Александр Дуванов
Алексей Рудь
Виктор Семенко

АЗЫ ПРОГРАММИРОВАНИЯ

ФАКУЛЬТАТИВНЫЙ КУРС

КНИГА ДЛЯ УЧЕНИКА

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 ученика. — СПб.: БХВ-Петербург, 2005. — 352 с.: ил.

ISBN 5-94157-583-1

Теоретические основы программирования излагаются в наглядных средах исполнителей Кукарача и Корректор, сложные темы становятся простыми и ясными.

Особое внимание уделяется основам формализации и построения алгоритмов, тестированию и отладке программ. Целый раздел посвящен принципам построения трансляторов. Программные среды исполнителей помещены на диске, прилагаемом к книге учителя из этого комплекта, который также содержит решения всех задач и дополнительные материалы к урокам. Закрепить практический опыт программирования поможет книга «Азы программирования. Задачник».

Для учащихся 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. 28,38.
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 5-94157-583-1

© Дуванов А. А., Рудь А. В., Семенко В. П., 2005
© Дуванов А. А., Русс А. А., иллюстрации, 2005
© Оформление, издательство "БХВ-Петербург", 2005

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

> — А нам и так хорошо живётся! — кричит нервный Шарик.
> — Вам плохо живётся, — объясняет тётя. — Только вы этого не понимаете. Вы по ошибке счастливы. Но я вам глаза раскрою.
>
> *Э. Успенский*

Эта книга — отражение самого творческого курса Роботландского сетевого университета, на котором школьники 5–8 классов приступают к изучению основ программирования. Ребята быстро забывают про то, что они ученики, становятся вровень со своими наставниками, а порой заставляют учителей испытывать зависть найденными решениями и новыми потрясающими задачами!

Программирование — удивительный род человеческой деятельности, который сродни волшебству. Несколько заклинаний на языке посвящённых — и «твёрдый» металл на письменном столе получает «мягкую» душу: компьютер оживает и ведёт себя «разумно». Переживания программиста, вероятно, сравнимы с переживанием Создателя. Ведь Он сотворил живых тварей при

помощи одного Своего слова. Наверное, фраза «и создал Он их по образу и подобию Своему» относятся как раз к программистам. Если вы понимаете, о чём я говорю, значит, в душе вы — программист и взяли в руки эту книгу не зря!

О волшебстве

Первый раз автор столкнулся с волшебством в 6 классе. Брат подарил мне книгу Е. Айсберга «Радио?.. Это очень просто!». Свой первый радиоприёмник я спаял на деревянной дощечке, которую выпилил ножовкой из доски, валявшейся в сарае. Пригладил её рубанком, отполировал шкуркой, приклеил клеем БФ транзисторы, конденсаторы, резисторы... Потом спаял всё это согласно схеме. Какой же была моя радость пополам с удивлением, когда эта деревяшка заговорила человеческим голосом! Своими руками я сотворил Буратино!

Стало понятно, что я волшебник. Это было удивительное ощущение! Второй раз испытал подобные чувства, когда написал первую программу. А потом каждый раз, создавая программные коды, не переставал удивляться (не перестаю и сейчас!), как можно словесными заклинаниями вдохнуть жизнь в железного (теперь) Буратино на моём рабочем столе!

Мне хочется передать эту радость творчества и это постоянное удивление нынешним ребятам и девчатам! Это желание — основная причина появления этой книги.

Программисты — счастливые люди, но жизнь у них трудна. Во-первых, нужно знать теорию волшебства, во-вторых — язык, на котором записываются заклинания, и, в-третьих — знать исполнителя, который эти заклинания выполняет, превращая их в ощущение жизни на экране.

В данной книге программирование — это:

1. Теория — основы построения алгоритмов.
2. Исполнитель — Кукарача, Корректор.
3. Язык — язык роботландских исполнителей.

Составляющие знаний программиста пронумерованы в порядке убывания важности.

Самым неважным оказывается язык, хотя новичок всегда думает, что язык — это и есть программирование. Это всё равно, как сказать, что писательство — это авторучка, а химия — это пробирки. Или что топор — это основа плотницкого мастерства. Конечно, для писателя важно знать, как менять стержень в авторучке, а для плотника — как затачивать топор. Но знания топора и авторучки не лежат в основе изделий, которые вырубаются из дерева и выписываются на бумаге.

Так и в программировании: язык — это второстепенный элемент, при помощи которого описываются наши знания основ построения алгоритмов в виде программы для конкретного исполнителя.

Если знаний основ формализации и построения алгоритмов нет, изучение языка программирования их не прибавит.

Как говорил классик программирования Эдгар Дийкстра — самый главный язык, который должен знать программист, — 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 можно перевести алгоритм в программу, записав несколько заклинаний на языке посвящённых!

Желаю приятного путешествия по страницам этой книги!

Для работы вам понадобится программная реализация исполнителей, описанных в книге. Этот продукт записан на диске, который прилагается к учительской части «Азов программирования».

Под отдельную обложку вынесен сборник задач, сформированный по материалам роботландских турниров. Если вы решились на покупку этой книги, то и задачник вам совершенно необходим. Ведь программирования без практики не бывает!

Автор исполнителей

Кукарача и Корректор,

руководитель Роботландского университета

Александр Александрович Дуванов

kurs@robotland.pereslavl.ru

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 является А. А. Дуванов. Для построения рисунков были использованы художественные заготовки А. А. Русса (художник Роботландии).

Вступление

> Было когда-то на свете двадцать пять оловянных солдатиков. Все они были сыновьями одной матери — старой оловянной ложки — и, значит, друг другу родными братьями. Они были очень красивы: ружьё на плече, грудь колесом, мундир красный с синим. Чудо что за солдатики!
>
> Они лежали, все двадцать пять, в картонной коробке. В ней было темно и тесно. Но вот однажды коробка открылась.
>
> *Г.-Х. Андерсен*

Ниже приводятся краткие объяснения понятиям *исполнитель*, *СКИ*, *среда*, *отказ*, *алгоритм*, *программа*, *транслятор*.

Эти базовые понятия будут уточняться и конкретизироваться на примерах работы с исполнителями Кукарача и Корректор. Сейчас просто послушайте, как звучат главные слова программистской магии!

Понятие исполнителя

Исполнителем называют устройство, которое умеет выполнять команды из фиксированного списка.

Этот набор «понятных» исполнителю команд называют системой команд этого исполнителя, или сокращенно **СКИ**.

Обстановку, в которой работает исполнитель, называют его **средой**.

Различные исполнители получают свои команды по-разному. Управляя автомобилем, например, надо нажимать на педали, дергать за рычаги и крутить руль. А исполнитель-курсор на экране компьютера получает свои команды `ВЛЕВО`, `ВПРАВО`, `ВВЕРХ`, `ВНИЗ`, когда человек нажимает на клавиатуре клавиши со стрелками или перемещает по коврику манипулятор мышь.

Управление исполнителем может привести к одной из двух **аварийных ситуаций** (**отказов**), одна из них условно называется «**Не понимаю**», другая — «**Не могу**».

Отказ «Не понимаю» возникает у всех исполнителей в одном случае — задаётся команда, не входящая в СКИ.

Отказ «Не могу» специфичен для каждой команды из СКИ — он возникает, когда условия среды не позволяют выполнить заданную команду (автомобиль не может ускоряться от педали газа, если не включена передача, курсор не может выйти за пределы экрана).

Для знакомства с исполнителем можно использовать схему, изображённую на рис. 1.

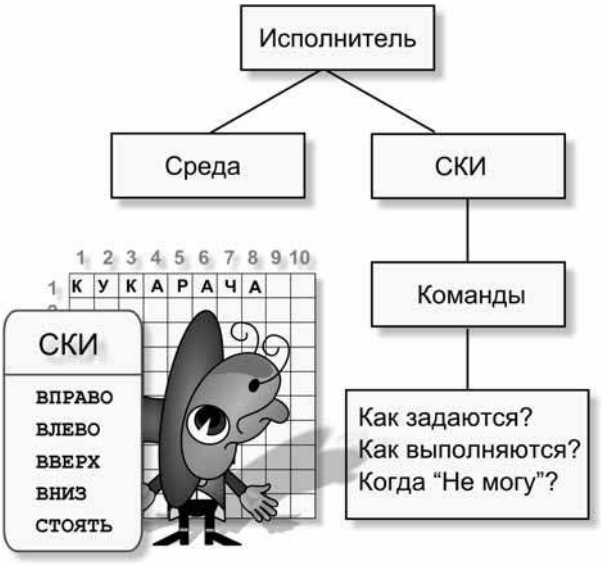


Рис. 1. Схема знакомства с исполнителем

Понятие алгоритма и программы

Алгоритм — это описание действий исполнителя для выполнения определённой работы.

Чтобы алгоритм стал понятен исполнителю, его записывают при помощи команд из СКИ. Такая запись алгоритма называется **программой**.

Для записи программ часто пользуются специальными **языками программирования**. В эти языки наряду с командами СКИ входят **процедуры**, описывающие новые команды (команды, которые конструируются программистом), и **управляющие структуры**, предназначенные для записи алгоритма.

Исполнитель понимает команды только из своей СКИ и выполнить программу, написанную на языке программирования, самостоятельно не может.

Для «перевода» текста программы в СКИ строят новый исполнитель, называемый **транслятором** (**интерпретатор** или **компилятор** языка). Процесс выполнения программы, написанной на языке программирования, упрощённо поясняет схема, изображённая на рис. 2.

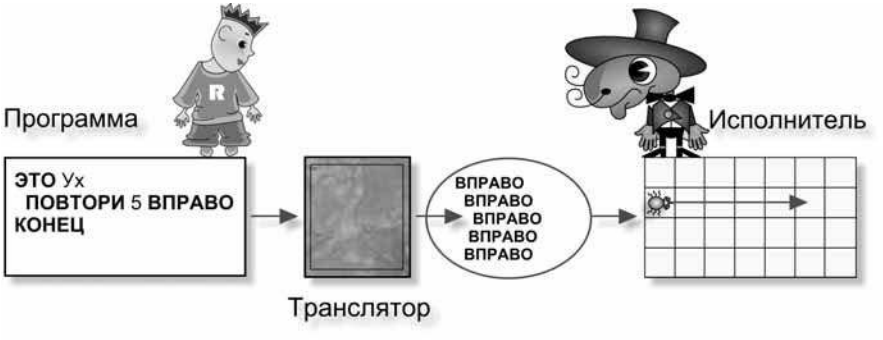


Рис. 2. Схема выполнения программы

Часть I

Кукарача

Глава 1

Кукарача и его среда обитания

1.1. Знакомство с Кукарачей

— Как тебя зовут?
Кот говорит:
— И не знаю как. И Барсиком меня звали, и Пушком, и Оболтусом. И даже Кис Кисычем я был. Только мне всё это не нравится. Я хочу фамилию иметь.

Э. Успенский

Вася. Послушай, Петя, что это за приложение со странным названием «Кукарача» «живёт» на моём компьютере (рис. 1.1)?

Рис. 1.1. Разговор братьев

Петя. Кукарача? Это исполнитель, который может ползать по клетчатому полю и собирать из кубиков разные слова. Давай я покажу тебе, как с ним работать.

Вася. Отлично! Запускаю Кукарачу.

Петя. Что ты видишь на экране?

Вася. Я вижу пустое клетчатое поле и на нём одинокое существо в левом верхнем углу (рис. 1.2).

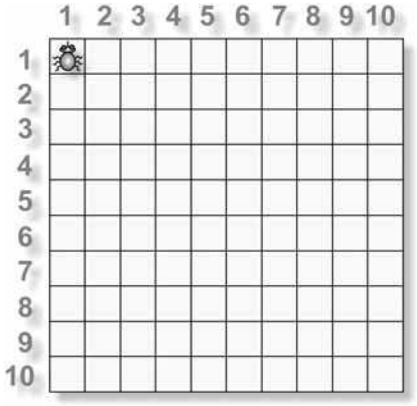


Рис. 1.2. Клетчатое поле с исполнителем

Петя. Это и есть среда исполнителя. А в углу в ожидании указаний стоит сам исполнитель — Кукарача. На рис. 1.3 показана его СКИ (**С**истема **К**оманд **И**сполнителя).

Рис. 1.3. Система команд Кукарачи

Давай расскажу, как исполнитель выполняет команды. По командам `ВЛЕВО`, `ВПРАВО`, `ВВЕРХ`, `ВНИЗ` он переползает в соседнюю клетку в указанном направлении. Команда `СТОЯТЬ` — это «пустая» команда. Выполняя её, Кукарача не делает никаких движений, он вообще ничего не делает!

Вася. Зачем же было включать такую команду в СКИ?

Петя. Она удобна при программировании развилок. Мы этим займёмся чуть позже.

Вася. А как задавать команды исполнителю?

Петя. Надо их записывать в редакторе команд и нажимать экранную кнопку *GO* (go по-английски означает «иди», читается «гоу») (рис. 1.4).

Рис. 1.4. Редактор команд

Вася. Хорошо, напишу команду `ВПРАВО` и нажму кнопку *GO*... Есть! Кукарача послушно сместился на одну клетку вправо (рис. 1.5).

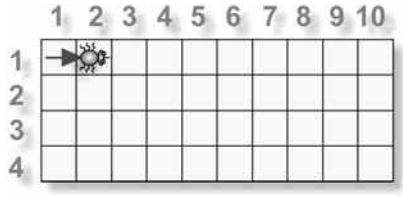


Рис. 1.5. Выполнена команда `ВПРАВО`

А теперь скомандую `ВВЕРХ`. Ага, хитрый исполнитель пошевелил лапками и написал «Не могу» (рис. 1.6)!

Рис. 1.6. Исполнитель сообщает «Не могу»

Петя. Кукарача может сталкивать с поля кубики, но сам покинуть свою среду не может. Вот и сообщает об этом.

Вася. Но на поле нет никаких кубиков!

Петя. Их можно поставить. Кубик — это клетка, в которую записана буква, цифра или другой символ. Из кубиков можно складывать слова (рис. 1.7).

Вася. Я дал команду толкнуть вверх кубик **к**, а исполнитель не понял (рис. 1.8)!

Рис. 1.7. Кубики на поле исполнителя

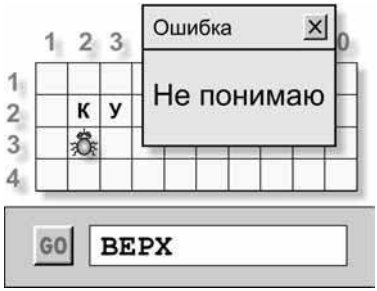


Рис. 1.8. Исполнитель сообщает «Не понимаю»

Петя. Ты записал слово с ошибкой. Команды **ВЕРХ** нет в СКИ, и Кукарача не знает, что надо делать!

Вопросы и задания

1. Что сделает Кукарача, получив команду **НАЛЕВО**?
2. Кукарача стоит в левом нижнем углу поля. Как он выполнит команду **ВНИЗ**?
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
```

1.2. Первая задача

Приказ-расписание

1. Подъём в 7.30 (ответственный Матроскин).
2. Завтрак в 8.25 (ответственный Шарик). Поедание совместное.
3. Топка печки в 9.00 (ответственный Матроскин).
4. Доставка дров (ответственный Шарик).
5. Обед в 14.00 (ответственный Матроскин). Поедание совместное.
6. Мытьё посуды, но не облизывание, в 14.30 (ответственный Шарик).

Э. Успенский

Петя. Не пора ли перейти от общих слов к решению задач?

Задача

Пусть Кукарача превратит слово КУЛИК в нашу фамилию (рис. 1.9).

Рис. 1.9

Вася. Ну, раз он может выталкивать с поля буквы, то это просто. Вот алгоритм решения этой задачи:

1. Выбросить лишние буквы.
2. Подойти к концу слова.
3. Придвинуть последнюю букву к началу слова.

Петя. Молодец! Первый пункт алгоритма предлагаю записать подробнее.

Вася. Можно. Тогда получится так:

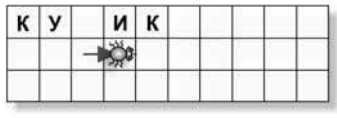
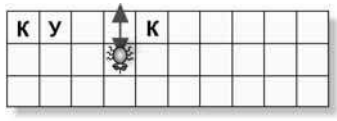
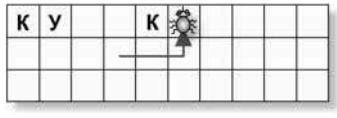
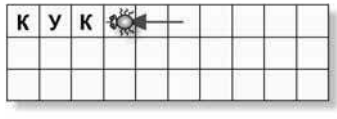
1. Выбросить лишние буквы.
   1.1. Выбросить Л.
   1.2. Подойти к И.
   1.3. Выбросить И.

2. Подойти к концу слова.
3. Придвинуть последнюю букву к началу слова.

Петя. Начинай выполнять свой план.

Вася с увлечением приступил к управлению исполнителем, а Петя по давней привычке записал его работу в тетрадь (табл. 1.1):

Таблица 1.1

Шаги алгоритма	Команды	Среда исполнителя
1. Выбросить лишние буквы		
1.1. Выбросить Л	ВВЕРХ ВНИЗ	
1.2. Подойти к И	ВПРАВО	
1.3. Выбросить И	ВВЕРХ ВНИЗ	
2. Подойти к концу слова	ВПРАВО ВПРАВО ВВЕРХ	
3. Придвинуть последнюю букву к началу слова	ВЛЕВО ВЛЕВО	

1.3. Задачи

1. Превратить молоток в моток (рис. 1.10).

Рис. 1.10

2. Превратить мишку в мышку (рис. 1.11).

Рис. 1.11

3. Починить колесо (рис. 1.12).

Рис. 1.12

4. Какую роль хочет играть Кукарача на поле (рис. 1.13)?

Рис. 1.13

5. Получить имя знакомой кошки Кукарачи (рис. 1.14).

Рис. 1.14

Глава 2

Вася экономит свой труд

2.1. Ток — это кот задом наперёд

— Что-то у нас кошачьим духом пахнет.
Не иначе как дядя Фёдор кота притащил.

Э. Успенский

Петя. Знаешь ли, Вася, что такое кот задом наперёд (рис. 2.1)?

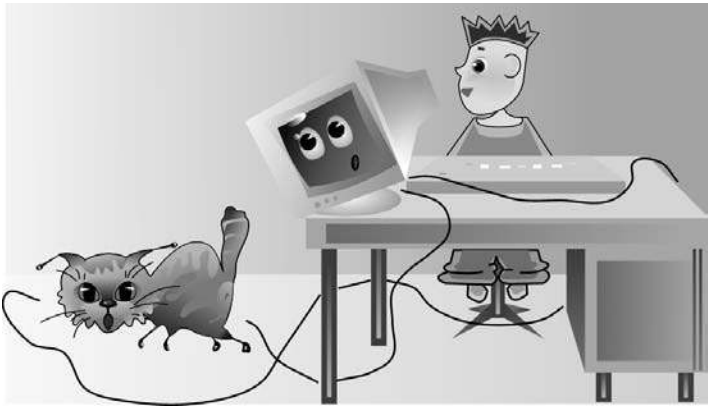


Рис. 2.1. Что такое кот задом наперёд?

Вася. Кот — он и в огороде кот. Не понял вопроса!

Петя. А будет кот задом наперёд совсем не котом, а тем, что заставляет работать наш компьютер.

Вася. Процессором что ли?

Петя. Чудак, если слово кот прочитать с конца, то получится слово...

Вася. А, понял! Получится слово «ток».

Петя. Так вот, давай заставим Кукарачу превратить кота в ток (рис. 2.2).

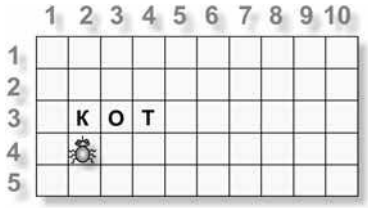


Рис. 2.2. Превратить кота в ток

Вася. Надо переставить первую и последнюю буквы. Записываю алгоритм и команды (табл. 2.1):

1. Букву К во вторую строку.
2. Букву К в клетку (2,4).
3. Букву К на место, а букву Т в четвёртую строку.
4. Букву Т в клетку (4,2).
5. Букву Т на место.

Теперь записываю команды.

Таблица 2.1

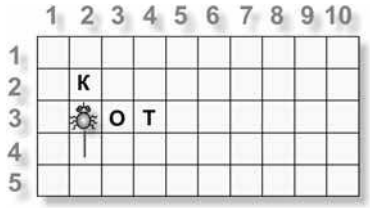
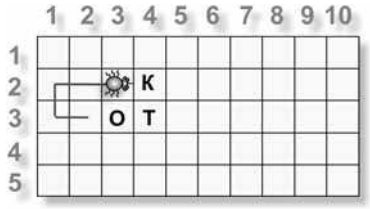
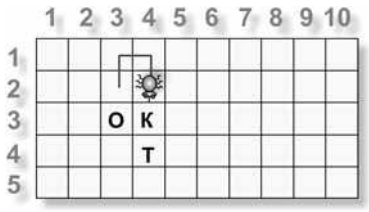
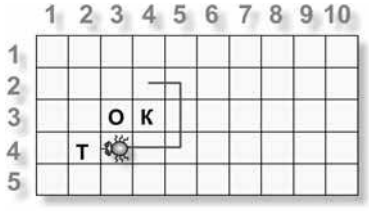
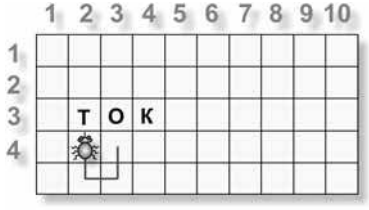
Шаги алгоритма	Команды	Среда исполнителя
1. Букву К во вторую строку	ВВЕРХ	
2. Букву К в клетку (2,4)	ВЛЕВО ВВЕРХ ВПРАВО ВПРАВО	

Таблица 2.1 *(окончание)*

Шаги алгоритма	Команды	Среда исполнителя
3. Букву К на место, а букву Т в четвёртую строку	ВВЕРХ ВПРАВО ВНИЗ	
4. Букву Т в клетку (4,2)	ВПРАВО ВНИЗ ВНИЗ ВЛЕВО ВЛЕВО	
5. Букву Т на место	ВНИЗ ВЛЕВО ВВЕРХ	

Петя. Прекрасно! Кукарача вернулся на то самое место, с которого начал свою работу. Это значит, что если мы сейчас повторим все команды с самого начала и до конца, то...

Вася. Ток опять превратится в кота, но только я бы поленился снова набирать все 16 команд ради удовольствия вернуть хвостатого на место. В конце концов, перед нами компьютер, у которого, как ты сам мне говорил, есть память. Почему мы её не используем, а вместо этого эксплуатируем мой детский труд?

Вопросы и задания

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

2.2. У компьютера есть память, и это хорошо

— На сегодня у нас такая программа намечается, — отвечает тётя. — Матроскина с Шариком мы бросаем в речку рыбу ловить. У нас по понедельникам будут рыбные дни. Пусть берут удочки и уходят удить.

Э. Успенский

На идею эксплуатации детского труда Петя отреагировал так.

Петя. Ну, ну, Вася, не заводись. Я всегда говорил, что лень — это двигатель прогресса. Что касается компьютерной памяти, тут ты прав. Сейчас покажу, как можно её использовать, решая алгоритмические задачи.

Вот справа на экране большое поле (рис. 2.3). Это редактор программ. В него можно один раз записать программу, а затем запускать её на выполнение столько раз, сколько потребуется, не набирая заново.

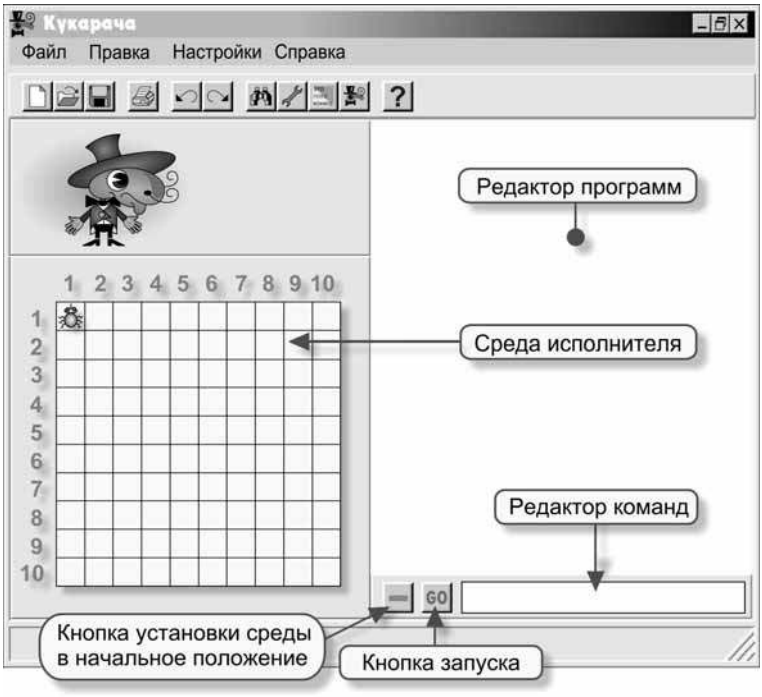


Рис. 2.3. Поле для записи программ

Вася. Хорошо, я наберу в этом поле программу, но как это сделать?

Петя. Войти в редактор программ можно, щёлкнув по его изображению мышкой, а программа записывается так:

```
это Имя
  ...
  команды
  ...
конец
```

Слова `это` и `конец` называют **ключевыми** словами. Между ними записываются команды.

Вася. А что за имя нужно написать после слова `это`?

Петя. Имя твоей программы. Это имя становится названием новой команды для Кукарачи, хотя такая команда, понятно, не входит в СКИ. Написав программу, ты можешь запустить её на выполнение, задав имя в редакторе команд.

Вася. Получается, что я учу Кукарачу новым командам! А какое имя ты посоветуешь мне сейчас написать?

Петя. Имя может быть любым, только оно не должно совпадать с названиями команд из СКИ и ключевыми словами. Внутри имени нельзя писать пробелы.

Назови программу `Перевертыш`, ведь исполнитель переворачивает слово задом наперёд.

Вася согласился с братом и набрал в редакторе программ свой текст.

Петя попросил Васю писать **комментарии** при записи программы. Он объяснил, что комментарии не обрабатываются компьютером, они записываются для удобства программиста. С комментариями программа становится более понятной — человеку её легче читать и отлаживать.

Для записи комментариев в программах Кукарачи используют две косых черты «//». Они сами и всё, что находится правее до конца строки, не принимаются во внимание при выполнении программы (рис. 2.4).

Петя. Ну вот, программа записана. Теперь набери в редакторе команд её имя и нажми кнопку *GO*.

Вася. Записываю `Перевертыш`, нажимаю *GO* (рис. 2.5).

Да, здорово! Кукарача как заводная игрушка трудится на поле, и, что удивительно, сначала вырабатывает ток, а при повторном запуске возвращает кота на место!

Петя. Так получается, потому что перед повторным запуском среда не приведена в начальное положение. Если сейчас снова нажать кнопку *GO* (не нажимая кнопку возврата в начальное положение), то на поле опять получится слово «ток».

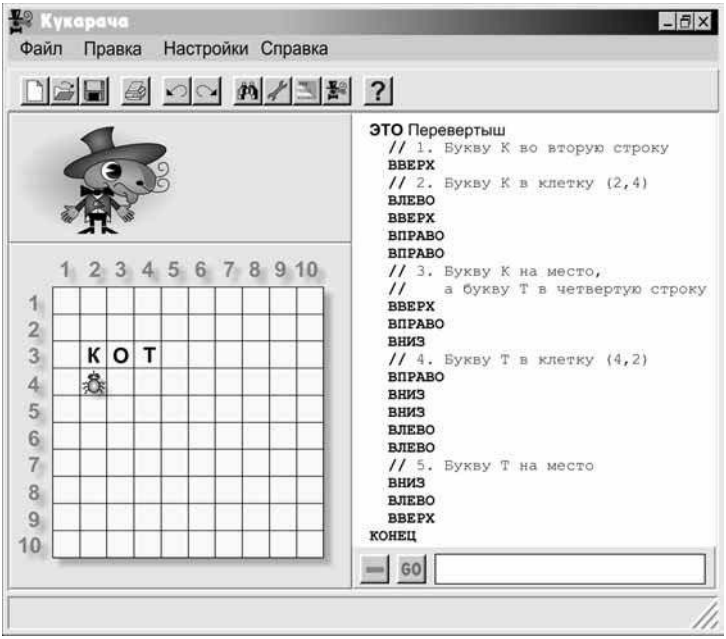


Рис. 2.4. Программа, записанная в программном поле

Рис. 2.5. Задана команда на выполнение

Вася щёлкает мышкой по кнопке *GO* и наблюдает за работой исполнителя.

Вася. Ток, конечно, получился, но что будет, если я его выключу?

Вопросы и упражнения

1. Что обозначают слова **это** и **конец** в программах Кукарачи?
2. Найдите ошибки в программе:

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

2.3. Неприятное свойство компьютерной памяти, которое исправляется наличием дисков

— Может быть, синяк ещё и не вскочит, — ответил Незнайка, смущённо вертя линейку в руках.

Н. Носов

Петя. Если ты выключишь компьютер, твой труд пропадёт, программа исчезнет из памяти, и завтра тебе придётся набирать её заново.

Вася. Помнится, ты намекал, что у компьютера есть магнитный диск для долговременного хранения информации. Говорил также, что информация хранится на диске подобно звуку на магнитной ленте нашего магнитофона. А ведь когда выключают магнитофон, на ленте ничего не пропадает.

Петя. Да, конечно, твою программу можно сохранить на диске, ты прав. Смышлёный у меня растёт брат! Меню окна практически любого компьютерного приложения содержит позицию *Файл* (рис. 2.6). Нужно щёлкнуть в этой позиции и сохранить работу на диске.

Рис. 2.6. Меню окна приложения «Кукарача»

Вася. Я запишу свою программу на диск под именем `kuk1`. С одной стороны, Кук — это моя фамилия, а с другой — сокращение от слова «Кукарача». Цифра «1» будет обозначать первую мою программу.

2.4. Программа может содержать много процедур

— Ну-ка, тащите сюда гаечный ключ, плоскогубцы, молоток и паяльник, — сказал Винтик малышкам.

Н. Носов

Вася. Мне понравилось писать программы. С их помощью можно научить Кукарачу новым командам. Плохо только, что программу для каждой такой команды придётся записывать в отдельный файл.

Петя. Совсем необязательно. Языковая конструкция `ЭТО ... КОНЕЦ`, которую мы сегодня рассматривали, называется **процедурой**. Программа может состоять из нескольких процедур, и каждую из них можно запускать на выполнение отдельно.

Вася. Понятно, ведь каждая процедура имеет своё собственное имя. Эти имена и будут именами «новых» команд исполнителя.

Петя. Всё правильно. Ну, ты ещё можешь повозиться с Кукарачей, а у меня сегодня много других дел...

2.5. Задачи

Для решения задач напишите программы, запустите их на выполнение, сохраните на диске.

1. Что кричит кукушка (рис. 2.7)? (КУКУ)

Рис. 2.7

2. Получите из этого неизвестного зверя домашнее животное (рис. 2.8).

Рис. 2.8

3. Как получить из пяти ног одно пятно (рис. 2.9)?

Рис. 2.9

4. Кому поёт Кукарача (рис. 2.10)? (ЮРИЮ)

Рис. 2.10

5. Исправьте пример, переместив один из кубиков (рис. 2.11).

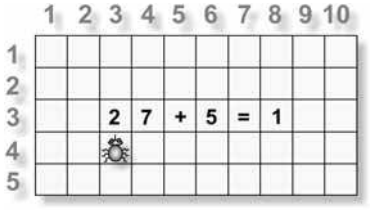


Рис. 2.11

Глава 3

Новые команды и их повторение

3.1. Кукарача говорит «Ах!»

— Ах, как интересно было! — подхватила Снежинка. — Один малыш упал в реку и чуть не утонул, но его вытащили из воды.

Н. Носов

Петя. Посмотри, какую программу я написал (рис. 3.1):

```
ЭТО Ух
   Ах
КОНЕЦ
```

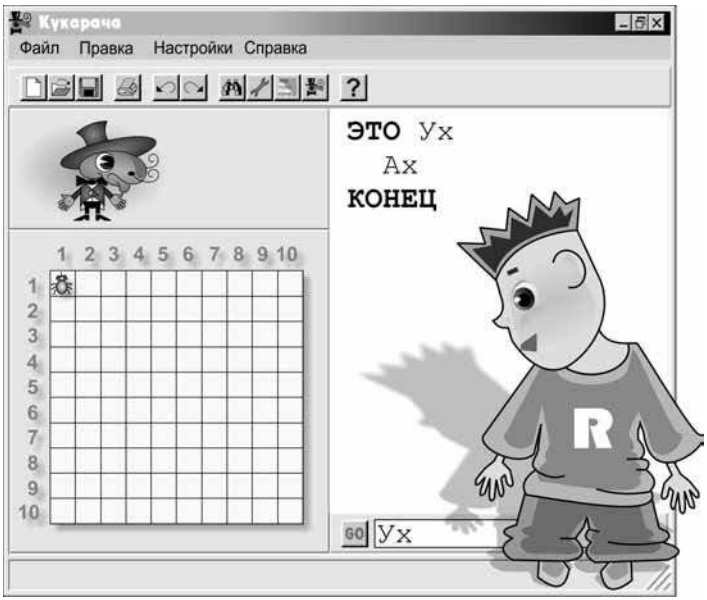


Рис. 3.1. Программа «Ух»

Вася. Ух!

Петя. Как ты думаешь, что произойдёт, если я сейчас нажму кнопку *GO*?

Вася. Думаю, Кукарача скажет «Ах!».

Петя. Ты почти угадал.

Петя щёлкнул по кнопке *GO* мышкой, Кукарача не стронулся с места, а на экране появилось окошко с надписью (рис. 3.2).

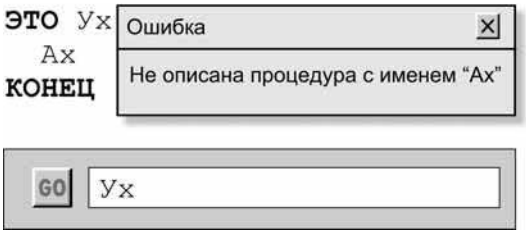


Рис. 3.2. Аварийное сообщение

Петя. Команды `Ах` нет в СКИ исполнителя. Нет описания этой команды и в программном поле. Вот почему появилось это сообщение.

А теперь я дополню программу так:

```
ЭТО Ух
  Ах // Это вызов процедуры Ах
КОНЕЦ


// Это описание процедуры Ах
ЭТО Ах
  ВПРАВО
  ВНИЗ
  ВЛЕВО
  ВВЕРХ
КОНЕЦ
```

Теперь всё в порядке. По команде `Ух` Кукарача выполнит процедуру `Ах`, т. е. совершит небольшую прогулку и вернётся в исходное положение (рис. 3.3).

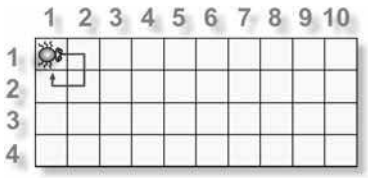


Рис. 3.3. Прогулка Кукарачи

Вася. Ты всё понятно объяснил, но только зачем такие сложности?

Петя. Э, брат! Сейчас ты всё поймёшь. Использование имён процедур в качестве команд существенно облегчает программирование.

Вопросы и задания

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

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

3.2. Процедурное программирование

> — Дай мне поездить на автомобиле. Я тоже хочу научиться управлять.
> — Ты не сумеешь, — сказал Торопыжка. — Это ведь машина. Тут понимать надо.
> — Чего тут ещё понимать! — ответил Незнайка. — Я видел, как ты управляешь. Дёргай за ручки да верти руль. Всё просто.
>
> *Н. Носов*

Петя. Вот, к примеру, такая задача.

Задача «НАРЫЧАЛО»

НАРЫЧАЛО зарычало, превратилось вдруг в НАЧАЛО. (Получить на поле слово НАЧАЛО (рис. 3.4).)

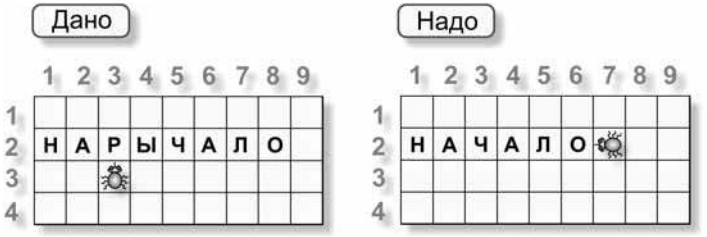


Рис. 3.4

Вася. Предлагаю такой алгоритм (табл. 3.1).

Таблица 3.1

Шаги алгоритма
1. Убрать лишние буквы РЫ
2. Идти в конец слова
3. Придвинуть ЧАЛО к НА

Петя. Хорошо. Давай перепишем твой алгоритм прямо на поле программы:

```
ЭТО Начало
  // 1. Убрать лишние буквы РЫ
  // 2. Идти в конец слова
  // 3. Придвинуть ЧАЛО к НА
КОНЕЦ
```

Вася. Ты написал процедуру, состоящую из одних комментариев!

Петя. Да, но зато комментарии содержат описания шагов алгоритма. Теперь после каждого комментария-шага я напишу вызов процедуры, которая будет выполнять работу, описанную в этом комментарии:

```
ЭТО Начало
  // 1. Убрать лишние буквы РЫ
  Убрать_РЫ
  // 2. Идти в конец слова
  Идти_в_конец
```

```
  // 3. Придвинуть ЧАЛО к НА
  Уплотнить
КОНЕЦ
```

Петя. Заметь, что когда имя процедуры состоит из нескольких слов, я соединяю их символом подчёркивания.

Вася. Думаю, ты это делаешь потому, что в именах нельзя использовать пробелы.

Петя. Верно! Теперь приступаем к описанию процедуры `Убрать_РЫ`. Я изобразил на рисунке 3.5, что она должна делать.

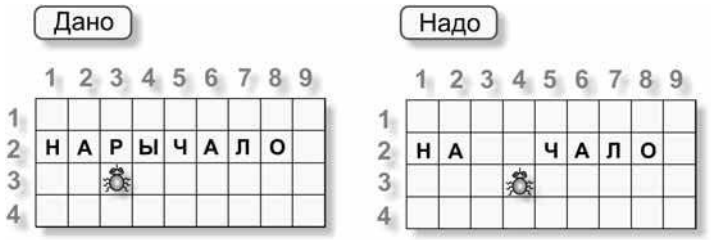


Рис. 3.5

Вася. Попробую программировать твоим способом:

```
ЭТО Убрать_РЫ
  Убрать_букву
  ВПРАВО
  Убрать_букву
КОНЕЦ


ЭТО Убрать_букву
  ВВЕРХ
  ВВЕРХ
  ВНИЗ
  ВНИЗ
КОНЕЦ
```

Вася. Вот теперь я понял преимущество процедур! Их вызов можно использовать многократно, и это здорово экономит усилия программиста!

Петя. Преимущество **процедурного программирования** ещё и в том, что программы становятся более простыми и наглядными, так как раскладываются на множество маленьких процедур, описывающих простые и короткие действия.

Вася. Где ты научился говорить так мудро?

Петя. Забываешь, что я учусь в университете, а ты пока школьник, хотя и смышлёный!

Вася. Приступаю к процедуре `Идти_в_конец` (рис. 3.6).

```
ЭТО Идти_в_конец
  // Движение по строке в клетку (3,9)
  ВПРАВО
  ВПРАВО
  ВПРАВО
  ВПРАВО
  ВПРАВО
  // На одну строку вверх в клетку (2,9)
  ВВЕРХ
КОНЕЦ
```

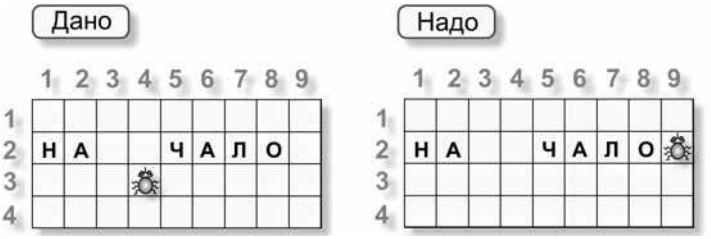


Рис. 3.6

Петя. Осталось написать процедуру `Уплотнить`.

Вася. Ну это совсем просто (рис. 3.7).

```
ЭТО Уплотнить
  ВЛЕВО
  ВЛЕВО
КОНЕЦ
```

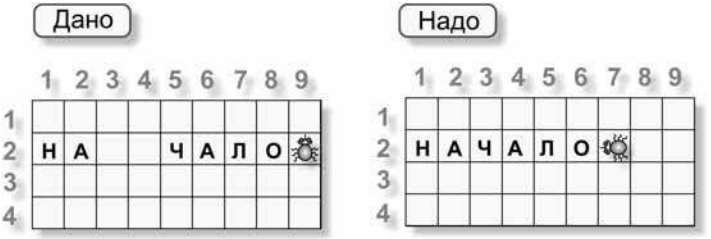


Рис. 3.7

Вопросы и задания

1. В чём заключается преимущество процедурного программирования?
2. Дайте обоснование выбранным именам процедур в задаче «НАРЫЧАЛО».

3.3. Команда повторения

> Когда эта песня кончилась, они затянули другую, потом ещё и ещё.
>
> *Н. Носов*

Петя. Покажу, как можно ещё упростить нашу программу. Управляя исполнителем, часто используют **команду повторения** (**команду цикла**). В языке Кукарачи эта команда имеет вид:

```
ПОВТОРИ число команда
```

ПОВТОРИ — это ключевое слово языка, «число» — это целое положительное число, обозначающее число повторений, «команда» — это любая команда языка, которую надо повторять. Например, вместо пяти команд **ВПРАВО** в процедуре Идти_в_конец можно написать одну команду цикла:

```
ПОВТОРИ 5 ВПРАВО
```

Вася. Это действительно облегчение! Сейчас я исправлю нашу программу:

```
ЭТО Начало
  // 1. Убрать лишние буквы РЫ
  Убрать_РЫ
  // 2. Идти в конец слова
  Идти_в_конец
  // 3. Придвинуть ЧАЛО к НА
  Уплотнить
КОНЕЦ

ЭТО Убрать_РЫ
  Убрать_букву
  ВПРАВО
  Убрать_букву
КОНЕЦ

ЭТО Убрать_букву
  ПОВТОРИ 2 ВВЕРХ
```

```
  ПОВТОРИ 2 ВНИЗ
КОНЕЦ

ЭТО Идти_в_конец
  // Движение по строке в клетку (3,9)
  ПОВТОРИ 5 ВПРАВО
  // На одну строку вверх в клетку (2,9)
  ВВЕРХ
КОНЕЦ

ЭТО Уплотнить
  ПОВТОРИ 2 ВЛЕВО
КОНЕЦ
```

Вопросы и задания

1. Посмотрите на следующие рисунки (рис. 3.8).
   1.1. Найдите в каждом рисунке повторяющиеся части.
   1.2. Можно ли выделить такую часть, повторением которой образован весь рисунок?

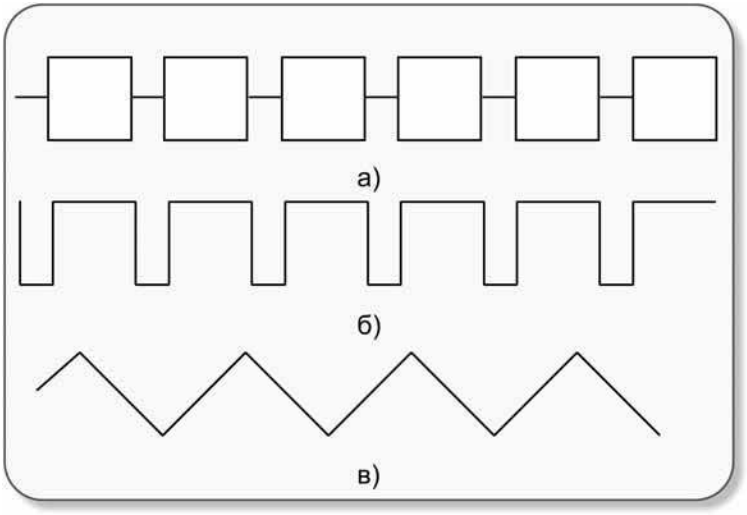


Рис. 3.8

2. Какие из приведённых далее команд повторения правильные, а какие — нет и почему?
   2.1. **ПОВТОРИ** 7 **СТОЯТЬ**
   2.2. **ПОВТОРИ** 2 **ВПРАВО**
   2.3. **ПОВТОРИ** 5-2 **ВНИЗ**
   2.4. **ПОВТОРИ** 100 **ВНИЗ ВПРАВО**
   2.5. **ПОВТОРИ** 8 Колесо
   2.6. **ПОВТОРИ** 7 **ПОВТОРИ** 2 **ВПРАВО**
   2.7. **ПОВТОРИ** 1 **ВПРАВО**

3.4. Интерпретатор и его странные сообщения

> «Кот Матроскин подойти к телефону не может. Он очень занят. Он на печи лежит».
>
> *Э. Успенский*

Петя. Как ты думаешь, понимает ли Кукарача команду **ПОВТОРИ**?

Вася. Конечно, понимает! Ведь я запустил программу, и он её выполнил, не сказав плохих слов.

Петя. Но ведь команда **ПОВТОРИ** не входит в СКИ исполнителя!

Вася. Да, действительно. В чём же тогда дело?

Петя. Дело в том, что программу выполняет не Кукарача, а **интерпретатор программ**. Он выполняет программу и в соответствии с ней передаёт Кукараче команды из его СКИ.

Вася. Что ты называешь интрепатором..., тьфу! интерпретатором? Это сам компьютер?

Петя. Нет. Ты видишь на экране Кукарачу, его среду, поле программы, разные кнопки, окна. Всё это появляется тогда, когда ты запускаешь компьютерную модель исполнителя и начинаешь с ней работать. Компьютерная модель Кукарачи — это программа для нашего компьютера, мы купили её у фирмы «Роботландия». Она состоит из многих частей-процедур, которые подобно твоим процедурам для Кукарачи выполняют определённую для них часть работы. Одна процедура отвечает за вывод на экран картинок, другая записывает информацию на диск, третья обрабатывает нажатие клавиш на клавиатуре компьютера. Интерпретатор программ Кукарачи — это одна из процедур этой программы. Именно она читает написанную тобой программу с «чужими» для Кукарачи словами: **ЭТО**, **КОНЕЦ**, **ПОВТОРИ** и выполняет её.

Вася. Как работает интерпретатор программ?

Петя. Интерпретатор сначала проверяет, нет ли в программе **синтаксических ошибок**, а затем начинает выполнять программу, передавая Кукараче понятные ему команды из его СКИ.

Например, выполняя команду **ПОВТОРИ** 5 **ВПРАВО**, интерпретатор передаст Кукараче последовательно пять команд **ВПРАВО** (рис. 3.9).

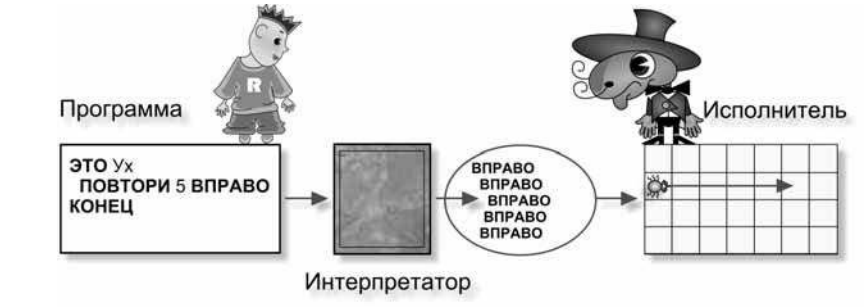


Рис. 3.9. Схема работы интерпретатора программ

Вася. Ты меня сегодня утомил сложными словами. Что такое синтаксическая ошибка?

Петя. Ничего, брат, привыкай. Выучил же ты слово «процессор», и эти новые слова как-нибудь запомнишь. Синтаксические ошибки — это разные описки, неточности, нарушения правил построения программ. Синтаксической ошибкой будет запись НАЛЕВО, вместо команды **ВЛЕВО**, слово ТОТ вместо ключевого слова **ЭТО** и так далее.

Вася. Сказал бы просто «ошибки», зачем ещё добавлять «синтаксические»?

Петя. Потому что существуют **логические ошибки**.

Вася. А это что такое?

Петя. Это когда программа написана правильно с точки зрения интерпретатора, но делает совсем не то, что ты хотел. Иными словами, логическая ошибка — это неправильно составленный алгоритм. Вот фраза «*для того, чтобы выключить компьютер, надо ударить по нему молотком*» написана совершенно правильно с точки зрения русского языка. В ней нет синтаксических ошибок, но логическая ошибка налицо!

Вася. Я буду писать свои программы совсем без ошибок, и тогда твои мудрые объяснения мне не потребуются.

Петя. Зелен ты, брат, не знаешь тяжёлого труда программиста! Писать программы можно хорошо и плохо, но без ошибок — почти никогда!

Вася. Убил наповал!

Петя. Синтаксическую ошибку не пропустит интерпретатор программ. Он обязательно выдаст окно с сообщением и выделит то место программы, где обнаружил ошибку.

Вася. Это уже хорошо.

Петя. Хорошо, но к сообщениям интерпретатора надо привыкнуть. К тому же место, где указана ошибка, не всегда то, где она на самом деле находится.

Вася. Неужели «Роботландия» сделала такой плохой интерпретатор?

Петя. Ха! Ведь интерпретатор не знает, что *хотел* написать программист, поэтому вынужден выдавать сообщения очень формально. Посмотри на следующую программу:

```
ЭТО Пример
  ВЛЕВА
КОНЕЦ
```

Интерпретатор выдаст сообщение «*Не описана процедура с именем ВЛЕВА*» (рис. 3.10). Откуда ему знать, что ты написал команду из СКИ с ошибкой?

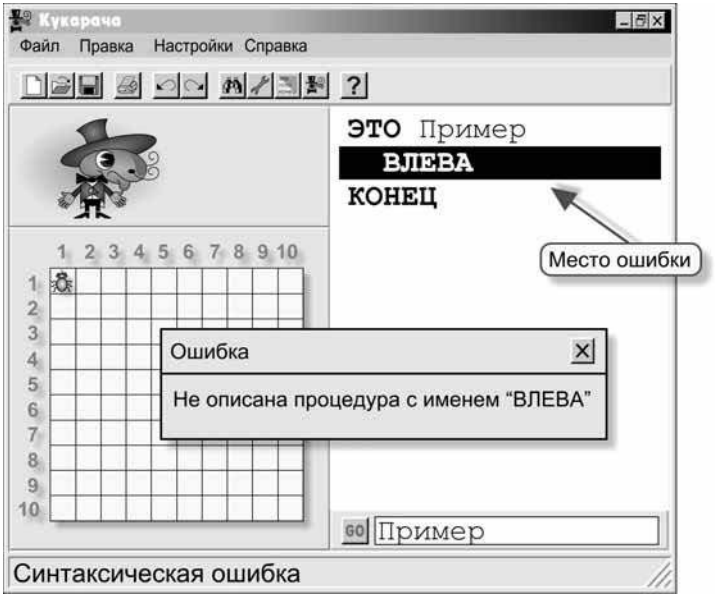


Рис. 3.10. Сообщение о синтаксической ошибке в программе

Вася. А уж в поиске логических ошибок помощи от интерпретатора совсем не жди!

Петя. Нет, он может здорово помочь, если запустить его в *пошаговом режиме*.

Вася. Интерпретатор умеет ходить?

Петя. Интерпретатор умеет работать в двух режимах — **непрерывном** и **пошаговом**. В непрерывном режиме он выполняет программу до конца, не останавливаясь, а в пошаговом прерывает работу после каждой команды, отмечая подсветкой то, что ему предстоит выполнить на следующем шаге (рис. 3.11). Наблюдая за тем, *что* выполняется на поле программ и *как* работает Кукарача, легко найти место логической ошибки.

Рис. 3.11. Работа интерпретатора в пошаговом режиме

Вопросы и задания

1. Используя сообщения интерпретатора, исправьте синтаксические ошибки в следующей программе:

```
ЭТО Пример 1
  Вправо
  ПОВТОРИТЬ 2 ВНИЗ
  ВЛЕВО
  ПОВТОРИ 2 раза ВВЕРХ
КОНЕЦ.
```

2. Используя пошаговый режим, найдите и исправьте логические ошибки в решении следующей задачи.

Задача

Превратите слово ЛИПА в слово ПИЛА (рис. 3.12).

Рис. 3.12

Решение

```
ЭТО Упражнение
  // 1.  Л в клетку (2,2)
  Часть_1
  // 2. Л в клетку (2,4)
  Часть_2
  // 3. Л в клетку (3,4)
  Часть_3
  // 4. В четвертую строку
  Часть_4
  // 5. П в клетку (4,2)
  Часть_5
  // 6. П в клетку (3,2)
  Часть_6
КОНЕЦ

// 1. Л в клетку (2,2)
ЭТО Часть_1
  ВВЕРХ
КОНЕЦ

// 2. Л в клетку (2,4)
ЭТО Часть_2
  ВЛЕВО
  ВВЕРХ
  ПОВТОРИ 3 ВПРАВО
КОНЕЦ
```

```
// 3. Л в клетку (3,4)
ЭТО Часть_3
  ВВЕРХ
  ВЛЕВО
  ВНИЗ
КОНЕЦ

// 4. В четвертую строку
ЭТО Часть_4
  ВВЕРХ
  ПОВТОРИ 2 ВПРАВО
  ПОВТОРИ 2 ВНИЗ
КОНЕЦ

// 5. П в клетку (4,2)
ЭТО Часть_5
  ПОВТОРИ 4 ВПРАВО
КОНЕЦ

// 6. П в клетку (3,2)
ЭТО Часть_6
  ВНИЗ
  ВЛЕВО
  ВВЕРХ
КОНЕЦ
```

3.5. Задачи

1. Помогите Кукараче взобраться по лесенке (рис. 3.13).

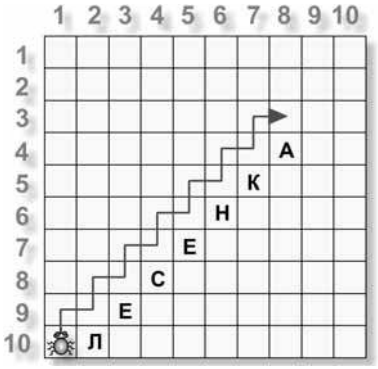


Рис. 3.13

2. Составьте программу прогулки Кукарачи по заданному маршруту так, чтобы он прошёл его пять раз (рис. 3.14).

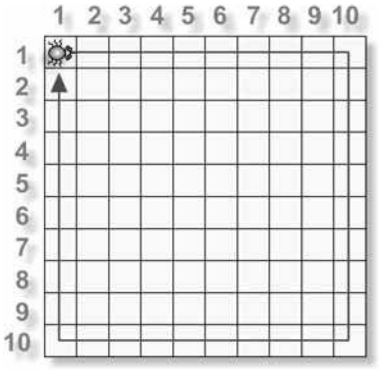


Рис. 3.14

3. Контроль поля. Заставьте исполнителя обойти все клетки поля так, чтобы в каждой из них он побывал только один раз.
4. Собрать слово ПАРОХОД (рис. 3.15).

Рис. 3.15

5. Получить на поле слово КОЛЕСО и закончить работу в первом столбце (рис. 3.16).

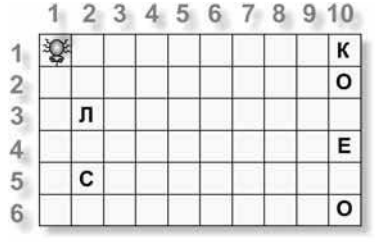


Рис. 3.16

6. Собрать слово МАШИНИСТ на 10 строке и закончить работу в клетке (1,10) (рис. 3.17).
7. Помогите Кукараче очистить поле от мусора (кубики X), не сдвигая остальных кубиков (рис. 3.18).

Рис. 3.17

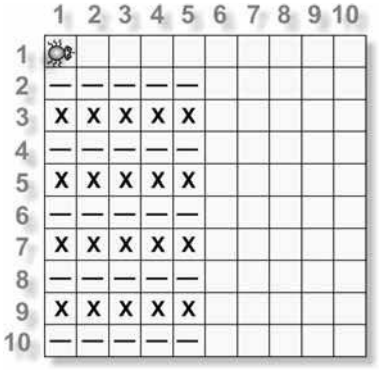


Рис. 3.18

8. Помогите Кукараче совершить прогулку в клетку (1,10), не сдвигая кубиков (рис. 3.19).

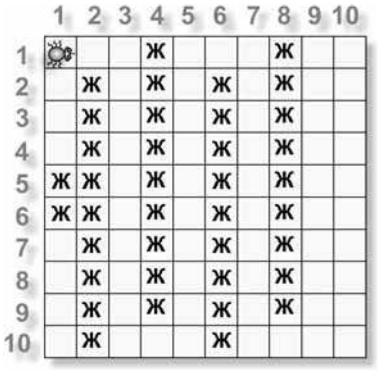


Рис. 3.19

Глава 4

Кукарача на распутье

4.1. Команда ветвления

— Ну, что это такое! — проворчал Ворчун. То встаньте, то лягте, то дышите, то не дышите!

Н. Носов

Вася. Если завтра будет хорошая погода, пойдём на рыбалку?

Петя. Хорошо! А если кубик на поле Кукарачи перевёрнут, то как тогда писать программу (рис. 4.1)?

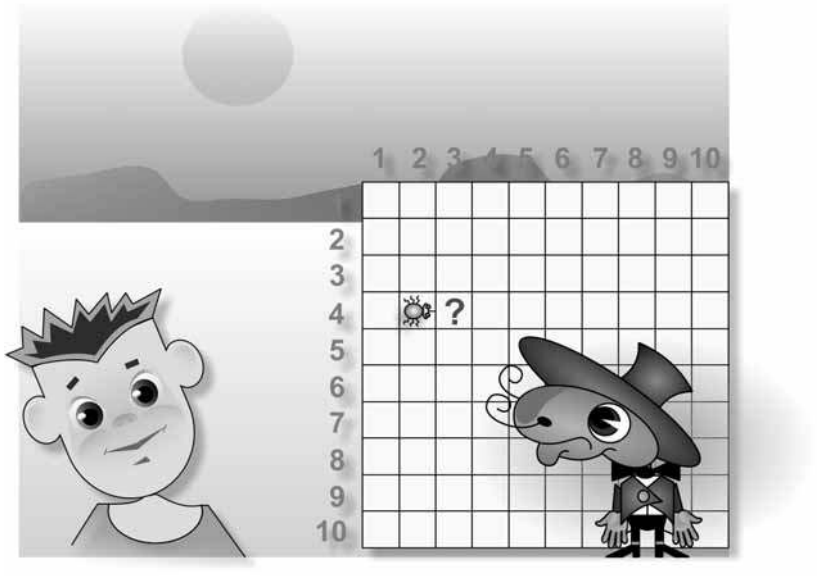


Рис. 4.1. Мечты о рыбалке

Вася. Это шутка?

Петя. Это приглашение заняться программированием условий. Посмотри на следующую задачу.

Задача 1

Помогите Кукараче собрать два слова: название животного и место, где этому животному нравится жить. На перевёрнутом кубике (на рис. 4.2 он изображён знаком вопроса) одна из букв О или И.

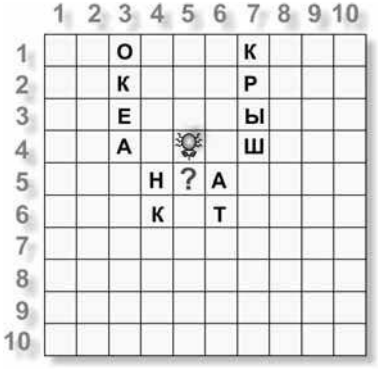


Рис. 4.2

Вася. Я понял! Если это буква О, то надо собрать слова КОТ и КРЫША, а если буква И — КИТ и ОКЕАН. Давай перевернём кубик и посмотрим, что на нём написано.

Петя. Считай, что поле Кукарачи недоступно. Например, оно подвергнуто радиоактивному заражению. Кукарача — робот, и он может там работать, а человек — нет. Но можно управлять исполнителем при помощи радио и компьютера.

Вася. Может ли исполнитель как-нибудь узнать, что написано на кубике?

Петя. Может. Когда Кукарача толкает кубик, он переворачивает его и видит надпись.

Вася. Ну, тогда напишу команду `вниз`. Кукарача толкнёт кубик, и я увижу надпись. После этого буду знать, как управлять исполнителем дальше (рис. 4.3).

Петя. Хотелось бы, чтобы робот работал автоматически без твоего присутствия, непрерывно, по заранее написанной программе. Представь, что Кукарачу отправили на Марс, связи с ним нет и увидеть, как он переворачивает кубики, нельзя.

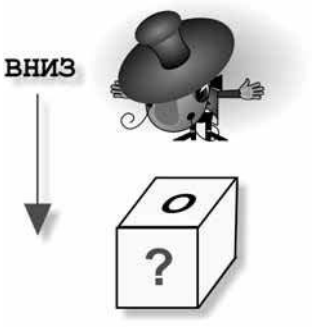


Рис. 4.3

Вася. Даже если бы связь с Марсом была, она не быстрая, ведь сигнал от Марса идёт так долго.

Петя. Верно! А роботу нужно принимать решение немедленно, он не должен ждать с Земли твоих сигналов.

Вася. Но я не знаю, как написать программу, которая бы принимала решение за меня!

Петя. Правильно, не знаешь. Это потому, что ты не знаком с **командой ветвления**. Посмотри, как она записывается:

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

Слова **ЕСЛИ**, **ТО**, **ИНАЧЕ** — ключевые слова языка программирования. Они как цемент связывают команду ветвления воедино. Работает новая команда так: интерпретатор программ проверяет условие и, если оно верное, выполняет первую команду, если ложное — вторую. В любом случае работает только одна команда, либо `команда1`, либо `команда2` (рис. 4.4).

Вася. А как записывается условие?

Петя. Обычно в качестве условия записывают символ (букву, цифру или другой знак). Интерпретатор будет сравнивать этот символ с тем, который перевернул Кукарача.

Вася. Попробую написать программу для нашей задачи.

```
// Программа для местности, заражённой радиацией
ЭТО Звери
  // Толкнуть кубик (он переворачивается, и Кукарача видит надпись).
  ВНИЗ
  // Проверить, что находится на кубике и выполнить нужную команду.
  ЕСЛИ И // Интерпретатор сравнивает И с тем,
```

```
        // что Кукарача увидел на кубике.
    ТО    ВЛЕВО   // На кубике И — собираем слово ОКЕАН
    ИНАЧЕ ВПРАВО  // На кубике О — собираем слово КРЫША
КОНЕЦ
```

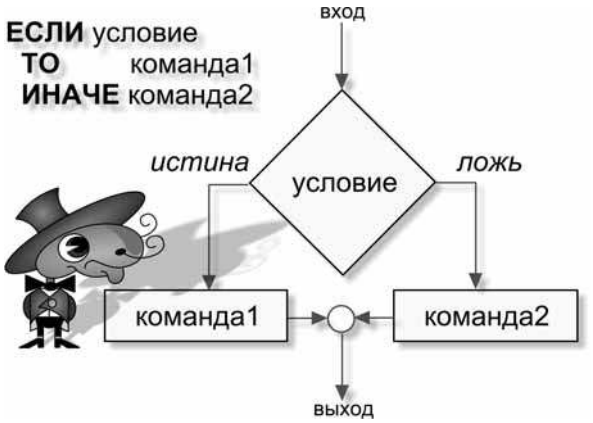


Рис. 4.4. Схема работы команды ветвления

Петя. Написано правильно. Теперь расскажи, как интерпретатор выполнит эту программу.

Вася. Сначала он передаст исполнителю команду **вниз**. Кукарача толкнёт кубик и увидит надпись на нём. А дальше…

Петя. Дальше интерпретатор, выполняя команду **если**, пошлёт Кукараче запрос: что на кубике? Кукарача ответит, и в зависимости от ответа интерпретатор передаст исполнителю либо команду **влево**, либо команду **вправо** (рис. 4.5).

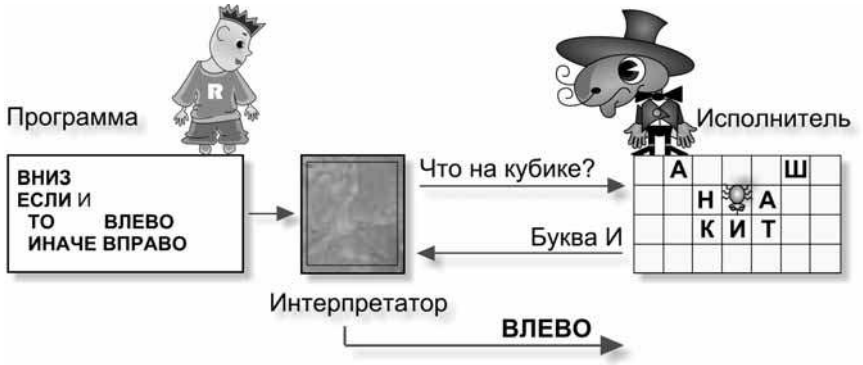


Рис. 4.5. Исполнитель ответил на вопрос интерпретатора

Вася. То есть интерпретатор будет работать за меня? Он спросит исполнителя, что тот видит на кубике, а затем, в соответствии с командой ветвления, задаст ему ту или иную работу.

Петя. Да. Только интерпретатор, как и Кукарача, не боится радиации. Он может даже на Марсе выполнять ту работу, которую ты запрограммировал в своём уютном кресле.

Вася. Вот она — сила программирования вообще и робототехники в частности!

Петя. Как ты думаешь, сколько запусков программы `Звери` нужно сделать, чтобы убедиться, что программа работает правильно?

Вася. Думаю, что два. Сначала надо проверить программу, когда на перевёрнутом кубике записана буква О, и второй раз, когда на перевёрнутом кубике — буква И.

Петя. Правильно. Вот ещё одна задача на ветвление.

Задача 2

На перевёрнутом кубике — либо буква Н, либо буква Р. Кукараче надо составить одно из слов — КИНО или ЦИРК (рис. 4.6).

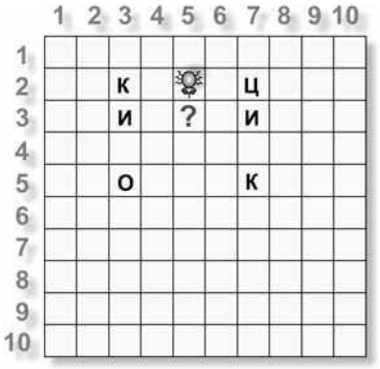


Рис. 4.6

Вася. Первая команда понятна: **вниз**. Кукарача толкнёт кубик и увидит на нём букву. Её надо поставить на место, но за один шаг Кукарачи это не сделать. Потребуется несколько команд. Можно ли в команде ветвления после слов **то** и **иначе** писать много команд?

Петя. Нет. Так же, как и в команде повторения. Но ведь тебе уже приходилось заменять группу команд одной.

Вася. Придётся писать процедуру. Даже две: для каждого варианта буквы — свою. Назову их Кино и Цирк. Вот моё решение:

```
ЭТО Отдых
  // Толкнуть кубик и посмотреть, что на нём
  ВНИЗ
  // Развилка
  ЕСЛИ Н
    ТО    Кино // Обнаружена буква Н
    ИНАЧЕ Цирк // Буквы Н нет
КОНЕЦ

// Сборка слова КИНО
ЭТО Кино
  ВПРАВО
  ВНИЗ
  ПОВТОРИ 2 ВЛЕВО
КОНЕЦ

// Сборка слова ЦИРК
ЭТО Цирк
  ВЛЕВО
  ВНИЗ
  ПОВТОРИ 2 ВПРАВО
КОНЕЦ
```

Вопросы и задания

1. Когда в программировании не обойтись без команды ветвления?
2. На перевёрнутом кубике одна из букв Е, Д, Л. Помогите Кукараче посадить на поле дерево (рис. 4.7).

Рис. 4.7

4.2. Особые случаи

— А потом я у них чай пил и незаметно пуговицу отрезал от курточки. Посмотрите, ваша ли это пуговица. Если пуговица ваша, значит, и мальчик ваш.

Э. Успенский

Петя. Давай закончим разговор про команду ветвления обзором особых случаев. Может случиться, что при верном условии исполнителю не надо делать никаких действий. Тогда после слова **то** записывают команду **стоять**. Вот задача, которая это поясняет.

Задача 3

Если на кубике находится подходящая буква, соберите слово ОЛЕНЬ, в противном случае пусть на поле останется слово ЛЕНЬ (рис. 4.8).

Рис. 4.8

Решение

```
ЭТО Олень
  ВВЕРХ
  ЕСЛИ О
    ТО    СТОЯТЬ
    ИНАЧЕ ВВЕРХ
КОНЕЦ
```

Может случиться по-другому: делать ничего не надо, когда условие не выполняется. В этом случае после ключевого слова **иначе** можно поместить пустую команду **стоять** либо опустить ветвь **иначе** вовсе.

Задача 4

Если можно, поставьте на поле столб (рис. 4.9).

Рис. 4.9

Решение

```
// 1 вариант
ЭТО Столб
  ВЛЕВО
  ЕСЛИ Б
    ТО    ВЛЕВО
    ИНАЧЕ СТОЯТЬ
КОНЕЦ
```

```
// 2 вариант
ЭТО Столб
  ВЛЕВО
  ЕСЛИ Б
    ТО ВЛЕВО
КОНЕЦ
```

Вася. Наконец ты объяснил, для чего нужна команда СТОЯТЬ!

Петя. Небольшое дополнение. В качестве условия можно использовать ключевое слово **ПУСТО**. Это условие будет истинным, когда в клетке, куда переполз исполнитель, нет кубика, и ложным в противном случае.

Задача 5

Если справа от Кукарачи стоит какой-нибудь кубик, удалить его с поля (рис. 4.10).

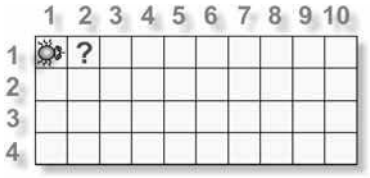


Рис. 4.10

Решение

```
ЭТО Удаление
  ВПРАВО
  ЕСЛИ ПУСТО
    ТО    СТОЯТЬ
    ИНАЧЕ ПОВТОРИ 8 ВПРАВО
КОНЕЦ
```

Вася. Ты же говорил, что в команде ветвления после слов **ТО** и **ИНАЧЕ** можно писать только одну команду?

Петя. Так оно и есть. После слова **ИНАЧЕ** записана ровно одна команда — команда повторения.

Вася. Да, действительно. Но тогда после слов **ТО** и **ИНАЧЕ** можно записывать и команды ветвления?

Петя. Конечно. Вот пример.

Задача 6

Под Кукарачей, во втором столбце расположена цифра. Её значение больше двух. Определить, какая это цифра, и поставить её в строку с соответствующим номером (рис. 4.11).

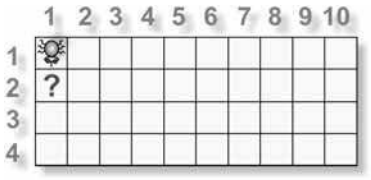


Рис. 4.11

Решение

```
ЭТО Установка_цифры
  ВНИЗ
  ЕСЛИ 4
    ТО    ВНИЗ
    ИНАЧЕ ЕСЛИ 5
            ТО    ПОВТОРИ 2 ВНИЗ
            ИНАЧЕ ЕСЛИ 6
                    ТО    ПОВТОРИ 3 ВНИЗ
```

```
                ИНАЧЕ ЕСЛИ 7
                        ТО    ПОВТОРИ 4 ВНИЗ
                        ИНАЧЕ ЕСЛИ 8
                                ТО ПОВТОРИ 5 ВНИЗ
                                ИНАЧЕ ЕСЛИ 9
                                        ТО ПОВТОРИ 6 ВНИЗ
КОНЕЦ
```

Вася. Выглядит страшновато.

Петя. Обычно такие конструкции (их называют **переключателями**) записывают так:

```
ЭТО Установка_цифры
  ВНИЗ
  ЕСЛИ       4 ТО ВНИЗ
  ИНАЧЕ ЕСЛИ 5 ТО ПОВТОРИ 2 ВНИЗ
  ИНАЧЕ ЕСЛИ 6 ТО ПОВТОРИ 3 ВНИЗ
  ИНАЧЕ ЕСЛИ 7 ТО ПОВТОРИ 4 ВНИЗ
  ИНАЧЕ ЕСЛИ 8 ТО ПОВТОРИ 5 ВНИЗ
  ИНАЧЕ ЕСЛИ 9 ТО ПОВТОРИ 6 ВНИЗ
КОНЕЦ
```

Вася. Да, это совсем другое дело.

Петя. Хотя в это и трудно поверить, но в последней программе записано то же самое, что и в предыдущей, но по-другому отформатировано. Эти две программы наглядно показывают, как важно правильно оформлять тексты. Интерпретатору программ оформление безразлично (так же, как и комментарии), а человеку — нет.

Вася. Понятно! Надеюсь, ты уже всё рассказал про команду ветвления?

Петя. Ещё одна важная деталь. При записи условия можно использовать ключевое слово **НЕ**. Это слово означает, что интерпретатор будет проверять условие наоборот: когда символ на кубике *не совпадает* с символом, указанным после слов **ЕСЛИ НЕ**, выполняется команда, стоящая вслед за словом **ТО**. В качестве примера я приведу другое решение задачи про оленя:

```
ЭТО Олень
  ВВЕРХ
  ЕСЛИ НЕ О
    ТО ВВЕРХ
КОНЕЦ
```

Вася. Да, это ключевое слово упростило запись программы.

Петя. Я предлагаю тебе самому решить побольше задач на ветвление, чтобы, как говорится, набить руку.

Вася. А заодно и голову...

Вопросы и задания

1. На перевёрнутых кубиках (рис. 4.12) буквы О и У (какая буква на каком кубике — неизвестно).

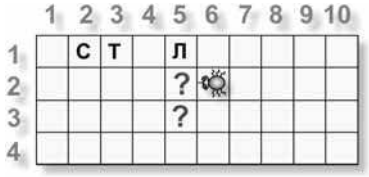


Рис. 4.12

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
```

4.3. Задачи

1. На одном из перевёрнутых кубиков буква И. Составьте слово ИГРА (рис. 4.13).

Рис. 4.13

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 (рис. 4.14)? (На перевёрнутом кубике либо X, либо 0. Кукарача должен поставить этот кубик в выигрышную позицию.)

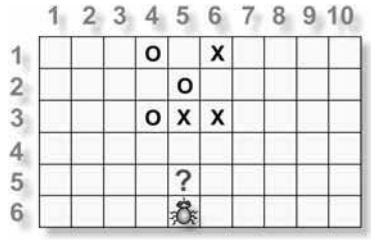


Рис. 4.14

3. На перевёрнутом кубике либо буква З, либо буква Л. Помогите Кукараче совершить космическое путешествие (рис. 4.15).

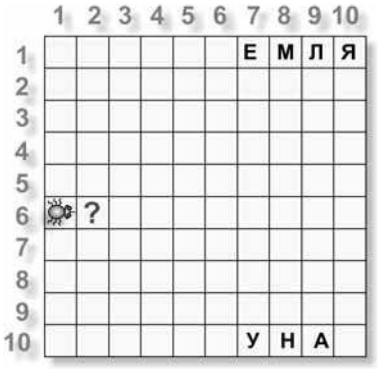


Рис. 4.15

4. На одном из кубиков — буква К. Составьте слово КЛАД (рис. 4.16).

Рис. 4.16

5. На одном из кубиков — буква М. Составить слово МЕДВЕДЬ и вернуть Кукарачу в исходную позицию. Лишнюю букву выбросить за пределы поля (рис. 4.17).

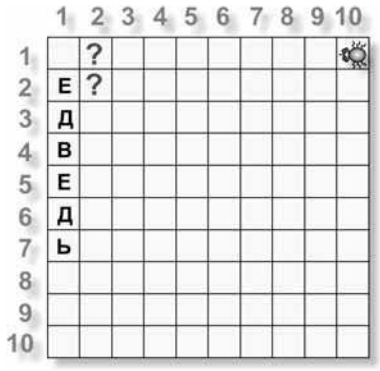


Рис. 4.17

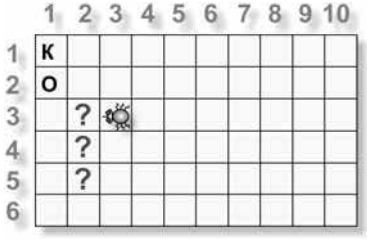


Рис. 4.18

6. Помогите Кукараче составить слово КОТ. На одном из кубиков есть буква Т (рис. 4.18).
7. Во второй строке злоумышленник перемешал буквы в слове ЭКРАН. Восстановить испорченное слово (рис. 4.19).

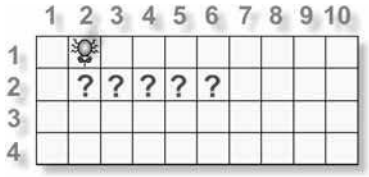


Рис. 4.19

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. 4.20).

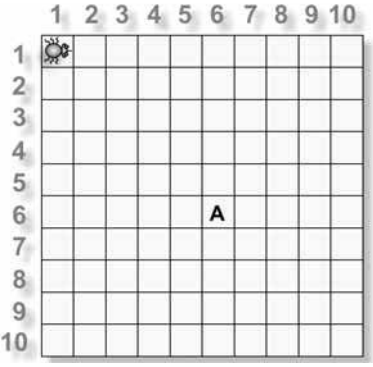


Рис. 4.20

Глава 5

Другой тип повторения

5.1. Когда неизвестно число повторений

> Несколько малышек подбежали к лежавшему на земле яблоку и, толкая его перед собой, покатили к ближайшему двору.
>
> *Н. Носов*

Петя. Я хочу предложить необычную задачу. Вот посмотри на мой рисунок (рис. 5.1).

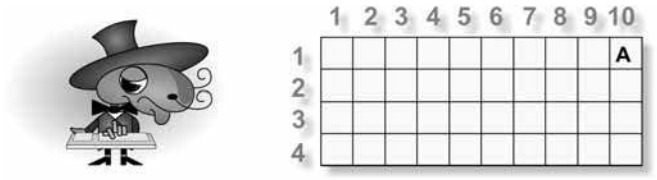


Рис. 5.1

Буква А стоит в углу, в клетке (1,10). Её надо удалить с поля.

Вася. А где же Кукарача?

Петя. Где-то в первой строке.

Вася. Как же я напишу программу, не зная положения исполнителя?

Петя. В этом-то и состоит задача: программа должна работать для *любого* положения Кукарачи в первой строке.

Вася. Если я напишу `повтори 9 вправо`, то где бы ни был Кукарача, он всегда выбросит букву с поля.

Петя. Да, так поступить можно. Но чаще всего выполнение этой команды будет заканчиваться аварийным сообщением «Не могу».

Вася. Понятно. Но как же поступить? Ведь Кукарача должен двигаться вправо, пока не толкнёт букву А. Число повторений заранее неизвестно, значит, команда **ПОВТОРИ** здесь не годится.

Петя. Верно подметил: надо двигаться вправо, *пока* не обнаружится буква А! Именно так, с ключевого слова **ПОКА**, и начинается ещё одна команда повторения. Вот как она записывается для нашей задачи:

```
ПОКА НЕ А ВПРАВО
```

Вася. Замечательно! Это то, что нужно.

Петя. В общем случае эта команда имеет вид:

```
ПОКА условие команда
```

Условие записывается точно так же, как и в команде ветвления. Работает команда так (рис. 5.2):

1. Сначала проверяется `условие`.
2. Если условие ложно, выполнение команды повторения заканчивается.
3. Если условие истинно, выполняется `команда`, записанная после условия, и всё начинается сначала: проверяется условие, выполняется команда и т. д.

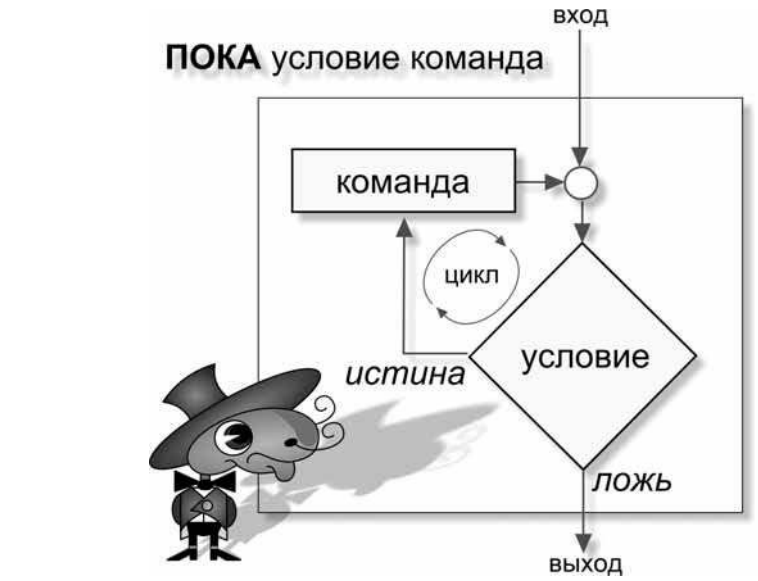


Рис. 5.2. Схема работы команды **ПОКА**

Вася. Значит, если условие ложно с самого начала, то команда, записанная после условия, не выполнится ни разу?

Петя. Правильно. В связи с этим хочу заметить, что код

```
ПОКА НЕ А ВПРАВО
```

для нахождения кубика А в некоторых случаях может оказаться неверным.

Вася. Почему?

Петя. Попробуй решить такую задачу.

Задача 1

Кукарача стоит в клетке (1,1). На поле расположены две буквы А. Одна — сразу под Кукарачей, другая — где-то во второй строке (рис. 5.3). Переместить первую букву в третью строку, а Кукарачу поставить на место второй.

Рис. 5.3

Вася. Очень просто! Вот моё решение:

```
ЭТО Вход
  ВНИЗ
  ПОКА НЕ А ВПРАВО
КОНЕЦ
```

Петя. Теперь запусти эту программу.

Вася. Кукарача толкает кубик вниз и останавливается. Почему он не идёт вправо?

Петя. Условие цикла ложно с самого начала — ведь перед циклом Кукарача толкнул букву А вниз, поэтому команда **ВПРАВО**, записанная в цикле, не выполнится ни разу.

Вася. Как же быть?

Петя. Надо «очистить» старое значение датчика кубиков, переместив Кукарачу в направлении второй буквы А:

```
ЭТО Вход
  Первый_кубик_вниз
  На_место_второго_кубика
```

```
КОНЕЦ
ЭТО Первый_кубик_вниз
  ВНИЗ
КОНЕЦ
ЭТО На_место_второго_кубика
  ВПРАВО // Шаг в направлении кубика
  ПОКА НЕ А ВПРАВО
КОНЕЦ
```

Вася. Я всё же хотел бы уточнить, как работает датчик кубиков.

Петя. В начальный момент (перед выполнением программы) в датчик кубиков помещается значение **ПУСТО**. Затем значение датчика меняется каждый раз, когда Кукарача передвигается в соседнюю клетку. Если в клетке был кубик (исполнитель толкнул его) в датчик записывается символ с кубика, в противном случае в датчик записывается **ПУСТО**.

Проверка условия состоит в сравнении значения датчика с выражением, которое записано после ключевого слова **ЕСЛИ** или **ПОКА**. Результат проверки: *истина*, если показание датчика соответствует этому выражению и *ложь* в противном случае.

Вася. Понятно!

Петя. Давай решим ещё несколько задач, чтобы привыкнуть к использованию команды **ПОКА**.

Задача 2

Найти проход в стене и дойти до края поля (в десятом столбце).

Петя. Здесь тоже нельзя использовать команду **ПОВТОРИ**, потому что расположение прохода неизвестно (на рис. 5.4 он изображён приблизительно). Мы не знаем, сколько надо сделать шагов до прохода, значит, неизвестно, какое число повторений надо указать в этой команде. Что же делать? Надо идти вдоль стены (кубики с буквой Ж), пока она есть. Потом пройти в обнаруженный проход.

Вася. Попробую записать это так:

```
ЭТО Выход
  Поиск_прохода
  На_край_доски
КОНЕЦ
```

Первая процедура написана, а вот как же искать проход?

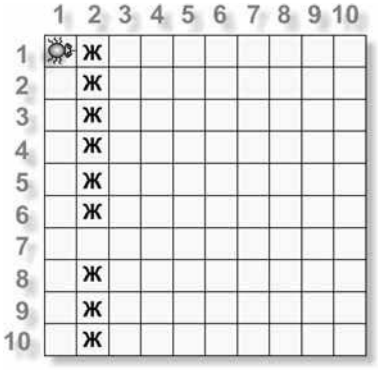


Рис. 5.4

Петя. Толкать кубики, пока не обнаружится пустое место.

Вася. Попробую так:

```
ЭТО Поиск_прохода
  ПОКА НЕ ПУСТО Шаг
КОНЕЦ
```

Петя. Команда `Шаг` не выполнится ни разу. Ведь в начальный момент Кукарача не толкал кубики, поэтому условие в команде **ПОКА** — ложно.

Вася. Хорошо, тогда напишу так:

```
ЭТО Поиск_прохода
  ВПРАВО
  ПОКА НЕ ПУСТО Шаг
КОНЕЦ
```

Петя. Это другое дело. Продолжай дальше.

Вася. Теперь напишу процедуру `Шаг`:

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

Петя. Очень хорошо.

Вася. Теперь осталось написать совсем простую процедуру На_край_доски:

```
ЭТО На_край_доски
  ПОВТОРИ 8 ВПРАВО
КОНЕЦ
```

Петя. Не мог бы ты привести примеры с циклом **ПОКА** из повседневной жизни?

После бурного обсуждения у братьев возникли следующие примеры.

1. Пока гвоздь не забит, ударяй по нему молотком.
2. Пока идут машины, стой у перехода.
3. Пока кастрюля грязная, три её щёткой.
4. Пока бочка не полна, лей в неё воду из шланга.
5. Иди на север, пока не выйдешь к реке.

Во всех этих примерах можно выделить повторяющиеся действия, но число повторов заранее неизвестно. Момент окончания повторения определяется некоторым условием, которое проверяется в процессе выполнения работы.

Вопросы и задания

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).
2. Кукарача спускается по диагонали из клетки (1,1) до буквы С, расположенной где-то на этой диагонали.
3. Робот-станок должен изготовить 100 одинаковых деталей.
4. Робот-станок должен изготовлять одинаковые детали в течение дневной рабочей смены.
5. Робот-станок должен изготовлять одинаковые детали до тех пор, пока у него не кончатся заготовки.
6. Компьютерная программа должна обрабатывать длительное нажатие символьной клавиши на клавиатуре.
7. Автомат отключает двигатель стиральной машины через 5 минут после включения.
8. Автомат отключает двигатель стиральной машины через 10 000 оборотов двигателя.

5.2. Как обмануть интерпретатор

— Вылезая из обыкновенного гамака, вы можете зацепиться ногой за верёвку и, упав, разбить себе нос, — сказал Шурупчик, поднимаясь с пола. — В моём механизированном гамаке эта опасность, как видите, полностью устранена.

Н. Носов

Петя. Покажу тебе ещё одну команду языка Кукарачи. Эта команда называется **составной**. Её единственное назначение — объединить несколько команд в одну группу:

```
{
  команды
}
```

Вася. А зачем нужна эта команда?

Петя. Её используют тогда, когда по правилам языка должна быть записана одна команда, а хочется написать несколько.

Составную команду удобно использовать на ветвях условной команды:

```
ЕСЛИ условие
  ТО     команда1  // только одна команда!
  ИНАЧЕ команда2  // только одна команда!
```

и в качестве тела цикла:

```
ПОВТОРИ число команда  // только одна команда!
ПОКА условие команда   // только одна команда!
```

Можно, например, так переписать твою программу с использованием составной команды:

```
ЭТО Выход
  // Поиск прохода
  ВПРАВО
  ПОКА НЕ ПУСТО
  {
    // вернуться в первый столбец
    ВЛЕВО
    // шагнуть на следующую строку
    ВНИЗ
    // толкнуть кубик
    ВПРАВО
  }
```

```
  // На край доски
  ПОВТОРИ 8 ВПРАВО
КОНЕЦ
```

Вася. Программа получилась более компактной.

Петя. Да, но использовать составную команду надо осторожно. Часто предпочтительней написать много маленьких процедур, чем всю работу описывать в одной длинной. Мы с тобой уже говорили о преимуществе процедурного программирования.

Вася. Так как же быть? Использовать составную команду или нет?

Петя. Использовать, но разумно: там, где это не ухудшит наглядность текста программы, не затруднит программирование и не увеличит длину программы.

Вася. Как может составная команда увеличить длину программы?

Петя. Если одни и те же действия должны выполняться неоднократно в разных частях программы, то выгоднее оформить их в отдельную процедуру, а не пользоваться составной командой.

Вася. Понятно.

Петя. Продолжим решение задач с использованием команды **ПОКА**.

Задача 3

Найти и вытолкнуть с поля кубик с буквой А (рис. 5.5).

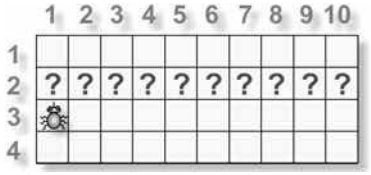


Рис. 5.5

Вася. Вот мое решение:

```
ЭТО Поиск_А
  // Испытание первого кубика.
  ВВЕРХ
  ПОКА НЕ А
  {
    // Возврат в третью строку.
    ВНИЗ
    // Смещение к следующему кубику.
    ВПРАВО
    // Проверка кубика.
```

```
    ВВЕРХ
  }
  // Выбрасывание буквы А с поля.
  ВВЕРХ
КОНЕЦ
```

Вопросы и задания

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

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. 5.6).

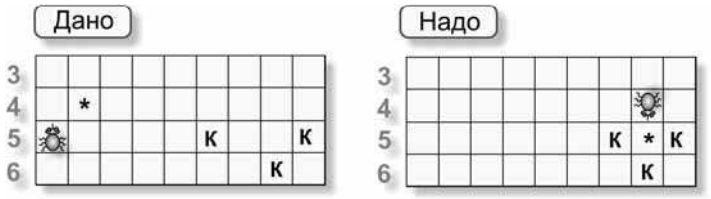


Рис. 5.6

2. Нарастить стену на один кирпич (кирпич изображается буквой Ж). После работы стена должна сместиться в шестой столбец. На рис. 5.7 показана наибольшая возможная длина стены в исходном состоянии среды.

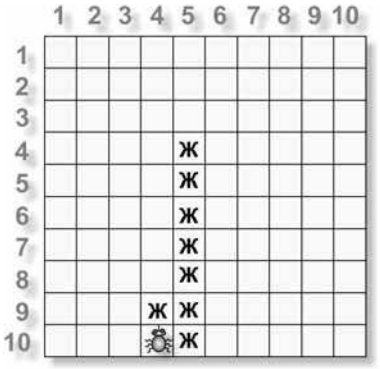


Рис. 5.7

3. Провести исполнителя по прямоугольнику А-Б-В-Г. Размеры прямоугольника заранее неизвестны (рис. 5.8).
4. На одном из перевёрнутых кубиков во втором столбце — буква Ш. Найдите её и составьте слово ИШАК. Букву Ф с поля надо убрать (рис. 5.9).

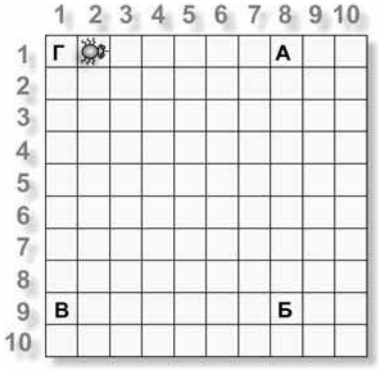


Рис. 5.8

Рис. 5.9

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. 5.10).
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. 5.11 показано возможное состояние среды в начальный момент.

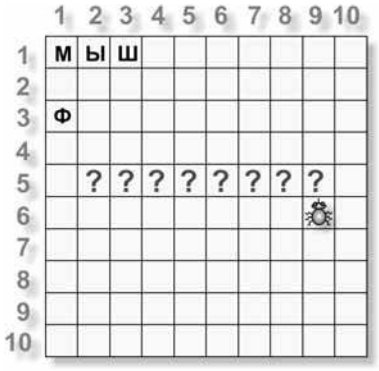


Рис. 5.10

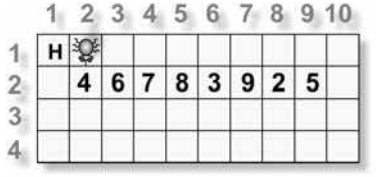


Рис. 5.11

Глава 6

Кукарача хочет укусить себя за хвост

6.1. Разговор о рекурсии

— Ах, эти малыши! Вечно они придумывают разные шалости!

Н. Носов

Петя. Вот рисунок среды и программа (рис. 6.1). Как Кукарача будет выполнять эту программу?

Рис. 6.1. Необычная программа

Вася. Очень странная программа, она кусает сама себя за хвост!

Петя. Это всё, что ты можешь сказать?

Вася. Сейчас я попробую выполнить её за Кукарачу. Первая команда **ВПРАВО**. Перемещаюсь в клетку (1,2). Следующая команда `До_упора`. Укус за хвост! Программа вызывает саму себя. Что делать?

Петя. Исполняй!

Вася. Хорошо. Процедура `До_упора` состоит из двух команд. Первая команда **ВПРАВО** смещает Кукарачу в клетку (1,3). Потом снова следует команда `До_упора`! Всё повторяется сначала.

Петя. Что же произойдёт в итоге?

Вася. Кукарача будет всё время смещаться вправо, пока... пока не остановится у края поля, умоляя: «Не могу!»

Петя. Такая программа называется **рекурсивной**. Рекурсия — это способ программирования, при котором программа вызывает саму себя. Если бы доска не имела границы (скажем, была бы склеена наподобие цилиндра), то Кукарача, выполняя программу `До_упора`, никогда бы не остановился. Такая рекурсия называется **бесконечной**.

А вот пример **конечной** рекурсии.

Задача 1

Найти и выбросить с поля букву А (рис. 6.2).

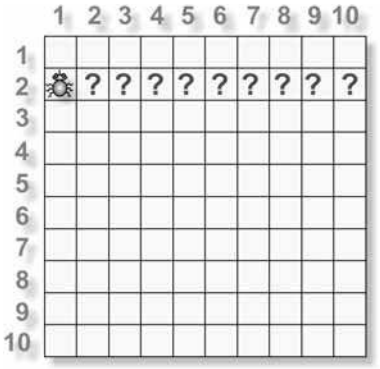


Рис. 6.2

Решение состоит из двух частей: поиска буквы А во второй строке и выбрасывания её с поля.

```
ЭТО Кубик_А
  Поиск
  Выброс
КОНЕЦ
```

Поиск выполняется так. Толкаем кубик, он переворачивается. Если на кубике обнаружена буква А, поиск закончен. В противном случае продвигаемся

вперёд и начинаем всё сначала (рекурсия): толкаем кубик, выполняем проверку и т. д., пока кубик А не будет найден.

```
ЭТО Поиск
  Шаг
  ЕСЛИ НЕ А
    ТО Поиск // Рекурсия!
КОНЕЦ

ЭТО Шаг
  ВВЕРХ
  ВПРАВО
  ВНИЗ
КОНЕЦ

ЭТО Выброс
  ПОВТОРИ 8 ВНИЗ
КОНЕЦ
```

Рекурсия в процедуре `Поиск` заканчивается, когда найдена буква А (рис. 6.3).

Рис. 6.3

Вася. Процедуру `Поиск` можно построить при помощи команды **ПОКА**:

```
ЭТО Поиск
  Шаг
  ПОКА НЕ А Шаг
КОНЕЦ
```

Петя. Верно! Команда повторения почти всегда предпочтительнее рекурсии, но бывают задачи, которые можно решить только рекурсивными методами. Рассмотрим их немного позже.

Сейчас хочу заметить, что рекурсия бывает **прямая** и **косвенная**. Рекурсия называется прямой, если процедура вызывает саму себя напрямую (рис. 6.4).

Рис. 6.4. Прямая рекурсия

Когда процедура вызывает себя через другую процедуру, рекурсия называется косвенной (рис. 6.5).

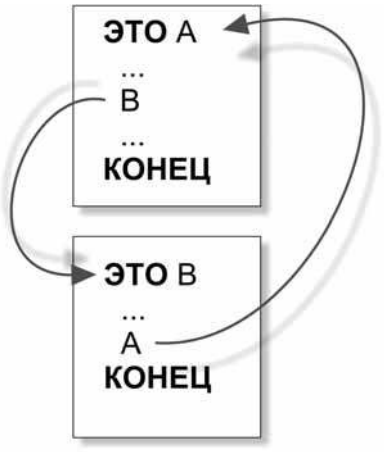


Рис. 6.5. Косвенная рекурсия

Увлечённые обсуждением рекурсии, братья придумали несколько примеров, в которых содержится рекурсия (рис. 6.6–6.8).

В последнем примере программа составлена для исполнителя, который умеет выполнять команду **НАЛЕЙ_СТАКАН** и проверять условие **БОЧКА НЕ ПОЛНА**.

Для этого примера можно написать и нерекурсивную программу:

```
ЭТО Налей_бочку
  ПОКА БОЧКА НЕ ПОЛНА НАЛЕЙ_СТАКАН
КОНЕЦ
```

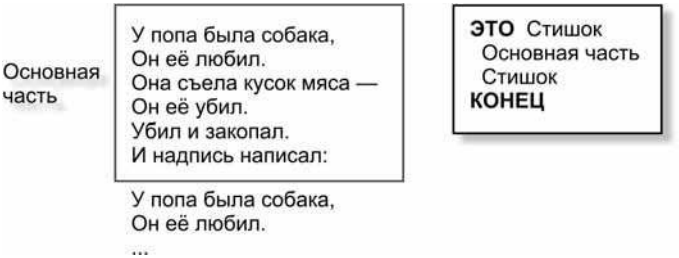


Рис. 6.6. Литературная бесконечная рекурсия

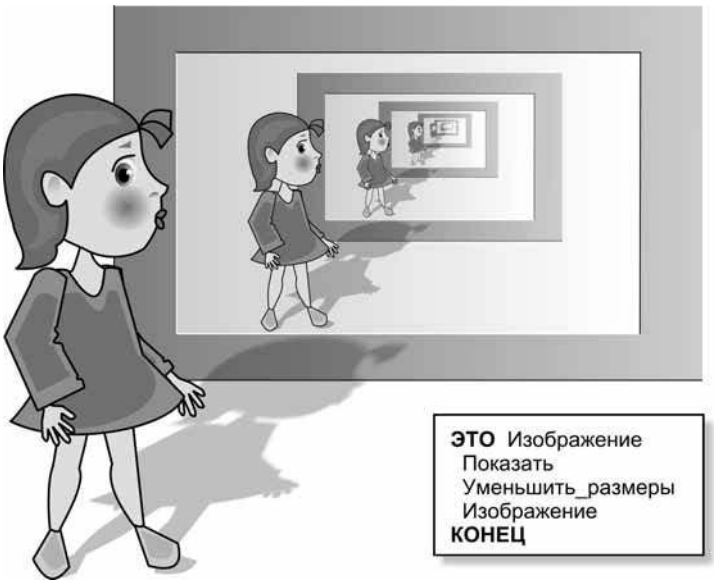


Рис. 6.7. Телевизионная бесконечная рекурсия

Рис. 6.8. Конечная косвенная рекурсия

6.2. Рекурсивный практикум

«Дудки! — подумал Незнайка. — Никакого сотрясения мозга у меня нет».

Н. Носов

Вася. Всё, что ты мне рассказал, вроде бы понятно, но уж очень непривычно.

Петя. Давай решать задачи. К рекурсии надо привыкнуть.

Задача 2

Проведите Кукарачу по диагонали до буквы К. Расположение буквы К на диагонали заранее неизвестно (рис. 6.9).

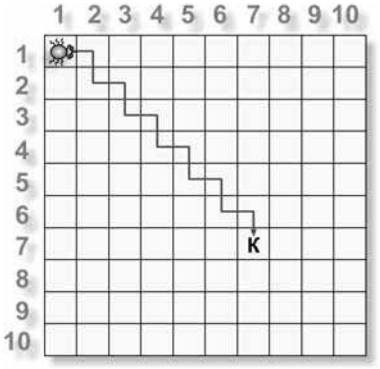


Рис. 6.9

Вася. Эту задачу можно легко решить при помощи команды **ПОКА**: нужно шагать по диагонали, пока не встретится буква К.

```
ЭТО Поиск_К
  Шаг
  ПОКА НЕ К Шаг
КОНЕЦ

ЭТО Шаг
  ВПРАВО
  ВНИЗ
КОНЕЦ
```

Петя. Хорошо! А теперь попробуй написать рекурсивный вариант.

Вася. Попробую.

```
ЭТО Поиск_К
  Шаг
  ЕСЛИ НЕ К
    ТО Поиск_К
  КОНЕЦ
```

Петя. Ну вот, видишь, у тебя получилось! Следующая задача похожа на предыдущую, но в ней есть одно тонкое место.

6.3. Головоломное программирование

— Ветер несёт нас прямо на север, — объявил Знайка. — Значит, обратно надо будет возвращаться на юг.

Н. Носов

Задача 3 (автор Лилитко Е. П.)

Кукарача находится в верхней строке поля. Где-то под ним (в том же столбце) расположена буква А. Требуется спуститься к букве (где бы она ни находилась), подвинуть её один раз и вернуться к исходной позиции в верхнюю строку. Сообщение «Не могу» допускается только в том случае, если буквы А не оказалось под Кукарачей (рис. 6.10).

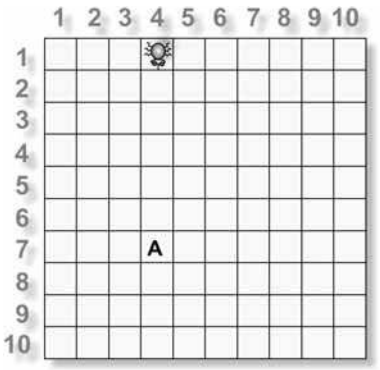


Рис. 6.10

Вася. Думаю, тонкость заключается в том, что Кукарача должен вернуться в исходную позицию!

Петя. Да, ты правильно подметил отличие. Ну и как написать решение этой задачи?

Вася. Совершенно не понимаю, как вернуть Кукарачу к месту старта. Ведь он должен запомнить, на сколько клеток спустился вниз, чтобы повторить столько же шагов вверх. Но как это сделать? Думаю, у задачи нет решения.

Петя. Ты ошибаешься. Вот решение задачи и, как ты видишь, очень короткое:

```
ЭТО Прогулка
  ВНИЗ
  ЕСЛИ НЕ А
    ТО Прогулка
  ВВЕРХ
КОНЕЦ
```

Вася. Короткое-то короткое, но непонятное. Кукарача найдёт букву А и по такой программе:

```
ЭТО Прогулка
  ВНИЗ
  ЕСЛИ НЕ А
    ТО Прогулка
КОНЕЦ
```

Твоя программа отличается от моей только одной командой **ВВЕРХ**. Значит, выполняя твою программу, Кукарача найдёт букву А, после чего поднимется всего на одну клетку вверх, что в общем случае неверно. Например, для такого положения Кукарачи и буквы А (рис. 6.11) нужно для возврата к старту сделать не один шаг вверх, а два.

Петя. Хорошо, давай проверим моё решение.

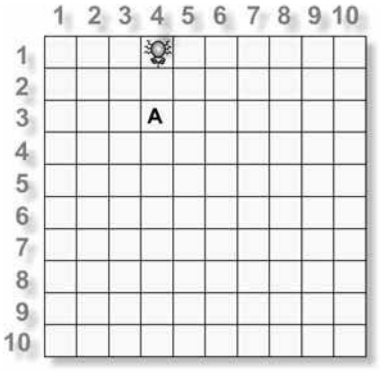


Рис. 6.11

Вася установил Кукарачу и букву, как на рисунке, набрал программу Пети и после запуска с удивлением обнаружил, что Кукарача, выполнив работу, вернулся в клетку, с которой начал своё путешествие.

Вася. Ничего не понимаю, колдовство какое-то!

Петя. Давай разберёмся не спеша... Первой выполняется команда `вниз` (рис. 6.12).

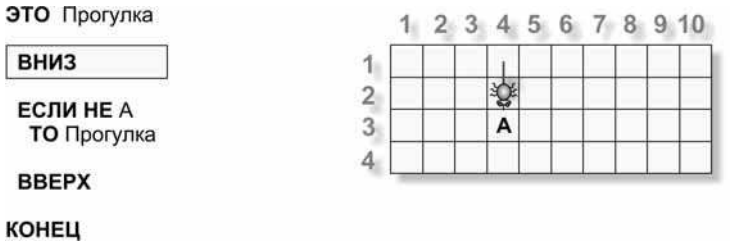


Рис. 6.12

Затем выполняется условная команда (рис. 6.13).

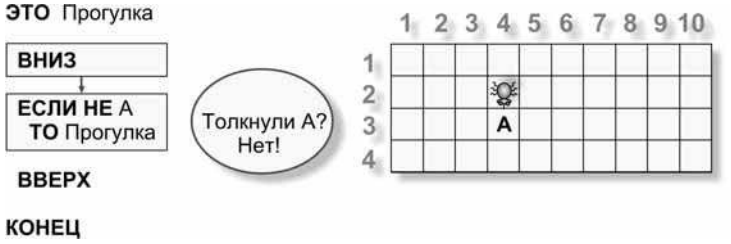


Рис. 6.13

Так как условие истинно, выполнение условной команды будет равносильно выполнению второго экземпляра программы `Прогулка` (рис. 6.14).

Рис. 6.14

Теперь условие ложно, и выполнится команда **ВВЕРХ** во втором экземпляре программы `Прогулка` (рис. 6.15).

Рис. 6.15

Работа второго экземпляра программы `Прогулка` завершена, и, тем самым, завершилось выполнение условной команды в первом экземпляре. Кукарача приступает к выполнению последней команды **ВВЕРХ** и возвращается в исходное положение (рис. 6.16).

Рис. 6.16

Вася. Сногсшибательное программирование! Чудеса, да и только. Можно ли здесь заменить рекурсию командой **ПОКА**?

Петя. С циклом **ПОКА** эту задачу не решить! Предлагаю полезное правило (рис. 6.17).

Когда нужно выполнить повторение, используй:	
ПОВТОРИ	★ известно число повторений
ПОКА	★ неизвестно число повторений
рекурсию	★ нужно “запомнить” число повторений

Рис. 6.17

6.4. Подводные камни

— У вашего больного на плече синяк, — сказала она Синеглазке. — Пойдите в аптеку, там дадут медовый пластырь.

Н. Носов

Вася. Я написал рекурсивную прогулку, но не с командой ветвления, а с командой цикла:

```
ЭТО Прогулка1
  ВНИЗ
  ПОКА НЕ А Прогулка1
  ВВЕРХ
КОНЕЦ
```

Петя. Ну и как, работает?

Вася. Нет, не работает, что очень обидно. Кукарача толкает кубик, делает шаг назад, возвращается, снова толкает кубик и так поступает многократно, пока не сваливает букву за пределы поля. Потом пытается покончить с собой, но одумывается и кричит «Не могу»!

Петя. Давай разбираться в том, что происходит. Сначала Кукарача выполняет команду **вниз** (рис. 6.18).

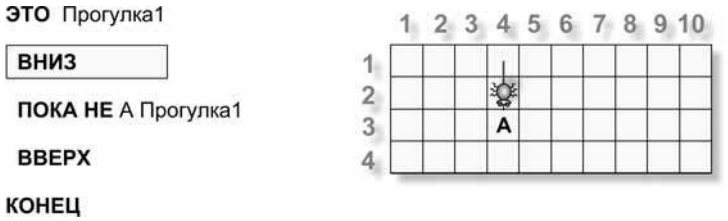


Рис. 6.18

Затем приступает к выполнению команды цикла. Цикл **пока** работает так:

1. Проверка условия.
2. Если условие ложно, цикл заканчивается, иначе выполняется команда `Прогулка1` и цикл продолжается.

Сейчас условие истинно, значит, начинает выполняться второй экземпляр процедуры `Прогулка1`, с первой команды **вниз** (рис. 6.19).

Теперь условие ложно и выполнится команда **ВВЕРХ** во втором экземпляре процедуры `Прогулка1`. Кукарача делает шаг назад (рис. 6.20).

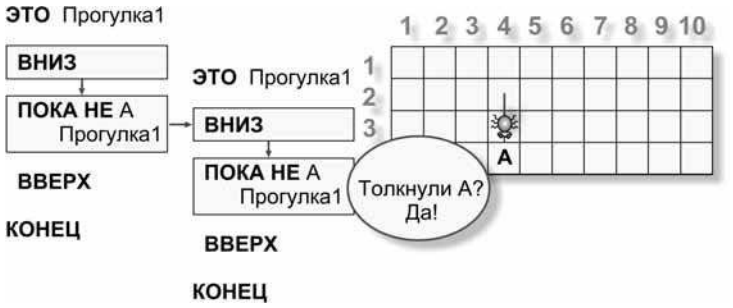


Рис. 6.19

Рис. 6.20

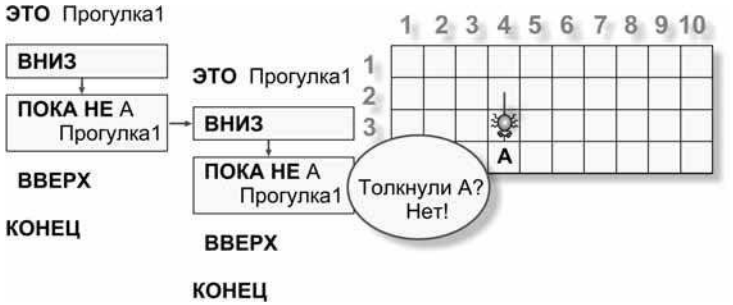


Рис. 6.21

Работа второго экземпляра процедуры `Прогулка1` завершена и выполнение цикла в первом экземпляре продолжается. Проверка условия даёт значение «истина», значит, снова начинает выполняться второй экземпляр процедуры `Прогулка1`, с первой команды **вниз** (рис. 6.21).

Условие истинно, и выполняется третий экземпляр процедуры `Прогулка1`. Кукарача толкает кубик. Условие ложно, шаг назад по команде **вверх**, возврат в цикл второго экземпляра (рис. 6.22).

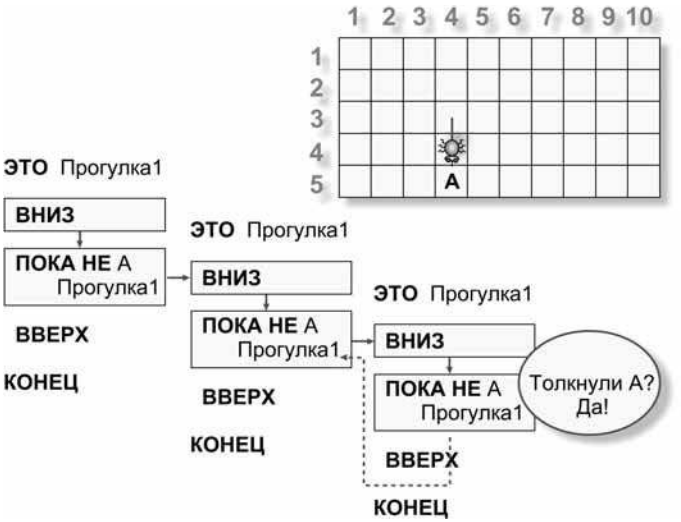


Рис. 6.22

Условие истинно, цикл продолжается.

Вася. Понятно! В первый экземпляр процедуры теперь не вернуться.

Петя. Точно. Кукарача будет «плясать» около кубика, толкая его к границе поля, а интерпретатор будет размножать экземпляры процедуры. Это безобразие завершится, когда у интерпретатора закончится свободная память или когда Кукарача попытается сбросить себя с поля!

Вася. Второе в нашем случае наступит быстрее!

Петя. Если поле было бы более длинным, у интерпретатора могла бы закончиться свободная память. Всему же есть пределы!

Вася. Никогда не буду записывать рекурсивный вызов процедуры из цикла!

Петя. Правильная мысль! Всё очень просто: возврат из рекурсии происходит внутрь цикла, и цикл продолжается.

Вася. А возврат из рекурсии в команде ветвления происходит за неё — ведь ветвление это не цикл, никаких повторов в нём нет.

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. 6.23).
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. 6.24).

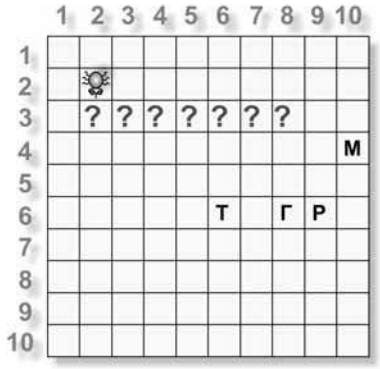


Рис. 6.23

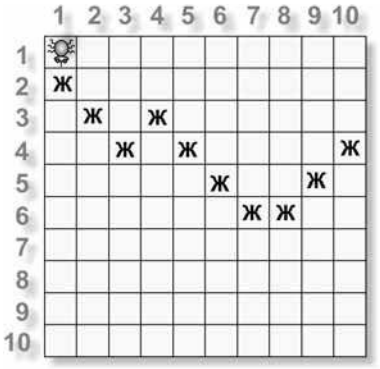


Рис. 6.24

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. 6.25).
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. 6.26).

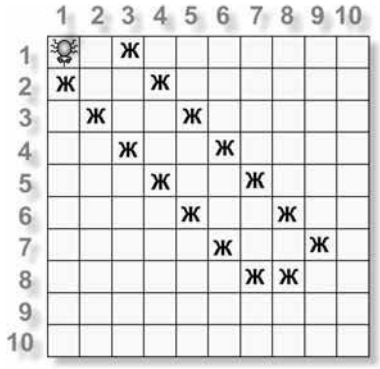


Рис. 6.25

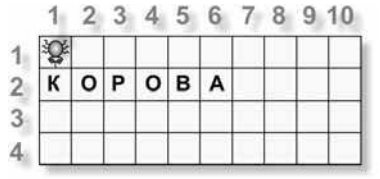


Рис. 6.26

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.

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.
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. 6.27) и после завершения трудов остановится в правом верхнем углу поля.

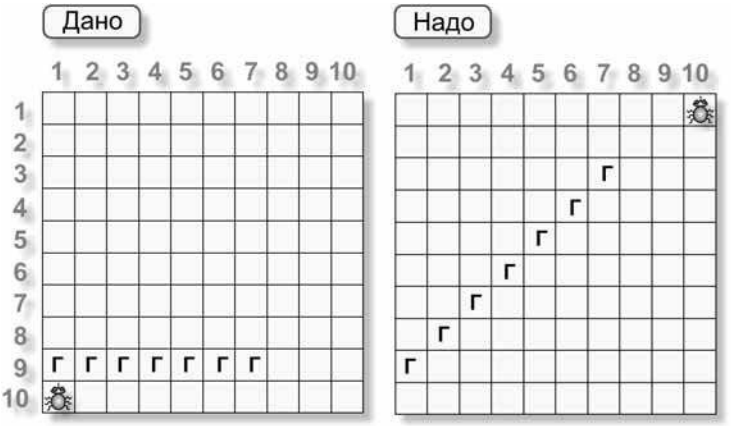


Рис. 6.27

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.

Глава 7

Задачи

— Ну, что тут рассказывать — развёл Незнайка руками. — Меня давно просили наши малыши что-нибудь придумать: «Придумай что-нибудь, братец, да придумай».

Н. Носов

В Роботландском университете на курсе 31 «Азы программирования» с 2002/2003 учебного года прижилась новая форма работы — «Задача недели».

Эта форма дополняет традиционные семестровые турниры юных программистов мини-турнирами, проводимыми каждую учебную неделю.

Конкурс «Задача недели» задуман:

- как подготовка к основным курсовым сражениям;
- как индикатор прохождения учебной программы;
- как способ общения.

И конечно, он направлен на активизацию творческого потенциала детей (студентов курса) и их преподавателей (руководителей детских команд).

Сначала учебные группы (команды) каждую неделю присылали на конкурс по одной задаче с решением и перекрёстно оценивали друг друга. Затем правила изменились: куратор публиковал на курсовом списке рассылки только условия задач, а решения принимались и оценивались дополнительно.

Такая система позволила получать общую оценку команды из нескольких слагаемых:

- баллы за условие придуманной задачи;
- баллы за решение «своей» задачи;
- баллы за решение «чужих» задач.

Конкурс не просто имел успех, он поразил студентов курса и преподавателей как вирус!

Весьма полезный вирус! Ведь придумать хорошую задачу совсем непросто! Создать красивое решение — тоже почётно! Оценить условие задачи, протестировать код и выставить баллы за решение — хороший практикум для начинающих магов.

В этой главе собраны лучшие задачи недели для Кукарачи за два учебных года, и мы предлагаем читателю получить удовольствие от их решения.

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.

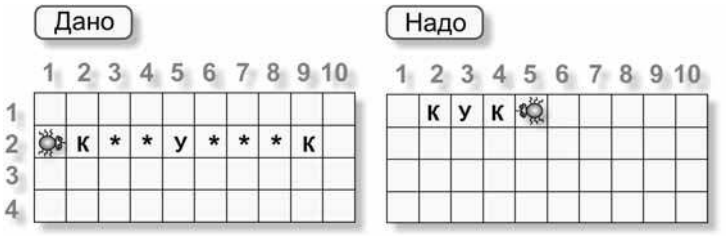


Рис. 7.1

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.

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.

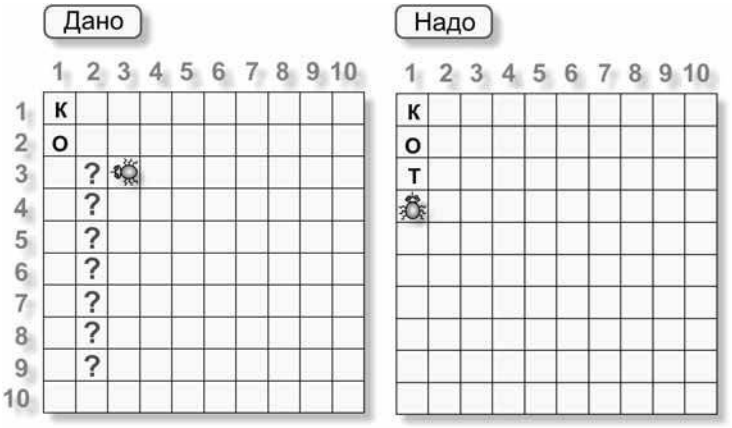


Рис. 7.2

Рис. 7.3

Рис. 7.4

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.

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 состояния среды показаны на рис. 7.5.

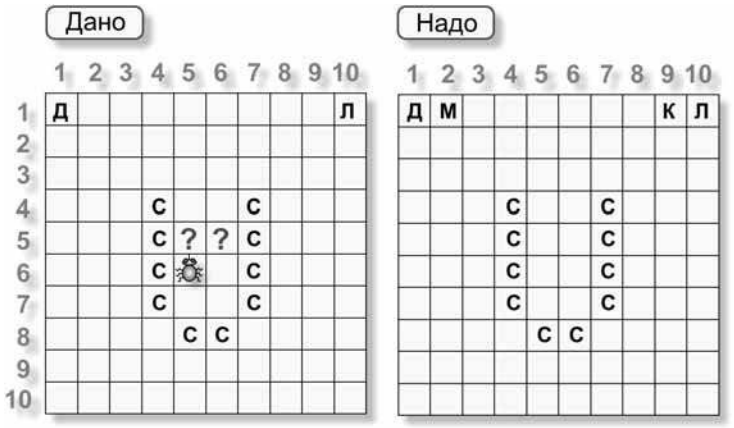


Рис. 7.5

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.

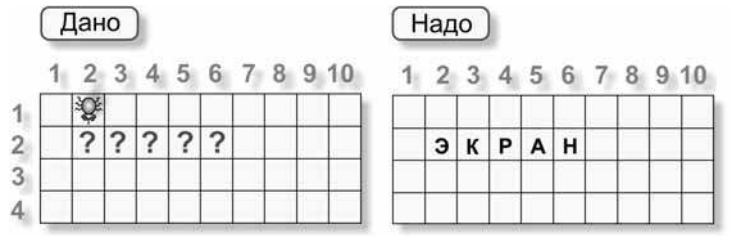


Рис. 7.6

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 состояний среды показан на рис. 7.7.

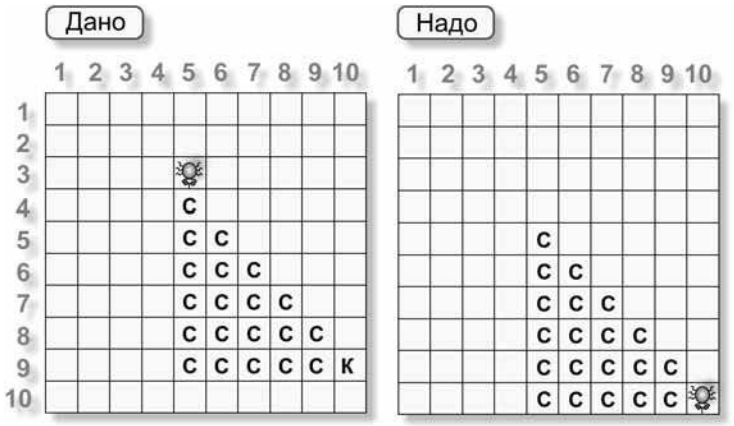


Рис. 7.7

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 состояний среды показан на рис. 7.8.

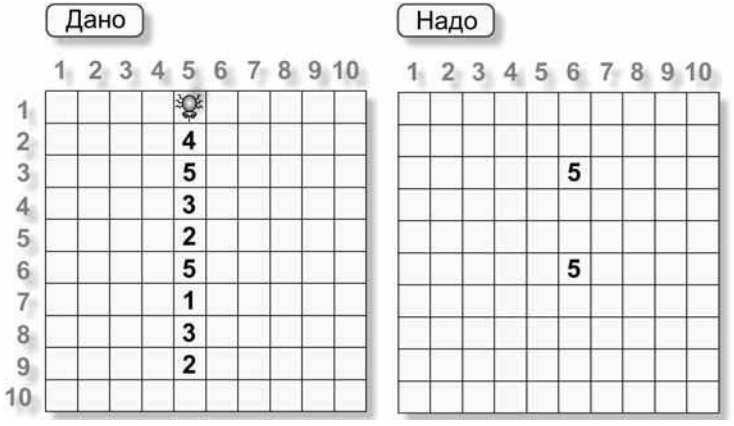


Рис. 7.8

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.

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.

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.

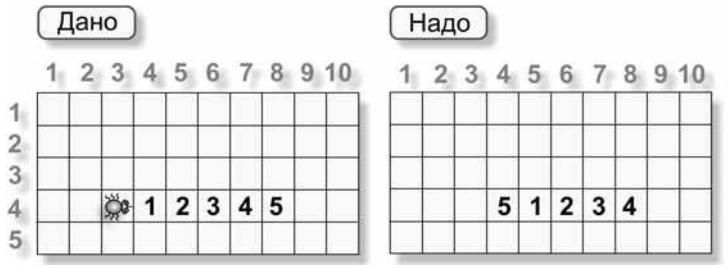


Рис. 7.9

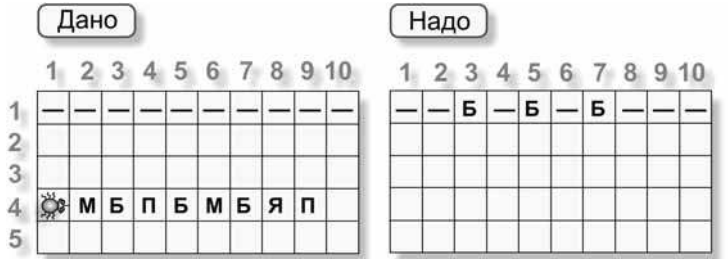


Рис. 7.10

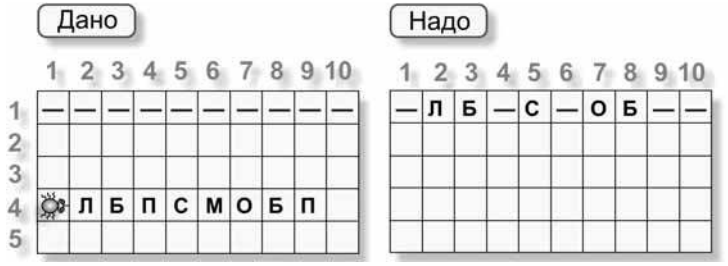


Рис. 7.11

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.

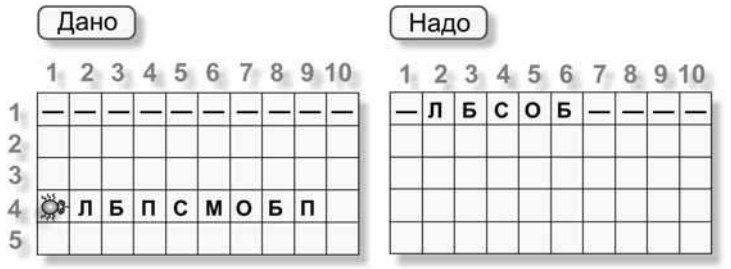


Рис. 7.12

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 состояний среды показан на рис. 7.13.

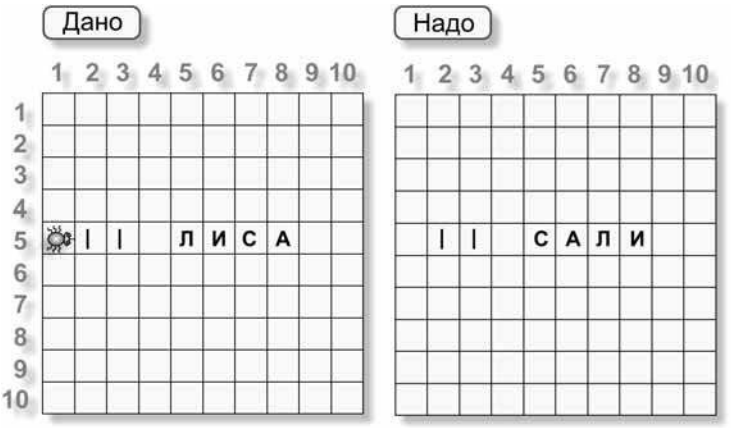


Рис. 7.13

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 положения среды показаны на рис. 7.14.

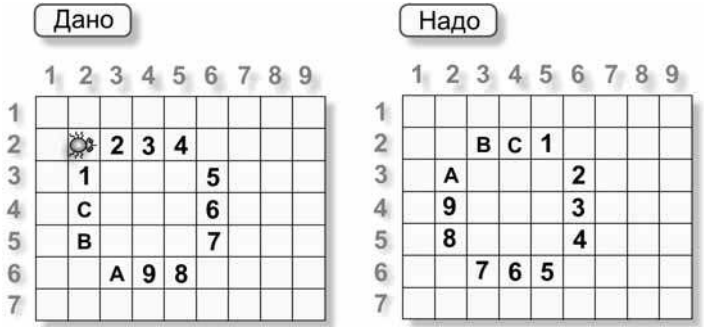


Рис. 7.14

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.

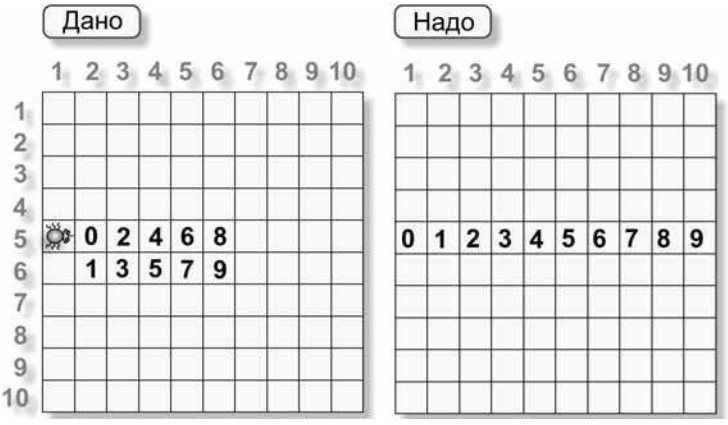


Рис. 7.15

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.

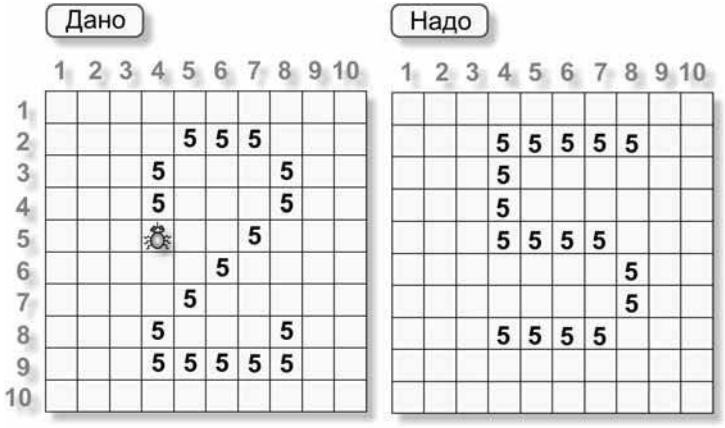


Рис. 7.16

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.

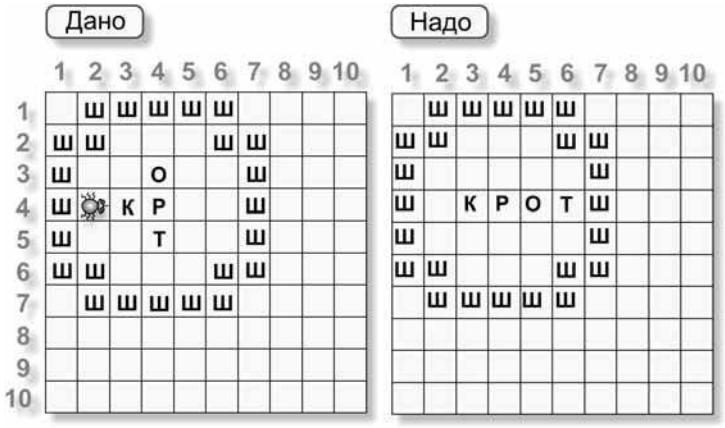


Рис. 7.17

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.

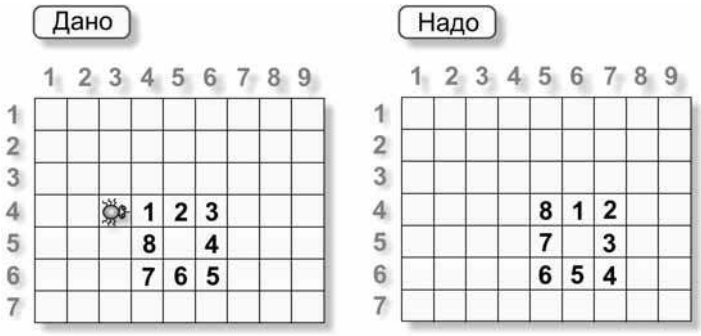


Рис. 7.18

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.

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.

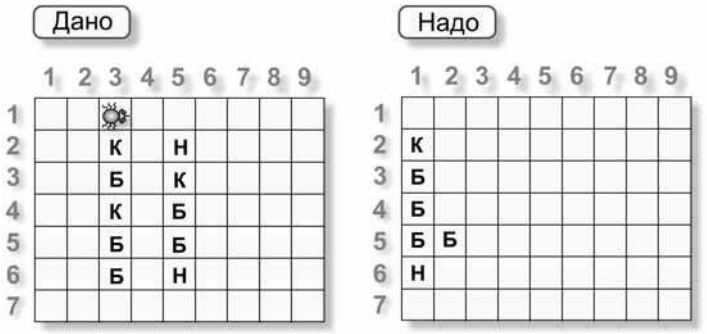


Рис. 7.19

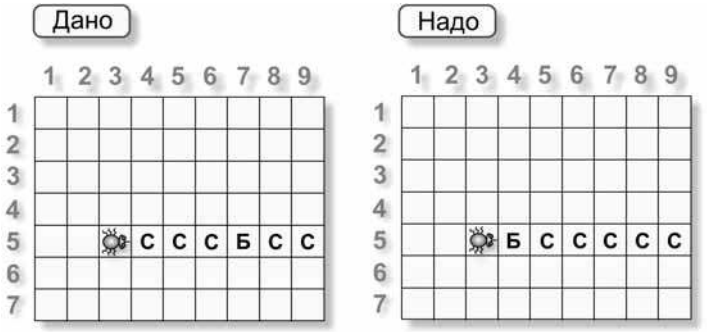


Рис. 7.20

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

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).

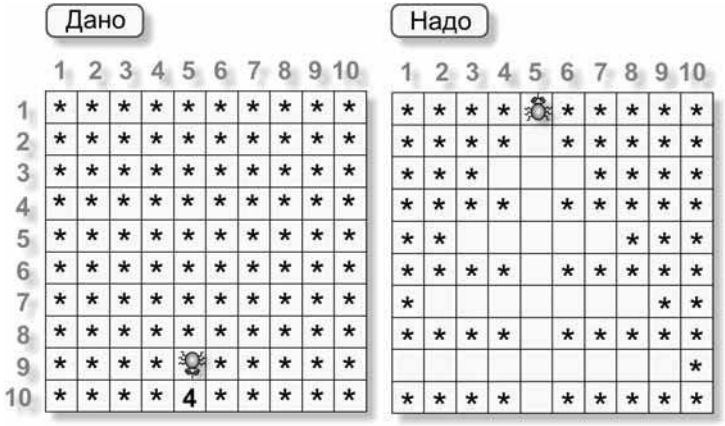


Рис. 7.22

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.

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 неизвестно.

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.

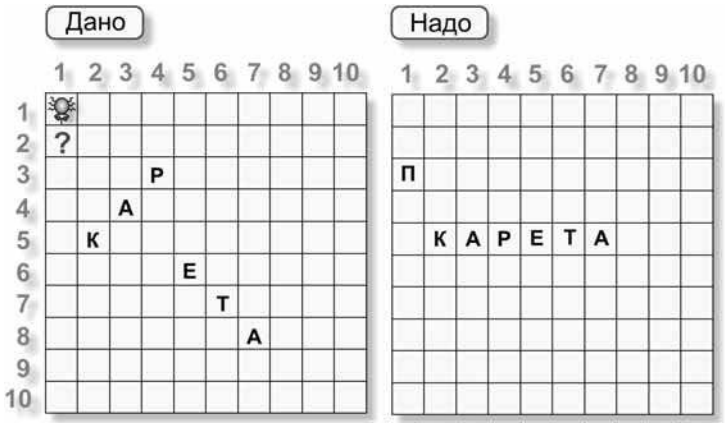


Рис. 7.23

Рис. 7.24

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.

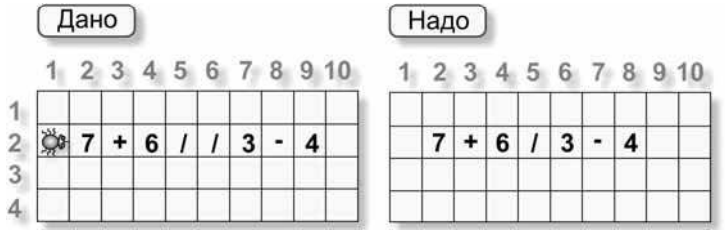


Рис. 7.25

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 показаны на рис. 7.26.

Рис. 7.26

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.

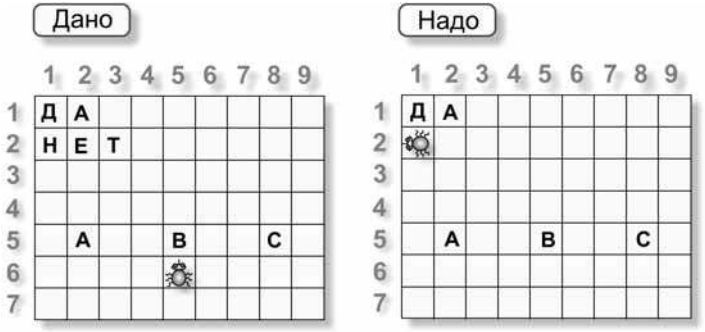


Рис. 7.27

Часть II

Корректор

Глава 8

Знакомство с исполнителем

Петя, студент университета, будущий программист, приехал домой на каникулы. Его брат Вася очень этому рад: ведь он тоже увлечён информатикой и использует каждую встречу с Петей, чтобы задать многочисленные вопросы и поупражняться в практическом программировании.

На этот раз приезд брата складывался особенно удачно: папа купил новый пакет программ с исполнителем Корректор, и сегодня Куки собрались у компьютера, чтобы познакомиться с этим новым исполнителем. Все, кроме мамы. Надежду Семёновну больше волновал праздничный ужин, затеянный в честь приезда старшего сына, и она осталась на кухне (рис. 8.1).

Рис. 8.1. Папа купил диск с Корректором

8.1. Корректор и его среда обитания

— А меня зовут Гена. Я работаю в зоопарке крокодилом.
— А что мы будем сейчас делать?
— Ничего. Давайте просто побеседуем.

Э. Успенский

Папа. Привет, ребята! Я принёс диск с новым роботландским исполнителем.

Петя. Новый исполнитель? Это интересно.

Вася. А как его зовут?

Папа. Сейчас Петя скопирует диск на винчестер нашего компьютера, и мы увидим Корректора — так зовут этого робота.

Вася. Корректор — звучное имя.

Папа. Корректорами называют людей, которые занимаются в издательствах правкой текста. Видимо, этот исполнитель не зря получил своё имя.

Петя. Готово! Я закончил копирование и запускаю приложение (рис. 8.2).

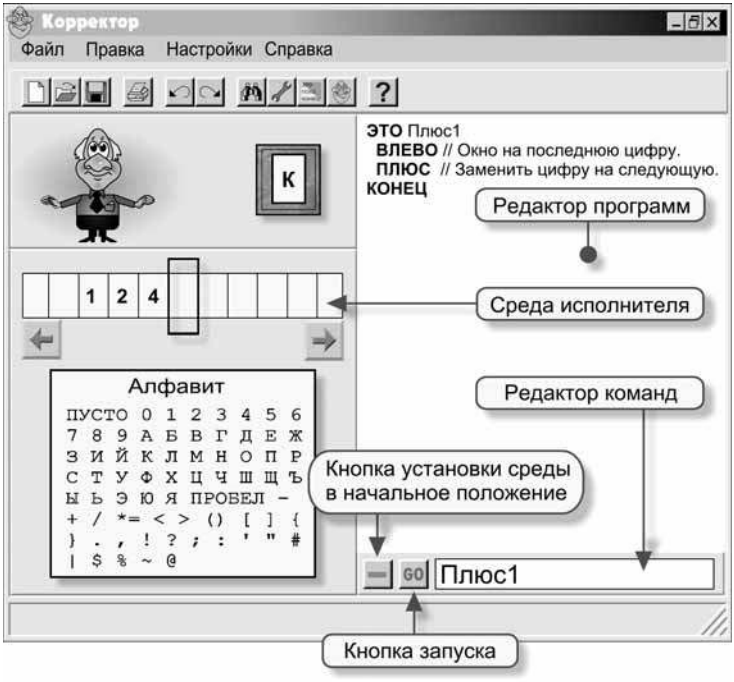


Рис. 8.2. Корректор на экране компьютера

Папа. Сейчас я перескажу то, что узнал про Корректора в Роботландии, где приобрёл диск, а подробное описание робота вы прочитаете в сопровождающей документации.

Итак, этого робота зовут Корректор. Он работает с длинной лентой, которая разбита на клетки или ячейки. В ячейку может быть записан один символ. Символ — это буква, цифра, знак «+» или что-нибудь ещё в этом роде. Полный набор допустимых символов составляет алфавит исполнителя.

Считается, что лента спрятана в непрозрачный для Корректора футляр. Исполнитель может видеть только одну ячейку ленты через специальное окно в футляре. Через это же окно Корректор может записывать на ленту любой символ из своего алфавита. Ещё он может перемещать окно вправо и влево вдоль ленты и таким образом записывать символы в любые ячейки на ней (рис. 8.3).

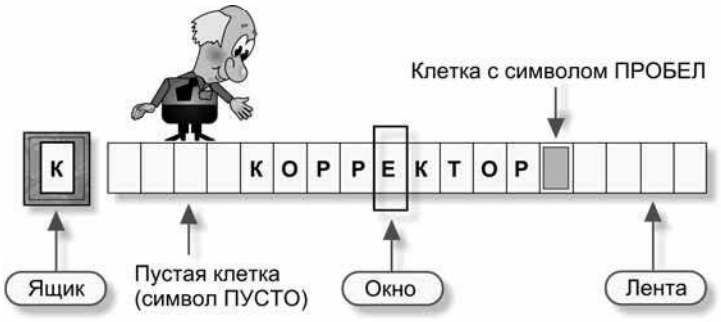


Рис. 8.3. Среда Корректора

Рис. 8.4. Система команд Корректора и алфавит

Вася. А может Корректор стирать символы на ленте?

Папа. Исполнитель очищает ячейку, записывая в неё специальный символ `пусто` — он стоит самым первым в алфавите исполнителя. `пусто` не надо путать с пробелом — это два разных символа!

Ещё у нашего робота есть ящик — специальная ячейка памяти. Корректор может снимать копию символа из окна в ящик или, наоборот, копировать символ из ящика на ленту.

На рис. 8.4 и в табл. 8.1 приводится полный набор команд Корректора и его алфавит.

Таблица 8.1

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

Порядок символов в алфавите очень важен. Команда `плюс` заставляет исполнителя заменить символ в окне на следующий по порядку в алфавите, а команда `минус` — на предыдущий. Команда `плюс` для последнего символа в алфавите и команда `минус` для первого приводят к отказу — Корректор выводит на экран сообщение «Не могу» (рис. 8.5).

Петя. Команды `плюс` и `минус` мне понятны. Проверим, как их понял Вася. Вот, братишка, для тебя несколько упражнений.

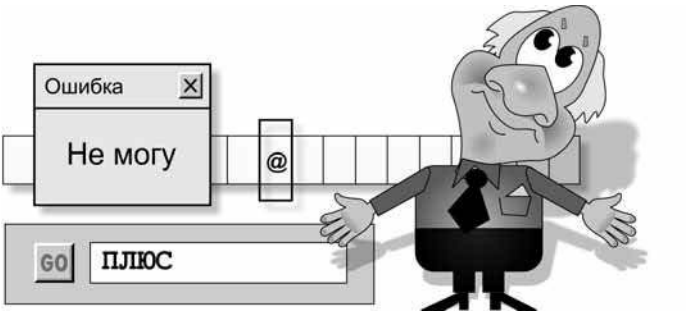


Рис. 8.5. Корректор сообщает об отказе

Вопросы и упражнения

Какой символ запишет Корректор на ленту по указанной команде и для показанного состояния среды (рис. 8.6–8.10)?

1. плюс

Рис. 8.6

2. минус

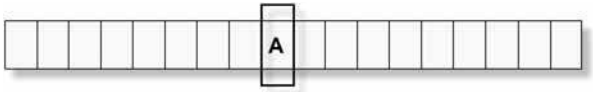


Рис. 8.7

3. плюс

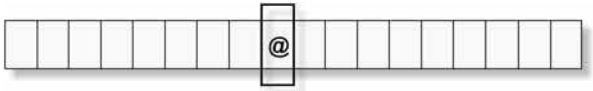


Рис. 8.8

4. минус

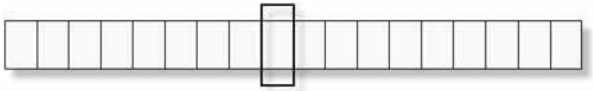


Рис. 8.9

5. минус

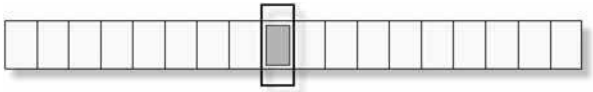


Рис. 8.10

8.2. Попробуем управлять

Палку в воду бросит, а телёнок принесёт. Скажет ему: «Лежать» — и Гаврюша лежит. Прикажет ему Матроскин: «Взять! Куси!» — тот сразу бежит и бодаться начинает.

Э. Успенский

Вася. Пустите меня за компьютер, я хочу поработать с Корректором!

Петя. Пожалуйста! Вот тебе задание: пусть исполнитель запишет на ленту слово КОТ.

Вася. А где мне писать команды?

Папа. Команды записываются в редакторе команд.

Вася. Первая команда понятна (рис. 8.11).

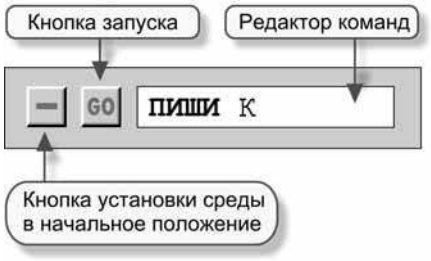


Рис. 8.11. Редактор команд Корректора и кнопки управления

Команду я написал, но Корректор ничего не сделал!

Папа. Чтобы Корректор выполнил команду, надо щёлкнуть мышкой по кнопке *GO* или нажать клавишу <Enter> на клавиатуре.

Вася. Ах, да! Понятно! Всё так, как было у Кукарачи.

Вася нажал <Enter>, и Корректор записал в оконную ячейку букву К (рис. 8.12).

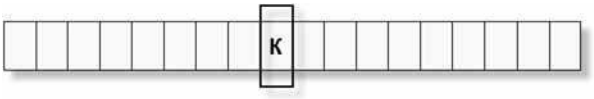


Рис. 8.12

Вася. Если теперь я задам команду `пиши о`, буква К, конечно, исчезнет. Понятно, сначала нужно переместить окно на ленте вправо!

Вася по порядку записывал и запускал следующие команды:

```
ПИШИ К
ВПРАВО
ПИШИ О
ВПРАВО
ПИШИ Т
```

В конце работы исполнителя лента выглядела так (рис. 8.13).

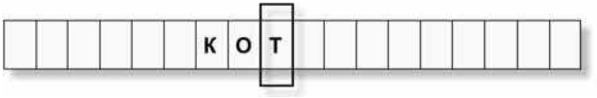


Рис. 8.13

Вопросы и упражнения

1. Вася написал команду **пиши** R. Корректор выдал сообщение «Не понимаю». Почему?
2. Задавая команды Корректору, напишите на ленте его имя.

8.3. Управление при помощи программы

> — Это по правилам, — отвечает Печкин. — Я, может, вас очень люблю. Я, может, плакать буду. А только правила нарушать нельзя.
>
> *Э. Успенский*

Вася. Работать с исполнителем в командном режиме неинтересно. Надо написать для него программу!

Отличается ли язык программирования Корректора от языка Кукарачи?

Папа. Они полностью совпадают. У этих исполнителей разные среды и СКИ, а язык программирования — один и тот же.

Петя. Напомню, что программа состоит из процедур, каждая из которых записывается так (рис. 8.14).

Слова **это** и **конец** — ключевые. Между ними записываются команды.

Вася. Я помню! После слова **это** нужно записывать имя процедуры. Это имя становится именем новой команды для исполнителя. Для запуска процедуры её имя набирают в редакторе команд и нажимают экранную кнопку *GO*.

Папа. Ты прав!

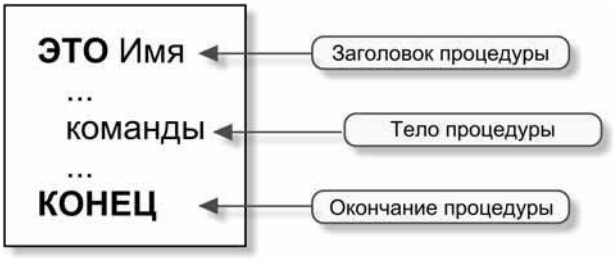


Рис. 8.14. Форма записи процедуры

Задача

На ленте записано чётное число. Исполнитель в начальный момент смотрит на пустую клетку за последней цифрой этого числа. Написать программу, которая добавит к числу на ленте единицу.

Вася. Как записать на ленту число, над которым будет работать Корректор?

Папа. Для этого нужно войти в редактор среды, щёлкнув мышью по изображению ленты на экране.

Вася записал на ленту Корректора число 124 и задумался (рис. 8.15).

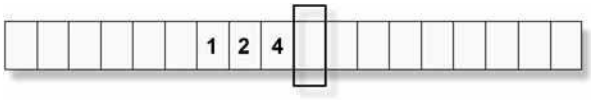


Рис. 8.15

Вася. В СКИ Корректора нет нужной команды.

Папа. Не торопись. Посмотри ещё раз на СКИ повнимательнее.

Вася. Я понял! Нужно использовать команду **плюс**, когда в окне видна последняя цифра числа. Ведь эта команда заменяет символ на следующий по алфавиту. Цифры в алфавите идут по порядку, значит, получится то, что нужно.

Вася щёлкнул мышкой по редактору программ и набрал такую программу:

```
ЭТО Плюс1
  ВЛЕВО // Установить окно на последнюю цифру.
  ПЛЮС  // Заменить цифру на следующую.
КОНЕЦ
```

Вася не поленился написать комментарии в своей короткой программе. Он с важным видом объяснил брату и отцу, что хотя комментарии не обрабатываются интерпретатором программ, но они здорово помогают программисту.

И Вася, конечно, прав. С комментариями программа становится более понятной: человеку её легче читать и отлаживать. Для записи комментариев в программах исполнителей Кукарача и Корректор используют две косых черты «//». Они сами и всё, что находится правее них до конца строки, не принимается во внимание интерпретатором при выполнении программы.

Папа. Ну вот, программа записана. Выйди из программного поля, набери в редакторе команд имя твоей процедуры и щёлкни мышкой по экранной кнопке *GO*.

Вася. Записываю `Плюс1`, нажимаю *GO*. Да, здорово! Корректор усердно работает со своей лентой, и что удивительно, действительно прибавляет к числу на ленте единицу (рис. 8.16)!

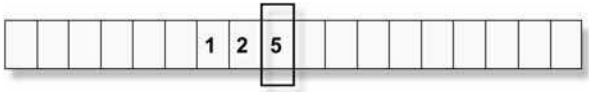


Рис. 8.16

Петя. Запусти программу ещё раз, я хочу посмотреть, как трудится исполнитель, это выглядит довольно забавно!

Вася. Ещё раз нажал кнопку *GO*.

Корректор послушно выполнил программу, но, к удивлению Васи, на ленте получилось число 135, а не 125, как он ожидал (рис. 8.17).

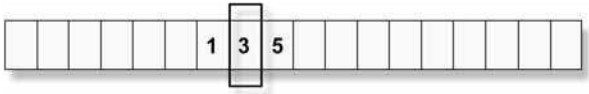


Рис. 8.17

Вася. Не понял!

Петя. Корректор начал выполнение твоей программы в том состоянии среды, которая осталась после первого запуска.

Папа. Для установки среды в исходное положение нужно перед запуском нажать экранную кнопку с изображением знака минус (рис. 8.18).

Рис. 8.18. Кнопка установки среды в начальное положение

Вася щёлкнул мышью по этой кнопке, на ленте восстановился начальный текст 124, а окно переместилось на первый пустой символ за ним. Теперь новый запуск программы прошёл правильно, без каких-либо неожиданностей.

Папа. Как вы думаете, почему в условии задачи говорится о чётном числе?

Вася. Думаю, программа будет работать и для нечётных чисел. Если я запишу на ленту число 123, то после выполнения программы оно изменится на 124. Ведь по команде плюс цифра 3 заменяется на следующую по алфавиту. В итоге тройка заменится на четвёрку.

Петя. Я понял, в чём дело. Всё дело в цифре 9!

Вася. Ну, конечно, я упустил это из виду! Число 9 команда плюс заменит буквой А. Папа исключил девятку, запретив использовать нечётные числа.

Папа. Всё правильно, вы разгадали мою хитрость! Теперь давайте запишем нашу задачу на диск и пойдём ужинать, а то мама уже начинает сердиться.

Вопросы и упражнения

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. 8.19 показан пример возможного начального состояния среды.

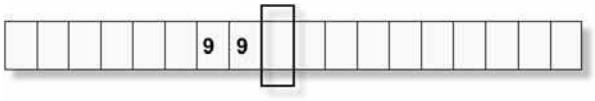


Рис. 8.19

Глава 9

Язык программирования

И стал он галчонка учить разговаривать. Целыми днями сидел около него и говорил:
— Кто там? Кто там? Кто там?

Э. Успенский

К сегодняшнему занятию наши герои тщательно подготовились: Петя внимательно прочитал документацию, папа устроил Васе экзамен по языку программирования (он такой же, как у Кукарачи). Результаты экзамена показали, что Васе не помешает освежить в памяти основные понятия и приёмы программирования. Было принято решение устроить вечер «воспоминаний» с традиционным чаем по-роботландски (рис. 9.1).

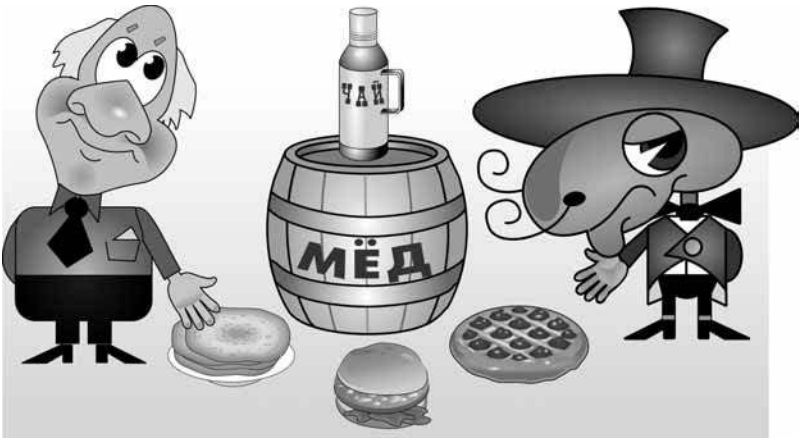


Рис. 9.1. Чаепитие по-роботландски

9.1. Процедурное программирование

На брюках было: два кармана спереди, два кармана сзади, два кармана по бокам и один карман внизу, на колене.

Н. Носов

Папа. Начнём работу с такой задачи.

Задача 1

На ленте Корректора записан один символ, и он виден через окошко. Требуется записать рядом ещё пять символов, которые в алфавите исполнителя следуют по порядку за исходным символом.

Вася. Значит, если на ленте записан символ 0, то нужно написать ещё символы 1, 2, 3, 4, 5 (рис. 9.2).

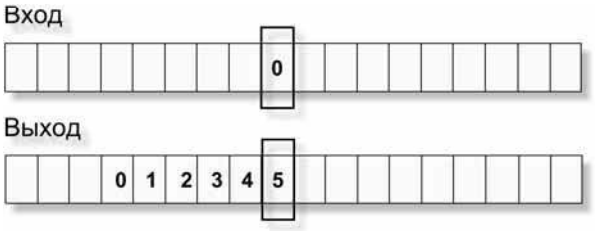


Рис. 9.2

Петя. А если на ленте записан символ 8, то новыми символами будут 9, А, Б, В, Г (рис. 9.3).

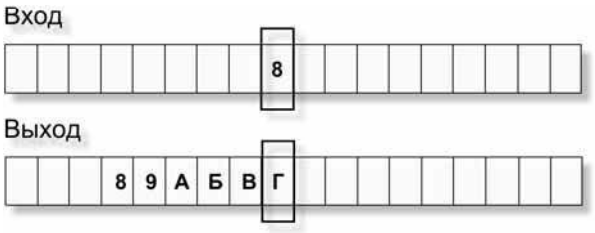


Рис. 9.3

Папа. Вы правильно поняли условие. Но сначала решим более простую задачу: пусть требуется записать на ленту не пять, а только один новый символ по указанному правилу.

Петя. Надо скопировать символ в соседнюю ячейку и применить к копии команду **плюс**.

Папа. А как скопировать символ?

Петя. В среде Корректора кроме ленты есть ящик — эта ячейка памяти вполне подходит для наших целей.

Вася. Мне всё ясно! Вот моё решение:

```
ЭТО Следующий_символ
  ЯЩИК+  // Копировать символ из окна в ящик
  ВПРАВО // Сместить окно в соседнюю ячейку
  ЯЩИК-  // Копировать символ из ящика в окно
  ПЛЮС   // Заменить символ на следующий по алфавиту
КОНЕЦ
```

Папа. Очень хорошо! Теперь я думаю, несложно решить задачу в начальной постановке.

Петя. Нет ничего проще. Нужно повторить процедуру `Следующий_символ` пять раз.

Вася. Получается длинная программа... В ней 20 команд:

```
ЭТО Символы_по_порядку
  ЯЩИК+
  ВПРАВО
  ЯЩИК-
  ПЛЮС

  ЯЩИК+
  ВПРАВО
  ЯЩИК-
  ПЛЮС

  ЯЩИК+
  ВПРАВО
  ЯЩИК-
  ПЛЮС

  ЯЩИК+
  ВПРАВО
  ЯЩИК-
```

```
  ПЛЮС

  ЯЩИК+
  ВПРАВО
  ЯЩИК-
  ПЛЮС
КОНЕЦ
```

Петя. Запись можно существенно сократить, используя в качестве команды вызов процедуры Следующий_символ.

Вася. Да, конечно, мы так делали, программируя Кукарачу. Я помню могущество процедур! Их вызов можно использовать многократно, и это здорово экономит усилия программиста! Вот моё новое решение:

```
ЭТО Символы_по_порядку
  Следующий_символ
  Следующий_символ
  Следующий_символ
  Следующий_символ
  Следующий_символ
КОНЕЦ
```

Папа. Преимущество процедурного программирования ещё и в том, что программы становятся более простыми и наглядными, так как раскладываются на множество маленьких процедур, описывающих простые и короткие действия.

Вася. Можно сказать, что процедуры расширяют СКИ, «дополняя» её новыми командами.

Петя. А можно, наоборот, считать, что команды из СКИ — это специальные стандартные процедуры, понятные интерпретатору без описаний!

Вася. Напомните, пожалуйста, что такое интерпретатор?

Петя. Исполнитель понимает и выполняет команды только из своей СКИ, такие как ВПРАВО, ВЛЕВО, ПЛЮС, МИНУС. Мы пишем команду Следующий_символ, а её нет в СКИ. Тем не менее, исполнитель прекрасно работает и не выдаёт сообщения «Не понимаю». Почему?

Вася. Мы пишем для него программу.

Петя. И что, он её выполняет?

Вася. Конечно, ты же сам видел, как он забавно трудился над лентой и ящиком.

Петя. И что, Корректор выполнял команду Следующий_символ? Она ведь не входит в его СКИ!

Вася. Ну, не знаю!

Петя. На самом деле программу выполняет не Корректор, а интерпретатор программ. Он читает программу и переводит её в СКИ исполнителя.

Вася. Кажется, я понимаю. Когда интерпретатор видит в моей программе команду `Следующий_символ`, он передаёт исполнителю 5 команд из его СКИ (рис. 9.4).

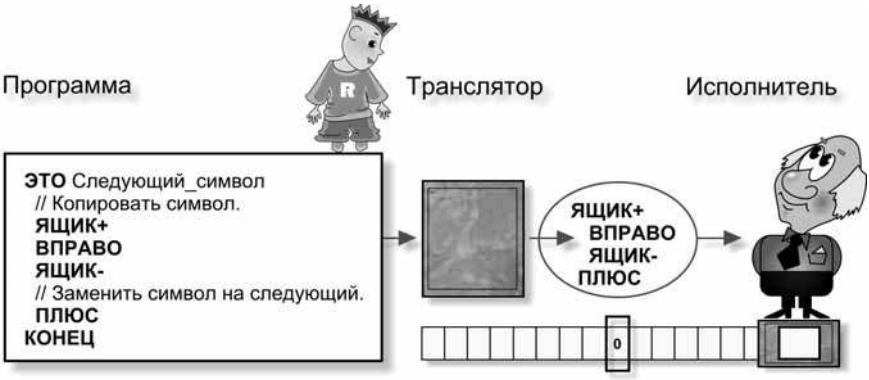


Рис. 9.4. Схема работы интерпретатора программ

Петя. Точно! Интерпретатор работает как переводчик. Он позволяет использовать язык программирования, а это здорово облегчает управление роботом.

Вася. А как выглядит интерпретатор на экране компьютера?

Петя. Никак. Интерпретатор — одна из процедур программы, моделирующей исполнителя на компьютере, и он не имеет на экране никакого внешнего отображения.

Вопросы и упражнения

1. Уменьшает ли использование команды вызова процедуры количество работы для исполнителя?
2. Как интерпретатор будет выполнять такую программу:

```
ЭТО Работа
  Подготовка
  Выполнение
  Завершение
КОНЕЦ
```

9.2. Циклы

> Если шофёр увидит, что кто-нибудь починяет машину, он обязательно подойдёт и тоже начнёт что-нибудь ковырять, подвинчивать болт или гайку или просто станет давать советы.
>
> *Н. Носов*

Папа. Запись нашей последней программы можно сделать более компактной и наглядной, если использовать команду цикла **ПОВТОРИ**.

Петя. Напомню эту конструкцию языка программирования. Она выглядит так:

```
ПОВТОРИ число команда
```

ПОВТОРИ — ключевое слово языка, «число» — целое положительное число, обозначающее число повторений, «команда» — любая команда языка (только одна!), которая подлежит повторению.

Вася. Вот новый вариант нашей программы. С командой цикла получилось гораздо веселее!

```
ЭТО Символы_по_порядку
  ПОВТОРИ 5 Следующий_символ
КОНЕЦ


ЭТО Следующий_символ
  ЯЩИК+  // Копировать символ из окна в ящик
  ВПРАВО // Сместить окно в соседнюю ячейку
  ЯЩИК-  // Копировать символ из ящика в окно
  ПЛЮС   // Заменить символ на следующий по алфавиту
КОНЕЦ
```

Петя. Эту программу можно записать в виде одной процедуры, используя составную команду:

```
ЭТО Символы_по_порядку
  ПОВТОРИ 5
  {
    ЯЩИК+  // Копировать символ из окна в ящик
    ВПРАВО // Сместить окно в соседнюю ячейку
    ЯЩИК-  // Копировать символ из ящика в окно
    ПЛЮС   // Заменить символ на следующий по алфавиту
  }
КОНЕЦ
```

Папа. Так, конечно, сделать можно. Составная команда позволяет объединять несколько команд в одну при помощи фигурных скобок. Но последняя программа сложнее для понимания.

Формулирую следующую задачу.

Задача 2

Лента Корректора пуста за исключением одной ячейки, в которую записан некоторый символ. Требуется найти этот символ и показать его в окне. На рис. 9.5 показан пример исходного и результирующего состояния среды.

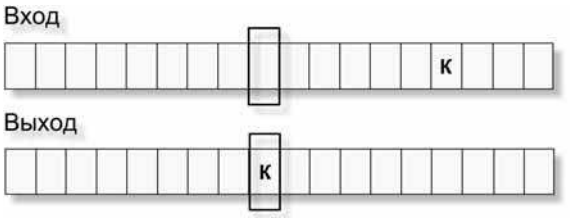


Рис. 9.5

Петя. Думаю, самое удобное средство языка для решения этой задачи — цикл **ПОКА**.

Вася. Я помню, как записывается этот цикл:

```
ПОКА условие команда
```

Работает цикл так (рис. 9.6):

1. Сначала проверяется условие.

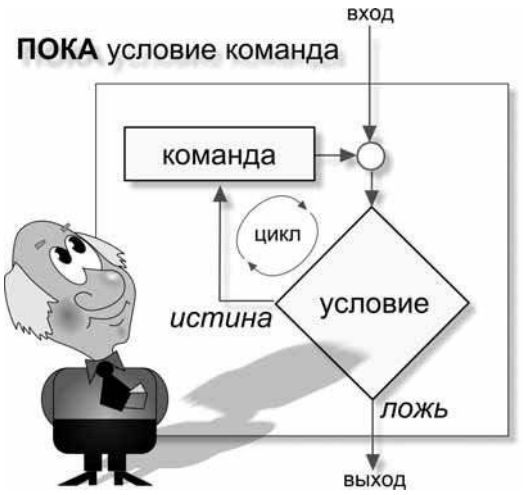


Рис. 9.6. Схема работы цикла **ПОКА**

2. Если условие ложно, выполнение команды заканчивается.
3. Если условие истинно, выполняется «команда», записанная после условия, и всё начинается сначала: проверяется условие, выполняется команда и т. д.

Петя. Верно! Заметь, что если условие ложно с самого начала, то команда не выполнится ни разу.

Папа. Осталось рассказать, как записываются и проверяются условия.

Петя. В качестве условия записывают символ. Интерпретатор будет сравнивать его с тем символом, который Корректор видит через окошко на ленте. Если они совпадают, условие принимает значение *истина*, если отличаются — значение *ложь*.

Папа. Для обозначения пробела и пустого места нужно использовать ключевые слова **ПРОБЕЛ** и **ПУСТО**. Эти же ключевые слова годятся и для команды **ПИШИ**.

Петя. В качестве условия можно записывать и ключевое слово **ЦИФРА**. Это условие принимает значение *истина*, когда в окне на ленте записан один из символов 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Папа. Ещё одна важная деталь. При записи условия можно использовать ключевое слово **НЕ**. Это слово заставляет интерпретатор проверять условие «наоборот»!

Вася. Сейчас ключевое слово **НЕ**, думаю, не потребуется. Ведь Корректор должен перемещать окно по ленте *пока* в нём **ПУСТО**. Как только в окошке появится непустой символ, работа цикла заканчивается:

```
ЭТО Поиск_символа
  ПОКА ПУСТО ВПРАВО
КОНЕЦ
```

Папа. Верное решение. А вот задача, где как раз удобно воспользоваться ключевым словом **НЕ**.

Задача 3

На ленте записан текст, содержащий пробелы. Установить окно на первый пробел, если в начальный момент в окно виден первый символ текста.

Вася. Корректор должен перемещать окно вправо, пока видимый в нём символ — не пробел:

```
ЭТО Поиск_пробела
  ПОКА НЕ ПРОБЕЛ ВПРАВО
КОНЕЦ
```

Петя. В инструкции указаны ещё несколько разновидностей условий. Они выполняют сравнение символа в оконной ячейке ленты с символом, расположенным в ящике:

- `Я=Л` — условие истинно, если символ в ящике совпадает с символом в окне.
- `Я#Л` — условие истинно, если символ в ящике отличается от символа в окне.
- `Я>Л` — условие истинно, если символ в ящике имеет больший порядковый номер в алфавите Корректора по сравнению с символом в окне.
- `Я<Л` — условие истинно, если символ в ящике имеет меньший порядковый номер в алфавите Корректора по сравнению с символом в окне.

Папа. Вот задача, в которой пригодятся условия, связанные с ящиком Корректора.

Задача 4

На ленте записан текст и окошко установлено на первый его символ. Известно, что первый символ встречается в тексте ещё несколько раз. Найти первое его повторение. На рис. 9.7 показан пример исходных данных и результат обработки.

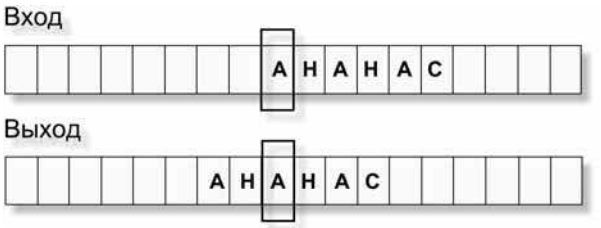


Рис. 9.7

Вася. Если бы у Корректора не было ящика, я бы не знал, как решить эту задачу. Ну, а с ящиком всё очень просто: сначала мы поместим в него первый символ текста, а затем будем перемещать окно вдоль текста, пока символы в ящике и на ленте отличаются:

```
ЭТО Поиск_повтора
  ЯЩИК+
  ВПРАВО
  ПОКА Я#Л ВПРАВО
КОНЕЦ
```

Петя. Давайте напишем программу с условием `ЦИФРА`. Это условие принимает значение истина, если в окошке на ленте записан один из символов 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Папа. Вот простая задача на использование этого типа условия.

Задача 5

Найти в записи на ленте первое вхождение цифры (известно, что цифры есть). В начальный момент окно установлено на начало записи. На рис. 9.8 показан пример исходных данных и результат обработки.

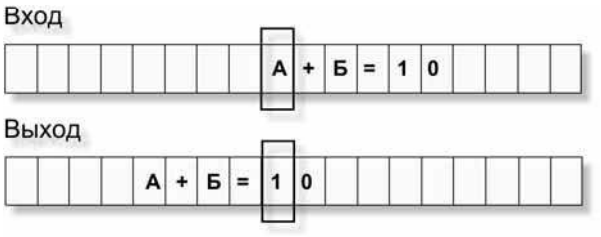


Рис. 9.8

Вася. Вот моё решение:

```
ЭТО Поиск_цифры
  ПОКА НЕ ЦИФРА ВПРАВО
КОНЕЦ
```

Вопросы и упражнения

1. Текст на ленте начинается, возможно, серией пробелов. Стереть все начальные пробелы. Окно перед работой установлено на первый символ текста. На рис. 9.9 показан пример исходных данных и результат обработки.

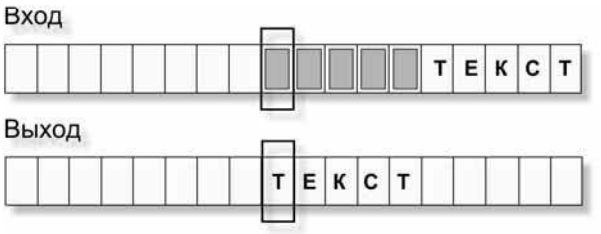


Рис. 9.9

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. 9.10 показан пример исходных данных и результат обработки.

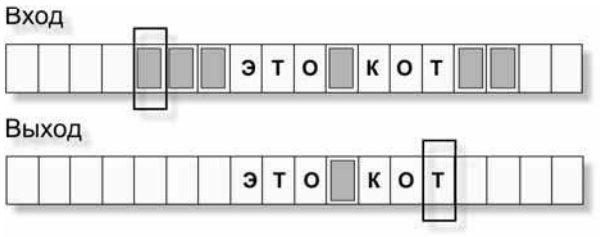


Рис. 9.10

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. 9.11 показан пример исходных данных и результат обработки.

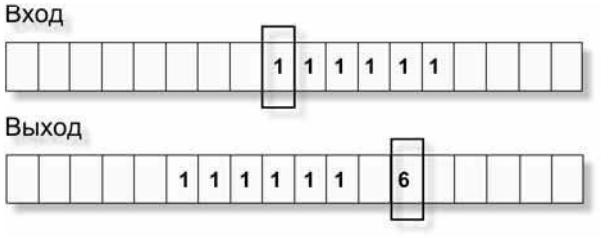


Рис. 9.11

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. 9.12 показан пример исходных данных и результат обработки.

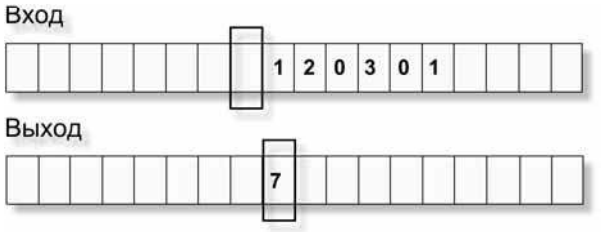


Рис. 9.12

9.3. Развилки

Матроскин говорит:
— А если они с Шариком на охоту пошли, на утреннюю зорьку? И вернутся только к вечеру?
— Такие события заранее планировать надо, — говорит тётя Тамара. — Я думаю, что с этого дня мы все будем жить по плану.

Э. Успенский

Вася. Решая задачи Кукарачи, мы использовали ещё одну команду языка программирования — команду ветвления.

Петя. И ты помнишь, как она записывается?

Вася. Помню, конечно! Команда ветвления содержит слова **ЕСЛИ**, **ТО**, **ИНАЧЕ**. Вот как она выглядит:

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

Слова **ЕСЛИ**, **ТО**, **ИНАЧЕ** — ключевые слова языка программирования. Они, как цемент, связывают команду ветвления воедино!

Работает команда так: интерпретатор программ проверяет условие и, если оно верное, выполняет первую команду, а если нет — выполняет вторую. В любом случае работает только одна команда: либо `команда1`, либо `команда2`. Условие в команде ветвления записывается и проверяется точно так же, как в команде цикла **ПОКА** (рис. 9.13).

Петя. Правильно! Решим такую задачу.

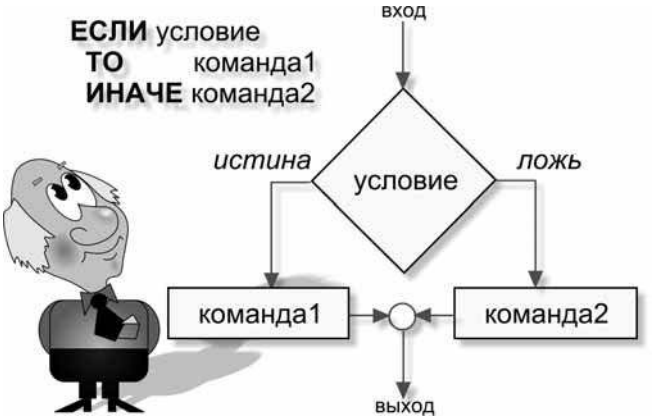


Рис. 9.13. Схема работы команды ветвления

Задача 6

На ленте записано целое число, большее 9, но меньшее 99. Увеличить его на один, если в начальный момент в окно виден последний символ числа. На рис. 9.14 показан пример исходных данных и результат обработки.

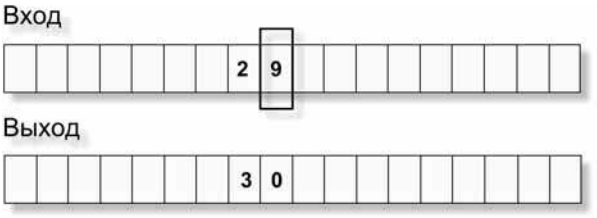


Рис. 9.14

Вася. Но такая задача решалась нами без всяких ветвлений одной командой **плюс**!

Петя. Не торопись. А если число оканчивается на цифру девять?

Вася. Да... Тогда команда **плюс** на место девятки запишет букву А.

Я понял! Тут, действительно, без развилки не обойтись. Сначала надо проверить последнюю цифру. Если она не девять, применить команду **плюс**, а если девять, обработать отдельно: вместо девятки записать ноль, сместиться влево и применить команду **плюс** к первой цифре числа: этот **плюс** сработает правильно всегда — ведь по условию первая цифра не может быть девяткой! Вот моё решение:

```
ЭТО Плюс1
  ЕСЛИ 9
    ТО    Девять // Особый случай
    ИНАЧЕ ПЛЮС
КОНЕЦ

ЭТО Девять
  ПИШИ 0
  ВЛЕВО
  ПЛЮС
КОНЕЦ
```

Папа. Обращаю ещё раз ваше внимание на то, что после слов **то** и **иначе** можно писать только одну команду. Кроме того, запись «**иначе** команда2» может отсутствовать, и тогда команда принимает вид:

```
ЕСЛИ условие ТО команда1
```

Эта запись равнозначна такой:

```
ЕСЛИ условие
  ТО    команда1
  ИНАЧЕ СТОЯТЬ
```

Команда **СТОЯТЬ** — пустая, выполняя её, Корректор не предпринимает никаких действий.

Петя. Посмотрите! Процедура Девять, расположенная на ветви **ТО** в условной команде Васиного решения, заканчивается командой **ПЛЮС**. Та же команда записана и на ветви **ИНАЧЕ**. Можно упростить программу, вынеся общую команду **ПЛЮС** за развилку, при этом ветвь **ИНАЧЕ** уже не потребуется:

```
ЭТО Плюс1
  ЕСЛИ 9 ТО { ПИШИ 0 ВЛЕВО }
  ПЛЮС
КОНЕЦ
```

Вася. Здорово!

Папа. Вот ещё одна «развилочная» задача.

Задача 7

На ленте записан текст, в котором злоумышленник заменил все буквы Л на К. Известно, что в исходном тексте букв К не было. Восстановить исходный текст. В начальный момент в окно виден первый символ записи.

Вася. Какой текст мне записать на ленту?

Папа. Ну, давай запишем текст «КИСА НА КУГУ», и пусть Корректор его расшифрует.

Вася вошёл в редактор среды, записал на ленту эту забавную фразу и стал думать, как написать решение (рис. 9.15).

Рис. 9.15

Вася. Корректор должен перемещать окно вдоль текста и проверять символы. Перемещение удобно записать при помощи цикла **ПОКА**: окно перемещается, пока в нём не **ПУСТО**. Пустая клетка — признак конца текста.

Начало программы может выглядеть так:

```
ЭТО Шифр
  ПОКА НЕ ПУСТО
  {
    Проверка
    Шаг
  }
КОНЕЦ
```

Петя. Неплохо! Мне кажется, занятия с Кукарачей не прошли для тебя даром.

Вася, довольный похвалой брата, продолжал свои рассуждения.

Вася. Проверка должна содержать команду ветвления: если в окне видна буква К, её нужно заменить буквой Л:

```
ЭТО Проверка
  ЕСЛИ К ТО ПИШИ Л
КОНЕЦ
```

Ну, а процедура Шаг совсем проста: она перемещает окно на одну клетку вправо:

```
ЭТО Шаг
  ВПРАВО
КОНЕЦ
```

Вопросы и упражнения

1. Как вы думаете, почему по условию задачи 6 число на ленте должно быть больше 9, но меньше 99?
2. На ленте Корректора сложилась такая обстановка (рис. 9.16).

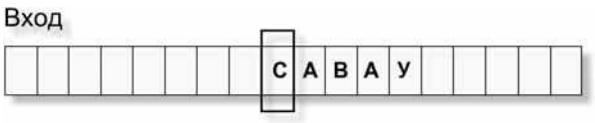


Рис. 9.16

Какое слово будет на ленте после выполнения следующей программы:

```
ЭТО Сава
  ПОВТОРИ 5
```

```
  {
    ЕСЛИ A
     ТО    ПОВТОРИ 14 ПЛЮС
     ИНАЧЕ МИНУС
     ВПРАВО
  }
КОНЕЦ
```

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. 9.17 показан пример исходных данных и результат обработки.

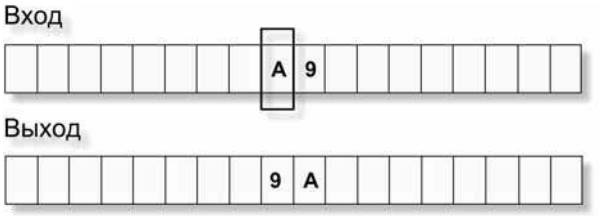


Рис. 9.17

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. 9.18 показан пример исходных данных и результат обработки.

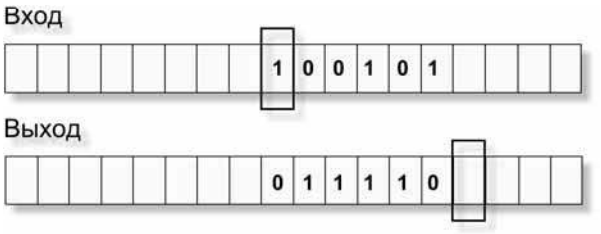


Рис. 9.18

9.4. Рекурсия

> Этот комбинезон застёгивался сверху на одну кнопку, которая была на затылке.
>
> *Н. Носов*

Петя. Мы ещё не рассматривали такой важный приём программирования, как рекурсия.

Вася. По работе с Кукарачей рекурсия мне хорошо известна. Если написать такую программу:

```
ЭТО Поход
  ВПРАВО
  Поход
КОНЕЦ
```

то Корректор будет безостановочно перемещать окно вдоль ленты, пока мы не выключим компьютер!

Петя. Такая программа называется рекурсивной. Рекурсия — это способ программирования, при котором программа вызывает сама себя. Рекурсия, которая не имеет проверки на окончание, как в твоей программе, называется бесконечной.

Папа. Попробуйте переписать программу `Шифр` (задача 7) в рекурсивном варианте.

Вася и Петя немного поспорили, но потом пришли к единому мнению и показали папе своё решение:

```
ЭТО Шифр
  ЕСЛИ К ТО ПИШИ Л // Проверка и замена, если нужно
  ВПРАВО           // Шаг к следующему символу
  ЕСЛИ НЕ ПУСТО    // Проверка на окончание
    ТО Шифр        // Рекурсивный вызов
КОНЕЦ
```

Вопросы и упражнения

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 оно присутствует в тексте, или на первый пустой символ за концом текста, если многоточие отсутствует. В начальный момент в окно виден первый символ текста. На рис. 9.19 показан пример исходных данных и результат обработки.

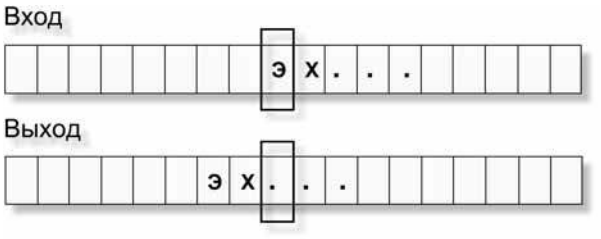


Рис. 9.19

2. В окно виден первый символ записи. Проверить, следуют ли символы в записи в порядке возрастания их номеров. На рис. 9.20 показан пример исходных данных и результат обработки.

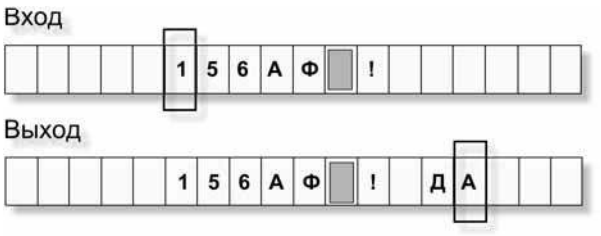


Рис. 9.20

Справочник по языку программирования

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, показанный на рис. 9.21.

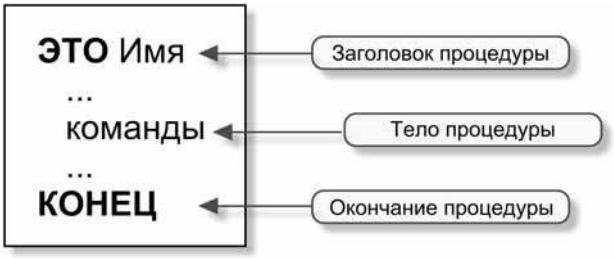


Рис. 9.21. Описание процедуры

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

Список команд приводится в табл. 9.1.

Таблица 9.1

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

Таблица 9.1 *(окончание)*

Команда языка	Как выполняется
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

Справочник по условиям Корректора

Список условий (без модификатора **НЕ**) приводится в табл. 9.2.

Таблица 9.2

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.

Глава 10

Отладка программ

Надежде Семёновне Кук по роду своей деятельности приходится много работать с музыкальными инструментами. Она знает: чтобы скрипка не звучала фальшиво, её надо тщательно настроить.

Так и в программировании: для того, чтобы программа работала правильно, её надо тщательно отладить (рис. 10.1).

Рис. 10.1. Отладка — важный этап разработки программ

Отладка — очень важный этап разработки программ. Об этом и пойдёт сегодня «мужской» разговор в семье Куков.

10.1. Корректор развлекается с котом

— А лечат-то как! Тьфу! Снаружи ставят медовые пластыри и внутрь дают мёду. Это ведь неправильно: снаружи надо йодом, а внутрь — касторку.

Н. Носов

Папа. Начнём работу с такой задачи.

Задача 1

На ленте написано трёхсимвольное слово, и в начальный момент окно установлено на его начало. Написать программу, которая меняет местами первый и последний символы слова. На рис. 10.2 показан пример исходного и результирующего состояний среды.

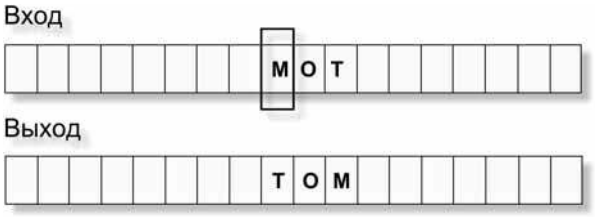


Рис. 10.2

Петя. Значит, если на ленте написано слово КОТ, Корректор должен заменить его словом ТОК.

Вася. А если на ленте записано слово СОН, Корректор превратит его в слово НОС.

Папа. Правильно! С условием задачи разобрались, давайте приступим к решению.

Петя. Я думаю, без ящика здесь не обойтись.

Папа. Предчувствие тебя не обмануло!

Вася. Копируем первый символ в ящик (рис. 10.3).

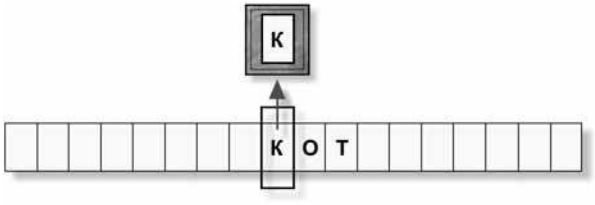


Рис. 10.3

Идём в конец слова и записываем на место последнего символа содержимое ящика (рис. 10.4). Потом...

Петя. Стоп! Посмотри, что ты натворил: последний символ стёрт с ленты безвозвратно!

Вася. Понял. Содержимое ящика нужно копировать рядом с последним символом (рис. 10.5).

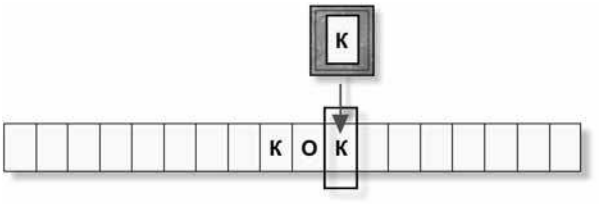


Рис. 10.4

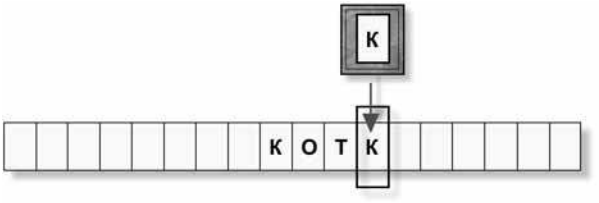


Рис. 10.5

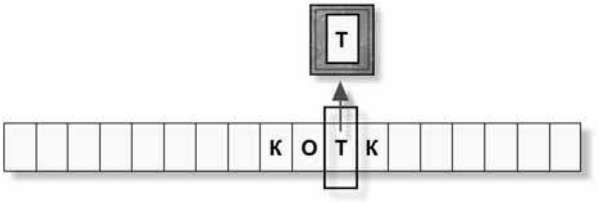


Рис. 10.6

Копируем в ящик последний символ слова (рис. 10.6).

Идём к началу слова и заменяем первый его символ содержимым ящика (рис. 10.7).

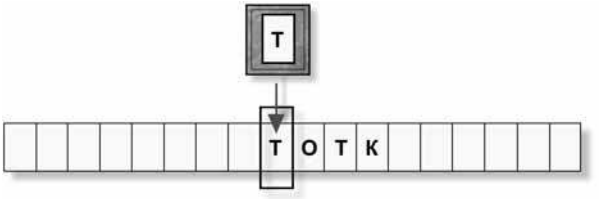


Рис. 10.7

Перемещаем окно на копию первого символа, записываем её в ящик и стираем с ленты (рис. 10.8).

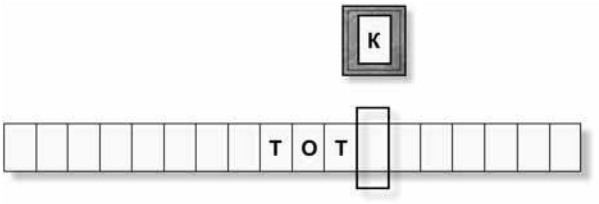


Рис. 10.8

Осталось только сместиться на одну ячейку влево и переписать символ из ящика на ленту (рис. 10.9).

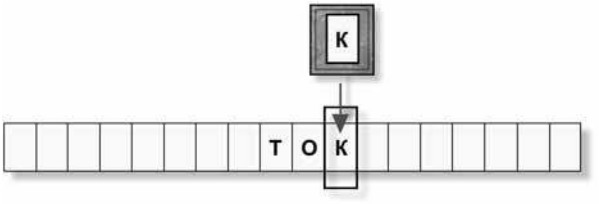


Рис. 10.9

Папа. Теперь оформи свой алгоритм в виде программы.

Вася написал в программном поле следующий текст:

```
ЭТО Перевертыш
  // Помещаем первый символ за словом.
  ЯЩИК +
  ПОВТОРИ 3 ВПРАВО
  ЯЩИК-
  // Последний символ на место первого.
  ВЛЕВО
  ЯЩИК+
  ПОВТОРИ 2 ВЛЕВО
  ЯЩИК-
  // Заменяем последний символ слова.
  ПОВТАРИ 3 ВПРАВО
  ЯЩИК+
  ПИШИ ПУСТО
  ЯЩИК_
КОНЕЦ
```

10.2. Синтаксические ошибки

Незнайка пожал плечами и сел на постели.
— Не нужно пожимать плечами, больной, — заметила Медуница. — Покажите язык.

Н. Носов

Папа. Ну, запускай свою программу.

Вася нажал экранную кнопку *GO* и увидел на экране сообщение (рис. 10.10).

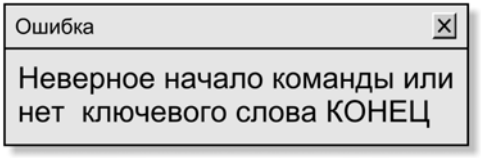


Рис. 10.10

Петя. Интерпретатор обнаружил в программе синтаксическую ошибку и выдал диагностическое сообщение.

Вася. Какие умные слова ты говоришь! Переведи, пожалуйста, это на человеческий язык.

Петя. **Синтаксическая ошибка** — это несоответствие текста программы правилам языка программирования. Интерпретатор, прежде чем выполнять программу, производит проверку её правильности. Сейчас он обнаружил ошибку и честно сказал об этом.

Вася. А где же ошибка?

Петя. Интерпретатор пометил место, в котором обнаружил ошибку (рис. 10.11).

Вася. Да, ошибку я теперь вижу: между словом ЯЩИК и знаком «+» не должно быть пробела. Но почему интерпретатор вместо того, чтобы прямо сказать «убери пробел», выдал такое странное сообщение?

Петя. Откуда ему знать, что ты написал команду из СКИ с ошибкой? Интерпретатор решил, что слово ЯЩИК — это имя новой процедуры, и она будет описана позже. Значит, со словом ЯЩИК всё в порядке. Следующий знак «+» он воспринял как неверный и выдал сообщение.

Вася исправил ошибку и запустил программу. На экране появилось прежнее сообщение, а пометка появилась теперь в другой строке (рис. 10.12).

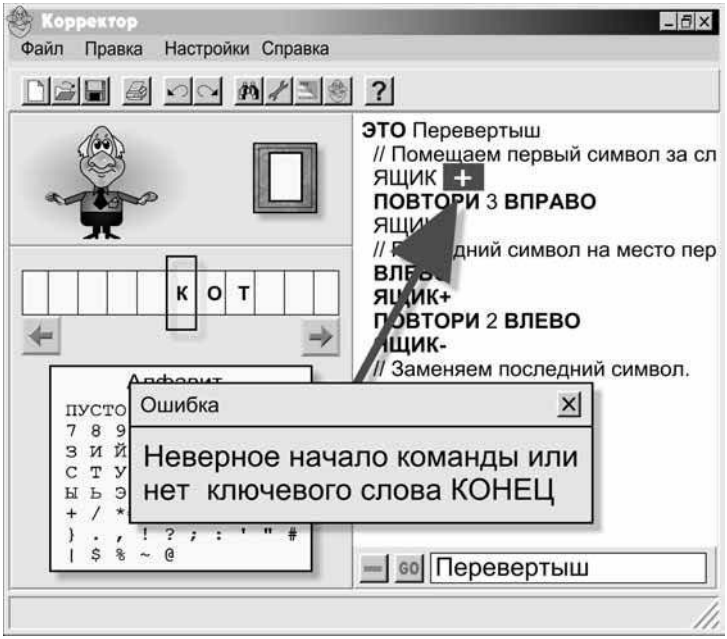


Рис. 10.11

Рис. 10.12

Петя. Та же ситуация. Слово `повтори` написано с ошибкой, и интерпретатор «не узнал» его. Он, конечно, решил, что `ПОВТАРИ` является именем процедуры, а цифра три, следующая за вызовом этой процедуры, — неверное начало новой команды.

Вася, смутившись своей описки, написал команду цикла правильно и снова запустил программу. Результат опять оказался отрицательным, но сообщение на экране было уже другим (рис. 10.13).

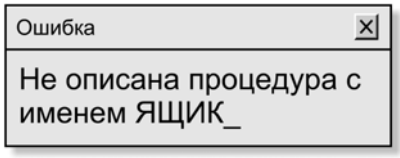


Рис. 10.13

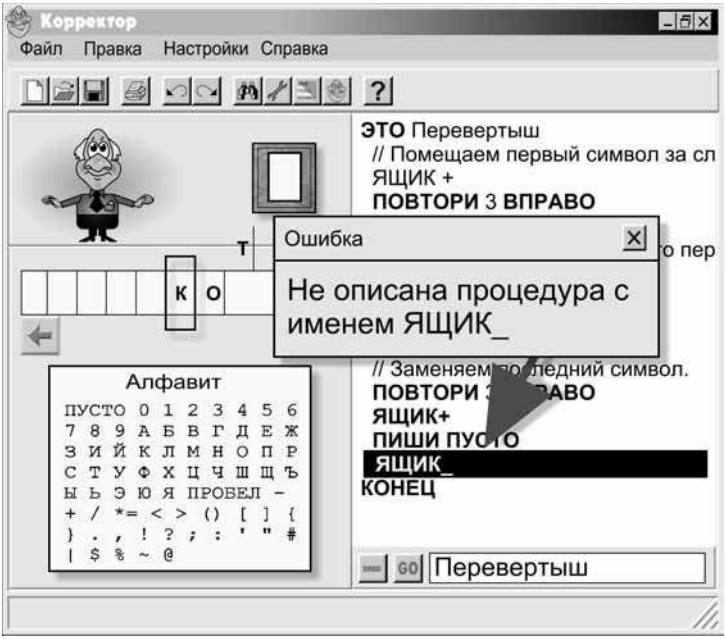


Рис. 10.14

На этот раз интерпретатор выделил строку `ЯЩИК_` (рис. 10.14).

Вася. Попробую сам объяснить, что произошло: команда из СКИ написана с ошибкой, поэтому интерпретатор не узнал её.

Петя. Верно. Он решил, что это вызов процедуры.

Вася исправил ошибку, запустил программу. На этот раз она стала выполняться, но из слова КОТ получился не ТОК, а непонятное слово КОТК.

Вася. ???

Петя. Синтаксических ошибок в твоей программе нет, и интерпретатор выполнил её, но в твоей программе есть **семантическая ошибка**!

Вопросы и упражнения

1. Что такое синтаксическая ошибка?
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
```

3. Исправьте синтаксические ошибки предыдущей программы в среде Корректора.
4. Что должна делать программа, описанная в вопросе 2?

10.3. Ошибки программирования

> — Так вот, — продолжал Незнайка. — Летим, значит, выше. Вдруг — бум! Не летим выше. Смотрим — на облако наскочили. Что делать? Взяли топор, прорубили в облаке дырку.
>
> *Н. Носов*

Петя. Ошибки программирования можно условно разделить на *алгоритмические* (ошибки в алгоритме) и на ошибки *кодирования* (ошибки в программе).

Вася. Алгоритм мы составили правильно.

Петя. Это так. У тебя ошибка в программе.

Вася. Но интерпретатор не обнаружил никаких ошибок!

Петя. Ошибки кодирования можно разделить на два класса: *синтаксические* и *семантические*. В программе нет синтаксических ошибок, поэтому интерпретатор не выдал никаких сообщений и запустил код на выполнение. Но в твоей программе есть семантическая ошибка.

Если программа написана правильно с точки зрения правил языка программирования, но делает не то, что предписано алгоритмом, по которому она

построена, то в программе есть семантические ошибки. Иными словами, семантическая ошибка — это погрешности, допущенные при переводе алгоритма в код программы, вызванные недопониманием смысла конструкций языка программирования или случайными описками.

Приведу простой пример. Программист записал в программном коде:

```
ПОВТОРИ 3 ПИШИ * ВПРАВО
```

рассчитывая получить на ленте три звёздочки.

Звёздочка, конечно, получилась одна. Правильный код должен быть таким:

```
ПОВТОРИ 3 { ПИШИ * ВПРАВО }
```

Программист допустил семантическую ошибку: написал синтаксически правильную конструкцию, которая делает совсем не то, что было задумано.

Классификация ошибок программирования изображена в виде схемы на рис. 10.15.

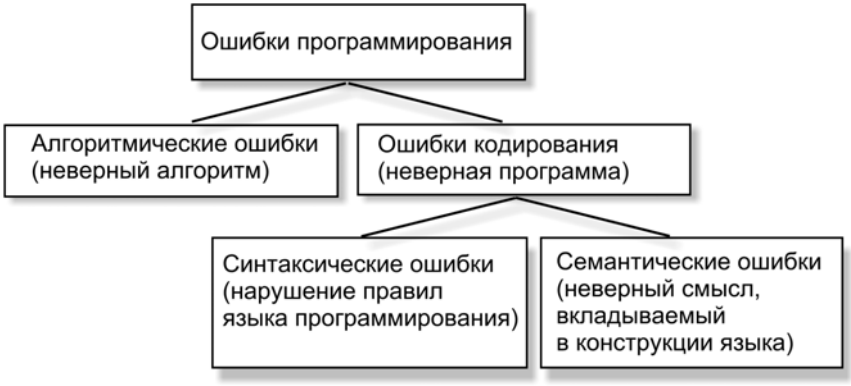


Рис. 10.15

Папа. Подводя итог разговору о классификации ошибок программирования, скажу так. Ошибки, допущенные на этапе разработки алгоритма, называют алгоритмическими. Ошибки, допущенные при переводе алгоритма в код программы, — ошибками кодирования. Если при этом нарушаются грамматические правила языка, то это синтаксическая ошибка, а если с грамматикой всё в порядке, то ошибка — семантическая.

Хочу заметить, что семантическая ошибка по своей сути близка алгоритмической ошибке. Ведь кодирование алгоритма — это продолжение работы над алгоритмом, детализация и конкретизация его шагов при помощи конструкций языка программирования. Можно сказать, что семантическая ошибка — это алгоритмическая ошибка, допущенная на этапе перевода алгоритма в программный код. Ошибка может появиться из-за неверного понимания работы конструкции языка программирования, из-за случайной

описки или из-за неверного проектирования алгоритмического шага. Приведённая на рис. 10.15 классификация отражает лишь место возникновения ошибки в процессе разработки программного продукта и только частично её суть.

Вася. А как найти семантическую ошибку?

Петя. Можно запустить выполнение программы по шагам.

Интерпретатор умеет работать в двух режимах — *непрерывном* и *пошаговом*. В непрерывном режиме он выполняет программу до конца, не останавливаясь, а в пошаговом прерывает работу после выполнения каждой команды, отмечая на экране подсветкой то, что ему предстоит выполнить на следующем шаге.

Наблюдая за тем, что выполняется на поле программ и что делает Корректор, легко найти место семантической ошибки.

Вася включил пошаговый режим, нажал кнопку *GO* и увидел, как Корректор скопировал букву К с ленты в свой ящик.

Выполняя программу по шагам, Вася увидел неверные действия Корректора почти в самом конце программы и сразу понял, в чём дело: перед последней командой **ЯЩИК-** отсутствовала команда **ВЛЕВО**. После исправления программа выглядела следующим образом:

```
ЭТО Перевертыш
  // Помещаем первый символ за словом.
  ЯЩИК+
  ПОВТОРИ 3 ВПРАВО
  ЯЩИК-
  // Последний символ на место первого.
  ВЛЕВО
  ЯЩИК+
  ПОВТОРИ 2 ВЛЕВО
  ЯЩИК-
  // Заменяем последний символ слова.
  ПОВТОРИ 3 ВПРАВО
  ЯЩИК+
  ПИШИ ПУСТО
  ВЛЕВО
  ЯЩИК-
КОНЕЦ
```

Вася. Сейчас я переключу режим и запущу Корректор на непрерывную работу без остановок.

На этот раз программа выполнилась без ошибок, и Корректор в результате своих трудов превратил слово КОТ в слово ТОК.

Папа. Я внимательно следил за отладкой программы и вижу, что вы с ней отлично справились. Но программа для решения нашей задачи может быть гораздо короче, если использовать команду **ОБМЕН**.

Петя. Да, я забыл про эту команду из СКИ Корректора.

Вася. А как она работает?

Петя. Эта команда меняет местами символы, записанные в оконной ячейке и ящике исполнителя.

Вася. Действительно, полезная команда! Вот новый алгоритм решения нашей задачи:

1. Поместить первый символ в ящик.
2. Идти в конец слова.
3. Обменять местами символы в ящике и на ленте.
4. Идти в начало слова.
5. Обменять местами символы в ящике и на ленте.

Ну, а программа, конечно, получается короче:

```
ЭТО Перевертыш
  // 1. Поместить первый символ в ящик.
  ЯЩИК+
  // 2. Идти в конец слова
  ПОВТОРИ 2 ВПРАВО
  // 3. Обменять местами символы в ящике и на ленте.
  ОБМЕН
  // 4. Идти в начало слова.
  ПОВТОРИ 2 ВЛЕВО
  // 5. Обменять местами символы в ящике и на ленте.
  ОБМЕН
КОНЕЦ
```

Вопросы и упражнения

1. Что такое семантическая ошибка?
2. Укажите ошибки в решении указанной ниже 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
```

10.4. Тестирование

> В это время корзина с силой ударилась о землю и перевернулась.
>
> *Н. Носов*

Вася. Программа написана, ошибки исправлены, значит, работа закончена, можно пойти погулять!

Петя. Погулять, конечно, можно, но работа над программой ещё не завершена.

Вася. Как так? Хочешь сказать, что программу можно написать ещё короче?

Петя. Программу нельзя считать завершённой, пока она не прошла этап *тестирования*.

Вася. Тестирование — это проверка?

Петя. Верно, проверка.

Вася. Но мы же проверили программу для слова КОТ, и киска превратилась в ТОК!

Петя. А ты уверен, что, например, слово МИР программа преобразует в слово РИМ?

Вася. Конечно, уверен! Ведь программа переставляет местами первую и третью буквы слова, и ей всё равно, какие это буквы!

Петя. А если исходная запись содержит не буквы, а другие символы, например, цифры, спецзнаки, пробелы, первый алфавитный символ `ПУСТО` или последний — @?

Вася. Так что, нужно проверить все группы из трёх символов, взятых из алфавита Корректора?

Петя. Практически это невозможно. На каждом из трёх мест может быть любой символ алфавита. Если в алфавите 100 символов, то всего получается $100 \times 100 \times 100 = 1000\,000$ комбинаций. Если выполнять проверку со скоростью 1 комбинация в минуту по 10 часов в день, то на полную проверку уйдет 1000 000/60/10/365 — около четырёх с половиной лет.

Вася. Ого! Так и состариться недолго, проверяя одну простую программу!

Петя. В практике проверки (тестирования) программ ограничиваются, конечно, существенно меньшим набором данных (*тестов*). В проверочные наборы включают несколько типичных входных данных и, желательно, все особые случаи.

В табл. 10.1 показан возможный набор тестов для нашей программы (пробел обозначен знаком ■, пустая ячейка — знаком □, а подчёркнутый символ показывает положение окна на ленте).

Таблица 10.1

Тест	Комментарий	Ожидаемый результат
КОТ	Типичные данные	ТОК
МИР	Типичные данные	РИМ
1□2	Особый случай	2□1
□!?	Особый случай	?!□
□:□	Особый случай	□:□
@□□	Особый случай	□□@

Вася. Я проверил программу для данных из таблицы. Результаты проверок совпали с ожидаемыми результатами. Значит, теперь можно считать, что программа работает правильно?

Петя. К сожалению, полной уверенности, конечно, нет. И здесь дело даже не в нашей конкретной программе: мы знаем алгоритм её работы, он действительно не зависит от конкретных значений символов. Но, вполне возможно, что на некоторых тестах проявятся ошибки интерпретатора программ, и наша программа станет работать неверно!

Вася. Вот как! Но в таком случае мы не виноваты! Виноват разработчик интерпретатора из Роботландии!

Петя. Это так, но пострадает пользователь программы, он не будет разбираться в истинной причине ошибки, а плохо подумает об авторе программы, то есть о Васе Куке, и будет прав: программы надо тщательно тестировать!

Вопросы и упражнения

Предложите наборы тестов для программ, решающих следующие задачи.

1. Скопировать первый символ на место последнего в записи.
2. Увеличить целое неотрицательное число на 1.
3. Уменьшить целое положительное число на 1.
4. Сосчитать число пробелов в тексте.
5. Удалить все пробелы в тексте.
6. Найти в записи символ, имеющий максимальный номер в алфавите Корректора.
7. Упорядочить символы в записи в порядке неубывания их номеров в алфавите Корректора.
8. На ленте через пробел записаны два числа. Установить окно на первую цифру наибольшего из них.
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.
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.

10.5. Задачи

1. Проверить, есть ли в записи на ленте цифры. В начальный момент в окно виден первый символ записи. На рис. 10.16 показан пример исходного и результирующего состояний среды.

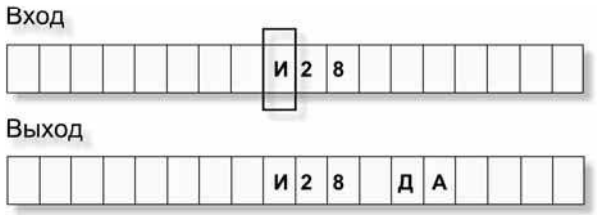


Рис. 10.16

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. 10.17 показан пример исходного и результирующего состояний среды.

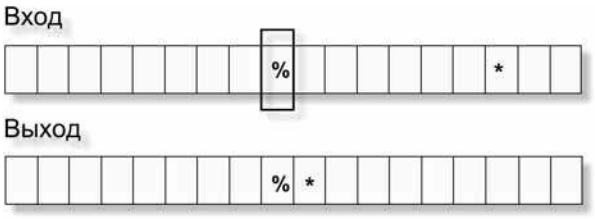


Рис. 10.17

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. 10.18 показан пример исходного и результирующего состояний среды.

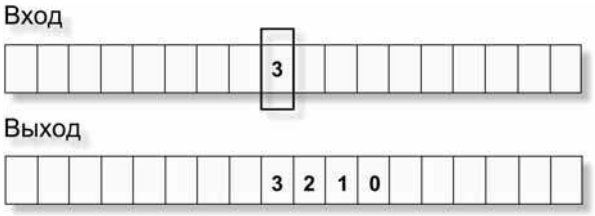


Рис. 10.18

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. 10.19 показан пример исходного и результирующего состояний среды.

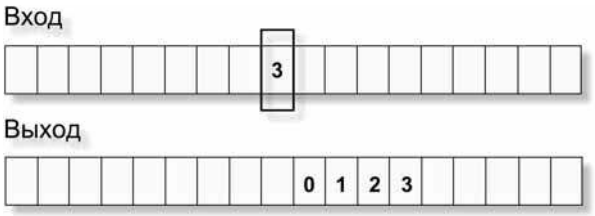


Рис. 10.19

Глава 11

Приёмы программирования Корректора

У Пети сегодня свободный день, и он посвятил его работе с Корректором. Решая разные задачи, Петя заметил, что в программах часто используются одни и те же приёмы программирования. Когда вечером у компьютера собралась обычная компания, Петя решил рассказать про свои находки папе и брату (рис. 11.1).

Рис. 11.1. У Пети много новых идей

11.1. Как найти конец текста

> Винтик перекинул верёвку через ветку яблони и велел Торопыжке привязать верхний конец верёвки к черенку яблока. Другой конец велел держать нескольким малышкам.
>
> *Н. Носов*

Папа. Итак, друзья, какой план работы у нас на сегодня?

Петя. У меня есть интересные идеи, которые помогут решать задачи Корректора.

Папа. Давай, послушаем.

Петя. При программировании Корректора часто приходится устанавливать окно на начало или конец текста.

Папа. Что ты понимаешь под текстом?

Петя. Ну, как всегда, это последовательность любых символов на пустой ленте.

Вася. А пробелы между словами тоже относятся к тексту?

Петя. Да, конечно. Название «текст» — это условное обозначение произвольной последовательности символов из алфавита исполнителя. Среди символов, составляющих текст, не должно быть специального символа **ПУСТО**, но могут быть пробелы, цифры или специальные знаки, такие, как «+», «–», «%» или что-нибудь ещё в этом роде. Вот я изобразил несколько примеров на рис. 11.2.

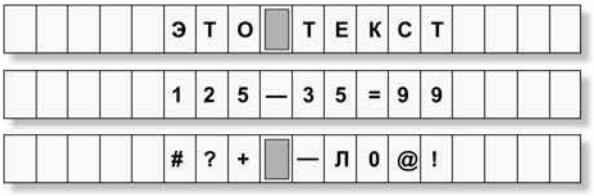


Рис. 11.2. Примеры записи текста на ленте Корректора

Папа. Сформулируй свою задачу.

Петя. Пожалуйста!

Задача 1

На ленте записан некоторый текст. Требуется установить окно на последний его символ. В начальный момент окно расположено произвольно внутри текста.

Вася. Понятно. Решение очень простое. Перемещаем окно вправо, пока в нём видны символы текста. Как только в окне появится пустая клетка — смещаем окно на одну ячейку назад:

```
ЭТО На_конец
  ПОКА НЕ ПУСТО ВПРАВО
  ВЛЕВО
КОНЕЦ
```

Петя. Верно. Это самое удачное решение, хотя можно предложить и вариант с рекурсией:

```
ЭТО На_конец
  ВПРАВО
  ЕСЛИ ПУСТО
    ТО ВЛЕВО
    ИНАЧЕ На_конец
КОНЕЦ
```

Вася. То же самое можно записать и по-другому:

```
ЭТО На_конец
  ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО На_конец
    ИНАЧЕ ВЛЕВО
КОНЕЦ
```

Вопросы и упражнения

1. На ленте записан некоторый текст. Требуется установить окно на первый его символ. В начальный момент окно расположено произвольно внутри текста (рис. 11.3).

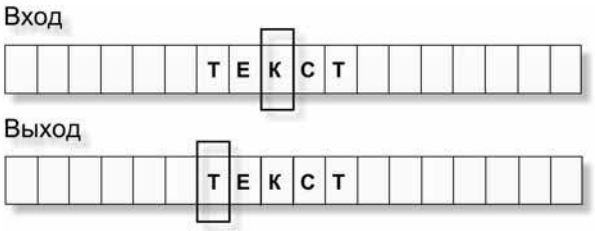


Рис. 11.3

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. 11.4).
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. 11.5).

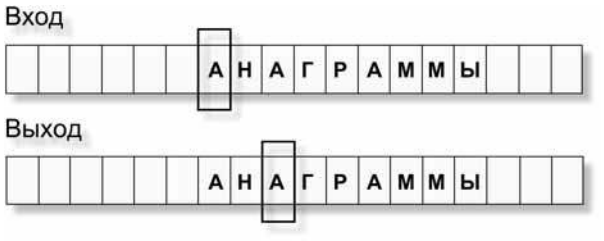


Рис. 11.4

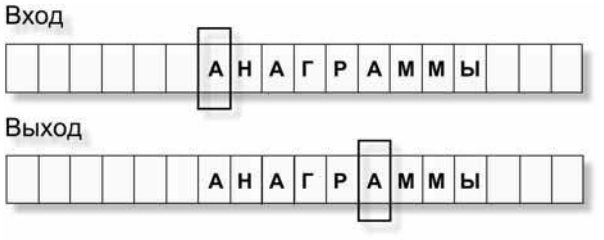


Рис. 11.5

11.2. Как вернуться в исходное место

— Что же Знайка будет делать без нас? — спросил Пончик.
— Ну что, — развёл руками Авоська. — Пойдёт себе потихоньку домой.

Н. Носов

Папа. Вариант с рекурсией может помочь вернуть окно Корректора в исходное положение.

Петя. Что ты имеешь в виду?

Папа. Представьте себе такую задачу.

Задача 2

На ленте записан текст и окно установлено где-то внутри него. Требуется поставить в конце текста точку и вернуть окно в исходное положение. До начала работы лента может, например, выглядеть так, как показано на рис. 11.6.

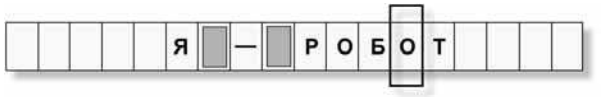


Рис. 11.6

А после выполнения программы лента должна иметь вид, представленный на рис. 11.7.

Рис. 11.7

Вася. Понимаю трудность. Поставить в конце текста точку очень просто, но как вернуть окно назад? Ведь мы не знаем, на сколько клеток оно переместилось вправо. И непонятно, как здесь может помочь рекурсия.

Петя. Понятно! Можно использовать отложенную команду. Вспомни наши занятия с Кукарачей: он находил букву и возвращался в исходное положение при помощи отложенной в рекурсии команды (рекурсивной пружинки).

Вася. Честно говоря, я не помню.

Петя. Ну, хорошо, оставим Кукарачу в покое. Вот решение задачи 2:

```
// Поставить в конце текста точку и вернуться назад.
ЭТО Точка
  ВПРАВО
  ЕСЛИ ПУСТО
   ТО ПИШИ .
   ИНАЧЕ Точка
  ВЛЕВО // Отложенная команда. Она выполнится столько раз, сколько
        // раз выполнялась команда ВПРАВО.
КОНЕЦ
```

Вася. Мне кажется, что эта программа неверная. Точка, действительно, будет поставлена в конце текста, но окно после этого переместится только на одну клетку влево и, конечно, в общем случае, не вернётся в исходное положение.

Петя. Давай разберёмся не спеша... Первой выполняется команда **ВПРАВО** (рис. 11.8).

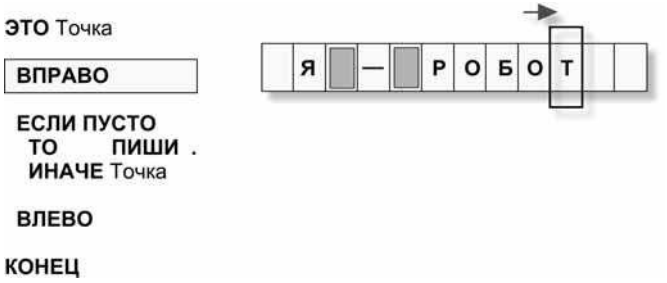


Рис. 11.8

Затем выполняется команда ветвления (рис. 11.9).

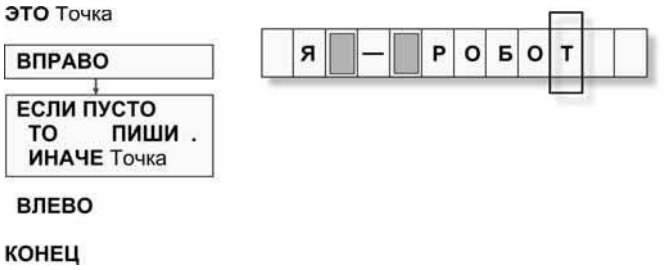


Рис. 11.9

Обрати внимание на то, что после выполнения команды ветвления должна выполняться команда, следующая за ней, т. е. команда **ВЛЕВО**.

Так как условие ложно, выполнение условной команды равносильно выполнению второго экземпляра процедуры `Точка`. Первая команда в ней — перемещение окна вправо на клетку (рис. 11.10).

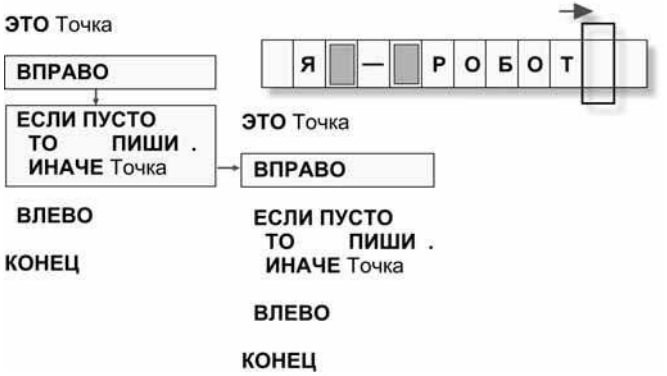


Рис. 11.10

Далее интерпретатор выполняет команду ветвления: условие истинно, значит, выполнится команда **ПИШИ** . (рис. 11.11).

Теперь выполняется команда **ВЛЕВО**, следующая за условной во втором экземпляре процедуры `Точка` (рис. 11.12).

Работа второго экземпляра процедуры `Точка` завершена и, тем самым, завершилось выполнение условной команды в первом экземпляре. Корректор приступает к выполнению последней команды **ВЛЕВО** и возвращает окно в исходное положение (рис. 11.13).

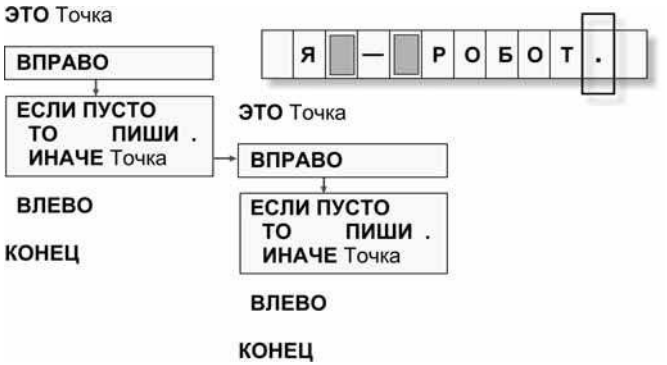


Рис. 11.11

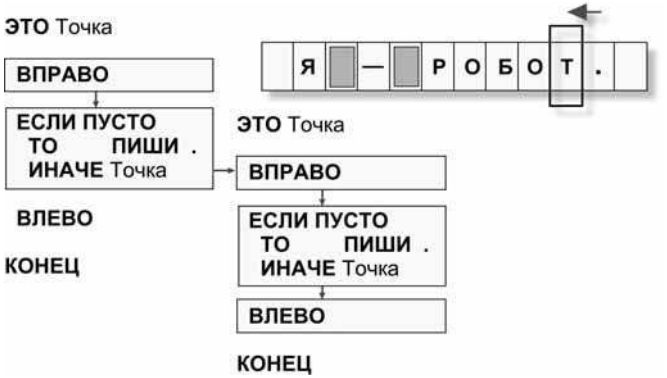


Рис. 11.12

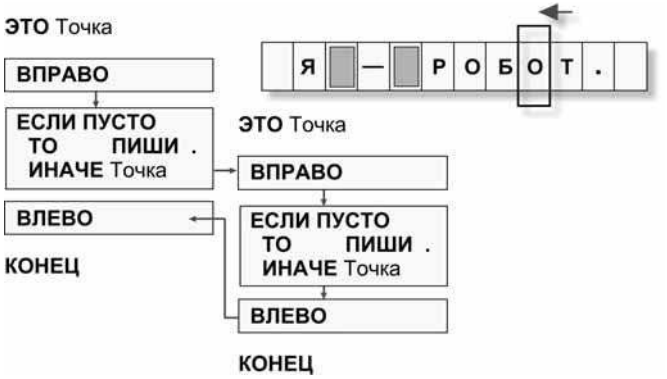


Рис. 11.13

Вася. Вот теперь я, наконец, вспомнил этот головоломный приём программирования! Твои слова и схемы мне понятны.

Петя. Я рад за тебя!

Папа. А я рад за вас обоих и немножко за себя — ведь я ваш папа!

Вопросы и упражнения

1. Окно установлено на один из символов текста. Заключите текст в круглые скобки и верните окно в исходное положение (рис. 11.14).

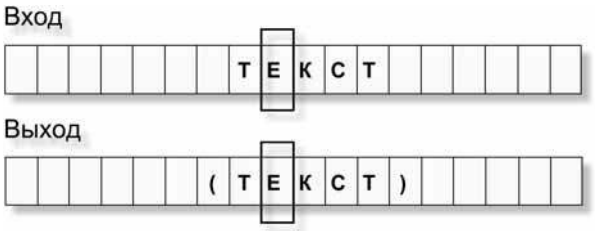


Рис. 11.14

2. Проверьте, 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. 11.15).

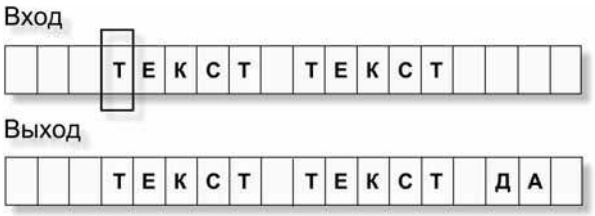


Рис. 11.15

11.3. Специальные символы-флаги

> Не разбирая дороги он забрёл на край города, наткнулся там на забор и набил на лбу шишку. Остановившись, он поднял глаза и увидел на заборе надпись: «Незнайка дурак».
>
> *Н. Носов*

Петя. При программировании Корректора можно использовать специальные символы для разных пометок, указателей. Я называю такие символы **флагами**. Давайте решим такую задачу.

Задача 3

Продублировать текст, записанный на ленте. В начальный момент окно установлено на первый его символ. На рис. 11.16 показан пример исходного и результирующего состояний среды.

Рис. 11.16

Вася. Точку в конце мы ставить умеем, но можно записывать на ленту не точку, а то, что находится в ящике!

Папа. Хорошее рассуждение. Ты намекаешь на то, что в ящик можно последовательно копировать символы текста и переносить их в конец?

Вася. Конечно! Вот моя программа:

```
ЭТО Дубль
  ПОКА НЕ ПУСТО
  {
    ЯЩИК+
    Перенести_символ
    ВПРАВО
  }
КОНЕЦ

// Эта процедура копирует символ из ящика в конец текста
// и возвращает окно в исходное положение.
// -----------------------------------------------------
ЭТО Перенести_символ
  ВПРАВО
  ЕСЛИ ПУСТО
    ТО    ЯЩИК-
    ИНАЧЕ Перенести_символ
  ВЛЕВО // Отложенная команда. Она выполнится столько раз,
        // сколько выполнялась команда ВПРАВО.
КОНЕЦ
```

Папа. Хорошая задача. В ней без флага действительно трудно обойтись: ведь выполнение твоей программы никогда не закончится!

Вася. Почему?

Папа. Копируя символы в конец, ты удлиняешь исходное слово, и поэтому условие в цикле **ПОКА** никогда не станет ложным.

Вася. Понимаю...

Петя. Выход очень простой: поставить за исходным текстом специальный символ — флаг конца текста.

Папа. При этом приходится допускать, что такого символа-флага в самом тексте нет.

Петя. Ну, я не думаю, что это ограничение очень серьёзное. В качестве символа флага можно выбрать редко используемый символ и честно предупредить пользователей нашей программы, чтобы они этот символ не использовали для записи текста на ленту.

Вася. Давайте в качестве флага использовать символ «|». Я вообще не знаю, для чего он нужен Корректору!

Петя. Вот ты и нашёл ему применение! Теперь алгоритм решения задачи примет следующий вид:

1. Поставить флаг в конец слова.
2. Копировать слово за флаг.
3. Стереть флаг.

Вася. А вот как запишется главная процедура программы:

```
ЭТО Дубль
  Поставить_флаг
  Копировать_слово
  Удалить_флаг
КОНЕЦ
```

Поставить флаг в конец текста и вернуться назад можно при помощи рекурсивной процедуры с отложенной командой **ВЛЕВО**:

```
ЭТО Поставить_флаг
  ВПРАВО
  ЕСЛИ ПУСТО
    ТО    ПИШИ |
    ИНАЧЕ Поставить_флаг
  ВЛЕВО
КОНЕЦ
```

Петя. Правильно, конечно. Но здесь можно обойтись и без рекурсивных хитростей — ведь положение окна нам известно: это начало текста. Значит, известно, куда надо вернуть окно после установки флага. Это легко программируется старым добрым циклом **ПОКА**:

```
ЭТО Поставить_флаг
  ПОКА НЕ ПУСТО ВПРАВО
  ПИШИ |
  ПОКА НЕ ПУСТО ВЛЕВО
  ВПРАВО
КОНЕЦ
```

Вася. Процедура `Копировать_слово` будет дублировать символы до тех пор, пока в окошке не появится флаг:

```
ЭТО Копировать_слово
  ПОКА НЕ |
  {
    ЯЩИК+
    Перенести_символ
    ВПРАВО
  }
КОНЕЦ
```

Для переноса символов годится моя процедура `Перенести_символ`, а процедура `Удалить_флаг` до смешного проста:

```
ЭТО Удалить_флаг
  ПИШИ ПУСТО
КОНЕЦ
```

Ведь цикл по копированию символов заканчивается тогда, когда в окошке появится флаг конца слова. Он-то и подлежит удалению.

Петя. Обман! Получилось не то, что требовалось: копия пристроена не в хвост текста, а через пустую клетку от него (рис. 11.17)!

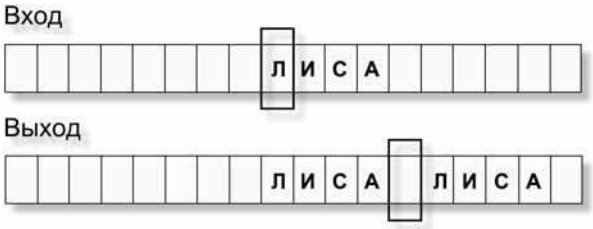


Рис. 11.17

Вася. Да, верно... Придётся удалять флаг по-другому — сдвигом копии на одну ячейку влево:

```
ЭТО Удалить_флаг
  ПОКА НЕ ПУСТО
  {
    ВПРАВО ЯЩИК+ ВЛЕВО ЯЩИК- // Символ справа копируется в текущую ячейку
    ВПРАВО                   // Окно на следующую ячейку
  }
КОНЕЦ
```

Петя. Это другое дело! Набери на компьютере эту программу и проверь её.

Вася набрал в программном поле свой текст, записывая комментарии в местах, которые считал особенно важными:

```
// Программа дублирования текста.
ЭТО Дубль
  Поставить_флаг
  Копировать_слово
  Удалить_флаг
КОНЕЦ

ЭТО Поставить_флаг
  ПОКА НЕ ПУСТО ВПРАВО
  ПИШИ |
  ПОКА НЕ ПУСТО ВЛЕВО
  ВПРАВО
КОНЕЦ

ЭТО Копировать_слово
  ПОКА НЕ |
  {
    ЯЩИК+
    Перенести_символ
    ВПРАВО
  }
КОНЕЦ

// Эта процедура копирует символ из ящика в конец текста
// и возвращает окно в исходное положение.
```

```
// -------------------------------------------------
ЭТО Перенести_символ
  ВПРАВО
  ЕСЛИ ПУСТО
    ТО    ЯЩИК-
    ИНАЧЕ Перенести_символ
  ВЛЕВО // Отложенная команда. Она выполнится столько раз,
        // сколько выполнялась команда ВПРАВО.
КОНЕЦ

ЭТО Удалить_флаг
  ПОКА НЕ ПУСТО
  {
    ВПРАВО ЯЩИК+ ВЛЕВО ЯЩИК- // Символ справа копируется в текущую ячейку
    ВПРАВО                   // Окно на следующую ячейку
  }
КОНЕЦ
```

Результат запуска этой программы теперь соответствует рис. 11.16.

Папа. А что если в качестве флага в нашей задаче использовать символ пустого места? Тогда мы бы сняли запрет на использование символа «|» в тексте.

Петя. Мысль интересная и, думаю, осуществимая.

Вася. Попробую воплотить её в жизнь! Главная процедура будет описывать два действия: копирование записи (через пустую ячейку справа от оригинала) и сдвиг копии влево на одну ячейку для удаления промежутка между оригиналом и копией.

```
ЭТО Дубль
  Копирование
  Сдвиг_копии
КОНЕЦ
```

Петя. Нет возражений.

Вася. При копировании исполнитель будет переносить по одному символу, пока текст не закончится:

```
ЭТО Копирование
  ПОКА НЕ ПУСТО
  {
    ЯЩИК+    // Запомнить символ.
    Перенос  // Скопировать символ на новое место.
```

```
    ВПРАВО   // Переместиться к следующему символу.
  }
КОНЕЦ
```

Перенос разделим на две части, каждую из которых будет выполнять отдельная процедура. Первая процедура Перенос рекурсивно перемещает окно вправо до пустого места и обеспечивает возврат окна к исходному символу в тексте при помощи отложенной команды **ВЛЕВО**:

```
ЭТО Перенос
  ВПРАВО
  ЕСЛИ ПУСТО
    ТО    Установка
    ИНАЧЕ Перенос
  ВЛЕВО
КОНЕЦ
```

Папа. Хорошее рассуждение.

Вася. Вторая процедура Установка будет вызвана тогда, когда в окне видна пустая клетка между текстом и его копией. В этой процедуре рекурсия не нужна: окно перемещается до конца копии и символ копируется на ленту сразу за ней.

Вернуться назад к флагу — пустой клетке можно при помощи цикла **ПОКА**. Возврат окна от флага к исходному символу выполнит отложенная команда **ВЛЕВО** в процедуре Перенос.

```
ЭТО Установка
  ВПРАВО
  ПОКА НЕ ПУСТО ВПРАВО
  ЯЩИК-
  ПОКА НЕ ПУСТО ВЛЕВО
КОНЕЦ
```

Написать процедуру Сдвиг_копии не составляет труда:

```
ЭТО Сдвиг_копии
  Сдвиг_символа
  ПОКА НЕ ПУСТО
  {
    ВПРАВО         // Окно на следующую ячейку
    Сдвиг_символа // Символ справа копируется в текущую ячейку
  }
```

```
КОНЕЦ


ЭТО Сдвиг_символа
  ВПРАВО ЯЩИК+ ВЛЕВО ЯЩИК- // Символ справа копируется в текущую ячейку
КОНЕЦ
```

Папа. Прекрасное решение, достойное Куков!

11.4. Задачи

На рис. 11.18–11.22 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.

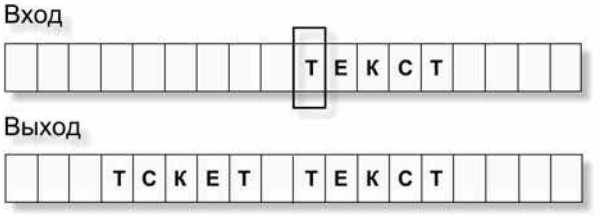


Рис. 11.18

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.

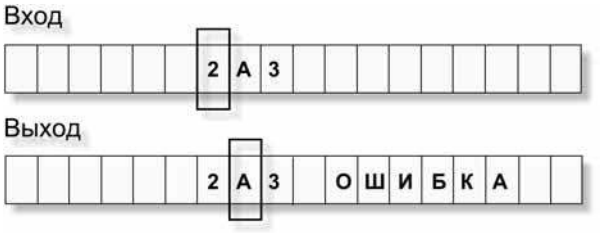


Рис. 11.19

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.

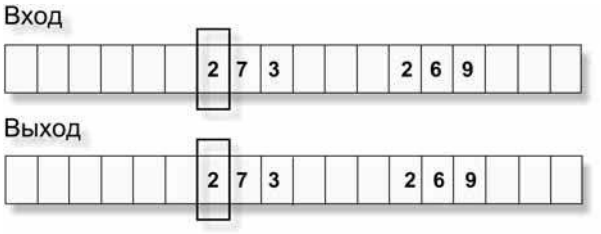


Рис. 11.20

4. В начальный момент в окно виден первый символ текста. Убрать из текста все повторные вхождения этого символа, а оставшуюся запись уплотнить.

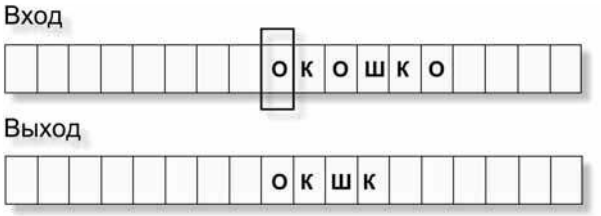


Рис. 11.21

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.

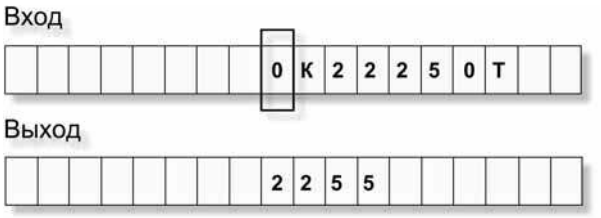


Рис. 11.22

Глава 12

Арифметика чисел, палочек и символов

— Ну, скажи рифму на слово «палка».
— Селёдка, — ответил Незнайка.

Н. Носов

Все Куки: и большие, и маленькие, — горазды на всякие выдумки. Сегодня они решили научить Корректора арифметике (рис. 12.1), обычной и не очень...

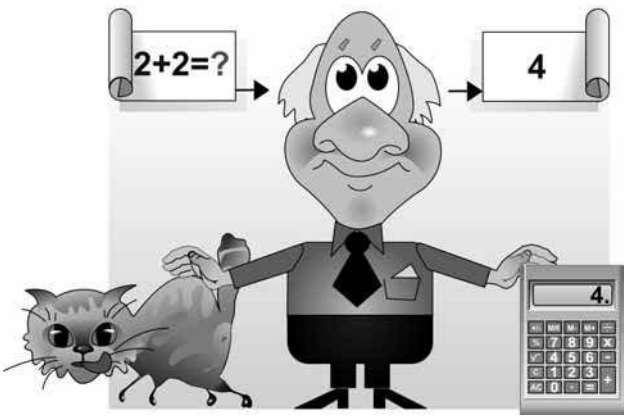


Рис. 12.1. Корректор в школе у Куков

12.1. Арифметика чисел

— Тридцать шесть и шесть у меня. Кажется, всё в порядке.
— Какое там в порядке, — кричит кот. — У вас же температура сорок два!
— Почему? — испугался Печкин.
— А потому что тридцать шесть у вас и ещё шесть. Сколько это вместе будет?
Почтальон посчитал на бумажке. Сорок два вышло.

Э. Успенский

Папа. Давайте научим Корректора выполнять арифметические операции над числами.

Вася. Но в СКИ нет нужных команд!

Папа. Сделаем это при помощи хитрости.

Петя. А какие числа мы будем рассматривать?

Папа. Давайте займёмся арифметикой целых неотрицательных чисел и, чтобы не говорить каждый раз так длинно, будем называть их просто числами. Вот первая задача.

Задача 1

К числу на ленте прибавить единицу. В начальный момент в окно видна младшая цифра числа.

Вася. Такая задача уже была. Решением является единственная команда **плюс**.

Папа. В задаче, которую мы решали, говорилось о сложении единицы с чётным числом, и это было не случайно.

Петя. Проблема возникает, когда число оканчивается на цифру девять, ведь в этом случае команда **плюс** запишет на её место букву А (она идёт следом за девяткой в алфавите исполнителя).

Вася. Да, я поспешил... А что же делать, если младшая цифра числа — девять?

Петя. Вместо неё надо записать нуль, затем прибавить единицу к следующему разряду или, что то же самое, прибавить единицу к исходному числу без последней цифры.

Папа. Посмотрите, решая исходную задачу, мы снова пришли к ней самой, но с меньшим количеством цифр в числе. Как называется такой приём?

Петя. Конечно, это рекурсия!

Вася. Пусть число на ленте равно 239. Вместо девяти записываем нуль и прибавляем 1 к числу 23. Получается 240, правильно!

Папа. А если на ленте число 299?

Вася. Самую младшую девятку заменяем нулём и прибавляем единицу к числу 29. При решении последней задачи девятку в числе 29 заменяем нулём и прибавляем один к числу 2. В итоге получается 300!

Петя. Я придумал красивую схему. Давайте обозначим нашу задачу так:

```
Плюс1(299)
```

Эта запись означает, что нужно прибавить один к числу 299. Тогда решение будет иметь вид:

```
Плюс1(299) = Плюс1(29)0 = Плюс1(2)00 = 300
```

Ведь известно, что `Плюс1(1) = 2`

Папа. Схема мне понравилась: в ней очень хорошо видна рекурсивность. А как решить задачу для числа 99?

Петя. Запишем так:

```
Плюс1(99) = Плюс1(9)0 = Плюс1()00
```

Понятно! Чтобы ответ был верным, нужно предусмотреть добавление единицы к пустому месту, при этом результат должен равняться числу один. Пустое место получается как бы равнозначно нулю:

```
Плюс1() = 1
```

Тогда:

```
Плюс1(99) = Плюс1(9)0 = Плюс1()00 = 100
```

Папа. Что-то я не слышу ничего от Васи!

Вася. С трудом успеваю следить за вашими схемами... Но, кажется, я всё понял. В качестве доказательства записываю программу для Корректора:

```
ЭТО Плюс1
  ЕСЛИ НЕ 9  // Если последняя цифра не 9,
    ТО ПЛЮС  // то задача решается командой ПЛЮС.
    ИНАЧЕ
    {
      ПИШИ 0 // Когда число заканчивается на 9, заменяем девятку нулём
      ВЛЕВО
      Плюс1  // и добавляем 1 к "сокращённому" числу.
    }
КОНЕЦ
```

Петя. В твоей программе упущен случай, когда число состоит из одних девяток.

Вася. Да, действительно. Такое число приводит к необходимости обработать пустую ячейку:

```
ЭТО Плюс1
  ЕСЛИ НЕ 9  // Если последняя цифра не 9,
    ТО       // то ...
    {
      ЕСЛИ ПУСТО    // Если пусто, то
       ТО    ПИШИ 1 // записываем 1,
       ИНАЧЕ ПЛЮС   // иначе записываем
    }               // следующий символ.
    ИНАЧЕ
    {        // Когда число заканчивается на 9,
      ПИШИ 0 // заменяем девятку нулём
      ВЛЕВО  // и добавляем
      Плюс1  // один к числу слева.
    }
КОНЕЦ
```

Петя. Я бы переписал этот код так:

```
ЭТО Плюс1
  ЕСЛИ       ПУСТО ТО ПИШИ 1
  ИНАЧЕ ЕСЛИ 9     ТО { ПИШИ 0 ВЛЕВО Плюс1 }
  ИНАЧЕ               ПЛЮС
КОНЕЦ
```

Папа. Да, последний вариант нагляднее! В нём отчётливо заметен переключатель на три положения:

- ячейка пуста;
- в ячейке цифра 9;
- в ячейке одна из остальных цифр.

Как вы думаете, сколько раз надо запустить эту программу, чтобы убедиться в правильности её работы?

Вася. Думаю, минимум 3 раза. Первый раз для числа, которое не оканчивается на девять, например, для числа 458, второй раз для числа, у которого первая цифра не девять, а последние — девятки, например, для числа 57 999 и третий раз для числа, состоящего из одних девяток, например, для числа 9999. Тем самым мы проверим все ветвления программы.

Петя. Молодец! Тестирование должно быть спланировано по возможности так, чтобы в программе проработали все её части во всех возможных комбинациях. Исходные данные, которые обрабатывает проверяемая программа, называются тестами. Для каждого теста нужно подготовить ожидаемый результат, который будет сравниваться с результатом работы программы. Обычно результаты тестирования оформляют в виде таблицы. Для твоих тестов таблица примет такой вид (табл. 12.1).

Таблица 12.1

Тест	Что проверяется	Ожидаемый результат	Результат по программе
458	Младший разряд не 9	459	
57 999	Число заканчивается девятками	59 000	
9 999	Число состоит из одних девяток	10 000	

Папа. Проверка программы будет более надёжной, если для каждой проверяемой ситуации заготовить не один, а несколько тестов.

Вася. Зачем?

Папа. Посмотри, ты ошибся при ручном подсчёте, и ожидаемый результат для числа 57 999 в таблице записан неверно! Правильная программа выдаст другой ответ, и при проверке выяснится, что ошибка допущена в тестах. Но может получиться так, что из-за ошибки программа выдаст результат, совпадающий с неверным тестом. Тогда программу с ошибкой ты будешь считать правильной.

Вот почему для каждой проверяемой ситуации надо иметь не один, а несколько тестов. Один раз ошибка в программе может случайно привести к такому же неверному ответу, который записан в таблице, но повторение такого совпадения маловероятно.

Вася. Убедительно! Вот моя новая тестовая таблица (табл. 12.2).

Таблица 12.2

Тест	Что проверяется	Ожидаемый результат	Результат по программе
458 14	Младший разряд не 9	459 15	
57 999 1999 999	Число заканчивается девятками	58 000 2000 000	
9999 99	Число состоит из одних девяток	10 000 100	

Вася занялся тестированием программы, и все тесты прошли хорошо: результаты программы совпали с ожидаемыми результатами, посчитанными вручную.

Петя. Тестирование — спасательный круг для программиста. Ведь в программе очень легко допустить ошибку и не заметить этого. Ошибка может затаиться в какой-нибудь очень редко работающей ветви программы, предусмотренной для обработки особого случая.

Представь, что наша программа является элементом системы, управляющей космическим полётом, и ветвь, обрабатывающая одни девятки, содержит ошибку. Вероятность того, что на вход поступит число из одних цифр девять, мала, поэтому программа долгое время будет работать прекрасно, но если всё же такое число появится, это приведёт к отходу аппарата от заданного курса и, возможно, падению его на Солнце.

Вася. Ну, а если программу протестировали, то можно спать спокойно!

Петя. Увы! Программист никогда не спит спокойно. Представь, что обработка числа из одних девяток не предусмотрена в алгоритме. Тогда в программе эта ветвь отсутствует и, конечно, «девяточный» тест программистом заготовлен не будет. Остальные тесты пройдут нормально, ты уснёшь спокойно, а космический корабль упадёт на Солнце.

Вася. Вот это да! А как же проверять программы и избавляться от ошибок?

Папа. Теме тестирования программ был посвящён один из моих репортажей.

Обычно это происходит так. Сначала программу проверяет автор. Когда он приходит к выводу, что исправил последнюю ошибку, передаёт программу эксперту — специалисту по тестированию.

В коллективе, занимающемся разработкой программ, такие специалисты необходимы. Они, подобно музыкальным или литературным критикам, имеют особый склад мышления, направленный на выявление самых незаметных, глубоко спрятанных изъянов и погрешностей.

Когда специалист по тестированию «выдыхается» (все его самые изощрённые тесты проходят нормально), программа поступает в опытную эксплуатацию, где длительное время обрабатывает данные, приближённые к реальным.

Только потом программа передаётся потребителю, и всё равно в ней, как правило, остается некоторое количество ошибок.

Тестирование предназначено для выявления ошибок, но оно не доказывает их отсутствия!

Вася. Вот это да! Всё так безнадёжно?

Папа. Увы, все большие программы именитых фирм, с которыми я постоянно работаю время от времени, работают неправильно или попросту «зависают», то есть перестают реагировать на нажатие клавиш и перемещение мыши.

Вася. Иными словами, как ни тестируй, а космический аппарат всё равно упадёт на Солнце?

Папа. В очень ответственных случаях вычисления выполняются одновременно по нескольким программам, написанным по разным алгоритмам, и в качестве результата принимается тот, который выдали большинство программ.

Теперь ты понимаешь, какая большая ответственность лежит на программистах?

Вася. Я всегда буду очень тщательно проверять свои программы.

Папа. Это похвальное решение!

Продолжим наши арифметические опыты с Корректором. Раз мы научили его прибавлять единицу к числу, давайте научим и отнимать.

Задача 2

От ненулевого числа на ленте отнять единицу. В начальный момент в окно видна младшая цифра числа.

Вася. Для всех чисел, которые не оканчиваются нулём, задача решается командой **минус**. Но вот с последним нулём возникают проблемы…

Петя. Нуль надо заменить девяткой, а от оставшегося числа (без последней цифры) снова отнять один. Опять рекурсия! Я изображу решение на примере числа 2400 с помощью моих схем:

```
Минус1(2400) = Минус1(240)9 = Минус1(24)99 = 2399
```

Когда число не заканчивается нулём, рекурсия заканчивается: `Минус1(24) = 23`.

Папа. Твои схемы нравятся мне всё больше и больше. Есть ли в задаче другие особые случаи, кроме последнего нуля?

Вася. Думаю, небольшие сложности возникнут для чисел, первая цифра которых единица, а остальные нули. Например,

```
Минус1(100) = Минус1(10)9 = Минус1(1)99 = 099
```

Петя. Да, незначащий нуль хотелось бы убрать.

Вася. Вот моя программа:

```
ЭТО Минус1
  Минус1_работа
  Минус1_проверка
КОНЕЦ


ЭТО Минус1_работа
  ЕСЛИ 0
   ТО
```

```
  {
    ПИШИ 9
    ВЛЕВО
    Минус1_работа
  }
  ИНАЧЕ МИНУС
КОНЕЦ
```

По моей задумке процедура `Минус1_проверка` должна убрать незначащий нуль у результата, если он есть. Эта ситуация возникает, когда окно смотрит на нуль, а слева от него цифр нет.

Петя. В одном случае это неправильно.

Вася. В каком?

Петя. Когда исходное число равно 1. В окно виден нуль, но стирать его не надо.

Вася. Понятно. Значит, нуль надо убирать, если слева от него есть цифра, а справа — нет.

Петя. Это другое дело!

Папа. Не забудьте, что в языке есть условие **ЦИФРА**. Сейчас как раз удобно его использовать.

И ещё, если у результата обнаружится незначащий нуль, то давайте будем стирать его сдвигом числа влево на одну позицию (рис. 12.2).

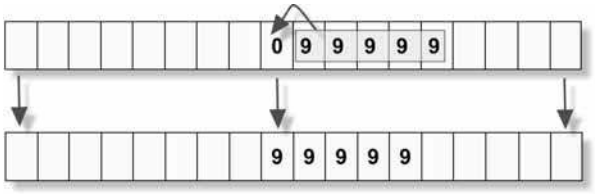


Рис. 12.2

Петя. На самом деле сдвигать число не надо: ведь за незначащим нулём идут одни девятки. Вместо нуля запишем девятку, а последнюю девятку сотрём!

Петя и Вася изрядно потрудились и записали решение, вызвав при этом одобрение папы:

```
ЭТО Минус1
  Минус1_работа
  Минус1_проверка
```

```
КОНЕЦ

ЭТО Минус1_работа
  ЕСЛИ 0
   ТО { ПИШИ 9 ВЛЕВО Минус1_работа }
   ИНАЧЕ МИНУС
КОНЕЦ

ЭТО Минус1_проверка
  ЕСЛИ 0
    ТО
    {
      ВЛЕВО
      ЕСЛИ НЕ ЦИФРА
        ТО
        {
          ВПРАВО ВПРАВО
          ЕСЛИ 9
            ТО Минус1_сдвиг
            ИНАЧЕ ВЛЕВО
        }
    }
КОНЕЦ

ЭТО Минус1_сдвиг
  // Вместо нуля запишем 9
  ВЛЕВО ПИШИ 9
  // Сотрём последнюю девятку
  ПОКА 9 ВПРАВО
  ВЛЕВО ПИШИ ПУСТО
  // Сместим окно на младшую цифру
  ВЛЕВО
КОНЕЦ
```

Задача 3

На ленте записан пример на сложение двух чисел. Вычислить результат. В начальный момент окно установлено на знак сложения. На рис. 12.3 изображена обработка возможного примера.

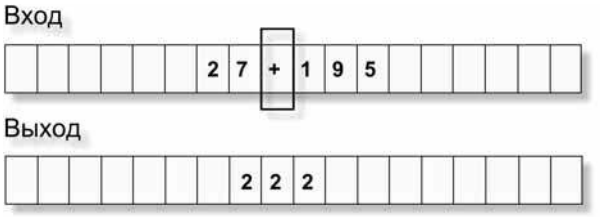


Рис. 12.3

Петя. Предлагаю такой алгоритм: к первому числу добавлять по единице, а от второго отнимать, пока второе число не станет нулём. Тогда на месте первого числа получится результат сложения.

Папа. Принимается!

Петя многозначительно посмотрел на Васю.

Вася. Вы, как всегда, предлагаете мудрёные алгоритмы, а я, как Золушка, должен трудиться не покладая рук. И добрые мыши не придут ко мне на помощь!

Петя. Не ворчи, братишка! Ты же хочешь научиться писать программы, а для этого надо написать их очень много.

Вася. Вот моё решение:

```
ЭТО Сложение
  ВПРАВО
  ПОКА НЕ 0
  {
    // Установим окно на конец второго числа.
    ПОКА НЕ ПУСТО ВПРАВО
    ВЛЕВО
    // Отнимем 1 от второго числа.
    Минус1
    // Установим окно на первое число.
    ПОКА НЕ + ВЛЕВО
    ВЛЕВО
    // Прибавим 1 к первому числу.
```

```
    Плюс1
    // Установим окно на начало второго числа.
    ПОКА НЕ + ВПРАВО
    ВПРАВО
  }
  // Оформим результат
  ПИШИ ПУСТО
  ВЛЕВО
  ПИШИ ПУСТО
КОНЕЦ
```

Рекурсия или не рекурсия?

Петя. Посмотрите! Я написал нерекурсивный вариант программы Плюс1:

```
ЭТО Плюс1
  ПОКА 9                // Пока в окошке видны девятки,
  {                     // заменяем их нулями, перемещая окно
    ПИШИ 0              // влево.
    ВЛЕВО
  }                     // Девятки закончились.
  ЕСЛИ НЕ ЦИФРА         // Если все цифры были девятками, то
    ТО    ПИШИ 1        // перед всеми нулями изменённого числа
    ИНАЧЕ ПЛЮС          // записываем 1, в противном случае
КОНЕЦ                   // увеличиваем цифру-не-девять на 1.
```

Папа. Это хорошая программа! Рекурсия красива, но расходует много памяти компьютера. Когда придуман короткий нерекурсивный алгоритм, ему надо отдавать предпочтение.

Вася. Ну тогда я предлагаю нерекурсивный вариант программы Минус1:

```
ЭТО Минус1
  ЕСЛИ 0
    ТО
    {
      ПОКА 0 { ПИШИ 9 ВЛЕВО }
      ЕСЛИ 1
        ТО    ПИШИ ПУСТО
        ИНАЧЕ МИНУС
    }
    ИНАЧЕ МИНУС
КОНЕЦ
```

Вася. Внешняя развилка помогает правильно обработать исходное число 1, записав в ответе 0, а не **ПУСТО**. Правда, в этом варианте нет сдвига числа влево для случая, когда его разрядность уменьшается, но это тоже легко построить:

```
ЭТО Минус1
  ЕСЛИ 0
    ТО
    {
      ПОКА 0 { ПИШИ 9 ВЛЕВО }
      ЕСЛИ 1
        ТО    Минус1_сдвиг
        ИНАЧЕ МИНУС
    }
    ИНАЧЕ МИНУС
КОНЕЦ

ЭТО Минус1_сдвиг
  ПИШИ 9
  ПОКА ЦИФРА ВПРАВО
  ВЛЕВО ПИШИ ПУСТО ВЛЕВО
КОНЕЦ
```

Задачи

На рис. 12.4–12.9 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 «–».

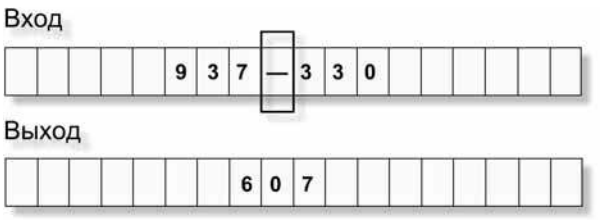


Рис. 12.4

2. На ленте записан пример вида: «число*2». Вычислить результат. В начальный момент окно установлено на знак умножения.

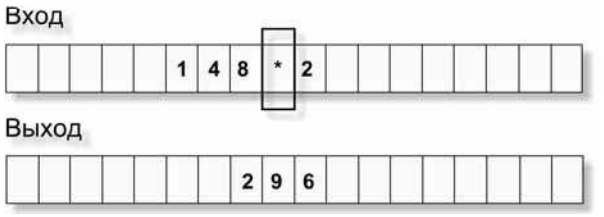


Рис. 12.5

3. На ленте записан пример вида: «число*4». Вычислить результат. В начальный момент окно установлено на знак умножения.

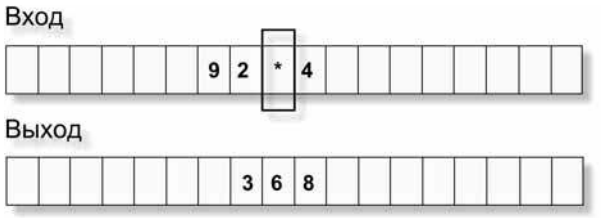


Рис. 12.6

4. На ленте записан пример вида: «число*16». Вычислить результат. В начальный момент окно установлено на знак умножения.

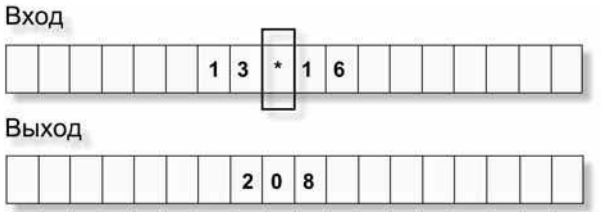


Рис. 12.7

5. На ленте записан пример вида: «число*3». Вычислить результат. В начальный момент окно установлено на знак умножения.

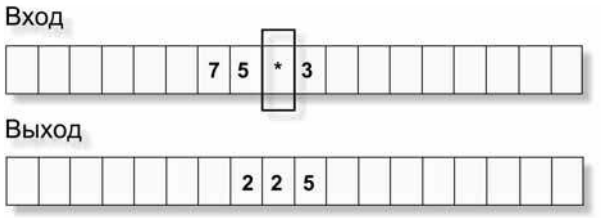


Рис. 12.8

6. На ленте записан пример вида: «число*6». Вычислить результат. В начальный момент окно установлено на знак умножения.

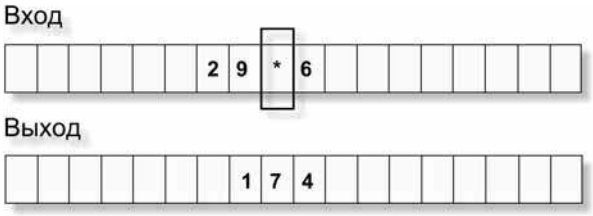


Рис. 12.9

12.2. Арифметика палочек

> Тогда дядя Фёдор говорит:
> — Пусть каждый из вас возьмёт палку и кинет её через забор. Чью палку он принесёт, тот для Гаврюши и главнее.
>
> *Э. Успенский*

Папа. Мы научили Корректора кое-каким арифметическим действиям над неотрицательными целыми числами. У меня есть предложение: давайте теперь научим Корректора арифметике палочек!

Петя. Арифметика палочек? Это что-то новое!

Папа. Будем считать значением «палочного» числа количество изображающих его палочек. Сами палочки будем записывать символом «|» (он входит в алфавит исполнителя). Вот как, например, на ленте выглядит число 7 (рис. 12.10).

Рис. 12.10

Действия над палочками понимаются как действия над числами, которые они изображают.

Задача 4

На ленте изображено два палочных числа, соединенных знаком сложения. Получить результат, если в начальный момент окно установлено на знак «+». Вариант начальной установки и результат работы программы показаны на рис. 12.11.

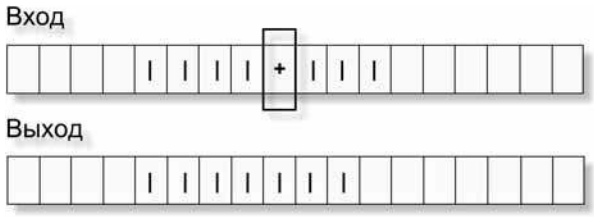


Рис. 12.11

Петя. Мне кажется, что это довольно простая задача: на место знака сложения надо записать палочку, а потом...

Вася. А потом переместить окно на последнюю палочку и стереть её. Вот моё решение:

```
ЭТО Сложение_палочек
  ПИШИ |
  ПОКА | ВПРАВО
  ВЛЕВО
  ПИШИ ПУСТО
КОНЕЦ
```

Петя. Сейчас я подумал о том, что мы не договорились, как будем изображать нуль в палочной арифметике.

Папа. Давайте изображать нуль символом **ПУСТО**.

Петя. Если так, то Васина программа будет правильно работать и тогда, когда одно из палочных чисел или даже оба равны нулю (рис. 12.12).

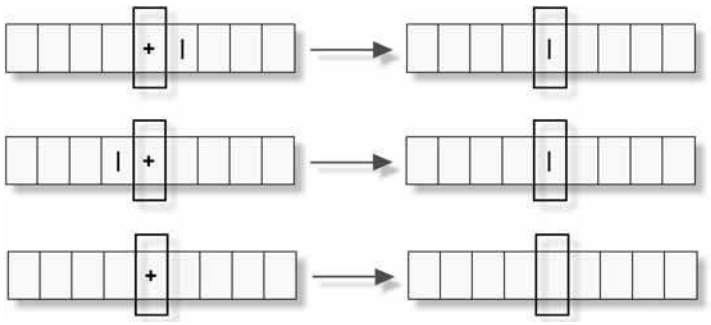


Рис. 12.12

Вася. Ну, надо же! Честно говоря, я про нулевые числа и не подумал. Это получилось случайно.

Папа. Следующая задача сложнее.

Задача 5

На ленте изображены два палочных числа, соединённых знаком вычитания. Найти модуль разности этих двух чисел (т. е. от большего из этих двух чисел нужно отнять меньшее). В начальный момент окно установлено на знак «–». Вариант начальной установки и результаты работы программы показаны на рис. 12.13.

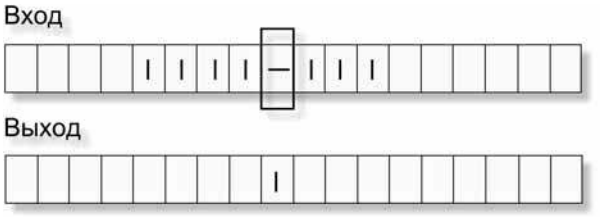


Рис. 12.13

Вася. Можно поступить так же, как это делалось при программировании вычитания обычных чисел: из обоих чисел вычитать по единице до тех пор, пока одно из них не станет нулём.

Петя. А как отнимать от палочного числа единицу?

Вася. Ну, это очень просто: нужно в записи числа стереть одну крайнюю палочку.

Петя. Прежде чем стирать пару палочек (одну у первого, а другую у второго числа), не забудь проверить, можно ли это сделать. Вдруг одно из чисел уже не содержит палочек, то есть уже стало нулём.

Папа. Или было нулём с самого начала.

Вася. Хорошо. Вот главная процедура:

```
ЭТО Вычитание_палочек
  // Можно ли стереть по одной палочке у обоих чисел?
  Проверка
  // Результат проверки: если в ящике знак "-", стирать палочки не надо.
  ОБМЕН
  ЕСЛИ -
   ТО ПИШИ ПУСТО
   ИНАЧЕ
   {
     ОБМЕН
     Стереть_две_палочки
```

```
    Вычитание_палочек
  }
КОНЕЦ
```

Сначала я выполняю проверку, а затем, если числа не нулевые, стираю по одной палочке у каждого числа процедурой `Стереть_две_палочки`, после чего рекурсивно возвращаюсь на начало главной процедуры.

Результат проверки я буду помещать в ящик. Знак «–» в нём означает, что одно из палочных чисел равно нулю, а знак «|» в ящике показывает, что числа не нулевые.

```
ЭТО Проверка
  // Сначала в ящик запишем знак "-", он будет обозначать, что стирать
  // палочки нельзя.
  ЯЩИК+
  // Проверим, есть ли палочка слева
  ПОКА НЕ ПУСТО ВЛЕВО
  ВПРАВО
  ЕСЛИ НЕ -
    ТО
    {
      // Проверим, есть ли палочка справа
      ПОКА НЕ ПУСТО ВПРАВО
      ВЛЕВО
      ЕСЛИ НЕ -
        ТО
        {
          // Палочки есть и слева и справа.
          // Запишем в ящик знак "|", он будет обозначать, что
          // стирать палочки можно.
          ЯЩИК+
          // Установим окно на знак "-"
          ПОКА НЕ - ВЛЕВО
        }
    }
КОНЕЦ

ЭТО Стереть_две_палочки
  // Стираем палочку слева
```

```
  ПОКА НЕ ПУСТО ВЛЕВО
  ВПРАВО
  ПИШИ ПУСТО
  ВПРАВО
  // Стираем палочку справа
  ПОКА НЕ ПУСТО ВПРАВО
  ВЛЕВО
  ПИШИ ПУСТО
  // Установим окно на знак вычитания
  ПОКА НЕ - ВЛЕВО
КОНЕЦ
```

Папа. Код получился довольно громоздким и малоэффективным: исполнителю приходится ходить по записи два раза: один раз для проверки чисел на 0, другой раз для стирания палочек. Хотелось бы совместить проверку с удалением палочек.

Петя. Предлагаю такой алгоритм.

Алгоритм работы процедуры `Вычитание_палочек`

1. Идём влево. Если запись заканчивается палочкой, стираем её, иначе стираем знак минус, и заканчиваем работу (ответ готов).
2. Идём вправо. Если запись заканчивается палочкой, стираем её и выполняем (рекурсивно) новый экземпляр процедуры `Вычитание_палочек`. Иначе заменяем знак минус на палочку, напрасно стёртую слева, и заканчиваем работу (ответ готов).

А вот код, записанный по этому алгоритму:

```
ЭТО Вычитание_палочек
  На_левый_конец
  ЕСЛИ -
    ТО     ПИШИ ПУСТО
    ИНАЧЕ
    {
      ПИШИ ПУСТО
      На_правый_конец
      ЕСЛИ -
        ТО    ПИШИ |
        ИНАЧЕ { ПИШИ ПУСТО Вычитание_палочек }
    }
КОНЕЦ
```

```
ЭТО На_левый_конец
  ВЛЕВО
  ПОКА НЕ ПУСТО ВЛЕВО
  ВПРАВО
КОНЕЦ

ЭТО На_правый_конец
  ВПРАВО
  ПОКА НЕ ПУСТО ВПРАВО
  ВЛЕВО
КОНЕЦ
```

Папа. Неплохое решение, но хорошо бы обойтись без рекурсивного размножения экземпляров процедуры `Вычитание_палочек`.

Петя. Заменяем рекурсию циклом **ПОКА**.

Алгоритм работы процедуры `Вычитание_палочек`

1. Идти на левый конец записи.
2. Пока в окне не знак минус, делать:
   2.1. Стереть палочку.
   2.2. Идти на правый конец записи.
   2.3. Если не знак минус, то:
      а) Стереть палочку.
      б) Идти на левый конец записи.
      Иначе:
      а) Заменить знак минус на палочку (напрасно стёртую слева).
      б) Описать справа знак минус для прекращения цикла.
3. На место знака минус записать **ПУСТО**.

А вот код, записанный по этому алгоритму:

```
ЭТО Вычитание_палочек
  На_левый_конец
  ПОКА НЕ -
  {
    ПИШИ ПУСТО
    На_правый_конец
    ЕСЛИ НЕ -
      ТО    { ПИШИ ПУСТО На_левый_конец }
```

```
      ИНАЧЕ { ПИШИ | ВПРАВО ПИШИ - }
  }
  ПИШИ ПУСТО
КОНЕЦ
```

Задачи

На рис. 12.14–12.17 приводятся примеры начальных состояний и результаты обработки.

1. Умножить палочное число на 2. В начальный момент окно установлено на первую слева палочку.

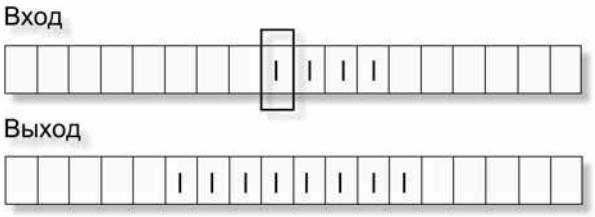


Рис. 12.14

2. Умножить палочное число на 3. В начальный момент окно установлено на первую слева палочку.

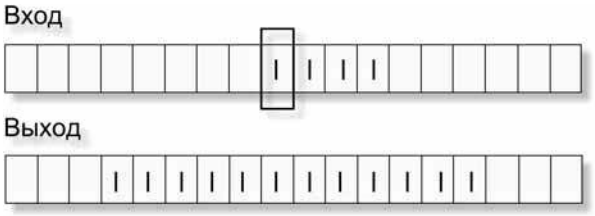


Рис. 12.15

3. Умножить палочное число на 6. В начальный момент окно установлено на первую слева палочку.

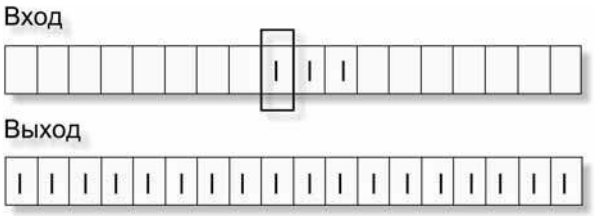


Рис. 12.16

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.

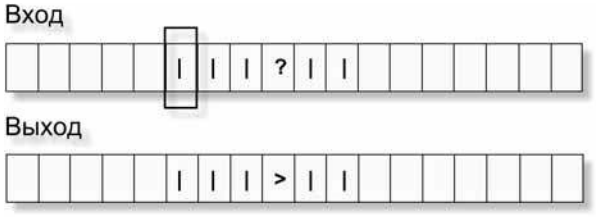


Рис. 12.17

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.
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.
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.

12.3. Арифметика символов

> Кот Матроскин очень хорошо сосиски складывал. Или котлеты. Или сосиски с котлетами. А вот сложить хлеб с колбасой он никак не мог. У него всё бутерброды получались.
>
> *Э. Успенский*

Папа. Для выполнения вычислений в среде Корректора в качестве чисел удобно использовать символы алфавита исполнителя, начиная с нуля.

Вася. Символы алфавита в качестве чисел?

Папа. Ну да. Просто каждому символу ставится в соответствие его порядковый номер в алфавите. Нумерацию удобно начинать с символа «0», присвоив ему номер 0 (табл. 12.3).

Таблица 12.3

Символ	Соответствующее число
0	0
1	1
2	2
...	...
9	9
А	10
Б	11
В	12
...	...

Будем называть символы алфавита Корректора, когда они используются для вычислений, **символьными числами**.

Значением символьного числа будем считать номер соответствующего символа в алфавите Корректора по отношению к символу «0» (нуль), номер которого полагается равным числу 0.

Под арифметическими действиями над символьными числами будем понимать соответствующие действия над их алфавитными номерами.

Примеры:

```
А / 2 = 5
В - А = 2
7 · 2 = Д
```

Петя. Получается, что мы будем работать с цифрами системы счисления по основанию N, если число (N − 1) соответствует последнему символу в алфавите Корректора.

Папа. Да, это так. Если бы алфавит Корректора заканчивался символом «9», мы бы работали с цифрами обычной десятичной системы счисления (N = 10).

Петя. А если бы алфавит Корректора содержал только 16 символов (не считая `ПУСТО`), то символы эти можно было бы рассматривать как цифры 16-ричной системы счисления (N = 16).

Вася. Напомните мне, что такое система счисления?

Петя. Система счисления — это способ записи чисел. Привычная для нас десятичная система счисления является позиционной: в ней вклад каждой цифры однозначно определяется её позицией в записи числа. Цифра

на первом месте справа означает количество единиц, на втором — количество десятков, на третьем — количество сотен и так далее. Например, запись 8735 означает:

$8735 = 8 \cdot 10^3 + 7 \cdot 10^2 + 3 \cdot 10^1 + 5 \cdot 10^0$

Та же самая запись в 16-ричной системе счисления означает:

$8735 = 8 \cdot 16^3 + 7 \cdot 16^2 + 3 \cdot 16^1 + 5 \cdot 16^0$

Получается, что число 8735 в 16-ричной системе счисления равно числу 34 613 в десятичной системе счисления. Это записывают так:

$8735_{16} = 34\ 613_{10}$

Вася. Существуют ли непозиционные системы счисления?

Петя. Да, конечно. С одной такой системой мы работали сегодня — это счёт палочками. Другой пример — римская система счисления.

Папа. Давайте вернёмся к символьным числам Корректора.

Работать с символьными числами очень удобно, ведь команды `плюс` и `минус` увеличивают и уменьшают их значение на 1 (без специальных процедур, как для обычных чисел), а проверки =, #, <, > также выполняются корректно (они относятся к порядковым номерам символов в алфавите Корректора, а значит, к значениям символьных чисел).

Перевод символьного числа в обычное число

Пусть x — целое неотрицательное число, меньшее 100. Тогда число x можно представить в виде:

$X = a_1 \cdot 10 + a_0$

Здесь:

- a_1 — старшая цифра числа x,
- a_0 — младшая цифра числа x.

Например:

$35 = 3 \cdot 10 + 5$

$7 = 0 \cdot 10 + 7$

Из этого представления следует, что:

- остаток от деления x на 10 равен младшей цифре числа x;
- целая часть от деления x на 10 равна старшей цифре.

Будем использовать эти рассуждения при построении алгоритма перевода символьного числа в обычное число, предполагая, что значение символьного числа не превышает 100.

Алгоритм перевода символьного числа в обычное число

Пусть x — исходное символьное число. Деление заменим вычитанием. Число вычитаний (целая часть от деления — старшая цифра исходного числа) будем подсчитывать в ячейке t (изначально пустой), а значение x после всех вычитаний будет изображать остаток от деления (младшая цифра исходного числа).

Алгоритм

1. Пока x не цифра, делать:
   1.1. Вычитание. Отнять от числа x число 10.
   1.2. Подсчёт целой части от деления. Если t — пусто, записать в него 1, иначе добавить 1.

Заметим, что если x изначально меньше 10, тело цикла не будет работать ни разу, то есть символьные числа-цифры меняться не будут.

Программа

```
// 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
```

Перевод обычного числа в символьное число

Если исходное число — цифра, то оно уже является символьным числом и перевод не требуется. Пусть исходное число записано в виде двух цифр a_1a_0, тогда соответствующее символьное число $x = a_1 \cdot 10 + a_0$.

Описание алгоритма. Добавляем к младшей цифре a_0 количество десяток, равное значению старшей цифры a_1.

Программа

```
// 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
    {
      // Добавляем к младшей цифре старшую, умноженную на 10.
      ПОКА НЕ 0
      {
        МИНУС
        ВПРАВО
        ПОВТОРИ 10 ПЛЮС
        ВЛЕВО
      }
      // Стираем старшую цифру.
      ПИШИ ПУСТО
    }
  // Устанавливаем окно на результат.
  ВПРАВО
КОНЕЦ
```

Пример использования символьных чисел

Задача 6

Окно установлено на первую цифру целого неотрицательного числа. Найти сумму цифр этого числа, если известно, что она лежит в диапазоне символьных чисел Корректора.

Программа

```
ЭТО Сумма_цифр
  // Обнуление ящика-сумматора
  ОБМЕН ПИШИ 0 ОБМЕН
  // Проход по цифрам числа и суммирование (в ящике).
  // Цифры исходного числа с ленты стираются.
  ПОКА ЦИФРА
  {
    // Добавление цифры к символьному числу в ящике
    ПОКА НЕ 0 { МИНУС ОБМЕН ПЛЮС ОБМЕН }
    // Стирание цифры числа (она стала теперь нулём)
    ПИШИ ПУСТО
    // Смещение окна к следующей цифре
    ВПРАВО
  }
  // Запись результата на ленту
  ОБМЕН
  // Перевод символьного числа в обычное число
  Символ_в_число
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
```

Задачи

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. 12.18.

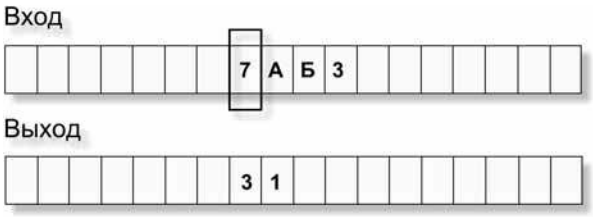


Рис. 12.18

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. 12.19.

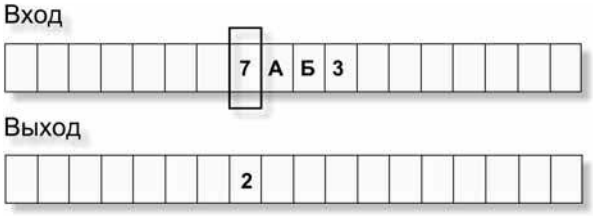


Рис. 12.19

3. Шифровка. Каждый символ исходного сообщения закодирован двумя символами с использованием следующей кодировочной табл. 12.4.

Таблица 12.4

Символ	Код	Символ	Код
0	00	А	10
1	01	Б	11
2	02	В	12
...	...	...	...
9	09		

Расшифровать сообщение. В начальный момент окно установлено на первый символ записи. На рис. 12.20 показан пример начального и конечного состояний среды исполнителя.

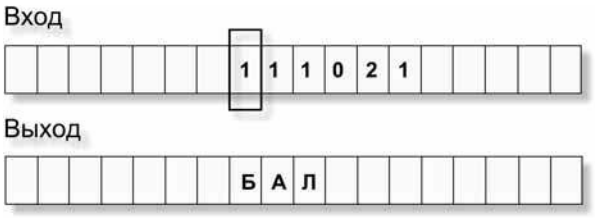


Рис. 12.20

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. 12.21.

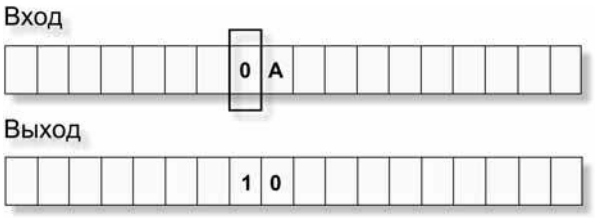


Рис. 12.21

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. 12.22.

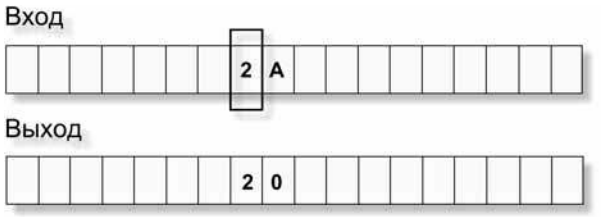


Рис. 12.22

Глава 13

Преобразования, подсчёты, редактирование

13.1. Длина текста

— Я могу картошку окучивать задними лапами. И посуду мыть — языком облизывать. И места мне не надо, я могу на улице спать.

Э. Успенский

Петя. В своих программах, которые я пишу в университете, мне часто требуется вычислять длину текста. Как ты думаешь, можно ли научить этому Корректора?

Вася. После того, как мы научили его азам арифметики, думаю, нет предела его возможностям (рис. 13.1)!

Папа. Интересная задача! Давайте сначала сформулируем её для случая, когда длина текста не превышает девяти символов.

Рис. 13.1. Корректор измеряет длину текста

Задача 1

На ленте записан текст, в котором не более 9 символов. Записать на ленту его длину. В начальный момент окно установлено на первый пустой символ перед текстом (рис. 13.2).

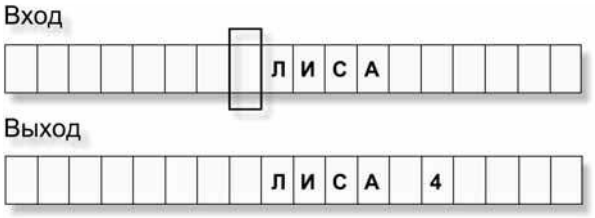


Рис. 13.2

Петя. Кажется, догадываюсь, почему ограничена длина: если текст небольшой, то для увеличения счётчика символов на один можно использовать команду **плюс**.

Папа. Ты прав. Именно это я имел в виду.

Вася. Что вы называете счётчиком?

Петя. Обычно подсчёты программируют так. Заводят специальную ячейку памяти — её-то и называют счётчиком — и записывают в неё нуль. Затем устраивают цикл для просмотра всех символов текста, добавляя в нём к счётчику единицу на каждом символе. Понятно, что когда цикл заканчивается, в счётчике окажется нужное значение.

Вася. Понятно. Предлагаю в качестве счётчика использовать ящик.

Петя. Почему бы и нет!

Вася. Тогда основная процедура запишется так:

```
ЭТО Длина
  Обнуление_счётчика
  Работа
  Запись_ответа
КОНЕЦ
```

Папа. Как вы запишете в ящик нуль?

Вася. Ну, это очень просто. В начальный момент окно установлено на пустой символ перед текстом. Записываю на ленту нуль и копирую его в ящик, затем восстанавливаю на ленте пустое место:

```
ЭТО Обнуление_счётчика
  ПИШИ 0 ЯЩИК+ ПИШИ ПУСТО
КОНЕЦ
```

Петя. Для обнуления счетчика гораздо лучше использовать команду **ОБМЕН**:

```
ЭТО Обнуление_счётчика
  ОБМЕН ПИШИ 0 ОБМЕН
КОНЕЦ
```

Последняя процедура запишет в ящик символ 0 и не испортит содержимое ленты при любом положении окна.

Папа. Хорошо. Теперь надо описать цикл просмотра всех символов текста.

Вася. Сначала командой **ВПРАВО** перемещаю окно на первый символ, а затем открываю цикл **ПОКА**:

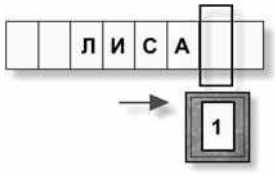
```
ЭТО Работа
  ВПРАВО
  ПОКА НЕ ПУСТО
  {
    Добавить_к_счётчику_1
    ВПРАВО
  }
КОНЕЦ
```

Петя. Теперь мы подошли к самой сложной процедуре, увеличивающей счётчик на один. Так как текст не длиннее 9 символов, то можно было бы использовать команду **ПЛЮС**, но, к сожалению, её нельзя применить к ящику (рис. 13.3).

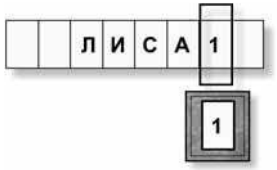
Рис. 13.3

Вася. Да, ты прав. Придется:

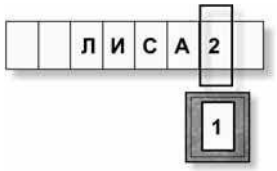
1. Идти в конец текста.



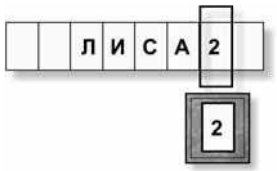
2. Копировать ящик на ленту.



3. Увеличивать цифру на ленте на один.

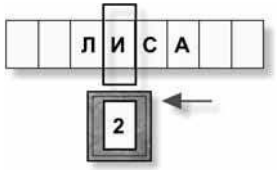


4. Копировать полученное число в ящик.



5. Стирать число на ленте.

6. Возвращать окно на прежнее место в тексте.



Папа. Корректору приходится постоянно «бегать» на конец текста, чтобы увеличивать значение ящика-счётчика на один. Предлагаю изменять счётчик прямо на месте при помощи команды **ОБМЕН**.

Вася. Ну, конечно, как я сразу не догадался! В самом деле, можно обменять местами содержимое ящика и ленты, добавить один к числу на ленте, а затем восстановить ленту новым обменом:

```
ЭТО Добавить_к_счётчику_1
  ОБМЕН ПЛЮС ОБМЕН
КОНЕЦ
```

Петя. Теперь осталось только написать процедуру для оформления ответа.

Вася. Когда цикл закончен, окно смотрит на первый пустой символ справа от текста. В следующую ячейку я запишу число из ящика:

```
ЭТО Запись_ответа
```

```
  ВПРАВО ЯЩИК-
КОНЕЦ
```

Папа. Предлагаю теперь изменить немного формулировку нашей задачи.

Задача 2

На ленте записан текст, число символов в котором меньше числа символов в алфавите Корректора. Записать на ленту его длину. В начальный момент окно установлено на первый пустой символ перед текстом.

Вася. Теперь на ленте могут быть более длинные тексты.

Петя. Новое ограничение на длину, я думаю, по-прежнему связано с возможностями команды **плюс**.

Папа. Ты прав!

Вася. Если действовать прежним образом, то на десятом символе в ящике получится буква А, потом буква Б и так далее.

Петя. Понятно! После просмотра всего текста, можно воспользоваться готовой процедурой Символ_в_число (см. *главу 12*) и преобразовать полученный символ в обычное десятичное число!

Братья после небольших споров и раздумий записали такую программу:

```
ЭТО Большая_длина
  Обнуление_счётчика
  Работа
  Запись_ответа
КОНЕЦ

ЭТО Обнуление_счётчика
  ОБМЕН ПИШИ 0 ОБМЕН
КОНЕЦ

ЭТО Работа
  ВПРАВО
  ПОКА НЕ ПУСТО
  {
    Добавить_к_счётчику_1
    ВПРАВО
  }
КОНЕЦ
```

```
ЭТО Добавить_к_счётчику_1
  ОБМЕН ПЛЮС ОБМЕН
КОНЕЦ


ЭТО Запись_ответа
  ВПРАВО ВПРАВО ЯЩИК-
  Символ_в_число
КОНЕЦ
```

Папа. Теперь можно попробовать решить задачу для любой длины текста.

Задача 3

На ленте записан произвольный текст. Вычислить его длину. В начальный момент окно установлено на первый пустой символ перед текстом.

Петя. Длина текста не может быть совсем произвольной. Ведь лента не бесконечна и на ней, кроме самого текста, должен поместиться ещё и ответ.

Папа. Да, ты прав! Будем считать, что это условие выполнено.

Вася. Кажется, знаю, как решить эту задачу! Можно отказаться от счётчика-ящика, а количество символов «запоминать» при помощи отложенной команды `Добавить_1` в рекурсивном просмотре текста:

```
ЭТО Любая_длина
  ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО     { Любая_длина Добавить_1 }
    ИНАЧЕ  Подготовка_к_записи_ответа
КОНЕЦ
```

Ответ надо записывать не справа, а слева от текста, т. к. разрядность результата заранее не известна. Процедура `Подготовка_к_записи_ответа` установит окно через пустую ячейку слева от текста и запишет на ленту начальное значение 0. Затем рекурсивная пружинка добавит к этому нулю столько единиц, сколько символов было в исходном тексте.

```
ЭТО Подготовка_к_записи_ответа
  // Установим окно через пустую ячейку слева от текста
  ВЛЕВО
  ПОКА НЕ ПУСТО ВЛЕВО
  ВЛЕВО
  // Запишем на ленту начальное значение результата
  ПИШИ 0
```

```
КОНЕЦ


ЭТО Добавить_1
  // Установим окно на младшую цифру текущего значения результата
  ПОКА НЕ ПУСТО ВПРАВО
  ВЛЕВО
  // Увеличим значение результата на 1
  Плюс1 // Процедура, написанная в главе 12
КОНЕЦ
```

Задачи

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. 13.4).

Рис. 13.4

2. На ленте записано число. Подсчитать сумму наименьшей и наибольшей его цифр. В начальный момент окно установлено на первый пустой символ перед записью (рис. 13.5).

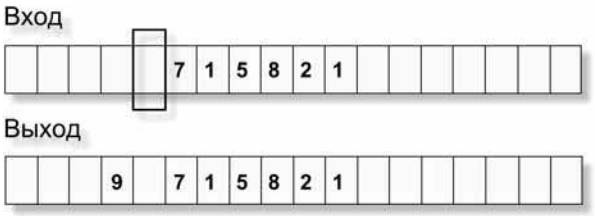


Рис. 13.5

3. На ленте записан текст. Записать на ленту число слов в нём, если известно, что их количество меньше числа символов в алфавите Корректо-

ра, а начальные и конечные пробелы в тексте отсутствуют. В начальный момент окно установлено на первый символ текста (рис. 13.6).

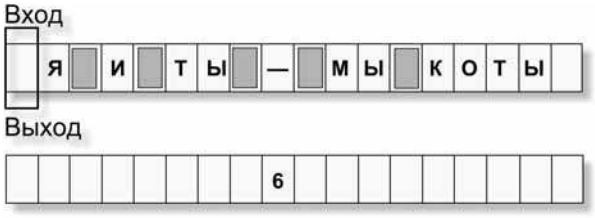


Рис. 13.6

13.2. Корректор оправдывает своё имя

«До свиданья. Ваш сын — дядя Фарик».

Э. Успенский

Петя. У меня есть предложение заняться редактированием текста, не зря же у Корректора такое имя!

Папа. Хорошая идея. Сначала давайте вспомним разделение ошибок набора текста на три класса.

Вася. Я помню! При вводе текста с клавиатуры можно сделать кучу ошибок, но большинство из них укладывается на три полочки:

1. Неверный символ — исправляется заменой.
2. Лишний символ — исправляется удалением.
3. Пропущенный символ — исправляется вставкой.

Папа. Начнём работать по порядку.

Задача 4

На ленте записан текст, в котором все вхождения некоторого символа нужно заменить другим символом. В начальный момент среда имеет вид, показанный на рис. 13.7.

Здесь через t обозначен символ, подлежащий замене, через z — символ, на который надо менять символ t.

Рис. 13.7

Например, конкретная начальная установка может иметь вид, показанный на рис. 13.8.

Рис. 13.8

Это означает, что в тексте КАЛЕСА нужно все буквы А заменить на букву О.

Вася. Предлагаю такой алгоритм. Сначала в ящик копируется символ t, подлежащий замене, а потом в цикле каждый символ текста сравнивается с содержимым ящика. Если совпадение есть, производится замена символа на ленте символом z, расположенным перед текстом:

```
ЭТО Замена
  // Подготовка
  ЯЩИК+ ПОВТОРИ 3 ВПРАВО
  ПОКА НЕ ПУСТО
  {
    ЕСЛИ Я=Л ТО Заменить_t_на_z
    ВПРАВО
  }
КОНЕЦ
```

Папа. Пока нет никаких возражений.

Вася. В процедуре замены есть одно неприятное место. Когда в тексте обнаружен символ, подлежащий замене, нужно:

1. Сходить за образцом z и заменить им символ на ленте.
2. Сходить за образцом t, чтобы продолжить поиск.

Петя. Думаю, что можно обойтись только одним походом.

Вася. Верно! Ведь образец t мы обнаружили на ленте, и ходить за ним никуда не надо! Вот окончание моей программы:

```
ЭТО Заменить_t_на_z
  Взять_образец_z
  // На ленту запишем z, а в ящик поместим t
  ОБМЕН
КОНЕЦ


ЭТО Взять_образец_z
```

```
  ВЛЕВО
  ЕСЛИ НЕ ПУСТО
    ТО Взять_образец_z
    ИНАЧЕ
    {
      ВЛЕВО
      ЯЩИК+
      ВПРАВО
    }
  ВПРАВО // Отложенная команда.
         // Она возвращает окно на текущее место в тексте.
КОНЕЦ
```

Папа. Теперь давайте решим задачу на удаление.

Задача 5

На ленте расположен текст. Перед ним через пустое место записан образец символа, все вхождения которого в текст подлежат удалению. В начальный момент окно установлено на символ-образец (рис. 13.9).

Рис. 13.9

Вася. Сначала копируем образец в ящик, а затем в цикле просматриваем все символы текста:

```
ЭТО Удаление
  // Подготовка
  ЯЩИК+ ВПРАВО ВПРАВО
  ПОКА НЕ ПУСТО
  {
    ЕСЛИ Я=Л ТО Удаление_символа
    ВПРАВО
  }
КОНЕЦ
```

Петя. Как ты собираешься удалять символ?

Вася. Запишу на его место ПУСТО.

Петя. Не годится! При удалении остаток текста надо сдвигать влево.

Вася. Да, верно! Это я поторопился.

Петя. И ещё, представь себе, что удаляемый символ в тексте идёт подряд несколько раз — тогда твоя программа будет работать неправильно.

Вася. Почему?

Петя. Когда условная команда, расположенная первой в цикле **ПОКА**, обнаруживает подлежащий удалению символ, происходит сдвиг текста, после чего окошко перемещается вправо. А если символ повторяется?

Вася. Тогда он копируется на место удалённого и исключается из проверки. Я изобразил эту неприятность на рис. 13.10.

Рис. 13.10

Да, но как же преодолеть эту трудность?

Петя. Очень просто. Вправо надо смещаться только тогда, когда не было удаления. А если удаление было, надо текущий символ проверять снова!

Вася. Понятно. Тогда начало моей программы будет таким:

```
ЭТО Удаление
  // Подготовка
  ЯЩИК+ ВПРАВО ВПРАВО
  ПОКА НЕ ПУСТО
  {
```

```
    ЕСЛИ Я=Л
     ТО    Удаление_символа
     ИНАЧЕ ВПРАВО
  }
КОНЕЦ
```

Петя. Теперь другое дело!

Вася. Работа по удалению символа разделяется на две части. Сначала должен быть сдвиг текста, но он использует ящик. Значит, после сдвига придётся сходить в начало текста за образцом:

```
ЭТО Удаление_символа
  Сдвиг_текста
  Копирование_образца
КОНЕЦ
```

Процедуры сдвига текста и копирования образца будут рекурсивными с отложенными командами, которые возвращают окно в исходное положение внутри текста:

```
ЭТО Сдвиг_текста
  ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО
    {
      ЯЩИК+
      ВЛЕВО
      ЯЩИК-
      ВПРАВО
      Сдвиг_текста
    }
    ИНАЧЕ
    {
      // Удаление последнего символа
      ВЛЕВО
      ПИШИ ПУСТО
      ВПРАВО
    }
  ВЛЕВО // Отложенная команда.
        // Она возвращает окно на текущее место в тексте.
КОНЕЦ
```

```
ЭТО Копирование_образца
  ВЛЕВО
  ЕСЛИ НЕ ПУСТО
    ТО    Копирование_образца
    ИНАЧЕ
    {
      ВЛЕВО
      ЯЩИК+
      ВПРАВО
    }
  ВПРАВО // Отложенная команда.
         // Она возвращает окно на текущее место в тексте.
КОНЕЦ
```

Папа. Хотелось бы избавить Корректора от утомительных походов за образцом.

Вася. А это возможно?

Папа. Попробуй, сегодня ты уже предлагал красивое решение подобной задачи.

Вася. Да, конечно, сохранить образец нам опять поможет команда **ОБМЕН**. Вот новое решение:

```
ЭТО Удаление
  Подготовка
  ПОКА НЕ ПУСТО
  {
    ЕСЛИ Я=Л
      ТО    Удаление_символа
      ИНАЧЕ ВПРАВО
  }
КОНЕЦ

ЭТО Подготовка
  ЯЩИК+ ВПРАВО ВПРАВО
КОНЕЦ

ЭТО Удаление_символа
  ВПРАВО
  ЕСЛИ НЕ ПУСТО
```

```
    ТО
    {
      ОБМЕН // Образец на ленту
      ВЛЕВО
      ЯЩИК-
      ВПРАВО
      ЯЩИК+ // Образец в ящик
      Удаление_символа
    }
    ИНАЧЕ
    {
      // Удаление последнего символа
      ВЛЕВО
      ПИШИ ПУСТО
      ВПРАВО
    }

  ВЛЕВО // Отложенная команда.
        // Она возвращает окно на текущее место в тексте.
КОНЕЦ
```

Папа. Очень хорошо! Ты существенно сократил не только текст программы, но и время исполнения. Теперь давайте решим задачу на вставку.

Задача 6

В тексте на ленте пропущено слово КОТ. Вставить это слово вместе с последующим пробелом. В начальный момент окно установлено на место вставки. Пример возможного начального состояния среды показан на рис. 13.11.

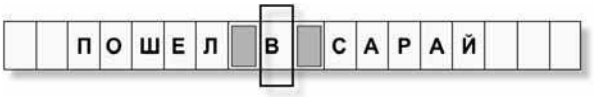


Рис. 13.11

Вася. Придётся сдвигать текст от места вставки, но не влево, как при удалении, а вправо.

Папа. На сколько ячеек предполагается сдвиг?

Вася. На длину вставляемого слова плюс последующий пробел, значит, на четыре ячейки. Вот начало моей программы:

```
ЭТО Вставка
  Сдвиг
  Вставка_слова
КОНЕЦ

ЭТО Вставка_слова
  ПИШИ К ВПРАВО
  ПИШИ О ВПРАВО
  ПИШИ Т ВПРАВО
  ПИШИ ПРОБЕЛ
КОНЕЦ
```

Вася. Процедура Сдвиг оказалась не так проста, как я ожидал... Надо поставить окно в конец текста и оттуда перемещать символы на 4 клетки вправо. Остановиться нужно тогда, когда сдвиг достигнет места вставки.

Петя. Здесь опять поможет рекурсия с отложенной командой, но эта команда — не команда из СКИ, а вызов процедуры.

```
ЭТО Сдвиг
  ВПРАВО
  ЕСЛИ НЕ ПУСТО ТО Сдвиг
  Сдвиг_символа // Отложенная команда.
                // Она выполнится столько же раз, сколько
                // команда ВПРАВО
КОНЕЦ

ЭТО Сдвиг_символа
  ВЛЕВО
  ЯЩИК+
  ПОВТОРИ 4 ВПРАВО
  ЯЩИК-
  ПОВТОРИ 4 ВЛЕВО
КОНЕЦ
```

Вася. Удивительно красивое решение! Даже не верится, что оно будет работать.

Братья запустили программу, и, к восхищению Васи, она заработала без каких-либо замечаний.

Задачи

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. 13.12).

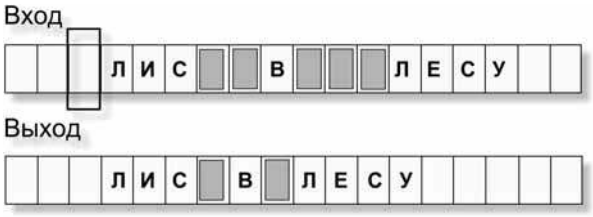


Рис. 13.12

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. 13.13).

Рис. 13.13

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. 13.14).

Рис. 13.14

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 символ «|», расположенный между образцом и текстом (рис. 13.15).

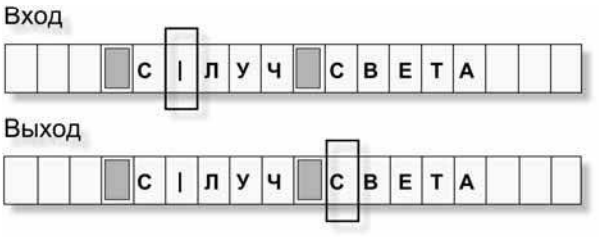


Рис. 13.15

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. 13.16).

Рис. 13.16

Глава 14

Трансляторы

14.1. Проверка объектов

— Да, у меня организм крепкий, а голова ещё крепче, — хвастливо сказал Незнайка. — У другого на моём месте обязательно было бы мозготрясение.

Н. Носов

Петя. Моя последняя курсовая работа была посвящена трансляторам. Очень интересная тема! Можно ли написать транслятор на языке Корректора?

Папа. Почему бы и нет! Напомни Васе, что это такое.

Вася. Думаю, что не забыл наши уроки. **Транслятор** — это специальная программа, которая переводит текст с языка программирования в программу, состоящую из команд СКИ (рис. 14.1).

Петя. Верно! Одна из важных задач трансляции — распознавание и анализ отдельных элементов текста программы. Например, часто необходима проверка, является ли объект текста программы числом.

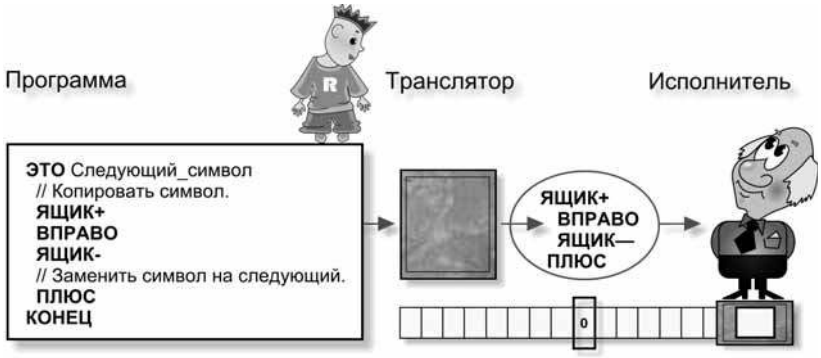


Рис. 14.1. Схема работы транслятора

Папа. Вот и давайте начнём наше занятие с такой задачи: проверить, является ли текст на ленте Корректора числом. В начальный момент окно установлено на первый символ текста.

Петя. Что мы будем понимать под числом?

Папа. Правильный вопрос. Ну, давайте для начала решим задачу для целых неотрицательных чисел без знака.

Задача 1

Проверить, является ли текст на ленте целым неотрицательным числом без знака. В начальный момент окно установлено на первый символ текста (рис. 14.2).

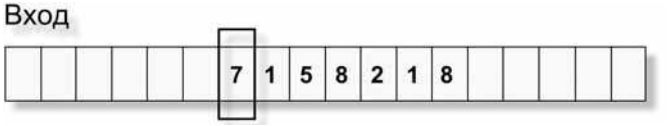


Рис. 14.2

Вася. Ну, это просто. Проверяем каждый символ текста — все они должны быть цифрами. Результат проверки будем записывать в ящик. Сначала в него поместим 1, а если обнаружим в записи не цифру, поместим в него 0:

```
// Проверка целого неотрицательного числа
ЭТО Проверка_числа
  Подготовка_ящика
  Работа
  Запись_ответа
КОНЕЦ

ЭТО Подготовка_ящика
  ОБМЕН ПИШИ 1 ОБМЕН
КОНЕЦ

ЭТО Работа
  ЕСЛИ ПУСТО ТО { ОБМЕН ПИШИ 0 ОБМЕН} // Если запись на ленте пустая.
  ПОКА НЕ ПУСТО
  {
    ЕСЛИ НЕ ЦИФРА
      ТО
      {
        ОБМЕН
```

```
        ПИШИ 0
        ОБМЕН
      }
    ВПРАВО
  }
КОНЕЦ

ЭТО Запись_ответа
  ВПРАВО
  ЯЩИК-
  ЕСЛИ 1
   ТО
   {
     ПИШИ Д ВПРАВО
     ПИШИ А
   }
   ИНАЧЕ
   {
     ПИШИ Н ВПРАВО
     ПИШИ Е ВПРАВО
     ПИШИ Т
   }
КОНЕЦ
```

На рис. 14.3 показаны примеры работы программы.

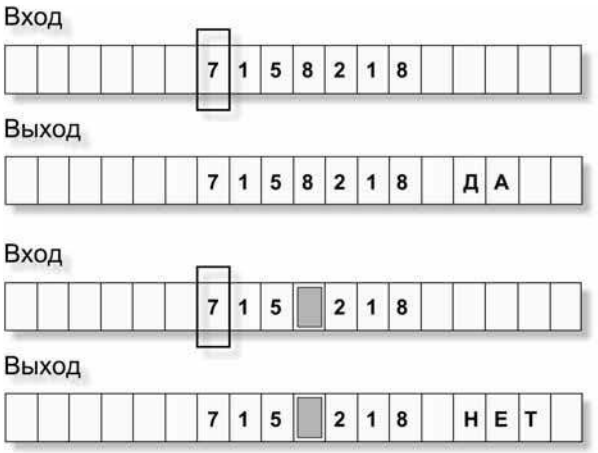


Рис. 14.3

Папа. Теперь решим задачу для любых целых чисел, положительных и отрицательных.

Задача 2

Проверить, является ли текст на ленте целым числом. В начальный момент окно установлено на первый символ текста (рис. 14.4).

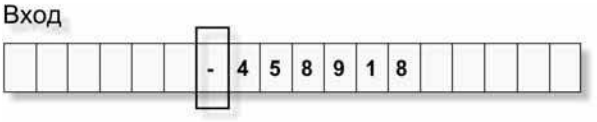


Рис. 14.4

Вася. Значит, перед цифрами может стоять знак минус, и первый символ нужно проверить отдельно.

Папа. Будем предполагать возможность появления числа и со знаком плюс. Правильными будут теперь считаться, например, числа: 4652, -657, +08675.

Вася. Решение получается небольшим изменением предыдущей программы:

```
// Проверка целого числа
ЭТО Проверка_числа
  Подготовка_ящика     // Совпадает с одноимённой процедурой в задаче 1
  Проверка_первого
  Проверка_остальных
  Запись_ответа        // Совпадает с одноимённой процедурой в задаче 1
КОНЕЦ

ЭТО Проверка_первого
  ЕСЛИ НЕ ЦИФРА ТО ЕСЛИ НЕ + ТО ЕСЛИ НЕ - ТО Ошибка
  ВПРАВО
КОНЕЦ

ЭТО Проверка_остальных
  ПОКА НЕ ПУСТО
  {
    ЕСЛИ НЕ ЦИФРА ТО Ошибка
    ВПРАВО
  }
```

```
КОНЕЦ

ЭТО Ошибка
  ОБМЕН ПИШИ 0 ОБМЕН
КОНЕЦ
```

На рис. 14.5 показан пример работы программы.

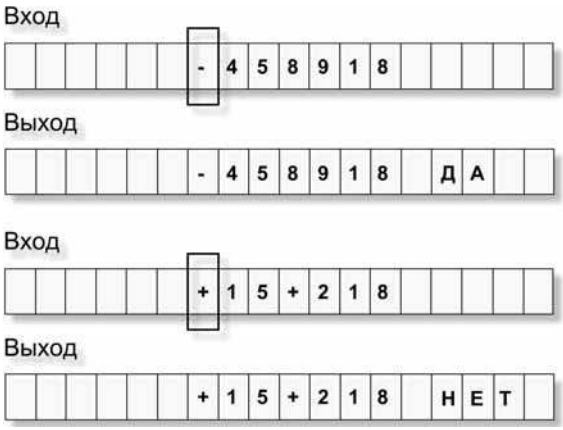


Рис. 14.5

Задачи

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. 14.6 показаны примеры работы программы.

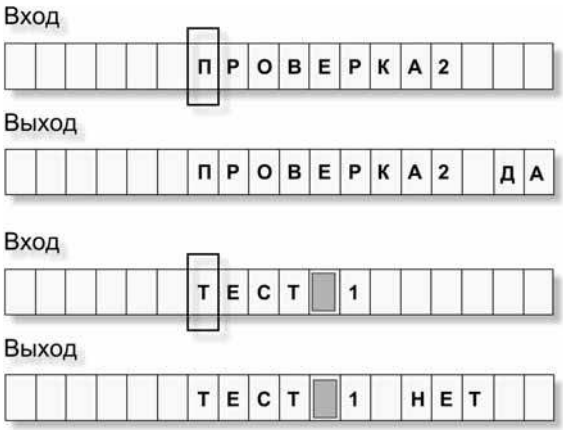


Рис. 14.6

2. На ленте записан текст. Проверить, является ли он числом, и записать результат проверки на ленте в виде табл. 14.1.

Таблица 14.1

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. 14.7 показаны примеры работы программы.

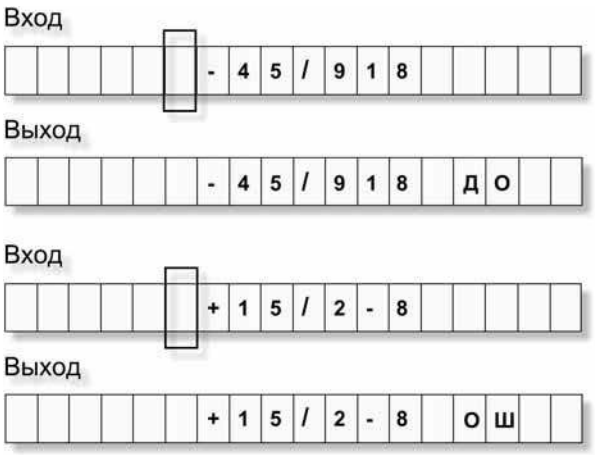


Рис. 14.7

14.2. Транслятор для Плюсика

— Заходите, — пригласил Винтика и Шпунтика Бублик. — Я вас познакомлю с Шурупчиком. Это интересная личность.

Н. Носов

Петя. Роботландский исполнитель Плюсик — хороший объект для упражнений по программированию трансляторов.

Папа. Согласен. Стек, на котором Плюсик проводит свои вычисления, почти всегда используют при программировании трансляторов, да и, вообще, это интересный исполнитель.

Петя. Для наших целей я немного изменил его СКИ. Вот описание исполнителя, пригодное для наших занятий.

Исполнитель Плюсик

Плюсик — это исполнитель, предназначенный для выполнения арифметических вычислений над целыми неотрицательными числами. Его среда состоит из стека, арифметического устройства (сокращенно АУ) и канала связи между стеком и АУ (рис. 14.8).

Рис. 14.8. Исполнитель Плюсик и его среда

СКИ Плюсика включает следующие команды (рис. 14.9).

Рис. 14.9. СКИ Плюсика

Папа. Давайте сначала вспомним, как работает стек, а затем уточним, как Плюсик выполняет свои команды.

Вася. **Стек** — это специальное устройство для хранения чисел. Оно похоже на детскую пирамидку: как её кольца по очереди нанизываются на стержень, так и числа по очереди поступают в стек (рис. 14.10).

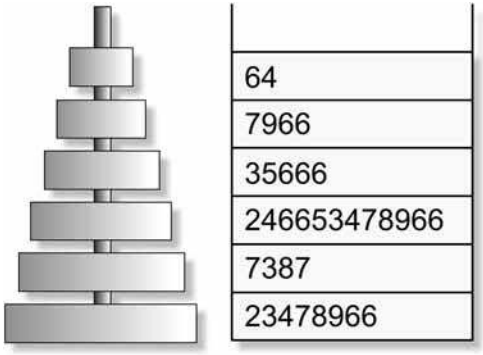


Рис. 14.10. Стек

Петя. Вообще говоря, в стеке могут храниться не только числа, но у Плюсика это действительно так.

Папа. А как извлекать информацию из стека?

Вася. Как в пирамидке мы не можем снять нижнее кольцо, не сняв предварительно верхние, так и в стеке: извлекать из хранилища можно только последнее число.

Папа. Очень хорошо! Теперь давайте рассмотрим, как Плюсик выполняет команды.

Петя. Первая команда П предназначена для записи чисел в стек. Несколько примеров представлены в табл. 14.2.

Таблица 14.2

Команда	Пояснение	Состояние стека после выполнения команды
П125	Положить в стек число 125	125
П25	Положить в стек число 25	25 125
П10	Положить в стек число 10	10 25 125

Вася. Вторая команда О проста и понятна: выполняя её, Плюсик очищает стек от всех чисел (рис. 14.11).

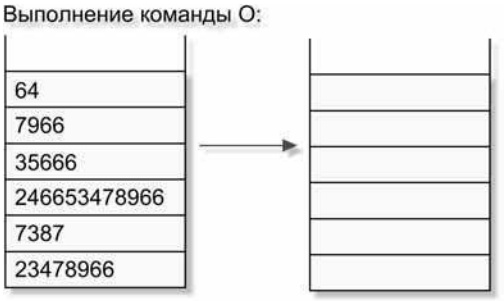


Рис. 14.11. Работа команды О

Петя. Остальные команды предназначены для выполнения вычислений. Они все работают одинаково. Из стека извлекаются последовательно два числа. По каналу связи они передаются в АУ, где над ними выполняется арифметическое действие, а результат по каналу связи снова помещается в стек (рис. 14.12).

Папа. Важно отметить, что последнее извлечённое число становится первым операндом предстоящей операции, а извлечённое первым — вторым.

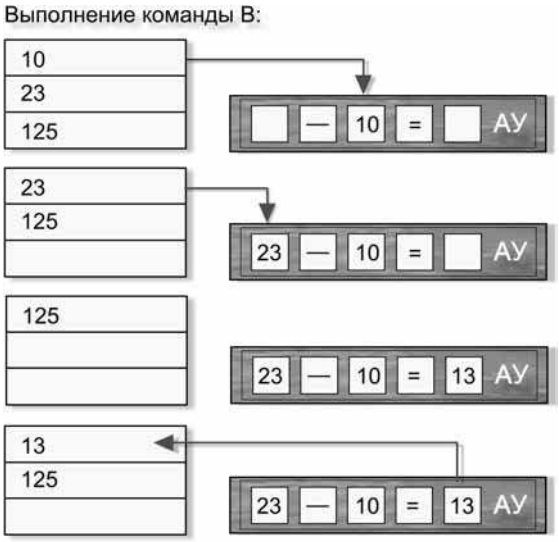


Рис. 14.12. Работа вычислительных команд

Петя. Да, я это продемонстрирую на типичной для Плюсика задаче: написать программу для вычисления арифметического примера:

```
2 * (127 - 32)
```

Вот как может выглядеть решение:

```
П2
П127
П32
В
У
```

Вася. А я нарисовал схемы выполнения Плюсиком этой программы (табл. 14.3).

Таблица 14.3

Команда	Пояснение	Состояние стека после выполнения команды
П2	Положить в стек число 2	2

Таблица 14.3 *(окончание)*

Команда	Пояснение	Состояние стека после выполнения команды
П127	Положить в стек число 127	127 2
П32	Положить в стек число 32	32 127 2
В	Извлечь из стека два числа, от последнего отнять первое, результат поместить в стек	95 2
У	Извлечь из стека два числа, последнее умножить на первое, результат поместить в стек	190

Папа. Ну, с Плюсиком мы разобрались, теперь давайте вернёмся к Корректору.

Петя. Предлагаю такую задачу.

Задача 3

Перевести пример на сложение двух чисел в программу для Плюсика. Например, запись 25 + 7 должна быть преобразована в текст П25П7С. В начальный момент в окошко виден первый символ примера (рис. 14.13).

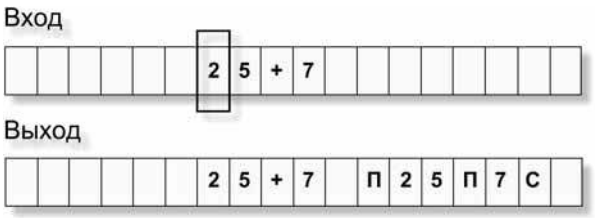


Рис. 14.13

Вася. Предлагаю такую главную процедуру:

```
ЭТО Транслятор
  Записать_команду_П
  Перенести_число      // Перенести первое число.
  Записать_команду_П
  ВПРАВО               // Перешагнуть через знак +.
  Перенести_число      // Перенести второе число.
  Записать_команду_С
КОНЕЦ
```

На рис. 14.14–14.19 пояснена работа каждой команды в этой процедуре.

Рис. 14.14. Записать_команду_П

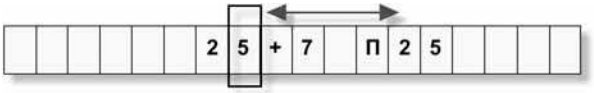


Рис. 14.15. Перенести_число

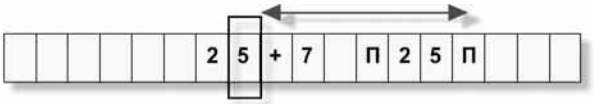


Рис. 14.16. Записать_команду_П

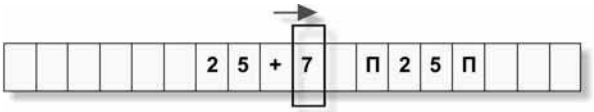


Рис. 14.17. ВПРАВО

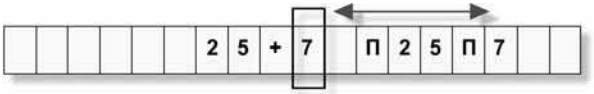


Рис. 14.18. Перенести_число

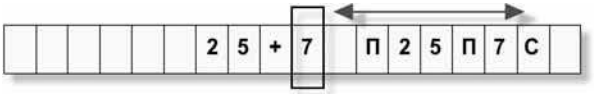


Рис. 14.19. Записать_команду_С

Папа. Названия процедур выбраны удачно: сразу становится понятен алгоритм решения. Ну, действуй дальше.

Вася. Результат будет записываться справа от исходного примера, через одну пустую клетку.

Начну с процедур записи команд, они мне кажутся более простыми. Сначала нужно дойти до пустого места, разделяющего пример и результат, затем переместить окно в конец результата и записать там нужный знак.

Петя. Потом необходимо вернуть окно на текущее место в исходном примере — это самое сложное!

Вася. Ну, этот фокус мы уже проходили: нам, как всегда, поможет отложенная команда в рекурсивной процедуре. Вот моё решение:

```
ЭТО Записать_команду_П
  ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО    Записать_команду_П
    ИНАЧЕ Поставить_П
  ВЛЕВО
КОНЕЦ

ЭТО Поставить_П
  ВПРАВО
  ПОКА НЕ ПУСТО ВПРАВО
  ПИШИ П
  ПОКА НЕ ПУСТО ВЛЕВО
КОНЕЦ

ЭТО Записать_команду_С
  ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО    Записать_команду_С
    ИНАЧЕ Поставить_С
  ВЛЕВО
КОНЕЦ

ЭТО Поставить_С
  ВПРАВО
  ПОКА НЕ ПУСТО ВПРАВО
  ПИШИ С
  ПОКА НЕ ПУСТО ВЛЕВО
КОНЕЦ
```

Петя. Процедуры, записывающие команды П и С, почти совпадают.

Вася. Понял твой намёк и придумал, как сократить программу: надо использовать ящик! Тогда можно обойтись одним экземпляром процедуры, записывающей в конец результата символ из ящика. В первом случае это символ «П», а во втором — «С».

Вот новый вариант моего решения (я внёс правку и в главную процедуру программы):

```
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
  Записать_символ
КОНЕЦ

ЭТО Записать_символ
  ВПРАВО
  ЕСЛИ НЕ ПУСТО
    ТО    Записать_символ
    ИНАЧЕ Записать
  ВЛЕВО
КОНЕЦ

ЭТО Записать
  ВПРАВО
  ПОКА НЕ ПУСТО ВПРАВО
  ЯЩИК-
  ПОКА НЕ ПУСТО ВЛЕВО
КОНЕЦ
```

Папа. Осталось написать процедуру, выполняющую копирование числа.

Вася. Будем использовать уже готовую процедуру `Записать_символ`:

```
ЭТО Перенести_число
 ПОКА ЦИФРА
 {
   ЯЩИК+
   Записать_символ
   ВПРАВО
 }
КОНЕЦ
```

Петя. Отлично! У меня есть повод гордиться младшим братом.

Но когда программа была запущена, обнаружилось, что работает она не совсем правильно: последняя команда `С` записывается не вплотную к предыдущей записи, а через пустое место от неё (рис. 14.20).

Рис. 14.20

Когда первое удивление прошло, братья быстро нашли ошибку и исправили её. Читателю предлагается сделать это самостоятельно.

Задачи

1. Найти и исправить логическую ошибку в программе `Транслятор`.
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.
3. Арифметический пример содержит только целые неотрицательные числа, соединенные 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.
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. 14.21.

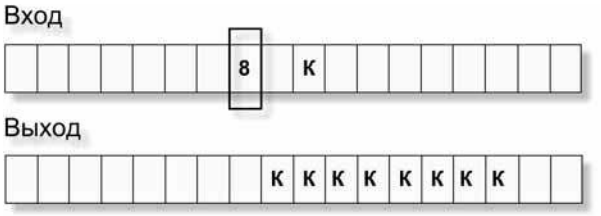


Рис. 14.21

5. Построить на базе Корректора нового исполнителя со следующей СКИ (табл. 14.4).

Таблица 14.4

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. 14.22).

Команды нового исполнителя, записанные друг за другом без всяких разделителей, образуют программу.

Рис. 14.22

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 т. д.).

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 трехсимвольное 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.

Глава 15

Задачи

— Что же тут придумаешь? — сказал Винтик. — Если бы у нас были мешки с песком, можно было бы сбросить один мешок.

Н. Носов

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.

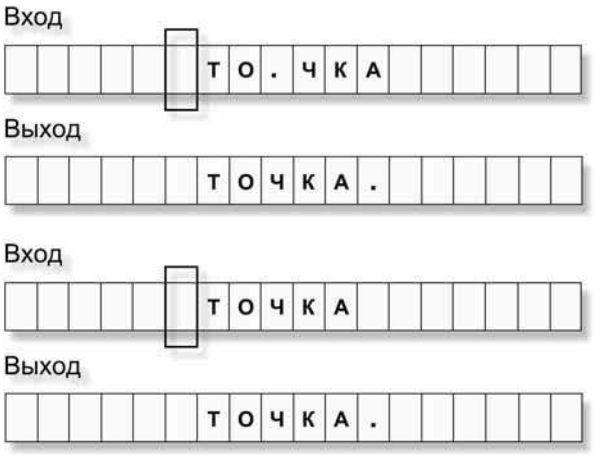


Рис. 15.1

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.

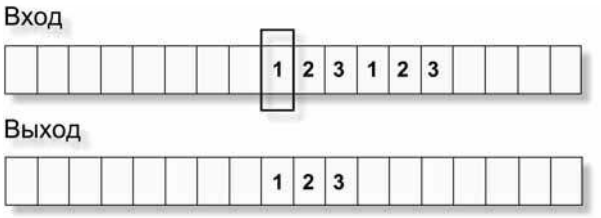


Рис. 15.2

3. Неисправная клавиатура (В. П. Семенко, Рубцовск)

   Окно установлено на первый символ записи. В результате неисправности клавиатуры каждый символ записи удвоился. Удалить повторы символов. Пример начального и конечного положений среды показан на рис. 15.3.

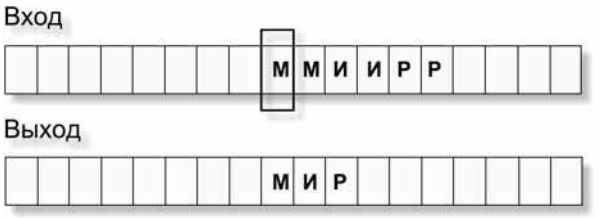


Рис. 15.3

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.

5. Голосование символов (В. П. Семенко, Рубцовск)

   Для надежности 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 показан на рис. 15.5.

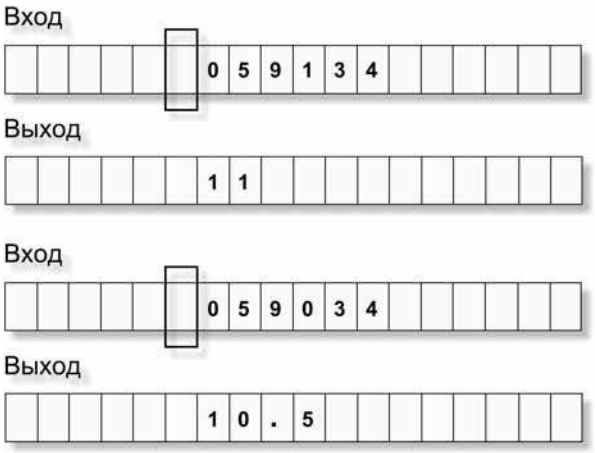


Рис. 15.4

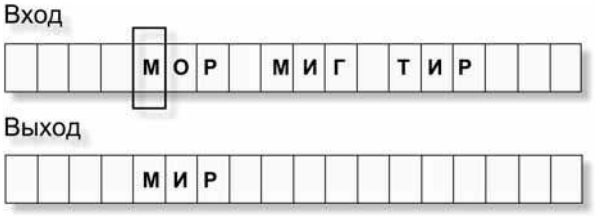


Рис. 15.5

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

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.

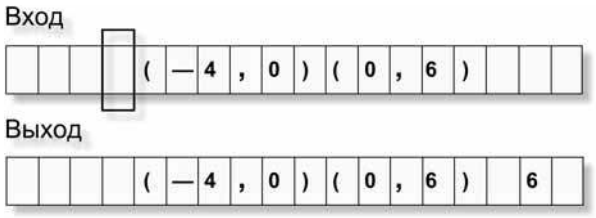


Рис. 15.7

8. Распаковка отрезка (В. П. Семенко, Рубцовск)

   Под отрезком на ленте будем понимать последовательность (слева направо) ячеек, в первой из которых записан символ H (начало отрезка), в последней K (конец отрезка), а промежуточные ячейки (тело отрезка) пусты. Число ячеек между символами H и K 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, то есть H0K — запакованный, а HK — 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.

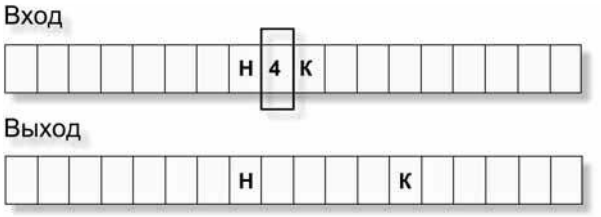


Рис. 15.8

9. Пятёрки (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.

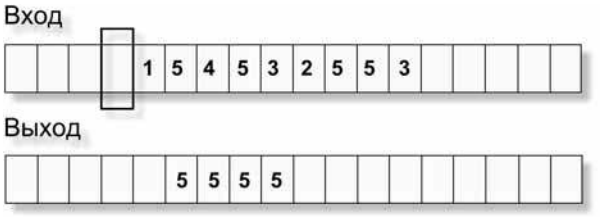


Рис. 15.9

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.

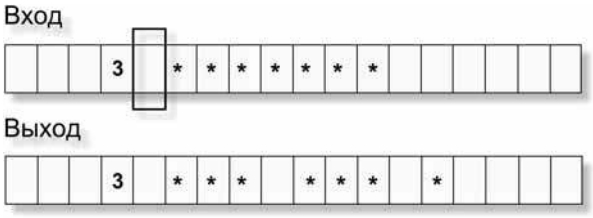


Рис. 15.10

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

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.

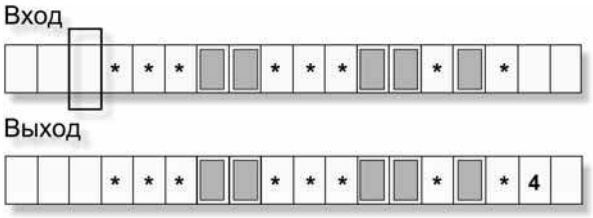


Рис. 15.12

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.

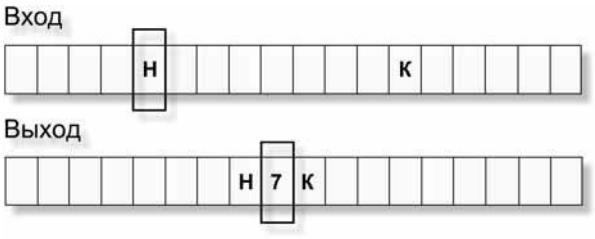


Рис. 15.13

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.

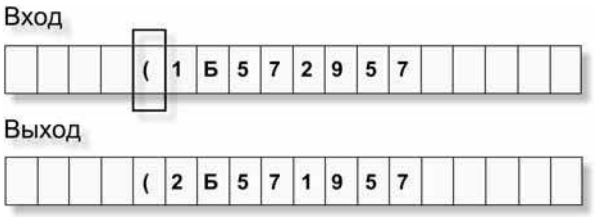


Рис. 15.14

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 по-

рядок 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.

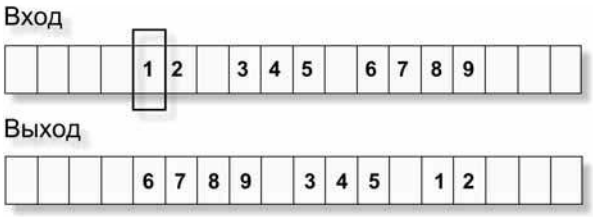


Рис. 15.15

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.

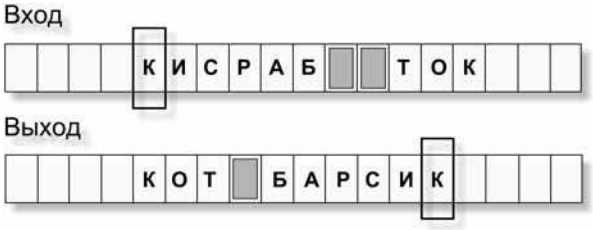


Рис. 15.16

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.

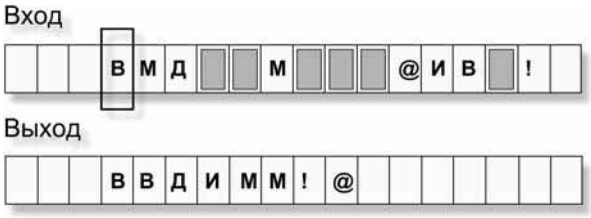


Рис. 15.17

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.

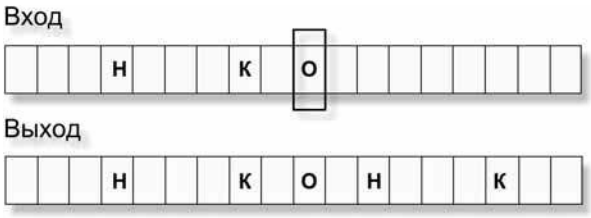


Рис. 15.18

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.

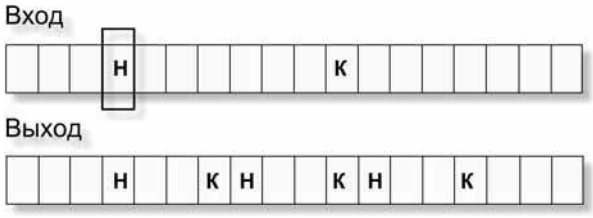


Рис. 15.19

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.

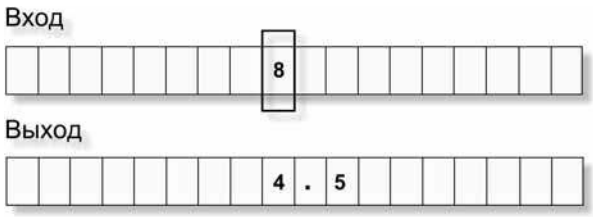


Рис. 15.20

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.

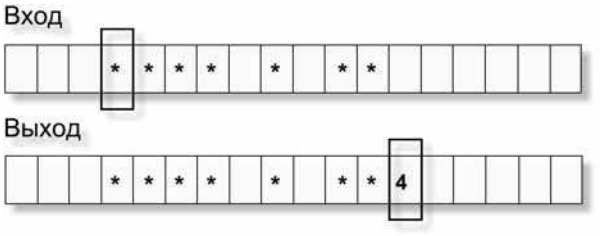


Рис. 15.21

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.

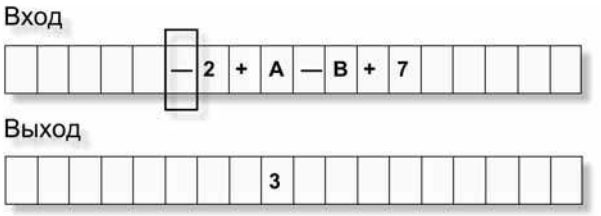


Рис. 15.22

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.

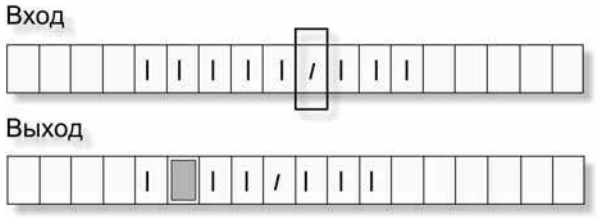


Рис. 15.23

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 · 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.

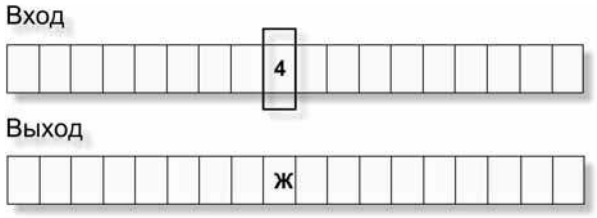


Рис. 15.24

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.

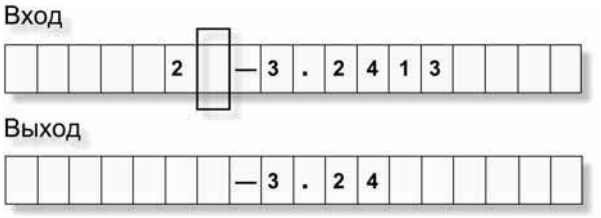


Рис. 15.25

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.

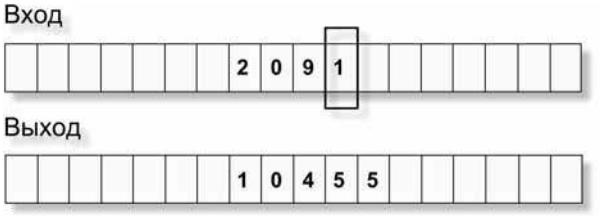


Рис. 15.26

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 то же, 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).

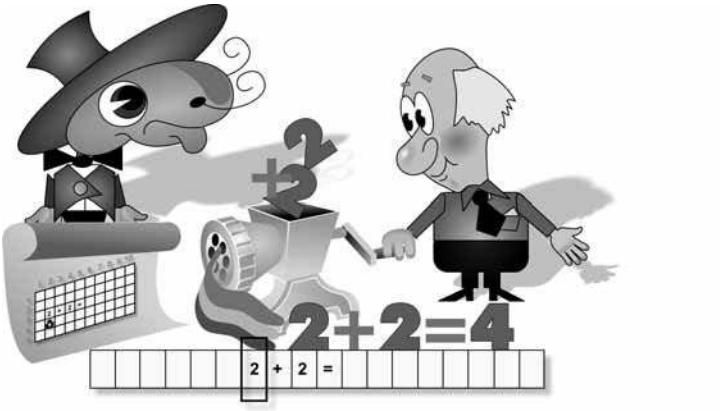


Рис. 16.1. Почему бы не написать для нас транслятор!

Глава 16

Язык Бэкуса-Наура

16.1. Понятие метаязыка

Язык, при помощи которого описывают другие языки, называется **метаязыком**. Понятно, что русский язык является метаязыком. Ведь на этом языке мы рассказываем, например, о языке роботландских исполнителей. Однако русский язык, как и любой другой естественный язык, слишком расплывчат и словообилен для строгих формальных описаний.

16.2. Определения на языке Бэкуса-Наура

Для описания формальных конструкций в программировании обычно применяют метаязык **Бэкуса-Наура**. Впервые этот язык был использован для описания языка программирования Алгол-60.

В языке Бэкуса-Наура используются следующие специальные символы (метасимволы) (табл. 16.1).

Таблица 16.1

Метасимвол	Пояснение
<>	Скобки, выделяющие определяемое понятие
::=	Составной символ, читается «по определению есть»
|	Символ, обозначающий альтернативу, читается «или»

Примеры определений.

Определение 1

```
<знак сложения> ::= +
```

Знак сложения по определению есть символ «+».

Определение 2

```
<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Цифра по определению есть: 0, или 1, или 2, или 3, или 4, или 5, или 6, или 7, или 8, или 9.

Наиболее интересно использование языка Бэкуса-Наура для создания рекурсивных определений.

Определение 3

```
<слово> ::= <буква> | <слово><буква>          (1)
<буква> ::= A | M                             (2)
```

Прочитать это определение можно так:

- «Слово» по определению есть «буква» или объект, про который мы знаем, что он — «слово», вслед за которым записана «буква».
- «Буква» — это либо символ A, либо символ M.

Как видите, первая строка в определении содержит рекурсию!

Проверим, соответствуют ли определению 3 объекты: A, MA, AB.

1. Объект A — слово, потому что: A — буква (правило 2), а буква является словом (правило 1).
2. Объект MA — слово, потому что: M — это слово (так как M — это буква по правилу 2, а значит, слово по правилу 1). Слово, за которым следует буква, является словом (правило 1), значит, MA — слово.
3. Объект AB — не слово, потому что за словом A следует не буква в смысле правила 2.

Как проверить соответствие объекта определению? Нужно следовать правилам, заданным в определении. Если правила рекурсивные, значит, и проверка будет содержать рекурсию. Проверим, например, что объект MAMA является словом в смысле определения 3.

1. Объект MAMA — слово, если словом является MAM (т. к. A — буква).
2. Объект MAM — слово, если словом является MA (т. к. M — буква).
3. Объект MA — слово, если словом является M (т. к. A — буква).
4. Объект M — слово (т. к. M — буква).

Заметим, что по определению 3 словом является любая (не пустая!) последовательность из букв A и M. Это наблюдение легко доказать методом математической индукции.

16.3. Математическая индукция

Математическая индукция — универсальный способ доказательства утверждения F, содержащего зависимость от целого числа n. Зависимость обозначается как F(n).

Доказательство справедливости утверждения F(n) для всех $n > n_0$ методом математической индукции проводится в три шага.

1. *База индукции.*

   Утверждение F(n) проверяется при начальном значении $n = n_0$ (например, при $n_0 = 1$).

2. *Индуктивное предположение.*

   Выдвигается индуктивное предположение. Оно заключается в том, что утверждение F(n-1) считается справедливым при некотором $n > n_0$.

3. *Индуктивный переход.*

   Доказывается справедливость F(n) с учётом индуктивного предположения, т. е. предположения о справедливости $F(n - 1)$. Доказательство основывается на попытке представить зависимость F(n) через $F(n - 1)$.

Итак, нужно доказать, что F(n) справедливо, если справедливо $F(n - 1)$. Это доказательство, наряду с проверкой F для одного конкретного значения $n = n_0$, позволяет утверждать, что F(n) справедливо всегда. Почему?

Пусть доказано, что утверждение F(n) справедливо, если справедливо $F(n - 1)$, и пусть для определённости $n_0 = 1$. Тогда можно утверждать, что F(n) справедливо и для любого $n > 1$, если справедливость F(1) проверена непосредственно.

В самом деле, это следует из нисходящей к F(1) цепочки рассуждений:

...

F(5) справедливо, если справедливо F(4)

F(4) справедливо, если справедливо F(3)

F(3) справедливо, если справедливо F(2)

F(2) справедливо, если справедливо F(1)

F(1) — база индукции, справедливость проверена непосредственно.

Мы будто заставляем работать некоторого исполнителя для проверки справедливости утверждения F(n) по следующей рекурсивной программе:

```
ЭТО Индукция(n)
  ЕСЛИ n = 1
    ТО ОТВЕТ("F - справедливо")
    ИНАЧЕ Индукция(n-1)
КОНЕЦ
```

Например, процедура `Индукция(3)` рекурсивно выполнит процедуру `Индукция(2)`. Процедура `Индукция(2)`, аналогично, выполнит процедуру `Индукция(1)`. В последней процедуре рекурсия заканчивается: ведь F(1) справедливо — это проверенная база индукции.

Приведённая программа, конечно, лишена всякого практического смысла. Она просто демонстрирует математическую индукцию как рекурсивный процесс, который, пользуясь доказанным индуктивным переходом, сводит проверку F(n) к проверке F(1).

Вернёмся к нашей задаче. Докажем, что любая последовательность F(n), состоящая из n символов, каждый из которых принимает значение A, либо M, является словом в смысле определения 3.

1. *База индукции.*

   F(1) состоит из одного символа A или M и в силу правила 2 является буквой, а значит, в силу правила 1 — словом.

2. *Индуктивное предположение.*

   Считаем, что F(n − 1) — слово для некоторого n > 1.

3. *Индуктивный переход.*

   Докажем, что F(n) — тоже слово. Представим F(n) для n > 1 в виде двух частей: последний символ и всё остальное. Тогда F(n) можно записать как:

   F(n) = F(n − 1)s (*)

   Здесь s — последний символ в F(n), а он может принимать значение A или M, т. е. является буквой в силу правила 2.

   Итак, мы представили F(n) как слово F(n − 1), за которым следует буква. Но тогда в силу правила 1 — F(n) тоже слово.

   Индуктивный переход доказан, а значит, доказано исходное утверждение.

16.4. Задачи

1. Задано определение:

Определение 4

```
<имя> ::= <буква> | <имя><буква> | <имя><цифра>
<буква> ::= X | Y
<цифра> ::= 1 | 2
```

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

Доказать следующее утверждение. Имя в смысле определения 4 — это любая последовательность из букв X, Y и цифр 1, 2, которая начинается с буквы.

2. Задано определение:

Определение 5

```
<имя> ::= <буква> | <имя><буква> | <имя><цифра>
<буква> ::= <пусто> | X | Y
<цифра> ::= 1 | 2
<пусто> ::=
```

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

Верны ли следующие утверждения.

а) Любая последовательность из символов X и 1 является именем в смысле определения 5.

б) Множество таких имён совпадает с множеством имён, задаваемых определением 5.

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)))

1. Запишите следующее определение при помощи языка Бэкуса-Наура.

   Пример — это несколько цифр, соединённых между собой знаком «–».

2. Запишите при помощи языка Бэкуса-Наура определение натурального числа.
3. Запишите при помощи языка Бэкуса-Наура определение целого числа.

4. 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 +

Глава 17

Кукарача и лексический анализ выражений

— Умеешь ты нести яйца? — спросила она утёнка.
— Нет.
— Так и держи язык на привязи!

Г.-Х. Андерсен

Откроем обман: мы не умеем писать трансляторы в среде Кукарачи!

На самом деле это обман только наполовину:

- мы умеем писать в среде Кукарачи лексические анализаторы, а это всё же составная часть транслятора;
- мы на самом деле умеем писать трансляторы в среде исполнителя Ректор!

17.1. Транслятор

Как известно, исполнитель не понимает языка программирования.

Когда в программе записывается код «**ПОВТОРИ** 5 **ВПРАВО**», мы знаем, что запись эта исполнителю недоступна. Ведь такой команды нет в его СКИ. Тем не менее, когда программа работает, Кукарача перемещается на пять клеток вправо!

Дело в том, что запись «**ПОВТОРИ** 5 **ВПРАВО**» — это команда языка программирования, а язык программирования «понимает» не исполнитель, а транслятор.

Транслятор — это специальная программа, которая переводит текст с языка программирования в последовательность команд из СКИ. Эта последовательность команд передаётся исполнителю, и Кукарача делает пять прыжков по полю (рис. 17.1).

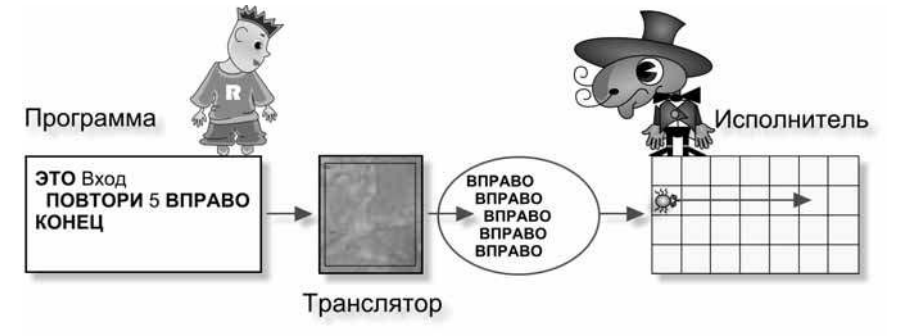


Рис. 17.1. Схема работы транслятора

17.2. Лексема

Одна из важных задач трансляции — сборка из последовательных символов смысловых элементов. Эти элементы называются **лексемами**.

Лексический анализ выражений — это выделение в тексте лексем, кирпичиков, из которых строятся конструкции языка. До лексического анализа текст рассматривается как однородная цепочка символов, после — как цепочка лексем.

Например, текст

```
ПОВТОРИ 5 ВПРАВО
```

содержит 16 символов (среди них два символа пробела). После лексического анализа текст преобразуется в цепочку из трёх лексем (табл. 17.1).

Таблица 17.1

Лексема	Что означает
ПОВТОРИ	Ключевое слово, начало цикла
5	Число
ВПРАВО	Ключевое слово, команда из СКИ

17.3. Диаграмма переходов

Один из методов решения задач лексического анализа основан на составлении диаграммы переходов с последующим буквальным переводом построенной диаграммы в программу.

Содержательная часть работы выпадает именно на составление схемы. Когда схема нарисована, остаётся просто переписать её на языке программирования.

Само программирование (кодирование) при этом выполняется почти механически без каких-либо дополнительных размышлений.

Построение диаграммы переходов проиллюстрируем на следующем примере.

Определение 1

```
<число> ::= <целое> | <дробь>
<целое> ::= <цифра> | <целое><цифра>
<дробь> ::= <целое> / <целое>
<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Диаграмма переходов для анализа записи на соответствие определению 1 изображена на рис. 17.2.

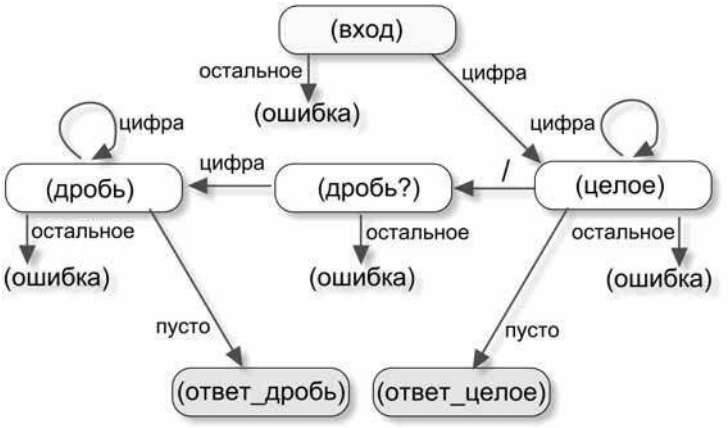


Рис. 17.2. Диаграмма переходов

Схема показывает состояния алгоритма (они обозначены словами в овальных блоках) и переходы алгоритма из одного состояния в другое (они изображены стрелками) в зависимости от значения очередного символа входной записи (символы надписаны на стрелках-переходах).

Состояние «ошибка» на схеме не изображено, но переходы в это состояние показаны.

Рассмотрим схему в динамике работы алгоритма.

- В начальный момент алгоритм находится в состоянии (вход).
- Любой очередной символ в этом состоянии, кроме символа «цифра», приводит к переходу в состояние (ошибка).
- Если в состоянии (вход) очередным символом является «цифра», алгоритм переходит в состояние (целое).

- Какой символ допустим в состоянии (целое)? Из определения следует, что таких символов три:
  - «цифра» — алгоритм остаётся в состоянии (целое);
  - «/» — алгоритм переходит в состояние (дробь?);
  - «пусто» — алгоритм переходит в состояние (ответ_целое).
- Состояние (дробь?) вводится дополнительно к состоянию (дробь) для того, чтобы запись, завершающуюся косой чертой, считать ошибочной.
- Состояние (дробь) обрабатывает очередной символ следующим образом:
  - «цифра» — алгоритм остаётся в состоянии (дробь);
  - «пусто» — алгоритм переходит в состояние (ответ_дробь);
  - любой другой символ — алгоритм переходит в состояние (ошибка).

Если из схемы удалить многочисленные переходы в состояние «ошибка», она будет выглядеть более компактно (рис. 17.3).

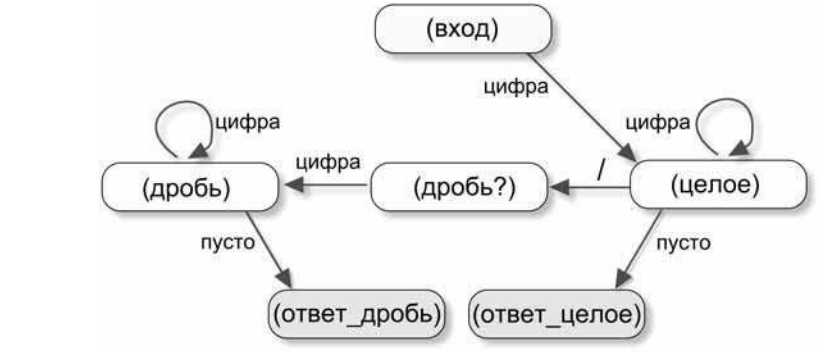


Рис. 17.3. Упрощённая диаграмма переходов

Вместо диаграммы можно строить (часто это гораздо удобнее) таблицу переходов (табл. 17.2).

Последние три строки описывают конечные состояния алгоритма. Если исполнитель оказался в одном их этих состояний, работа программы заканчивается, а название состояния отражает результат анализа исходной записи.

Когда диаграмма (или таблица) построена, остаётся простая и фактически рутинная работа: закодировать диаграмму (или таблицу) на языке программирования.

Рекомендуется каждое состояние отображать в программе одноимённой процедурой, а переходы между состояниями — вызовами соответствующих процедур-состояний. Такой способ записи буквально повторяет схему (таблицу) и практически не требует дополнительных комментариев.

Таблица 17.2

Состояние	Следующий символ			
	цифра	**/**	**пусто**	**остальное**
(вход)	(целое)	(ошибка)	(ошибка)	(ошибка)
(целое)	(целое)	(дробь?)	(ответ_целое)	(ошибка)
(дробь?)	(дробь)	(ошибка)	(ошибка)	(ошибка)
(дробь)	(дробь)	(ошибка)	(ответ_дробь)	(ошибка)
(ответ_дробь)				
(ответ_целое)				
(ошибка)				

17.4. Программа для Кукарачи

Постановка задачи

Пусть запись расположена во второй строке, а исполнитель — перед ней (рис. 17.4).

Рис. 17.4. Начальное состояние среды

Проверить запись на соответствие определению 1.

При этом:

- Если запись есть «целое» — установить исполнитель за записью в первой строке (рис. 17.5).

Рис. 17.5. Кукарача показывает, что запись есть «целое»

- Если запись есть «дробь» — установить исполнитель в конце записи во второй строке (рис. 17.6).

Рис. 17.6. Кукарача показывает, что запись есть «дробь»

- Если в записи есть ошибки, установить исполнителя под первой из них (рис. 17.7).

Рис. 17.7. Кукарача показывает место первой ошибки

Программа

```
ЭТО вход                                  // Состояние (вход)
  шаг                                     // Смотрим следующий символ
  ЕСЛИ  ЦИФРА ТО целое
  ИНАЧЕ          ошибка
КОНЕЦ

ЭТО целое                                 // Состояние (целое)
  шаг                                     // Смотрим следующий символ
  ЕСЛИ       ПУСТО  ТО ответ_целое
  ИНАЧЕ ЕСЛИ ЦИФРА  ТО целое
  ИНАЧЕ ЕСЛИ /      ТО дробь?
  ИНАЧЕ                ошибка
КОНЕЦ

ЭТО дробь?                                // Состояние (дробь?)
  шаг                                     // Смотрим следующий символ
  ЕСЛИ ЦИФРА ТО дробь
  ИНАЧЕ          ошибка
КОНЕЦ
```

```
ЭТО дробь                              // Состояние (дробь)
  шаг                                  // Смотрим следующий символ
  ЕСЛИ       ПУСТО ТО ответ_дробь
  ИНАЧЕ ЕСЛИ ЦИФРА ТО дробь
  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_целое
  ВВЕРХ
КОНЕЦ

ЭТО ответ_дробь
  СТОЯТЬ
КОНЕЦ

ЭТО шаг                                // Толкнуть следующий
  ВНИЗ ВПРАВО ВВЕРХ                    // символ
КОНЕЦ
```

Не правда ли, программа получилось пусть и не очень короткая, но простая, предельно понятная и даже скучная? Она полностью соответствует диаграмме на рис. 17.2 (и соответствующей табл. 17.2).

Совершенно очевидно, что в этой задаче (и всех других аналогичных) самое трудное — построить правильную диаграмму. Вопрос кодирования диаграммы на языке программирования является второстепенным и несущественным. Программирование здесь не содержит никаких идей, и могло бы выполняться автоматически при наличии специального исполнителя-транслятора, который получал бы на входе построенную диаграмму (таблицу переходов), а на выходе выдавал бы готовую программу. Эту программу даже не нужно было бы тестировать.

Конечно, правильность программы означает лишь, что она полностью соответствует диаграмме, является другой формой её записи. Но если диаграмма построена неверно, то неверной будет и эквивалентная ей программа.

Итак, успех построения лексического анализатора выражений по заданному формальному определению, подобному определению 1, полностью зависит от успеха построения правильной диаграммы переходов.

В приведённой программе для Кукарачи есть только одно «тонкое» место — процедура `ошибка`:

```
ЭТО ошибка
  шаг                        // Перемещение остатка записи
  ЕСЛИ НЕ ПУСТО ТО ошибка // в первую строку
  ВЛЕВО                      // Возврат к месту ошибки
КОНЕЦ
```

Благодаря отложенной в рекурсии команде **ВЛЕВО**, Кукарача после перемещения остатка записи в первую строку вернётся к месту ошибки.

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. 17.8).

Рис. 17.8

Если проверяемая запись — число, установить исполнитель в конец записи (рис. 17.9).

Рис. 17.9

Если запись — не число, поставить исполнитель в третью строку поля (рис. 17.10).

Рис. 17.10

2. Решить задачу 1, но если запись не является числом, установить исполнитель под первым неверным символом в записи (рис. 17.11).

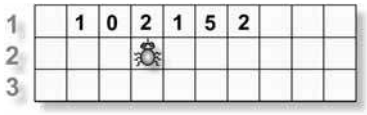


Рис. 17.11

Если запись — число, установить исполнитель за последним её символом (рис. 17.12).

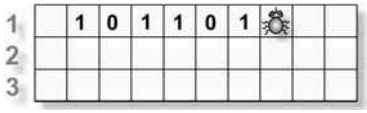


Рис. 17.12

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. 17.13).

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

Примеры результатов выполнения программы (рис. 17.16–17.19).

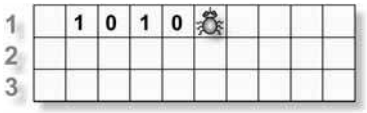


Рис. 17.16

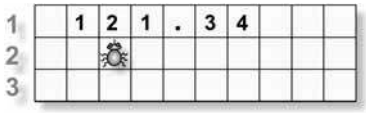


Рис. 17.17

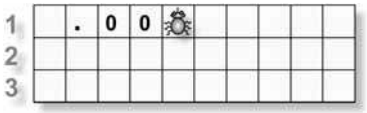


Рис. 17.18

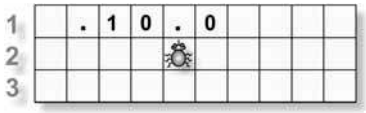


Рис. 17.19

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. 17.20).

Рис. 17.20

Если проверяемая запись — число, установить исполнитель в конец записи (рис. 17.21).

Рис. 17.21

Если запись — не число, поставить исполнитель под первым неверным символом (рис. 17.22).

Рис. 17.22

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. 17.23).

Рис. 17.23

Если проверяемая запись — выражение, установить исполнитель в конец записи (рис. 17.24).

Рис. 17.24

Если запись — не выражение, поставить исполнитель под первым неверным символом (рис. 17.25).

Рис. 17.25

Глава 18

Ах уж эта рекурсия!

> Машина имеет четыре скорости: первую, вторую, третью и четвёртую, а также задний и боковой ход. В задней части машины имеется приспособление для стирки белья. Стирка может производиться во время движения на любой скорости. В спокойном состоянии, то есть на остановках, машина рубит дрова, месит глину и делает кирпичи, а также чистит картошку.
>
> *Н. Носов*

Обозначенный выше эпиграф — про наших ребят: Кукарачу и Корректора (рис. 18.1). Кто бы мог подумать, что с помощью этих простых исполнителей можно решать сложные программистские задачи!

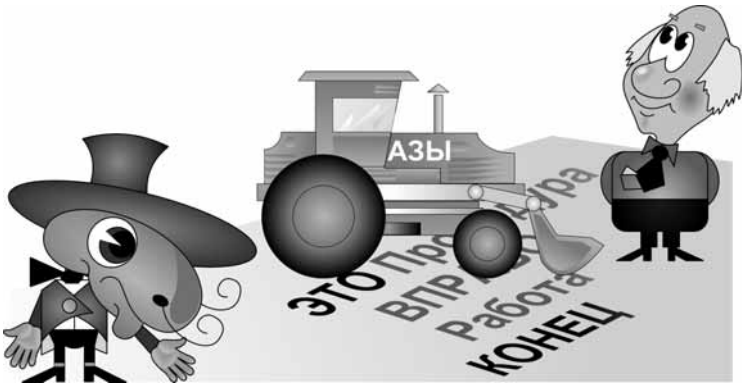


Рис. 18.1. Кукарача и Корректор — парни на все руки!

18.1. Рекурсивная пружинка

В предыдущей главе на языке Бэкуса-Наура было записано:

Определение 1

```
<число> ::= <целое> | <дробь>
<целое> ::= <цифра> | <целое><цифра>
<дробь> ::= <целое> / <целое>
<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

В среде Кукарачи был построен лексический анализатор, который проверял запись на соответствие этому определению.

В этом лексическом анализаторе было использовано небольшое рекурсивное «безумие»:

```
ЭТО ошибка
  шаг                          // Перемещение остатка записи
  ЕСЛИ НЕ ПУСТО ТО ошибка      // в первую строку
  ВЛЕВО                        // Возврат к месту ошибки
КОНЕЦ


ЭТО шаг
  ВНИЗ ВПРАВО ВВЕРХ
КОНЕЦ
```

При этом доверчивым читателям сообщалось, что Кукарача сделает влево не один шаг, а столько, сколько надо, чтобы встать точно под обнаруженной в записи ошибкой!

Проверим это забавное утверждение!

Пусть исходное состояние среды будет таким, как на рис. 18.2.

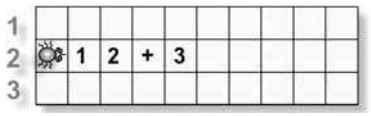


Рис. 18.2

Исследуя запись, исполнитель встретил ошибку (знак «+») (рис. 18.3).

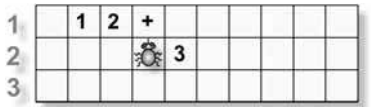


Рис. 18.3

Теперь Кукарача должен сдвинуть остаток записи в первую строку и встать точно под ошибочным символом (рис. 18.4).

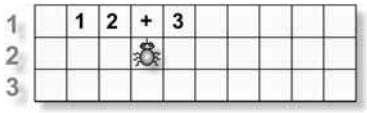


Рис. 18.4

Неужели процедура `ошибка` всё это сделает?

Давайте проследим работу исполнителя по шагам.

Первой выполняется команда `шаг` (рис. 18.5).

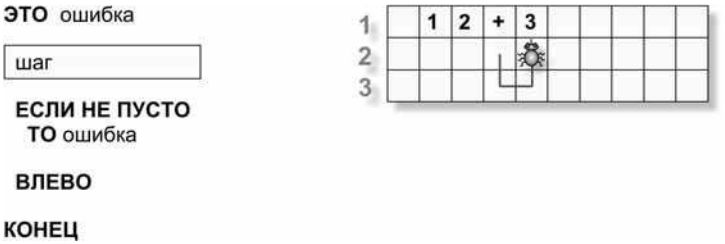


Рис. 18.5

Затем выполняется условная команда (рис. 18.6).

Рис. 18.6

После выполнения условной команды должна выполняться команда, следующая за ней, т. е. команда **`ВЛЕВО`**. Но условная команда ещё не закончила свою работу: т. к. условие истинно, начнёт выполняться второй экземпляр процедуры `ошибка` (рис. 18.7).

Теперь условие ложно, и выполнится команда **`ВЛЕВО`** во втором экземпляре процедуры `ошибка` (рис. 18.8).

Рис. 18.7

Рис. 18.8

Работа второго экземпляра процедуры `ошибка` завершена, и тем самым завершилось выполнение условной команды в первом экземпляре. Кукарача приступает к выполнению последней команды **`влево`** и останавливается точно под знаком «+» (рис. 18.9).

Рис. 18.9

Фокус разоблачён! Команда **`влево`** выполнится столько раз, сколько работала процедура `шаг` в рекурсивных вызовах. А это гарантирует, что исполнитель вернётся к месту ошибки, где бы ошибка ни располагалась в записи!

Разоблачение основано на понимании, что рекурсия — это не переход на начало процедуры. Рекурсивный вызов — это выполнение *нового экземпляра* процедуры: он может порождать «отложенные» команды, которые начнут работать после завершения рекурсии и будут выполняться столько раз, сколько было рекурсивных вызовов.

Ребята из Барнаула (руководитель команды Анатолий Владимирович Бычков) нашли в далёком 1998 году очень выразительный и ёмкий ассоциативный термин для описания работы отложенных в рекурсии команд — «рекурсивная пружина».

Кукарача, делая рекурсивный `шаг`, «сжимает пружину». По окончанию рекурсии «пружина разжимается»: команда **`влево`** будет работать столько раз, сколько «витков было закручено».

Как решается эта задача на «настоящем» языке программирования, таком, как, например, Бейсик? Очень просто: заводим переменную (ячейку памяти), считаем в ней шаги вправо, а потом — Кукарача, шагай столько же раз влево!

Но Кукарача — парень без головы! Ну, нет у него памяти! Совсем! И вот рекурсия выступает как протез отсутствующего органа.

С рекурсией Кукарача (ого!) умеет считать и запоминать!

Вам понравилась рекурсия? Очаровательная штучка, правда?

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. 18.10).

Рис. 18.10

Если проверяемая запись — выражение, установить исполнитель в конец записи (рис. 18.11).

Рис. 18.11

Если запись — не выражение, поставить исполнитель под первым неверным символом (рис. 18.12).

Рис. 18.12

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. 18.13).

Рис. 18.13

Если проверяемая запись — узор, установить исполнитель в конец записи (рис. 18.14).

Рис. 18.14

Если запись — не узор, поставить исполнитель под первым неверным символом (рис. 18.15).

Рис. 18.15

3. Введём определение узора:

Определение 3

```
<узор> ::= <простой> | <фигура><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. 18.16).

Рис. 18.16

Если проверяемая запись — узор, установить исполнитель в конец записи (рис. 18.17).

Рис. 18.17

Если запись — не узор, поставить исполнитель под первым неверным символом (рис. 18.18).

Рис. 18.18

Глава 19

Лексический анализатор в среде Корректора

— Вы меня не понимаете! — сказал утёнок.
— Если уж мы не понимаем, так кто тебя поймёт! Что ж, ты хочешь быть умнее кота и нашей госпожи, не говоря уже обо мне? Не дури, а будь благодарен за всё, что для тебя сделали! Тебя приютили, пригрели, ты попал в такое общество, в котором можешь кое-чему научиться.

Г.-Х. Андерсен

Для Корректора программировать лексические анализаторы приятнее (рис. 19.1).

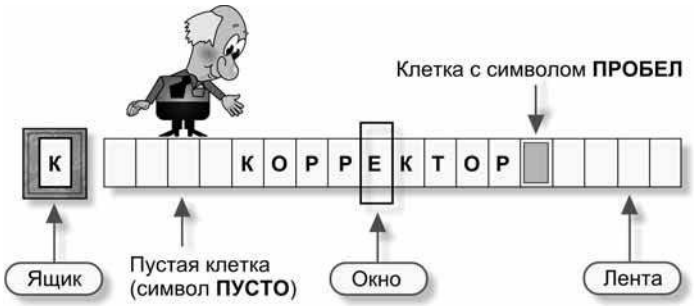


Рис. 19.1. Среда Корректора

Этот исполнитель, в отличие от Кукарачи, умеет записывать на ленту символы, а значит, может написать результат анализа «по-человечески», вместо кукарачинских намёков на ответ своим положением на поле (рис. 19.2). В остальном анализаторы Корректора — кальки с анализаторов Кукарачи.

Рис. 19.2. СКИ Корректора

Решим для Корректора задачу, которая в *главе 2* была решена в среде Кукарачи.

19.1. Постановка задачи

Задано определение:

Определение 1

```
<число> ::= <целое> | <дробь>
<целое> ::= <цифра> | <целое><цифра>
<дробь> ::= <целое> / <целое>
<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Программа должна проверить запись на ленте и поместить результат проверки справа от неё в виде двух букв: ОШ, ЦЛ или ДР (табл. 19.1).

Таблица 19.1

Код результата	Что означает
ОШ	Запись числом не является
ЦЛ	Запись есть «целое»
ДР	Запись есть «дробь»

В начальный момент окошко расположено перед записью, а в конце работы — сразу за ней, если в записи нет ошибок. Когда запись неверная, окошко должно указывать на первый ошибочный символ. На рис. 19.3 показаны возможные варианты обработки записей.

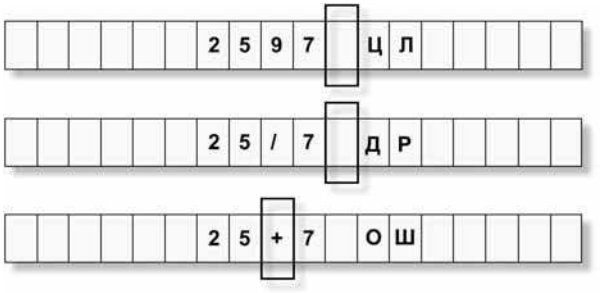


Рис. 19.3

Решение

Диаграмму переходов демонстрирует рис. 19.4.

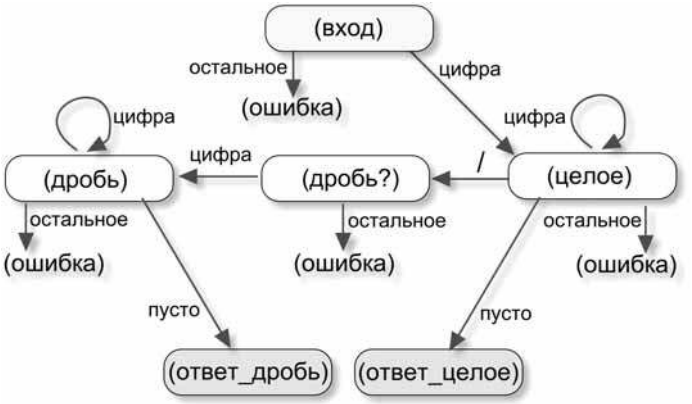


Рис. 19.4

Этой диаграмме соответствует таблица переходов (табл. 19.2).

Таблица 19.2

Состояние	Следующий символ			
	цифра	/	**пусто**	**остальное**
(вход)	(целое)	(ошибка)	(ошибка)	(ошибка)
(целое)	(целое)	(дробь?)	(ответ_целое)	(ошибка)
(дробь?)	(дробь)	(ошибка)	(ошибка)	(ошибка)
(дробь)	(дробь)	(ошибка)	(ответ_дробь)	(ошибка)
(ответ_дробь)				
(ответ_целое)				
(ошибка)				

Перепишем эту таблицу в виде программы для Корректора:

```
// Окошко перед записью
ЭТО вход                              // Состояние (вход)
  ВПРАВО                              // Смотрим следующий символ
  ЕСЛИ  ЦИФРА ТО целое
  ИНАЧЕ          ошибка
КОНЕЦ

ЭТО целое                             // Состояние (целое)
  ВПРАВО                              // Смотрим следующий символ
  ЕСЛИ       ПУСТО  ТО ответ_целое
  ИНАЧЕ 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_дробь
  ИНАЧЕ ЕСЛИ ЦИФРА ТО дробь
  ИНАЧЕ ошибка
КОНЕЦ

// Запишем ответ ОШ и вернёмся к месту ошибки
ЭТО ошибка
  ВПРАВО                               // Перемещение окна в конец записи
  ЕСЛИ ПУСТО
    ТО ответ_ошибка                    // и запись ответа.
    ИНАЧЕ ошибка
  ВЛЕВО                                // Возврат к месту ошибки.
КОНЕЦ
```

```
ЭТО ответ_ошибка
  ВПРАВО ПИШИ О
  ВПРАВО ПИШИ Ш
  ПОВТОРИ 2 ВЛЕВО
КОНЕЦ

ЭТО ответ_целое
  ВПРАВО ПИШИ Ц
  ВПРАВО ПИШИ Л
  ПОВТОРИ 2 ВЛЕВО
КОНЕЦ

ЭТО ответ_дробь
  ВПРАВО ПИШИ Д
  ВПРАВО ПИШИ Р
  ПОВТОРИ 2 ВЛЕВО
КОНЕЦ
```

19.2. Задачи

1. Задано определение:

Определение 2

```
<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 нее в виде двух букв (табл. 19.3).

Таблица 19.3

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 состояний среды после работы программы показаны на рис. 19.5.

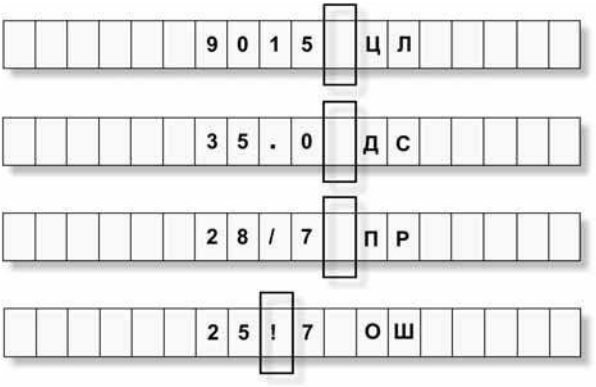


Рис. 19.5

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 состояний среды после работы программы показаны на рис. 19.6.

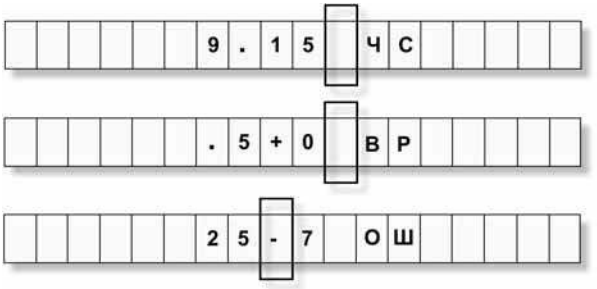


Рис. 19.6

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 состояний среды после работы программы показаны на рис. 19.7.

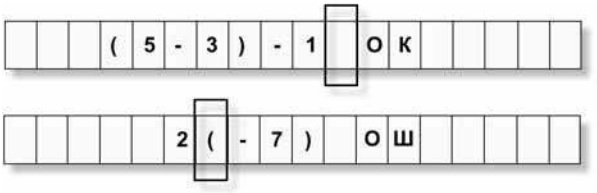


Рис. 19.7

Глава 20

Построение трансляторов

И всё доступно уж, эхма!
Теперь для нашего ума!

Н. Носов (стихи Цветика)

Дорогой читатель! Вы ещё с нами? Если это так, то усаживайтесь поудобнее и пристегните ремни: виражи будут крутые!

20.1. Интерпретаторы и компиляторы

Трансляторы разделяются на два класса: **интерпретаторы** и **компиляторы**.

Компилятор и исполнитель работают последовательно (сначала компилятор, потом исполнитель), а интерпретатор с исполнителем — совместно (интерпретатор анализирует конструкцию языка и тут же передаёт исполнителю соответствующие команды на выполнение).

Компилятор

Транслятор этого класса переводит программу с языка программирования в систему команд исполнителя. На этом работа компилятора заканчивается. Исполнитель выполняет последовательность команд, которую для него подготовил компилятор.

Интерпретатор

Транслятор этого класса совмещает разбор программы на языке программирования с передачей команд исполнителю.

20.2. План работы

Для начала построим простой интерпретатор для вычисления арифметического примера. Это для разминки. Никаких открытий, только здравый смысл и опыт прошлых занятий.

Затем, разогревшись, вступим в сферы перевода на польский (построение компилятора) и вычислений на стеке (построение интерпретатора).

Если вы ограничите своё чтение только «простым транслятором», мы не обидимся. Ну а если пойдёте за нами до конца, будем аплодировать!

20.3. Простой транслятор

Построим интерпретатор для вычисления простых примеров на сложение.

Задача 1

Задано определение:

Определение 1

```
<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 вычисления меньше числа, равного длине алфавита Корректора.

Примеры начальных и конечных состояний среды показаны на рис. 20.1–20.3.

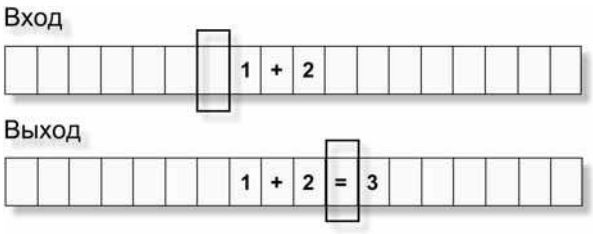


Рис. 20.1

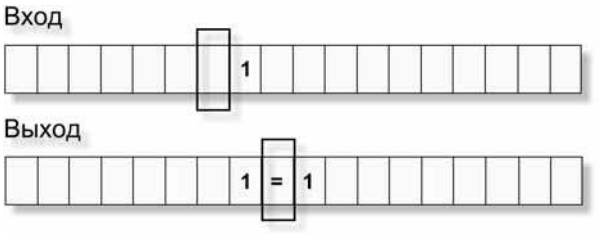


Рис. 20.2

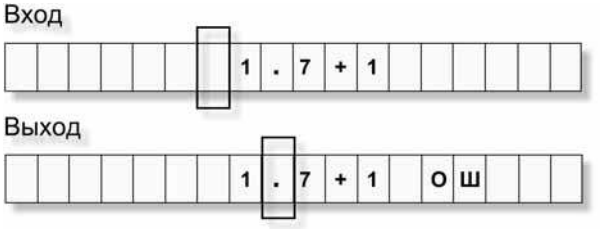


Рис. 20.3

Решение

Диаграмма переходов очень проста (рис. 20.4).

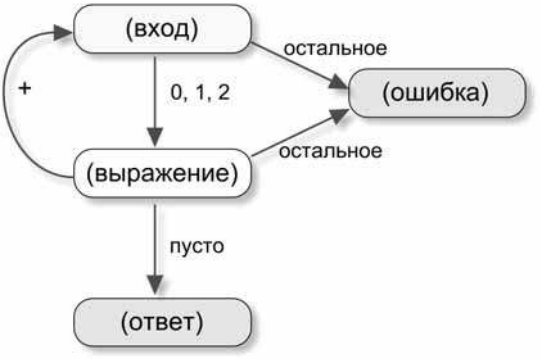


Рис. 20.4. Диаграмма переходов

Таблица переходов, соответствующая диаграмме (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)				

Однако 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.

```
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 добавить1      // без вычислений. Если 1 или 2, перед
  ИНАЧЕ ЕСЛИ 2 ТО добавить2      // переходом в состояние (выражение)
                                 // "добавим" число в ящик.
  ИНАЧЕ ошибка                   // Любой другой символ в состоянии (вход) -
КОНЕЦ                            // ошибка.

ЭТО выражение                    // Состояние (выражение).
  ВПРАВО                         // Смотрим следующий символ.
  ЕСЛИ       +     ТО вход       // Если это "+", переходим в состояние
                                 //(вход)- выражение продолжается.
  ИНАЧЕ ЕСЛИ ПУСТО ТО ответ      // Если это ПУСТО, выражение закончилось.
  ИНАЧЕ ошибка                   // Любой другой символ - ошибка.
КОНЕЦ
```

```
ЭТО добавить1                  // Вспомогательная переходная процедура
  ОБМЕН                        // между состояниями (вход) и (выражение).
  ПЛЮС                         // В ней к ящику "добавляется" 1.
  ОБМЕН                        // Конечно, команда ПЛЮС не арифметическая,
  выражение                    // она просто заменяет символ на следующий по
КОНЕЦ                          // алфавиту Корректора. Но, как увидим
                               // дальше, это действие совершенно
                               // эквивалентно операции "+1".

ЭТО добавить2                  // Вспомогательная переходная процедура
  ОБМЕН                        // между состояниями (вход) и (выражение).
  ПЛЮС ПЛЮС                    // В ней к ящику "добавляется" 2.
  ОБМЕН
  выражение
КОНЕЦ

ЭТО ошибка                     // Идём от ошибочного символа до конца
  ВПРАВО                       // записи, сжимая рекурсивную пружину.
  ЕСЛИ ПУСТО                   // Записываем в конце записи сообщение об
    ТО ответ_ошибка            // ошибке и под воздействием "энергии"
    ИНАЧЕ ошибка               // закрученных витков возвращаем окно на
  ВЛЕВО                        // место ошибки.
КОНЕЦ

ЭТО ответ_ошибка               // Сообщение об ошибке.
  ВПРАВО ПИШИ О
  ВПРАВО ПИШИ Ш
  ПОВТОРИ 2 ВЛЕВО              // Переместиться назад на 2 клетки необходимо
КОНЕЦ                          // для того, чтобы точно попасть "под
                               // пружину".

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
```

Вероятно, вас немного удивило замечание 2 в условии задачи («предполагается, что результат вычислений выражения меньше числа, равного длине алфавита Корректора»).

Теперь, конечно, понятна причина этого ограничения!

Ведь обычная операция «добавить один» подменяется в среде Корректора операцией **ПЛЮС**, которая совсем не арифметическая:

```
ПЛЮС (от ПУСТО) = 0
ПЛЮС (от 0) = 1
ПЛЮС (от 1) = 2
...
ПЛЮС (от 9) = А
ПЛЮС (от А) = Б
...
ПЛЮС (от последнего символа алфавита) = "Не могу"
```

Видим, что «сумматор» Корректора переполняется на вычислении:

ПЛЮС (от последнего символа алфавита)

Задача 2

Задано определение:

Определение 2

```
<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 начальных и конечных состояний среды показаны на рис. 20.5–20.7.

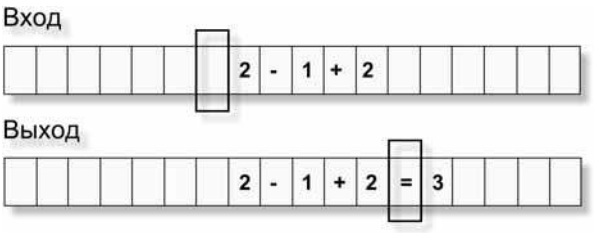


Рис. 20.5

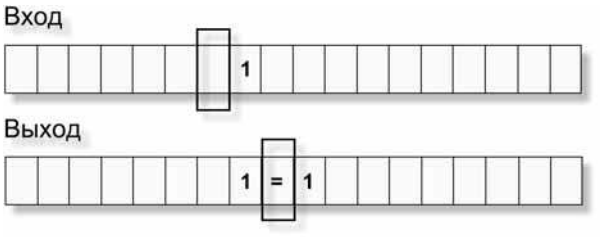


Рис. 20.6

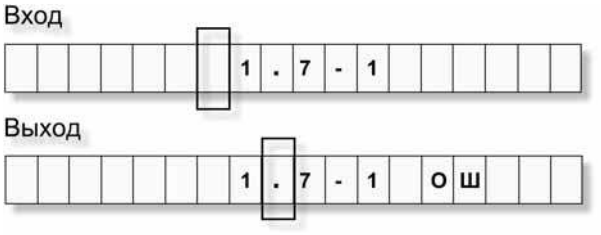


Рис. 20.7

20.4. Построение компилятора

Ну, а теперь рассмотрим по-настоящему сложную задачу. Она предлагалась в 1998 году на роботландском турнире юных программистов, и, представьте, нашлись ребята, которые её решили!

Это сильно огорчило куратора курса. Задача специально была подложена в надежде, что её никто не решит. Куратор думал, что будет повод для нравоучения: «Вот вам и простой Корректор! Слабо! Дело, ребята, не в языке программирования, а в знаниях методов программирования и опыте».

Куратора оконфузили ребята из волгоградской команды Владимира Анатольевича Петрова. Обычный максимальный балл в турнирных задачах — 10. Для этой задачи куратор вывел балл 15. И ребята его заработали!

Задача 3

Задано определение:

Определение 3

```
<выражение> ::= <число> |                                    (1)
                <выражение> - <выражение> |                    (2)
                <выражение> + <выражение> |                    (3)
                (<выражение>)                                  (4)
<число> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9          (5)
```

Написать программу, которая проверяет правильность записи выражения. Если запись правильная, то нужно за ней на ленте записать результат вычисления выражения, в противном случае — сообщение ОШ и установить окошко на первый неверный символ или на место обнаружения ошибки в случае ошибок со скобками.

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 вычисления выражения и все промежуточные результаты меньше числа, равного длине алфавита Корректора.

Замечание 3

Считается, что конечный и все промежуточные результаты вычислений неотрицательные.

Примеры начальных и конечных состояний среды показаны на рис. 20.8 и 20.9.

Лексический анализатор основан на диаграмме переходов из одного состояния в другие и дополнительном подсчёте баланса круглых скобок (на схеме (рис. 20.10) он не отражён).

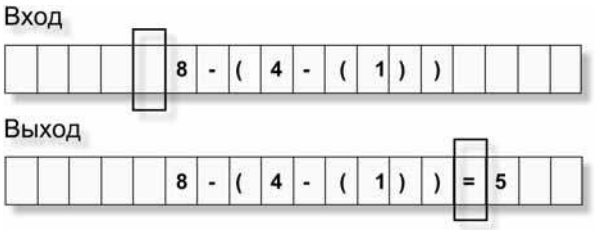


Рис. 20.8

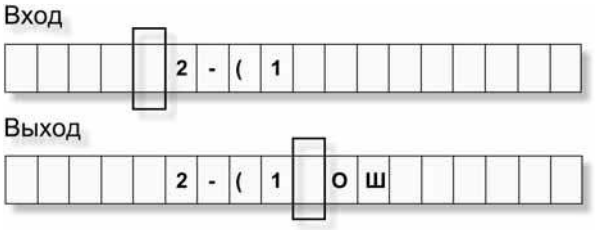


Рис. 20.9

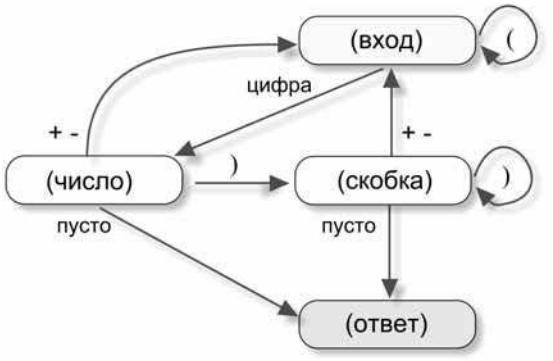


Рис. 20.10. Диаграмма переходов

На диаграмме не показаны переходы в состояние «ошибка», но они отражены в табл. 20.2 переходов, представленной ниже.

Таблица 20.2

Состояние	Следующий символ					
	цифра	**+ -**	**(**	**)**	**пусто**	**остальное**
(вход)	(число)	(ошибка)	(вход)	(ошибка)	(ошибка)	(ошибка)
(число)	(ошибка)	(вход)	(ошибка)	(скобка)	(ответ)	(ошибка)
(скобка)	(ошибка)	(вход)	(ошибка)	(скобка)	(ответ)	(ошибка)
(ответ)						
(ошибка)						

Для слежения за балансом круглых скобок можно использовать ящик Корректора (изначально пустой). На каждой открывающей скобке будем применять к символу в ящике операцию `плюс`, а на каждой закрывающей — `минус`. Если после просмотра всего выражения ящик не будет пуст, это ошибка — баланс скобок в записи нарушен.

Итак, диаграмма состояний будет «следить» за правомочностью появления круглой скобки в данном месте записи, а ящик — за балансом скобок во всей записи в целом.

Основываясь на высказанных ранее идеях, записать программу анализатора достаточно легко.

Но сейчас нас больше интересует не лексический анализ выражения, а его вычисление. Вот уж здесь действительно скачок сложности сразу на несколько порядков!

Первый вариант решения (который, кстати, и реализовали ребята Владимира Анатольевича) выглядит так. Ищем в записи пару круглых скобок `(...)`,

таких, которые внутри себя скобок уже не содержат. Далее вычисляем выражение внутри этих скобок и вставляем результат на ленту, заменив им всю конструкцию `(...)`. Такие действия нужно повторять до тех пор, пока в исходном выражении скобок уже не останется.

Алгоритмы, которые просматривают запись много раз, называются многопроходными. Описанный выше алгоритм *очень* многопроходный, а значит, неэффективный.

В информатике используют другой метод:

1. Переводят исходное выражение в обратную польскую запись (компиляция исходной записи в польскую).
2. Вычисляют польскую запись на стеке (интерпретация польской записи как программы для вычисления выражения).

Этот алгоритм популярен по следующей причине: если один раз перевести выражение в польскую (бесскобочную) запись, то в дальнейшем можно многократно производить по нему быстрые (однопроходные) вычисления.

Запись выражения по-польски и стековые вычисления

Самой первой темой в курсе роботландского университета «Азы программирования» традиционно обозначен блок, связанный с исполнителем Плюсик.

Плюсик — это модель стекового калькулятора. Среда исполнителя состоит из стека и вычислителя (арифметического устройства), соединённых между собой информационным каналом связи (рис. 20.11).

Рис. 20.11. Среда Плюсика

Стек служит для хранения чисел. Числа помещаются в стек по одному, подобно тому, как кольца детской пирамидки нанизываются на стержень. За один раз взять из стека можно только одно число с вершины числовой пирамиды. Копия числа в стеке не сохраняется, зато становится доступным следующее число, расположенное «ниже» взятого.

Система команд Плюсика

В табл. 20.3 представлена система команд Плюсика.

Таблица 20.3

Команда	Описание
ЗАПОМНИ число	Число, записанное в команде, отправляется на хранение в стек
СЛОЖИ	Все вычислительные команды работают одинаково. Из стека последовательно извлекаются два числа и отправляются вычислителю для выполнения соответствующей арифметической операции. Результат помещается в стек. Число, извлечённое из стека вторым, считается первым операндом операции. Отказ возникает когда: в стеке меньше двух чисел; результат вычислений — отрицательное число; выполняется деление на 0. Команда ДЕЛИ выполняет деление нацело
ВЫЧТИ	
УМНОЖЬ	
ДЕЛИ	
ОЧИСТИ	Удаление всех чисел из стека

Вычислить выражение типа `8 - (2 + 3)` — задача не из лёгких! Практически все трансляторы сначала переводят его в обратную польскую запись, а уж потом, подобно Плюсику, вычисляют на стеке.

Просим прощения! Мы не обсудили ещё устройство обратной польской записи!

Всё очень просто! В обычной записи знак операции располагается между операндами, а в обратной польской — после. Вот и всё!

Обычная запись операции: `a + b`

Обратная польская запись: `a b +`

Обычная запись выражения: `a + b * (c + d / e) / f`

Обратная польская запись: `a b c d e / + * f / +`

Замечание

Запись типа `+ab` выражения `a + b` называется польской. Она впервые была введена польским философом Лукашевичем в связи с формулами символической логики. Соответственно запись типа `ab+` для суммы двух чисел называется обратной польской записью.

Иными словами прямая польская запись базируется на структуре:

```
<операция> <операнд> <операнд>
```

А обратная польская запись (которая нас интересует) на структуре:

```
<операнд> <операнд> <операция>
```

В некоторых источниках обратная польская запись называется постфиксной, а прямая — префиксной. Этимология таких названий совершенно прозрачна.

У. Маккиман, Дж. Хорнинг, Д. Уортман в книге «Генератор компиляторов» называют обычную запись выражений — инфиксной записью, а обратную польскую — суффиксной.

Сравните:

- обычная запись выражения: `2 + 5 * (1 + 3 / 2) / 7`
- обратная польская запись выражения: `2 5 1 3 2 / + * 7 / +`
- программа для Плюсика:

```
ЗАПОМНИ 2
ЗАПОМНИ 5
ЗАПОМНИ 1
ЗАПОМНИ 3
ЗАПОМНИ 2
ДЕЛИ
СЛОЖИ
УМНОЖЬ
ЗАПОМНИ 7
ДЕЛИ
СЛОЖИ
```

Видим, что программа для Плюсика практически является другой формой обратной польской записи выражения. Верно и обратное утверждение: обратная польская запись является программой для вычисления выражения на стеке.

Исполнитель, который выполняет эту программу, должен работать совершенно так же, как Плюсик. Такие исполнители содержатся практически в любом трансляторе.

Одним словом, для вычисления выражения в обратной польской записи нужно построить Плюсик.

А этот исполнитель предельно прост. Выражение типа `2 5 1 3 2 / + * 7 / +` он рассматривает как запись программы работы со стеком и вычислителем.

Алгоритм работы Плюсика.

1. Очистить стек.
2. Продолжать, пока запись-программа не закончится:
   2.1. Если очередной элемент записи — число, поместить его в стек и перейти к следующему элементу записи.
   2.2. Если очередной элемент записи — операция, выполнить её следующим образом:
      2.2.1. Извлечь из стека два числа и выполнить над ними операцию, считая при этом последнее извлечённое число первым операндом.
      2.2.2. Результат операции поместить в стек.

После завершения программы в стеке останется одно число — результат вычислений.

Пошаговое выполнение программы `2 5 1 3 2 / + * 7 / +` показано на рис. 20.12.

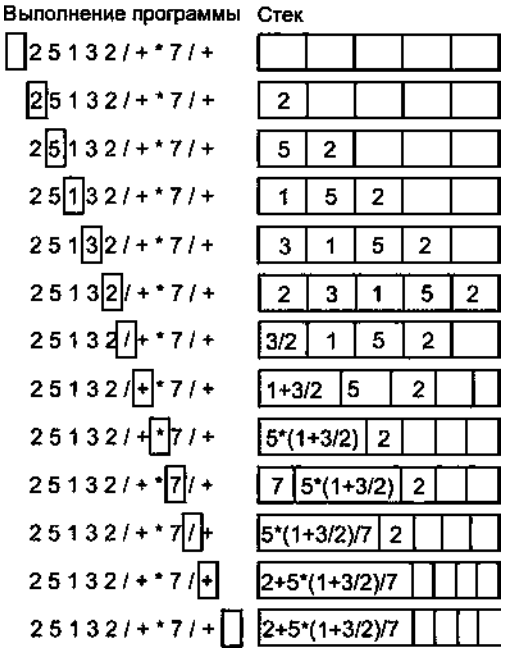


Рис. 20.12

Итак, если выражение записано «по-польски», вычислить его совсем не трудно. Даже Корректором.

А как перевести обычную запись в польскую?

Для Плюсика даются следующие рекомендации для построения программ (читай — для компиляции обычной записи в польскую):

- для преобразования обычного выражения в программу для Плюсика нужно просматривать выражение последовательно слева направо, записывая команды **ЗАПОМНИ** `число`;
- если после очередной записи команды **ЗАПОМНИ** выясняется, что над последними двумя числами в стеке можно выполнить арифметическую операцию, записывается соответствующая команда.

Очень хорошая рекомендация! Но для человека. Как проверить условие «можно выполнить арифметическую операцию»?

Это непросто! Особенно мешают круглые скобки. Тем более что вложение их друг в друга носит рекурсивный характер.

Посмотрите на выражение:

```
7 - (1 + 3)
```

Как перевести его в польскую запись?

Сложность в том, что исполнитель не видит запись целиком! В каждый момент ему виден только один символ (который в нашей задаче одновременно является лексемой).

Следовательно, нужно придумать, как, перемещаясь от символа к символу, сформировать обратную польскую запись выражения, не видя его целиком.

Вы правы, это невозможно без хитрости! Нужно дополнительное средство, которое расширит «кругозор» исполнителя за пределы одного символа и позволит осуществить отложенные действия.

Конечно, первый символ записи «`7`» можно сразу поместить в результирующую строку. Следующий символ «`-`» — уже нельзя. Его нужно где-то временно запомнить, чтобы потом в нужный момент вставить в польскую запись.

Когда же наступит подходящий момент для знака «`-`»? Тогда, когда за первым операндом «`7`» будет переведен в польскую запись второй операнд операции «`-`». В нашем случае таким операндом будет польская запись выражения «`(1+3)`» (рис. 20.13).

Теперь на входе символ «`(`». Что с ним делать? Придется его тоже запомнить и хранить до тех пор, пока не встретится парный знак «`)`». После этого знак «`(`» из хранилища можно убрать (рис. 20.14).

Объект «`1`» помещаем в выходную польскую запись, а знак «`+`» запоминаем до «лучших» времен (рис. 20.15).

Рис. 20.13

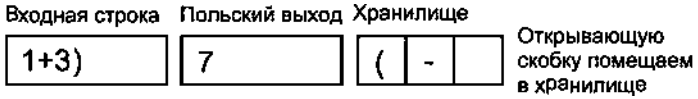


Рис. 20.14

Рис. 20.15

Рис. 20.16

Продолжая действовать в том же духе, получим польскую запись выражения (рис. 20.16).

Наши действия над «хранилищем» наглядно показывают, что память эта является стеком.

Второй вывод, который можно сделать из наших опытов, позволит написать формальный алгоритм преобразования обычного выражения в польскую запись.

Вывод такой: символы перемещались из входной строки в выходную или в стек, а из стека в выходную строку в зависимости от их приоритетов.

Понятие приоритета можно уточнить, если ввести числовые характеристики, соответствующие проделанным опытам (табл. 20.4).

Таблица 20.4

Объект	Приоритет в стеке	Приоритет во входной строке
+, -	2	1
число	4	3
(	0	5
)		0

Замечание 1

Скобку «)» никогда не помещаем в стек.

Замечание 2

Приоритет открывающей скобки в стеке равен приоритету закрывающей скобки во входной строке, чтобы по формальному признаку равенства приоритетов открывающую скобку из стека можно было убрать тогда, когда парная к ней закрывающая скобка появится во входной строке.

Теперь алгоритм преобразования можно изобразить следующим образом (рис. 20.17).

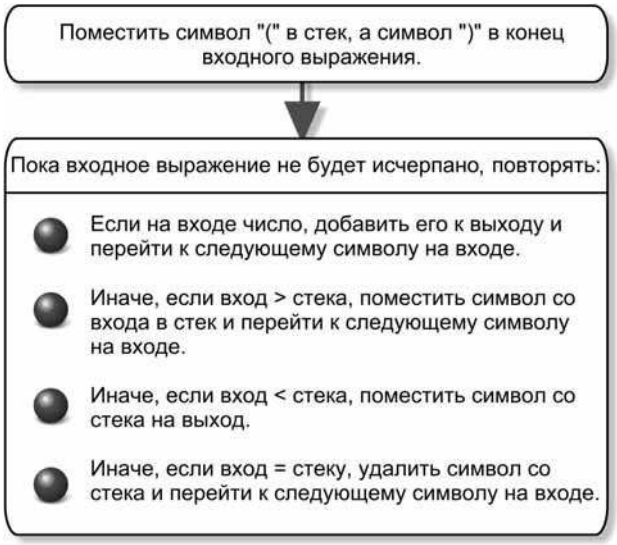


Рис. 20.17

Инициализация перед входом в цикл введена для единообразия действий внутри цикла (когда не надо обрабатывать отдельно пустой стек и пустую входную строку).

Заметим также, что без первой ветви переключателя в цикле вполне можно обойтись (правда ценой лишних действий исполнителя алгоритма): число в выходную строку может передаваться не сразу, а единообразно через стек (рис. 20.18).

Рис. 20.18

Прокрутим по этому алгоритму выражение `7 - (1 + 3)` (табл. 20.5).

Таблица 20.5

Шаг	Состояние входа, выхода и стека	Проверка приоритетов и выбор действия
1	вход: 7 - (1 + 3)) выход: стек: (	(вход = 3) > (стек = 0) Помещаем число в стек
2	вход: - (1 + 3)) выход: стек: 7 (	(вход = 1) < (стек = 4) Помещаем число на выход
3	вход: - (1 + 3)) выход: 7 стек: (	(вход = 1) > (стек = 0) Помещаем операцию в стек

Таблица 20.5 *(окончание)*

Шаг	Состояние входа, выхода и стека	Проверка приоритетов и выбор действия
4	вход: (1 + 3)) выход: 7 стек: - (	(вход = 5) > (стек = 2) Помещаем скобку в стек
5	вход: 1 + 3)) выход: 7 стек: (- (	(вход = 3) > (стек = 0) Помещаем число в стек
6	вход: + 3)) выход: 7 стек: 1 (- (	(вход = 1) < (стек = 4) Помещаем число на выход
7	вход: + 3)) выход: 7 1 стек: (- (	(вход = 1) > (стек = 0) Помещаем операцию в стек
8	вход: 3)) выход: 7 1 стек: + (- (	(вход = 3) > (стек = 2) Помещаем число в стек
9	вход:)) выход: 7 1 стек: 3 + (- (	(вход = 0) < (стек = 4) Помещаем число на выход
10	вход:)) выход: 7 1 3 стек: + (- (	(вход = 0) < (стек = 2) Помещаем операцию на выход
11	вход:)) выход: 7 1 3 + стек: (- (	(вход = 0) = (стек = 0) Удаляем скобку из стека
12	вход:) выход: 7 1 3 + стек: - (	(вход = 0) < (стек = 2) Помещаем операцию на выход
13	вход:) выход: 7 1 3 + - стек: (	(вход = 0) = (стек = 0) Удаляем скобку из стека
14	вход: выход: 7 1 3 + - стек:	Выполнение закончено

Замечание

Если бы кроме операций «+» и «–» в записи присутствовали бы более «старшие» операции «*» и «/», то алгоритм преобразования остался бы прежним, а вот таблицу приоритетов пришлось бы изменить (табл. 20.6).

Таблица 20.6

Объект	Приоритет в стеке	Приоритет во входной строке
+, -	2	1
*, /	4	3
число	6	5
(	0	7
)		0

Программа для Корректора

Решение представим в виде 4 последовательных независимых друг от друга шагов.

1. Проверка соответствия записи определению.
2. Преобразование выражения в обратную польскую запись.
3. Вычисление польской записи на стеке.
4. Запись результата на ленту в виде обычного десятичного числа.

Соответственно главная процедура программы будет иметь вид:

```
ЭТО выражение
  проверка          // Процедура "проверка" запишет в ящик Корректора
  ОБМЕН             // символ "#" как признак того, что ошибок в записи
  ЕСЛИ # ТО         // нет.
  {
    ОБМЕН           // Продолжаем работу только в том случае,
    преобразование  // если ошибок в записи не было.
    вычисление
    результат
  }
  ИНАЧЕ ОБМЕН
КОНЕЦ
```

1. Проверка соответствия записи определению

```
// Окошко установлено на первый символ ПУСТО перед записью.
// В случае верной записи, окошко будет поставлено здесь же при
// выходе из процедуры "проверка".
ЭТО проверка
  ОБМЕН ПИШИ ПУСТО ОБМЕН  // Запись символа ПУСТО в ящик и переход
  вход                    // в состояние (вход).
КОНЕЦ

ЭТО вход
  ВПРАВО
  ЕСЛИ       ЦИФРА ТО число
  ИНАЧЕ ЕСЛИ ( ТО вход1
  ИНАЧЕ ошибка
КОНЕЦ

ЭТО вход1
  ОБМЕН ПЛЮС ОБМЕН
  вход
КОНЕЦ

ЭТО скобка)
  ВПРАВО
  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)
    ИНАЧЕ ошибка
  }
КОНЕЦ

ЭТО число
```

```
  ВПРАВО
  ЕСЛИ       ПУСТО   ТО ответ
  ИНАЧЕ ЕСЛИ +       ТО вход
  ИНАЧЕ ЕСЛИ -       ТО вход
  ИНАЧЕ ЕСЛИ )       ТО скобка)
  ИНАЧЕ ошибка
КОНЕЦ

ЭТО ответ
  ВПРАВО
  ОБМЕН
  ЕСЛИ НЕ ПУСТО ТО { ОБМЕН ошибка }
  ИНАЧЕ
  {
    ПИШИ #               // Запишем в ящик символ "#" - признак
    ОБМЕН                // успешной проверки.
    ПИШИ ПУСТО           // Переместим окошко в исходное состояние -
                         // на первый символ ПУСТО перед записью.
    ПОКА ПУСТО ВЛЕВО     // Найдём конец записи.
    ПОКА НЕ ПУСТО ВЛЕВО // А теперь найдём начало.
  }
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
```

2. Преобразование выражения в обратную польскую запись

```
ЭТО преобразование
  подготовка          // Инициализации перед входом в цикл.
  работа              // Цикл.
КОНЕЦ

ЭТО подготовка
  подготовка1         // В конец записи дописываем ")"
                      // и формируем стек с элементом "(".

  ПИШИ #              //  Начало выходной (пока пустой) записи.

  ВЛЕВО
  ПОКА НЕ # ВЛЕВО     // Вернёмся к началу входной записи и
  ВЛЕВО ВЛЕВО         // установим окошко на первый символ.
  ПОКА НЕ ПУСТО ВЛЕВО
  ВПРАВО
КОНЕЦ

ЭТО подготовка1       // Идём в конец входной записи
  ВПРАВО              // и "считаем" её длину.
  ЕСЛИ НЕ ПУСТО
    ТО    подготовка1
    ИНАЧЕ подготовка2
  ВПРАВО              // Рекурсивная пружинка отодвинет вправо
КОНЕЦ                 // от стека на длину входной записи.

ЭТО подготовка2
  ПИШИ )                 // Припишем к концу записи скобку ")".
  ПОВТОРИ 2 ВПРАВО       // Создадим стек и запишем в него
  ПИШИ #  ВПРАВО ПИШИ (  // скобку "(".
  ВПРАВО
КОНЕЦ

ЭТО работа            // Основной цикл алгоритма преобразования.
  ПОКА НЕ ПУСТО       // Цикл работает, пока не кончатся символы
  {                   // во входной записи.
```

```
    ЯЩИК+               // Запомним символ со входа и сравним его
                        // приоритет с символом на стеке.
    один_такт           // Результат сравнения поместим в ящик:
                        // > — вход > стек, нужно символ занести в стек;
                        // = — вход = стек, нужно символ убрать из стека;
                        // < — вход < стек, символ из стека на выход.
                        // В процедуре "один_такт" вся необходимая
                        // работа совмещена с проверкой.

    ОБМЕН               // Посмотрим, что в ящике.

    ЕСЛИ > ТО
    {                   // вход > стека, входной символ занесён в стек
      ОБМЕН             // в процедуре один_такт
      ВПРАВО
    }

    ИНАЧЕ ЕСЛИ = ТО     // вход = стек, стирание символа из стека
    {                   // было выполнено в процедуре "один_такт".
      ОБМЕН
      ВПРАВО
    }
    ИНАЧЕ ОБМЕН         // вход < стек, символ из стека был перенесён
  }                     // на выход в процедуре "один_такт"
КОНЕЦ

ЭТО один_такт
  ВПРАВО                // Идём вправо до начала стека (символ "#"),
  ЕСЛИ НЕ # ТО один_такт // сжимая рекурсивную пружину для возврата
  ИНАЧЕ проверить       // на текущую позицию во входной строке
  ВЛЕВО
КОНЕЦ

ЭТО проверить           // Сейчас окошко на символе "#" (начало стека).
  ПОКА НЕ ПУСТО ВПРАВО  // Устанавливаем окошко на верхушку
  ВЛЕВО                 // стека.
  сравнение             // Сравниваем символ в ящике (текущий символ из
```

```
                          // входной строки) с символом в верхушке стека
                          // и формируем результат на ленте в виде:
                          // > — вход > стек
                          // = — вход = стек
                          // < — вход < стек
                               //  Что дала проверка:
  ЕСЛИ       = ТО стек_равен  // вход = стек - убираем верхушку,
                               // "=" помещаем в ящик (в стек_равен)
  ИНАЧЕ ЕСЛИ < ТО стек_больше // вход < стек - верхушку стека на выход,
                               // признак "<" в ящик (в стек_больше)
  ИНАЧЕ ОБМЕН                  // вход > стек - символ со входа в стек,
                               // признак ">" помещаем в ящик (ОБМЕН)
  ПОКА НЕ # ВЛЕВО              // Выводим окошко на начало стека (символ "#"),
КОНЕЦ                          // чтобы рекурсивная пружина в процедуре
                               // один_такт вернула нас точно на место
                               // текущего символа во входной строке.

ЭТО сравнение                 // Сравниваем символ в ящике (текущий символ
                              // из входной строки) с символом в верхушке
                              // стека и записываем результат проверки на ленту.
  ЕСЛИ       -     ТО сравнение_со_знаком
  ИНАЧЕ ЕСЛИ +     ТО сравнение_со_знаком
  ИНАЧЕ ЕСЛИ ЦИФРА ТО сравнение_с_цифрой
  ИНАЧЕ ЕСЛИ (     ТО сравнение_со_скобкой
КОНЕЦ

ЭТО сравнение_со_знаком          // В верхушке стека - знак операции.
  ОБМЕН                          // Приоритет этого символа = 2. Формируем
  ЕСЛИ ЦИФРА ТО   символ_в_стек  // результат проверки по таблице:
  ИНАЧЕ ЕСЛИ ( ТО символ_в_стек  // вх.симв. приоритет условие      рез.
  ИНАЧЕ ПИШИ <                   // +, -        1      вход < стека  2
КОНЕЦ                            // число       3      вход > стека  0
                                 // (           5      вход > стека  0
                                 // )           0      вход < стека  2

ЭТО сравнение_с_цифрой          // В верхушке стека - цифра.
  ОБМЕН                          // Приоритет этого символа = 4. Формируем
```

```
  ЕСЛИ (                    // результат проверки по следующей таблице:
    ТО символ_в_стек        // вх.симв. приоритет условие      рез.
    ИНАЧЕ ПИШИ <            // +, -        1      вход < стека  <
КОНЕЦ                       // число       3      вход < стека  <
                            // (           5      вход > стека  >
                            // )           0      вход < стека  <

ЭТО сравнение_со_скобкой    // В верхушке стека открывающаяся скобка
  ОБМЕН                     //.Приоритет этого символа = 0. Формируем
  ЕСЛИ )                    // результат проверки по следующей таблице:
    ТО  ПИШИ =              // вх.симв. приоритет условие      рез.
    ИНАЧЕ символ_в_стек     // +, -        1      вход > стека  >
КОНЕЦ                       // число       3      вход > стека  >
                            // (           5      вход > стека  >
                            // )           0      вход = стеку  =

ЭТО стек_равен                 // Удаление верхушки стека для случая,
  ОБМЕН                        // когда вход = стек
  ПИШИ ПУСТО
КОНЕЦ

ЭТО стек_больше             // Перенос символа из стека на выход
                            // в случае, когда вход < стек.
                            // Символ из верхушки стека сейчас
                            // в ящике, а в верхушке стека -
                            // результат проверки - "<"

  ПОКА НЕ # ВПРАВО          // Идём на начало выходной записи.
  ПОКА НЕ ПУСТО ВПРАВО      // Идём на конец выходной записи.

  ОБМЕН                     // Символ из ящика помещаем в конец
                            // выходной записи, а пустой символ
                            // с ленты - в ящик.

  ПОКА НЕ # ВЛЕВО           // Идём на начало выходной записи.
  ВЛЕВО
  ПОКА ПУСТО ВЛЕВО          // Идём на конец стека.
```

```
  ОБМЕН                          // Затираем верхушку стека, а в ящик
                                 // помещаем результат проверки (этот
КОНЕЦ                            // результат временно хранился в верхушке).

ЭТО символ_в_стек
  ОБМЕН
  ВПРАВО ЯЩИК- ВЛЕВО
  ОБМЕН
  ПИШИ > // Признак: вход > стека
КОНЕЦ
```

3. Вычисление польской записи на стеке

Сейчас на ленте сложилась следующая обстановка (рис. 20.19).

Рис. 20.19

Окошко смотрит на первый пустой символ за исходной записью, стек пуст, а на выходе (сразу за вторым знаком «#») обратная польская запись исходного выражения.

Для вычисления польской записи будем использовать стек.

Арифметические вычисления выполним командами **ПЛЮС** и **МИНУС**. Конечно, в итоге мы получим не десятичное число, а некоторый символ из алфавита Корректора (по условию результат не превышает длины алфавита, т. е. символов Корректора нам хватит для изображения ответа). Преобразованием ответа-символа в обычное десятичное число займётся специальная процедура `результат`.

```
ЭТО вычисление
  на_польскую_запись       // Переместим окошко на начало
                           // польской записи.
  ПОКА НЕ ПУСТО            // Выполнять вычисления на стеке, пока
  {                        // символы в польской записи не кончатся.
    ЯЩИК+                  // Очередной символ помещаем в ящик.
    ЕСЛИ ЦИФРА ТО в_стек   // Если это - число, заносим в стек.
    ИНАЧЕ         операция // Если - операция, выполняем на стеке.
```

```
    ВПРАВО                  // Сдвиг окошка к следующему символу
  }                         //
  ПОКА НЕ # ВЛЕВО           // Перемещаем окно на начало стека
  ВЛЕВО                     // (символ "#"). В стеке сейчас один
  ПОКА НЕ # ВЛЕВО           // символ - результат вычислений.
КОНЕЦ

ЭТО на_польскую_запись      // Перемещение на начало польской записи.
  ВЛЕВО
  ПИШИ ПУСТО
  ПОКА НЕ # ВПРАВО
  ВПРАВО
  ПОКА НЕ # ВПРАВО
  ВПРАВО
КОНЕЦ

ЭТО в_стек                  // Занести число в верхушку стека
  ВЛЕВО                     // и вернуться к текущему символу
  ЕСЛИ НЕ # ТО в_стек       // в польской записи.
  ИНАЧЕ        в_стек1      // Для начала выходим на начало польской
  ВПРАВО                    // записи, сжимая рекурсивную пружину.
КОНЕЦ

ЭТО в_стек1
  ВЛЕВО                 // Теперь устанавливаем окно на первый
  ПОКА ПУСТО ВЛЕВО      // пустой символ за стеком и записываем
  ВПРАВО                // в него число (оно в ящике).
  ЯЩИК-
  ПОКА НЕ # ВПРАВО      // Возврат окна на начало польской записи.
КОНЕЦ

ЭТО операция              // Выполнить операцию на стеке и вернуться
  ВЛЕВО                   // к текущему символу в польской записи.
  ЕСЛИ НЕ # ТО операция
  ИНАЧЕ        операция1  // Для начала выходим на начало польской
```

```
  ВПРАВО                  // записи, сжимая рекурсивную пружину.
КОНЕЦ

ЭТО операция1             // Теперь устанавливаем окно на первый
  ВЛЕВО                   // пустой символ за стеком и переключаемся
  ПОКА ПУСТО ВЛЕВО        // по значению операции (знак операции
  ВЛЕВО                   // в ящике).
  ОБМЕН
  ЕСЛИ + ТО сложить
  ИНАЧЕ      вычесть      // Выполнение операции сводится к замене
  ПИШИ ПУСТО              // двух символов в верхушке стека одним -
  ВЛЕВО                   // результатом выполнения операции.
  ЯЩИК-
  ПОКА НЕ # ВПРАВО        // Возврат окна на начало польской записи.
КОНЕЦ

ЭТО сложить               // Операция сложения.
  ОБМЕН
  ЯЩИК+                   // Алгоритм:
  ВПРАВО                  // К первому слагаемому прибавляем 1, а из
  ПОКА НЕ 0               // второго вычитаем 1, до тех пор, пока
  {                       // второе слагаемое не станет нулем.
    ОБМЕН ПЛЮС ОБМЕН МИНУС
  }
КОНЕЦ

ЭТО вычесть               // Операция вычитания.
  ОБМЕН
  ЯЩИК+                   // Алгоритм:
  ВПРАВО                  // От уменьшаемого и вычитаемого отнимаем
  ПОКА НЕ 0               // по единице, пока вычитаемое не станет
  {                       // нулем.
    ОБМЕН МИНУС ОБМЕН МИНУС
  }
КОНЕЦ
```

4. Запись результата на ленту в виде обычного десятичного числа

Ситуация на ленте сейчас такая (рис. 20.20).

Рис. 20.20

В стеке записан символ (на рис. 20.20 он обозначен как «n»). Этот символ — результат вычислений — нужно преобразовать в обычное десятичное число.

```
ЭТО результат
  ПИШИ ПУСТО           // Стираем "#" у стека - он больше не понадобится.
  ВПРАВО               // Окошко смещаем на символ-результат.
  Символ_в_число       // Переводим символ в число
  ПОКА НЕ ПУСТО ВЛЕВО // Записываем перед результатом
  ПИШИ =               // знак "=".
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
```

Ссылки на задачи

Ниже представлена группировка задач по темам, изложенным в книге «Азы программирования. Магия для начинающих». Присутствуют ссылки только на задачи, предназначенные для самостоятельного решения. Названия задач роботландских конкурсов из книги «Азы программирования. Задачи роботландских турниров» (в табл. 1–3 ссылки на эту книгу обозначены ссылкой «Задачник») сопровождаются указанием сложности в баллах.

Часть I. Кукарача

Таблица 1

Номер по порядку	Название задачи	Ссылка на тему книги «Азы программирования. Магия для начинающих»	Адрес задачи	Страница
1.	Молоток в моток	I.1	I.1.3.1	26
2.	Мишку в мышку	I.1	I.1.3.2	27
3.	Починить колесо	I.1	I.1.3.3	27
4.	Роль Кукарачи	I.1	I.1.3.4	27
5.	Имя знакомой кошки	I.1	I.1.3.5	27
6.	Что кричит кукушка	I.2	I.2.5.1	36
7.	Домашнее животное	I.2	I.2.5.2	36
8.	Из пяти ног одно пятно	I.2	I.2.5.3	36
9.	Кому поет Кукарача	I.2	I.2.5.4	37
10.	Исправление примера	I.2	I.2.5.5	37

Таблица 1 *(продолжение)*

Номер по порядку	Название задачи	Ссылка на тему книги «Азы программирования. Магия для начинающих»	Адрес задачи	Страница
11.	Лесенка	I.3	I.3.5.1	52
12.	Прогулка по маршруту	I.3	I.3.5.2	53
13.	Контроль поля	I.3	I.3.5.3	53
14.	Пароход	I.3	I.3.5.4	53
15.	Колесо	I.3	I.3.5.5	53
16.	Машинист	I.3	I.3.5.6	54
17.	Очистить поле от мусора	I.3	I.3.5.7	54
18.	Прогулка	I.3	I.3.5.8	54
19.	Провернувшийся циферблат	I.3	I.7.14	104
20.	Цифровая головоломка	I.3	I.7.15	104
21.	Опять пятёрка	I.3	I.7.16	105
22.	Крот	I.3	I.7.17	105
23.	Квадрат	I.3	I.7.18	106
24.	Прогулка (3 балла)	I.3	Задачник 2.1.1	31
25.	Кубик в прямоугольнике (4 балла)	I.3	Задачник 3.1.1	46
26.	Три мешка (3 балла)	I.3	Задачник 8.1.1	125
27.	Разнеси подарки (3 балла)	I.3	Задачник 8.1.2	126
28.	Квадратик (3 балла)	I.3	Задачник 9.1	140
29.	Восстановление связи (2 балла)	I.3	Задачник 9.2	140
30.	Хитрый гость (5 баллов)	I.3	Задачник 9.3	141
31.	Крот в отпуске (4 балла)	I.3	Задачник 9.4	141
32.	Посадить дерево	I.4	I.4.1.2	60
33.	Игра	I.4	I.4.3.1	67

Таблица 1 *(продолжение)*

Номер по порядку	Название задачи	Ссылка на тему книги «Азы программирования. Магия для начинающих»	Адрес задачи	Страница
34.	Крестики-нолики	I.4	I.4.3.2	67
35.	Космическое путешествие	I.4	I.4.3.3	68
36.	Клад	I.4	I.4.3.4	68
37.	Медведь	I.4	I.4.3.5	68
38.	Кот	I.4	I.4.3.6	69
39.	Экран	I.4	I.4.3.7	69
40.	Найти букву и встать на её место	I.4	I.4.3.8	69
41.	Кук	I.4	I.7.1	98
42.	Больной кот	I.4	I.7.2	98
43.	Шумы на линии	I.4	I.7.3	98
44.	Чаепитие в Кукарачинске	I.4	I.7.4	100
45.	Новогодние подарки	I.4	I.7.5	100
46.	Экран на прежнем месте	I.4	I.7.6	100
47.	Значение цифры (3 балла)	I.4	Задачник 1.7	26
48.	Умножение цифры на 11 (4 балла)	I.4	Задачник 2.1.6	34
49.	Кукарача — перевозчик (8 баллов)	I.4	Задачник 6.1.7	103
50.	Кукарача — цветовод (6 баллов)	I.4	Задачник 9.5	141
51.	Цифры по порядку (5 баллов)	I.4	Задачник 9.6	142
52.	Мяч в корзину	I.5	I.5.3.1	79
53.	Нарастить стену	I.5	I.5.3.2	79

Таблица 1 *(продолжение)*

Номер по порядку	Название задачи	Ссылка на тему книги «Азы программирования. Магия для начинающих»	Адрес задачи	Страница
54.	Провести по прямоугольнику А-Б-В-Г	I.5	I.5.3.3	80
55.	Ишак	I.5	I.5.3.4	80
56.	Мышь	I.5	I.5.3.5	80
57.	Упорядочить цифры	I.5	I.5.3.6	80
58.	Горка	I.5	I.7.7	101
59.	Какие оценки нужны Кукараче	I.5	I.7.8	101
60.	Последний станет первым	I.5	I.7.9	102
61.	Грибы	I.5	I.7.10	102
62.	Грибы-2	I.5	I.7.11	102
63.	Камень, ножницы, бумага	I.5	I.7.19	106
64.	Белая мышка	I.5	I.7.20	106
65.	Уа-Ау	I.5	I.7.21	107
66.	Новогодняя сказка	I.5	I.7.22	108
67.	Машина времени	I.5	I.7.23	108
68.	Зелёный ряд	I.5	I.7.24	108
69.	Пример	I.5	I.7.25	110
70.	Торт	I.5	I.7.26	110
71.	Расставь цифры (4 балла)	I.5	Задачник 2.1.2	32
72.	Сложение двух палочных чисел (5 баллов)	I.5	Задачник 2.1.3	32
73.	Баскетбол (4 балла)	I.5	Задачник 2.1.8	35
74.	Умножение на 10	I.5	Задачник 2.1.9	35
75.	Ёлка (3 балла)	I.5	Задачник 6.1.1	101

Таблица 1 *(продолжение)*

Номер по порядку	Название задачи	Ссылка на тему книги «Азы программирования. Магия для начинающих»	Адрес задачи	Страница
76.	Точка (4 балла)	I.5	Задачник 6.1.2	101
77.	Стыковка-1 (4 балла)	I.5	Задачник 7.1.1	113
78.	Первый и последний (4 балла)	I.5	Задачник 8.1.3	126
79.	Красные и синие (8 баллов)	I.5	Задачник 9.7	142
80.	Репка (7 баллов)	I.5	Задачник 9.8	143
81.	Лабиринт (5 баллов)	I.5	Задачник 9.9	144
82.	Тигр	I.6	I.6.5.1	93
83.	Кубики на строку ниже	I.6	I.6.5.2	93
84.	Починить коридор	I.6	I.6.5.3	94
85.	Длина слова	I.6	I.6.5.4	94
86.	Умножение на 2	I.6	I.6.5.5	94
87.	Новый исполнитель	I.6	I.6.5.6	95
88.	Построение лесенки	I.6	I.6.5.7	95
89.	Подсчитать число букв и очистить поле	I.6	I.6.5.8	95
90.	Грибы-3	I.6	I.7.12	103
91.	Циклическая перестановка	I.6	I.7.13	103
92.	Отрезки	I.6	I.7.27	111
93.	Деление на два (3 балла)	I.6	Задачник 1.1	23
94.	Треугольник (5 баллов)	I.6	Задачник 1.3	25
95.	Прямоугольник (6 баллов)	I.6	Задачник 1.4	25
96.	Вычитание двух палочных чисел (10 баллов)	I.6	Задачник 2.1.4	33

Таблица 1 *(продолжение)*

Номер по порядку	Название задачи	Ссылка на тему книги «Азы программирования. Магия для начинающих»	Адрес задачи	Страница
97.	Деление палочного числа на 2 (5 баллов)	I.6	Задачник 2.1.5	33
98.	Подсчёт букв О (8 баллов)	I.6	Задачник 2.1.7	35
99.	Разворот слова (6 баллов)	I.6	Задачник 2.1.10	37
100.	Количество согласных (8 баллов)	I.6	Задачник 2.1.11	37
101.	Середина отрезка (6 баллов)	I.6	Задачник 2.1.12	37
102.	Кубик с цифрой (5 баллов)	I.6	Задачник 3.1.2	47
103.	Уплотнение записи (7 баллов)	I.6	Задачник 3.1.3	47
104.	Квадрат (5 баллов)	I.6	Задачник 3.1.9	53
105.	Срединный перпендикуляр (6 баллов)	I.6	Задачник 3.1.10	53
106.	Вниз на Pab (8 баллов)	I.6	Задачник 3.1.11	54
107.	Прямоугольник (10 баллов)	I.6	Задачник 3.1.12	55
108.	Деление отрезка (6 баллов)	I.6	Задачник 3.1.13	55
109.	Сложение строк (5 баллов)	I.6	Задачник 4.1.1	67
110.	Удаление символов (5 баллов)	I.6	Задачник 4.1.2	69
111.	Удаление символов-2 (10 баллов)	I.6	Задачник 4.1.3	69
112.	Построение окружности (8 баллов)	I.6	Задачник 4.1.8	75
113.	Поворот треугольника (8 баллов)	I.6	Задачник 4.1.10	76

Таблица 1 *(продолжение)*

Номер по порядку	Название задачи	Ссылка на тему книги «Азы программирования. Магия для начинающих»	Адрес задачи	Страница
114.	Смещение линейки (8 баллов)	I.6	Задачник 4.1.11	77
115.	Построение лесенки (8 баллов)	I.6	Задачник 4.1.12	77
116.	Построение лесенки-2 (8 баллов)	I.6	Задачник 4.1.13	77
117.	Программируемое перемещение (3 балла)	I.6	Задачник 5.1.1	85
118.	Деление пополам (5 баллов)	I.6	Задачник 5.1.2	86
119.	Ракета (5 баллов)	I.6	Задачник 5.1.3	87
120.	Опрокидывание столбика (8 баллов)	I.6	Задачник 5.1.4	88
121.	Разобрать по брёвнышку (8 баллов)	I.6	Задачник 5.1.5	88
122.	Кирпич на другой столбик (7 баллов)	I.6	Задачник 5.1.6	89
123.	Ханойская башня (15 баллов)	I.6	Задачник 5.1.7	90
124.	Сортировка (6 баллов)	I.6	Задачник 6.1.3	102
125.	Распилить по метке (4 балла)	I.6	Задачник 6.1.4	102
126.	Дрова на зиму (6 баллов)	I.6	Задачник 6.1.5	102
127.	Распилить натрое (8 баллов)	I.6	Задачник 6.1.6	103
128.	Обмен (6 баллов)	I.6	Задачник 6.1.8	104
129.	Треугольник (9 баллов)	I.6	Задачник 6.1.9	104
130.	Проще некуда (10 баллов)	I.6	Задачник 6.1.10	104

Таблица 1 *(окончание)*

Номер по порядку	Название задачи	Ссылка на тему книги «Азы программирования. Магия для начинающих»	Адрес задачи	Страница
131.	Стыковка-2 (5 баллов)	I.6	Задачник 7.1.2	114
132.	Выбрать меньшее (6 баллов)	I.6	Задачник 7.1.3	114
133.	Джип от джина (6 баллов)	I.6	Задачник 7.1.4	114
134.	Выстроить в ряд (7 баллов)	I.6	Задачник 7.1.5	115
135.	Поворот шахматной доски (9 баллов)	I.6	Задачник 7.1.7	116
136.	Инверсия (10 баллов)	I.6	Задачник 7.1.8	116
137.	Букет цветов (5 баллов)	I.6	Задачник 8.1.4	127
138.	Земельный участок (6 баллов)	I.6	Задачник 8.1.5	127
139.	Горка-1 (6 баллов)	I.6	Задачник 8.1.6	128
140.	А и Б (8 баллов)	I.6	Задачник 8.1.7	129
141.	Найди друга (9 баллов)	I.6	Задачник 8.1.8	129
142.	Горка-2 (10 баллов)	I.6	Задачник 8.1.9	130
143.	Просто обмен (11 баллов)	I.6	Задачник 8.1.10	130
144.	Чаще и меньше (12 баллов)	I.6	Задачник 8.1.11	131

Часть II. Корректор

Таблица 2

Номер по порядку	Название задачи	Ссылка на тему книги «Азы программирования. Магия для начинающих»	Адрес задачи	Страница
1.	От нечётного числа отнять единицу	II.8	II.8.3.1	124
2.	Удалить начальные пробелы	II.9	II.9.2.1	134
3.	Удалить начальные и конечные пробелы	II.9	II.9.2.2	134
4.	Сумма единиц	II.9	II.9.2.3	135
5.	Сумма цифр	II.9	II.9.2.4	135
6.	Упорядочить два символа	II.9	II.9.3.3	140
7.	Двери	II.9	II.9.3.4	140
8.	Многоточия	II.9	II.9.4.1	141
9.	Проверить порядок	II.9	II.9.4.2	142
10.	Радиус окружности	II.9	II.15.7	246
11.	Распаковка отрезка	II.9	II.15.8	246
12.	Пятёрки	II.9	II.15.9	247
13.	Распиливание бревна	II.9	II.15.10	247
14.	ЧУ-ЩУ	II.9	II.15.11	248
15.	Подсчитаем брёвнышки	II.9	II.15.12	248
16.	Копия в обратном порядке (5 баллов)	II.9	Задачник 2.2.1	39
17.	Дополнение до 9 (4 балла)	II.9	Задачник 2.2.8	41
18.	Разметка отрезка (3 балла)	II.9	Задачник 3.2.11	64
19.	Последовательность-1 (2 балла)	II.9	Задачник 5.2.1	93

Таблица 2 *(продолжение)*

Номер по порядку	Название задачи	Ссылка на тему книги «Азы программирования. Магия для начинающих»	Адрес задачи	Страница
20.	Последовательность-2 (3 балла)	II.9	Задачник 5.2.2	93
21.	Последовательность-3 (3 балла)	II.9	Задачник 5.2.3	94
22.	Последовательность-4 (4 балла)	II.9	Задачник 5.2.4	94
23.	Шифр-1 (6 баллов)	II.9	Задачник 8.2.6	96
24.	Есть ли цифры в записи	II.10	II.10.5.1	158
25.	Уплотнение двух символов	II.10	II.10.5.2	159
26.	Записать в порядке убывания	II.10	II.10.5.3	159
27.	Записать в порядке возрастания	II.10	II.10.5.4	159
28.	Запаковка отрезка	II.10	II.15.13	249
29.	Друзья	II.10	II.15.14	249
30.	Перестановка	II.10	II.15.15	249
31.	Левое зеркало	II.10	II.15.16	250
32.	Дефрагментация ленты	II.10	II.15.17	250
33.	Окно на первый символ записи	II.11	II.11.1.1	163
34.	Первое повторное вхождение символа	II.11	II.11.1.2	163
35.	Последнее вхождение символа	II.11	II.11.1.3	163
36.	Заключить в круглые скобки	II.11	II.11.2.1	168
37.	Проверка записей на совпадение	II.11	II.11.2.2	168

Таблица 2 *(продолжение)*

Номер по порядку	Название задачи	Ссылка на тему книги «Азы программирования. Магия для начинающих»	Адрес задачи	Страница
38.	Продублировать в обратном порядке	II.11	II.11.4.1	175
39.	Проверка записи целого числа	II.11	II.11.4.2	175
40.	Какое число больше	II.11	II.11.4.3	175
41.	Убрать повторные вхождения символа	II.11	II.11.4.4	176
42.	Распаковка архива	II.11	II.11.4.5	176
43.	Симметричный отрезок	II.11	II.15.18	251
44.	Деление отрезка	II.11	II.15.19	251
45.	Среднее арифметическое	II.11	II.15.20	252
46.	Самое длинное бревно	II.11	II.15.21	252
47.	Палиндром (10 баллов)	II.11	Задачник 2.2.2	39
48.	Отношение чисел (10 баллов)	II.11	Задачник 2.2.3	40
49.	Закраска окружности (3 балла)	II.11	Задачник 3.2.7	63
50.	Одинаковые символы (5 баллов)	II.11	Задачник 4.2.4	79
51.	Поиск максимума (3 балла)	II.11	Задачник 6.2.2	106
52.	Разгадай шифр-1 (6 баллов)	II.11	Задачник 6.2.4	107
53.	Разгадай шифр-2 (5 баллов)	II.11	Задачник 6.2.5	108
54.	Середина (5 баллов)	II.11	Задачник 7.2.4	119
55.	Дата (4 балла)	II.11	Задачник 8.2.4	134
56.	Вычитание двух чисел	II.12	II.12.1.1	188
57.	Умножить на 2	II.12	II.12.1.2	188

Таблица 2 *(продолжение)*

Номер по порядку	Название задачи	Ссылка на тему книги «Азы программирования. Магия для начинающих»	Адрес задачи	Страница
58.	Умножить на 4	II.12	II.12.1.3	189
59.	Умножить на 16	II.12	II.12.1.4	189
60.	Умножить на 3	II.12	II.12.1.5	189
61.	Умножить на 6	II.12	II.12.1.6	190
62.	Умножить палочное число на 2	II.12	II.12.2.1	196
63.	Умножить палочное число на 3	II.12	II.12.2.2	196
64.	Умножить палочное число на 6	II.12	II.12.2.3	196
65.	Отношение двух палочных чисел	II.12	II.12.2.4	197
66.	Первые палочные числа	II.12	II.12.2.5	197
67.	Первые чётные палочные числа	II.12	II.12.2.6	197
68.	Первые нечётные палочные числа	II.12	II.12.2.7	197
69.	Сумма символов	II.12	II.12.3.1	203
70.	Особые цифры	II.12	II.12.3.2	203
71.	Шифровка	II.12	II.12.3.3	204
72.	Вычитание символов	II.12	II.12.3.4	204
73.	Умножение символов	II.12	II.12.3.5	204
74.	Точка	II.12	II.15.1	243
75.	Повтор	II.12	II.15.2	244
76.	Неисправная клавиатура	II.12	II.15.3	244
77.	Складывай и вычитай	II.12	II.15.22	253
78.	Неправильная дробь	II.12	II.15.23	254

Таблица 2 *(продолжение)*

Номер по порядку	Название задачи	Ссылка на тему книги «Азы программирования. Магия для начинающих»	Адрес задачи	Страница
79.	Возведение в квадрат	II.12	II.15.24	254
80.	Округление	II.12	II.15.25	255
81.	Умножение на пять	II.12	II.15.26	255
82.	Деление на 2 (7 баллов)	II.12	Задачник 2.2.4	40
83.	Пример на сложение (7 баллов)	II.12	Задачник 2.2.5	40
84.	Деление палочками (6 баллов)	II.12	Задачник 2.2.11	43
85.	Закраска пересечения окружностей (8 баллов)	II.12	Задачник 3.2.8	63
86.	Последовательность Фибоначчи (6 баллов)	II.12	Задачник 3.2.9	63
87.	Последовательность Фибоначчи-2 (8 баллов)	II.12	Задачник 3.2.10	64
88.	Построение отрезка (5 баллов)	II.12	Задачник 3.2.12	65
89.	Автомат (10 баллов)	II.12	Задачник 4.2.5	80
90.	Последовательность-5 (4 балла)	II.12	Задачник 5.2.5	95
91.	Последовательность-6 (6 баллов)	II.12	Задачник 5.2.6	96
92.	НОД двух чисел (7 баллов)	II.12	Задачник 5.2.8	96
93.	Сколько палочек? (2 балла)	II.12	Задачник 6.2.1	106
94.	Умножение (7 баллов)	II.12	Задачник 6.2.6	108
95.	От и до (3 балла)	II.12	Задачник 7.2.1	118
96.	Сколько нулей, сколько единиц? (3 балла)	II.12	Задачник 7.2.2	118

Таблица 2 *(продолжение)*

Номер по порядку	Название задачи	Ссылка на тему книги «Азы программирования. Магия для начинающих»	Адрес задачи	Страница
97.	Счастливый билет (4 балла)	II.12	Задачник 7.2.3	119
98.	Смешанное сложение (5 баллов	II.12	Задачник 7.2.5	119
99.	Горка из чисел (5 баллов)	II.12	Задачник 7.2.6	120
100.	Умножение палочек (8 баллов)	II.12	Задачник 7.2.10	122
101.	Сложение отрезков (4 балла)	II.12	Задачник 8.2.1	132
102.	Два отрезка (4 балла)	II.12	Задачник 8.2.2	133
103.	Деление отрезка пополам (7 баллов)	II.12	Задачник 8.2.3	133
104.	Хитрое умножение (5 баллов)	II.12	Задачник 8.2.5	134
105.	Число букв О в тексте	II.13	II.13.1.1	213
106.	Сумма наименьшей и наибольшей цифр числа	II.13	II.13.1.2	213
107.	Число слов в тексте	II.13	II.13.1.3	213
108.	Убрать лишние пробелы	II.13	II.13.2.1	222
109.	Добавить пробелы после знаков препинания	II.13	II.13.2.2	222
110.	Заменить # образцом	II.13	II.13.2.3	222
111.	Проверить вхождение образца в запись	II.13	II.13.2.4	223
112.	Упорядочить символы по возрастанию	II.13	II.13.2.5	223
113.	Полусумма	II.13	II.15.4	244
114.	Голосование символов	II.13	II.15.5	244

Таблица 2 *(продолжение)*

Номер по порядку	Название задачи	Ссылка на тему книги «Азы программирования. Магия для начинающих»	Адрес задачи	Страница
115.	Количество перевёртышей	II.13	II.15.6	245
116.	Количество цифр в тексте (6 баллов)	II.13	Задачник 2.2.6	41
117.	Кратность трём (10 баллов)	II.13	Задачник 2.2.7	41
118.	Испорченное слово (6 баллов)	II.13	Задачник 2.2.9	42
119.	Алфавит записи (6 баллов)	II.13	Задачник 2.2.10	42
120.	НОД (10 баллов)	II.13	Задачник 2.2.12	43
121.	Популярная цифра (5 баллов)	II.13	Задачник 4.2.1	78
122.	Пересечение записей (8 баллов)	II.13	Задачник 4.2.2	78
123.	Сортировка записи (5 баллов)	II.13	Задачник 4.2.3	79
124.	Из двоичной в десятичную (10 баллов)	II.13	Задачник 4.2.6	81
125.	Из римской в десятичную (10 баллов)	II.13	Задачник 4.2.7	81
126.	Из десятичной в двоичную (8 баллов)	II.13	Задачник 5.2.9	96
127.	Самое длинное слово (9 баллов)	II.13	Задачник 5.2.10	97
128.	Ханойская башня (10 баллов)	II.13	Задачник 5.2.11	98
129.	Среднее арифметическое (5 баллов)	II.13	Задачник 6.2.3	107
130.	Деление с остатком (7 баллов)	II.13	Задачник 6.2.7	108
131.	Возведение в степень (9 баллов)	II.13	Задачник 6.2.8	109

Таблица 2 *(окончание)*

Номер по порядку	Название задачи	Ссылка на тему книги «Азы программирования. Магия для начинающих»	Адрес задачи	Страница
132.	Сократить дробь (10 баллов)	II.13	Задачник 6.2.9	109
133.	Антиавтомат (8 баллов)	II.13	Задачник 6.2.10	110
134.	Скобки (7 баллов)	II.13	Задачник 6.2.11	110
135.	Жребий на пальцах (6 баллов)	II.13	Задачник 7.2.7	120
136.	Кому выходить? (7 баллов)	II.13	Задачник 7.2.8	121
137.	Кто останется? (10 баллов)	II.13	Задачник 7.2.9	121
138.	Шифр-2 (8 баллов)	II.13	Задачник 8.2.7	135
139.	Тарабарский язык (7 баллов)	II.13	Задачник 8.2.8	136
140.	Перевод с тарабарского (7 баллов)	II.13	Задачник 8.2.9	136
141.	Раздвоитель (8 баллов)	II.13	Задачник 8.2.10	137
142.	Координаты вектора (5 баллов)	II.13	Задачник 8.2.11	137
143.	Шахматное поле (9 баллов)	II.13	Задачник 8.2.12	138
144.	Проверка идентификатора	II.14	II.14.1.1	229
145.	Проверить число	II.14	II.14.1.2	230
146.	Пример на сложение в программу для Плюсика	II.14	II.14.2.2	239
147.	Пример на сложение и вычитание в программу для Плюсика	II.14	II.14.2.3	239
148.	Трансляция `повтори`	II.14	II.14.2.4	239
149.	Модель нового исполнителя	II.14	II.14.2.5	240

Часть III. Транслятор?.. Это очень просто!

Таблица 3

Номер по порядку	Название задачи	Ссылка на тему книги «Азы программирования. Магия для начинающих»	Адрес задачи	Страница
1.	Проверка числа-1 (Кукарача)	III.17	III.17.5.1	272
2.	Проверка числа-2 (Кукарача)	III.17	III.17.5.2	273
3.	Проверка числа-3 (Кукарача)	III.17	III.17.5.3	273
4.	Проверка числа-4 (Кукарача)	III.17	III.17.5.4	274
5.	Проверка выражения (Кукарача)	III.17	III.17.5.5	275
6.	Проверка выражения (Кукарача)	III.18	III.18.2.1	281
7.	Проверка узора-1 (Кукарача)	III.18	III.18.2.2	282
8.	Проверка узора-2 (Кукарача)	III.18	III.18.2.3	283
9.	Анализатор-5 (9 баллов)	III.18	Задачник 3.1.8	51
10.	Морковь и капуста (5 баллов)	III.18	Задачник 4.1.4	70
11.	Морковь и капуста-2 (8 баллов)	III.18	Задачник 4.1.5	72
12.	Странные тексты (6 баллов)	III.18	Задачник 4.1.6	73
13.	Странные тексты-2 (7 баллов)	III.18	Задачник 4.1.7	74

Таблица 3 *(окончание)*

Номер по порядку	Название задачи	Ссылка на тему книги «Азы программирования. Магия для начинающих»	Адрес задачи	Страница
14.	Проверка программы (6 баллов)	III.18	Задачник 5.1.8	90
15.	Проверка программы-2 (6 баллов)	III.18	Задачник 5.1.9	91
16.	Исполнитель Кроха (10 баллов)	III.18	Задачник 5.1.10	91
17.	Проверка записи (8 баллов)	III.18	Задачник 6.1.11	104
18.	Кукарача — грамотей (8 баллов)	III.18	Задачник 7.1.6	116
19.	Проверка числа (Корректор)	III.19	III.19.2.1	289
20.	Проверка выражения-1 (Корректор)	III.19	III.19.2.2	290
21.	Проверка выражения-2 (Корректор)	III.19	III.19.2.3	291
22.	Малыш (30 баллов)	III.19	Задачник 4.2.8	82
23.	Проверка программы (6 баллов)	III.19	Задачник 5.2.7	96
24.	Проверка выражения (12 баллов)	III.19	Задачник 6.2.12	111
25.	Хохотунчики (8 баллов)	III.19	Задачник 7.2.11	122

Предметный указатель