Программирование:
ВВОДНЫЙ КУРС

Учебное пособие
Рекомендовано Министерством образования
Российской Федерации

МЦНМО
1995

Программирование: вводный курс.
Под редакцией Д. Школьника
М.: МЦНМО, 1995

При подготовке текста в значительной
степени использованы задания для учащихся
школы № 57, составленные А. Шенем.

Текст учебника: Р. Авданин,
А. Суханов, В. Хименко, Д. Школьник

Авторы благодарят
В. Радионова за многочисленные замечания.

Тексты программ: Р. Авданин, М. Вьюгин,
А. Коган, В. Кордонский, А. Тилипман, В. Хименко,
А. Шень, Д. Школьник, Е. Школьник

Иллюстрации: М. Иванова

Учебное пособие
Рекомендовано Министерством образования Российской Федерации

Излагается начальный курс программирования. Предлагается свыше ста задач, к большей части которых даны решения. Для учителей информатики, старшеклассников, студентов младших курсов высших учебных заведений. Пособие может быть использовано на кружковых и факультативных занятиях в общеобразовательных учреждениях, а также в школах с углубленным изучением математики и информатики.

ISBN 5-900916-02-2

©МЦНМО, 1995
©Sophie Bockholdt (Deutschland), 1995, обложка

Предисловие

В настоящее время обучение старшеклассников программированию (например, в рамках предмета «Основы информатики и вычислительной техники») стало нормой. Разумеется, в эпоху бурного развития информационных технологий наивно ожидать появления сколько-либо полного и стабильного учебника. В то же время потребность в учебнике есть, и немалая. Мы попытались хотя бы частично решить проблему, обобщив в предлагаемой книге наш опыт занятий программированием с учащимися математических классов московской школы № 57.

Приступая к работе над этой книгой, мы планировали ее как аналог известных пособий по различным разделам высшей математики — например, задачника Б. П. Демидовича по математическому анализу или И. В. Проскурякова по линейной алгебре. На наш взгляд, программирование не в меньшей степени, нежели упомянутые математические дисциплины, приемлет обучение на систематически подобранных задачах.

Данная книга является первой частью и представляет собой именно вводный курс, рассчитанный на первый год обучения — наша практика показывает, что материал этой книги может быть изучен в режиме двух уроков в неделю в девятом классе средней школы.

К сожалению, реальное положение таково, что большинство студентов младших курсов вузов либо не изучали программирования в школе вовсе, либо занимались «сухим плаванием», либо (что, по нашему мнению, еще хуже) изучали Бейсик. Мы полагаем, что материал этой книги может послужить основой семестрового вводного курса.

Для практического решения содержащихся в книге задач нами был разработан ряд программ. Все они содержатся на дискете, прилагаемой к книге. Все содержимое дискеты, включая TEX-вариант второй части курса — книги А. Шеня «Программирование: теоремы и задачи» — является свободно распространяемым и может быть использовано в некоммерческих (в том числе научных или образовательных) целях без согласования с авторами. Мы настаиваем, однако, на распространении этих материалов в неизменном виде — если вы внесли коррективы, переименуйте файл и укажите, какие изменения внесены вами.

Содержание

ВВЕДЕНИЕ, или С ЧЕГО НАЧАТЬ?

Вероятно, каждый учитель (и не только программирования), приходя работать с новым классом, задает себе этот сакраментальный вопрос. Наверное, лучший ответ на него дал Л. Кэролл, сказав «начни с начала».

Разумеется, при построении вводного курса можно было пойти несколькими путями. Нам показалось естественным начинать не с каких-либо распространенных языков программирования, а со специальных учебных языков с несложным синтаксисом.

Мы считаем, что нельзя обучать нескольким вещам одновременно — если изучается новый язык, то первые программы не должны вызывать алгоритмических сложностей, и наоборот, при изучении нетривиального алгоритма ученик должен свободно владеть синтаксисом языка. Вот почему мы начинаем изучение программирования с языка управления Роботом.

Идея начального курса программирования, основанного на управлении движущимся по плоскости объектом, не нова (в частности, использована С. Пейпертом в языке Лого); на мехмате МГУ с 1980 г. курс программирования (А. Г. Кушниренко) начинался с программ управления Роботом. Используемый нами набор команд Робота — это некоторое подмножество набора, предложенного А. Г. Кушниренко.

В предлагаемой нами системе подготовки программ для Робота содержится синтаксически ориентированный редактор, использование которого делает невозможным написание синтаксически неверных программ и «принуждает» к написанию понятных, структурированных программ. Ее первая версия была написана в 1990 г. А. Тилипманом и В. Кордонским. Поставляемая версия программы написана в основном В. Хименко.

Обучение программированию математически одаренных детей, на наш взгляд, не должно сводиться к изучению лишь одной «ветви» современного программирования — процедурным (алголоподобным) языкам. В качестве иллюстрации мы приводим функциональный язык RL, который был разработан С. А. Романенко для экспериментов с автоматическим преобразованием программ. Интерпретатор RL, прилагаемый к данной книге, написан Р. Авданиным и Д. Школьником.

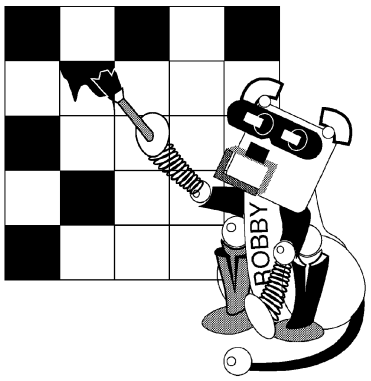
Нам представляется полезным знакомить старшеклассников с основами математической логики при помощи программы моделирования схем из функциональных элементов, разработаной Р. Авданиным, А. Г. Коганом, А. Шенем, М. Вьюгиным и Е. Школьником.

Часть I

ПОИГРАЕМ?

Глава I

ИГРА В РОБОТА



1. ПРАВИЛА ИГРЫ

Робот. Команды и проверки

Робот находится на плоскости, разбитой на квадратные клетки. Некоторые клетки могут быть разделены стенами. Кроме того, некоторые клетки могут быть закрашены.

Робот находится в одной из клеток. Робот умеет выполнять пять элементарных команд и столько же проверок:

`шаг на север`	`на севере свободно`
`шаг на юг`	`на юге свободно`
`шаг на восток`	`на востоке свободно`
`шаг на запад`	`на западе свободно`
`закрасить`	`закрашено`

Каждая из команд `шаг на ...` выполнима, если и только если в соответствующем направлении нет стены, и сдвигает Робота на одну клетку.

Попытка выполнить команду `шаг на ...` в ситуации, когда в соответствующем направлении стена, приводит к аварийному прекращению выполнения программы.

Команда `закрасить` выполнима всегда (клетку можно красить повторно). Она закрашивает клетку, в которой находится Робот.

Условия. Образование новых команд

Назовем *условием* то, что можно построить по правилам:

- проверка есть условие;
- если u_1 и u_2 — условия, то (u_1 **и** u_2) — условие, которое истинно тогда и только тогда, когда истинны оба условия u_1 и u_2;
- если u_1 и u_2 — условия, то (u_1 **или** u_2) — условие, которое истинно тогда и только тогда, когда истинно хотя бы одно из условий u_1 и u_2;
- если u — условие, то (**не** u) — условие, которое истинно тогда и только тогда, когда условие u ложно.

Назовем *командой* то, что можно построить по правилам:

- элементарная команда — команда;
- последовательное выполнение нескольких (в том числе одной или нуля) команд — команда;
- повторение команды k, пока истинно некоторое условие u — команда:

  пока u **выполнять**
  | k
  конец

  При этом команда k выполнится столько раз, сколько нужно для того, чтобы условие u стало ложным.

  Образованная таким образом команда называется циклом, u — условием цикла, k — телом цикла;
- пусть u_1, u_2, …, u_n — условия, а k_1, k_2, …, k_n — команды. Тогда можно образовать новую команду k, выполнение которой состоит в том, что среди

условий u_1, u_2, ..., u_n отыскивается первое истинное (если таких нет, то применение k эквивалентно пустой последовательности команд), после чего выполняется соответствующая ему команда:

```
выбор
| при условии: u_1
| | k_1
| конец
| при условии: u_2
| | k_2
| конец
| ⋮
| при условии: u_n
| | k_n
| конец
конец
```

Образованные по этим правилам команды могут, в свою очередь, использоваться при образовании других команд, и т.д.

Введем множество всех состояний Робота, каждое из которых задается указанием того, где стоят стены, какие клетки закрашены, и где находится Робот. Это множество включает в себя также состояние ОТКАЗ, которое возникает при попытке выполнить команду `шаг на ...` в случае, когда в указанном направлении находится стена. Каждая команда переводит Робота из одного состояния в другое.

Программа и процедура

Программа Робота имеет вид:

```
программа ИмяПрограммы
начало
```

```
    команда
конец
```

Язык Робота позволяет присваивать командам имена. Для того, чтобы присвоить команде имя, нужно вставить в программу описание процедуры:

```
процедура ИмяПроцедуры
    команда
конец
```

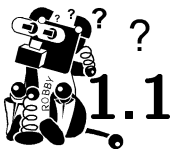
Все описания процедур помещаются в программе после заголовка (первой строки) до слова `начало`.

Обращение к команде, описанной процедурой, в программе (или в описанных ниже процедурах) имеет вид:

ИмяПроцедуры

2. РАСКРАСКИ

В приведенных ниже формулировках задач указаны предусловие (**Дано**) и постусловие (**Надо**). Требуется построить команду, которая применима в любом состоянии, удовлетворяющем предусловию, и завершается в состоянии, удовлетворяющем постусловию.

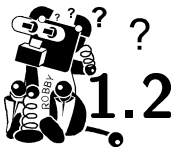


Дано: *Робот в огороженном прямоугольнике.*
Надо: *Робот у северной стены.*

Решение:

```
программа Север
начало
    пока на севере свободно выполнять
        шаг на север
    конец
конец
```

◀



Дано: *Робот в огороженном прямоугольнике.*
Надо: *Робот в северо-западном углу.*

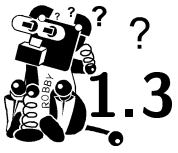
Решение: Тело процедуры `НаСеверДоУпора` совпадает с телом предыдущей программы. Процедура `НаЗападДоУпора` получается из этой процедуры заменой «север» → «запад».

```
программа вСевероЗападныйУгол
процедура НаСеверДоУпора
|   пока на севере свободно выполнять
|   |   шаг на север
|   конец
конец
процедура НаЗападДоУпора
|   пока на западе свободно выполнять
|   |   шаг на запад
|   конец
конец
начало
|   НаСеверДоУпора
|   НаЗападДоУпора
конец
```

◀

Инвариантом цикла называется утверждение, справедливое после любого (в том числе нулевого) числа повторений тела цикла.

Таким образом, если имеется цикл с условием u и инвариантом i, постусловие для данного цикла есть (`не` u) `и` i.



Дано: *Робот в южной клетке вертикального коридора шириной в одну клетку.*
Надо: *Все клетки коридора закрашены.*

Решение: Рассмотрим цикл с инвариантом «клетка с Роботом и все клетки ниже Робота закрашены».

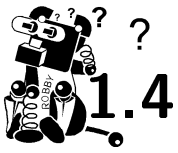
```
программа ЗакраситьКоридор
начало
|  закрасить
|  пока на севере свободно выполнять
|  |  шаг на север
|  |  закрасить
|  конец
конец
```

После выполнения цикла инвариант цикла остается истинным, а условие цикла становится ложным, то есть Робот стоит в верхней клетке и столбец под ним закрашен, что и требовалось. ◀

▶ **Упражнение 1.1.** *Изменить приведенное решение предыдущей задачи, считая инвариантом утверждение «все клетки коридора ниже Робота закрашены».*

Решение:

```
программа ЗакраситьКоридор
начало
|  пока на севере свободно выполнять
|  |  закрасить
|  |  шаг на север
|  конец
|  закрасить
конец
```
◀



Дано: *Робот в юго-западном углу прямоугольника.*

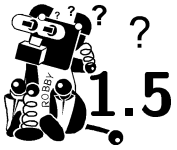
Надо: *Прямоугольник закрашен.*

Решение: Изменим решение задачи 1.3 так, чтобы Робот после закраски столбца возвращался к его нижней (южной) границе (процедура `ЗакраситьСтолбецИВернуться`).

Для закраски всего прямоугольника (тело программы) построим цикл с инвариантом «столбец с Роботом и все столбцы западнее Робота закрашены». Если в результате выполнения цикла Робот окажется у восточной стены, задача будет решена.

```
программа ЗакраситьПрямоугольник
процедура ЗакраситьСтолбецИВернуться
|  закрасить
|  пока на севере свободно выполнять
|  |  шаг на север
|  |  закрасить
|  конец
|  пока на юге свободно выполнять
|  |  шаг на юг
|  конец
конец
начало
|  ЗакраситьСтолбецИВернуться
|  пока на востоке свободно выполнять
|  |  шаг на восток
|  |  ЗакраситьСтолбецИВернуться
|  конец
конец
```

◀



Дано: *Робот в нижней клетке незакрашенного коридора шириной в одну клетку.*

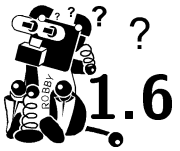
Надо: *Клетки закрашены через одну (начиная с первой).*

Решение: Рассмотрим цикл с инвариантом «клетка с Роботом и все клетки ниже Робота имеют верный цвет». Для определенности будем считать, что Робот находится в самой верхней из закрашенных клеток (это условие может быть нарушено лишь в одном случае: в коридоре четное число клеток и Робот находится у северной стены).

```
программа ЗакраситьСтолбецЧерезКлетку
начало
```

```
    закрасить
    пока на севере свободно выполнять
    |  шаг на север
    |  выбор
    |  |  при условии: на севере свободно
    |  |  |  шаг на север
    |  |  |  закрасить
    |  |  конец
    |  конец
    конец
конец
```

◀



1.6

Дано: *Робот в левом нижнем углу огороженного незакрашенного прямоугольника.*

Надо: *Прямоугольник закрашен в шахматном порядке, левая нижняя клетка закрашена*[1]*.*

Решение: Процедура `ЗакраситьСтолбецЧерезКлеткуИНазад` повторяет процедуру `ЗакраситьСтолбецЧерезКлетку` из решения предыдущей задачи с тем отличием, что после закраски столбца Робот возвращается к южной стене.

Тело программы строится аналогично телу этой процедуры (с учетом того, что Робот движется с запада на восток и вместо закраски одной клетки красится целый столбец).

Закраска четного столбца (в случае, когда высота столбца больше единицы) после выполнения шага на север сводится к закраске нечетного столбца.

```
программа ЗакраситьПрямоугольникВШахматномПорядке
процедура ЗакраситьСтолбецЧерезКлеткуИНазад
|  закрасить
|  пока на севере свободно выполнять
|  |  шаг на север
```

[1] Высота или ширина прямоугольника (или обе) могут быть равны единице.

```
    выбор
      при условии: на севере свободно
        шаг на север
        закрасить
      конец
    конец
  конец
  пока на юге свободно выполнять
    шаг на юг
  конец
конец
начало
  ЗакрситьСтолбецЧерезКлеткуИНазад
  пока на востоке свободно выполнять
    шаг на восток
    выбор
      при условии: на севере свободно
        шаг на север
        ЗакрситьСтолбецЧерезКлеткуИНазад
      конец
    конец
    выбор
      при условии: на востоке свободно
        шаг на восток
        ЗакрситьСтолбецЧерезКлеткуИНазад
      конец
    конец
  конец
конец                                                    ◀
```

В разделе 5 приведены довольно изящные рекурсивные решения задач 1.5 и 1.6.

3. ЛАБИРИНТЫ

В этом разделе рассматриваются задачи, связанные с нахождением некоторого пути в заданном лабиринте.

Дано: *Робот у южной стены огороженного прямоугольника, все стены внутри которого горизонтальны и находятся на одной широте.*

Надо: *Робот у северной стены.*

Решение: Чем ряд у северной стены прямоугольника отличается от ряда у горизонтальной перегородки? Тем, что для любой клетки у северной стены справедливо условие `не на севере свободно`, тогда как хотя бы для одной клетки ряда у перегородки выполняется условие `на севере свободно`.

В терминах Робота это различие может быть описано циклом с инвариантом «у всех клеток западнее Робота с северной стороны стена».
Выполнение этого цикла должно завершаться в двух случаях:

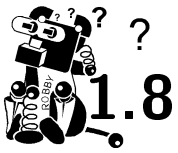
- Робот дошел до клетки, у которой свободно на севере, или
- Робот дошел до восточной стены.

Выполнение этой процедуры (назовем ее `ПоискПроходаНаСевер`) заканчивается, когда Робот оказывается у прохода на север (если он имеется) или в северо-восточном углу прямоугольника.

```
программа кСевернойСтенеЧерезПерегородку
процедура ПоискПроходаНаСевер
|  НаЗападДоУпора[1]
|  пока (не на севере свободно и на востоке
|  свободно) выполнять
|  |  шаг на восток
|  конец
конец
```

[1] Эта процедура описана в решении задачи 1.2.

```
начало
|  НаСеверДоУпора[1]
|  ПоискПроходаНаСевер
|  НаСеверДоУпора
конец                                                    ◀
```



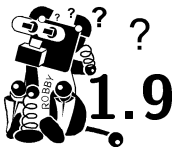
1.8

Дано: *В огороженном прямоугольнике есть только несплошные горизонтальные стены.*

Надо: *Робот у северной границы.*

Решение: В этой программе мы вновь используем свойство процедуры ПоискПроходаНаСевер, состоящее в том, что Робот останавливается в клетке, для которой выполнено условие на севере свободно (если такая есть в данном ряду).

```
программа НаСеверЧерезГоризонтальныеСтены
начало
|  ПоискПроходаНаСевер[2]
|  пока на севере свободно выполнять
|  |  НаСеверДоУпора[3]
|  |  ПоискПроходаНаСевер
|  конец
конец                                                    ◀
```



1.9

Дано: *В огороженном прямоугольнике есть вертикальные и горизонтальные стены, не примыкающие к границе прямоугольника и друг к другу.*

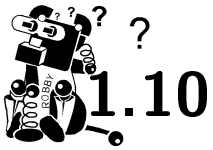
Надо: *Робот в северо-западном углу прямоугольника.*

[1] Эта процедура описана в решении задачи 1.2.
[2] Эта процедура описана в решении задачи 1.7.
[3] Эта процедура описана в решении задачи 1.2.

Решение: Эта задача прекрасно иллюстрирует то, как важно сформулировать постусловие. Действительно, северо-западный угол — это единственная клетка прямоугольника, для которой справедливо условие `(не на севере свободно) и (не на западе свободно)`.

```
программа вСевероЗападныйУгол
начало
| пока (на севере свободно или
| на западе свободно) выполнять
| | выбор
| | | при условии: на севере свободно
| | | | шаг на север
| | | конец
| | | при условии: на западе свободно
| | | | шаг на запад
| | | конец
| | конец
| конец
конец
```

Докажем, что данный цикл закончит свою работу. Введем систему координат с началом в северо-западном углу. Пусть Робот начинает работу с клетки (x, y). В результате выполнения тела цикла одна из координат уменьшится на единицу. В то же время, ни одна из координат не может стать отрицательной, следовательно, когда-нибудь цикл закончит свою работу (при этом, легко видеть, тело цикла выполнится $x + y$ раз). ◂



1.10

Дано: *В огороженном прямоугольнике есть вертикальные и горизонтальные стены, не примыкающие друг к другу (но, возможно, примыкающие к границе) и не нарушающие связности.*

Надо: *Робот в северо-западном углу прямоугольника.*

Решение: Заметим, что отрезок вида 94.005.33159.385.332 36.675.3332.005.334 ...

может находиться только у северной границы лабиринта. Процедура кСевернойСтене заканчивает свою работу, когда Робот натыкается на такой отрезок.

Процедура ПоискОбхода аналогична процедуре ПоискПроходаНаСевер с тем отличием, что Робот движется с севера на юг вдоль западной перегородки.

```
программа вСевероЗападныйУголЛабиринта
процедура кСевернойСтене
|  ПоискПроходаНаСевер[1]
|  пока на севере свободно выполнять
|  |  НаСеверДоУпора[2]
|  |  ПоискПроходаНаСевер
|  конец
конец
процедура ПоискОбхода
|  НаСеверДоУпора
|  пока (не на западе свободно и на юге свободно)
|  выполнять
|  |  шаг на юг
|  конец
конец
процедура ПоСевернойСтеневУгол
|  ПоискОбхода
|  пока на западе свободно выполнять
|  |  шаг на запад
|  |  ПоискОбхода
|  конец
|  НаСеверДоУпора
конец
начало
|  кСевернойСтене
|  ПоСевернойСтеневУгол
конец                                                    ◀
```

[1] Эта процедура описана в решении задачи 1.7.
[2] Эта процедура описана в решении задачи 1.2.

1.11

Дано: *В огороженном прямоугольнике есть не примыкающая к границе огороженная со всех сторон клетка; других стен нет, Робот в юго-западном углу.*

Надо: *Робот в клетке, соседней с огороженной.*

Решение:

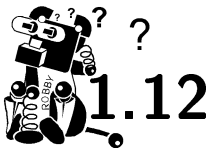
```
программа НайтиКлетку
процедура ПоискВСтолбце
|  пока (на севере свободно и на востоке свободно)
|  выполнять
|  |  шаг на север
|  конец
|  выбор
|  |  при условии: не на севере свободно
|  |  |  пока на юге свободно выполнять
|  |  |  |  шаг на юг
|  |  |  конец
|  |  конец
|  конец
конец
начало
|  ПоискВСтолбце
|  пока на востоке свободно выполнять
|  |  шаг на восток
|  |  ПоискВСтолбце
|  конец
конец
```

◀

4. ДОКАЗАТЕЛЬСТВА НЕВОЗМОЖНОСТИ

Во всех задачах, которые мы до сих пор решали, мы не пользовались проверками `закрашено` и процедурами, использующими еще не определенные команды.

В задачах данного пункта доказывается, что при соблюдении этих условий нельзя построить команды, выполняющие определенные задания.



Дано: *Робот находится в n-ой клетке ограниченного слева и неограниченного справа коридора высотой в одну клетку.*

Надо: *Робот находится в $2n$-ой клетке.*

Решение:

Первый способ

Любой нормально завершающейся программе Робота для данного поля можно однозначно поставить в соответствие функцию с натуральными аргументом и значением, определяемую так: $f(n)$ есть номер клетки (считая от левого конца коридора), на которой Робот остановится, если первоначально он находился в клетке с номером n.

Докажем, что верно следующее утверждение: для любой программы на данном поле, которая успешно завершается при любом положении Робота, существует константа C (зависящая только от программы) такая, что либо

$$f(n) < C, \tag{4.1}$$

либо

$$|f(n) - n| < C. \tag{4.2}$$

Доказательство: Для начала заменим нашу программу на эквивалентную ей программу, вообще не содержащую процедур. Для этого вместо каждого вызова вставим тело соответствующей процедуры.

За число C примем количество строк в получившейся программе.

Пусть для данного n программа нормально завершилась.

1) Допустим, что Робот побывал хотя бы один раз в крайней западной («угловой») клетке. Рассмотрим, что будет после того, как он побывал в этой клетке в последний раз. Если он там и остановился, истинно утверждение 4.1. Если же нет, то заметим, что во всех остальных клетках значения проверок (а значит, и их комбинаций) одинаковы (на юге и севере — стены, на западе и востоке свободно). Следовательно, ни одно из условий цикла, которые встретятся исполнителю, не может выполняться (иначе этот цикл будет выполняться всегда и программа никогда не закончит работу). Отсюда следует, что количество шагов, которые сделает Робот, не может превышать количества строк программы (так как все остальные строчки могут привести к сдвигу Робота не более чем на один шаг и к ранее выполненной строчке исполнитель уже не возвращается).

2) Если Робот ни разу не побывает в «угловой» клетке, верно утверждение 4.2 (доказывается так же, как 4.1 в п. 1).

Второй способ

Пусть S — множество всех возможных состояний Робота. Будем говорить, что задана схема, если

1) На плоскости нарисовано несколько точек (будем называть их вершинами), соединенных стрелками.

2) Одна из вершин объявлена начальной, другая — конечной.

3) На некоторых стрелках написаны условия (то есть логические комбинации проверок), на других — исходные команды Робота, а на третьих ничего не написано — в последнем случае мы будем говорить, что написана пустая команда.

4) Из конечной вершины не выходит ни одной стрелки, из прочих выходит либо одна стрелка с командой, либо произвольное число стрелок с проверками.

Пусть заданы схема и начальное состояние Робота. Определим понятие исполнения схемы в этом состоянии. Поместим фишку в начальную вершину. Затем будем передвигать ее, одновременно меняя состояние Робота. Если вершина с фишкой является конечной, то исполнение заканчивается. Если из вершины выходит одна стрелка с командой Робота (возможно, пустой), то передвигаем по ней фишку, одновременно выполняя команду (ничего не делая, если команда пустая). Если этого сделать нельзя, происходит отказ. Если из вершины выходит несколько стрелок с условиями, то двигаем фишку по любой из тех стрелок, для которых условие истинно в данном состоянии Робота (состояние Робота при этом не меняем). Если все условия ложны, происходит отказ.

Лемма. *Для всякой команды k, полученной по правилам раздела 1, существует схема H со следующими свойствами:*

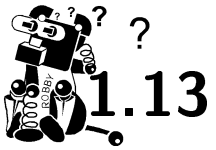
1) *исполнение схемы H происходит однозначно (всегда истинно не более одного условия);*

2) *если команда k применима в состоянии x и переводит Робота в состояние y, то исполнение схемы H в состоянии x завершается в состоянии y;*

3) *если команда k неприменима в состоянии x, то исполнение схемы H в состоянии x либо никогда не заканчивается, либо приводит к отказу.*

Лемма доказывается индукцией по построению команды k.

Вернемся к решению задачи. Пусть k — удовлетворяющая условию команда и H — соответствующая схема. Обозначим число вершин этой схемы через N. Рассмотрим исполнение H в состоянии с $n > N$. Для каждой границы между клетками на участке от n до $2n$ рассмотрим вершину схемы, в которой был Робот, когда он в последний раз пересекал эту границу. Поскольку границ n (то есть больше, чем вершин схемы), найдутся разные границы $k/k+1$ и $l/l+1$, где $k < l$, которым соответствует одна вершина, т.е. исполнение схемы H после пересечения второй границы должно повторить (со сдвигом) исполнение H после пересечения первой границы. А это противоречит тому, что выполнение программы заканчивается. ◂



1.13

Дано: *Робот находится в коридоре шириной в одну клетку на расстоянии n от южной стены.*

Надо: *Робот находится на расстоянии n от северной стены.*

Указание: Выберите длину коридора $k > 6C$, где C — количество строк в программе и начальное положение Робота n такое, что $C < n < 2C$. ◂

1.14

Дано: *На бесконечном поле есть одна горизонтальная стена неизвестной ширины, Робот стоит снизу от нее.*

Надо: *Робот оказался на одну клетку севернее исходного положения (обойдя стену).* ◂

▸ **Упражнение 1.2.** *Верно ли обратное лемме утверждение: для каждой схемы H, движение по которой однозначно, можно построить команду k, связанную с ней так, как об этом говорится в лемме?* ◂

5. РЕКУРСИЯ

Внесем изменение в правила построения команд Робота: разрешим использовать в процедурах команды, описанные любыми процедурами программы, а не только теми процедурами, которые определены ранее.

Определение. Рекурсивный аналог цикла

Рекурсией называется ситуация, когда процедура обращается к самой себе прямо или косвенно (то есть имеется цепочка вызовов $P_1 \to P_2 \to \ldots \to P_n \to P_1$).

Рассмотрим схему выполнения цикла:

```
программа ПримерЦикла
начало
| пока и выполнять
```

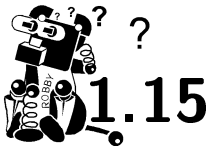
```
    |   k
    конец
конец
```

Команда k выполняется ровно столько раз, сколько необходимо для того, чтобы условие u стало ложным. Тот же эффект может быть достигнут следующим образом:

```
программа ПримерРекурсии
процедура ЕщеРазок
|  выбор
|  |  при условии: u
|  |  |  k
|  |  |  ЕщеРазок
|  |  конец
|  конец
конец
начало
|  ЕщеРазок
конец
```

Решение задач

Все задачи этого пункта были рассмотрены в разделе «Раскраски». Некоторые из приводимых ниже решений иллюстрирует рассуждения, приведенные выше, а некоторые приводятся потому, что они проще, нежели нерекурсивные.



1.15

Дано: *Робот в нижней клетке вертикального коридора.*

Надо: *Робот в верхней клетке.*

Решение[1]*:*

```
программа Север
процедура шаг
|  выбор
|  |  при условии: на севере свободно
|  |  |  шаг на север
|  |  |  шаг
|  |  конец
|  конец
конец
начало
|  шаг
конец
```
◀

При написании этой рекурсивной программы нами использовано следующее утверждение: для того, чтобы из данной клетки попасть в северную, достаточно попасть в следующую клетку, а из следующей попасть в северную.

1.16

Дано: *Робот в юго-западном углу.*
Надо: *Поле закрашено и Робот в исходном положении.*

Решение[2]*:*

```
программа ЗакрситьПолеИВернуться
процедура ЗакрситьСтолбецИНазад
|  выбор
|  |  при условии: на севере свободно
|  |  |  шаг на север
|  |  |  ЗакрситьСтолбецИНазад
|  |  |  шаг на юг
|  |  конец
|  конец
|  закрасить
конец
процедура ЗакрситьВсеПоле
```

[1] Сравните с решением задачи 1.1.
[2] Сравните с решением задачи 1.4.

```
    выбор
      при условии: на востоке свободно
        шаг на восток
        ЗакраситьВсеПоле
        шаг на запад
      конец
    конец
    ЗакраситьСтолбецИНазад
конец
начало
    ЗакраситьВсеПоле
конец
```
◀

В данной программе мы нашли способ объединить два цикла — закраску столбца и возвращение Робота — в одну рекурсивную процедуру. Действительно, каждому шагу на север соответствует шаг на юг.

1.17

Дано: *Робот в нижней клетке столбца шириной в одну клетку.*

Надо: *Клетки закрашены через одну, Робот в исходном положении.*

Решение[1]*:*

```
программа ЗакраситьСтолбецЧерезКлеткуИВернуться
процедура НачатьСПустой
    выбор
      при условии: на севере свободно
        шаг на север
        НачатьСЗакрашенной
        шаг на юг
      конец
    конец
конец
процедура НачатьСЗакрашенной
    выбор
      при условии: на севере свободно
        шаг на север
```

[1] Сравните с решением задачи 1.5.

```
        НачатьСПустой
        шаг на юг
      конец
    конец
    закрасить
конец
начало
  НачатьСЗакрашенной
конец
```

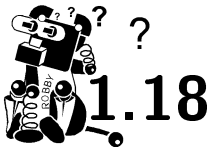
◀

Этот пример интересен тем, что в нем мы впервые используем косвенную рекурсию. В постановке задачи заложена некоторая симметрия — чередование пустых и закрашенных клеток. Как следствие этой симметрии имеем:

- для того, чтобы закрасить столбец через одну клетку, начиная с данной клетки, достаточно перейти в следующую клетку, закрасить столбец, оставляя первую клетку чистой, вернуться и закрасить данную клетку;

- для того, чтобы закрасить столбец через одну клетку, не закрашивая первую, достаточно перейти в следующую клетку и применить 1).

Использование рекурсии позволяет избежать необходимого при циклическом решении разбора вырожденных случаев (когда одна или обе стороны поля имеют длину, равную единице).



1.18

Дано: *Робот в юго-западном углу.*
Надо: *Прямоугольник закрашен в шахматном порядке, Робот в исходном положении.*

Решение[1]*:*

Попробуем применить те же рассуждения, что и при решении предыдущей задачи, учитывая чередование столбцов.

```
программа ЗакраситьПолеВШахматномПорядке
процедура ЧетныйСтолбец
|  выбор
|  |  при условии: на востоке свободно
|  |  |  шаг на восток
|  |  |  НечетныйСтолбец
|  |  |  шаг на запад
|  |  конец
|  конец
|  НачатьСПустой[2]
конец
процедура НечетныйСтолбец
|  выбор
|  |  при условии: на востоке свободно
|  |  |  шаг на восток
|  |  |  ЧетныйСтолбец
|  |  |  шаг на запад
|  |  конец
|  конец
|  НачатьСЗакрашенной[3]
конец
начало
|  НечетныйСтолбец
конец                                                    ◀
```

[1] Сравните с решением задачи 1.6.

[2] Эта процедура описана в решении задачи 1.17.

[3] Эта процедура описана в решении задачи 1.17.

Доказательства рекурсивных программ

Как всегда, есть два вопроса:

1) почему программа заканчивает работу?
2) почему она работает правильно, если заканчивает работу?

Начнем со второго вопроса. Достаточно проверить, что содержащая рекурсивный вызов процедура работает правильно, исходя из предположения, что вызываемая ею одноименная процедура работает правильно. В самом деле, если это так, то все процедуры в цепочке рекурсивно вызываемых процедур работают правильно (убеждаемся в этом, идя от конца цепочки к началу).

В этом рассуждении фактически используется метод математической индукции: мы проверяем, что последняя[1] из вызываемых процедур (одноименных!) работает верно (база индукции), и показываем, что из предположения о правильности 1-й, 2-й, ..., $(N-1)$-й процедур[2] (предположение индукции) следует правильность N-й процедуры (индуктивный переход).

Теперь перейдем к первому вопросу. Почему заканчивает работу рекурсивная программа, приведенная в качестве решения задачи 1.15? Ответ прост: при каждом новом вызове расстояние до стены уменьшается, и потому бесконечная цепочка вызовов невозможна.

Подобные рассуждения применимы и в других задачах: указывается некоторая характеристика, которая уменьшается с каждым вызовом, но не может уменьшаться беспредельно[3].

---

[1] Мы предположили, что программа заканчивает работу.

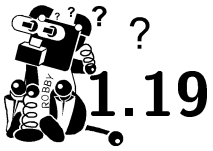
[2] Мы нумеруем процедуры с конца: 1-я закончившаяся — это последняя вызванная, 2-я закончившаяся — это предпоследняя и т.д.

[3] Мы использовали этот прием и при доказательстве правильности нерекурсивных программ. Смотрите, например, задачу 1.9.

6. РЕКУРСИЯ (ПРОДОЛЖЕНИЕ)

Использование рекурсии позволяет найти решения для некоторых задач, неразрешимых без рекурсии (для некоторых из этих задач в разделе 4 приведены доказательства неразрешимости).

Несколько примеров



Дано: *Робот находится в некоторой клетке идущего с севера на юг коридора шириной в одну клетку.*

Надо: *Закрасить самую северную клетку коридора и вернуться на место.*

Решение: Применим знакомый прием (см. задачи раздела 5): если перед рекурсивным вызовом делается шаг в некотором направлении, а после возврата из процедуры делается шаг в противоположном направлении, то количество реально сделанных шагов в обоих направлениях одинаково.

```
программа ЗакраситьСевер
процедура ЗакраситьСевернуюКлеткуИНазад
| выбор
| | при условии: на севере свободно
| | | шаг на север
```

```
        ЗакрситьСевернуюКлеткуИНазад
        шаг на юг
      конец
      при условии: не на севере свободно
        закрасить
      конец
    конец
конец
начало
  ЗакрситьСевернуюКлеткуИНазад
конец
```

◀

1.20

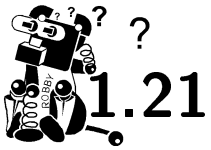
Дано: *Робот находится под ни к чему не примыкающей горизонтальной стеной рядом с ней.*

Надо: *Сместиться на одну клетку севернее (обогнув стену).*

Решение: Фактически здесь обыгрывается та же идея, что и в предыдущей задаче.

```
программа СквозьСтену
процедура ОбогнутьСтену
  выбор
    при условии: на севере свободно
      шаг на север
    конец
    при условии: не на севере свободно
      шаг на восток
      ОбогнутьСтену
      шаг на запад
    конец
  конец
конец
начало
  ОбогнутьСтену
конец
```

◀



1.21

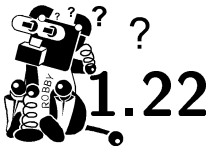
Дано: *Робот стоит в горизонтальном коридоре высотой в одну клетку. В восточном направлении коридор бесконечен, на западе — стена.*

Надо: *Отойти от западной стены на вдвое большее расстояние, чем было изначально.*

Решение: Некоторая вариация на тему предыдущих задач: каждому шагу на запад (к стене) соответствует два шага на восток.

```
программа ОтойтиОтСтеныНаВдвоеБольшееРасстояние
процедура УдвоитьРасстояние
|  выбор
|  |  при условии: на западе свободно
|  |  |  шаг на запад
|  |  |  УдвоитьРасстояние
|  |  |  шаг на восток
|  |  |  шаг на восток
|  |  конец
|  конец
конец
начало
|  УдвоитьРасстояние
конец
```

◀



1.22

Дано: *Робот находится в лабиринте с конечным числом доступных клеток. Ни одна клетка не закрашена.*

Надо: *Закрасить все доступные клетки.*

Решение: Главная трудность, с которой мы сталкиваемся при решении этой задачи, состоит в том, чтобы правильно сформулировать требования к программе в ситуации,

когда некоторые клетки закрашены, — так, чтобы прошел «шаг индукции».

Один из возможных вариантов решения таков: программа закрашивает все незакрашенные клетки, к которым можно пройти из текущей клетки по незакрашенным клеткам[1].

```
программа ЗакраситьЛабиринт
процедура ЗакраситьВсеДоступныеКлетки
| выбор
|   при условии: не закрашено
|   | закрасить
|   | выбор
|   |   при условии: на севере свободно
|   |   | шаг на север
|   |   | ЗакраситьВсеДоступныеКлетки
|   |   | шаг на юг
|   |   конец
|   | конец
|   | выбор
|   |   при условии: на юге свободно
|   |   | шаг на юг
|   |   | ЗакраситьВсеДоступныеКлетки
|   |   | шаг на север
|   |   конец
|   | конец
|   | выбор
|   |   при условии: на востоке свободно
|   |   | шаг на восток
|   |   | ЗакраситьВсеДоступныеКлетки
|   |   | шаг на запад
|   |   конец
|   | конец
|   | выбор
|   |   при условии: на западе свободно
|   |   | шаг на запад
|   |   | ЗакраситьВсеДоступныеКлетки
|   |   | шаг на восток
```

[1] В частности, если текущая клетка закрашена, то ничего делать не следует.

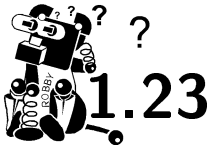
```
        конец
      конец
    конец
  конец
конец
начало
  ЗакрситьВсеДоступныеКлетки
конец
```

Проверим, что наша процедура работает правильно — в предположении, что вызываемые одноименные процедуры работают правильно. Если клетка закрашена, то доступных клеток нет и ничего не закрашивается. Если какая-то клетка закрашивается в ходе работы программы, то она доступна из одного из соседей (незакрашенных), а значит, и из исходной клетки (вначале незакрашенной). Осталось проверить, что любая доступная клетка будет закрашена. Если некоторая клетка, не совпадающая с начальной, доступна по незакрашенным, то она доступна из одного из незакрашенных соседей, причем путь не проходит через исходную клетку. Рассмотрим первого из таких соседей (в том порядке, в котором они просматриваются в программе): при соответствующем рекурсивном вызове клетка будет закрашена.

Почему мы берем именно первого соседа? Дело в том, что до рассматриваемого рекурсивного вызова процедуры могли быть и другие вызовы, в результате которых некоторые клетки стали закрашенными. Нам нужно быть уверенными, что от этого наша клетка не перестала быть доступной.

Осталось доказать, что выполнение программы заканчивается. В самом деле, после каждого рекурсивного вызова число незакрашенных клеток уменьшается (хотя бы на единицу). ◀

▶ **Упражнение 1.3.** *Что будет, если команду* `закрасить` *переставить в конец процедуры (после четырех конструкций выбора)?* ◀



Дано: *Робот стои́т у горизонтальной стены. Стена бесконечна в обе стороны. Известно, что в стене есть проход, но не известно, с какой стороны от Робота.*

Надо: *Найти проход.*

Решение: Оговорим сразу, что любые действия прекращаются, как только Робот оказывается в клетке, у которой на севере свободно.

Идея состоит в том, что поскольку неизвестно, в какую сторону следует идти, необходимо время от времени менять направление движения, увеличивая при этом амплитуду.

Принцип действия процедуры `ПоЗакрашеннымНаЗапад`: Робот находится ниже горизонтальной стены, несколько (возможно, ноль) клеток западнее Робота (в том числе и та, в которой стои́т Робот) закрашено. Робот идет по закрашенным клеткам до первой незакрашенной, закрашивает ее и выполняет процедуру `ПоЗакрашеннымНаВосток`, которая действует так же, но «запад» и «восток» меняются местами.

```
программа НайтиПроход
процедура ПоЗакрашеннымНаЗапад
|  пока (закрашено и не на севере свободно)
|  выполнять
|  |  шаг на запад
|  конец
|  выбор
|  |  при условии: не на севере свободно
|  |  |  закрасить
|  |  |  ПоЗакрашеннымНаВосток
|  |  конец
|  конец
конец
процедура ПоЗакрашеннымНаВосток
```

```
    пока (закрашено и не на севере свободно)
    выполнять
      шаг на восток
    конец
    выбор
      при условии: не на севере свободно
        закрасить
        ПоЗакрашеннымНаЗапад
      конец
    конец
конец
начало
  ПоЗакрашеннымНаЗапад
конец
```

◀

1.24

Дано: *Робот в некоторой клетке горизонтального коридора конечной ширины.*

Надо: *Робот в симметричной по вертикали относительно середины коридора клетке.*

Решение: Если заметить, что для двух последовательных по вертикали клеток симметричные им клетки также последовательны (но в обратном порядке), задача становится очевидной.

```
программа ПоискСимметричнойКлетки
процедура вСимметричнуюКлетку
  выбор
    при условии: на юге свободно
      шаг на юг
      вСимметричнуюКлетку
      шаг на юг
    конец
    при условии: не на юге свободно
      НаСеверДоУпора[1]
```

[1] Эта процедура описана в решении задачи 1.2.

```
    конец
  конец
конец
начало
  вСимметричнуюКлетку
конец
```

◀

1.25

Дано: *Некоторые клетки конечного коридора закрашены.*

Надо: *Закрасить симметричные клетки.*

Решение: Заметим, что двукратное применение процедуры вСимметричнуюКлетку возвращает нас на исходное место.

```
программа ЗакраситьСимметричныеКлетки
процедура ЗакрасьСимметричную
  выбор
    при условии: закрашено
      вСимметричнуюКлетку[1]
      закрасить
      вСимметричнуюКлетку
    конец
  конец
конец
начало
  НаСеверДоУпора[2]
  пока на юге свободно выполнять
    ЗакрасьСимметричную
    шаг на юг
  конец
  ЗакрасьСимметричную
конец
```

◀

[1] Эта процедура описана в решении задачи 1.24.
[2] Эта процедура описана в решении задачи 1.2.

Задачи для самостоятельного решения

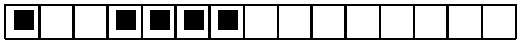
1.26 *Можно ли решить задачу 1.23 без использования проверки* `закрашено`*?* ◂

1.27 **Дано:** *В произвольном лабиринте с конечным числом доступных клеток имеется одна закрашенная.*
Надо: *Робот в этой клетке.* ◂

1.28 *Попробуйте получить следующие узоры как для бесконечного в одну сторону, так и для конечного коридора:*

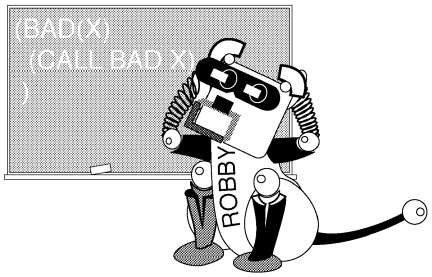
а) 
(одна клетка закрашена, одна пропущена, две закрашены, одна пропущена, три закрашены, одна пропущена и т.д.);

б) 
(одна клетка закрашена, две пропущены, три закрашены, четыре пропущены, пять закрашены и т.д.);

в) 
(одна клетка закрашена, две пропущены, четыре закрашены, восемь пропущены, шестнадцать закрашены, тридцать две пропущены и т.д.). ◂

Глава II

ИГРА В RL



1. ПРАВИЛА ИГРЫ

Выражения

Мы начнем с определений и поясняющих их примеров.

▷ *Символом* (или *атомом*) мы будем называть любую последовательность, составленную из букв; при этом под буквами мы понимаем буквы английского алфавита, причем большие и маленькие буквы считаем одинаковыми; знак подчеркивания тоже считается буквой.

Примеры:

`A, B, symbol, AnotherSymbol`	—	символы
`НеСимвол, a2, abc-d`	—	не символы

▷ *Термом* называется любой символ, а также любое выражение, заключенное в скобки. При этом *выражением* называется любая (в том числе и пустая) последовательность термов, разделенных пробелами.

Примеры выражений:

`A B CCC D`	—	состоит из четырех символов
`(A B)`	—	состоит из одного терма, представляющего собой выражение из двух символов в скобках
`(()) B CCC`	—	выражение из трех термов, первый из которых есть выражение из единственного терма в скобках
	—	пустое выражение

Обратите внимание, что пробелы вокруг скобок ставить не обязательно.

Программы и функции

В языке RL все объекты — программы, входные данные, результаты работы — являются выражениями.

▷ *Программа* на языке RL представляет собой выражение, состоящее из одного или нескольких термов. Эти термы называются *функциями*.

Каждая функция имеет следующий вид:
$(NAME\ (ARG_1\ \ldots\ ARG_n)\ BODY)$,
где

$NAME$	—	символ, называемый именем функции;
$ARG_1\ \ldots\ ARG_n$	—	символы, называемые аргументами функции;
$BODY$	—	терм, называемый телом функции.

Число аргументов может быть равным нулю — в этом случае в описании функции закрывающая скобка следует сразу за открывающей.

Имена функций, описанных в программе, не должны совпадать.

Каждая функция вычисляет некоторое выражение, называемое ее значением и зависящее от значений аргументов. Результатом работы программы является значение первой функции программы при значениях ее аргументов, заданных пользователем при запуске программы.

Значением функции называется значение терма — ее тела. Что такое значение терма, определяется в следующих разделах.

Параметры. Значение терма при данных значениях параметров

Для каждого терма определяется список символов, называемых его параметрами, и значение терма при заданных значениях этих параметров. Значениями термов, как и их параметров, являются выражения, а также специальный символ НЕОПРЕДЕЛЕНО.

Значения и параметры данного терма определяются через значения и параметры термов, входящих в этот терм. Ясно, что этот процесс должен когда-то заканчиваться, так что некоторые термы должны получать значения непосредственно.

Стандартные термы

▷ Терм (ABORT) не имеет параметров, и его значением является символ НЕОПРЕДЕЛЕНО.

Пусть имеется выражение X.

▷ Терм (QUOTE X) не имеет параметров, и его значением является выражение X.

Например, значением терма (QUOTE) является пустое выражение, а значением терма (QUOTE (A) () B) является выражение из трех термов: (A) () B

Другим примером терма, значение которого определяется непосредственно, является аргумент функции. Его значение определяется при вызове функции (об этом подробнее будет рассказано позже), в частности, в начале работы программы пользователь задает значения всех аргументов первой (возможно, единственной) функции программы. Значения всех остальных термов определяются через уже известные значения.

Пусть $T_1 \ldots T_n$ — термы, *знач*$T_1 \ldots$ *знач*T_n — выражения, являющиеся их значениями.

Параметры всех описываемых далее термов — все параметры термов $T_1 \ldots T_n$, входящих в их описание.

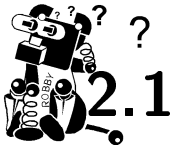
▷ Значение терма (FIRST T_1) — это первый терм выражения *знач*T_1. Если же выражение *знач*T_1 пусто или не определено, значением этого терма считается символ НЕОПРЕДЕЛЕНО.

▷ Значением терма (BF T_1) является выражение, полученное отбрасыванием первого терма выражения *знач*T_1. Если выражение *знач*T_1 пусто или не определено, то значением этого терма является символ НЕОПРЕДЕЛЕНО.

▷ Аналогично определяются значения термов (LAST T_1) и (BL T_1). Значением терма (LAST T_1) является значение последнего терма выражения *знач*T_1; значением терма (BL T_1) является выражение, полученное отбрасыванием последнего терма выражения *знач*T_1.

▷ Значением функции называется значение терма — ее тела.

▶ **Упражнение 2.1.** *Чему равно значение терма* (BL T_1)*, если* *знач*T_1 *состоит из одного терма?* ◀

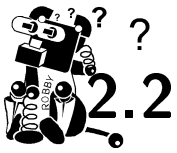


Опишите функцию SECOND*, значение которой — второй терм выражения, являющегося значением ее аргумента.*

Решение: Второй терм выражения, являющегося значением терма X — это первый терм выражения (BF X). Значит, мы можем записать:

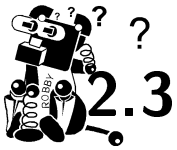
```
(SECOND (X)
  (FIRST (BF X))
)
```

Если при этом значение `X` не определено, то значение `(BF X)` тоже не определено, значит, не определено и значение `(FIRST (BF X))`. Если же значение `X` состоит из одного терма, то значение терма `(BF X)` — пустое выражение, а значение `(FIRST (BF X))` снова не определено. ◀



Опишите функцию, значением которой является предпоследний терм значения ее аргумента.

▷ Значение терма `(EXPR` T_1 ... T_n`)` есть выражение *знач*T_1 ... *знач*T_n. Иными словами, значение этого терма получается «сцеплением» значений его параметров.



Опишите функцию, значением которой является выражение, состоящее из двух термов: первого и последнего терма значения ее аргумента.

Решение:

```
(FIRSTandLAST (X)
  (EXPR
    (FIRST X)
    (LAST X)
  )
)
```

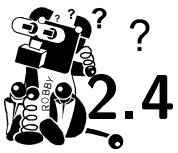
◀

Замечание. Приведенная функция имеет небольшой «дефект»: если значение `X` состоит из одного терма, то значением этой функции будет выражение, состоящее из дважды повторенного значения `X`.

▷ Значение терма `(EQUAL` T_1 T_2`)` есть символ `TRUE`, если выражения *знач*T_1 и *знач*T_2 определены и совпадают, символ `FALSE`, если эти выражения определены, но не совпадают, и символ `НЕОПРЕДЕЛЕНО`, если хотя бы одно из этих выражений не определено.

▷ Значение терма (SYMBOL T_1) есть символ TRUE, если значение выражения *знач*T_1 определено и состоит из одного символа, символ FALSE, если оно определено и состоит из одного терма — выражения в скобках, и символ НЕОПРЕДЕЛЕНО во всех остальных случаях[1].

▷ Значение терма (IF T_1 T_2 T_3) есть выражение *знач*T_2, если выражение *знач*T_1 есть символ TRUE; выражение *знач*T_3, если выражение *знач*T_1 есть символ FALSE; символ НЕОПРЕДЕЛЕНО во всех остальных случаях.



Описать функцию, значение которой на выражении A *равно* B*, на выражении* B *равно* A*, а на остальных выражениях не определено.*

Решение:

```
(AB (X)
  (IF (EQUAL X (QUOTE A))
    (QUOTE B)
    (IF (EQUAL X (QUOTE B))
      (QUOTE A)
      (ABORT)
    )
  )
)
```

◂

Описать функцию AND*, значение которой на паре аргументов равно* TRUE*, если значения обоих аргументов равны* TRUE*, и равно* FALSE *во всех остальных случаях.*

Решение:

```
(AND (X Y)
  (IF (EQUAL X (QUOTE TRUE))
```

[1] То есть не определено или является выражением, состоящим более чем из одного терма.

```
(IF (EQUAL Y (QUOTE TRUE)
  (QUOTE TRUE)
  (QUOTE FALSE)
)
(QUOTE FALSE)
)
)
```
◀

▶ **Упражнение 2.2.** *Можно ли упростить решение предыдущей задачи так:*

```
(AND (X Y)
  (IF (EQUAL X (QUOTE TRUE))
    Y
    (QUOTE FALSE)
  )
) ?
```
◀

Описать функцию OR*, значение которой на паре аргументов равно* TRUE*, если хотя бы один аргумент равен* TRUE*, и равно* FALSE *во всех остальных случаях.* ◀

▷ Значением терма (BR T_1) является выражение, состоящее из одного терма, представляющего собой выражение *знач* T_1 в скобках.

▷ Значение терма (CONT T_1) есть выражение X, если *знач* T_1 есть выражение X, заключенное в скобки; символ НЕОПРЕДЕЛЕНО во всех остальных случаях.

Теперь мы можем определить значение терма

(CALL *NAME* PAR_1 ... PAR_n)

при данных значениях параметров и при данной программе. При этом терм *NAME* должен быть символом, функция с именем *NAME* должна быть описана в данной программе и число ее аргументов должно быть равно n — если хотя бы одно из этих требований не выполнено, то значение

терма не определено. Если же эти требования выполнены, то значение терма (CALL $NAME\ PAR_1 \ldots PAR_n$) равно значению функции с именем $NAME$ на аргументах, равных значениям термов $PAR_1 \ldots PAR_n$.

Другими словами, для вычисления значения терма (CALL $NAME\ PAR_1 \ldots PAR_n$) мы должны сначала вычислить значения термов $PAR_1 \ldots PAR_n$ (при данных значениях параметров и данной программе), а затем вычислить значение терма — тела функции $NAME$ при значениях $ARG_1 \ldots ARG_n$, равных значениям $PAR_1 \ldots PAR_n$, то есть вычислить значение этого терма при соответствующих значениях аргументов.

▶ **Упражнение 2.3.** *Попробуйте применить наши правила к такой функции:*

```
(BAD (X)
  (CALL BAD X)
)
```

Решение: По правилам значение этой функции на выражении E равно значению терма (CALL BAD X), когда значение его параметра X равно E. Но значение этого терма по определению равно значению функции BAD на аргументе E — получился порочный круг. ◀

Мы будем считать, что если наши правила определения значений не позволяют его определить из-за порочного круга (одно значение сводится к другому, другое — к третьему и т.п.), то значение является неопределенным.

Описать функцию, значение которой на выражениях с четным числом аргументов равно TRUE, *а на выражениях с нечетным числом аргументов —* FALSE.

Решение:

```
(EVEN (X)
  (IF (EQUAL X (QUOTE))
```

```
    (QUOTE TRUE)
    (IF (CALL EVEN (BF X))
      (QUOTE FALSE)
      (QUOTE TRUE)
    )
  )
)
```
◀

▶ **Упражнение 2.4.** *Возможен ли такой порочный круг в приведенном выше описании функции* EVEN*?* ◀

Описать функцию, значение которой состоит из тех же термов, что и значение ее аргумента, но записанных в обратном порядке. Например, из A (B C) D *получается* D (B C) A*.*

Решение:

```
(REV (X)
  (IF ( EQUAL (QUOTE) X)
    (QUOTE)
    (EXPR (CALL REV (BF X)) (FIRST X))
  )
)
```
◀

Для более сложных задач, как правило, применима схема решения, которую мы сейчас приведем.

Опишем функции ОБРАБОТАТЬ_ВЫРАЖЕНИЕ (X) и ОБРАБОТАТЬ_ТЕРМ (T), устроенные следующим образом:

```
(ОБРАБОТАТЬ_ВЫРАЖЕНИЕ (X)
  (IF ( EQUAL (QUOTE) X)
    (QUOTE)
    (EXPR
      (CALL ОБРАБОТАТЬ_ТЕРМ (FIRST X))
      (CALL ОБРАБОТАТЬ_ВЫРАЖЕНИЕ (BF X))
    )
  )
)
```

```
(ОБРАБОТАТЬ_ТЕРМ (T)
  (IF (SYMBOL T)
    ...
    ...
  )
)
```

2.9 *Описать функцию, которая меняет порядок термов в выражении и всех его подвыражениях (например, ее значение на выражении* `(A B C)(D E F)` *должно равняться* `(F E D)(C B A)`*).*

Решение:

```
(REVERS_EXPR (X)
  (IF (EQUAL (QUOTE) X)
    (QUOTE)
    (EXPR
      (CALL REVERS_TERM(LAST X))
      (CALL REVERS_EXPR(BL X))
    )
  )
)
(REVERS_TERM (T)
  (IF (SYMBOL T)
    T
    (BR (CALL REVERS_EXPR(CONT T)))
  )
)
```

◀

2.10 *Описать функцию, которая удаляет из выражения все скобки (например, из* `((A)(BC))D` *получается* `A B C D`*).*

Решение:

```
(TREAT_EXPR (X)
    (IF (EQUAL (QUOTE) X)
```

```
      (QUOTE)
      (EXPR
        (CALL TREAT_TERM(FIRST X))
        (CALL TREAT_EXPR(BF X))
      )
    )
)

(TREAT_TERM (T)
    (IF (SYMBOL T)
      (EXPR T)
      (CALL TREAT_EXPR(CONT T))
    )
)
```

◀

2. ЗАДАЧИ И УПРАЖНЕНИЯ

2.11 *Определить функцию, значение которой равно* TRUE *на выражениях, в которых число термов кратно трем, и равно* FALSE *на всех остальных аргументах.* ◀

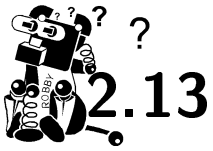
2.12 *Описать функцию, которая удваивает каждый терм в выражении (сами термы остаются без изменений).* ◀

Решение:

```
(DOUBLE (X)
  (IF (EQUAL X (QUOTE))
    (QUOTE)
    (EXPR
        (FIRST X)
```

```
          (FIRST X)
          (CALL DOUBLE(BF X)))
    )
  )
```

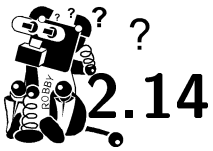
◀



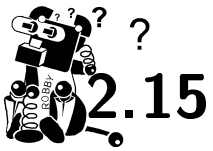
Удалить из выражения все символы BAD. ◀

Унарная арифметика

Рассмотрим следующий способ представления натуральных чисел: выражение из n символов I представляет число n (унарная запись). При этом пустое выражение представляет число ноль.



Описать функцию IS_NUMBER, *которая принимает значение* TRUE, *если ее аргумент представляет собой унарную запись некоторого числа, и значение* FALSE *в противном случае.* ◀



Описать функцию, значение которой есть унарное представление числа термов в выражении, являющимся ее аргументом.

Решение: По существу, от нас требуется заменить каждый терм в выражении, являющемся значением аргумента функции, на символ I.

```
(COUNT_TERMS(X)
  (IF (EQUAL X (QUOTE))
    (QUOTE)
    (EXPR
```

```
      (QUOTE I)
      (CALL COUNT_TERMS (BF X))
    )
  )
)
```
◀

2.16

Описать функцию сложения двух чисел в унарном представлении.

Решение: Мы проверим, являются ли значения аргументов функции унарными числами (функция IS_NUMBER), после чего сложим («сцепим») аргументы.

```
(UNARY_ADD(X Y)
  (IF (CALL IS_NUMBER X)
    (IF (CALL IS_NUMBER Y)
      (EXPR X Y)
      (ABORT)
    )
    (ABORT)
  )
)

(IS_NUMBER(X)
  (IF (EQUAL X (QUOTE))
    (QUOTE TRUE)
    (IF (EQUAL (FIRST X) (QUOTE I))
      (CALL IS_NUMBER (BF X))
      (QUOTE FALSE)
    )
  )
)
```
◀

▶ **Упражнение 2.5**. *Приведите решение предыдущей задачи, которое использовало бы функцию* IS_NUMBER *только один раз.* ◀

2.17 *Описать функцию умножения двух чисел в унарном представлении.*

Решение: Мы воспользуемся описанными в задаче 2.16 функциями UNARY_ADD и IS_NUMBER.

```
(UNARY_MULT(X Y)
  (IF (CALL IS_NUMBER (EXPR X Y))
    (IF (EQUAL Y (QUOTE))
      (QUOTE)
      (CALL UNARY_ADD
        X
        (CALL UNARY_MULT X (BF Y))
      )
    )
    (ABORT)
  )
)
```

◀

2.18 *Описать функцию возведения в степень в унарном представлении.* ◀

2.19 *Описать функцию «усеченного» вычитания (если вычитаемое больше уменьшаемого, то получается ноль).*

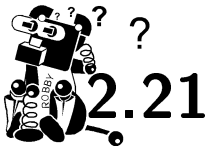
Решение: Решение основывается на равенстве: $a - b = (a - 1) - (b - 1)$.

```
(UNARY_SUB(X Y)
  (IF (CALL IS_NUMBER (EXPR X Y))
    (IF (EQUAL X (QUOTE))
      (QUOTE)
```

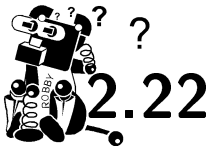
```
        (IF (EQUAL Y (QUOTE))
          X
          (CALL UNARY_SUB (BF X) (BF Y))
        )
      )
      (ABORT)
    )
  )
```

◀

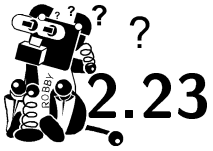
2.20 *Описать функцию, дающую значение* TRUE, *если первый аргумент меньше второго или равен ему, и* FALSE *в противном случае.* ◀



2.21 *Описать функцию нахождения частного от деления двух унарных чисел.* ◀



2.22 *Описать функцию нахождения остатка от деления двух унарных чисел.* ◀



2.23 *Описать функцию подсчета числа скобок в выражении, используя унарную запись результата.*

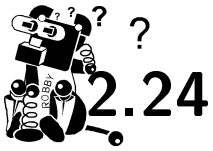
Решение: Решение базируется на следующем соображении: число скобок в выражении равно сумме чисел скобок в термах, его составляющих. При этом для терма число скобок равно нулю, если он является символом, и на два

больше числа скобок в выражении, полученном применением CONT к этому терму.

```
(BRACKETS_IN_EXPR(X)
  (IF (EQUAL X (QUOTE))
    (QUOTE)
    (EXPR
      (CALL BRACKETS_IN_TERM (FIRST X))
      (CALL BRACKETS_IN_EXPR (BF X))
    )
  )
)

(BRACKETS_IN_TERM(T)
  (IF (SYMBOL T)
    (QUOTE)
    (EXPR
      (QUOTE I I)
      (CALL BRACKETS_IN_EXPR (CONT T))
    )
  )
)
```

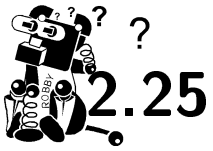
◀



2.24 *Описать функцию проверки простоты числа в унарной записи.* ◀

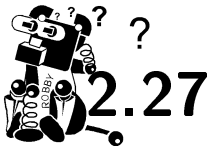
Бинарная арифметика

Наряду с унарной записью можно пользоваться двоичной, используя буквы O, I как цифры 0, 1.



2.25 *Описать функцию* IS_BIN_NUMBER, *значение которой есть символ* TRUE, *если значение ее аргумента представляет собой двоичную запись некоторого числа, и символ* FALSE *в противном случае.* ◀

2.26 *Описать функции преобразования из унарной системы в двоичную и наоборот.* ◂



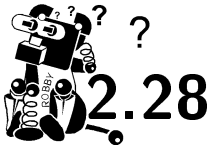
2.27 *Описать функцию сложения в двоичной системе, не использующую унарную запись в качестве промежуточной.*

Решение: Заметим, что $a + b = (a + 1) + (b - 1)$. Следовательно, нам достаточно описать функции увеличения и уменьшения двоичного числа на единицу.

```
(BINARY_ADD(X Y)
  (IF (ZERO Y)
    X
    (CALL BINARY_ADD (CALL INC X) (CALL DEC Y))
  )
)
(INC(X)
  (IF (EQUAL X (QUOTE))
    (QUOTE I)
    (IF (EQUAL (LAST X) (QUOTE O)))
      (EXPR (BL X) (QUOTE I))
      (EXPR (CALL INC (BL X)) (QUOTE O))
    )
  )
)
(DEC(X)
  (IF (EQUAL (LAST X) (QUOTE I))
    (EXPR (BL X) (QUOTE O))
    (EXPR (CALL DEC (BL X)) (QUOTE I))
  )
)
(ZERO(X)
  (IF (EQUAL X (QUOTE))
    (QUOTE TRUE)
    (IF (EQUAL (FIRST X) (QUOTE O))
```

```
        (CALL ZERO (BF X))
        (QUOTE FALSE)
      )
    )
  )
```
◀

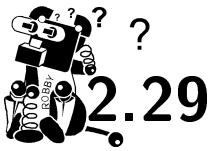
Обратите внимание на функцию ZERO. Она принимает значение TRUE на выражениях, состоящих из произвольного числа символов 0.



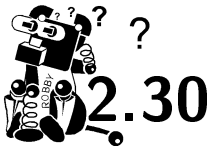
Решить предыдущую задачу, используя алгоритм сложения в столбик. ◀

▶ **Упражнение 2.6.** *Решить задачи раздела 2.13, используя двоичную запись.* ◀

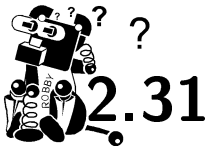
Разные задачи



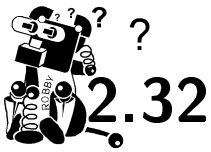
Путь Робота в лабиринте записан в виде выражения из букв E W N S *(восток, запад, север, юг). Удалить из этого пути лишние шаги (т.е. участки* E W, W E, N S, S N*).* ◀



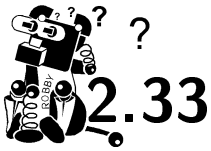
Путь Робота на плоскости без стенок записан в виде выражения. Определить, возвращается ли Робот в конце пути в исходное положение. ◀



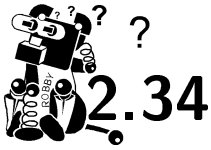
2.31 *Описать функцию, которая преобразует выражение вида* $(A_{11}\ A_{12}\ldots A_{1n})\ (A_{21}\ A_{22}\ldots A_{2n})\ldots(A_{m1}\ A_{m2}\ldots A_{mn})$, *где* A_{ij} *— символы, в выражение* $(A_{11}\ A_{21}\ldots A_{m1})\ (A_{12}\ A_{22}\ldots A_{m2})\ldots(A_{1n}\ A_{2n}\ldots A_{mn})$ *(«переход от записи по столбцам к записи по строкам»).* ◀



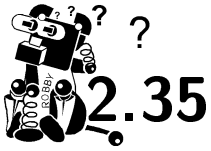
2.32 *Удалить из последовательности повторяющиеся символы (результат содержит те же символы, но по одному разу).* ◀



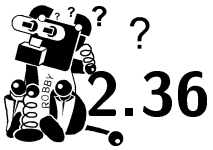
2.33 *Переставить символы в последовательности так, чтобы одинаковые символы стояли рядом.* ◀



2.34 *По двум последовательностям символов построить их пересечение, то есть последовательность, содержащую их общие символы. (Каждая из последовательностей не содержит повторений, и результат тоже не должен содержать повторений.)* ◀



2.35 *Дана последовательность различных символов; получить последовательность термов, содержащую все перестановки данных символов (каждая перестановка — один терм).* ◀

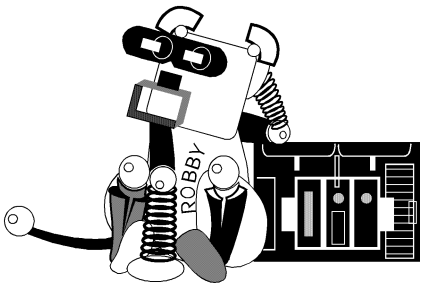


2.36

Дана последовательность символов. Построить выражение, которое получится, если заменить в ней все символы `LB` *на открывающие скобки, а символы* `RB` *— на закрывающие.* ◀

Глава III

ИГРА В СХЕМЫ



1. ПРАВИЛА ИГРЫ

Общие определения

Всякая схема имеет несколько (быть может, ни одного) входов и несколько (не менее одного) выходов. На каждом входе может быть один из двух сигналов: нуль или единица. Сигнал на каждом выходе определяется сигналами на входе (является функцией от сигналов на входе) и также может принимать значение, равное нулю или единице.

Мы будем изучать схемы, образованные некоторым образом из стандартных схем, описанных ниже.

Стандартные схемы

Схема `and` («И») имеет два входа и один выход. Сигнал на выходе равен единице, если оба входных сигнала равны единице, и нулю во всех остальных случаях.

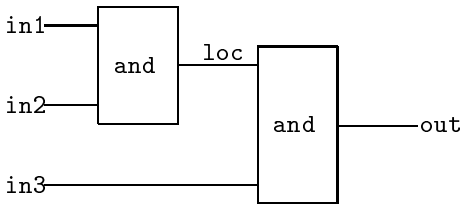
Схема `or` («ИЛИ») также имеет два входа и один выход, но сигнал на выходе равен единице, если хотя бы один из входных сигналов равен единице.

Схема отрицания `not` («НЕ») имеет один вход и один выход. Сигнал на ее выходе противоположен сигналу на входе.

Наконец, схемы `zero` («НУЛЬ») и `one` («ЕДИНИЦА») вовсе не имеют входов. Сигнал на их выходе постоянен и равен нулю и единице соответственно.

Правила образования новых схем

Мы продемонстрируем правила образования новых схем на примере. Соберем схему с тремя входами, на выходе которой будет единица, только если на всех трех входах будет единица. Графически ее можно изобразить в виде



Выход левой схемы `loc` является «локальным» (не выходит наружу построенной схемы, которую мы назовем `and3`).

Мы будем задавать схемы не рисунками, а описаниями. Изображенная выше схема описывается так:

```
scheme (in1 in2 in3) and3 (out):
local loc
  (in1 in2) and (loc)
  (loc in3) and (out)
end
```

Расположение слов по строкам значения не имеет, и наши отступы — это лишь декоративная наглядность.

При этом локальный (как и выходной) контакт должен встречаться в правой части ровно один раз. (Если он встречается более одного раза, может произойти «короткое замыкание», если на один и тот же провод одна схема захочет подать 0, а другая 1. Если же он не встречается, значение на проводе будет не определено.)

Имена контактов и схем могут быть любыми последовательностями букв (русских и латинских) и цифр,

начинающимися с буквы.

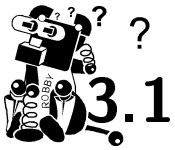
Вокруг списков входов и выходов обязательно должны стоять скобки. Порядок строк внутри описания роли не играет.

Обратите внимание на двоеточие, отделяющее заголовок от дальнейшего описания.

Описанную таким образом схему можно использовать наряду со стандартными.

При этом схема не должна содержать «обратной связи». Это означает, что ситуация, когда выход одной подсхемы использован как вход другой, ее выход — как вход третьей и т.п., а выход последней схемы — как вход первой, является недопустимой.

Кроме того, описания схем не должны содержать прямой или косвенной рекурсии.

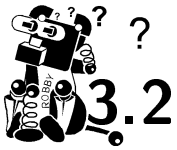


Определить схему «И» с четырьмя аргументами.

Решение:

```
scheme (in1 in2 in3 in4) and4 (out):
local loc
  (in1 in2 in3) and3 (loc)
  (loc in4) and (out)
end
```
◀

Замечание. В описаниях разных схем могут использоваться одни и те же имена контактов (это не играет роли). Но внутри одной схемы имена должны быть уникальны. В файле не может быть двух описаний схемы с одним и тем же именем. Имена стандартных схем также не могут быть переопределены.



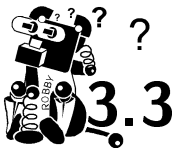
Привести другое решение предыдущей задачи, не использующее функции `and3`.

Решение:

```
scheme (in1 in2 in3 in4) and4 (out):
local loc1 loc2
  (in1 in2) and (loc1)
  (in3 in4) and (loc2)
  (loc1 loc2) and (out)
end
```

◀

2. ПОСТРОЕНИЕ ПРОСТЕЙШИХ СХЕМ

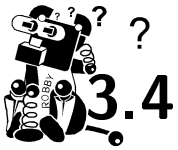


Составить схему, вычисляющую функцию «ИЛИ» трех аргументов (на выходе единица, если хотя бы на одном из входов единица).

Решение:

```
scheme (in1 in2 in3) or3 (out):
local temp
  (in1 in2) or (temp)
  (temp in3) or (out)
end
```

◀



Составить схему, вычисляющую функцию «ИЛИ» четырех аргументов. ◀

Составить схему с тремя входами, у которой выход равен единице или нулю в зависимости от того, каких входов больше (если есть хотя бы две единицы, то на выходе единица, если два нуля, то ноль). ◀

Составить схему `xor`[1]*, на выходе которой сумма двух входов по модулю 2 (на выходе единица, если ровно на одном входе единица).*

Решение:

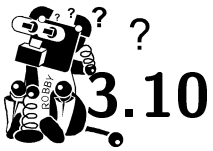
```
scheme (a b) xor (out):
local AnotB BnotA notA notB
  (a) not (notA)
  (b) not (notB)
  (a notB) and (AnotB)
  (b notA) and (BnotA)
  (AnotB BnotA) or (out)
end
```
◀

Построить схему с четырьмя входами, значение выхода которой изменяется при изменении значения любого ее входа. ◀

[1] От английского *exclusive or*

Составить схему с двумя входами и одним выходом, который равен единице только в том случае, когда значение первого входа меньше значения второго входа. ◀

Составить «элементарную сортирующую ячейку» с двумя входами и двумя выходами, которая пропускает сигналы со входов на выходы, если значение первого входа не меньше значения второго входа, и меняет их местами в противном случае. ◀

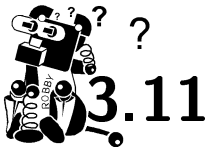


Составить схему с тремя входами `a`, `b` *и* `c` *и одним выходом, действующую согласно таблице:*

a	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*
b	*0*	*0*	*1*	*1*	*0*	*0*	*1*	*1*
c	*0*	*1*	*0*	*1*	*0*	*1*	*0*	*1*
выход	*1*	*0*	*0*	*0*	*0*	*1*	*0*	*1*

Решение: Составим схему `notAnotBnotC`, равную единице только на нулевых входах, схему `AnotBC`, равную единице только на входах (1,0,1) и функцию `ABC`, равную единице только на (1,1,1). Далее применим операцию «ИЛИ» к трем этим функциям. ◀

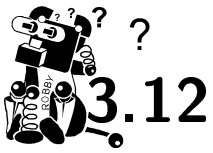
3. НЕСКОЛЬКО ОБЩИХ УТВЕРЖДЕНИЙ



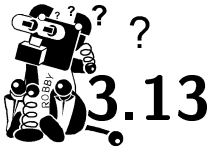
3.11 *Доказать, что любая функция из $\{0,1\}^n$ в $\{0,1\}$ может быть реализована схемой с n входами и одним выходом.*

Решение: Поступаем как в предыдущей задаче (для каждого единичного значения функции строим схему, равную единице только при соответствующих значениях входов, затем применяем «ИЛИ»). ◂

▷ Функция из $\{0,1\}^n$ в $\{0,1\}$ называется монотонной, если при увеличении любого из аргументов (с нуля до единицы) значение функции не уменьшается.

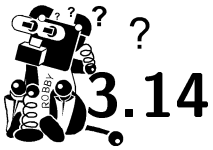


3.12 *Доказать, что любая монотонная функция может быть реализована схемой, не использующей стандартную схему* `not`. ◂



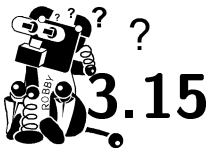
3.13 *Доказать, что стандартную схему* `not` *нельзя собрать из стандартных схем остальных четырех типов.*

Решение: Все остальные стандартные схемы — и, следовательно, любые их комбинации — монотонны. ◂

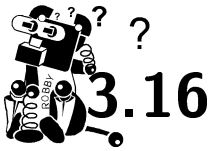


3.14 *Доказать, что любая функция может быть реализована схемой, составленной только из элементов* `and` *и* `not`. ◂

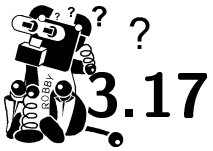
4. ОПЕРАЦИИ НАД ЧИСЛАМИ



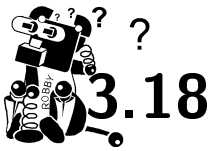
3.15 *Описать схему, осуществляющую сложение двух двухразрядных двоичных чисел (четыре входа — два для каждого слагаемого) и три выхода (числа на выходе могут быть больше, чем на входе).* ◂



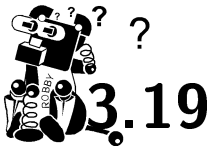
3.16 *Та же задача для четырехразрядных чисел.* ◂



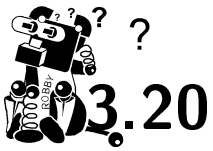
3.17 *Та же задача для восьмиразрядных чисел.* ◂



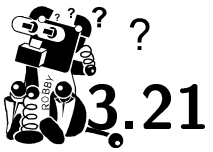
3.18 *Описать схему, осуществляющую сравнение по величине двух восьмиразрядных чисел (на выходе единица, если первое число больше второго).* ◂



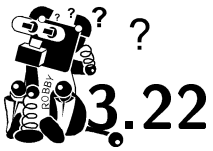
3.19 *Описать схему, осуществляющую вычитание двух восьмиразрядных чисел.* ◂



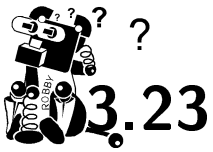
3.20 *Описать схему, осуществляющую умножение двух восьмиразрядных чисел.* ◂



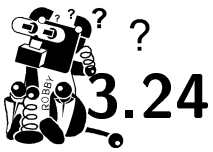
3.21 *Описать схему, проверяющую, кратно ли трем восьмиразрядное двоичное число на ее входе.* ◂



3.22 *Описать схему, которая делит записанное в двоичной системе число на максимальную степень двойки, на которую оно делится.* ◂

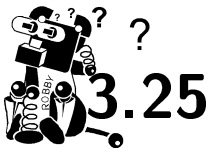


3.23 *Описать схему, осуществляющую деление восьмиразрядного числа на четырехразрядное (имеющую на выходе восемь разрядов для частного и четыре для остатка).* ◂



Описать схему с семью входами, на выходе которой единица, если более трех входов имеют единицы. ◂

▷ Схема называется *сортирующей*, если число выходов равно числу входов, на выходах столько же единиц, сколько на входах и нули на выходах предшествуют единицам.

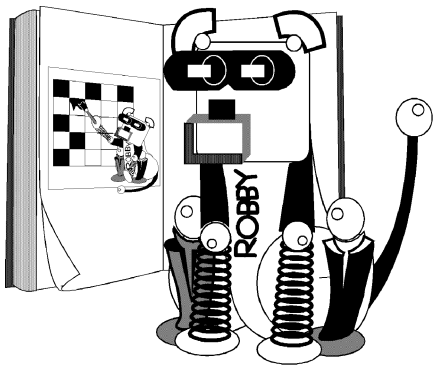


Составить сортирующую схему с восемью входами и выходами, используя в качестве элементов только сортирующие схемы с двумя входами и выходами. ◂

II

Глава IV

ПРОГРАММА ROBOT



В этой главе подробно описывается работа с программой ROBOT, предназначенной для написания и исполнения программ на языке ROBOT (ему посвящена глава 1).

1. КОМПЛЕКТ ПОСТАВКИ

На прилагаемой к этой книге дискете в директории ROBOT содержатся:

- В поддиректории BIN — исполняемый файл интерпретатора.

- В поддиректории EXAMPLES — примеры программ на языке ROBOT

2. ТРЕБОВАНИЯ К КОМПЬЮТЕРУ

Для работы с программой требуется IBM-совместимый компьютер с оперативной памятью не менее 256 килобайт, работающий под управлением DOS версии 3. 20 и выше.

3. СОЗДАНИЕ ПРОГРАММ

Для того, чтобы начать работу с программой, необходимо набрать в командной строке строку

`ROBOT` *ИмяФайла*

и нажать [Enter] (здесь *ИмяФайла* обозначает название имени файла (без расширения), в котором расположена программа). Заметим, что в программе ROBOT имя файла всегда совпадает с названием программы. Кроме того, если не указывать ИмяФайла, то подразумевается файл ROBOT.RBT). Мы предполагаем также, что путь к файлу ROBOT.EXE известен.

В случае удачного запуска на экране появится программа, которая была ранее создана под этим названием, или заготовка программы:

```
программа ИмяФайла
начало
| <команда>
конец
```

Вы можете перемещаться по тексту программы с помощью клавиш [↑], [↓], [←] и [→]. Встав на неконкретизированную конструкцию, вы можете нажать [Enter], после чего возникнет меню, указывающее возможные способы конкретизации, например, меню конкретизации, вызываемое при нажатии клавиши [Enter] на слове <*команда*>:

```
Элементарная команда ▶
Команда цикла
Команда выбора
Вызов процедуры ▶
Комментарий
```

Вы можете выбирать подпункты с помощью клавиш

↑ и ↓ и указывать на выбранный подпункт с помощью клавиши Enter. Пункты, отмеченные с помощью символа ▶, вызывают подменю, например, выбор пункта Элементарная команда ▶ вызывает появление следующего подменю:

```
шаг на север
шаг на юг
шаг на запад
шаг на восток
закрасить
```

Для того, чтобы вставить команду **перед** предыдущей, необходимо нажать клавишу Insert в тот момент, когда курсор находится на следующей команде (или на слове конец охватывающей команды).

Вы можете деконкретизировать конструкцию, нажав клавишу Delete в тот момент, когда курсор находится на деконкретизируемой конструкции.

Внимание! Нажатие Delete на составной конструкции, занимающей, например, 100 строк, без всяких подтверждений заменит ее на слово <команда>. Если же вы нажмете клавишу Delete на одной из ветвей выбора или на слове <команда>, то соответствующая часть программы просто исчезнет.

Для редактирования названия процедуры нажмите клавишу Enter на ее названии, для вставления новой процедуры нажмите Insert на начале предыдущей (или на слове начало главной программы), а для удаления процедуры нажмите Delete на слове процедура.

После того, как текст программы введен, вы можете перейти на поле, нажав клавишу Tab.

По полю вы можете двигаться с помощью клавиш ↑, ↓, ← и →. Если вы нажмете клавишу * на цифровой клавиатуре, то тем самым вы переместите стартовую позицию Робота в ту точку, в которой находится курсор (при этом помните, что Робот не может находиться

между клетками).

Для закрашивания/стирания клетки и установки/снятия стенок нажмите клавишу Insert (ее функция меняется в зависимости от того, где находится курсор, — над полем или над стенками, — и от состояния поля).

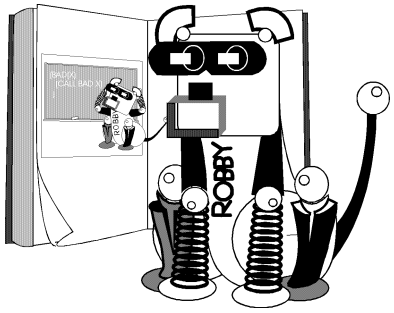
Когда поле приведено в состояние, требуемое по условию задачи, нажмите клавишу F9, и вы сможете наблюдать процесс выполнения программы Роботом.

После обнаружения ошибок в поведении Робота вы можете нажать клавишу Esc, чтобы прервать его работу, и клавишу Tab, чтобы вернуться к редактированию текста программы.

Выйти из программы вы можете, нажав клавишу F10, а получить краткую справку (как и во множестве других программ), нажав клавишу F1.

Глава V

ПРОГРАММА IRL



В этой главе подробно описывается работа с программой IRL, предназначенной для написания и исполнения программ на языке RL (ему посвящена глава 2).

1. КОМПЛЕКТ ПОСТАВКИ

На прилагаемой к этой книге дискете в директории IRL содержатся:

- В поддиректории BIN — исполняемый файл интегрированной среды, а также файл помощи.
- В поддиректории EXAMPLES — примеры программ на языке RL

Для работы необходимы только файлы, содержащиеся в директории BIN.

2. ТРЕБОВАНИЯ К КОМПЬЮТЕРУ

Для работы с программой требуется IBM-совместимый компьютер с оперативной памятью не менее 512 килобайт, работающий под управлением DOS версии 3. 20 и выше.

3. СОЗДАНИЕ ПРОГРАММ

Начало работы

Для того, чтобы начать работу с программой, необходимо набрать в командной строке строку

```
IRL
```

и нажать Enter. Мы предполагаем, что путь к файлу IRL.EXE известен.

В случае удачного запуска на экране появится окно с сообщением о версии и разработчиках программы. Для продолжения работы нажмите Enter.

Для того, чтобы перейти к редактированию программы, необходимо считать ее из уже существующего файла или же создать новую.

Создание новой программы

Для создания новой программы нажмите F4. После этого откроется окно, в котором можно редактировать текст программы. Это окно будет иметь заголовок `Untitled`. При попытке сохранить набранный текст в виде файла система предложит вам дать файлу имя, после чего изменит заголовок окна.

Открытие существующего файла

Для того, чтобы перейти к редактированию уже существующего файла, нажмите F3. После этого откроется

окно, в котором нужно либо указать имя файла, либо выбрать его из списка. Для переключения между строкой ввода имени файла, списком файлов, а также «кнопками» (они расположены в правой части окна) используются клавиши Tab и Shift + Tab.

После того, как выбрано имя файла, надо нажать на клавишу Enter. В этом случае откроется окно, в которое будет загружено содержимое выбранного файла.

Запись на диск

Для того, чтобы сохранить набранный текст в виде файла, нажмите F2. Если вы создали новую программу, то IRL попросит вас выбрать имя файла для вашей программы. Если же вы редактировали файл, загруженный с диска, то изменения будут сохранены в файле с тем же именем (при этом предыдущая версия будет утеряна).

Редактирование

После того, как открыто окно редактирования, можно переходить к процессу написания и выполнения программ.

Перемещение по тексту

Клавиша	Перемещение
← / →	один символ влево/вправо
Ctrl + ←	слово влево
Ctrl + →	слово вправо
↑ / ↓	одна строка вверх/вниз
Home / End	начало/конец строки
PgUp / PgDn	страница вверх/вниз
Ctrl + PgUp	начало текста
Ctrl + PgDn	конец текста
Ctrl + Home	верх экрана
Ctrl + End	низ экрана

Примечание. Знак + означает, что нужно нажать первую клавишу и, не отпуская ее, вторую.

Вставка и удаление

Клавиша	Действие
Ins	смена режима Insert/Overwrite
Del	удаление текущего символа
BS	удаление символа слева от курсора
Ctrl + Y	удаление текущей строки

4. ИСПОЛНЕНИЕ ПРОГРАММ

Теперь, когда вы подготовили программу, можно попробовать выполнить ее.

Запуск

Для того, чтобы начать исполнение программы, нажмите Ctrl + F9.

Если ваша программа содержит ошибки, из-за которых она не может быть выполнена, то в появившемся окне вы увидите сообщение об этом.

Задание значений аргументов

Если программа не содержит синтаксических ошибок, то начинается ее исполнение. Сначала запрашиваются значения аргументов первой функции. Происходит это следующим образом: на экране появляется окно, в котором можно ввести значение аргумента. (Напомним, что оно должно соответствовать правилам RL, изложенным в главе 2). После того, как вы ввели выражение, являющееся значением аргумента, нажмите клавишу Enter.

Если первая функция имеет несколько аргументов, то описанное действие повторяется для каждого из них.

Окно сообщений

После окончания исполнения программы появляется окно, в котором содержится результат работы программы или сообщение об ошибке. Для того, чтобы закрыть это окно, нажмите клавишу Enter или Alt + F3.

Сообщения об ошибках

Если в окне сообщений содержится информация об ошибке, то после закрытия этого окна курсор устанавливается на ту конструкцию, в которой произошла ошибка.

Сообщения об ошибках подробно описываются в системе помощи (см. раздел 7). Кроме того, в системе помощи содержатся рекомендации по устранению наиболее распространенных ошибок. Поэтому в данном разделе список диагностируемых ошибок мы не приводим.

5. КОМАНДЫ МЕНЮ

Меню расположено в верхней строке экрана. Выход в меню осуществляется нажатием F10.

Вообще говоря, меню (как, в сущности, и редактор) программы IRL выдержано в стиле фирмы Borland International. Поэтому читателю может оказаться полезным какое-нибудь пособие по работе в системах Borland Pascal или Borland C++.

Меню состоит из пяти элементов: File, Edit, Search, Window и Help. Между элементами можно перемещаться с помощью стрелок. При нажатии клавиши Enter выбранный элемент «раскрывается» — появляется подменю, пе-

ремещаться по которому можно с помощью вертикальных стрелок.

В данном разделе мы рассмотрим элементы File, Edit, Search и Window. Меню Help будет описано в разделе 7.

Меню File

Open...	F3	Загрузка файла в новое окно
New		Создание нового файла
Save	F2	Сохранение текущего файла
Save as...		То же, с выбором имени файла
Change dir...		Изменение текущей директории
DOS Shell		Временный выход в DOS
Exit	Alt+X	Окончание работы

Меню Edit

Программа IRL имеет одно невидимое окно, которое называется буфером промежуточного копирования. Этот буфер (его еще называют Clipboard) удобно использовать для переноса части текста из одного окна в другое.

Undo	Отмена предыдущей команды
Cut	Удаление выделенного текста
Copy	Копирование выделенного текста в Clipboard
Paste	Вставка содержимого Clipboard в окно
Clear	Удаление блока

Примечание. Работа с блоками подробно описывается в разделе 6.

Меню Search

Find	Поиск выражения в файле
Replace	Поиск с заменой
Search again	Повтор последней операции

Меню Window

Size/move	Изменить размеры/положение окна
Zoom	Расширить/восстановить размер окна
Tile	Расположить окна равномерно по экрану
Cascade	Расположить окна вложенными
Next	Перейти к следующему окну
Previous	Перейти к предыдущему окну
Close	Закрыть окно

Кроме описанных в таблице команд имеется еще одно средство работы с окнами: некоторые окна имеют номер (он изображается в левом верхнем углу рамки окна). При нажатии комбинации клавиш Alt + *цифра* происходит переход в указанное окно.

6. ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ

Этот раздел посвящен нескольким «архитектурным излишествам», без которых, разумеется, можно обойтись, но наличие которых делает процесс редактирования текста программы более комфортным.

Блоки

Программа IRL позволяет выделить часть текста (или весь текст). Такой выделенный участок называется блоком. Блок может быть скопирован, перенесен в другое место или удален.

Клавиша	Действие
Ctrl + K, B	отметить начало блока
Ctrl + K, K	отметить конец блока
Ctrl + K, C	скопировать блок
Ctrl + K, V	перенести блок
Ctrl + K, Y	удалить блок

Примечание. Запятая между клавишами означает, что надо нажать первую из них (в данном случае Ctrl + K), отпустить ее и нажать вторую.

Кроме описанного в таблице, есть еще один способ выделения блока: нажать клавишу Shift и, не отпуская ее, расширять блок с помощью стрелок.

Еще раз о запуске IRL

В начале этой главы мы объяснили, как запустить программу IRL. На самом деле в командной строке после слова IRL могут следовать имена одного или нескольких файлов. В этом случае при запуске IRL не выдает сообщения об авторах и версии, а загружает указанные файлы (каждый в отдельное окно).

«Горячие» клавиши

Некоторые, наиболее часто используемые, функции меню могут быть выполнены с помощью так называемых «горячих» клавиш.

Клавиша	Меню, Команда
F2	File, Save
F3	File, Open
F4	File, New
Alt + X	File, Exit
Ctrl + Ins	Edit, Copy
Shift + Ins	Edit, Paste
Ctrl + Del	Edit, Clear
Shift + Del	Edit, Cut
Ctrl + F5	Window, Size/Move
F5	Window, Zoom
F6	Window, Next
Shift + F6	Window, Previous
Alt + F3	Window, Close

Использование устройства «мышь»

Использование «мыши» позволяет заметно упростить работу с программой. В нижней строке экрана расположена так называемая строка состояния. Команды, находя-

щиеся в этой строке, могут быть вызваны путем нажатия левой кнопки «мыши».

Доступ к элементам меню

Для доступа к элементам меню нужно переместить курсор «мыши» на требуемый элемент меню, после чего нажать на левую кнопку «мыши». Выбранный элемент раскроется. После этого (не отпуская кнопку) можно выбрать одну из команд меню.

Перемещение между окнами

«Мышь» может быть также использована для перемещения между окнами: для этого нужно лишь поместить ее курсор в нужное окно и нажать на одну из кнопок «мыши».

Изменение размеров/положения окна

Для изменения положения окна нужно переместить курсор «мыши» на верхнюю часть рамки окна, после чего нажать на левую кнопку «мыши» и, не отпуская ее, переместить окно.

Для изменения размеров окна нужно проделать все то же самое, только курсор «мыши» необходимо переместить не в верхнюю часть рамки, а в ее правый нижний угол (туда, где рамка состоит из одинарной линии, а не из двойной).

7. СИСТЕМА ПОМОЩИ

Программа IRL имеет систему помощи, в которой содержится практически вся информация, приведенная в данной главе.

Подменю Help главного меню содержит следующие элементы:

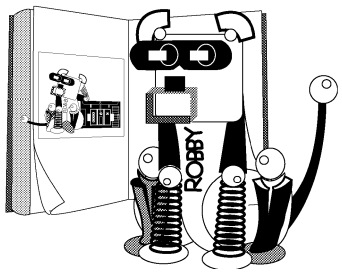
Contents	Список разделов системы помощи
Language	Шпаргалка к языку
How Use	Как пользоваться системой помощи
About	Информация о программе
Authors	Информация об авторах

При вызове системы (это можно сделать еще и путем нажатия клавиши F1) в центре экрана появляется окно, в котором содержится текст подсказки. По этому тексту можно перемещаться точно таким же образом, как и по тексту в окне редактора. Некоторые части содержимого окна выделяются цветом (на монохромных дисплеях — яркостью или подчеркиванием). Этим частям соответствуют другие разделы помощи, в которые можно попасть нажав клавишу Enter. Чтобы вернуться назад, нужно нажать Alt + F1.

Для того, чтобы закончить работу с системой помощи, нажмите клавишу Esc или Alt + F3.

Глава VI

ПРОГРАММА SCHEME



Работа с программой Scheme очень напоминает работу работу с программой IRL, описанной в предыдущей главе. В этой главе мы опишем только отличия, вызванные разницей в синтаксисе языков.

1. КОМПЛЕКТ ПОСТАВКИ

На прилагаемой к данной книге дискете в директории SCHEME содержатся:

- В поддиректории BIN — исполняемый файл интегрированной среды и файл помощи.
- В поддиректории EXAMPLES — примеры схем

Для работы необходим только файл SCHEME.EXE.

2. ТРЕБОВАНИЯ К КОМПЬЮТЕРУ

Для работы с программой требуется IBM-совместимый компьютер с оперативной памятью не менее 512 килобайт, работающий под управлением DOS версии 3. 20 и выше.

3. НАЧАЛО РАБОТЫ

Запуск программы можно произвести, набрав в командной строке команду

```
scheme
```

При этом подразумевается, что файл SCHEME.EXE находится в известной (текущей или содержащейся в переменной PATH) директории.

4. ИСПОЛНЕНИЕ СХЕМ

После окончания описания схемы можно попытаться проверить ее работу.

Запуск

Для того, чтобы перейти к исполнению схемы, необходимо нажать Ctrl+F9.

Если описанная схема не содержит ошибок (о том, какие ошибки диагностируются программой, смотрите в следующем разделе), то на экране появляется графическое изображение главной схемы. На ней изображаются только входные и выходные контакты. С помощью клавиш ↑ и ↓ можно перемещаться от одного входного контакта к другому, а нажатием клавиши Enter — менять значение на контакте (с нуля на единицу и наоборот).

Диагностика ошибок

Интерпретатор схем распознает следующие типы ошибок:

1) синтаксическая ошибка (отсутствие скобки, двоеточия и т.п.);
2) локальный или выходной контакт встречается более одного раза в правой части или вовсе не встречается;
3) входной контакт встречается в правой части: тем самым контакт описываемой схемы, объявленный входом, ведет себя как выход;
4) обратная связь внутри схемы;
5) разное число параметров. Число входов или выходов в описании схемы не совпадает с число входов или выходов при ее использовании;
6) рекурсия в описаниях;
7) совпадение имен контактов внутри схемы;
8) попытка описания двух схем с одинаковым именем или переопределения стандартной схемы;
9) упоминается контакт, не фигурирующий в списке входов, выходов и локальных контактов, с которых начинается описание;
10) используется схема, которая не описана.

Сообщение об ошибке содержит номер ошибки (согласно приведенному списку) и номер строки, в которой эта ошибка предположительно находится.

ДИСТРИБУТИВНАЯ ДИСКЕТА

В этом разделе мы опишем содержимое дискеты, прилагаемой к книге.

В корневой директории диска находится программа INSTALL.EXE, предназначенная для установки программ, описанных в данной книге, на ваш компьютер.

Программа установки позволяет вам задать некоторые параметры. В частности, вы можете установить на свой компьютер лишь часть программ.

Программы, описанные в последних трех главах, могут быть скопированы из соответствующих директорий вручную.

В директории BOOK имеются поддиректории PART1, PART2 и NOTES. В первых двух хранятся TEX-версии соответственно первой и второй частей предлагаемого курса (первую часть вы держите в руках). Они сжаты с помощью INFO-ZIP и могут быть извлечены с помощью программ UNZIP или PKUNZIP. В директории NOTES хранятся материалы, которые могут, по нашему мнению, помочь преподавателю в составлении заданий для учащихся.

Программы, хранящиеся в директориях ROBOT, IRL и SCHEME, могут в целях экономии места быть сжаты программами LZEXE или PKLITE.

Мы с глубокой признательностью примем все пожелания, отзывы и замечания по адресу:

121019, Москва, Малый Знаменский пер., 7
Московская государственная
Пятьдесят седьмая школа

или по электронной почте:

book@sch57.mcn.msk.su

Кроме того, мы высылаем по электронной почте новые версии программ, сопровождающих курс. Для включения в список к рассылке пришлите сообщение по указанному выше адресу.

Издательство Московского Центра
непрерывного математического образования

Директор издательского проекта И. В. Ященко
Ведущий менеджер С. В. Маркелов
Технический редактор О. В. Сипачева

Лицензия ЛР б 071150 от 11.04.95г.
Подписано в печать 29.06.95. Формат 84 × 108/32
Бумага офсетная б1. Печать офсетная. Печ. л. 6,5
Тираж 5000. Заказ б

МЦНМО
121019, Москва, Малый Знаменский пер., 7.

Отпечатано с оригинал-макета в АО «Астра-7»
121019, Москва, Филипповский п., 13.