в противном случае
Я>Л	*Истина*, если символ в ящике имеет больший порядковый номер в алфавите Корректора по сравнению с номером символа в окне, *ложь* в противном случае
Я<Л	*Истина*, если символ в ящике имеет меньший порядковый номер в алфавите Корректора по сравнению с номером символа в окне, *ложь* в противном случае

Перед любым условием может быть написан ключевой модификатор НЕ, который меняет смысл условия на противоположный.

Язык программирования исполнителей

Разделитель слов

Разделителем слов в языке служит один или несколько пробелов, а также конец строки.

Программа

Программа представляет собой последовательность процедур, записанных в любом порядке. Имя процедуры может быть использовано наравне с командой из СКИ исполнителя.

Процедура

Описание процедуры имеет вид, представленный на рис. 5.

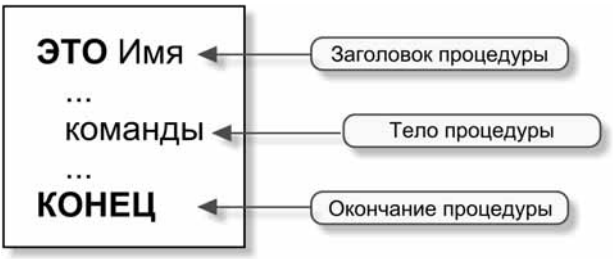

Рис. 5. Описание процедуры

Имя процедуры

Имя процедуры должно начинаться с буквы. Оно не должно содержать пробелов и не может совпадать с ключевыми словами языка программирования.

Комментарии

Комментарий имеет вид:

```
// текст комментария (до конца строки)
```

Комментарии могут располагаться как в процедурах, так и между ними.

Команды

Команды исполнителя представлены в табл. 7.

Таблица 7

Команда языка	**Как выполняется**
<команда из СКИ>	**Команда исполнителя** Выполнение определяется описанием команды в СКИ
<имя процедуры>	**Вызов процедуры** Выполняется процедура с указанным именем
ПОВТОРИ <число> <команда>	**Цикл ПОВТОРИ** Повторение выполнения команды указанное число раз
ПОКА <условие> <команда>	**Цикл ПОКА** Повторение выполнения команды, пока условие имеет значение *истина*. Проверка условия — перед выполнением команды
ЕСЛИ <условие> **ТО** <команда1> **ИНАЧЕ** <команда2>	**Команда ветвления** Проверяется условие и выполняется либо <команда1> (при значении условия: *истина*), либо <команда2> (при значении условия: *ложь*). Часть «**ИНАЧЕ** <команда2>» может быть опущена
{ <список команд> }	**Составная команда** Объединение нескольких команд в одну

Часть I

Кукарача: методика занятий и решение задач

Глава 1

Кукарача и его среда обитания

Общие рекомендации и дополнения

В этом разделе среда исполнителя предъявлена в виде клетчатого поля размером 10×10 (рис. 1.1).

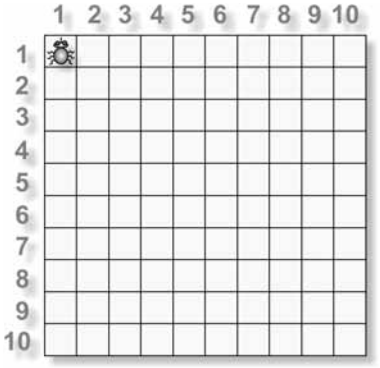

Рис. 1.1. Клетчатое поле с исполнителем

Однако многие интересные задачи будут относиться к более широкому клетчатому пространству и даже бесконечному во всех направлениях.

Кукарача ползает внутри поля, подчиняясь командам **влево**, **вправо**, **вверх**, **вниз**. Команда **стоять** является «пустой», она не меняет среду и положение исполнителя в ней (рис. 1.2).

Когда Кукарача натыкается на один или несколько кубиков, он толкает их перед собой, пока не вытолкнет за пределы поля. Сам исполнитель покинуть среду не в состоянии (сообщение «Не могу»).

Для представления Кукарачи можно использовать общую схему знакомства с исполнителем, вынесенную на классную доску, кодоскоп или демонстрационный экран (рис. 1.3).

Рис. 1.2. Система команд Кукарачи

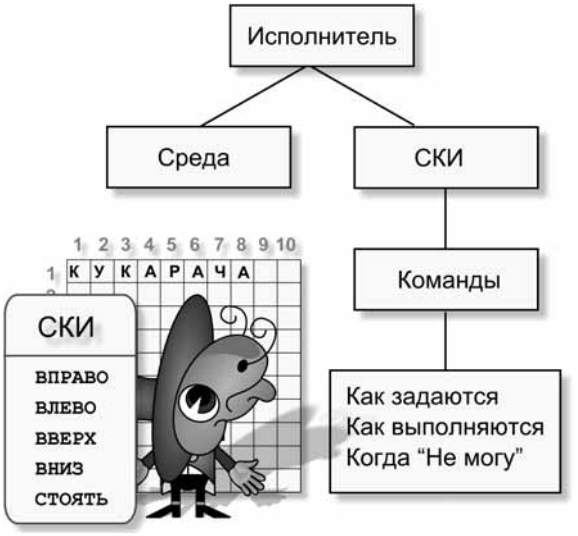

Рис. 1.3. Схема знакомства с исполнителем

Следуя нисходящему методу работы над проектом (технология «сверху вниз»: сначала общие идеи, затем проработка деталей), рекомендуется следующая последовательность этапов решения задачи:

1. Обсуждение задачи и построение траектории движения исполнителя.
2. Составление алгоритма в общем виде на русском языке.
3. Детализация алгоритма (возможно в нескольких уровнях) на русском языке.
4. Запись программы (последовательность команд из СКИ).
5. Выполнение.

Можно записывать программу параллельно с выполнением, так как это делал Петя в задаче с куликом. Такая работа полезна, т. к. наглядно показывает взаимосвязь алгоритма и программы. При этом хорошо видно, например, что одному пункту алгоритма может соответствовать несколько команд из СКИ.

Ответы на вопросы и решение задач

1.1. Знакомство с Кукарачей

1. Что сделает Кукарача, получив команду **НАЛЕВО**?

   Ответ. Покажет сообщение «Не понимаю».

2. Кукарача стоит в левом нижнем углу поля. Как он выполнит команду **ВНИЗ**?

   Ответ. Покажет сообщение «Не могу».

3. Кукарача находится в клетке (1,1). Где он окажется после выполнения следующих команд?

   а)
   ```
   ВПРАВО
   ВНИЗ
   ВПРАВО
   ВНИЗ
   ВНИЗ
   ```
   б)
   ```
   ВПРАВО
   ВПРАВО
   ВНИЗ
   ВЛЕВО
   ВЛЕВО
   ```
   в)
   ```
   ВПРАВО
   ВНИЗ
   ВЛЕВО
   ВНИЗ
   ВНИЗ
   ```
   г)
   ```
   ВПРАВО
   ВНИЗ
   ВЛЕВО
   СТОЯТЬ
   ```

   Ответ. а) в клетке (4,3); б) в клетке (2,1); в) в клетке (4,1); г) в клетке (2,2).

1.3. Задачи

1. Превратить молоток в моток (рис. 1.4).

Рис. 1.4

Решение

В табл. 1.1 приводится алгоритм решения и команды исполнителя, реализующие шаги алгоритма.

Таблица 1.1

Шаги алгоритма	Команды
1. Выбросить Л и шагнуть к следующей букве	`ВВЕРХ ВНИЗ ВПРАВО`
2. Выбросить О и шагнуть к следующей букве	`ВВЕРХ ВНИЗ ВПРАВО`
3. Подойти к концу слова	`ВПРАВО ВПРАВО ВПРАВО ВВЕРХ`
4. Уплотнить запись	`ВЛЕВО ВЛЕВО`

2. Превратить мишку в мышку (рис. 1.5).

Рис. 1.5

Решение

В табл. 1.2 приводится алгоритм решения и команды исполнителя, реализующие шаги алгоритма.

Таблица 1.2

Шаги алгоритма	Команды
1. Выбросить И	`ВВЕРХ ВВЕРХ ВНИЗ ВНИЗ`
2. Подойти к Ы	`ВНИЗ ВПРАВО ВПРАВО ВВЕРХ`
3. Поставить Ы	`ВЛЕВО ВНИЗ ВЛЕВО ВВЕРХ`

3. Починить колесо (рис. 1.6).

Рис. 1.6

Решение

В табл. 1.3 приводится алгоритм решения и команды исполнителя, реализующие шаги алгоритма.

Таблица 1.3

Шаги алгоритма	Команды
1. Поставить О	`ВВЕРХ`
2. Подойти к началу слова	`ВЛЕВО ВЛЕВО ВВЕРХ`
3. Уплотнить запись	`ВПРАВО ВПРАВО ВПРАВО`

4. Какую роль хочет играть Кукарача на поле (рис. 1.7)?

Рис. 1.7

Решение

В табл. 1.4 приводится алгоритм решения и команды исполнителя, реализующие шаги алгоритма.

Таблица 1.4

Шаги алгоритма	Команды
1. Сместить РОЛЬ	`ВПРАВО ВПРАВО ВПРАВО ВПРАВО ВПРАВО`
2. Подойти к К	`ВЛЕВО ВЛЕВО ВНИЗ ВНИЗ ВПРАВО`
3. Поставить К	`ВВЕРХ ВНИЗ`
4. Подойти к О	`ВПРАВО`
5. Поставить О	`ВВЕРХ`

5. Получить имя знакомой кошки Кукарачи (рис. 1.8).

Рис. 1.8

Решение

Возможные варианты имен МУРА, МУРКА, МУШКА, МАША, МАШКА.

Вариант МУРА.

В табл. 1.5 приводится алгоритм решения и команды исполнителя, реализующие шаги алгоритма.

Таблица 1.5

Шаги алгоритма	Команды
1. Выбросить Ш	`ВВЕРХ`
2. Выбросить слог КА	`ВПРАВО ВПРАВО`

Вариант МУРКА.

Предварительный алгоритм (табл. 1.6).

Таблица 1.6

Шаги алгоритма
1. Выбросить АШ
2. Подойти к началу слова
3. Придвинуть МУР к КА

Окончательный алгоритм (табл. 1.7).

Таблица 1.7

Шаги алгоритма	Команды
1. Выбросить АШ	
1.1. Выбросить Ш	`ВВЕРХ ВНИЗ`
1.2. Подойти к А	`ВЛЕВО`
1.3. Выбросить А	`ВВЕРХ ВНИЗ`
2. Подойти к началу слова	`ВЛЕВО ВЛЕВО ВЛЕВО ВЛЕВО ВВЕРХ`
3. Придвинуть МУР к КА	`ВПРАВО ВПРАВО`

Глава 2

Вася экономит свой труд

Общие рекомендации и дополнения

Программа для Кукарачи записывается в поле редактора программ и состоит из процедур. Каждая процедура имеет вид, показанный на рис. 2.1.

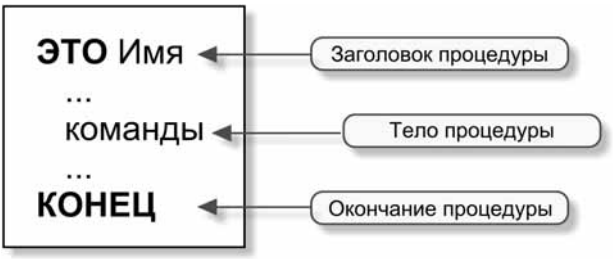

Рис. 2.1. Устройство процедуры

Слова **это** и **конец** являются ключевыми словами языка программирования. Ключевые слова (и команды СКИ) следует употреблять только по назначению. Нельзя, например, использовать их в качестве имён процедур.

Чтобы запустить процедуру, нужно написать её имя в редакторе команд и нажать экранную кнопку *GO* (рис. 2.2).

Запись программы рекомендуется сопровождать комментариями. Комментарий начинается с двух символов «//» и продолжается до конца строки. Комментарии не порождают никаких команд исполнителю, их можно располагать в любом месте программы, как внутри процедур, так и между ними.

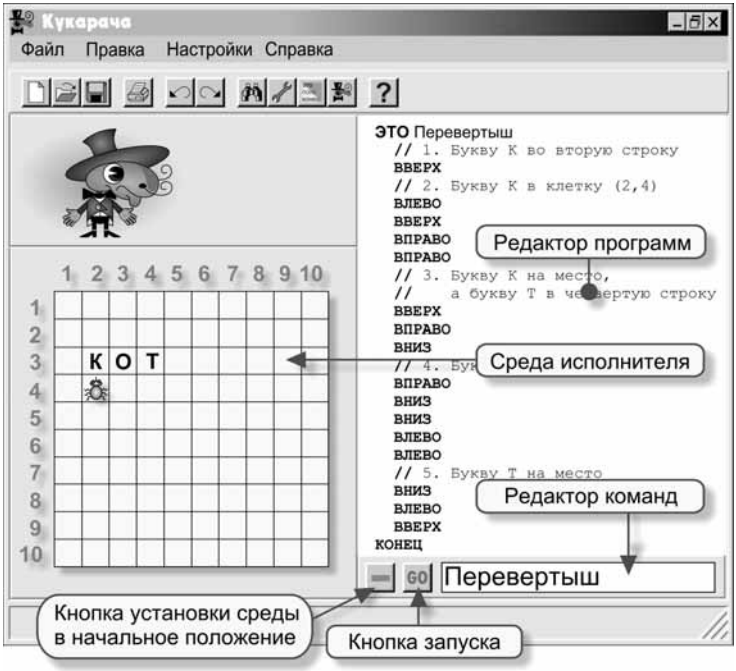

Рис. 2.2. Запуск процедуры

Стиль записи программ

Разделителем слов в языке программирования роботландских исполнителей служит пробел или конец строки.

Посмотрите на такую запись программы:

```
ЭТО Выход Поиск_прохода На_край_доски КОНЕЦ ЭТО Поиск_прохода ВПРАВО
ПОКА НЕ ПУСТО Шаг КОНЕЦ ЭТО Шаг ВЛЕВО ВНИЗ ВПРАВО КОНЕЦ ЭТО На_край_доски
ПОВТОРИ 8 ВПРАВО КОНЕЦ
```

С точки зрения интерпретатора — это правильный код, и он будет успешно выполнен.

Человек же читать плотно набитые строки не привык. Совсем другое дело, когда тот же самый код записывается так:

```
// Задача. Найти проход в стене и дойти
// до края поля (в десятом столбце).
ЭТО Выход
  Поиск_прохода
```

```
  На_край_доски
КОНЕЦ

ЭТО Поиск_прохода
  ВПРАВО
  ПОКА НЕ ПУСТО Шаг
КОНЕЦ

ЭТО Шаг
  // Вернуться в первый столбец.
  ВЛЕВО
  // Шагнуть на следующую строку.
  ВНИЗ
  // Толкнуть кубик.
  ВПРАВО
КОНЕЦ

ЭТО На_край_доски
  ПОВТОРИ 8 ВПРАВО
КОНЕЦ
```

Что изменилось? Последний вариант кода содержит пустоты, структурирован и снабжен комментариями.

Пустоты

Человек способен воспринимать информацию только маленькими порциями.

Поэтому обычные тексты в книгах не покрывают сплошь всю страницу. Пустоты помогают фиксировать внимание на небольших фрагментах, улучшая тем самым информационное пищеварение.

А ведь программы похожи на текст, они пишутся в текстовых редакторах, они выглядят как текст, они и есть текст!

В программном коде полезно отделять пустыми строками процедуры и в каждой строке записывать не более одной команды. Последнее правило можно нарушать лишь тогда, когда команды, записанные в одной строке, выполняют отдельный шаг алгоритма и есть причины, по которым не хочется выделять эту группу в отдельную процедуру (смотрите решения задач в конце этой главы).

Структурирование

Работая с отдельным фрагментом, необходимо понимать его место в общей ткани повествования. В силу этого книгу делят на части, части — на главы, главы — на параграфы, внутри параграфа записывают абзацы. Абзац содержит предложения, предложение — слова. Таким образом, наше восприятие настраивается на иерархическую структуру материала, связывая каждое слово с полным иерархическим деревом.

В программном коде выделить иерархическую структуру помогают отступы от левого края (структурная лесенка). Это общепринятое правило записи иерархических конструкций.

Иерархия, показанная на рис 2.3, представлена графически.

Рис. 2.3. Пример иерархического дерева

В текстовой записи эта иерархия выглядит так:

```
Личность
  Фамилия
  Возраст
  Адрес
    Город
    Улица
    Дом
```

Смещение вправо означает переход от родителя к потомку.

Тот же принцип лежит в основе структурной лесенки в записи программных кодов. Ключевые слова **ЭТО** и **КОНЕЦ** записываются с первой позиции. Команды, составляющие тело процедуры, размещаются со смещением (на две позиции) вправо.

Когда команда языка сама имеет иерархическую структуру, поступают подобным образом.

```
ЕСЛИ А
  ТО    ВПРАВО
  ИНАЧЕ ВЛЕВО
```

В приведённом примере команды, расположенные на ветвях условной конструкции, выровнены по вертикали, что дополнительно улучшает дизайн кода.

Ещё одна иллюстрация структурой лесенки на сложном коде:

```
ЕСЛИ A
  ТО
  {
    ВПРАВО
    ПОКА ПУСТО
    {
      ВНИЗ
      ВЛЕВО
    }
  }
  ИНАЧЕ Подход
```

Отход от структурной лесенки допустим только в компактных конструкциях, которые лучше воспринимаются в одной строке:

```
ПОВТОРИ 8 ВНИЗ
ЕСЛИ ПУСТО ТО ВПРАВО
ПОКА НЕ A ВНИЗ
```

Комментарии

Роль комментариев трудно переоценить. Обычно программирование сводится к записи шагов алгоритма на русском языке с последующим переводом их в конструкции языка программирования. При этом описания шагов алгоритма сохраняются в тексте программы в виде комментариев.

Дополнительные комментарии могут пояснять отдельные важные или «тонкие» места в программе.

Нужно сразу приучить себя к комментариям. Труд, потраченный на их ввод, окупается экономией времени при отладке и редактировании программы.

Комментарии не нужны лишь тогда, когда имена процедур содержательно их заменяют.

Имена процедур

Все ключевые слова роботландского языка записываются в верхнем регистре. Если имена процедур тоже записывать большими буквами, то визуально программа станет слишком однородной, следовательно, плохо читаемой. Глаз скользит по поверхности текста программы, ему не за что зацепиться.

Нижний регистр для имён процедур (за исключением первой заглавной буквы) позволяет разрушить однородность программного текста, внести в него «шершавость». Эта шершавость образуется в правильных местах программного полотна, именно там, где намечены структурные узлы.

```
ЭТО Вход
  Шаг
  ...
КОНЕЦ

ЭТО Шаг
  ...
КОНЕЦ
```

Когда код нужно оформлять в отдельную процедуру

Дробление программы на процедуры должно выполняться по правилам функционального разбиения алгоритма на части, методом нисходящей разработки (сверху вниз; от общего к частному).

Бывают и отступления от нисходящего метода, например, при использовании процедур, написанных ранее, но принцип функциональности должен работать всегда.

Программа, разбитая на смысловые процедуры, гораздо легче читается, отлаживается и сопровождается, чем программа, состоящая из процедур с механическим делением на части по объёму кода.

Посмотрите на этот код:

```
ЭТО Вход
  Шаг
  ЕСЛИ + ТО { ВВЕРХ ВНИЗ Вход ВЛЕВО }
  ИНАЧЕ ЕСЛИ НЕ ПУСТО ТО Вход
  ИНАЧЕ ВВЕРХ
КОНЕЦ

ЭТО Шаг
  ВНИЗ
  ВПРАВО
  ВВЕРХ
КОНЕЦ
```

Казалось бы, зачем выделять в отдельную процедуру код «**ВНИЗ ВПРАВО ВВЕРХ**», если он встречается в программе ровно один раз?

Дело в том, что команда `Шаг` лучше отражает суть действий, чем безликий набор «**ВНИЗ ВПРАВО ВВЕРХ**».

Иногда имеет смысл выделить в процедуру даже одну команду.

Визуальное отделение процедур

Позитивную неоднородность в визуальный ряд текста программы вносит отделение процедур друг от друга. Отделять процедуры можно пустой строкой или цепочкой символов, расположенных за началом комментария.

Вот реальный пример кода из работы школьника:

```
ЭТО ТЕКСТ
ШАГ
ЕСЛИ а ТО СЛОВО
ИНАЧЕ ЕСЛИ м ТО СЛОВО
ИНАЧЕ ОШИБКА
КОНЕЦ
ЭТО СЛОВО
ШАГ
ЕСЛИ а ТО СЛОВО
ИНАЧЕ ЕСЛИ м ТО СЛОВО
ИНАЧЕ ЕСЛИ . ТО ОТВЕТ1
ИНАЧЕ ЕСЛИ _ ТО СЛОВО1
ИНАЧЕ ЕСЛИ , ТО СЛОВО1
ИНАЧЕ ЕСЛИ ПУСТО ТО ОТВЕТ
ИНАЧЕ ОШИБКА
КОНЕЦ
ЭТО СЛОВО1
ШАГ
ЕСЛИ а ТО СЛОВО
ИНАЧЕ ЕСЛИ м ТО СЛОВО
ИНАЧЕ ОШИБКА
КОНЕЦ
```

У вас зарябило в глазах?

Уверен, что если попросить вас вникнуть в этот код, то для начала вы раздвинете процедуры. А ещё, конечно, сделаете структурные отступы, измените регистр в именах процедур.

Получится так:

```
ЭТО Текст
  Шаг
  ЕСЛИ       а ТО Слово
  ИНАЧЕ ЕСЛИ м ТО Слово
  ИНАЧЕ          Ошибка
КОНЕЦ

ЭТО Слово
  Шаг
  ЕСЛИ       а     ТО Слово
  ИНАЧЕ ЕСЛИ м     ТО Слово
  ИНАЧЕ ЕСЛИ .     ТО Ответ1
  ИНАЧЕ ЕСЛИ _     ТО Слово1
  ИНАЧЕ ЕСЛИ ,     ТО Слово1
  ИНАЧЕ ЕСЛИ ПУСТО ТО Ответ
  ИНАЧЕ               Ошибка
КОНЕЦ

ЭТО Слово1
  Шаг
  ЕСЛИ       а ТО Слово
  ИНАЧЕ ЕСЛИ м ТО Слово
  ИНАЧЕ          Ошибка
КОНЕЦ
```

Это уже читается, правда?

Ответы на вопросы и решение задач

2.1. Ток — это кот задом наперёд

1. Изменится ли состояние среды Кукарачи, если Петя и Вася захотят переставлять местами буквы Н и С в слове СОН? Изменится ли программа? Почему? Заставьте Кукарачу сделать требуемую перестановку.

   Ответ. Состояние среды определяется положением в ней исполнителя, кубиков и символами, которые расположены на кубиках. Символы стали другими, значит, состояние среды изменилось. Программу менять не надо.

В ней задана перестановка первого и последнего символа трёхсимвольной записи. Какие это конкретно символы, программа не проверяет.

2. Потребуется ли новая программа, если ребята захотят переставить первую и последнюю буквы в слове СОРТ? Почему?

   Ответ. А вот для этой задачи потребуется новая программа, т. к. старая работает только с трёхсимвольной записью.

2.2. У компьютера есть память, и это хорошо

1. Что обозначают слова **это** и **конец** в программах Кукарачи?

   Ответ. Эти ключевые слова языка программирования обозначают начало и конец процедуры.

2. Найдите ошибки в программе:

```
Это Очень хорошая команда
  СТОЯТЬ  (Кукарача стоит на месте)
Конец.
```

   Ответ. Ошибки в ключевых словах «**Это**» и «**Конец.**». нужно: **ЭТО** и **КОНЕЦ** (без точки). Пробелы в записи имени процедуры. Неверная запись комментария (должен начинаться с символов «//»).

2.5. Задачи

1. Что кричит кукушка (рис. 2.4)? (КУКУ)

Рис. 2.4

Решение

```
ЭТО Куку
  // Убрать ШКА
  ВВЕРХ ВНИЗ // Убрать А
  ВЛЕВО      // Подойти к К
  ВВЕРХ ВНИЗ // Убрать К
  ВЛЕВО      // Подойти к Ш
  ВВЕРХ      // Убрать Ш
КОНЕЦ
```

2. Получите из этого неизвестного зверя домашнее животное (рис. 2.5).

Рис. 2.5

Решение

```
ЭТО Собака
  // 1. Поставить О
    // 1.1. Извлечь О
    ВНИЗ
    // 1.2. Подойти к О
    ВВЕРХ
    ВПРАВО ВПРАВО ВПРАВО
    ВНИЗ ВНИЗ
    ВЛЕВО ВЛЕВО
    // 1.3. Передвинуть О
    ВЛЕВО ВЛЕВО
    // 1.4. Поставить О на место
    ВНИЗ ВЛЕВО ВВЕРХ
  // 2. Поставить А
    // 2.1. Подойти к А
    ВЛЕВО ВЛЕВО
    ВВЕРХ ВВЕРХ
    ВПРАВО
    // 2.2. Передвинуть А
    ВПРАВО ВПРАВО
    // 2.3. Поставить А на место
    ВВЕРХ ВПРАВО ВНИЗ
КОНЕЦ
```

3. Как получить из пяти ног одно пятно (рис. 2.6)?

Рис. 2.6

Решение

```
ЭТО Пятно
  // 1. Убрать Ь
  ВВЕРХ ВВЕРХ ВНИЗ ВНИЗ
  // 2. Подойти к Г
  ВПРАВО ВПРАВО ВПРАВО ВПРАВО
  // 3. Убрать Г
  ВВЕРХ ВВЕРХ ВНИЗ
  // 4. Уплотнить
  ВЛЕВО ВЛЕВО
КОНЕЦ
```

4. Кому поёт Кукарача (рис. 2.7)? (ЮРИЮ)

Рис. 2.7

Решение

```
ЭТО Юра
  // 1. Убрать ПО
    // 1.1.  Толкнуть букву (П) и подойти к следующей
    ВВЕРХ ВНИЗ ВПРАВО
    // 1.2. Толкнуть_букву (О) и подойти к следующей
    ВВЕРХ ВНИЗ ВПРАВО
```

```
  // 2. Убрать А
  ВНИЗ ВПРАВО ВВЕРХ ВВЕРХ
  // 3. Подойти к Р
  ВНИЗ ВНИЗ ВПРАВО
  // 4. Переместить РИЮ
    // 4.1.Толкнуть букву (Р) и подойти к следующей
    ВВЕРХ ВНИЗ ВПРАВО
    // 4.2.Толкнуть букву (И) и подойти к следующей
    ВВЕРХ ВНИЗ ВПРАВО
    // 4.3.Толкнуть букву (Ю) и подойти к следующей
    ВВЕРХ ВНИЗ ВПРАВО
  // 5. Уплотнить
  ВВЕРХ ВВЕРХ ВЛЕВО
КОНЕЦ
```

5. Исправьте пример, переместив один из кубиков (рис. 2.8).

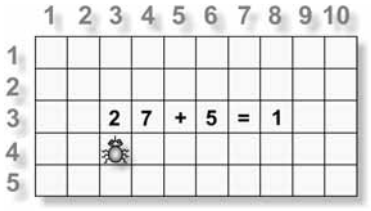

Рис. 2.8

Решение

```
ЭТО Исправление
  // 2 - в клетку (2,3)
  //////////
  ВВЕРХ   //
  //////////
  // 2 - в клетку (2,9)
  //////////
  ВЛЕВО   //
  ВВЕРХ   //
```

```
  ВПРАВО  //
  ВПРАВО  //
  ВПРАВО  //
  ВПРАВО  //
  ВПРАВО  //
  ВПРАВО  //
  //////////
  // 2 - в клетку (3,9)
  //////////
  ВВЕРХ   //
  ВПРАВО  //
  ВНИЗ    //
  //////////
КОНЕЦ
```

Глава 3

Новые команды и их повторение

Общие рекомендации и дополнения

Процедурное программирование

Программирование с помощью процедур (процедурное программирование) существенно облегчает труд.

Во-первых, написав процедуру, можно использовать её, вызывая неоднократно из разных мест программы.

Во-вторых, разбивая программу на маленькие кирпичики, программист получает возможность описывать короткие простые действия.

В-третьих, разделение на процедуры позволяет работать над большим проектом коллективу программистов.

Команда повторения

Команда **ПОВТОРИ** используется, когда нужно повторить несколько раз одни и те же действия. Эта команда имеет вид:

ПОВТОРИ число команда

После ключевого слова **ПОВТОРИ** записывается число повторений и команда (только одна!), которая будет повторена указанное число раз.

ПОВТОРИ 2 ВЛЕВО или **ВЛЕВО ВЛЕВО?**

С точки зрения правописания программ, обе формы записи имеют право на существование.

Команда **ПОВТОРИ** вводится в язык для сокращения записи. Никакой другой смысловой нагрузки, в отличие, например, от команды **ПОКА** эта конструкция не несёт.

ПОВТОРИ 2 ВЛЕВО всегда эквивалентно **ВЛЕВО ВЛЕВО**.

ПОКА ПУСТО ВЛЕВО иногда эквивалента **СТОЯТЬ**, иногда **ВЛЕВО**, иногда **ВЛЕВО ВЛЕВО**.

Записи «**ПОВТОРИ** 2 **ВЛЕВО**» и «**ВЛЕВО ВЛЕВО**» имеют примерно одинаковую длину, но всё же последняя запись немного короче. Если учесть ещё, что транслятор при работе цикла вынужден дополнительно проверять условие его окончания, то последняя форма записи (без **ПОВТОРИ**) оказывается более эффективной как по длине кода, так и по времени его выполнения.

Интерпретатор программ

Интерпретатор программ — это одна из процедур программы, моделирующей исполнителя Кукарачу на компьютере. Интерпретатор выполняет программы, записанные на языке программирования. Сначала интерпретатор проверяет программу на соответствие правилам языка. Когда обнаруживается синтаксическая ошибка, интерпретатор показывает окно с сообщением и отмечает в программном поле подозрительное место.

Часто место обнаружения ошибки не совпадает с местом её расположения. Отыскать ошибку помогает «игра за интерпретатор»: формальный анализ кода программы.

Интерпретатор может помочь отыскать и логическую ошибку (ошибку в алгоритме), если запустить его в пошаговом режиме. При пошаговом режиме интерпретатор отмечает выполняемые команды и работает с остановкой после каждого шага.

Классификация ошибок программирования

Обсуждая правила записи алгоритма на языке программирования, братья подробно рассмотрели синтаксические ошибки. Они возникают при нарушении формальных правил использования языковых конструкций. Однако программист может совершить и более сложные промахи, которые приводят к ошибкам в синтаксически правильных программах.

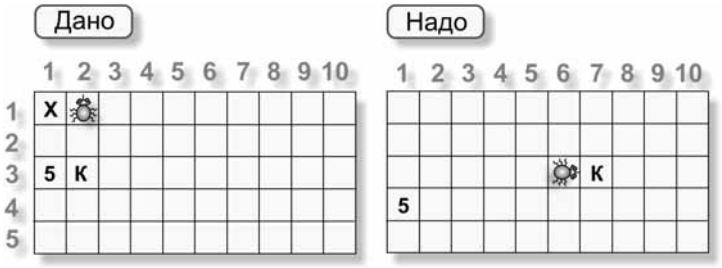

Рис. 3.1. Пример, иллюстрирующий условие задачи

Построим классификацию ошибок программирования, опираясь на этапы разработки программного продукта. В качестве иллюстраций будем опираться на решение следующей задачи.

Задача. Исполнитель расположен в клетке (1,2). На поле поставлено три кубика. Первый кубик в клетке (1,1), второй с цифрой где-то в первом столбце, ниже первой строки, а третий — сразу справа от второго. Требуется удалить первый кубик с поля, а третий переместить вправо по горизонтали на число клеток, задаваемое цифрой на втором кубике (рис. 3.1).

Построение алгоритма

Ошибки, возникающие при описании алгоритма, назовём *алгоритмическими* ошибками.

Пример неправильного алгоритма. Перемещаем исполнителя влево на одну клетку (удаляем первый кубик), затем вниз на две клетки (проверяем второй кубик), наконец, перемещаем третий кубик на нужное число клеток вправо.

Ошибка в рассуждении: описанная идея будет работать только для состояния среды, изображённого в условии задачи в качестве примера.

Пример правильного алгоритма. Перемещаем исполнителя влево, затем вниз, пока он не толкнёт кубик с цифрой, наконец, перемещаем третий кубик вправо на число клеток, равное цифре на втором кубике.

Построение программы

Ошибки, возникающие при построении программы, назовём *ошибками кодирования.*

В свою очередь ошибки кодирования разделим на *семантические* (смысловые) и *синтаксические* (грамматические).

Пример семантической ошибки

```
ЭТО Вход
  ВЛЕВО
  ПОКА ПУСТО ВНИЗ
  ЕСЛИ       1 ТО ВПРАВО
  ИНАЧЕ ЕСЛИ 2 ТО ПОВТОРИ 2 ВПРАВО
  ИНАЧЕ ЕСЛИ 3 ТО ПОВТОРИ 3 ВПРАВО
  ИНАЧЕ ЕСЛИ 4 ТО ПОВТОРИ 4 ВПРАВО
  ИНАЧЕ ЕСЛИ 5 ТО ПОВТОРИ 5 ВПРАВО
  ИНАЧЕ ЕСЛИ 6 ТО ПОВТОРИ 6 ВПРАВО
  ИНАЧЕ ЕСЛИ 7 ТО ПОВТОРИ 7 ВПРАВО
  ИНАЧЕ ЕСЛИ 8 ТО ПОВТОРИ 8 ВПРАВО
  ИНАЧЕ ЕСЛИ 9 ТО ПОВТОРИ 9 ВПРАВО
КОНЕЦ
```

Программист не учёл, что при работе цикла сначала проверяется условие, а затем, если условие истинно, выполняется команда, образующая его тело.

Пример синтаксической ошибки

```
ЭТО Вход
  ВЛЕВО
  ВНИС // Ошибка здесь
  ПОКА ПУСТО ВНИЗ
  ВПРАВО
  ЕСЛИ        1 ТО ВПРАВО
  ИНАЧЕ ЕСЛИ 2 ТО ПОВТОРИ 2 ВПРАВО
  ИНАЧЕ ЕСЛИ 3 ТО ПОВТОРИ 3 ВПРАВО
  ИНАЧЕ ЕСЛИ 4 ТО ПОВТОРИ 4 ВПРАВО
  ИНАЧЕ ЕСЛИ 5 ТО ПОВТОРИ 5 ВПРАВО
  ИНАЧЕ ЕСЛИ 6 ТО ПОВТОРИ 6 ВПРАВО
  ИНАЧЕ ЕСЛИ 7 ТО ПОВТОРИ 7 ВПРАВО
  ИНАЧЕ ЕСЛИ 8 ТО ПОВТОРИ 8 ВПРАВО
  ИНАЧЕ ЕСЛИ 9 ТО ПОВТОРИ 9 ВПРАВО
КОНЕЦ
```

На схеме рис. 3.2 представлена классификация ошибок программирования.

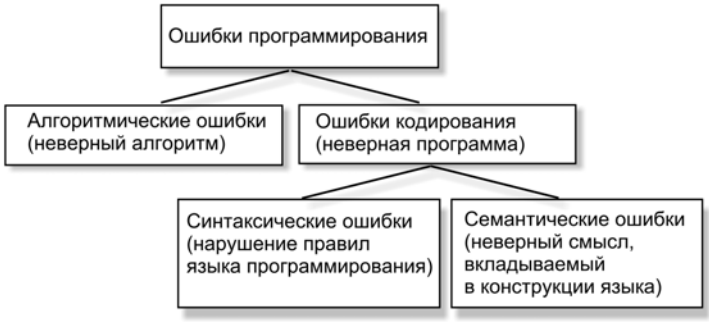

Рис. 3.2. Классификация ошибок программирования

В практике программирования чаще рассматривают другую классификацию ошибок, основанную не на времени совершения ошибки, а на времени её обнаружения.

Синтаксическая ошибка — обнаруживается на этапе трансляции программы.

Например, несуществующая команда ВНИС в коде программы.

Семантическая ошибка — приводит к отказу работы программы на этапе выполнения. Например, команда **ВЛЕВО**, выполняемая в тот момент, когда исполнитель расположен в первом столбце.

Алгоритмическая ошибка — обнаруживается после завершения работы программы по неверному конечному состоянию среды. Например, кубик, который должен быть удалён, остался на поле.

Такая классификация способна одну и ту же погрешность программиста относить к разным типам ошибок, в зависимости от времени проявления допущенной погрешности.

Пример (предложил Я. Н. Зайдельман)

Пусть в программе на Си должно быть написано деление `1/x1`. Программист ошибся и написал `1/x`. Эта ошибка может проявиться следующим образом:

- синтаксически — в виде сообщения компилятора — если в программе не описана переменная с именем `x` или она имеет неподходящий тип;
- семантически — в виде сообщения исполняющей системы — если переменная `x` есть, но в момент деления она равна нулю;
- алгоритмически — в виде неверного результата вычислений, если переменная `x` существует и в момент деления не равна нулю;
- вообще никак — если в момент деления `x` равно `x1`.

Ответы на вопросы и решение задач

3.1. Кукарача говорит «Ах!»

1. Выполнит ли Кукарача прогулку, показанную на рис. 3.3, по команде `Ух`, если в программном поле записано:

```
ЭТО Ух
  Ох  // Это вызов процедуры Ох
  Эх  // Это вызов процедуры Эх
КОНЕЦ
// Это описание процедуры Ох
ЭТО Ох
  ВПРАВО
  ВНИЗ
КОНЕЦ
```

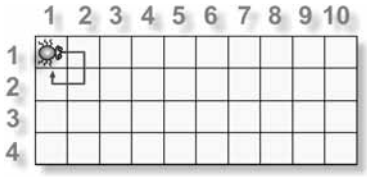

Рис. 3.3

```
// Это описание процедуры Эх
ЭТО Эх
  ВЛЕВО
  ВВЕРХ
КОНЕЦ
```

Ответ. Да.

2. Что произойдёт, если вызовы процедур `Ох` и `Эх` поменять местами в процедуре `Ух`?

   Ответ. Кукарача покажет сообщение «Не могу».

3. Что произойдёт, если описание процедур `Ох` и `Эх` поменять местами в программном поле?

   Ответ. Команда `Ух` будет работать правильно: порядок следования описаний процедур не имеет значения.

3.2. Процедурное программирование

1. В чём заключается преимущество процедурного программирования?

   Ответ. Одну и ту же процедуру можно использовать многократно (экономия кода). Программа разбивается на небольшие части, каждую из которых легко написать и отладить (упрощение программирования).

2. Дайте обоснование выбранным именам процедур в задаче «НАРЫЧАЛО».

   Ответ. Каждое имя соответствует той работе, которую должен выполнить исполнитель.

3.3. Команда повторения

1. Посмотрите на следующие рисунки (рис. 3.4).

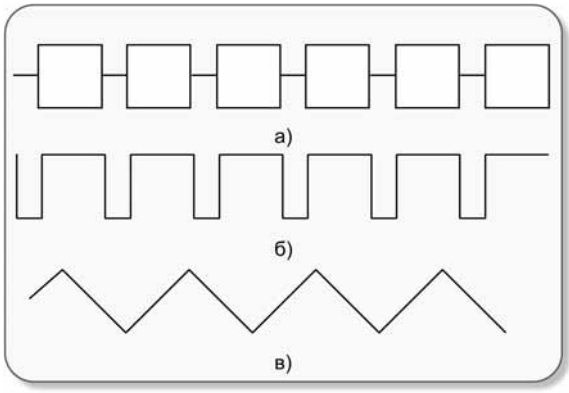

Рис. 3.4

1.1. Найдите в каждом рисунке повторяющиеся части.

Ответ. Возможно несколько вариантов ответа. Один из них показан на рис. 3.5.

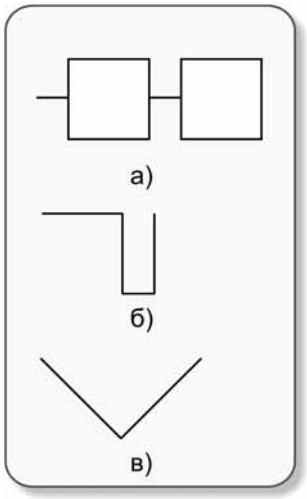

Рис. 3.5

1.2. Можно ли выделить такую часть, повторением которой образован весь рисунок?

Ответ. Показан на рис. 3.6.

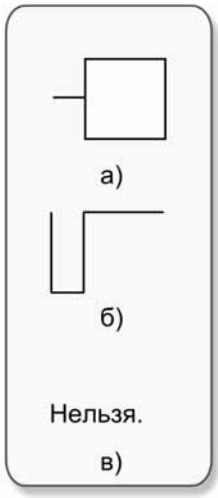

Рис. 3.6

2. Какие из приведённых ниже команд повторения правильные, а какие — нет и почему?

   2.1. **ПОВТОРИ** 7 **СТОЯТЬ**

   Ответ. Формально команда правильная, хотя сомнительно, что полезная.

   2.2. **Повтори** 2 **ВПРАВО**

   Ответ. Первое ключевое слово написано с синтаксической ошибкой, нужно писать **ПОВТОРИ**.

   2.3. **ПОВТОРИ** 5-2 **ВНИЗ**

   Ответ. Число повторений должно быть числом и не может быть арифметическим выражением.

   2.4. **ПОВТОРИ** 100 **ВНИЗ ВПРАВО**

   Ответ. В этом коде задано: повторить 100 раз команду **ВНИЗ**, а затем один раз выполнить команду **ВПРАВО**. Если программист считает, что 100 раз будут выполняться две команды «**ВНИЗ ВПРАВО**», то он ошибается.

   2.5. **ПОВТОРИ** 8 Колесо

   Ответ. Если программа содержит описание процедуры Колесо, то команда записана без ошибок.

   2.6. **ПОВТОРИ** 7 **ПОВТОРИ** 2 **ВПРАВО**

   Ответ. Формально запись совершенно правильная: после числа повторений следует одна команда (команда повторения), хотя конструкция выглядит странно. Эта конструкция эквивалента одной команде повторения: **ПОВТОРИ** 14 **ВПРАВО**

   2.7. **ПОВТОРИ** 1 **ВПРАВО**

   Ответ. Запись правильная, но неразумная, лучше просто записать команду **ВПРАВО**.

3.5. Задачи

1. Помогите Кукараче взобраться по лесенке (рис. 3.7).

Решение

```
ЭТО Лесенка
  ПОВТОРИ 7 Ступень
КОНЕЦ

ЭТО Ступень
  ВВЕРХ
  ВПРАВО
КОНЕЦ
```

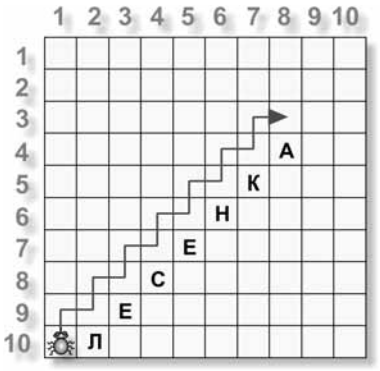

Рис. 3.7

2. Составьте программу прогулки Кукарачи по заданному маршруту так, чтобы он прошёл его пять раз (рис. 3.8).

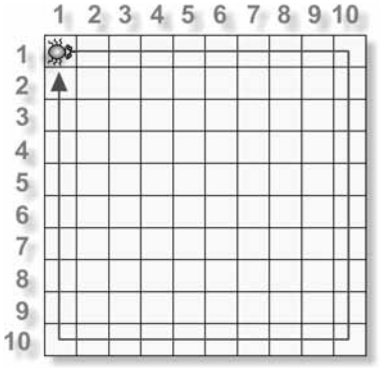

Рис. 3.8

Решение

```
ЭТО Прогулка
  ПОВТОРИ 5 Маршрут
КОНЕЦ

ЭТО Маршрут
  ПОВТОРИ 9 ВПРАВО
  ПОВТОРИ 9 ВНИЗ
  ПОВТОРИ 9 ВЛЕВО
  ПОВТОРИ 9 ВВЕРХ
КОНЕЦ
```

3. Контроль поля. Заставьте исполнителя обойти все клетки поля так, чтобы в каждой из них он побывал только один раз.

Решение

```
ЭТО Контроль
  ПОВТОРИ 4 Две_строки_и_вниз
  Две_строки
КОНЕЦ

ЭТО Две_строки_и_вниз
  Две_строки
  ВНИЗ
КОНЕЦ

ЭТО Две_строки
  ПОВТОРИ 9 ВПРАВО
  ВНИЗ
  ПОВТОРИ 9 ВЛЕВО
КОНЕЦ
```

4. Собрать слово ПАРОХОД (рис. 3.9).

Рис. 3.9

Решение

```
ЭТО Пароход
  Поставить_ПАР
  Поставить_ХОД
КОНЕЦ

ЭТО Поставить_ПАР
  ПОВТОРИ 3 ВНИЗ
```

```
  ВПРАВО
КОНЕЦ

ЭТО Поставить_ХОД
  Подойти_к_ХОД
  Сместить_ХОД_вправо
  Сместить_ХОД_вверх
КОНЕЦ

ЭТО Подойти_к_ХОД
  ВЛЕВО ВНИЗ
КОНЕЦ

ЭТО Сместить_ХОД_вправо
  ПОВТОРИ 5 ВПРАВО
КОНЕЦ

ЭТО Сместить_ХОД_вверх
  // Подойти к Х
  ВНИЗ ВПРАВО
  // Смещать по буквам
  ПОВТОРИ 3 Букву_вверх
КОНЕЦ

ЭТО Букву_вверх
  ВВЕРХ ВНИЗ
  ВПРАВО
КОНЕЦ
```

5. Получить на поле слово КОЛЕСО и закончить работу в первом столбце (рис. 3.10).

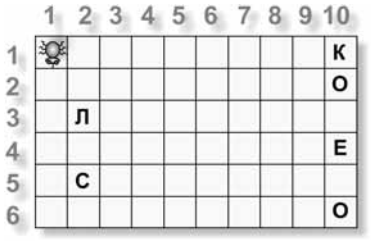

Рис. 3.10

Решение

Нарисуем траекторию движения исполнителя (рис. 3.11).

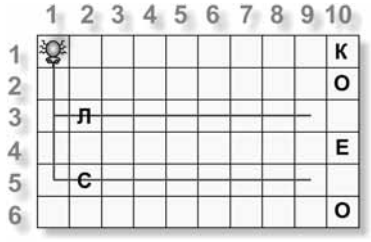

Рис. 3.11

Заметим, что она состоит из двух одинаковых частей, поэтому программа будет записана следующим образом:

```
ЭТО Колесо
  ПОВТОРИ 2 Установка_буквы
КОНЕЦ


ЭТО Установка_буквы
  ПОВТОРИ 2 ВНИЗ
  ПОВТОРИ 8 ВПРАВО
  ПОВТОРИ 8 ВЛЕВО
КОНЕЦ
```

6. Собрать слово МАШИНИСТ на 10 строке и закончить работу в клетке (1,10) (рис. 3.12).

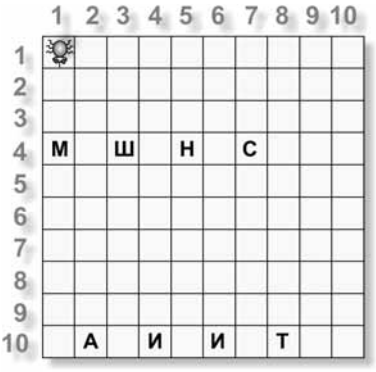

Рис. 3.12

Решение

```
ЭТО Машинист
  // 1. Подойти к М
  ВНИЗ ВНИЗ
  // 2. Поставить буквы
  ПОВТОРИ 4 Установка_буквы
  // 3. Сместиться в клетку (1,10)
  ВВЕРХ ВВЕРХ ВПРАВО
КОНЕЦ

ЭТО Установка_буквы
  ПОВТОРИ 6 ВНИЗ
  ПОВТОРИ 6 ВВЕРХ
  ВПРАВО ВПРАВО
КОНЕЦ
```

7. Помогите Кукараче очистить поле от мусора (кубики X), не сдвигая остальных кубиков (рис. 3.13).

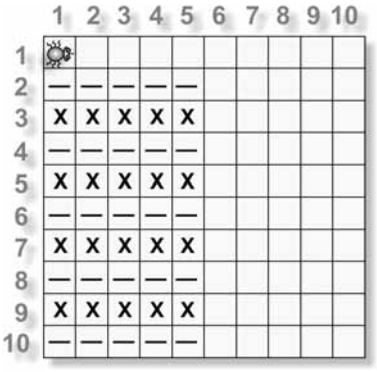

Рис. 3.13

Решение

```
ЭТО Уборка
  ПОВТОРИ 4 Уборка_строки
КОНЕЦ

ЭТО Уборка_строки
```

```
  ПОВТОРИ 5 ВПРАВО
  ВНИЗ ВНИЗ
  ПОВТОРИ 5 ВЛЕВО
КОНЕЦ
```

8. Помогите Кукараче совершить прогулку в клетку (1,10), не сдвигая кубиков (рис. 3.14).

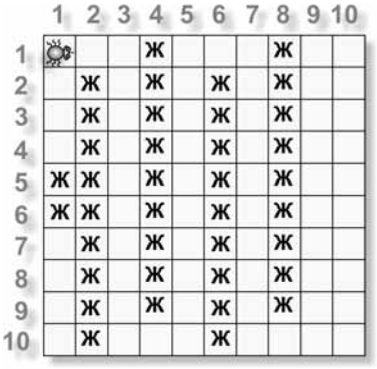

Рис. 3.14

Решение

```
ЭТО Прогулка
  ПОВТОРИ 2 Виток
  ВПРАВО
КОНЕЦ

ЭТО Виток
  ВПРАВО ВПРАВО
  ПОВТОРИ 9 ВНИЗ
  ВПРАВО ВПРАВО
  ПОВТОРИ 9 ВВЕРХ
КОНЕЦ
```

Глава 4

Кукарача на распутье

Общие рекомендации и дополнения

Исполнители, лишённые «органов чувств» (датчиков), не имеют со средой обратной связи, а значит, способны работать только тогда, когда состояние среды известно заранее.

Пусть Кукараче нужно сдвинуть 3 кубика из второй строки в первую и встать за концом полученного ряда (рис. 4.1).

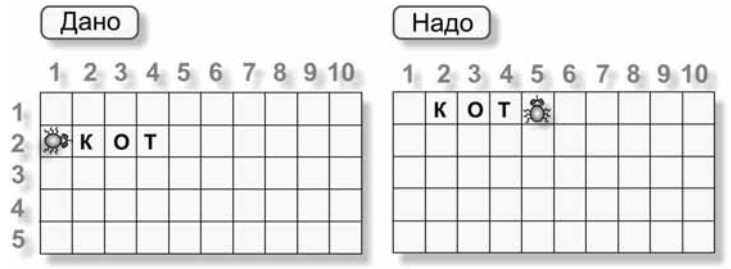

Рис. 4.1. Переместить кубики в первую строку и встать за концом ряда

Программа будет содержать последовательность перемещающих команд, без всякого анализа среды:

```
ЭТО Вход
  ПОВТОРИ 4 Шаг
  ВВЕРХ
КОНЕЦ
ЭТО Шаг
  ВНИЗ ВПРАВО ВВЕРХ
КОНЕЦ
```

Предположим теперь, что число кубиков во второй строке заранее не известно. Исполнитель, лишённый возможности отличить пустую клетку от клетки с кубиком, не способен решить такую задачу.

Но Кукарача умеет анализировать клетку, переместившись в неё: у него есть «датчик», который сигнализирует о кубике и определяет надпись на нём. Такое устройство исполнителя позволяет установить в программе обратную связь со средой:

```
ЭТО Вход
  Шаг
  ПОКА ПУСТО Шаг
  ВВЕРХ
КОНЕЦ
ЭТО Шаг
  ВНИЗ ВПРАВО ВВЕРХ
КОНЕЦ
```

В приведённой ранее программе анализ состояния среды выполняется в команде цикла **ПОКА** (материал следующей главы), в этом уроке рассматривается команда ветвления **ЕСЛИ**, работа которой также основана на проверке показаний датчика исполнителя.

После ключевого слова **ПОКА** в команде цикла и после ключевого слова **ЕСЛИ** в команде ветвления записывается условие, выполнение (или не выполнение) которого определяет работу команды цикла или команды ветвления. Когда условие выполняется, говорят, что оно принимает значение *истина*, когда не выполняется, говорят, что оно принимает значение *ложь*.

Ещё раз отметим, что если состояние среды полностью известно до работы исполнителя, никакие проверки в программе не нужны.

В языке программирования команда ветвления имеет следующий вид (или, как говорят программисты, формат):

```
ЕСЛИ условие
  ТО    команда1
  ИНАЧЕ команда2
```

Условие — это символ, ключевое слово **ПУСТО** (исполнитель сместился в клетку без кубиков) или ключевое слово **ЦИФРА** (исполнитель толкнул один из символов 0, 1, 2, 3, 4, 5, 6, 7, 8, 9).

Условие может начинаться с ключевого слова **НЕ**, и тогда смысл его меняется на обратный.

Набор возможных условий перечислен в табл. 4.1.

Таблица 4.1

Запись условия	Результат проверки
символ	*Истина*, если Кукарача толкнул кубик с указанным в условии символом, *ложь* в противном случае
НЕ символ	*Ложь*, если Кукарача толкнул кубик с указанным в условии символом, *истина* в противном случае
ПУСТО	*Истина*, если клетка, в которую сместился Кукарача, была пуста, *ложь* в противном случае
НЕ ПУСТО	*Ложь*, если клетка, в которую сместился Кукарача, была пуста, *истина* в противном случае
ЦИФРА	*Истина*, если Кукарача толкнул кубик с цифрой (один из символов 0, 1, 2, 3, 4, 5, 6, 7, 8, 9), *ложь* в противном случае
НЕ ЦИФРА	*Ложь*, если Кукарача толкнул кубик с цифрой (один из символов 0, 1, 2, 3, 4, 5, 6, 7, 8, 9), *истина* в противном случае

Записи «команда1» и «команда2» в определении условной команды — любые команды языка.

После слов **ТО** и **ИНАЧЕ** можно записывать только в одну команду.

Запись «**ИНАЧЕ** команда2» может отсутствовать, тогда команда принимает вид:

```
ЕСЛИ условие
  ТО  команда1
```

Эту конструкцию нагляднее записывать в одну строку:

```
ЕСЛИ условие ТО команда1
```

Команда ветвления работает следующим образом: интерпретатор программ запрашивает состояние среды у исполнителя и проверяет условие. Если условие истинно, интерпретатор выполняет команду, записанную после ключевого слова **ТО**, в противном случае — после ключевого слова **ИНАЧЕ**.

Отдельные рекомендации, пояснения и примеры

Следующие примеры на ветвление взяты из повседневной жизни.

1. Планирование воскресного дня: если будет хорошая погода, пойду гулять, если нет, буду читать книгу.
2. Телефонное правило: если, набрав номер, вы слышите короткие гудки, то положите трубку, иначе ждите соединения.

3. Как отличить ужа от ядовитой змеи: если на голове рептилии имеется два жёлтых пятнышка, то это уж.

Общее, что важно увидеть во всех примерах, — ситуация выбора на основе информации, получаемой в момент выполнения ветвящейся инструкции. Важно подчеркнуть проблему, возникающую перед составителем инструкции: он заранее не знает, с каким вариантом на практике встретится исполнитель. Значит, он должен предусмотреть все случаи, отсюда и возникают ветвления.

Для проверки программ с ветвлениями недостаточно одного запуска. Чтобы проверить все ветви алгоритма, необходимо запускать программу со всеми возможными комбинациями значений всех условий в программе.

При записи условия в команде ветвления разрешается использовать ключевое слово **НЕ**, обозначающее логическую операцию отрицания. Можно привести множество примеров, где операция отрицания используется в повседневной жизни:

1. Если завтра НЕ будет дождя, пойду в поход.
2. Если НЕ зададут уроков, буду кататься на велосипеде.
3. Если дома НЕТ хлеба, схожу в магазин.
4. Если на светофоре НЕ горит зелёный свет, надо подождать.

Переключатели

Не надо записывать переключатель подобным образом:

```
ЕСЛИ +          ТО Уборка ИНАЧЕ
ЕСЛИ НЕ ПУСТО ТО Слово  ИНАЧЕ ВВЕРХ
```

Кажется, что это не переключатель, а две условных команды. Надо записывать так:

```
ЕСЛИ        +         ТО Уборка
ИНАЧЕ ЕСЛИ НЕ ПУСТО ТО Слово
ИНАЧЕ                  ВВЕРХ
```

Такая запись переключателя:

```
ЭТО Слово
 Шаг
 ЕСЛИ а
  ТО Слово
  ИНАЧЕ ЕСЛИ м
```

```
        ТО Слово
        ИНАЧЕ ЕСЛИ .
               ТО Точка
               ИНАЧЕ ЕСЛИ _
                      ТО Текст
                      ИНАЧЕ ЕСЛИ ,
                             ТО Текст
                             ИНАЧЕ Ошибка
КОНЕЦ
```

тоже не является хорошей.

Для переключения во многих языках программирования используется специальная команда. В Си, например, эта конструкция имеет вид:

```
switch (выражение)
{
  case значение1:
       команды
       break;

  case значение2:
       команды
       break;
  ...

  case значениеN:
       команды
       break;
  default:
       команды
}
```

В языке Кукарачи такой команды нет, но она легко моделируется при помощи выражения «**ИНАЧЕ ЕСЛИ**».

В каком-то смысле можно рассматривать это построение как особую языковую конструкцию и записывать подобно **switch**.

В рекомендуемой далее записи гораздо легче увидеть переключение при помощи вертикального выравнивания (рис. 4.2).

Сравните фрагменты кода в столбцах табл. 4.2.

Рис. 4.2. Форма записи переключателя

Таблица 4.2

Запись 1	**Запись 2**

ЭТО Цифру_в_нужную_строку
  ЕСЛИ 3 **ТО** Цифра_3
  ИНАЧЕ ЕСЛИ 4 **ТО** Цифра_4
  ИНАЧЕ ЕСЛИ 5 **ТО** Цифра_5
  ИНАЧЕ ЕСЛИ 6 **ТО** Цифра_6
  ИНАЧЕ ЕСЛИ 7 **ТО** Цифра_7
  ИНАЧЕ ЕСЛИ 8 **ТО** Цифра_8
  ИНАЧЕ ЕСЛИ 9 **ТО** Цифра_9
КОНЕЦ
	ЭТО Цифру_в_нужную_строку
	  ЕСЛИ 3 **ТО** Цифра_3
	  ЕСЛИ 4 **ТО** Цифра_4
	  ЕСЛИ 5 **ТО** Цифра_5
	  ЕСЛИ 6 **ТО** Цифра_6
	  ЕСЛИ 7 **ТО** Цифра_7
	  ЕСЛИ 8 **ТО** Цифра_8
	  ЕСЛИ 9 **ТО** Цифра_9
	КОНЕЦ

Запись 1 — это переключатель, запись 2 — нет, хотя многие начинающие программисты записывают переключение по образцу записи 2. Ведь эта запись проще!

Дело в том, что вторая запись может привести к трудноуловимой ошибке.

Допустим, надо увеличить `x` на 1, если `x` равен 3 и на 2, если он равен 4.

Напишем так:

```
ЕСЛИ x = 3 ТО x:=x+1                (запись 2)
ЕСЛИ x = 4 ТО x:=x+2
```

Предположим, что перед вычислениями x равнялся 3.

Чему будет равен x после работы этих двух команд? Ведь не четырём, как хотелось, правда? Он станет равным 6.

А если записать так:

```
ЕСЛИ        x = 3 ТО x:=x+1          (запись 1)
ИНАЧЕ ЕСЛИ x = 4 ТО x:=x+2
```

то все будет работать правильно.

Итак, запись 2 на роль переключателя не годится: она может привести к ошибке. Кроме того, запись 2, несмотря на свою простоту, приводит к неэффективным программам. В таких записях все проверки работают всегда, независимо от значения переменной. В записи 1 количество выполненных проверок зависит от значения переменной и может равняться одному.

Запись 1 не эквивалентна записи 2 даже для Кукарачи.

Рассмотрим два варианта кода:

```
// Вариант 1
ЭТО Код2
  ВНИЗ
  ЕСЛИ        1 ТО ВПРАВО
  ИНАЧЕ ЕСЛИ 2 ТО ПОВТОРИ 2 ВПРАВО
  ВВЕРХ
КОНЕЦ

// Вариант 2
ЭТО Код1
  ВНИЗ
  ЕСЛИ 1 ТО ВПРАВО
  ЕСЛИ 2 ТО ПОВТОРИ 2 ВПРАВО
  ВВЕРХ
КОНЕЦ
```

На рис. 4.3 и 4.4 показаны результаты выполнения этих кодов. Они совершенно разные.

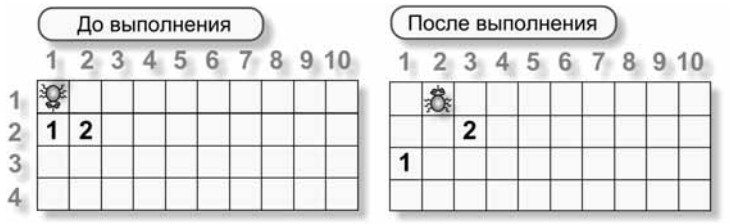

Рис. 4.3. Результат выполнения варианта 1

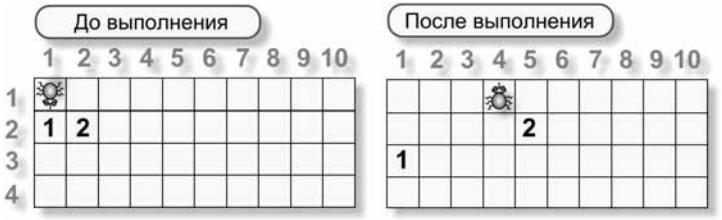

Рис. 4.4. Результат выполнения варианта 2

Ответы на вопросы и решение задач

4.1. Команда ветвления

1. Когда в программировании не обойтись без команды ветвления?

   Ответ. Без команды ветвления не обойтись, когда состояние среды становится известным только в процессе работы исполнителя (для Кукарачи это: местоположение исполнителя, количество кубиков на поле, их координаты и надписи на них). Если состояние среды полностью известно заранее, ветвления не нужны.

   Примеры.

   а) Космический корабль-исследователь приближается к неизвестной планете. Корабль послан в полёт без людей, в автоматическом режиме. Это означает, что корабль управляется по заранее написанной программе. Представляете, сколько разных условий, значения которых заранее неизвестны, должны обрабатываться в такой программе? Например, при посадке на планету нужно выпускать либо шасси-колеса, либо шасси-лыжи, либо шасси-поплавки. Выбор зависит от поверхности, на которую происходит посадка.

   б) Компьютерная игрушка перед тем, как сгенерировать на экране число препятствий и задать скорость движения «злодеев», проверяет, какой уровень сложности выбрал пользователь.

   в) Когда на клавиатуре нажимается клавиша с буквой, на экран выводится графический образ этой буквы. Но заранее неизвестно, какая клавиша будет нажата. Значит, процедура, обрабатывающая нажатия клавиш, должна предусмотреть реакцию на каждую из них.

2. На перевёрнутом кубике одна из букв Е, Д, Л. Помогите Кукараче посадить на поле дерево (рис. 4.5).

Рис. 4.5

Решение

```
ЭТО Дерево
  ВПРАВО
  ЕСЛИ        Е ТО Ель
  ИНАЧЕ ЕСЛИ Д ТО Дуб
  ИНАЧЕ            Липа
КОНЕЦ

ЭТО Дуб
  ПОВТОРИ 4 ВПРАВО
КОНЕЦ

ЭТО Ель
  Дуб
  ВНИЗ ВПРАВО
  ПОВТОРИ 5 ВВЕРХ
КОНЕЦ

ЭТО Липа
  Дуб
  ВВЕРХ ВПРАВО
  ПОВТОРИ 4 ВНИЗ
КОНЕЦ
```

4.2. Особые случаи

1. На перевёрнутых кубиках буквы О и У (какая буква на каком кубике — неизвестно) (рис. 4.6).

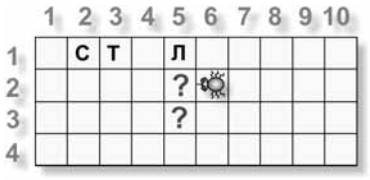

Рис. 4.6

Какое слово соберёт Кукарача, выполняя команды

а) `Мебель1`;

б) `Мебель2`;

в) `Мебель3`;

г) `Мебель4`;

д) `Мебель5`;

е) `Мебель6`;

ж) `Мебель7`;

з) `Мебель8`,

если в поле программы написан следующий текст:

```
//-------------------------------------
ЭТО Мебель1
  ВЛЕВО
  ЕСЛИ У
    ТО    Работа1
    ИНАЧЕ Работа2
КОНЕЦ
//-------------------------------------
ЭТО Мебель2
  ВЛЕВО
  ЕСЛИ НЕ У
    ТО    Работа1
    ИНАЧЕ Работа2
КОНЕЦ
```

```
//-------------------------------------
ЭТО Мебель3
  ВЛЕВО
  ЕСЛИ У
    ТО    Работа2
    ИНАЧЕ Работа1
КОНЕЦ
//-------------------------------------
ЭТО Мебель4
  ВЛЕВО
  ЕСЛИ НЕ У
    ТО    Работа2
    ИНАЧЕ Работа1
КОНЕЦ
//-------------------------------------
ЭТО Мебель5
  ВЛЕВО
  ЕСЛИ О
    ТО    Работа1
    ИНАЧЕ Работа2
КОНЕЦ
//-------------------------------------
ЭТО Мебель6
  ВЛЕВО
  ЕСЛИ НЕ О
    ТО    Работа1
    ИНАЧЕ Работа2
КОНЕЦ
//-------------------------------------
ЭТО Мебель7
  ВЛЕВО
  ЕСЛИ О
    ТО    Работа2
    ИНАЧЕ Работа1
КОНЕЦ
//-------------------------------------
ЭТО Мебель8
```

```
  ВЛЕВО
  ЕСЛИ НЕ О
    ТО    Работа2
    ИНАЧЕ Работа1
КОНЕЦ
//-------------------------------------
ЭТО Работа1
  ВНИЗ
  ВЛЕВО
  ВВЕРХ
КОНЕЦ
//-------------------------------------
ЭТО Работа2
  ВПРАВО
  ВНИЗ
  ВЛЕВО
  ВНИЗ
  ВЛЕВО
  ПОВТОРИ 2 ВВЕРХ
КОНЕЦ
```

Ответ. Процедуры `Мебель1`, `Мебель4`, `Мебель6`, `Мебель7` соберут слово СТУЛ, а процедуры `Мебель2`, `Мебель3`, `Мебель5`, `Мебель8` соберут слово СТОЛ, независимо от расположения букв на скрытых кубиках.

4.3. Задачи

1. На одном из перевёрнутых кубиков буква И. Составьте слово ИГРА (рис. 4.7).

Рис. 4.7

Решение

Описание алгоритма. Толкнуть первый кубик и, если это И, поставить его на место, иначе поставить на место второй кубик.

Программа

```
ЭТО Игра
  ВНИЗ
  ЕСЛИ И
    ТО    Первый_кубик
    ИНАЧЕ Второй_кубик
КОНЕЦ
ЭТО Первый_кубик
  ВНИЗ
КОНЕЦ

ЭТО Второй_кубик
  ВЛЕВО
  ПОВТОРИ 4 ВНИЗ
  ВПРАВО
  ВВЕРХ
КОНЕЦ
```

Тесты

Для проверки программы необходимо (и достаточно) запустить её два раза, например с данными табл. 4.3.

Таблица 4.3

Кубик в клетке (2,2)	Кубик в клетке (5,2)	Комментарий	Результат
И	Ш	Буква И на первом кубике	ИГРА
Ш	И	Буква И на втором кубике	ИГРА

2. Ниф-Ниф и Нуф-Нуф играют в крестики-нолики, Ниф-Ниф ставит крестики, а Нуф-Нуф — нолики. До победы остался один ход. Но чей это ход? (На перевёрнутом кубике либо X, либо 0. Кукарача должен поставить этот кубик в выигрышную позицию) (рис. 4.8).

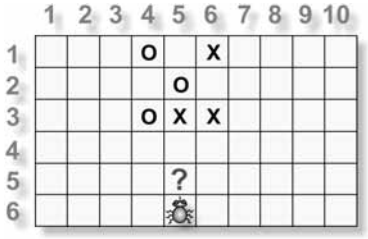

Рис. 4.8

Решение

Описание алгоритма. Толкнуть кубик. Если это X, поставить его в клетку (2,6), иначе поставить кубик в клетку (2,4).

Программа

```
ЭТО Крестики_нолики
  ВВЕРХ
  ЕСЛИ X
    ТО    Крестик
    ИНАЧЕ Нолик
КОНЕЦ

ЭТО Крестик
  ВЛЕВО ВВЕРХ
  ВПРАВО ВПРАВО
  ВНИЗ ВПРАВО
  ВВЕРХ ВВЕРХ
  ВПРАВО ВВЕРХ
  ВЛЕВО
КОНЕЦ

ЭТО Нолик
  ВПРАВО ВВЕРХ
  ВЛЕВО ВЛЕВО
  ВНИЗ ВЛЕВО
  ВВЕРХ ВВЕРХ
  ВЛЕВО ВВЕРХ
  ВПРАВО
КОНЕЦ
```

Тесты

Для проверки программы необходимо (и достаточно) запустить её два раза (табл. 4.4).

Таблица 4.4

Кубик в клетке (5,5)	Комментарий	Результат
X	На кубике крестик	Крестик в клетке (2,6)
0	На кубике нолик	Нолик в клетке (2,4)

3. На перевёрнутом кубике либо буква З, либо буква Л. Помогите Кукараче совершить космическое путешествие (рис. 4.9).

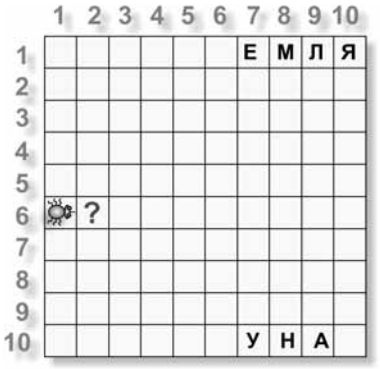

Рис. 4.9

Решение

Описание алгоритма. Толкнуть кубик. Если это З, поставить его в клетку (1,6), иначе поставить кубик в клетку (9,6).

Программа

```
ЭТО Космос
  ПОВТОРИ 4 ВПРАВО
   ЕСЛИ З
     ТО    Земля
     ИНАЧЕ Луна
КОНЕЦ


ЭТО Земля
```

```
  ВНИЗ ВПРАВО
  ПОВТОРИ 5 ВВЕРХ
КОНЕЦ

ЭТО Луна
  ВВЕРХ ВПРАВО
  ПОВТОРИ 4 ВНИЗ
КОНЕЦ
```

Тесты

Для проверки программы необходимо (и достаточно) запустить её два раза (табл. 4.5).

Таблица 4.5

Кубик в клетке (6,2)	Результат
З	ЗЕМЛЯ
Л	ЛУНА

4. На одном из кубиков буква К. Составьте слово КЛАД (рис. 4.10).

Рис. 4.10

Решение

Описание алгоритма. Толкнуть первый кубик вправо. Если это К, то работа закончена, иначе поставить на место второй кубик.

Программа

```
ЭТО Клад
  ВПРАВО
  ЕСЛИ НЕ К ТО Второй_кубик
КОНЕЦ

ЭТО Второй_кубик
```

```
  ВНИЗ ВНИЗ
  ВПРАВО
  ВВЕРХ
КОНЕЦ
```

Тесты

Для проверки программы необходимо (и достаточно) запустить её два раза (табл. 4.6).

Таблица 4.6

Кубик в клетке (2,2)	Кубик в клетке (3,3)	Комментарий	Результат
К	Ш	Буква К на первом кубике	КЛАД
Ш	К	Буква К на втором кубике	КЛАД

5. На одном из кубиков — буква М. Составить слово МЕДВЕДЬ и вернуть Кукарачу в исходную позицию. Лишнюю букву выбросить за пределы поля (рис. 4.11).

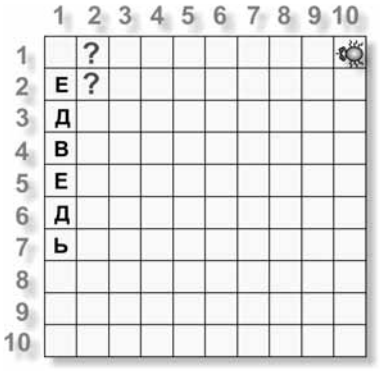

Рис. 4.11

Решение

Алгоритм

1. Толкнуть первый кубик влево.
2. Если это М, то удалить второй кубик, иначе удалить первый кубик и поставить на место второй.
3. Вернуть Кукарачу на место.

Программа

```
ЭТО Медведь
  // Толкнём первый кубик
  ПОВТОРИ 8 ВЛЕВО
  // Поставим нужный кубик
  ЕСЛИ М
    ТО    Первый_кубик
    ИНАЧЕ Второй_кубик
  // Вернём Кукарачу на место
ПОВТОРИ 8 ВПРАВО
КОНЕЦ

ЭТО Первый_кубик
  ВПРАВО
  ВНИЗ ВНИЗ
  ВЛЕВО
  ВВЕРХ ВВЕРХ
КОНЕЦ

ЭТО Второй_кубик
  ВПРАВО
  ВНИЗ ВНИЗ
  ВЛЕВО
  ВВЕРХ
  ВПРАВО
  ВВЕРХ
  ВЛЕВО
КОНЕЦ
```

Комментарий. Возврат Кукарачи на место начинается с клетки (1,2). Процедуры `Первый_кубик` и `Второй_кубик` обеспечивают такое положение исполнителя по окончании работы.

Тесты

Для проверки программы необходимо (и достаточно) запустить её два раза (табл. 4.7).

Таблица 4.7

Кубик в клетке (1,2)	Кубик в клетке (2,2)	Результат
М	Ш	МЕДВЕДЬ
Ш	М	МЕДВЕДЬ

6. Помогите Кукараче составить слово КОТ. На одном из кубиков есть буква Т (рис. 4.12).

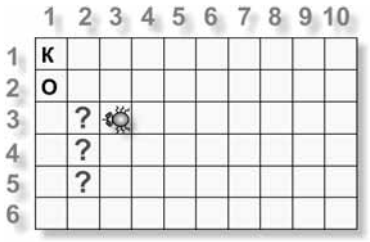

Рис. 4.12

Решение

Алгоритм. По очереди проверяем кубики, пока не найдём букву Т.

Программа

```
ЭТО Кот
  // Толкнём первый кубик
  ВЛЕВО // Если это Т, задача решена
  ЕСЛИ НЕ Т ТО Второй_кубик
КОНЕЦ

ЭТО Второй_кубик
  // Удалим первый кубик с поля
  ВЛЕВО
  // Подойдём ко второму кубику
  ВПРАВО ВПРАВО ВНИЗ
  // Толкнём второй кубик
  ВЛЕВО
  ЕСЛИ НЕ Т
    ТО Третий_кубик
    ИНАЧЕ Поставить_второй_кубик
КОНЕЦ
```

```
ЭТО Поставить_второй_кубик
  ВНИЗ ВЛЕВО ВВЕРХ
КОНЕЦ

ЭТО Третий_кубик
  // Удалим второй кубик с поля
  ВЛЕВО
  // Подойдём ко второму кубику
  ВПРАВО ВПРАВО ВНИЗ
  // Толкнём второй кубик
  ВЛЕВО
  Поставить_третий_кубик
КОНЕЦ

ЭТО Поставить_третий_кубик
  ВНИЗ ВЛЕВО ВВЕРХ ВВЕРХ
КОНЕЦ
```

Тесты

Для проверки программы необходимо (и достаточно) запустить её три раза (табл. 4.8).

Таблица 4.8

Кубик в клетке (3,2)	Кубик в клетке (4,2)	Кубик в клетке (5,2)	Результат
Т	Ш	Л	КОТ
Ш	Т	Л	КОТ
Л	Ш	Т	КОТ

7. Во второй строке злоумышленник перемешал буквы в слове ЭКРАН. Восстановить испорченное слово (рис. 4.13).

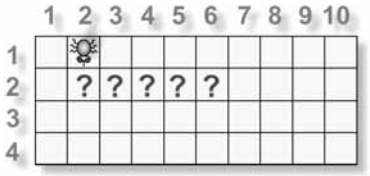

Рис. 4.13

Решение

Описание алгоритма. Решение основано на следующей идее: переместить буквы в строки 3–7, а затем собрать слово в 10 столбце. При этом буква Э перемещается в третью строку, буква К — в четвёртую и т. д., наконец, буква Н перемещается в 7 строку. Таким образом, независимо от того, как стояли буквы изначально, они будут стоять в правильном порядке по строкам. Остаётся только сместить их в один столбец.

Идею решения демонстрирует следующая схема:

```
1 ..........    ..........    ..........
2 .КРЭНА....    ..........    ..........
3 ..........    ...Э......    .........Э
4 .......... -> .К........ -> .........К
5 ..........    ..Р.......    .........Р
6 ..........    .....А....    .........А
7 ..........    ....Н.....    .........Н
```

Программа

```
ЭТО Экран
  ПОВТОРИ 5 Буква_в_нужную_строку
  Исполнитель_в_угол
  Установка_слова
КОНЕЦ

ЭТО Буква_в_нужную_строку
  Работа
  ВПРАВО
КОНЕЦ

ЭТО Работа
  ВНИЗ
  ЕСЛИ       К ТО Поставить_К
  ИНАЧЕ ЕСЛИ Р ТО Поставить_Р
  ИНАЧЕ ЕСЛИ А ТО Поставить_А
  ИНАЧЕ ЕСЛИ Н ТО Поставить_Н
  ВВЕРХ
КОНЕЦ

ЭТО Поставить_К
```

```
  ВНИЗ
  ВВЕРХ
КОНЕЦ

ЭТО Поставить_Р
  ПОВТОРИ 2 ВНИЗ
  ПОВТОРИ 2 ВВЕРХ
КОНЕЦ

ЭТО Поставить_А
  ПОВТОРИ 3 ВНИЗ
  ПОВТОРИ 3 ВВЕРХ
КОНЕЦ

ЭТО Поставить_Н
  ПОВТОРИ 4 ВНИЗ
  ПОВТОРИ 4 ВВЕРХ
КОНЕЦ

ЭТО Исполнитель_в_угол
  ПОВТОРИ 6 ВЛЕВО
КОНЕЦ

ЭТО Установка_слова
  ПОВТОРИ 2 ВНИЗ
  ПОВТОРИ 5 Установка_буквы
КОНЕЦ

ЭТО Установка_буквы
  ПОВТОРИ 8 ВПРАВО
  ПОВТОРИ 8 ВЛЕВО
  ВНИЗ
КОНЕЦ
```

Тесты

Для проверки именно этого алгоритма достаточно запустить программу один раз с произвольным расположением букв Э, К, Р, А, Н на кубиках — ведь для этого алгоритма безразлично, на каком кубике какая буква находится.

Для проверки же всех возможных вариантов начальных состояний пришлось бы выполнить 5! = 120 проверочных запусков.

8. (Автор Е. П. Лилитко) Кукарача расположен в верхнем левом углу поля. Где-то на поле находится одна буква. Других кубиков на поле нет. Требуется найти букву и встать на её место. Сообщение «Не могу» допускается только в том случае, если буквы на поле не оказалось (рис. 4.14).

Решение

```
ЭТО Поиск
  ПОВТОРИ 5 Вниз_вверх
КОНЕЦ


ЭТО Вниз_вверх
  ПОВТОРИ 9 ЕСЛИ ПУСТО ТО ВНИЗ
  ЕСЛИ ПУСТО ТО ВПРАВО
  ПОВТОРИ 9 ЕСЛИ ПУСТО ТО ВВЕРХ
  ЕСЛИ ПУСТО ТО ВПРАВО
КОНЕЦ
```

Тесты

Для полной проверки программы для поля 10×10 нужно выполнить 100 запусков, располагая кубик по очереди во всех клетках поля. На практике можно ограничиться 9 проверками, учитывая только крайние и средние положения кубика. Возможные положения буквы для тестирования показаны на рис. 4.15.

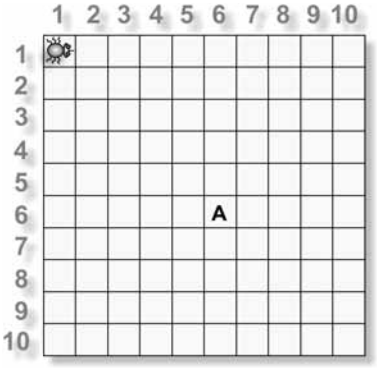

Рис. 4.14

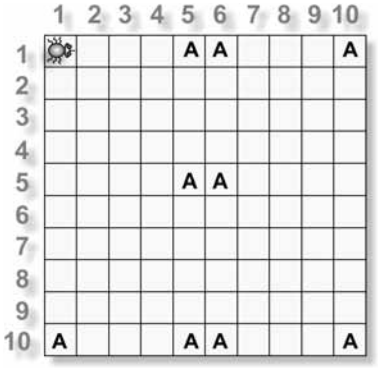

Рис. 4.15

Глава 5

Другой тип повторения

Общие рекомендации и дополнения

Как и в случае с командой ветвления, одного запуска программы с циклом **пока** недостаточно для проверки её правильности. При этом проверить все варианты, как правило, не представляется возможным. Обычно проверяют несколько типичных случаев и все предельные. Для Кукарачи предельными можно считать случаи, когда на поле установлено минимально и максимально возможное число кубиков, когда кубики расположены в предельно допустимых позициях и принимают предельно возможные значения.

При такой проверке полной уверенности в правильности программы нет, но число ошибок существенно уменьшается.

Составная команда не зря была введена самой последней из всех конструкций языка. Важно с самого начала привыкнуть к процедурному стилю, который помогает формулировать алгоритм, упрощает программирование и делает тексты программ более наглядными. Составная команда провоцирует написание длинных и запутанных кодов, хотя её разумное использование часто очень полезно.

Ответы на вопросы и решение задач

5.1. Когда неизвестно число повторений

В каких случаях при программировании придётся воспользоваться командой **пока**, а в каких — командой **повтори**?

1. Кукарача спускается по диагонали из клетки (1,1) до буквы С, расположенной в клетке (8,8).

   Ответ. **повтори** — расположение буквы С известно заранее, значит, заранее известно число шагов по диагонали.

2. Кукарача спускается по диагонали из клетки (1,1) до буквы С, расположенной где-то на этой диагонали.

   Ответ. **ПОКА** — расположение буквы заранее неизвестно.

3. Робот-станок должен изготовить 100 одинаковых деталей.

   Ответ. **ПОВТОРИ** — робот должен повторить 100 раз одни и те же действия, приводящие к изготовлению одной детали.

4. Робот-станок должен изготовлять одинаковые детали в течение дневной рабочей смены.

   Ответ. **ПОКА** — условием в этой команде будет проверка текущего времени.

5. Робот-станок должен изготовлять одинаковые детали до тех пор, пока у него не кончатся заготовки.

   Ответ. **ПОКА** — условием будет проверка наличия заготовок.

6. Компьютерная программа должна обрабатывать длительное нажатие символьной клавиши на клавиатуре.

   Ответ.

```
ПОКА пользователь не отпустил клавишу
{
  Повторять вывод на экран символа,
  соответствующего нажатой клавише
}
```

7. Автомат отключает двигатель стиральной машины через 5 минут после включения.

   Ответ.

```
Включить двигатель
ПОКА не прошли 5 минут
{
  Проверять время
}
Выключить двигатель
```

8. Автомат отключает двигатель стиральной машины через 10 000 оборотов двигателя.

   Ответ.

```
Включить двигатель
ПОКА число оборотов меньше 10 000
{
  Считать обороты
}
Выключить двигатель
```

5.2. Как обмануть интерпретатор

1. Когда лучше воспользоваться дополнительной процедурой, а когда составной командой?

   Ответ. Когда фрагмент программного кода решает отдельную подзадачу, возникшую при проектировании общего решения, его надо оформлять как процедуру.

   Когда фрагмент программного кода повторяется в программе многократно, его надо оформлять как процедуру.

   Если фрагмент программного кода не решает отдельных подзадач, не повторяется многократно, то он лучше воспринимается в виде составной команды — тела цикла или ветви условной команды.

5.3. Задачи

1. Загнать мяч (символ «*») в корзину, образованную буквами К. Заодно «починить» корзину — придвинуть первую букву К к остальным. Положение корзины заранее неизвестно, известно только, что её верхний край расположен на пятой строке (рис. 5.1).

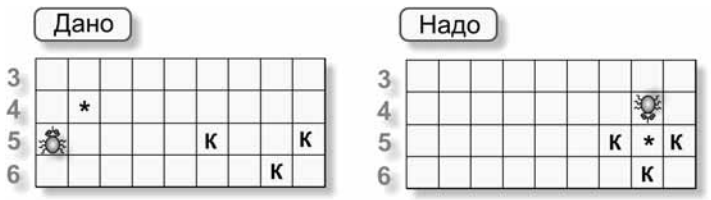

Рис. 5.1

Решение

Описание алгоритма. Будем перемещать исполнителя вместе с мячом (символом «*») к корзине, пока он не натолкнётся на букву К. Затем переместим мяч в корзину.

Программа

```
ЭТО Мяч
  Шаг_к_корзине
  ПОКА НЕ К
  {
    Мяч_к_корзине
    В_строку_с_корзиной
```

```
    Шаг_к_корзине
  }
  Мяч_в_корзину
КОНЕЦ

ЭТО Шаг_к_корзине
  ВПРАВО
КОНЕЦ

ЭТО Мяч_к_корзине
  ВЛЕВО ВВЕРХ ВПРАВО
КОНЕЦ

ЭТО В_строку_с_корзиной
  ВНИЗ
КОНЕЦ

ЭТО Мяч_в_корзину
  Мяч_к_корзине
  ВПРАВО ВВЕРХ ВПРАВО ВНИЗ
КОНЕЦ
```

Замечание. Отсутствие комментариев в программе компенсируется развёрнутыми содержательными именами процедур.

Тесты

На поле 10×10 количество возможных начальных состояний среды равно шести. Необходимо проверить, по крайней мере, три из этих состояний (по расположению ближней по отношению к исполнителю буквы К в пятой строке):

- буква К расположена рядом с исполнителем в клетке (5,2);
- буква К расположена в среднем положении, например, в клетке (5,5);
- буква К расположена максимально далеко от исполнителя в клетке (5,7).

2. Нарастить стену на один кирпич (кирпич изображается буквой Ж). После работы стена должна сместиться в шестой столбец. На рис. 5.2 показана наибольшая возможная длина стены в исходном состоянии среды.

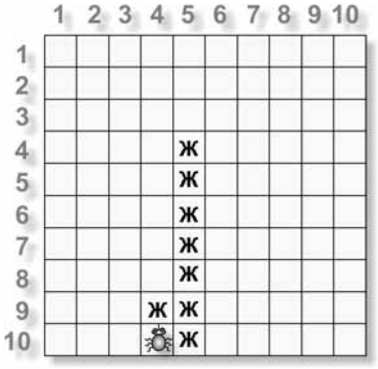

Рис. 5.2

Решение

Описание алгоритма. Будем перемещать «кирпич» вверх, пока не обнаружим конец стены. Затем установим «кирпич» на стену.

Программа

```
ЭТО Стена
  // Толкнуть кирпич стены в шестой столбец
  ВПРАВО
  // Перемещаем кирпич, пока не кончится стена
  ПОКА Ж Шаг
  Кирпич_на_стену
КОНЕЦ

ЭТО Шаг
  // Кирпич вверх
  ВЛЕВО ВВЕРХ
  // Толкнуть кирпич стены в шестой столбец
  ВПРАВО
КОНЕЦ

ЭТО Кирпич_на_стену
  ВЛЕВО ВЛЕВО ВВЕРХ ВПРАВО ВПРАВО
  ВВЕРХ ВПРАВО ВНИЗ
КОНЕЦ
```

Тесты

На поле 10×10 количество возможных начальных состояний среды равно восьми. Желательно проверить, по крайней мере, четыре из этих состояний (по числу кирпичей стены в исходном состоянии среды):

- в стене нет кирпичей (первый особый случай);
- стена из одного «кирпича» (второй особый случай);
- стена из трёх кирпичей (один из промежуточных вариантов);
- стена из семи «кирпичей» (третий особый случай).

3. Провести исполнителя по прямоугольнику А-Б-В-Г. Размеры прямоугольника заранее неизвестны (рис. 5.3).

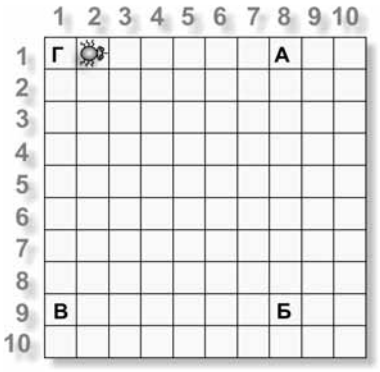

Рис. 5.3

Решение

Алгоритм

1. Идти вправо, пока пусто.
2. Идти вниз, пока пусто.
3. Идти влево, пока пусто.
4. Идти вверх, пока пусто.

Программа

```
ЭТО А_Б_В_Г
  ВПРАВО
  ПОКА ПУСТО ВПРАВО
  ВНИЗ
  ПОКА ПУСТО ВНИЗ
  ВЛЕВО
```

```
  ПОКА ПУСТО ВЛЕВО
  ВВЕРХ
  ПОКА ПУСТО ВВЕРХ
КОНЕЦ
```

Тесты

Желательно проверить, по крайней мере, два особых положения букв на поле и одно промежуточное. Положение кубиков для каждого теста приводятся в табл. 5.1.

Таблица 5.1

Номер теста	А	Б	В	Г
1	(1,3)	(2,3)	(2,1)	(1,1)
2	(1,7)	(8,7)	(8,1)	(1,1)
3	(1,10)	(10,10)	(10,1)	(1,1)

4. На одном из перевёрнутых кубиков во втором столбце — буква Ш. Найдите её и составьте слово ИШАК. Букву Ф с поля надо убрать (рис. 5.4).

Рис. 5.4

Решение

Алгоритм

1. Найти букву Ш во втором столбце.
2. Переместить букву Ш в шестой столбец.
3. Найти букву Ф и выбросить её с поля.
4. Поставить букву Ш на место.

Программа

```
ЭТО Ишак
  Найти_Ш
  Ш_в_шестой_столбец
  Удалить_Ф
  Поставить_Ш
КОНЕЦ

ЭТО Найти_Ш
  ВПРАВО
  ПОКА НЕ Ш Поиск
КОНЕЦ

ЭТО Поиск
  ВЛЕВО ВНИЗ ВПРАВО
КОНЕЦ

ЭТО Ш_в_шестой_столбец
  ПОВТОРИ 3 ВПРАВО
КОНЕЦ

ЭТО Удалить_Ф
  // Определиться на "местности" (поиск Ф)
  ВЛЕВО
  ПОКА НЕ Ф ВНИЗ
  // Выбросить Ф с поля
  ВНИЗ
КОНЕЦ

ЭТО Поставить_Ш
  // Исполнителя в первую строку
  ПОВТОРИ 9 ВВЕРХ
  // Исполнителя в клетку (1,6)
  ВПРАВО ВПРАВО
  // Букву Ш на место
  ПОВТОРИ 8 ВНИЗ
КОНЕЦ
```

Тесты

Нужно проверить, по крайней мере, два крайних положения буквы Ш и одно промежуточное (табл. 5.2).

Таблица 5.2

Номер теста	Положение буквы Ш
1	(2,2)
2	(5,2)
3	(8,2)

5. На одном из перевёрнутых кубиков в пятой строке поля Кукарачи — буква Ь. Найдите её и составьте слово МЫШЬ. В первом столбце третьей строки расположена буква Ф — её надо спихнуть в ров (рис. 5.5).

Рис. 5.5

Решение

Алгоритм

1. Найти букву Ь в пятой строке.
2. Переместить букву Ь во вторую строку.
3. Найти букву Ф и выбросить её с поля.
4. Поставить букву Ь на место.

Программа

```
ЭТО Мышь
  Найти_Ь
  Ь_во_вторую_строку
```

```
  Удалить_Ф
  Поставить_Ь
КОНЕЦ

ЭТО Найти_Ь
  ВВЕРХ
  ПОКА НЕ Ь Поиск
КОНЕЦ

ЭТО Поиск
  ВНИЗ ВЛЕВО ВВЕРХ
КОНЕЦ

ЭТО Ь_во_вторую_строку
  ПОВТОРИ 2 ВВЕРХ
КОНЕЦ

ЭТО Удалить_Ф
  // Определиться на "местности"
  ПОКА НЕ Ф ВЛЕВО
  // Удалить Ф
  ВВЕРХ
КОНЕЦ

ЭТО Поставить_Ь
  // Букву Ь в 9 столбец
  ПОВТОРИ 7 ВПРАВО
  // Букву Ь в первую строку
  ВНИЗ ВПРАВО ВВЕРХ
  // Подойти к букве Ь справа
  ВПРАВО ВВЕРХ
  // Поставить Ь на место
  ПОВТОРИ 5 ВЛЕВО
КОНЕЦ
```

Тесты

Нужно проверить, по крайней мере, два крайних положения буквы Ь и одно промежуточное (табл. 5.3).

Таблица 5.3

Номер теста	Положение буквы Ь
1	(5,2)
2	(5,5)
3	(5,9)

6. Во второй строке со второй позиции записано число, все цифры которого разные. Упорядочить цифры числа в порядке возрастания, если известно, что среди них нет нулей и единиц. Букву Н, расположенную в клетке (1,1), удалить. На рис. 5.6 показано возможное состояние среды в начальный момент.

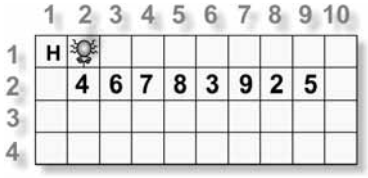

Рис. 5.6

Решение

Алгоритм. Будем использовать ту же идею, что и программа для задачи 7 про ЭКРАН из предыдущей главы: перемещаем каждую цифру в строку, номер которой на единицу больше этой цифры, затем смещаем все цифры в один столбец.

```
ЭТО Порядок
  Распределение_цифр_по_строкам
  Сборка_числа_в_10_столбце
КОНЕЦ

ЭТО Распределение_цифр_по_строкам
  // Проверка клетки под исполнителем
  ВНИЗ
  ПОКА НЕ ПУСТО
  {
    Цифру_в_нужную_строку
    // Исполнитель в первую строку
    ВВЕРХ
    // Смещение в следующий столбец
```

```
    ВПРАВО
    // Проверка клетки под исполнителем
    ВНИЗ
  }
КОНЕЦ

ЭТО Сборка_числа_в_10_столбце
  // Исполнителя в клетку (1,1)
  ВВЕРХ
  ПОКА ПУСТО ВЛЕВО
  // Установка цифр в последнем столбце
  ВНИЗ
  ПОВТОРИ 8
  {
    ВНИЗ
    ПОВТОРИ 8 ВПРАВО
    ПОВТОРИ 8 ВЛЕВО
  }
КОНЕЦ

// Переключатель
ЭТО Цифру_в_нужную_строку
  ЕСЛИ       3 ТО Цифра_3
  ИНАЧЕ ЕСЛИ 4 ТО Цифра_4
  ИНАЧЕ ЕСЛИ 5 ТО Цифра_5
  ИНАЧЕ ЕСЛИ 6 ТО Цифра_6
  ИНАЧЕ ЕСЛИ 7 ТО Цифра_7
  ИНАЧЕ ЕСЛИ 8 ТО Цифра_8
  ИНАЧЕ ЕСЛИ 9 ТО Цифра_9
КОНЕЦ

ЭТО Цифра_3
  ВНИЗ
  ВВЕРХ
КОНЕЦ

ЭТО Цифра_4
  ПОВТОРИ 2 ВНИЗ
```

```
  ПОВТОРИ 2 ВВЕРХ
КОНЕЦ

ЭТО Цифра_5
  ПОВТОРИ 3 ВНИЗ
  ПОВТОРИ 3 ВВЕРХ
КОНЕЦ

ЭТО Цифра_6
  ПОВТОРИ 4 ВНИЗ
  ПОВТОРИ 4 ВВЕРХ
КОНЕЦ

ЭТО Цифра_7
  ПОВТОРИ 5 ВНИЗ
  ПОВТОРИ 5 ВВЕРХ
КОНЕЦ
ЭТО Цифра_8
  ПОВТОРИ 6 ВНИЗ
  ПОВТОРИ 6 ВВЕРХ
КОНЕЦ

ЭТО Цифра_9
  ПОВТОРИ 7 ВНИЗ
  ПОВТОРИ 7 ВВЕРХ
КОНЕЦ
```

Тесты

Программу можно проверить для числа, состоящего из одной цифры (особый случай), восьми цифр (особый случай), пяти цифр (промежуточный вариант) и для варианта, когда на поле нет ни одной цифры (хотелось бы, чтобы и в этом случае программа нормально завершала работу).

Глава 6

Кукарача хочет укусить себя за хвост

Рекурсия — это метод программирования, при котором процедура сама (**прямая** рекурсия) или через другие процедуры (**косвенная** рекурсия) вызывает саму себя. Рекурсия может никогда не заканчиваться, и в этом случае она называется **бесконечной**. Когда программист при помощи условий обеспечивает выход из рекурсивного цикла, рекурсия становится **конечной**.

Более подробное обсуждение рекурсии приводится в части III «Транслятор?.. Это очень просто!»

Ответы на вопросы и решение задач

6.5. Задачи

1. На одном из перевёрнутых кубиков — буква И. Найдите её и составьте слово ТИГР. Используйте кубик с буквой М для определения положения Кукарачи (рис. 6.1).

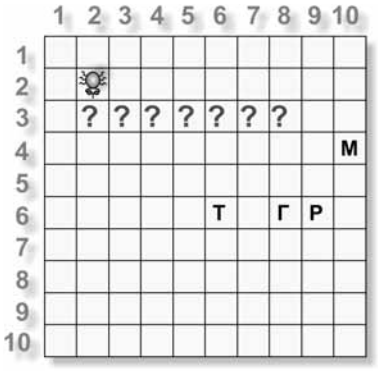

Рис. 6.1

Решение

Алгоритм главной процедуры `Тигр`

1. Толкнуть кубик.
2. Проверить кубик:
   2.1. Если это не И, то сделать:
      2.1.1. Подойти к следующему кубику.
      2.1.2. Вызвать процедуру `Тигр` (рекурсия).
   2.2. Если это И, то сделать:
      2.2.1. Толкнуть кубик в пятую строку
      2.2.2. Вызвать процедуру `Установка` (установка кубика на место).

Описание алгоритма процедуры `Установка`. Перемещать кубик И вправо, пока исполнитель не обнаружит букву М. Когда М обнаружена, положение исполнителя становится известным, и букву И легко поставить на место.

Программа

```
ЭТО Тигр
  // Толкнуть кубик
  ВНИЗ
  // Проверить кубик
  ЕСЛИ НЕ И
    // Если не И, то продолжить поиск
    ТО { ВВЕРХ ВПРАВО Тигр }
    // Если И, установить кубик на место
    ИНАЧЕ { ВНИЗ Установка }
КОНЕЦ
// Перед выполнением процедуры исполнитель в 4-ой строке,
// а кубик под ним строкой ниже.
ЭТО Установка
  // Шаг в направлении М
  ВПРАВО
  ПОКА ПУСТО Шаг
  Установка_И
КОНЕЦ


ЭТО Шаг
  // Букву И на клетку вправо
```

```
  ВЛЕВО ВЛЕВО ВНИЗ ВПРАВО
  // Исполнителя в четвёртую строку
  ВВЕРХ ВПРАВО
  // Шаг в направлении М
  ВПРАВО
КОНЕЦ

ЭТО Установка_И
  ВНИЗ ВЛЕВО ВЛЕВО
  ВВЕРХ ВЛЕВО ВНИЗ
КОНЕЦ
```

Тесты

Нужно проверить, по крайней мере, два крайних положения буквы И и одно промежуточное (табл. 6.1).

Таблица 6.1

Номер теста	Положение буквы И	Комментарий
1	(3,2)	Особый случай
2	(3,8)	Особый случай
3	(3,5)	Промежуточный вариант

2. Исполнитель располагается в левом верхнем углу поля. Кубики с буквами стоят ниже него по одному в каждом столбце (их расположение в столбце произвольно, но не выше второй строки). Написать программу, которая перемещает все эти кубики на одну строку ниже (рис. 6.2).

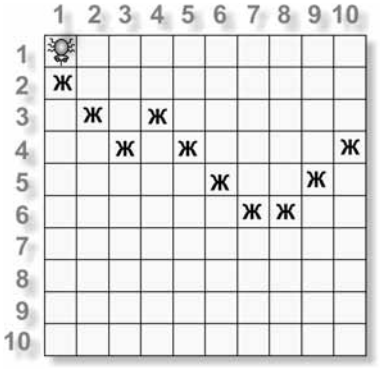

Рис. 6.2

Решение

Алгоритм главной процедуры Смещение

1. Повторить 9 раз (для первых девяти столбцов):
   1.1. Сместить кубик вниз и вернуть исполнителя в первую строку (вызов процедуры Смещение_кубика).
   1.2. Переместить исполнителя в следующий столбец.
2. Отдельно сместить кубик в десятом столбце (вызов процедуры Смещение_кубика) без перемещения исполнителя в следующий столбец (чтобы не выйти за пределы поля).

Описание процедуры Смещение_кубика. Рекурсивно перемещаем исполнителя вниз (команда **ВНИЗ**), пока клетки пусты. Возврат на прежнее место обеспечивается отложенной в рекурсии командой **ВВЕРХ** («рекурсивной пружинкой» — этот ассоциативный термин предложила в 1999 году команда Роботландского университета под руководством Анатолия Владимировича Бычкова).

Программа

```
ЭТО Смещение
  // Работать в первых девяти столбцах
  ПОВТОРИ 9
  {
    // Сместить кубик вниз и вернуть исполнителя в первую строку
    Смещение_кубика
    // Переместить исполнителя в следующий столбец
    ВПРАВО
  }
  // Работать в десятом столбце
  Смещение_кубика
КОНЕЦ

// Смещение кубика вниз и возврат исполнителя в первую строку
ЭТО Смещение_кубика
  ВНИЗ
  ЕСЛИ ПУСТО ТО Смещение_кубика
  ВВЕРХ // Рекурсивная пружинка
КОНЕЦ
```

Тесты

Для каждого столбца (кроме десятого) выполняется в цикле один и тот же алгоритм:

1. Смещение кубика вниз.
2. Возврат в исходное положение.
3. Переход к следующему столбцу.

Отдельно обрабатывается лишь десятый столбец (нет перехода к следующему столбцу).

В силу этого тестирование может быть выполнено всего одним запуском программы, при этом нужно обязательно поставить один кубик во вторую строку, один в десятую (он будет удалён с поля), а остальные расположить произвольно в промежуточных строках.

3. Исполнитель располагается в клетке (1,1) поля внутри коридора неизвестной длины. Внизу коридора нижняя стенка сломана. Починить коридор и вернуть исполнителя в исходное положение (рис. 6.3).

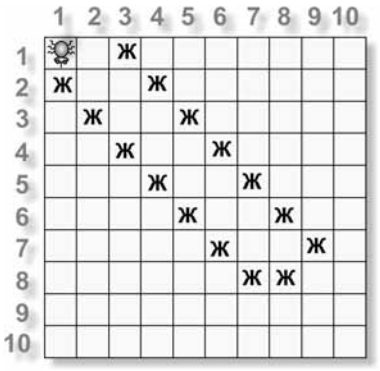

Рис. 6.3

Решение

Описание алгоритма. Рекурсивно перемещаем исполнителя по диагонали (команды **вправо вниз**), пока клетки пусты. Возврат на прежнее место обеспечивается отложенной в рекурсии группой команд **вверх влево** (рекурсивной пружинкой).

Программа

```
это Ремонт
  вправо вниз
  если пусто то Ремонт
```

```
  // Движение исполнителя назад после выхода из рекурсии
  ВВЕРХ ВЛЕВО // Рекурсивная пружинка
КОНЕЦ
```

Тесты

Проверим работу исполнителя в самом коротком, самом длинном коридоре (для поля 10×10) и коридоре промежуточной длины (табл. 6.2).

Таблица 6.2

Номер теста	Положение «поломанного» кирпича	Комментарий
1	(2,2)	Самый короткий коридор
2	(9,9)	Самый длинный коридор
3	(5,5)	Коридор промежуточной длины

4. Исполнитель находится в левом верхнем углу. Во второй строке, начиная с первой позиции, записано слово не длиннее 7 символов. Поставить Кукарачу в строку, номер которой равен длине этого слова (рис. 6.4).

Рис. 6.4

Решение

Описание алгоритма. Рекурсивно перемещаем исполнителя вдоль слова, толкая его буквы в третью строку (команды **ВНИЗ ВВЕРХ ВПРАВО**), пока слово не закончится. Перемещение исполнителя вниз на количество букв в слове обеспечим рекурсивной пружинкой **ВНИЗ**. Правда, номер строки при этом оказывается на 3 больше длины слова. В главной процедуре выполним соответствующую корректировку положения исполнителя.

Программа

```
ЭТО Счет
  Работа
  // Корректировка положения исполнителя
  ПОВТОРИ 3 ВВЕРХ
```

```
КОНЕЦ

ЭТО Работа
  ВНИЗ
  ЕСЛИ НЕ ПУСТО
    ТО
    {
      ВВЕРХ
      ВПРАВО
      Работа
    }
  // Движение исполнителя вниз после выхода из рекурсии
  ВНИЗ
КОНЕЦ
```

Тесты

Проверим работу программы для самого короткого слова (А), самого длинного (РАКУШКА) и для слова средней длины (ЛИСА).

Замечание

Частичную корректировку положения исполнителя (на одну команду **ВВЕРХ**) можно выполнить перед работой рекурсивной пружинки:

```
ЭТО Работа
  ВНИЗ
  ЕСЛИ НЕ ПУСТО
    ТО
    {
      ВВЕРХ
      ВПРАВО
      Работа
    }
    ИНАЧЕ ВВЕРХ // Корректировка положения исполнителя
                 // перед работой рекурсивной пружинки.
  ВНИЗ // Рекурсивная пружинка
КОНЕЦ
```

В главной процедуре теперь нужно выполнить не три, а только две команды **ВВЕРХ**, а главное, максимальная длина слова, при котором задача решается в описанных условиях, возрастает до 8 символов.

5. (Автор Е. П. Лилитко) Кукарача расположен в верхнем левом углу поля. Где-то под ним в первом столбце находится кубик. Требуется умножить номер строки, в которой стоит кубик, на два и поставить Кукарачу в соответствующую строку.

Решение

Описание алгоритма. Рекурсивно перемещаем исполнителя вниз, пока он не толкнёт кубик с буквой А. Рекурсивная пружинка (команда **вниз**) удвоит число шагов исполнителя, а в главной процедуре запишем поправку, так, чтобы номер строки его конечного положения был равен удвоенному номеру строки начального положения кубика.

Программа

```
// Автор решения: Е. П. Лилитко
ЭТО Умножение
  Умножить_на_2
  // Поправка положения исполнителя
  ВНИЗ
КОНЕЦ


ЭТО Умножить_на_2
  ВНИЗ
  ЕСЛИ ПУСТО ТО Умножить_на_2
  ВНИЗ
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 6.3.

Таблица 6.3

Номер теста	Положение	Комментарий
1	(2,1)	Первый особый случай
2	(5,1)	Второй особый случай
3	(3,1)	Промежуточный вариант
4	(4,1)	Промежуточный вариант

Замечание

Эти четыре теста исчерпывают все возможные начальные состояния среды для поля в 10 строк. Но программа является универсальной и будет работать для поля любой высоты.

6. (Автор Е. П. Лилитко) Требуется построить нового исполнителя, «управляемого данными», поведение которого будет полностью зависеть от расположения букв на поле. Изначально Кукарача движется вниз до тех пор, пока не встретит одну из букв В, Н, Л, П или С. Встретив одну из этих букв, он продолжает движение соответственно вверх, вниз, влево, вправо или немедленно останавливается. На дальнейшем пути Кукарачи может вновь повстречаться одна из приведённых букв, и тогда он вновь изменяет направление вышеописанным способом. Для нового исполнителя создайте программу (установите в нужные места буквы-команды и Кукарачу), заставляющую ходить его по расширяющейся спирали.

Решение

Описание алгоритма. Программа состоит из четырёх процедур, которые моделируют выполнение соответствующих команд новым исполнителем:

- `Идти_вниз` (работает после того, как исполнитель толкнул кубик Н). Рекурсивное движение вниз, пока не будет обнаружен кубик с новой командой:
  - П, Л, В — переход к выполнению соответствующей процедуры;
  - Н — продолжение рекурсивного движения вниз;
  - С — завершение рекурсии без отложенных команд (рекурсивных пружинок).
- `Идти_вверх` (после того, как исполнитель толкнул кубик В). Рекурсивное движение вверх, пока не будет обнаружен кубик с новой командой:
  - П, Л, Н — переход к выполнению соответствующей процедуры;
  - В — продолжение рекурсивного движения вверх;
  - С — завершение рекурсии без отложенных команд (рекурсивных пружинок).
- `Идти_влево` (после того, как исполнитель толкнул кубик Л). Рекурсивное движение влево, пока не будет обнаружен кубик с новой командой:
  - В, Н, П — переход к выполнению соответствующей процедуры;
  - Л — продолжение рекурсивного движения влево;
  - С — завершение рекурсии без отложенных команд (рекурсивных пружинок).
- `Идти_вправо` (после того, как исполнитель толкнул кубик П). Рекурсивное движение вниз, пока не будет обнаружен кубик с новой командой:
  - В, Н, Л — переход к выполнению соответствующей процедуры;
  - П — продолжение рекурсивного движения вправо;
  - С — завершение рекурсии без отложенных команд (рекурсивных пружинок).

По условию задачи программа должна начинать работу с процедуры Идти_вниз.

Программа

```
// Автор решения: Е. П. Лилитко
ЭТО Вход
  Идти_вниз
КОНЕЦ

ЭТО Идти_вниз
  ВНИЗ
  ЕСЛИ           П ТО Идти_вправо
  ИНАЧЕ ЕСЛИ     Л ТО Идти_влево
  ИНАЧЕ ЕСЛИ     В ТО Идти_вверх
  ИНАЧЕ ЕСЛИ НЕ  С ТО Идти_вниз
КОНЕЦ

ЭТО Идти_вверх
  ВВЕРХ
  ЕСЛИ           П ТО Идти_вправо
  ИНАЧЕ ЕСЛИ     Л ТО Идти_влево
  ИНАЧЕ ЕСЛИ     Н ТО Идти_вниз
  ИНАЧЕ ЕСЛИ НЕ  С ТО Идти_вверх
КОНЕЦ

ЭТО Идти_влево
  ВЛЕВО
  ЕСЛИ           П ТО Идти_вправо
  ИНАЧЕ ЕСЛИ     В ТО Идти_вверх
  ИНАЧЕ ЕСЛИ     Н ТО Идти_вниз
  ИНАЧЕ ЕСЛИ НЕ  С ТО Идти_влево
КОНЕЦ

ЭТО Идти_вправо
  ВПРАВО
  ЕСЛИ           Л ТО Идти_влево
  ИНАЧЕ ЕСЛИ     В ТО Идти_вверх
  ИНАЧЕ ЕСЛИ     Н ТО Идти_вниз
  ИНАЧЕ ЕСЛИ НЕ  С ТО Идти_вправо
КОНЕЦ
```

Программа для нового исполнителя «Поход по спирали» показана на рис. 6.5.

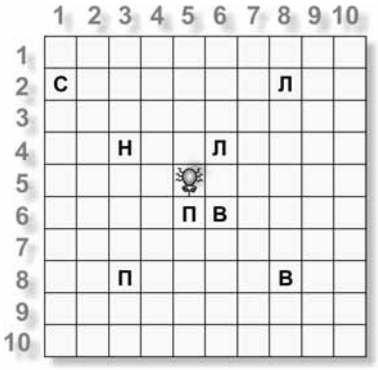

Рис. 6.5

Тесты

Начальная установка среды «Поход по спирали» является одним из возможных тестов, но явно недостаточным для проверки построенной программы. Необходимо протестировать программу и для других исходных данных, так чтобы проверить работу всех частей программного кода. Два возможных теста показаны на рис. 6.6.

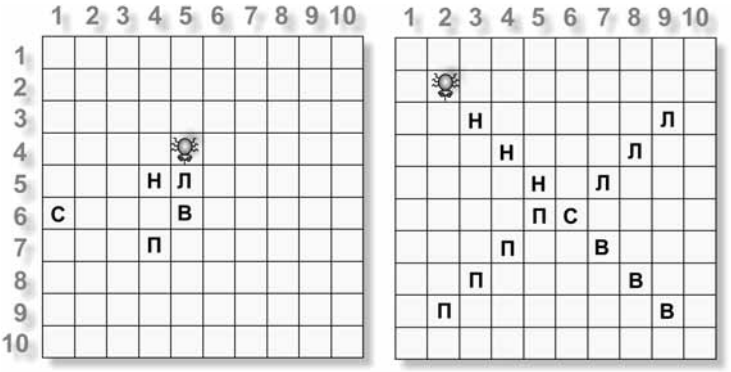

Рис. 6.6

7. Кукарача расположен в нижнем левом углу поля. Прямо над ним расположена плотная строка кубиков. Кубики начинаются с первого столбца, и их не больше восьми. Написать программу, выполняя которую Кукарача построит лесенку (как на рис. 6.7) и после завершения трудов остановится в правом верхнем углу поля.

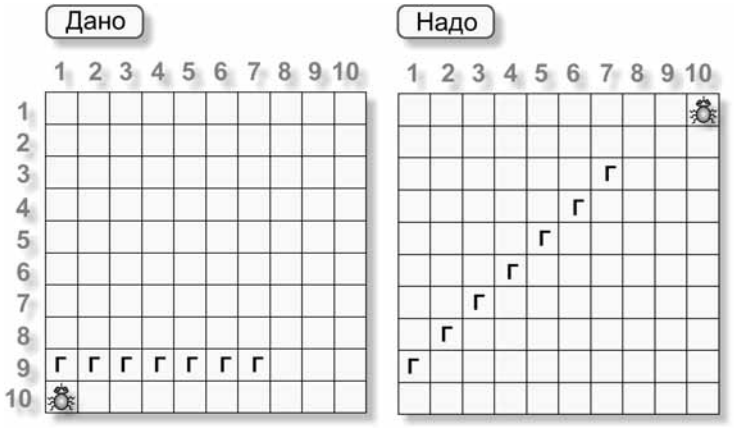

Рис. 6.7

Решение

Алгоритм главной процедуры Лесенка

1. Переместить исполнителя ко второму кубику.
2. Повторить 7 раз:
   2.1. Сместить горизонтальный ряд кубиков (без первого в ряду) на одну строку вверх (вызов процедуры На_строку_вверх).
3. Переместить исполнителя в правый верхний угол.

Описание процедуры На_строку_вверх. Рекурсивно перемещаем кубики вверх (команды **ВВЕРХ ВНИЗ ВПРАВО**), пока кубики в ряду не закончатся. Возврат на прежнее место обеспечивается отложенной в рекурсии командой **ВЛЕВО** (рекурсивной пружинкой). Для продолжения построения лесенки необходимо передвинуть исполнителя ко второму кубику в построенном ряду.

Программа

```
ЭТО Лесенка
  ВПРАВО
  ПОВТОРИ 7 На_строку_вверх
  ПОВТОРИ 2 ВВЕРХ
  ВПРАВО
КОНЕЦ

ЭТО На_строку_вверх
  ВВЕРХ
```

```
  ЕСЛИ НЕ ПУСТО
    ТО // Ступень
    {
      ВНИЗ
      ВПРАВО
      На_строку_вверх
      ВЛЕВО // Рекурсивная пружинка
    }
    ИНАЧЕ ВПРАВО // Коррекция
КОНЕЦ
```

Тесты

Нужно проверить работу программы для самого короткого ряда (1 кубик), самого длинного (8 кубиков) и для ряда средней длины (4 кубика).

8. (Автор Е. П. Лилитко) Кукарача расположен в нижнем левом углу поля. Первая строка и первый столбец поля пусты за исключением буквы Г, расположенной точно в левом верхнем углу поля. На остальном пространстве поля разбросано некоторое количество букв А, не более чем по одной на строку. Таким образом, некоторые строки содержат одну букву А, а некоторые — ни одной. Требуется очистить поле от букв, попутно подсчитывая их количество. В конце работы Кукарача должен остановиться в строке, номер которой равен количеству сброшенных в ров букв А. То есть если изначально на поле была одна буква А, Кукарача должен остановиться в первой строке, если две, то во второй и т. д. Сообщение «Не могу» допускается только в том случае, если ни одной буквы А на поле не оказалось.

Решение

Описание алгоритма. Будем рекурсивно перемещать исполнителя по строкам, пока не обнаружим кубик с буквой Г (всё поле пройдено). При этом каждый раз, когда встречается кубик с буквой А, будем передавать управление на специальную процедуру `Запомнить`, содержащую отложенную в рекурсии команду **ВНИЗ** (рекурсивную пружинку). После завершения рекурсии эта команда будет выполнена столько раз, сколько на поле было кубиков с буквой. Так как исполнитель начинает движение вниз с первой строки, необходима поправка (одна команда **ВВЕРХ**). Она предусмотрена в главной процедуре `Сосчитать`.

Программа

```
// Автор решения: Е. П. Лилитко
ЭТО Сосчитать
```

```
  Проход
  ВВЕРХ
КОНЕЦ

ЭТО Проход
  ПОВТОРИ 9 ВПРАВО
  ЕСЛИ А
    ТО    Запомнить
    ИНАЧЕ Новый_Проход
КОНЕЦ

ЭТО Новый_Проход
  ПОВТОРИ 9 ВЛЕВО
  ВВЕРХ
  ЕСЛИ НЕ Г ТО Проход
КОНЕЦ

ЭТО Запомнить
    Новый_Проход
    ВНИЗ
КОНЕЦ
```

Тесты

Нужно проверить работу программы, когда на поле:

- ❒ одна буква А (особый случай);
- ❒ 9 букв А (особый случай);
- ❒ 5 букв А (одно из промежуточных количеств букв А).

Дополнительно нужно проверить разные варианты расположения букв А на поле:

- ❒ во втором столбце (особый случай);
- ❒ в десятом столбце (особый случай);
- ❒ в 5 столбце (промежуточное положение).

Глава 7

Решения задач

Задачи недели 2002/2003 учебного года

Задачи к главе 4

1. Кук (В. П. Семенко, Рубцовск).

   Кукарача стоит в клетке (2,1). Справа от него находится слово КУК, между буквами которого вкрались пять лишних символов «*». Распределение звёздочек между буквами заранее неизвестно. Восстановить слово КУК, а лишние символы вытолкнуть с поля. Пример начального и конечного состояний среды показан на рис. 7.1.

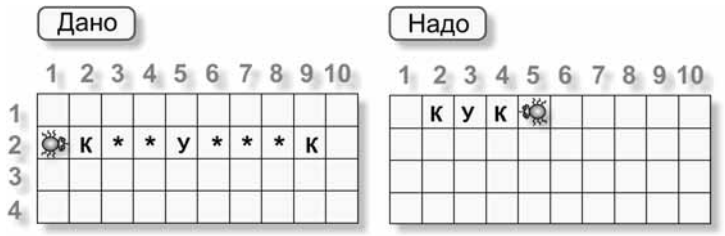

Рис. 7.1

Решение

Алгоритм

1. Толкай все 8 кубиков, и каждый раз проверяй, что толкнул: если это «*», то вытолкни кубик с поля и приготовься толкать следующий.
2. Склей слово КУК.

Программа

```
// Автор решения: А. В. Рудь
ЭТО Кук
  ПОВТОРИ 8
  {
    Толкни_кубик
    ЕСЛИ * ТО Вытолкни
  }
  Склей_слово
КОНЕЦ

ЭТО Толкни_кубик
  ВНИЗ ВПРАВО ВВЕРХ
КОНЕЦ

ЭТО Вытолкни
  ВВЕРХ ВНИЗ
КОНЕЦ

ЭТО Склей_слово
  ВПРАВО ВВЕРХ
  ПОВТОРИ 5 ВЛЕВО
КОНЕЦ
```

Замечание (В. П. Семенко)

Задачи такого типа приучают к процедурному программированию. Основная процедура отражает алгоритм решения, а вспомогательные процедуры реализуют «новые команды» исполнителя. Задача хороша и с пропедевтической точки зрения: к ней можно вернуться после знакомства с циклом **ПОКА** и предложить в условии слово произвольной длины с любым количеством звёздочек (с помощью рекурсии выброшенные звёздочки можно «подсчитать» и склеить слово).

Тесты

Набор тестов приводится в табл. 7.1.

Таблица 7.1

Начальное состояние среды	Комментарий	Ожидаемый результат
¤*****КУК	Все звёздочки в начале записи	КУК
¤КУК*****	Все звёздочки в конце записи	КУК
¤К***У**К	Все звёздочки внутри записи	КУК
¤*К**У*К*	Произвольное расположение звёздочек	КУК

2. Больной кот (Г. М. Федченко, Благовещенск).

   Вылечить кота: найти кубик с буквой Т и поставить её в клетку (3,1). Среди 7 перевёрнутых кубиков только один с буквой Т. Все лишние кубики нужно удалить с поля. Начальное и конечное состояния среды показаны на рис. 7.2.

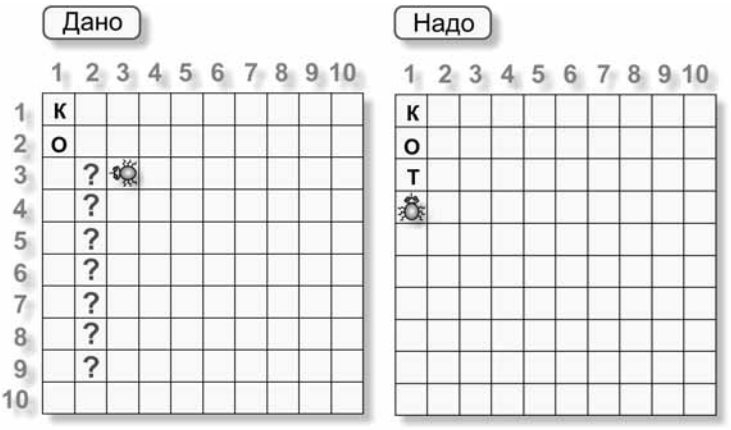

Рис. 7.2

Решение

Алгоритм

1. Проверить все 7 кубиков. На каждом шаге проверки выполнить:
   1.1. Толкнуть кубик влево.
   1.2. Если это не Т, вытолкнуть кубик с поля (за левую границу).
   1.3. Подойти к следующему кубику.
2. Встать в первый столбец и шесть раз шагнуть вверх (при этом кубик с буквой Т будет придвинут к слову, где бы он ни находился).

Программа

```
// Автор решения: А. В. Рудь
ЭТО Кот
  // Проверить все 7 кубиков
  ПОВТОРИ 7 Проверка
  // Поставить букву Т на место
  Установка
КОНЕЦ

ЭТО Проверка
```

```
  // Перевернуть кубик
  ВЛЕВО
  // Если это не буква Т, выбросить кубик с поля
  ЕСЛИ НЕ Т ТО Выбросить
  // Подойти к следующему кубику
  ВПРАВО ВНИЗ
КОНЕЦ

ЭТО Выбросить
  ВЛЕВО ВПРАВО
КОНЕЦ

// Поставить букву Т на место
ЭТО Установка
  ПОВТОРИ 2 ВЛЕВО
  ПОВТОРИ 6 ВВЕРХ
КОНЕЦ
```

Тесты

Проверка каждого из семи вариантов расположения буквы Т на перевёрнутых кубиках приведёт к полной проверке работы программы.

3. Шумы на линии (Г. М. Федченко, Благовещенск).

Кукарача получил сообщение: четырёхбуквенное слово. Но во время передачи в сообщение попали шумы — пять цифр. Полученное сообщение находится во второй строке, а Кукарача под первым его символом. Восстановить исходный текст. Пример начального и конечного состояний среды показан на рис. 7.3.

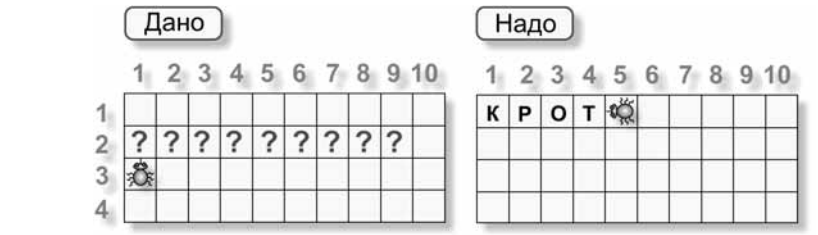

Рис. 7.3

Решение

Описание алгоритма. Толкнуть каждый кубик вверх. Если на кубике цифра, вытолкнуть его с поля. Когда все кубики проверены, уплотнить оставшиеся.

Алгоритм

1. Проверка. Проверить все 9 кубиков. На каждом шаге проверки выполнить:
   1.1. Толкнуть кубик вверх.
   1.2. Если это цифра, вытолкнуть кубик с поля (за верхнюю границу).
   1.3. Подойти к следующему кубику.
2. Уплотнение. Встать в первую строку и пять раз шагнуть влево (на число удалённых кубиков).

Программа

```
// Автор решения: А. В. Рудь
ЭТО Расшифровка
  ПОВТОРИ 9 Проверка
  Уплотнение
КОНЕЦ

ЭТО Проверка
  // Перевернуть кубик
  ВВЕРХ
  // Если это цифра, выбросить кубик с поля
  ЕСЛИ ЦИФРА ТО { ВВЕРХ ВНИЗ }
  // Подойти к следующему кубику
  ВНИЗ ВПРАВО
КОНЕЦ

ЭТО Уплотнение
  // Встать в первую строку
  ПОВТОРИ 2 ВВЕРХ
  // Пять раз шагнуть влево (на число удалённых кубиков)
  ПОВТОРИ 5 ВЛЕВО
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 7.2.

Таблица 7.2

Начальное состояние среды	Комментарий	Ожидаемый результат
12345КРОТ	Все шумы в начале записи	КРОТ
КРОТ12345	Все шумы в конце записи	КРОТ
К1Р23О45Т	Все шумы в середине записи	КРОТ
1К2Р3О4Т5	Произвольное расположение шумов в записи	КРОТ
1К23РО4Т5		КРОТ

4. Чаепитие в Кукарачинске (Кристина Иванова, 6 класс, Кострома).

Научить Кукарачу накрывать стол для чаепития. Если в клетке (5,2) буква А, то получить слова КРУЖКА и ЧАЙНИК. В противном случае — убрать со стола, т. е. все кубики столкнуть с рабочего поля. Начальное состояние среды и пример конечного состояния показаны на рис. 7.4.

Рис. 7.4

Решение

Описание алгоритма. Перевернуть кубик над Кукарачей. Если на нём окажется буква А, раздвинуть пары кубиков в строках 4, 6, 8, иначе очистить столбцы 6 и 7, затем точно так же очистить столбцы 3 и 4.

Программа

```
// Автор решения: А. В. Рудь
ЭТО Чай
  ВВЕРХ
```

```
    ЕСЛИ А
     ТО    Накрыть_стол
     ИНАЧЕ Уборка
КОНЕЦ

ЭТО Накрыть_стол
  ПОВТОРИ 3 Раздвижка
КОНЕЦ

ЭТО Раздвижка
  ПОВТОРИ 2 ВНИЗ
  ВЛЕВО
  ПОВТОРИ 2 ВПРАВО
  ВЛЕВО
КОНЕЦ

ЭТО Уборка
  ВВЕРХ ВПРАВО ВНИЗ
  Туда-сюда // Уборка чайника
  ВНИЗ
  ПОВТОРИ 4 ВЛЕВО
  Туда-сюда // Уборка кружки
КОНЕЦ

ЭТО Туда-сюда
  ПОВТОРИ 8 ВНИЗ
  ВПРАВО
  ПОВТОРИ 9 ВВЕРХ
КОНЕЦ
```

Тесты

Тестирование исчерпывается двумя вариантами надписи на кубике в клетке (5,2): А и не А (например, Б).

5. Новогодние подарки (Николай Истомин, 5 класс, п. Турочак).

   В мешке, образованном кубиками С, у Кукарачи новогодние подарки: машина (кубик М) для Димы (кубик Д) и кукла (кубик К) для Лены (кубик Л). Доставить подарки ребятам (поставить в соответствующие клетки). Начальное и конечное состояние среды показаны на рис. 7.5.

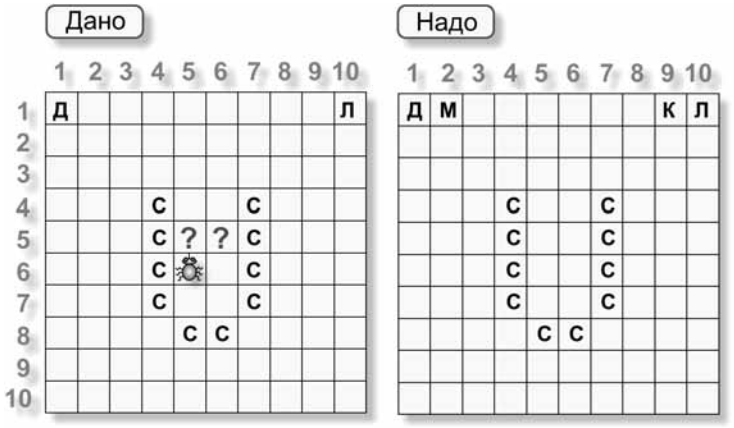

Рис. 7.5

Решение 1

Описание алгоритма. Кукарача толкает кубик. Если это — кукла, несёт её Лене, затем возвращается за машиной и доставляет её Диме. Если машина, то сначала поход к Диме, а затем — к Лене.

Программа

```
// Автор решения: Рудь А. В.
ЭТО Подарки
  ВВЕРХ
  ЕСЛИ К
    ТО     Сначала_Лене
    ИНАЧЕ Сначала_Диме
КОНЕЦ

ЭТО Сначала_Лене
  ПОВТОРИ 3 ВВЕРХ   // Из мешка
  ВЛЕВО ВВЕРХ
  ПОВТОРИ 4 ВПРАВО  // К Лене
  ПОВТОРИ 3 ВЛЕВО   // В мешок за машиной
  ПОВТОРИ 5 ВНИЗ
  ВПРАВО
  ПОВТОРИ 4 ВВЕРХ   // Из мешка
  ВПРАВО ВВЕРХ
  ПОВТОРИ 4 ВЛЕВО   // К Диме
```

```
КОНЕЦ

ЭТО Сначала_Диме
  ПОВТОРИ 3 ВВЕРХ   // Из мешка
  ВПРАВО  ВВЕРХ
  ПОВТОРИ 3 ВЛЕВО   // К Диме
  ПОВТОРИ 2 ВПРАВО  // Обратно в мешок
  ПОВТОРИ 5 ВНИЗ
  ВПРАВО
  ПОВТОРИ 4 ВВЕРХ   // Из мешка
  ВЛЕВО  ВВЕРХ
  ПОВТОРИ 3 ВПРАВО  // К Лене
КОНЕЦ
```

Решение 2

Описание алгоритма. Первый подарок (кубик над Кукарачей) «вынем из мешка» — установим в клетку (1,5). Если вынули куклу (кубик К) — доставим Лене, иначе Диме (процедуры Лене или Диме), причём после «вручения» подарка вернём исполнитель в клетку (1,5). Затем сходим за вторым подарком и установим его в эту же клетку (процедура Вынуть_второй). Второй подарок доставляется с помощью одной из упомянутых процедур Диме или Лене.

Программа

```
// Автор решения: Рудь А. В.
ЭТО Подарки
  ПОВТОРИ 4 ВВЕРХ  // Вынуть подарок
  ЕСЛИ К           // Что вынули?
    ТО    { Лене Вынуть_второй Диме }
    ИНАЧЕ { Диме Вынуть_второй Лене }
КОНЕЦ

ЭТО Лене
  ВЛЕВО ВВЕРХ
  ПОВТОРИ 4 ВПРАВО  // Куклу Лене
  ПОВТОРИ 3 ВЛЕВО   // В клетку (1,5)
КОНЕЦ

ЭТО Вынуть_второй
```

```
  ПОВТОРИ 5 ВНИЗ     // В мешок
  ВПРАВО
  ПОВТОРИ 4 ВВЕРХ    // Из мешка
  ВПРАВО ВВЕРХ
  ВЛЕВО ВНИЗ ВЛЕВО
КОНЕЦ

ЭТО Диме
  ВПРАВО ВВЕРХ
  ПОВТОРИ 3 ВЛЕВО    // Машину Диме
  ПОВТОРИ 2 ВПРАВО   // В клетку (1,5)
КОНЕЦ
```

Тесты

Нужно проверить два варианта надписей на скрытых кубиках:

- МК
- КМ

6. Экран на прежнем месте (Илья Алутин, 9 класс, Благовещенск).

Во второй строке злоумышленник перемешал буквы в слове ЭКРАН. Восстановить испорченное слово на прежнем месте. Начальное и конечное состояния среды показаны на рис. 7.6.

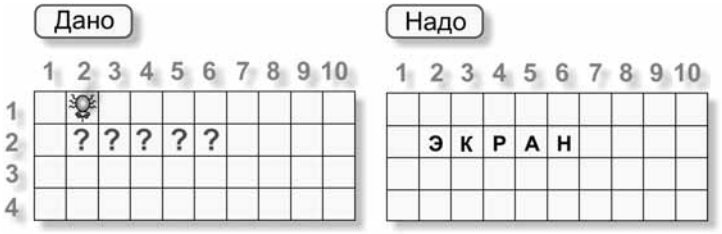

Рис. 7.6

Решение

Описание алгоритма. Идея решения показана на схеме:

```
.КРЭНА.     .......     .Э.....     .Э.....     .ЭКРАН.
.......     ...Э...     .К.....     ..К....     .......
....... -> .К..... -> .Р..... -> ...Р... -> .......
```

```
.......    ..Р....    .А.....    ....А..    .......
.......    .....А.    .Н.....    .....Н.    .......
.......    ....Н..    .......    .......    .......
```

Пояснение. Сдвигаем кубик с буквой Э в 3-ю строку, кубик с буквой К — в 4-ю, с буквой Н — в 7-ю. Теперь сдвигаем все кубики во 2-й столбец. Слово ЭКРАН читается сверху вниз, осталось его «повернуть». Построим из букв «лесенку», затем превратим её в строку.

Программа

```
// Автор решения: А. В. Рудь
ЭТО Экран
  ПОВТОРИ 5 Сдвиг_по_строкам
  Все_во_2_столбец
  Буквы_лесенкой
  Буквы_на_место
КОНЕЦ
//-------------------------------------
ЭТО Сдвиг_по_строкам
  ВНИЗ
  ЕСЛИ       Э ТО Оставить
  ИНАЧЕ ЕСЛИ К ТО На_1_вниз
  ИНАЧЕ ЕСЛИ Р ТО На_2_вниз
  ИНАЧЕ ЕСЛИ А ТО На_3_вниз
  ИНАЧЕ ЕСЛИ Н ТО На_4_вниз
  ВПРАВО
КОНЕЦ
//-------------------------------------
ЭТО Оставить
  ВВЕРХ
КОНЕЦ
//-------------------------------------
ЭТО На_1_вниз
  ВНИЗ
  ПОВТОРИ 2 ВВЕРХ
КОНЕЦ
//-------------------------------------
ЭТО На_2_вниз
  ПОВТОРИ 2 ВНИЗ
  ПОВТОРИ 3 ВВЕРХ
```

```
КОНЕЦ
//-------------------------------------
ЭТО На_3_вниз
  ПОВТОРИ 3 ВНИЗ
  ПОВТОРИ 4 ВВЕРХ
КОНЕЦ
//-------------------------------------
ЭТО На_4_вниз
  ПОВТОРИ 4 ВНИЗ
  ПОВТОРИ 5 ВВЕРХ
КОНЕЦ
//-------------------------------------
ЭТО Все_во_2_столбец
  ПОВТОРИ 2 ВНИЗ
  ПОВТОРИ 4 ВЛЕВО
  ПОВТОРИ 4 Челнок
КОНЕЦ
//-------------------------------------
ЭТО Челнок
  ПОВТОРИ 4 ВПРАВО
  ВНИЗ
  ПОВТОРИ 4 ВЛЕВО
КОНЕЦ
//-------------------------------------
ЭТО Буквы_лесенкой
  // Э на место
  ВНИЗ ВЛЕВО ВВЕРХ
  // Н лесенкой
  ВЛЕВО ВВЕРХ
  ПОВТОРИ 4 ВПРАВО
  ПОВТОРИ 4 ВЛЕВО
  // А лесенкой
  ВВЕРХ
  ПОВТОРИ 3 ВПРАВО
  ПОВТОРИ 3 ВЛЕВО
  // Р лесенкой
  ВВЕРХ
```

```
  ПОВТОРИ 2 ВПРАВО
  ПОВТОРИ 2 ВЛЕВО
  // К лесенкой
  ВВЕРХ
  ВПРАВО
КОНЕЦ
//-----------------------------------
ЭТО Буквы_на_место
  // К на место
  ВНИЗ ВПРАВО ВВЕРХ
  // Р на место
  ПОВТОРИ 2 ВНИЗ
  ВПРАВО
  ПОВТОРИ 2 ВВЕРХ
  // А на место
  ПОВТОРИ 3 ВНИЗ
  ВПРАВО
  ПОВТОРИ 3 ВВЕРХ
  // Н на место
  ПОВТОРИ 4 ВНИЗ
  ВПРАВО
  ПОВТОРИ 4 ВВЕРХ
КОНЕЦ
```

Тесты

Возможно 5! = 120 различных сочетаний из 5 букв. Проверка каждого варианта приведёт к полной проверке программы. Такое тестирование имеет смысл, если неизвестен алгоритм решения (тестирование чёрного ящика).

Так как алгоритм нам известен и он инвариантен к порядку следования букв в записи, то достаточно проверить несколько вариантов (5–6).

Задачи к главе 5

7. Горка (Николай Истомин, 5 класс, п. Турочак).

   Заставьте Кукарачу скатиться с горки (встать на место кубика с буквой К). Высота горки заранее неизвестна. Пример начального и конечного состояния среды показан на рис. 7.7.

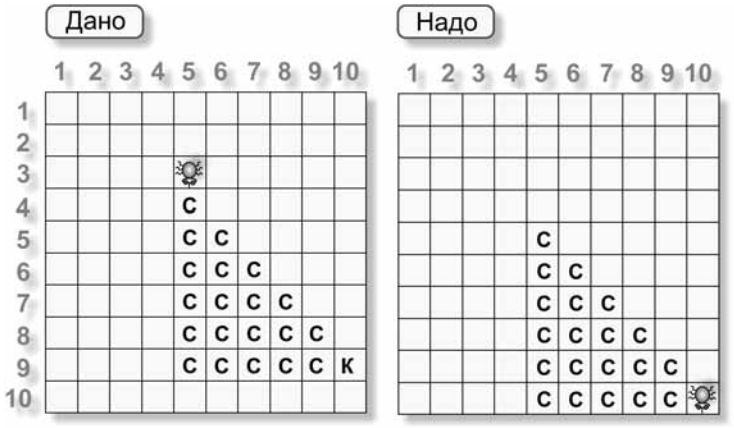

Рис. 7.7

Решение

Алгоритм

1. Переместить Кукарачу на 1 клетку вниз.
2. Пока исполнитель не толкнёт кубик с буквой К, двигать его по «склону» командами **ВПРАВО ВНИЗ**.

Программа

```
// Автор решения: А. В. Рудь
ЭТО Горка
  ВНИЗ
  ПОКА НЕ К Катись
КОНЕЦ

ЭТО Катись
  ВПРАВО ВНИЗ
КОНЕЦ
```

Тесты

- Горка из одного кубика:

```
......¤
.......К
```

- Горка из трёх кубиков:

```
.....¤.
.....С.
.....СК
```

- ❒ Большая горка:

```
....¤..
....C..
....CC.
....CCK
```

8. Какие оценки нужны Кукараче (А. А. Кашников, Кострома).

Исполнитель находится в пятом столбце, ниже плотно стоят кубики с цифрами. Кубики с цифрой 5 нужно оставить, а все остальные убрать с поля. Разрешается сдвинуть цифры 5 на клетку вправо. Пример начального и конечного состояния среды показан на рис. 7.8.

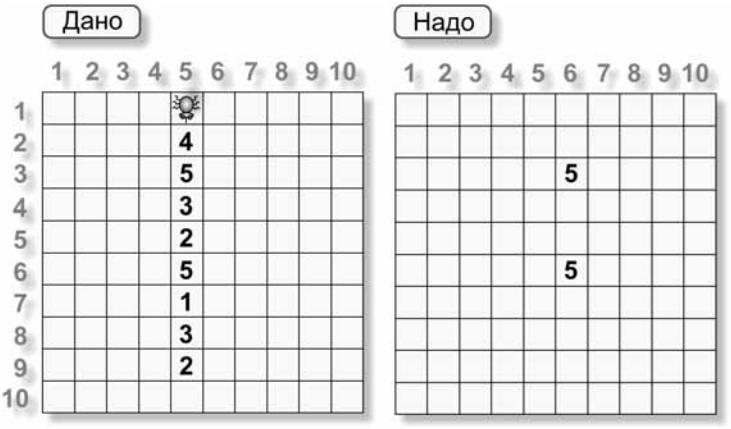

Рис. 7.8

Решение

Алгоритм

1. Толкаем ближайший кубик вправо.
2. Пока есть кубик, выполнять:
   2.1. Если не пятерка — выбрасываем кубик и возвращаемся обратно.
   2.2. Толкаем следующий кубик.

Программа

```
// Автор решения: А. В. Рудь
ЭТО Вход
  ВЛЕВО ВНИЗ ВПРАВО       // Толкаем 1-й кубик
  ПОКА НЕ ПУСТО Проверка // Пока есть кубик выполняем проверку
КОНЕЦ
```

```
// Проверяем кубик и, если на нём не 5, выбрасываем с поля
ЭТО Проверка
  ЕСЛИ НЕ 5 ТО Выброс
  ВЛЕВО ВНИЗ ВПРАВО // К следующему кубику
КОНЕЦ

// Выбрасываем кубик с поля и возвращаемся назад
ЭТО Выброс
  ПОВТОРИ 5 ВПРАВО
  ПОВТОРИ 5 ВЛЕВО
КОНЕЦ
```

Тесты

- ❒ На поле нет кубиков.
- ❒ На поле один кубик.
  1. Кубик 5.
  2. Кубик не 5 (например, 4)
- ❒ На поле много кубиков
  1. Все кубики с цифрой 5.
  2. Нет ни одного кубика с цифрой 5.
  3. Разные комбинации кубиков, среди которых есть кубики с цифрой 5 и кубики с другими надписями.

9. Последний станет первым (В. П. Семенко, Рубцовск).

Кукарача находится в клетке с координатами (4,3). Справа от него плотная строка из кубиков (не менее двух). Переставить последний кубик на первое место. Пример начального и конечного состояний среды показан на рис. 7.9.

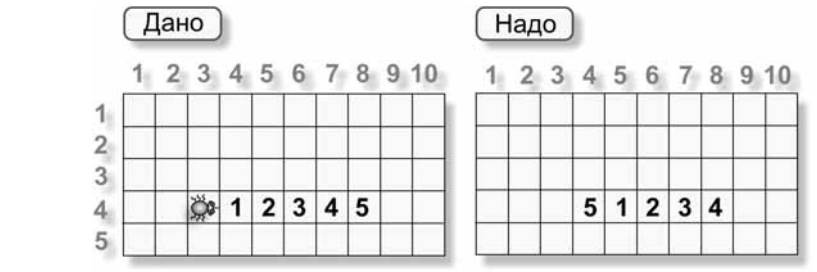

Рис. 7.9

Решение

Алгоритм

1. Сделать шаг вправо вдоль записи и толкнуть снизу вверх первый кубик (процедура `Шаг_вправо`).
2. Пока есть кубики, толкать их вверх и смещаться по записи вправо (процедура `В_конец_записи`).
3. Поставить последний кубик над записью во вторую строку, а исполнителя справа от него (процедура `Подготовка`).
4. Толкать кубики записи вниз, а последний кубик влево, пока запись не закончится (процедура `В_начало_записи`).
5. Поставить последний кубик на первое место в записи (процедура `Последний_на_место`).

Программа

```
// Автор решения: А. В. Рудь
ЭТО Вход
  Шаг_вправо
  В_конец_записи
  Подготовка
  В_начало_записи
  Последний_на_место
КОНЕЦ

// Пока есть кубики, толкать их вверх и смещаться по записи вправо
// ---------------------------------------------------------------
ЭТО В_конец_записи
  ПОКА НЕ ПУСТО  Шаг_вправо
КОНЕЦ

// Поставить последний кубик над записью во вторую
// строку, а исполнителя справа от него
// ---------------------------------------------
ЭТО Подготовка
  ВЛЕВО
  ВВЕРХ
  ВПРАВО
  ВВЕРХ
```

```
  ПОВТОРИ 2 ВЛЕВО
КОНЕЦ

// Толкать кубики записи вниз, а последний кубик
// влево, пока запись не закончится
// ---------------------------------------------
ЭТО В_начало_записи
  ВНИЗ
  ПОКА НЕ ПУСТО Шаг_влево
КОНЕЦ

// Поставить последний кубик на первое место в записи
// --------------------------------------------------
ЭТО Последний_на_место
  ПОВТОРИ 2 ВВЕРХ
  ВЛЕВО
  ПОВТОРИ 2 ВНИЗ
  ВЛЕВО ВНИЗ ВПРАВО
КОНЕЦ

// Шагнуть вправо вдоль записи, одновременно толкнув кубик вверх
// -------------------------------------------------------------
ЭТО Шаг_вправо
  ВНИЗ ВПРАВО ВВЕРХ
КОНЕЦ

// Шагнуть влево вдоль записи, одновременно толкнув кубик вниз
// -----------------------------------------------------------
ЭТО Шаг_влево
  ВВЕРХ ВЛЕВО ВНИЗ
КОНЕЦ
```

Замечание

Кукарача идёт вправо вдоль записи «налегке». Задача будет более интересной, если предложить в условии обменять местами первый и последний кубики записи. Тогда исполнитель при движении и вправо, и влево должен будет тащить за собой кубик.

Тесты

- ❒ На поле 2 кубика.
- ❒ На поле 3 кубика.
- ❒ На поле много кубиков (например, 5).

10. Грибы (Кирилл Харин, 9 класс, Качканар).

Справа от Кукарачи, расположенного в клетке (4,1), выросло некоторое количество грибов (от 0 до 99). Повесьте белые грибы (кубики Б) в первую строку на верёвку, а остальные кубики передвиньте за пределы верёвки. Пример начального и конечного состояний среды показан на рис. 7.10.

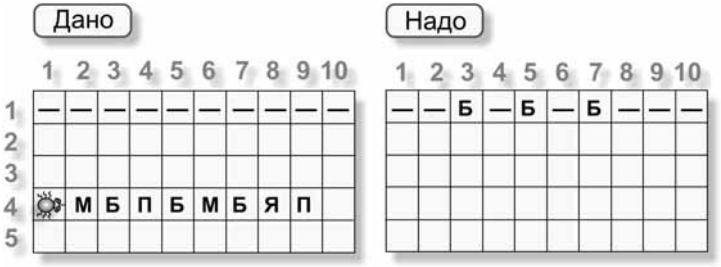

Рис. 7.10

Решение

Описание алгоритма. Белые грибы будем вешать на верёвку, а другие сдвигать вправо по третьей строке за пределы верёвки.

Алгоритм

1. Проверим ближайший к Кукараче гриб, толкнув кубик вверх.
2. Пока грибы не кончились, выполнять:
   2.1. Если гриб нехороший (не Б), продвинем его на 2 клетки вправо в третьей строке, иначе повесим на верёвку.
   2.2. Перейдём к следующему грибу.

Программа

```
// Автор решения: А. В. Рудь
ЭТО Вход
  Шаг
  ПОКА НЕ ПУСТО Ищи
КОНЕЦ
```

```
ЭТО Ищи
  ЕСЛИ НЕ Б
   ТО    Сдвигай
   ИНАЧЕ Суши
  Шаг
КОНЕЦ

ЭТО Сдвигай
  ВЛЕВО ВВЕРХ
  ПОВТОРИ 2 ВПРАВО
  ВЛЕВО ВНИЗ
КОНЕЦ

ЭТО Суши
  ПОВТОРИ 2 ВВЕРХ
  ВНИЗ
  ВПРАВО ВЛЕВО // Сдвиг плохих грибов
  ВНИЗ
КОНЕЦ

ЭТО Шаг
  ВНИЗ ВПРАВО ВВЕРХ
КОНЕЦ
```

Тесты

- ❒ На поле нет грибов.
- ❒ На поле 1 гриб:
  1. Б;
  2. П.
- ❒ На поле много грибов:
  1. Все кубики с буквой Б.
  2. Нет ни одного кубика с буквой Б.
  3. Разные комбинации надписей.

11. Грибы-2 (А. В. Рудь, Снежинск).

    В лесу, кроме белых, растут другие хорошие грибы: **л**исички (Л), подо**с**иновики (С) и **о**пята (О). Их тоже надо посушить на зиму. Пример начального и конечного состояний среды показан на рис. 7.11.

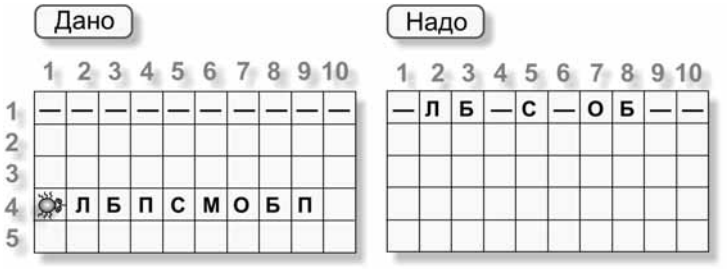

Рис. 7.11

Решение

Описание алгоритма. Отличие от алгоритма предыдущей задачи только в способе выбора грибов: в процедуре Ищи потребуется запрограммировать переключатель:

```
ЭТО Ищи    // Проверка букв
  ЕСЛИ        Б   ТО Суши
  ИНАЧЕ ЕСЛИ Л   ТО Суши
  ИНАЧЕ ЕСЛИ О   ТО Суши
  ИНАЧЕ ЕСЛИ С   ТО Суши
  ИНАЧЕ              Сдвигай
  Шаг
КОНЕЦ
```

Программа. В программе для предыдущей задачи потребуется только заменить процедуру Ищи.

Тесты

- ❒ На поле нет грибов.
- ❒ На поле 1 гриб (Б, Л, О, С, П).
- ❒ На поле много грибов:
  1. Все кубики с хорошими грибами.
  2. Нет ни одного кубика с хорошими грибами.
  3. Разные комбинации надписей.

Задачи к главе 6

12. Грибы-3 (А. В. Рудь, Снежинск).

Когда Кукарача в предыдущей задаче развесил грибы, то огорчился: «Где густо, а где пусто!» Как повесить съедобные грибы на верёвке плотно? Пример начального и конечного состояний среды показан на рис. 7.12.

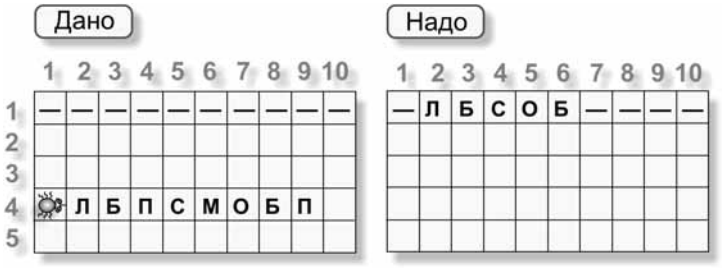

Рис. 7.12

Решение

Описание алгоритма. Сначала рассортируем грибы: съедобные — во вторую строку, несъедобные — как и в предыдущих задачах, в третью, для постепенного удаления за пределы верёвки.

Чтобы удалить пустоты между грибами, надо знать количество выброшенных несъедобных грибов, значит, не обойтись без рекурсии и отложенной команды (рекурсивной пружинки), которую нужно поместить в процедуру `Сдвигай`. Рекурсивная пружинка сработает столько раз, сколько раз вызывалась процедура `Сдвигай`, т. е. столько, сколько было несъедобных грибов, создавших пустоты между съедобными грибами.

Программа

```
// Автор решения: А. В. Рудь
// Сначала сортируем грибы (несъедобные в третью строку
// со сдвигом вправо, съедобные во вторую строку без сдвига)
// и уплотняем съедобные. Затем сдвигаем съедобные грибы на
// верёвку в первую строку.
// -------------------------------------------------------
ЭТО Вход
  Сортируй_и_уплотни
  Грибы_на_веревку
КОНЕЦ
```

```
// Косвенная рекурсия:
// Сортируй_и_уплотни -> Ищи -> Суши -> Сортируй_и_уплотни
// Сортируй_и_уплотни -> Ищи -> Сдвигай -> Сортируй_и_уплотни
// Последняя рекурсивная цепочка содержит рекурсивную пружинку
// в процедуре Сдвигай.
// Процедура Подготовься_к_уплотнению будет работать
// после окончания рекурсии, но перед действием рекурсивной
// пружинки. В ней исполнитель устанавливается во вторую строку,
// чтобы рекурсивная пружинка работала на уплотнение строки
// со съедобными грибами.
// -------------------------------------------------------------
ЭТО Сортируй_и_уплотни
  Шаг_вправо
  ЕСЛИ НЕ ПУСТО
    ТО Ищи
    ИНАЧЕ Подготовься_к_уплотнению
КОНЕЦ

ЭТО Ищи
  ЕСЛИ       Б ТО Суши
  ИНАЧЕ ЕСЛИ Л ТО Суши
  ИНАЧЕ ЕСЛИ О ТО Суши
  ИНАЧЕ ЕСЛИ С ТО Суши
  ИНАЧЕ          Сдвигай
КОНЕЦ

ЭТО Суши
  ВВЕРХ ВПРАВО ВЛЕВО ВНИЗ
  Сортируй_и_уплотни // Замыкание рекурсии
КОНЕЦ

ЭТО Сдвигай
  ВЛЕВО ВВЕРХ ВПРАВО ВПРАВО ВЛЕВО ВНИЗ
  Сортируй_и_уплотни // Замыкание рекурсии
  Уплотни            // Рекурсивная пружинка
КОНЕЦ
```

```
ЭТО Уплотни
  ВЛЕВО
КОНЕЦ

ЭТО Подготовься_к_уплотнению
  ВВЕРХ ВВЕРХ
КОНЕЦ

ЭТО Грибы_на_веревку
  Шаг_влево
  ПОКА НЕ ПУСТО Шаг_влево
КОНЕЦ

ЭТО Шаг_вправо
  ВНИЗ ВПРАВО ВВЕРХ
КОНЕЦ

ЭТО Шаг_влево
  ВНИЗ ВЛЕВО ВВЕРХ
КОНЕЦ
```

Тесты

- ❒ На поле нет грибов.
- ❒ На поле 1 гриб (Б, Л, О, С, П).
- ❒ На поле много грибов:
  1. Все кубики с хорошими грибами.
  2. Нет ни одного кубика с хорошими грибами.
  3. Разные комбинации надписей.

13. Циклическая перестановка (В. П. Семенко, Рубцовск).

Кукарача стоит в клетке (5,1). Справа от него две последовательности кубиков, разделённые между собой пустой клеткой. Выполнить циклическую перестановку вправо кубиков во второй последовательности на число, равное количеству кубиков в первой последовательности. Пример начального и конечного состояния среды показан на рис. 7.13.

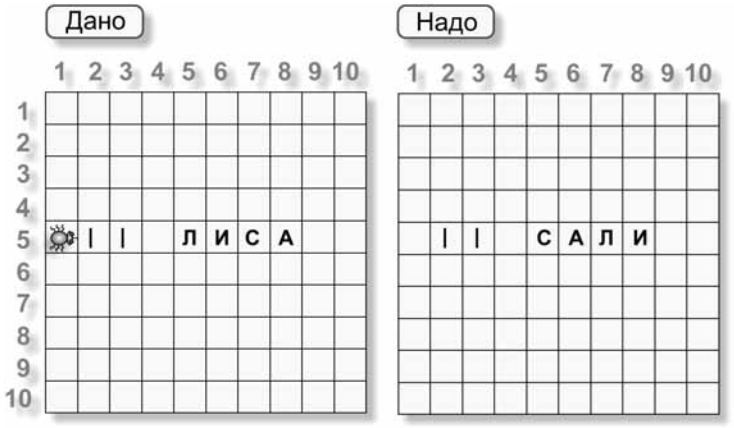

Рис. 7.13

Решение

Алгоритм

1. Рекурсивно подсчитать кубики первой последовательности (процедура `Вход`).
2. Во второй последовательности рекурсивной пружинкой ставить последний кубик на первое место столько раз, сколько было кубиков в первой последовательности (процедура `Сдвиг`).

Программа

```
ЭТО Вход
  // Начальное состояние среды:
  // ¤||.ЛИСА.  (5 строка)
  Вход1  // Циклический сдвиг
  // Состояние среды после выполнения процедуры Вход1:
  // .||.....  (4 строка)
  // ...¤САЛИ.
  // Сместить кубики первой последовательности в исходную строку
  ВВЕРХ
  Шаг_влево
  ПОКА НЕ ПУСТО Шаг_влево
  ВНИЗ
  // Теперь состояние среды:
  // ¤||.САЛИ.  (5 строка)
КОНЕЦ
```

```
// Рекурсивно идём вправо, пока кубики в первой
// последовательности не закончатся.
// Затем начинает работать рекурсивная пружинка Сдвиг.
// Эта процедура переставляет (один раз) последний кубик
// во второй последовательности на первое место.
// Состояние среды перед выполнением процедуры Вход1:
// ¤||.ЛИСА.  (5 строка)
// Состояние среды после выполнения процедуры Вход1:
// .||......  (4 строка)
// ...¤САЛИ.
// -----------------------------------------------------
ЭТО Вход1
  Шаг_вправо
  ЕСЛИ НЕ ПУСТО
    ТО
    {
      Вход1
      Сдвиг // Рекурсивная пружинка
    }
КОНЕЦ

// Циклический сдвиг второй последовательности на один кубик.
// Состояние среды перед выполнением процедуры Сдвиг:
// .||......
// ...¤ЛИСА.
// Состояние среды после выполнения процедуры Сдвиг:
// ..........
// .||.......
// ...¤АЛИС..
// ------------------------------------------------------------
ЭТО Сдвиг
  // Сдвинуть всё слово вправо (освобождение первого места
  // в записи для последнего кубика)
  ВПРАВО
  // Рекурсивно найти последний кубик и рекурсивной
  // пружинкой поставить его над первым местом.
  Сдвиг1
  // Состояние среды после выполнения процедуры Сдвиг1:
```

```
  // ....А¤....
  // .||.......
  // .....ЛИС..
  // Установить кубик на первое место
  ВВЕРХ ВЛЕВО ВНИЗ ВНИЗ
  // Теперь состояние среды:
  // ..........
  // .||.¤.....
  // ....АЛИС..
  // Установить исполнителя в исходное положение
  // перед выполнением нового сдвига
  ВЛЕВО ВНИЗ
КОНЕЦ

// Рекурсивный поиск последнего кубика и установка
// его рекурсивной пружиной над первым местом.
// Процедура Подготовка готовит среду к работе
// рекурсивной пружины:
// - смещает последний кубик на строку выше;
// - устанавливает исполнителя справа от кубика.
// -------------------------------------------------
ЭТО Сдвиг1
  Шаг_вправо
  ЕСЛИ НЕ ПУСТО
    ТО
    {
      Сдвиг1
      // Рекурсивная пружинка
      ВЛЕВО      // Смещение кубика влево
      ВНИЗ ВВЕРХ // Смещение записи вниз
    }
    ИНАЧЕ Подготовка
КОНЕЦ

// Процедура Подготовка готовит среду к работе
// рекурсивной пружины:
// - смещает последний кубик на строку выше;
// - устанавливает исполнителя справа от кубика
```

```
// -------------------------------------------------
ЭТО Подготовка
  ВЛЕВО ВВЕРХ  // Последний кубик: на строку выше
  ВПРАВО ВВЕРХ // Исполнитель: справа от кубика
КОНЕЦ

ЭТО Шаг_вправо
  ВНИЗ ВПРАВО ВВЕРХ
КОНЕЦ

ЭТО Шаг_влево
  ВВЕРХ ВЛЕВО ВНИЗ
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 7.3

Таблица 7.3

Первая последовательность	Вторая последовательность	Ожидаемый результат
Нет кубиков	ЛИСА	ЛИСА
|	Нет кубиков	
|	Л	Л
|	ОН	НО
|	ЛИСА	АЛИС
||	Нет кубиков	
||	Л	Л
||	ОН	ОН
||	ЛИСА	САЛИ
|||	Нет кубиков	
|||	Л	Л
|||	ОН	НО
|||	ЛИСА	ИСАЛ

Задачи недели 2003/2004 учебного года

Задачи к главам 1–3

14. Провернувшийся циферблат (В. П. Семенко, Рубцовск).

Кубики с числами 1, 2, 3, 4, 5, 6, 7, 8, 9, A (число 10), B (число 11), C (число 12) образуют на поле циферблат часов. Циферблат провернулся, и Кукарача решил починить часы. Напишите нужную программу. Начальное и конечное положение среды показано на рис. 7.14.

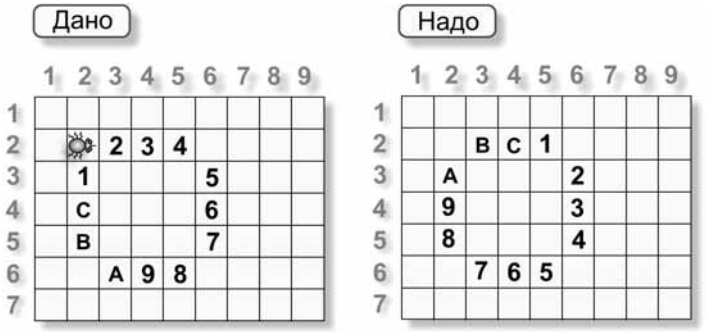

Рис. 7.14

Решение

Описание алгоритма. Циферблат провернулся на три часа против часовой стрелки, значит, необходимо выполнить его поворот на три часа по часовой стрелке. Для этого достаточно написать процедуру, отвечающую за поворот на один час и повторить её три раза.

Программа

```
// Автор решения: В. П. Семенко
ЭТО Вход
  ПОВТОРИ 3 Поворот
КОНЕЦ

// Поворот циферблата на 1 час по часовой стрелке
// ---------------------------------------------
ЭТО Поворот
  ВПРАВО ВВЕРХ
  ПОВТОРИ 3 ВПРАВО
```

```
  ВНИЗ ВПРАВО
  ПОВТОРИ 4 ВНИЗ
  ВЛЕВО ВНИЗ
  ПОВТОРИ 4 ВЛЕВО
  ВВЕРХ ВЛЕВО
  ПОВТОРИ 4 ВВЕРХ
  ВПРАВО
КОНЕЦ
```

Комментарий. Так как задача ориентирована на закрепление материала глав 1–3, то в условии указан конкретный поворот циферблата на 3 часа. Интересно решить эту задачу для произвольного исходного поворота циферблата (после изучения цикла **ПОКА** в *главе 5*).

Новое решение может иметь вид:

```
ЭТО Вход
  ВПРАВО              // Толкнём верхнюю часть циферблата
  ПОКА НЕ В Поворот // Пока это не В, выполняем поворот циферблата
  Возврат             // Когда толкнули В, возвращаем верхнюю часть на
КОНЕЦ                 // прежнее место.


ЭТО Поворот
  ВВЕРХ
  ПОВТОРИ 3 ВПРАВО
  ВНИЗ ВПРАВО
  ПОВТОРИ 4 ВНИЗ
  ВЛЕВО ВНИЗ
  ПОВТОРИ 4 ВЛЕВО
  ВВЕРХ ВЛЕВО
  ПОВТОРИ 4 ВВЕРХ
  ПОВТОРИ 2 ВПРАВО
КОНЕЦ


// Возврат верхней части циферблата на прежнее место.
// Состояние среды перед выполнением процедуры:
// ..¤BC1.
// .A...2.
// -------------------------------------------------
ЭТО Возврат
```

```
    ВВЕРХ
    ПОВТОРИ 4 ВПРАВО
    ВНИЗ ВЛЕВО
КОНЕЦ
```

Дальнейшее усложнение задачи: исходный поворот циферблата в произвольную (неизвестную заранее) сторону. Программа должна сама определить направление поворота.

Тесты

Для проверки программы в исходной формулировке (поворот на 3 часа) достаточно выполнить один запуск с состоянием среды, описанным в условии.

Для проверки решения задачи о произвольном повороте нужны такие тесты:

- циферблат провернулся на один час;
- циферблат провернулся на более чем один час (2, 3, 11);
- циферблат не проворачивался.

15. Цифровая головоломка (В. П. Семенко, Рубцовск).

Кукарача находится в клетке (5,1). Справа от него, начиная с клетки (5,2), расположен плотный ряд кубиков с чётными цифрами от 0 до 8. Под чётными цифрами, в шестом ряду, расположены кубики с нечётными цифрами от 1 до 9 (рис. 7.15). Помогите Кукараче выстроить цифры в горизонтальный ряд в порядке возрастания.

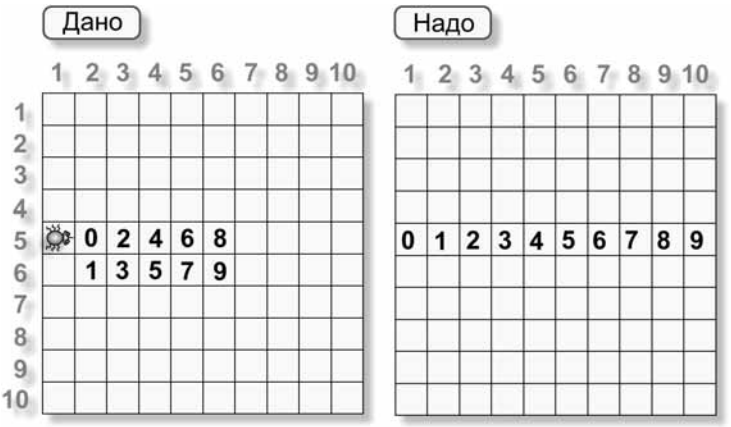

Рис. 7.15

Решение

Описание алгоритма. Будем формировать результирующую последовательность, дополняя её в цикле парами кубиков.

Начальная установка:

¤02468 (исполнитель перед первым кубиком первой пары)

.13579

После первого оборота цикла:

01¤2468 (исполнитель перед первым кубиком второй пары)

...3579

После второго оборота цикла:

0123¤468 (исполнитель перед первым кубиком третьей пары)

.....579

...

После пятого оборота цикла:

0123456789¤ (исполнитель перед первым кубиком шестой пары, которой, правда, нет)

Таким образом, алгоритм решения имеет вид:

1. Повторить 5 раз:
   1.1. Дополнить результат новой парой кубиков.
   1.2. Установить исполнителя перед первым кубиком очередной пары.

Программа

```
ЭТО Вход
  ПОВТОРИ 5
  {
    Установи_пару
    Встань_в_ряд
  }
КОНЕЦ

// Установка пары цифр в 5 строку
// ------------------------------
ЭТО Установи_пару
  ПОВТОРИ 2 { ВНИЗ ВПРАВО }
  ПОВТОРИ 2 ВВЕРХ
  ВЛЕВО
```

```
  ПОВТОРИ 2 ВПРАВО
  ПОВТОРИ 2 { ВВЕРХ ВЛЕВО }
  ВНИЗ
  ВВЕРХ
КОНЕЦ

// Возвращение в верхний ряд кубика очередной пары
// и установка исполнителя слева от него
// ---------------------------------------------
ЭТО Встань_в_ряд
  ПОВТОРИ 2 { ВВЕРХ ВПРАВО }
  ПОВТОРИ 3 ВНИЗ
  ВЛЕВО
  ВНИЗ
КОНЕЦ
```

Комментарий. Это решение легко модифицируется для произвольного числа пар кубиков (с помощью цикла **ПОКА**).

Замечание

Для работы программы нужно поле шириной в 12 столбцов. Можно написать решение, в котором исполнителю не придётся выходить за пределы стандартного поля 10×10, но тогда последние цифры нужно обрабатывать отдельно:

```
// Автор решения: В. П. Семенко
ЭТО Вход
  ПОВТОРИ 3
  {
    Установи_пару
    Встань_в_ряд
  }
  Установи_пару
  Последняя_пара
КОНЕЦ

// Особая установка последней пары цифр на поле 10×10
// --------------------------------------------------
ЭТО Последняя_пара
  ПОВТОРИ 2 ВВЕРХ
```

```
    ВПРАВО
    ПОВТОРИ 5 ВНИЗ
    ВЛЕВО
    ПОВТОРИ 2 ВНИЗ
    ВПРАВО
    ПОВТОРИ 2 ВВЕРХ
    ВНИЗ ВПРАВО ВВЕРХ
КОНЕЦ
```

Тесты

Для проверки программы достаточно выполнить один запуск с состоянием среды, описанным в условии.

16. Опять пятёрка (Михаил Суворов, 8 класс, Тверь).

На турнире исполнителей Кукараче поставили 5, но злоумышленник испортил оценку. Помогите восстановить справедливость. Начальное и конечное состояния среды показаны на рис. 7.16.

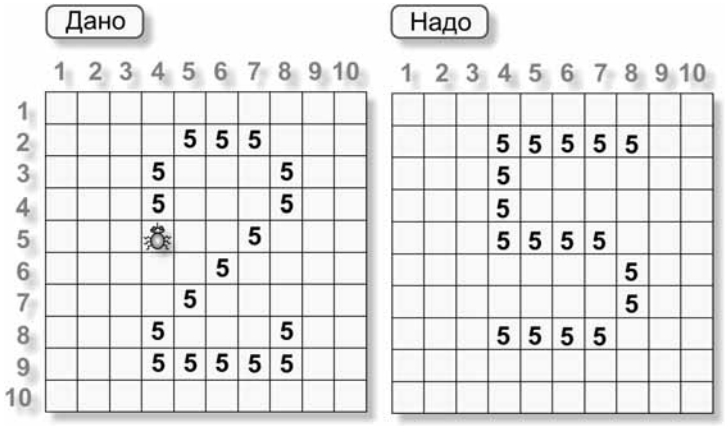

Рис. 7.16

Решение

Алгоритм

1. Построить верхнюю горизонталь.
2. Построить левую вертикаль.
3. Построить среднюю горизонталь.
4. Построить нижнюю горизонталь.
5. Построить правую вертикаль.

Программа

```
// Автор решения: В. П. Семенко

ЭТО Вход
  Верхняя_горизонталь
  Левая_вертикаль
  Средняя_горизонталь
  Нижняя_горизонталь
  Правая_вертикаль
КОНЕЦ

// Вход:         Выход:
// 2|....555...  2|...55555..
// 3|...5...5..  3|...5...¤..
// 4|...5...5..  4|........5.
// 5|...¤..5...  5|......5...
// 6|.....5....  6|.....5....
// 7|....5.....  7|....5.....
// 8|...5...5..  8|...5...5..
// 9|...55555..  9|...55555..
ЭТО Верхняя_горизонталь
  ВВЕРХ
  ПОВТОРИ 4 ВПРАВО
  ВВЕРХ
КОНЕЦ

// Вход:         Выход:
// 2|...55555..  2|...55555..
// 3|...5...¤..  3|...5......
// 4|........5.  4|...5¤.....
// 5|......5...  5|......5...
// 6|.....5....  6|.....5....
// 7|....5.....  7|....5.....
// 8|...5...5..  8|...5...5..
// 9|...55555..  9|...55555..
```

```
ЭТО Левая_вертикаль
  ПОВТОРИ 2 ВПРАВО
  ВНИЗ
  ПОВТОРИ 5 ВЛЕВО
КОНЕЦ

// Вход:         Выход:
// 2|...55555.. 2|...55555..
// 3|...5...... 3|...5......
// 4|...5¤..... 4|...5......
// 5|......5... 5|...5555...
// 6|.....5.... 6|..........
// 7|....5..... 7|..........
// 8|...5...5.. 8|.......5..
// 9|...55555.. 9|....55555¤
ЭТО Средняя_горизонталь
  ПОВТОРИ 3 ВПРАВО
  ВНИЗ
  ПОВТОРИ 3 ВЛЕВО
  ПОВТОРИ 2 { ВНИЗ ВЛЕВО }
  ПОВТОРИ 2 ВНИЗ
  ВПРАВО
  ПОВТОРИ 2 ВВЕРХ
  Шаг_вправо
  ПОВТОРИ 2 ВЛЕВО
  ВВЕРХ ВПРАВО
  ПОВТОРИ 3 Шаг_вправо
  ПОВТОРИ 3 { ВПРАВО ВНИЗ }
КОНЕЦ

// Вход:         Выход:
// 2|...55555.. 2|...55555..
// 3|...5...... 3|...5......
// 4|...5...... 4|...5......
// 5|...5555... 5|...5555...
// 6|.......... 6|..........
```

```
// 7|.......... 7|.......5..
// 8|.......5.. 8|...55555..
// 9|....55555¤ 9|...¤......
ЭТО Нижняя_горизонталь
  ВЛЕВО
  ПОВТОРИ 5 Шаг_влево
КОНЕЦ

// Вход:         Выход:
// 2|...55555.. 2|...55555..
// 3|...5...... 3|...5......
// 4|...5...... 4|...5......
// 5|...5555... 5|...5555...
// 6|.......... 6|.......5..
// 7|.......5.. 7|.......5..
// 8|...55555.. 8|...5555¤..
// 9|...¤...... 9|..........
ЭТО Правая_вертикаль
  ПОВТОРИ 4 ВПРАВО
  ВВЕРХ
КОНЕЦ

ЭТО Шаг_вправо
  ВНИЗ ВПРАВО ВВЕРХ
КОНЕЦ

ЭТО Шаг_влево
  ВНИЗ ВЛЕВО ВВЕРХ
КОНЕЦ
```

Тесты

Для проверки программы достаточно выполнить один запуск с состоянием среды, описанным в условии.

17. Крот (Н. В. Табашин, Лукоянов).

Помогите Кукараче составить слово КРОТ. Заграждения (кубики с буквой Ш) двигать нельзя. Начальное и конечное состояния среды показаны на рис. 7.17.

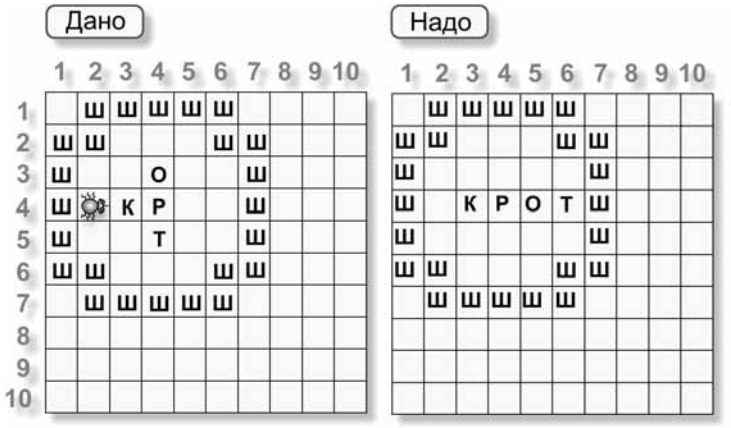

Рис. 7.17

Решение

Алгоритм

1. Установить О в клетку (3,5).
2. Установить Т в клетку (4,5).
3. Установить Р в клетку (3,4) и поставить Т на место.
4. Установить РО на место.

Программа

```
// Автор решения: Дмитрий Ерилин (8 класс, ст. Новохопёрск)

ЭТО Вход
  О_в_3_5
  Т_в_4_5
  Р_в_3_4_Т_на_место
  РО_на_место
КОНЕЦ

// Изменение среды процедурой:
// ...О..      ...¤О.
// .¤КР.. --> ..КР..
// ...Т..      ...Т..
// ----------------------------------------
ЭТО О_в_3_5
  ВВЕРХ
```

```
  ПОВТОРИ 2 ВПРАВО
КОНЕЦ

// Изменение среды процедурой:
// ...¤О.      ....О.
// ..КР.. --> ..КРТ.
// ...Т..      ....¤.
// -----------------------------------------
ЭТО Т_в_4_5
  ПОВТОРИ 2 ВЛЕВО
  ПОВТОРИ 2 ВНИЗ
  ПОВТОРИ 2 ВПРАВО
  ВНИЗ ВПРАВО ВВЕРХ
КОНЕЦ

// Изменение среды процедурой:
// ....О.      ..¤РО.
// ..КРТ. --> ..К..Т
// ....¤.      ......
// -----------------------------------------
ЭТО Р_в_3_4_Т_на_место
  ВЛЕВО ВВЕРХ ВПРАВО
  ВНИЗ
  ПОВТОРИ 3 ВЛЕВО
  ПОВТОРИ 2 ВВЕРХ
  ВПРАВО
КОНЕЦ

// Изменение среды процедурой:
// ..¤РО.      ....¤.
// ..К..Т --> ..КРОТ
// ......      ......
// -----------------------------------------
ЭТО РО_на_место
  ПОВТОРИ 2 { ВВЕРХ ВПРАВО ВНИЗ }
КОНЕЦ
```

Тесты

Для проверки программы достаточно выполнить один запуск с состоянием среды, описанным в условии.

18. Квадрат (Роман Ченцов, 7 класс, Рубцовск).

Квадрат образован цифрами 1, 2, 3, 4, 5, 6, 7, 8. Провернуть цифры на одну позицию по часовой стрелке. Начальное и конечное состояния среды показаны на рис. 7.18.

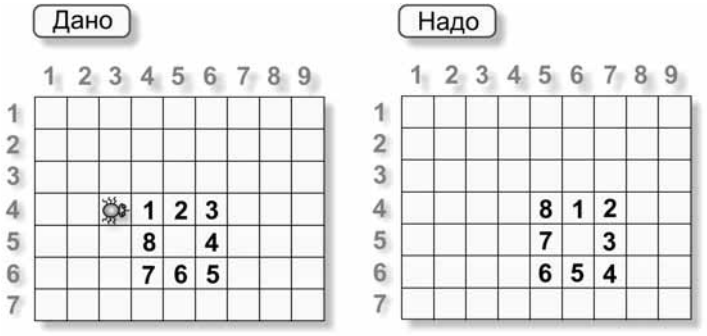

Рис. 7.18

Решение

Алгоритм

В этой простой задаче, как и в предыдущей, приходится внимательно кодировать простые перемещения исполнителя. Число шагов (время выполнения программы) зависит от очередности установки цифр.

1. Установить на место цифры 1, 2, 8 и 7.
2. Установить на место цифры 6 и 5.
3. Установить на место цифры 3 и 4.

Программа

```
// Автор решения: В. П. Семенко

ЭТО Вход
  На_место_1,2,8,7
  На_место_6,5
  На_место_3,4
КОНЕЦ
```

```
// Изменение среды процедурой:
// .¤123..     ..8123.
// ..8.4.. --> ..7¤4..
// ..765..     ....65.
// ---------------------------
ЭТО На_место_1,2,8,7
  ВПРАВО ВЛЕВО
  ПОВТОРИ 3 ВНИЗ
  ПОВТОРИ 2 { ВПРАВО ВВЕРХ }
КОНЕЦ

// Изменение среды процедурой:
// ..8123.     ..8123.
// ..7¤4.. --> ..7..4.
// ....65.     ..65¤.
// ---------------------------
ЭТО На_место_6,5
  ВПРАВО ВЛЕВО
  ПОВТОРИ 2 ВНИЗ
  ПОВТОРИ 3 ВПРАВО
  ВВЕРХ
  ПОВТОРИ 2 ВЛЕВО
КОНЕЦ

// Изменение среды процедурой:
// ..8123.     ..812..
// ..7..4. --> ..7.3..
// ..65¤.      ..654¤.
// ---------------------------
ЭТО На_место_3,4
  ПОВТОРИ 2 ВПРАВО
  ПОВТОРИ 3 ВВЕРХ
  ВЛЕВО ВНИЗ
  ПОВТОРИ 2 { ВПРАВО ВНИЗ ВЛЕВО }
КОНЕЦ
```

Тесты

Для проверки программы достаточно выполнить один запуск с состоянием среды, описанным в условии.

Задачи к главам 4–5

19. Камень, ножницы, бумага (В. П. Семенко, Рубцовск).

В третьем и пятом столбцах, начиная со второй строки, расположены плотные ряды кубиков неизвестной, но равной длины (поле считается бесконечным вниз). На каждом кубике написана одна из трёх букв: К (камень), Н (ножницы), Б (бумага). Кукарача проверяет пары, играя в такую игру:

- К тупит Н (Н удаляется с поля, К ставится в первый столбец);
- Н режут Б (Б удаляется с поля, Н ставится в первый столбец);
- Б накрывает К (К удаляется с поля, Б ставится в первый столбец);
- если кубики в паре одинаковые, то они остаются на поле в первом и втором столбце.

Напишите программу для этой игры. Начальное положение исполнителя — клетка (1,3). Пример начального и конечного состояния среды показан на рис. 7.19.

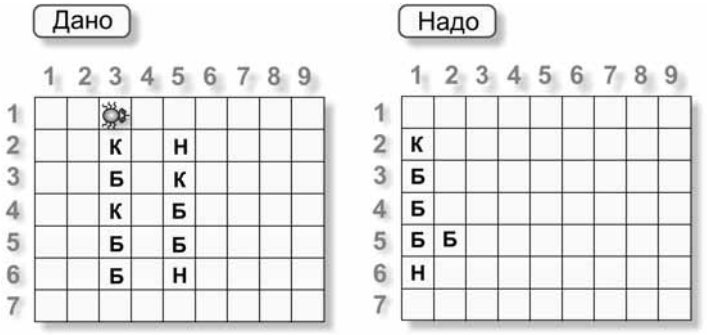

Рис. 7.19

Решение

Алгоритм

1. Шаг в первую строку с кубиками.
2. Пока кубики не закончатся, делать:
   2.1. Проверить, какой кубик «главнее». Главный кубик поставить в первый столбец, а второй удалить с поля. Если кубики одинаковые, то поставить их в первые два столбца.
   2.2. Шаг в следующую строку.

Программа

```
// Автор решения: В. П. Семенко

ЭТО Вход
  Шаг
  ПОКА НЕ ПУСТО
  {
    Проверка
    Шаг
  }
КОНЕЦ

// Толкнуть кубик, расположенный слева строкой ниже
// ------------------------------------------------
ЭТО Шаг
  ВПРАВО ВНИЗ ВЛЕВО
КОНЕЦ

// Проверяем, какая буква на левом кубике
// --------------------------------------
ЭТО Проверка
  ЕСЛИ       К ТО Проверка_для_К
  ИНАЧЕ ЕСЛИ Н ТО Проверка_для_Н
  ИНАЧЕ ЕСЛИ Б ТО Проверка_для_Б
КОНЕЦ

// Слева буква К. Проверим кубик справа
// ------------------------------------
ЭТО Проверка_для_К
  Что_справа
  ЕСЛИ       К ТО Сомкни
  ИНАЧЕ ЕСЛИ Н ТО Удалить_правый
  ИНАЧЕ ЕСЛИ Б ТО Удалить_левый
КОНЕЦ

// Слева буква Н. Проверим кубик справа
// ------------------------------------
```

```
ЭТО Проверка_для_Н
  Что_справа
  ЕСЛИ       Н ТО Сомкни
  ИНАЧЕ ЕСЛИ Б ТО Удалить_правый
  ИНАЧЕ ЕСЛИ К ТО Удалить_левый
КОНЕЦ

// Слева буква Б. Проверим кубик справа
// -------------------------------------
ЭТО Проверка_для_Б
  Что_справа
  ЕСЛИ       Б ТО Сомкни
  ИНАЧЕ ЕСЛИ К ТО Удалить_правый
  ИНАЧЕ ЕСЛИ Н ТО Удалить_левый
КОНЕЦ

// Толкнуть кубик справа
// ---------------------
ЭТО Что_справа
  ПОВТОРИ 2 ВПРАВО
КОНЕЦ

// На кубиках одинаковые буквы. Они обе остаются на поле
// -----------------------------------------------------
ЭТО Сомкни
  ВВЕРХ
  ПОВТОРИ 2 ВПРАВО
  ВНИЗ
  ПОВТОРИ 4 ВЛЕВО
КОНЕЦ

// Левый кубик удаляем, правый - в первый столбец
// ----------------------------------------------
ЭТО Удалить_левый
  Сомкни
  ВЛЕВО ВПРАВО
КОНЕЦ
```

```
// Правый кубик удаляем, левый - в первый столбец
// ---------------------------------------------
ЭТО Удалить_правый
  ПОВТОРИ 5 ВПРАВО
  ПОВТОРИ 8 ВЛЕВО
  ВПРАВО
КОНЕЦ
```

Комментарий. Если кубиков на поле нет, то после выполнения процедуры Шаг (стоящей перед циклом) условие цикла **НЕ ПУСТО** ложно, т. е. и цикл, и программа закончат свою работу.

Тесты

- ❒ На поле кубики отсутствуют, ожидаемый результат: исполнитель останавливается.
- ❒ Проверить все возможные сочетания кубиков в парах: КН, КБ, КК, НК, НБ, НН, БК, БН, ББ.

20. Белая мышка (В. П. Семенко, Рубцовск).

Кот Мурлыка придумал для Кукарачи такую задачу.

Ты находишься в клетке (5,3). Справа, начиная с клетки (5,4), расположен плотный ряд кубиков с буквами С (серая мышка) неизвестной длины. Среди серых есть одна белая мышка — кубик с буквой Б.

Я не стану завтракать, если белая мышь будет стоять рядом с тобой, как показано на рис. 7.20, а серые мышки не разбегутся, т. е. по-прежнему будут образовывать плотный ряд с белой мышкой в начале.

Помогите Кукараче спасти мышиную компанию.

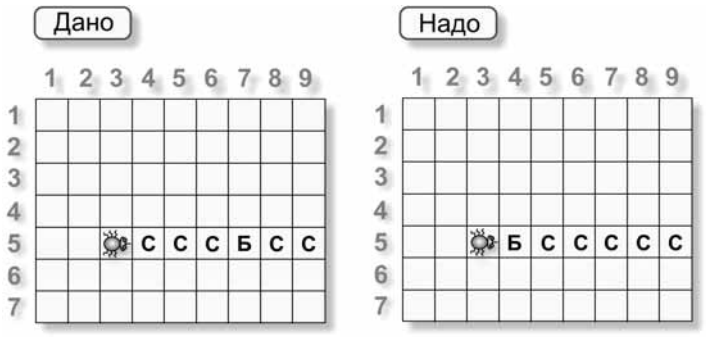

Рис. 7.20

Решение

Алгоритм главной процедуры

1. Проверяем кубики, пока не толкнём Б.
2. Поставим Б на место (процедура Установка_Б).

Алгоритм процедуры Установка_Б.

1. Перемещаем Б влево вдоль проверенного ряда кубиков (заодно смещая их обратно на 5 строку), пока кубики в ряду не кончатся.
2. Поставим Б на первое место и уплотним запись.

Программа

```
// Автор решения: А. В. Рудь
ЭТО Вход
  Шаг_вправо
  ПОКА НЕ Б Шаг_вправо
  Установка_Б
КОНЕЦ

// Нашли белую мышку. Уплотняем ряд и ставим Б на первое место.
// Состояние среды перед выполнением процедуры:
// .СССБ...
// ....¤СС.
// ----------------------------------
ЭТО Установка_Б
  // Сместим Б на одну строку выше
  ВВЕРХ
  // Проверим, есть ли слева от белой мыши серая
  ВПРАВО ВВЕРХ ПОВТОРИ 2 ВЛЕВО ВНИЗ
  // Тащим Б влево, пока кубики С не закончатся.
  // Изменение среды телом цикла за один шаг:
  // ............     ............
  // .....Б......     ....Б.......
  // ...ССС¤..... --> ...СС¤......
  // ......С.СС..     .....СС.СС..
  ПОКА НЕ ПУСТО Шаг_влево
  // Состояние среды после завершения цикла:
  // ............
  // .Б..........
  // ..¤........
```

```
  // ...ССС.СС..
  // Поставим Б на место
  ПОВТОРИ 2 ВВЕРХ
  ВЛЕВО
  ПОВТОРИ 2 ВНИЗ
  ВЛЕВО ВНИЗ
  ПОВТОРИ 2 ВПРАВО
КОНЕЦ

ЭТО Шаг_вправо
  ВНИЗ ВПРАВО ВВЕРХ
КОНЕЦ

ЭТО Шаг_влево
  ВВЕРХ ВЛЕВО ВНИЗ
КОНЕЦ

// Сдвиг Б влево в соседний столбец
// --------------------------------
ЭТО Сдвиг
  ВВЕРХ
  ПОВТОРИ 2 ВПРАВО
  ВВЕРХ ВЛЕВО ВНИЗ ВЛЕВО
КОНЕЦ
```

Комментарий. Работа программы приведёт к отказу «Не могу», если среди кубиков нет кубика с буквой Б. Но такого быть не может, т. к. по условию задачи на поле есть ровно одна белая мышка.

Тесты

- ❒ Кубик Б стоит первым.
- ❒ Кубик Б находится внутри ряда.
- ❒ Кубик Б стоит последним.
- ❒ Ряд состоит только из одного кубика (по условию это может быть только кубик Б).

21. Уа-Ау (Сергей Наумов, 6 класс, ст. Новохопёрск, Воронежская обл.).

В лесу Кукарача услышал плач «уа, уа...». Он нашёл муравьишку и посоветовал бедняжке вместо УА кричать АУ — так получается громче!

Кукарача находится в клетке (2,1), а справа от него плотный ряд кубиков, среди которых встречаются кубики УА (рис. 7.21). Перетасуйте кубики, заменяя УА на АУ так, чтобы ни одного УА в записи не осталось.

Поле считается неограниченным вправо. После выполнения программы Кукарачу нужно вернуть в клетку (2,1), а кубики должны по-прежнему образовывать плотный ряд справа от него.

Рис. 7.21

Решение

Алгоритм

1. Выполнять сортировку записи, пока в ней не закончатся сочетания УА, в том числе и образованные вновь перестановкой кубиков (процедура `Сортировка`). Результирующая запись при этом смещается на строку ниже.
2. Возврат записи и исполнителя в исходное положение (процедура `Кубики_на_место`).

Программа

```
// Автор решения: В. П. Семенко

ЭТО Вход
  Сортировка
  Кубики_на_место
КОНЕЦ

// Перестановка кубиков в записи, пока в ней не останется
// сочетаний УА. После работы процедуры ряд кубиков окажется
// смещённым на строку ниже:
// .........     .........
// ¤ttttttt. --> ........¤
// .........     .ttttttt.
```

```
// .........     .........
// ---------------------------------------
ЭТО Сортировка
  Шаг_вправо
  ПОКА НЕ ПУСТО Проверить_УА
КОНЕЦ

// Возврат ряда в исходную строку:
// .........     .........
// .........¤ --> ¤ttttttt.
// .ttttttt.     .........
// .........     .........
ЭТО Кубики_на_место
  // Подготовка
  ВНИЗ

  // Кубики на место
  Шаг_влево
  ПОКА НЕ ПУСТО Шаг_влево

  // Исполнителя на место
  ВВЕРХ
КОНЕЦ

ЭТО Шаг_вправо
  ВВЕРХ ВПРАВО ВНИЗ
КОНЕЦ

ЭТО Шаг_влево
  ВНИЗ ВЛЕВО ВВЕРХ
КОНЕЦ

// Проверить кубик.
// Если это У, проверить следующий.
// Если получилось УА, выполнить процедуру Перестановка
// Работа процедуры, когда не У:
// .........     .........
```

```
// ...¤tttt. --> ....¤ttt. (продолжение текущей проверки)
// .ttt.....     .tttt....
// .........     .........
// Работа процедуры на УА:
// .........     .........
// ...¤Attt. --> .¤tAУttt. (перестановка УА в АУ и новая
// .ttУ.....     .t.......  проверка кубиков с самого начала)
// .........     .........
// Работа процедуры на УПУСТО:
// .........     .........
// ...¤..... --> ....¤.... (продолжение текущей проверки)
// .ttУ.....     .ttУ.....
// .........     .........
// Работа процедуры на УХ: (продолжение текущей проверки)
// .........     .........
// ...¤Xttt. --> ....¤ttt.
// .ttУ.....     .ttУХ.....
// .........     .........
//--------------------------------------------------------------------
ЭТО Проверить_УА
  ЕСЛИ У
    ТО
    {
      ПОКА У Шаг_вправо
      ЕСЛИ       А     ТО Перестановка
      ИНАЧЕ ЕСЛИ ПУСТО ТО ВЛЕВО
    }
  Шаг_вправо
КОНЕЦ

// Поменять УА на АУ, начало записи вернуть в исходную строку,
// а исполнителя поставить слева от ряда, как в начальном
// состоянии среды.
// Состояние среды перед началом работы процедуры:
// .........
// ....¤ttt.
// .ttУА....
```

```
// .........
// ---------------------------------------
ЭТО Перестановка

// Поменять УА на АУ
  ВНИЗ ВВЕРХ ВЛЕВО
  ПОВТОРИ 2 ВНИЗ
  ВВЕРХ
  ПОВТОРИ 2 ВПРАВО
  ВНИЗ ВЛЕВО
  ПОВТОРИ 2 ВНИЗ
  ПОВТОРИ 2 ВЛЕВО
  ВВЕРХ ВПРАВО
  ВНИЗ ВПРАВО
  ПОВТОРИ 3 ВВЕРХ
  ПОВТОРИ 2 ВНИЗ
  ВЛЕВО
  ПОВТОРИ 2 ВВЕРХ

  // Состояние среды:
  // .........
  // ...АУttt.
  // .tt¤.....
  // .........

  // Вернём начало записи в исходную строку
  Шаг_влево
  ПОКА НЕ ПУСТО Шаг_влево

  // Состояние среды:
  // .........
  // .ttАУttt.
  // ¤.........
  // .........

  // Вернём исполнителя в исходное положение
  ВВЕРХ
```

```
  // Состояние среды:
  // .........
  // ¤ttАУttt.
  // .........
  // .........
КОНЕЦ
```

Комментарий. Программа начинает проверку с самого начала, как только в записи встречается сочетание УА. Например, для исходной записи УАУАУА получим серию таких преобразований:

УАУАУА

АУ**УА**УА

А**УА**УУА

ААУУ**УА**

ААУ**УА**У

АА**УА**УУ

АААУУУ

Тесты

- Только одно буквосочетание УА, исправление которого не приводит к новым УА. Нужно рассмотреть случаи, когда УА начинает, заканчивает запись или расположена в её середине (табл. 7.4).

Таблица 7.4

Начальная запись	Конечная запись
УА1234	АУ1234
12УА34	12АУ34
1234УА	1234АУ

- Только одно буквосочетание УА, исправление которого приводит к новым УА. Нужно рассмотреть случаи, когда УА начинает запись или расположена в её середине (табл. 7.5).

Таблица 7.5

Начальная запись	Конечная запись
УАААА	ААААУ
ААУАА	ААААУ

- Несколько буквосочетаний УА (табл. 7.6).

Таблица 7.6

Начальная запись	Конечная запись
УА1УА2УА	АУ1АУ2АУ
УАУАУА	АААУУУ

- Нет ни одного кубика с У.
- Нет ни одного сочетания УА, но есть кубики с У.
- Пустая последовательность (нет кубиков).
- Один кубик с не У (например, Т).
- Один кубик У.
- Два кубика, на которых нет У (например, ТТ).
- Два кубика: не УА, но есть У: УТ, ТУ.
- УА, АУ.

22. Новогодняя сказка (Н. В. Табашин, Лукоянов).

Зима. Снег. На скрытом кубике — возраст ёлки: число 1, 2, 3 или 4. Кукарача должен очистить от снега ёлку, учитывая её возраст. Каждый год ель подрастает на ярус вверх, а ветви прежних лет увеличиваются в длину (рис. 7.22).

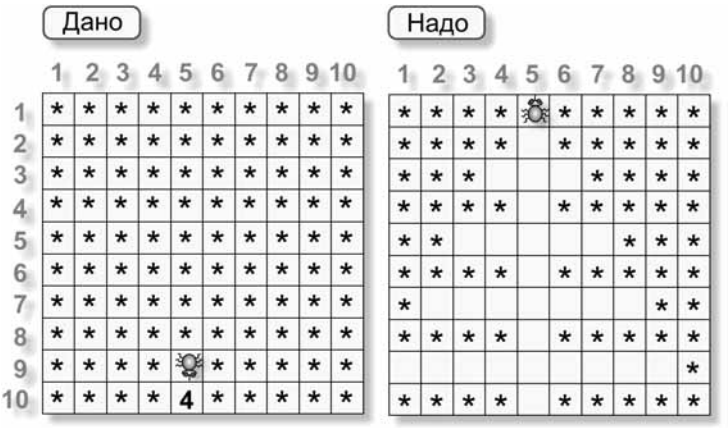

Рис. 7.22

Решение

Описание алгоритма

Исполнитель должен толкнуть кубик, стоящий под ним, и определить n — цифру на кубике (1, 2, 3 или 4). В зависимости от n выполняется соответствующая процедура (Один_год, Два_года, Три_года, Четыре_года), которая «очищает от снега» ветки ёлки. Если ёлочке больше года, то в процедуре для n лет используется вызов процедуры для (n − 1) лет.

Программа

```
// Автор решения: В. П. Семенко

ЭТО Вход
  ВНИЗ
  // Сколько лет ёлке?
  ЕСЛИ        1 ТО Один_год
  ИНАЧЕ ЕСЛИ 2 ТО Два_года
  ИНАЧЕ ЕСЛИ 3 ТО Три_года
  ИНАЧЕ ЕСЛИ 4 ТО Четыре_года
КОНЕЦ

ЭТО Один_год
  ВВЕРХ ВЛЕВО
  ПОВТОРИ 2 ВПРАВО
  ВЛЕВО
  ПОВТОРИ 2 ВВЕРХ
КОНЕЦ

ЭТО Два_года
  ВВЕРХ
  ПОВТОРИ 2 ВЛЕВО
  ПОВТОРИ 4 ВПРАВО
  ПОВТОРИ 2 ВЛЕВО
  ВВЕРХ
  Один_год
КОНЕЦ

ЭТО Три_года
  ВВЕРХ
```

```
  ПОВТОРИ 3 ВЛЕВО
  ПОВТОРИ 6 ВПРАВО
  ПОВТОРИ 3 ВЛЕВО
  ВВЕРХ
  Два_года
КОНЕЦ

ЭТО Четыре_года
  ВВЕРХ
  ПОВТОРИ 4 ВЛЕВО
  ПОВТОРИ 8 ВПРАВО
  ПОВТОРИ 4 ВЛЕВО
  ВВЕРХ
  Три_года
КОНЕЦ
```

Тесты

Каждая из цифр 1, 2, 3 и 4 на скрытом кубике должна быть поочередно протестирована.

23. Машина времени (С. В. Пинженина, Челябинск).

Помогите Кукараче построить транспорт, соответствующий записи на пульте управления в клетке (2,1): Б (будущее) или П (прошлое). Слово можно строить по вертикали или горизонтали. Начальное положение исполнителя в клетке (1,1), конечное положение несущественно, детали транспорта (буквы А, А, Е, К, Р, Т) рассыпаны, как показано на рис. 7.23.

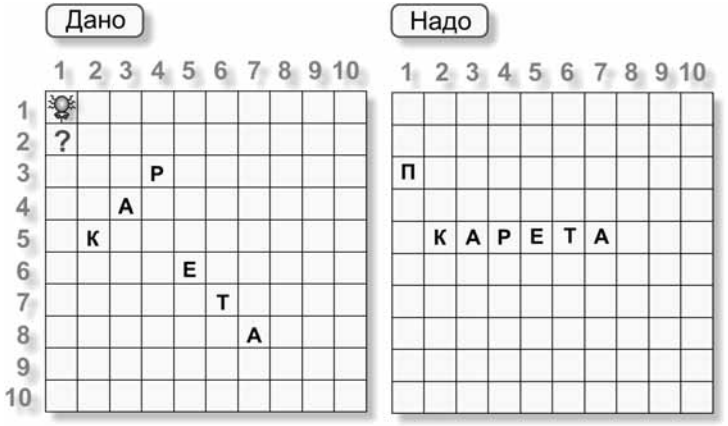

Рис. 7.23

Решение

Описание алгоритма

Проверить, какая буква на скрытом кубике в клетке (2,1). Если там Б (будущее), то построить в пятом столбце слово РАКЕТА, если на кубике буква П, построить в пятой строке слово КАРЕТА, а если на кубике какой-то другой символ, то вернуть исполнитель в начальное положение.

Программа

```
// Автор решения: Данил Орлов, 7 класс, Рубцовск

ЭТО Вход
  ВНИЗ
  // Какая буква на кубике?
  ЕСЛИ       П ТО Карета
  ИНАЧЕ ЕСЛИ Б ТО Ракета
  ИНАЧЕ          ВВЕРХ
КОНЕЦ

// На кубике Б (будущее): строим по вертикали слово РАКЕТА
// -----------------------------------------------------
ЭТО Ракета
  ВПРАВО
  ПОВТОРИ 2 ВНИЗ
  ВПРАВО             // Установка А
  ПОВТОРИ 2 ВЛЕВО
  ВНИЗ
  ПОВТОРИ 2 ВПРАВО   // Установка К

  // Подходим к ЕТА
  ПОВТОРИ 4 { ВНИЗ ВПРАВО }
  ВПРАВО ВВЕРХ
  ПОВТОРИ 3 ВЛЕВО    // Установка А
  ПОВТОРИ 2 ВПРАВО
  ВВЕРХ
  ПОВТОРИ 2 ВЛЕВО    // Установка Т
  ВПРАВО ВВЕРХ ВЛЕВО // Установка Е
КОНЕЦ
```

```
// На кубике П (прошлое): строим по горизонтали слово КАРЕТА
// -------------------------------------------------------
ЭТО Карета
  ПОВТОРИ 2 ВПРАВО
  ПОВТОРИ 2 ВНИЗ      // Установка А
  ПОВТОРИ 2 ВВЕРХ
  ВПРАВО
  ПОВТОРИ 2 ВНИЗ      // Установка Р

  // Подходим к ЕТА
  ПОВТОРИ 4 { ВПРАВО ВНИЗ }
  ВНИЗ ВЛЕВО
  ПОВТОРИ 3 ВВЕРХ     // Установка А
  ПОВТОРИ 2 ВНИЗ
  ВЛЕВО
  ПОВТОРИ 2 ВВЕРХ     // Установка Т
  ВНИЗ ВЛЕВО
  ВВЕРХ               // Установка Е
КОНЕЦ
```

Тесты

- ❒ На скрытом кубике буква Б. Результат — РАКЕТА.
- ❒ На скрытом кубике буква П. Результат — КАРЕТА.
- ❒ На скрытом кубике не Б и не П. Результат — исполнитель возвращается в клетку (1,1).

24. Зелёный ряд (С. В. Пинженина, Челябинск).

В Роботландии завёлся злоумышленник, который всегда готов в зелёный ряд кресел впихнуть кресло другого цвета!

В третьей строке, начиная с третьего столбца, стоит плотный ряд зелёных кресел (кубики З) неизвестной длины. Среди них находится, возможно, одно кресло другого цвета (кубик с другим символом).

Удалить с поля лишнее кресло и добавить зелёное со склада, если оно там есть.

Склад — плотный ряд кубиков, расположенный во втором столбце, начиная с пятой строки. Число кресел на складе заранее не известно.

Начальное положение Кукарачи — клетка (1,1). В конце работы зелёные кресла по-прежнему образуют плотный горизонтальный ряд, начиная

с клетки (3,3). Поле можно считать неограниченным вниз и вправо. Примеры начального и конечного состояния среды показаны на рис. 7.24.

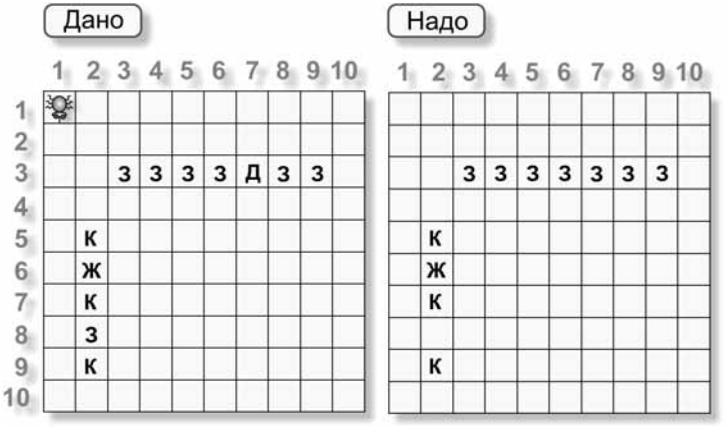

Рис. 7.24

Решение

Алгоритм

1. Подходим к зелёному ряду.
2. Пока З (зелёные кресла), двигаемся вправо, толкая кубики снизу вверх.
3. Нашли не З. Проверяем, что найдено:
   3.1. Если Д, то производим замену на З (процедура `Замена`).
   3.2. В противном случае в ряду нет Д (кресел не зелёного цвета) — устанавливаем кубики ряда в начальное положение (процедура `Возврат`).

Алгоритм процедуры `Замена`

1. Удаляем с поля Д.
2. Ищем конец ряда и делаем ряд плотным (один сдвиг влево).
3. Передвигаем кресла в третью (исходную) строку. При этом ряд будет начинаться с клетки (3,3), как того требует условие.
4. Идём на склад искать З (зелёное кресло):
   4.1. Делаем шаг вниз.
   4.2. Пока не нашли конец склада, делаем:
      4.2.1. Если З найдено, то перемещаем З в зелёный ряд (процедура `Зелёный_ряд`).
      4.2.2. Иначе продолжаем поиск, делая шаг вниз.

Алгоритм процедуры Зелёный_ряд

1. Устанавливаем З в клетку (4,5).
2. Добавляем З к ряду из З слева.

Алгоритм процедуры Возврат

Пока не нашли начало ряда, толкаем кубики из второй строки в третью.

Программа

```
// Автор решения: В. П. Семенко

ЭТО Вход
  // Подходим к зелёному ряду
  ВПРАВО
  ПОВТОРИ 2 ВНИЗ

  // Ищем не З в ряду кубиков
  Шаг_вправо
  ПОКА З Шаг_вправо

  // Нашли не З
  ЕСЛИ ПУСТО
    ТО    { ВВЕРХ Возврат } // Нашли ПУСТО
    ИНАЧЕ Замена            // Найдено кресло другого цвета
КОНЕЦ

// Установка зелёного ряда на исходное место.
// Преобразование среды процедурой:
// .........     .........
// ...ЗЗЗЗ¤. --> .. ¤......
// .........     ...ЗЗЗ..
// ----------------------------------
ЭТО Возврат
  Шаг_влево
  ПОКА НЕ ПУСТО Шаг_влево
КОНЕЦ

// Удаление кресла не зелёного цвета и поход
// на склад за зелёным креслом
```

```
// -------------------------------------------
ЭТО Замена
  ПОВТОРИ 2 ВВЕРХ // Удаляем
  ПОВТОРИ 2 ВНИЗ

  // Ищем конец ряда
  Шаг_вправо
  ПОКА НЕ ПУСТО Шаг_вправо

  // Делаем ряд плотным
  ВВЕРХ ВЛЕВО

  // Ставим ряд в исходное положение
  Возврат

  // Идём на склад за зелёным креслом
  На_склад
КОНЕЦ

// Поиск зелёного кресла на складе.
// Состояние среды перед входом в процедуру:
// .........
// ..¤......
// ...3333..
// .........
// ..С......
// ..К......
// ..Л......
// ..А......
// ..Д......
// ------------------------------
ЭТО На_склад
  // Подход к складу
  ПОВТОРИ 2 ВНИЗ

  // Поиск 3 на складе.
  // Если 3 найдено, то цикл ПОКА искусственно прерывается.
```

```
  // Для прерывания цикла в процедуре Добавить_З (вызывается
  // из процедуры Зеленый_ряд) исполнитель смещается в заведомо
  // пустую клетку.
  Шаг_вниз
  ПОКА НЕ ПУСТО
    ЕСЛИ З
      ТО Зеленый_ряд
      ИНАЧЕ Шаг_вниз
КОНЕЦ

// На складе найдена З.
// Перемещаем З в клетку (4,5).
// Состояние среды перед входом в процедуру:
// .........
// ...ЗЗЗЗ..
// .....*... <-- знаком * обозначена клетка (4,5)
// ...С.....
// ...К.....
// ..¤З.....
// ..Л......
// ..А......
// ..Д......
// ---------------------------
ЭТО Зеленый_ряд
  ПОВТОРИ 2 ВПРАВО
  ВНИЗ ВПРАВО ВВЕРХ ВЛЕВО ВВЕРХ ВЛЕВО
  ПОКА НЕ ПУСТО Шаг_вверх
  Добавить_З
КОНЕЦ

// Добавляем З в зелёный ряд
// Состояние среды перед входом в процедуру:
// .........
// ...ЗЗЗЗ..
// .. ¤.З...
// ..С......
// ..К......
```

```
// .........
// ..Л......
// ..А......
// ..Д......
// --------------------
ЭТО Добавить_З

  // Уплотняем склад
  ВЛЕВО ВНИЗ

  // Добавляем З
  ПОВТОРИ 4 ВПРАВО
  ВВЕРХ
  ПОВТОРИ 3 ВЛЕВО
  ВНИЗ ВЛЕВО ВВЕРХ ВЛЕВО ВВЕРХ ВПРАВО

  // Эта команда (исполнитель смещается в заведомо пустую клетку)
  // служит для прерывания цикла ПОКА в процедуре На_склад. Возврат
  // в тело этого цикла происходит сразу после выполнения
  // нижерасположенной команды.
  ВВЕРХ
КОНЕЦ

ЭТО Шаг_вправо
  ВНИЗ ВПРАВО ВВЕРХ
КОНЕЦ

ЭТО Шаг_влево
  ВВЕРХ ВЛЕВО ВНИЗ
КОНЕЦ

ЭТО Шаг_вниз
  ВЛЕВО ВНИЗ ВПРАВО
КОНЕЦ

ЭТО Шаг_вверх
  ВПРАВО ВНИЗ ВПРАВО
```

```
  ВВЕРХ ВЛЕВО ВВЕРХ
  ВЛЕВО
КОНЕЦ
```

Комментарий

1. В процедуре `Замена` (она вызывается, когда в зелёном ряду есть кубик с буквой Д) необходимо дойти до конца ряда, уплотнить его и сдвинуть ряд в третью (исходную) строку так, чтобы начало ряда было в клетке (3,3). Это нужно для того, чтобы удовлетворить условию задачи: «в конце работы зелёные кресла по-прежнему образуют плотный горизонтальный ряд, начиная с клетки (3,3)».

   В самом деле, если в зелёном ряду нет кубика Д (или на складе нет З), то ряд после сдвига будет находиться в правильном положении.

   Если в зелёном ряду была буква Д, то буква З со склада будет поставлена в его начало со сдвигом вправо: зелёный ряд снова будет находиться в правильном положении.

2. Цикл **ПОКА НЕ ПУСТО** (поиск З на складе) в процедуре `На_склад` искусственно прерывается, как только найден кубик З. В процедуре `Добавить_З` для этого предусмотрена команда **ВВЕРХ**, которая толкает заведомую пустоту.

Тесты

- ❒ Зелёный ряд пуст.
- ❒ Склад пуст.
- ❒ В зелёном ряду есть З, но нет Д.
- ❒ Зелёный ряд состоит из одной буквы Д, и на складе нет З.
- ❒ Зелёный ряд состоит из одной буквы Д, и на складе есть З.
- ❒ Зелёный ряд состоит из двух кубиков ДЗ или ЗД, и на складе нет З.
- ❒ Зелёный ряд состоит из двух кубиков ДЗ или ЗД, и на складе есть З.
- ❒ Зелёный ряд состоит из нескольких З и одной Д, и на складе нет З.
- ❒ Зелёный ряд состоит из нескольких З и одной Д, и на складе есть З.

25. Пример (Владимир Югатов, Рубцовск).

Во второй строке, начиная с клетки (2,2), расположена запись арифметических действий (+, -, *, /) над натуральными числами. Кукарача стоит перед записью в той же строке, а пример начинается и заканчивается числом.

Один из знаков арифметических действий по ошибке повторяется два раза подряд. Нужно исправить ошибку (удалить с поля лишний знак)

и уплотнить запись. Конечное положение исполнителя и записи несущественны. Примеры начального и конечного состояния среды показаны на рис. 7.25.

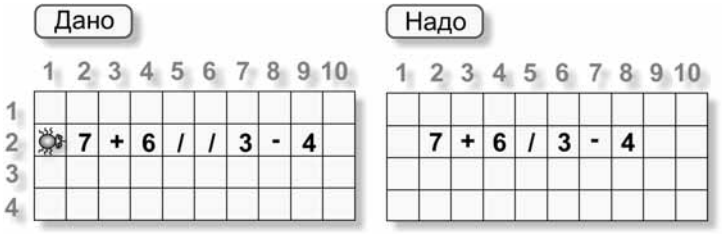

Рис. 7.25

Решение

Описание алгоритма

Толкаем кубики записи снизу вверх, пока она не закончится. Когда знак найден, проверяем кубик за ним. Если на этом кубике — снова арифметический знак, удаляем его с поля. После завершения цикла (найдена пустая клетка), уплотняем запись сдвигом её конца влево на одну клетку.

Программа

```
// Автор решения: В. П. Семенко
ЭТО Вход
  Шаг                              // Толкаем первый кубик
  ПОКА НЕ ПУСТО { Проверка Шаг }   // Находим и удаляем лишний знак
  Уплотнить                        // Уплотняем запись
КОНЕЦ

ЭТО Шаг
  ВНИЗ ВПРАВО ВВЕРХ
КОНЕЦ

ЭТО Уплотнить
  ВВЕРХ ВЛЕВО
КОНЕЦ

ЭТО Проверка
  ЕСЛИ НЕ ЦИФРА                    // Проверяем, что на кубике?
    ТО Найден_знак                 // На кубике не цифра
КОНЕЦ
```

```
ЭТО Найден_знак
  Шаг
  ЕСЛИ НЕ ЦИФРА                    // Посмотрим, что за знаком
    ТО Вытолкни                    // Вытолкнем повторный знак
КОНЕЦ

ЭТО Вытолкни
  ВВЕРХ ВНИЗ
КОНЕЦ
```

Комментарий. Программа будет работать и для пустой записи, и для записи, в которой нет ошибок. Но если в записи больше одной пары повторяющихся знаков арифметических действий, то программа сработает неправильно: запись не уплотнится. Задачу с таким обобщением можно решать в *главе 6*: ошибки можно «посчитать» при помощи рекурсии, а уплотнение выполнить рекурсивной пружинкой.

Тесты

- ❒ Пустая запись.
- ❒ Правильная запись с действием над цифрами (например, 1 + 2/3).
- ❒ Правильная запись с действием над числами (например, 24/12 + 18).
- ❒ Неправильная запись с одним арифметическим действием (например, 24//12).
- ❒ Неправильная запись с двумя и более арифметическими действиями (например, 24 + 12**3).

26. Торт (Михаил Суворов, Илья Мокс, 8 класс, Тверь).

Кукарача находится в клетке (1,1) на поле размером 10×10. Плотный горизонтальный ряд кубиков начинается с клетки (2,2). Число кубиков в ряду не больше 7, и среди них есть кубики Т, О, Р и Т.

Помогите Кукараче подарить Гене торт на день рождения и прогнать злую Шапокляк с поля.

Пример начального состояния среды и конечное состояние среды показан на рис. 7.26.

Решение

Алгоритм

1. Пока кубики в ряду есть, толкать их сверху вниз и искать буквы Т, О и Р, расставляя их в нужные строки:

   1.1. Т в 4 строку.

1.2. О в 5 строку.

1.3. Р в 6 строку.

Лишние буквы, если они есть, окажутся в третьей строке.

Рис. 7.26

2. Очистить третью строку (удалить кубики с поля).
3. Расставить буквы Т, О, Р и Т в удобном порядке:

```
.........
..T......
...O.T...
....P....
.........
```

4. Подарить Гене ТОРТ, разместив его в девятой строке.
5. Прогнать ШАПОКЛЯК (удалить кубики с поля).

Программа

```
// Автор решения: Юлия Чикачкова, 9 класс, ст. Новохоперск

ЭТО Вход
  // Толкаем первый кубик ряда
  ВПРАВО ВНИЗ

  // Состояние среды
  //   123456789
  // 1 .........
```

```
// 2 .¤АТРОКТ.Ш
// 3 .А.......А
// 4 .........П
// 5 .........О
// 6 .........К

// Ищем в ряду буквы Т, О, Р
ПОКА НЕ ПУСТО Поиск

// Состояние среды
//   123456789
// 1 ..........
// 2 .........¤Ш
// 3 .АА...К..А
// 4 ...Т...Т.П
// 5 .....О...О
// 6 ....Р....К

Очистка_третьей_строки

// Состояние среды
//   123456789
// 1 ..........
// 2 ..........
// 3 ..........
// 4 ¤..Т...Т.А
// 5 .....О...П
// 6 ....Р....О

Расстановка_Т_О_Р_Т

// Состояние среды
//   123456789
// 1 ..........
// 2 ..........
// 3 ..........
// 4 .....Т....
```

```
  // 5 ......О.ТП
  // 6 ......¤Р.О

  Подарить_ТОРТ

  // Состояние среды
  //    123456789
  //  3 .........¤
  //  4 ..........
  //  5 .........П
  //  6 .........О
  //  7 .........К
  //  8 .........Л
  //  9 .....ТОРТЯ
  // 10 .....ГЕНАК

  Прогнать_ШАПОКЛЯК
КОНЕЦ

ЭТО Поиск
  ЕСЛИ       Т ТО Ставь_Т
  ИНАЧЕ ЕСЛИ О ТО Ставь_О
  ИНАЧЕ ЕСЛИ Р ТО Ставь_Р
  ИНАЧЕ Шаг

КОНЕЦ

// Буквы Т ставим в 4 строку
ЭТО Ставь_Т
  ВНИЗ ВВЕРХ
  Шаг
КОНЕЦ

// Букву О ставим в 5 строку
ЭТО Ставь_О
  ПОВТОРИ 2 ВНИЗ
  ПОВТОРИ 2 ВВЕРХ
  Шаг
КОНЕЦ
```

```
// Букву Р ставим в 4 строку
ЭТО Ставь_Р
  ПОВТОРИ 3 ВНИЗ
  ПОВТОРИ 3 ВВЕРХ
  Шаг
КОНЕЦ

ЭТО Шаг
  ВВЕРХ ВПРАВО ВНИЗ
КОНЕЦ

ЭТО Очистка_третьей_строки
  // Выброс Ш
  ПОКА ПУСТО ВПРАВО

  // Очистка третьей строки
  ВНИЗ
  ПОВТОРИ 9 ВЛЕВО
  ВНИЗ
КОНЕЦ

ЭТО Расстановка_Т_О_Р_Т
  ПОВТОРИ 6 ВПРАВО
  ВВЕРХ
  ПОВТОРИ 2 ВПРАВО
  ВНИЗ             // Последнюю Т в (5,9)
  ВПРАВО
  ПОВТОРИ 3 ВЛЕВО // Первую Т в (4,6)
  ВВЕРХ
  ПОВТОРИ 6 ВЛЕВО
  ПОВТОРИ 2 ВНИЗ
  ПОВТОРИ 5 ВПРАВО // Букву О в (5,7)
  ПОВТОРИ 5 ВЛЕВО
  ВНИЗ
  ПОВТОРИ 6 ВПРАВО // Букву Р в (6,8)
КОНЕЦ
```

```
ЭТО Подарить_ТОРТ
  ПОВТОРИ 2 ВЛЕВО
  ПОВТОРИ 3 ВВЕРХ
  ВПРАВО
  ПОВТОРИ 4
  {
    ПОВТОРИ 5 ВНИЗ
    ПОВТОРИ 5 ВВЕРХ
    ВПРАВО
  }
КОНЕЦ

ЭТО Прогнать_ШАПОКЛЯК
  ПОВТОРИ 7 ВНИЗ
КОНЕЦ
```

Комментарий. Эта задача напоминает задачу про ЭКРАН (задача 7 из *главы 4*), но имеет особенность: в слове ТОРТ две буквы одинаковые.

Тесты

Для проверки программы можно предложить два теста:

- В ряду, начиная с клетки (2,2), четыре кубика с буквами Т, О, Р и Т, и кубики располагаются в указанном порядке.
- В ряду, начиная с клетки (2,2), произвольное количество кубиков, от 5 до 7, среди них обязательно есть кубики с буквами Т, О, Р и Т, которые располагаются в ряду в произвольном порядке.

Ожидаемым результатом для любого теста является конечное положение, указанное на рис. 7.26, т. е. в десятой строке написано слово ГЕНА, а над ним, в девятой строке, написано слово ТОРТ.

Задачи к главе 6

27. Отрезки (В. П. Семенко, Рубцовск).

    В пятой строке поля, не ближе второго столбца, стоят три кубика: A, B и C. Это концы отрезков AB и BC. Кукарача находится под средним кубиком B.

    Длину отрезка Кукарача измеряет числом пустых клеток между его концами. Проверить, равны ли длины отрезков AB и BC, и оставить на поле правильное сообщение (рис. 7.27).

Поле считать бесконечным вправо. Положение отрезков на поле измениться не должно.

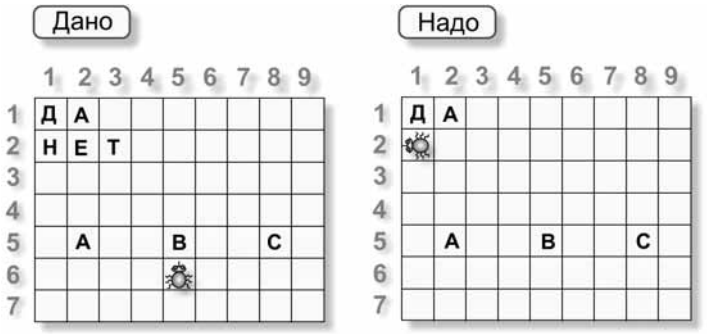

Рис. 7.27

Решение

Алгоритм

1. «Измерить» длину левого отрезка, рекурсивно двигаясь до левого кубика.
   1.1. Когда левый кубик найден (сдвинут), поставить его на прежнее место.
   1.2. Рекурсивной пружинкой сместиться вправо на двойную длину левого отрезка.
2. Сместиться в клетку, где должен стоять кубик при равенстве отрезков.
3. Проверить, что толкнул исполнитель.
   3.1. Если клетка пустая, то отрезки не равны, и необходимо удалить с поля слово ДА.
   3.2. Если в клетке кубик, то отрезки равны. Вернуть кубик на прежнее место и удалить с поля слово НЕТ.

Программа

```
ЭТО Вход
  // Измерить длину левого отрезка и сместиться от его
  // начала на двойную длину.
  Измерение
  // Проверить, есть ли кубик над исполнителем (на том
  // месте, где он должен стоять при равенстве отрезков).
  // Если да, то длины отрезков равны.
```

```
  ВВЕРХ
  ЕСЛИ ПУСТО
    ТО     Удали_ДА
    ИНАЧЕ  Удали_НЕТ
КОНЕЦ

// Рекурсивный поиск кубика А
// --------------------------
ЭТО Измерение
  ВЛЕВО ВВЕРХ
  ЕСЛИ ПУСТО
    ТО { ВНИЗ Измерение }
    ИНАЧЕ На_место // Поставить кубик А на прежнее место
  // Рекурсивная пружинка для смещения исполнителя
  // вправо на двойную длину первого отрезка.
  ПОВТОРИ 2 ВПРАВО
КОНЕЦ

ЭТО На_место
  // Вернуть кубик А на исходное место
  ВЛЕВО ВВЕРХ ВВЕРХ ВПРАВО ВНИЗ
  ВЛЕВО ВНИЗ ВНИЗ ВПРАВО
КОНЕЦ

ЭТО Удали_ДА
  // Сместиться в первую строку
  ПОВТОРИ 4 ВВЕРХ
  // Удалить слово ДА
  Удалить_два_кубика
КОНЕЦ

ЭТО Удалить_два_кубика
  ПОКА ПУСТО ВЛЕВО
  ВЛЕВО
КОНЕЦ

ЭТО Удали_НЕТ
```

```
  // Поставить кубик С на прежнее место
  ВПРАВО ВВЕРХ ВВЕРХ ВЛЕВО ВНИЗ
  // Сместиться во вторую строку
  ВВЕРХ ВВЕРХ
  // Удалить слово НЕТ
  Удалить_два_кубика
  // Удалить третий кубик
  ВЛЕВО
КОНЕЦ
```

Тесты

- ❒ Левый отрезок имеет нулевую длину.
- ❒ Левый отрезок имеет единичную длину.
- ❒ Левый отрезок имеет большую длину.

Часть II

Корректор

Глава 8

Знакомство с исполнителем

Корректор и Кукарача

Возможности Корректора существенно выше возможностей Кукарачи. Далее приводится сравнительный анализ этих исполнителей.

Ввод информации

Оба исполнителя не имеют средств для ввода данных во время работы программы. Данные, подлежащие обработке, формируются в среде перед запуском программы (начальная установка среды).

- В среде Кукарачи задаются надписи на кубиках, их положение и положение исполнителя на поле.
- В среде Корректора задаётся содержимое ячеек ленты и положение окна на ленте.

Вывод информации

Кукарача

Формирует результат обработки при помощи:

- изменения положения кубиков на поле. Пример: из слова КОТ получается слово ТОК;
- удаления кубиков с поля. Пример: из слова МУРАШКА получается слово МУРА;
- своего положения на поле. Пример: положение исполнителя в шестой строке показывает, что количество букв в слове КОРОВА равно 6.

Кукарача не способен создать новый кубик или изменить надпись. Все кубики, необходимые для формирования результата, должны быть поставлены на поле перед выполнением программы.

Выполнить сложение 2 + 3 Кукарача сможет лишь тогда, когда на поле есть кубик с цифрой 5. Если такого кубика нет, то у Кукарачи остаётся только косвенная возможность показать результат, например, своим положением на поле (в пятой строке или пятом столбце). Можно, конечно, предложить и совсем экзотическую идею: продемонстрировать результат, переместив пример 2 + 3 в пятую строку.

Корректор

Способен создавать записи на ленте командой **пиши** (новые «кубики»), менять символы при помощи команд **плюс**, **минус** (изменение надписи на «кубике»).

Таким образом, команды **пиши**, **плюс**, **минус** позволяют Корректору создавать результирующую информацию более гибким и более привычным для программирования способом по отношению к скромным возможностям Кукарачи.

Память

Кукарача

Кукарача — очень «легкомысленный» исполнитель: у него нет средств для хранения новой информации.

Можно было бы запоминать данные, используя положение кубиков на поле. Например, кубик в клетке (2,3) можно интерпретировать как число 2, или число 3, или как кортеж (2,3). Но Кукарача не умеет определять координаты клеток.

Единственный способ «запомнить» число в среде Кукарачи основан на конструкции с рекурсивным вызовом.

Общая схема рекурсивной «памяти» показана на следующем условном коде:

```
ЭТО Рекурсивная_память
  команды1
  ЕСЛИ условие
    ТО
    {
      команды2
      Рекурсивная_память
    }
    ИНАЧЕ
```

```
    {
      команды3
    }
  команды4
КОНЕЦ
```

Работает эта конструкция так.

1. В рекурсивном цикле (пока условие истинно) выполняется группа команд `{ команды1 команды2 }` и *запоминается* число повторений. Понятно, что число повторений запоминает не исполнитель, а интерпретатор, для которого каждый рекурсивный вызов равнозначен выполнению нового экземпляра процедуры `Рекурсивная_память`.
2. Когда условие становится ложным, рекурсия заканчивается: один раз выполняется группа `{ команды1 команды3 }` (как правило, `команды3` подготавливают среду ко входу в цикл выполнения отложенной `команды4`).
3. Группа `{ команды4 }` срабатывает столько раз, сколько работала группа `{ команды1 }` (на один раз больше числа срабатывания `команды2`). Так происходит потому, что `команды4` работают в каждом рекурсивном экземпляре процедуры `Рекурсивная_память` и в оригинале.

Корректор

Рекурсивной «памятью» обладает любой исполнитель, над которым надстроен интерпретатор, поддерживающий рекурсивные программы. В том числе рекурсивная память есть и у Корректора.

Кроме того, у Корректора есть лента, над ячейками которой исполнитель способен выполнять модифицирующие действия **ПИШИ**, **ПЛЮС**, **МИНУС**, а значит, ленту можно рассматривать как память, похожую на ОЗУ (оперативное запоминающее устройство) компьютера. Наконец, у Корректора есть ещё одна память — ящик. Эта память маленькая по объёму (одна ячейка), но быстрая (не зависит от положения окна на ленте). Ящику Корректора можно сопоставить регистровую память компьютера.

Система команд

В системе команд Кукарачи четыре команды, меняющие положение исполнителя на поле (**ВВЕРХ**, **ВНИЗ**, **ВПРАВО**, **ВЛЕВО**), одна пустая команда (**СТОЯТЬ**) и группа команд для проверки клетки, в которую шагнул исполнитель.

Проверочные команды обеспечивают обратную связь с исполнителем, а значит, позволяют программировать развилки, конечные рекурсии и циклы **ПОКА**. Эти команды записываются как условия соответствующих конструкций, и результатом их работы является одно из двух сообщений: *истина* или *ложь*.

Список команд-условий для Кукарачи:

- символ (результат — истина, если в клетке кубик с указанным символом);
- **ПУСТО** (результат — истина, если клетка пуста);
- **ЦИФРА** (результат — истина, если в клетке кубик с десятичной цифрой).

Этот список возрастает вдвое при помощи модификатора **НЕ**, который меняет результат проверки на обратный.

У Корректора команд больше и они сложнее. Пустую команду **СТОЯТЬ** дополняют две команды, перемещающие окно по ячейкам ленты (**ВПРАВО**, **ВЛЕВО**). Три команды реализуют передачу данных между лентой и ящиком (**ЯЩИК+**, **ЯЩИК-**, **ОБМЕН**). Три команды редактируют данные на ленте (**ПИШИ**, **ПЛЮС**, **МИНУС**).

Расширен и список команд-условий:

- символ (результат — истина, если в окне записан указанный символ);
- **ПУСТО** (результат — истина, если в окне записан символ **ПУСТО**);
- **ПРОБЕЛ** (результат — истина, если в окне записан символ **ПРОБЕЛ**);
- **ЦИФРА** (результат — истина, если в окне записана десятичная цифра);
- **Я=Л** (результат — истина, если символ в окне совпадает с символом в ящике);
- **Я#Л** (результат — истина, если символ в окне не совпадает с символом в ящике);
- **Я>Л** (результат — истина, если алфавитный номер символа в ящике больше алфавитного номера символа в окне);
- **Я<Л** (результат — истина, если алфавитный номер символа в ящике меньше алфавитного номера символа в окне).

Перед любым условием может быть записан модификатор **НЕ**, который меняет результат проверки на противоположный.

Принципиально отличие Корректора от Кукарачи заключено в двух группах команд:

- команды **ПИШИ**, **ПЛЮС**, **МИНУС** позволяют редактировать содержимое ячеек;
- команды-условия **Я=Л**, **Я#Л**, **Я>Л**, **Я<Л** позволяют сравнивать ячейки с вычисляемым в программе значением (отличным от константы).

Язык программирования

Одинаков для обоих исполнителей.

Общие рекомендации

Первое знакомство с Корректором должно пройти достаточно быстро. Ученик знакомится с новой средой и новой системой команд исполнителя, а язык программирования и интерфейс экранной модели остаются прежними (как у Кукарачи).

Для представления Корректора можно использовать общую схему знакомства с исполнителем, вынесенную на классную доску, кодоскоп или демонстрационный экран (рис. 8.1).

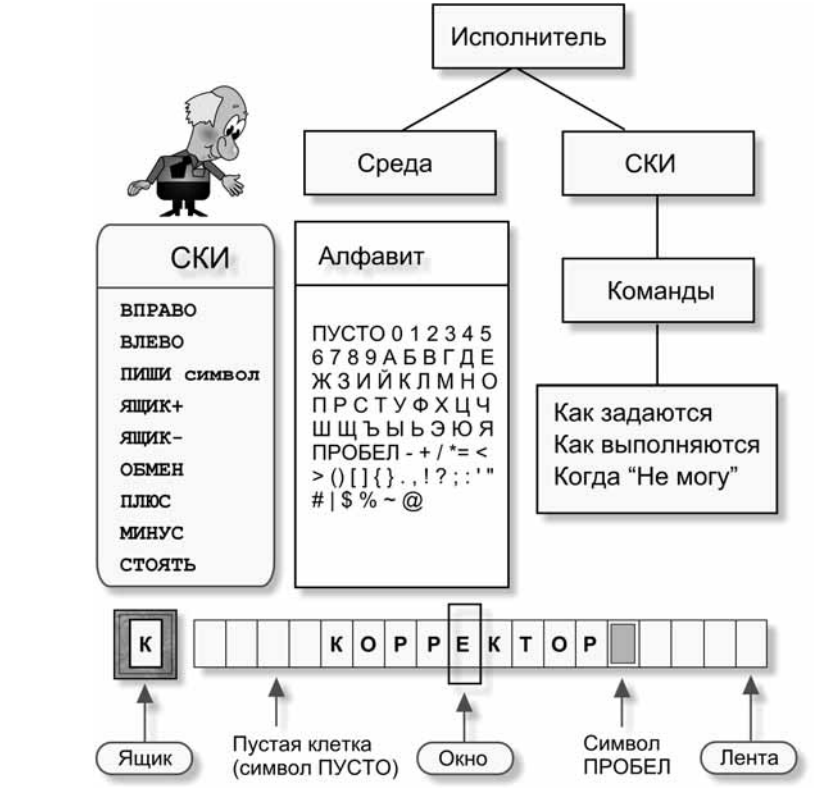

Рис. 8.1

При рассмотрении алфавита Корректора обратите внимание на то, что символы `ПРОБЕЛ` и `ПУСТО` являются разными. Изначально все ячейки ленты и ящик пусты, то есть заполнены символами `ПУСТО`.

Ответы на вопросы и задания

8.1. Корректор и его среда обитания

Какой символ запишет Корректор на ленту по указанной команде и для показанного состояния среды.

1. плюс (рис. 8.2).

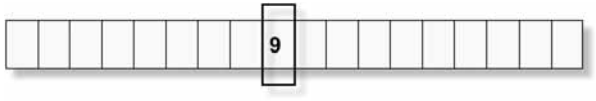

Рис. 8.2

Ответ. Символ А.

2. минус (рис. 8.3).

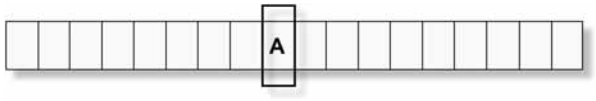

Рис. 8.3

Ответ. Символ 9.

3. плюс (рис. 8.4).

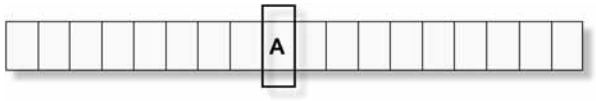

Рис. 8.4

Ответ. Покажет сообщение «Не могу».

4. минус (рис. 8.5).

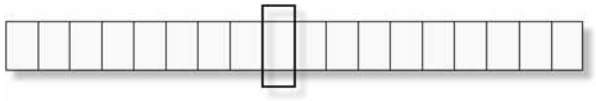

Рис. 8.5

Ответ. Покажет сообщение «Не могу».

5. минус (рис. 8.6).

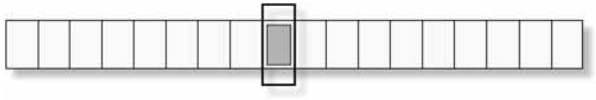

Рис. 8.6

Ответ. Символ Я.

8.2. Попробуем управлять

1. Вася написал команду **пиши** R. Корректор выдал сообщение «Не понимаю». Почему?

   Ответ. Символ R не входит в алфавит Корректора.

2. Задавая команды Корректору, напишите на ленте его имя.

Решение

```
пиши К вправо
пиши О вправо
пиши Р вправо
пиши Р вправо
пиши Е вправо
пиши К вправо
пиши Т вправо
пиши О вправо
пиши Р
```

8.3. Управление при помощи программы

1. Написать программу, которая отнимает от нечётного числа на ленте единицу. В начальный момент окно установлено на первый пустой символ справа от записи числа. На рис. 8.7 показан пример возможного начального состояния.

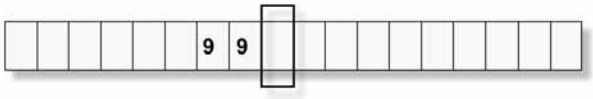

Рис. 8.7

Решение

```
это Минус1
  влево  // Установить окно на последнюю цифру
  минус  // Заменить цифру на предыдущую
конец
```

Глава 9

Язык программирования

Ответы на вопросы и задания

В тестах, рекомендованных для проверки программ, пробел обозначен знаком ■, пустая ячейка — знаком □, а подчёркнутый символ показывает положение окна на ленте.

9.1. Процедурное программирование

1. Уменьшает ли использование команды вызова процедуры количество работы для исполнителя?

   Ответ. Нет, не уменьшает, хотя размер программы с процедурами может быть существенно меньше. Каждый раз, когда интерпретатор встречает в программе вызов процедуры, он фактически заменяет его командами, содержащимися в этой процедуре. Процедурное программирование не сокращает работу исполнителя, но упрощает работу программиста, делает код программы проще и, как правило, короче.

2. Как интерпретатор будет выполнять такую программу:

```
ЭТО Работа
  Подготовка
  Выполнение
  Завершение
КОНЕЦ
```

   Ответ. Интерпретатор последовательно выполнит три процедуры: `Подготовка`, `Выполнение` и `Завершение`. Если в программе не будет обнаружено процедур с такими именами, интерпретатор покажет окно с сообщением об ошибке.

   Выполнение процедуры состоит в последовательном выполнении команд, содержащихся в её описании:

   - Команда из СКИ передаётся на выполнение исполнителю.

- ❒ Конструкция языка выполняется согласно её описанию.
- ❒ Если очередная команда снова является вызовом процедуры, интерпретатор последовательно выполняет команды, в ней содержащиеся.

9.2. Циклы

1. Текст на ленте начинается, возможно, серией пробелов. Стереть все начальные пробелы. Окно перед работой установлено на первый символ текста. На рис. 9.1 показан пример исходных данных и результат обработки.

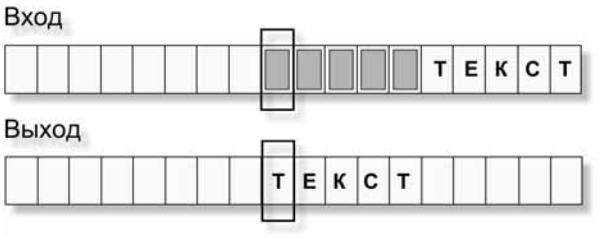

Рис. 9.1

Решение

Алгоритм. Идея алгоритма: пока виден пробел, удаляем его и двигаем окно вправо. Работа заканчивается, когда в окне появится символ, отличный от пробела.

Программа

```
ЭТО Удаление_первых_пробелов
  ПОКА ПРОБЕЛ
  {
    ПИШИ ПУСТО
    ВПРАВО
  }
КОНЕЦ
```

Комментарий. В цикле **ПОКА** проверка выполняется перед выполнением команд в теле цикла, поэтому, если пробела в окне нет с самого начала, выполнение программы заканчивается без каких-либо действий исполнителя.

Тесты

Проверим программу для текста, в котором есть символы, отличные от пробела (табл. 9.1).

Таблица 9.1

Тест	Комментарий	Ожидаемый результат
КОТ	Текст начинается не с пробела	КОТ
□КОТ	Текст начинается с одного пробела	КОТ
□□КОТ	В тексте много пробелов	КОТ

Проверим для текста, в котором кроме пробелов нет других символов (табл. 9.2).

Таблица 9.2

Тест	Комментарий	Ожидаемый результат
□	Текст состоит из одного пробела	□
□□	Текст состоит из двух пробелов	□

Проверим для случая, когда в тексте вообще нет ни одного символа (табл. 9.3).

Таблица 9.3

Тест	Комментарий	Ожидаемый результат
□	Лента пуста	□

2. На ленте записан текст, а окно установлено на первый его символ. Удалить, если они есть, все начальные и конечные пробелы в этом тексте. На рис. 9.2 показан пример исходных данных и результат обработки.

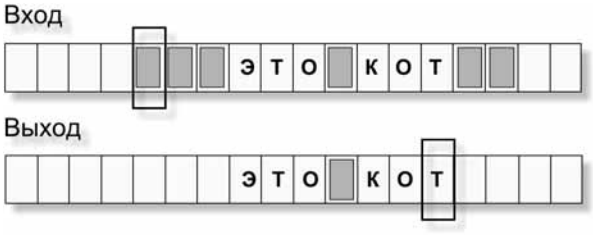

Рис. 9.2

Решение

Алгоритм

1. Удалить первые пробелы (процедура `Удаление_первых_пробелов` из решения предыдущей задачи).
2. Переместить окно на конец записи.
   2.1. Найти первый пустой символ справа за записью.
   2.2. Сместиться влево на последний символ записи.
3. Удалить конечные пробелы.
   3.1. Пока виден пробел, удаляем его и двигаем окно влево.

Программа

```
ЭТО Удаление
  Удаление_первых_пробелов
  На_конец_записи
  Удаление_конечных_пробелов
КОНЕЦ

ЭТО На_конец_записи
  ПОКА НЕ ПУСТО ВПРАВО
  ВЛЕВО
КОНЕЦ

ЭТО Удаление_конечных_пробелов
  ПОКА ПРОБЕЛ { ПИШИ ПУСТО ВЛЕВО }
КОНЕЦ
```

Тесты. В табл. 9.4 представлен минимальный набор тестов. Дополнительно стоит проверить работу программы для следующих случаев:

- ❒ внутри текста нет пробелов;
- ❒ текст состоит из одного символа, не являющегося пробелом;
- ❒ текст начинается ровно с одного пробела;
- ❒ текст заканчивается ровно одним пробелом.

Таблица 9.4

Тест	Комментарий	Ожидаемый результат
_эта□кот	Нет начальных и конечных пробелов	эта□кот_
_□□эта□кот	Начальные пробелы есть, конечных нет	эта□кот_

Таблица 9.4 *(окончание)*

Тест	Комментарий	Ожидаемый результат
ЭТО□КОТ□□	Конечные пробелы есть, начальных нет	ЭТО□КОТ
□□ЭТО□КОТ□□	Есть начальные и конечные пробелы	ЭТО□КОТ
	Текст состоит из одного пробела	□
	Текст состоит из нескольких пробелов	□
□	Лента пуста	□

3. На ленте записано число, состоящее из одних единиц (не более 9), и в окно видна первая его цифра. Записать следом за числом сумму его цифр. На рис. 9.3 показан пример исходных данных и результат обработки.

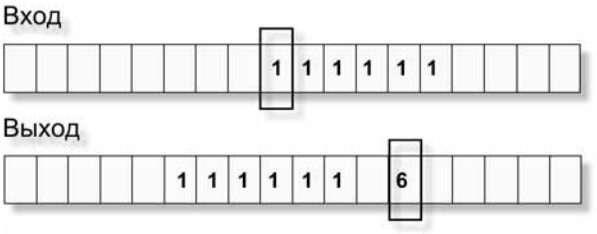

Рис. 9.3

Решение

Алгоритм. Копируем первую единицу в ящик, а затем перемещаем окно по записи, увеличивая число в ящике на каждом шаге. Для добавления единицы временно помещаем число из ящика на ленту (команда ОБМЕН), применяем команду ПЛЮС, возвращаем изменённое значение в ящик и восстанавливаем ленту (команда ОБМЕН).

1. Занесём первую единицу в ящик.
2. Добавляем остальные единицы.
3. Записываем результат.

Программа

```
ЭТО Сумма_единиц
  // Занесём первую единицу в ящик
```

```
  ЯЩИК+
  // Добавляем остальные единицы
  ВПРАВО
  ПОКА НЕ ПУСТО
  {
    ОБМЕН  // Текущую сумму поместим на ленту
    ПЛЮС   // Увеличим сумму на 1
    ОБМЕН  // Вернём сумму в ящик и восстановим ленту
    ВПРАВО // Шаг к следующей ячейке
  }
  // Записываем результат
  ВПРАВО ЯЩИК-
КОНЕЦ
```

Комментарий. Было бы неплохо, если бы программа правильно работала и для пустой записи, выдавая ответ 0. Это несложно устроить при помощи условной команды, которая обсуждается в следующем разделе этой главы.

Программа

```
ЭТО Новая_сумма_единиц
  ЕСЛИ ПУСТО
    ТО ПИШИ 0
    ИНАЧЕ Сумма_единиц
КОНЕЦ
```

Но можно предложить решение, учитывающее пустую запись, и без развилки:

```
ЭТО Сумма_единиц
  ОБМЕН ПИШИ 0 ОБМЕН // Обнулим счётчик единиц (в ящике)
  ПОКА НЕ ПУСТО      // Считаем единицы
  {
    ОБМЕН  // Текущую сумму поместим на ленту
    ПЛЮС   // Увеличим сумму на 1
    ОБМЕН  // Вернём сумму в ящик и восстановим ленту
    ВПРАВО // Шаг к следующей ячейке
  }
  // Записываем результат
  ВПРАВО ЯЩИК-
КОНЕЦ
```

Тесты. В табл. 9.5 представлен набор тестов для последнего варианта программы.

Таблица 9.5

Тест	Комментарий	Ожидаемый результат
1	Одна единица	1□1
111	Несколько единиц	111□3
111111111	Максимальное число единиц	111111111□9
□	Пустая лента	0

4. На ленте записано число, а окно установлено на первый пустой символ перед ним. Записать вместо числа сумму его цифр, предполагая, что она не больше 9. На рис. 9.4 показан пример исходных данных и результат обработки.

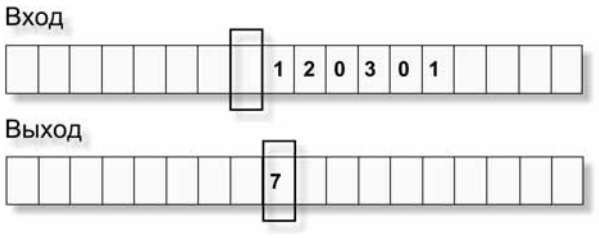

Рис. 9.4

Решение

Алгоритм. Для подсчёта суммы цифр будем использовать ящик. Сначала занесём в него число 0, а затем будем просматривать символы записи и увеличивать содержимое ящика на величины цифр из окна исполнителя.

Добавление цифры к сумме в ящике и замена её на ленте символом `пусто` будем выполнять при помощи следующего алгоритма:

1. Пока в окне не 0, делать:
   1.1. Выполнить обмен содержимого окна и ящика.
   1.2. Увеличить сумму на 1.
   1.3. Выполнить обмен содержимого окна и ящика.
   1.4. Уменьшить цифру на 1.
2. Записать в окно `пусто`.

Полный алгоритм имеет вид:

1. Занести в ящик число 0 — начальное значение суммы.
2. Сместить окно на начало записи.
3. Пока в окне не **ПУСТО**, делать:
   3.1. Пока в окне не 0, делать:
      3.1.1. Выполнить обмен содержимого окна и ящика.
      3.1.2. Увеличить сумму на 1.
      3.1.3. Выполнить обмен содержимого окна и ящика.
      3.1.4. Уменьшить цифру на 1.
   3.2. Записать в окно **ПУСТО**.
   3.3. Сместить окно на одну ячейку вправо.
4. Записать результат.

Программа

```
ЭТО Сумма_цифр
  // 1. Занести в ящик число 0 — начальное значение суммы
  ПИШИ 0 ОБМЕН
  // 2. Сместить окно на начало записи
  ВПРАВО
  // 3. Пока в окне не ПУСТО, идти по записи и добавлять цифры
  ПОКА НЕ ПУСТО
  {
    // 3.1. Пока в окне не 0, увеличивать сумму и уменьшать цифру
    ПОКА НЕ 0
    {
      ОБМЕН  // 3.1.1. Текущую сумму поместить на ленту
      ПЛЮС   // 3.1.2. Увеличить сумму на 1
      ОБМЕН  // 3.1.3. Вернуть сумму в ящик и восстановить ленту
      МИНУС  // 3.1.4. Уменьшить цифру на 1
    }
    // 3.2. Вместо нуля записать в окно ПУСТО
    ПИШИ ПУСТО
    // 3.3. Сместить окно на одну ячейку вправо
    ВПРАВО
  }
  // 4. Записать результат
  ЯЩИК-
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 9.6.

Таблица 9.6

Тест	Комментарий	Ожидаемый результат
□5	Одна ненулевая цифра в записи	5
□0	Одна нулевая цифра в записи	0
□2013	Несколько цифр в записи	6
□4005	Максимальное значение суммы	9
□	Пустая лента	0

Замечание

Программа будет правильно работать и для значений суммы, больших 9, если рассматривать результирующий символ на ленте как ответ, закодированный порядковым номером этого символа в алфавите Корректора (считая, что порядковый номер нуля есть нуль). Ограничением величины суммы теперь станет порядковый номер последнего символа в алфавите Корректора.

9.3. Развилки

1. Как вы думаете, почему по условию задачи 6 число на ленте должно быть больше 9, но меньше 99?

   Ответ. Условие сформулировано так, чтобы исходное число и результат имели ровно по две цифры.

2. На ленте Корректора сложилась такая обстановка (рис. 9.5).

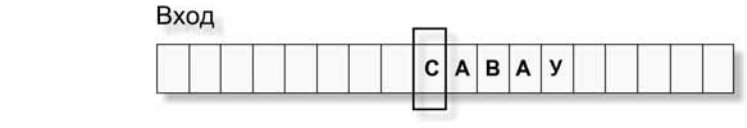

Рис. 9.5

Какое слово будет на ленте после выполнения следующей программы:

```
ЭТО Сава
  ПОВТОРИ 5
  {
    ЕСЛИ А
```

```
    ТО    ПОВТОРИ 14 ПЛЮС
    ИНАЧЕ МИНУС
    ВПРАВО
  }
КОНЕЦ
```

Ответ. На ленте окажется слово РОБОТ.

3. На ленте в соседних ячейках записаны два символа. Упорядочить их по возрастанию порядковых номеров в алфавите Корректора. В начальный момент окно установлено на первый символ. На рис. 9.6 показан пример исходных данных и результат обработки.

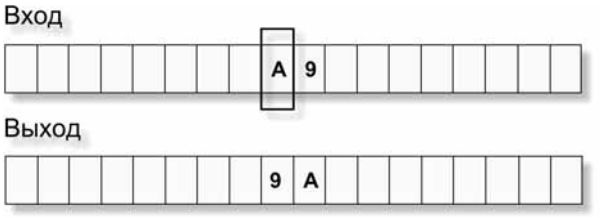

Рис. 9.6

Решение

Алгоритм. Сравним первый символ (скопировав его в ящик) со вторым и, если порядка нет, обменяем символы на ленте местами.

1. Скопируем первый символ в ящик.
2. Установим окно на второй символ.
3. Сравним символы. Если символ в ящике имеет больший порядковый номер, то поменяем символы на ленте местами. Для этого сделаем:
   3.1. Поменяем местами содержимое окна и ящика.
   3.2. Сместим окно на первый символ.
   3.3. Заменим первый символ вторым (из ящика).

Программа

```
ЭТО Порядок
  // 1. Скопируем первый символ в ящик
  ЯЩИК+
  // 2. Установим окно на второй символ
  ВПРАВО
```

```
  // 3. Сравним символы
  ЕСЛИ Я>Л ТО
  {
    // Если порядка нет, поменяем символы местами
    ОБМЕН
    ВЛЕВО
    ЯЩИК-
  }
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 9.7.

Таблица 9.7

Тест	Комментарий	Ожидаемый результат
ЮЮ	Одинаковые символы	ЮЮ
Ф5	Первый символ «больше» второго	5Ф
1Я	Первый символ «меньше» второго	1Я
В□	Один из символов равен первому символу алфавита ПУСТО	□В
@Р	Один из символов равен последнему символу алфавита @	Р@

4. В длинном коридоре много дверей без промежутков между ними. Нужно открыть все закрытые двери и закрыть все открытые. Закрытая дверь изображается на ленте цифрой 0, а открытая — цифрой 1. В начальный момент в окне видна первая дверь. На рис. 9.7 показан пример исходных данных и результат обработки.

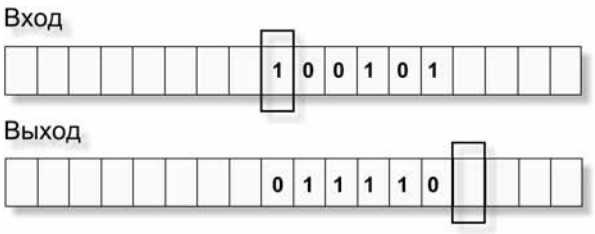

Рис. 9.7

Решение

Алгоритм. Перемещаем окно вдоль записи и заменяем 0 на 1, а 1 на 0.

1. Пока в окне не пусто, делать:
   1.1. Проверим символ:
      1.1.1. Если символ равен 0, заменим его на 1, иначе заменим его на 0.
   1.2. Переместим окно на следующую ячейку ленты.

Программа

```
ЭТО Двери
  // 1. Пока в окне не пусто, перемещаем окно по записи
  //    и преобразуем символы
  ПОКА НЕ ПУСТО
  {
    // 1.1. Проверка символа
    ЕСЛИ 0
      ТО    ПИШИ 1 // 0 меняем на 1
      ИНАЧЕ ПИШИ 0 // 1 меняем на 0
    // 1.2. Окно на следующую ячейку ленты
    ВПРАВО
  }
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 9.8.

Таблица 9.8

Тест	Комментарий	Ожидаемый результат
□	Лента пуста	□
1	Один символ 1	0□
0	Один символ 0	1□
10110	Несколько символов	01001□

9.4. Рекурсия

1. На ленте записан некоторый текст. Проверить, есть ли в нём многоточие (три и более подряд идущих знака «.»). Установить окно на начало первого многоточия, если они присутствуют в тексте или на первый пустой символ за концом текста, если многоточие отсутствует. В начальный момент в окно виден первый символ текста. На рис. 9.8 показан пример исходных данных и результат обработки.

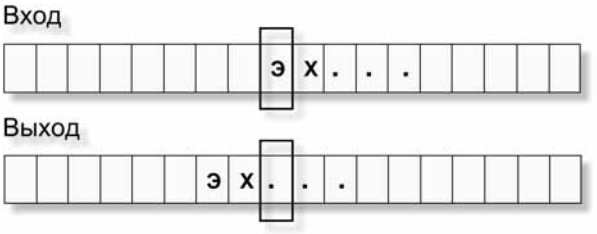

Рис. 9.8

Решение

Алгоритм. Будем рекурсивно проверять символы, пока не встретим **пусто** или три подряд идущих точки.

Алгоритм Многоточие

1. Проверим текущий символ.
   1.1. Если это символ **пусто**, то конец работы (на алгоритм Конец).
   1.2. Если это точка, то проверим, нет ли справа от неё ещё двух точек (на алгоритм Проверка).
   1.3. В противном случае продолжаем работу (на алгоритм Дальше).

Алгоритм Проверка

1. Переместим окно на следующий символ.
   1.1. Если это точка, то:
      1.1.1. Переместим окно на следующий символ.
      1.1.2. Если это точка, то:
         1.1.2.1. Сместим окно на первую из обнаруженных точек.
         1.1.2.2. Переходим на алгоритм Конец.
      1.1.3. Если это не точка, то переходим к выполнению алгоритма Дальше.
   1.2. Если это не точка, то переходим к выполнению алгоритма Дальше.

Алгоритм Дальше

1. Смещаем окно к следующему символу.
2. Переходим к алгоритму Многоточие (рекурсивное продолжение).

Алгоритм Конец

1. Исполнитель ничего не делает (пустая команда).

Программа

```
ЭТО Многоточие
  ЕСЛИ ПУСТО ТО   Конец
  ИНАЧЕ ЕСЛИ . ТО Проверка
  ИНАЧЕ           Дальше
КОНЕЦ

ЭТО Проверка
  ВПРАВО
  ЕСЛИ .
    ТО
    {
      ВПРАВО
      ЕСЛИ .
        ТО
        {
          ВЛЕВО ВЛЕВО
          Конец
        }
        ИНАЧЕ Дальше
    }
    ИНАЧЕ Дальше
КОНЕЦ

ЭТО Дальше
  ВПРАВО
  Многоточие
КОНЕЦ

ЭТО Конец
  СТОЯТЬ
КОНЕЦ
```

Замечание

В программе организована косвенная рекурсия: `Многоточие` — `Дальше` — `Многоточие`. Процедура `Многоточие` проверяет текущий символ, а процедура `Дальше` смещает окно к следующему символу. Прерывание рекурсивного цикла выполняет процедура `Конец`.

Тесты

Набор тестов приводится в табл. 9.9.

Таблица 9.9

Тест	Комментарий	Ожидаемый результат
□	Пустая лента	□
`.` `..`	Только многоточие	`.` `...`
`.` `..КОТ`	Многоточие, за которым идут другие символы	`.` `..КОТ`
`КОТ...□ЭТО`	Многоточие среди других символов	`КОТ.` `..□ЭТО`
`КОТ.□ЭТО...`	Многоточие, перед которым есть одиночная точка в тексте	`КОТ.□ЭТО.` `..`
`1..2□1...3`	Многоточие, перед которым в тексте есть пара точек	`1..2□1.` `..3`
`ЭТО□КОТ`	В тексте нет точек	`ЭТО□КОТ□`
`КОТ.□ЛИСА.`	В тексте нет многоточия, но есть отдельные точки	`КОТ.□ЛИСА.□`
`1..2□1..2`	В тексте нет многоточия, но есть пары точек	`1..2□1..2□`

2. В окно виден первый символ записи. Проверить, следуют ли символы в записи в порядке возрастания их номеров. На рис. 9.9 показан пример исходных данных и результат обработки.

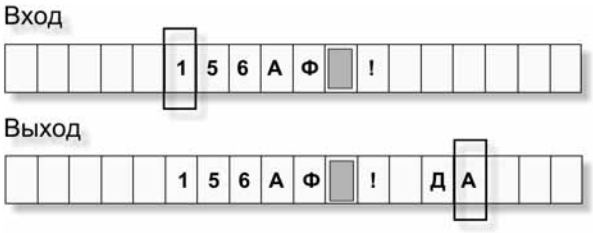

Рис. 9.9

Решение

Алгоритм. Будем проверять по паре символов, смещаясь вправо, пока не обнаружим **ПУСТО** (символы следуют в порядке возрастания номеров, ответ «да») или не обнаружим пару, которая расположена в неправильном порядке. В последнем случае для записи ответа «нет» нужно сместить окно на конец записи.

Алгоритм Проверка

1. Запомним текущий символ в ящике и сместим окно к следующему символу.
2. Проверим символ в окне:
   2.1. Если это символ **ПУСТО**, то запись ответа «да» (на алгоритм Да).
   2.2. В противном случае проверим порядок символа в ящике и символа в окне (на алгоритм Порядок).

Алгоритм Проверка

1. Сравним символ в ящике с символом в окне:
   1.1. Если они имеют правильный порядок номеров, продолжаем проверку (рекурсивно на алгоритм Проверка).
   1.2. В противном случае запишем ответ «нет» (на алгоритм Нет).

Алгоритм Да

1. Запишем ответ «да».

Алгоритм Нет

1. Сместим окно на конец записи.
2. Запишем ответ «нет».

```
ЭТО Проверка
  // Запомним текущий символ в ящике и
  // сместим окно к следующему символу
  ЯЩИК+ ВПРАВО
  // Проверим на конец записи
  ЕСЛИ ПУСТО
    ТО Да
    ИНАЧЕ Порядок
КОНЕЦ

ЭТО Порядок
  // Проверим на порядок в записи
  ЕСЛИ Я<Л
```

```
    ТО Проверка
    ИНАЧЕ Нет
КОНЕЦ

ЭТО Да
  ВПРАВО ПИШИ Д
  ВПРАВО ПИШИ А
КОНЕЦ

ЭТО Нет
  ПОКА НЕ ПУСТО ВПРАВО // На конец записи
  ВПРАВО ПИШИ Н
  ВПРАВО ПИШИ Е
  ВПРАВО ПИШИ Т
КОНЕЦ
```

Замечание

В программе организована косвенная рекурсия: Проверка — Порядок — Проверка. Процедура Проверка проверяет конец записи, а процедура Порядок — порядок символов в записи.

Тесты

Набор тестов приводится в табл. 9.10.

Таблица 9.10

Тест	Комментарий	Ожидаемый результат
□	Пустая лента	□ ДА
Р	Один символ на ленте	Р ДА
26АЯ@	Упорядоченная запись	26АЯ@ ДА
174 АЯ@	Неупорядоченная запись	174 АЯ@ НЕТ

Глава 10

Отладка программ

Ответы на вопросы и задания

В тестах, рекомендованных для проверки программ, пробел обозначен знаком ■, пустая ячейка — знаком □, а подчёркнутый символ показывает положение окна на ленте.

10.2. Синтаксические ошибки

1. Что такое синтаксическая ошибка?

   Ответ. Синтаксическая ошибка — это несоответствие текста программы правилам языка программирования.

2. Найдите синтаксические ошибки в следующей программе, не запуская её в среде Корректора.

   Программа

```
Это программа
  Запомнить первый символ
  Идти на конец слова
  Записать на место последнего
Конец


Это Запомнить первый
  Ящик +
Конец.

Это Идти на конец
  ПОКА НЕТ ПУСТА ПРАВО
  влево
```

```
Конец

ЭТО Записать на место последнего
  Ящик-
Конец
```

Решение

Ошибки в записи ключевых слов показаны в табл. 10.1.

Таблица 10.1

Запись в программе	Верное ключевое слово
Это	**ЭТО**
Конец	**КОНЕЦ**
Конец.	**КОНЕЦ**
Ящик +	**ЯЩИК+**
НЕТ	**НЕ**
ПУСТА	**ПУСТО**
ПРАВО	**ВПРАВО**
влево	**ВЛЕВО**
Ящик-	**ЯЩИК-**

Наиболее вероятные ошибки, связанные с именами процедур, показаны в табл. 10.2 и 10.3.

Таблица 10.2

Запись в программе	Возможная правильная запись
Запомнить первый символ	Запомнить_первый_символ
Идти на конец слова	Идти_на_конец_слова
Записать на место последнего	Записать_на_место_последнего

Таблица 10.3

Имя в вызове процедуры	Имя в описании процедуры
Запомнить первый символ	Запомнить первый
Идти на конец слова	Идти на конец

3. Исправьте синтаксические ошибки предыдущей программы в среде Корректора.

Решение

Исправленная программа

```
ЭТО программа
  Запомнить_первый_символ
  Идти_на_конец_слова
  Записать_на_место_последнего
КОНЕЦ


ЭТО Запомнить_первый_символ
  ЯЩИК+
КОНЕЦ

ЭТО Идти_на_конец_слова
  ПОКА НЕ ПУСТО ВПРАВО
  ВЛЕВО
КОНЕЦ

ЭТО Записать_на_место_последнего
  ЯЩИК-
КОНЕЦ
```

4. Что должна делать программа, описанная в вопросе 2?

   Ответ. Эта программа копирует первый символ в записи на место последнего.

10.3. Ошибки программирования

1. Что такое семантическая ошибка?

   Ответ. Семантическая ошибка — это смысловая ошибка в синтаксически правильной программе.

2. Укажите ошибки в решении указанной далее задачи.

   Задача

   Написать программу, которая увеличивает целое неотрицательное число на 1. В начальный момент в окно видна первая слева цифра числа.

Решение задачи

Алгоритм

1. Переместить окошко на младшую цифру числа.
2. Увеличить младшую цифру на 1.

Программа

```
ЭТО Плюс_1
  На_младшую_цифру
  Добавить_1
КОНЕЦ

ЭТО На_младшую_цифру
  ПОКА НЕ ПУСТО ВПРАВО
КОНЕЦ

ЭТО Добавить_1
  ПЛЮС
КОНЕЦ
```

Решение

Алгоритмическая ошибка. Если младшая цифра — 9, то алгоритм работать не будет.

Семантическая ошибка. Процедура `На_младшую_цифру` устанавливает окно не на младшую цифру числа, а на пустую ячейку сразу за ней.

10.4. Тестирование

Предложите наборы тестов для программ, решающие следующие задачи.

1. Скопировать первый символ на место последнего в записи.

Решение

Решение приводится в табл. 10.4.

Таблица 10.4

Тест	Комментарий	Ожидаемый результат
КОТ	Типичные данные	КОК
12	Типичные данные	11
1	Особый случай (один символ в записи)	1
□	Особый случай (нет символов в записи)	□

2. Увеличить целое неотрицательное число на 1.

Решение

Решение приводится в табл. 10.5.

Таблица 10.5

Тест	Комментарий	Ожидаемый результат
1347	Типичные данные	1348
7589	Последняя цифра — 9	7590
453999	Число оканчивается несколькими девятками	454000
9999	Число состоит из одних девяток	10000

3. Уменьшить целое положительное число на 1.

Решение

Решение приводится в табл. 10.6.

Таблица 10.6

Тест	Комментарий	Ожидаемый результат
1347	Типичные данные	1346
7580	Последняя цифра — 0	7579
453000	Число оканчивается несколькими нулями	452999
10000	Число представляет собой единицу с несколькими нулями	9999

4. Сосчитать число пробелов в тексте.

Решение

Решение приводится в табл. 10.7.

Таблица 10.7

Тест	Комментарий	Ожидаемый результат
ТЕКСТ□С□ПРОБЕЛАМИ	Типичные данные	2
□□□ТЕКСТ	Пробелы в начале текста	3

Таблица 10.7 *(окончание)*

Тест	Комментарий	Ожидаемый результат
ТЕКСТ□□	Пробелы в конце текста	2
□□□□ □	Текст из одних пробелов	4 1
ТЕКСТ Т	Текст без пробелов	0
□	Пустая лента	0
□□ТЕКСТ□□С□ПРОБЕЛАМИ□□□	Пробелы в начале, в середине и в конце текста	8
1□2□□□3□□4□□56□□7□□8	Число пробелов больше 9	12

5. Удалить все пробелы в тексте.

Решение

Решение приводится в табл. 10.8.

Таблица 10.8

Тест	Комментарий	Ожидаемый результат
1□23□7	Типичные данные	1237
□□7580□ □75□□80□□ □□75□□□80□□	Пробелы на первых и последних местах	7580 7580 7580
453000	В записи нет пробелов	453000
□ □□□□□	Текст состоит из одних пробелов	□ □
□	В тексте нет символов	□

6. Найти в записи символ, имеющий максимальный номер в алфавите Корректора.

Решение

Решение приводится в табл. 10.9.

Таблица 10.9

Тест	Комментарий	Ожидаемый результат
ЛИСА	Типичные данные	С
@ЛИСА	Искомый символ на первом месте	@
ЛИСА@	Искомый символ на последнем месте	@
А	Текст состоит из одного символа	А
□	В тексте нет символов	□

7. Упорядочить символы в записи в порядке неубывания их номеров в алфавите Корректора.

Решение

Решение приводится в табл. 10.10.

Таблица 10.10

Тест	Комментарий	Ожидаемый результат
МОЛОКО 743534	Типичные данные	КЛМООО 334457
123456789	Символы в порядке неубывания алфавитных номеров	123456789
987654321	Символы в порядке невозрастания порядковых номеров	123456789
А	Текст состоит из одного символа	А
□	В тексте нет символов	□

8. На ленте через пробел записаны два числа. Установить окно на первую цифру наибольшего из них.

Решение

Решение приводится в табл. 10.11.

Таблица 10.11

Тест	Комментарий	Ожидаемый результат
325□34 743□534 8□1	Первое число больше	325□34 743□534 8□1

Таблица 10.11 *(окончание)*

Тест	Комментарий	Ожидаемый результат
787□4646 67□85 1□5	Второе число больше	787□4646 67□85 1□5
35□35	Числа равны	35□35или 35□35
12345□1745 12745□12345 12349□12345	Числа имеют одинаковую длину и несколько первых равных цифр. Тест необходим для проверки той части алгоритма, которая будет сравнивать по разрядам числа одинаковой длины	12345□1745 12745□12345 12349□12345

9. На ленте записано целое десятичное число. Проверить, нет ли ошибок в его записи. Если ошибки есть, написать за текстом слово ОШИБКА и установить окно на первую слева ошибку. В противном случае записать за числом сообщение ВЕРНО.

Решение

Тесты для правильных записей в табл. 10.12.

Таблица 10.12

Тест	Комментарий	Ожидаемый результат
325 7 15748463269	Число без знака	325□ВЕРНО 7□ВЕРНО 15748463269□ВЕРНО
+325 +7 +15748463269	Число со знаком +	+325□ВЕРНО +7□ВЕРНО +15748463269□ВЕРНО
–325 –7 –15748463269	Число со знаком –	–325□ВЕРНО –7□ВЕРНО –15748463269□ВЕРНО

Тесты для записей с ошибками в табл. 10.13.

Таблица 10.13

Тест	Комментарий	Ожидаемый результат
32А5 +Ф156 –7676* Щ	Число содержит не цифру	32А5□ОШИБКА +Ф156□ОШИБКА –7676*□ОШИБКА Щ□ОШИБКА
+ –	Запись содержит только знак числа. Ошибочным должен быть отмечен первый пустой символ за знаком (на этом месте ожидается цифра, а её нет)	+□ОШИБКА –□ОШИБКА
□	Запись пуста	□ОШИБКА
3+25 7–	Знак числа не на месте	3+25□ОШИБКА 7–□ОШИБКА
++36843 ––6236 +–7283 –+6889	Несколько знаков перед набором цифр	++36843□ОШИБКА ––6236□ОШИБКА +–7283□ОШИБКА –+6889□ОШИБКА
12+34–56 +12.48А	Несколько разных ошибок (проверить, правильно ли определяется первая слева ошибка)	12+34–56□ОШИБКА +12.48А□ОШИБКА

10. На ленте записана десятичная дробь. Проверить, нет ли ошибок в записи. Установить окно на первую ошибку или на первую пустую ячейку за записью, если ошибок нет.

Решение

Тесты для правильных записей в табл. 10.14.

Таблица 10.14

Тест	Комментарий	Ожидаемый результат
3.25 .7 157.	Дробь без знака	3.25□ .7□ 157.□

Таблица 10.14 *(окончание)*

Тест	Комментарий	Ожидаемый результат
+3.25 +.7 +15748463269.	Дробь со знаком +	+3.25□ +.7□ +15748463269.□
–3.25 –.7 –15748463269.	Дробь со знаком –	-3.25□ –.7□ –15748463269.□

Тесты для записей с ошибками в табл. 10.15.

Таблица 10.15

Тест	Комментарий	Ожидаемый результат
22323 –7999 +7999	Дробь не содержит десятичной точки. Ошибочным должен быть отмечен первый пустой символ за числом (на этом месте ожидалась точка, а её нет)	22323□ –7999□ +7999□
3.2A5 +Ф1.56 –.7676*	Дробь содержит не цифру	3.2A5 +Ф1.56 –.7676*
+ –	Запись содержит только знак числа. Ошибочным должен быть отмечен первый пустой символ за знаком (на этом месте ожидается цифра или точка, а их нет)	+□ –□
+. –.	Запись содержит только знак с точкой	+.□ –.□
□	Запись пуста	□
3+2.5 .7– .+8 .–65	Знак числа не на месте	3+2.5 .7– .+8 .–65

Таблица 10.15 *(окончание)*

Тест	Комментарий	Ожидаемый результат
.788.8 ..868 466.. 777..878	Несколько десятичных точек	.788.8 ..868 466.. 777..878
++36.843 −−.6236 +−7283. −+68.89	Несколько знаков перед дробью	++36.843 −−.6236 +−7283. −+68.89

10.5. Задачи

1. Проверить, есть ли в записи на ленте цифры. В начальный момент в окно виден первый символ записи. На рис. 10.1 показан пример исходного и результирующего состояний среды.

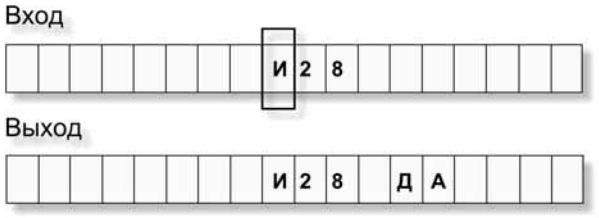

Рис. 10.1

Решение

Алгоритм

1. Занести в ящик символ 0 (признак отсутствия цифр в записи).
2. Пока в окне не `пусто`, делать:
   2.1. Проверять каждый символ записи. Если символ — цифра, записать в ящик символ 1 (признак присутствия цифры в записи).
3. Проверить ящик:
   3.1. Если в ящике символ 0, записать на ленту НЕТ.
   3.2. В противном случае записать на ленту ДА.

Программа

```
ЭТО Проверка
  // 1. Занести в ящик символ 0 (признак отсутствия цифр в записи)
  Установить_ящик
  // 2. Проверять каждый символ записи
  ПОКА НЕ ПУСТО { Проверить_символ ВПРАВО }
  // 3. Проверить ящик и записать ответ
  Записать_ответ
КОНЕЦ

ЭТО Установить_ящик
  ОБМЕН ПИШИ 0 ОБМЕН
КОНЕЦ

ЭТО Проверить_символ
  ЕСЛИ ЦИФРА ТО { ОБМЕН ПИШИ 1 ОБМЕН }
КОНЕЦ

ЭТО Записать_ответ
  ВПРАВО ЯЩИК-
  ЕСЛИ 0
    ТО
    {
      ПИШИ Н ВПРАВО
      ПИШИ Е ВПРАВО
      ПИШИ Т
    }
    ИНАЧЕ
    {
      ПИШИ Д ВПРАВО
      ПИШИ А
    }
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 10.16.

Таблица 10.16

Тест	Комментарий	Ожидаемый результат
?%?23;A	В середине записи есть цифры	?%?23;A□ДА
3□ЖУКА	Запись начинается с цифры	3□ЖУКА□ДА
КЛЕТКА□3	Запись кончается цифрой	КЛЕТКА□3□ДА
76464674	В записи только цифры	76464674□ДА
6	Запись состоит из одной цифры	6□ДА
Ф	Запись состоит из одного символа — не цифры	Ф□НЕТ
МОЛОКО	Длинная запись не содержит цифр	МОЛОКО□НЕТ
□	Запись пуста	□НЕТ

2. На ленте записано два непустых символа. Один из них виден в окошке, второй расположен где-то справа от окна. Написать программу, которая устанавливает символы в соседних клетках. На рис. 10.2 показан пример исходного и результирующего состояний среды.

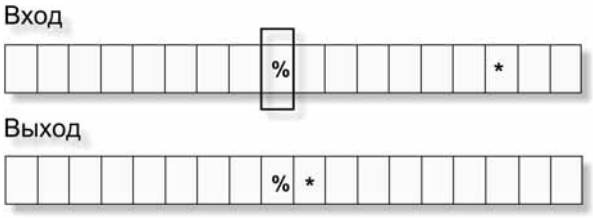

Рис. 10.2

Решение

Алгоритм

1. Искать второй символ, перемещая окно вправо.
2. Занести символ в ящик.
3. Искать первый символ, перемещая окно влево.
4. Записать символ из ящика рядом с символом на ленте.

Программа

```
ЭТО Уплотнить
  Искать_второй_символ
  Второй_символ_в_ящик
```

```
  Искать_первый_символ
  Записать_на_ленту_второй_символ
КОНЕЦ

ЭТО Искать_второй_символ
  ВПРАВО
  ПОКА ПУСТО ВПРАВО
КОНЕЦ

ЭТО Второй_символ_в_ящик
  ОБМЕН
КОНЕЦ

ЭТО Искать_первый_символ
  ПОКА ПУСТО ВЛЕВО
КОНЕЦ

ЭТО Записать_на_ленту_второй_символ
  ВПРАВО
  ОБМЕН
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 10.17.

Таблица 10.17

Тест	Комментарий	Ожидаемый результат
А□□□Б	Между символами несколько пустых ячеек	АБ
1□2	Между символами одна пустая ячейка	12
±1	Между символами нет пустых ячеек	+1

3. Записать в порядке убывания алфавитных номеров все символы Корректора, начиная с того, который в начальный момент виден в окошке, до символа ПУСТО. На рис. 10.3 показан пример исходного и результирующего состояний среды.

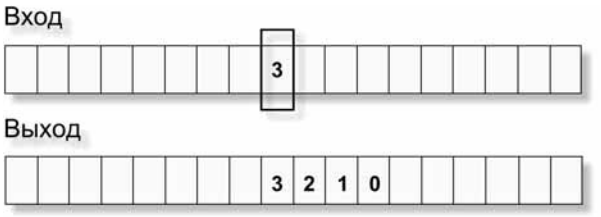

Рис. 10.3

Решение

Алгоритм

1. Пока ячейка в окне не пуста, делать:
   1.1. Запомнить содержимое окна в ящике.
   1.2. Сдвинуть окно вправо.
   1.3. Записать символ из ящика на ленту.
   1.4. Записать на ленту символ с алфавитным порядковым номером на единицу меньше.

Программа

```
ЭТО Вход
  ПОКА НЕ ПУСТО
  {
    ЯЩИК+
    ВПРАВО
    ЯЩИК-
    МИНУС
  }
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 10.18.

Таблица 10.18

Тест	Комментарий	Ожидаемый результат
□	Пустая лента	□
0	На ленте символ 0	0□
A	На ленте один из символов алфавита	A9876543210□

4. Записать в порядке возрастания алфавитных номеров все символы Корректора, начиная с символа **ПУСТО** и кончая тем, который в начальный момент виден в окошке. На рис. 10.4 показан пример исходного и результирующего состояний среды.

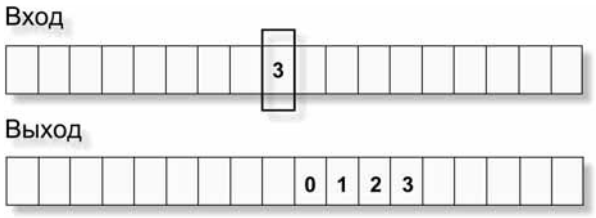

Рис. 10.4

Решение

Алгоритм. Алгоритм решения этой задачи совпадает с алгоритмом решения задачи 3 за исключением направления сдвига окна по ленте: в предыдущей задаче сдвиг выполнялся вправо, здесь выполняется влево.

1. Пока ячейка в окне не пуста, делать:
   1.1. Обменять содержимое окна и ленты.
   1.2. Сдвинуть окно влево.
   1.3. Записать символ из ящика на ленту.
   1.4. Записать на ленту символ с алфавитным порядковым номером на единицу меньше.

Программа

```
ЭТО Вход
  ПОКА НЕ ПУСТО
  {
    ЯЩИК+
    ВЛЕВО
    ЯЩИК-
    МИНУС
  }
КОНЕЦ
```

Комментарий. Приведённый выше код использует особенности среды Корректора и кажется немного «нечестным». Далее приводится другой вариант решения этой задачи. Символ в окне сначала рекурсивно

«уменьшается» до ПУСТО, затем рекурсивная пружинка записывает на ленту нужную последовательность:

```
ЭТО Вход
  ЕСЛИ НЕ ПУСТО ТО { МИНУС Вход Запись }
КОНЕЦ


ЭТО Запись
  ЯЩИК+ ВПРАВО ЯЩИК- ПЛЮС
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 10.19.

Таблица 10.19

Тест	Комментарий	Ожидаемый результат
□	Пустая лента	□
0	На ленте символ 0	□0
A	На ленте один из символов алфавита	□0123456789A

Глава 11

Приёмы программирования Корректора

Алгоритмические формулы

Новичку может показаться, что программирование — это специфический вид искусства, построенного на сплошной хитрости: программисту дают задачу, он ждёт озарения, а затем записывает программу, руководствуясь интуицией, вдохновением, и получает уникальный код, сродни неповторимым художественным полотнам.

Конечно, мы не будем отрицать весомое значение хитрости в программистском деле, равно как и в любом другом. Но фундаментом программирования является технология, а искусство накладывается на производственный процесс, как штукатурка на кирпичные стены.

Программист должен быть способен без всякой хитрости написать алгоритм решения задачи, используя универсальные алгоритмические приёмы. Эта работа сродни математическому построению, в котором решение задачи записывается в виде цепочки математических формул.

В инструментальном чемоданчике программиста есть свои универсальные технологические приёмы, которые мы будем называть *алгоритмическими формулами*. Программист должен знать эти формулы и уметь собирать из них алгоритм без всякой хитрости, подобно строителю, который возводит стену без тяжёлых раздумий над каждым кирпичом, который он добавляет на кладку.

Технология — основа мастерства.

Сборка решения из набора универсальных алгоритмических формул позволяет ускорить разработку и, что очень важно, позволяет получать легко читаемые, а значит, легко сопровождаемые программы.

Искусство (хитрость) проявляется как желание улучшить универсальный алгоритм, учитывая особенности конкретной задачи.

Хитрость позволяет получать более эффективные (по длине кода; по времени выполнения) решения, но её оборотная сторона — плохо читаемые, трудно отлаживаемые и тяжело сопровождаемые программы.

Когда хитрость связана с изменением структуры алгоритмического построения, но не меняет сами алгоритмические формулы, программисты идут на неё с лёгкой душой. Всё, что нужно сделать в таком случае — это понятно описать придуманный уникальный алгоритм, и читатель программы не запутается в кодах: ведь они по-прежнему построены из стандартных кирпичиков.

Когда хитрость связана с изменением алгоритмической формулы, она опасна: читабельность программы ухудшается, а значит, ухудшаются и другие свойства продукта, связанные с его отладкой и сопровождением.

Если программист вынужден пойти на изменение универсальной алгоритмической формулы ради значительного повышения эффективности работы программы, то он должен каждое такое изменение снабдить подробными комментариями, чтобы хоть как-то помочь отладке и сопровождению.

Когда алгоритмическая формула меняется не ради ощутимых прибавок эффективности программы, то такое изменение должно рассматриваться как признак профессиональной несостоятельности автора кода.

Перейдём от общих слов к обсуждению конкретных алгоритмических формул, которые настоятельно рекомендуем использовать при записи программ.

В числовых расчётах, иллюстрирующих алгоритмические формулы, будем использовать символы алфавита Корректора, считая, что каждому символу ставится в соответствие его порядковый номер в алфавите. Нумерацию будем начинать с символа «0», присвоив ему номер 0 (табл. 11.1).

Таблица 11.1

Символ	**Соответствующее число**
0	0
1	1
2	2
...	...
9	9
A	10
Б	11
B	12
...	...

Будем считать, что результаты вычислений (конечные и промежуточные) не выходят из диапазона «символьных» чисел Корректора.

Обработка записи известной длины n

Исходное состояние: окно установлено на первый символ записи.

Алгоритмическая формула

```
Начальные_установки
ПОВТОРИ n
{
  Обработка_символа
  Переход_к_следующему_символу
}
```

Пример

Запись на ленте состоит из 9 символов. Подсчитать число цифр в этой записи, если в начальный момент окно установлено на первый символ записи. Пример начального и конечного состояния среды показан на рис. 11.1.

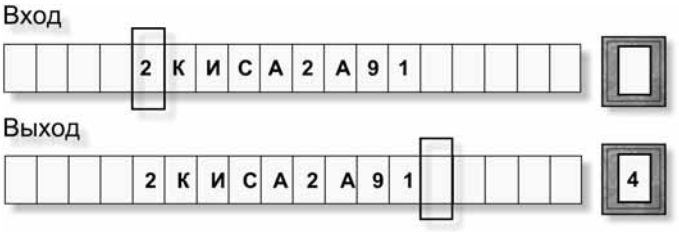

Рис. 11.1

Решение

```
// Обнуление счётчика
  ОБМЕН ПИШИ 0 ОБМЕН
// Просмотр записи
ПОВТОРИ 9
{
  // Если символ - цифра, добавить к счётчику 1
  ЕСЛИ ЦИФРА ТО { ОБМЕН ПЛЮС ОБМЕН }
  // Окно на следующий символ записи
  ВПРАВО
}
// Результат расположен в ящике
```

Здесь:

```
Начальные_установки = ОБМЕН ПИШИ 0 ОБМЕН
Обработка_символа = ЕСЛИ ЦИФРА ТО { ОБМЕН ПЛЮС ОБМЕН }
Переход_к_следующему_символу = ВПРАВО
```

Обработка записи неизвестной длины

Исходное состояние: окно установлено на первый символ записи.

Алгоритмическая формула

```
Начальные_установки
ПОКА НЕ ПУСТО
{
  Обработка_символа
  Переход_к_следующему_символу
}
```

Пример

Подсчитать число цифр в записи, если в начальный момент окно установлено на первый её символ. Пример начального и конечного состояния среды показан на рис. 11.2.

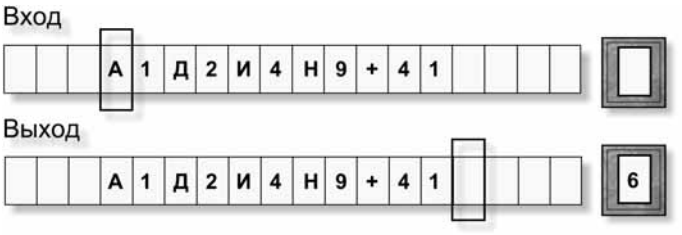

Рис. 11.2

Решение

```
// Обнуление счётчика
ОБМЕН ПИШИ 0 ОБМЕН
// Просмотр записи
ПОКА НЕ ПУСТО
{
  // Если символ - цифра, добавить к счётчику 1
  ЕСЛИ ЦИФРА ТО { ОБМЕН ПЛЮС ОБМЕН }
  // Окно на следующий символ записи
  ВПРАВО
}
// Результат расположен в ящике
```

Первая попытка изменить формулу

```
ОБМЕН ПИШИ 0 ОБМЕН
ПОКА НЕ ПУСТО Работа
...

ЭТО Работа
  ЕСЛИ ЦИФРА ТО { ОБМЕН ПЛЮС ОБМЕН }
  ВПРАВО
КОНЕЦ
```

Комментарий. Чтобы проверить, не забыл ли программист закодировать смещение окна к следующему символу, нужно анализировать процедуру `Работа`. Вывод: плохо.

Вторая попытка изменить формулу

```
ОБМЕН ПИШИ 0 ОБМЕН
ПОКА НЕ ПУСТО
{
  ЕСЛИ ЦИФРА
    ТО    Работа
    ИНАЧЕ ВПРАВО
}
...

ЭТО Работа
  ОБМЕН ПЛЮС ОБМЕН
  ВПРАВО
КОНЕЦ
```

Комментарий. По внешнему виду цикла можно предположить, что переход к следующему символу выполняется только в том случае, если текущий символ не является цифрой. Вывод: очень плохо.

Поиск объекта, его изменение и возврат в исходное положение

Алгоритмическая формула

```
ЭТО Поиск
  Шаг_к_объекту
  ЕСЛИ НЕ ОБЪЕКТ
```

```
    ТО Поиск
    ИНАЧЕ { Изменение_объекта Окно_под_пружинку }
  Обратный_шаг
КОНЕЦ
```

Пример

В записи на ленте, возможно, встречаются цифры. Записать их отдельно через пустую ячейку справа от исходной записи. В начальный момент окно установлено на первый символ записи. Пример начального и конечного состояния среды показан на рис. 11.3.

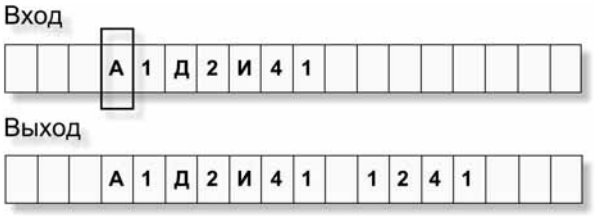

Рис. 11.3

Решение

Сначала применим алгоритмическую формулу для обработки записи неизвестной длины:

```
ЭТО Вход
  ПОКА НЕ ПУСТО
  {
    ЕСЛИ ЦИФРА ТО { ЯЩИК+ Дописать } // Дописать цифру к результату
    ВПРАВО // К следующему символу исходной записи
  }
КОНЕЦ
```

Процедура `Дописать` будет выполнять поиск объекта (результирующую последовательность с цифрами), его изменение (приписывание очередной цифры из исходной записи) и возврат в исходное положение в исходной записи:

```
ЭТО Дописать
  ВПРАВО          // Шаг к объекту
  ЕСЛИ НЕ ПУСТО   // ОБЪЕКТ — символ ПУСТО за исходной записью
    ТО Дописать
    ИНАЧЕ { Изменение_объекта Окно_под_пружинку }
  ВЛЕВО           // Обратный шаг
```

```
КОНЕЦ

ЭТО Изменение_объекта
  // Найдём первую пустую ячейку за концом записи с цифрами
  ВПРАВО
  ПОКА НЕ ПУСТО ВПРАВО
  // Допишем цифру из ящика
  ЯЩИК-
КОНЕЦ

ЭТО Окно_под_пружинку
  ВЛЕВО
  ПОКА НЕ ПУСТО ВЛЕВО
КОНЕЦ
```

Подсчёт двух количеств за один проход

Окно установлено на пустую ячейку перед записью.

Алгоритмическая формула

Первое количество будем подсчитывать в ящике исполнителя, второе запоминать отложенной в рекурсии командой.

```
ЭТО Подсчеты
  // Обнуление первого счётчика (ящика)
  ОБМЕН ПИШИ 0 ОБМЕН
  // Подсчёт двух количеств и запись результата
  Работа
КОНЕЦ

ЭТО Работа
  ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО
    {
      ЕСЛИ       Объект1 ТО { Изменить_счетчик1 Работа }
      ИНАЧЕ ЕСЛИ Объект2 ТО { Работа Изменить_счетчик2 }
      ИНАЧЕ                 Работа
    }
    ИНАЧЕ
```

```
    {
      ВПРАВО // Отступить от исходной записи вправо
      ЯЩИК+  // Записать значение первого счётчика
      ВПРАВО // Сместить окно в ячейку второго счётчика
      ПИШИ 0 // Обнулить второй счётчик
    }
КОНЕЦ
```

Пример 1

Подсчитать в записи на ленте число единиц и двоек. Результат вывести на ленту через пустую ячейку от записи в двух соседних ячейках. Пример начального и конечного состояния среды показан на рис. 11.4.

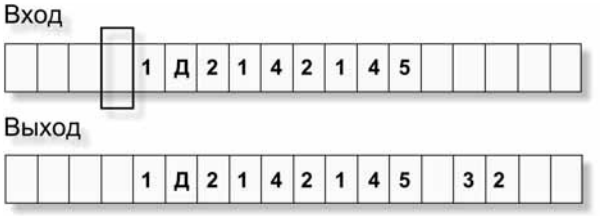

Рис. 11.4

Решение

```
ЭТО Подсчеты
  // Обнуление первого счётчика (ящика)
  ОБМЕН ПИШИ 0 ОБМЕН
  // Подсчёт двух количеств и запись результата
  Работа
КОНЕЦ

ЭТО Работа
  ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО
    {
      ЕСЛИ       1 ТО { Изменить_счетчик1 Работа }
      ИНАЧЕ ЕСЛИ 2 ТО { Работа Изменить_счетчик2 }
      ИНАЧЕ             Работа
    }
    ИНАЧЕ
```

```
    {
      ВПРАВО // Отступить от исходной записи вправо
      ЯЩИК-  // Записать значение первого счётчика
      ВПРАВО // Сместить окно в ячейку второго счётчика
      ПИШИ 0 // Обнулить второй счётчик
    }
КОНЕЦ

ЭТО Изменить_счетчик1
  ОБМЕН ПЛЮС ОБМЕН
КОНЕЦ

ЭТО Изменить_счетчик2
  ПЛЮС
КОНЕЦ
```

Пример 2

На ленте записано число. Подсчитать сумму его цифр и количество цифр 5 в нём. В начальный момент окно расположено слева от записи. Пример начального и конечного состояния среды показаны на рис. 11.5.

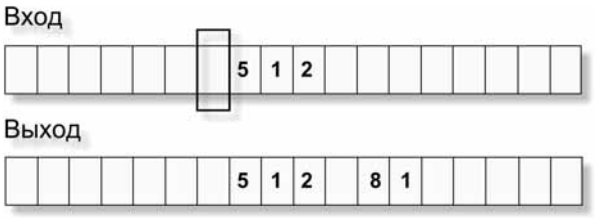

Рис. 11.5

Решение

```
ЭТО Подсчеты
  // Обнуление первого счётчика (ящика) — для подсчёта суммы цифр
  ОБМЕН ПИШИ 0 ОБМЕН
  // Подсчёт двух количеств и запись результата
  Работа
КОНЕЦ

ЭТО Работа
  ВПРАВО
```

```
  ЕСЛИ НЕ ПУСТО
    ТО
    {
      ЕСЛИ 5
      ТО
      { Изменить_счетчик1 Работа Изменить_счетчик2 }
      ИНАЧЕ
      { Изменить_счетчик1 Работа }
    }
    ИНАЧЕ
    {
      ВПРАВО // Отступить от исходной записи вправо
      ЯЩИК-  // Записать значение первого счётчика
      ВПРАВО // Сместить окно в ячейку второго счётчика
      ПИШИ 0 // Обнулить второй счётчик
    }
КОНЕЦ

ЭТО Изменить_счетчик2
  ПЛЮС
КОНЕЦ

ЭТО Изменить_счетчик1 // Добавить к ящику очередную цифру
  ЕСЛИ НЕ 0
    ТО
    {
      МИНУС ОБМЕН ПЛЮС ОБМЕН
      Изменить_счетчик1
      ПЛЮС  // Восстановить цифру в исходной записи
    }
КОНЕЦ
```

Вставка символа в запись

Алгоритмическая формула

```
// Вставка символа из ящика в запись.
// Место вставки показывает окно (любой символ записи
// или первый символ за записью).
```

```
// Вход: ЗАИСЬ (в ящике вставляемый символ П)
// Выход: ЗАПИСЬ (в ящике ПУСТО)
// Идея В. П. Семенко
ЭТО Вставка
  ОБМЕН ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО    Вставка
    ИНАЧЕ ОБМЕН
  ВЛЕВО // Пружинка для возврата на вставленный символ
КОНЕЦ
```

Комментарий. Символ из ящика записывается на ленту, а символ с ленты копируется в ящик. Теперь надо вставить символ из ящика в ячейку справа — отчётливо видны циклические действия. Повторение заканчивается, когда окно смещается на пустую ячейку за записью:

ЗАИСЬ (в ящике П)

ЗАПСЬ (в ящике И)

ЗАПИЬ (в ящике С)

ЗАПИС□ (в ящике Ь)

ЗАПИСЬ (в ящике □) — работа ветви **ИНАЧЕ**

ЗАПИСЬ (в ящике □) — работа рекурсивной пружинки **ВЛЕВО**

Процедура использует рекурсивный цикл, а не цикл **ПОКА**, чтобы вернуть окно на место вставки после завершения сдвига остатка записи вправо.

Удаление символа из записи

Алгоритмическая формула

```
// Удаление символа записи.
// Место удаления показывает окно (любой символ записи).
// После удаления окно смещено на одну позицию влево по
// отношению к исходному положению, а в ящике - удалённый символ.
// Вход: ЗАКПИСЬ
// Выход: ЗАПИСЬ (в ящике удалённый символ К)
// Идея В. П. Семенко
ЭТО Удаление
  ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО Удаление
```

```
    ИНАЧЕ { ЯЩИК+ ВЛЕВО }
  ОБМЕН ВЛЕВО // Пружинка для сдвига остатка на одну ячейку влево
КОНЕЦ
```

Комментарий. Рекурсивный цикл находит конец записи, а рекурсивная пружинка смещает остаток записи на одну ячейку влево:

ЗАКПИСЬ

ЗАКПИСЬ□ — работа рекурсивного цикла

ЗАКПИСЬ (в ящике ПУСТО) — работа ветви ИНАЧЕ

ЗАКПИС□ (в ящике Ь) — здесь и далее работа рекурсивной пружинки

ЗАКПИЬ□ (в ящике С)

ЗАКПСЬ□ (в ящике И)

ЗАКИСЬ□ (в ящике П)

ЗАПИСЬ□ (в ящике К)

Определение чётности суммы слагаемых

Исходное состояние: окно установлено на первый символ плотной последовательности символьных чисел. Определить чётность суммы этих чисел.

Комментарий. В приведённом далее варианте кода суммируются не сами символьные числа, а их остатки от деления на 2, что позволяет существенно расширить область исходных данных, для которых код будет работать без выхода из диапазона символьных чисел Корректора.

Вариант кода

```
// Определение чётности суммы слагаемых
// Вход: окно на первом символьном числе последовательности
// Выход: окно в ячейке за концом исходной записи
// Чётность суммы определяется по содержимому этой ячейки:
// 0 - сумма чётное число
// 1 - сумма нечётное число
ЭТО Четность_суммы
  Обнулить_счетчик
  Вычислить_сумму_остатков
  Определить_четность_суммы
КОНЕЦ


ЭТО Обнулить_счетчик
  ОБМЕН ПИШИ 0 ОБМЕН
```

```
КОНЕЦ

ЭТО Вычислить_сумму_остатков
  ПОКА НЕ ПУСТО { Добавить ВПРАВО }
КОНЕЦ

// Добавить остаток от деления на 2
ЭТО Добавить
  Остаток_от_деления_на_2
  ЕСЛИ 1 ТО { ОБМЕН ПЛЮС ОБМЕН }
КОНЕЦ

ЭТО Остаток_от_деления_на_2
  ЕСЛИ НЕ 0 ТО ЕСЛИ НЕ 1 ТО { МИНУС МИНУС Остаток_от_деления_на_2 }
КОНЕЦ

ЭТО Определить_четность_суммы
  ЯЩИК- // Сумму остатков записать на ленту
  Остаток_от_деления_на_2
КОНЕЦ
```

Недостатки приведённого кода:

- ❒ исходная запись на ленте будет запорчена;
- ❒ при длинной исходной записи возможен выход за диапазон символьных чисел Корректора при подсчёте суммы остатков от деления на 2.

Первый недостаток легко исправляется рекурсивной пружинкой **ПЛЮС ПЛЮС** при вычислении остатка от деления на 2 символьного числа.

Вторую неприятность можно преодолеть, используя ящик не для суммирования остатков, а для хранения признака чётности текущей суммы:

- ❒ 0 — текущая сумма чётна;
- ❒ 1 — текущая сумма нечётна.

В итоге получается алгоритмическая формула.

Алгоритмическая формула

```
// Определение чётности суммы слагаемых
// Вход: окно установлено на первый символ плотной
// последовательности символьных чисел.
// Выход: окно в ячейке за концом исходной записи
```

```
// Чётность суммы определяется по содержимому этой ячейки (или ящика):
// 0 - сумма чётное число
// 1 - сумма нечётное число
ЭТО Четность_суммы
  ОБМЕН ПИШИ 0 ОБМЕН // Установить начальное значение признака чётности
  ПОКА НЕ ПУСТО { Текущая_четность ВПРАВО } // Вычислить чётность суммы
  ЯЩИК- // Записать признак чётности из ящика на ленту
КОНЕЦ
// Вычислить чётность текущей суммы
ЭТО Текущая_четность
  ЕСЛИ НЕ 0 ТО ЕСЛИ НЕ 1
    ТО
    {
      МИНУС МИНУС
      Текущая_четность
      ПЛЮС ПЛЮС // Восстановление числа на ленте
    } // Вычислить текущую чётность перед восстановлением числа на ленте
    ИНАЧЕ Изменить_признак
КОНЕЦ

// Изменение признака чётности текущей суммы в ящике
ЭТО Изменить_признак
  ЕСЛИ 1
    ТО
    {
      ОБМЕН
      ЕСЛИ 0
        ТО ПИШИ 1
        ИНАЧЕ ПИШИ 0
      ОБМЕН
    }
КОНЕЦ
```

Замечание

Если в главной процедуре `Четность_суммы` убрать последнюю команду **ЯЩИК-**, то признак чётности суммы будет содержать только ящик (без записи результата на ленту).

Алгоритмические формулы для логических выражений

На момент написания этой книги в языке Кукарачи и Корректора нет логических операций **и**, **или**. Этот недостаток должен быть исправлен, а для работы со сложными логическими выражениями в текущей версии предлагаются алгоритмические формулы, вывод которых описан далее.

Операция *И*

Построим алгоритмическую формулу для проверки условия `A` **и** `B`. Если бы в языке существовала операция **и**, то можно было бы написать:

```
ЕСЛИ A И B ТО Работа
```

Эту конструкцию можно смоделировать так:

```
ЕСЛИ A
  ТО ЕСЛИ B
       ТО Работа
```

Аналогично, запись вида:

```
ЕСЛИ A И B И C ТО Работа
```

моделируется так:

```
ЕСЛИ A
  ТО ЕСЛИ B
       ТО ЕСЛИ C
            ТО Работа
```

Отформатируем модель более удобным способом, подобно тому, как это делалось при построении переключателя.

Конструкция

```
ЕСЛИ A И B ТО Работа
```

Алгоритмическая формула

```
ЕСЛИ A ТО ЕСЛИ B ТО Работа
```

Другой вариант записи:

```
ЕСЛИ    A
ТО ЕСЛИ B
ТО Работа
```

Конструкция

```
ЕСЛИ A И B И C ТО Работа
```

Алгоритмическая формула

ЕСЛИ A **ТО ЕСЛИ** B **ТО ЕСЛИ** C **ТО** Работа

Другой вариант записи:

ЕСЛИ A
ТО ЕСЛИ B
ТО ЕСЛИ C
ТО Работа

Конструкция

ЕСЛИ A_1 **И** A_2 **И** ... **И** A_{N-1} **И** A_N **ТО** Работа

Алгоритмическая формула

ЕСЛИ A_1
ТО ЕСЛИ A_2
...
ТО ЕСЛИ A_{N-1}
ТО ЕСЛИ A_N
ТО Работа

Пример

Координаты точек на плоскости записаны на ленту в виде:

$(x_1,y_1)(x_2,y_2)$...

Здесь: x_i,y_i — координаты i-ой точки — символьные числа, которым может предшествовать знак «–». Предполагается, что знаки «(», «)», «,» и «–» — особые, и символьные числа не могут принимать таких значений.

Требуется среди всех чисел, задающих координаты, найти наибольшее число без учёта его знака.

Символы в записи располагаются плотно, без пробелов и пустых ячеек. Окошко установлено на первую пустую ячейку слева от записи.

Результат нужно записать в виде символьного числа справа через одну пустую ячейку от исходных данных. Пример начального и конечного состояния среды показаны на рис. 11.6.

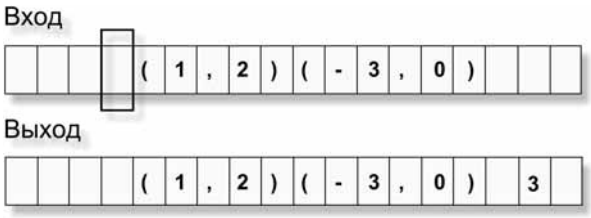

Рис. 11.6

Замечание

Символ алфавита Корректора будем называть символьным числом тогда, когда он используются для вычислений. Значением символьного числа будем считать номер соответствующего символа в алфавите Корректора по отношению к символу «0» (нуль). Номер символа «0» полагается равным числу 0 *(см. главу 5)*.

Решение

Описание алгоритма

Для хранения текущего кандидата будем использовать ящик. Занесём в него число 0 и пройдёмся вдоль записи. Всякий раз, когда на ленте обнаруживается число, большее числа в ящике, обновляем ящик этим значением.

Алгоритм

1. Записываем в ящик число 0.
2. Находим в записи наибольшее символьное число. Пока запись не закончилась, делаем:
   2.1. Если текущий символ — координата и она больше числа в ящике, заменяем содержимое ящика этой координатой.
   2.2. Продвигаем окно к следующему символу.
3. Записываем содержимое ящика на ленту.

Программа

```
ЭТО Вход
  Подготовка
  Поиск_максимума
  Запись_результата
КОНЕЦ

// Запишем в ящик число 0 - начальный
// кандидат для максимума
// ---------------------------------
ЭТО Подготовка
  ПИШИ 0 ОБМЕН
КОНЕЦ

ЭТО Поиск_максимума
  ВПРАВО
  ПОКА НЕ ПУСТО
  {
```

```
    Проверить
    ВПРАВО
  }
КОНЕЦ

// Если символ не является вспомогательным и
// он больше числа в ящике, заносим его в ящик.
// ------------------------------------------
ЭТО Проверить
  ЕСЛИ    НЕ (
  ТО ЕСЛИ НЕ )
  ТО ЕСЛИ НЕ ,
  ТО ЕСЛИ НЕ -
  ТО ЕСЛИ Я<Л
  ТО ЯЩИК+
КОНЕЦ

// Максимум содержится в ящике.
// Запишем его на ленту
// ---------------------------
ЭТО Запись_результата
  ВПРАВО ОБМЕН
КОНЕЦ
```

Операция *И:* модель с ветвью *ИНАЧЕ*

Рассмотрим вариант с ветвью **ИНАЧЕ**:

```
ЕСЛИ А И В ТО Работа1 ИНАЧЕ Работа2
```

Эта конструкция моделируется так:

```
ЕСЛИ А
  ТО    ЕСЛИ В
          ТО    Работа1
          ИНАЧЕ Работа2
  ИНАЧЕ Работа2
```

Последнюю запись можно переписать в виде компактной алгоритмической формулы:

Алгоритмическая формула

ЕСЛИ A
ТО ЕСЛИ B
ТО Работа1
ИНАЧЕ Работа2
ИНАЧЕ Работа2

В общем случае, для конструкции:

ЕСЛИ A_1 **И** A_2 **И** ... **И** A_{N-1} **И** A_N **ТО** Работа1 **ИНАЧЕ** Работа2

Алгоритмическая формула

ЕСЛИ A_1
ТО ЕСЛИ A_2
...
ТО ЕСЛИ A_{N-1}
ТО ЕСЛИ A_N
ТО Работа1
ИНАЧЕ Работа2
...
ИНАЧЕ Работа2
ИНАЧЕ Работа2
ИНАЧЕ Работа2

Число ветвей **ИНАЧЕ** совпадает с числом ветвей **ТО**.

Пример

Если символ на ленте является цифрой и эта цифра больше нуля, уменьшить её на 1, в противном случае записать на месте символа пробел.

Решение

```
ЕСЛИ ЦИФРА
ТО ЕСЛИ НЕ 0
ТО МИНУС
ИНАЧЕ ПИШИ ПРОБЕЛ
ИНАЧЕ ПИШИ ПРОБЕЛ
```

Замечание

А если в приведённом ранее решении убрать последнюю ветвь **ИНАЧЕ**? Приведёт ли такое сокращение к прежнему результату?

```
ЕСЛИ ЦИФРА
ТО ЕСЛИ НЕ 0
ТО МИНУС
ИНАЧЕ ПИШИ ПРОБЕЛ
```

Ответ отрицательный. Оставшаяся ветвь **ИНАЧЕ** относится к вложенной команде ветвления, значит, при нарушении условия во внешней команде ветвления пробел на ленту записан не будет. Таким образом, пробел будет записываться только тогда, когда на ленте записан нуль и ни при каком другом символе, что противоречит условию задачи.

Операция *ИЛИ*

```
ЕСЛИ A ИЛИ B ТО Работа
```

Эту операцию можно моделировать при помощи переключателя.

Алгоритмическая формула

```
ЕСЛИ        A ТО Работа
ИНАЧЕ ЕСЛИ B ТО Работа
```

В общем случае:

ЕСЛИ A_1 **ИЛИ** A_2 **ИЛИ ... ИЛИ** A_{N-1} **ИЛИ** A_N **ТО** Работа

Алгоритмическая формула

```
ЕСЛИ        A1    ТО Работа
ИНАЧЕ ЕСЛИ A2    ТО Работа
...
ИНАЧЕ ЕСЛИ AN-1 ТО Работа
ИНАЧЕ ЕСЛИ AN    ТО Работа
```

Пример

Если символ на ленте — одна из цифр 2, 5 или 8, заменить её цифрой, на единицу большей исходной.

Решение

```
ЕСЛИ        2 ТО ПЛЮС
ИНАЧЕ ЕСЛИ 5 ТО ПЛЮС
ИНАЧЕ ЕСЛИ 8 ТО ПЛЮС
```

Конструкция:

```
ЕСЛИ A ИЛИ B ТО Работа1 ИНАЧЕ Работа2
```

записывается в следующем виде:

```
ЕСЛИ       A ТО Работа1
ИНАЧЕ ЕСЛИ B ТО Работа1
ИНАЧЕ          Работа2
```

В общем виде конструкцию:

```
ЕСЛИ A1 ИЛИ A2 ИЛИ ... ИЛИ AN-1 ИЛИ AN ТО Работа1 ИНАЧЕ Работа2
```

можно переписать так:

```
ЕСЛИ       A1   ТО Работа1
ИНАЧЕ ЕСЛИ A2   ТО Работа1
...
ИНАЧЕ ЕСЛИ AN-1 ТО  Работа1
ИНАЧЕ ЕСЛИ AN   ТО Работа1
ИНАЧЕ              Работа2
```

Пример

Если символ на ленте — одна из цифр 2, 5 или 8, заменить её цифрой, на единицу большей исходной, любой другой символ заменить пробелом.

Решение

```
ЕСЛИ       2 ТО ПЛЮС
ИНАЧЕ ЕСЛИ 5 ТО ПЛЮС
ИНАЧЕ ЕСЛИ 8 ТО ПЛЮС
ИНАЧЕ           ПИШИ ПРОБЕЛ
```

Ответы на вопросы и задания

В тестах, рекомендованных для проверки программ, пробел обозначен знаком ■, пустая ячейка — знаком □, а подчёркнутый символ показывает положение окна на ленте.

11.1. Как найти конец текста

1. На ленте записан некоторый текст. Требуется установить окно на первый его символ. В начальный момент окно расположено произвольно внутри текста (рис. 11.7).

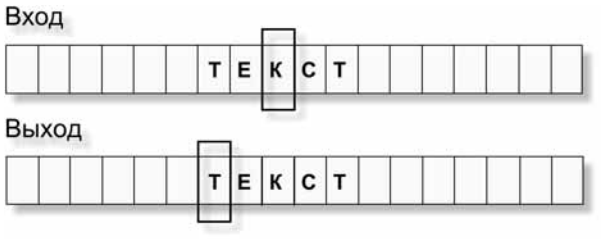

Рис. 11.7

Решение

Алгоритм

1. Пока в окне расположен символ, отличный от **ПУСТО**, перемещать окно влево.
2. Сместить окно вправо (на первый символ текста).

Программа

```
ЭТО На_начало
  ПОКА НЕ ПУСТО ВЛЕВО
  ВПРАВО
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 11.2.

Таблица 11.2

Тест	Комментарий	Ожидаемый результат
ЖИРАФ	Окно в середине длинного текста	ЖИРАФ
877565	Окно на первом символе длинного текста	877565
2+3=5	Окно на последнем символе длинного текста	2+3=5
К	Текст состоит из одного символа	К
□	Текст пуст	□

2. Найти первое повторное вхождение первого символа в тексте на ленте. В начальный момент окно установлено на начало текста. Установить окно на найденном символе или на первой пустой ячейке за концом текста, если повторов первого символа в тексте нет (рис. 11.8).

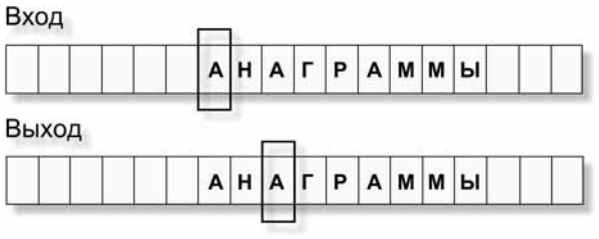

Рис. 11.8

Решение

Алгоритм

1. Скопировать первый символ текста в ящик.
2. Перемещать окно вправо, пока в нём не окажется символ, равный символу в ящике или символ **ПУСТО** (конец текста).

Программа

```
ЭТО Вход
  ЯЩИК+
  Повтор
КОНЕЦ


ЭТО Повтор
  ВПРАВО
  ЕСЛИ НЕ ПУСТО ТО ЕСЛИ Я#Л ТО Повтор
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 11.3.

Таблица 11.3

Тест	Комментарий	Ожидаемый результат
111111111 255727278 3774333	Первый символ повторяется несколько раз	111111111 255727278 3774333
КОЛОБОК КУКАРАЧА	Первый символ повторяется один раз	КОЛОБОК КУКАРАЧА
1234567	Первый символ не повторяется	1234567□

Таблица 11.3 *(окончание)*

Тест	Комментарий	Ожидаемый результат
ФФ	Текст состоит из двух одинаковых символов	ФФ
АХ	Текст состоит из двух разных символов	АХ
+	Текст состоит из одного символа	+□
□	Текст пуст	□

3. Найти последнее вхождение первого символа в тексте на ленте. В начальный момент окно установлено на начало текста. Установить окно на найденном символе или на первой пустой ячейке за концом текста, если повторов первого символа в тексте нет (рис. 11.9).

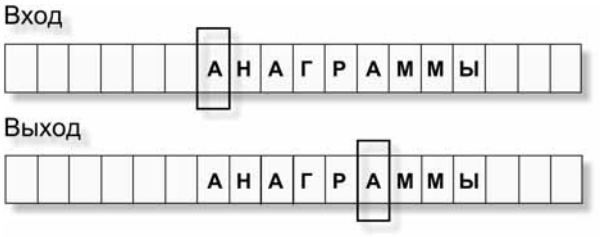

Рис. 11.9

Решение

Алгоритм

1. Скопировать первый символ текста в ящик.
2. Установить окно на последний символ текста.
3. Перемещать окно влево, пока в нём не окажется символ, равный символу в ящике.
4. Если этот символ является первым символом текста, установить окно на первый пустой символ за концом текста.

Программа

```
ЭТО Вход
  // Скопировать первый символ текста в ящик
  ЯЩИК+
  // Установить окно на последний символ текста
  На_конец
```

```
  // Перемещать окно влево, пока в нём не окажется
  // символ, равный символу в ящике
  Повтор
  // Если этот символ является первым символом текста,
  // установить окно на первый пустой символ за концом текста
  ВЛЕВО
  ЕСЛИ ПУСТО
    ТО { ВПРАВО За_конец }
    ИНАЧЕ ВПРАВО
КОНЕЦ

ЭТО Повтор
  ПОКА Я#Л ВЛЕВО
КОНЕЦ

ЭТО На_конец
  За_конец
  ВЛЕВО
КОНЕЦ

ЭТО За_конец
  ПОКА НЕ ПУСТО ВПРАВО
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 11.4.

Таблица 11.4

Тест	Комментарий	Ожидаемый результат
111111111 255727278 3774333	Первый символ повторяется несколько раз	111111111 255727278 3774333
КОЛОБОК КУКАРАЧА	Первый символ повторяется один раз	КОЛОБОК КУКАРАЧА
1234567	Первый символ не повторяется	1234567□
ФФ	Текст состоит из двух одинаковых символов	ФФ

Таблица 11.4 *(окончание)*

Тест	Комментарий	Ожидаемый результат
АХ	Текст состоит из двух разных символов	АХ□
+	Текст состоит из одного символа	+□
□	Текст пуст	□

11.2. Как вернуться в исходное место

На рис. 11.10 и 11.11 к задачам показаны примеры исходного и результирующего состояний среды.

1. Окно установлено на один из символов текста. Заключить текст в круглые скобки и вернуть окно в исходное положение.

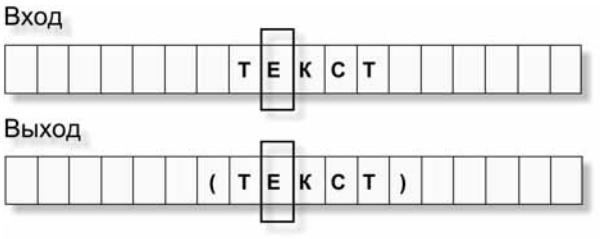

Рис. 11.10

Решение

Описание алгоритма. Задача решается при помощи последовательного вызова двух рекурсивных процедур, каждая из которых записывает скобку (первая слева, вторая справа) и рекурсивной пружинкой возвращает окно в исходное место записи.

Программа

```
ЭТО Вход
  Левая_скобка
  Правая_скобка
КОНЕЦ

ЭТО Левая_скобка
  ВЛЕВО
  ЕСЛИ НЕ ПУСТО
```

```
    ТО Левая_скобка
    ИНАЧЕ ПИШИ (
  // Рекурсивная пружинка: возврат в исходное место
  ВПРАВО
КОНЕЦ

ЭТО Правая_скобка
  ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО Правая_скобка
    ИНАЧЕ ПИШИ )
  // Рекурсивная пружинка: возврат в исходное место
  ВЛЕВО
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 11.5.

Таблица 11.5

Тест	Комментарий	Ожидаемый результат
□	Пустая запись	(□)
1	Запись из одного символа	(1)
123 123 123	Запись из нескольких символов	(123) (123) (123)

2. Проверить, совпадают ли посимвольно две записи одинаковой длины. В начальный момент окно установлено на первый символ первой записи, а вторая отделяется от первой одной пустой ячейкой.

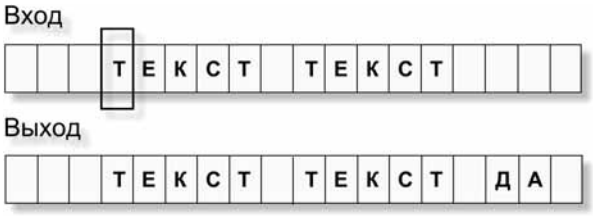

Рис. 11.11

Решение

Описание алгоритма. В основной процедуре `Вход` в цикле **ПОКА** сравниваем символы первой записи с символами второй записи, расположенными на тех же местах (процедура `Сравнение_символов`).

Если символы не совпадают, записываем в ящик 0 — признак несовпадения и завершаем цикл просмотра, смещая окно на пустое место между двумя записями.

Если символы совпадают, продолжаем проверку записей.

Программа

```
ЭТО Вход
  // Сравниваем посимвольно две записи
  // Цикл по символам первой записи:
  ПОКА НЕ ПУСТО { Сравнение_символов ВПРАВО }
  // За конец второй записи
  ВПРАВО
  ПОКА НЕ ПУСТО ВПРАВО
  ВПРАВО
  // Запись результата (если в ящике 0, то записи не совпадают)
  ЯЩИК-
  ЕСЛИ 0
   ТО Ответ_нет
   ИНАЧЕ Ответ_да
КОНЕЦ

ЭТО Сравнение_символов
  ОБМЕН
  На_символ_второй_записи
  ЕСЛИ Я#Л
    ТО Записи_не_совпадают
    ИНАЧЕ Продолжение_проверки
КОНЕЦ

ЭТО Записи_не_совпадают
  На_текущее_место_в_первой_записи
  // Восстановим символ в первой записи и
  // запишем в ящик 0 - признак несовпадения записей.
  ПИШИ 0 ОБМЕН
```

```
  // Установим окно на конец первой записи,
  // чтобы прервать цикл просмотра.
  ПОКА НЕ ПУСТО ВПРАВО
  ВЛЕВО
КОНЕЦ

ЭТО На_текущее_место_в_первой_записи
  ПОКА НЕ ПУСТО ВЛЕВО
  ВЛЕВО
  ПОКА НЕ ПУСТО ВЛЕВО
КОНЕЦ

ЭТО Продолжение_проверки
  На_текущее_место_в_первой_записи
  // Восстановим символ в первой записи
  ОБМЕН
КОНЕЦ

ЭТО На_символ_второй_записи
  ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО На_символ_второй_записи
    ИНАЧЕ За_конец_второй_записи
  // Рекурсивная пружинка: смещение на символ второй записи,
  // который расположен на том же месте, что и проверяемый
  // символ в первой записи.
  ВЛЕВО
КОНЕЦ

ЭТО За_конец_второй_записи
  ВПРАВО
  ПОКА НЕ ПУСТО ВПРАВО
КОНЕЦ

ЭТО Ответ_да
  ПИШИ Д ВПРАВО ПИШИ А
КОНЕЦ
```

```
ЭТО Ответ_нет
  ПИШИ Н ВПРАВО ПИШИ Е ВПРАВО ПИШИ Т
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 11.6.

Таблица 11.6

Тест	Комментарий	Ожидаемый результат
□	Пустые записи	ДА
1□1 1□2	Записи из одного символа	1□1□ДА 1□2□НЕТ
123□123 123□124 163□123 223□123	Записи из нескольких символов	123□123□ДА 123□124□НЕТ 163□123□НЕТ 223□123□НЕТ

11.3. Задачи

На рис. 11.12–11.16 к задачам показаны примеры исходного и результирующего состояний среды.

1. Продублировать текст, записав его на ленту в обратном порядке. В начальный момент окно установлено на первый символ текста.

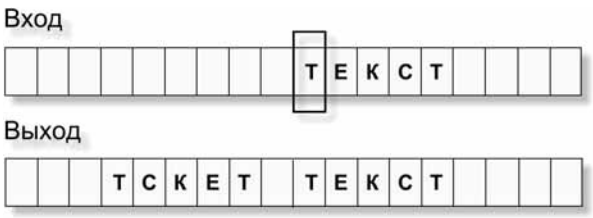

Рис. 11.12

Решение

Будем записывать результат слева от исходного текста.

Алгоритм

1. Пока не пусто, делать:
   1.1. Скопировать символ из окна в ящик.
   1.2. Добавить символ из ящика в начало результирующей записи (слева от исходного текста) и вернуть окно на текущее место в исходном тексте.
   1.3. Переместить окно к следующему символу исходного текста.

Программа

```
ЭТО Вход
  ПОКА НЕ ПУСТО
  {
   // Запомнить текущий символ в исходном тексте
    ЯЩИК+
    // Поставить символ перед результирующим текстом и вернуться
    // в текущее место в исходном тексте
    Работа
    // Переместить окно к следующему символу исходного текста
    ВПРАВО
  }
КОНЕЦ

ЭТО Работа
  ВЛЕВО
  ЕСЛИ НЕ ПУСТО
    ТО Работа
    ИНАЧЕ Установка_символа
  ВПРАВО // Рекурсивная пружинка для возврата в текущее
         // положение в исходном тексте
КОНЕЦ

ЭТО Установка_символа
  ВЛЕВО
  ПОКА НЕ ПУСТО ВЛЕВО
  ЯЩИК-
  ПОКА НЕ ПУСТО ВПРАВО
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 11.7.

Таблица 11.7

Тест	Комментарий	Ожидаемый результат
12345	В тексте несколько символов	54321□12345
Ф	В тексте один символ	Ф□Ф
□	Текст пуст	□

2. На ленте записано целое десятичное число. Проверить, нет ли ошибок в его записи. Если ошибки есть, написать за текстом слово ОШИБКА и установить окно на первую слева ошибку. В противном случае записать за числом сообщение ВЕРНО. В начальный момент окно установлено на первый символ записи.

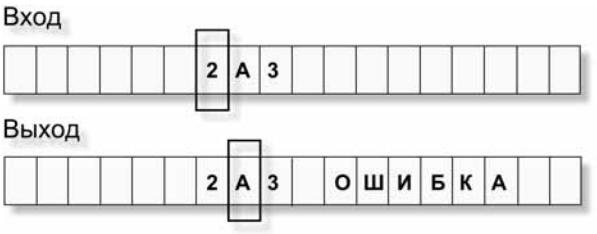

Рис. 11.13

Решение

Описание алгоритма. Отдельно проверим знак числа (процедура `Вход`), первую цифру (процедура `Целое_без_знака`) и остальные цифры (процедура `Цифры`).

Программа

```
// Проверка знака
ЭТО Вход
  ЕСЛИ       +     ТО { ВПРАВО Целое_без_знака }
  ИНАЧЕ ЕСЛИ -     ТО { ВПРАВО Целое_без_знака }
  ИНАЧЕ ЕСЛИ ЦИФРА ТО Целое_без_знака
  ИНАЧЕ              Ошибка
КОНЕЦ
// Проверка первой цифры числа
```

```
ЭТО Целое_без_знака
  ЕСЛИ ЦИФРА ТО Цифры
  ИНАЧЕ         Ошибка
КОНЕЦ

// Проверка остальных цифр числа
ЭТО Цифры
  ВПРАВО
  ПОКА ЦИФРА ВПРАВО
  ЕСЛИ ПУСТО
    ТО     Верно
    ИНАЧЕ  Ошибка
КОНЕЦ

ЭТО Ошибка
  ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО Ошибка
    ИНАЧЕ
    {
       ВПРАВО ПИШИ О
       ВПРАВО ПИШИ Ш
       ВПРАВО ПИШИ И
       ВПРАВО ПИШИ Б
       ВПРАВО ПИШИ К
       ВПРАВО ПИШИ А
       ПОВТОРИ 6 ВЛЕВО
     }
  ВЛЕВО // Возврат к месту ошибки
КОНЕЦ

ЭТО Верно
  ВПРАВО ПИШИ В
  ВПРАВО ПИШИ Е
  ВПРАВО ПИШИ Р
  ВПРАВО ПИШИ Н
  ВПРАВО ПИШИ О
КОНЕЦ
```

Тесты

Тесты для правильных записей приводятся в табл. 11.8.

Таблица 11.8

Тест	Комментарий	Ожидаемый результат
325 7 15748463269	Число без знака	325□ВЕРНО 7□ВЕРНО 15748463269□ВЕРНО
+325 +7 +15748463269	Число со знаком +	+325□ВЕРНО +7□ВЕРНО +15748463269□ВЕРНО
–325 –7 –15748463269	Число со знаком –	–325□ВЕРНО –7□ВЕРНО –15748463269□ВЕРНО

Тесты для записей с ошибками приводятся в табл. 11.9.

Таблица 11.9

Тест	Комментарий	Ожидаемый результат
32А5 +Ф156 –7676* Щ	Число содержит не цифру	32А5□ОШИБКА +Ф156□ОШИБКА –7676*□ОШИБКА Щ□ОШИБКА
+ –	Запись содержит только знак числа. Ошибочным должен быть отмечен первый пустой символ за знаком (на этом месте ожидается цифра, а её нет)	+□ОШИБКА –□ОШИБКА
□	Запись пуста	□ОШИБКА
3+25 7–	Знак числа не на месте	3+25 7–
++36843 ––6236 +–7283 –+6889	Несколько знаков перед набором цифр	++36843□ОШИБКА ––6236□ОШИБКА +–7283□ОШИБКА –+6889□ОШИБКА

3. На ленте через три пустых ячейки друг от друга записаны два целых неотрицательных числа. Установить окно на первую цифру большего из них, если известно, что числа имеют одинаковое количество разрядов и незначащих (первых) нулей в них нет. В начальный момент окно установлено на первую цифру первого числа.

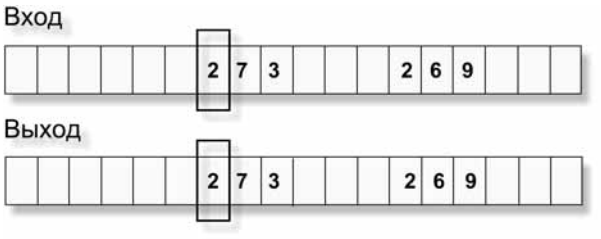

Рис. 11.14

Решение

Описание алгоритма. Будем по порядку сравнивать цифры в числах, начиная со старшего разряда. Если все цифры равны, то и числа равны, в противном случае наибольшее число определяется по первой несовпавшей паре цифр.

По условию задачи исходное состояние среды имеет вид:

`число1□□□число2`

Две ячейки (обозначим их через N и D) перед вторым числом будем использовать для хранения:

- в ячейке N — значения счётчика шагов исполнителя по первой записи;
- в ячейке D — копии текущей цифры первого числа.

Таким образом, во время работы программы лента будет иметь вид:

`число1□NDчисло2`

Будем перемещать окно по первому числу, сравнивая каждую его цифру с соответствующей цифрой второго числа (процедура `Сравнение`). Результат сравнения будем записывать в ящик (процедура `Сравнить`) в виде:

- 0 — признак «числа равны»;
- 1 — признак «первое число больше»;
- 2 — признак «второе число больше».

Для нахождения цифры второго числа, отстоящей на такое же число ячеек от начала второго числа, что и текущая цифра первого числа от начала первого числа, будем использовать счётчик шагов в ячейке N. Сначала этот счётчик обнулим (процедура `Обнулить_счетчик_шагов`), а затем

будем добавлять к нему по единице на каждом шаге по цифрам первого числа (в процедуре Сравнить). Значение счётчика будем использовать для установки окна на цифру второго числа, которая соответствует текущей цифре первого числа.

Программа

```
ЭТО Вход
  Обнулить_счетчик_шагов // Записать 0 в ячейку N
  Сравнение
КОНЕЦ

// Записать 0 в ячейку N
ЭТО Обнулить_счетчик_шагов
  ПОКА НЕ ПУСТО ВПРАВО
  ВПРАВО ПИШИ 0
  ВЛЕВО ВЛЕВО
  ПОКА НЕ ПУСТО ВЛЕВО
  ВПРАВО
КОНЕЦ

// Рекурсивно продвигаем окно по первому числу, сравнивая его
// цифры с соответствующими цифрами второго числа (в процедуре
// Сравнить_цифры). По итогам сравнения выполняем следующие
// действия:
// * Цифра 1-го числа больше - прерываем рекурсию (Первое_больше)
// * Цифра 2-го числа больше - прерываем рекурсию (Второе_больше)
// * Цифры равны - если это были последние цифры, прерываем
//                 рекурсию (Числа_равны), если нет, продолжаем
//                 работу (рекурсивный вызов Сравнение).
// -------------------------------------------------------------
ЭТО Сравнение
  ЯЩИК+          // Запомнить текущую цифру 1-ого числа
  Сравнить_цифры // Сравнить с цифрой 2-го числа
  // Проверить результат сравнения
  ОБМЕН
  ЕСЛИ       1 ТО { ОБМЕН Первое_больше }
  ИНАЧЕ ЕСЛИ 2 ТО { ОБМЕН Второе_больше }
  ИНАЧЕ
```

```
  {
    ОБМЕН
    ВПРАВО          // Продвинуть окно на следующую цифру 1-го числа
    ЕСЛИ ПУСТО ТО Числа_равны
    ИНАЧЕ         Сравнение
  }
КОНЕЦ

ЭТО Сравнить_цифры
  ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО    Сравнить_цифры
    ИНАЧЕ Сравнить
  ВЛЕВО // Возврат на текущее место в первом числе
КОНЕЦ

// Вход:  окно установлено на пустую ячейку за 1-ым числом
// Выход: окно установлено на пустую ячейку за 1-ым числом,
//        ящик содержит результат проверки:
//        0 - цифры равны
//        1 - цифра первого числа больше
//        2 - цифра второго числа больше
// -------------------------------------------------------
ЭТО Сравнить
  // На счетчик
  ВПРАВО
  // Записать символ из первой записи в ячейку D
  ВПРАВО ЯЩИК-
  // Записать в ящик значение счетчика (содержимое ячейки N)
  ВЛЕВО ЯЩИК+
  // Установить окно на символ второй записи
  // по счетчику N и запомнить его в ящике
  ВПРАВО ВПРАВО
  ОБМЕН
  ПОКА НЕ 0 { МИНУС ОБМЕН ВПРАВО ОБМЕН }
  ЯЩИК-
  // Установить окно на счетчик
```

```
  ПОКА НЕ ПУСТО ВЛЕВО
  ВПРАВО
  // Увеличить счетчик на 1
  ПЛЮС
  // Установить окно на ячейку D
  ВПРАВО
  // Сравнить цифры
  ЕСЛИ       Я=Л ТО ПИШИ 0 // Признак "числа равны"
  ИНАЧЕ ЕСЛИ Я<Л ТО ПИШИ 1 // Признак "первое число больше"
  ИНАЧЕ          ПИШИ 2 // Признак "второе число больше"
  ОБМЕН
  // Под пружинку
  ВЛЕВО ВЛЕВО
КОНЕЦ

ЭТО Первое_больше
  Удалить_ND
  // На первую цифру первого числа
  ВЛЕВО ВЛЕВО ВЛЕВО
  ПОКА НЕ ПУСТО ВЛЕВО
  ВПРАВО
КОНЕЦ

ЭТО Удалить_ND
  ПОКА НЕ ПУСТО ВПРАВО
  ПОКА ПУСТО ВПРАВО
  ПИШИ ПУСТО ВПРАВО
  ПИШИ ПУСТО
КОНЕЦ

ЭТО Второе_больше
  Удалить_ND
  // На первую цифру второго числа
  ВПРАВО
КОНЕЦ

ЭТО Числа_равны
  Второе_больше
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 11.10.

Таблица 11.10

Тест	Комментарий	Ожидаемый результат
7647□□□7647 7□□□7	Числа равны	7647□□□7647 7□□□7
7649□□□7641 4569□□□4548 7□□□0	Первое число больше	7649□□□7641 4569□□□4548 7□□□0
7641□□□7649 4569□□□4578 1□□□8	Второе число больше	7641□□□7649 4569□□□4578 1□□□8

4. В начальный момент в окно виден первый символ текста. Убрать из текста все повторные вхождения этого символа, а оставшуюся запись уплотнить.

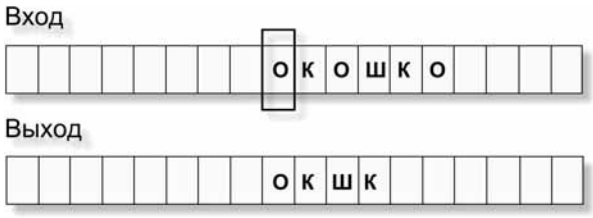

Рис. 11.15

Решение

Идея алгоритма. Скопируем первый символ в ящик и будем просматривать текст со второго символа. Каждый раз, когда обнаруживается символ, совпадающий с символом в ящике, будем сдвигать остаток записи влево, затирая ненужный символ и одновременно уплотняя запись.

Алгоритм

1. Скопировать первый символ текста в ящик.
2. Сместить окно ко второму символу.
3. Пока в окне не `пусто`, делать:
   3.1. Если символ в ящике совпадает с символом на ленте, удалить символ на ленте и уплотнить запись.

3.2. Если символ в ящике отличается от символа на ленте, передвинуть окно к следующему символу.

Программа

```
ЭТО Вход
  ЯЩИК+
  ВПРАВО
  ПОКА НЕ ПУСТО Работа
КОНЕЦ

ЭТО Работа
  ЕСЛИ Я=Л
   ТО Уплотнить
   ИНАЧЕ ВПРАВО
КОНЕЦ

ЭТО Уплотнить
  ВПРАВО
  Символ_влево
  ЕСЛИ НЕ ПУСТО ТО Уплотнить
  ВЛЕВО // Рекурсивная пружинка возвращает окно
        // к текущему символу просмотра
КОНЕЦ

ЭТО Символ_влево
  ОБМЕН ВЛЕВО ЯЩИК- ВПРАВО ОБМЕН
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 11.11.

Таблица 11.11

Тест	Комментарий	Ожидаемый результат
КИТ К	Первый символ не повторяется в тексте	КИТ К

Таблица 11.11 *(окончание)*

Тест	Комментарий	Ожидаемый результат
1124	Первый символ повторяется в тексте один раз	124
1213		123
1231		123
11123	Первый символ повторяется в тексте много раз	123
121113		123
1211311		123
1111111		1
□	Пустой текст	□

5. На ленте записан архив. В архиве каждому символу предшествует цифра 0 (символ не входит в исходный текст) или цифра 2 (символ повторяется в исходном тексте дважды). Восстановить по архиву исходную запись. В начальный момент окно установлено на начало архива.

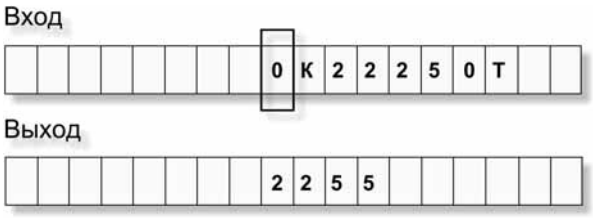

Рис. 11.16

Решение

Описание алгоритма. Просматриваем архив по две ячейки (первая ячейка — кратность символа в исходном тексте, вторая — сам символ). Если кратность символа равна нулю, просто перемещаем окно к новой паре ячеек, если кратность равна 2, записываем символ два раза в конец исходного текста (который строим справа от архива, через пустую ячейку от него).

Программа

```
// Пока не ПУСТО, работаем над архивом, анализируя по паре символов
ЭТО Вход
  ПОКА НЕ ПУСТО Работа
КОНЕЦ
```

```
// Анализируем первый символ пары
ЭТО Работа
  ЕСЛИ 0
    ТО    Ноль
    ИНАЧЕ Два
КОНЕЦ

// Если первый символ пары 0, просто переходим к следующей паре
ЭТО Ноль
  ВПРАВО ВПРАВО
КОНЕЦ

// Если первый символ пары 2, добавляем две копии
  // символа в конец формируемого результата
ЭТО Два
  // Запомнить символ из архива
  Запомнить_символ
  // Добавить две копии запомненного символа в конец результата
  Добавить
  // Сместить окно на следующую пару символов в архиве
  ВПРАВО
КОНЕЦ

// Запомнить символ из архива
ЭТО Запомнить_символ
  ВПРАВО ЯЩИК+
КОНЕЦ

// Добавить два символа в конец формируемого результата.
// Сначала рекурсивно находим конец архива, добавляем два
// символа к результату и при помощи рекурсивной пружинки
// возвращаем окно на текущее место в архиве.
// -------------------------------------------------
ЭТО Добавить
  ВПРАВО
  ЕСЛИ НЕ ПУСТО ТО Добавить
  ИНАЧЕ Добавить_к_результату
```

```
  ВЛЕВО // Рекурсивная пружинка: возврат к текущему месту в архиве
КОНЕЦ


ЭТО Добавить_к_результату
  // На конец результата
  ВПРАВО
  ПОКА НЕ ПУСТО ВПРАВО
  // Добавить к результату
  ЯЩИК- ВПРАВО ЯЩИК-
  // Установить окно перед началом результата
  ПОКА НЕ ПУСТО ВЛЕВО
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 11.12.

Таблица 11.12

Тест	Комментарий	Ожидаемый результат
01 21	В архиве одна пара ячеек	01□ 21□11
0122 2102 0102 2122 0022002300	В архиве несколько пар ячеек	0122□22 2102□11 0102□ 2122□1122 0022002300□2233

Глава 12

Арифметика чисел, палочек и символов

Структурное программирование и библиотечные процедуры

Структурный стиль программирования предполагает нисходящую разработку программного продукта.

Сначала проектируется главная (входная) процедура, — она описывает основные шаги алгоритма и, как правило, содержит только вызовы процедур, реализующие эти шаги. Затем описываются дочерние процедуры, и с ними поступают подобным образом: все крупные и логически связанные построения заменяют вызовами подчинённых процедур.

Программист продвигается в своём строительстве кода, постепенно детализируя алгоритм на каждом уровне процедурной иерархии, пока не будут построены все терминальные процедуры, расположенные на листьях процедурного дерева.

Преимущество нисходящей разработки очевидны: на каждом шаге проектировщик описывает крупные блоки, оставляя детальное развитие сюжета вспомогательным процедурам.

Таким образом, удаётся создавать большие коды, без страха увязнуть в деталях, как обычно получается при попытке рассказать всю программу одним алгоритмическим предложением.

С другой стороны, работая над новым проектом, конструктор полон желания повысить эффективность своего труда за счёт использования своих прежних наработок и наработок коллег профессионального цеха.

Возникает здоровая идея не закапывать однажды созданные процедуры, реализующие часто востребованные действия в недрах своего кода, а наоборот, выделять их в специальную копилку для последующего использования.

Так и возникают сборники готовых процедур, которые программисты называют *библиотеками*.

Имея под рукой набор библиотечных процедур, программист укорачивает дерево проекта, размещая на его листьях готовые и уже однажды отлаженные блоки.

В текущей реализации исполнителя Корректор (которая сопровождает эту книгу на CD) есть возможность добавления к записанной в среде программе дополнительного кода из файла. Настоятельно рекомендуется использовать эту интерфейсную операцию для подсоединения заранее созданных библиотек. Одна такая библиотека, `lib.kor` записана на CD к этой книге.

При подготовке процедур для библиотеки необходимо особое внимание уделить заголовочным комментариям, которые описывают состояние среды на входе, выходе, особенности работы и ограничения использования.

Без таких описаний использование библиотечных процедур становится проблематичным.

В решениях, описанных далее в этой главе, отсутствует описание процедур `Минус1` и `Символ_в_число` — они считаются библиотечными. Коды этих процедур приводятся в *главе 12* ученической книги, а в файле `lib.kor` они снабжены следующими заголовочными комментариями:

```
// Вычитание единицы.
// Начальное положение окна: на младшей цифре числа
// Конечное положение окна: на одной из цифр результата
// Замечания
// 1. Положение первой цифры результата совпадает с положением первой
//    цифры исходного числа
// 2. Результат завершается символом ПУСТО, если количество
//    цифр в результате меньше количества цифр в исходном числе
ЭТО Минус1
…
КОНЕЦ

// Перевод числа в символьное число.
// Вход:  окно установлено на младшую цифру,
//        если старшей цифры нет, то слева пустая ячейка.
// Выход: окно установлено на результат (место младшей цифры),
//        а в ячейку слева записано ПУСТО.
// Ограничение: процедура работает для целых неотрицательных
//              чисел, значение которых меньше 100.
// -------------------------------------------------------
ЭТО Число_в_символ
…
КОНЕЦ
```

Решение задач

В тестах, рекомендованных для проверки программ, пробел обозначен знаком ■, пустая ячейка — знаком □, а подчёркнутый символ показывает положение окна на ленте.

12.1. Арифметика чисел

На рис. 12.1–12.6 приводятся примеры начальных состояний и результаты обработки.

1. На ленте записан пример на вычитание двух чисел (уменьшаемое не меньше вычитаемого). Вычислить результат. В начальный момент окно установлено на знак «–».

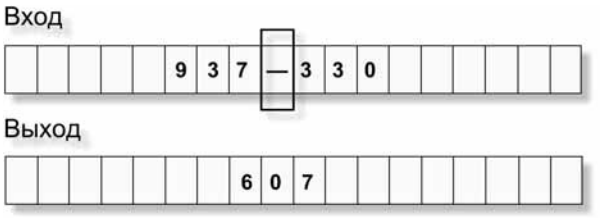

Рис. 12.1

Решение

Описание алгоритма. Вычитать из чисел по 1, пока вычитаемое не станет нулём.

Алгоритм процедуры `Вычитание`

1. Проверить число справа.
   1.1. Если 0, то закончить работу.
   1.2. Если не 0, то делать:
      1.2.1. Установить окно на младшую цифру вычитаемого.
      1.2.2. Отнять от вычитаемого 1 (вызов процедуры `Минус1`).
      1.2.3. Установить окно на младшую цифру уменьшаемого.
      1.2.4. Отнять от уменьшаемого 1 (вызов процедуры `Минус1`).
      1.2.5. Подготовить среду к следующему обороту рекурсивного цикла: установить окно на знак «–».
      1.2.6. Продолжить вычитание (рекурсивный вызов процедуры `Вычитание`)

Программа

```
ЭТО Вычитание
  // Проверка на окончание:
  ВПРАВО
  ЕСЛИ 0
    ТО    // Если справа 0, вычитание закончено
    { ПИШИ ПУСТО ВЛЕВО ПИШИ ПУСТО }
    ИНАЧЕ // Если справа не 0, продолжить вычитание
    {
       // Окно на младшую цифру вычитаемого
       ПОКА НЕ ПУСТО ВПРАВО
       ВЛЕВО
       // От вычитаемого отнять 1
       Минус1
       // Окно на младшую цифру уменьшаемого
       ПОКА НЕ - ВЛЕВО
       ВЛЕВО
       ПОКА НЕ ЦИФРА ВЛЕВО
       // От уменьшаемого отнять 1
       Минус1
       // Подготовить среду к следующему шагу рекурсивного цикла:
       // установить окно на знак -
       ПОКА НЕ - ВПРАВО
       // Рекурсивный шаг: продолжить вычитание
       Вычитание
    }
КОНЕЦ
```

Замечание

Процедура `Минус1` описана в *главе 12* книги ученика.

Тесты

Набор тестов приводится в табл. 12.1.

Таблица 12.1

Тест	Комментарий	Ожидаемый результат
23_0	Вычитаемое равно 0	23
65_65	Результат равен 0	0
100_75	Первое число после вычитания единицы становится короче	25
575_100	Второе число после вычитания единицы становится короче	475
100_100	Оба числа после вычитания единицы становятся короче	0
456_177	Случайные числа	279

2. На ленте записан пример вида: «число*2». Вычислить результат. В начальный момент окно установлено на знак умножения.

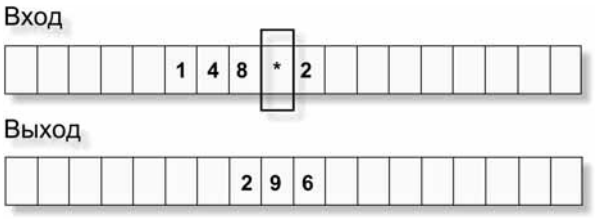

Рис. 12.2

Рекомендация

Привести пример к виду «число+число» и использовать процедуру сложения двух чисел.

3. На ленте записан пример вида: «число*4». Вычислить результат. В начальный момент окно установлено на знак умножения.

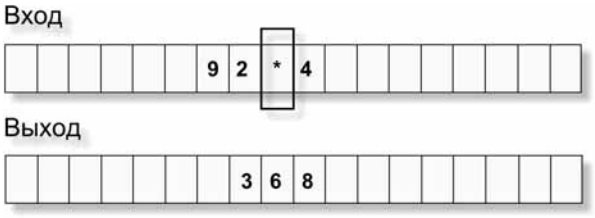

Рис. 12.3

Рекомендация

Два раза применить процедуру, умножающую число на 2.

4. На ленте записан пример вида: «число*16». Вычислить результат. В начальный момент окно установлено на знак умножения.

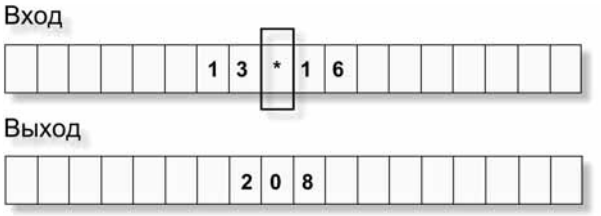

Рис. 12.4

Рекомендация

Два раза применить процедуру, умножающую число на 4.

5. На ленте записан пример вида: «число*3». Вычислить результат. В начальный момент окно установлено на знак умножения.

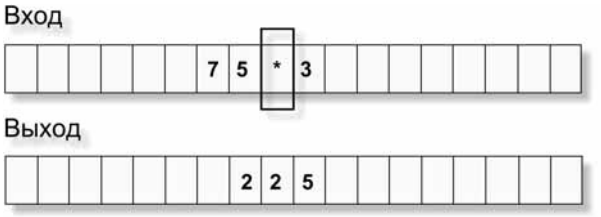

Рис. 12.5

Рекомендация

Использовать процедуру для сложения двух чисел.

6. На ленте записан пример вида: «число*6». Вычислить результат. В начальный момент окно установлено на знак умножения.

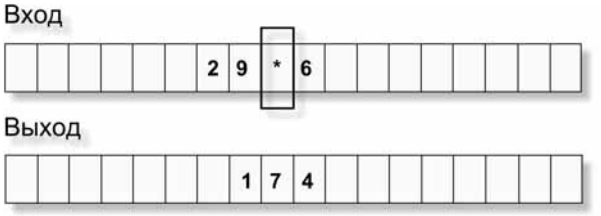

Рис. 12.6

Рекомендация

Использовать процедуру для умножения на 2 и процедуру для умножения на 3.

12.2. Арифметика палочек

На рис. 12.7–12.10 приводятся примеры начальных состояний и результаты обработки.

1. Умножить палочное число на 2. В начальный момент окно установлено на первую слева палочку.

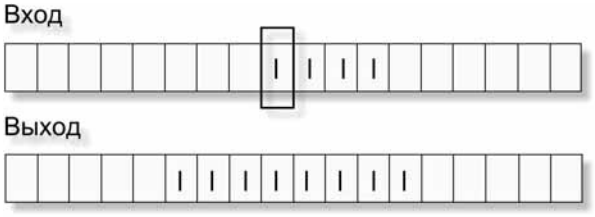

Рис. 12.7

Решение 1

Этот вариант работает только для исходных чисел, не превышающих по значению длину алфавита Корректора.

Описание алгоритма. Сначала подсчитаем количество исходных палочек (процедура `Длина`), а затем допишем на ленту столько же новых палочек (процедура `Добавить`).

Описание работы процедуры `Длина`

Для подсчёта будем использовать ящик. Занесём в него символ 0 и будем перемещать окно вдоль палочного числа, увеличивая значение ящика командой **плюс** (временно записывая содержимое ящика на ленту) на каждой палочке. В ящике будут последовательно храниться числа 1 (одна палочка), 2 (две палочки), ..., 9 (девять палочек), А (десять палочек), Б (одиннадцать палочек) и т. д.

Описание работы процедуры `Добавить`

Пока в ящике не окажется символ 0, будем записывать палочки на ленту и уменьшать значение ящика командой **минус** (временно записывая содержимое ящика на ленту).

Программа

```
ЭТО Умножить_2
  Длина
  Добавить
КОНЕЦ
```

```
// Подсчёт числа палочек
ЭТО Длина
  // Занесём в ящик (счётчик палочек)0
  ОБМЕН ПИШИ 0 ОБМЕН
  // Считаем палочки: добавляем 1 к счётчику на каждой палочке
  ПОКА НЕ ПУСТО
 {
    ОБМЕН ПЛЮС ОБМЕН ВПРАВО
 }
КОНЕЦ

// Пишем на ленту палочки, отнимая от счётчика по 1,
// пока в счётчике не окажется число 0
ЭТО Добавить
  ОБМЕН
  ПОКА НЕ 0
  {
    // Отнять от счётчика 1
    МИНУС ОБМЕН
    // Записать палочку
    ПИШИ |
    // Сдвинуть окно вправо и проверить счётчик
    ВПРАВО ОБМЕН
  }
  ОБМЕН
КОНЕЦ
```

Решение 2

Вариант с рекурсивной пружинкой. У него нет ограничения на длину палочного числа.

Описание алгоритма. Рекурсивно перемещаем окно вдоль палочного числа. Отложенные в рекурсии команды `{ ПИШИ | ВПРАВО }` запишут на ленту на одну палочку больше, чем надо: количество записанных палочек будет равно числу рекурсивных экземпляров программы плюс один (работа группы `{ ПИШИ | ВПРАВО }` в оригинале). Поэтому в главной (не рекурсивной) процедуре `Умножить_2` сместим окно на вторую палочку перед вызовом рекурсивной процедуры `Умножить_2_работа`. Заодно обработаем в главной процедуре случай палочного нуля (писать на ленту палочки в этом случае вообще не надо).

Программа

```
ЭТО Умножить_2
  // Отдельная обработка палочного нуля
  ЕСЛИ | ТО { ВПРАВО Умножить_2_работа }
КОНЕЦ

// Умножение ненулевых палочных чисел
ЭТО Умножить_2_работа
  ЕСЛИ |
    ТО { ВПРАВО Умножить_2_работа }
  ПИШИ | // Рекурсивная пружинка из двух команд:
  ВПРАВО // продублирует на ленту исходные палочки
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 12.2.

Таблица 12.2

Тест	Комментарий	Ожидаемый результат
□	Палочное число равно нулю	□
|	Палочное число равно единице	||
|||||	Большое палочное число	||||||||||

2. Умножить палочное число на 3. В начальный момент окно установлено на первую слева палочку.

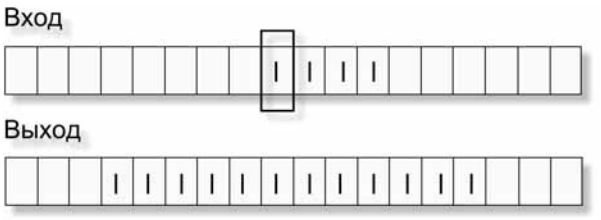

Рис. 12.8

Решение

Описание алгоритма. Приведённое далее решение практически повторяет рекурсивный вариант программы умножения на 2. Изменения касаются

только рекурсивной пружины: она содержит запись на ленту не одной, а двух палочек.

Программа

```
ЭТО Умножить_3
  // Отдельная обработка палочного нуля
  ЕСЛИ | ТО { ВПРАВО Умножить_3_работа }
КОНЕЦ


// Умножение ненулевых палочных чисел
ЭТО Умножить_3_работа
  ЕСЛИ |
    ТО { ВПРАВО Умножить_3_работа }
  // Рекурсивная пружинка
  ПИШИ | ВПРАВО
  ПИШИ | ВПРАВО
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 12.3.

Таблица 12.3

Тест	Комментарий	Ожидаемый результат
□	Палочное число равно нулю	□
|	Палочное число равно единице	| | |
|||||	Большое палочное число	|||||||||||||||

3. Умножить палочное число на 6. В начальный момент окно установлено на первую слева палочку.

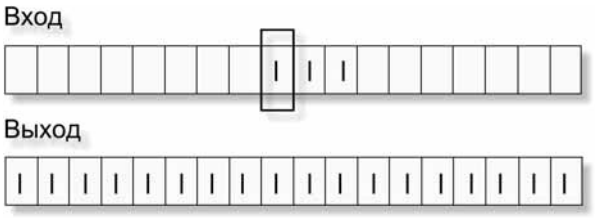

Рис. 12.9

Решение

Описание алгоритма. Рекурсивная пружина в этом решении записывает на ленту пять палочек на каждом шаге.

Программа

```
ЭТО Умножить_6
  // Отдельная обработка палочного нуля
  ЕСЛИ | ТО { ВПРАВО Умножить_6_работа }
КОНЕЦ

// Умножение ненулевых палочных чисел
ЭТО Умножить_6_работа
  ЕСЛИ |
    ТО { ВПРАВО Умножить_6_работа }
  // Рекурсивная пружинка
  ПОВТОРИ 5 { ПИШИ | ВПРАВО }
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 12.4.

Таблица 12.4

Тест	Комментарий	Ожидаемый результат
□	Палочное число равно нулю	□
|	Палочное число равно единице	||||||
|||||	Большое палочное число	||||||||||||||||||||||||||||

4. На ленте записаны два палочных числа, соединённые знаком вопроса. Заменить знак «?» на один из знаков «<», «>» или «=» в зависимости от отношения чисел. В начальный момент окно установлено на первый непустой символ записи.

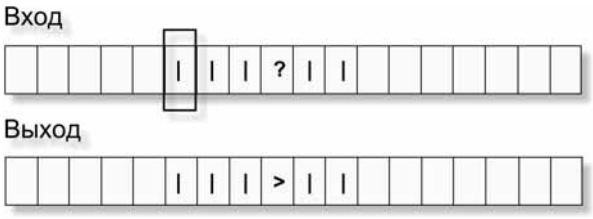

Рис. 12.10

Решение

Описание алгоритма. Подсчитаем количество палочек в первом числе (процедура Проверка1), используя ящик как счётчик. Затем перемещаем окно по записи второго числа и отнимаем по единице из счётчика на каждой палочке (процедура Проверка2). Так делаем до тех пор, пока второе число не закончится или в счётчике не окажется **ПУСТО**. Запишем ответ, проверяя значение счётчика:

- **ПУСТО** — второе число больше;
- 0 — числа равны;
- любое другое значение — первое число больше.

Программа

```
ЭТО Проверка
  // Обнулим счётчик перед началом подсчёта палочек в первом числе.
  ОБМЕН ПИШИ 0 ОБМЕН
  Проверка1 // Подсчитаем палочки в первом числе.
  Проверка2 // Сравним с количеством палочек второго числа.
  Найти_символ_?
  // Посмотрим, что в счётчике, и запишем ответ
  ОБМЕН
  ЕСЛИ       ПУСТО ТО ПИШИ <
  ИНАЧЕ ЕСЛИ 0     ТО ПИШИ =
  ИНАЧЕ               ПИШИ >
КОНЕЦ

ЭТО Найти_символ_?
  ПОКА НЕ ? ВЛЕВО
КОНЕЦ

ЭТО Проверка1
  ПОКА НЕ ?
  {
    ОБМЕН ПЛЮС ОБМЕН
    ВПРАВО
  }
КОНЕЦ

ЭТО Проверка2
```

```
  ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО
    {
      ОБМЕН
      ЕСЛИ НЕ ПУСТО
        ТО    { МИНУС ОБМЕН Проверка2 }
        ИНАЧЕ ОБМЕН
    }
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 12.5.

Таблица 12.5

Тест	Комментарий	Ожидаемый результат
□?□	Числа равны	□=□
|?|		|=|
||?||		||=||
|||?|||		|||=|||
|?□	Первое число больше	|>□
||?|		||>|
|||?|		|||>|
□?|	Второе число больше	□<|
|?||		|<||
|?|||		|<|||

5. Записать на ленту через запятые последовательность из первых n палочных чисел, начиная с единицы. Построить решение для n = 5, но так, чтобы программа работала и для других n после замены в ней числовой константы.

Решение

Алгоритм (идея Я. Н. Зайдельмана)

1. Записать в ящик единицу — количество палочек в первом числе.
2. Повторить 5 раз:
   2.1. Записать очередное число (количество палочек числа задано в ящике).

2.2. Занести в ящик количество палочек для следующего числа.

2.3. Записать разделитель (запятую).

3. Удалить последнюю запятую.

Замечание

Для записи очередного числа используется рекурсивная процедура Запись_числа. Эта процедура записывает на ленту палочки, уменьшая до нуля значение ящика. Рекурсивная пружинка восстанавливает значение ящика и увеличивает его на один для правильного построения следующего числа.

Программа

```
ЭТО Последовательность
  // Записать в ящик единицу — количество палочек в первом числе.
  ОБМЕН ПИШИ 1 ОБМЕН
  // Записать остальные числа
  ПОВТОРИ 5
  {
    Запись_числа     // Записать очередное число.
    ЯЩИК+ // Занести в ящик количество палочек в следующем числе.
    ПИШИ ,           // Записать разделитель.
  }
  // Удалить последнюю запятую
  ПИШИ ПУСТО
КОНЕЦ

ЭТО Запись_числа
  ВПРАВО
  ОБМЕН
  ЕСЛИ НЕ 0
    ТО { МИНУС ОБМЕН ПИШИ | Запись_числа ПЛЮС }
    ИНАЧЕ ПИШИ 1
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 12.6.

Таблица 12.6

Тест	Комментарий	Ожидаемый результат
□	n = 1 (ПОВТОРИ 1)	|
□	n = 1 (ПОВТОРИ 1)	|
□	n = 2 (ПОВТОРИ 2)	|,||
□	n = 5 (ПОВТОРИ 5)	|,||,|||,||||,|||||

6. Записать на ленту через запятые последовательность из первых n чётных палочных чисел, начиная с двойки. Построить решение для n = 5, но так, чтобы программа работала и для других n после замены в ней числовой константы.

Решение

Программа для решения этой задачи отличается от программы решения задачи 5 следующими небольшими модификациями:

- Перед циклом **ПОВТОРИ** в ящик записывается число 2 — количество палочек в первом числе:

```
ОБМЕН ПИШИ 2 ОБМЕН
```

- Перед работой рекурсивной пружины на ленту записывается число 2 (для увеличения количества палочек в следующем числе на 2):

```
  ИНАЧЕ ПИШИ 2
```

Программа

```
ЭТО Четные
  // Записать в ящик двойку — количество палочек в первом числе.
  ОБМЕН ПИШИ 2 ОБМЕН
  // Записать остальные числа
  ПОВТОРИ 5
  {
    Запись_числа      // Записать очередное число.
    ЯЩИК+ // Занести в ящик количество палочек в следующем числе.
    ПИШИ ,            // Записать разделитель.
  }
  // Удалить последнюю запятую
  ПИШИ ПУСТО
КОНЕЦ
```

```
ЭТО Запись_числа
  ВПРАВО
  ОБМЕН
  ЕСЛИ НЕ 0
    ТО { МИНУС ОБМЕН ПИШИ | Запись_числа ПЛЮС }
    ИНАЧЕ ПИШИ 2
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 12.7.

Таблица 12.7

Тест	Комментарий	Ожидаемый результат
□	n = 1 (ПОВТОРИ 1)	||
□	n = 2 (ПОВТОРИ 2)	||,||||
□	n = 5 (ПОВТОРИ 5)	||,||||,||||||,||||||||,||||||||||

7. Записать на ленту через запятые последовательность из первых n нечётных палочных чисел, начиная с единицы. Построить решение для n = 5, но так, чтобы программа работала и для других n после замены в ней числовой константы.

Решение

Программа для решения этой задачи отличается от программы решения задачи 5 одной строкой:

- Перед работой рекурсивной пружины на ленту записывается число 2 (для увеличения количества палочек в следующем числе на 2):

```
ИНАЧЕ ПИШИ 2
```

Программа

```
ЭТО Нечетные
  // Записать в ящик единицу — количество палочек в первом числе.
  ОБМЕН ПИШИ 1 ОБМЕН
  // Записать остальные числа
  ПОВТОРИ 5
  {
    Запись_числа     // Записать очередное число.
    ЯЩИК+ // Занести в ящик количество палочек в следующем числе.
```

```
    ПИШИ ,            // Записать разделитель.
  }
  // Удалить последнюю запятую
  ПИШИ ПУСТО
КОНЕЦ

ЭТО Запись_числа
  ВПРАВО
  ОБМЕН
  ЕСЛИ НЕ 0
    ТО { МИНУС ОБМЕН ПИШИ | Запись_числа ПЛЮС }
    ИНАЧЕ ПИШИ 2
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 12.8.

Таблица 12.8

Тест	Комментарий	Ожидаемый результат
□	n = 1 (ПОВТОРИ 1)	|
□	n = 2 (ПОВТОРИ 2)	|,|||
□	n = 5 (ПОВТОРИ 5)	|,|||,|||||,|||||||,|||||||||

12.3. Арифметика символов

1. Сумма символов. Последовательность символов записана на ленту плотно, без промежутков. В начальный момент окно установлено на первый символ записи. Найти сумму символьных чисел при условии, что она лежит в допустимом диапазоне, и отобразить её на ленте в виде обычного числа. Пример начального и конечного состояний среды показан на рис. 12.11.

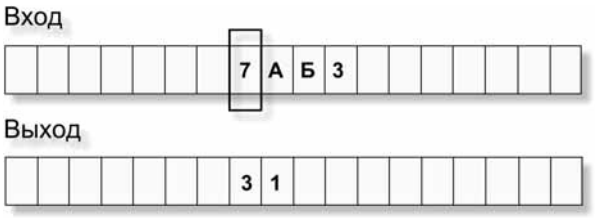

Рис. 12.11

Решение

Алгоритм

1. Записать в ящик число 0 — начальное значение суммы.
2. Пока запись на ленте не закончится, делать:
   2.1. Добавить к ящику численное значение текущего символа.
   2.2. Продвинуть окно к следующей ячейке.
3. Записать на ленту результат.

Программа

```
ЭТО Вход
  Обнулить_сумматор
  Суммировать
  Записать_результат
КОНЕЦ

ЭТО Обнулить_сумматор
  ОБМЕН ПИШИ 0 ОБМЕН
КОНЕЦ

ЭТО Суммировать
  ПОКА НЕ ПУСТО
  {
    // Добавить к ящику численное значение текущего символа
    // и записать в ячейку ПУСТО
    ПОКА НЕ 0 { МИНУС ОБМЕН ПЛЮС ОБМЕН }
    ПИШИ ПУСТО
    // Продвинуть окно к следующей ячейке.
    ВПРАВО
  }

ЭТО Записать_результат
  ЯЩИК-
  Символ_в_число
КОНЕЦ
```

Замечание

Процедура `Символ_в_число` описана в *главе 12* книги ученика.

Тесты

Набор тестов приводится в табл. 12.9.

Таблица 12.9

Тест	Комментарий	Ожидаемый результат
□	Пустая лента	0
5 A @	Один символ в записи	5 10 69
1AE4	Много символов в записи	31

2. Особые цифры. Подсчитать в записи количество цифр, которые больше 2, но меньше 9. В начальный момент окно установлено на первый символ записи. Пример начального и конечного состояний среды показан на рис. 12.12.

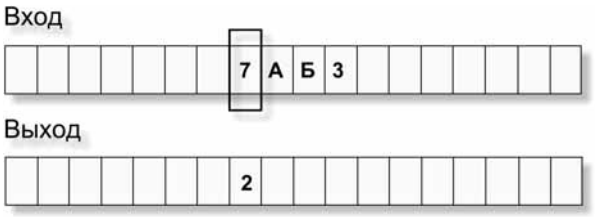

Рис. 12.12

Решение

Алгоритм

1. Записать в ящик число 0 — начальное значение счётчика особых цифр.
2. Пока запись на ленте не закончится, делать:
   2.1. Добавить к счётчику 1, если символ на ленте — особое число.
   2.2. Продвинуть окно к следующей ячейке.
3. Записать на ленту результат.

Программа

```
ЭТО Вход
  Обнулить_счетчик
  Просмотр_записи
```

```
  Записать_результат
КОНЕЦ

ЭТО Обнулить_счетчик
  ОБМЕН ПИШИ 0 ОБМЕН
КОНЕЦ

ЭТО Просмотр_записи
  ПОКА НЕ ПУСТО
  {
    // Добавить к счётчику 1, если символ на ленте — особое число.
    ЕСЛИ    ЦИФРА
    ТО ЕСЛИ НЕ 0
    ТО ЕСЛИ НЕ 1
    ТО ЕСЛИ НЕ 9
    ТО { ОБМЕН ПЛЮС ОБМЕН }
    // Продвинуть окно к следующей ячейке.
    ВПРАВО
  }

ЭТО Записать_результат
  ЯЩИК-
  Символ_в_число
КОНЕЦ
```

Замечание

Процедура `Символ_в_число` описана в *главе 12* книги ученика.

Тесты

Набор тестов приводится в табл. 12.10.

Таблица 12.10

Тест	Комментарий	Ожидаемый результат
□	Пустая лента	0
КОТ	Нет особых символов в записи	0
01901		0
А0Р91		0

Таблица 12.10 *(окончание)*

Тест	Комментарий	Ожидаемый результат
5 120 А5091Б	Один особый символ в записи	1 1 1
Л7К4Д19 32875	Несколько особых символов в записи	2 5
23456789 230454568789	Все особые символы присутствуют в записи	8 11

3. Шифровка. Каждый символ исходного сообщения закодирован двумя символами с использованием следующей кодировочной таблицы (табл. 12.11).

Таблица 12.11

Символ	Код	Символ	Код
0	00	9	09
1	01	А	10
2	02	Б	11
...	...	В	12
		...	...

Расшифровать сообщение. В начальный момент окно установлено на первый символ записи. На рис. 12.13 показан пример начального и конечного состояний среды исполнителя.

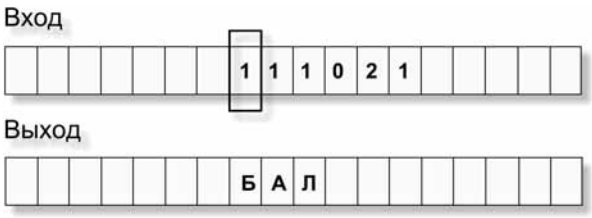

Рис. 12.13

Решение

Алгоритм

1. Если запись не пуста, то делать:
   1.1. Отдельно расшифровать первый символ: код `xy` заменить на □`s` (`xy` — код символа `s`) и передвинуть окно на код второго символа.
   1.2. Расшифровать остальные символы.
   1.3. Передвинуть окно на результат.

Алгоритм процедуры `Расшифровка_остальных`

1. Пока запись на ленте не закончится, делать:
   1.1. Расшифровать символ.
   1.2. Добавить символ к цепочке расшифрованных символов и передвинуть окно на следующий код в записи.

Описание алгоритма процедуры `Добавить`

Расшифрованный символ копируем в ящик и стираем с ленты. В рекурсивном цикле ищем конец цепочки расшифрованных символов. Дописываем символ из ящика и рекурсивной пружиной возвращаем окно в исходную запись.

Программа

```
ЭТО Вход
  ЕСЛИ НЕ ПУСТО
  ТО
  {
    Расшифровка_первого
    Расшифровка_остальных
    На_результат
  }
КОНЕЦ

ЭТО Расшифровка_первого
  Расшифровка_символа
  ВПРАВО // Окно на первый символ второго кода
КОНЕЦ

ЭТО Расшифровка_остальных
  ПОКА НЕ ПУСТО
  {
```

```
    Расшифровка_символа
    // Добавить символ к цепочке расшифрованных символов
    // и переместить окно к коду следующего символа.
    Добавить
  }
КОНЕЦ

ЭТО Добавить
  ЯЩИК+
  ПИШИ ПУСТО
  Добавить1
КОНЕЦ

ЭТО Добавить1
  ВЛЕВО
  ЕСЛИ ПУСТО
    ТО    Добавить1
    ИНАЧЕ { ВПРАВО ЯЩИК- }
  // Рекурсивная пружинка: на код следующего символа записи.
  ВПРАВО
КОНЕЦ

ЭТО На_результат
  ПОКА ПУСТО ВЛЕВО
КОНЕЦ

ЭТО Расшифровка_символа
  ЕСЛИ 0 ТО ПИШИ ПУСТО
  ВПРАВО
  Число_в_символ
КОНЕЦ

// Перевод числа в символьное число.
// Вход:  окно установлено на младшую цифру,
//        если старшей цифры нет, то слева пустая ячейка.
// Выход: окно установлено на результат (место младшей цифры),
//        а в ячейку слева записано ПУСТО.
```

```
// Ограничение: процедура работает для целых неотрицательных
//              чисел, значение которых меньше 100.
// ---------------------------------------------------------
ЭТО Число_в_символ
  // Установим окно на старшую цифру.
  ВЛЕВО
  ЕСЛИ НЕ ПУСТО
    ТО
    {
      // Умножаем старшую цифру на 10 и добавляем к младшей цифре.
      ПОВТОРИ 10
      {
        ЯЩИК+ ВПРАВО ОБМЕН
        ПОКА НЕ 0 { МИНУС ОБМЕН ПЛЮС ОБМЕН }
        ОБМЕН ВЛЕВО
      }
    }
  // Стираем старшую цифру и устанавливаем окно на результат.
  ПИШИ ПУСТО ВПРАВО
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 12.12.

Таблица 12.12

Тест	Комментарий	Ожидаемый результат
□	Пустая лента	0
00 09 10 69	Закодирован один символ	0 9 A @
0306 1069	Закодировано два символа	36 A@
1011121314	Закодировано несколько символов	АБВГД

4. Вычитание символов. Окно установлено на первый из двух символов на ленте. Найти модуль разности этих символьных чисел и записать ответ в виде обычного числа. Пример начального и конечного состояний среды показан на рис. 12.14.

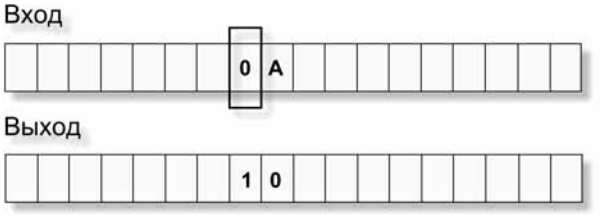

Рис. 12.14

Решение

Описание алгоритма. Если числа равны, то результат равен 0, если нет, то результат будет сформирован на месте наибольшего числа при выполнении следующих действий: пока наименьшее из двух чисел не равно нулю, вычитать из каждого из них по единице.

Программа

```
ЭТО Вход
  // Сравниваем числа и переключаем работу по трём процедурам.
  ЯЩИК+ ВПРАВО
  ЕСЛИ       Я=Л ТО Числа_равны
  ИНАЧЕ ЕСЛИ Я<Л ТО Второе_больше
  ИНАЧЕ              Второе_меньше
  // Переводим результат в обычное число
  Символ_в_число
КОНЕЦ

ЭТО Числа_равны
  // Стираем лишний символ
  ПИШИ ПУСТО
  // Записываем результат
  ВЛЕВО ПИШИ 0
КОНЕЦ

ЭТО Второе_больше
  ВЛЕВО
```

```
  ПОКА НЕ 0 { МИНУС ВПРАВО МИНУС ВЛЕВО }
  // Стираем лишний символ
  ПИШИ ПУСТО
  // Устанавливаем окно на результат
  ВПРАВО
КОНЕЦ

ЭТО Второе_меньше
  ПОКА НЕ 0 { МИНУС ВЛЕВО МИНУС ВПРАВО }
  // Стираем лишний символ
  ПИШИ ПУСТО
  // Устанавливаем окно на результат
  ВЛЕВО
КОНЕЦ
```

Замечание

Процедура `Символ_в_число` описана в *главе 12* книги ученика.

Тесты

Набор тестов приводится в табл. 12.13.

Таблица 12.13

Тест	Комментарий	Ожидаемый результат
□	Пустая лента	0
00 99 @@	Числа равны	0 0 0
0A AЯ	Второе число больше	10 31
@0 +Ъ	Второе число меньше	69 8

5. Умножение символов. Окно установлено на первый из двух символов на ленте. Найти произведение этих символьных чисел и записать ответ в виде обычного числа. Считать, что произведение лежит в диапазоне символьных чисел Корректора. Пример начального и конечного состояний среды показан на рис. 12.15.

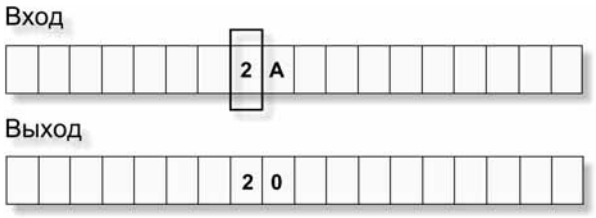

Рис. 12.15

Решение

Описание алгоритма. Если одно из чисел равно нулю, то и результат равен нулю, если нет, то умножение заменим сложением: первое число будем складывать само с собой столько раз, сколько указывает второй сомножитель.

Программа

```
ЭТО Вход
  ЕСЛИ 0
    ТО
    {
      ПИШИ ПУСТО ВПРАВО ПИШИ 0
    }
    ИНАЧЕ
    {
      ВПРАВО
      ЕСЛИ 0
        ТО    { ПИШИ ПУСТО ВЛЕВО ПИШИ 0 }
        ИНАЧЕ Умножение
    }
  Символ_в_число
КОНЕЦ


ЭТО Умножение
  // Подготовим ящик к накоплению результата.
  ОБМЕН ПИШИ 0 ОБМЕН
  // Будем складывать первое число с самим собой такое количество раз,
  // которое задаёт второе число. Второе число на ленте будем
  //  использовать как счётчик сложений, вычитая из него единицу при
  // каждом сложении.
```

```
  ПОКА НЕ 0
  {
    // Сохраним копию первого числа слева от него, не испортив при
    // этом содержимое ящика, в котором накапливается результат.
    ВЛЕВО ОБМЕН
    ВЛЕВО ЯЩИК-
    ВПРАВО ОБМЕН
    // Добавим к содержимому ящика копию первого числа
    ВЛЕВО // Окно на копию первого числа
    ПОКА НЕ 0 { МИНУС ОБМЕН ПЛЮС ОБМЕН }
    // Уменьшим значение второго числа на 1.
    ВПРАВО ВПРАВО
    МИНУС
  }
  // Очистим ленту и запишем на неё из ящика результат.
  ПОВТОРИ 2 { ПИШИ ПУСТО ВЛЕВО }
  ПИШИ ПУСТО
  ЯЩИК-
КОНЕЦ
```

Замечание

Процедура `Символ_в_число` описана в *главе 12* книги ученика.

Тесты

Набор тестов приводится в табл. 12.14.

Таблица 12.14

Тест	Комментарий	Ожидаемый результат
□	Пустая лента (ошибка в данных)	НЕ МОГУ
00	Одно из чисел равно нулю	0
09		0
@0		0
@1	Числа ненулевые	69
1@		69
A2		20
5Б		55

Глава 13

Преобразования, подсчёты, редактирование

Решение задач

В тестах, рекомендованных для проверки программ, пробел обозначен знаком ■, пустая ячейка — знаком □, а подчёркнутый символ показывает положение окна на ленте.

13.1. Длина текста

1. Подсчитать число букв О в тексте, если известно, что оно меньше числа символов в алфавите Корректора. В начальный момент окно установлено на первый пустой символ перед текстом (рис. 13.1).

Рис. 13.1

Решение

Алгоритм

1. Обнулить счётчик.
2. Перемещать окно по тексту и добавлять к счётчику 1 на каждой букве О в тексте.
3. Привести символьное число в ящике к обычной десятичной записи.

Программа

```
ЭТО Подсчет_О
  Подготовка_счетчика
  Подсчет
  Оформление_результата
КОНЕЦ

ЭТО Подготовка_счетчика
  ОБМЕН ПИШИ 0 ОБМЕН
КОНЕЦ

ЭТО Подсчет
  ВПРАВО
  ПОКА НЕ ПУСТО
  {
    ЕСЛИ О ТО { ОБМЕН ПЛЮС ОБМЕН }
    ВПРАВО
  }
КОНЕЦ

ЭТО Оформление_результата
  ВПРАВО ВПРАВО ЯЩИК-
  Символ_в_число
КОНЕЦ
```

Замечание

Процедура `Символ_в_число` описана в *главе 12* книги ученика.

Тесты

Набор тестов приводится в табл. 13.1.

Таблица 13.1

Тест	Комментарий	Ожидаемый результат
□	Пустое слово	□0
□1 □0	Слово из одного символа	1□□0 0□□1

Таблица 13.1 *(окончание)*

Тест	Комментарий	Ожидаемый результат
□ОБЛАКО □КОТ □ЛИСА	Слово из нескольких символов	ОБЛАКО□2 КОТ□□1 ЛИСА□0
□ООООО...ОООООООО	Число букв О на единицу меньше числа символов в алфавите Корректора	ООООО...ОООООООО□72

2. На ленте записано число. Подсчитать сумму наименьшей и наибольшей его цифр. В начальный момент окно установлено на первый пустой символ перед записью (рис. 13.2).

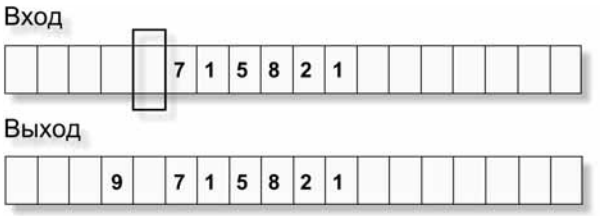

Рис. 13.2

Решение

Алгоритм главной процедуры

1. Найти максимальную цифру.
2. Запомнить на ленте максимальную цифру.
3. Найти минимальную цифру.
4. Сложить минимальную и максимальную цифры.
5. Записать ответ в виде обычного числа.

Алгоритм процедуры `Найти_макс` (результат формируется в ящике)

1. Записать в ящик **пусто** (символ с наименьшим алфавитным номером).
2. Переместить окно на первый символ записи.
3. Пока символы записи не закончатся, делать:
   3.1. Если алфавитный номер символа в окне больше алфавитного номера символа в ящике, записать символ из окна в ящик.
   3.2. Сместить окно к следующему символу.

Программа

```
ЭТО Макс+Мин
  Найти_макс
  Запомнить_макс
  Найти_мин
  Сложить
  Записать_ответ
КОНЕЦ

// Нахождение символа с максимальным алфавитным номером
// Результат формируется в ящике
ЭТО Найти_макс
  ЯЩИК+
  ВПРАВО
  ПОКА НЕ ПУСТО
  {
    ЕСЛИ Я<Л ТО ЯЩИК+
    ВПРАВО
  }
КОНЕЦ

// Найденный максимальный символ записывается на ленту
// через две пустые ячейки справа от записи
ЭТО Запомнить_макс
  ВПРАВО ВПРАВО ЯЩИК- ПОВТОРИ 3 ВЛЕВО
КОНЕЦ

// Нахождение символа с минимальным алфавитным номером
// Результат формируется в ящике
ЭТО Найти_мин
  ЯЩИК+
  ВЛЕВО
  ПОКА НЕ ПУСТО
  {
    ЕСЛИ Я>Л ТО ЯЩИК+
    ВЛЕВО
```

```
  }
КОНЕЦ

// Сложение цифр
ЭТО Сложить
  // Перемещаем окно к записанной на ленте максимальной цифре
  ВПРАВО
  ПОКА НЕ ПУСТО ВПРАВО
  ВПРАВО ВПРАВО
  // Формируем на ленте символ, алфавитный номер которого равен
  // сумме алфавитных номеров цифр в ящике и на ленте
  ОБМЕН // Помещаем на ленту минимум для ускорения сложения
        // (смотрите замечание в конце листинга)
  ПОКА НЕ 0 { МИНУС ОБМЕН ПЛЮС ОБМЕН }
  // Символ-сумма найден, он в ящике. Запишем его на ленту
  ЯЩИК-
КОНЕЦ

ЭТО Записать_ответ
  Символ_в_число // Процедура из главы 12
КОНЕЦ
```

Замечание

Перед нахождением символа, алфавитный номер которого равнялся бы сумме алфавитных номеров минимальной и максимальной цифры (в процедуре `Сложить`), Вася совершенно резонно записывает команду **ОБМЕН**. Перед выполнением этой команды на ленте записана максимальная цифра, а в ящике минимальная. Обмен позволяет ускорить вычисления, т. к. единица будет вычитаться из минимального числа, а добавляться к максимальному, что в большинстве случаев существенно сократит число оборотов цикла нахождения суммы. Однако можно было бы обойтись без этой команды, если в первом проходе числа находить не максимальную цифру (как сделал Вася), а минимальную.

Тесты

Набор тестов приводится в табл. 13.2.

Таблица 13.2

Тест	Комментарий	Ожидаемый результат
□0 □9	Число содержит только одну цифру	0□0 18□9
□31663 □11711	Искомая сумма меньше 10	9□31663 8□11711
□8182	Искомая сумма равна 10	10□8182
□99999 □3018	Искомая сумма больше 10	18□99999 11□3018

3. На ленте записан текст. Записать на ленту число слов в нём, если известно, что их количество меньше числа символов в алфавите Корректора, а начальные и конечные пробелы в тексте отсутствуют. В начальный момент окно установлено на первый символ текста (рис. 13.3).

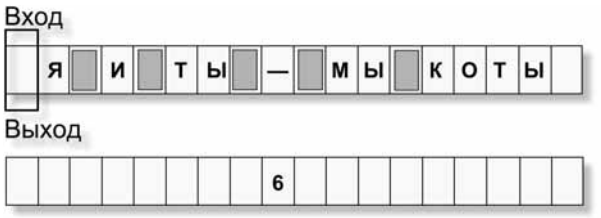

Рис. 13.3

Решение

Алгоритм главной процедуры

1. Если лента пуста, записать ответ 0, иначе:
   1.1. Записать в счётчик 1.
   1.2. Подсчитать число групп из подряд идущих пробелов в тексте.
   1.3. Записать ответ.

Программа

```
ЭТО Число_слов
  ЕСЛИ ПУСТО
    ТО ПИШИ 0
    ИНАЧЕ
    {
```

```
      В_счетчик_1
      Работа
      Ответ
    }
КОНЕЦ

ЭТО В_счетчик_1
  ОБМЕН ПИШИ 1 ОБМЕН
КОНЕЦ

ЭТО Добавить_к_счетчику_1
  ОБМЕН ПЛЮС ОБМЕН
КОНЕЦ

ЭТО Работа
  ПОКА НЕ ПУСТО
  {
    ЕСЛИ ПРОБЕЛ
      ТО
      {
        Добавить_к_счетчику_1
        Пропустить_пробелы
      }
    ВПРАВО
  }
КОНЕЦ

ЭТО Пропустить_пробелы
  ПОКА ПРОБЕЛ ВПРАВО
КОНЕЦ

ЭТО Ответ
  ВПРАВО ВПРАВО ЯЩИК-
  Символ_в_число
КОНЕЦ

ЭТО Записать_ответ
  Символ_в_число // Процедура из главы 12
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 13.3.

Таблица 13.3

Тест	Комментарий	Ожидаемый результат
□	Лента пуста	0
КОТ 1	В тексте одно слово	КОТ□□1 1□□1
1□2 11□□22 1□□□22	В тексте два слова	1□2□□2 11□□22□□2 1□□□22□□2
КОТ□И□ЛИСА 1□□2□□□3 1□2□...□9 1□2□...□10 1□2□...□11	В тексте много слов	КОТ□И□ЛИСА□□3 1□□2□□□3□□3 1□2□...□9□□9 1□2□...□10□10 1□2□...□11□11
1□2□...□72	Число слов в тексте на 1 меньше числа символов в алфавите Корректора	1□2□...□72□72

13.2. Корректор оправдывает своё имя

1. На ленте записан текст. Убрать из него лишние пробелы (каждую группу пробелов заменить одним пробелом). В начальный момент окно установлено на первый пустой символ перед текстом (рис. 13.4).

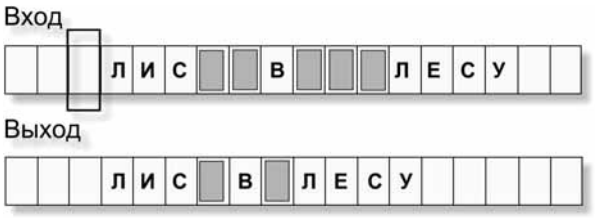

Рис. 13.4

Решение

Алгоритм главной процедуры

1. Сместить окно на первый символ текста.
2. Пока не **ПУСТО**, делать:
   2.1. Если текущий символ пробел:
      2.1.1. Сдвинуть окно к следующему символу.
      2.1.2. Пока **ПРОБЕЛ**, удалять текущий символ из текста.
   2.2. Переместить окно на следующий символ.

Программа

```
ЭТО Лишние_пробелы
  ВПРАВО
  ПОКА НЕ ПУСТО
    {
      ЕСЛИ ПРОБЕЛ
        ТО
        {
          // Если за первым пробелом есть
          // ещё пробелы - удаляем их
          ВПРАВО
          ПОКА ПРОБЕЛ { Удаление ВПРАВО }
        }
      ВПРАВО // К следующему символу
    }
КОНЕЦ

// Удаление символа записи.
// Место удаления показывает окно (любой символ записи).
// После удаления окно смещено на одну позицию влево по
// отношению к исходному положению, а в ящике - удалённый символ.
// Вход: ЗАКПИСЬ
// Выход: ЗАПИСЬ (в ящике удаленный символ К)
// Идея В. П. Семенко
ЭТО Удаление
  ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО Удаление
```

```
    ИНАЧЕ { ЯЩИК+ ВЛЕВО }
  ОБМЕН ВЛЕВО // Пружинка для сдвига остатка на одну ячейку влево
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 13.4.

Таблица 13.4

Тест	Комментарий	Ожидаемый результат
□	Пустой текст	□
□□ □□□ □□□□□	Текст из одних пробелов	□ □ □
□8□88□9 □8□□88□□□9 □□8□□88□□□9□ □□□□8□□88□□□9□□□	Разные варианты расположения пробелов и разное их количество	8□88□9 8□88□9 □8□88□9□ □8□88□9□

2. На ленте записан текст. Добавить пробел после каждого из знаков препинания (.,!?:;), если пробел отсутствует. В начальный момент окно установлено на первый пустой символ перед текстом (рис. 13.5).

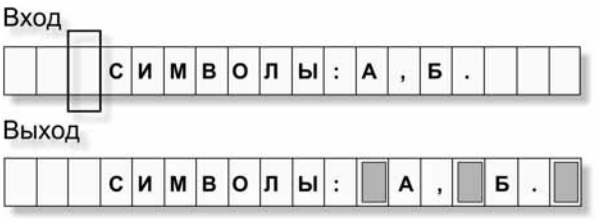

Рис. 13.5

Решение

Описание алгоритма. Перемещаем окно вправо по тексту и каждый раз, когда в тексте встречается знак препинания без пробела, вставляем его при помощи процедуры `Вставка`.

Программа

```
ЭТО Вставка_пробелов
  ВПРАВО
  ПОКА НЕ ПУСТО
  {
    ЕСЛИ       . ТО Работа
    ИНАЧЕ ЕСЛИ , ТО Работа
    ИНАЧЕ ЕСЛИ ! ТО Работа
    ИНАЧЕ ЕСЛИ ? ТО Работа
    ИНАЧЕ ЕСЛИ : ТО Работа
    ИНАЧЕ ЕСЛИ ; ТО Работа
    // Передвинуть окно на следующий символ
    ВПРАВО
  }
КОНЕЦ

ЭТО Работа
  // Сместимся к месту вставки
  ВПРАВО
  // Занесём в ящик символ пробела
  ОБМЕН ПИШИ ПРОБЕЛ ОБМЕН
  // Обратимся к процедуре, вставляющей символ из ящика
  Вставка
КОНЕЦ

// Вставка символа из ящика в запись.
// Место вставки показывает окно (любой символ записи
// или первый символ за записью).
// Вход: ЗАИСЬ (в ящике вставляемый символ П)
// Выход: ЗАПИСЬ (в ящике ПУСТО)
// Идея В. П. Семенко
ЭТО Вставка
  ОБМЕН ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО    Вставка
    ИНАЧЕ ОБМЕН
  ВЛЕВО // Пружинка для возврата на вставленный символ
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 13.5.

Таблица 13.5

Тест	Комментарий	Ожидаемый результат
□	Пустой текст	□
□1 □КОТ□БАЗИЛИО	Текст без знаков препинания	1 КОТ□БАЗИЛИО
□.,!?:; □1.2,3!4?5:6; □1.□2,□□3!□4?5:6;7 □.□!□28□,□88□□□	Разные варианты расположения пробелов и знаков препинания в тексте	.□,□!□?□:□;□ 1.□2,□3!□4?□5:□6;□ 1.□2,□□3!□4?□5:□6;□7 □.□!□28□,□88□□□

3. На ленте записан текст. В нём встречаются символы «#». Перед текстом записан образец, отделённый от основного текста символом «|». Получить новый текст, в котором все символы «#» заменены этим образцом. В начальный момент окно установлено на символ «|», расположенный между словом и текстом (рис. 13.6).

Рис. 13.6

Решение

Описание алгоритма. Будем по символу создавать копию текста справа от него, через пустую ячейку. Перенос обычного символа будем выполнять процедурой `Копировать_символ`, а вместо символа «#» будем переносить в формируемый результат образец процедурой `Копировать_образец`.

Программа

```
ЭТО Замена
  ВПРАВО
  ПОКА НЕ ПУСТО
```

```
  {
    ЕСЛИ #
      ТО Копировать_образец
      ИНАЧЕ
      {
        ЯЩИК+
        Копировать_символ
      }
    ВПРАВО
  }
КОНЕЦ

// Добавление символа в конец результата и возврат
// окна к текущему символу в исходном тексте.
ЭТО Копировать_символ
  ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО    Копировать_символ
    ИНАЧЕ Дописать_символ
  // Отложенная команда
  ВЛЕВО  // Она вернёт окно к текущему месту
КОНЕЦ

ЭТО Дописать_символ
  ВПРАВО
  ПОКА НЕ ПУСТО ВПРАВО
  ЯЩИК-
  ПОКА НЕ ПУСТО ВЛЕВО
КОНЕЦ

ЭТО Копировать_образец
  ВЛЕВО
  ЕСЛИ НЕ ПУСТО
    ТО    Копировать_образец
    ИНАЧЕ Перенос_образца
  // Отложенная команда
  ВПРАВО // Она вернёт окно к текущему месту
```

```
КОНЕЦ


ЭТО Перенос_образца
  ВПРАВО
  ПОКА НЕ |
  {
    ЯЩИК+
    Копировать_символ
    ВПРАВО
  }
  // Вернуться к началу образца,
  // чтобы рекурсивная пружинка в процедуре Копировать_образец
  // начала работать с правильного места.
  ВЛЕВО
  ПОКА НЕ ПУСТО ВЛЕВО
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 13.6.

Таблица 13.6

Тест	Комментарий	Ожидаемый результат
КОТ|	Пустой текст	КОТ|
|#1#2#3#	Пустой образец	|#1#2#3#□123
|	Пустой текст и пустой образец	|
КОТ|<#|#>	В тексте присутствует символ |	КОТ|<#|#>□<КОТ|КОТ>
###|1#2#	В образце присутствует символ #	###|1#2#□1###2###
0|1#####	Разные варианты расположения знаков # в тексте	0|1#####□100000
КОТ□|###		КОТ|###□КОТ□КОТ□КОТ□
КОТ|1#2##3		КОТ|1#2##3□1КОТ2КОТКОТ3
КОТ|123#456#		КОТ|123#456#□123КОТ456КОТ

4. Перед текстом на ленте записан двухсимвольный образец, который отделён от текста знаком «|». Проверить, входит ли этот образец в текст как

составная часть (известно, что образец не может быть на последнем месте в тексте). Если вхождение найдено, установить окно на его конец, в противном случае установить окно на первую пустую ячейку за текстом. В начальный момент окно установлено на первый символ образца (рис. 13.7).

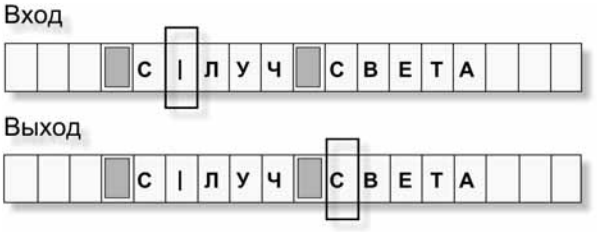

Рис. 13.7

Решение

Описание алгоритма. Копируем первый символ образца в ящик и просматриваем символы текста, пока они не закончатся (в тексте нет образца) или пока не появится символ в тексте, совпадающий с символом в ящике (первым символом образца). Просмотр символов текста выполняем при помощи цикла `пока`.

Когда символ текста совпадает с первым символом образца, проверяем, не совпадает ли идущий за ним символ со вторым символом образца. Для этого рекурсивно возвращаемся к началу текста и копируем второй символ образца в ящик, а затем, при помощи рекурсивной пружинки, возвращаем окно к текущему месту в тексте.

Если и второй символ образца совпал с соответствующим символом на ленте, образец найден. Прерываем цикл `пока`, записывая на ленту `пусто`. За пределами цикла проверяем причину его завершения:

- Текущий символ `пусто`, но за ним есть непустые символы: образец найден, цикл искусственно прерван записью символа `пусто` внутрь текста. Заменяем `пусто` вторым символом образца и завершаем работу.
- Текущий символ `пусто`, следующий символ за ним также `пусто`. Образец в тексте не найден, завершаем работу.

Замечание

Искусственное завершение цикла `пока` стало возможно из-за наличия в условии ограничения: текст не может завершаться образцом (иначе нельзя было бы понять, почему завершился цикл: все символы просмотрены и образец не найден или образец располагается в последних двух ячейках текста). Можно решить задачу, исключив из условия это ограничение, но тогда придётся заменить цикл `пока` рекурсивным просмотром текста.

Программа

```
ЭТО Поиск
  ПОВТОРИ 2 ВЛЕВО
  // Запомним первый символ образца
  ЯЩИК+
  // Передвинем окно на начало текста
  ПОВТОРИ З ВПРАВО
  // Ищем в тексте первый символ образца
  ПОКА НЕ ПУСТО
  {
    ЕСЛИ Я=Л
     ТО
     {
       // Запомнить в ящике второй символ образца
       Запомнить_второй_символ_образца
       ПОВТОРИ 2 ВПРАВО
       ЕСЛИ Я=Л
         ТО ПИШИ ПУСТО // Для прерывания цикла
         ИНАЧЕ
         {
           // Восстановим в ящике первый символ образца
           // и продолжим поиск
           ВЛЕВО ЯЩИК+ ВПРАВО ВПРАВО
         }
     }
     ИНАЧЕ ВПРАВО
  }
  // Если образец найден, восстановим второй его символ на ленте
  ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО { ВЛЕВО ЯЩИК- }
    ИНАЧЕ ВЛЕВО
КОНЕЦ

ЭТО Запомнить_второй_символ_образца
  ВЛЕВО
  ЕСЛИ НЕ |
```

```
    ТО
    {
      Запомнить_второй_символ_образца
      ВПРАВО // Возврат на текущее место в тексте
    }
    ИНАЧЕ
    {
      ВЛЕВО ЯЩИК+ ВПРАВО
    }
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 13.7.

Таблица 13.7

Тест	Комментарий	Ожидаемый результат
12|	Пустой текст	12|□
12|3	В тексте нет образца	12|3□
12|345		12|345□
12|12A	В тексте есть образец	12|12A
12|A1Б12B		12|A1Б12B
12|1A1212Б		12|1A1212Б
12|12112112A		12|12112112A

5. На ленте записан текст. Упорядочить его символы по возрастанию их порядковых номеров в алфавите Корректора. В начальный момент окно установлено на первый символ текста (рис. 13.8).

Рис. 13.8

Решение

Описание алгоритма. Будем использовать метод «пузырька». Суть метода: просматривать символы текста, начиная со второго, и попутно менять местами текущий и предыдущий символы, если их алфавитные номера не возрастают. Такие просмотры нужно выполнять до тех пор, пока в тексте не будет сделано ни одной перестановки: все символы упорядочены.

Рекурсивная процедура `Упорядочивание`, код которой представлен далее, обеспечивает такой цикл просмотров. В начале каждого просмотра в ящик записывается символ **ПУСТО** — признак упорядоченной записи. Если в записи выполняется перестановка символов, то в ящик заносится 1 — признак неупорядоченности. Процедура `Упорядочивание` заканчивает свою работу, как только после очередного просмотра в ящике оказывается символ **ПУСТО**.

Программа

```
ЭТО Упорядочивание
  ВПРАВО
  // Поместим в ящик ПУСТО - признак упорядоченности
  ОБМЕН ПИШИ ПУСТО ОБМЕН
  // Просмотр текста, начиная со второго символа
  ПОКА НЕ ПУСТО
  {
    ОБМЕН ВЛЕВО
    // Если текущий и предыдущий символ не упорядочены,
    // переставляем их местами.
    ЕСЛИ Я<Л
      ТО
      {
        ОБМЕН ВПРАВО ОБМЕН
        // Была перестановка, пишем в ящик 1 - флаг перестановки
        ОБМЕН ПИШИ 1 ОБМЕН
      }
      ИНАЧЕ { ВПРАВО ОБМЕН }
    ВПРАВО
  }
  // Нужно ли делать новый проход?
  ОБМЕН
  ЕСЛИ НЕ ПУСТО
    ТО
```

```
    {
      // Флаг установлен. Стираем его и делаем новый проход
      ПИШИ ПУСТО
      Назад
      Упорядочивание
    }
КОНЕЦ

// Возврат на начало текста
ЭТО Назад
  ВЛЕВО
  ПОКА НЕ ПУСТО ВЛЕВО
  ВПРАВО
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 13.8.

Таблица 13.8

Тест	Комментарий	Ожидаемый результат
□	Пустой текст	□
1 | 2345 12345A|@	Текст упорядочен	1 | 2345 12345A|@
|1 |1| 543221 |543215 @12430	Разные варианты неупорядоченного текста	1| 1|| 122345 123455| 01234@

Глава 14

Трансляторы

Эта главу следует рассматривать как пропедевтическую прелюдию к задачам трансляции, которые подробно обсуждаются в *части III «Транслятор?.. Это очень просто!»*.

Ответы на вопросы и решения задач

В тестах, рекомендованных для проверки программ, пробел обозначен знаком ■, пустая ячейка — знаком □, а подчёркнутый символ показывает положение окна на ленте.

14.1. Проверка объектов

1. Идентификатор — это последовательность из букв и цифр, которая начинается с буквы. Проверить, является ли текст на ленте идентификатором. В начальный момент окно установлено на первый символ текста. На рис. 14.1 показаны примеры работы программы.

Рис. 14.1

Решение

Описание алгоритма. Пусть процедура `Проверка_символа` копирует текущий символ в ящик, а на ленту записывает результат анализа этого символа:

- 0 — символ оказался цифрой;
- 1 — символ оказался буквой;
- 2 — **ПУСТО**;
- 3 — другой символ.

Проверим первый символ записи в процедуре `Вход`, а остальные в рекурсивной процедуре `Идентификатор`.

Программа

```
ЭТО Вход
  Проверка_символа
  ЕСЛИ 1 ТО Идентификатор
  ИНАЧЕ     Ошибка
КОНЕЦ


ЭТО Идентификатор
  ОБМЕН
  ВПРАВО
  Проверка_символа
  ЕСЛИ       0 ТО Идентификатор
  ИНАЧЕ ЕСЛИ 1 ТО Идентификатор
  ИНАЧЕ ЕСЛИ 2 ТО Ответ_да
  ИНАЧЕ           Ошибка
КОНЕЦ

// Процедура записывает на ленту
// код результата проверки символа:
// 0 - цифра
// 1 - буква
// 2 - ПУСТО
// 3 - другой символ
// Сам проверяемый символ процедура отправляет в ящик
ЭТО Проверка_символа
  ЕСЛИ       ПУСТО ТО { ОБМЕН ПИШИ 2 }
```

```
  ИНАЧЕ ЕСЛИ ЦИФРА ТО { ОБМЕН ПИШИ 0 }
  ИНАЧЕ
  {
    ОБМЕН ПИШИ 9 ОБМЕН
    ЕСЛИ Я<Л
      ТО
      {
        ОБМЕН ПИШИ ПРОБЕЛ ОБМЕН
        ЕСЛИ Я>Л
          ТО { ОБМЕН ПИШИ 1 }
          ИНАЧЕ { ОБМЕН ПИШИ 3 }
      }
      ИНАЧЕ { ОБМЕН ПИШИ 3 }
  }
КОНЕЦ

ЭТО Ответ_да
  ОБМЕН
  ВПРАВО ПИШИ Д
  ВПРАВО ПИШИ А
КОНЕЦ

ЭТО Ошибка
  ОБМЕН
  ПОКА НЕ ПУСТО ВПРАВО
  ВПРАВО ПИШИ Н
  ВПРАВО ПИШИ Е
  ВПРАВО ПИШИ Т
КОНЕЦ
```

Замечание

Если в процедуру Ошибка встроить рекурсивную пружинку, то после записи ответа НЕТ окно будет устанавливаться на место обнаруженной ошибки:

```
ЭТО Ошибка
  ОБМЕН
  Ошибка1
КОНЕЦ
```

```
ЭТО Ошибка1
  ЕСЛИ НЕ ПУСТО
    ТО { ВПРАВО Ошибка1 }
    ИНАЧЕ
    {
      ВПРАВО ПИШИ Н
      ВПРАВО ПИШИ Е
      ВПРАВО ПИШИ Т
      ВЛЕВО ВЛЕВО
    }
  ВЛЕВО // Возврат окна на место ошибки
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 14.1.

Таблица 14.1

Тест	Комментарий	Ожидаемый результат
А	Запись является идентификатором	А□ДА
Я		Я□ДА
Р1		Р1□ДА
КОТ		КОТ□ДА
К123		К123□ДА
К1Т2П		К1Т2П□ДА
□	Запись не является идентификатором	□НЕТ
1		1□НЕТ
1А		1А□НЕТ
Л23#5		Л23#5□НЕТ
АН-1		АН-1□НЕТ
КОТ□И□ЛИС		КОТ□И□ЛИС□НЕТ

2. На ленте записан текст. Проверить, является ли он числом, и записать результат проверки на ленте в виде табл. 14.2.

Таблица 14.2

Запись результата	Что означает
ЦП	Целое положительное и нуль без знака и со знаком «+» (примеры: 0, +0, 75, +83)
ЦО	Целое отрицательное и нуль со знаком минус (примеры: –0, –17)
ДП	Простая положительная дробь (пример: 55/65, +85/10)
ДО	Простая отрицательная дробь (пример: –25/45)
ОШ	Ошибка в записи

Замечание

Записи с нулём в знаменателе считаются формально правильными.

В начальный момент окно установлено на первый пустой символ перед записью. На рис. 14.2 показаны примеры работы программы.

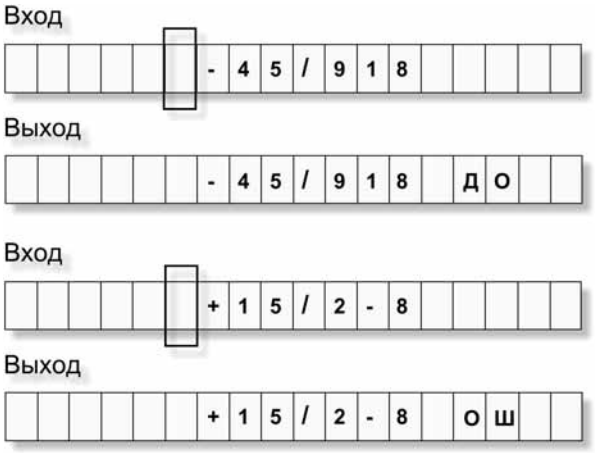

Рис. 14.2

Решение

Описание алгоритма

Первым символом в правильной записи может быть знак «–», знак «+» или цифра. Любой другой символ (в том числе и `пусто`) означает ошибку (рис. 14.3).

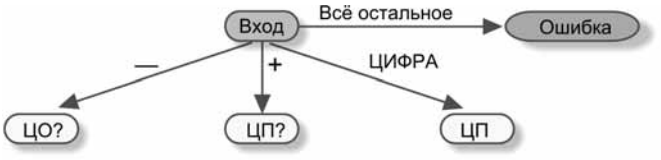

Рис. 14.3

Эти рассуждения можно записать в виде главной процедуры `Вход` (процедура `Шаг` перемещает окно к следующему символу):

```
ЭТО Вход
  Шаг
  ЕСЛИ         -      ТО Целое_отрицательное?
  ИНАЧЕ ЕСЛИ   +      ТО Целое_положительное?
  ИНАЧЕ ЕСЛИ ЦИФРА ТО Целое_положительное
  ИНАЧЕ                  Ошибка
КОНЕЦ
```

На схеме (рис. 14.3) эти процедуры обозначены соответственно как состояния «Вход», «ЦО?», «ЦП?», «ЦП», «Ошибка».

Посмотрим, что должна делать процедура `Целое_отрицательное?`. В эту процедуру передаётся управление, когда первый символ записи оказался знаком «–». Понятно, что за этим знаком обязана идти цифра, всё остальное является ошибкой (рис. 14.4).

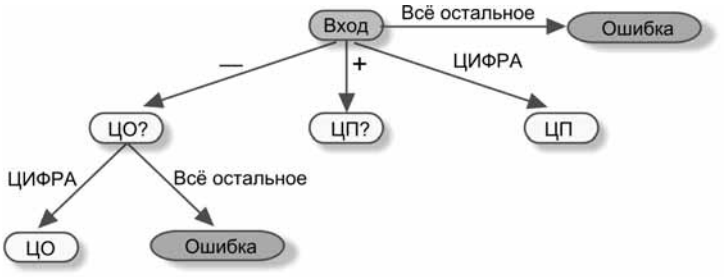

Рис. 14.4

Значит, процедура `Целое_отрицательное?` должна иметь вид:

```
ЭТО Целое_отрицательное?
  Шаг
  ЕСЛИ ЦИФРА ТО Целое_отрицательное
  ИНАЧЕ         Ошибка
КОНЕЦ
```

На схеме (рис. 14.4) процедура `Целое_отрицательное` обозначена как состояние «ЦО». В этом состоянии правильными символами являются:

- цифра. Управление рекурсивно передаётся на процедуру `Целое_отрицательное` (состояние ЦО);
- **ПУСТО**. Управление передаётся на процедуру `Ответ_ЦО` (запись на ленту ответа ЦО);
- знак «/». Управление передаётся на процедуру `Дробь_отрицательная?`.

Любой другой символ является ошибочным, и управление передаётся на процедуру `Ошибка`.

Теперь схема принимает вид, изображённый на рис. 14.5 (чтобы не загромождать схему, на ней не показаны переходы в состояние «Ошибка»).

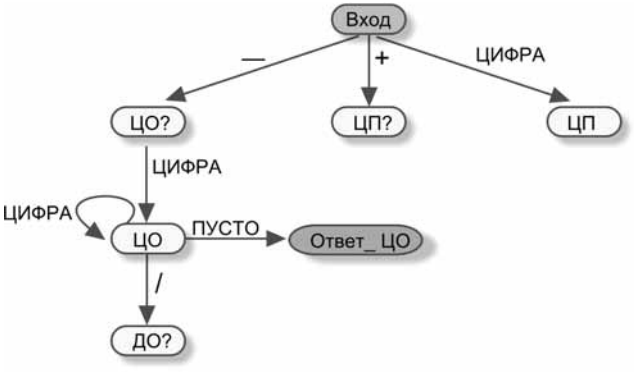

Рис. 14.5

Соответственно процедура `Целое_отрицательное` кодируется так:

```
ЭТО Целое_отрицательное
  Шаг
  ЕСЛИ        ЦИФРА ТО Целое_отрицательное
  ИНАЧЕ ЕСЛИ  /     ТО Дробь_отрицательная?
  ИНАЧЕ ЕСЛИ ПУСТО  ТО Ответ_ЦО
  ИНАЧЕ                Ошибка
КОНЕЦ
```

Продолжая анализировать другие возможные состояния, возникающие при проверке исходной записи, построим полную её схему (рис. 14.6).

На схеме (рис. 14.6) переходы в состояние «Ошибка» также не показаны. По этой схеме очень легко написать код программы.

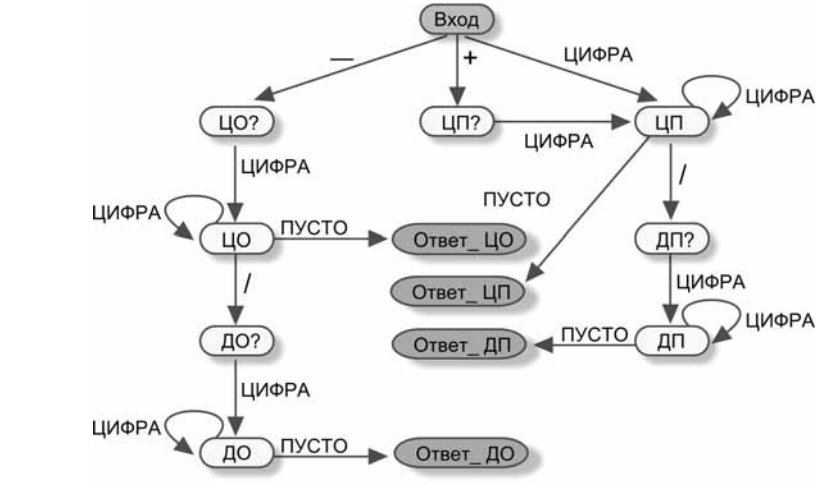

Рис. 14.6

Программа

```
ЭТО Вход
  Шаг
  ЕСЛИ        -      ТО Целое_отрицательное?
  ИНАЧЕ ЕСЛИ  +      ТО Целое_положительное?
  ИНАЧЕ ЕСЛИ ЦИФРА ТО Целое_положительное
  ИНАЧЕ                  Ошибка
КОНЕЦ

ЭТО Целое_отрицательное?
  Шаг
  ЕСЛИ ЦИФРА ТО Целое_отрицательное
  ИНАЧЕ         Ошибка
КОНЕЦ

ЭТО Целое_отрицательное
  Шаг
  ЕСЛИ        ЦИФРА ТО Целое_отрицательное
  ИНАЧЕ ЕСЛИ  /      ТО Дробь_отрицательная?
  ИНАЧЕ ЕСЛИ ПУСТО ТО Ответ_ЦО
  ИНАЧЕ                  Ошибка
КОНЕЦ
```

```
ЭТО Дробь_отрицательная?
  Шаг
  ЕСЛИ ЦИФРА ТО Дробь_отрицательная
  ИНАЧЕ         Ошибка
КОНЕЦ

ЭТО Дробь_отрицательная
  Шаг
  ЕСЛИ       ЦИФРА ТО Дробь_отрицательная
  ИНАЧЕ ЕСЛИ ПУСТО ТО Ответ_ДО
  ИНАЧЕ               Ошибка
КОНЕЦ

ЭТО Целое_положительное?
  Шаг
  ЕСЛИ ЦИФРА ТО Целое_положительное
  ИНАЧЕ         Ошибка
КОНЕЦ

ЭТО Целое_положительное
  Шаг
  ЕСЛИ ЦИФРА ТО Целое_положительное
  ИНАЧЕ ЕСЛИ ПУСТО ТО Ответ_ЦП
  ИНАЧЕ ЕСЛИ / ТО Дробь_положительная?
  ИНАЧЕ         Ошибка
КОНЕЦ

ЭТО Дробь_положительная?
  Шаг
  ЕСЛИ ЦИФРА ТО Дробь_положительная
  ИНАЧЕ         Ошибка
КОНЕЦ

ЭТО Дробь_положительная
  Шаг
  ЕСЛИ ЦИФРА ТО Дробь_положительная
```

```
  ИНАЧЕ ЕСЛИ ПУСТО ТО Ответ_ДП
  ИНАЧЕ          Ошибка
КОНЕЦ

ЭТО Шаг
  ВПРАВО
КОНЕЦ

ЭТО За_конец
  ПОКА НЕ ПУСТО Шаг
  Шаг
КОНЕЦ

ЭТО Ответ_ЦО
  За_конец
  ПИШИ Ц Шаг ПИШИ О
КОНЕЦ

ЭТО Ответ_ЦП
  За_конец
  ПИШИ Ц Шаг ПИШИ П
КОНЕЦ

ЭТО Ответ_ДО
  За_конец
  ПИШИ Д Шаг ПИШИ О
КОНЕЦ

ЭТО Ответ_ДП
  За_конец
  ПИШИ Д Шаг ПИШИ П
КОНЕЦ


ЭТО Ошибка
  За_конец
  ПИШИ О Шаг ПИШИ Ш
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 14.3.

Таблица 14.3

Тест	Комментарий	Ожидаемый результат
□1	Целое положительное	1□ЦП
□+4		+4□ЦП
□75		75□ЦП
□+83		+83□ЦП
□-1	Целое отрицательное	-1□ЦО
□-17		-17□ЦО
□1/7	Дробь положительная	1/7□ДП
□+1/7		+1/7□ДП
□55/65		55/65□ДП
□+85/10		+85/10□ДП
□-1/2	Дробь отрицательная	-1/2□ДО
□-25/45		-25/45□ДО
□	Ошибка в записи	□ОШ
□+		+□ОШ
□-		-□ОШ
□/		/□ОШ
□+/		+/□ОШ
□-/		-/□ОШ
□+/		+/□ОШ
□1/		1/□ОШ
□+1/		+1/□ОШ
□-1/		-1/□ОШ
□/2		/2□ОШ
□12A3		12A3□ОШ
□2-3		2-3□ОШ
□2+3		2+3□ОШ

14.2. Транслятор для Плюсика

1. Найти и исправить логическую ошибку в программе Транслятор.

Решение

Программа, написанная ребятами, работает правильно за исключением одной детали: последняя команда С устанавливается не вплотную к другим командам, а через пустую ячейку. Причина кроется в алгоритме работы процедуры Записать символ. Сначала эта процедура ищет пустую ячейку, разделяющую исходную запись примера от программы его вычисления, а затем пустую ячейку за концом программы (чтобы записать в эту ячейку содержимое ящика). При каждом поиске выполняется хотя бы один шаг вправо.

Команда С формируется, когда исходный текст уже полностью просмотрен и окно установлено на пустую ячейку между примером и программой, поэтому один поиск (один шаг **ВПРАВО**) оказывается лишним.

Ситуацию можно выправить, если перед соответствующим вызовом процедуры Записать_символ написать команду **ВЛЕВО** (на последний символ исходного примера):

```
ЭТО Транслятор
  // Запись команды П
  ОБМЕН ПИШИ П ОБМЕН
  Записать_символ
  Перенести_число // Первое число
  // Запись команды П
  ОБМЕН ПИШИ П ОБМЕН
  Записать_символ
  ВПРАВО // Перешагнуть через знак +
  Перенести_число // Второе число
  // Запись команды С
  ОБМЕН ПИШИ С ОБМЕН
  ВЛЕВО  // ← Эта команда обеспечит запись команды С вплотную
  Записать_символ
КОНЕЦ
```

Но более правильным будет не включать в процедуру Транслятор «заплатку» **ВЛЕВО**, а изменить процедуру Записать_символ так, чтобы при поиске конца исходного примера шаг вправо выполнялся только на непустой ячейке:

```
ЭТО Записать_символ
  ЕСЛИ НЕ ПУСТО
```

```
    ТО { ВПРАВО Записать_символ }
    ИНАЧЕ Записать
  ВЛЕВО
КОНЕЦ

ЭТО Записать
  ВПРАВО
  ПОКА НЕ ПУСТО ВПРАВО
  ЯЩИК-
  ПОКА НЕ ПУСТО ВЛЕВО
  ВПРАВО
КОНЕЦ
```

2. Перевести пример на сложение нескольких чисел в программу для Плюсика. Например, запись 25 + 7 + 20 должна быть преобразована в текст П25П7СП20С. В начальный момент в окошко виден первый символ записи примера.

Решение

Алгоритм

1. Записать команду П и первое число, смещаясь вправо по символам исходной записи.
2. Пока не закончатся знаки + в исходной записи, делать:
   2.1. Записать команду П, число, следующее за знаком +, и команду С.

Программа

```
ЭТО Транслятор
  Запись_Пчисло
  ПОКА + Запись_ПчислоС
КОНЕЦ

ЭТО Запись_Пчисло
  // Запись команды П
  ОБМЕН ПИШИ П ОБМЕН
  Записать_символ
  // Запись цифр числа
  ПОКА ЦИФРА
  {
    ЯЩИК+
```

```
    Записать_символ
    ВПРАВО
  }
КОНЕЦ

ЭТО Запись_ПчислоС
  ВПРАВО // Пропустим знак +
  Запись_Пчисло
  // Запись команды С
  ОБМЕН ПИШИ С ОБМЕН
  Записать_символ
КОНЕЦ
```

Замечание

Процедура `Записать_символ` описана в решении предыдущей задачи.

Тесты

Набор тестов приводится в табл. 14.4.

Таблица 14.4

Тест	Комментарий	Ожидаемый результат
1 123	Запись состоит из одного числа	1□П1 123□П123
1+2 123+456	Сложение двух чисел	1+2□П1П2С 123+456□П123П456С
1+2+12+123	Сложение нескольких чисел	1+2+12+123□П1П2СП12СП123С

3. Арифметический пример содержит только целые неотрицательные числа, соединённые знаками «+» и «−». Написать программу для преобразования примера в программу для Плюсика. Например, запись 100 − 20 + 67 должна быть преобразована в программу `П100П20ВП67С`. В начальный момент в окошко виден первый символ записи примера.

Решение

Решение этой задачи легко получить из решения предыдущей заменой цикла **ПОКА** + в главной процедуре на рекурсивный цикл с двумя проверками (на знак + и на знак −). Соответственно придётся написать ещё одну простую процедуру `Запись_ПчислоВ`.

Программа

```
ЭТО Транслятор
  Запись_Пчисло
  Транслятор1
КОНЕЦ

ЭТО Транслятор1
  ЕСЛИ       - ТО { Запись_ПчислоВ Транслятор1 }
  ИНАЧЕ ЕСЛИ + ТО { Запись_ПчислоС Транслятор1 }
КОНЕЦ

ЭТО Запись_ПчислоВ
  ВПРАВО // Пропустим знак +
  Запись_Пчисло
  // Запись команды В
  ОБМЕН ПИШИ В ОБМЕН
  Записать_символ
КОНЕЦ
```

Замечание

Процедуры `Запись_Пчисло`, `Запись_ПчислоС` и `Записать_символ` описаны в решении предыдущей задачи.

Тесты

Набор тестов приводится в табл. 14.5.

Таблица 14.5

Тест	Комментарий	Ожидаемый результат
1 123	Запись состоит из одного числа	1□П1 123□П123
1+2 123+456 1-2 123-456	Два числа в записи	1+2□П1П2С 123+456□П123П456С 1-2□П1П2В 123-456□П123П456В
1+2-12+123 12-2+12+123 1+234-12-1 122-22-12+123	Несколько чисел в записи	1+2-12+123□П1П2СП12ВП123С 12-2+12+123□П12П2ВП12СП123С 1+234-12-1□П1П234СП12ВП1В 122-22-12+123□П122П22ВП12ВП123С

4. Язык программирования некоторого исполнителя содержит такую конструкцию цикла **ПОВТОРИ**:

   `ЧислоКоманда`

   Здесь:

   - `Число` — целое неотрицательное число, меньшее числа символов в алфавите Корректора.
   - — символ **ПУСТО**.
   - `Команда` — кодируется одним символом.

   Эта запись означает, что выполнение команды нужно повторить заданное число раз.

   Написать программу, которая переводит команду цикла в последовательность команд, повторённых нужное количество раз.

   В начальный момент окно установлено на первый символ записи.

   Пример начального и конечного состояний среды показан на рис. 14.7.

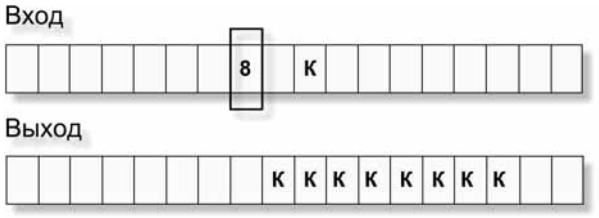

Рис. 14.7

Решение

Алгоритм. Будем рекурсивно отнимать от числа на ленте по единице, пока число не станет нулём, а рекурсивная пружинка запишет на ленту нужное количество команд. Чтобы не портить исходную запись, будем работать с копией числа.

Программа

```
ЭТО Вход
  // Сделать копию числа справа от команды
  Скопировать_число
  // Скопировать команду в ящик
  ВПРАВО ЯЩИК+
  // Сместиться на младшую цифру копии числа
  ВПРАВО ВПРАВО
```

```
  ПОКА НЕ ПУСТО ВПРАВО
  ВЛЕВО
  Записать_результат
  // Удалить одну лишнюю команду
  ПИШИ ПУСТО
КОНЕЦ

// Процедура Скопировать_число копирует число на ленту справа от
// команды, через пустую ячейку от неё. После копирования окно
// расположено на пустой ячейке между числом и командой
// в исходной записи.
ЭТО Скопировать_число
  ПОКА ЦИФРА { ЯЩИК+ Копировать_символ ВПРАВО }
КОНЕЦ

ЭТО Копировать_символ
  ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО Копировать_символ
    ИНАЧЕ
    {
      ПОВТОРИ 3 ВПРАВО
      ПОКА НЕ ПУСТО ВПРАВО
      ЯЩИК-
      ПОКА НЕ ПУСТО ВЛЕВО
      ВЛЕВО ВЛЕВО
    }
  ВЛЕВО
КОНЕЦ

// Будем рекурсивно отнимать от числа на ленте, пока
// оно не станет 0, а рекурсивная пружинка столько же
// раз повторит запись команды на ленту
ЭТО Записать_результат
  ЕСЛИ 0 ТО
  {
    ВЛЕВО
```

```
ЕСЛИ ПУСТО
  ТО
  {
    ВПРАВО ВПРАВО
    ЕСЛИ ПУСТО
      ТО
      {
        // Число на ленте равно нулю.
        // Удалим этот нуль и установим окно
        // справа от команды.
        // С этого места начнет работать рекурсивная
        // пружинка, записывающая последовательность
        // команд на ленту.
        ВЛЕВО
        ПИШИ ПУСТО
        ПОКА ПУСТО ВЛЕВО
        ВПРАВО
      }
      ИНАЧЕ
      {
        // На младшую цифру числа
        ВЛЕВО
        Минус1
        // На младшую цифру числа
        ПОКА ЦИФРА ВПРАВО
        ВЛЕВО
        Записать_результат
      }
  }
  ИНАЧЕ
  {
    // На младшую цифру числа
    ВПРАВО
    ПОКА ЦИФРА ВПРАВО
    ВЛЕВО
    Минус1
    // На младшую цифру числа
```

```
        ПОКА ЦИФРА ВПРАВО
        ВЛЕВО
        Записать_результат
      }
  }
  ИНАЧЕ
  {
    Минус1
    // На младшую цифру числа
    ПОКА ЦИФРА ВПРАВО
    ВЛЕВО
    Записать_результат
  }
  Повторение_команды // Рекурсивная пружинка
КОНЕЦ

ЭТО Повторение_команды
  ВПРАВО ЯЩИК-
КОНЕЦ
```

Замечание

Процедура `Минус 1` описана в *главе 12*.

Тесты

Набор тестов приводится в табл. 14.6.

Таблица 14.6

Тест	Комментарий	Ожидаемый результат
0□К	Ноль повторений	0□К
1□К	Одно повторение	1□К□К
5□К	Повторение задаётся одной цифрой	5□К□КККК
10□К	Десять повторений	10□К□КККККККК
12□К	Много повторений	12□К□КККККККККК

5. Построить на базе Корректора нового исполнителя со следующей СКИ (табл. 14.7).

Таблица 14.7

Команда новой СКИ	Что означает в старой СКИ
П	ВПРАВО
Л	ВЛЕВО
+	ПЛЮС
-	МИНУС
Зs	ПИШИ s
Я+	ЯЩИК+
Я-	ЯЩИК-
О	ОБМЕН

Информация на ленте Корректора имеет следующую структуру (рис. 14.8).

Рис. 14.8

Команды нового исполнителя, записанные друг за другом без всяких разделителей, образуют программу.

Рабочая область занимает 3 клетки, их содержимое в начальный момент произвольно. Рабочую область можно использовать следующим образом:

- первая ячейка — как временное хранилище символов;
- вторая ячейка — для моделирования ящика нового исполнителя;
- третья ячейка — для хранения положения окна нового исполнителя (0 — окно на первом символе данных, 1 — окно на втором символе данных и так далее).

Данные, подлежащие обработке программой (написанной для нового исполнителя), расположены за рабочей областью (предполагается, что среди данных нет символа @).

Считается, что окно нового исполнителя перед работой всегда устанавливается на первую ячейку данных.

В начальный момент Корректор «смотрит» на первую команду программы нового исполнителя.

Исполняя программу, Корректор, в соответствии с командами, обрабатывает данные. Обработка заканчивается, когда выполнена последняя команда.

Написать программы для нового исполнителя, решающие следующие задачи:

а) Записать на ленту слово КОТ.

б) Перевернуть трёхсимвольное слово в области данных «задом наперёд».

в) В области данных записано трёхзначное число, каждая цифра которого больше единицы. Отнять от этого числа 111.

г) В области данных записано трёхзначное число, каждая цифра которого не больше восьми. Прибавить к этому числу 111.

д) Придумать другие задачи для нового исполнителя и написать для него программы, решающие эти задачи.

Решение

Описание алгоритма. Построим главную процедуру нового исполнителя по следующему алгоритму:

1. Записать в третью ячейку рабочей области число 0, установить тем самым окно нового исполнителя на первый символ его данных.
2. Выполнить в цикле все команды программы, записанной на ленте для нового исполнителя.
3. Установить окно Корректора на символ данных нового исполнителя, в соответствии с положением его окна после выполнения программы.

Каждую команду нового исполнителя оформим в виде отдельной процедуры Корректора (со вспомогательными процедурами). Для возврата к текущему месту программы, записанной на ленте, будем использовать рекурсивную пружинку.

Для примера разберём более подробно реализацию команды П:

```
//-------------------------------------
ЭТО Команда_П
  ВПРАВО
  ЕСЛИ НЕ @ ТО Команда_П
  ИНАЧЕ        Окно_плюс_1
  ВЛЕВО // На текущее место в программе
КОНЕЦ
```

```
ЭТО Окно_плюс_1
  // Увеличить значение положения окна на 1
  ПОВТОРИ 3 ВПРАВО
  ПЛЮС
  ПОВТОРИ 3 ВЛЕВО
КОНЕЦ
//-------------------------------------
```

Процедура Команда_П рекурсивно перемещает окно Корректора до начала рабочей области (первый символ @), выполняет процедуру Окно_плюс_1, а затем рекурсивной пружинкой **ВЛЕВО** возвращает окно Корректора на исходное место в программу нового исполнителя.

Процедура Окно_плюс_1 смещает окно Корректора на третью ячейку рабочей области, в которой записано положение окна нового исполнителя, увеличивает значение символьного числа на 1 и возвращает окно Корректора на левую границу рабочей области (символ @), откуда рекурсивная пружина в процедуре Команда_П сместит окно на текущее место в программу нового исполнителя.

Программа

```
ЭТО Исполнитель
  Обнулить_положение_окна
  ПОКА НЕ @
  {
    ЕСЛИ       П ТО Команда_П
    ИНАЧЕ ЕСЛИ Л ТО Команда_Л
    ИНАЧЕ ЕСЛИ + ТО Команда_+
    ИНАЧЕ ЕСЛИ - ТО Команда_-
    ИНАЧЕ ЕСЛИ З ТО Команда_З
    ИНАЧЕ ЕСЛИ Я ТО Команда_Я
    ИНАЧЕ ЕСЛИ О ТО Команда_О
    ВПРАВО
  }
  Установить_окно
КОНЕЦ
//-------------------------------------
ЭТО Обнулить_положение_окна
  ПОКА НЕ @ ВПРАВО
  ПОВТОРИ 3 ВПРАВО
  ПИШИ 0
```

```
  ПОВТОРИ 3 ВЛЕВО
  ПОКА НЕ ПУСТО ВЛЕВО
  ВПРАВО
КОНЕЦ
//-------------------------------------
ЭТО Установить_окно
  ПОВТОРИ 3 ВПРАВО
  ЯЩИК+
  ВПРАВО ВПРАВО
  ОБМЕН
  ПОКА НЕ 0
  {
    МИНУС
    ОБМЕН
    ВПРАВО
    ОБМЕН
  }
  ОБМЕН
КОНЕЦ
//-------------------------------------
ЭТО Команда_П
  ВПРАВО
  ЕСЛИ НЕ @ ТО Команда_П
  ИНАЧЕ        Окно_плюс_1
  ВЛЕВО // На текущее место в программе
КОНЕЦ

ЭТО Окно_плюс_1
  // Увеличить значение положения окна на 1
  ПОВТОРИ 3 ВПРАВО
  ПЛЮС
  ПОВТОРИ 3 ВЛЕВО
КОНЕЦ
//-------------------------------------
ЭТО Команда_Л
  ВПРАВО
  ЕСЛИ НЕ @ ТО Команда_Л
```

```
  ИНАЧЕ         Окно_минус_1
  ВЛЕВО // На текущее место в программе
КОНЕЦ

ЭТО Окно_минус_1
  // Уменьшить значение положения окна на 1
  ПОВТОРИ 3 ВПРАВО
  МИНУС
  ПОВТОРИ 3 ВЛЕВО
КОНЕЦ
//-------------------------------------
ЭТО Команда_+
  ВПРАВО
  ЕСЛИ НЕ @ ТО Команда_+
  ИНАЧЕ         Плюс1
  ВЛЕВО // На текущее место в программе
КОНЕЦ

ЭТО Плюс1
  Установить_окно
  // Заменим символ следующим по алфавиту
  ПЛЮС
  // Вернёмся на конец программы (под рекурсивную пружинку)
  ПОКА НЕ @ ВЛЕВО
  ВЛЕВО ПОКА НЕ @ ВЛЕВО
КОНЕЦ
//-------------------------------------
ЭТО Команда_-
  ВПРАВО
  ЕСЛИ НЕ @ ТО Команда_-
  ИНАЧЕ         Минус1
  ВЛЕВО // На текущее место в программе
КОНЕЦ

ЭТО Минус1
  Установить_окно
  // Заменим символ предыдущим по алфавиту
```

```
  МИНУС
  // Вернёмся на конец программы (под рекурсивную пружинку)
  ПОКА НЕ @ ВЛЕВО
  ВЛЕВО ПОКА НЕ @ ВЛЕВО
КОНЕЦ
//-------------------------------------
ЭТО Команда_3
  // Запомним символ, который надо записать на ленту
  ВПРАВО ЯЩИК+
  Пиши1
КОНЕЦ

ЭТО Пиши1
  ВПРАВО
  ЕСЛИ НЕ @ ТО Пиши1
  ИНАЧЕ        Пиши2
  ВЛЕВО // На текущее место в программе
КОНЕЦ

ЭТО Пиши2
  // Запомним символ в первой ячейке рабочей области
  ВПРАВО ЯЩИК-
  // Запомним в ящике положение окна
  ВПРАВО ВПРАВО
  ЯЩИК+
  // Сместимся на соответствующее место в данных
  ВПРАВО ВПРАВО
  ОБМЕН
  ПОКА НЕ 0
  {
    МИНУС
    ОБМЕН
    ВПРАВО
    ОБМЕН
  }
  ОБМЕН
  // Сходить за символом, сохранённым в рабочей области
```

```
  Пиши3
  // Записать символ на ленту
  ЯЩИК-
  // Вернёмся на конец программы (под рекурсивную пружинку)
  ПОКА НЕ @ ВЛЕВО
  ВЛЕВО ПОКА НЕ @ ВЛЕВО
КОНЕЦ

ЭТО Пиши3
  ВЛЕВО
  ЕСЛИ НЕ @
    ТО Пиши3
    ИНАЧЕ
    {
      ПОВТОРИ 3 ВЛЕВО
      ЯЩИК+
      ПОВТОРИ 3 ВПРАВО
    }
  ВПРАВО // Возврат на место записи символа
КОНЕЦ
//-------------------------------------
ЭТО Команда_Я
  ВПРАВО
  ЕСЛИ + ТО Команда_Я+
  ИНАЧЕ     Команда_Я-
КОНЕЦ

ЭТО Команда_Я+
  ВПРАВО
  ЕСЛИ НЕ @ ТО Команда_Я+
  ИНАЧЕ        Ящик1+
  ВЛЕВО // На текущее место в программе
КОНЕЦ

ЭТО Ящик1+
  // Запомним в ящике положение окна
  ПОВТОРИ 3 ВПРАВО
```

```
  ЯЩИК+
  // Сместимся на соответствующее место в данных
  ВПРАВО ВПРАВО
  ОБМЕН
  ПОКА НЕ 0
  {
    МИНУС
    ОБМЕН
    ВПРАВО
    ОБМЕН
  }
  ОБМЕН
  // Запомним символ в ящике
  ЯЩИК+
  // Запишем этот символ во вторую ячейку рабочей области
  ПОКА НЕ @ ВЛЕВО
  ВЛЕВО ВЛЕВО ЯЩИК-
  // Вернёмся на конец программы (под рекурсивную пружинку)
  ВЛЕВО ПОКА НЕ @ ВЛЕВО
КОНЕЦ

ЭТО Команда_Я-
  ВПРАВО
  ЕСЛИ НЕ @ ТО Команда_Я-
  ИНАЧЕ        Ящик1-
  ВЛЕВО // На текущее место в программе
КОНЕЦ

ЭТО Ящик1-
  // Запомним в ящике положение окна
  ПОВТОРИ 3 ВПРАВО
  ЯЩИК+
  // Сместимся на соответствующее место в данных
  ВПРАВО ВПРАВО
  ОБМЕН
  ПОКА НЕ 0
  {
```

```
      МИНУС
      ОБМЕН
      ВПРАВО
      ОБМЕН
    }
    ОБМЕН
    // Запишем в ячейку содержимое второй ячейки рабочей
    // области (содержимое ящика нового исполнителя)
    Ящик2-
    ЯЩИК-
    // Вернёмся на конец программы (под рекурсивную пружинку)
    ПОКА НЕ @ ВЛЕВО
    ВЛЕВО ПОКА НЕ @ ВЛЕВО
КОНЕЦ

ЭТО Ящик2-
    ВЛЕВО
    ЕСЛИ НЕ @
      ТО Ящик2-
      ИНАЧЕ
      {
        ПОВТОРИ 2 ВЛЕВО
        ЯЩИК+
        ПОВТОРИ 2 ВПРАВО
      }
    ВПРАВО // Возврат на место записи символа
КОНЕЦ
//-------------------------------------
ЭТО Команда_О
    ВПРАВО
    ЕСЛИ НЕ @ ТО Команда_О
    ИНАЧЕ        Обмен1
    ВЛЕВО // На текущее место в программе
КОНЕЦ

ЭТО Обмен1
```

```
  // Запомним в ящике положение окна
  ПОВТОРИ 3 ВПРАВО
  ЯЩИК+
  // Сместимся на соответствующее место в данных
  ВПРАВО ВПРАВО
  ОБМЕН
  ПОКА НЕ 0
  {
    МИНУС
    ОБМЕН
    ВПРАВО
    ОБМЕН
  }
  ОБМЕН
  // Запомним символ в ящике
  ЯЩИК+
  // Обменяем этот символ с символом во второй ячейке рабочей области
  Обмен2
  ЯЩИК-
  // Вернёмся на конец программы (под рекурсивную пружинку)
  ПОКА НЕ @ ВЛЕВО
  ВЛЕВО ПОКА НЕ @ ВЛЕВО
КОНЕЦ

ЭТО Обмен2
  ВЛЕВО
  ЕСЛИ НЕ @
    ТО Обмен2
    ИНАЧЕ
    {
      ПОВТОРИ 2 ВЛЕВО
      ОБМЕН
      ПОВТОРИ 2 ВПРАВО
    }
  ВПРАВО // Возврат на место записи символа
КОНЕЦ
```

Тесты

Сначала нужно отдельно протестировать работу каждой команды нового исполнителя, а только потом приступать к тестированию, в котором используется смешанный набор команд нового исполнителя.

В табл. 14.8 приводятся тесты, соответствующие решению задач, описанных в условии, и решение одной новой задачи.

Таблица 14.8

Тест	Задача для нового исполнителя	Ожидаемый результат
ЗКПЗОПЗТ@□□□@	Записать на ленту слово КОТ	ЗКПЗОПЗТП@□□2@КОТ
Я+ППОЛЛО@□□□@КОТ	Перевернуть трёхсимвольное слово в области данных «задом наперёд»	Я+ППОЛЛО@□□0@ТОК
-П-П-@□□□@349	В области данных записано трёхзначное число, каждая цифра которого больше единицы. Отнять от этого числа число 111	-П-П-@□□2@238
+П+П+@□□□@108	В области данных записано трёхзначное число, каждая цифра которого не больше восьми. Прибавить к этому числу число 111	+П+П+@□□2@219
ППЯ+ЛЯ-ПЗ□ПЗ□ЛЛ@□□□@ОКНО	В области данных записано 4-символьное слово. Удалить из него второй и последний символ, а оставшиеся символы уплотнить	ППЯ+ЛЯ-ПЗ□ПЗ□@□□□@ОН

Глава 15

Решения задач

В тестах, рекомендованных для проверки программ, пробел обозначен знаком ■, пустая ячейка — знаком □, а подчёркнутый символ показывает положение окна на ленте.

Задачи недели 2002/2003 учебного года

Задачи, рекомендуемые к главам 11 и 12

1. Точка (В. П. Семенко, Рубцовск).

   Текст на ленте может содержать не более одного символа «.» (точка). Переставить (или поставить) точку в конец записи. В начальный момент окно установлено на первый символ текста. Примеры начального и конечного положений среды показаны на рис. 15.1.

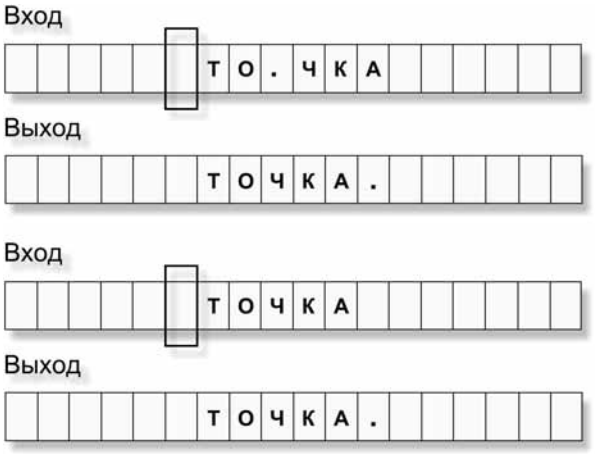

Рис. 15.1

Решение

Описание алгоритма

1. Перемещаем окно по записи до символа «.» или до символа **ПУСТО**, если точки в записи нет. В последнем случае на место **ПУСТО** записываем точку.
2. Сдвигаем остаток записи влево на один символ (затирая символ «.»).
3. Записываем точку в конце текста.

Программа

```
ЭТО Точка
  Поиск_точки
  Сдвиг_остатка
  Запись_точки
КОНЕЦ

ЭТО Поиск_точки
  ПОКА НЕ .
  {
    ВПРАВО
    ЕСЛИ ПУСТО ТО ПИШИ .
  }
КОНЕЦ

ЭТО Сдвиг_остатка
  ПОКА НЕ ПУСТО
  {
    ВПРАВО ЯЩИК+ ВЛЕВО ЯЩИК-
    ВПРАВО
  }
КОНЕЦ

ЭТО Запись_точки
  ВЛЕВО ПИШИ .
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 15.1.

Таблица 15.1

Тест	Комментарий	Ожидаемый результат
□	Пустой текст	.
□.	В тексте один символ — точка	.
□A	В тексте один символ — не точка	A.
□МОЛОКО	В длинном тексте нет точки	МОЛОКО.
□.4578	Длинный текст начинается с точки	4578.
□КОРОВА.	Длинный текст заканчивается точкой	КОРОВА.
□КОТ.□И□ЛИСА	Точка внутри длинного текста	КОТ□И□ЛИСА.

2. Повтор (В. П. Семенко, Рубцовск).

Текст на ленте повторяется дважды. Окно на первом символе записи. Убрать с ленты повторный вариант текста. Пример начального и конечного положений среды показан на рис. 15.2.

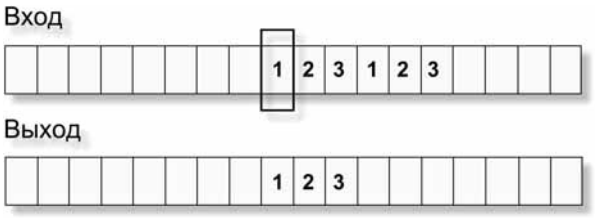

Рис. 15.2

Решение

Описание алгоритма. Рекурсивно проходим всю запись, двигая окно вправо на две ячейки на каждом шаге. Рекурсивной пружинкой (отложенными командами) стираем правую половину.

Программа

```
// Автор решения: А. В. Рудь
ЭТО Вход
  ПОВТОРИ 2 ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО  Вход      // Рекурсивный проход
    ИНАЧЕ ВЛЕВО  // Окно на последний символ записи
  // Рекурсивная пружинка для стирания правой половины записи
  ПИШИ ПУСТО ВЛЕВО
КОНЕЦ
```

Замечание

На каждом шаге рекурсивного цикла окно сдвигается на *две* ячейки вправо, «откладывая» стирание *одной* ячейки. Таким образом, после завершения рекурсии «пружинка» сработает столько раз, сколько пар символов было в записи, т. е. будет стёрта половина исходной записи.

Тесты

Набор тестов приводится в табл. 15.2.

Таблица 15.2

Тест	Комментарий	Ожидаемый результат
□	Пустая запись	□
11	Повторение одного символа	1
1212 123123	Повторение нескольких символов	12 123

3. Неисправная клавиатура (В. П. Семенко, Рубцовск).

Окно установлено на первый символ записи. В результате неисправности клавиатуры каждый символ записи удвоился. Удалить повторы символов. Примеры начального и конечного положений среды показаны на рис. 15.3.

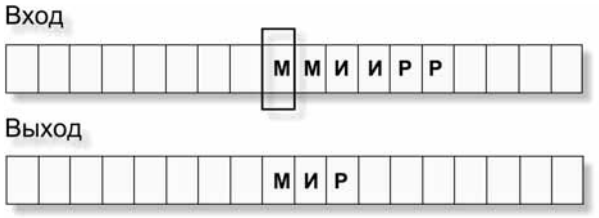

Рис. 15.3

Решение

Описание алгоритма. На каждом шаге просмотра будем смещать окно на две ячейки вправо и сдвигать остаток записи влево на одну ячейку.

Алгоритм

1. Пока запись не закончится, делать:
   1.1. Сместить окно на две ячейки вправо.
   1.2. Сдвинуть остаток записи с текущего символа влево.

Программа

```
ЭТО Вход
  ПОКА НЕ ПУСТО
  {
    ПОВТОРИ 2 ВПРАВО
    Сдвиг
  }
КОНЕЦ

ЭТО Сдвиг
  // Перенос символа влево
  ЯЩИК+ ВЛЕВО ЯЩИК- ВПРАВО
  // Шаг по записи вправо
  ВПРАВО

  ЕСЛИ НЕ ПУСТО
    ТО    Сдвиг // Рекурсивный цикл
    ИНАЧЕ { ВЛЕВО ПИШИ ПУСТО } // Удаляем последний символ
  // Пружинка: возврат в исходное место записи
  // для продолжения основного цикла в процедуре Вход
  ВЛЕВО
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 15.3.

Таблица 15.3

Тест	Комментарий	Ожидаемый результат
□	Пустая запись	□
11	Запись из одного повторённого символа	1
1122 112233	Запись из нескольких повторённых символов	12 123

Задачи, рекомендуемые к главам 12 и 13

4. Полусумма (В. П. Семенко, Рубцовск).

   На ленте записан непустой плотный ряд цифр. Окно исполнителя находится слева перед записью. Найти полусумму цифр ряда. Примеры начального и конечного положений среды показаны на рис. 15.4.

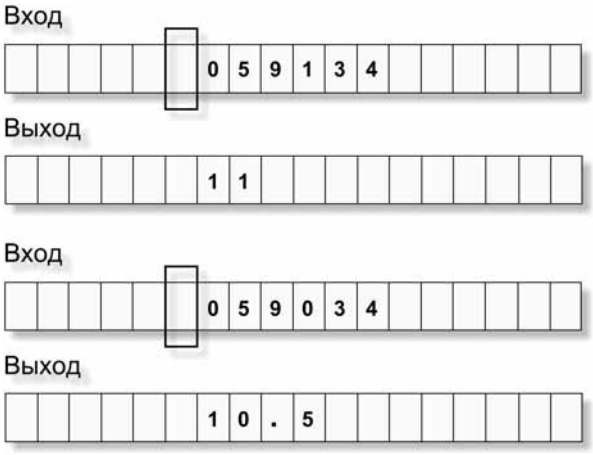

Рис. 15.4

Решение

Алгоритм 1

Для лучшего понимания алгоритма его пункты иллюстрированы текущим содержимым ленты и ящика при обработке исходной записи `4216`.

Исходное состояние ленты: `□4216`.

1. Нахождение суммы. Перемещаем окно вдоль записи, накапливая в ящике сумму цифр в виде символьного числа и стирая исходную запись.

   Лента: `□`.

   Ящик: `г` (символьное число, равнозначное числу 13).

2. Подготовка к нахождению полусуммы. Записываем на ленту 0 (значение полусуммы), а справа содержимое ящика (значение суммы).

   Лента: `0г`.

3. Нахождение полусуммы. Пока значение суммы больше единицы, делаем:

   3.1. От значения суммы отнимаем 2, а к значению полусуммы добавляем 1.

   Лента: `61`.

4. Запись дробной части результата. Проверим значение суммы.
   4.1. Если значение суммы равно 0, записываем в ячейку с суммой **ПУСТО**.
   4.2. Если значение суммы равно 1, записываем в ячейку с суммой точку, а справа символ 5 (то есть записываем десятичную дробь «.5»).

   Лента: `6.5`.
5. Запись целой части результата. Преобразуем значение полусуммы из символьного числа в обычное число.

   Лента: `6.5`.

Программа 1

```
ЭТО Вход
  Нахождение_суммы
  Подготовка_полусуммы
  Нахождение_полусуммы
  Запись_дробной_части
  Запись_целой_части
КОНЕЦ

ЭТО Нахождение_суммы
  // Подготовка счётчика
  ПИШИ 0 ОБМЕН
  // Окно на первую цифру
  ВПРАВО
  // Суммирование всех цифр записи
  Сумма
КОНЕЦ

ЭТО Сумма
  // Просмотр всех цифр записи
  ПОКА НЕ ПУСТО
  {
    // Увеличение ящика на значение цифры в окне.
    // Значение ящика увеличиваем на 1, а значение цифры на ленте
    // уменьшаем на 1, пока цифра не станет нулём.
    ПОКА НЕ 0 { МИНУС ОБМЕН ПЛЮС ОБМЕН }
    // Вместо цифры (которая стала нулём) записываем
```

```
    // ПУСТО и продвигаем окно к следующей ячейке.
    ПИШИ ПУСТО ВПРАВО
  }
КОНЕЦ

// Подготовка к нахождению полусуммы.
// Записываем на ленту 0 (значение полусуммы),
// а справа содержимое ящика (значение суммы).
// ---------------------------------------
ЭТО Подготовка_полусуммы
  ПИШИ 0 ВПРАВО ОБМЕН
КОНЕЦ

// Нахождение полусуммы. Пока значение суммы больше единицы, от
// значения суммы отнимаем 2, а к значению полусуммы добавляем 1.
// -------------------------------------------------------------
ЭТО Нахождение_полусуммы
 ЕСЛИ НЕ 0
   ТО
   {
     ЕСЛИ НЕ 1
       ТО
       {
         МИНУС МИНУС
         ВЛЕВО
         ПЛЮС
         ВПРАВО
         Нахождение_полусуммы
       }
   }
КОНЕЦ

ЭТО Запись_дробной_части
  ЕСЛИ 0
    ТО ПИШИ ПУСТО
    ИНАЧЕ { ПИШИ . ВПРАВО ПИШИ 5 ВЛЕВО }
  // Установка окна на символьное число - значение
```

```
  // целой части полусуммы.
  ВЛЕВО
КОНЕЦ

ЭТО Запись_целой_части
  Символ_в_число // Процедура, описанная в главе 12
КОНЕЦ
```

Замечание

Программа, написанная по алгоритму 1, способна получить результат, если только значение суммы лежит в диапазоне символьных чисел Корректора, а значение полусуммы, соответственно, в два раза меньше.

Следующий алгоритм позволяет расширить область допустимых значений и работать для таких исходных чисел, полусумма (а не сумма) цифр которых лежит в диапазоне символьных чисел Корректора.

Алгоритм 2 (идея Алексея Носова, Рубцовск)

Для лучшего понимания алгоритма его пункты иллюстрированы текущим содержимым ленты и ящика при обработке исходной записи `9367`.

Исходное состояние ленты: `□9367`.

1. Нахождение полусуммы за один просмотр записи. Рекурсивно перемещаем окно вдоль записи и накапливаем в ящике не сумму исходных цифр, а сумму их половинок (в качестве половинки нечётной цифры берётся целая часть от деления её на два). Недостающие 0.5 на каждой нечётной цифре учтём при помощи специально предусмотренной рекурсивной пружины (процедура `Половинка`).

   Лента: `Б3` (`Б` — сумма половинок цифр, `3` — число неучтённых в сумме 0.5).

2. Добавить к полученной сумме неучтённые в ней 0.5. Пока число половинок больше единицы, делаем:

   2.1. От числа половинок отнимаем 2, а к значению суммы добавляем 1.

   Лента: `В1` (`В` — целая часть суммы, `1` — признак наличия в сумме дробной части 0.5).

3. Запись дробной части результата. Проверим значение числа оставшихся половинок.

   3.1. Если 0, записываем в ячейку `ПУСТО`.

   3.2. Если 1, записываем в ячейку точку, а справа символ 5 (то есть записываем десятичную дробь «.5»).

   Лента: `В.5`.

4. Запись целой части результата. Преобразуем значение суммы из символьного числа в обычное число.

   Лента: `12.5`.

Программа 2

```
ЭТО Вход
  // Подготовка ящика-сумматора
  ПИШИ 0 ОБМЕН
                          // Пример на ленте: 9367
  Найти_полусумму         // ->               Б3
  Добавить_половинки      // ->               В1
  Запись_дробной_части    // ->               В.5
  Запись_целой_части      // ->              12.5
КОНЕЦ

// Рекурсивное нахождение полусуммы
// Цепочка косвенной рекурсии:
// Найти_полусумму -> Сложение -> Найти_полусумму
// Описание. Каждая цифра заменяется целой частью от
// деления её на 2, и это значение добавляется к ящику (в
// процедуре Сложение).
// Для нечётных цифр дополнительно предусмотрена рекурсивная
// пружинка (процедура Половинка).
// Перед работой рекурсивной пружины работает процедура
// Подготовка. Эта процедура записывает на ленту содержимое
// ящика (полусумма без половинок) и следом число 0, устанавливая
// на него окно.
// ---------------------------------------------------------------
ЭТО Найти_полусумму
  ВПРАВО
  ЕСЛИ       1 ТО { ПИШИ 0 Сложение Половинка }
  ИНАЧЕ ЕСЛИ 2 ТО { ПИШИ 1 Сложение }
  ИНАЧЕ ЕСЛИ 3 ТО { ПИШИ 1 Сложение Половинка }
  ИНАЧЕ ЕСЛИ 4 ТО { ПИШИ 2 Сложение }
  ИНАЧЕ ЕСЛИ 5 ТО { ПИШИ 2 Сложение Половинка }
  ИНАЧЕ ЕСЛИ 6 ТО { ПИШИ 3 Сложение }
  ИНАЧЕ ЕСЛИ 7 ТО { ПИШИ 3 Сложение Половинка }
  ИНАЧЕ ЕСЛИ 8 ТО { ПИШИ 4 Сложение }
```

```
  ИНАЧЕ ЕСЛИ 9 ТО { ПИШИ 4 Сложение Половинка }
  ИНАЧЕ Подготовка
КОНЕЦ

ЭТО Сложение
  // Добавляем значение цифры с ленты к ящику
  // (ячейку уменьшаем, а ящик увеличиваем)
  ПОКА НЕ 0 { МИНУС ОБМЕН ПЛЮС ОБМЕН }
  // Вместо цифры (которая стала нулём) записываем ПУСТО
  ПИШИ ПУСТО
  // Замыкаем рекурсивный цикл
  Найти_полусумму
КОНЕЦ

// Записываем на ленту 0, к которому будем затем добавлять
// рекурсивной пружинкой количество неучтённых половинок.
// -------------------------------------------------------
ЭТО Подготовка
  ОБМЕН ВПРАВО ПИШИ 0
КОНЕЦ

ЭТО Половинка
  ПЛЮС
КОНЕЦ

// Добавление к полученной сумме неучтённые в ней 0.5.
// Пока число половинок больше единицы, от числа половинок
// отнимаем 2, а к значению суммы добавляем 1.
// -------------------------------------------------------
ЭТО Добавить_половинки
  ЕСЛИ НЕ 0
    ТО
    {
      ЕСЛИ НЕ 1
        ТО
        {
          МИНУС МИНУС ВЛЕВО ПЛЮС ВПРАВО
```

```
            Добавить_половинки
          }
      }
КОНЕЦ

ЭТО Запись_дробной_части
  ЕСЛИ 0
    ТО ПИШИ ПУСТО
    ИНАЧЕ { ПИШИ . ВПРАВО ПИШИ 5 ВЛЕВО }
  // Установка окна на символьное число -
  // значение целой части суммы.
  ВЛЕВО
КОНЕЦ

ЭТО Запись_целой_части
  Символ_в_число // Процедура, описанная в главе 12
КОНЕЦ
```

Замечание

Можно написать решение без рекурсивной пружинки, но оно требует двух проходов по записи. На первом проходе (слева направо) цифры на ленте заменяются «половинками», а в ящике накапливается сумма поправок 0.5 для нечётных цифр. При втором проходе (справа налево) к ящику добавляются половинные цифры, записанные на ленту при первом проходе.

Тесты

Набор тестов приводится в табл. 15.4.

Таблица 15.4

Тест	Комментарий	Ожидаемый результат
□	Пустая запись	0
□0	Одна цифра	0
□6		3
□9		4.5
□1234567890	Несколько цифр	22.5
□4646464646		25
□46323…868	Предельное значение для полусуммы	69.5

5. Голосование символов (В. П. Семенко, Рубцовск).

   Для надёжности сообщение было передано по линии связи трижды, все три текста были приняты Корректором и записаны на ленту через символ **ПУСТО**. Восстановить исходное сообщение, если известно, что каждый его символ был верно принят не менее двух раз. В начальный момент окно установлено на первый символ первой из трёх записей. Пример начального и конечного положений среды показаны на рис. 15.5.

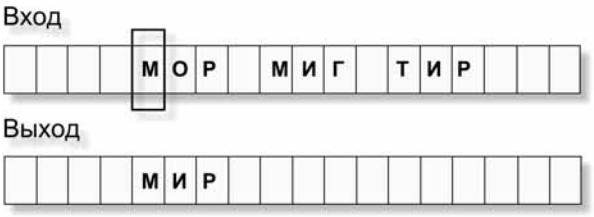

Рис. 15.5

Решение

Описание алгоритма. Будем формировать результат справа от последнего из трёх текстов, стирая по символам исходное сообщение.

Алгоритм

1. Пока символы в первом варианте текста не закончатся, делать:
   1.1. Копировать первый символ первого текста в ящик и стирать его с ленты.
   1.2. Сравнить содержимое ящика с первым символом второго текста.
      1.2.1. Если символы совпали, удалить первые символы второго и третьего текста и дописать символ из ящика справа к результату.
      1.2.2. Если символы не совпали, записать в ящик первый символ третьего текста, удалить с ленты первые символы второго и третьего текста и дописать символ из ящика справа к результату.
   1.3. Установить окно на первый символ первого текста.
2. Сместить окно на первый символ результата.

Программа

```
ЭТО Вход
  ПОКА НЕ ПУСТО Работа
  На_первый_символ_результата
```

```
КОНЕЦ

ЭТО Работа
  // Занести первый символ первого текста в ящик
  ЯЩИК+
  // Удалить первый символ первого текста
  ПИШИ ПУСТО
  // Установить окно на первый символ второго текста
  К_следующему_тексту
  // Сравнить содержимое ящика с первым символом второго текста.
  ЕСЛИ Я=Л
    ТО
    {
      ПИШИ ПУСТО           // Удалить 1-ый символ 2-го текста
      К_следующему_тексту // Окно на 1-ый символ 3-го текста
    }
    ИНАЧЕ
    {
      ПИШИ ПУСТО           // Удалить 1-ый символ 2-го текста
      К_следующему_тексту // Окно на 1-ый символ 3-го текста
      ЯЩИК+                // В ящик 1-ый символ 3-го текста
    }
  // Удалить 1-й символ 3-го текста
  ПИШИ ПУСТО
  // Добавить содержимое ящика к результату
  За_конец_результата // Окно за конец результата
  ЯЩИК-                // Добавить символ к результату
  // Установить окно на 1-й символ первого текста
  На_начало
КОНЕЦ

ЭТО К_следующему_тексту
  ВПРАВО
  ПОКА НЕ ПУСТО ВПРАВО
  ПОКА ПУСТО ВПРАВО
КОНЕЦ

ЭТО За_конец_результата
```

```
  ВПРАВО
  ПОКА НЕ ПУСТО ВПРАВО
  ВПРАВО
  ПОКА НЕ ПУСТО ВПРАВО
КОНЕЦ

ЭТО На_начало
    ПОКА НЕ ПУСТО ВЛЕВО
    ВЛЕВО
    ЕСЛИ НЕ ПУСТО
      ТО
      {
        ПОВТОРИ 2
        {
          ПОКА НЕ ПУСТО ВЛЕВО
          ПОКА ПУСТО ВЛЕВО
        }
        ПОКА НЕ ПУСТО ВЛЕВО
        ВПРАВО
      }
КОНЕЦ

ЭТО На_первый_символ_результата
  ПОВТОРИ 2 ВПРАВО
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 15.5.

Таблица 15.5

Тест	Комментарий	Ожидаемый результат
□	Пустая лента	□
1□2□1	Один символ в сообщении	1
2□2□1		2
1□2□2		2
2□2□2		2
МОР□МИГ□ТИР	Несколько символов в сообщении	МИР

6. Количество перевёртышей (Артём Букирь, 8 класс, Тольятти).

На ленте записано несколько слов. Перед первым словом, за последним словом и между словами записано по одному пробелу. Отыскать в тексте слова-перевёртыши и записать их количество на ленту. Перевёртышей в тексте может быть не более девяти. Слова, состоящие из одной буквы, тоже считаются перевёртышами. Начальный текст можно не сохранять. В начальный момент окно установлено на первый пробел записи. Пример начального и конечного положений среды показан на рис. 15.6.

Рис. 15.6

Решение

Описание алгоритма. Счётчик перевёртышей поместим на ленте сразу перед записью (перед первым пробелом). Командой **плюс** будем добавлять к счётчику единицу каждый раз, когда обнаружим перевёртыш в записи. Первый раз команда **плюс** будет применена к пустой клетке и запишет в неё 0 — начальное значение счётчика.

Проверку слова на перевёртыш будем выполнять в цикле, сравнивая и затирая пробелом первый и последний его символы.

После проверки слово заменяется пробелами, так что очередное проверяемое слово всегда будет первым справа от счётчика после цепочки пробелов.

Программа

```
// Автор решения: В. П. Семенко
ЭТО Вход
  // Обнуление счётчика перевёртышей и установка окна
  // на 1-й символ 1-го слова, уцелевшего в записи.
  // Если все слова уже обработаны, окно будет установлено на
  // первую пустую ячейку за цепочкой пробелов, оставшихся от записи.
  Прибавь_1
  // Поиск перевёртышей
  ПОКА НЕ ПУСТО Поиск_перевертышей
```

```
  // Удалить пробелы, оставшиеся от записи, и установить окно на счётчик
  ВЛЕВО
  ПОКА ПРОБЕЛ { ПИШИ ПУСТО ВЛЕВО }
КОНЕЦ

ЭТО Поиск_перевертышей
  // Запомним первый символ слова и заменим его пробелом.
  ЯЩИК+ ПИШИ ПРОБЕЛ
  // На 2-ой символ слова
  ВПРАВО
  ЕСЛИ ПРОБЕЛ
    ТО Нечетный_перевертыш
    ИНАЧЕ Проверка
КОНЕЦ

ЭТО Нечетный_перевертыш
  // Заменяем пробелом единственный (оставшийся) символ слова.
  ВЛЕВО ПИШИ ПРОБЕЛ
  // Добавим единицу к счётчику и устанавливаем окно
  // на 1-й символ 1-го слова уцелевшего в записи.
  Прибавь_1
КОНЕЦ

ЭТО Проверка
  // Установим окно на последний символ слова
  На_последний_символ
  // Сравним последний символ с первым (он в ящике)
  ЕСЛИ Я=Л
    ТО Продолжение_проверки
    ИНАЧЕ Удаление_слова  // и установка окна на следующее слово
КОНЕЦ

ЭТО Продолжение_проверки
  // Стираем последний символ (он оказался равным первому)
  ПИШИ ПРОБЕЛ
  // Смещаем окно влево
  ВЛЕВО
```

```
  // Если символов слева нет - был перевёртыш
  ЕСЛИ ПРОБЕЛ
    ТО Прибавь_1  // Чётный перевёртыш
    ИНАЧЕ
    {
      // Устанавливаем окно на первый символ слова и
      // продолжаем его проверку рекурсивным циклом.
      ПОКА НЕ ПРОБЕЛ ВЛЕВО
      ВПРАВО
      Поиск_перевертышей // Замыкание рекурсивного цикла
    }
КОНЕЦ

// Удаление слова справа и установка окна
// на первый символ следующего слова
// -------------------------------------
ЭТО Удаление_слова
  ПОКА НЕ ПРОБЕЛ { ПИШИ ПРОБЕЛ ВЛЕВО }
  ПОКА ПРОБЕЛ ВПРАВО
КОНЕЦ

// Добавление единицы к счётчику и установка окна
// на 1-й символ 1-го слова, уцелевшего в записи.
// ----------------------------------------------
ЭТО Прибавь_1
  ПОКА ПРОБЕЛ ВЛЕВО
  ПЛЮС
  ВПРАВО
  ПОКА ПРОБЕЛ ВПРАВО
КОНЕЦ

ЭТО На_последний_символ
  ВПРАВО
  ПОКА НЕ ПРОБЕЛ ВПРАВО
  ВЛЕВО
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 15.6.

Таблица 15.6

Тест	Комментарий	Ожидаемый результат
□□	Пустая запись (два пробела)	0
□12□	Нет перевёртышей	0
□12□345□6789□0□		0
□1□232□шалаш□	Несколько перевёртышей	3
□1□11□111□1111□11111□		5
□1□22□345□667□5555□767□		3

Задачи недели 2003/2004 учебного года

Задачи к главам 8 и 9

7. Радиус окружности (В. П. Семенко, Рубцовск).

   На плоскости нарисовано несколько окружностей с центрами в начале координат. Для каждой окружности заданы координаты одной из точек её пересечения с координатными осями. Найти радиус большей окружности.

   Формат исходных данных: $(x_1,y_1)(x_2,y_2)\ldots$

   Здесь: x_i, y_i — координаты точки для i-ой окружности — символьные числа, которым может предшествовать знак «–». Знаки «(», «)», «,» и «–» — особые, символьные числа не могут принимать таких значений. Символы в записи располагаются плотно, без пробелов и пустых ячеек. Окно установлено на первую пустую ячейку слева от записи.

   Результат записать в виде символьного числа справа через одну пустую ячейку от исходных данных.

   На рис. 15.7 показан пример начального и конечного состояний среды.

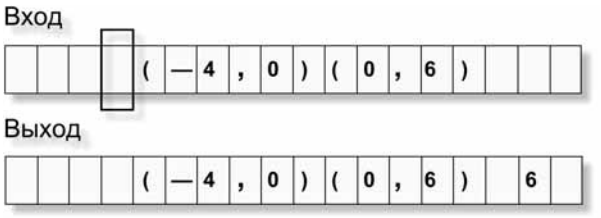

Рис. 15.7

Решение

Описание алгоритма. Среди всех чисел, задающих координаты, нужно найти наибольшее число без учёта его знака. Для хранения текущего кандидата будем использовать ящик. Занесём в него число 0 и пройдёмся вдоль записи. Всякий раз, когда на ленте обнаруживается число, большее числа в ящике, обновляем ящик этим значением.

Алгоритм

1. Записываем в ящик число 0.
2. Нахождение наибольшего символьного числа: пока запись не закончилась, делать:
   2.1. Если текущий символ — координата и она больше числа в ящике, заменяем содержимое ящика этой координатой.
   2.2. Продвигаем окно к следующему символу.
3. Записываем содержимое ящика на ленту.

Программа

```
ЭТО Вход
  Подготовка
  Поиск_максимума
  Запись_результата
КОНЕЦ

// Запишем в ящик число 0 - начальный кандидат для максимума
// -------------------------------------------------------
ЭТО Подготовка
  ПИШИ 0 ОБМЕН
КОНЕЦ

ЭТО Поиск_максимума
  ВПРАВО
```

```
  ПОКА НЕ ПУСТО
  {
    Проверить
    ВПРАВО
  }
КОНЕЦ

// Если символ не является вспомогательным и
// он больше числа в ящике, заносим его в ящик.
// ---------------------------------------------
ЭТО Проверить
  ЕСЛИ    НЕ (
  ТО ЕСЛИ НЕ )
  ТО ЕСЛИ НЕ ,
  ТО ЕСЛИ НЕ -
  ТО ЕСЛИ Я<Л
  ТО ЯЩИК+
КОНЕЦ

// Максимум содержится в ящике. Запишем его на ленту
// -------------------------------------------------
ЭТО Запись_результата
  ВПРАВО ОБМЕН
КОНЕЦ
```

Замечание 1

Обратите внимание на удобную запись вложенной команды ветвления. Каждое новое вложенное условие добавляется к прежним через логическую операцию И. То есть команда, расположенная на последней ветви ТО, срабатывает тогда и только тогда, когда выполнены все входящие в выражение условия. В нашем случае это можно условно записать так:

```
ЕСЛИ НЕ ( И НЕ ) И НЕ , И НЕ – И Я<Л ТО ЯЩИК+
```

К сожалению, в существующей версии языка условия нельзя связывать логическими операциями. Приходится их моделировать (модели логических операций описаны в *главе 11*).

Замечание 2

Программа работает и в том случае, если лента исполнителя изначально пуста, предлагая в качестве ответа разумное значение — число 0.

Тесты

Набор тестов приводится в табл. 15.7.

Таблица 15.7

Тест	Комментарий	Ожидаемый результат
□	Пустая лента	0
□(-4,0) □(0,A) □(0,0)	Одна пара координат	4 A 0
□(-7,0)(0,@) □(A,0)(-B,0)(0,7)	Несколько пар координат	@ B

8. Распаковка отрезка (В. П. Семенко, Рубцовск).

Под отрезком на ленте будем понимать последовательность (слева направо) ячеек, в первой из которых записан символ Н (начало отрезка), в последней К (конец отрезка), а промежуточные ячейки (тело отрезка) пусты. Число ячеек между символами Н и К будем называть длиной отрезка.

Отрезок можно запаковать, записав его в трёх последовательных ячейках:

HnK

Здесь: n — символьное число, задающее длину отрезка.

Отрезок называется вырожденным, если он имеет нулевую длину, т. е. Н0К — запакованный, а НК — распакованный вырожденные отрезки.

На ленте задан запакованный отрезок, окошко Корректора находится на символе, задающем его длину. Переписать на ленту этот отрезок в распакованном виде.

На рис. 15.8 показан пример начального и конечного состояний среды.

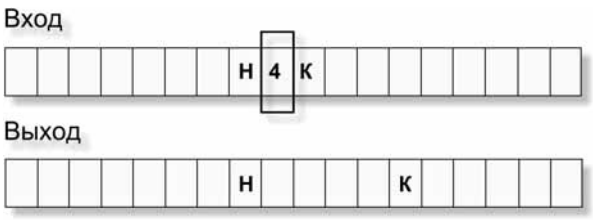

Рис. 15.8

Решение

Алгоритм

1. Проверим длину отрезка:
   1.1. Если длина равна 0, то отрезок вырожденный: обработаем его отдельно.
   1.2. Если длина не равна 0, выполним распаковку.

Алгоритм процедуры `Распаковка`

1. Пока длина отрезка не равна нулю, делать:
   1.1. Уменьшить длину на 1.
   1.2. Переместить окно на одну ячейку вправо.
2. Записать на ленту К.

Программа

```
// Автор решения В. П. Семенко
ЭТО Вход
  // Проверим длину отрезка
  ЕСЛИ 0
    ТО    Вырожденный_отрезок
    ИНАЧЕ Распаковка
КОНЕЦ

ЭТО Вырожденный_отрезок
  // Рядом с Н запишем К, а исходное К удалим
  ПИШИ К ВПРАВО ПИШИ ПУСТО
КОНЕЦ

ЭТО Распаковка
  // Двигаемся по ленте вправо, уменьшая длину отрезка,
  // пока длина не станет равной нулю
  ПОКА НЕ 0
  {
    // Уменьшаем длину на 1 и сохраняем её в ящике
    МИНУС ЯЩИК+
    // Очищаем ячейку
    ПИШИ ПУСТО
    // Сдвигаем окно вправо и записываем на ленту длину отрезка
    ВПРАВО ЯЩИК-
```

```
  }
  // Запись конца отрезка
  ПИШИ К
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 15.8.

Таблица 15.8

Тест	Комментарий	Ожидаемый результат
Н0К	Вырожденный отрезок	НК
Н1К	Длина отрезка равна 1	Н□К
Н4К	Длина отрезка больше 1	Н□□□□К

9. Пятерки (Владимир Югатов, Владимир Кротенко, 8 класс, Рубцовск).

   В роботландской школе Кукарача получал только оценки 5, которые Корректор записывал на свою ленту. Однажды, когда друзья пили чай, Злоумышленник испортил запись, вписав в неё единицы, двойки, тройки и четвёрки.

   Удалите лишние оценки и уплотните запись. Окошко Корректора находится перед первым символом записи.

   На рис. 15.9 показан пример начального и конечного состояний среды.

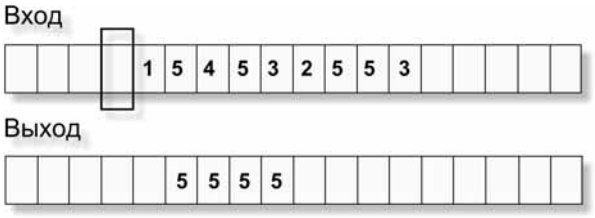

Рис. 15.9

Решение

Алгоритм 1

1. Удалим все лишние символы в начале записи до первого символа 5 (или до конца записи, если в ней нет пятерок).
2. Пропустим все пятёрки, плотно расположенные за первой.

3. Перемещаем окно по остатку записи, стирая символы. Если среди символов попадается символ 5, дописываем его к цепочке пятёрок, начатой в пункте 1.

Программа 1

```
ЭТО Вход
  Удаление_первых_лишних
  // Найден символ 5 или конец записи.
  // Если найден символ 5, продолжаем работу.
  ЕСЛИ 5
    ТО
    {
      // Пропускаем плотную цепочку пятёрок
      ПОКА 5 ВПРАВО
      // Просмотр (и стирание) остатка записи.
      // Найденные символы 5 добавляем к цепочке пятёрок.
      ПОКА НЕ ПУСТО
      {
        ЕСЛИ 5
          ТО    { Добавь_5 ПИШИ ПУСТО }
          ИНАЧЕ { ПИШИ ПУСТО ВПРАВО }
        // Стирание обработанного символа и продвижение окна к
        // следующему символу записи выполняется отдельно в каждой
        // ветви условной команды.
        // На ветви ТО продвижение по записи выполняется в процедуре
        // Добавь_5, за счёт чего экономятся по две команды на каждую
        // пятёрку в записи.
      }
    }
КОНЕЦ

// Добавь_5: рекурсивный поиск существующей цепочки пятёрок слева,
// дописывание новой пятёрки к цепочке и возврат в исходную запись.
// Замечание. Перед работой отложенной команды ВПРАВО окно умышленно
// не возвращается влево "под рекурсивную пружину". То есть рекурсивная
// пружина установит окно на одну ячейку правее его текущего положения
// в исходной записи. Это позволяет на каждой пятёрке записи экономить
// выполнение двух команд: ВЛЕВО - установка "под пружину", ВПРАВО -
```

```
// продвижение к следующему символу записи.
// ---------------------------------------------------------------------
ЭТО Добавь_5
  ВЛЕВО
  ЕСЛИ НЕ 5
    ТО    Добавь_5
    ИНАЧЕ { ВПРАВО ПИШИ 5 }
  ВПРАВО
КОНЕЦ

ЭТО Удаление_первых_лишних
  ВПРАВО
  ПОКА НЕ ПУСТО
  {
    ЕСЛИ 5
     ТО    ВЛЕВО // Прерывание цикла
     ИНАЧЕ { ПИШИ ПУСТО ВПРАВО }
  }
  ВПРАВО
КОНЕЦ
```

Алгоритм 2

Можно пройтись по записи, стирая оценки и считая пятёрки. Затем записать нужное количество символов 5 на пустую ленту. Программа будет работать только в том случае, если количество пятёрок не превышает число символов в алфавите Корректора.

Программа 2

```
// Автор решения: Владимир Югатов, 8 класс, Рубцовск
ЭТО Вход
  // Обнуляем ящик - счетчик пятёрок
  ПИШИ 0 ОБМЕН
  // Просмотр записи
  ВПРАВО
  ПОКА НЕ ПУСТО
  {
    ЕСЛИ 5 ТО Считаем_пятёрки
    ПИШИ ПУСТО // Стираем символ
    ВПРАВО     // Окно к следующему символу записи
```

```
  }
  // Записываем на ленту нужное количество символов 5
  Пишем_пятерки
КОНЕЦ

// Увеличиваем количество пятёрок на 1
// -----------------------------------
ЭТО Считаем_пятерки
  ОБМЕН ПЛЮС ОБМЕН
КОНЕЦ

ЭТО Пишем_пятерки
  ОБМЕН
  ПОКА НЕ 0
  {
    // Уменьшаем счётчик на 1
    МИНУС ОБМЕН
    // Записываем в ячейку 5
    ПИШИ 5
    // Продвигаем окно
    ВПРАВО ОБМЕН
  }
  // Стираем 0
  ПИШИ ПУСТО
КОНЕЦ
```

Алгоритм 3

Ещё интереснее выглядит программа с использованием рекурсивной пружинки. В рекурсивном цикле все символы записи стираются, а для символа 5 дополнительно предусматривается отложенная команда (вызов процедуры `Запись5`), которая и запишет на ленту нужное число пятёрок.

Программа 3

```
// Автор решения: В. П. Семенко
ЭТО Вход
  ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО
      ЕСЛИ 5
```

```
        ТО    { ПИШИ ПУСТО Вход Запись5 }
        ИНАЧЕ { ПИШИ ПУСТО Вход }
КОНЕЦ

ЭТО Запись 5
  ПИШИ 5 ВПРАВО
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 15.9.

Таблица 15.9

Тест	Комментарий	Ожидаемый результат
□	Лента пуста	□
□5	Запись состоит из одной пятёрки	5
□2	Запись состоит из одной не пятёрки	□
□3443497	В длинной записи нет пятёрок	□
□5555555	В длинной записи есть пятёрки	5555555
□56467554		555

10. Распиливание бревна (Артём Букирь, 9 класс, Тольятти).

На ленте, справа от окна, расположено «бревно» — цепочка символов «*». Число звёздочек в бревне называется длиной бревна. «Распилить» бревно на брёвнышки, длина которых задаётся символьным числом в ячейке слева от окна (символ «0» означает, что пилить не надо). Брёвна должны отделяться друг от друга одной пустой ячейкой. Длина последнего брёвнышка может быть меньше остальных.

На рис. 15.10 показан пример начального и конечного состояний среды.

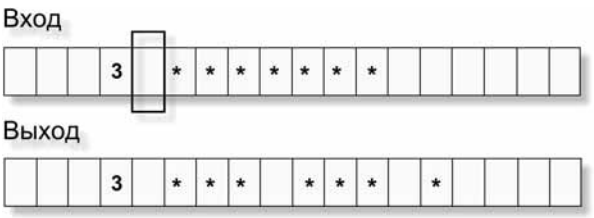

Рис. 15.10

Решение

Алгоритм

1. Если в ячейке слева не 0 (бревно пилить нужно), делаем:
   1.1. Запоминаем в ящике длину отпиливаемых брёвнышек.
   1.2. Перемещаемся на начало бревна.
   1.3. Если бревно нужно пилить, то пилим (процедура `Работа`).

Алгоритм процедуры `Работа`

1. Пока бревно не закончилось, делаем:
   1.1. Отпиливаем брёвнышко (процедура `Распил`).
   1.2. Заносим в ящик длину брёвнышка для следующего распила.
   1.3. Смещаем окно на остаток бревна.
2. Бревно распилено, стираем лишний символ — счётчик длины распила.

Описание алгоритма процедуры `Распил`

Пока бревно не закончилось и брёвнышко не отпилено, рекурсивно продвигаем окно вдоль бревна. На месте распила записываем **ПУСТО**, в конец бревна дописываем «*» и отложенной командой восстанавливаем исходную длину брёвнышка для продолжения работы.

Программа

```
// Автор решения: В. П. Семенко
ЭТО Вход
  ВЛЕВО
  // Проверяем, нужно ли пилить бревно
  ЕСЛИ НЕ 0
    ТО
    {
      // Запоминаем длину брёвнышка
      ЯЩИК+
      ПОВТОРИ 2 ВПРАВО
      // Проверяем, есть ли бревно
      ЕСЛИ НЕ ПУСТО ТО Работа
    }
КОНЕЦ

ЭТО Работа
  // Пока бревно не закончилось
  ПОКА НЕ ПУСТО
```

```
  {
    // Отпиливаем брёвнышко
    Распил
    // Записываем в ящик длину брёвнышка для следующего распила.
    ОБМЕН
    // На следующий символ
    ВПРАВО
  }
  // Стираем лишний символ
  ВЛЕВО ПИШИ ПУСТО
КОНЕЦ

ЭТО Распил
  ОБМЕН
  // Отмеряем длину брёвнышка
  МИНУС
  // Проверяем, не закончилось ли брёвнышко.
  // Брёвнышко закончилось, когда окно располагается за ним,
  // поэтому проверка не на 0, а на ПУСТО.
  ЕСЛИ НЕ ПУСТО
    ТО
    {
      ОБМЕН ВПРАВО
      // Проверяем, не закончилось ли бревно
      ЕСЛИ НЕ ПУСТО ТО Распил // Рекурсивный цикл
    }
    ИНАЧЕ
    {
      // "Отодвигаем" отпиленную часть
      // (дописываем в конец символ *; он расположен в ящике)
      ВПРАВО
      ПОКА НЕ ПУСТО ВПРАВО
      ОБМЕН  // На ленту - *, в ящик - ПУСТО
      // Возвращаем окно на место последнего распила
      ПОКА НЕ ПУСТО ВЛЕВО
    }
  // Рекурсивная пружинка: восстанавливаем длину брёвнышка
  ПЛЮС
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 15.10.

Таблица 15.10

Тест	Комментарий	Ожидаемый результат
0□***	Длина брёвнышка равна нулю — пилить не нужно	***
2□	Бревна нет	□
4□***	Длина брёвнышка больше длины бревна	***
1□***** 2□***** 3□******	Длина брёвнышка меньше длины бревна (рассмотреть отдельно случаи, когда длина бревна кратна длине брёвнышка и когда нет)	*□*□*□*□* **□**□* ***□***
3□***	Длина брёвнышка равна длине бревна	***

11. ЧУ-ЩУ (Михаил Суворов, 8 класс, Тверь).

Заменить в тексте на ленте слоги ЧЮ и ЩЮ на ЧУ и ЩУ. В начальный момент окно установлено на первую пустую ячейку перед текстом.

На рис. 15.11 показан пример начального и конечного состояний среды.

Рис. 15.11

Решение

Описание алгоритма

Поочередно проверяем все символы текста. Если встретили букву Ч или Щ, то проверяем, какая буква стоит справа: если это буква Ю, то вместо неё пишем У.

Программа

```
// Автор решения: В. П. Семенко
ЭТО Вход
```

```
  // Шаг на первый символ текста
  ВПРАВО
  // Пока текст не закончился, выполняем проверку.
  // Шаг к следующему символу расположен в процедуре Проверка.
  ПОКА НЕ ПУСТО Проверка
КОНЕЦ

ЭТО Проверка
  // Если Ч или Щ, то проверяем, какая буква стоит справа
  ЕСЛИ        Ч ТО Что_справа
  ИНАЧЕ  ЕСЛИ Щ ТО Что_справа
  ИНАЧЕ            ВПРАВО // Шаг к следующему символу
КОНЕЦ

ЭТО Что_справа
  ВПРАВО                  // Шаг к следующему символу
  ЕСЛИ Ю ТО ПИШИ У
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 15.11.

Таблица 15.11

Тест	Комментарий	Ожидаемый результат
□	Пустая запись	□
□ЧУЩУ	В тексте нет ошибок	ЧУЩУ
□ЧЮЩЮ	В тексте есть ошибки, но нет сдвоенных согласных ЧЧ, ЩЩ	ЧУЩУ
□ЧЧУЩЩЮ	В тексте есть ошибки после сдвоенных согласных ЧЧ, ЩЩ	ЧЧУЩЩУ

12. Подсчитаем брёвнышки (Дмитрий Исаев, 10 класс, Лукоянов).

На ленте Корректора расположены несколько брёвен в виде плотных рядов из символов «*». Брёвна отделяются друг от друга символом ПРОБЕЛ (одним или более). Подсчитайте, сколько брёвен находится на ленте. В начальный момент окно установлено на первый символ записи.

На рис. 15.12 показан пример начального и конечного состояний среды.

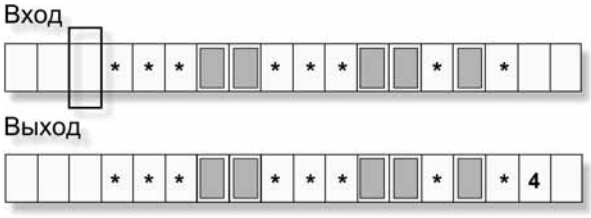

Рис. 15.12

Решение

Алгоритм

1. Обнуляем счётчик (ящик).
2. Пока не пусто, т. е. запись не закончилась, считаем брёвна:
   2.1. Доходим до конца текущего бревна.
   2.2. Увеличиваем счётчик на 1.
   2.3. Переходим к следующему бревну.
3. Записываем ответ на ленту.

Программа

```
// Автор решения: В. П. Семенко
ЭТО Вход
  // Обнуляем счётчик (ящик)
  ОБМЕН ПИШИ 0 ОБМЕН
  // Пока запись не закончилась, считаем брёвна
  ПОКА НЕ ПУСТО
  {
    // Доходим до конца бревна и увеличиваем счетчик на 1
    Подсчет
    // Переходим к следующему бревну
    ПОКА ПРОБЕЛ ВПРАВО
  }
  // Записываем ответ на ленту
  ОБМЕН
КОНЕЦ

ЭТО Подсчет
  // Доходим до конца бревна
  ПОКА * ВПРАВО
```

```
    // Увеличиваем счётчик на 1
    ОБМЕН ПЛЮС ОБМЕН
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 15.12.

Таблица 15.12

Тест	Комментарий	Ожидаемый результат
□	Брёвен на ленте нет	0
*	Одно бревно из одной звёздочки	1
****	Одно бревно произвольной длины	1
□***	Два бревна произвольной длины, отделённые одним пробелом	2
□□□□***	Два бревна произвольной длины, отделённые более, чем одним пробелом	2
****□□***□**□□□*	Больше двух брёвен с разным числом пробелов между ними	4

Задачи к главе 10

13. Запаковка отрезка (В. П. Семенко, Рубцовск).

На ленте записаны два символа: Н — начало отрезка и К — конец отрезка. Конец отрезка правее его начала. Запаковать отрезок, т. е. записать его в трёх последовательных ячейках в виде HnK, где n — длина отрезка, символьное число, равное количеству пустых ячеек между символами Н и К в исходном отрезке. В начальный момент окно находится на символе Н. После работы программы в нём видно символьное число — длина запакованного отрезка.

На рис. 15.13 показан пример начального и конечного состояний среды.

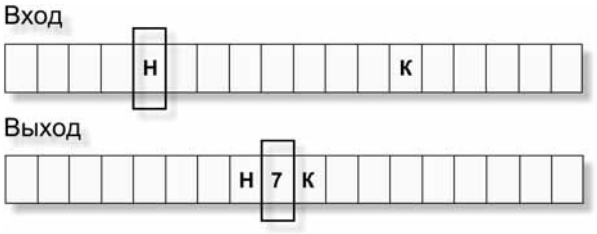

Рис. 15.13

Решение

Алгоритм

1. Стираем начало отрезка символ Н.
2. Рекурсивно идём по ленте, пока не обнаружим конец отрезка символ К. Рекурсивной пружинкой записываем слева от К длину отрезка.
3. Слева от ячейки с длиной записываем символ Н (начало отрезка).
4. Устанавливаем окно на ячейку с длиной отрезка.

Программа

```
// Автор решения Иван Безрядин, 9 кл., Снежинск
ЭТО Вход
  // 1. Стираем Н
  ПИШИ ПУСТО
  // 2. Рекурсивно считаем длину отрезка (ищем К) и записываем
  //    длину в пустую ячейку перед К
  Подсчет_длины
  // 3. Записываем на ленте слева от длины символ Н
  ВЛЕВО ПИШИ Н
  // 4. Устанавливаем окно на длину отрезка
  ВПРАВО
КОНЕЦ

ЭТО Подсчет_длины
  ВПРАВО
  ЕСЛИ НЕ К
    ТО    Подсчет_длины // Рекурсивный цикл
    ИНАЧЕ ВЛЕВО // Найден символ К: окно - на ячейку, в которой
                // должна быть записана длина.
  // Рекурсивная пружинка: записывает в пустую ячейку символьное
  // число - длину отрезка.
  ПЛЮС
КОНЕЦ
```

Замечание

Рекурсивная пружинка ПЛЮС добавит столько единиц к ячейке, сколько было сделано шагов вправо в рекурсивном цикле. То есть на одну единицу больше, чем надо, ведь последний шаг был сделан в ячейку с символом К. Но так как первоначально в ячейке записан не 0, а символ ПУСТО (его номер в алфавите на единицу меньше номера символа 0), то ответ будет получен правильный.

Тесты

Набор тестов приводится в табл. 15.13.

Таблица 15.13

Тест	Комментарий	Ожидаемый результат
НК	Вырожденный отрезок	Н0К
Н□К	Длина отрезка равна 1	Н1К
Н□□□□К	Длина отрезка больше 1	Н4К

14. Друзья (В. П. Семенко, Рубцовск).

Назовём символы друзьями, если они в алфавите стоят рядом, например А и Б, Ю и Я. Друзьями также будем считать символы 0 и @.

Известно, что среди символов записи на ленте есть только одна пара друзей, причём каждый друг встречается в записи ровно один раз. Найти эти символы и поменять их местами. В начальный момент в окно виден первый символ записи.

На рис. 15.14 показан пример начального и конечного состояний среды.

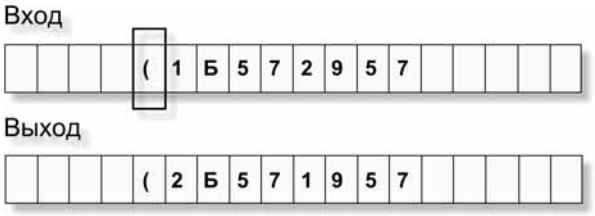

Рис. 15.14

Решение

Алгоритм

1. Пока запись не закончилась, делать:
   1.1. Заносим символ в ящик, а на его место — `ПУСТО`.
   1.2. В остатке записи ищем друга к символу в ящике. Если друг найден, переставляем друзей местами и смещаем окно на пустую клетку слева от записи, чтобы прервать цикл. В противном случае возвращаем окно на текущий символ записи и восстанавливаем его.
   1.3. Смещаем окошко на следующий символ записи.

Программа

```
// Автор решения В. П. Семенко
ЭТО Вход
  // Просмотр записи
  ПОКА НЕ ПУСТО
  {
    // Текущий символ - в ящик, а на его место - ПУСТО
    ОБМЕН
    Поиск_друга
    ОБМЕН
    // На следующий символ записи
    ВПРАВО
  }
КОНЕЦ

ЭТО Поиск_друга
  // На следующий символ записи
  ВПРАВО
  ПОКА НЕ ПУСТО
  {
    // Проверяем, являются ли символы в окошке и в ящике друзьями
    Проверка
    // Если в процедуре Проверка друг найден, то на его
    // место записывается товарищ из ящика, сам он - на
    // место товарища на ленте, а окно устанавливается
    // на пустую клетку перед записью, чтобы прекратить
    // работу (прервать выполнение циклов).
    ЕСЛИ НЕ ПУСТО ТО ВПРАВО // Друг не найден, продолжаем поиск
  }
  // Друг в остатке записи не найден, возвращаем окно на её начало
  ВЛЕВО
  ПОКА НЕ ПУСТО ВЛЕВО
КОНЕЦ

ЭТО Проверка
  ЕСЛИ       0 ТО Особый_случай_0
  ИНАЧЕ ЕСЛИ @ ТО Особый_случай_@
```

```
  ИНАЧЕ         Друг_слева?
КОНЕЦ

ЭТО Особый_случай_0
  ОБМЕН
  ЕСЛИ        @ ТО Поменять_местами
  ИНАЧЕ ЕСЛИ 1 ТО Поменять_местами
  ИНАЧЕ            ОБМЕН
КОНЕЦ

ЭТО Особый_случай_@
  ОБМЕН
  ЕСЛИ        0 ТО Поменять_местами
  ИНАЧЕ ЕСЛИ ~ ТО Поменять_местами
  ИНАЧЕ            ОБМЕН
КОНЕЦ

ЭТО Друг_слева?
  МИНУС
  // Символ в ящике такой же, как в окошке?
  ЕСЛИ Я=Л
    ТО  // Друг слева найден!
    {
      // Восстанавливаем символ на ленте
      ПЛЮС
      // Меняем местами содержимое ящика и окна, первый из друзей
      // теперь на месте второго, а второй - в ящике).
      ОБМЕН
      // Найденного друга ставим на место товарища на ленте
      Поменять_местами
    }
    ИНАЧЕ
    {
      // Восстанавливаем ленту и проверяем на друга справа
      ПЛЮС
      Друг_справа?
    }
```

```
КОНЕЦ

ЭТО Друг_справа?
  ПЛЮС
  // Символ в ящике такой же, как в окошке?
  ЕСЛИ Я=Л
    ТО // Друг справа найден
    {
      // Восстанавливаем символ на ленте
      МИНУС
      // Меняем местами содержимое ящика и окна (первый из друзей
      // теперь на месте второго, а второй - в ящике).
      ОБМЕН
      // Найденного друга ставим на место товарища на ленте
      Поменять_местами
    }
    ИНАЧЕ МИНУС // Восстанавливаем ленту
КОНЕЦ

// Перед началом работы процедуры:
// первый из друзей на месте второго, а второй - в ящике.
ЭТО Поменять_местами
  // Перемещаем окно на место первого из друзей
  ПОКА НЕ ПУСТО ВЛЕВО
  // Записываем в эту ячейку второго из друзей
  ОБМЕН
  // Так как друзей больше нет и задача решена, то искусственно прерываем
  // работу циклов ПОКА НЕ ПУСТО, сместив окно на первую пустую
  // ячейку слева от записи.
  ПОКА НЕ ПУСТО ВЛЕВО
КОНЕЦ
```

Комментарий. В процедуре `Поменять_местами` последняя команда **ПОКА НЕ ПУСТО ВЛЕВО** устанавливает окно на первую пустую ячейку слева от записи. Эта команда искусственно прерывает работу циклов **ПОКА НЕ ПУСТО** в процедурах `Вход` и `Поиск_друга`.

Тесты

Набор тестов приводится в табл. 15.14.

Таблица 15.14

Тест	Комментарий	Ожидаемый результат
□	Лента пуста	□
A	Запись состоит из одного символа	A
19	Запись состоит из двух символов, но это не друзья	19
12	Запись состоит из двух символов-друзей	21
A0@Я @A0Я	Особые друзья	A@0Я 0A@Я
1AAAAA0	Друзья начинают и заканчивают запись	0AAAAA1
AA12AAAAA	Друзья-соседи	AA21AAAAA
A4AAAAA3AA	Друзья внутри, но не рядом	A3AAAAA4AA

15. Перестановка. (Идея Сергея Соболева, Андрея Яковлева, 8 класс, ст. Новохопёрск, Воронежской обл.; редакция В. П. Семенко).

На ленте расположены три записи, отделённые друг от друга одной пустой ячейкой. Поменять первую и третью запись местами, сохраняя порядок символов в каждой записи. В начальный момент в окно виден первый символ первой записи.

На рис. 15.15 показан пример начального и конечного состояний среды.

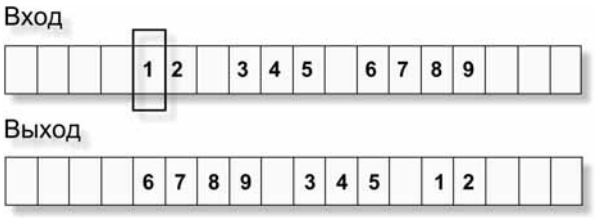

Рис. 15.15

Решение

Алгоритм

Пусть на ленте три записи: 12□345□5678.

1. Переставляем запись1 через одну пустую ячейку справа от записи3.

   Лента: 345□5678□12.

2. Переставляем запись3 через одну пустую ячейку слева от записи2.

   Лента: 5678□345□□□□□12.

3. Придвигаем запись1 к записи2 так, чтобы между ними была только одна пустая ячейка.

   Лента: 5678□345□12.

Программа

```
// Автор решения В. П. Семенко
ЭТО Вход
  // 12□345□5678
  Перестановка_123_231
  // □345□5678□12
  // На конец записи3 (она на втором месте)
  ПОКА ПУСТО ВПРАВО
  ПОКА НЕ ПУСТО ВПРАВО
  ВПРАВО
  ПОКА НЕ ПУСТО ВПРАВО
  ВЛЕВО
  // 345□5678□12
  Перестановка_231_321
  // 5678□345□□□□□12
  // На начало записи1 (она на третьем месте)
  ПОКА ПУСТО ВПРАВО
  // 5678□345□□□□□12
  Уплотнение
  // 5678□345□12□□□□□
  // На конец записи1 (она на третьем месте)
  ПОКА ПУСТО ВЛЕВО
  // 5678□345□12
КОНЕЦ

// Перестановка запись1 за запись3
ЭТО Перестановка_123_231
  ПОКА НЕ ПУСТО
  {
    ОБМЕН
    Перенос_символа_123_231
    ВПРАВО
```

```
  }
КОНЕЦ

ЭТО Перенос_символа_123_231
  // На правый конец формируемой записи
  ПОВТОРИ 3
  {
    ВПРАВО
    ПОКА НЕ ПУСТО ВПРАВО
  }
  ВПРАВО
  ПОКА НЕ ПУСТО ВПРАВО
  // Дописать очередной символ
  ОБМЕН
  // На текущее место в первой записи
  ПОВТОРИ 3
  {
    ПОКА НЕ ПУСТО ВЛЕВО
    ВЛЕВО
  }
  ПОКА НЕ ПУСТО ВЛЕВО
КОНЕЦ

ЭТО Перестановка_231_321
  ПОКА НЕ ПУСТО
  {
    ОБМЕН
    Перенос_символа_231_321
    ВЛЕВО
  }
КОНЕЦ

ЭТО Перенос_символа_231_321
  // На левый конец формируемой записи
  ПОВТОРИ 2
  {
    ВЛЕВО
```

```
    ПОКА НЕ ПУСТО ВЛЕВО
  }
  ВЛЕВО
  ПОКА НЕ ПУСТО ВЛЕВО
  // Дописать очередной символ
  ОБМЕН
  // На текущее место во второй записи
  ПОВТОРИ 2
  {
    ПОКА НЕ ПУСТО ВПРАВО
    ВПРАВО
  }
  ПОКА НЕ ПУСТО ВПРАВО
КОНЕЦ

ЭТО Уплотнение
  // Перенос первого символа
  ЯЩИК+ ВЛЕВО
  ПОКА ПУСТО ВЛЕВО
  ВПРАВО ВПРАВО
  ЯЩИК- ВПРАВО
  ПОКА ПУСТО ВПРАВО
  ПИШИ ПУСТО
  ВПРАВО
  // Перенос остальных символов
  ПОКА НЕ ПУСТО
  {
    ЯЩИК+
    ПИШИ ПУСТО
    Перенос_символа
    // Переход к следующему символу выполняет
    // процедура Перенос_символа
  }
КОНЕЦ

ЭТО Перенос_символа
  ВЛЕВО
```

```
  ЕСЛИ ПУСТО
    ТО    Перенос_символа
    ИНАЧЕ { ВПРАВО ЯЩИК- } // Дописывание символа
  // Рекурсивная пружинка:
  // возврат в запись на следующий символ
  ВПРАВО
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 15.15.

Таблица 15.15

Тест	Комментарий	Ожидаемый результат
1□2□3	Три записи по одному символу	3□2□1
12□345□6789	Три записи с произвольным количеством символов	6789□345□12

16. Левое зеркало (Константин Елечко, 9 класс, Петропавловск, Казахстан).

Как-то раз Корректор нашёл на своей ленте странную записку. Слова в ней были написаны с помощью зеркала, да не простого, а кривого. Помогите Корректору восстановить текст, если известно, что записку надо читать справа налево, а между словами расположено неизвестное количество пробелов (зеркало ведь кривое), из которых нужно оставлять по одному.

Записка начинается и заканчивается не пробелами. Начальное положение окна на первом слева символе записи, его положение после работы программы — на том символе, который станет в записи последним.

На рис. 15.16 показан пример начального и конечного состояний среды.

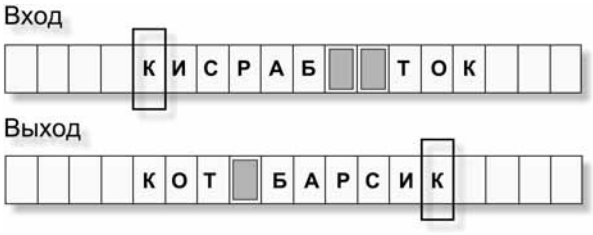

Рис. 15.16

Решение

Алгоритм

1. Если запись не пуста, то делать:
   1.1. Переместить окно на второй символ записи.
   1.2. Пока запись не закончилась, делать:
      1.2.1. Перенести символ на первое пустое место слева, а на его исходном месте записать пробел.
      1.2.2. Перейти к следующему символу записи. При этом, если перенесённый символ был пробелом, пропустить все пробелы, которые за ним следуют.
   1.3. Удалить все пробелы справа.

Программа

```
// Автор решения В. П. Семенко
ЭТО Вход
  ЕСЛИ НЕ ПУСТО ТО Работа
КОНЕЦ

ЭТО Работа
  // На второй символ записи
  ВПРАВО
  // Перенос записи, начиная со второго символа влево.
  // На месте переносимого символа записываем ПРОБЕЛ.
  ПОКА НЕ ПУСТО
  {
    // Перенос символа
    ЯЩИК+
    Перенос_символа
    // Запись пробела на текущее место записи и переход к следующему
    // символу записи. Если перенесённый символ был пробелом,
    // пропускаются все пробелы, которые за ним следуют.
    ЕСЛИ ПРОБЕЛ ТО ПОКА ПРОБЕЛ ВПРАВО
    ИНАЧЕ { ПИШИ ПРОБЕЛ ВПРАВО }
  }
  // Удалить пробелы справа от результата
  ВЛЕВО
```

```
  ПОКА ПРОБЕЛ { ПИШИ ПУСТО ВЛЕВО }
КОНЕЦ

ЭТО Перенос_символа
  ВЛЕВО
  ЕСЛИ НЕ ПУСТО
    ТО    Перенос_символа // Рекурсивный цикл
    ИНАЧЕ ЯЩИК- // Запись символа
  // Рекурсивная пружинка:
  // возврат на текущее место в исходной записи
  ВПРАВО
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 15.16.

Таблица 15.16

Тест	Комментарий	Ожидаемый результат
□	Лента пуста	□
А	Текст состоит из одного символа	А
ЯСАВ	Текст состоит из одного слова	ВАСЯ
ЯСАВ□ТОК	Текст состоит из двух слов с одним пробелом	КОТ□ВАСЯ
ЯСАВ□□□ТОК	Текст состоит из двух слов с несколькими пробелами	КОТ□ВАСЯ
ИКУК□И□□ТОК	Текст состоит из нескольких слов	КОТ□И□КУКИ

17. Дефрагментация ленты (Н. В. Табашин, Лукоянов).

    На ленте Корректора скопилось много ячеек от стёртых файлов (символы ПРОБЕЛ). Выполните дефрагментацию ленты. (Дефрагментация — это уплотнение информации на магнитном носителе с целью повышения скорости работы компьютера). После дефрагментации файлы (символы) должны составлять плотную запись без пробелов и располагаться в порядке возрастания алфавитных номеров.

    Символы на ленте могут быть любые. На рис. 15.17 показан пример начального и конечного состояний среды (можно считать, что В — это

Windows, Д — драйверы, И — игры, М — музыка, @ — почта, ! — папка с важными файлами).

В начальный момент в окно виден первый символ записи.

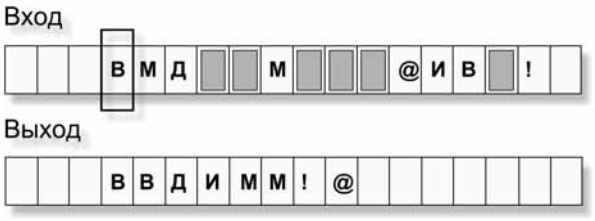

Рис. 15.17

Решение

Описание алгоритма

Сначала упорядочим символы в записи в порядке возрастания их алфавитных номеров. При этом пробелы, если они есть, станут образовывать плотную группу, разделяя запись на две части. Удалим группу пробелов, придвинув вторую часть записи к первой.

Описание алгоритма упорядочивания записи

Перемещаем окно вдоль записи, переставляя текущий элемент и минимальный элемент, найденный в остатке записи, начиная с текущего.

Программа

```
// Автор решения: В. П. Семенко
ЭТО Вход
  // Если запись не пуста
  ЕСЛИ НЕ ПУСТО
    ТО
    {
      // Упорядочим символы в записи в порядке возрастания их
      // алфавитных номеров. При этом пробелы, если они есть, станут
      // образовывать плотную группу, разделяя запись на две части.
      Сортировка
      // Удалим группу пробелов (если они есть), придвинув
      // вторую часть записи к первой.
      Пробелы
    }
КОНЕЦ
```

```
ЭТО Сортировка
  ПОКА НЕ ПУСТО
  {
    // Запомним текущий символ записи
    ЯЩИК+
    // Рекурсивно найдём минимальный элемент в остатке записи
    // справа, начиная с текущего.
    // Найденный минимальный элемент запомним в ящике.
    // Рекурсивной пружинкой вернём окно на место текущего элемента.
    Поиск_минимального
    // Минимальный элемент запишем на место текущего
    ЯЩИК-
    // Передвинем окно к следующему элементу записи
    ВПРАВО
  }
КОНЕЦ

// Рекурсивный поиск минимального элемента в остатке записи
// справа (начиная с текущего) и возврат окна в текущее
// место рекурсивной пружинкой.
// Описание алгоритма поиска.
// Идём вдоль записи с текущим элементом в ящике.
// Каждый раз, когда на ленте оказывается элемент, меньший
// элемента в ящике, меняем содержимое ленты и ящика местами.
// В конце просмотра в ящике окажется минимальный элемент записи,
// и все элементы на ленте будут сохранены (возможно, с изменением
// их положения в записи).
// -------------------------------------------------------
ЭТО Поиск_минимального
  ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО
    {
      // Если элемент на ленте меньше элемента в ящике,
      // выполняем обмен ленты и ящика.
      ЕСЛИ Я>Л ТО ОБМЕН
     Поиск_минимального
```

```
    }
   // Рекурсивная пружинка: возврат в исходное место записи
   ВЛЕВО
КОНЕЦ

// Запись на ленте упорядочена.
// Окно установлено на первый пустой символ за записью.
// Запись имеет один из следующих видов:
// Часть1ПробелыЧасть2  (пробелы в середине)
// Часть1Пробелы        (пробелы в конце)
// Запись               (нет пробелов)
// Пробелы              (одни пробелы в записи)
// -------------------------------------------------
ЭТО Пробелы
  // Очищаем ящик
  ЯЩИК+
  ВЛЕВО // На последний символ записи
  ЕСЛИ ПРОБЕЛ
    ТО Удаление_пробелов
    ИНАЧЕ Поиск_пробелов
КОНЕЦ

// Запись имеет один из следующих видов:
// Часть1Пробелы        (пробелы в конце)
// Пробелы              (одни пробелы в записи)
// Удалим пробелы
ЭТО Удаление_пробелов
  ПОКА ПРОБЕЛ
  {
    ПИШИ ПУСТО
    ВЛЕВО
  }
КОНЕЦ

// Запись имеет один из следующих видов:
// Часть1ПробелыЧасть2  (пробелы в середине)
// Запись               (нет пробелов)
//-----------------------------------------
```

```
ЭТО Поиск_пробелов
  ЕСЛИ    НЕ ПРОБЕЛ
  ТО ЕСЛИ НЕ ПУСТО
  ТО { ВЛЕВО Поиск_пробелов }
  ИНАЧЕ Сдвиг
  ИНАЧЕ Сдвиг
КОНЕЦ

// Найден: либо пробел, либо окно установилось
// на первый пустой символ перед записью.
// Если найден символ ПУСТО, то работа закончена
// (в записи пробелов нет), иначе выполняем сдвиг записи
// и удаление пробелов.
//-----------------------------------------------------
ЭТО Сдвиг
  ЕСЛИ НЕ ПУСТО
    ТО
    {
      // Запись имеет вид:
      // Часть1ПробелыЧасть2  (пробелы в середине)
      // Придвигаем Часть1 к Части2, заменяя её пробелами,
      // затем удаляем лишние конечные пробелы.
      ВПРАВО
      ПОКА НЕ ПУСТО
      {
        ЯЩИК+
        ПИШИ ПРОБЕЛ
        ПОКА ПРОБЕЛ ВЛЕВО
        ВПРАВО
        ЯЩИК-
        ВПРАВО
        ПОКА ПРОБЕЛ ВПРАВО
        // Переход к следующему символу выполняется в последнем цикле.
      }
      ВЛЕВО
      Удаление_пробелов
    }
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 15.17.

Таблица 15.17

Тест	Комментарий	Ожидаемый результат
□	Лента пуста	□
□□□□□□□□	Запись состоит из одних пробелов	□
A □	Запись состоит из одного символа	A □
98765554321 1234567889 2915826734	В записи нет пробелов	12345556789 1234567889 1223456789
9158□26734 □9□□158□2673□4□	В записи есть пробелы, а все остальные символы меньше пробела	123456789
□□+□*□□-/□*<□>=□	В записи есть пробелы, а все остальные символы больше пробела	-+/*=<>
□531□□+□*□□-/□*=□	В записи есть пробелы и символы, большие и меньшие пробела	135-+/*=

Задачи к главе 11

18. Симметричный отрезок (В. П. Семенко, Рубцовск).

    В окошке Корректора видна буква О — центр симметрии. Где-то слева от неё распакованный отрезок (см. определение отрезка в задаче 8). Постройте распакованный отрезок, симметричный исходному относительно центра О.

    На рис. 15.18 показан пример начального и конечного состояний среды.

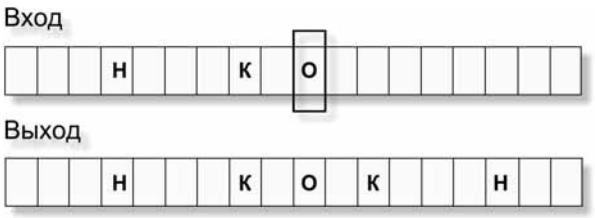

Рис. 15.18

Решение

Описание алгоритма

Задача решается при помощи пружинок двух рекурсивных процедур, вложенных одна в другую (процедура Длина_КО вложена в процедуру Длина_НК).

Алгоритм

1. Перемещаем окно на начало исходного отрезка.
2. Рекурсивно измеряем длину исходного отрезка НК (процедура Длина_НК).
3. Рекурсивно измеряем расстояние КО (процедура Длина_КО).
4. Откладываем КО (рекурсивной пружинкой в процедуре Длина_КО).
5. Пишем Н (перед работой рекурсивной пружинки в процедуре Длина_КО).
6. Откладываем НК (рекурсивной пружинкой в процедуре Длина_НК).
7. Пишем К.

Программа

```
// Автор решения: Владимир Кротенко, 8 кл., Рубцовск
ЭТО Вход
  На_начало_исходного_отрезка
  Построение_симметричного_отрезка
КОНЕЦ

ЭТО На_начало_исходного_отрезка
  ПОКА НЕ Н ВЛЕВО
КОНЕЦ

ЭТО Построение_симметричного_отрезка
  // 1) Измеряем длину исходного отрезка НК
  // 2) Измеряем расстояние КО
  // 3) Откладываем КО
  // 4) Пишем Н
  // 5) Откладываем НК
  Длина_НК
  // 6) Пишем К
  ПИШИ К
КОНЕЦ
```

```
ЭТО Длина_НК
  // 1) Рекурсивно измеряем длину исходного отрезка НК
  ВПРАВО
  ЕСЛИ ПУСТО
    ТО Длина_НК // Рекурсивный цикл
    ИНАЧЕ
    {
      // 2) Измеряем расстояние КО
      // 3) Откладываем КО
      Длина_КО
      // 4) Пишем Н
      ПИШИ Н
    }
  // 5) Откладываем НК (рекурсивная пружинка)
  ВПРАВО
КОНЕЦ

ЭТО Длина_КО
  // 1) Рекурсивно измеряем расстояние КО
  ВПРАВО
  ЕСЛИ ПУСТО
    ТО Длина_КО // Рекурсивный цикл
  // 2) Откладываем КО (рекурсивная пружинка)
  ВПРАВО
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 15.18.

Таблица 15.18

Тест	Комментарий	Ожидаемый результат
НКО НК□О НК□□О	Вырожденный отрезок	НКОКН НК□О□КН НК□□О□□КН
Н□КО Н□К□О Н□□К□О Н□□К□□О Н□□□□К□□□О	Невырожденный отрезок	Н□КОК□Н Н□К□О□К□Н Н□□К□О□К□□Н Н□□К□□О□□К□□Н Н□□□□К□□□О□□□К□□□□Н

19. Деление отрезка (В. П. Семенко, Рубцовск).

На ленте — распакованный отрезок длиной 3k (k — натуральное число). Разделить отрезок на три равные части, т. е. построить плотно один за другим три распакованных отрезка, каждый из которых имеет длину k. В начале работы в окне видна буква Н — начало исходного отрезка.

На рис. 15.19 показан пример начального и конечного состояний среды.

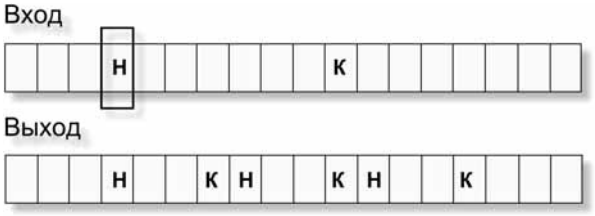

Рис. 15.19

Решение

Описание алгоритма

Подсчитаем длину k исходного отрезка, рекурсивно продвигая по нему окно. За один шаг будем продвигаться на три ячейки, чтобы отложенная команда (**плюс**) сработала k/3 раз. Затем построим три отрезка длиной k/3.

Построение каждого отрезка сделаем процедурой с рекурсивной пружинкой, которая будет восстанавливать длину построенного отрезка (чтобы можно было построить следующий отрезок).

Алгоритм

1. Стираем Н — начало исходного отрезка.
2. Находим длину k = 1 / 3 длины исходного отрезка (процедура `Деление_на_3`).
3. Строим три отрезка с длиной k (процедура `Построение`).
4. Удаляем с ленты значение k.

Алгоритм `Деление_на_3`

1. Смещаемся на три ячейки вправо.
2. Проверяем, что в окошке:
   2.1. Если **пусто**, то К не найдено, вызываем процедуру `Деление_на_3` (рекурсия).
   2.2. Если не **пусто** (в окошке К), то пишем на её месте 0.
3. Восстанавливаем на ленте длину k отложенной командой **плюс**, которая будет выполнена втрое меньше раз, чем было сделано шагов вправо.

Алгоритм Построение

1. Из окошка записываем в ящик k.
2. Пишем H — начало нового отрезка.
3. Строим новый отрезок (процедура Отрезок).

Алгоритм Отрезок

1. Длину k из ящика записываем на ленту.
2. Проверяем значение k:
   2.1. Если длина не 0, то уменьшаем длину на 1 и вызываем процедуру Отрезок, откладывая команду **плюс** для восстановления длины k.
   2.2. Если длина 0, то отрезок построен, пишем K — конец нового отрезка.

Программа

```
// Автор решения: В. П. Семенко
ЭТО Вход
  // Стираем H - начало отрезка и продвигаем окно на его тело
  ПИШИ ПУСТО ВПРАВО
  // Находим 1/3 длины исходного отрезка.
  // После выполнения процедуры - в окне результат.
  Деление_на_3
  // Строим три отрезка длины, равной 1/3 от длины исходного отрезка.
  // После выполнения процедуры Построение слева от окна — построенный
  // отрезок, а в окне - его длина.
  ПОВТОРИ 3 Построение
  // Стираем с ленты длину отрезка - она больше не нужна.
  ПИШИ ПУСТО
КОНЕЦ

ЭТО Деление_на_3
  // Смещаем окно на три ячейки вправо
  ПОВТОРИ 3 ВПРАВО
  ЕСЛИ ПУСТО
    ТО Деление_на_3 // Рекурсивный цикл
    // Нашли K, пишем на этом месте 0
    ИНАЧЕ ПИШИ 0 // Записываем начальное значение длины
  // Рекурсивная пружина:
  // Формируем на ленте 1/3 длины исходного отрезка
  ПЛЮС
```

```
КОНЕЦ

// Вход: в окне длина отрезка
// Выход: слева построенный отрезок, в окне длина построенного отрезка
// ---------------------------------------------------------------------
ЭТО Построение
  // Записываем в ящик длину отрезка
  ОБМЕН
  // Пишем на ленте Н - начало отрезка и смещаемся на его тело
  ПИШИ Н ВПРАВО
  // Строим отрезок
  Отрезок
  // Конец отрезка (символ К) будет записан в процедуре Отрезок
КОНЕЦ

ЭТО Отрезок
  // Длину непостроенной части отрезка запишем на ленту
  ОБМЕН
  ЕСЛИ НЕ 0
    ТО
    {
      // Уменьшим длину непостроенного отрезка и продвинемся
      // на одну ячейку по его телу.
      МИНУС ОБМЕН ВПРАВО
      Отрезок // Рекурсивный цикл
      // Рекурсивная пружинка:
      // восстанавливает длину построенного отрезка
      ПЛЮС
    }
    ИНАЧЕ
    {
      // Тело отрезка закончилось. Записываем К - конец отрезка.
      ПИШИ К
      // Смещаем окно в следующую ячейку и подготавливаем
      // её для работы рекурсивной пружинки (записываем 0)
      ВПРАВО ПИШИ 0
    }
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 15.19.

Таблица 15.19

Тест	Комментарий	Ожидаемый результат
Н□□□К	Длина исходного отрезка равна 3	Н□К Н□К Н□К
Н□□□□□□□□□К	Длина исходного отрезка больше 3	Н□□□К Н□□□К Н□□□К

20. Среднее арифметическое (В. П. Семенко, Рубцовск).

Средним арифметическим называется отношение суммы слагаемых к их количеству.

Среднее арифметическое = $(a_1 + a_2 + a_3 + \ldots + a_n)/n$.

Например, средним арифметическим для чисел 1, 2 и 9 будет число 4.

Среднее арифметическое = $(1 + 2 + 9)/3 = 4$.

А для чисел 1, 2, 3, 4, 5, 6, 7 и 8 среднее арифметическое равно числу 4.5.

Среднее арифметическое = $(1 + 2 + 3 + 4 + 5 + 6 + 7 + 8)/8 = 4.5$.

Через окно на ленте видно символьное число n, не меньшее двух. Найти среднее арифметическое всех натуральных чисел от 1 до n включительно и записать результат в виде символьного числа (для целого результата) или символьного числа, за которым через десятичную точку записано другое символьное число, изображающее дробную часть.

Положение окошка после исполнения программы несущественно. Исходные данные на ленте можно не сохранять.

На рис. 15.20 показан пример начального и конечного состояний среды.

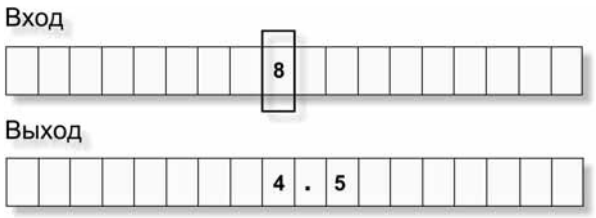

Рис. 15.20

Решение

Немного математики

Обозначим через S(n) сумму натуральных чисел от 1 до n включительно, а через M(n) — среднее арифметическое этих чисел.

Тогда:

$S(n) = 1 + 2 + \ldots + (n - 1) + n$ (1)

или

$S(n) = n + (n - 1) + \ldots + 2 + 1$ (2)

Сложим почленно равенства (1) и (2):

$2S(n) = n \cdot (n + 1)$

Откуда:

$S(n) = n \cdot (n + 1)/2$

Тогда:

$M(n) = (n + 1)/2$ (3)

Последнюю формулу для реальных вычислений перепишем в виде:

$M(n) = n/2 + 1/2$ (4)

Вычисление по формуле (4) позволяет расширить область допустимых значений для n без аварийных сообщений исполнителя.

В самом деле, вычисление по формуле (3) возможно только для $n < MAX$ (MAX — максимальное символьное число Корректора или максимальное целое число в других системах программирования). Вычисление по формуле (4) возможно и для $n = MAX$.

Описание алгоритма

Рекурсивно найдём остаток от деления числа n на 2, заменив деление вычитанием. Величина остатка подскажет значение дробной части у результата M(n) = n / 2 + 1 / 2: если остаток равен 0, результат — дробное число, если 1 — целое. Целую часть добавим к дробной за счёт рекурсивной пружинки, которая будет запоминать число вычтенных двоек.

Программа

```
// Автор идеи Евгений Сериков, Лукоянов; реализация — В. П. Семенко
// Дано:   n
// Найти:  M(n) = n/2+1/2
// ----------------------
ЭТО Вход
  ЕСЛИ 1
    ТО    Дробная_часть
    ИНАЧЕ Работа
КОНЕЦ


ЭТО Дробная_часть
```

```
  ПИШИ 0 ВПРАВО ПИШИ . ВПРАВО ПИШИ 5
КОНЕЦ


ЭТО Работа
  // Уменьшаем символьное число на 2
  ПОВТОРИ 2 МИНУС
  ЕСЛИ НЕ 0
    ТО ЕСЛИ НЕ 1 ТО Работа // Остаток больше 1: Рекурсивный цикл
    ИНАЧЕ
    {
      // В окошке 0, значит, n - чётное, а M(n)=n/2+1/2 - дробь.
      // Дописываем дробную часть .5
      Дробная_часть
      // Возвращаем окно на место целой части
      ПОВТОРИ 2 ВЛЕВО
    }
  // Рекурсивная пружинка:
  // добавляем к остатку (0 или 1) целую часть от деления n на 2
  ПЛЮС
КОНЕЦ
```

Комментарий. Отложенная команда **ПЛЮС** начинает работать, если в окошке цифра 0 или цифра 1.

Тесты

Набор тестов приводится в табл. 15.20.

Таблица 15.20

Тест	Комментарий	Ожидаемый результат
1	Нижняя граница возможных значений n	0.5
2		1.5
3		2
8	«Типичное» n	4.5
9		5
~	Верхняя граница возможных значений n	Ш.5
@		Щ

21. Самое длинное бревно (Н. В Табашин, Лукоянов).

На ленте Корректора расположено несколько брёвен (плотные цепочки символов «*»). Брёвна отделяются друг от друга символом **ПРОБЕЛ**. Найти длину самого длинного бревна и записать ответ на ленту в виде символьного числа. Длина бревна — это количество звёздочек, составляющих запись бревна на ленте. В начале работы окно находится на первом символе записи, в конце — справа от записи вплотную к ней, и в нём виден ответ.

На рис. 15.21 показан пример начального и конечного состояний среды.

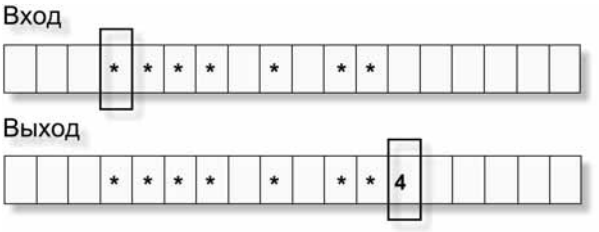

Рис. 15.21

Решение

Описание алгоритма

Для нахождения длины максимального бревна будем использовать ящик. Сначала запишем в ящик 0. Затем просмотрим запись по брёвнам. Каждый раз, когда длина просмотренного бревна оказывается больше числа в ящике, меняем содержимое ящика на это значение. После просмотра всех брёвен в ящике окажется искомое значение.

Программа

```
// Автор решения: В. П. Семенко
ЭТО Вход
  // Занесём в ящик начальное значение для длины
  // максимального бревна
  ОБМЕН ПИШИ 0 ОБМЕН
  // Просмотр записи по брёвнам
  ПОКА НЕ ПУСТО
  {
    // Просмотр бревна
    Длина_бревна
    // Если длина бревна больше числа в ящике,
```

```
    // заменяем число в ящике этим значением.
    ЕСЛИ Я<Л ТО ЯЩИК+
    ПИШИ ПРОБЕЛ
    // На следующее бревно
    ВПРАВО
  }
  // Запись результата
  ВЛЕВО ЯЩИК-
КОНЕЦ

ЭТО Длина_бревна
  ВПРАВО
  ЕСЛИ *
    ТО    Длина_бревна // Рекурсивный цикл
    ИНАЧЕ ПИШИ 0
  // Рекурсивная пружинка:
  // записывает длину бревна на ленту.
  ПЛЮС
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 15.21.

Таблица 15.21

Тест	Комментарий	Ожидаемый результат
□	Лента пуста	0
*	Одно бревно	1
***		3
	Несколько брёвен	
*□***	Последнее бревно длиннее	3
□**		4
****□*□**□***	Первое бревно длиннее	4
□*□****□**	Длинное бревно внутри	6
□□***□***	Одинаковые брёвна	3
**********□*□*	Бревно длиннее 9	A

Задачи к главе 12

22. Складывай и вычитай (В. П. Семенко, Рубцовск).

На ленте записан бесскобочный пример на сложение и вычитание символьных чисел. Выполнить действия и записать ответ в виде символьного числа (возможно, со знаком минус).

В начале работы программы окошко находится на первом символе записи.

Дополнительные договорённости:

- Символы «–» и «+» — это только знаки действий и символьными числами быть не могут.
- Конечные и промежуточные результаты вычислений не выходят из диапазона [–n, n], где n — номер последнего символа из алфавита Корректора относительно символа 0.
- Символ `пусто` числом не является.
- Исходные данные на ленте можно не сохранять.

На рис. 15.22 показан пример начального и конечного состояний среды.

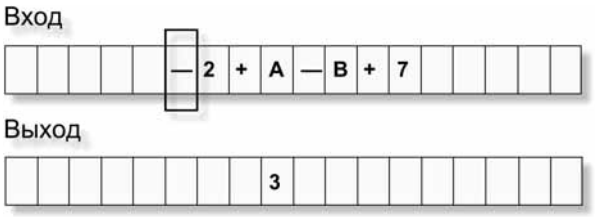

Рис. 15.22

Решение

Описание алгоритма

Если запись начинается не со знака числа, допишем слева знак +. Таким образом, запись единообразно будет представлять собой последовательность пар: ЗнакЧисло.

В качестве накопителя результата будем использовать ящик. Сначала запишем в ящик 0, а затем будем добавлять в ящик числа, учитывая их знаки и знак числа в ящике.

Знак числа в ящике будем хранить на ленте перед очередной парой ЗнакЧисло (исходную запись на ленте будем стирать по мере продвижения).

При выполнении операции над числом в ящике и числом на ленте будем учитывать их знаки следующим образом:

1. Знаки одинаковые. Действие: сложение. Знак результата: совпадает с общим знаком операндов.

2. Знаки разные. Действие: вычитание из большего меньшего (сравнение по модулю, вычитание модулей). Знак результата: совпадает со знаком большего числа.

Программа

```
ЭТО Вход
  // Допишем знак, если его нет перед первым числом.
  ЕСЛИ    НЕ -
  ТО ЕСЛИ НЕ +
  ТО { ВЛЕВО ПИШИ + }
  // Обнулим сумматор
  ОБМЕН ПИШИ 0 ОБМЕН
  // Запишем на ленту знак числа в ящике
  ВЛЕВО ПИШИ + ВПРАВО
  Вычисления
  // Запись ответа (из ящика)
  ЯЩИК-
КОНЕЦ

ЭТО Вычисления
  // Цикл по парам ЗнакЧисло
  ПОКА НЕ ПУСТО
  {
    Выполнить_действие
    // Смещение к следующей паре ЗнакЧисло
    ВПРАВО
  }
КОНЕЦ

ЭТО Выполнить_действие
  // Сохраним значение ящика на ленте и сравним
  // знаки числа в ящике и числа на ленте
  ОБМЕН ВЛЕВО
  ЕСЛИ Я=Л
   ТО    Знаки_совпадают
   ИНАЧЕ Знаки_разные
КОНЕЦ
```

```
ЭТО Знаки_совпадают
  // Сложим числа и припишем к результату общий знак
  // Удалим с ленты знак числа в ящике
  ПИШИ ПУСТО
  // Восстановим знак числа на ленте и значение ящика
  ВПРАВО ОБМЕН
  // Выполним сложение
  ВПРАВО
  ПОКА НЕ 0 { МИНУС ОБМЕН ПЛЮС ОБМЕН }
  // Перемещаем знак числа в ящике к следующей паре ЗнакЧисло
  ВЛЕВО ОБМЕН ВПРАВО ЯЩИК-
  ВЛЕВО ЯЩИК+ ПИШИ ПУСТО ВПРАВО
КОНЕЦ

ЭТО Знаки_разные
  // Надо из большего из двух чисел (в ящике и на ленте)
  // отнять меньшее и приписать ему знак большего числа.
  // Восстановим знак числа на ленте и значение ящика
  ВПРАВО ОБМЕН
  // Сравним числа
  ВПРАВО
  ЕСЛИ Я<Л
    ТО    Лента-Ящик
    ИНАЧЕ Ящик-Лента
КОНЕЦ

ЭТО Лента-Ящик
  ОБМЕН
  ПОКА НЕ 0 { МИНУС ОБМЕН МИНУС ОБМЕН }
  // Стираем прежний знак ящика и перемещаем
  // знак числа на ленте к следующей паре ЗнакЧисло
  ВЛЕВО ОБМЕН ВПРАВО ОБМЕН
  ВЛЕВО ОБМЕН ПИШИ ПУСТО
  ВЛЕВО ПИШИ ПУСТО
  ВПРАВО ВПРАВО
КОНЕЦ
```

```
ЭТО Ящик-Лента
  ПОКА НЕ 0 { МИНУС ОБМЕН МИНУС ОБМЕН }
  // Прежний знак ящика перемещаем к следующей паре ЗнакЧисло
  ВЛЕВО ВЛЕВО ОБМЕН ВПРАВО ВПРАВО ОБМЕН
  ВЛЕВО ВЛЕВО ОБМЕН ПИШИ ПУСТО
  ВПРАВО ПИШИ ПУСТО
  ВПРАВО
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 15.22.

Таблица 15.22

Тест	Комментарий	Ожидаемый результат
–2	Одно число на ленте	-2
А		А
+@		+@
А-2	Два числа на ленте	8
А+2		В
А-Б		-1
-А-2		-В
-А+2		-8
+А-2		8
+А+2		В
А-Б+1-А+Б	Несколько чисел на ленте	1

23. Неправильная дробь (В. П. Семенко, Рубцовск).

На ленте Корректора записана неправильная обыкновенная палочная дробь: числитель больше знаменателя или равен ему. Числитель дроби отделяется от знаменателя символом /.

Выделить из неправильной дроби целую часть и записать результат в виде смешанной палочной дроби, отделяя целую часть от дробной символом ПРОБЕЛ. В начальный момент в окошко виден символ /. Исходные данные на ленте можно не сохранять.

На рис. 15.23 показан пример начального и конечного состояний среды.

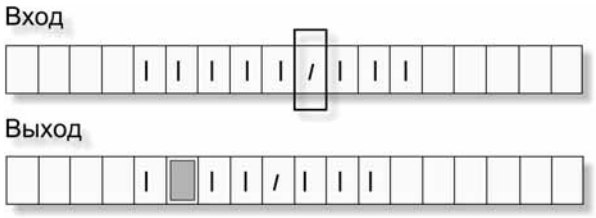

Рис. 15.23

Решение

Немного математики

Дробь вида m/n, где m — число целое, а n — натуральное, называется обыкновенной дробью.

Обыкновенная дробь, числитель которой больше знаменателя или равен ему, называется неправильной.

Для выделения из неправильной дроби целой части, необходимо числитель дроби разделить на знаменатель с остатком. Тогда частное от деления будет целой частью результата, остаток от деления — числителем дробной части, а знаменатель дробной части останется прежним. Например:

9 / 2 = 4 1 / 2

9 / 3 = 3

В последнем примере остаток от деления 9 на 3 равен нулю, значение дроби 0/3 также равно 0. В этом случае дробная часть не пишется.

Описание алгоритма основной процедуры `Вход`

1. Переводим знаменатель дроби в символьное число (процедура `Знаменатель`).
2. Переводим числитель дроби в символьное число (процедура `Числитель`).
3. Делим числитель на знаменатель — выделяем целую часть (процедура `Деление`).
4. Проверяем, какое число получится в результате деления:
   4.1. Если остаток равен знаменателю, то добавим к частному 1 (остаток теперь 0), и результат — число целое (переход на процедуру `Целое_число`).
   4.2. Если остаток меньше знаменателя, то результат — число дробное (переход на процедуру `Дробное число`).

Описание процедуры `Знаменатель`

Рекурсивно проходим вправо по палочкам знаменателя и отложенной командой **плюс** формируем символьное число, равное числу пройденных палочек.

Описание процедуры Числитель

Рекурсивно проходим влево по палочкам числителя и отложенной командой **плюс** формируем символьное число, равное числу пройденных палочек.

Описание алгоритма процедуры Деление

Пока числитель больше знаменателя, из числителя вычитаем знаменатель и считаем, сколько раз выполнена эта операция.

Алгоритм процедуры Целое_число

1. Стираем знаменатель и символ /.
2. Прибавляем к целой части 1.
3. Переводим символьное число в палочное.

Алгоритм процедуры Дробное_число

1. Сохраняем целую часть и числитель дробной части на ленте.
2. Знаменатель дробной части переводим в палочное число.
3. Числитель дробной части переводим в палочное число.
4. Отделяем дробную часть от целой символом **ПРОБЕЛ**.
5. Переводим целую часть в палочное число.

Программа

```
// Автор решения: В. П. Семенко
ЭТО Вход
  // |||||/|||
  Знаменатель
  // |||||/3
  ВЛЕВО
  Числитель
  // 5/3
  ОБМЕН // Копируем числитель в ящик
  Деление
  // 1/3 (в ящике: 2)
  ЕСЛИ Я=Л
    ТО Целое_число
    ИНАЧЕ Дробное_число
КОНЕЦ

ЭТО Знаменатель
  // Переводим знаменатель в символьное число
```

```
  ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО
    {
      ПИШИ ПУСТО
      Знаменатель
    }
    ИНАЧЕ
    {
      // Возвращаемся к символу /
      ПОКА ПУСТО ВЛЕВО
      ВПРАВО
    }
  // Восстанавливаем знаменатель в виде символьного числа
  ПЛЮС
КОНЕЦ

ЭТО Числитель
  // Переводим числитель в символьное число
  ВЛЕВО
  ЕСЛИ НЕ ПУСТО
    ТО
    {
      ПИШИ ПУСТО
      Числитель
    }
    ИНАЧЕ
    {
      // Возвращаемся к символу /
      ПОКА ПУСТО ВПРАВО
      ВЛЕВО
    }
  // Восстанавливаем числитель в виде символьного числа
  ПЛЮС
КОНЕЦ

ЭТО Деление
  // Находим частное
```

```
  ПИШИ 0
  // Смещаем окно на знаменатель
  ПОВТОРИ 2 ВПРАВО
  // Пока числитель больше знаменателя, выделяем целую часть
  ПОКА Я>Л Дели
КОНЕЦ

ЭТО Дели
  // Проверяем значение знаменателя
  ЕСЛИ НЕ 0
    ТО
    {
       // Вычитаем из числителя и знаменателя по 1
       ПОВТОРИ 2 { МИНУС ОБМЕН }
       Дели
       // Откладываем команду для восстановления знаменателя
       ПЛЮС
    }
    ИНАЧЕ
    {
      // Прибавляем к целой части 1
      ПОВТОРИ 2 ВЛЕВО
      ПЛЮС
      ПОВТОРИ 2 ВПРАВО
    }
КОНЕЦ

ЭТО Целое_число
  // Я=Л т.е. дробь вида 2 3/3
  ПИШИ ПУСТО ОБМЕН
  ПОВТОРИ 2 { ПИШИ ПУСТО ВЛЕВО }
  ПЛЮС // Прибавляем к частному 1
  // Записываем ответ в виде палочного числа
  ПОКА НЕ 0
  {
    МИНУС ОБМЕН ПИШИ |
    ВЛЕВО ОБМЕН
```

```
  }
  ПИШИ ПУСТО
КОНЕЦ

ЭТО Дробное_число
  // 1/3 (в ящике: 2)
  ПОВТОРИ 2 ВЛЕВО
  ОБМЕН ВЛЕВО ОБМЕН
  ПОВТОРИ 3 ВПРАВО
  // 12/3 (в ящике: ПУСТО)
  // Знаменатель дробной части переводим в палочное число
  ПОКА НЕ 0
  {
    МИНУС ОБМЕН ПИШИ |
    ВПРАВО ОБМЕН
  }
  // 12/|||0
  // Стираем 0
  ПИШИ ПУСТО
  // 12/|||□
  // Перемещаем окно на числитель
  ВЛЕВО
  ПОКА НЕ / ВЛЕВО
  ВЛЕВО
  // 12/|||
  // Переводим в палочное число числитель
  ПОКА НЕ 0
  {
    МИНУС ОБМЕН ПИШИ |
    ПОВТОРИ 2 { ВЛЕВО ОБМЕН }
    ВПРАВО
  }
  // 10||/|||
  // Отделяем дробную часть от целой
  ПИШИ ПРОБЕЛ
  // 1■||/|||
  ВЛЕВО
```

```
  // Переводим целую часть в палочное число
  ПОКА НЕ 0
  {
    МИНУС ОБМЕН ПИШИ |
    ВЛЕВО ОБМЕН
  }
  ПИШИ ПУСТО
  //□|■|||/|||
КОНЕЦ
```

Комментарий. Программа будет правильно работать для любой обыкновенной дроби, и для правильной, например, ||/|||, и для дроби, значение которой равно 0, например, /|||. Но в ней не предусмотрена обработка ошибки деления на 0. Такую проверку легко сделать, переписав основную процедуру Вход в следующем виде:

```
ЭТО Вход
  Знаменатель
  ЕСЛИ НЕ 0
    ТО
    {
      ВЛЕВО
      Числитель
      ОБМЕН // Копируем числитель в ящик
      Деление
      ЕСЛИ Я=Л
        ТО Целое_число
        ИНАЧЕ Дробное_число
    }
    ИНАЧЕ Деление_на_0
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 15.23.

Таблица 15.23

Тест	Комментарий	Ожидаемый результат
|/| |||/|||	Числитель и знаменатель исходной дроби равны	| |

Таблица 15.23 *(окончание)*

Тест	Комментарий	Ожидаемый результат
||||/||	Числитель больше знаменателя и кратен ему	||
||||/||| ||||/|	Числитель больше знаменателя и не кратен ему	|□|/||| ||||

24. Возведение в квадрат (В. П. Семенко, Рубцовск).

Корректору предложили возвести натуральное число в квадрат. Он усмехнулся — элементарно! Ведь умножение можно заменить сложением, например: $5 \cdot 5 = 5 + 5 + 5 + 5 + 5$. А потом подумал и нашёл другой способ.

Решите и вы задачу по-другому.

В начальный момент в окне расположено натуральное символьное число. Замените его другим символьным числом — квадратом исходного. Считается, что квадрат исходного числа может быть изображён символом алфавита Корректора.

На рис. 15.24 показан пример начального и конечного состояний среды.

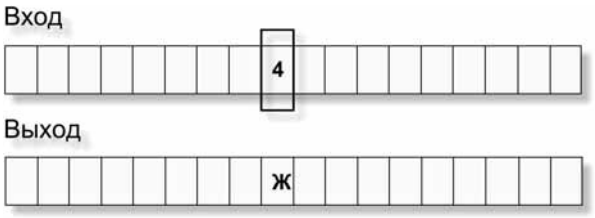

Рис. 15.24

Решение

Немного математики

Известно, что квадрат любого натурального числа n равен сумме n первых натуральных нечётных чисел.

$1^2 = 1$

$2^2 = 1 + 3$

$3^2 = 1 + 3 + 5$

$4^2 = 1 + 3 + 5 + 7$

…

$n^2 = 1 + 3 + 5 + 7 + 9 + \ldots + n$

Алгоритм решения основан на сложении первых n натуральных нечётных чисел.

Алгоритм

Исходное число будем использовать как счётчик числа нечётных чисел, а ячейку справа от него — для формирования самих нечётных чисел. Сумму будем накапливать в ящике.

1. Если в окошке не 1, то делаем (иначе ответ уже есть):
   1.1. Уменьшаем исходное число на 1; записываем первое нечётное число 1 в соседнюю ячейку справа и в ящик.
   1.2. Пока не сложили нужное количество нечётных чисел:
      1.2.1. Уменьшаем счётчик нечётных чисел на 1.
      1.2.2. Записываем следующее нечётное число (увеличиваем текущее на 2) и добавляем его к числу в ящике.
   1.3. Записываем ответ из ящика на ленту.

Программа

```
// Автор решения: В. П. Семенко
ЭТО Вход
  // 4
  ЕСЛИ НЕ 1
    ТО
    {
      МИНУС  // Уменьшаем счётчик нечётных чисел на 1
      ВПРАВО
      ПИШИ 1 // Записываем первое нечётное число на ленту
      ЯЩИК+  // и в ящик
      ВЛЕВО
      // 31 (Ящик: 1)
      Работа
      // Ж
    }
КОНЕЦ

ЭТО Работа
  // 31 (Ящик: 1)
  // Пока не сложили все нечётные числа, выполняем сложение
  ПОКА НЕ 0 Сложение
  // 077 (Ящик: Ж)
  // Очищаем рабочие ячейки и записываем результат
```

```
  ПОВТОРИ 3 { ПИШИ ПУСТО ВПРАВО }
  ОБМЕН
  // Ж
КОНЕЦ

ЭТО Сложение
  МИНУС ВПРАВО
  ПОВТОРИ 2 ПЛЮС // Следующее нечётное число
  // Сохраним копию нечётного числа справа
  ОБМЕН ВПРАВО ЯЩИК- ВЛЕВО ОБМЕН
  Прибавь
  // Восстановим нечётное число
  ОБМЕН ВПРАВО ЯЩИК+ ВЛЕВО ОБМЕН
  // Окно на счётчик нечётных чисел
  ВЛЕВО
КОНЕЦ

ЭТО Прибавь
  ПОКА НЕ 0 { МИНУС ОБМЕН ПЛЮС ОБМЕН }
КОНЕЦ
```

Замечание

На самом деле Корректор перемудрил! Если умножение заменять сложением, например: $5 \cdot 5 = 5 + 5 + 5 + 5 + 5$, то программа получается короче, а количество основной работы на ленте остаётся прежним.

Программа

```
ЭТО Вход
  Копия
  Работа
  Результат
КОНЕЦ

ЭТО Копия
  ЯЩИК+ ВПРАВО ЯЩИК- ВЛЕВО // Копия числа n
КОНЕЦ

ЭТО Работа
  ПОКА НЕ 1
  {
```

```
    ВПРАВО Добавить_n ВЛЕВО
    МИНУС
  }
КОНЕЦ

ЭТО Добавить_n
МИНУС
  ЕСЛИ НЕ ПУСТО ТО { ОБМЕН ПЛЮС ОБМЕН Добавить_n }
  ПЛЮС // Восстановление n (пружинка)
КОНЕЦ

ЭТО Результат
  ЯЩИК- ВПРАВО ПИШИ ПУСТО ВЛЕВО
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 15.24.

Таблица 15.24

Тест	Ожидаемый результат
1	1
2	4
3	9
4	Ж
8	#

25. Округление (Владимир Слотин, 8 класс, Тольятти).

Справа от окошка записано десятичное число. Округлить его с точностью, заданной символьным числом слева от окошка. Точность — это количество цифр дробной части результата. Исходные данные на ленте можно не сохранять.

На рис. 15.25 показан пример начального и конечного состояний среды.

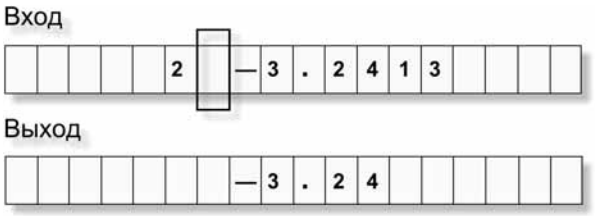

Рис. 15.25

Решение

Алгоритм

Сначала выполним подготовку (процедура `Подготовка`): запомним в ящике число, задающее точность, и установим окно слева от округляемого числа. Затем приступим к округлению (процедура `Работа`).

Алгоритм процедуры `Работа`

1. Смещаемся на первый символ числа.
2. Проверяем символ в окошке:
   2.1. Если в окошке «.», то число дробное.
      2.1.1. Устанавливаем окно на цифру, влияющую на округление (процедура `Точность`).
      2.1.2. Округляем дробное число (процедура `Округление`).
   2.2. Если в окошке **ПУСТО**, то число целое.
      2.2.1. Записываем символ «.».
      2.2.2. Считаем, сколько нулей нужно дописать (процедура `Точность`).
      2.2.3. Дописываем нули.
   2.3. Если в окошке иной символ, то рекурсивно вызываем процедуру `Работа`.

Описание алгоритма процедуры `Точность`

Пока число, задающее точность округления, хранящееся в ящике, не равно 0, уменьшаем его на 1 и двигаемся вправо.

Описание алгоритма `Округление`

Сравниваем число, влияющее на округление с числом 5, удаляем цифры справа и при необходимости прибавляем 1 к последней оставшейся цифре, учитывая, что это может быть и 9.

Программа

```
// Автор решения: Анна Громова, 9 класс, Тверь
ЭТО Вход
  Подготовка
  Работа
КОНЕЦ

// Запомнить в ящике точность и установить окно слева от числа
ЭТО Подготовка
  ВЛЕВО ОБМЕН ВПРАВО
КОНЕЦ
```

```
ЭТО Работа
  ВПРАВО
  ЕСЛИ .
    // Число дробное, становимся на цифру, влияющую на округление.
    ТО  { Точность Округление }
    ИНАЧЕ ЕСЛИ ПУСТО
            // Число целое, дописываем нули
            ТО
            {
              ПИШИ . Точность
              ПИШИ ПУСТО Нули
            }
            ИНАЧЕ  Работа
КОНЕЦ

ЭТО Точность
  ВПРАВО ОБМЕН
  ПОКА НЕ 0
  {
    МИНУС ОБМЕН
    ВПРАВО ОБМЕН
  }
КОНЕЦ

ЭТО Округление
  ПИШИ 5
  ОБМЕН
  ЕСЛИ ПУСТО
    ТО Нули
    ИНАЧЕ ЕСЛИ Я>Л
            ТО
            {
              Удаление_цифр
              ЕСЛИ . ТО ПИШИ ПУСТО
            }
            ИНАЧЕ
            {
```

```
            Удаление_цифр
            Плюс1
          }
КОНЕЦ

ЭТО Плюс1
 ЕСЛИ .
   ТО
   {
     ПИШИ ПУСТО ВЛЕВО
     Плюс1
   }
   ИНАЧЕ ЕСЛИ НЕ 9
         ТО ПЛЮС
         ИНАЧЕ
         {
           ПИШИ 0 ВЛЕВО
           ЕСЛИ ЦИФРА
             ТО Плюс1
             ИНАЧЕ ЕСЛИ .
                   ТО { ВЛЕВО Плюс1 }
                   ИНАЧЕ { ЯЩИК+ ПИШИ 1 ВЛЕВО ЯЩИК- }
         }
КОНЕЦ

// Убрать, если надо, лишние цифры
// ------------------------------
ЭТО Удаление_цифр
  ПОКА НЕ ПУСТО
  {
    ПИШИ ПУСТО
    ВПРАВО
  }
  ПОКА ПУСТО ВЛЕВО
КОНЕЦ

ЭТО Нули
  ВЛЕВО
```

```
  ПОКА ПУСТО
  {
    ПИШИ 0
    ВЛЕВО
  }
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 15.25.

Таблица 15.25

Тест	Комментарий	Ожидаемый результат
0□1 0□-1 0□23 0□-32	Округлить число с точностью до единиц	1 -1 23 -32
1□1 1□-1 1□23 1□-32 1□12.99 1□-32.89 1□32.4 1□0.99	Округлить число с точностью до десятых	1.0 -1.0 23.0 -32. 13.0 -32.9 32.4 1.0
2□1 2□76.3421 2□45.5455 2□-99.99999	Округлить с точностью до сотых	1.00 76.34 45.55 -100.00

26. Умножение на пять (Евгений Сериков, 10 класс, Лукоянов).

    На ленте Корректора записано целое число и окно расположено на младшей его цифре. Умножить число на 5, не заменяя умножение сложением.

    На рис. 15.26 показан пример начального и конечного состояний среды.

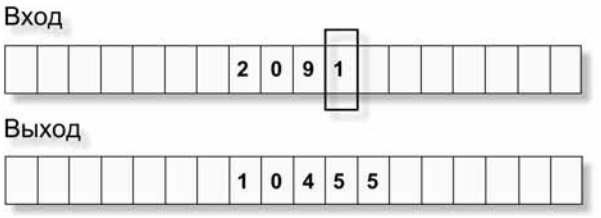

Рис. 15.26

Решение

Описание алгоритма

Сначала число умножаем на 10, дописывая нуль справа, а затем результат делим на 2.

Алгоритм основной процедуры `Вход`

1. Умножаем число на 10 (приписываем справа 0).
2. Пока цифры в числе не закончились, делим на 2 каждую его цифру (процедура `Деление_на_2`).
3. Удаляем незначимый ноль в старшем разряде, если он там есть (процедура `Старший_разряд`).

Описание алгоритма процедуры `Деление_на_2`

Деление заменим вычитанием. В рекурсивном цикле отнимаем от цифры по 2 единицы, пока не получим вместо неё 0 (цифра была чётной) или `пусто` (цифра была нечётной). Затем рекурсивной пружиной (отложенной командой `плюс`) записываем вместо цифры число вычитаний — оно равно целой части от деления цифры на 2. При этом если исходная цифра была нечётной, добавляем 5 единиц к разряду справа.

Алгоритм процедуры `Старший разряд`

1. Перемещаем окошко вправо.
2. Проверяем, что в окошке:
   2.1. Если в окошке 0, то число положительное, и в старшем разряде незначимый 0 — удаляем его.
   2.2. Если в окошке «–», то заносим «–» в ящик и перемещаем окошко на старший разряд.
      2.2.1. Если в окошке 0, то на его месте пишем «–».
      2.2.2. Если в окошке не 0, то «–» восстанавливаем на старом месте.

Программа

```
// Автор решения Евгений Сериков, Лукоянов
ЭТО Вход
  // Умножаем число на 10, дописывая справа 0
  ВПРАВО ПИШИ 0 ВЛЕВО
  // Пока число не закончилось, делим его на 2
  ПОКА НЕ ПУСТО
  {
    ЕСЛИ    НЕ 0
    ТО ЕСЛИ НЕ -
    ТО Деление_на_2
    ВЛЕВО
  }
  // Удаляем незначимый ноль в старшем разряде
  Старший_разряд
КОНЕЦ

ЭТО Деление_на_2
  // Из цифры разряда отнимаем 2
  ПОВТОРИ 2 МИНУС
  ЕСЛИ НЕ 0
    ТО ЕСЛИ НЕ ПУСТО
         ТО Деление_на_2
         ИНАЧЕ
           // Делили нечётное: разряд справа увеличиваем на 5
         { // (число, большее 9, в сумме не получится)
           ВПРАВО
           ПОВТОРИ 5 ПЛЮС
           ВЛЕВО
         }
  // Восстанавливаем целую часть от деления на 2
  ПЛЮС
КОНЕЦ

ЭТО Старший_разряд
  // На старший разряд
  ВПРАВО
```

```
ЕСЛИ 0
  // Число положительное и в старшем разряде незначащий ноль
  ТО ПИШИ ПУСТО
  ИНАЧЕ ЕСЛИ -
          // Число отрицательное
          ТО
          {
            ПИШИ ПУСТО // Вместо -
            // На старший разряд
            ВПРАВО
            ЕСЛИ 0
              // Удаляем ноль
              ТО ПИШИ -
              // Знак "-" пишем на место
              ИНАЧЕ { ВЛЕВО ПИШИ - }
           }
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 15.26.

Таблица 15.26

Тест	Ожидаемый результат
□	□
0	0
1	5
-1	-5
100	500
999	4995
-10693	-53465

Часть III

Транслятор?.. Это очень просто!

Напрасно я объяснял своим друзьям, что телевизионная техника не только не проста, но согласно выражению Незнайкина дьявольски сложна, что она затрагивает различные области физики, что положение ещё усложняется из-за отсутствия международного стандарта.
Ничего не помогло. Я должен был покориться и написать «Телевидение?.. Это очень просто!»

Е. Айсберг

Этот раздел для тех, кто твёрдо уверен, что кулинария и мясорубка — это не одно и тоже, и ради хорошей кухни готов покрутить ручку, пренебрегая всякими электрическими излишествами. Ведь «ручная» котлета сохраняет больше витаминов!

Если переходить от метафор к реальным планам, то речь в этом разделе пойдёт о программировании трансляторов в средах Кукарачи и Корректора.

Вероятно, читатель, знакомый с этой, по выражению Незнайкина, дьявольски сложной темой, не смог удержать ироническую улыбку: как можно писать трансляторы в среде Кукарачи, если в ней нет переменных, и даже в среде Корректора, с его примитивной лентой и ящиком?

Тем не менее, трансляторы в этих средах — историческая правда. На курсе «Азы программирования» Роботландского университета школьники 5–8 классов программируют трансляторы в средах Кукарачи и Корректора!

Предлагаем читателю последовать в эту не простую, но чрезвычайно интересную тему из области профессионального программирования (рис. 16.1).

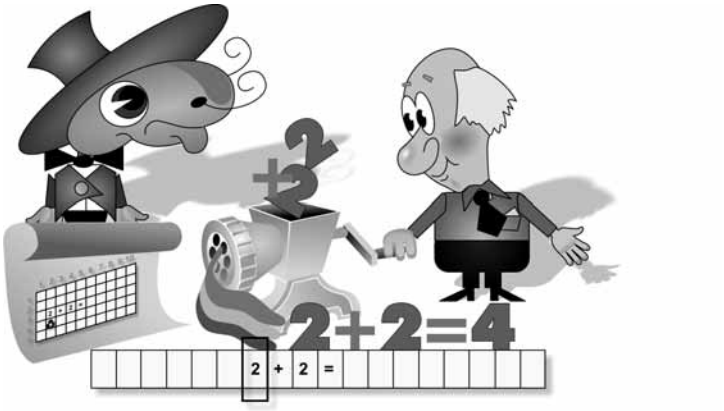

Рис. 16.1. Почему бы не написать для нас транслятор!

Глава 16

Язык Бэкуса-Наура

Ответы на вопросы и задания

1. Задано определение:

Определение 4

```
<имя> ::= <буква> | <имя><буква> | <имя><цифра>   (1)
<буква> ::= X | Y                                 (2)
<цифра> ::= 1 | 2                                 (3)
```

Проверить, являются ли приведённые далее объекты именами в смысле определения 4.

а) X; б) XY; в) X1; г) 2Y; д) XXXXXXX;
е) Y321; ж) 1212; з) X21; и) x21

Решение

Следующие объекты не являются именами в смысле определения 4:

- г) 2Y — имя не может начинаться с цифры;
- ж) 1212 — имя не может начинаться с цифры;
- и) x21 — объект «x» не является буквой.

Остальные объекты — имена в смысле определения 4. Приведём примеры двух доказательств.

Докажем, что объект X1 является именем в смысле определения 4. В силу (1) объект X1 — имя, если X — имя. Но X — буква в силу (2), а значит, имя в силу (1).

Докажем, что объект 2Y не является именем в смысле определения 4. Этот объект был бы именем по правилу (1), если бы именем был бы объект 2. Но последний объект именем не является: к нему нельзя применить правило (1).

Доказать следующее утверждение. Имя в смысле определения 4 — это любая последовательность из букв X, Y и цифр 1, 2, которая начинается с буквы.

Решение

Докажем это утверждение методом математической индукции.

Обозначим последовательность из n символов, каждый из которых является либо буквой X, Y, либо цифрой 1, 2 через F(n). При этом будем предполагать, что F(n) начинается с буквы (в смысле определения 4).

База индукции

F(1) = X, либо F(1) = Y

Объекты X и Y являются буквами по (2), а значит, именами по (1).

База индукции проверена.

Индуктивное предположение

Пусть F(n − 1) есть имя для некоторого n > 1.

Индуктивный переход

Представим F(n) при n > 1 как F(n − 1), следом за которым записана буква или цифра в смысле определения 4. Но такой объект есть имя по (1), т. к. F(n − 1) является именем по индуктивному предположению, а имя, вслед за которым написана буква или цифра, является именем по правилу (1).

Индуктивный переход доказан.

2. Задано определение:

Определение 5

```
<имя> ::= <буква> | <имя><буква> | <имя><цифра>   (1)
<буква> ::= <пусто> | X | Y                        (2)
<цифра> ::= 1 | 2                                  (3)
<пусто> ::=                                        (4)
```

Проверить, являются ли приведённые далее объекты именами в смысле определения 5.

а) X; б) XY; в) X1; г) 2Y;

д) XXXXXXX; е) Y321; ж) 1212

Решение

Все перечисленные в а) — ж) объекты являются именами в смысле определения 5. Докажем, например, что объект 2Y является именем.

В самом деле, этот объект по правилу (1) (<имя><буква>) будет именем, если именем будет объект «2», т. к. объект «Y» является буквой по правилу (2).

Объект «2» будет именем по правилу (1) (<имя><цифра>), если именем будет пустое место перед цифрой 2, т. к. объект «2» является цифрой по правилу (3).

Но пустое место есть буква по правилу (2), а значит, — имя по правилу (1).

Верны ли следующие утверждения.

а) Любая последовательность из символов X и 1 является именем в смысле определения 5.

Решение

Верно. Это легко можно доказать методом математической индукции.

Обозначим последовательность из n символов, каждый из которых либо X, либо 1 через F(n).

База индукции

F(1) = X, либо F(1) = 1

Объект X является буквой по (2), а значит, именем по (1).

Объект «1» можно рассматривать как пустое место, за которым написана цифра 1. Так как объект «пусто» является буквой, то объект «1» является именем по правилу (1).

База индукции проверена.

Индуктивное предположение

Пусть F(n − 1) есть имя для некоторого n > 1.

Индуктивный переход

Представим F(n) при n > 1 как F(n − 1), следом за которым записана буква — X или цифра 1. Но такой объект есть имя по (1), т. к. F(n − 1) является именем по индуктивному предположению, «X» — буква по (2), а «1» — цифра по (3).

б) Множество таких имён совпадает с множеством имён, задаваемых определением 5.

Решение

Нет, например, именем по определению 5 является объект «Y».

3. Задано определение:

Определение 6

```
<выражение> ::= <число> |(<выражение>) | <число> + <выражение>
<число> ::= 20 | 30
```

Проверить, являются ли приведённые далее объекты выражениями в смысле определения 6.

а) 2; б) 20; в) 2030; г) 20 + 30;

д) (20); е) (20 + 30); ж) 20 + (30 + 30); з) 20 +;

и) + 30; к) (20 + (30 + (20 + 20)))

Ответ.

Записи:

б) 20, г) 20 + 30, д) (20), е) (20 + 30),

ж) 20 + (30 + 30), к) (20 + (30 + (20 + 20)))

являются выражениями в смысле определения 6.

4. Запишите следующее определение при помощи языка Бэкуса-Наура.

Пример — это несколько цифр, соединённых между собой знаком «–».

Решение

```
<пример> ::= <цифра> - <цифра>|<пример> - <цифра>
<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

5. Запишите при помощи языка Бэкуса-Наура определение натурального числа.

Решение

```
<натуральное число> ::= <ненулевая цифра>|
                        <натуральное число ><цифра>
<ненулевая цифра> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<цифра> ::= 0 | <ненулевая цифра>
```

6. Запишите при помощи языка Бэкуса-Наура определение целого числа.

Решение

Вариант 1 (ведущие нули разрешены)

```
<целое число> ::= <целое без знака>|<знак><целое без знака>
```

```
<целое без знака> ::= <цифра>|< целое без знака><цифра>
<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<знак> ::= + | -
```

Вариант 2 (ведущие нули запрещены)

```
<целое число> ::= <целое без знака>|<знак><целое без знака>
<целое без знака> ::= <ненулевая цифра>|<целое без знака><цифра>
<ненулевая цифра> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<цифра> ::= 0 |<ненулевая цифра>
<знак> ::= + | -
```

7. Задано определение:

Определение 7

```
<выражение> ::= <терм> | <выражение> + <терм>
<терм> ::= <первичный> | <терм> * <первичный>
<первичный> ::= X | Y | Z
```

Проверить, являются ли приведённые далее объекты выражениями в смысле определения 7.

а) Z; б) X * Y + Z; в) X + Y + Z; г) (X + Y) * Z;
д) X * X; е) Y + Y * Y + Y; ж) XY; з) + X; и) Z +

Ответ.

Записи:

а) Z, б) X * Y + Z, в) X + Y + Z, д) X * X, е) Y + Y * Y + Y

являются выражениями в смысле определения 7.

Глава 17

Кукарача и лексический анализ выражений

Задачи на лексический анализ выражений занимают в информатике важное место.

Описанный в книге ученика способ решения применяется достаточно часто. При этом диаграмма переходов записывается в виде двухмерного массива (таблицы). Покажем, как это делается при программировании на языке Си.

Определение 1

```
<число> ::= <целое> | <дробь>
<целое> ::= <цифра> | <целое><цифра>
<дробь> ::= <целое>/<целое>
<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Таблица переходов, соответствующая этому определению, имеет вид (табл. 17.1).

Таблица 17.1

Предыдущее состояние		**Класс нового символа**			
		цифра	**/**	**пусто**	**остальное**
		0	1	2	3
(вход)	0	1	4	4	4
(целое)	1	1	2	stop	4
(дробь?)	2	3	4	4	4
(дробь)	3	3	4	stop	4
(ошибка)	4	stop	stop	stop	stop

На Си эта таблица записывается в виде массива `diagramma`:

```
int diagramma [5][4] =
{
  {1,    4,    4,    4},
  {1,    2,    stop, 4},
  {3,    4,    4,    4},
  {3,    4,    stop, 4},
  {stop, stop, stop, stop}
};
```

Элемент `diagramma[i][j] = m` показывает, что стрелка на диаграмме, выходящая из состояния i при обнаружении символа с номером j, приводит в состояние m, кроме значения m = stop. В последнем случае анализатор должен выдать в качестве результата число i (или соответствующий ему текст — название состояния).

Когда массив переходов задан, анализ выражения может быть выполнен при помощи такого кода:

```
int i = 0;
int j = diagramma[i][NextSymbolType()];
while (j != stop)
{                                                    (*)
  i = j;
  j = diagramma[i][NextSymbolType()];
}
```

Функция `NextSymbolType()` читает следующий символ входной записи и возвращает класс этого символа, т. е. в соответствии с таблицей переходов выдаёт:

- 0, если символ есть цифра;
- 1, если символ есть «/»;
- 2, если символ есть пусто;
- 3 в остальных случаях.

Анализатор (*) совершенно не зависит от вида определения 1. При замене определения другим (какой угодно сложности) лексический анализатор (*) останется прежним и будет содержать те же 7 строк программного кода. Меняться будет только массив `diagramma`.

В «большой» информатике проблема та же, что и в «малой»: нужно правильно составить диаграмму переходов по заданному определению.

Приведённый ранее код демонстрирует технику, при которой управление работой программы выполняют данные (массив `diagramma`), а не код (он инвариантен по отношению к структуре проверяемой записи).

Решение задач

17.5. Задачи

1. Введём определение числа:

Определение 2

```
<число> ::= <цифра> | <число><цифра>
<цифра> ::= 0 | 1
```

Напишите программу для Кукарачи, которая проверяет, является ли запись на его поле числом в смысле определения 2. В начальный момент исполнитель расположен перед кубиками с записью во второй строке (рис. 17.1).

Рис. 17.1

Если проверяемая запись — число, установить исполнитель в конец записи (рис. 17.2).

Рис. 17.2

Если запись — не число, поставить исполнитель в третью строку поля (рис. 17.3).

Рис. 17.3

Решение

Таблица переходов представлена табл. 17.2.

Таблица 17.2

Состояние	Следующий символ		
	0, 1	пусто	остальное
(вход)	(число)	(ошибка)	(ошибка)
(число)	(число)	(ответ_число)	(ошибка)
(ответ_число)			
(ошибка)			

Описание состояний

В каждом состоянии анализируется следующий символ записи и, в зависимости от его значения, принимается решение о переходе в новое состояние.

(вход)

Это состояние входное для алгоритма. Алгоритм анализирует первый символ записи.

Если символ — цифры 0 или 1, алгоритм переходит в состояние (число).

Любой другой символ — переход в состояние (ошибка).

(число)

Алгоритм анализирует очередной символ записи.

Если символ — цифры 0 или 1, алгоритм остаётся в состоянии (число).

Если символ — **ПУСТО**, алгоритм переходит в состояние (ответ_число).

Любой другой символ — переход в состояние (ошибка).

Программа

```
ЭТО вход                              // Состояние (вход)
  шаг                                 // Смотрим следующий символ
  ЕСЛИ        0 ТО число
  ИНАЧЕ ЕСЛИ 1 ТО число
  ИНАЧЕ            ошибка
КОНЕЦ

ЭТО число                             // Состояние (число)
  шаг                                 // Смотрим следующий символ
  ЕСЛИ        ПУСТО  ТО ответ_число
  ИНАЧЕ ЕСЛИ 0  ТО      число
```

```
  ИНАЧЕ ЕСЛИ 1  ТО      число
  ИНАЧЕ                 ошибка
КОНЕЦ

ЭТО ошибка
  ПОКА НЕ ПУСТО шаг                 // Перемещение остатка записи
  ВНИЗ                              // В положение "ошибка"
КОНЕЦ

ЭТО ответ_число
  ВВЕРХ
КОНЕЦ

ЭТО шаг                             // Толкнуть следующий
  ВНИЗ ВПРАВО ВВЕРХ                 // символ
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 17.3.

Таблица 17.3

Начальное состояние среды	Комментарий	Ожидаемый результат
¤□	Пустая запись	□ ¤
¤0	Правильная запись	0¤
¤01101	Правильная запись	01101¤
¤2	Запись с ошибками	2 ¤
¤02	Запись с ошибками	02 ¤
¤011020101	Запись с ошибками	011020101 ¤

2. Решить задачу 1, но если запись не является числом, установить исполнитель под первым неверным символом в записи (рис. 17.4).

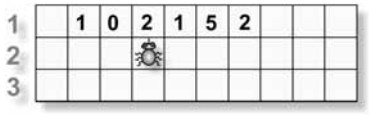

Рис. 17.4

Если запись — число, установить исполнитель за последним её символом (рис. 17.5).

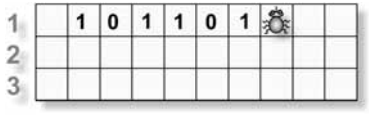

Рис. 17.5

Решение

В решении задачи 1 изменится только процедура `ошибка`:

```
ЭТО ошибка
  шаг                              // Перемещение остатка записи
  ЕСЛИ НЕ ПУСТО ТО ошибка          // в первую строку
  ВЛЕВО                            // Возврат к месту ошибки
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 17.4.

Таблица 17.4

Начальное состояние среды	Комментарий	Ожидаемый результат
¤□	Пустая запись	□ ¤
¤0	Правильная запись	0¤
¤01101	Правильная запись	01101¤
¤2	Запись с ошибками	2 ¤
¤02	Запись с ошибками	02 ¤
¤011020101	Запись с ошибками	011020101 ¤

3. Введём определение числа:

Определение 3

```
<число>   ::= <целое> | <дробное>
<целое>   ::= <цифра> | <целое><цифра>
<дробное> ::= .<целое> | <целое>. | <целое>.<целое>
<цифра>   ::= 0 | 1
```

Напишите программу для Кукарачи, которая проверяет, является ли запись на его поле числом в смысле определения 3. В начальный момент исполнитель расположен перед кубиками с записью во второй строке (рис. 17.6).

Рис. 17.6

Если проверяемая запись — число, установить исполнитель в конец записи (рис. 17.7).

Рис. 17.7

Если запись — не число, поставить исполнитель под первым неверным символом (рис. 17.8).

Рис. 17.8

Примеры результатов выполнения программы представлены на рис. 17.9–17.12).

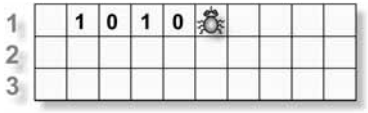

Рис. 17.9

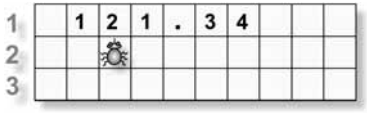

Рис. 17.10

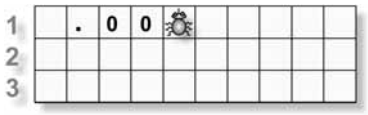

Рис. 17.11

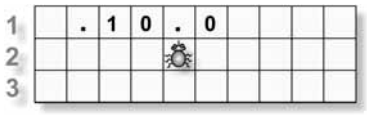

Рис. 17.12

Решение

Таблица переходов представлена в табл. 17.5.

Таблица 17.5

Состояние	Следующий символ			
	0, 1	.	пусто	остальное
(вход)	(целое)	(дробное?)	(ошибка)	(ошибка)
(целое)	(целое)	(дробное)	(ответ_число)	(ошибка)
(дробное?)	(дробное)	(ошибка)	(ошибка)	(ошибка)
(дробное)	(дробное)	(ошибка)	(ответ_число)	(ошибка)
(ответ_число)				
(ошибка)				

Программа

```
ЭТО вход                              // Состояние (вход)
  шаг                                 // Смотрим следующий символ
  ЕСЛИ        0 ТО целое
  ИНАЧЕ ЕСЛИ  1 ТО целое
  ИНАЧЕ ЕСЛИ  . ТО дробное?
  ИНАЧЕ           ошибка
КОНЕЦ

ЭТО целое                             // Состояние (целое)
  шаг                                 // Смотрим следующий символ
  ЕСЛИ       ПУСТО  ТО ответ_число
  ИНАЧЕ ЕСЛИ 0   ТО    целое
  ИНАЧЕ ЕСЛИ 1   ТО    целое
  ИНАЧЕ ЕСЛИ .   ТО    дробное
  ИНАЧЕ                ошибка
КОНЕЦ
```

```
ЭТО дробное?                          // Состояние (дробное?)
  шаг                                 // Смотрим следующий символ
  ЕСЛИ        0  ТО      дробное
  ИНАЧЕ ЕСЛИ  1  ТО      дробное
  ИНАЧЕ                  ошибка
КОНЕЦ

ЭТО дробное                           // Состояние (дробное)
  шаг                                 // Смотрим следующий символ
  ЕСЛИ        ПУСТО  ТО ответ_число
  ИНАЧЕ ЕСЛИ  0  ТО      дробное
  ИНАЧЕ ЕСЛИ  1  ТО      дробное
  ИНАЧЕ                  ошибка
КОНЕЦ

ЭТО ошибка
  шаг                                 // Перемещение остатка записи
  ЕСЛИ НЕ ПУСТО ТО ошибка             // в первую строку
  ВЛЕВО                               // Возврат к месту ошибки
КОНЕЦ

ЭТО ответ_число
  ВВЕРХ
КОНЕЦ

ЭТО шаг                               // Толкнуть следующий
  ВНИЗ ВПРАВО ВВЕРХ                   // символ
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 17.6.

Таблица 17.6

Правильные записи	Записи с ошибками
.0	Пустая запись
1.	.
01	2.1
1.0	1.1.1
11.00	

4. Введём определение числа:

Определение 4

```
<число>    ::= <целое> | <дробь>
<целое>    ::= <цифра> | <целое><цифра>
<дробь>    ::= <простая> | <десятичная>
<простая>  ::= <целое>/<целое>
<десятичная> ::= .<целое> | <целое>. | <целое>.<целое>
<цифра>    ::= 0 | 1
```

Напишите программу для Кукарачи, которая проверяет, является ли запись на его поле числом в смысле определения 4. В начальный момент исполнитель расположен перед кубиками с записью во второй строке (рис. 17.13).

Рис. 17.13

Если проверяемая запись — число, установить исполнитель в конец записи (рис. 17.14).

Рис. 17.14

Если запись — не число, поставить исполнитель под первым неверным символом (рис. 17.15).

Рис. 17.15

Решение

Таблица переходов представлена в табл. 17.7.

Таблица 17.7

Состояние	Следующий символ				
	0, 1	**.**	**/**	**пусто**	**остальное**
(вход)	(целое)	(десятичная?)	(ошибка)	(ошибка)	(ошибка)
(целое)	(целое)	(десятичная)	(простая?)	(ответ_число)	(ошибка)
(десятичная?)	(десятичная)	(ошибка)	(ошибка)	(ошибка)	(ошибка)
(десятичная)	(десятичная)	(ошибка)	(ошибка)	(ответ_число)	(ошибка)
(простая?)	(простая)	(ошибка)	(ошибка)	(ошибка)	(ошибка)
(простая)	(простая)	(ошибка)	(ошибка)	(ответ_число)	(ошибка)
(ответ_число)					
(ошибка)					

Программа

```
ЭТО вход                              // Состояние (вход)
  шаг                                 // Смотрим следующий символ
  ЕСЛИ        0 ТО целое
  ИНАЧЕ ЕСЛИ  1 ТО целое
  ИНАЧЕ ЕСЛИ  . ТО десятичная?
  ИНАЧЕ             ошибка
КОНЕЦ
ЭТО целое                             // Состояние (целое)
  шаг                                 // Смотрим следующий символ
  ЕСЛИ        ПУСТО  ТО ответ_число
  ИНАЧЕ ЕСЛИ  0   ТО     целое
  ИНАЧЕ ЕСЛИ  1   ТО     целое
  ИНАЧЕ ЕСЛИ  .   ТО     десятичная
  ИНАЧЕ ЕСЛИ  /   ТО     простая?
  ИНАЧЕ                  ошибка
КОНЕЦ

ЭТО десятичная?                       // Состояние (десятичная?)
  шаг                                 // Смотрим следующий символ
  ЕСЛИ        0   ТО     десятичная
  ИНАЧЕ ЕСЛИ  1   ТО     десятичная
```

```
  ИНАЧЕ                ошибка
КОНЕЦ

ЭТО десятичная                      // Состояние (десятичная)
  шаг                               // Смотрим следующий символ
  ЕСЛИ       ПУСТО  ТО ответ_число
  ИНАЧЕ ЕСЛИ 0  ТО     десятичная
  ИНАЧЕ ЕСЛИ 1  ТО     десятичная
  ИНАЧЕ                ошибка
КОНЕЦ

ЭТО простая?                        // Состояние (простая?)
  шаг                               // Смотрим следующий символ
  ЕСЛИ       0  ТО     простая
  ИНАЧЕ ЕСЛИ 1  ТО     простая
  ИНАЧЕ                ошибка
КОНЕЦ

ЭТО простая                         // Состояние (простая)
  шаг                               // Смотрим следующий символ
  ЕСЛИ       ПУСТО  ТО ответ_число
  ИНАЧЕ ЕСЛИ 0  ТО     простая
  ИНАЧЕ ЕСЛИ 1  ТО     простая
  ИНАЧЕ                ошибка
КОНЕЦ

ЭТО ошибка
  шаг                               // Перемещение остатка записи
  ЕСЛИ НЕ ПУСТО ТО ошибка           // в первую строку
  ВЛЕВО                             // Возврат к месту ошибки
КОНЕЦ

ЭТО ответ_число
  ВВЕРХ
КОНЕЦ

ЭТО шаг                             // Толкнуть следующий
  ВНИЗ ВПРАВО ВВЕРХ                 // символ
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 17.8.

Таблица 17.8

Правильные записи	Записи с ошибками
`.0`	Пустая запись
`1.`	`.`
`01`	`2.1`
`1.0`	`1.1.1`
`11.00`	`/`
`1/0`	`/1`
`0/10`	`1/`
	`1/1/1`
	`1.1/1`

5. Введём определение выражения:

Определение 5

```
<выражение> ::= <число> | <выражение> + <число>
<число>     ::= <цифра> | <число><цифра>
<цифра> ::= 0 | 1
```

Напишите программу для Кукарачи, которая проверяет, является ли запись на его поле выражением в смысле определения 5. В начальный момент исполнитель расположен перед кубиками с записью во второй строке (рис. 17.16).

Рис. 17.16

Если проверяемая запись — выражение, установить исполнитель в конец записи (рис. 17.17).

Рис. 17.17

Если запись — не выражение, поставить исполнитель под первым неверным символом (рис. 17.18).

Рис. 17.18

Решение

Таблица переходов представлена табл. 17.9.

Таблица 17.9

Состояние	Следующий символ			
	0, 1	**+**	**пусто**	**остальное**
(вход)	(число)	(ошибка)	(ошибка)	(ошибка)
(число)	(число)	(знак_плюс)	(ответ_выражение)	(ошибка)
(знак_плюс)	(число)	(ошибка)	(ошибка)	(ошибка)
(ответ_выражение)				
(ошибка)				

Описание состояний

(вход)

Это состояние входное для алгоритма. Алгоритм анализирует первый символ записи.

Если символ — цифры 0 или 1, алгоритм переходит в состояние (число).

Любой другой символ — переход в состояние (ошибка).

(число)

В этом состоянии алгоритм остаётся, пока не закончится цепочка цифр. Символ `пусто` успешно завершает анализ (выражение по определению обязано заканчиваться числом), знак «+» переключает алгоритм в состояние (знак_плюс), любой другой символ означает ошибку.

(знак_плюс)

В это состояние алгоритм попадает после обнаружения знака «+». Следующим символом в правильной записи может быть только цифра,

и при её обнаружении алгоритм возвращается в состояние (число), чтобы продолжить проверку записи.

Программа

```
ЭТО вход                              // Состояние (вход)
  шаг                                 // Смотрим следующий символ
  ЕСЛИ        0 ТО число
  ИНАЧЕ ЕСЛИ 1 ТО число
  ИНАЧЕ           ошибка
КОНЕЦ

ЭТО число                             // Состояние (число)
  шаг                                 // Смотрим следующий символ
  ЕСЛИ       ПУСТО  ТО ответ_выражение
  ИНАЧЕ ЕСЛИ 0  ТО     число
  ИНАЧЕ ЕСЛИ 1  ТО     число
  ИНАЧЕ ЕСЛИ +  ТО     знак_плюс
  ИНАЧЕ                ошибка
КОНЕЦ

ЭТО знак_плюс                         // Состояние (знак_плюс)
  шаг                                 // Смотрим следующий символ
  ЕСЛИ        0  ТО     число
  ИНАЧЕ ЕСЛИ 1  ТО     число
  ИНАЧЕ                ошибка
КОНЕЦ

ЭТО ошибка
  шаг                                 // Перемещение остатка записи
  ЕСЛИ НЕ ПУСТО ТО ошибка             // в первую строку
  ВЛЕВО                               // Возврат к месту ошибки
КОНЕЦ

ЭТО ответ_выражение
  ВВЕРХ
КОНЕЦ

ЭТО шаг                               // Толкнуть следующий
  ВНИЗ ВПРАВО ВВЕРХ                   // символ
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 17.10.

Таблица 17.10

Правильные записи	Записи с ошибками
`0`	Пустая запись
`10+00`	`+`
`1+01+011`	`2`
	`+1`
	`1+`
	`1+1+`
	`+1+1+1`

Глава 18

Ах уж эта рекурсия!

Удивительная рекурсия

Часто рекурсия сильно упрощает программирование, делая программу удивительно короткой.

Но всё же основным инструментом программиста остаётся цикл `пока`. Эта управляющая структура нагляднее рекурсии, проще в использовании и часто эффективнее при выполнении. Поэтому, если задача решается при помощи цикла `пока`, не надо стрелять из пушки по воробьям!

Сказанное совсем не означает, что рекурсия находится на задворках человеческой мысли. Рекурсия — мощный инструмент математики и, конечно, информатики. Часто рекурсивные описания настолько легки и красивы, насколько тяжелы и громоздки попытки обойтись циклами.

Посмотрите, например, как лаконично описывается при помощи рекурсии последовательность чисел Фибоначчи:

$f(1) = 1$

$f(2) = 1$

$f(n) = f(n - 1) + f(n - 2)$, для всех $n > 2$

Напоминание о числах Фибоначчи

Числа Фибоначчи определяются следующей бесконечной последовательностью целых чисел: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …

Каждый элемент этой последовательности, начиная с третьего, равен сумме двух непосредственно предшествующих элементов. Первые два члена последовательности равны единице.

Эти числа являются решением задачи, которую итальянский математик Фибоначчи сформулировал в 1202 году:

«Каждый месяц самка из пары кроликов приносит двух детёнышей (самца и самку). Через два месяца новая самка сама приносит пару кроликов. Нужно

найти число кроликов в конце года, если в начале года была одна новорождённая пара и в течение этого года кролики не умирали».

Ответ: к концу года число пар кроликов достигнет числа f(12) = 144, т. е. поголовье возрастет до 288 кроличьих единиц.

Несколько примеров рекурсивных описаний.

1. **Факториал.**

   $0! = 1$

   $n! = n \cdot (n-1)!$

2. **Сумма последовательности.**

   Пусть дана бесконечная последовательность чисел

   $a_1, a_2, \ldots, a_n, \ldots$

   Если обозначить через S(n) сумму n членов этой последовательности, то

   $S(1) = a_1$

   $S(k) = S(k-1) + ak$

3. **Бинарное дерево.**

   Объект O, показанный на рис. 18.1, назовём бинарным деревом.

Рис. 18.1. Бинарное дерево

Тогда если b1 и b2 — бинарные деревья, то объект, показанный на рис. 18.2, по определению тоже является бинарным деревом.

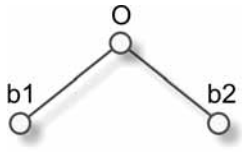

Рис. 18.2. Бинарное дерево

4. **Решение задачи о ханойских башнях.**

   Пусть H(n, i, j, k) описывает алгоритм переноса n колец с i-го стержня на j-ый, используя k-ый стержень как рабочий (i, j, k принимают значения из множества {1, 2, 3} и различны между собой) (рис. 18.3).

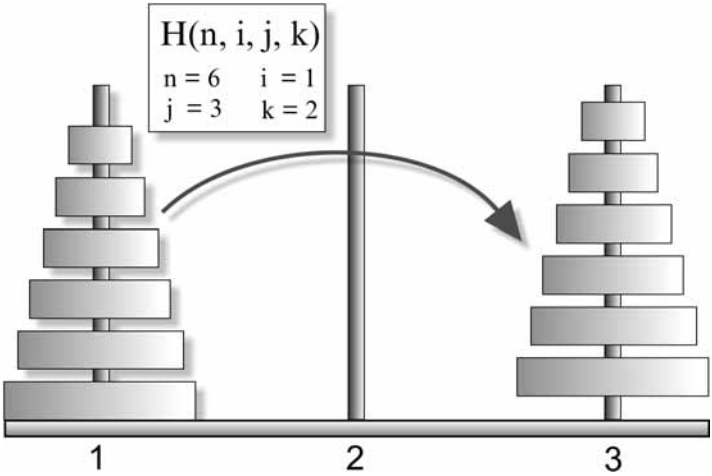

Рис. 18.3. Задача о ханойских башнях

Решение можно изобразить в виде рекурсивной схемы, показанной на рис. 18.4.

$$H(1, i, j, k) = i \longrightarrow j$$

$$H(n, i, j, k) = \begin{cases} 1.\ H(n-1, i, k, j) \\ 2.\ i \longrightarrow j \\ 3.\ H(n-1, k, j, i) \end{cases}$$

Рис. 18.4. Рекурсивная схема решения задачи о ханойских башнях

Замечание

Операция «стрелка» введена для обозначения переноса кольца с одного стержня на другой. Например, запись 1 → 3 обозначает перенос кольца с первого стержня на третий.

Подводные камни рекурсии

Рекурсия таит в себе подводные камни.

Первый камень

Использовать рекурсию разумно тогда, когда рекурсивна сама природа задачи. Искусственно построенная рекурсия (как альтернатива циклу) редко бывает оправдана. Например, для суммирования членов последовательности лучше применить цикл, а не рекурсию.

В чём заключается опасность рекурсии при машинных вычислениях? Дело в том, что при каждом рекурсивном вызове расходуется дополнительно часть машинной памяти (в ней сохраняются адреса возврата и локальные переменные процедур, если язык допускает использование переменных). При «глубоких» рекурсивных вызовах вся доступная память компьютера может быстро исчерпаться.

Но даже в некоторых естественно-рекурсивных задачах рекурсию использовать не всегда разумно. В качестве примера рассмотрим упоминаемую ранее последовательность чисел Фибоначчи.

Каждое обращение к f(n) при n > 2 приводит к двум рекурсивным обращениям. Посмотрите, как быстро растёт число вызовов процедуры f при рекурсивном вычислении (рис. 18.5).

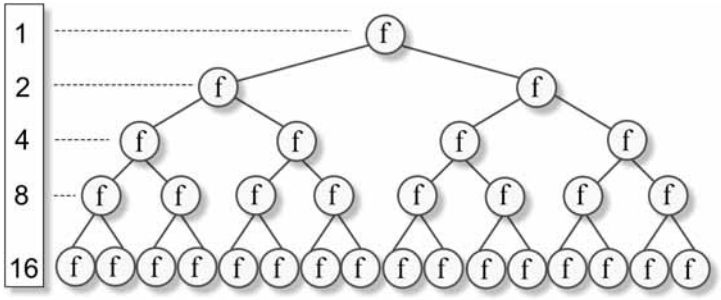

Рис. 18.5. Рекурсивное нахождение числа Фибоначчи

Число рекурсивных вызовов растёт как степень двойки, т. е. быстрее, чем размножаются кролики! Заметим, правда, что «степенной» рост рекурсивного дерева f(n) не означает, что число вершин будет измеряться степенью двойки. Дерево не получится симметричным, некоторые ветви будут заканчиваться раньше других, и в итоге общее число вершин будет соответствовать именно числу Фибоначчи.

Числа Фибоначчи лучше вычислять при помощи цикла **ПОКА**:

```
ЭТО f(n)
  i:= 1          // i - счётчик чисел Фибоначчи
  x:= 1          // x - сумматор
  y:= 0          // y - вспомогательная переменная
  ПОКА i < n
  {
    x:= x + y    // суммируем два предыдущих значения
    y:= x - y    // первое из них запоминаем в y
    i:= i + 1
  }
  ВЕРНУТЬ x
КОНЕЦ
```

Второй камень

Рекурсия как замена команде безусловного перехода **GOTO** ещё хуже, чем использование самого **GOTO**.

Рассматривать рекурсию как переход в начало процедуры неправильно: ведь рекурсия это не переход, а выполнение нового экземпляра процедуры.

Третий камешек (в адрес косвенной рекурсии)

Прямая рекурсия предпочтительнее косвенной. Косвенная рекурсия — это замаскированная форма прямой, и маскировка ухудшает читабельность программы, её отладку и сопровождение.

Классификация типов рекурсии

Прямая и косвенная рекурсия

В математике и информатике рекурсивной называют такую функцию или процедуру, которая при работе обращается к себе самой, прямо или косвенно. Соответственно говорят о прямой и косвенной рекурсии. При прямой рекурсии процедура вызывает себя в своём собственном теле, например:

```
ЭТО прямая
  ...
  ЕСЛИ ... ТО прямая
  ...
КОНЕЦ
```

Косвенная рекурсия образуется цепочкой процедур, и эта цепочка замыкается в рекурсивное кольцо, например:

```
ЭТО процедура0
  ...
  процедура1
  ...
КОНЕЦ

ЭТО процедура1
  ...
  процедура2
  ...
КОНЕЦ

ЭТО процедура2
  ...
  процедура0
  ...
КОНЕЦ
```

В примере цепочка `процедура0` → `процедура1` → `процедура2` → `процедура0` образует косвенную рекурсию. При этом `процедура0` является рекурсивной, т. к. вызывает сама себя. Правда, этот вызов происходит через обращение к процедурам `процедура1` и `процедура2`. Понятно, что каждая из процедур рекурсивной цепочки (и `процедура1`, и `процедура2`) тоже являются рекурсивными.

Сама по себе косвенная рекурсия не содержит новых идей. Это другая форма записи прямой рекурсии, если, конечно, промежуточные процедуры не содержат дополнительных рекурсий.

Пример сложного переплетения косвенных рекурсий можно обнаружить в программе лексического анализатора. Однако эта сложность скрадывается предварительным построением диаграммы (таблицы) переходов.

Когда диаграмма построена, мы формально переводим её в программу, даже не задумываясь о том, сколько сложных косвенных рекурсий при этом монтируется в простой с виду программный код.

Это то самое исключение из правил, когда косвенная рекурсия не затрудняет понимание алгоритма работы программы.

Рассмотрим два примера.

Определение 1

```
<выражение> ::= <число> | <выражение> + <число>
<число>   ::= 0 | 1 | 2
```

Диаграмма переходов, соответствующая определению 1, показана на рис. 18.6.

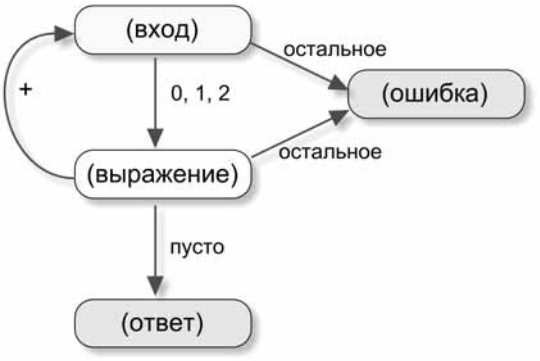

Рис. 18.6

На этой схеме изображены 4 состояния-процедуры: `ответ`, `выражение`, `вход` и `ошибка`.

Процедура `ответ` рекурсивной не является.

Процедура `ошибка` на схеме рекурсивной не показана, хотя алгоритм её работы содержит прямую рекурсию с отложенной командой для перемещения исполнителя к месту обнаруженной ошибки.

Процедуры `выражение` и `вход` содержат вызов друг друга, значит, образуют цепочку косвенной рекурсии.

Определение 2

```
<выражение> ::= <число> |
                <выражение> - <выражение> |
                <выражение> + <выражение> |
                (<выражение>)
<число>   ::=  0 |1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Диаграмма переходов, соответствующая определению 2, показана на рис. 18.7 (на диаграмме не показаны переходы в состояние «ошибка» и не учтена проверка на баланс скобок на уровне всего выражения).

Посмотрите, какое здесь переплетение рекурсий!

Например, процедура `вход` содержит прямую рекурсию и одновременно две косвенных: `вход число вход` и `вход число скобка вход`.

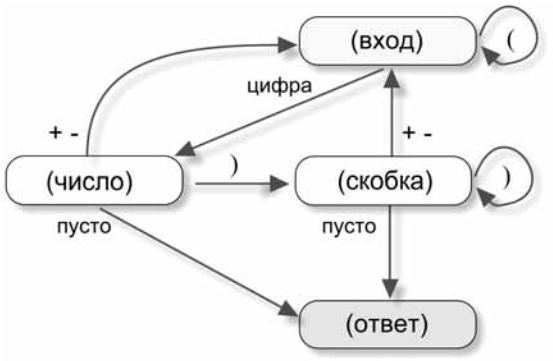

Рис. 18.7

Терминальные и нетерминальные рекурсии

Терминал — это оконечное устройство, концевик. В частности, терминалом называют монитор (часто вместе с клавиатурой) как оконечное устройство компьютера для диалога с пользователем. Раньше, когда использовали большие ЭВМ типа ЕС или БЭСМ, терминалы (монитор + клавиатура) выносили в отдельные залы, где несколько пользователей одновременно работали на своих терминалах с одной ЭВМ. Иногда такой терминал имел собственный процессор и небольшую память: получалось, что терминал — это небольшой компьютер. Но это не мешало ему быть терминалом, т. к. он работал не самостоятельно, а служил оконечным устройством другой ЭВМ.

Терминальная рекурсия — это рекурсия-концевик, т. е. такой рекурсивный вызов, после которого других команд в процедуре выполняться не должно. Понятно, что терминальная рекурсия может быть как прямой, так и косвенной.

Посмотрите на процедуры `вход` и `выражение`:

```
ЭТО вход
  ВПРАВО
  ЕСЛИ        0 ТО выражение
  ИНАЧЕ ЕСЛИ 1 ТО добавить1
  ИНАЧЕ ЕСЛИ 2 ТО добавить2
  ИНАЧЕ ошибка
КОНЕЦ

ЭТО выражение
  ВПРАВО
  ЕСЛИ        +      ТО вход
```

```
  ИНАЧЕ ЕСЛИ ПУСТО ТО ответ
  ИНАЧЕ ошибка
КОНЕЦ
```

В процедуре `вход` косвенная терминальная рекурсия обеспечивается вызовом процедуры `выражение`.

Обратите внимание на то, что запись:

```
ЕСЛИ       0 ТО выражение
ИНАЧЕ ЕСЛИ 1 ТО добавить1
ИНАЧЕ ЕСЛИ 2 ТО добавить2
ИНАЧЕ ошибка
```

является одной условной командой в отличие от записи:

```
ЕСЛИ 0 ТО выражение
ЕСЛИ 1 ТО добавить1
ЕСЛИ 2 ТО добавить2
ИНАЧЕ ошибка
```

Последняя запись содержит три команды и, если бы именно так было записано в процедуре `вход`, рекурсивный вызов был бы уже нетерминальным, а следовательно, процедура `вход` работала бы совсем по-другому. Она содержала бы две «отложенные» в рекурсии команды:

```
ЕСЛИ 1 ТО добавить1
ЕСЛИ 2 ТО добавить2 ИНАЧЕ ошибка
```

Типичный пример прямой нетерминальной рекурсии содержится в процедуре `ошибка`:

```
ЭТО ошибка
  ВПРАВО
  ЕСЛИ ПУСТО
    ТО    ответ_ошибка
    ИНАЧЕ ошибка
  ВЛЕВО
КОНЕЦ
```

Нетерминальная рекурсия порождает отложенные команды, рекурсивные «пружинки».

Конечная и бесконечная рекурсия

Как цикл может быть конечным и бесконечным, так и рекурсия (это ведь тоже повторение!) может быть бесконечной или нет.

Для того чтобы бесконечных повторений (циклических или рекурсивных) не было, нужна проверка на окончание.

Бесконечная прямая терминальная рекурсия

```
ЭТО поход
  ВПРАВО
  поход
КОНЕЦ
```

Кукарача кричит «Не могу» на правом краю поля.

Конечная прямая терминальная рекурсия

```
ЭТО поиск
  ВПРАВО
  ЕСЛИ ПУСТО ТО поиск
КОНЕЦ
```

Кукарача находит кубик с символом и останавливается.

Конечная прямая нетерминальная рекурсия

```
ЭТО поиск
  ВПРАВО
  ЕСЛИ ПУСТО ТО поиск
  ВЛЕВО
КОНЕЦ
```

Кукарача находит кубик с символом и возвращается на прежнее место.

Решение задач

18.2. Задачи

1. Введём определение выражения.

Определение 1

```
<выражение> ::= <число> + <число> | <выражение> + <число>
<число>   ::= <цифра> | <число><цифра>
<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Напишите программу для Кукарачи, которая проверяет, является ли запись на его поле выражением в смысле определения 1. В начальный

момент исполнитель расположен перед кубиками с записью во второй строке (рис. 18.8).

Рис. 18.8

Если проверяемая запись — выражение, установить исполнитель в конец записи (рис. 18.9).

Рис. 18.9

Если запись — не выражение, поставить исполнитель под первым неверным символом (рис. 18.10).

Рис. 18.10

Решение

Таблица переходов представлена табл. 18.1.

Таблица 18.1

Состояние	Следующий символ			
	цифра	**+**	**пусто**	**остальное**
(вход)	(первое_число)	(ошибка)	(ошибка)	(ошибка)
(первое_число)	(первое_число)	(плюс)	(ошибка)	(ошибка)
(плюс)	(выражение)	(ошибка)	(ошибка)	(ошибка)
(выражение)	(выражение)	(плюс)	(ответ_ выражение)	(ошибка)
(ответ_выражение)				
(ошибка)				

Описание состояний

В каждом состоянии анализируется следующий символ записи и, в зависимости от его значения, принимается решение о переходе в новое состояние.

(вход)

Это состояние входное для алгоритма. Алгоритм анализирует первый символ записи.

Если символ — цифра, алгоритм переходит в состояние (первое_число).

Любой другой символ — переход в состояние (ошибка).

(первое_число)

Алгоритм анализирует очередной символ записи.

Если символ — цифра, алгоритм остаётся в состоянии (первое_число).

Если символ — «+», алгоритм переходит в состояние (плюс).

Любой другой символ — переход в состояние (ошибка).

(плюс)

Алгоритм анализирует очередной символ записи.

Если символ — цифра, алгоритм переходит в состояние (выражение).

Любой другой символ — переход в состояние (ошибка).

(выражение)

Алгоритм анализирует очередной символ записи.

Если символ — цифра, алгоритм остаётся в состоянии (выражение).

Если символ — «+», алгоритм переходит в состояние (плюс).

Если символ — **ПУСТО**, алгоритм переходит в состояние (ответ_выражение).

Любой другой символ — переход в состояние (ошибка).

Программа

```
ЭТО вход                          // Состояние (вход)
  шаг                             // Смотрим следующий символ
  ЕСЛИ  ЦИФРА ТО первое_число
  ИНАЧЕ          ошибка
КОНЕЦ


ЭТО первое_число                  // Состояние (первое_число)
```

```
  шаг                                  // Смотрим следующий символ
  ЕСЛИ       ЦИФРА  ТО первое_число
  ИНАЧЕ ЕСЛИ +      ТО плюс
  ИНАЧЕ                ошибка
КОНЕЦ

ЭТО плюс                               // Состояние (плюс)
  шаг                                  // Смотрим следующий символ
  ЕСЛИ ЦИФРА    ТО выражение
  ИНАЧЕ           ошибка
КОНЕЦ

ЭТО выражение                          // Состояние (выражение)
  шаг                                  // Смотрим следующий символ
  ЕСЛИ       ПУСТО  ТО ответ_выражение
  ИНАЧЕ ЕСЛИ ЦИФРА  ТО выражение
  ИНАЧЕ ЕСЛИ +      ТО плюс
  ИНАЧЕ                ошибка
КОНЕЦ

ЭТО ошибка
  шаг                                  // Перемещение остатка записи
  ЕСЛИ НЕ ПУСТО ТО ошибка              // в первую строку
  ВЛЕВО                                // Возврат к месту ошибки
КОНЕЦ

ЭТО ответ_выражение
  ВВЕРХ
КОНЕЦ

ЭТО шаг                                // Толкнуть следующий
  ВНИЗ ВПРАВО ВВЕРХ                    // символ
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 18.2.

Таблица 18.2

Правильные записи	Записи с ошибками
10+00	Пустая запись
1+01+011	0
	+
	2
	+1
	1+
	1+1+
	1++1
	+1+1+1

2. Введём определение узора:

Определение 2

```
<узор> ::= <простой>| <узор><фигура><простой>
<простой> ::= <простой1> | <простой2>
<простой1> ::= ! | <простой1>!
<простой2> ::= ? | <простой2>?
<фигура> ::= % | #
```

Напишите программу для Кукарачи, которая проверяет, является ли запись на его поле узором в смысле определения 2. В начальный момент исполнитель расположен перед кубиками с записью во второй строке (рис. 18.11).

Рис. 18.11

Если проверяемая запись — узор, установить исполнитель в конец записи (рис. 18.12).

Рис. 18.12

Если запись — не узор, поставить исполнитель под первым неверным символом (рис. 18.13).

Рис. 18.13

Решение

Таблица переходов представлена табл. 18.3.

Таблица 18.3

Состояние	Следующий символ					
	!	?	%	#	пусто	остальное
(вход)	(простой1)	(простой2)	(ошибка)	(ошибка)	(ошибка)	(ошибка)
(простой1)	(простой1)	(ошибка)	(вход)	(вход)	(узор)	(ошибка)
(простой2)	(ошибка)	(простой2)	(вход)	(вход)	(узор)	(ошибка)
(узор)						
(ошибка)						

Программа

```
ЭТО вход                                  // Состояние (вход)
  шаг                                     // Смотрим следующий символ
  ЕСЛИ        ! ТО простой1
  ИНАЧЕ ЕСЛИ ? ТО простой2
  ИНАЧЕ           ошибка
КОНЕЦ

ЭТО простой1                              // Состояние (простой1)
  шаг                                     // Смотрим следующий символ
  ЕСЛИ       ПУСТО  ТО ответ_узор
  ИНАЧЕ ЕСЛИ !      ТО простой1
  ИНАЧЕ ЕСЛИ %      ТО вход
  ИНАЧЕ ЕСЛИ #      ТО вход
  ИНАЧЕ       ошибка
```

```
КОНЕЦ

ЭТО простой2                          // Состояние (простой2)
  шаг                                 // Смотрим следующий символ
  ЕСЛИ        ПУСТО  ТО ответ_узор
  ИНАЧЕ ЕСЛИ ?       ТО простой2
  ИНАЧЕ ЕСЛИ %       ТО вход
  ИНАЧЕ ЕСЛИ #       ТО вход
  ИНАЧЕ       ошибка
КОНЕЦ

ЭТО ошибка
  шаг                                 // Перемещение остатка записи
  ЕСЛИ НЕ ПУСТО ТО ошибка             // в первую строку
  ВЛЕВО                               // Возврат к месту ошибки
КОНЕЦ

ЭТО ответ_узор
  ВВЕРХ
КОНЕЦ

ЭТО шаг                               // Толкнуть следующий
  ВНИЗ ВПРАВО ВВЕРХ                   // символ
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 18.4.

Таблица 18.4

Правильные записи	Записи с ошибками
!	Пустая запись
????	%
!!!%!!!!	#
??#!!!%????	#!!!
	%???
	!!!!????
	????%

3. Введём определение узора:

Определение 3

```
<узор> ::= <простой>| <узор><фигура><простой>
<простой>   ::= /\ | []
<фигура> ::= % | #
```

Напишите программу для Кукарачи, которая проверяет, является ли запись на его поле узором в смысле определения 3. В начальный момент исполнитель расположен перед кубиками с записью во второй строке (рис. 18.14).

Рис. 18.14

Если проверяемая запись — узор, установить исполнитель в конец записи (рис. 18.15).

Рис. 18.15

Если запись — не узор, поставить исполнитель под первым неверным символом (рис. 18.16).

Рис. 18.16

Решение

Таблица переходов представлена табл. 18.5.

Таблица 18.5

Состояние	Следующий символ							
	/	\	[	]	%	#	пусто	остальное
(вход)	(знак/)	(ош)	(знак[)	(ош)	(ош)	(ош)	(ош)	(ош)
(знак/)	(ош)	(простой)	(ош)	(ош)	(ош)	(ош)	(ош)	(ош)
(знак[)	(ош)	(ош)	(ош)	(простой)	(ош)	(ош)	(ош)	(ош)
(простой)	(ош)	(ош)	(ош)	(ош)	(вход)	(вход)	(узор)	(ош)
(узор)								
(ош)								

Программа

```
ЭТО вход                              // Состояние (вход)
  шаг                                 // Смотрим следующий символ
  ЕСЛИ       / ТО знак/
  ИНАЧЕ ЕСЛИ [ ТО знак[
  ИНАЧЕ          ошибка
КОНЕЦ

ЭТО знак/                             // Состояние (знак/)
  шаг                                 // Смотрим следующий символ
  ЕСЛИ  \ ТО простой
  ИНАЧЕ      ошибка
КОНЕЦ

ЭТО знак[                             // Состояние (знак[)
  шаг                                 // Смотрим следующий символ
  ЕСЛИ  ] ТО простой
  ИНАЧЕ      ошибка
КОНЕЦ

ЭТО простой                           // Состояние (простой)
  шаг                                 // Смотрим следующий символ
  ЕСЛИ       ПУСТО  ТО ответ_узор
  ИНАЧЕ ЕСЛИ % ТО     вход
```

```
  ИНАЧЕ ЕСЛИ #  ТО      вход
  ИНАЧЕ                 ошибка
КОНЕЦ

ЭТО ошибка
  шаг                                  // Перемещение остатка записи
  ЕСЛИ НЕ ПУСТО ТО ошибка              // в первую строку
  ВЛЕВО                                // Возврат к месту ошибки
КОНЕЦ

ЭТО ответ_узор
  ВВЕРХ
КОНЕЦ

ЭТО шаг                                // Толкнуть следующий
  ВНИЗ ВПРАВО ВВЕРХ                    // символ
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 18.6.

Таблица 18.6

Правильные записи	Записи с ошибками
/\	Пустая запись
[]	%
/\%[]	#
[]#/\%[]	/%\
	[]/\
	%/\/\
	/\#/\%

Глава 19

Лексический анализатор в среде Корректора

Решение задач

19.2. Задачи

1. Задано определение:

Определение

```
<число> ::= <целое> | <дробь>
<целое> ::= <цифра> | <целое><цифра>
<дробь> ::= <простая> | <десятичная>
<простая> ::= <целое>/<целое>
<десятичная> ::= .<целое> | <целое>. | <целое>.<целое>
<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Напишите программу, которая проверяет запись на ленте и помещает результат проверки справа от неё в виде двух букв (табл. 19.1).

Таблица 19.1

Код результата	Что означает
ОШ	Запись числом не является
ЦЛ	Запись есть «`целое`»
ДС	Запись есть «`десятичная`»
ПР	Запись есть «`простая`»

В начальный момент окошко расположено перед записью, а в конце работы — сразу за ней, если в записи нет ошибок. Если запись неверная, окошко должно указывать на первый ошибочный символ.

Примеры работы программы показаны на рис. 19.1.

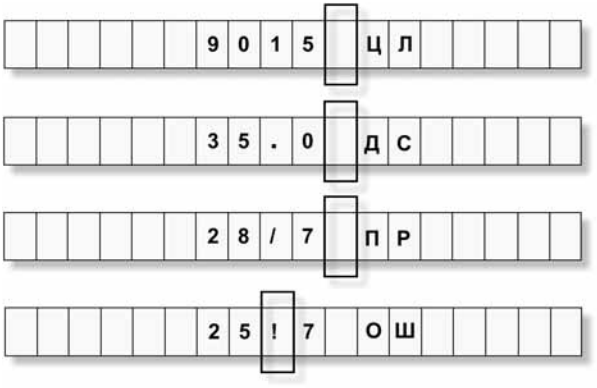

Рис. 19.1

Решение

Таблица переходов представлена в табл. 19.2.

Таблица 19.2

Состояние	Следующий символ				
	цифра	**.**	**/**	**пусто**	**осталь-ное**
(вход)	(целое)	(десятичная?)	(ошибка)	(ошибка)	(ошибка)
(целое)	(целое)	(десятичная)	(про-стая?)	(ответ_ЦЛ)	(ошибка)
(десятичная?)	(десятичная)	(ошибка)	(ошибка)	(ошибка)	(ошибка)
(десятичная)	(десятичная)	(ошибка)	(ошибка)	(ответ_ДС)	(ошибка)
(простая?)	(простая)	(ошибка)	(ошибка)	(ошибка)	(ошибка)
(простая)	(простая)	(ошибка)	(ошибка)	(ответ_ПР)	(ошибка)
(ответ_ЦЛ)					
(ответ_ДС)					
(ответ_ПР)					
(ошибка)					

Описание состояний

В каждом состоянии анализируется следующий символ записи и, в зависимости от его значения, принимается решение о переходе в новое состояние.

(вход)

Это состояние входное для алгоритма. Алгоритм анализирует первый символ записи.

Если символ — цифра, алгоритм переходит в состояние (целое).

Если точка — в состояние (десятичное?).

Любой другой символ — переход в состояние (ошибка).

(целое)

Алгоритм анализирует очередной символ записи.

Если символ — цифра, алгоритм остаётся в состоянии (целое).

Если символ — «/», алгоритм переходит в состояние (простая?).

Если символ — «.», алгоритм переходит в состояние (десятичная).

Если символ — `пусто`, алгоритм переходит в состояние (ответ_ЦЛ).

Любой другой символ — переход в состояние (ошибка).

(простая?)

Алгоритм анализирует следующий за «/» символ записи.

Если символ — цифра, алгоритм переходит в состояние (простая).

При любом другом символе — в состояние (ошибка).

(простая)

Алгоритм анализирует очередной символ записи.

Если символ — цифра, алгоритм остаётся в состоянии (простая).

Если символ — `пусто`, алгоритм переходит в состояние (ответ_ПР).

Любой другой символ — переход в состояние (ошибка).

(десятичная?)

Алгоритм анализирует второй символ записи, следующий за «.».

Если символ — цифра, алгоритм переходит в состояние (десятичная).

При любом другом символе — в состояние (ошибка).

(десятичная)

Алгоритм анализирует очередной символ записи.

Если символ — цифра, алгоритм остаётся в состоянии (десятичная).

Если символ — `пусто`, алгоритм переходит в состояние (ответ_ДС).

Любой другой символ — переход в состояние (ошибка).

Программа

```
ЭТО вход                              // Состояние (вход)
  ВПРАВО                              // Смотрим следующий символ
  ЕСЛИ       ЦИФРА ТО целое
  ИНАЧЕ ЕСЛИ .     ТО десятичная?
  ИНАЧЕ              ошибка
КОНЕЦ

ЭТО целое                             // Состояние (целое)
  ВПРАВО                              // Смотрим следующий символ
  ЕСЛИ       ПУСТО ТО ответ_ЦЛ
  ИНАЧЕ ЕСЛИ ЦИФРА ТО целое
  ИНАЧЕ ЕСЛИ .     ТО десятичная
  ИНАЧЕ ЕСЛИ /     ТО простая?
  ИНАЧЕ              ошибка
КОНЕЦ

ЭТО десятичная?                       // Состояние (десятичная?)
  ВПРАВО                              // Смотрим следующий символ
  ЕСЛИ ЦИФРА ТО десятичная
  ИНАЧЕ         ошибка
КОНЕЦ

ЭТО десятичная                        // Состояние (десятичная)
  ВПРАВО                              // Смотрим следующий символ
  ЕСЛИ       ПУСТО ТО ответ_ДС
  ИНАЧЕ ЕСЛИ ЦИФРА ТО десятичная
  ИНАЧЕ              ошибка
КОНЕЦ

ЭТО простая?                          // Состояние (простая?)
  ВПРАВО                              // Смотрим следующий символ
  ЕСЛИ ЦИФРА ТО простая
  ИНАЧЕ         ошибка
КОНЕЦ

ЭТО простая                           // Состояние (простая)
```

```
  ВПРАВО                              // Смотрим следующий символ
  ЕСЛИ       ПУСТО ТО ответ_ПР
  ИНАЧЕ ЕСЛИ ЦИФРА ТО простая
  ИНАЧЕ                ошибка
КОНЕЦ

// Запишем ответ ОШ и вернёмся к месту ошибки
ЭТО ошибка
  ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО    ошибка
    ИНАЧЕ ответ_ОШ
  ВЛЕВО                               // Возврат к месту ошибки
КОНЕЦ

ЭТО ответ_ОШ
  ВПРАВО ПИШИ О ВПРАВО ПИШИ Ш
  ПОВТОРИ 2 ВЛЕВО
КОНЕЦ

ЭТО ответ_ЦЛ
  ВПРАВО ПИШИ Ц ВПРАВО ПИШИ Л
  ПОВТОРИ 2 ВЛЕВО
КОНЕЦ
ЭТО ответ_ДС
  ВПРАВО ПИШИ Д ВПРАВО ПИШИ С
  ПОВТОРИ 2 ВЛЕВО
КОНЕЦ

ЭТО ответ_ПР
  ВПРАВО ПИШИ П ВПРАВО ПИШИ Р
  ПОВТОРИ 2 ВЛЕВО
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 19.3.

Таблица 19.3

Правильные записи	Ответ	Записи с ошибками
.0	ДС	Пустая запись
1.	ДС	.
01	ЦЛ	2.1
1.0	ДС	1.1.1
11.00	ДС	/
1/0	ПР	/1
0/10	ПР	1/
		1/1/1
		1.1/1

2. Задано определение:

Определение 3

```
<выражение> ::= <число>+<число> | <выражение>+<число>
<число> ::= <целое> | <дробь>
<целое> ::= <цифра> | <целое><цифра>
<дробь> ::= .<целое> | <целое>. | <целое>.<целое>
<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Напишите программу, которая проверяет запись на ленте и помещает результат проверки справа от неё в виде двух букв (табл. 19.4).

Таблица 19.4

Код результата	Что означает
ОШ	Запись не является ни числом, ни выражением
ЧС	Запись есть «число»
ВР	Запись есть «выражение»

В начальный момент окошко расположено перед записью, а в конце работы — сразу за ней, если в записи нет ошибок. Если запись содержит ошибки, окошко должно указывать на первый ошибочный символ.

Примеры работы программы показаны на рис. 19.2.

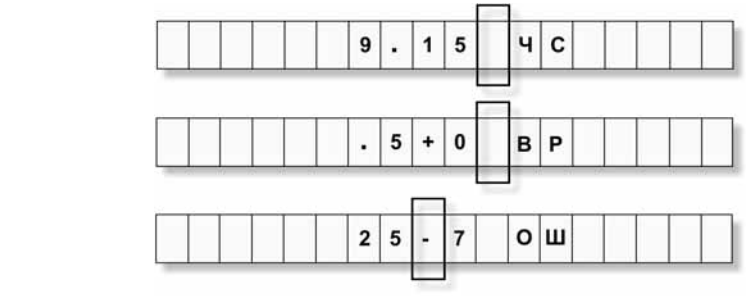

Рис. 19.2

Решение

Если при анализе записи ошибок не обнаружено, то запись является выражением тогда, когда в ней был хотя бы один знак «+», или числом, если знаков «+» не было. Иными словами, необходимо фиксировать появление знаков «+», чтобы после полного анализа записи написать правильный ответ. Эта часть алгоритма не отражена в таблице переходов (табл. 19.5) (чтобы не загромождать её деталями), но очень легко программируется при помощи ящика Корректора.

Таблица переходов представлена табл. 19.5.

Таблица 19.5

Состояние	Следующий символ				
	цифра	**.**	**+**	**пусто**	**остальное**
(вход)	(целое)	(дробь?)	(ошибка)	(ошибка)	(ошибка)
(целое)	(целое)	(дробь)	(начало_числа)	(ответ)	(ошибка)
(начало_числа)	(целое)	(дробь?)	(ошибка)	(ошибка)	(ошибка)
(дробь?)	(дробь)	(ошибка)	(ошибка)	(ошибка)	(ошибка)
(дробь)	(дробь)	(ошибка)	(начало_числа)	(ответ)	(ошибка)
(ответ)					
(ошибка)					

Программа

```
ЭТО вход                              // Состояние (вход)
  ПИШИ 0      // Установим ящик
  ЯЩИК+       // в состояние:
  ПИШИ ПУСТО // знаков "+" нет
```

```
  ВПРАВО                          // Смотрим следующий символ
  ЕСЛИ       ЦИФРА ТО целое
  ИНАЧЕ ЕСЛИ .     ТО дробь?
  ИНАЧЕ ошибка
КОНЕЦ

ЭТО целое                         // Состояние (целое)
  ВПРАВО                          // Смотрим следующий символ
  ЕСЛИ       ЦИФРА ТО целое
  ИНАЧЕ ЕСЛИ .     ТО дробь
  ИНАЧЕ ЕСЛИ +     ТО начало_числа
  ИНАЧЕ ЕСЛИ ПУСТО ТО ответ
  ИНАЧЕ ошибка
КОНЕЦ

ЭТО начало_числа                  // Состояние (начало_числа)
  ОБМЕН  // Установим ящик
  ПИШИ 1 // в состояние:
  ОБМЕН  // знак "+" был
  ВПРАВО                          // Смотрим следующий символ
  ЕСЛИ       ЦИФРА ТО целое
  ИНАЧЕ ЕСЛИ .     ТО дробь?
  ИНАЧЕ ошибка
КОНЕЦ

ЭТО дробь?                        // Состояние (дробь?)
  ВПРАВО                          // Смотрим следующий символ
  ЕСЛИ ЦИФРА ТО дробь
  ИНАЧЕ ошибка
КОНЕЦ

ЭТО дробь                         // Состояние (дробь)
  ВПРАВО                          // Смотрим следующий символ
  ЕСЛИ       ПУСТО ТО ответ
  ИНАЧЕ ЕСЛИ ЦИФРА ТО дробь
  ИНАЧЕ ЕСЛИ +     ТО начало_числа
  ИНАЧЕ ошибка
```

```
КОНЕЦ

ЭТО ответ
  ОБМЕН
  ЕСЛИ 0 ТО ответ_число
  ИНАЧЕ     ответ_выражение
КОНЕЦ

ЭТО ответ_число
  ОБМЕН
  ВПРАВО ПИШИ Ч
  ВПРАВО ПИШИ Л
  ПОВТОРИ 2 ВЛЕВО
КОНЕЦ

ЭТО ответ_выражение
  ОБМЕН
  ВПРАВО ПИШИ В
  ВПРАВО ПИШИ Р
  ПОВТОРИ 2 ВЛЕВО
КОНЕЦ

ЭТО ошибка
  ВПРАВО
  ЕСЛИ ПУСТО
    ТО ответ_ошибка
    ИНАЧЕ ошибка
  ВЛЕВО
КОНЕЦ

ЭТО ответ_ошибка
  ВПРАВО ПИШИ О
  ВПРАВО ПИШИ Ш
  ПОВТОРИ 2 ВЛЕВО
КОНЕЦ
```

Тесты

Набор тестов приводится в табл. 19.6.

Таблица 19.6

Правильные записи	Ответ	Записи с ошибками
.0	ЧС	Пустая запись
1.	ЧС	.
1+2.	ВР	+1
1.+1	ВР	+1.+1
1.+.1	ВР	1+
.0+6+9.	ВР	1+.
		1+.+1
		1.1.1+2
		.+1.

3. Задано определение:

Определение 4

```
<выражение> ::= <число> |
                <выражение> + <выражение> |
                <выражение> - <выражение> |
                (<выражение>)
<число>    ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Напишите программу, которая проверяет правильность записи выражения.

Если запись правильная, то нужно за ней на ленте записать ОК. Если запись ошибочная, то нужно записать ОШ и дополнительно установить окошко на первый неверный символ или на место обнаружения ошибки в случае ошибок со скобками.

В начальный момент окошко расположено перед записью.

Примеры работы программы показаны на рис. 19.3.

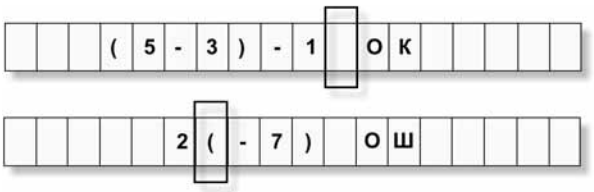

Рис. 19.3

Решение

Определение выражения, приведённое в условии задачи, содержит скачок сложности из-за наличия безобидной, на первый взгляд, строки с круглыми скобками:

```
<выражение> ::= <число> |                          (1)
                <число> + <выражение> |            (2)
                <число> - <выражение> |            (3)      (*)
                (<выражение>)                      (4)
<число>   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9  (5)
```

До сих пор рекурсивность определений (по Бэкусу) была простой и достаточно очевидной. В самом деле, из записи

```
<число> ::= <цифра> | <цифра><число>
<цифра> ::= 0 | 1
```

совершенно понятно, что число — это любая последовательность из нулей и единиц.

Скромная строка `(<выражение>)` в определении (*) позволяет считать выражениями записи типа:

(1 + (7 − (9 + 8)))

7 + (8) − (1 − 7) + 6

и т. д.

Иными словами, определение (*) почти соответствует нашему интуитивному представлению о том, какие могут быть обычные выражения с круглыми скобками. Почему почти? Ну, например, запись «5 − (−3)» выражением в смысле определения не будет, в то время как в обычной жизни это совершенно нормальная запись, значение которой равно 8.

Докажем, например, что запись «7 + (6 − (9))» является выражением в смысле определения (*).

Из (2) следует, что запись «7 + (6 − (9))» будет выражением, если выражением является запись «(6 − (9))».

Из (4) следует, что запись «(6 − (9))» будет выражением, если выражением является запись «6 − (9)».

Из (3) следует, что запись «6 − (9)» будет выражением, если выражением является запись «(9)».

Из (4) следует, что запись «(9)» будет выражением, если выражением является запись «9».

Но запись «9» — это выражение из (1), ведь «9» — это число по правилу (5).

Поработать с рекурсивными определениями очень полезно. Эта область информатики соприкасается с математикой и лингвистикой.

Запись «5 + 3» — это выражение, а «5 + F» — нет. Это понятно. Но тогда (чуть сложнее) «5 + 3 − 7» — тоже выражение. Ведь «5 + 3» — выражение и «7» — выражение тоже. Понятно, что любая цепочка цифр, между которыми стоят знаки «+» и «−», является выражением. Это легко доказать по индукции.

Рассмотрим объект:

V_n = <цифра><знак><цифра><знак>...<цифра> (**)

(Индекс n информирует, что в записи (**) — n знаков «+» , «−».)

Объект

V_1 = <цифра><знак><цифра>

выражением является (база индукции).

Предположим, что V_{n-1} — является выражением при некотором n > 1. Докажем, что тогда V_n тоже является выражением (индуктивный переход).

Доказательство

Очевидно, что

V_n = V_{n-1}<знак><цифра> (***)

Но так как V_{n-1} — выражение (по индуктивному предположению), то объект (***) является выражением по определению (*).

Индуктивный переход доказан.

Можно рассуждать и так.

V_1 — выражение по (*). Но если к выражению добавить конструкцию <знак><цифра>, то по (*) получим выражение V_2. Следовательно, если мы будем продолжать в том же духе (добавляя к выражению конструкцию <знак><цифра>), то всегда будем получать выражения для любого n. То есть V_n всегда будет выражением.

Несколько иллюстраций.

Примеры выражений:

1. (2);
2. ((2));
3. (2 + 3);
4. (2 + 3 − 7);
5. (2 + 3) − (7 + 6);
6. ((2 + 3) − (7 + 6));
7. (2) − (7);

8. ((2) − (7));
9. ((2 + 3) − (7 + 6)) + 5.

Примеры записей с ошибками:

1. ();
2. (2 (+ 3));
3. (2 + 3));
4.)2(;
5. (2 + 3;
6. 1 + (−5).

Докажем, что запись «1 + (−5)» выражением не является.

Она была бы выражением по (*), если бы выражением была запись «(−5)». А запись «(−5)» была бы выражением, если бы выражением была запись «-5». Но последняя запись выражением не является, ибо ни одна часть определения (*) под разборку записи «−5» не подходит.

Этот пример расходится с нашим представлением о правильных выражениях со скобками. Происходит так потому, что число со знаком в обычном понимании не является числом в смысле определения (*). По определению (*) число — это цифра и знака не содержит.

Алгоритм решения задачи основан на таблице переходов из одного состояния в другие плюс дополнительный подсчёт баланса круглых скобок (в таблице он не отражён).

Для слежения за балансом круглых скобок будем использовать ящик Корректора (изначально пустой). На каждой открывающейся скобке будем применять к ящику операцию `плюс`, а на каждой закрывающейся — `минус`. Если после просмотра всего выражения ящик не будет пуст, то это — ошибка: баланс скобок в записи нарушен.

Итак, алгоритм, представленный таблицей переходов, будет «следить» за правомочностью появления круглой скобки в текущем месте записи, а ящик — за балансом скобок на уровне записи в целом.

Когда, например, за числом расположена открывающаяся скобка (ошибка!), таблица переходов (табл. 19.7) этот случай не пропустит. А вот распознать ошибку в записи «2 + 3)» таблица бессильна! Ведь закрывающая скобка за числом — ситуация вполне нормальная. Но эта ошибка будет обнаружена после анализа содержимого ящика.

Таблица переходов представлена табл. 19.7.

Таблица 19.7

Состояние	Следующий символ					
	цифра	**(**	**)**	**+,-**	**пусто**	**остальное**
(вход)	(число)	(вход)	(ошибка)	(ошибка)	(ошибка)	(ошибка)
(число)	(ошибка)	(ошибка)	(скобка_закр)	(вход)	(ответ)	(ошибка)
(скобка_закр)	(ошибка)	(ошибка)	(скобка_закр)	(вход)	(ответ)	(ошибка)
(ответ)						
(ошибка)						

Программа

```
// Ящик будем использовать для подсчёта
// баланса круглых скобок в выражении.
// Поместим в ящик ПУСТО и перейдём в состояние (вход).
ЭТО вход0
  ОБМЕН ПИШИ ПУСТО ОБМЕН
  вход
КОНЕЦ

ЭТО вход                          // Состояние (вход)
  ВПРАВО                          // Смотрим следующий символ
  ЕСЛИ       ЦИФРА ТО число       // На ленте число.
  ИНАЧЕ ЕСЛИ ( ТО вход1           // Процедура вход1 выполнит над
  ИНАЧЕ ошибка                    // ящиком операцию ПЛЮС и вернёт
КОНЕЦ                             // алгоритм состояние (вход)

ЭТО вход1                         // Встретилась скобка "(".
  ОБМЕН ПЛЮС ОБМЕН                // Применим к ящику операцию ПЛЮС.
  вход
КОНЕЦ

ЭТО скобка_закр                   // Состояние (скобка_закр)
  ВПРАВО                          // Смотрим следующий символ
  // Применим к ящику операцию МИНУС, если можно, и проверим
  // корректность следующего символа.
```

```
  ОБМЕН
  ЕСЛИ ПУСТО ТО { ОБМЕН ВЛЕВО ошибка }
    ИНАЧЕ
    {
      МИНУС
      ОБМЕН
      ЕСЛИ        + ТО вход
      ИНАЧЕ ЕСЛИ  - ТО вход
      ИНАЧЕ ЕСЛИ ПУСТО ТО ответ
      ИНАЧЕ ЕСЛИ ) ТО скобка_закр
      ИНАЧЕ ошибка
    }
КОНЕЦ

ЭТО число                          // Состояние (число)
  ВПРАВО                           // Смотрим следующий символ
  ЕСЛИ ПУСТО ТО ответ
  ИНАЧЕ ЕСЛИ + ТО вход
  ИНАЧЕ ЕСЛИ - ТО вход
  ИНАЧЕ ЕСЛИ ) ТО скобка_закр
  ИНАЧЕ ошибка
КОНЕЦ

ЭТО ответ
  ВПРАВО
  ОБМЕН
  // Прежде чем писать на ленту ОК, проверим, не нарушен
  // ли в записи баланс скобок.
 ЕСЛИ НЕ ПУСТО
   ТО { ОБМЕН ВЛЕВО ошибка }
   ИНАЧЕ { ПИШИ О ВПРАВО ПИШИ К ВЛЕВО ВЛЕВО }
КОНЕЦ

ЭТО ошибка
  ВПРАВО
  ЕСЛИ ПУСТО
    ТО ответ_ошибка
```

```
    ИНАЧЕ ошибка
    ВЛЕВО
КОНЕЦ

ЭТО ответ_ошибка
  ВПРАВО ПИШИ О ВПРАВО ПИШИ Ш
  ПОВТОРИ 2 ВЛЕВО
КОНЕЦ
```

Тесты

В качестве тестов можно использовать примеры правильных выражений и записей с ошибками, приведённые ранее.

Глава 20

Построение трансляторов

Решение задачи

Задача 2

Задано определение:

Определение 1

```
<выражение> ::= <число> | <выражение> + <число> | <выражение> - <число>
<число>     ::= 0 | 1 | 2
```

Написать программу для Корректора, которая вычисляет выражение, если оно правильное, или записывает на ленту сообщение ОШ, если запись содержит ошибку. В случае ошибок в записи, окно должно указывать на первый неверный символ.

В начальный момент окошко расположено перед записью.

> ***Замечание 1***
>
> Значение выражения должно быть записано на ленте в виде обычного целого десятичного числа.

> ***Замечание 2***
>
> Предполагается, что результат вычислений выражения меньше числа, равного длине алфавита Корректора.

> ***Замечание 3***
>
> Предполагается, что результат и все промежуточные вычисления не отрицательные.

Примеры работы программы показаны на рис. 20.1–20.3.

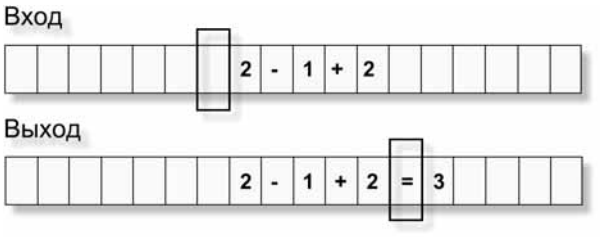

Рис. 20.1

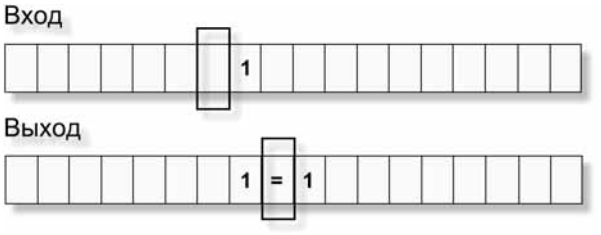

Рис. 20.2

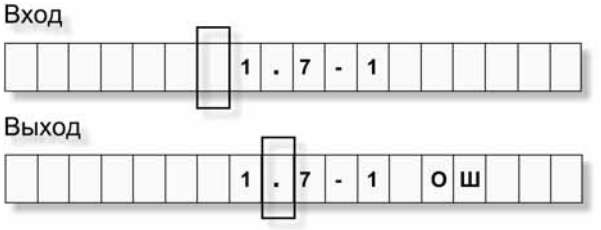

Рис. 20.3

Решение

Таблица переходов представлена табл. 20.1.

Таблица 20.1

Состояние	Следующий символ			
	0, 1, 2	+, –	пусто	остальное
(вход)	(выражение)	(ошибка)	(ошибка)	(ошибка)
(выражение)	(ошибка)	(вход)	(ответ)	(ошибка)
(ответ)				
(ошибка)				

Нам нужно не просто проверить выражение, но и вычислить его. Поэтому в программе появляются дополнительные процедуры, связанные не с переходом в другие состояния анализатора выражения, а с вычислениями.

Программа

```
ЭТО вход0                    // Эта заглавная процедура готовит
  ОБМЕН                      // ящик для вычислений перед входом в
  ПИШИ 0                     // состояние (вход) алгоритма.
  ОБМЕН                      // Подготовка состоит в записи символа 0
  вход                       // в ящик. Эту ячейку памяти Корректора будем
КОНЕЦ                        // использовать как сумматор (накопитель
                             // результата).

ЭТО вход                      // Состояние (вход).
  ВПРАВО                      // Смотрим следующий символ.
  ЕСЛИ       0 ТО выражение   // Если 0, переходим в состояние (выражение)
  ИНАЧЕ ЕСЛИ 1 ТО вычисление1 // без вычислений. Если 1 или 2, перед
  ИНАЧЕ ЕСЛИ 2 ТО вычисление2 // переходом в состояние (выражение)
                             // добавим число в ящик (или отнимем).
  ИНАЧЕ ошибка               // Любой другой символ в состоянии (вход) -
КОНЕЦ                        // ошибка.

ЭТО выражение                // Состояние (выражение).
  ВПРАВО                     // Смотрим следующий символ.
  ЕСЛИ       +     ТО вход   // Если это "+" или "-", переходим в
  ИНАЧЕ ЕСЛИ -     ТО вход   // состояние (вход)- выражение продолжается.
  ИНАЧЕ ЕСЛИ ПУСТО ТО ответ // Если это ПУСТО, выражение закончилось.
  ИНАЧЕ ошибка               // Любой другой символ - ошибка.
КОНЕЦ

ЭТО вычисление1
  ВЛЕВО
  ЕСЛИ -
    ТО    { ВПРАВО отнять1 }
    ИНАЧЕ { ВПРАВО добавить1 }
  выражение
КОНЕЦ
```

```
ЭТО вычисление2
  ВЛЕВО
  ЕСЛИ -
    ТО    { ВПРАВО отнять2 }
    ИНАЧЕ { ВПРАВО добавить2 }
  выражение
КОНЕЦ

ЭТО добавить1
  ОБМЕН ПЛЮС ОБМЕН
КОНЕЦ

ЭТО добавить2
  ОБМЕН ПЛЮС ПЛЮС ОБМЕН
КОНЕЦ

ЭТО отнять1
  ОБМЕН МИНУС ОБМЕН
КОНЕЦ

ЭТО отнять2
  ОБМЕН МИНУС МИНУС ОБМЕН
КОНЕЦ

ЭТО ошибка
  ВПРАВО
  ЕСЛИ ПУСТО
    ТО ответ_ошибка
    ИНАЧЕ ошибка
  ВЛЕВО
КОНЕЦ

ЭТО ответ_ошибка
  ВПРАВО ПИШИ О
  ВПРАВО ПИШИ Ш
  ПОВТОРИ 2 ВЛЕВО
КОНЕЦ
```

```
ЭТО ответ                    // Запись ответа в виде числа на ленте.
  ВПРАВО ВПРАВО              // Для преобразования символа в число
                             // используется процедура Символ_в_число,
                             // описанная в главе 12 части II.
  ЯЩИК-                      // Символьное число из ящика — на ленту.
  Символ_в_число
                             //  Последний дизайнерский штрих:
  ПОКА НЕ ПУСТО ВЛЕВО        //  знаком "=" отделим результат
  ПИШИ =                     //  от примера.
КОНЕЦ

// Перевод символьного числа в обычное число.
// Вход:  окно установлено на символьное число.
// Выход: окно установлено на младшую цифру результата,
//        которая теперь занимает ячейку символьного числа.
// Ограничение: процедура работает только для символьных
//              чисел, значение которых меньше 100.
// ------------------------------------------------------------
ЭТО Символ_в_число
  ПОКА НЕ ЦИФРА
  {
   ПОВТОРИ 10 МИНУС
   ВЛЕВО
   ЕСЛИ ПУСТО
    ТО    ПИШИ 1
    ИНАЧЕ ПЛЮС
   ВПРАВО
  }
КОНЕЦ
```