№ 42 (235) ноябрь 1999

еженедельное
ПРИЛОЖЕНИЕ К ГАЗЕТЕ
«ПЕРВОЕ СЕНТЯБРЯ»

ИНФОРМАТИК

Норберт Винер и его “наука об управлении”

26 ноября 1999 года исполняется 105 лет со дня рождения основоположника кибернетики, американского математика Норберта Винера (1894—1964)

Термин *кибернетика* (от греческого слова, означающего *кормчий, рулевой, управляющий*) впервые после древних греков употребил в 1834 г. французский ученый Андре Мари Ампер (1775—1836) в предложенной им классификации наук для обозначения еще не существовавшей тогда науки об управлении человеческим обществом [1]. Вскоре после Ампера этот термин был забыт, а возродил его Норберт Винер в названии своей книги [2], первое издание которой появилось в 1948 г. Именно с этой датой принято связывать рождение кибернетики как самостоятельной науки.

В своей работе Винер показал пути создания общей теории управления и заложил основы методов рассмотрения процессов управления и связи для различных систем с единой точки зрения. Он утверждал, что такие процессы являются прежде всего процессами передачи, хранения и переработки информации, то есть различных сигналов, сообщений, сведений, и определил кибернетику как “науку об управлении и связи в животном и машине”. Человеческое общество выпало из данного определения, и, чувствуя этот недостаток, Винер подготовил труд “Кибернетика и общество”, который вышел в свет в 1954 г.

Развертывавшееся в США и Западной Европе во второй половине 1950-х годов широкое использование ЭВМ и базирующихся на них автоматизированных систем управления (АСУ) требовало создания научных основ проектирования таких машин и систем. Поскольку теоретические работы по кибернетике того времени не давали ответа на вопросы, поднятые практикой, большинство специалистов в области ЭВМ и АСУ за рубежом стали скептически относиться к самой кибернетике. При этом на основе новых научных методов и результатов, появлявшихся в связи с задачами проектирования ЭВМ и АСУ, сформировалась новая наука, получившая в США и Англии название *computer science* (наука о компьютерах), а, например, во Франции — *informatics* (термин *информатика* (в качестве общего названия для группы дисциплин, занимающихся различными аспектами применения и разработки компьютеров) и сегодня применяется в основном только в европейских странах). Термин же *кибернетика* стали чаще всего употреблять в более узком смысле, имея в виду здесь в основном аналогии, существующие между машинами и живыми организмами, и философские вопросы, возникающие в связи с

Окончание читайте на с. 32

Читайте в номере

Материалы сборника

Оценка качества подготовки выпускников основной школы по информатике

Сборник подготовили:

А.А. Кузнецов, Л.Е. Самовольнова, Н.Д. Угринович

Продолжение. Начало в № 38, 39, 41/99

Примерная программа курса информатики 10—11-е классы (68 часов)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Настоящая программа составлена на основе “Обязательного минимального содержания обучения информатике”, рекомендованного Минобразованием РФ. Она представляет собой один из возможных вариантов построения программы базового курса информатики, изучаемого в старшем звене школы. Содержание программы полностью соответствует требованиям к результатам обучения по информатике, отраженным в проекте стандарта по этому учебному предмету.

Следует отметить, что “Обязательный минимум содержания образования по информатике” определяет только набор элементов содержания обучения и требований к уровню усвоения учебного материала. Он не задает последовательность, логику изучения материала курса, введения и развития его понятий. Это — функция программы обучения, которую выстраивает каждый учитель.

Предлагаемая примерная программа базового курса информатики носит рекомендательный характер, ее назначение — служить ориентиром при разработке конкретных школьных программ обучения информатике. В зависимости от методических позиций учителя, его взглядов на структуру курса последовательность и объем изучения различных тем курса могут меняться.

СОДЕРЖАНИЕ ОБУЧЕНИЯ

1. Информация и информационные процессы (4 часа)

Основные понятия: информация, информационные процессы, информационная картина мира, информационное общество, информационная культура.

Понятие информации. Информационные процессы: получение, передача, преобразование, хранение и использование информации. Информационные процессы в живой природе, обществе, технике. Информационные основы процессов управления.

Информационная деятельность человека. Информационная культура человека. Информационное общество: его особенности и основные черты.

Практические работы:

1. Отработка навыков ввода информации с помощью клавиатуры (клавиатурный тренажер).
2. Отработка навыков работы с графическим интерфейсом (мышь).

2. Представление информации (12 часов)

Основные понятия: кодирование информации, двоичная система счисления, количество информации, бит, байт.

Формы представления информации. Язык как способ представления информации. Кодирование. Двоичный алфавит. Двоичная система счисления.

Количество информации. Единицы измерения информации. Двоичное кодирование различных форм представления информации (числовой, текстовой, графической, звуковой).

Практическая работа:

Определение информационной емкости различных носителей информации.

3. Компьютер (10 часов)

Основные понятия: процессор, оперативная память, внешняя память, устройство ввода информации, устройство вывода информации, файл, операционная система, компьютерный вирус, антивирусная программа.

Функциональная организация компьютера. Магистрально-модульный принцип построения компьютера. Периферийные и внутренние устройства компьютера: назначение и основные характеристики. Программный принцип управления компьютером. Виды памяти в компьютере. Основные носители информации и их важнейшие характеристики.

Файлы. Операции с файлами. Операционная система. Основные виды программного обеспечения компьютера. Различные способы ввода информации в компьютер. Инсталляция программ.

Техника безопасности и санитарно-гигиенические нормы при работе на компьютере.

Защита сохранности информации. Компьютерные вирусы: методы распространения, профилактика заражения. Антивирусные программы.

Практические работы:
1. Форматирование дискеты, создание системной дискеты.
2. Работа с файлами: копирование, переименование, удаление.
3. Инсталляция программного продукта.
4. Лечение “зараженной” компьютерным вирусом дискеты.

4. Моделирование и формализация (8 часов)

Основные понятия: моделирование, формализация, информационная модель, информационная технология решения задач, компьютерный эксперимент.

Моделирование. Формальная и неформальная постановка задачи. Основные принципы формализации.

Понятие об информационной технологии решения задач. Этапы решения задачи на компьютере: постановка задачи, построение модели, разработка алгоритма и программы, отладка и исполнение программы, анализ результатов. Компьютерный эксперимент.

Практические работы:
1. Построение простой информационной модели.
2. Проведение компьютерного эксперимента.

5. Алгоритмы и исполнители (16 часов)

Основные понятия: алгоритм, исполнитель, система команд исполнителя, алгоритмический язык, блок-схема, линейный, разветвляющийся, циклический и вспомогательный алгоритмы, система программирования.

Понятие алгоритма, примеры алгоритмов. Исполнители алгоритмов. Система команд исполнителя. Примеры исполнителей (Робот, Черепашка).

Свойства алгоритмов. Формальное исполнение алгоритмов. Возможность автоматизации деятельности человека на основе алгоритмов.

Средства представления и записи алгоритмов (алгоритмический язык, блок-схемы).

Основные алгоритмические конструкции (цикл, ветвление, процедура и т.д.) и их использование для построения алгоритмов. Библиотеки алгоритмов.

Представление о системах программирования: состав, назначение компонентов. Знакомство с одним из языков программирования. Основные структуры данных. Присваивание. Переменная: имя, тип, значение. Массив: имя, тип данных, размерность. Функции, подпрограммы.

Практические работы:
1. Построение блок-схемы алгоритма, записанного на естественном языке.
2. Пошаговое исполнение алгоритма для исполнителя Робот или Черепашка.
3. Кодирование заданного в виде блок-схемы алгоритма на языке программирования, ввод и исполнение полученной программы.

6. Информационные технологии (18 часов)

Технологии обработки текста и графики

Основные понятия: текстовый редактор, графический редактор.

Текстовый редактор: назначение, основные функции. Редактирование и форматирование текста. Выбор шрифта. Различные форматы текстовых файлов (документов). Кодировки русских букв. Параметры печати.

Графические редакторы. Основные инструменты, операции. Палитры цветов. Создание и редактирование изображений. Различные форматы графических файлов. Печать графических файлов.

Практические работы:
1. Редактирование и форматирование заданного текста, выбор параметров шрифта.
2. Преобразование формата текстового файла (документа).
3. Редактирование заданного изображения с использованием различных инструментов и операций.
4. Преобразование формата графического файла и его палитры цветов.
5. Распечатка текстового или графического файла на принтере с заданными параметрами печати.

Технология обработки числовой информации

Основные понятия: электронные таблицы, адрес ячейки, типы и формат данных, стандартные функции.

Электронные таблицы: назначение, основные функции, настройка таблиц. Структура электронных таблиц (строка, столбец, ячейка). Типы (числа, формулы, текст) и формат данных. Вычисления с использованием стандартных функций. Редактирование структуры таблицы. Построение диаграмм. Использование электронных таблиц для решения задач.

Практические работы:
1. Ввод данных и вычисления в электронной таблице.
2. Решение задачи на построение графика в электронных таблицах.

Технология хранения, поиска и сортировки информации

Основные понятия: базы данных, таблица, картотека, управление базой данных.

Систематизация и хранение информации. Базы данных, принципы их построения и функционирования. Представление о системах управления базами данных (СУБД). Форма представления баз данных (таблица, картотека). Ввод и редактирование записей. Сортировка и поиск записей. Изменение структуры базы данных.

Практические работы:
1. Ввод и редактирование записей в базе данных.
2. Изменение структуры базы данных.
3. Сортировка записей в заданной базе данных.
4. Поиск записей в заданной базе данных.
5. Создание базы данных “Записная книжка”.

Компьютерные коммуникации

Основные понятия: компьютерные сети, модем, электронная почта, телеконференции, файловый архив, технология World Wide Web, Интернет.

Передача информации. Линии связи, их основные компоненты и характеристики.

Компьютерные телекоммуникации: назначение, структура, ресурсы. Локальные и глобальные компьютерные сети. Основные услуги компьютерных сетей: электронная почта, телеконференции, файловые архивы.

Гипертекст. Основы технологии World Wide Web.

Сеть Интернет. Информационные ресурсы. Поиск информации.

Практические работы:
1. Передача и получение сообщений по электронной почте.
2. Поиск информации в глобальной сети Интернет.

Образцы итоговых заданий по оценке качества подготовки выпускников основной школы по информатике

В помощь учителю предлагаются 6 вариантов образцов итоговых заданий по оценке качества подготовки выпускников основной школы. Каждый вариант содержит по 24 вопроса, с помощью которых учитель может проверить качество усвоения пройденного материала. К каждому вопросу предлагаются 4 варианта ответа, из которых следует выбрать правильный. Вопросы подобраны таким образом, чтобы можно было определить качество подготовки учащихся по всем темам курса информатики, включенным в обязательный минимум содержания образования. Количество вопросов в каждом варианте может быть разным: и больше, и меньше. Здесь приводятся лишь образцы итоговых заданий. Задания могут меняться. Главное, чтобы каждый вариант итоговой проверки полностью охватывал весь теоретический и практический материал курса, включенный в обязательный минимум содержания.

Ниже приводится таблица соответствия номеров вопросов итоговых заданий пунктам требований к уровню подготовки выпускников по отдельным темам курса, а также 6 вариантов итоговых заданий. Вопросы каждого варианта могут использоваться учителем также и при осуществлении текущего контроля в качестве образцов заданий при изучении отдельных тем.

№ вопроса	Контролируемые темы
	1. Информация и информационные процессы
1	Представление о развитии ВТ и становлении информационного общества
	2. Представление информации
2	Знание единиц измерения количества информации
3	Умение определять количество информации
4	Умение записывать десятичные числа в двоичной системе счисления
	3. Компьютер
5	Знание основных устройств компьютера и их функций
6	Знание правил техники безопасности, технической эксплуатации, сохранности информации и защиты ее от вирусов при работе на компьютере
7	Знание, что такое файл и файловая система
	4. Моделирование и формализация
8	Представление о процессе моделирования и решение задач на компьютере
	5. Алгоритмы и исполнители
9	Представление об алгоритме, исполнителе, системе команд исполнителя
10	Знание основных алгоритмических конструкций
11	Умение формально выполнять алгоритмы
12	Присваивание и переменная в программировании
	6. Информационные технологии
	Технологии обработки текста и графики
13	Представление о возможностях текстовых редакторов
14	Знание о существовании различных форматов и кодировок текстовых документов
15	Представление о различных типах графических редакторов и их возможностях
16	Представление о существовании различных форматов графических файлов
17	Представление о мультимедиа-технологии
	Технология обработки числовой информации
18	Знание структуры и возможностей электронных таблиц
19	Умение решать задачи в электронных таблицах
	Технология хранения, поиска и сортировки информации
20	Знание структуры и возможностей баз данных
21	Умение решать задачи на поиск и сортировку записей
	Компьютерные коммуникации
22	Представление о технических характеристиках модемов и линий связи
23	Представление об информационных сервисах Интернета
24	Представление об основах технологии WWW

ВАРИАНТ 1

1. Массовое производство персональных компьютеров началось...

1) *в 40-е годы*
2) *в 50-е годы*
3) *в 80-е годы*
4) *в 90-е годы*

2. За единицу измерения количества информации принят...

1) *1 бод*
2) *1 бит*
3) *1 байт*
4) *1 Кбайт*

3. В детской игре "Угадай число" первый участник загадал целое число в промежутке от 1 до 8. Второй участник задает вопросы: "Загаданное число больше числа _?" Какое максимальное количество вопросов при правильной стратегии (интервал чисел в каждом вопросе делится пополам) должен задать второй участник, чтобы отгадать число?

1) *1*
2) *2*
3) *3*
4) *4*

4. Как записывается десятичное число "5" в двоичной системе счисления?

1) *101*
2) *110*
3) *111*
4) *100*

5. Производительность работы компьютера (быстрота выполнения операций) зависит от...

1) *размера экрана дисплея*
2) *частоты процессора*
3) *напряжения питания*
4) *быстроты нажатия на клавиши*

6. Какое устройство может оказывать вредное воздействие на здоровье человека?

1) *принтер*
2) *монитор*
3) *системный блок*
4) *модем*

7. Файл — это...

1) *единица измерения информации*
2) *программа в оперативной памяти*
3) *текст, распечатанный на принтере*
4) *программа или данные на диске*

8. Модель есть замещение изучаемого объекта другим объектом, который отражает...

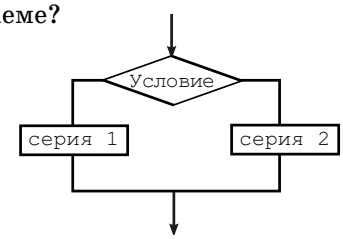
1) *все стороны данного объекта*
2) *некоторые стороны данного объекта*
3) *существенные стороны данного объекта*
4) *несущественные стороны данного объекта*

9. Алгоритмом является...

1) *последовательность команд, которую может выполнить исполнитель*
2) *система команд исполнителя*
3) *математическая модель*
4) *информационная модель*

10. Алгоритмическая структура какого типа изображена на блок-схеме?

1) *цикл*
2) *ветвление*
3) *подпрограмма*
4) *линейная*



11. Какая из последовательностей команд приведет исполнителя Черепашка в первоначальное место и положение?

1) *вперед (1 см), направо (90°), вперед (1 см), направо (90°), вперед (1 см), направо (90°), вперед (1 см), направо (90°)*
2) *вперед (1 см), направо (90°), вперед (1 см), направо (90°), вперед (1 см), направо (90°), вперед (1 см)*
3) *вперед (1 см), направо (90°), вперед (1 см), направо (90°), вперед (1 см), направо (90°)*
4) *вперед (1 см), направо (90°), вперед (1 см), направо (90°)*

12. Что изменяет операция присваивания?

1) *значение переменной*
2) *имя переменной*
3) *тип переменной*
4) *тип алгоритма*

13. Минимальным объектом, используемым в текстовом редакторе, является...

1) *слово*
2) *точка экрана (пиксель)*
3) *абзац*
4) *знакоместо (символ)*

14. Количество различных кодировок букв русского алфавита составляет...

1) *одну*
2) *две (MS-DOS, Windows)*
3) *три (MS-DOS, Windows, Macintosh)*
4) *пять (MS-DOS, Windows, Macintosh, КОИ-8, ISO)*

15. Инструментами в графическом редакторе являются...

1) *линия, круг, прямоугольник*
2) *выделение, копирование, вставка*
3) *карандаш, кисть, ластик*
4) *наборы цветов (палитры)*

16. Растровый графический файл содержит черно-белое изображение (без градаций серого) размером 100 × 100 точек. Каков информационный объем этого файла?

1) *10 000 бит*
2) *10 000 байт*
3) *10 Кбайт*
4) *1000 бит*

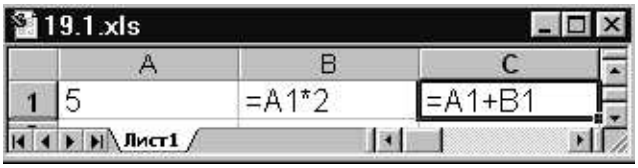
17. В состав мультимедиа-компьютера обязательно входит...

1) *проекционная панель*
2) *CD-ROM-дисковод и звуковая плата*
3) *модем*
4) *плоттер*

18. В электронных таблицах выделена группа ячеек A1:B3. Сколько ячеек входит в эту группу?

1) *6*
2) *5*
3) *4*
4) *3*

19. Результатом вычислений в ячейке C1 будет:



1) *5*
2) *10*
3) *15*
4) *20*

20. Основным элементом базы данных является...

1) *поле*
2) *форма*
3) *таблица*
4) *запись*

21. Какую строку будет занимать запись Pentium после проведения сортировки по возрастанию в поле **Опер. память?**

21.wdb

	Компьютер	Опер. память	Винчестер
1	Pentium	16	2Гб
2	386DX	4	300Мб
3	486DX	8	800Мб
4	Pentium II	32	4Гб

1) *1*
2) *2*
3) *3*
4) *4*

22. Модем, передающий информацию со скоростью 28 800 бит/с, может передать две страницы текста (3600 байт) в течение...

1) *1 секунды*
2) *1 минуты*
3) *1 часа*
4) *1 дня*

23. Какой из способов подключения к Интернету обеспечивает наибольшие возможности для доступа к информационным ресурсам...

1) *удаленный доступ по коммутируемому телефонному каналу*
2) *постоянное соединение по оптоволоконному каналу*
3) *постоянное соединение по выделенному телефонному каналу*
4) *терминальное соединение по коммутируемому телефонному каналу*

24. Гипертекст — это...

1) *очень большой текст*
2) *структурированный текст, в котором могут осуществляться переходы по выделенным меткам*
3) *текст, набранный на компьютере*
4) *текст, в котором используется шрифт большого размера*

Таблица номеров правильных ответов на вопросы теста для 1-го варианта

№ вопроса	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
№ правильного ответа	3	2	3	1	2	2	4	3	1	2	1	1	4	4	3	1	2	1	3	4	3	1	2	2

ВАРИАНТ 2

1. Общим свойством машины Бэббиджа, современного компьютера и человеческого мозга является способность обрабатывать…

1) *числовую информацию*
2) *текстовую информацию*
3) *звуковую информацию*
4) *графическую информацию*

2. Чему равен 1 байт?

1) *10 бит* 3) *8 бит*
2) *10 Кбайт* 4) *1 бод*

3. В детской игре “Угадай число” первый участник загадал целое число в промежутке от 1 до 16. Второй участник задает вопросы: “Загаданное число больше числа _?” Какое максимальное количество вопросов при правильной стратегии (интервал чисел в каждом вопросе делится пополам) должен задать второй участник, чтобы отгадать число?

1) *2* 3) *4*
2) *3* 4) *5*

4. Как записывается десятичное число 6 в двоичной системе счисления?

1) *101* 3) *111*
2) *110* 4) *100*

5. При выключении компьютера вся информация стирается…

1) *на гибком диске*
2) *на CD-ROM-диске*
3) *на жестком диске*
4) *в оперативной памяти*

6. В каком направлении от монитора вредные излучения максимальны?

1) *от экрана вперед*
2) *от экрана назад*
3) *от экрана вниз*
4) *от экрана вверх*

7. Файловую систему обычно изображают в виде дерева, где “ветки” — это каталоги (папки), а “листья” — это файлы (документы). Что может располагаться непосредственно в корневом каталоге, т.е. на “стволе” дерева?

1) *каталоги и файлы*
2) *только каталоги*
3) *только файлы*
4) *ничего*

8. Модель содержит информации…

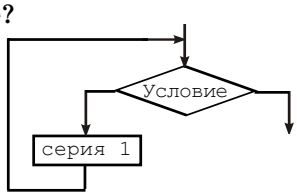
1) *столько же, сколько и моделируемый объект*
2) *меньше, чем моделируемый объект*
3) *больше, чем моделируемый объект*
4) *не содержит информации*

9. Какой из документов является алгоритмом?

1) *правила техники безопасности*
2) *инструкция по получению денег в банкомате*
3) *расписание уроков*
4) *список класса*

10. Алгоритмическая структура какого типа изображена на блок-схеме?

1) *цикл*
2) *ветвление*
3) *подпрограмма*
4) *линейная*



11. Какой путь пройдет исполнитель Черепашка после выполнения последовательности команд: вперед (1 см), направо (90°), вперед (1 см), направо (90°), вперед (1 см), направо (90°), вперед (1 см), направо (90°)?

1) *0 см* 3) *3 см*
2) *2 см* 4) *4 см*

12. Переменная в программировании считается полностью заданной, если известны ее…

1) *тип, имя* 3) *тип, значение*
2) *имя, значение* 4) *тип, имя, значение*

13. В процессе редактирования текста изменяется...

1) *размер шрифта*
2) *параметры абзаца*
3) *последовательность символов, слов, абзацев*
4) *параметры страницы*

14. Количество стандартных кодировок букв латинского алфавита составляет...

1) *одну*
2) *две (MS-DOS, Windows)*
3) *три (MS-DOS, Windows, Macintosh)*
4) *пять (MS-DOS, Windows, Macintosh, КОИ-8, ISO)*

15. Палитрами в графическом редакторе являются...

1) *линия, круг, прямоугольник*
2) *выделение, копирование, вставка*
3) *карандаш, кисть, ластик*
4) *наборы цветов*

16. Растровый графический файл содержит черно-белое изображение с 16 градациями серого цвета размером 10 × 10 точек. Каков информационный объем этого файла?

1) *100 бит*
2) *400 байт*
3) *400 бит*
4) *100 байт*

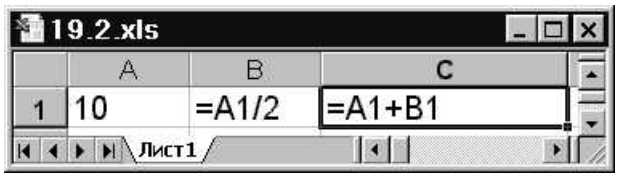
17. Звуковая плата с возможностью 16-битного двоичного кодирования позволяет воспроизводить звук с...

1) *8 уровнями интенсивности*
2) *16 уровнями интенсивности*
3) *256 уровнями интенсивности*
4) *65 536 уровнями интенсивности*

18. В электронных таблицах выделена группа ячеек A1:C2. Сколько ячеек входит в эту группу?

1) *6*
2) *5*
3) *4*
4) *3*

19. Результатом вычислений в ячейке C1 будет:



1) *20*
2) *15*
3) *10*
4) *5*

20. База данных представлена в табличной форме. Запись образует...

1) *поле в таблице*
2) *имя поля*
3) *строку в таблице*
4) *ячейку*

21. Какие записи будут найдены после проведения поиска в поле **Опер. память** с условием **> 8**?

21.wdb

	Компьютер	Опер. память	Винчестер
1	Pentium	16	2Гб
2	386DX	4	300Мб
3	486DX	8	800Мб
4	Pentium II	32	4Гб

1) *1, 2*
2) *2, 3*
3) *3, 4*
4) *1, 4*

22. Модем, передающий информацию со скоростью 28 800 бит/с, за 1 с может передать...

1) *две страницы текста (3600 байт)*
2) *рисунок (36 Кбайт)*
3) *аудиофайл (360 Кбайт)*
4) *видеофайл (3,6 Мбайта)*

23. Электронная почта (e-mail) позволяет передавать...

1) *только сообщения*
2) *только файлы*
3) *сообщения и приложенные файлы*
4) *видеоизображение*

24. HTML (*Hyper Text Markup Language*) является...

1) *сервером Интернета*
2) *средством создания web-страниц*
3) *транслятором языка программирования*
4) *средством просмотра web-страниц*

Таблица номеров правильных ответов на вопросы теста для 2-го варианта

№ вопроса	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
№ правильного ответа	1	3	3	2	4	2	1	2	2	1	4	4	3	1	4	3	4	1	2	3	4	1	3	2

ИНФОРМАТИКА 7—9

А.Г. Кушниренко,
Г.В. Лебедев,
Я.Н. Зайдельман

Избранные главы нового учебника

Выпуск 2

Выпуск 1 опубликован в № 39/99

§ 8. Вспомогательные алгоритмы

8.1. Понятие о вспомогательном алгоритме

Вернемся к примеру, разобранному в конце предыдущего параграфа (*рис.* 10*). Составляя алгоритм решения задачи, мы выделили повторяющуюся часть пути Робота, записали эту часть в одну строку алгоритма и повторили эту строку необходимое число раз.

Если бы мы ошиблись при записи повторяющейся части, эту ошибку пришлось бы исправлять в каждой строке. Если мы захотим изменить алгоритм, чтобы приспособить его для решения другой задачи (например, пройти лабиринт на *рис.* 11), правку опять придется вносить в каждую строку.

Избежать этого можно, если вместо повторения строки использовать ***вызов вспомогательного алгоритма***.

Оформим прохождение части лабиринта, показанной на *рис.* 10*б*, в виде отдельного алгоритма.

```
алг участок                                    (А8)
   дано  | Робот в начале участка (рис.10б)
   надо  | Робот в конце участка  (рис.10б)
нач
 | |вверх; вверх; вправо; вниз; вниз; вправо
кон
```

Теперь, составляя алгоритм прохождения лабиринта, можно не переписывать команды прохождения участка, а вызвать уже написанный алгоритм участок.

```
алг из А в Б                                   (А9)
  дано  | Робот в клетке А (рис.10а)
  надо  | Робот в клетке Б (рис.10а)
нач
 | участок; участок; участок; участок; участок
кон
```

Запись участок в алгоритме из А в Б — это ***команда вызова вспомогательного алгоритма*** с именем участок. Выполняя эту команду, компьютер приостанавливает выполнение алгоритма из А в Б, выполняет алгоритм участок, а затем продолжает выполнение алгоритма из А в Б.

8.2. Основные и вспомогательные алгоритмы

В рассмотренном примере алгоритм участок называется ***вспомогательным*** для алгоритма из А в Б, а алгоритм из А в Б — ***основным*** для алгоритма участок.

Приказ на выполнение вспомогательного алгоритма называется ***вызовом*** этого вспомогательного алгоритма.

В общем случае если в записи алгоритма *X* встречается вызов алгоритма *Y*, то алгоритм *Y* называется ***вспомогательным*** для *X*, а алгоритм *X* называется ***основным*** для *Y*.

Алгоритмы можно рассматривать как основные и вспомогательные только по отношению друг к другу, сами по себе они ни основными, ни вспомогательными не являются.

Любой алгоритм можно использовать как вспомогательный, для этого достаточно указать его имя в команде вызова в каком-то другом алгоритме. Здесь можно провести аналогию с математикой: при доказательстве теорем мы часто используем уже известные теоремы, не доказывая их заново. Точно так же при составлении алгоритмов можно вызвать уже написанный алгоритм, не переписывая его.

Использование уже написанных алгоритмов как вспомогательных позволяет свести новую задачу к уже решенным. Такой метод построения алгоритмов иногда называют ***программированием снизу вверх***.

8.3. Пример использования вспомогательных алгоритмов

Пусть требуется написать алгоритм, который проводит Робота из клетки А в клетку Б и закрашивает клетки, отмеченные точками (*рис.* 13). На рисунке показан один из возможных путей Робота при решении этой задачи. Видно, что Робот должен закрасить три прямоугольных "блока" размером 3 × 5 клеток, а между закрашиваниями блоков два раза обойти стену. Запишем эту мысль в виде алгоритма:

```
алг из А в Б с закрашиванием                  (А10)
  дано  | Робот в точке А (рис.13)
  надо  | Робот в точке Б (рис.13),
        |нужные клетки закрашены
```

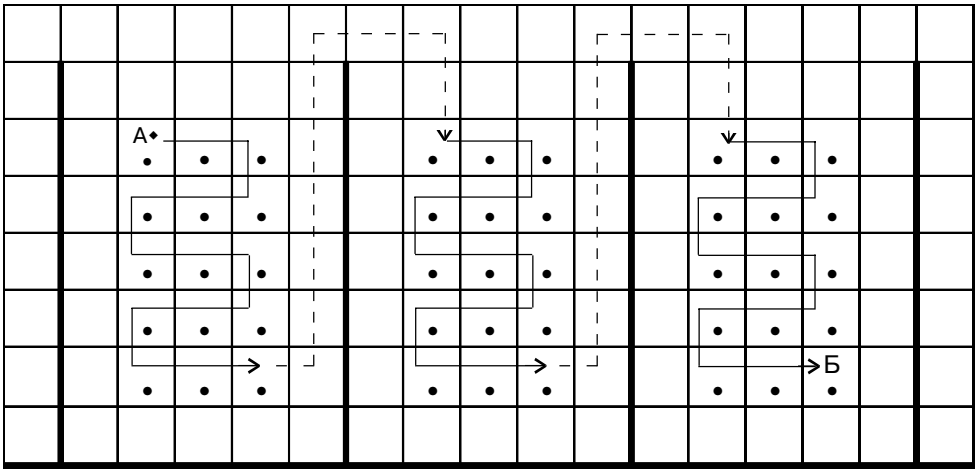


Рис. 13

* Рисунки 10, 11, 12 см. в первом выпуске (№ 39/99).

```
нач
| закрашивание блока
| обход стены
| закрашивание блока
| обход стены
| закрашивание блока
кон
```

Теперь составим вспомогательные алгоритмы, используемые в основном алгоритме из А в Б с закрашиванием (А10):

```
алг закрашивание блока                          (А11)
  дано  | Робот в левом верхнем углу блока
        | 3х5 клеток
  надо  | блок закрашен, Робот в его правом
        | нижнем углу
нач
| закрасить; вправо; закрасить; вправо;
|   закрасить; вниз
| закрасить; влево;  закрасить; влево;
|   закрасить; вниз
| закрасить; вправо; закрасить; вправо;
|   закрасить; вниз
| закрасить; влево;  закрасить; влево;
|   закрасить; вниз
| закрасить; вправо; закрасить; вправо;
|   закрасить
кон
```

```
алг обход стены                                 (А12)
  дано  | Робот в правом нижнем углу блока
        |(рис.13)
  надо  | Робот в левом верхнем углу
   | следующего блока (рис.13)
нач
| вправо; вверх; вверх; вверх; вверх;
| вверх; вверх
| вправо; вправо; вниз; вниз
кон
```

8.4. Метод последовательного уточнения

В разобранном примере мы сначала написали основной алгоритм из А в Б с закрашиванием (А10), используя еще не написанные вспомогательные алгоритмы для действий закрашивание блока и обход стены, и лишь потом составили эти вспомогательные алгоритмы (А11) и (А12). Такой метод составления алгоритмов называется ***методом последовательного уточнения***.

В общем случае метод последовательного уточнения состоит в том, что исходная задача разбивается на ряд крупных частей (подзадач) и составляется основной алгоритм, в котором для решения подзадач используются вызовы еще не написанных вспомогательных алгоритмов. Таким образом, сначала полностью записывается основной алгоритм и только потом начинается составление вспомогательных. При составлении вспомогательных алгоритмов, в свою очередь, могут использоваться вспомогательные алгоритмы для выполнения более мелких действий и т.д.

Метод последовательного уточнения облегчает составление алгоритмов, так как позволяет решать задачу по частям и использовать в качестве вспомогательных алгоритмы еще не решенных задач.

Построение алгоритмов методом последовательного уточнения иногда называют ***программированием сверху вниз***.

8.5. Заголовки вспомогательных алгоритмов

Что нужно знать об алгоритме, чтобы его можно было использовать как вспомогательный?

Во-первых, необходимо знать имя этого алгоритма, чтобы можно было записать команду вызова.

Во-вторых, необходимо знать, в какой обстановке алгоритм может быть исполнен.

В-третьих, что получается в результате выполнения алгоритма.

Вся эта информация содержится в заголовке алгоритма — в строках **алг**, **дано**, **надо**.

В то же время для использования алгоритма как вспомогательного необязательно знать, как именно он работает, то есть необязательно знать тело алгоритма. Здесь опять видна аналогия с математикой: чтобы пользоваться теоремой, достаточно знать ее формулировку, а в доказательстве нет необходимости.

Таким образом, заголовок любого алгоритма содержит *всю* информацию, необходимую для использования этого алгоритма как вспомогательного. При программировании снизу вверх для написания очередного алгоритма нужно знать заголовки (и только заголовки!) ранее написанных алгоритмов.

При программировании сверху вниз роль заголовка меняется. В основном алгоритме записывается вызов еще не написанного вспомогательного алгоритма. Этот вызов фактически задает заголовок будущего вспомогательного алгоритма. В самом деле, в вызове указано имя алгоритма (строка **алг**); обстановка, сложившаяся при выполнении основного алгоритма к моменту вызова, задает начальные условия вспомогательного алгоритма (строка **дано**), необходимый результат описывает итог работы (строка **надо**).

Например, написав алгоритм из А в Б с закрашиванием, мы фактически задали заголовки алгоритмов закрашивание блока и обход стены.

8.6. Разделение труда между компьютером и исполнителями

Алгоритм — это описание последовательности действий компьютера. Исполнители ничего ни про какие алгоритмы не знают.

Например, при выполнении алгоритма из А в Б с закрашиванием (А10) именно компьютер “разбирается” в записи алгоритма на алгоритмическом языке, “понимает”, что сначала надо выполнить алгоритм закрашивание блока (А11), выполняет его и т.д. Робот же лишь выполняет последовательно поступающие от компьютера команды.

Схематически работа компьютера и Робота при выполнении этого алгоритма показана в табл. 8.

Таблица 8

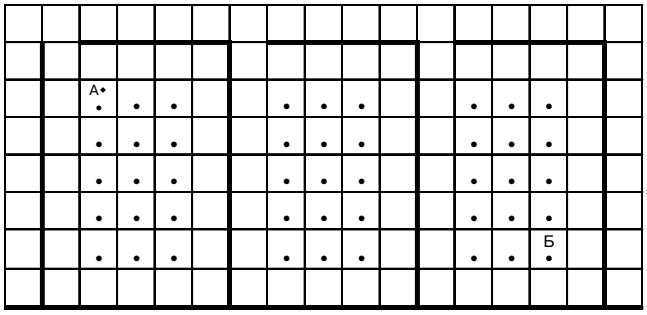
Диалог компьютера и Робота

Компьютер	Робот
начинает выполнять алгоритм из А в Б с закрашиванием	
вызывает алгоритм закрашивание блока	
командует Роботу закрасить	закрашивает клетку
командует Роботу вправо	делает шаг вправо
командует Роботу закрасить	закрашивает клетку
командует Роботу вправо	делает шаг вправо
командует Роботу закрасить	закрашивает клетку
командует Роботу вниз	делает шаг вниз

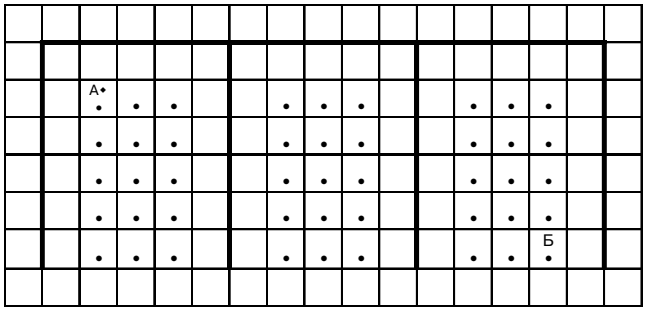
Задачи и упражнения

1. Измените алгоритм закрашивание блока так, чтобы Робот шел другим путем. Изменится ли при этом заголовок алгоритма? Почему?

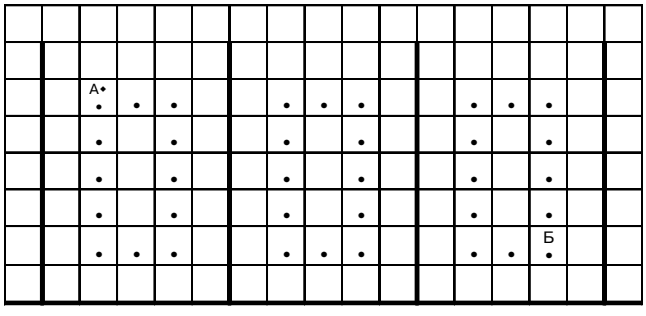
2. Составьте алгоритм, который переводит Робота из А в Б и закрашивает клетки, отмеченные точками (*рис.* 14).



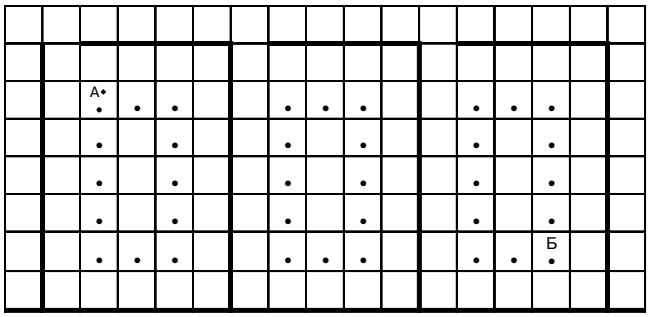
а



б



в



г

Рис. 14

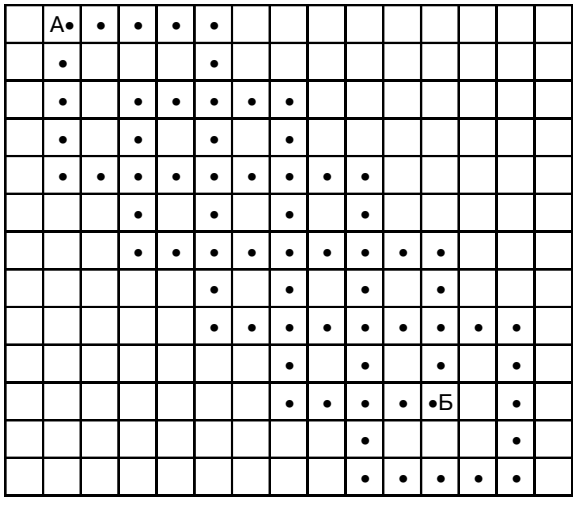
3. Составьте алгоритм, при выполнении которого в стек Калькулятора добавляется число "55". Используя его как вспомогательный, получите на экране число "3026".

4. Даны алгоритмы:

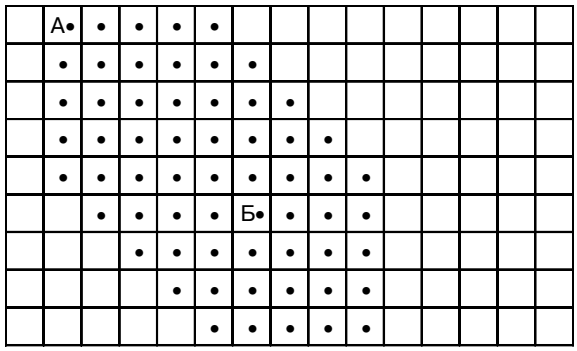
```
алг фигура                                          (A13)
нач
  | фрагмент; фрагмент; фрагмент;
      фрагмент; фрагмент
кон

алг фрагмент                                        (A14)
нач
  | закрасить; вправо; закрасить; вправо;
      закрасить; вправо
  | закрасить; вниз;   закрасить; вниз;
      закрасить; вниз
  | закрасить; влево;  закрасить; влево;
      закрасить; влево
  | закрасить; вверх;  закрасить; вверх;
      закрасить
  | вправо; вправо; вправо; вниз; вниз
кон
```

Нарисуйте результат выполнения алгоритма фигура (закрашенные клетки, начальное и конечное положения Робота).

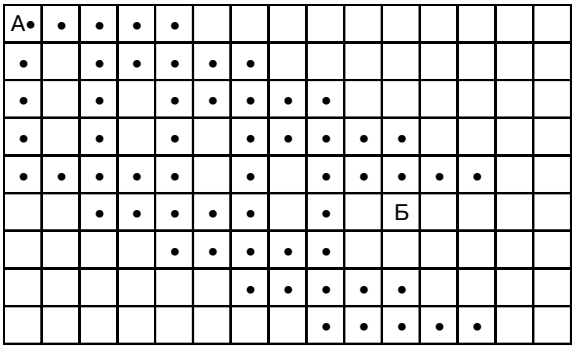
5. Измените вспомогательный алгоритм фрагмент (A14) так, чтобы при выполнении алгоритма (A13) Робот переместился из А в Б и закрасил клетки, отмеченные точками (*рис.* 15).



а



б



в

Рис. 15

6. Составьте алгоритмы `вверх с подсчетом`, `вниз с подсчетом`, `влево с подсчетом`, `вправо с подсчетом`, при выполнении которых Робот делает шаг в указанном направлении, а Счетчик увеличивает число на единицу. Как можно, используя эти алгоритмы, переписать алгоритмы управления Роботом, чтобы узнать общее количество шагов, сделанных Роботом?
7. Робот находится в левом нижнем углу квадрата 9×9 клеток. Горизонтальные и вертикальные ряды пронумерованы от 1 до 9 (*рис.* 16). В начало некоторого алгоритма управления Роботом вставили команду `начать работу`, все команды движения заменили на движение с подсчетом (вместо `вправо` — `вправо с подсчетом` и т.д.). В результате при движении Робота по полю на экране Счетчика все время видны текущие координаты Робота. Например, когда Робот находится в клетке А, Счетчик показывает 26, в клетке Б — 83. Составьте алгоритмы начала работы и движения с подсчетом.

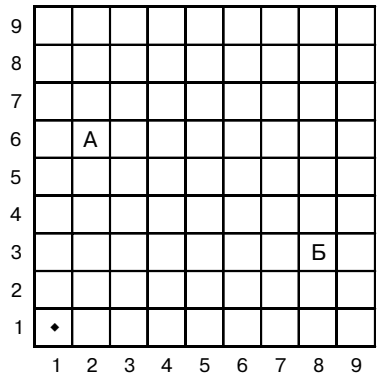
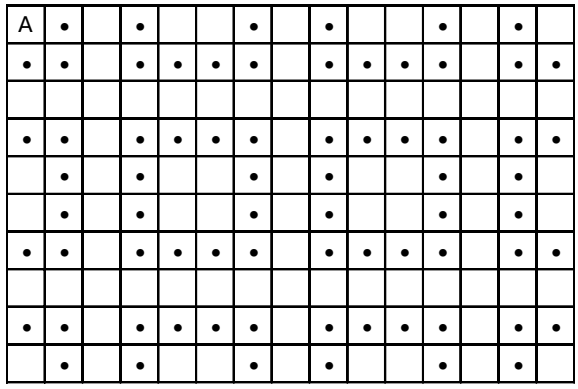
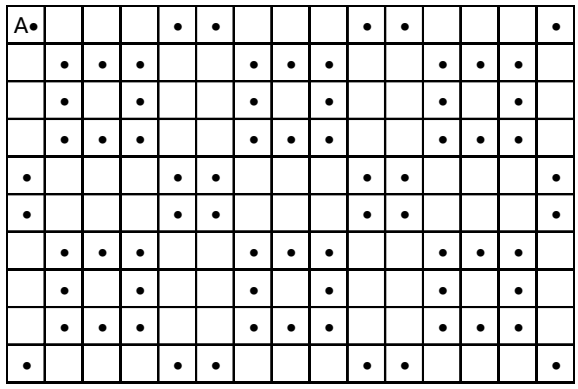


Рис. 16

8. Известно, что вспомогательный алгоритм `фрагмент` закрашивает некоторые клетки в квадрате 4×4, причем путь Робота при выполнении этого алгоритма начинается и заканчивается в левом верхнем углу квадрата. Составьте алгоритм, который закрашивает 25 одинаковых фрагментов в квадрате 20×20 клеток.
9. Составьте алгоритм, который закрашивает в шахматном порядке квадрат 10×10, начиная с левого верхнего угла.
10. Составьте алгоритм, при выполнении которого Робот перемещается из клетки А в клетку Б и закрашивает клетки, отмеченные точками (*рис.* 17).



а



б

Рис. 17

11. Сколько команд `закрасить` ЭВМ даст Роботу при выполнении алгоритма упражнения 4? Сколько клеток при этом будет закрашено:

а) дважды;

б) трижды?

§ 9. Цикл `n раз`

9.1. Пример алгоритма с циклом `n раз`

При составлении алгоритмов довольно часто встречаются случаи, когда некоторую последовательность команд нужно выполнить несколько раз подряд (см., например, алгоритм `А3`). Для упрощения записи алгоритма в таких случаях можно использовать специальную ***составную команду алгоритмического языка*** — цикл **`n раз`**:

```
алг из А в Б                                    (А15)
 | дано | Робот в клетке А (рис.10а)
 | надо | Робот в клетке Б (рис.10а)
нач
   нц 5 раз
    | вверх; вверх; вправо; вниз; вниз; вправо
   кц
кон
```

При выполнении этого алгоритма компьютер 5 раз повторит последовательность вызовов `вверх; вверх; вправо; вниз; вниз; вправо`, и Робот окажется в клетке Б (*рис.* 10*а*).

9.2. Общий вид цикла n раз

В общем виде цикл **n раз** записывается так:

```
нц число повторений  раз
|  тело цикла (последовательность команд)
кц
```

Служебные слова **нц** (начало цикла) и **кц** (конец цикла) пишутся строго одно под другим и соединяются вертикальной чертой. Правее этой черты записывается повторяемая последовательность команд (***тело цикла***).

Число повторений — произвольное целое число.

При выполнении алгоритма последовательность команд в теле цикла повторяется указанное число раз. Правила алгоритмического языка допускают задание любого целого числа повторений. Оно может быть нулевым и даже отрицательным. Эти случаи не считаются ошибочными, просто тело цикла не будет выполнено ни разу, а компьютер сразу перейдет к выполнению команд, записанных после **кц**.

9.3. Короткие алгоритмы могут описывать длинные последовательности действий

Использование цикла позволяет очень коротко записать довольно длинные последовательности действий. Рассмотрим пример.

```
алг закрасить ряд из 10 клеток                  (А16)
  дано  | на поле Робота стен нет
  надо  | Робот закрасил 10 клеток вправо
        | и вернулся в исходное положение
нач
 | нц 10 раз
 |    закрасить; вправо
 | кц
 | нц 10 раз
 |    влево
 | кц
кон
```

При выполнении этого короткого алгоритма компьютер даст Роботу 30 команд: сначала 10 раз будет выполнена пара команд `закрасить; вправо`, а затем 10 команд `влево`.

Если вместо числа 10 поставить 100, Робот выполнит уже 300 команд, а ведь длина алгоритма при этом не увеличится!

Таким образом, короткие алгоритмы могут описывать очень длинные последовательности действий.

9.4. Внутри цикла можно вызывать вспомогательные алгоритмы

```
алг закрасить прямоугольник                     (А17)
 дано | на поле Робота стен нет
 надо | закрашен прямоугольник размером 10х6
      | Робот в исходном положении
нач
 | нц 6 раз
 |  | закрасить ряд из 10 клеток
 |  | вниз
 | кц
 | нц 6 раз
 |  | вверх
 | кц
кон
```

9.5. Внутри цикла могут быть другие циклы

Задачу из алгоритма (А17) можно решить и без использования вспомогательного алгоритма, вставив текст алгоритма `закрасить ряд из 10 клеток` непосредственно в текст основного алгоритма.

```
алг закрасить прямоугольник                     (А18)
 дано  | на поле Робота стен нет
 надо  | закрашен прямоугольник размером 10х6
       | Робот в исходном положении
нач
 | нц 6 раз
 |  | нц 10 раз
 |  |  | закрасить; вправо
 |  | кц
 |  | нц 10 раз
 |  |  | влево
 |  | кц
 |  | вниз
 | кц
 | нц 6 раз
 |  |  вверх
 | кц
кон
```

Цикл, который находится внутри другого, называется ***вложенным циклом***. В данной задаче использование вложенных циклов не очень удобно: алгоритм (А18) выглядит более громоздким и менее понятным, чем (А17).

9.6. Краткость и скорость не всегда совпадают

Используя цикл **n раз**, можно очень коротко записать решение некоторых задач. Вот как, например, можно получить на экране Калькулятора число "1998":

```
алг число 1998                                  (А19)
  дано | стек Калькулятора пуст
  надо | на экране число 1998
нач
 | запомнить 1
 | нц 1997 раз
 |  | запомнить 1
 |  | сложить
 | кц
 | показать
кон
```

Этот алгоритм намного короче решающего ту же задачу алгоритма (А2), и его намного проще придумать. Но означает ли это, что алгоритм (А19) лучше?

При выполнении алгоритма (А19) компьютер передаст Калькулятору 3996 команд, а в алгоритме (А2) их будет всего 12. Таким образом, короткий алгоритм в данном случае работает в 333 раза медленнее!

Такая ситуация встречается в информатике довольно часто. Для многих задач можно придумать простой и короткий алгоритм, на выполнение которого потребуется много времени, но для той же самой задачи может существовать другой, более длинный и сложный алгоритм, который работает намного быстрее. Создание таких быстрых алгоритмов — одна из важных задач информатики.

Задачи и упражнения

1. Решите задачи 6 и 9 из § 7 с использованием цикла **n раз**.
2. Решите задачу 8 из § 8 с использованием цикла **n раз**.
3. Решение каких еще задач из § 8 можно упростить, используя цикл **n раз**? Составьте соответствующие алгоритмы.
4. Составьте алгоритм получения на экране Калькулятора числа 13^6.

§ 10. Цикл пока

10.1. Команды-вопросы Робота

До сих пор все составленные нами алгоритмы управления Роботом приводили к выполнению определенной, заранее известной последовательности действий. Компьютер лишь приказывал Роботу выполнить какие-то действия, но никак не анализировал их результаты и обстановку на поле Робота.

Представьте себе начальника, который отдает приказы, но не получает никаких донесений о результатах их выполнения; повара, который не может попробовать блюдо; шофера, ведущего автомобиль с закрытыми глазами. Понятно, что так далеко не уедешь; если мы хотим составлять алгоритмы для решения сложных задач, то надо уметь не только командовать исполнителем, но и анализировать обстановку, в которой он оказался.

У Робота для этого есть 12 команд-вопросов, отвечая на которые, он сообщает информацию об обстановке вокруг себя. В этом параграфе мы рассмотрим 10 из них. Вот они:

```
сверху стена          сверху свободно
снизу  стена          снизу  свободно
справа стена          справа свободно
слева  стена          слева  свободно
клетка закрашена      клетка чистая
```

Смысл этих вопросов ясен из их названий. Получив команду-вопрос, Робот проверяет соответствующее условие и отвечает **да** или **нет**.

Теперь мы можем рассмотреть вторую секцию пульта дистанционного управления Роботом (*рис.* 8). Каждому из 10 вопросов соответствует кнопка в этой части пульта. Для получения ответов Робота на пульте есть две лампочки, обозначенные **да** и **нет**. После нажатия кнопки с командой-вопросом загорается лампочка, соответствующая ответу Робота.

10.2. Использование команд-вопросов при ручном управлении Роботом

Задача. Где-то ниже Робота на неизвестном расстоянии есть стена. Нужно подвести Робота вплотную к этой стене.

Если мы командуем Роботом вручную, без компьютера, то задачу можно решить так: спросим у Робота — снизу свободно? Если Робот ответит **нет**, значит, он уже у стены и задача решена. Если же Робот ответит **да**, то можно скомандовать вниз и опять спросить: снизу свободно? Если Робот ответит **да**, опять скомандовать вниз, спросить: снизу свободно? — и т.д., пока Робот не ответит **нет**. Скомандуем ли мы при этом вниз 0, 3, 8 или 1998 раз, заранее неизвестно — это зависит от начального расположения Робота относительно стены.

Наши действия при решении этой задачи коротко можно описать так: пока на вопрос снизу свободно? Робот отвечает **да**, надо командовать вниз и повторять вопрос.

10.3. Цикл пока

Наша цель, однако, не ручное управление Роботом, а составление алгоритма для компьютера. Поэтому приведенную выше последовательность действий надо описать на алгоритмическом языке. Алгоритм должен быть универсальным, т.е. не должен зависеть от расстояния между Роботом и стеной. Для этого в алгоритмическом языке есть специальная составная команда — цикл **пока**:

```
алг вниз до стены                          (А20)
нач
  нц пока снизу свободно
    вниз
  кц
кон
```

Аналогично можно составить алгоритмы влево до стены, вправо до стены, вверх до стены, которыми мы далее часто будем пользоваться как вспомогательными.

10.4. Диалог "компьютер — Робот" при выполнении цикла пока

Пусть в начальный момент Робот находится в клетке А (*рис.* 18). Тогда при выполнении алгоритма вниз до стены (А20) диалог “компьютер — Робот” будет таким:

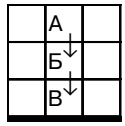


Рис. 18

компьютер : снизу свободно?	**Робот** : **да**
компьютер : вниз	**Робот** : смещается вниз (в клетку Б)
компьютер : снизу свободно?	**Робот** : **да**
компьютер : вниз	**Робот** : смещается вниз (в клетку В)
компьютер : снизу свободно?	**Робот** : **нет**

Поскольку Робот ответил **нет**, т.е. записанное после **пока** условие не соблюдается, то на этом выполнение цикла **пока** и алгоритма вниз до стены заканчивается.

10.5. Общий вид цикла пока

В общем виде цикл **пока** записывается так:

```
нц пока условие
|     тело цикла (последовательность команд)
кц
```

Слова **нц** и **кц** имеют тот же смысл, что и в цикле **n раз**, — они отмечают начало и конец цикла.

При выполнении цикла компьютер повторяет следующие действия:

а) проверяет записанное после служебного слова **пока** условие;

б) если условие не соблюдается (Робот ответил **нет**), то выполнение цикла завершается и компьютер начинает выполнять команды, записанные после **кц**. Если же условие соблюдается (Робот ответил **да**), то компьютер выполняет тело цикла, снова проверяет условие и т.д.

Графическая схема выполнения цикла **пока** изображена на *рис.* 19.

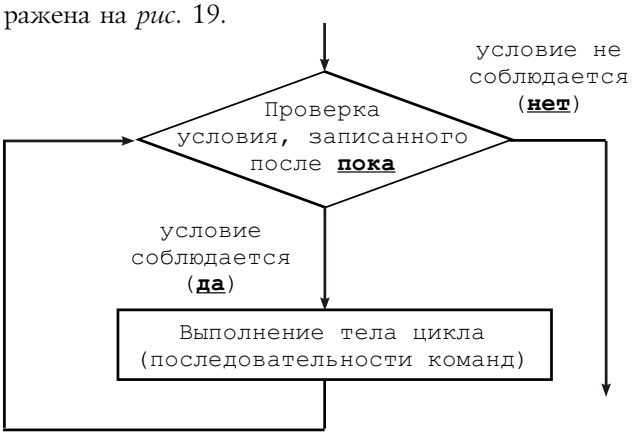


Рис. 19

10.6. Цикл n раз и цикл пока

Циклы **n раз** и **пока** оформляются в алгоритмическом языке почти одинаково. Это неудивительно: ведь обе эти команды задают ***цикл*** — повторяющуюся последовательность команд. Служебные слова **нц** и **кц** указывают, что исполняется цикл, а заголовок цикла указывает конкретный механизм его выполнения.

Однако у этих двух циклов есть одно существенное отличие. Начиная выполнять цикл **n раз**, компьютер знает, сколько раз придется повторить тело цикла. При исполнении цикла **пока** это не так — компьютер каждый раз проверяет условие цикла и не может заранее определить, когда выполнение закончится. Определить количество повторений цикла **пока** можно только после того, как цикл завершен.

Отсюда ясно, в каких случаях какой цикл следует использовать. Если к моменту начала цикла количество повторений известно, можно воспользоваться циклом **n раз**. Если же количество повторений заранее определить нельзя, необходим цикл **пока**.

10.7. Условия в цикле пока

В цикле **пока** могут использоваться ***простые*** и ***составные*** условия.

Простое условие — это обычно какая-то проверка. Примером простого условия может служить любая команда-вопрос Робота. Позже мы познакомимся с другими видами простых условий.

Составное условие составляется из одного или нескольких простых с использованием служебных слов **и**, **или**, **не**. Смысл этих слов и правила построения с их помощью составных условий те же, что и при построении запросов к базе данных.

Если между простыми условиями стоит **или**, составное условие считается выполненным в том случае, когда выполнено хотя бы одно из простых условий.

Рассмотрим подробнее проверку составного условия с **или**. Сначала проверяется первое простое условие. Если оно выполнено, то составное условие заведомо выполнено и второе простое условие даже не проверяется. Если же первое условие не выполнено, проверяется второе условие и результат этой проверки становится результатом всего составного условия.

Например, если Робот стоит в углу (*рис.* 20), то при проверке условия сверху стена **или** справа стена окончательный ответ ясен уже после первой проверки (сверху стена), поэтому второе условие не проверяется, компьютер даже не будет задавать Роботу вопрос: справа стена? Если записать то же самое составное условие как справа стена **или** сверху стена?, то после первой проверки окончательный результат неясен, поэтому будет проверено и второе условие.

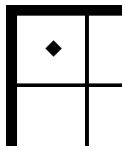


Рис. 20

Если между простыми условиями стоит **и**, то для выполнения составного условия необходимо, чтобы верными оказались оба простых.

Если в составном условии с **и** первое условие оказалось неверным, второе уже не проверяется, так как результатом все равно будет **нет**.

10.8. Свойства цикла пока

1. Тело цикла может не выполниться ни разу.

Если условие в цикле **пока** не соблюдается с самого начала, то тело цикла не выполняется ни разу. Например, если в алгоритме вниз до стены (А20) Робот на первый же вопрос снизу свободно? ответит **нет** (т.е. если снизу от него с самого начала будет стена), то компьютер не вызовет команду вниз ни разу.

Важно понимать, что ситуация, когда цикл ни разу не выполняется, не является отказом. Это нормальный вариант выполнения алгоритма, который приводит к правильным результатам.

2. Выполнение цикла может не завершиться.

Выполнение цикла **пока** может не завершиться, если условие все время будет соблюдаться. Такая ситуация обычно возникает из-за ошибок в составлении алгоритма. Она называется ***зацикливанием***.

Рассмотрим, например, такой фрагмент алгоритма:

```
нц пока справа свободно
|   вправо; влево
кц
```

Если справа от Робота нет стены, он будет бесконечно топтаться на месте, совершая шаги вправо и влево. Исполнение алгоритма никогда не закончится.

3. Условие цикла не проверяется в процессе выполнения тела цикла.

Условие в цикле **пока** проверяется только *перед* выполнением тела цикла, но не проверяется в процессе выполнения.

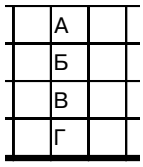


Рис. 21

Пример 1. Пусть Робот находится в клетке А (*рис.* 21) и компьютер выполняет цикл

```
нц пока снизу свободно
|  вниз
|  вниз
кц
```

Тогда диалог “компьютер — Робот” будет таким:

компьютер : снизу свободно?	**Робот** : **да**
компьютер : вниз	**Робот** : смещается вниз (в клетку Б)
компьютер : вниз	**Робот** : смещается вниз (в клетку В)
компьютер : снизу свободно?	**Робот** : **да**
компьютер : вниз	**Робот** : смещается вниз (в клетку Г)
компьютер : вниз	**Робот** : отказ

Таким образом, в процессе выполнения тела цикла компьютер условие *не проверяет* и вопросов Роботу не задает. Если в какой-то момент условие перестанет соблюдаться, то компьютер все равно будет выполнять тело цикла до конца.

Пример 2. Пусть Робот находится в клетке А (*рис.* 22) и компьютер выполняет цикл

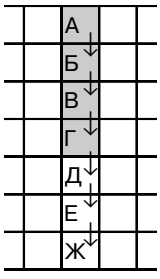


Рис. 22

```
нц пока клетка закрашена
|  вниз
|  вниз
|  вниз
кц
```

Тогда диалог “компьютер — Робот” будет таким:

компьютер : клетка закрашена?	**Робот** : **да**
компьютер : вниз	**Робот** : смещается вниз (в клетку Б)
компьютер : вниз	**Робот** : смещается вниз (в клетку В)
компьютер : вниз	**Робот** : смещается вниз (в клетку Г)
компьютер : клетка закрашена?	**Робот** : **да**
компьютер : вниз	**Робот** : смещается вниз (в клетку Д)
компьютер : вниз	**Робот** : смещается вниз (в клетку Е)
компьютер : вниз	**Робот** : смещается вниз (в клетку Ж)
компьютер : клетка закрашена?	**Робот** : **нет**

В процессе второго выполнения тела цикла условие клетка закрашена перестанет соблюдаться (клетки Д и Е не закрашены). Однако компьютер доведет выполнение тела цикла до конца и Робот окажется на две клетки ниже клетки Д.

4. Перед каждым выполнением тела цикла условие обязательно выполняется.

Это очевидное свойство: если компьютер приступает к выполнению тела цикла, значит, условие выполнено.

5. Сразу после окончания цикла условие не выполняется.

Это свойство тоже очевидно: цикл заканчивается в тот момент, когда при очередной проверке условие оказалось не выполненным.

10.9. Составление алгоритмов с циклом пока

Всякий раз, когда число повторений каких-то действий заранее неизвестно, используется цикл **пока**. Составление таких циклов — трудная задача. Короткий цикл **пока** может описывать длинную последовательность действий. Чтобы не запутаться и что-нибудь не забыть, лучше всего придумывать цикл **пока** по частям:

1) Понять, когда цикл должен закончиться, т.е. сформулировать ***условие окончания*** цикла. Записать после **пока** противоположное условие — ***условие продолжения*** цикла.
2) Выяснить, что и как будет меняться в цикле, описать промежуточные состояния после нескольких повторений цикла.
3) Описать, что происходит при однократном выполнении цикла (принято говорить “за один ***шаг*** цикла”), т.е. записать тело цикла.
4) Проверить, что цикл рано или поздно закончится, а не будет повторяться вечно.

10.10. Примеры построения алгоритмов

1. Закрашивание ряда.

Робот находится у левой стены внутри прямоугольника, огороженного со всех сторон стенами (*рис.* 23). Внутри прямоугольника стен нет, размеры прямоуголь-

ника неизвестны. Требуется закрасить горизонтальный ряд клеток от исходного положения Робота до правой стены и вернуть Робота в исходное положение.

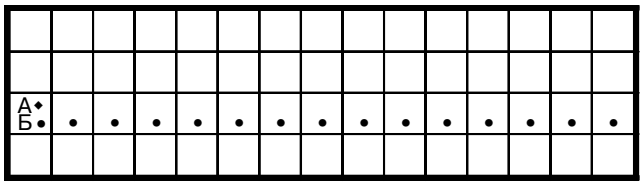


Рис. 23

Ясно, что алгоритм должен состоять из двух частей: сначала Робота надо довести до правой стены, по дороге закрашивая клетки, а затем вернуть в исходное положение.

```
алг закрасить ряд вправо и вернуться (А21)
 дано | Робот стоит у левой стены внутри
      | огороженного с четырех сторон
      | прямоугольника (рис.23)
 надо | горизонтальный ряд, в левой клетке
      | которого стоял Робот, полностью
      | закрашен. Робот в исходном положении
нач
| вправо до стены с закрашиванием
| влево до стены
кон
```

Вспомогательный алгоритм влево до стены мы уже составили (см. п. 10.3), займемся теперь алгоритмом вправо до стены с закрашиванием. В соответствии с методом последовательного уточнения, записав вызов этого алгоритма, мы фактически задали его заголовок:

```
алг вправо до стены с закрашиванием
  дано | где-то справа от Робота есть стена
  надо | Робот у стены, все клетки от
       | исходной до стены закрашены
```

Построим тело алгоритма. Поскольку расстояние до стены неизвестно, надо использовать цикл **пока**. После цикла Робот должен стоять у стены, следовательно, условие окончания цикла — справа стена. Тогда условие продолжения цикла — справа свободно.

На каждом шаге цикла Робот должен сместиться вправо и закрасить клетку. Получается такой цикл:

```
нц пока справа свободно
| вправо
| закрасить
кц
```

Ясно, что зацикливания не произойдет: по условию задачи стена где-то справа есть, при каждом выполнении цикла Робот делает шаг вправо и расстояние до стены уменьшается. Когда оно станет равным нулю, цикл закончится.

При выполнении этого цикла окажутся закрашенными все клетки правее исходного положения Робота, но сама эта клетка останется незакрашенной. Поэтому перед выполнением цикла нужно отдельно закрасить исходную клетку. Необходимость дополнительной команды закрасить можно объяснить и так: количество закрашенных клеток должно быть на одну больше, чем количество шагов, сделанных Роботом (это видно из рисунка); в цикле выполняется один шаг и одно закрашивание, следовательно, необходимо одно дополнительное закрашивание вне цикла.

Окончательно получаем алгоритм:

```
алг вправо до стены с закрашиванием (А22)
дано | где-то справа от Робота есть стена
 надо | Робот у стены, все клетки от
      | исходной до стены закрашены
нач
| закрасить
| нц пока справа свободно
| |  вправо
| |  закрасить
| кц
кон
```

2. Закрашивание коридора произвольной длины.

Робот стоит в левом конце горизонтального коридора, нижняя стена которого сплошная, а в верхней имеется несколько выходов. Составить алгоритм, который переводит Робота из клетки А в клетку Б (*рис.* 24) и закрашивает все клетки коридора (на *рис.* 24 отмечены точками).

Как составить такой алгоритм? Поскольку число клеток в коридоре неизвестно, то при записи алгоритма не обойтись без цикла **пока**. В какой же момент цикл должен окончиться? Он должен окончиться, когда Робот окажется в клетке Б. Чем клетка Б отличается от клеток коридора? Как видно из *рис.* 24, у клетки Б нет стены снизу, а у всех клеток коридора стена снизу есть. Поэтому после **пока** можно написать условие снизу стена. За один шаг цикла Робот должен закрашивать очередную клетку и переходить в следующую. Количество закрашенных клеток здесь равно количеству шагов (клетка Б не закрашивается!), поэтому дополнительные команды не нужны.

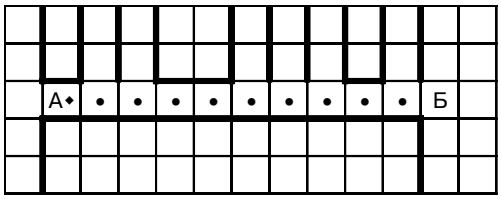


Рис. 24

```
алг закрасить коридор (А23)
  дано | Робот в левой клетке
       | горизонтального коридора (рис.24)
  надо | Робот вышел из коридора вправо,
       | коридор закрашен
нач
| нц пока снизу стена
| |  закрасить
| |  вправо
| кц
кон
```

3. Выход в левый верхний угол в лабиринте.

Робот находится внутри прямоугольного лабиринта, огороженного с четырех сторон стенами. Внутри лабиринта стены имеют вид отрезков прямых и не касаются друг друга и наружных стен (*рис.* 25). Составить алгоритм, который в любом таком лабиринте перемещает Робота в левый верхний угол.

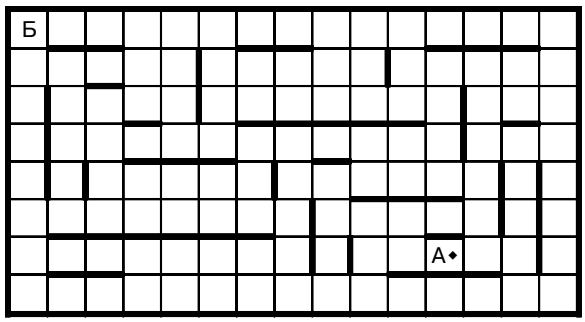


Рис. 25

Поскольку размеры лабиринта неизвестны, нам не обойтись без цикла **пока**. Когда этот цикл должен закончиться, т.е. чем конечное положение Робота (клетка Б) отличается от всех остальных? Видно, что в клетке Б стены есть и слева и сверху, а во всех остальных клетках хотя бы одной из этих стен нет. Значит, условие окончания цикла — сверху стена **и** слева стена, т.е. цикл надо продолжать до тех пор, пока либо слева, либо сверху от клетки свободно. Это условие записывается так: слева свободно **или** сверху свободно.

Внутри цикла надо смещать Робота по направлению к углу. Воспользуемся методом последовательного уточнения — введем вспомогательный алгоритм сместиться к углу (его мы составим потом) и запишем основной алгоритм:

```
алг в левый верхний угол лабиринта        (A24)
  дано | Робот где-то в лабиринте без углов
       | (рис.25)
  надо | Робот в левом верхнем углу
       | лабиринта
нач
| нц пока слева свободно или сверху свободно
|  | сместиться к углу
| кц
кон
```

Теперь составим вспомогательный алгоритм сместиться к углу:

```
алг сместиться к углу                     (A25)
  дано | Робот где-то в лабиринте без углов
       | (рис.32), либо слева, либо сверху
       |  от Робота свободно
  надо | Робот сместился к левому верхнему
       | углу
нач
| вверх до стены
| влево до стены
кон
```

Как проверить, что цикл в алгоритме (A24) рано или поздно закончится? Можно рассуждать так: при каждом выполнении тела цикла (т.е. при каждом выполнении алгоритма сместиться к углу) расстояние от Робота до левого верхнего угла лабиринта уменьшается. Значит, рано или поздно Робот окажется в углу и цикл закончится.

Заметим, что Робота можно провести той же дорогой и не используя составных условий, например, так:

```
алг в левый верхний угол лабиринта        (A26)
  дано | Робот в лабиринте, вид которого
       | изображен на рис. 25
  надо | Робот в левом верхнем углу
       | лабиринта
нач
| влево до стены
| нц пока сверху свободно
|  | вверх до стены
|  | влево до стены
| кц
кон
```

Задачи и упражнения

1. Составьте диалог компьютера и Робота при выполнении цикла

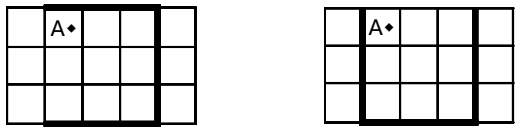
а)
```
нц пока клетка чистая
| закрасить
кц
```

б)
```
нц пока клетка закрашена
| закрасить
кц
```

в ситуации, когда Робот стоит:
1) в закрашенной клетке,
2) в незакрашенной.

2. Расположение Робота показано на *рис.* 26. Составьте диалог компьютера и Робота при выполнении цикла

```
нц пока сверху свободно
| вправо
кц
```



а *б*

Рис. 26

3. Переделайте алгоритм вправо до стены с закрашиванием (A22), используя в нем цикл:

```
нц пока справа свободно
| закрасить; вправо
кц
```

4. Используя в качестве вспомогательного алгоритм закрасить ряд вправо и вернуться (A21), составьте алгоритм:

```
алг закрасить прямоугольник
  дано | Робот стоит в левом верхнем
       | углу внутри огороженного с
       | четырех сторон прямоугольника
  надо | весь прямоугольник закрашен,
       |Робот в исходном положении
```

5. Решите задачу 4, считая, что про начальное положение Робота в прямоугольнике ничего не известно.
6. На поле Робота стен нет. Робот находится в левом верхнем углу прямоугольника из закрашенных клеток. Составьте алгоритм, переводящий Робота в правый нижний угол прямоугольника.
7. Составьте алгоритмы со следующими заголовками:

а) **алг** закрасить до стены вправо и вернуться
дано | где-то правее Робота есть стена
надо | закрашен ряд клеток между Роботом и стеной (рис.27), Робот в исходном положении

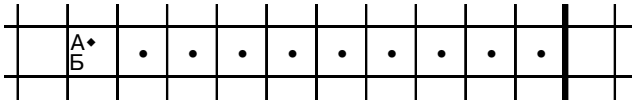


Рис. 27

б) **алг** закрасить до закрашенной клетки вправо и вернуться
дано | где-то правее Робота есть закрашенная клетка (рис.28)
надо | закрашен ряд клеток между Роботом и этой клеткой, Робот в исходном положении (рис.28)

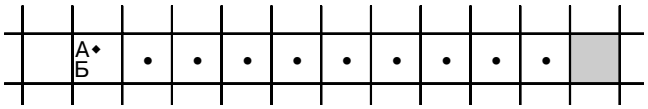


Рис. 28

в) **алг** закрасить коридор
дано | Робот где-то в горизонтальном коридоре
надо | закрашены все клетки коридора, кроме стартовой (клетки А), Робот в исходном положении (рис.29)

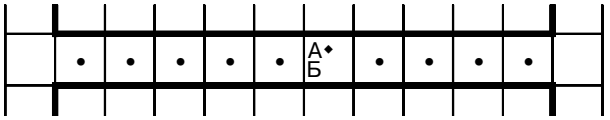


Рис. 29

г) **алг** закрасить коридор
дано | Робот где-то в горизонтальном коридоре
надо | закрашены все клетки коридора (рис.30), Робот в исходном положении (рис.30)

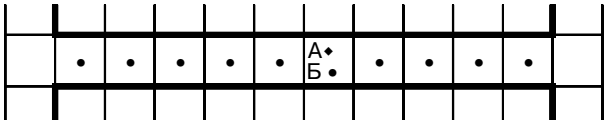


Рис. 30

д) **алг** закрасить угол
дано | Робот внутри прямоугольника, огороженного стенами
надо | закрашены все клетки правее и выше стартовой (рис.31) Робот в исходном положении

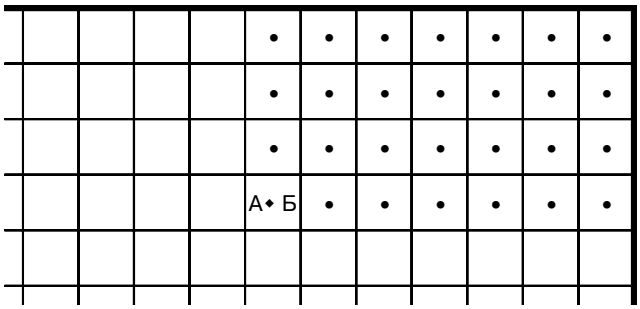


Рис. 31

8. Дано, что на поле Робота только одна стена, и эта стена расположена строго горизонтально. Робот находится в одной из клеток, прилегающих к стене сверху (*рис.* 32). Точные размеры стены и точное расположение Робота неизвестны. Составьте алгоритм, при выполнении которого Робот:
а) окажется в одной из клеток, прилегающих к стене снизу;
б) закрасит все клетки, прилегающие к стене сверху;
в) закрасит все клетки, прилегающие к стене снизу;
г) закрасит все прилегающие к стене клетки.

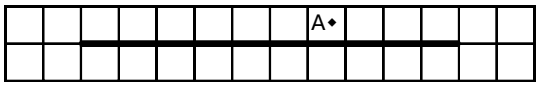


Рис. 32

9. Робот находится внутри прямоугольника, огороженного с четырех сторон стенами. Внутри прямоугольника стен нет. Составьте алгоритм, при выполнении которого Робот закрашивает все клетки внутри прямоугольника, прилегающие к стенам.
10. Составьте алгоритм со следующим заголовком:

алг обход прямоугольника
дано | Робот над верхней стороной прямоугольника, огороженного стенами; снаружи прямоугольника стен нет
надо | Робот под нижней стороной прямоугольника

11. Робот находится в левом верхнем углу в прямоугольнике, огороженном с четырех сторон стенами. Внутри прямоугольника имеется горизонтальная стена с одним проходом, идущая от левого до правого края прямоугольника (проход не прилегает ни к левой, ни к правой стене прямоугольника). Составьте алгоритм, при выполнении которого Робот переместится в правый нижний угол прямоугольника (*рис.* 33).

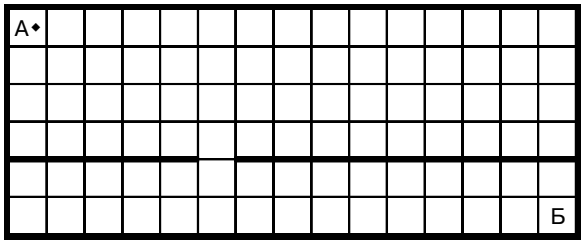


Рис. 33

12. Составьте алгоритм для закраски всех клеток вокруг прямоугольной стены (*рис.* 34).

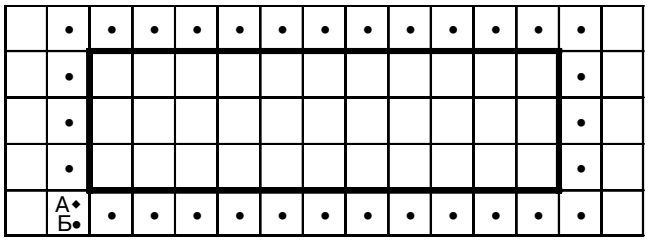


Рис. 34

13. Составьте алгоритм для закраски всех клеток вокруг Т-образной стены неизвестного размера (*рис.* 35).

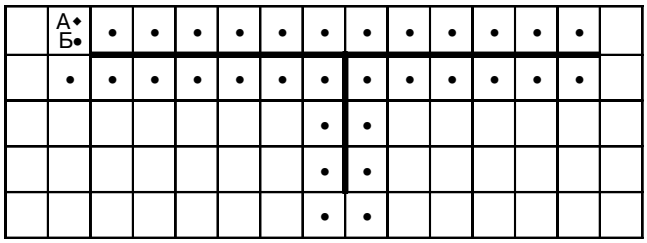


Рис. 35

§ 11. Команды ветвления и контроля

11.1. Пример алгоритма с командой **если**

Задача. Робот на дежурстве. Робот охраняет помещение, состоящее из двух соседних клеток (*рис.* 36). Неизвестно, в какой из двух клеток находится Робот. Необходимо перевести его в другую клетку.

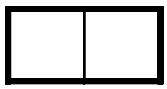


Рис. 36. Охраняемое помещение

В режиме непосредственного управления Роботом задача решается просто. Чтобы узнать, в какой клетке сейчас находится Робот, зададим ему вопрос: `справа свободно?` Ответ **да** означает, что Робот в левой клетке и должен сделать шаг вправо. При ответе **нет** Робот в правой комнате и должен сдвинуться `влево`.

Наши действия при непосредственном управлении Роботом можно описать так: если справа свободно, то скомандовать `вправо`, иначе скомандовать `влево`.

При решении этой задачи мы задавали Роботу вопрос и в зависимости от ответа выбирали следующее действие. Для описания такого выбора в алгоритмическом языке существует специальная команда **если**. Вот как можно записать алгоритм решения рассмотренной задачи:

```
алг дежурство                                    (A27)
  дано | Робот в одной из клеток
       | "двухкомнатного" помещения
       | (рис.36)
  надо | Робот в другой клетке
нач
  если справа свободно
    то вправо
    иначе влево
  все
кон
```

Рассмотрим еще одну задачу. Робот стоит в левом конце горизонтального коридора, нижняя стена которого сплошная, а в верхней имеется несколько выходов. Надо перевести Робота из клетки А в клетку Б и закрасить все клетки коридора, из которых есть выход вверх (*рис.* 37). Длина коридора, количество выходов и их точное расположение заранее неизвестны.

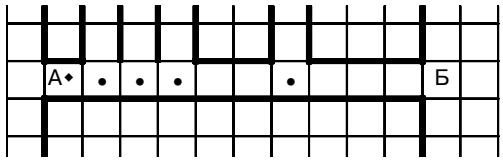


Рис. 37

Вспомним алгоритм (`A23`) из предыдущего параграфа, решающий аналогичную задачу, но с закрашиванием всех клеток коридора. Новая задача отличается только тем, что красить надо не все клетки коридора, а лишь те, где есть выход вверх. Другими словами, если сверху свободно, то клетку надо закрасить, иначе — красить не надо. Попробуем записать это в виде команды **если**:

```
если сверху свободно
  то закрасить
  иначе ...
все
```

А что собственно “иначе”? Как записать, что иначе ничего делать не надо? Для таких случаев существует краткая форма команды **если**, в которой слова **иначе** вообще нет:

```
если сверху свободно
  то закрасить
все
```

При выполнении этой команды компьютер спросит Робота: `сверху свободно?` Если Робот ответит **да**, то компьютер скомандует Роботу: `закрасить`. Если же Робот ответит **нет**, то компьютер вызов команды `закрасить` пропустит.

```
алг разметка выходов из коридора          (A28)
  дано | Робот в левой клетке горизонтального
       | коридора (рис.37)
  надо | Робот вышел из коридора вправо,
       | клетки коридора, из которых
       | есть выход вверх, закрашены
нач
| нц пока снизу стена
|   | если сверху свободно
|   | | то закрасить
|   | все
|   | вправо
| кц
кон
```

11.2. Общий вид команды **если**

Общий вид команды **если** таков:

```
если условие          или      если условие
| то  серия 1                   | то  серия 1
| иначе  серия 2                все
все
```

Служебные слова **если**, **то**, **иначе** имеют обычный смысл. Слово **все** означает конец команды. Это слово пишется строго под словом **если** и соединяется с ним вертикальной чертой. Между **то** и **иначе** — в одной или нескольких строках — записывается последовательность команд алгоритмического языка (серия 1). Между **иначе** и **все** записывается другая последовательность команд (серия 2). Серия 2 вместе со служебным словом **иначе** может отсутствовать.

При выполнении команды **если** компьютер сначала проверяет условие, записанное между **если** и **то** (задает Роботу вопрос). Если условие выполнено, то выполняется серия 1, а если нет — то серия 2 (если она есть).

Если условие не соблюдается (Робот ответил **нет**), а серия 2 вместе с **иначе** отсутствует, то компьютер сразу переходит к выполнению команд, записанных после слова **все**.

Графическая схема выполнения команды **если** приведена на *рис.* 38 (*а* — полная форма, *б* — сокращенная).

11.3. Команды контроля

Вспомните: когда вы объясняете, как пройти к определенному месту, то наверняка говорите что-нибудь вроде `заверните за угол, там будет видна булочная`. Это значит, что после выполнения команды `заверните за угол` надо проверить условие `видна булочная`. Проверка этого условия нужна не для выбора каких-то действий (как в командах **если** и **выбор**), а для того, чтобы убедиться: все идет нормально. Если булочной нет, значит, где-то раньше была допущена ошибка и дальнейшие действия бессмысленны.

Подобные контрольные утверждения применяются и при записи алгоритмов. Смысл их состоит в том, чтобы проверить, соответствуют ли условия выполнения алгоритма нашим представлениям о них. Если что-

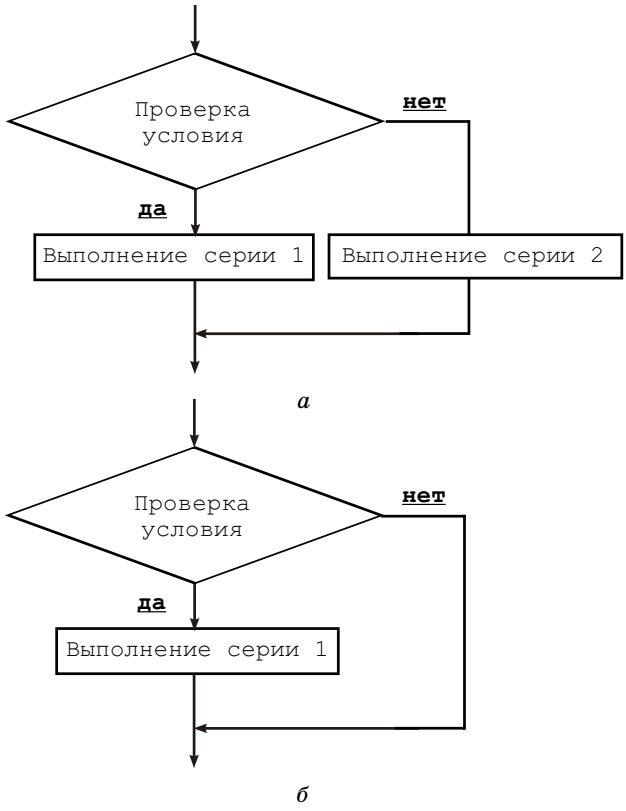


Рис. 38

то не так, это свидетельствует либо о несоответствии условий, либо об ошибке в алгоритме. В обоих случаях работу лучше прекратить.

Для записи контрольных условий в алгоритмическом языке есть специальная команда контроля

утв условие

После служебного слова **утв** (утверждение) записывается контрольное условие. Если при выполнении команды **утв** контрольное условие оказывается нарушенным, компьютер прекращает выполнение алгоритма и сообщает, что возник отказ. Если же условие верно, то выполнение алгоритма нормально продолжается так, как если бы команды **утв** не было вовсе.

Контрольные условия можно указать не только в команде **утв**, но и после слов **дано** и **надо** в заголовке алгоритма. Условие после **дано** проверяется в начале выполнения алгоритма, а условие после **надо** — в конце.

Вместо контрольного условия, которое проверяет компьютер, в команде **утв** (так же, как и в **дано**/**надо**) можно записать комментарий, предназначенный только для человека и облегчающий понимание алгоритма.

11.4. Пример алгоритма с командой **утв**

Задача. Робот стоит в левой клетке горизонтального коридора, от которого вверх отходят тупики размером в одну клетку. Требуется вывести Робота из коридора вправо (в клетку Б), а тупики закрасить (*рис.* 39).

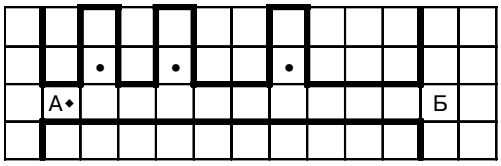


Рис. 39

Для решения этой задачи мы можем составить следующий алгоритм:

```
алг закрасить тупики                                    (А29)
  дано  | Робот стоит в левой клетке
        | горизонтального коридора,
        | выше некоторых клеток коридора
        | есть тупики  размером
        | в одну клетку (рис.39)
  надо  | Робот вышел из коридора вправо,
    | клетки всех  тупиков закрашены
нач
| нц пока снизу стена
| | если сверху свободно
| | | то
| | | вверх
| | | утв | Робот в тупике
| | | закрасить
| | | вниз
| | все
| | вправо
| кц
кон
```

В этом алгоритме команда **утв** содержит только комментарий, поясняющий алгоритм. Мы можем, однако, заменить ее на команду

```
утв  слева стена и сверху стена и справа стена
```

В этом случае при выполнении алгоритма компьютер будет проверять указанное условие и попытка выполнить алгоритм в неподходящей обстановке (т.е. в обстановке, отличающейся от *рис.* 39) приведет к отказу.

11.5. Выбор из многих вариантов

Команда **если** позволяет выбрать один из двух вариантов возможных действий. Но иногда бывают ситуации, когда выбор нужно сделать из большего числа вариантов. Рассмотрим, например, такую задачу.

Робот на распутье. Робот находится в клетке, из которой есть выход (вокруг клетки меньше четырех стен). Необходимо перевести Робота в соседнюю клетку.

Для решения этой задачи нужно, задавая Роботу вопросы, узнать, в каком направлении есть выход, и дать соответствующую команду движения. Однако запись этого алгоритма с помощью команды **если** оказывается очень громоздкой:

```
алг уйти из клетки                                      (А30)
  дано  | Робот в клетке, из которой есть
        | выход
  надо  | Робот вышел из исходной клетки
нач
| если сверху свободно
| | то вверх
| | иначе
| | если справа свободно
| | | то вправо
| | | иначе
| | | если снизу свободно
| | | | то вниз
| | | | иначе
| | | | утв слева свободно
| | | | влево
| | | все
| | все
| все
кон
```

Для упрощения записи алгоритма при выборе из многих вариантов в алгоритмическом языке существует команда **выбор**. Используя эту команду, решение нашей задачи можно записать намного понятнее:

```
алг уйти из клетки                                      (А31)
  дано  | Робот в клетке, из которой есть
        | выход
  надо  | Робот вышел из исходной клетки
нач
| выбор
| | при сверху свободно : вверх
| | при справа свободно : вправо
| | при снизу  свободно : вниз
| | иначе утв слева свободно; влево
| все
кон
```

11.6. Общий вид команды выбор

В общем виде команда **выбор** записывается так:

```
выбор
| при условие 1 : серия 1
| при условие 2 : серия 2
| . . .
| при условие n : серия n
| иначе  серия n+1
все
```

```
выбор
| при условие 1 : серия 1
| при условие 2 : серия 2
| при условие n : серия n
| . . .
все
```

Количество условий и соответствующих им серий команд может быть любым. Как и в команде **если**, серия $n + 1$ вместе со служебным словом **иначе** может отсутствовать.

При выполнении команды **выбор** компьютер последовательно проверяет условия: условие 1, условие 2 и т.д. Как только очередное условие окажется верным, компьютер выполняет соответствующую ему серию команд и прекращает выполнение команды **выбор**,

переходя к командам, записанным после слова **все**. Оставшиеся условия не проверяются и серии не выполняются!

Например, если Робот стоит в чистом поле без стен, то при выполнении алгоритма уйти из клетки первое условие (сверху свободно) оказывается верным, компьютер командует Роботу вверх, и на этом исполнение алгоритма заканчивается.

Если все условия проверены и ни одно из них не оказалось верным, выполняется серия команд, записанная после слова **иначе**, или, если слова **иначе** нет, никакие команды не выполняются.

Графическая схема выполнения команды **выбор** показана на *рис.* 40.

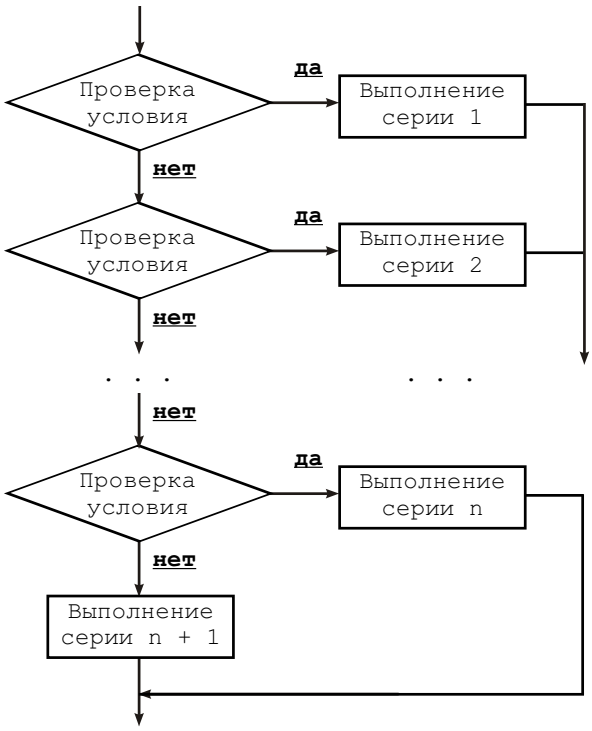


Рис. 40

Задачи и упражнения

1. На поле Робота стен нет. В ряду из десяти клеток правее Робота некоторые клетки закрашены. Составьте алгоритм, который закрашивает клетки
   а) ниже каждой закрашенной;
   б) выше и ниже каждой закрашенной;
   в) левее каждой закрашенной;
   г) правее каждой закрашенной;
   д) левее и правее каждой закрашенной.
2. Воспользовавшись вспомогательными алгоритмами вверх до стены, вниз до стены, вправо до стены, влево до стены, составьте алгоритмы со следующими заголовками:

а) **алг** переход в противоположный угол
дано | Робот стоит в каком-то углу
| прямоугольника, огороженного
| стенами. Других стен нет
надо | Робот в противоположном углу

б) **алг** к противоположной стене
дано | Робот стоит у стены (но не в углу)
| внутри прямоугольника, обнесенного
| со всех сторон стенами; внутри
| прямоугольника других стен нет
надо | Робот перешел к противоположной стене

3. Петя решил сделать алгоритм уйти из клетки более понятным без использования команды **выбор**. Вот что у него получилось:

```
алг уйти из клетки                                    (А32)
  дано  | Робот в клетке, из которой есть выход
  надо  | Робот вышел из исходной клетки
нач
| если сверху свободно
|  | то вверх
| все
| если справа свободно
|  | то вправо
| все
| если снизу свободно
|  | то вниз
| все
| если слева свободно
|  | то влево
| все
кон
```

Всегда ли после выполнения этого алгоритма Робот выйдет из исходной клетки?

4. Робот внутри коридора без боковых выходов, идущего в неизвестном направлении. Составьте алгоритм, выводящий Робота из коридора.
5. Робот внутри тупика неизвестного направления. Составьте алгоритм, выводящий Робота из тупика.
6. Робот на перекрестке, от которого отходят один коридор (без боковых выходов) и три тупика. Составьте алгоритм, после выполнения которого Робот окажется с противоположной стороны коридора.
7. Робот в левой клетке коридора неизвестной длины, уходящего вправо. Из некоторых клеток коридора есть выходы вверх, некоторые клетки коридора закрашены. Составьте алгоритм, после выполнения которого Робот выйдет из коридора, а на экране Счетчика будет
   а) длина коридора;
   б) количество закрашенных клеток коридора;
   в) количество незакрашенных клеток коридора;
   г) количество боковых выходов;
   д) количество отмеченных выходов (отмеченным считается выход, у которого соответствующая клетка коридора закрашена).

Продолжение следует

Транслятор?.. Это очень просто!

А.А. Дуванов

Машина имеет четыре скорости: первую, вторую, третью и четвертую, а также задний и боковой ход. В задней части машины имеется приспособление для стирки белья. Стирка может производиться во время движения на любой скорости. В спокойном состоянии, то есть на остановках, машина рубит дрова, месит глину и делает кирпичи, а также чистит картошку.

Николай Носов

Продолжение. Начало в № 41/99

Эпиграф к этой статье — будто бы в точности про наших ребят: Кукарачу и Корректора. Кто бы мог подумать, что с помощью этих простых исполнителей можно решать сложные программистские задачи!

АХ УЖ ЭТА РЕКУРСИЯ!

В прошлый раз в лексическом анализаторе для Кукарачи автор использовал рекурсивную процедуру ошибка:

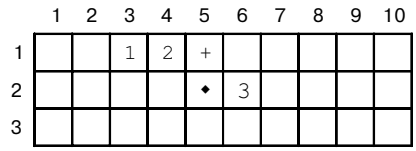
```
ЭТО ошибка                          | Перемещение
   шаг                              | остатка записи
   ЕСЛИ НЕ ПУСТО ТО ошибка          | в первую строку
   ВЛЕВО                            | Возврат к месту
КОНЕЦ                               | ошибки
```

Вспомним, что процедура шаг выглядит так:

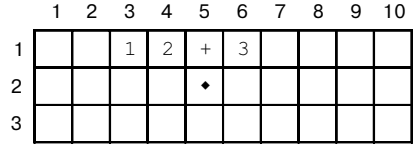
```
ЭТО шаг
   ВНИЗ ВПРАВО ВВЕРХ
КОНЕЦ
```

Автор уверял доверчивых читателей, что это будет работать: Кукарача якобы сделает влево не один шаг, а столько, сколько надо, чтобы встать точно под обнаруженной в записи ошибкой!

Пусть, исследуя запись, исполнитель встретил ошибку (знак “+”):



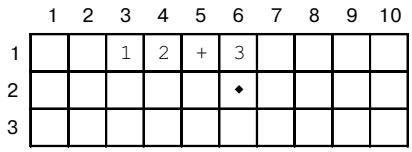
Теперь Кукарача должен сдвинуть остаток записи в первую строку и встать точно под ошибочным символом:



Неужели процедура ошибка это все сделает?

Давайте проследим работу исполнителя по шагам.

Первой выполняется команда шаг:

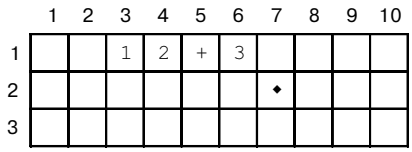



```
ЭТО ошибка
  шаг
   ЕСЛИ НЕ ПУСТО ТО ошибка
   ВЛЕВО
КОНЕЦ
```

Затем выполняется условная команда:

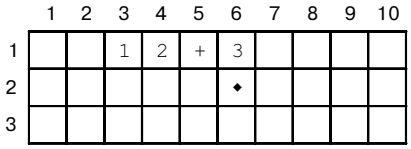
```
ЭТО ошибка
   шаг
   ЕСЛИ НЕ ПУСТО ТО ошибка
   ВЛЕВО
КОНЕЦ
```

Так как условие истинно, начнет выполняться копия процедуры ошибка:



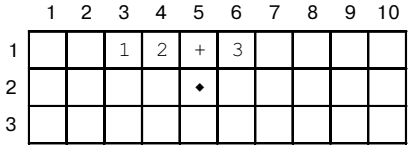

```
ЭТО ошибка
  шаг
   ЕСЛИ НЕ ПУСТО ТО ошибка
   ВЛЕВО
КОНЕЦ

ЭТО ошибка
  шаг
   ЕСЛИ НЕ ПУСТО ТО ошибка
   ВЛЕВО
КОНЕЦ
```

Теперь условие ложно и выполнится команда ВЛЕВО в копии программы ошибка:




```
ЭТО ошибка
  шаг
  ЕСЛИ НЕ ПУСТО ТО ошибка
  ВЛЕВО
КОНЕЦ

ЭТО ошибка
  шаг
  ЕСЛИ НЕ ПУСТО ТО ошибка
  ВЛЕВО
КОНЕЦ
```

Работа копии программы ошибка завершена, и тем самым завершилось выполнение условной команды в оригинале. Кукарача приступает к выполнению последней команды ВЛЕВО и устанавливается точно под знаком "+":




```
ЭТО ошибка
  шаг
  ЕСЛИ НЕ ПУСТО ТО ошибка
  ВЛЕВО
КОНЕЦ

ЭТО ошибка
  шаг
  ЕСЛИ НЕ ПУСТО ТО ошибка
  ВЛЕВО
КОНЕЦ
```

Фокус разоблачен! Команда ВЛЕВО выполнится столько раз, сколько работала процедура шаг в рекурсивных вызовах. А это гарантирует, что исполнитель вернется к месту ошибки со 100%-ной точностью, где бы ни располагалась ошибка в записи!

Разоблачение фокуса заключено в простом понимании, что рекурсия — это не GOTO (переход на начало процедуры). Рекурсивный вызов — это выполнение **копии** процедуры: он может порождать "отложенные команды", которые начнут выполняться на "обратном ходе" рекурсии. И будут выполняться столько раз, сколько было рекурсивных вызовов.

Ребята из Барнаула (руководитель команды — Анатолий Владимирович Бычков) нашли в прошлом году очень выразительный и емкий ассоциативный термин для описания работы отложенных в рекурсии команд — "рекурсивная пружина".

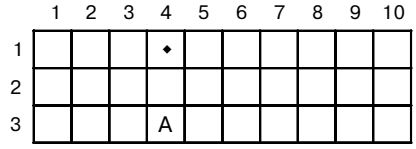
Кукарача, делая рекурсивный шаг, сжимает пружину. По окончании рекурсии пружина разжимается: команда ВЛЕВО сработает столько раз, сколько витков было "закручено".

Как можно решить эту задачу на Бейсике? Да очень просто: завели бы переменную (ячейку памяти), считали бы в ней шаги вправо, а потом: Кукарача, шагай столько же раз влево!

Но Кукарача — парень без головы! Ну, плохо соображает! Совсем! С рекурсией Кукарача (ого!) умеет считать и запоминать!

Вот совсем простое упражнение с рекурсивной пружинкой (ответ в конце статьи).

Упражнение (автор — Е.П. Лилитко). Кукарача находится в верхней строке поля. Где-то под ним (в том же столбце) расположена буква "А". Требуется спуститься к букве (где бы она ни находилась), подвинуть ее один раз и вернуться к исходной позиции в верхнюю строку.



Вам понравилась рекурсия? Но и ворчуны среди вас наверняка найдутся. Специально для них небольшое отступление против рекурсии.

ПОДВОДНЫЕ КАМНИ РЕКУРСИИ

Рекурсия порой очень сильно упрощает программирование, делая текст программы удивительно коротким.

Но все же основным инструментом программиста остается цикл ПОКА. Эта управляющая структура нагляднее рекурсии, проще в использовании и часто эффективнее при выполнении. Поэтому если задача решается при помощи цикла ПОКА, то не надо стрелять из пушки по воробьям. Хотя, конечно, изящные рекурсивные процедуры выглядят очень элегантно.

Сказанное совсем не означает, что рекурсия находится на задворках человеческой мысли. Рекурсия, безусловно, мощный и красивый инструмент математики и, конечно, информатики. Часто рекурсивные описания настолько легки и красивы, насколько тяжелы и громоздки попытки обойтись циклами. Посмотрите, например, как лаконично описывается при помощи рекурсии последовательность чисел Фибоначчи:

$f(1) = 1, f(2) = 1,$

$f(n) = f(n-1) + f(n-2)$, для всех $n > 2$

Напоминание о числах Фибоначчи

Числа Фибоначчи определяются следующей бесконечной последовательностью целых чисел: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Каждый n-й элемент этой последовательности, начиная с третьего, равен сумме двух непосредственно предшествующих элементов. Первые два члена последовательности равны единице.

Эти числа являются решением следующей задачи (итальянский математик Фибоначчи сформулировал ее в 1202 году):

"Каждый месяц самка из пары кроликов приносит двоих детенышей (самца и самку). Через два месяца новая самка сама приносит пару кроликов. Нужно найти число кроликов в конце года, если в начале года была одна новорожденная пара и в течение этого года кролики не умирали".

Ответ: к концу года число пар кроликов достигнет числа $f(12) = 144$, то есть поголовье возрастет до 288 кроличьих единиц.

Однако надо заметить, что на практике такое размножение кроликов ни к чему хорошему не приведет. Родственные "браки" повлекут быстрое вырождение потомства. Когда-то в детстве автор разводил кроликов и, не имея соответствующего образования (и должной ответственности перед живой природой!), пришел к этому выводу на практике. И потом, насколько я помню, крольчихи достигают половой зрелости не через 2 месяца после рождения, а через три. Последнее замечание наводит на мысль о новой задаче: описать и построить последовательность для решения задачи Фибоначчи при условии, что кролики вступают в "брак" через 3 месяца после рождения. Сколько кроликов будет в крольчатнике к концу года, если колония начала плодиться с одной пары?

Заядлые крольчатники сделают еще одно замечание: очень редко самка приносит только двоих детенышей. Обычный приплод — 6—12 (однажды крольчиха подарила мне 18 ушастиков!). Кроме того, опытные кролиководы не позволяют самкам становиться мамами так часто! Это еще один повод формулировать новые рекурсивные задачи.

Несколько примеров рекурсивных описаний.

1. Факториал.

$0! = 1$

$n! = n \cdot (n - 1)!$

2. Сумма последовательности.

Пусть дана бесконечная последовательность чисел: $a_1, a_2, \ldots, a_n, \ldots$

Если обозначить через $S(n)$ сумму n членов этой последовательности, то

$$S(1) = a_1$$
$$S(k) = S(k - 1) + a_k$$

3. Решение задачи о Ханойских башнях.

Пусть $H(n, i, j, k)$ описывает алгоритм переноса n колец с i-го стержня на j-й, используя k-й стержень как рабочий (i, j, k принимают значения из множества $\{1, 2, 3\}$ и различны между собой).

Тогда:

$$H(1, i, j, k) = i \to j$$

$$H(n, i, j, k) = \begin{vmatrix} \mathrm{H}(n - 1, i, k, j) \\ i \to j \\ H(n - 1, k, j, i) \end{vmatrix}$$

Замечание. Операция $\to$ введена для обозначения переноса кольца с одного стержня на другой. Например, запись $1 \to 3$ обозначает перенос кольца с первого стержня на третий.

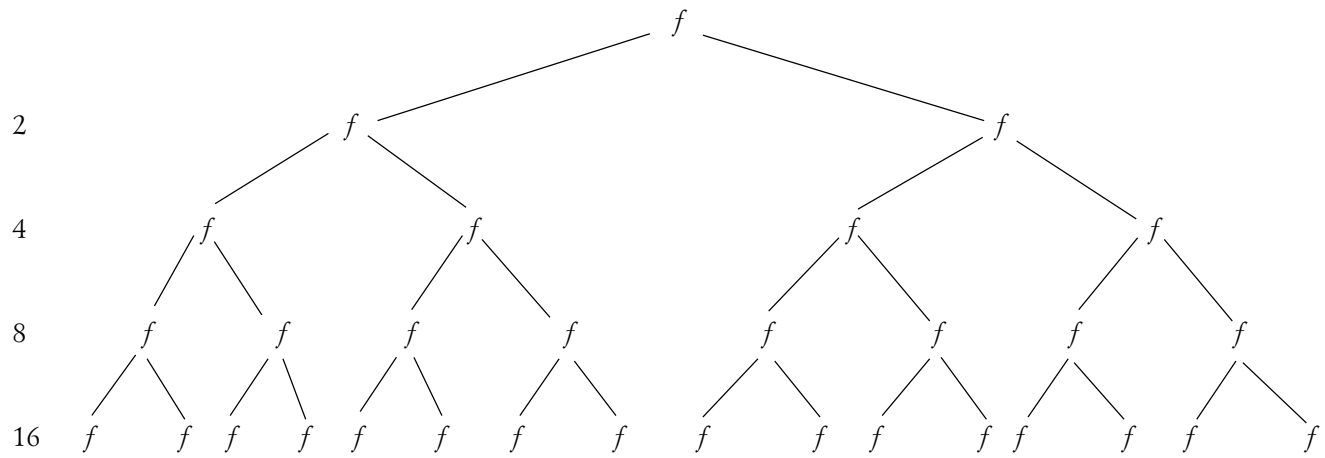
Следует помнить, что использование рекурсии таит в себе "подводные камни".

Первый камень. Использовать рекурсию разумнее всего тогда, когда природа задачи сама по себе рекурсивна. Искусственно построенная рекурсия (как альтернатива циклу) редко бывает оправданна. Например, для суммирования членов последовательности лучше применить цикл, а не рекурсию. В чем заключается опасность рекурсии при машинных вычислениях? Дело в том, что при каждом рекурсивном вызове расходуется дополнительно часть машинной памяти (в ней сохраняются адрес возврата и локальные переменные процедуры, если язык допускает использование переменных). При "глубоких" рекурсивных вызовах вся доступная память компьютера может быстро исчерпаться.

Но даже в некоторых естественно-рекурсивных задачах рекурсию использовать не всегда разумно. В качестве примера рассмотрим упоминаемую выше последовательность чисел Фибоначчи.

Каждое обращение к $f(n)$ при $n > 2$ приводит к двум рекурсивным обращениям. Посмотрите, как быстро растет число вызовов процедуры f при рекурсивном вычислении чисел Фибоначчи.

Число рекурсивных вызовов растет как степень двойки, то есть так же быстро, как размножаются кролики!



Числа Фибоначчи лучше вычислять при помощи цикла ПОКА:

```
ЭТО f(n)
  i:= 1        // i — счетчик чисел Фибоначчи
  x:= 1        // x — сумматор
  y:= 0        // y — вспомогательная переменная
ПОКА i < n
{
  x:= x + y    // суммируем два предыдущих
               // значения
  y:= x — y    // первое из них запоминаем в y
  i:= i + 1
}
ВЕРНУТЬ x
КОНЕЦ
```

Второй камень. Использовать рекурсию как замену отсутствующей в языке конструкции GOTO еще вреднее, чем использовать сам GOTO. Рассматривать рекурсию как переход в начало процедуры неправильно, ведь рекурсия — это не переход, а выполнение копии процедуры; выполнение без всякого перехода, в том самом месте, где стоит рекурсивный вызов.

Третий камень. Прямая рекурсия всегда предпочтительнее косвенной*. Косвенная рекурсия — это замаскированная форма прямой, а всякая маскировка действий ухудшает как проектирование программы, так и ее отладку, не говоря уже о попытках разобраться в текстах программы постороннему глазу (и даже глазу автора программы через неделю после написания текста).

К нашим баранам

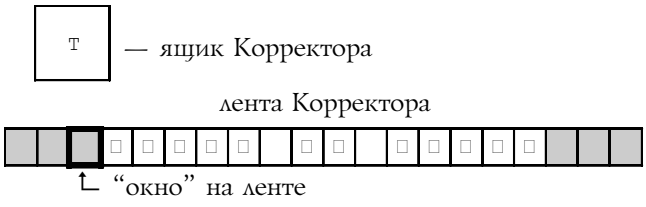
Рекурсия — очень интересная тема. Быть может, мы еще доберемся до важной практической части: как сделать так, чтобы рекурсии в рекурсии было как можно меньше! Интрига обозначена, а теперь...

ЗНАКОМЬТЕСЬ: КОРРЕКТОР

Корректор работает с лентой, которая разбита на клетки (ячейки). В ячейку можно записать один символ. Символ — это буква, цифра, знак "+" или что-нибудь еще в этом роде. Полный набор допустимых символов составляет алфавит исполнителя.

Считается, что лента спрятана в непрозрачный для Корректора футляр. Исполнитель может видеть только одну ячейку ленты через специальное окно в футляре и записывать в "оконную" ячейку любые символы из своего алфавита. Он может также перемещать окно вправо и влево по ленте и, таким образом, записывать символы в любые клетки на ней.

Еще робот имеет "ящик" — специальную ячейку памяти. В ящике можно хранить один символ.



Ниже приводятся полный набор команд Корректора и его алфавит.

ВПРАВО	— переместить окно на одну клетку ленты вправо.
ВЛЕВО	— переместить окно на одну клетку ленты влево.
ПИШИ символ	— записать указанный символ в клетку на ленте.
ЯЩИК+	— копировать символ из окна в ящик.
ЯЩИК—	— копировать символ из ящика в окно.
ОБМЕН	— поменять местами содержимое ящика и окна.
ПЛЮС	— заменить символ на ленте символом, следующим за данным в алфавите Корректора. Команда приводит к отказу (ситуация "Не могу"), когда в оконной ячейке перед ее выполнением записан последний символ алфавита.
МИНУС	— заменить символ на ленте символом, предыдущим по порядку в алфавите Корректора. Команда приводит к отказу (ситуация "Не могу"), когда оконная ячейка пуста (в ней записан символ ПУСТО).
СТОЯТЬ	— пустая команда; ее выполнение не вызывает никаких изменений в среде Корректора.

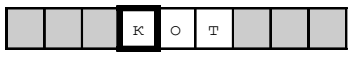
Алфавит Корректора

ПУСТО 1 2 3 4 5 6 7 8 9 а б в г д е ж з и й к л м н о п р с т у ф х ц ч ш щ ъ ы ь э ю я ПРОБЕЛ — + / * = < > ( ) [ ] { } . , ! ? ; : ' "# | $ % ~ @

Замечание. ПУСТО — это специальный символ. Мы будем рисовать его на ленте серой ячейкой (чтобы не путать с пробелом, который идет в алфавите после буквы "я").

Пример программы для Корректора

Задача. На ленте написано трехсимвольное слово, и в начальный момент окно установлено на его начало. Написать программу, которая меняет местами первый и последний символы.



* Если процедура А вызывает процедуру В, а та в свою очередь вызывает А, то это называется косвенной рекурсией.

Решение

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

Корректор был придуман автором статьи в 1986 году (двумя годами позже Кукарачи). Как давно это было! Эти ребята — ровесники школьной информатики.

ЛЕКСИЧЕСКИЙ АНАЛИЗАТОР КОРРЕКТОРА

Для Корректора программировать лексические анализаторы приятнее. Ведь этот исполнитель, в отличие от Кукарачи, умеет писать на ленту символы, а значит, может написать результат анализа "по-человечески", вместо Кукарачиных намеков на ответ своим положением на поле. В остальном анализатор Корректора для рассмотренного в прошлый раз определения и соответствующей диаграммы переходов — калька анализатора Кукарачи.

Постановка задачи

Программа должна проверить запись (конечной длины) на ленте и поместить результат проверки справа от нее в виде двух букв — "ош", "цл" или "др".

Здесь:

ош — запись числом не является;

цл — запись есть "целое";

др — запись есть "дробь".

В начальный момент окошко расположено перед записью, а в конце работы — сразу за ней, если в записи нет ошибок.

Когда запись неверная, окошко должно указывать на первый ошибочный символ.

```
// Окошко перед записью
ЭТО вход              Состояние (вход)
  ВПРАВО              Смотрим следующий символ
  ЕСЛИ ЦИФРА ТО целое
  ИНАЧЕ ошибка
КОНЕЦ

ЭТО целое                      Состояние (целое)
  ВПРАВО                       Смотрим следующий
  ЕСЛИ ПУСТО ТО ответ_целое    символ
  ИНАЧЕ ЕСЛИ ЦИФРА ТО целое
  ИНАЧЕ ЕСЛИ /      ТО дробь?
  ИНАЧЕ ошибка
КОНЕЦ

ЭТО дробь?            Состояние (дробь?)
  ВПРАВО              Смотрим следующий символ
  ЕСЛИ ЦИФРА ТО дробь
  ИНАЧЕ ошибка
КОНЕЦ

ЭТО дробь             Состояние (дробь)
  ВПРАВО              Смотрим следующий символ
  ЕСЛИ ПУСТО ТО ответ_дробь
  ИНАЧЕ ЕСЛИ ЦИФРА ТО дробь
  ИНАЧЕ ошибка
КОНЕЦ

// Запишем ответ "ош" и
// вернемся к месту ошибки
ЭТО ошибка
  ВПРАВО                       Перемещение окна
                               в конец записи
  ЕСЛИ ПУСТО ТО ответ_ошибка
  ИНАЧЕ ошибка                 и запись ответа
  ВЛЕВО                        Возврат к месту
КОНЕЦ                          ошибки

ЭТО ответ_ошибка
  ВПРАВО ПИШИ о
  ВПРАВО ПИШИ ш
  ПОВТОРИ 2 ВЛЕВО
КОНЕЦ

ЭТО ответ_целое
  ВПРАВО ПИШИ ц
  ВПРАВО ПИШИ л
  ПОВТОРИ 2 ВЛЕВО
КОНЕЦ

ЭТО ответ_дробь
  ВПРАВО ПИШИ д
  ВПРАВО ПИШИ р
  ПОВТОРИ 2 ВЛЕВО
КОНЕЦ
```

* * *

В следующий раз попробуем продвинуться в более сложные темы трансляции. А теперь по сложившейся традиции — несколько задач Корректора из конкурса Роботландского сетевого университета прошлого года.

Первые три задачи — это обычный лексический анализ (разной сложности).

А вот в задачах 4 и 5 нужно написать транслятор! В этих упражнениях после проверки выражения нужно его ВЫЧИСЛЯТЬ! А как? Ведь Корректору незнакомы арифметические операции. Для этого парня что "+2", что "дядя Вася"— всего лишь цепочка символов, ничуть не лучше и не хуже одна другой.

Ничего не поделаешь, нужен транслятор.

ЗАДАЧА 1 (5 БАЛЛОВ)

Введем следующее определение:

Определение

```
<число> ::= <целое> | <дробь>
<целое> ::= <цифра> | <целое><цифра>
<дробь> ::= <простая> | <десятичная>
<простая> ::= <целое>/<целое>
<десятичная> ::= .<целое> | <целое>.
                 | <целое>.<целое>
<цифра> ::= 0|1|2|3|4|5|6|7|8|9
```

Задание. Напишите программу, которая проверяет запись на ленте и помещает результат проверки справа от нее в виде двух букв:

ош — запись числом не является;

цл — запись есть "целое";

дс — запись есть "десятичная";

пр — запись есть "простая".

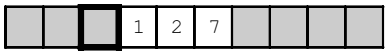
В начальный момент окошко расположено перед записью, а в конце работы — сразу за ней, если в записи нет ошибок.

Если запись неверная, окошко должно указывать на первый ошибочный символ.

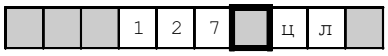
Примеры работы программы

1

Начальное состояние

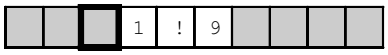


После работы программы

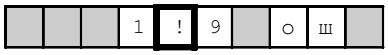


2

Начальное состояние



После работы программы



ЗАДАЧА 2 (8 БАЛЛОВ)

Введем следующее определение:

Определение

```
<выражение> ::= <число>+<число> |
                <выражение>+<число>
<число> ::= <целое> | <дробь>
<целое> ::= <цифра> | <целое><цифра>
<дробь> ::= .<целое> | <целое>. |
            <целое>.<целое>
<цифра> ::= 0|1|2|3|4|5|6|7|8|9
```

Задание. Напишите программу, которая проверяет запись на ленте и помещает результат проверки справа от нее в виде двух букв:

ош — запись не является ни числом, ни выражением;

чс — запись есть "число";

вр — запись есть "выражение".

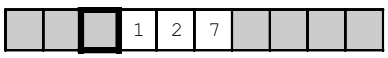
В начальный момент окошко расположено перед записью, а в конце работы — сразу за ней, если в записи нет ошибок.

Если запись содержит ошибки, окошко должно указывать на первый ошибочный символ.

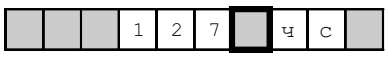
Примеры работы программы

1

Начальное состояние

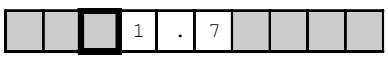


После работы программы

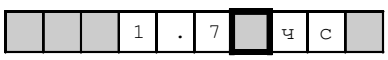


2

Начальное состояние

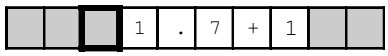


После работы программы

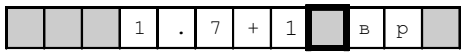


3

Начальное состояние

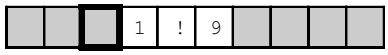


После работы программы



4

Начальное состояние



После работы программы

ЗАДАЧА 3 (10 БАЛЛОВ)

Введем следующее определение:

Определение

```
<выражение> ::= <число> |
<выражение>+<выражение> |
<выражение>-<выражение> |
(<выражение>)
<число> ::= 0|1|2|3|4|5|6|7|8|9
```

Задание. Напишите программу, которая проверяет правильность записи выражения. Если запись правильная, то нужно за ней на ленте записать "ОК".

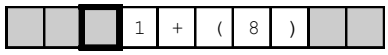
Если запись ошибочная, то нужно записать "ош" и дополнительно установить окошко на первый неверный символ или на место обнаружения ошибки в случае ошибок со скобками.

В начальный момент окошко расположено перед записью.

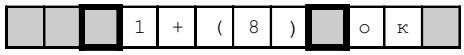
Примеры работы программы

1

Начальное состояние



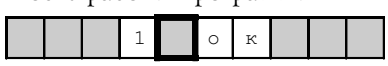
После работы программы



2

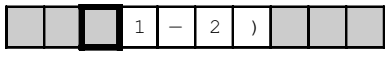
Начальное состояние

После работы программы

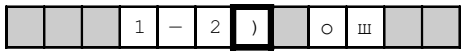


3

Начальное состояние



После работы программы



ЗАДАЧА 4 (9 БАЛЛОВ)

Введем следующее определение:

Определение

```
<выражение>::=<число>|<выражение>+<число>
<число>::=0|1|2
```

Задание. Напишите программу, которая вычисляет выражение, если оно правильное, или записывает на ленту сообщение "ош", если запись содержит ошибку. Если запись ошибочная, то дополнительно к сообщению "ош" окошко должно указывать на первый неверный символ.

Замечание 1. Значение выражения должно быть записано на ленте в виде обычного целого десятичного числа.

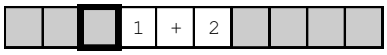
Замечание 2. Предполагается, что результат вычисления выражения не превышает длины алфавита Корректора.

В начальный момент окошко расположено перед записью.

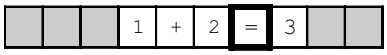
Примеры работы программы

1

Начальное состояние



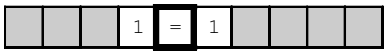
После работы программы



2

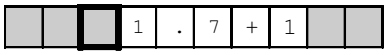
Начальное состояние

После работы программы

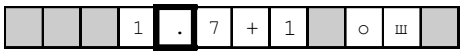


3

Начальное состояние



После работы программы



ЗАДАЧА 5 (9 БАЛЛОВ)

Введем следующее определение:

Определение

```
<выражение>::=<число>|<выражение>+<число>
              |<выражение>-<число>
<число>::=0|1|2
```

Задание. Напишите программу, которая вычисляет выражение, если оно правильное, или записывает на ленту сообщение "ош", если запись содержит ошибку. Если запись ошибочная, то дополнительно к сообщению "ош" окошко должно указывать на первый неверный символ.

Замечание 1. Значение выражения должно быть записано на ленте в виде обычного целого десятичного числа.

Замечание 2. Предполагается, что конечный и промежуточный результаты вычислений не превышают числа, равного длине алфавита Корректора.

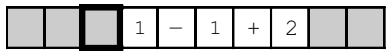
Замечание 3. Если конечный или промежуточный результат вычисления выражения окажется меньше нуля, записать на ленту сообщение "ав" (авария) и установить окошко на знак операции, которая приводит к отрицательному результату.

В начальный момент окошко расположено перед записью.

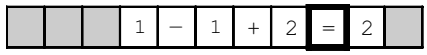
Примеры работы программы

1

Начальное состояние



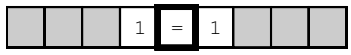
После работы программы



2

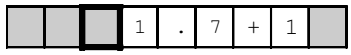
Начальное состояние

После работы программы

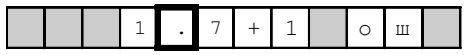


3

Начальное состояние

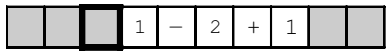


После работы программы

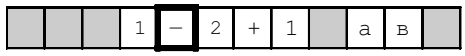


4

Начальное состояние



После работы программы



* * *

Ответ на упражнение

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

На первый взгляд кажется, что программа неверная: команда ВНИЗ, записанная последней, работает только один раз, и Кукарача в общем случае не вернется в исходное положение на поле. Но это не так. Команда ВНИЗ сработает ровно столько же раз, сколько команда ВВЕРХ, обеспечивая перемещение исполнителя в исходную точку.

Продолжение следует

Норберт Винер и его “наука об управлении”

Окончание. Начало на с. 1

социальными последствиями автоматизации. Лишь в самом конце 1960-х годов наметились пути сближения между “кибернетиками” и “вычислителями” [1].

В нашей стране развитие кибернетики пошло по иному пути. После первой отрицательной реакции постепенно (к началу 1960-х годов) определилось более широкое толкование кибернетики, полностью охватывавшее не только теорию ЭВМ, но и многочисленные применения в различных областях, начиная от автоматизации обработки научных данных до управления большими экономическими системами.

Сегодня можно утверждать, что появление мощных универсальных компьютеров, создание оптимальных и самонастраивающихся систем управления, развитие теории информации, теории автоматического управления, прикладной математики, программирования, достижения в области искусственного интеллекта, робототехники, телемеханики и многие другие научные и практические результаты прямо или косвенно связаны с работами в области кибернетики, которая давно уже завоевала не только право на существование, но и академическое признание.

Литература

1. *В.М. Глушков.* Кибернетика//Энциклопедия кибернетики. Т. 1. Киев: Гл. редакция Украинской Советской Энциклопедии, 1975.

2. *Н.Винер.* Кибернетика, или Управление и связь в животном и машине: 2-е изд. М.: Гл. редакция изданий для зарубежных стран издательства “Наука”, 1983.

Гл. редактор С.Л. Островский
Зам. гл. редактора Е.Б. Докшицкая
Редакция: И.Н. Фалина, Н.Л. Беленькая, Н.П. Медведева
Дизайн и компьютерная верстка: Н.И. Пронская
Корректоры: Е.Л. Володина, С.М. Подберезина

выходит четыре раза в месяц
При перепечатке ссылка на ИНФОРМАТИКУ обязательна, рукописи не возвращаются

121165, Киевская, 24
тел. 249 4896
Отдел рекламы
тел. 249 9870

Учредитель: ООО “Чистые пруды”
Регистрационный номер 012868
Отпечатано в типографии ОАО ПО “Пресса-1”. 125865, ГСП, Москва, ул. “Правды”, 24.
Тираж 5000 экз.
Заказ №

ИНДЕКС ПОДПИСКИ
для индивидуальных подписчиков 32291
комплекта приложений 32744

Тел. (095)249 3138, 249 3386. Факс (095)249 3184

Internet: inf@1september.ru
Fidonet: 2:5020/69.32
WWW: http://www.1september.ru